[2020-fall]
Principles and Practices of Software Development

Team 8 Final Report

https://adoor.world
https://github.com/swsnu/swpp2020-team8

Curie Yoo <2016-19979> curieyoo@snu.ac.kr
JaeWon Kim <2014-17831> slove0318@snu.ac.kr
Jina Park <2015-17261> koyrkr@snu.ac.kr
Jinsun Goo <2015-13796> gjinsun96@gmail.com

I. Introduction

A. Project Abstract

adoor is a prompt-driven social media platform ideal for building intimacy.

Upon successful sign in, users will be able to view a group of 'Recommended Questions' which they can choose to answer. A user-based collaborative filtering (ML) model will be leveraged to recommend specific 'Questions' to each user as 'Recommended Questions'.

Though responding to 'Questions' is the core part of our service, users are not restricted to posting responses to given questions. They may access questions other than the 'Recommended Questions' from the 'Question Feed' or write text-based articles of any lengths and purpose, including questions for which they can receive 'Responses' from friends (i.e. 'Custom Questions').

Users may also choose to customize the audience of their 'Posts' (i.e. 'Custom Questions', 'Responses', 'Articles'); 'Posts' can be shared with either their friends or other users of *adoor* (i.e. 'Anonymous Post'), or both. Also, reactions such as 'Likes', 'Comments' and 'Replies' can be set to be private between the author and the recipient. Furthermore, any user who wishes to find out their friend's opinions on a given question can request a response for the question easily with a few clicks.

Overall, *adoor* welcomes users with a wide range of thought provoking and individualized Question prompts along with the flexibility and focus of the modes of share, opening a door to active intrapersonal and interpersonal communication.

B. Background and Motivation

The Demand for a New Form of Social Media

As of mid 2019, four out of the five top social networks and messaging services worldwide are owned by a single Tech Giant, namely *Facebook*. Yet, 'networking' such services provide, may not be as effective as they seem or should be. According to a survey on social media usage conducted by Open Survey (2020), the rate of using major social networking platforms to consume media contents has steadily increased, while that to interact with friends and acquaintances has continued to decrease to less than 30%. Even worse is the fact that interactions are also quite limited in terms of diversity; on Instagram, for example, an overwhelming number of posts frequently viewed or shared are related to 'tourism', 'daily life', and 'fashion/clothing/commercialization'.

"Town Square" vs. "Chill House Party" Models of Social Media

In the article "A Software Engineer's Advice for Saving Social Media? Keep It Small" by Meg Miller, the author delivers an interesting suggestion on how services should be designed in a way that prioritizes social relationships. Miller quotes Darius Kazemi, a renowned Internet artist who runs a small social media site, *Friend Camp*; Kazemi refers to Facebook as the "digital equivalent of a town square," and advocates for sites that are more like "a chill house party with a considered guest list". The key, he emphasizes, is keeping the social network small so that it can serve specifically for the needs of the community - in ways that social networks with millions of users never could.

Revisiting the Old Social Media Platforms

We know for a fact that there is a demand for social media somewhat different from Facebook or Instagram, the ones currently dominating the culture of social media; in fact, Cyworld, a private social media for close relationships, was 'the' social media platform for several years. Since the current model of "town square" social media gained popularity, people have constantly been reminiscing their very own 'cyber world'. People have been turning to Blogs, Finstagram accounts and various other alternatives for their very own 'chill house parties'. The needs of the many Cyworld users and the troubles current social media users are facing are not gone, but unmet.

C. Target Customers & Market Landscape

Target Customers

- Anyone that is looking to build strong, close, and caring relationships with friends.
- Anyone that has a need for a social media platform where they can feel secure and cared for.
- Anyone that wants to learn more about themselves and their friends - whether it be about their values, habits, or favorite flavors of popcorn.

Market Competitors

	Close Friends Only	Light Conversations	Organized Writing	Daily Prompts	Prompts About Friends	Send Prompts (i.e. tag)	User Analysis
Kakaotalk	X	O	X	X	X	O	O
Diary (apps)	O	X	O	X	X	X	X
Finstagram	O	O	X	X	X	O	O
Naver Blog	O	X	O	X	X	X	O
Facebook	X	O	O	X	O	O	O
adoor	O	O	O	O	O	O	O

Three Major Strengths of *adoor*

Daily Recommendation of Thought-provoking Questions

Users of *adoor* are given a chance to broaden their horizons through an individually curated group of questions. These prompts will comprise a wide range of topics, from light conversation starters to more in depth and personal ones, and will serve as seeds of thoughts to the users on a daily basis.

Focus on Communicating Thoughts

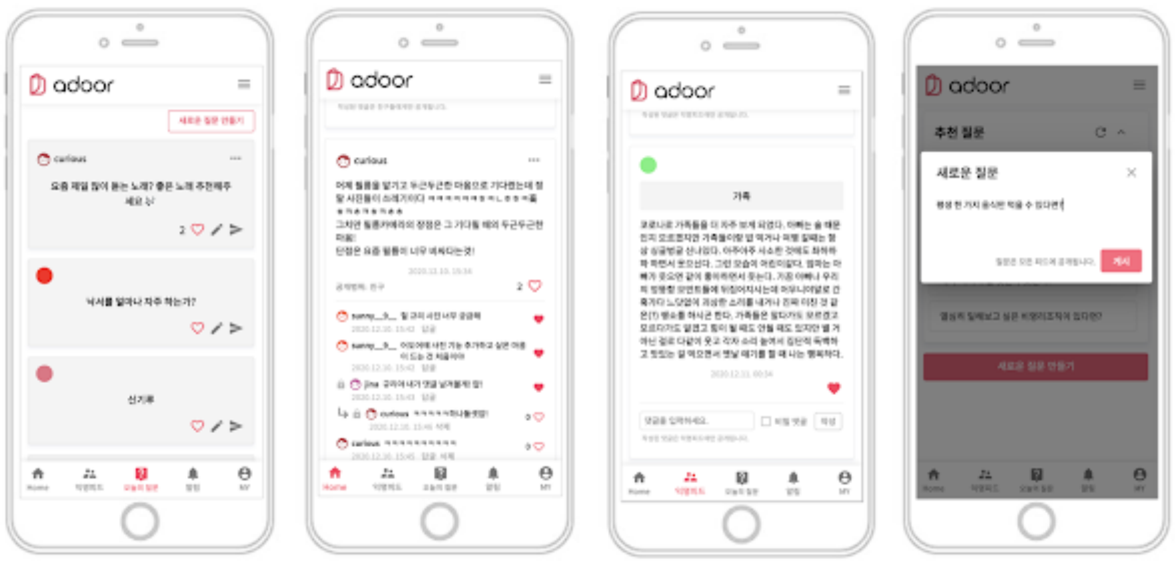
Unlike many other social media platforms run by Big Tech companies, *adoor* has a subscription revenue model, and is thus ad-free. Users will be able to concentrate solely on thinking and communicating without the distraction of irrelevant ads. Also, we make editing and publishing easy for contents of varying modes and lengths by leveraging useful features from other relevant services such as Facebook and Github.

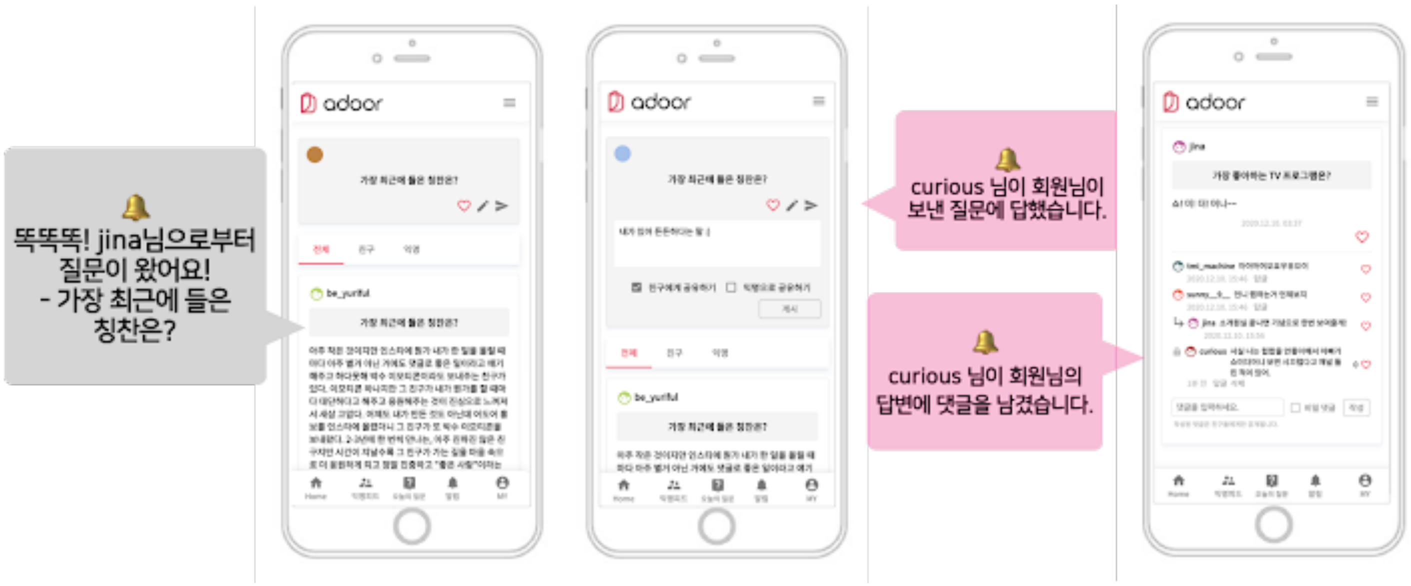
Sharing with and Learning from a Wider Audience

Users can share and view posts not only with their friends but also any with other users of *adoor* via the Anonymous Feed. Any curious minds will benefit hugely from interacting with different users on a range of topics they have never encountered elsewhere. They can even directly publish questions and seek opinions from a larger circle of community. (Any toxic questions will be cleaned up by a not-yet(?)-but-in-30-years-existing AI for *adoor* ☺️)

D. Key Features

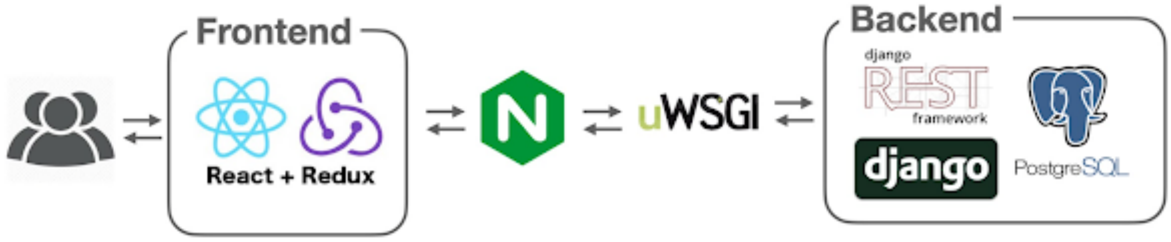
- Answer to various questions updated daily
- Get personalized question recommendation based on user behavior and preferences
- Share your thoughts freely through responses, articles and custom questions
- View friends' & Anonymous people's posts
- Be friends & Communicate with them
    - Communicate through public/private comments and likes
    - Send questions to your friends
    - Answer about what your friends are curious about you by responding to received questions




똑똑똑! jina님으로부터 질문이 왔어요! - 가장 최근에 들은 칭찬은?
adoor
가장 최근에 들은 칭찬은?
adoor
가장 최근에 들은 칭찬은?
curious 님이 회원님이 보낸 질문에 답했습니다.
curious 님이 회원님의 답변에 댓글을 남겼습니다.
adoor
가장 좋아하는 TV 프로그램은?

