Principles and Practices of Software Development

Final Report

MoiSha: New Interest-based SNS for SNU

Team 2

김성은, 안진우, 이재윤, 정영수

Contents

1. Introduction 4

1.1. Motivation for Service 4

1.2. Target Customer/ Market Landscape 4

1.3. Key Feature 5

1.4. User Stories/ Scenario 5

2. Design 6

2.1. Required/Designed Features 6

2.2. Frontend Design 7

2.2.1. Sign In Page 8

2.2.2. Sign Up Page 8

2.2.3. Pick Interest Page 8

2.2.4. Feed Page 8

2.2.5. Interest Detail Page 8

2.2.6. New Feed Page 9

2.3. Backend Design 10

2.3.1. DB Design 10

2.3.2. API Design 11

3. Implementation and Environment 12

3.1. Development Environment 13

3.2. Deployed Environment 13

3.3. Backend Implementation 14

3.3.1. DB Implementation 14

3.3.2. API Implementation 17

3.4. Frontend Implementation 18

3.4.1. Core Module 18

3.4.2. Feed Module 19

3.4.3. Interest Module 20

3.4.4. Intro/Sign up Module 21
3.4.5. Profile Module 22
3.4.6. Shared/Search Module 22
4. Unit Test/ Load Test Result 24
4.1. Angular Unit Test: Karma 24
4.2. Django Unit Test: Django Test 25
4.3. Load Test: Locust 26
5. Conclusion 27
6. References 29

1. Introduction

1.1. Motivation for Service

As modern society progresses and becomes personalized accordingly, people increasingly feel that they are disconnected from others, and thus feel alienated. This is no exception in SNU campus. As more students try to take a minor or a double major, a common phenomenon inside classroom is that students are struggling alone.

Currently available "SNS" services such as KakaoTalk, Facebook, and SNULife are not solving the problem. KakaoTalk or Facebook only tends to connect those who already know each other, while SNULife only forms a vast community of anonymous users. Because of such user anonimity, SNULife's main functionality is collecting data and sharing idea. They do not meet the need of connecting physically nearby people, who more likely will share a common interest.

MoiSha is a new and friendly "SNS" service specialized for SNU students. Have you ever craved for a past exam paper? Have you ever wanted to discuss with a colleague about your latest interests? Moisha will open a window for that. You will be able to talk to your classmates more easily. You will also be able to form any kind of social meetings more freely. With MoiSha, you are not alone, as long as in SNU.

1.2. Target Customer/ Market Landscape

Our main customer base will be SNU students. Limiting the client base to SNU students is our first effort to gather people who might share interests. In other words, all users of MoiSha share a common interest of 'attending Seoul National University'. Through MoiSha service, these students can further narrow down their interests and meet those people around them.1.3.

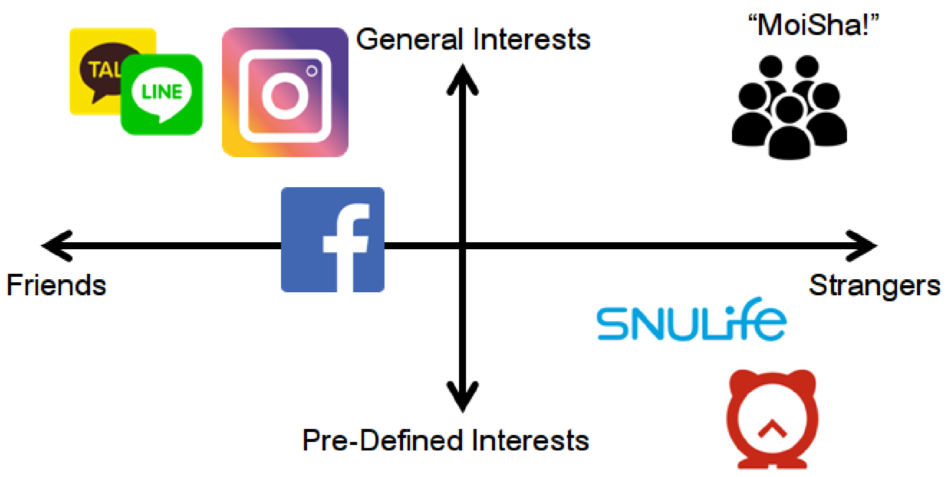


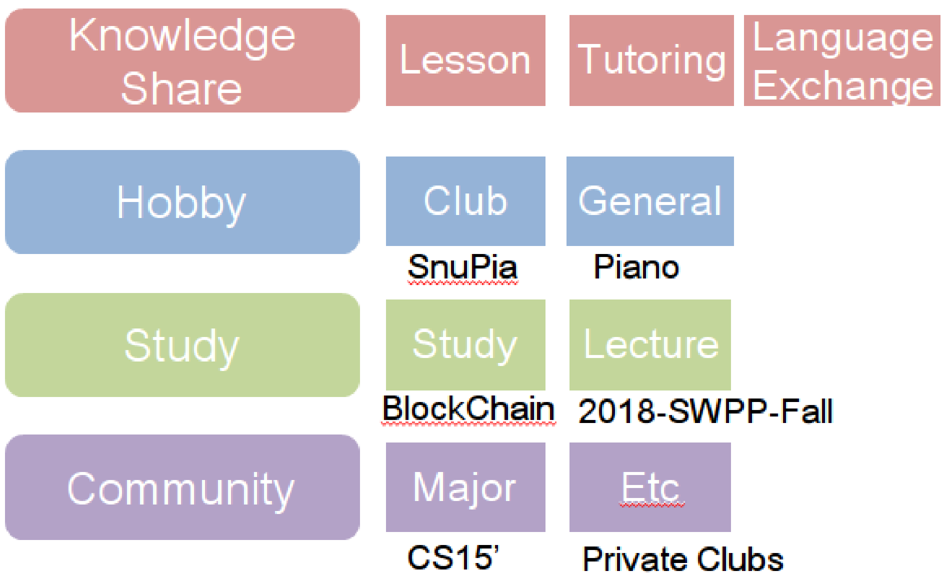
Figure 1: Market Landscape

Figure 1 shows competitive landscape *MoiSha* will face. Currently, there are two types of services in the Market. However, currently available "SNS" services are not solving the problem of the people sitting alone in classrooms. First Group, includes services such as *Kakao Talk, Line, Instagram, Facebook,* while offer pages/features for general interests, focus on build connection between people who already know each other. And the Second Group, includes services such as *Everytime* and *SNULife,* only forms a vast community of anonymous users. To be more specific, *SNULife* hardly allows people to meet outside the online community because of the user anonymity. Its main functionality is restricted to collecting data and sharing thoughts.

MoiSha is different from other services in a way that it connects people they did not know before while it provides pages(boards) about general(user-defined) interests. The idea of 'grouping by interest' will attract people without much burden. Our vision is to make SNU campus a more interactive place, a place where students can freely make new social communities.

1.3. Key Feature

MoiSha connects people based on 'interests'. User can think of any interest as keyword, search if an adequate one already exists, and create a new page if not. Some of the categories that MoiSha expect people to come up with are:



s

Figure 2: Examples of Possible Interests

Students benefit from using MoiSha in two ways. As materials accumulate for each page, users may later refer to them as an archive. At the same time, users can also get informed with any new posts for the 'interest' pages they are subscribed to.

1.4. User Stories/ Scenario

a) New user interested in "cooking"

Given a user is interested in cooking.

When the user sign up and search for interest with keyword “Cooking”,

If there was already interest called cooking,

Then the user can join the interest and find other users that also are interested in cooking.

If not the user can create new interest named cooking,

Then the user will become a manager of the interest and others will join.

b) A user want to make a question about class to former student

Given a user want to make a question about class and want to find students who took the class at last semester.

When he/she join the interest “2018 Spring SWPP” and write an article with tag “QNA”,

Then he/she will be able to get answers since the article will be displayed on the main feeds to former students.

2. Design

This section is about how we initially designed structure of our services. For implementation of our service, we planned to use *Angular* framework for frontend of our service and *Django* Framework for backend of our service. And also, since we planned to use *PostgreSQL*, we designed our DB schema using terms of RDB schema which will be explained later.

2.1. Required/Designed Features

Most of features were designed at early stage of project, and there were two core components; the main feed and the interest detail. Table 1 shows required features of design stage.

Table 1: Required Features at the Design Stage

Feature	Description	Actual Implementation (Y/N)
Sign in/ Sign up	Users should be able to sign in/up at the intro page.	Y
Pick interests	Newly signed-up users should be able to pick interests at first.(Will be explained at the frontend design)	N
Main feed	New articles to interests a user joined should be collected and displayed at the main feed	Y
Side bar	Sidebar should list all interests a user joined.	Y
Navigation bar	Navigation bar should consist of home button, search bar, profile button, and logout button.	Y
Search interest	Users should be able to search interests with keyword.	Y

Filter with tags	Users should be able to filter the articles displayed at the main feed.	Y
Different article type	Different type of articles should be displayed in different ways. I.E. *Event* type article focuses on the time and the location.	Y
Write article	Users should be able to write new article.	Y
Create interest	Users should be able to create new interests.	Y
Interest recommendation	It should be able to recommend new interests which a user might interested in.	Y
Edit/Delete article	Users should be able to edit and delete article only when he/she wrote them.	Y
Write/Edit/Delete comments	Users should be able to write comments. And they also should be able to edit/delete comments only when he/she wrote them.	Y
View/Modify user information	Users should be able to view and modify user information.	Y

2.2. Frontend Design

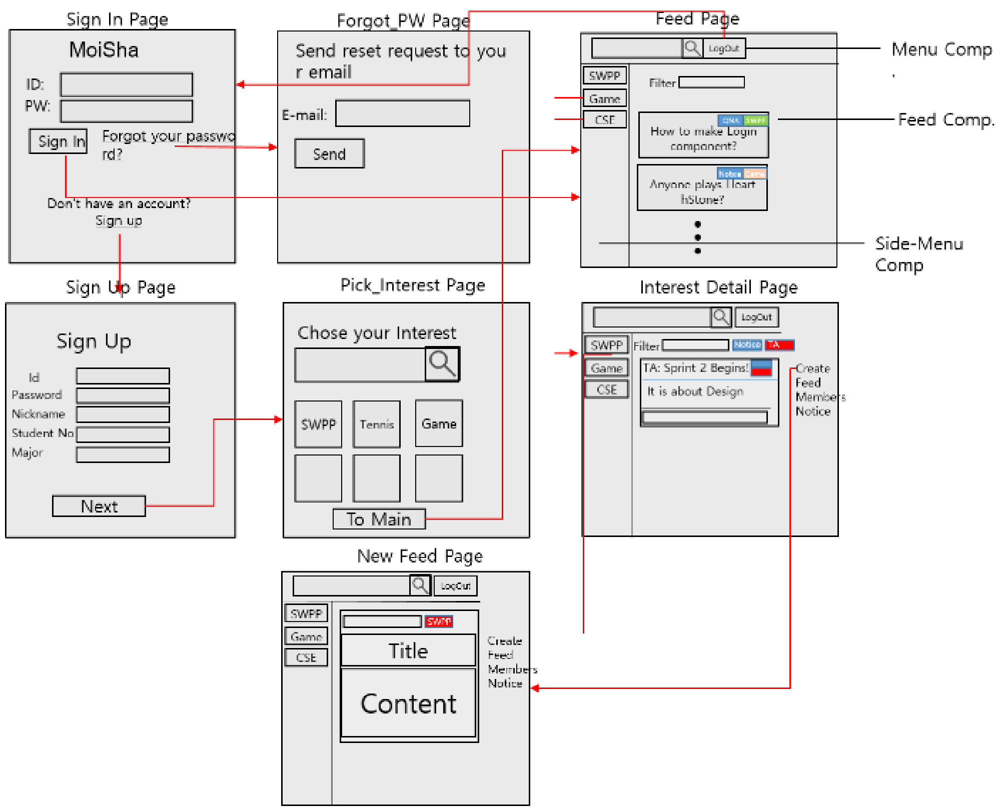


Figure 3: UI Flow Diagram of Design Phase

Figure 3 shows the UI flow diagram of early stage of project. Since we decided to use *Angular6* framework to implement frontend of our service, each page was designed with angular component. Some of them is described in the figure 3. For example, Feed Page includes 3 components; Menu Component, Feed Component, and Side-Menu Component.

2.2.1. Sign In Page

- This page act as front page to our Service.
- It has two user-input fields (email-formatted id and password), and three buttons (sign_in_btn, forgot_pw_btn, signup_btn)
- sign_in_btn: check the validity of user inputs (id and pw), and if it confirms, redirects to Feed Page.
- forgot_pw_btn: this button is for users who forgot their password. It redirects to Forgot_PW Page.
- signup_btn: it redirects to Sign Up Page.

2.2.2. Sign Up Page

- Our Member(User) Model has 5 attributes: id(email-formatted), password, nickname, student Number, major.
- There is validation and if necessary, check-duplicate process for each fields.
- When all fields are checked and push ‘next’ button, new user entity is created. Then it redirects to Pick Interest Page.

2.2.3. Pick Interest Page

This page is one time accessible for newly signed-up users.

- Users can search and pick(join) interests they are interested in.
- When To Main button is pushed, it redirects to Feed page.

2.2.4. Feed Page

Feed Page is one of our key-feature pages. It consists of three components: Menu Comp, Feed Comp, and Side-Menu Comp.

- Menu Component has search bar for searching new interest, and logout button. If a user pushes the logout button, it redirects to Sign In Page.
- In Side Menu Component, there will be interest buttons for each interest user joined.
- Feed Component has filter feature: user can filter the feeds according to their tags
- New feeds for interests which user joined will be listed in feed component. Upon clicked, there will be modal-popup of feed-detail.

2.2.5. Interest Detail Page

Interest Detail Page is also one of our key-feature pages. It consists of three components: Menu Comp, Interest-Detail Comp, Side-Menu Comp.

- Interest Detail Component has similar features with Feed Page.

- The difference for this stage of design is two.
- It contains only feeds from current detail-interest, and there will be drop-down menu on the right corner of screen.
- When drop-down menu is clicked, feed contents will be changed accordingly.
- Users can create new feed from detail page.

2.2.6. New Feed Page

Users can create new feeds in this page.

- It has two text-input fields: title and content.
- Users can also search/create/put tags to feeds being created.

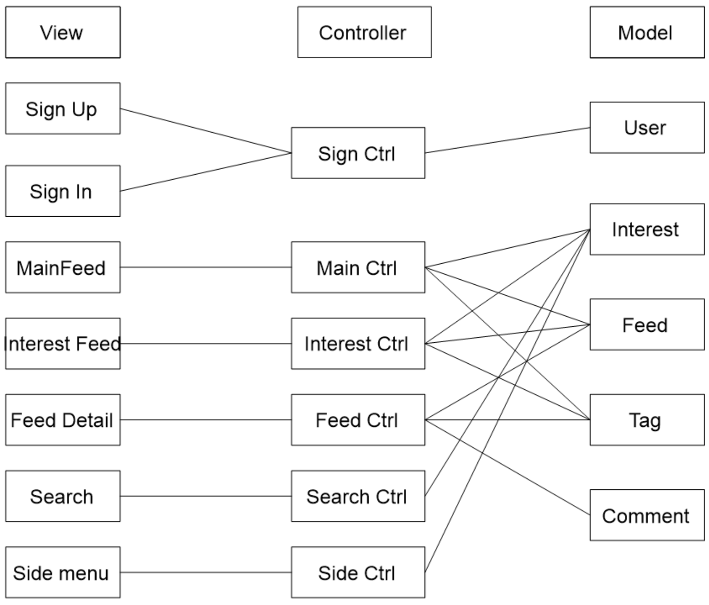


Figure 4: MVC Pattern Design

Figure 4 shows the relations between MVC. Left side is the View part, middle is the Controller part and right side is the Model part. Interaction between Controller and Model will be handled by *Django ORM*. All data will be delivered by JSON format. Interaction between View and Controller will be handle by both *Angular* and *Django*.

2.3. Backend Design

2.3.1. DB Design

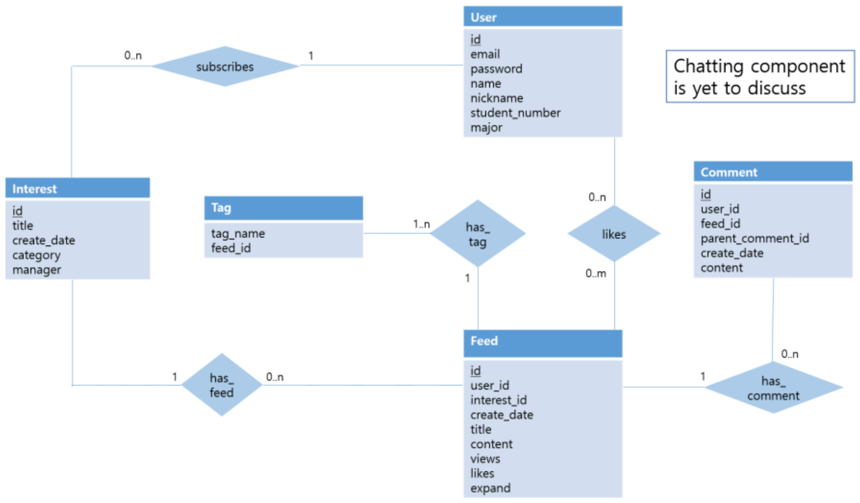


Figure 5: ER Diagram at the Design Stage

Figure 5 shows ER Diagram of Design Stage. At the Design Stage, we concluded 5 table would be needed; Interest, Tag, User, Feed, Comment.

Rectangle components represent entity sets and rhombus components represent relationship sets. Entity attributes are listed inside the entity rectangle, and underlined attribute is the primary key. A relationship set also form a table in the database schema. Some are decided to implement based on the nature of our service system, and some are implemented to provide some utility in manipulating the data.

Lines in between the sets have a number, which indicates the cardinality constraint (min to max). For example, a ‘user’ can ‘subscribe’ to many ‘interests’ and many ‘users ’can express ‘likes’ on many ‘feeds’.