II. Design

A. System Architecture

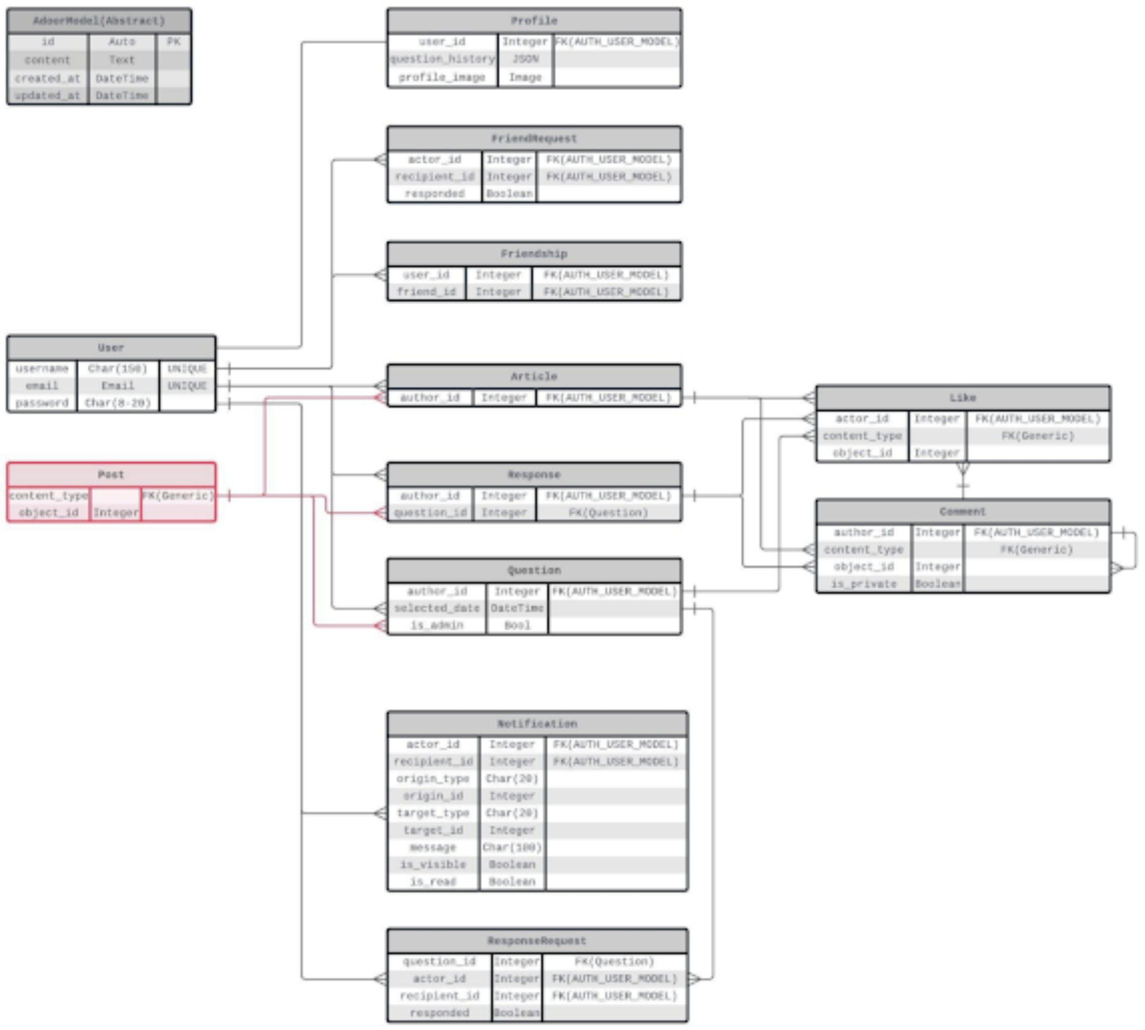


We used Django REST Framework for Backend, and React for Frontend. For DB, we used Postgresql DB because it is not suitable for SQLite to process multiple calls at the same time.

We used uWSGI to host Django as web servers, and we deployed both Backend and Frontend servers to a AWS EC2 instance using Nginx. All requests sent by the users are processed by Nginx, and requests that must be sent to the backend are passed through uWSGI. Frontend is distributed to port 443(HTTPS's default port), Backend is distributed to port 80, and requests between Frontend and Backend are made through port 8000 proxy servers.

B. Backend

1. Model - ER Diagram


AdoorModel(Abstract)
id | Auto | PK
content | Text
created_at | DateTime
updated_at | DateTime
Profile
user_id | Integer | FK(AUTH_USER_MODEL)
question_history | JSON
profile_image | Image
FriendRequest
actor_id | Integer | FK(AUTH_USER_MODEL)
recipient_id | Integer | FK(AUTH_USER_MODEL)
responded | Boolean
Friendship
user_id | Integer | FK(AUTH_USER_MODEL)
friend_id | Integer | FK(AUTH_USER_MODEL)
User
username | Char(150) | UNIQUE
email | Email | UNIQUE
password | Char(8-20)
Post
content_type | | FK(Generic)
object_id | Integer
Article
author_id | Integer | FK(AUTH_USER_MODEL)
Like
actor_id | Integer | FK(AUTH_USER_MODEL)
content_type | | FK(Generic)
object_id | Integer
Response
author_id | Integer | FK(AUTH_USER_MODEL)
question_id | Integer | FK(Question)
Comment
author_id | Integer | FK(AUTH_USER_MODEL)
content_type | | FK(Generic)
object_id | Integer
is_private | Boolean
Question
author_id | Integer | FK(AUTH_USER_MODEL)
selected_date | DateTime
is_admin | Bool
Notification
actor_id | Integer | FK(AUTH_USER_MODEL)
recipient_id | Integer | FK(AUTH_USER_MODEL)
origin_type | Char(20)
origin_id | Integer
target_type | Char(20)
target_id | Integer
message | Char(100)
is_visible | Boolean
is_read | Boolean
ResponseRequest
question_id | Integer | FK(Question)
actor_id | Integer | FK(AUTH_USER_MODEL)
recipient_id | Integer | FK(AUTH_USER_MODEL)
responded | Boolean

2. API

Model	API	GET	POST	PUT	DELETE
User	/api/user/	Get user list	X	X	X
	/api/user/:id/	Get specified user	X	X	X
	/api/user/token/	Get token for user	X	X	X
	/api/user/token/refresh/	Refresh token for user	X	X	X
	/api/user/token/verify/	Verify token for user	X	X	X
	/api/user/login/	X	Sign user in	X	X
	/api/user/signup/	X	Create new user	X	X
	/api/user/select-questions/	Get sign up question list for user	X	X	X
	/api/user/me/	Get current user profile detail	Update current user info (including sign up questions)	X	X
	/api/user/me/friends/	Get friend list for user	X	X	X
	/api/user/search/:query/	Get result of username search	X	X	X
	/api/user/friend/:friend_id/	X	X	X	Delete friend for user
Friend Request	/api/user/friend-requests/	Get sent friend request list	Create new friend request from actor to recipient	X	X
	/api/user/friend-requests/:recipient_id/	X	X	X	Delete friend request from actor to recipient

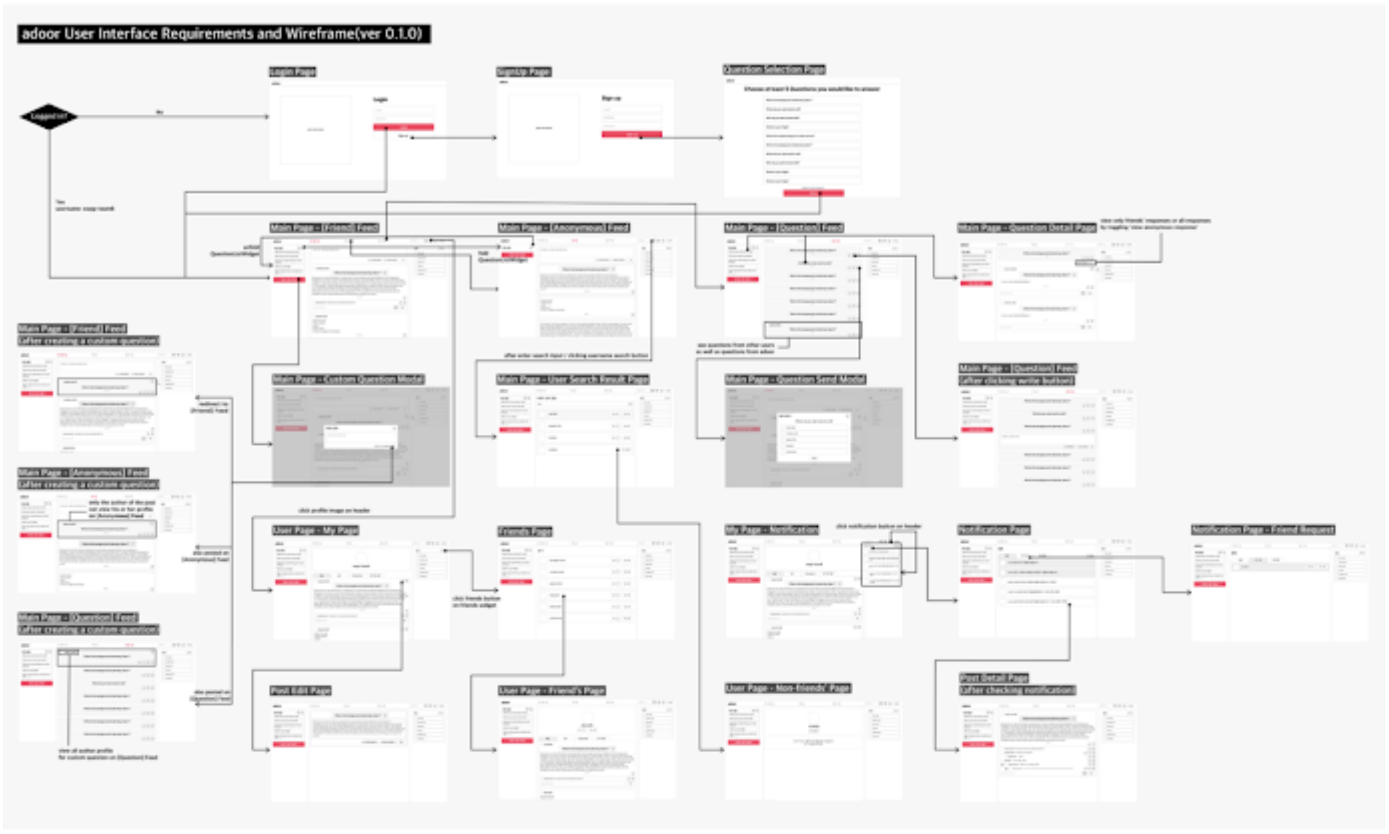
	/api/user/friend-requests/:recipient_id/respond/	X	X	Accept or Reject friend request from actor to recipient	X
	/api/user/friend-requests/:recipient_id/respond/	X	X	Accept or Reject friend request from actor to recipient	X
Feed	/api/feed/all/	Get all feed list	X	X	X
	/api/feed/friend/	Get friends feed	X	X	X
	/api/feed/anonymous/	Get anonymous feed	X	X	X
	/api/feed/user/:id/	Get feed for specified user	X	X	X
Question	/api/feed/questions/	Get question list	X	X	X
	/api/feed/questions/daily/	Get daily question list	X	X	X
	/api/feed/questions/daily/recommended/	Get daily recommended question list	X	X	X
	/api/feed/questions/:id/	Get specified question	X	X	X
	/api/feed/questions/:id/friend/	Get specified response in friend feed	X	X	X
	/api/feed/questions/:id/anonymous/	Get specified response in anonymous feed	X	X	X
	/api/feed/questions/custom-questions/	X	Create new custom question	X	X
	/api/feed/questions/custom-questions/:id/	Get specified custom question	X	Edit specified custom question	Delete specified custom question

Response Request	/api/feed/questions/response-request/	X	Post (send) response request	X	X
	/api/feed/questions/:question_id /response-request/:recipient_id	X	X	X	Delete specified response request
	/api/feed/questions/:question_id /response-request/	Get response request for specified question	X	X	X
Article	/api/feed/articles/	X	Create new article	X	X
	/api/feed/articles/:id/	Get specified article	X	Edit specified article	Delete specified article
Response	/api/feed/responses/	X	Create new response	X	X
	/api/feed/responses/:id/	Get specified response	X	Edit specified response	Delete specified response
Comment	/api/comments/	X	Create new comment	X	X
	/api/comments/:id/	X	X	X	Delete specified comment
Like	/api/likes/	X	Create new like	X	X
	/api/likes/:id/	X	X	X	Delete specified like
Notification	/api/notifications/	Get notification list	X	Mark all notifications as read	X
	/api/notifications/:id/	X	X	Mark specified notification as read	X