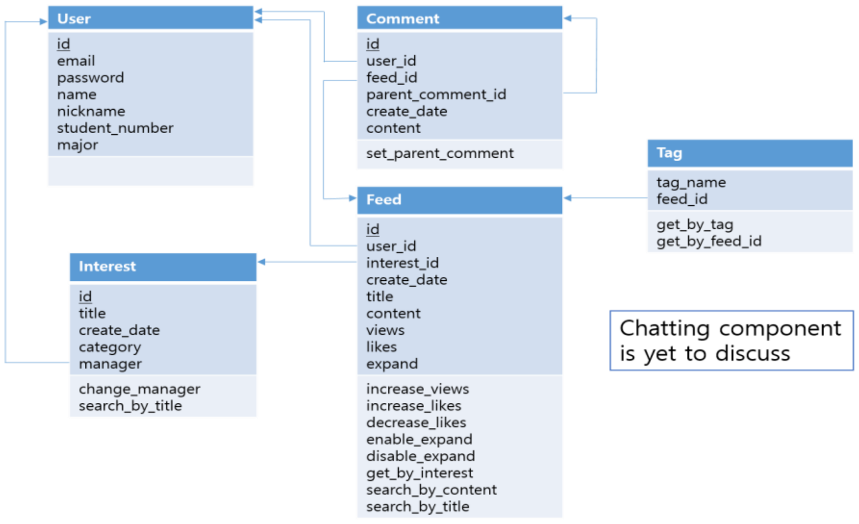


Figure 6: Relation Schema at the Design Stage

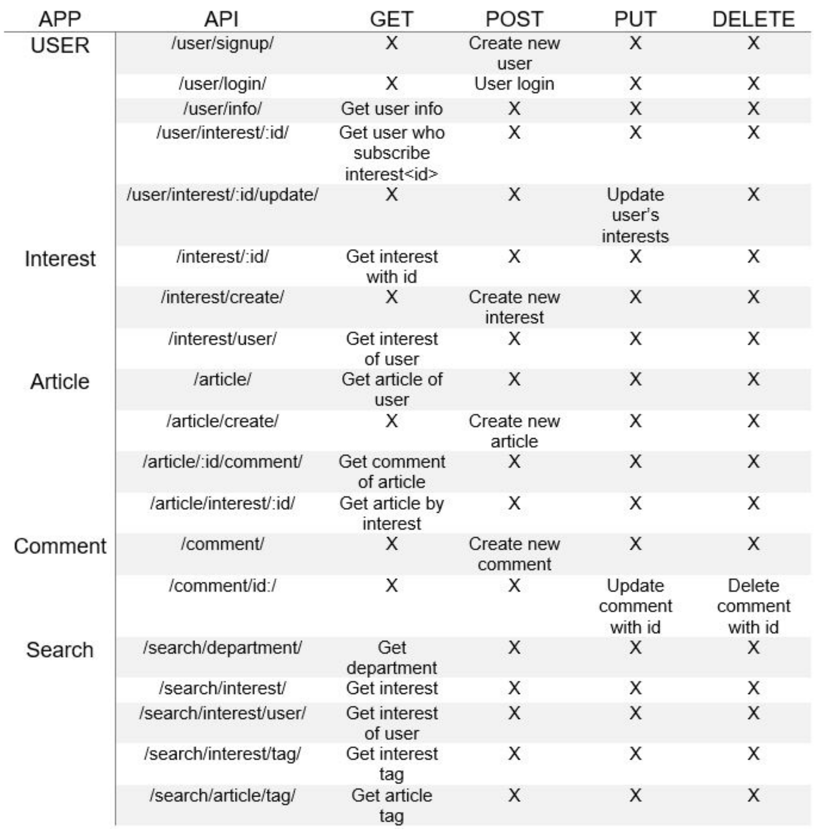
Figure 6 shows relation schema at the design stage. Relation schema is the representation of our *Django* model. In the *Angular* system, each of these models will match to a service that provides methods to manipulate. Underlined attributes are primary keys, and arrow represents foreign key constraints. Because *Django* saves whole object for foreign key, the arrow points to the object, not the primary key of an object.

Bottom row of each schema represents the functions needed to manipulate the data. For example, in order to implement a search bar, interest schema must provide a specific search function based on the requirement set.

2.3.2. API Design

For backend Design, we use models from above MVC architecture and RESTful API. Table 2 shows specification of RESTful API at the design stage.

Table 2: API Design



APP	API	GET	POST	PUT	DELETE
USER	/user/signup/	X	Create new user	X	X
	/user/login/	X	User login	X	X
	/user/info/	Get user info	X	X	X
	/user/interest/:id/	Get user who subscribe interest<id>	X	X	X
	/user/interest/:id/update/	X	X	Update user's interests	X
Interest	/interest/:id/	Get interest with id	X	X	X
	/interest/create/	X	Create new interest	X	X
	/interest/user/	Get interest of user	X	X	X
Article	/article/	Get article of user	X	X	X
	/article/create/	X	Create new article	X	X
	/article/:id/comment/	Get comment of article	X	X	X
	/article/interest/:id/	Get article by interest	X	X	X
Comment	/comment/	X	Create new comment	X	X
	/comment/id:/	X	X	Update comment with id	Delete comment with id
Search	/search/department/	Get department	X	X	X
	/search/interest/	Get interest	X	X	X
	/search/interest/user/	Get interest of user	X	X	X
	/search/interest/tag/	Get interest tag	X	X	X
	/search/article/tag/	Get article tag	X	X	X

3. Implementation and Environment

There were 6 each-2-week-long sprints total for our projects. We grouped two sprints into one development cycles. We concluded that in the first stage of development, it was better to implement the overall basic features and the complex features later on. In the first cycle which included sprint 1 and 2, the market landscape analysis, target customer analysis, and design were conducted. It then started actual development from Sprint 3. In the second cycle, which included sprint 3 and 4, the overall implementation of all pages is aimed at. And complex functions such as recommendation of interest and Feed Scrolling are all implemented in the third cycle, which included 5 and 6. Each Sprint and its contents are as follows:

Table 3: Project Timeline

Cycle	Sprint	Work
Design	Sprint 1	Decision of Service Feature, Proposal
	Sprint 2	Service Design
Basic Skeleton	Sprint 3 Sprint 4	Feature: Sign in, Sign up, Main Feed Page, Interest Detail Page, Interest Feed Page.(include backend APIs and unit test)
Advanced Feature	Sprint 5 Sprint 6	Feature: Interest Recommendation, User Profile Page, Interest/Feed scroll-paging. Deployment, Load Test, Optimization, CI Set-up(include backend APIs and unit test)

3.1. Development Environment

As mentioned before, we used *django* framework for our backend implementation and *Angular* framework for our frontend implementation. Moreover, since search with keyword (interest search/ auto complete search of department and tags) is included to our service, we used *elasticsearch* to implemented them. Figure 7 shows our development structure.

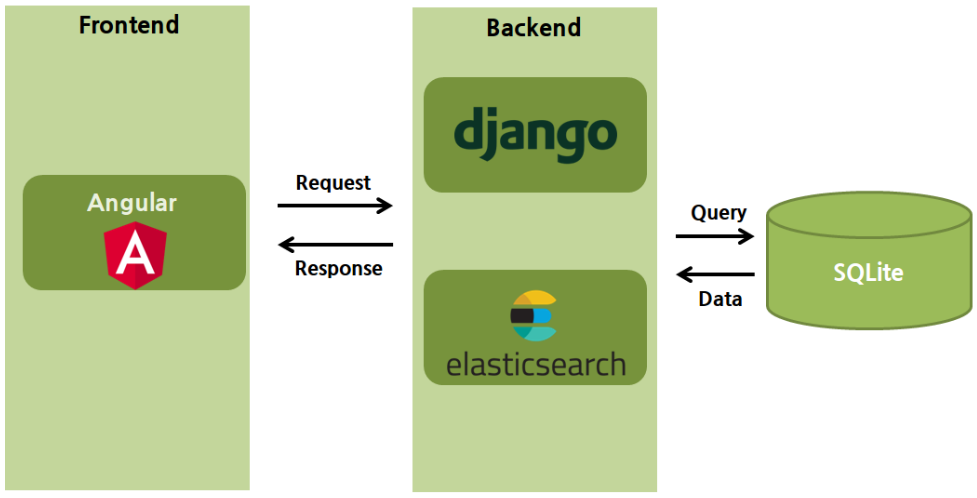


Figure 7: Development Environment

3.2. Deployed Environment

To deploy, we used a AWS EC2 instance as a server and a AWS RDS instance as a DB. At the server side, we used *Nginx* as a front static-serving man. If dynamic contents are needed, *Nginx* passes those request to *uWSGI* in which *django* app is running. Figure 8 shows our deployed structure.

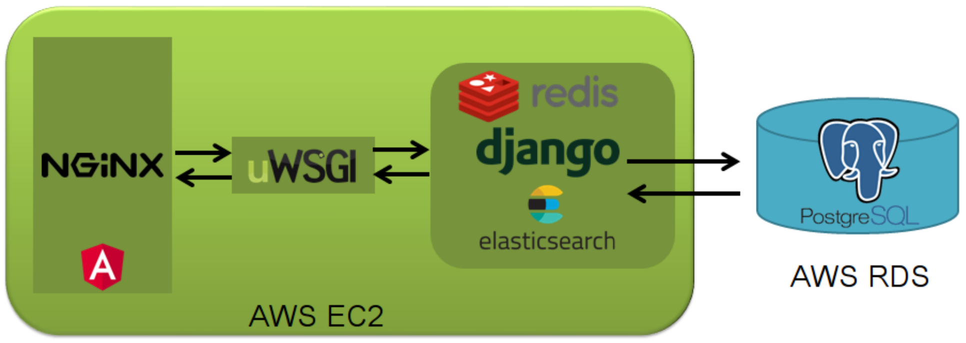


Figure 8: Deployed Environment

To optimize application, we also used *redis* as in-memory caching engine. Even if *django's* native *SQLite* is easy to configure and use, *SQLite* can't process multiple concurrent requests well. Because of its weakness, we used *PostgreSQL* as a DBMS.

3.3. Backend Implementation

3.3.1. DB Implementation

As we implement our service, we decided some of DB Design should change for additional features or implementation issues. They will be detailed in sections of each model(Table).

- Tag Color

Tag Color is newly created model. As implementation stage proceeds, it was needed to classify tags with distinct color. So, new model "Tag Color" is added. It has three field; id, name, RGB. RGB is formatted as #RRGGBB.

Table 4: Tag Color

Column/Field Name	Filed Type	Description
Id(pk)	Integer Field	Auto-created id field.
Name	Char Field	Name of the color. Max length is 20.
RGB	Char Field	RGB Value of the color. Max length is 7.

- Article Tag

Article Tag has one change. Originally, it had only two fields; id and name. But to set distinct colors to each tags when rendered, "color" field is added.

Table 5: Article Tag

Column/Field Name	Filed Type	Description
Id(pk)	Integer Field	Auto-created id field.
Name	Char Field	Name of the tag. Max length is 20.
Color	Foreign Key	Id of tag color.

- Article

Since we decided not to implement like or view counts feature to our service, for Article Model, likes, views, and expand fields are deleted.

Table 6: Article

Column/Field Name	Type	Description
Id(pk)	Integer Field	Auto-created id field.
Title	Char Field	Title of an article. Max length is 80.
Author	Foreign Key	Author of an article.
Content	Text Field	Content of an article.
CreatedDate	DateTime Field	Date/time when an article is created.
Interest	Foreign Key	Id of interest to which an article is posted.
Tags	ManyToMany Field	Article tags that are tagged to an article.

- Interest Tag

It has same fields and function as Article Tags.

Table 7: Interest Tag

Column/Field Name	Type	Description
Id(pk)	Integer Field	Auto-created id field.
Name	Char Field	Name of the tag. Max length is 20.
Color	Foreign Key	Id of tag color.

- Interest

Model for each interests users create.

Table 8: Interest

Column/Field Name	Type	Description
Id(pk)	Integer Field	Auto-created id field.
Name	Char Field	Name of an interest. Max length is 20.
Detail	Char Field	Simple introduce message of an interest. Max length is 300.
CreatedDate	DateTime Field	Date/time when an interest is created.
Tags	ManyToMany Field	Interest tags that are tagged to an interest
PhotoURL	Char Field	URI of interest deatil photo.

- InterestJaccard

It is newly create model used to implement interest recommendation feature. Jaccard score is defined as Equation 1 where A, B is set of people who joined first and second interest. For every pairs of interest, batch process is implemented using django background process. At present, the batch process runs every 15 minutes.

$$\mathrm{J(A, B)} = \frac{|\mathrm{A} \cap B|}{|A \cup B\,|}$$

Equation 1: Jaccard Score

Table 9: Interest Jaccard

Column/Field Name	Type	Description
Id(pk)	Integer Field	Auto-created id field.
First	Foreign Key	Id of First Interest. For recommendation, first interest is always the one user joined.
Second	Foreign Key	Id of Second Interest.
Score	Float Field	Jaccard Score between first and second interest.

- Comment

There were no changes from initial design for comment model.

Table 10: Comment

Column/Field Name	Type	Description
Id(pk)	Integer Field	Auto-created Id field.
Author	Foreign Key	Author of a comment
Article	Foreign Key	Article to which a comment is written.
Comment	Foreign Key	Comment to which a comment is written. Non blank only when a comment is written to a comment.
Content	Text Field	Content of a comment

CreatedDate	DateTime Field	Date/Time when a comment is created.
Deleted	Boolean Field	True if a comment is deleted

- College

It has only two fields: id and name.

Column/Field Name	Type	Description
Id(pk)	Integer Field	Auto-created id field.
Name	Char Field	Name of a College. Max length is 128.

- Department

It has three fields; id, name, and college.

Column/Field Name	Type	Description
Id(pk)	Integer Field	Auto-created id field.
Name	Char Field	Name of a department. Max length is 128
College	Foreign Key	College to which a department belongs.

- User

User Model has one change: “interests” field is added.

Column/Field Name	Type	Description
Id(pk)	Integer Field	Auto-created id field.
Email	Char Field	Email of a user. It acts as a username.
Name	Char Field	Name of a user.
studentId	Integer Field	Student ID of a user. It is unique key.
Major	Foreign Key	Major of a user.
NickName	Char Field	Nickname of a user. It is unique key
Interests	ManyToMany Field	List of interests a user joined.

Changed ER Diagram is described at figure 9. Since frontend implementation will be discussed later section, relation diagram will not be covered here.

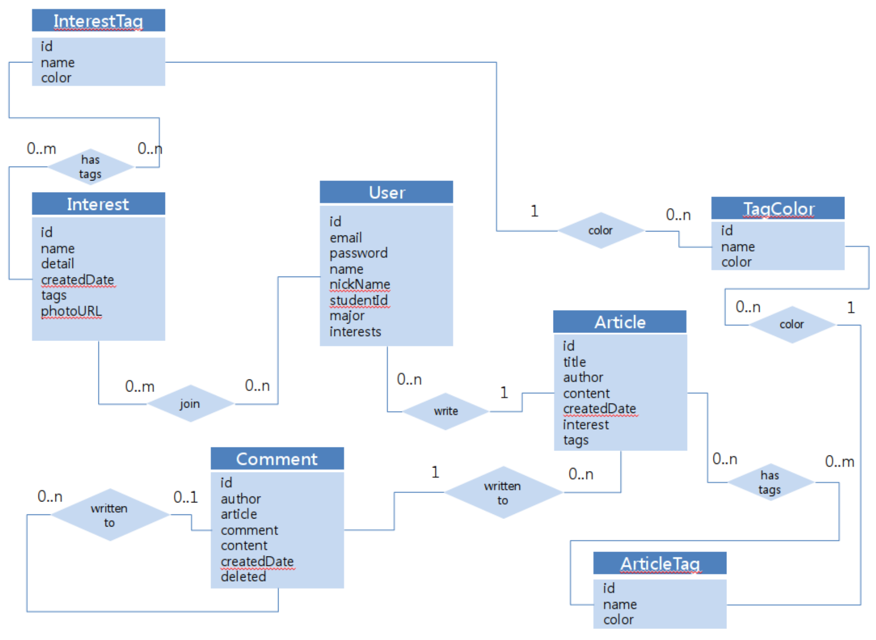


Figure 9: ER Diagram for Implementation

3.3.2. API Implementation

We implemented total 33 APIs(*Django* views) at the backend. All of them were implemented using *django restframework*, and 6 includes django *haystack/elasticsearch* stack. Table 11 shows implemented APIs in detail.

Table 11: Implemted APIs

APP	API	GET	POST	PUT	DELETE
User	api/user/signup/	X	Create new user	X	X
	api/user/login/	X	User login	X	X
	api/user/info/	Get user info	X	X	X
	/api/user/interest/:id/	Get user who joinned interest<id>	X	X	X
	api/user/interest/:id/update/	X	X	Update user’s joined status	X
	api/user/check/	Check duplicate email/nickname/student Id	X	X	X
	api/user/modify/	X	X	Modify user information	X
Interest	/api/interest/:id/	Get interest <id>	X	X	X
	/api/interest/create/	X	Create new interest	X	X
	/api/interest/user/	Get joined users of an interest	X	X	X
	/api/interest/tags/	Get all interest tags.	X	X	X
	/api/interest/recommend/	Get recommended	X	X	X

		interest to an user			
	/api/interest/recommend/tag/	Get recommended interest to an user and filter by tag	X	X	X
	/api/interest/recommend/:id/	Get recommended interest to an interest	X	X	X
Article	/api/article/	Get articles by user	X	X	X
	/api/article/create/	X	Create new article	X	X
	/api/article/:id/comment/	Get comments of article	X	X	X
	/api/article/interest/:id/	Get articles by interest	X	X	X
	/api/article/interest/:id/tags/	Get articles by interest and filter by tag			
	/api/article/tags/	Get articles by user and filter by tag			
	/api/article/:id/	Get detail of an article			
	/api/article/:id/edit/	X	X	Edit an article	X
	/api/article/:id/delete/	X	X	X	Delete an article
	/api/article/tag/	Get all article tags	X	X	X
Comment	/api/comment/	X	Create a comment	X	X
	/api/comment/:id/	X	X	Edit a comment	Delete a comment
Search	/api/search/department/	Get department (auto-complete)	X	X	X
	/api/search/interest/	Search interest with keyword	X	X	X
	/api/search/interest/user/	Search interest by user (auto-complete)	X	X	X
	/api/search/interesttag/	Search interest tags(auto-complete)	X	X	X
	/api/search/articletag/	Search article tags(auto-complete)	X	X	X
	/api/search/interest/tag/	Search interest with keyword and filter by tags	X	X	X

3.4. Frontend Implementation

To implemented frontend of *Moisha*, we first divided its feature to four 8 modules; core, feed, interest, intro, profile, search, shared, signup modules.

3.4.1. Core Module

In core module, all of services are declared. The reason why all of services should be declared in one module is to use http interceptor. Since our service use token authorization, all of API calls should include header “Authorization: token {{token key}}”. To append authorization header, it is required to intercept all API requests and append authorization header to them. It is implemented as Custom API interceptor extends HttpInterceptor.

```
@Injectable()
export class ApiUrlInterceptor implements HttpInterceptor {
  intercept(
    req: HttpRequest<any>,
    next: HttpHandler
  ): Observable<HttpEvent<any>> {
    var request
    if(req.url !== '/user/login/' && req.url !== '/user/signup/' && !(new RegExp('^\\/search\\/department').test(req.url)))
      request = req.clone({ url: environment.apiURL + req.url,
    headers: req.headers.set('Authorization', 'token ' + localStorage.getItem(TOKEN_KEY))});
    else
      request = req.clone({ url: environment.apiURL + req.url});
    return next.handle(request);
  }
}
```

Except login/signup for which authorization token should not be included, custom authorization header is added.