C. Frontend

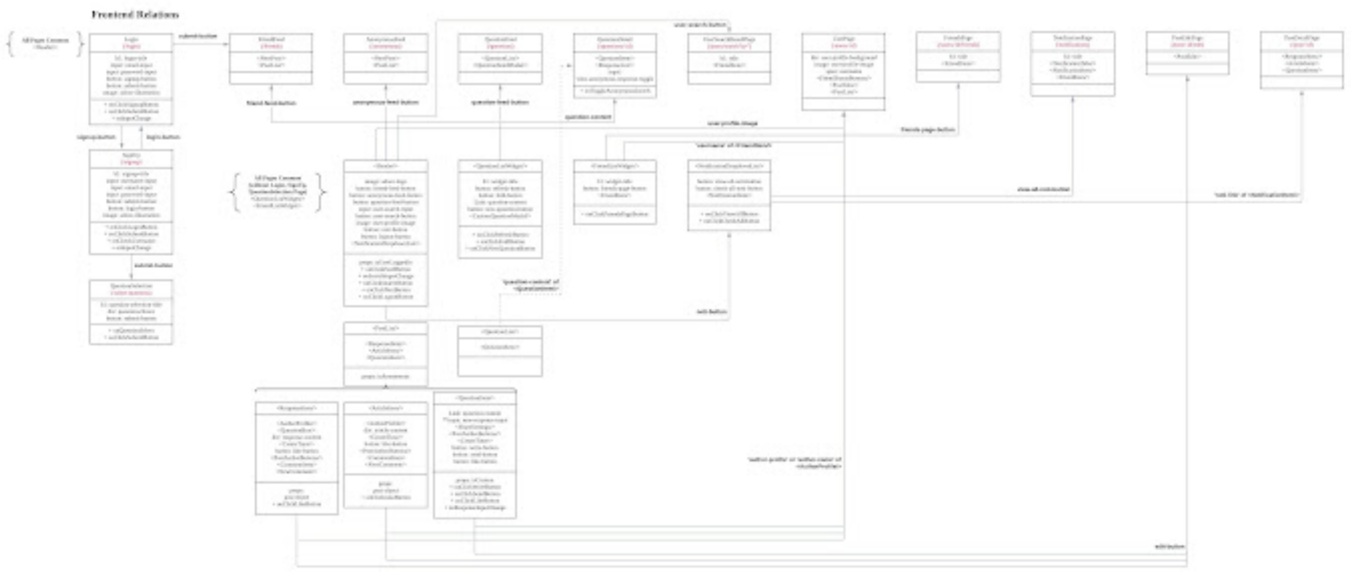
1. User Interface and Wireframe

[Wireframe] Google Drive Link
[All Pages] Google Drive Link


adoor User Interface Requirements and Wireframe(ver 0.1.0)
Logged in?
Login Page
SignUp Page
Question Selection Page
Main Page - [Friend] Feed
Main Page - [Anonymous] Feed
Main Page - [Question] Feed
Main Page - Question Detail Page
Main Page - [Friend] Feed (after creating a custom question)
Main Page - Custom Question Modal
Main Page - User Search Result Page
Main Page - Question Send Modal
Main Page - [Question] Feed (after clicking write button)
Main Page - [Anonymous] Feed (after creating a custom question)
User Page - My Page
Friends Page
My Page - Notification
Notification Page
Notification Page - Friend Request
Main Page - [Question] Feed (after creating a custom question)
Post Edit Page
User Page - Friend's Page
User Page - Non-friends' Page
Post Detail Page (after checking notification)

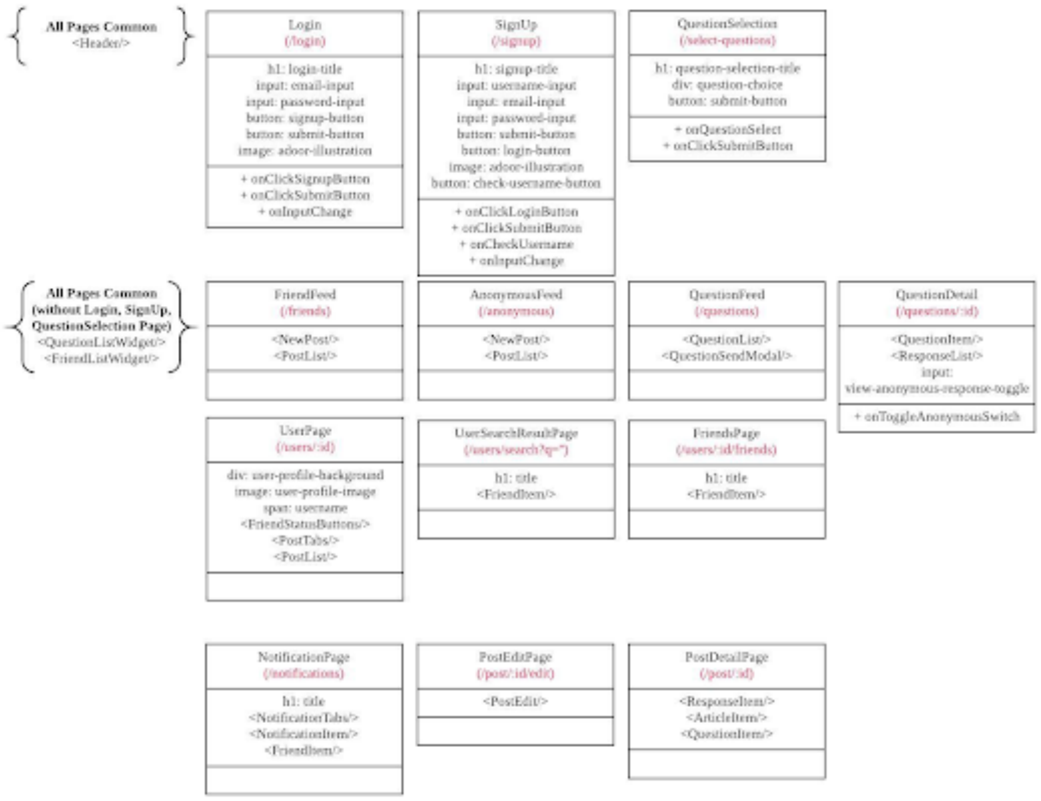
2. Component Relationships

[Component Relationships] Google Drive Link

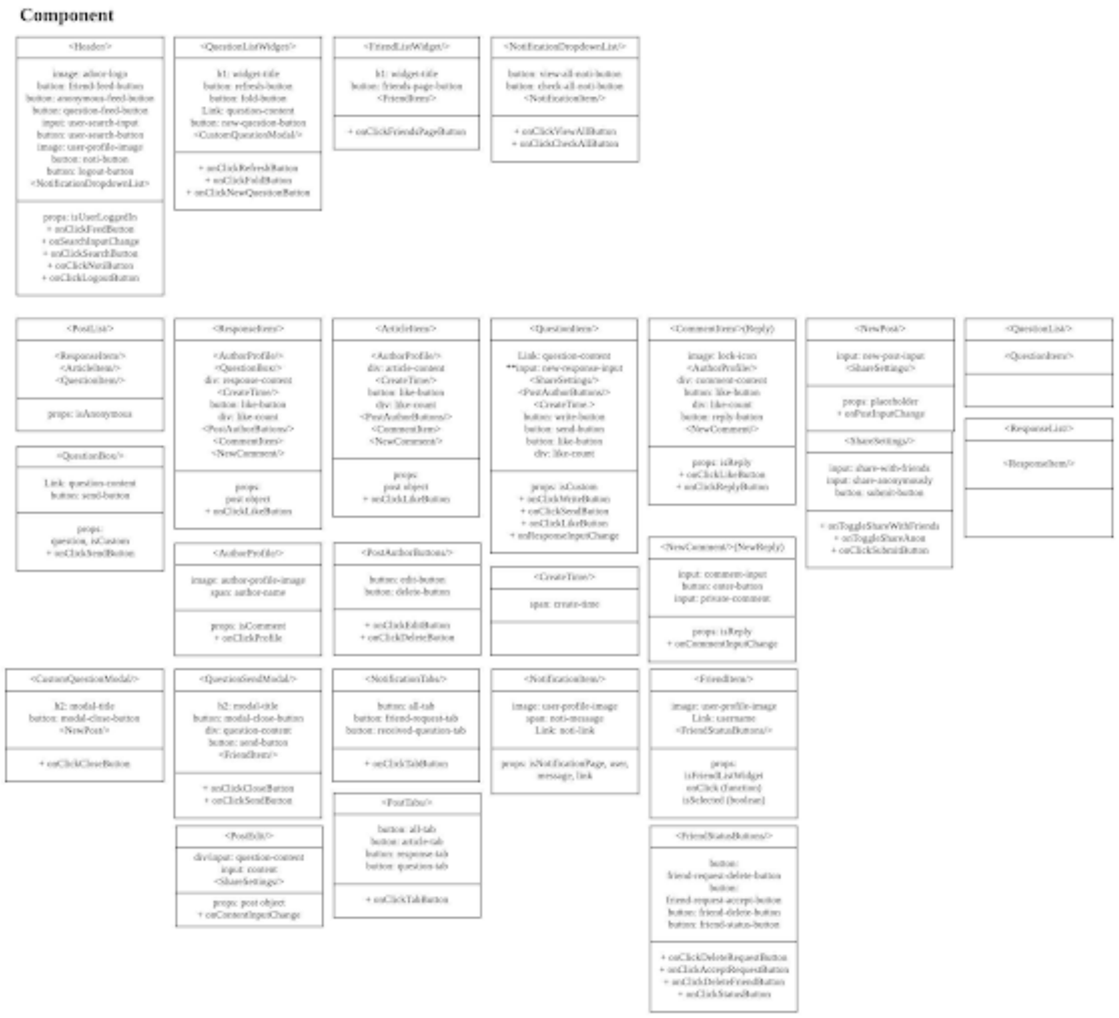


3. Pages & Components

- Pages



- Components



4. Reducers

- States of Reducer

Reducer	**State**	**Description**
User	**currentUser**	contains user information
	loginError	contains login error
	signUpError	contains signup error
	selectedUser	contains user information for selected UserPage
	selectQuestion	contains whether signed-up user completed selection of preferred questions
Questions	**sampleQuestions**	array of sample questions to pick @ sign up page
	randomQuestions	random questions (updated when refresh button clicked)
	recommendedQuestions	recommended questions for user

	selectedQuestion	question object @ question detail page
	selectedQuestionResponses	response objects of selected question
	selectedQuestionResponseRequest	array of response requests for selected question
	next	next request URL for infinite scroll
	maxPage	max page for response list on QuestionDetailPage
	dailyQuestions	daily questions displayed @ question feed
Posts	**anonymousPosts**	array of anonymous post objects
	friendPosts	array of friends' post objects
	selectedUserPosts	array of posts of currently visited user
	selectedUserId	id of selected user
	selectedPost	post object at post detail page
	selectedPostFailure	error when get selected post(i.e 403/404)
	next	next request URL for infinite scroll
Friends	**friendList**	array of friend objects with profile
	friendRequestError	error info when friend request error
Notifications	**receivedNotifications**	array of all received notifications
	next	next request URL for infinite scroll
search	**searchObj**	search results object for user search

- ## Actions of Reducer

Reducer	**Action**	**Function**
User	**requestLogin({ email, password })**	send email/password, and receive current user profile
	logout()	logout current user

	requestSignUp({ email, username, password, questions })	send user info(email, username, password, favorite questions), check validation, and login
	postSelectedQuestions	send selected questions
	getCurrentUser	get current user information
	getSelectedUser	get selected user information for UserPage
Questions	**getSampleQuestions**	get sample questions for question selection
	getRecommendedQuestions(userId)	get recommended questions for current user
	getDailyQuestions()	get all daily questions for Question feed
	aappendDailyQuestions()	append next page of daily questions
	resetSelectedQuestion()	reset stated related to selected question(selectedQuestion, selectedQuestionResponses, selectedResponseRequests)
	getRandomQuestions()	get random questions when refresh button on Question List Widget is clicked
	getResponsesByQuestionWithType(id, type)	get responses by selected question with type(all/friends/anonymous)
	appendResponsesByQuestionWithType(id, type)	append next page of responses by selected question with type(all/friends/anonymous)
	getResponseRequestsByQuestion(id)	get response requests by questionId
	createResponseRequest(responseRequestObj)	create new response request with a response request object
	deleteResponseRequest(qid, rid)	delete response request with questionId, requesteeId
Posts	**createPost(postObj)**	create a new post with post object
	getPostsByType(type[anon/friends/user], userId)	get posts(responses, articles, questions) with type[friend/anonymous/user] (and userId for user type)
	appendPosts(type[anon/frineds/user])	append posts by feed type
	resetPosts()	reset all posts state when page changes