3.4.2. Feed Module

Feed module includes main feed page, article detail page, and article create/edit page. To implement those pages, 11 components are implemented.

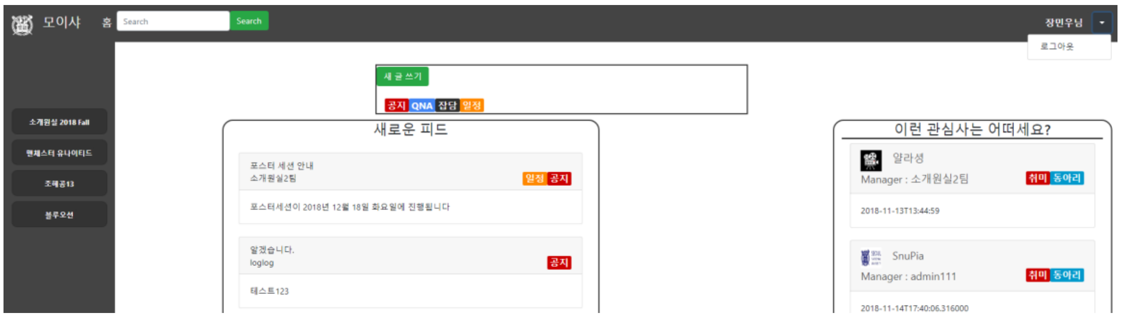
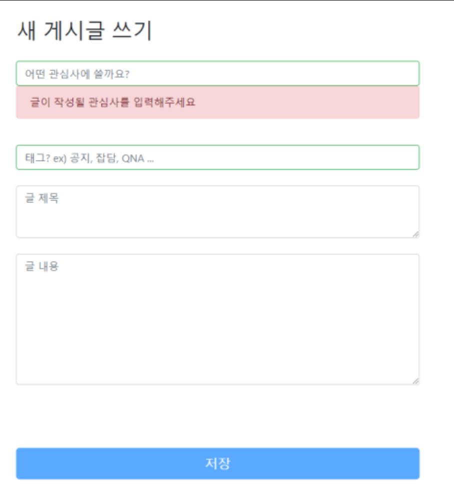
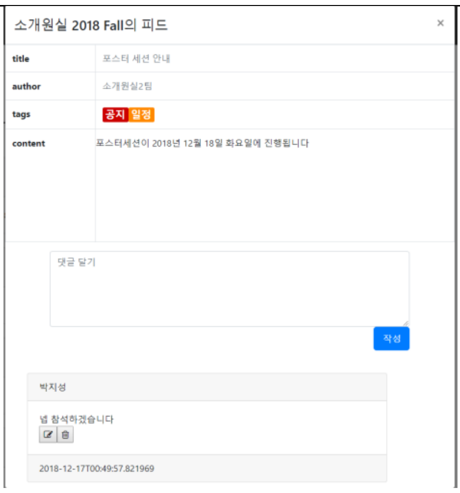


Figure 10: Main Feed Page(/)



새 게시글 쓰기
어떤 관심사에 쓸까요?
글이 작성될 관심사를 입력해주세요
태그? ex) 공지, 잡담, QNA ...
글 제목
글 내용
저장

Figure 11: Article Create Page(/create)



소개원실 2018 Fall의 피드
×
title
포스터 세션 안내
author
소개원실2팀
tags
공지
일정
content
포스터세션이 2018년 12월 18일 화요일에 진행됩니다
댓글 달기
작성
박지성
넵 참석하겠습니다
2018-12-17T00:49:57.821969

Figure 12: Article Detail Page

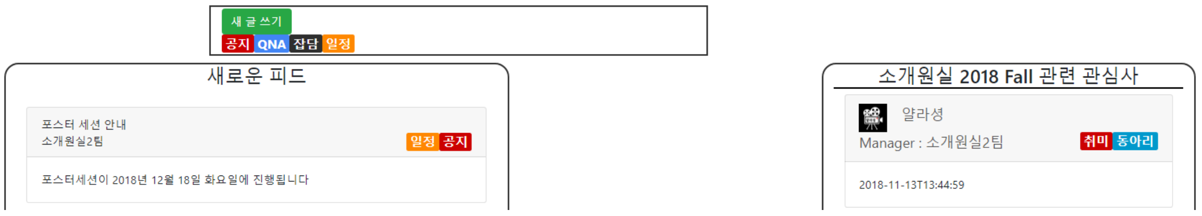
Table 12: Feed Module

Component	Description
Article	A preview article component. It is included in feed-list component. When clicked, article-detail component will be rendered as custom modal.
Article-create	Article create page. It includes article-form.
Article-detail	Article detail component. all the comments to the article is listed. when

	clicked outside of modal, it dismisses.
Article-edit	Article edit page. It includes article-edit-form.
Article-edit-form	Custom form-group for article editing.
Article-form	Custom form-group for article creation.
Comment	A comment component. when clicked, users can write comment to the comment.
Feed	Main feed page. It includes Feed-List Component.
Feed-list	Feed list component. Users can filter feeds by article tags. When scrolled down, more feeds are fetched.
Home	Home Component. It has router-outlet.
Write-reply	Write reply component. It has text input for comment content.

3.4.3. Interest Module

Interest Module includes interest detail page, interest people detail page, and interest feed page. To implement those pages, 11 components are implemented.



새로운 피드

포스터 세션 안내
소개원실2팀
일정 공지

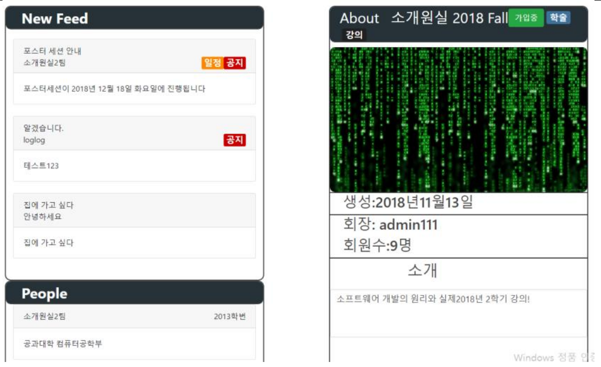
포스터세션이 2018년 12월 18일 화요일에 진행됩니다

소개원실 2018 Fall 관련 관심사

얄라셩
Manager : 소개원실2팀
취미 동아리

2018-11-13T13:44:59

Figure 13; Interest Feed Page(/interest/:id/feed)



포스터 세션 안내
소개원실2팀
일정 공지

포스터세션이 2018년 12월 18일 화요일에 진행됩니다

알겠습니다.
loglog
공지

테스트123

집에 가고 싶다
안녕하세요

집에 가고 싶다

소개원실2팀 2013학번

공과대학 컴퓨터공학부

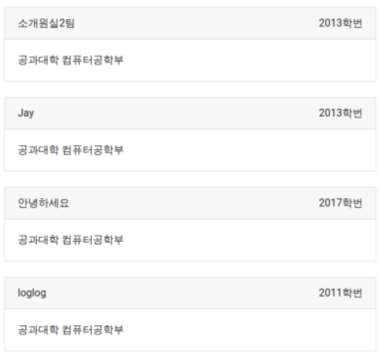
About 소개원실 2018 Fall

생성:2018년11월13일
회장: admin111
회원수:9명

소개

소프트웨어 개발의 원리와 실제2018년 2학기 강의!

Figure 14: Interest Detail Page(/interest/:id)



소개원실2팀 2013학번

공과대학 컴퓨터공학부

Jay 2013학번

공과대학 컴퓨터공학부

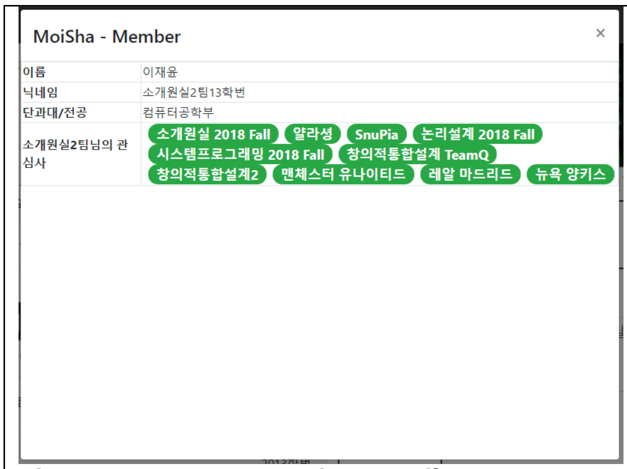
안녕하세요 2017학번

공과대학 컴퓨터공학부

loglog 2011학번

공과대학 컴퓨터공학부

Figure 15: Interest People Page(/interest/:id/people)


MoiSha - Member

이름	이재윤
닉네임	소개원실2팀13학번
단과대/전공	컴퓨터공학부
소개원실2팀님의 관심사	소개원실 2018 Fall, 얄라셩, SnuPia, 논리설계 2018 Fall, 시스템프로그래밍 2018 Fall, 창의적통합설계 TeamQ, 창의적통합설계2, 맨체스터 유나이티드, 레알 마드리드, 뉴욕 양키스

Figure 16: Interest Member Detail Page

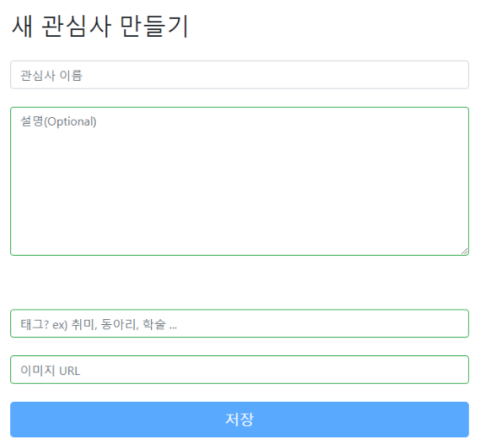

새 관심사 만들기

Figure 17: Interest Create Page(/interest/create)

Table 13: Interest Module

Component	Description
Interest-create	Interest create page. It includes interest-form component.
Interest-form	Custom form-group for interest creation.
Interest-feed	Interest feed page. It includes feed list component.
Interest-detail	Interest detail page. it includes interest people list component with only three member, feed list component with only three articles, interest info component.
Interest-home	Home Component. It has router-outlet
Interest-info	It has interest photo, introduce message, manager of the interest, and join button.
Interest-people	People preview component. When clicked, people-detail component will be rendered as custom modal.
Interest-people-detail	People Detail component. It has information of member; nickname, student id(only year of admission), all interests user joined.
Interest-people-list	People list component. It has list of interest-people component. When scrolled down, more people are fetched.
Interest-people-home	Interest people page. It includes interest-people-list component.

3.4.4. Intro/Sign up Module

Intro module has only one page: sign in page. Every not-permitted or not-defined url access will be redirected to Sign in page. When signed in, every not-defined url access will be redirected to Main feed page.

Sign up module also has only one page: sign up page. After successfully signed-up, users will be redirected to main feed page.

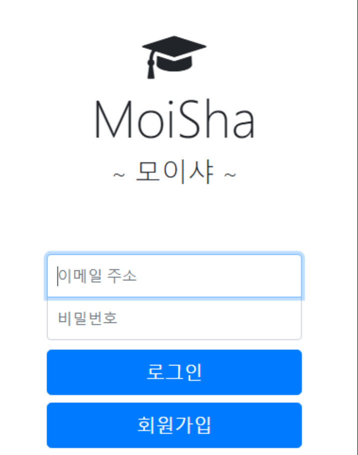


Figure 18: Sign In Page

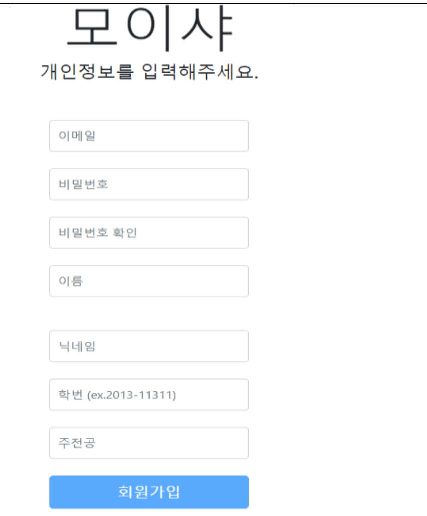
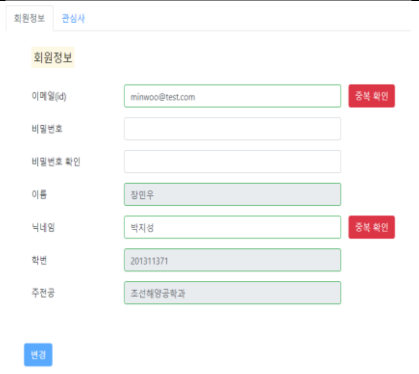


Figure 19: Sign Up Page

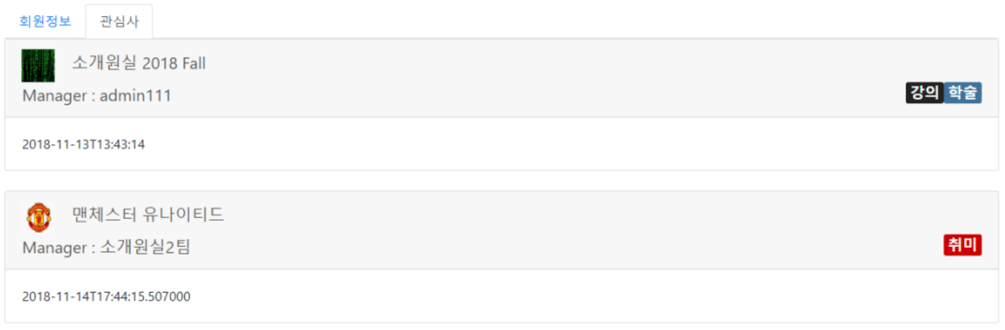


이메일(id)	minwoo@test.com
비밀번호	
비밀번호 확인	
이름	장민우
닉네임	박지성
학번	201311371
주전공	조선해양공학과

Figure 20: Profile Page (Info Tab)

3.4.5. Profile Module

Profile Module has one page and three components. Users can change their email and nickname at the info tab. And also users can check every interests he/she joined.



소개원실 2018 Fall
Manager : admin111
2018-11-13T13:43:14

맨체스터 유나이티드
Manager : 소개원실2팀
2018-11-14T17:44:15.507000

Figure 21: Profile Page(Interest Tab)

Table 14: Profile Module

Component	Description
Profile	It has two tabs; info and interest tabs. Each tab is rendered with profile-interest or profile-user info component.
Profile-interest	All interests user joined is listed.
Profile-Userinfo	User information such as email, name, nickname is listed. And user can modify their user information.

3.4.6. Shared/Search Module

Shared Module imports/declares general purpose component or any common module that more than two other module need to import. Common module includes modules like RouterModule, CommonModule, FormsModule. Feed Module has to import interest list component and Interest Module has to import feed list component. When Interest List Component is declared in the Interest Module, there is circular dependency problem (Feed Module needs components from Interest Module, and Interest Module needs Components from Feed Module). To avoid circular dependency, interest-item and interest-list component is declared in shared module.

Table 15: Shared Module

Component	Description
Interest-item	Interest Preview Item. When clicked, users are redirected to interest detail page.
Interest-list	It has multiple interest item and list them in ng for block.
Loading	Loading-indicating component when a page is waiting async calls.
Navbar	Navigation bar component.
Side-bar	Sidebar component.

Search Module has one page: search result page. At any page of Mosiha, users can search interests with keyword at the navigation bar. When search button is clicked, they will be redirected to search page. And users can create new interest in the search page.

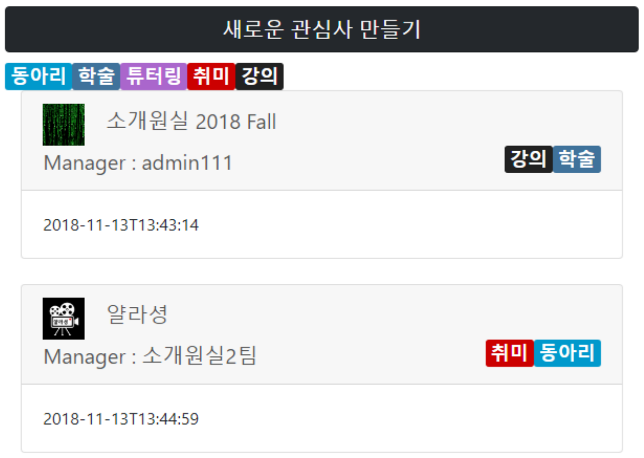


Figure 22: Search Page

Resulting URL-Page mapping is described in table 16.

Table 16: URL-Page Mapping of Moisha

URL	Page. Component
/intro	Sign in Page
/signup	Sign Up Page
/	Main Feed Page
/create	Create Article Page
/:id/edit	Edit Article Page
/interest/:id	Interest Detail Page
/interest/:id/feed	Interest Feed Page
/interest/:id/people	Interest People Page
/search	Search Page
/profile	User Profile Page
**(else)	Sign in Page or Main Feed Page.

As a result, all implemented features are:

- Sign In / Sign Up

- Main Feed, Tag Filtering
- Recommendation based on user’s interest: Jaccard Score, Batch Processing
- Article Write/Edit/Delete
- Article Detail View: Comment Write/Edit/Delete
- Search Interest using *elasticsearch*
- Create New Interest
- Check/Modify User Profile/Information
- Interest Detail
- Interest Feed/People