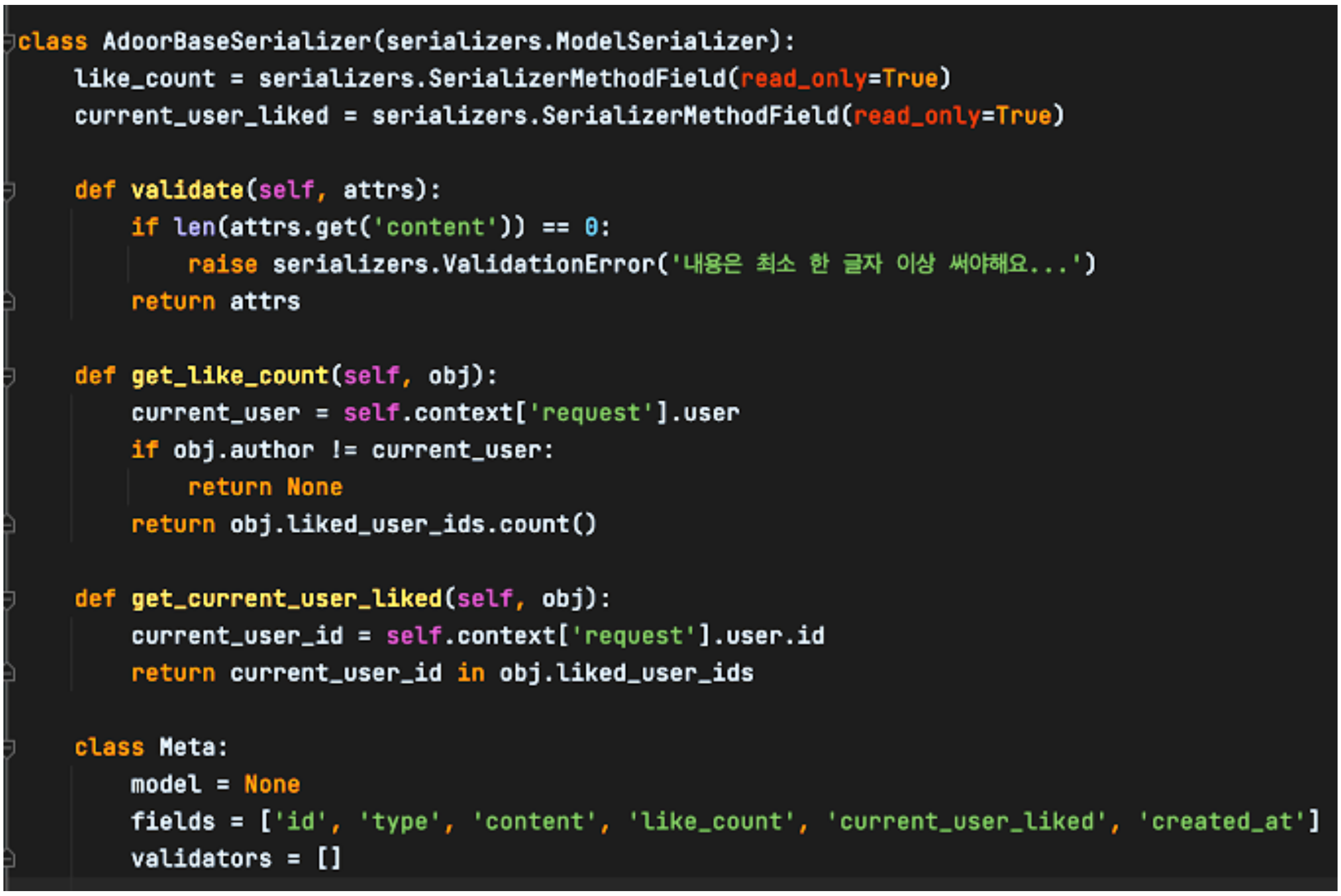
	editSelectedPost(postObj)	edit selected post
	deletePost(postId, type)	delete selected post
	getSelectedUserPosts(userId)	get posts of selected user
	getPostByType(postId)	get selected post for Post Detail Page
	toggleLike(type, id)	toggle like for selected post/comment/reply
	createComment(commentObj, postId)	create a comment for selected post
	deleteComment(commentId)	delete selected post
	createReply(commentObj, commentId)	create a reply for selected comment
	deleteReply(commentId)	delete selected reply
Friends	**getFriendList()**	get all friends(userId, username)
	acceptFriendRequest(friendId)	add friend to friend list when accept friend request
	deleteFriend(friendId)	delete friend from friend list when delete friend request
	deleteFriendRequest(userId)	delete friend request by userId
	rejectFriendRequest(userId)	reject friend request
Notifications	**getNotifications()**	get notifications
	appendNotifications()	append notifications
	readNotification(notificationId)	read selected notification
	readAllNotifications()	read all notifications
Search	**fetchSearchResults**	get all user search results

III. Implementation Details

A. Backend

1. Serializers and Generic Views

We leveraged serializers and generic class-based views Django REST Framework provides to implement our API. They allow for the management of consistent and robust views in a simple yet elegant way. Below is the base serializer for any models involving user-content, such as but not limited to Article, Response, and Comment.



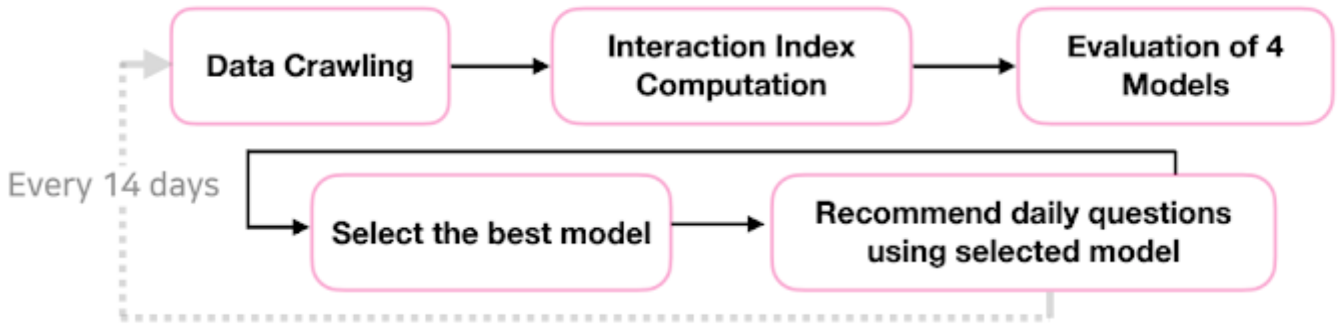
```python
class AdoorBaseSerializer(serializers.ModelSerializer):
    like_count = serializers.SerializerMethodField(read_only=True)
    current_user_liked = serializers.SerializerMethodField(read_only=True)

    def validate(self, attrs):
        if len(attrs.get('content')) == 0:
            raise serializers.ValidationError('내용은 최소 한 글자 이상 써야해요...')
        return attrs

    def get_like_count(self, obj):
        current_user = self.context['request'].user
        if obj.author != current_user:
            return None
        return obj.liked_user_ids.count()

    def get_current_user_liked(self, obj):
        current_user_id = self.context['request'].user.id
        return current_user_id in obj.liked_user_ids

    class Meta:
        model = None
        fields = ['id', 'type', 'content', 'like_count', 'current_user_liked', 'created_at']
        validators = []
```

2. ML Feature

Every two weeks, a cron job is called to first crawl question sets and user interaction data. User interaction data is evaluated in the following manner:

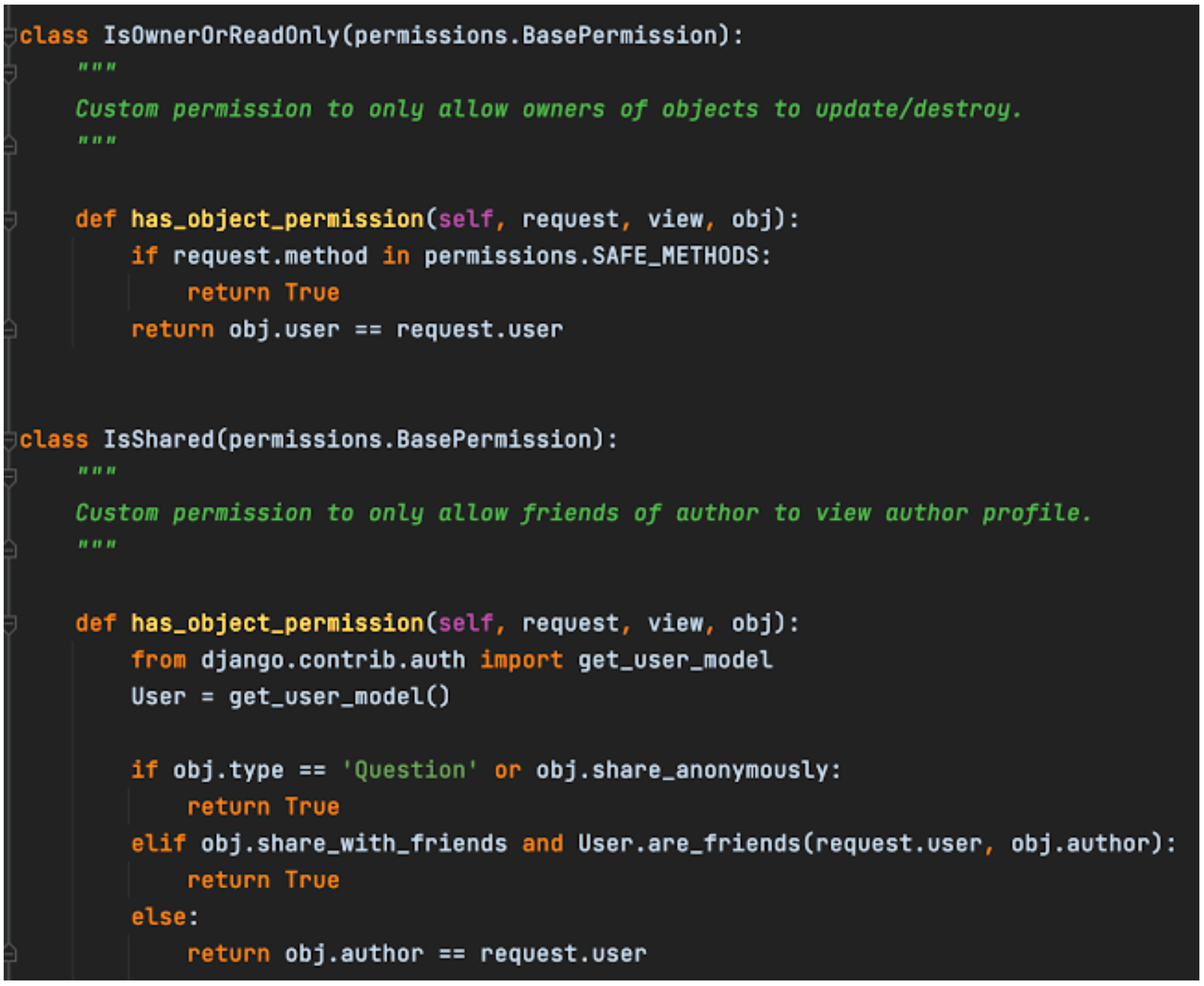
Questions that the user has answered	4
Custom Questions the user created	3.5
Response Requests that the user sent to friends	2.5
Questions selected upon sign up	2
Questions that the user liked	1

Then, a function for generating recommendation indices is called; we evaluate the performance of four different models - Popularity model, Content-based model, Collaborative Filtering model, and a Hybrid model of the Content-based and Collaborative Filtering model - in analyzing the acquired data. The model with the best performance is used for the following 14 days to generate daily question recommendations for each user. The process of this feature is illustrated in the following diagram:


Data Crawling
Interaction Index Computation
Evaluation of 4 Models
Every 14 days
Select the best model
Recommend daily questions using selected model