4. Unit Test/ Load Test Result

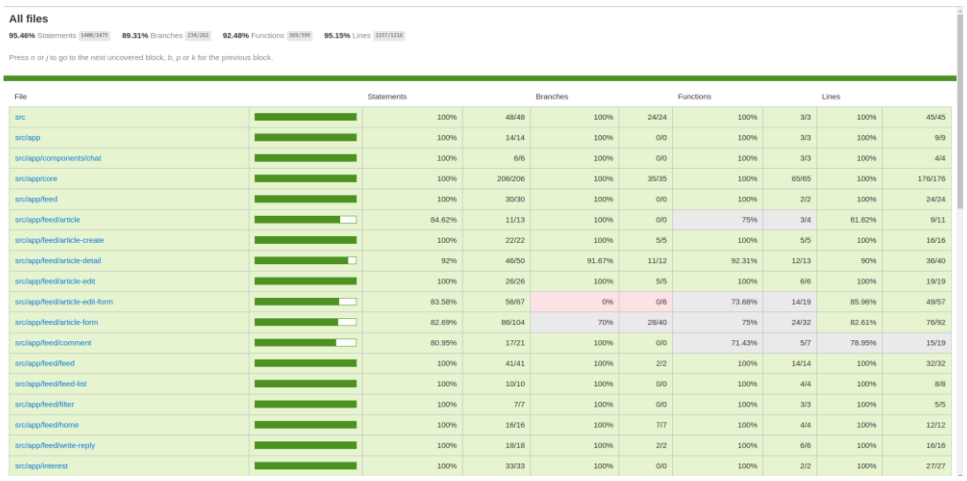
To implement our Service, we tried to follow Test-Driven-Development. Thus, we configured *Travis CI* to run unit test whenever commit to master branch take places.

```
sudo: required
services:
    elasticsearch
before_script:
    - sleep 10
before_install:
    curl -O
https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/deb/elasticsearch/2.4.1/elasticsearch-2.4.1.deb
&&
    sudo dpkg -i --force-confnew elasticsearch-2.4.1.deb && sudo service elasticsearch restart
matrix:
    include:
      - language: node_js
        node_js:
          - "8"
        before_install:
          - cd Moisha-Front
        install:
          - npm install -g @angular/cli
          - npm install
        script:
          - npm run testCI

      - language: python
        python: 3.6
        before_install:
          - cd Moisha_Back
        install:
          - pip install -r requirements.txt
        script:
          - python manage.py test
```

4.1. Angular Unit Test: Karma

We used *Karma* to run unit test for *angular* framework. Figure 23 shows result of *Karma* Unit Test. There are total 169 specs, and branch coverage is 89.31% at present. Some components have relatively low coverage than the other. For example, article edit form has only 83.58% coverage for statement and 0% for branches. This imbalance is largely due to Angular's characteristic.



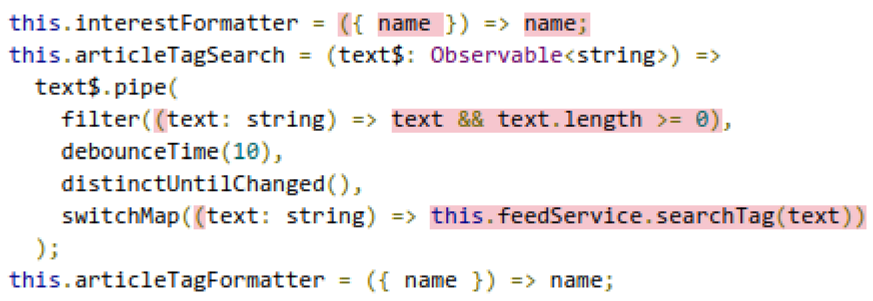
All files

95.46% Statements 1408/1475 **89.31%** Branches 234/262 **92.48%** Functions 369/399 **95.15%** Lines 1157/1216

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements		Branches		Functions		Lines	
src		100%	48/48	100%	24/24	100%	3/3	100%	45/45
src/app		100%	14/14	100%	0/0	100%	3/3	100%	9/9
src/app/components/chat		100%	6/6	100%	0/0	100%	3/3	100%	4/4
src/app/core		100%	206/206	100%	35/35	100%	65/65	100%	176/176
src/app/feed		100%	30/30	100%	0/0	100%	2/2	100%	24/24
src/app/feed/article		84.62%	11/13	100%	0/0	75%	3/4	81.82%	9/11
src/app/feed/article-create		100%	22/22	100%	5/5	100%	5/5	100%	16/16
src/app/feed/article-detail		92%	46/50	91.67%	11/12	92.31%	12/13	90%	36/40
src/app/feed/article-edit		100%	26/26	100%	5/5	100%	6/6	100%	19/19
src/app/feed/article-edit-form		83.58%	56/67	0%	0/6	73.68%	14/19	85.96%	49/57
src/app/feed/article-form		82.69%	86/104	70%	28/40	75%	24/32	82.61%	76/92
src/app/feed/comment		80.95%	17/21	100%	0/0	71.43%	5/7	78.95%	15/19
src/app/feed/feed		100%	41/41	100%	2/2	100%	14/14	100%	32/32
src/app/feed/feed-list		100%	10/10	100%	0/0	100%	4/4	100%	8/8
src/app/feed/filter		100%	7/7	100%	0/0	100%	3/3	100%	5/5
src/app/feed/home		100%	16/16	100%	7/7	100%	4/4	100%	12/12
src/app/feed/write-reply		100%	18/18	100%	2/2	100%	6/6	100%	16/16
src/app/interest		100%	33/33	100%	0/0	100%	2/2	100%	27/27

Figure 23: Karma Test Result



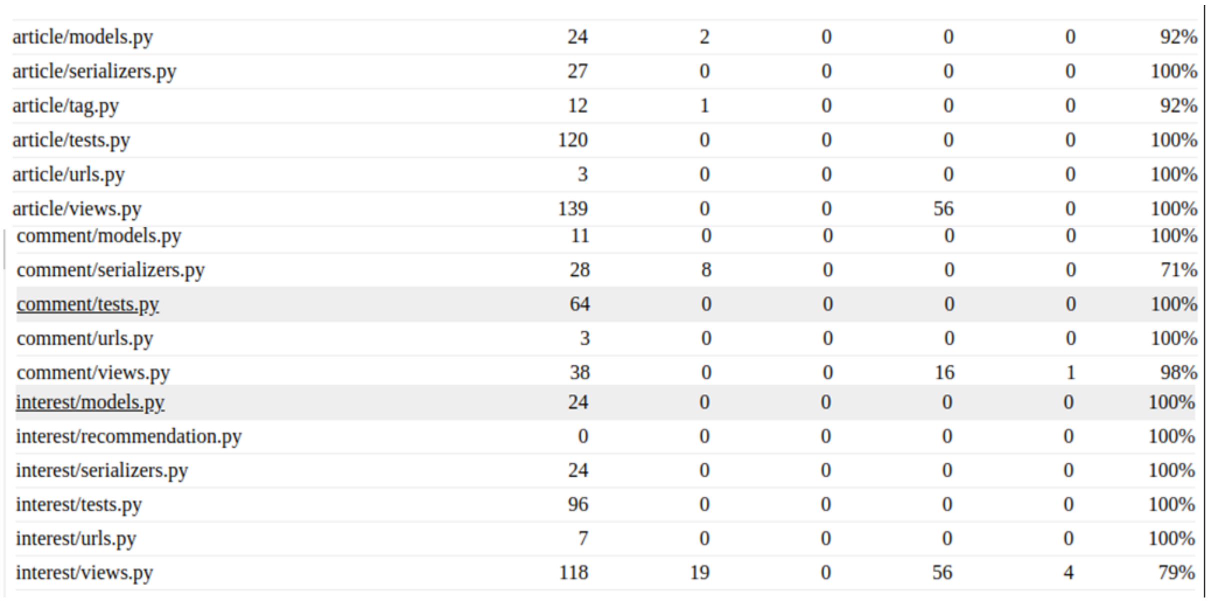
```
this.interestFormatter = ({ name }) => name;
this.articleTagSearch = (text$: Observable<string>) =>
  text$.pipe(
    filter((text: string) => text && text.length >= 0),
    debounceTime(10),
    distinctUntilChanged(),
    switchMap((text: string) => this.feedService.searchTag(text))
  );
this.articleTagFormatter = ({ name }) => name;
```

Figure 24: Not-Test-Covered Example

Figure 24 shows an example that can't be tested easily. For custom pipe and events, there is no easy/convincing way to test it.

4.2. Django Unit Test: Django Test

And for *Django,* we used *django*'s native test module. There are 34 tests and shows 91% branch coverage.

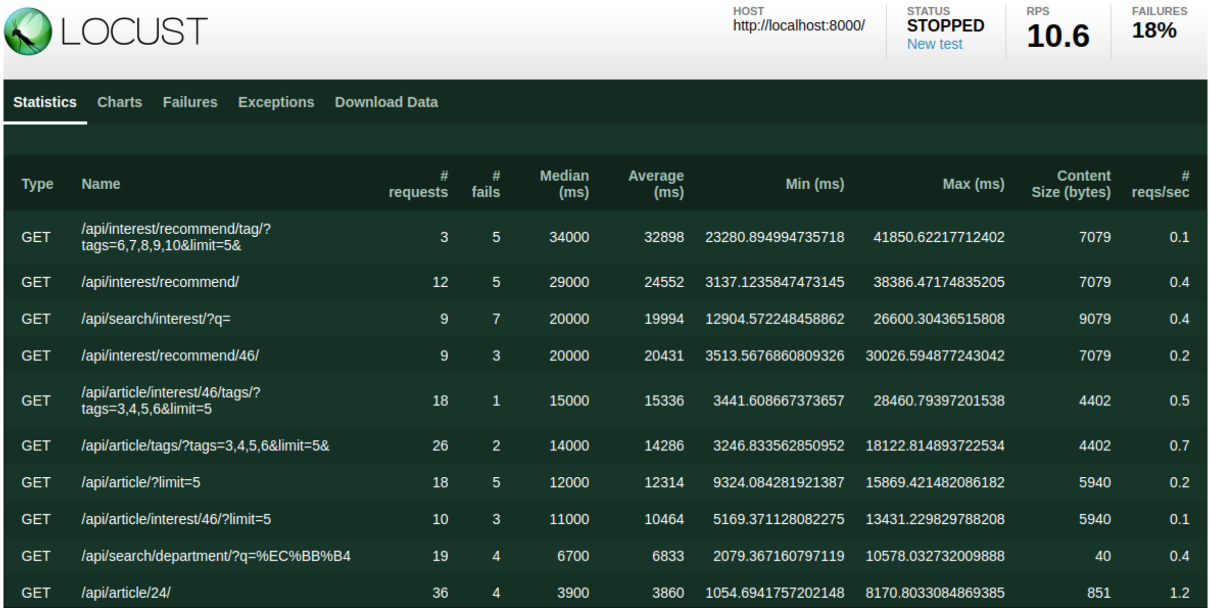


article/models.py	24	2	0	0	0	92%
article/serializers.py	27	0	0	0	0	100%
article/tag.py	12	1	0	0	0	92%
article/tests.py	120	0	0	0	0	100%
article/urls.py	3	0	0	0	0	100%
article/views.py	139	0	0	56	0	100%
comment/models.py	11	0	0	0	0	100%
comment/serializers.py	28	8	0	0	0	71%
comment/tests.py	64	0	0	0	0	100%
comment/urls.py	3	0	0	0	0	100%
comment/views.py	38	0	0	16	1	98%
interest/models.py	24	0	0	0	0	100%
interest/recommendation.py	0	0	0	0	0	100%
interest/serializers.py	24	0	0	0	0	100%
interest/tests.py	96	0	0	0	0	100%
interest/urls.py	7	0	0	0	0	100%
interest/views.py	118	19	0	56	4	79%