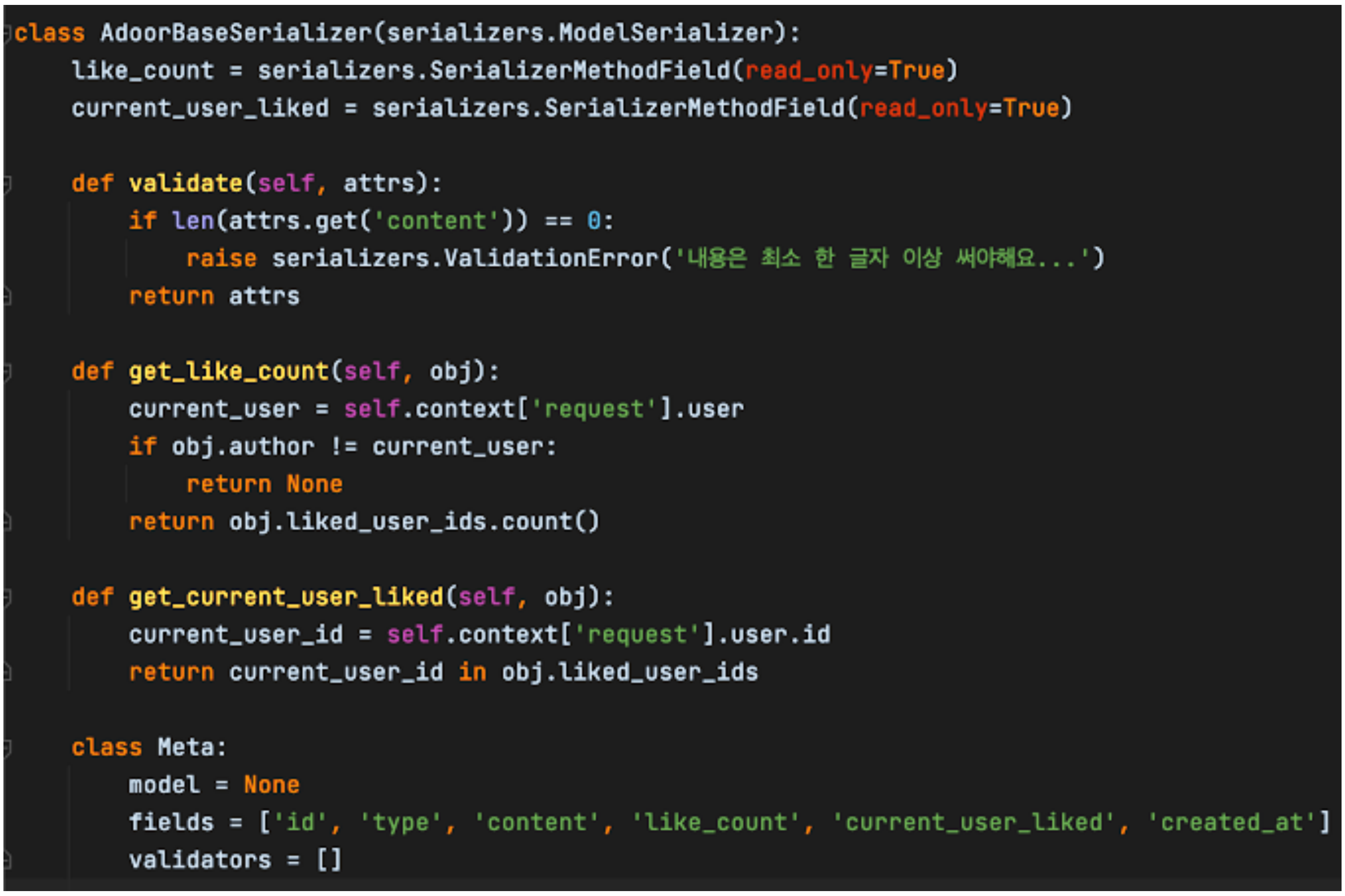
3. Authorization

We believe that providing a safe and secure environment for the users is an extremely important feature for *adoor*, and we tried our best to reflect that in our serializers by implementing specific permissions and sending specific responses only. Some examples of such are as follows:



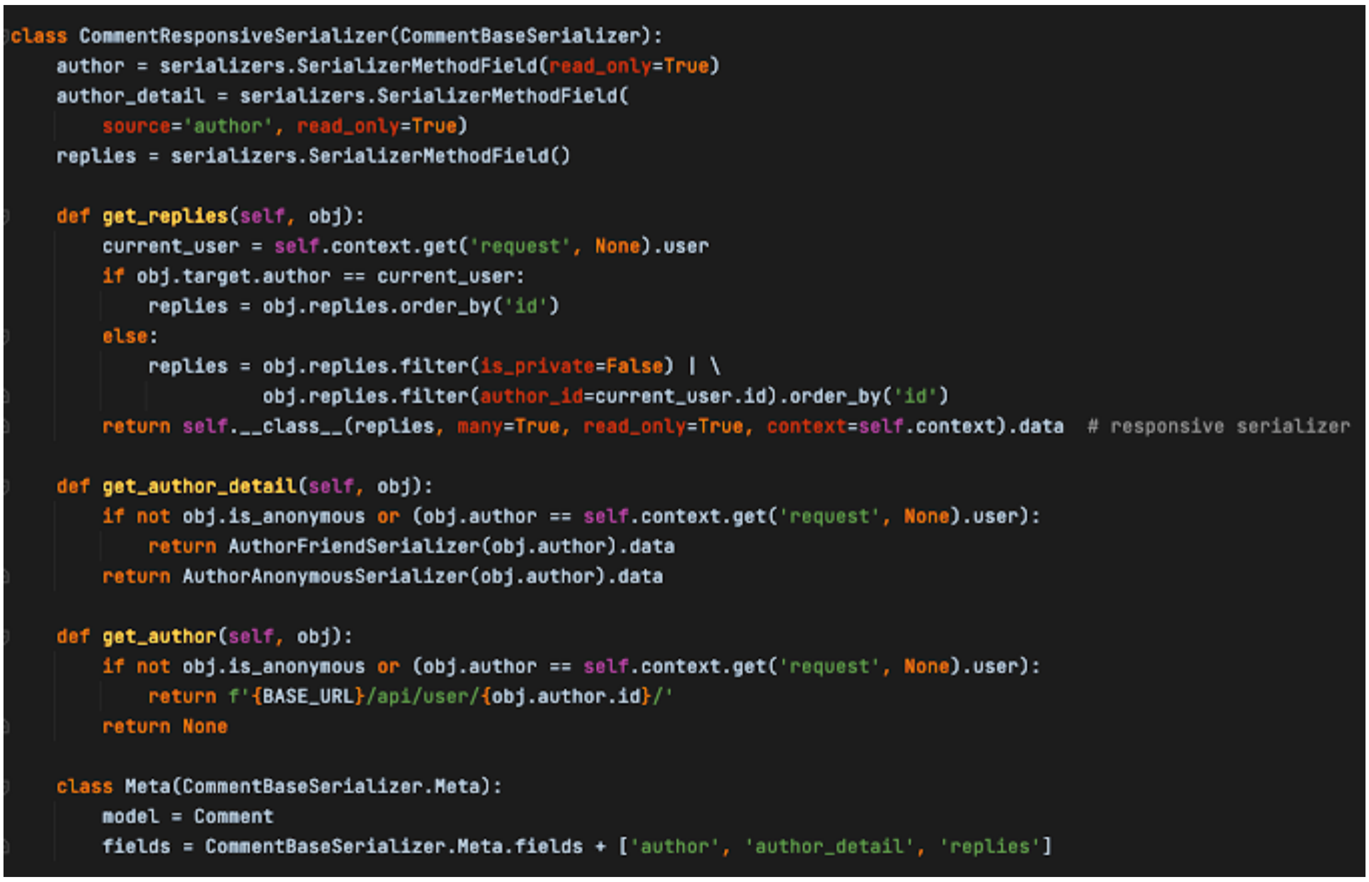
```python
class IsOwnerOrReadOnly(permissions.BasePermission):
    """
    Custom permission to only allow owners of objects to update/destroy.
    """

    def has_object_permission(self, request, view, obj):
        if request.method in permissions.SAFE_METHODS:
            return True
        return obj.user == request.user


class IsShared(permissions.BasePermission):
    """
    Custom permission to only allow friends of author to view author profile.
    """

    def has_object_permission(self, request, view, obj):
        from django.contrib.auth import get_user_model
        User = get_user_model()

        if obj.type == 'Question' or obj.share_anonymously:
            return True
        elif obj.share_with_friends and User.are_friends(request.user, obj.author):
            return True
        else:
            return obj.author == request.user
```

<Permissions>



```python
class AdoorBaseSerializer(serializers.ModelSerializer):
    like_count = serializers.SerializerMethodField(read_only=True)
    current_user_liked = serializers.SerializerMethodField(read_only=True)

    def validate(self, attrs):
        if len(attrs.get('content')) == 0:
            raise serializers.ValidationError('내용은 최소 한 글자 이상 써야해요...')
        return attrs

    def get_like_count(self, obj):
        current_user = self.context['request'].user
        if obj.author != current_user:
            return None
        return obj.liked_user_ids.count()

    def get_current_user_liked(self, obj):
        current_user_id = self.context['request'].user.id
        return current_user_id in obj.liked_user_ids

    class Meta:
        model = None
        fields = ['id', 'type', 'content', 'like_count', 'current_user_liked', 'created_at']
        validators = []
```

<Base Serializer Used for All Content-related Models>

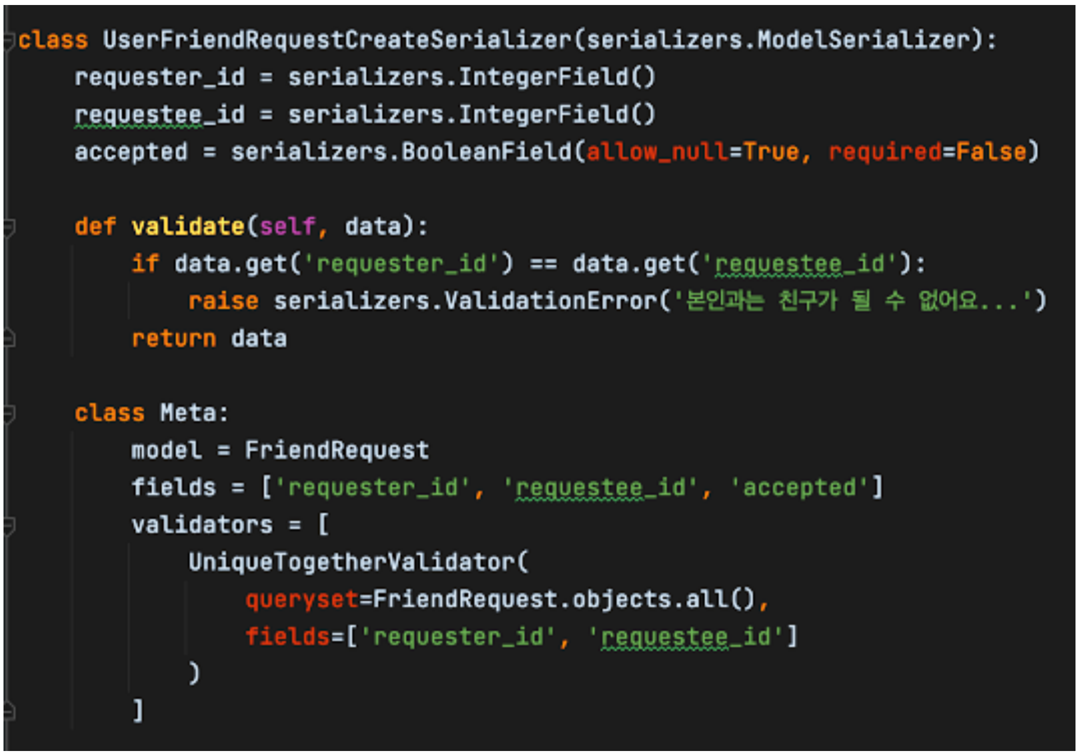


```python
class CommentResponsiveSerializer(CommentBaseSerializer):
    author = serializers.SerializerMethodField(read_only=True)
    author_detail = serializers.SerializerMethodField(
        source='author', read_only=True)
    replies = serializers.SerializerMethodField()

    def get_replies(self, obj):
        current_user = self.context.get('request', None).user
        if obj.target.author == current_user:
            replies = obj.replies.order_by('id')
        else:
            replies = obj.replies.filter(is_private=False) | \
                      obj.replies.filter(author_id=current_user.id).order_by('id')
        return self.__class__(replies, many=True, read_only=True, context=self.context).data  # responsive serializer

    def get_author_detail(self, obj):
        if not obj.is_anonymous or (obj.author == self.context.get('request', None).user):
            return AuthorFriendSerializer(obj.author).data
        return AuthorAnonymousSerializer(obj.author).data

    def get_author(self, obj):
        if not obj.is_anonymous or (obj.author == self.context.get('request', None).user):
            return f'{BASE_URL}/api/user/{obj.author.id}/'
        return None

    class Meta(CommentBaseSerializer.Meta):
        model = Comment
        fields = CommentBaseSerializer.Meta.fields + ['author', 'author_detail', 'replies']
```