Figure 25: Django Test Result

4.3. Load Test: Locust

To perform load test of our service, we configured locust test cases for *django* app. Setting was 300 concurrent users(hatches 30 users per second), and min_wait time for each user was 3 seconds, and max_wait time was 15 seconds.

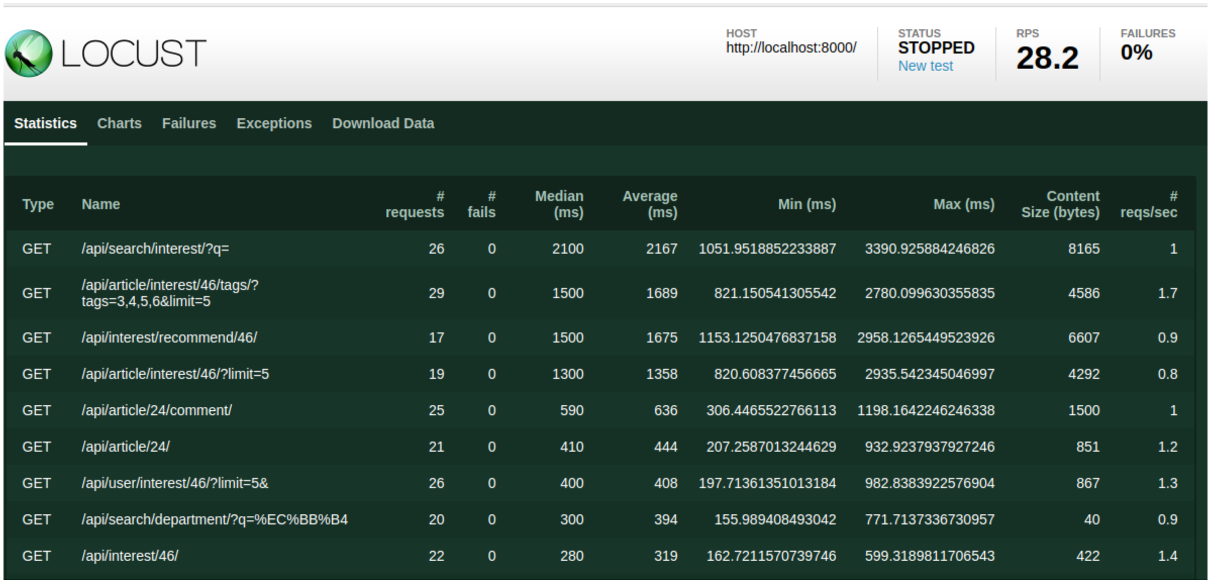


Type	Name	# requests	# fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Content Size (bytes)	# reqs/sec
GET	/api/interest/recommend/tag/?tags=6,7,8,9,10&limit=5&	3	5	34000	32898	23280.894994735718	41850.62217712402	7079	0.1
GET	/api/interest/recommend/	12	5	29000	24552	3137.1235847473145	38386.47174835205	7079	0.4
GET	/api/search/interest/?q=	9	7	20000	19994	12904.572248458862	26600.30436515808	9079	0.4
GET	/api/interest/recommend/46/	9	3	20000	20431	3513.5676860809326	30026.594877243042	7079	0.2
GET	/api/article/interest/46/tags/?tags=3,4,5,6&limit=5	18	1	15000	15336	3441.608667373657	28460.79397201538	4402	0.5
GET	/api/article/tags/?tags=3,4,5,6&limit=5&	26	2	14000	14286	3246.833562850952	18122.814893722534	4402	0.7
GET	/api/article/?limit=5	18	5	12000	12314	9324.084281921387	15869.421482086182	5940	0.2
GET	/api/article/interest/46/?limit=5	10	3	11000	10464	5169.371128082275	13431.229829788208	5940	0.1
GET	/api/search/department/?q=%EC%BB%B4	19	4	6700	6833	2079.367160797119	10578.032732009888	40	0.4
GET	/api/article/24/	36	4	3900	3860	1054.6941757202148	8170.8033084869385	851	1.2

Figure 26: Load Test Result before Optimization

Figure 26 shows result of load test before optimizing our backend application. As one can notice, It was not capable of handle 300 concurrent users, since some of the calls took more than 30 seconds. We noticed there were tendency that requests took longer time have bigger content size, thus we mainly focused on reducing content size by not getting not-needed data. For example, for main feed page, we implemented in a way that every data about comment of the article is fetched at first. To optimize it, we changed its implement in a way that comment data is only fetched when article detail page is rendered. Also we changed our DB to *postgre SQL*, and used caching. These are optimization techniques we used.

- DBMS Change/Deploy
- DB Indexing
- Query Change
- Caching
- Reduces content size(Exclude not-needed data)



Type	Name	# requests	# fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Content Size (bytes)	# reqs/sec
GET	/api/search/interest/?q=	26	0	2100	2167	1051.9518852233887	3390.925884246826	8165	1
GET	/api/article/interest/46/tags/?tags=3,4,5,6&limit=5	29	0	1500	1689	821.150541305542	2780.099630355835	4586	1.7
GET	/api/interest/recommend/46/	17	0	1500	1675	1153.1250476837158	2958.1265449523926	6607	0.9
GET	/api/article/interest/46/?limit=5	19	0	1300	1358	820.608377456665	2935.542345046997	4292	0.8
GET	/api/article/24/comment/	25	0	590	636	306.4465522766113	1198.1642246246338	1500	1
GET	/api/article/24/	21	0	410	444	207.2587013244629	932.9237937927246	851	1.2
GET	/api/user/interest/46/?limit=5&	26	0	400	408	197.71361351013184	982.8383922576904	867	1.3
GET	/api/search/department/?q=%EC%BB%B4	20	0	300	394	155.989408493042	771.7137336730957	40	0.9
GET	/api/interest/46/	22	0	280	319	162.7211570739746	599.3189811706543	422	1.4

Figure 27: Load Test Result after Optimization

Figure 27 shows Load Test result after Optimization. At present, no requests took long than 1.5 seconds.

5. Conclusion

We went out of class and thought about what kind of services would be used by consumers, and as a result, we created Moisha. To this end, we analyzed parts of everyday life that SNS were not properly covering and concluded that there was no service to connect strangers and make them friends. So there was an interest-based Moisha. Even if now service is bit simple and does not have cool UI, if it is well made, users will definitely use it.

One of the constant challenges of working on the project during the semester was to implement the unit test. At first, it was cumbersome and difficult, but as I got used to the project based on TDD, I realized the importance of unit testing. In particular, it was very useful to know immediately that there was a bug in case the unit test fails after committed to master branch.

And the hardest part was the deployment of the service that was created. Although we have previously created web programs in other classes, they were all done on local development environment. Although this process (Deployment) was not as smooth as I expected, it was the most important step to learn about web programming and servers. In particularly, in case of Authorization Token, even though it was not included to request header in localhost environment, the login was maintained, so there was no problem. However, when I actually did the deployment, none of API calls were working because *django* can't identify user from request header. It took several hours to know this fact, and another hour to correct it with Custom Http Interceptor with Authorization Token.

The bug always occurs at something that is not even imagined before, and I realized again that unit testing is important for this. In particular, resolving the issue that the API call did not work after the deployment was a meaningful process because it was previously a part of which was completely unknown due to our lack of knowledge. But we learned a lot from this part again by actually resolving issues while performing the deployment.

6. References

- Used Framework/ Libraries/ softwares

1. Angular Framework: One Framework. Mobile & Desktop

   (https://angular.io/)

2. Django Framework: The Web framework for perfectionists with deadlines

   (https://www.djangoproject.com/)

3. Locust Framework: A modern load testing framework.

   (https://locust.io/)

4. Postgre SQL: The world's most advanced open source database

   (https://www.postgresql.org/)

5. SQLite: Small. Fast. Reliable. Choose any three.

   (https://www.sqlite.org/index.html)

6. Elasticsearch: Open Source Search & Analytics

   (https://www.elastic.co/)

7. Nginx: High Performance Load Balancer, Web Server, & Reverse Proxy

   (https://www.nginx.com/)

8. Uwsgi: uWSGI application server container

   (https://github.com/unbit/uwsgi)

9. Redis: Database for the Instant Experience

   (https://redis.io/)