<A Comment Serializer for the User Page>

4. Validation

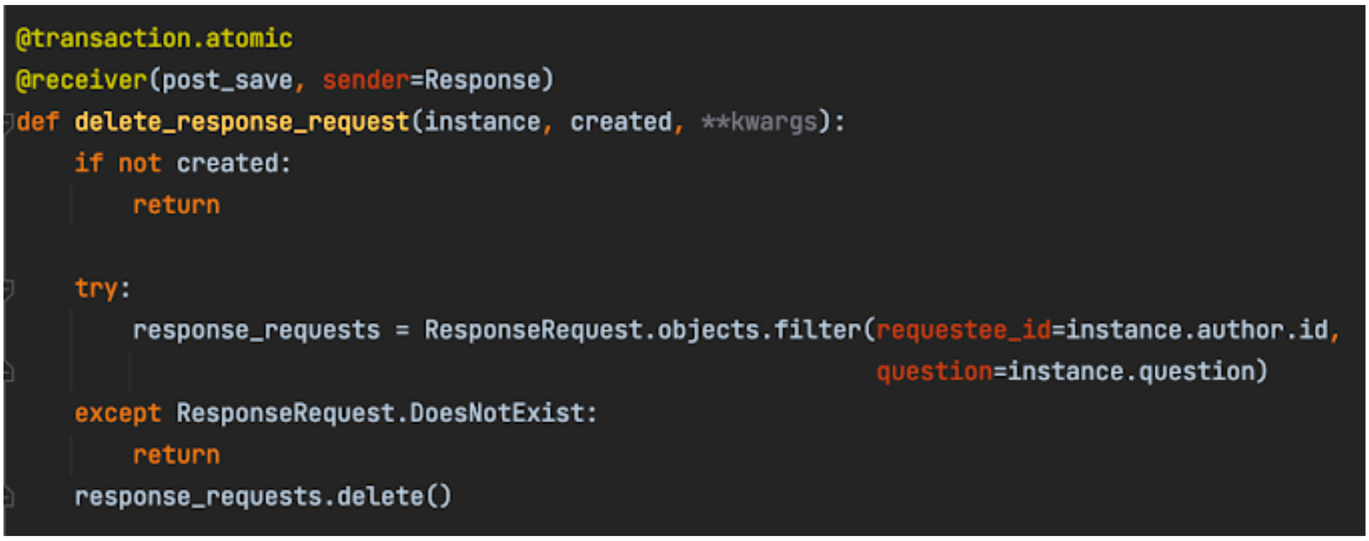
Validating our serializers was another important task for us. We maintained a robust API by implementing various validations such as the following:



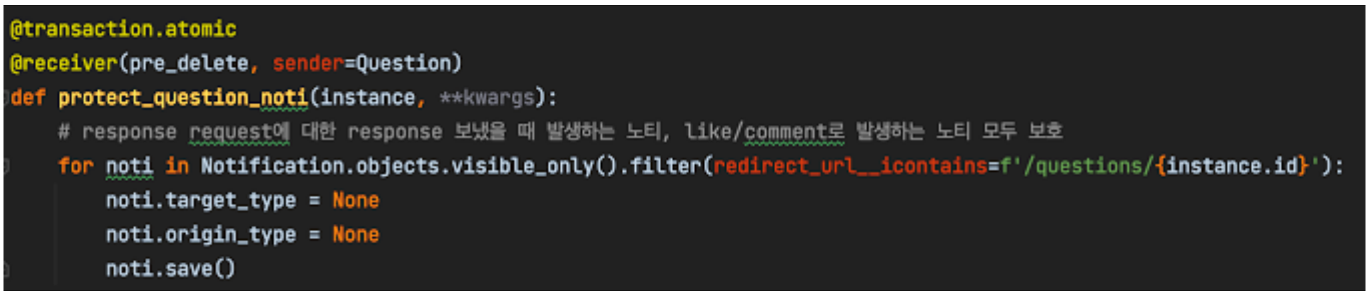
```python
class UserFriendRequestCreateSerializer(serializers.ModelSerializer):
    requester_id = serializers.IntegerField()
    requestee_id = serializers.IntegerField()
    accepted = serializers.BooleanField(allow_null=True, required=False)

    def validate(self, data):
        if data.get('requester_id') == data.get('requestee_id'):
            raise serializers.ValidationError('본인과는 친구가 될 수 없어요...')
        return data

    class Meta:
        model = FriendRequest
        fields = ['requester_id', 'requestee_id', 'accepted']
        validators = [
            UniqueTogetherValidator(
                queryset=FriendRequest.objects.all(),
                fields=['requester_id', 'requestee_id']
            )
        ]
```

5. Signals

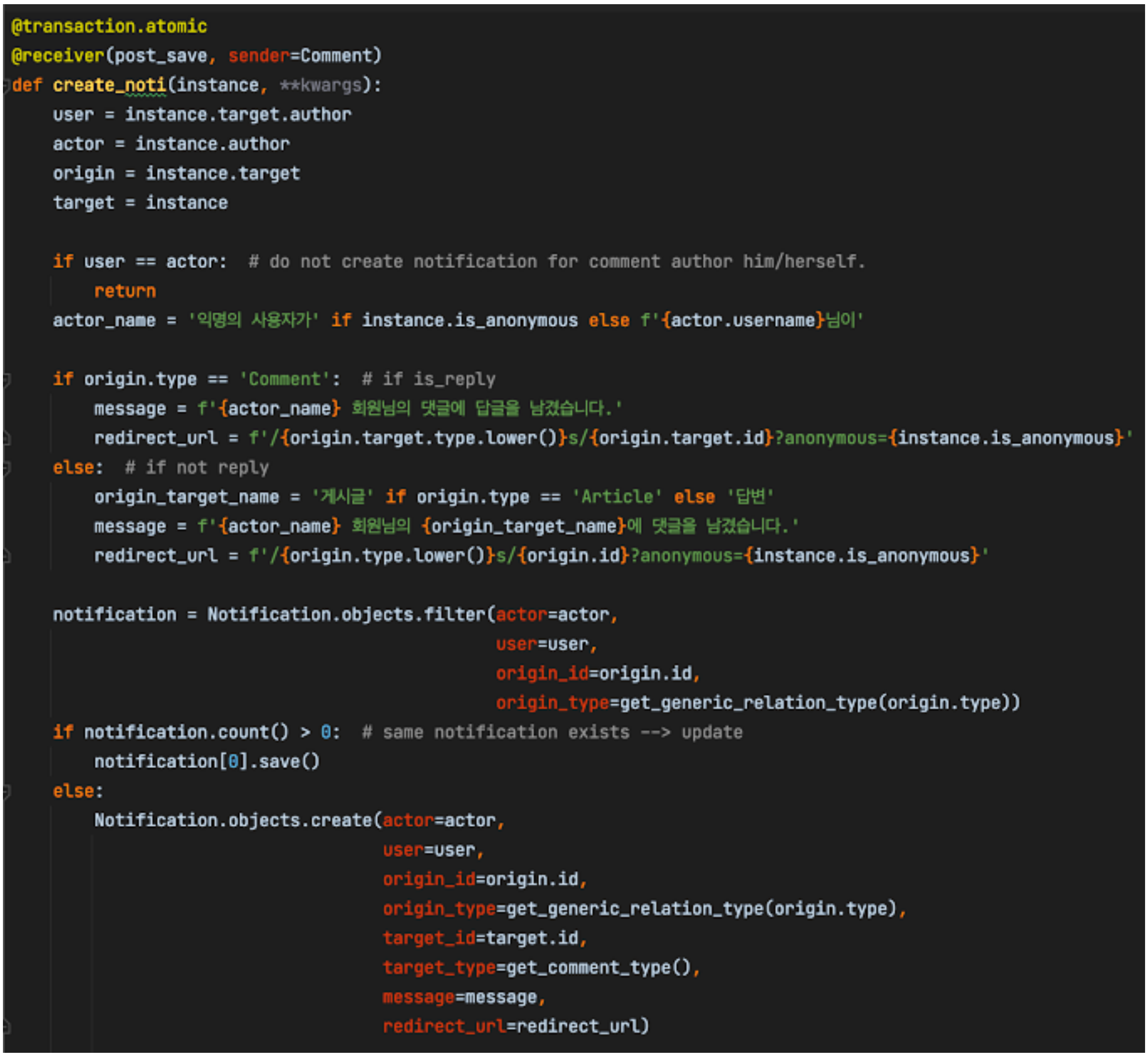
Django signals were used to create notifications and prevent the notifications from being deleted when the target (i.e. direct reference model for the notification) or origin (i.e. the origin of the target for the notification) is deleted. Examples of such receivers are as follows:



```python
@transaction.atomic
@receiver(post_save, sender=Response)
def delete_response_request(instance, created, **kwargs):
    if not created:
        return

    try:
        response_requests = ResponseRequest.objects.filter(requestee_id=instance.author.id,
                                                           question=instance.question)
    except ResponseRequest.DoesNotExist:
        return
    response_requests.delete()
```



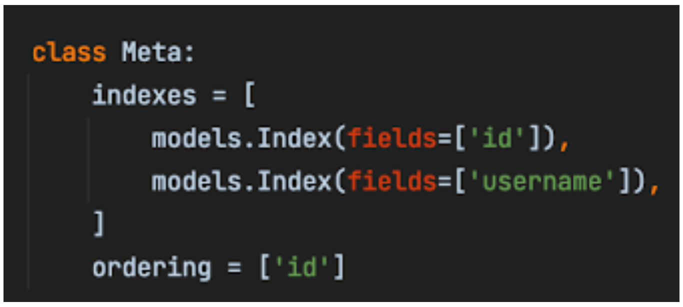
```python
@transaction.atomic
@receiver(pre_delete, sender=Question)
def protect_question_noti(instance, **kwargs):
    # response request에 대한 response 보냈을 때 발생하는 노티, like/comment로 발생하는 노티 모두 보호
    for noti in Notification.objects.visible_only().filter(redirect_url__icontains=f'/questions/{instance.id}'):
        noti.target_type = None
        noti.origin_type = None
        noti.save()
```



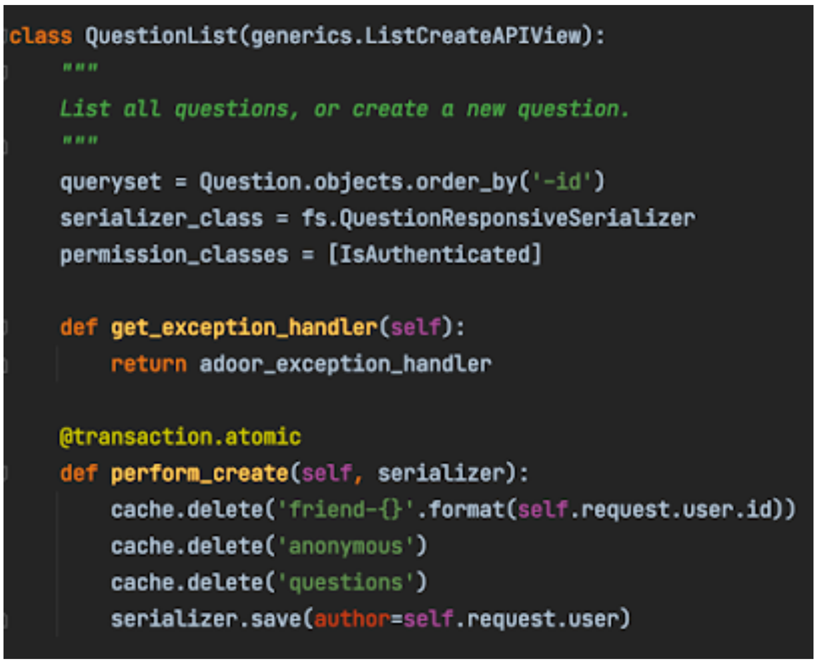
```python
@transaction.atomic
@receiver(post_save, sender=Comment)
def create_noti(instance, **kwargs):
    user = instance.target.author
    actor = instance.author
    origin = instance.target
    target = instance

    if user == actor:  # do not create notification for comment author him/herself.
        return
    actor_name = '익명의 사용자가' if instance.is_anonymous else f'{actor.username}님이'

    if origin.type == 'Comment':  # if is_reply
        message = f'{actor_name} 회원님의 댓글에 답글을 남겼습니다.'
        redirect_url = f'/{origin.target.type.lower()}s/{origin.target.id}?anonymous={instance.is_anonymous}'
    else:  # if not reply
        origin_target_name = '게시글' if origin.type == 'Article' else '답변'
        message = f'{actor_name} 회원님의 {origin_target_name}에 댓글을 남겼습니다.'
        redirect_url = f'/{origin.type.lower()}s/{origin.id}?anonymous={instance.is_anonymous}'

    notification = Notification.objects.filter(actor=actor,
                                               user=user,
                                               origin_id=origin.id,
                                               origin_type=get_generic_relation_type(origin.type))
    if notification.count() > 0:  # same notification exists --> update
        notification[0].save()
    else:
        Notification.objects.create(actor=actor,
                                    user=user,
                                    origin_id=origin.id,
                                    origin_type=get_generic_relation_type(origin.type),
                                    target_id=target.id,
                                    target_type=get_comment_type(),
                                    message=message,
                                    redirect_url=redirect_url)
```

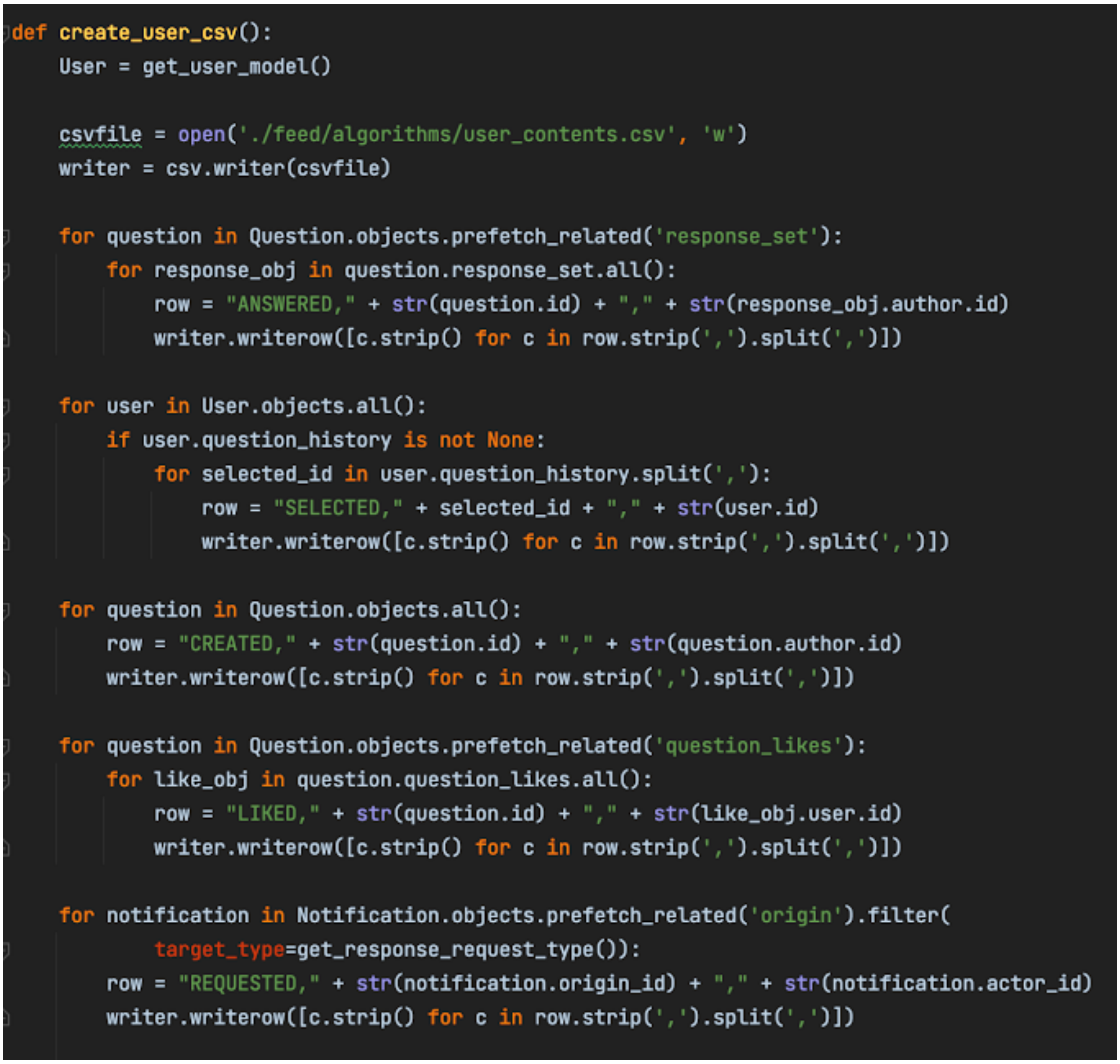
6. Optimization

Due to various authorization issues and nested models, the speed of our API is sub-optimal. Thus, we optimized our API by indexing our models, caching, query optimization, and more. An example of such are the following:

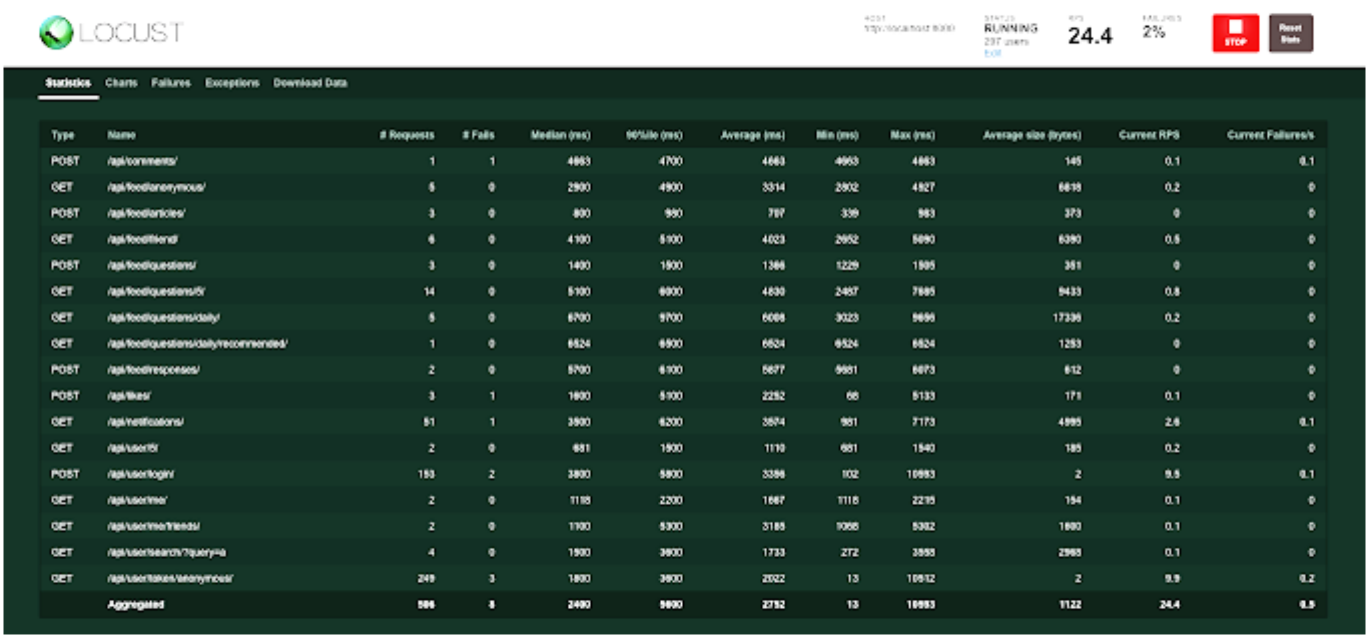


```python
class Meta:
    indexes = [
        models.Index(fields=['id']),
        models.Index(fields=['username']),
    ]
    ordering = ['id']
```

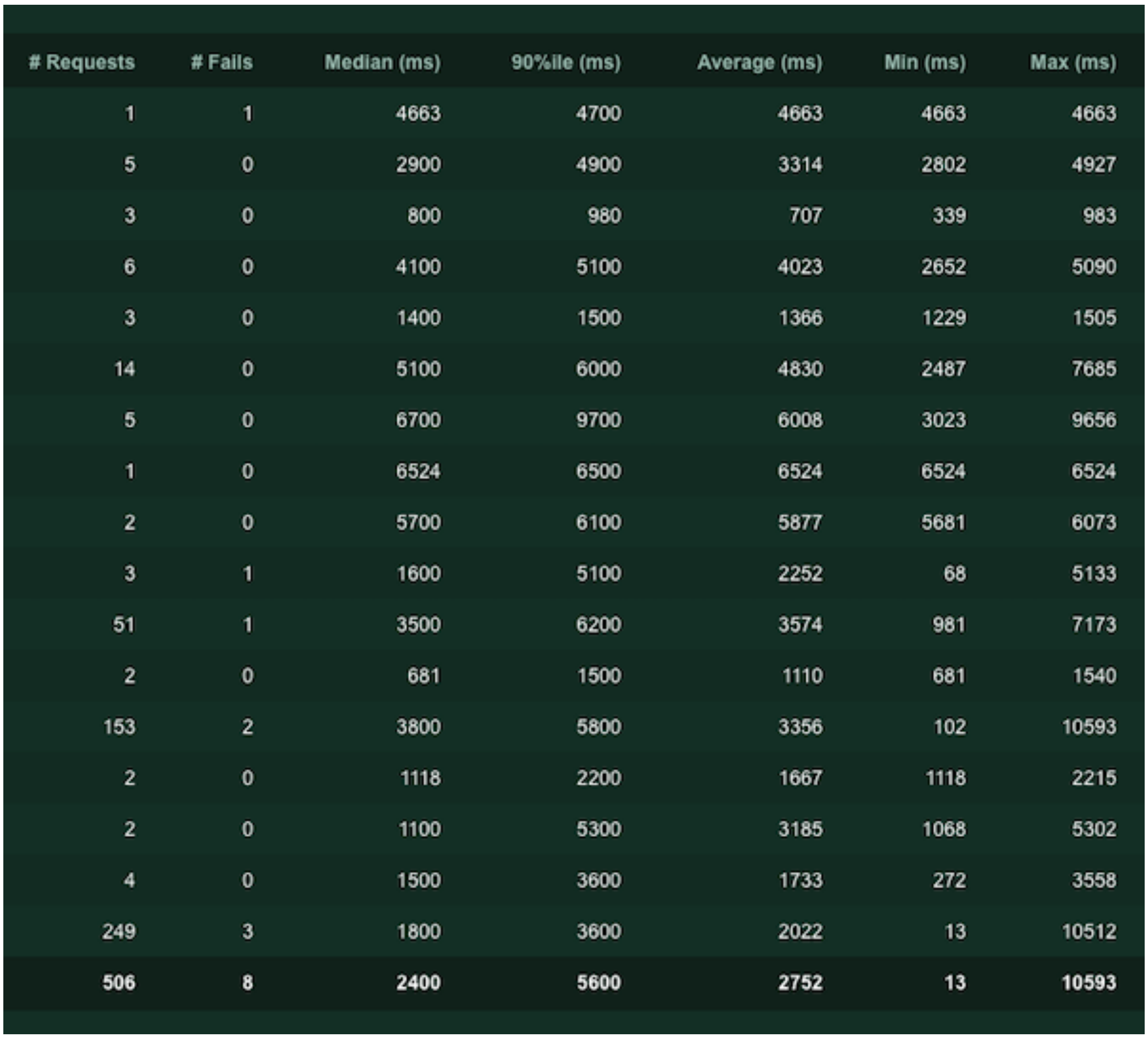

```python
class QuestionList(generics.ListCreateAPIView):
    """
    List all questions, or create a new question.
    """
    queryset = Question.objects.order_by('-id')
    serializer_class = fs.QuestionResponsiveSerializer
    permission_classes = [IsAuthenticated]

    def get_exception_handler(self):
        return adoor_exception_handler

    @transaction.atomic
    def perform_create(self, serializer):
        cache.delete('friend-{}'.format(self.request.user.id))
        cache.delete('anonymous')
        cache.delete('questions')
        serializer.save(author=self.request.user)
```


```python
def create_user_csv():
    User = get_user_model()

    csvfile = open('./feed/algorithms/user_contents.csv', 'w')
    writer = csv.writer(csvfile)

    for question in Question.objects.prefetch_related('response_set'):
        for response_obj in question.response_set.all():
            row = "ANSWERED," + str(question.id) + "," + str(response_obj.author.id)
            writer.writerow([c.strip() for c in row.strip(',').split(',')])

    for user in User.objects.all():
        if user.question_history is not None:
            for selected_id in user.question_history.split(','):
                row = "SELECTED," + selected_id + "," + str(user.id)
                writer.writerow([c.strip() for c in row.strip(',').split(',')])

    for question in Question.objects.all():
        row = "CREATED," + str(question.id) + "," + str(question.author.id)
        writer.writerow([c.strip() for c in row.strip(',').split(',')])

    for question in Question.objects.prefetch_related('question_likes'):
        for like_obj in question.question_likes.all():
            row = "LIKED," + str(question.id) + "," + str(like_obj.user.id)
            writer.writerow([c.strip() for c in row.strip(',').split(',')])

    for notification in Notification.objects.prefetch_related('origin').filter(
            target_type=get_response_request_type()):
        row = "REQUESTED," + str(notification.origin_id) + "," + str(notification.actor_id)
        writer.writerow([c.strip() for c in row.strip(',').split(',')])
```

The result of our Locust load testing with 300 concurrent users (30 user hatch per second), with a min wait time of 5 seconds and a max wait time of 15 seconds is as follows:



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	/api/comments/	1	1	4663	4700	4663	4663	4663	145	0.1	0.1
GET	/api/feed/anonymous/	5	0	2900	4900	3314	2802	4927	6618	0.2	0
POST	/api/feed/articles/	3	0	800	980	707	339	983	373	0	0
GET	/api/feed/friend/	6	0	4100	5100	4023	2652	5090	6390	0.6	0
POST	/api/feed/questions/	3	0	1400	1500	1366	1229	1505	351	0	0
GET	/api/feed/questions/5/	14	0	5100	6000	4830	2487	7685	9433	0.8	0
GET	/api/feed/questions/daily/	5	0	6700	9700	6008	3023	9656	17336	0.2	0
GET	/api/feed/questions/daily/recommended/	1	0	6524	6500	6524	6524	6524	1253	0	0
POST	/api/feed/responses/	2	0	5700	6100	5877	5681	6073	612	0	0
POST	/api/likes/	3	1	1600	5100	2252	68	5133	171	0.1	0
GET	/api/notifications/	51	1	3500	6200	3574	981	7173	4995	2.6	0.1
GET	/api/user/5/	2	0	681	1500	1110	681	1540	185	0.2	0
POST	/api/user/login/	153	2	3800	5800	3356	102	10593	2	9.5	0.1
GET	/api/user/me/	2	0	1118	2200	1667	1118	2215	154	0.1	0
GET	/api/user/me/friends/	2	0	1100	5300	3185	1068	5302	1600	0.1	0
GET	/api/user/search/?query=a	4	0	1500	3600	1733	272	3558	2965	0.1	0
GET	/api/user/token/anonymous/	249	3	1800	3600	2022	13	10512	2	9.9	0.2
	Aggregated	506	8	2400	5600	2752	13	10593	1122	24.4	0.5



# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)
1	1	4663	4700	4663	4663	4663
5	0	2900	4900	3314	2802	4927
3	0	800	980	707	339	983
6	0	4100	5100	4023	2652	5090
3	0	1400	1500	1366	1229	1505
14	0	5100	6000	4830	2487	7685
5	0	6700	9700	6008	3023	9656
1	0	6524	6500	6524	6524	6524
2	0	5700	6100	5877	5681	6073
3	1	1600	5100	2252	68	5133
51	1	3500	6200	3574	981	7173
2	0	681	1500	1110	681	1540
153	2	3800	5800	3356	102	10593
2	0	1118	2200	1667	1118	2215
2	0	1100	5300	3185	1068	5302
4	0	1500	3600	1733	272	3558
249	3	1800	3600	2022	13	10512
506	**8**	**2400**	**5600**	**2752**	**13**	**10593**

B. Frontend

The components were all developed as stateless functional components. We then used react-hooks to operate state management and life cycle functions and work with redux. The useEffect was used to control the life cycle and the useState was used to obtain the effect of setState. Using useSelector and useDispatch, the state and action of the redux are linked to the component.

For a directory structure, we used a more simple component-page structure similar to the container-presenter pattern. A page-by-page component was placed in the pages directory, and the building block components in it were placed in the components directory.

The status of various API calls (wait, success, failure) was managed collectively through loading reducer. All action types have been declared in compliance with regex rules such as @@_REQUEST, @@_SUCCESS, @@_FAILURE. The loading reducer then determines whether the API is loading or completed by determining the regex of all action types occurring in the redux. Loading reducer was useful for introducing loading components such as spinners and skeleton components on various pages.

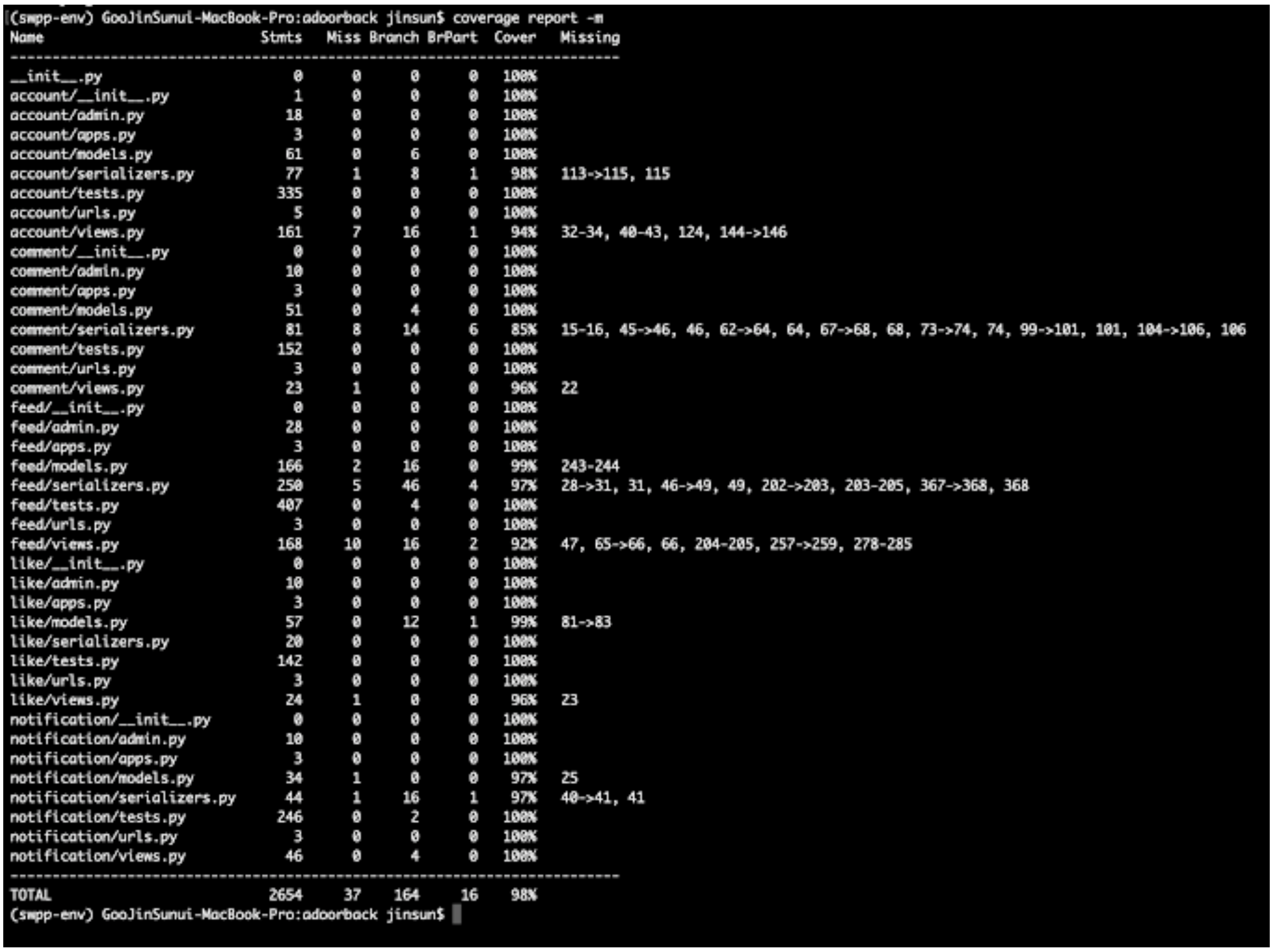
We used styled-components instead of css to style the application. The method is a "CSS-in-JS" style, which is convenient because styling can also be done on a component-by-component basis without className declaration. Furthermore, we were able to style along the react pattern because we could pass the props like real components and vary the styles according to the props. We also used components and icons of Material ui core together. It was useful for elements such as buttons and input.

Used libraries include *intersection-observer* for infinite scrolling implementation, *use-onclickoutside* module for closing drop-down when clicking outside of drop-down, and *date-fns* for created time computations (such as "30 minutes ago, just before") for comment or post.

IV. Testing

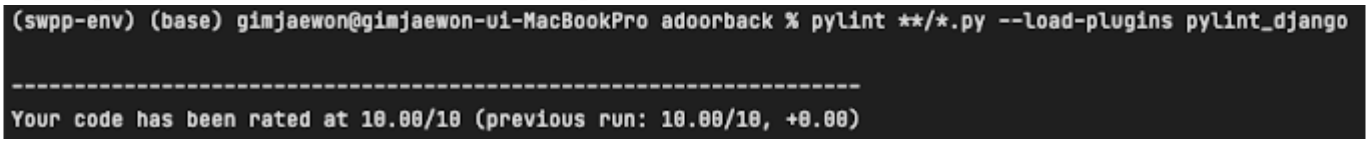
A. Backend

We used django-test-plus to implement and run unit tests on all the model structures. Also, we implemented API tests focusing on JSON responses and permissions. Especially for testing permissions in each API test, we made tests in response to the number of possible cases related to permissions, and we were able to confirm that they all passed. There are 80 tests in total, and it shows 98% coverage.



```
(swpp-env) GooJinSunui-MacBook-Pro:adoorback jinsun$ coverage report -m
Name                           Stmts   Miss Branch BrPart  Cover   Missing
--------------------------------------------------------------------------
__init__.py                        0      0      0      0   100%
account/__init__.py                1      0      0      0   100%
account/admin.py                  18      0      0      0   100%
account/apps.py                    3      0      0      0   100%
account/models.py                 61      0      6      0   100%
account/serializers.py            77      1      8      1    98%   113->115, 115
account/tests.py                 335      0      0      0   100%
account/urls.py                    5      0      0      0   100%
account/views.py                 161      7     16      1    94%   32-34, 40-43, 124, 144->146
comment/__init__.py                0      0      0      0   100%
comment/admin.py                  10      0      0      0   100%
comment/apps.py                    3      0      0      0   100%
comment/models.py                 51      0      4      0   100%
comment/serializers.py            81      8     14      6    85%   15-16, 45->46, 46, 62->64, 64, 67->68, 68, 73->74, 74, 99->101, 101, 104->106, 106
comment/tests.py                 152      0      0      0   100%
comment/urls.py                    3      0      0      0   100%
comment/views.py                  23      1      0      0    96%   22
feed/__init__.py                   0      0      0      0   100%
feed/admin.py                     28      0      0      0   100%
feed/apps.py                       3      0      0      0   100%
feed/models.py                   166      2     16      0    99%   243-244
feed/serializers.py              250      5     46      4    97%   28->31, 31, 46->49, 49, 202->203, 203-205, 367->368, 368
feed/tests.py                    407      0      4      0   100%
feed/urls.py                       3      0      0      0   100%
feed/views.py                    168     10     16      2    92%   47, 65->66, 66, 204-205, 257->259, 278-285
like/__init__.py                   0      0      0      0   100%
like/admin.py                     10      0      0      0   100%
like/apps.py                       3      0      0      0   100%
like/models.py                    57      0     12      1    99%   81->83
like/serializers.py               20      0      0      0   100%
like/tests.py                    142      0      0      0   100%
like/urls.py                       3      0      0      0   100%
like/views.py                     24      1      0      0    96%   23
notification/__init__.py           0      0      0      0   100%
notification/admin.py             10      0      0      0   100%
notification/apps.py               3      0      0      0   100%
notification/models.py            34      1      0      0    97%   25
notification/serializers.py       44      1     16      1    97%   40->41, 41
notification/tests.py            246      0      2      0   100%
notification/urls.py               3      0      0      0   100%
notification/views.py             46      0      4      0   100%
--------------------------------------------------------------------------
TOTAL                           2654     37    164     16    98%
(swpp-env) GooJinSunui-MacBook-Pro:adoorback jinsun$
```

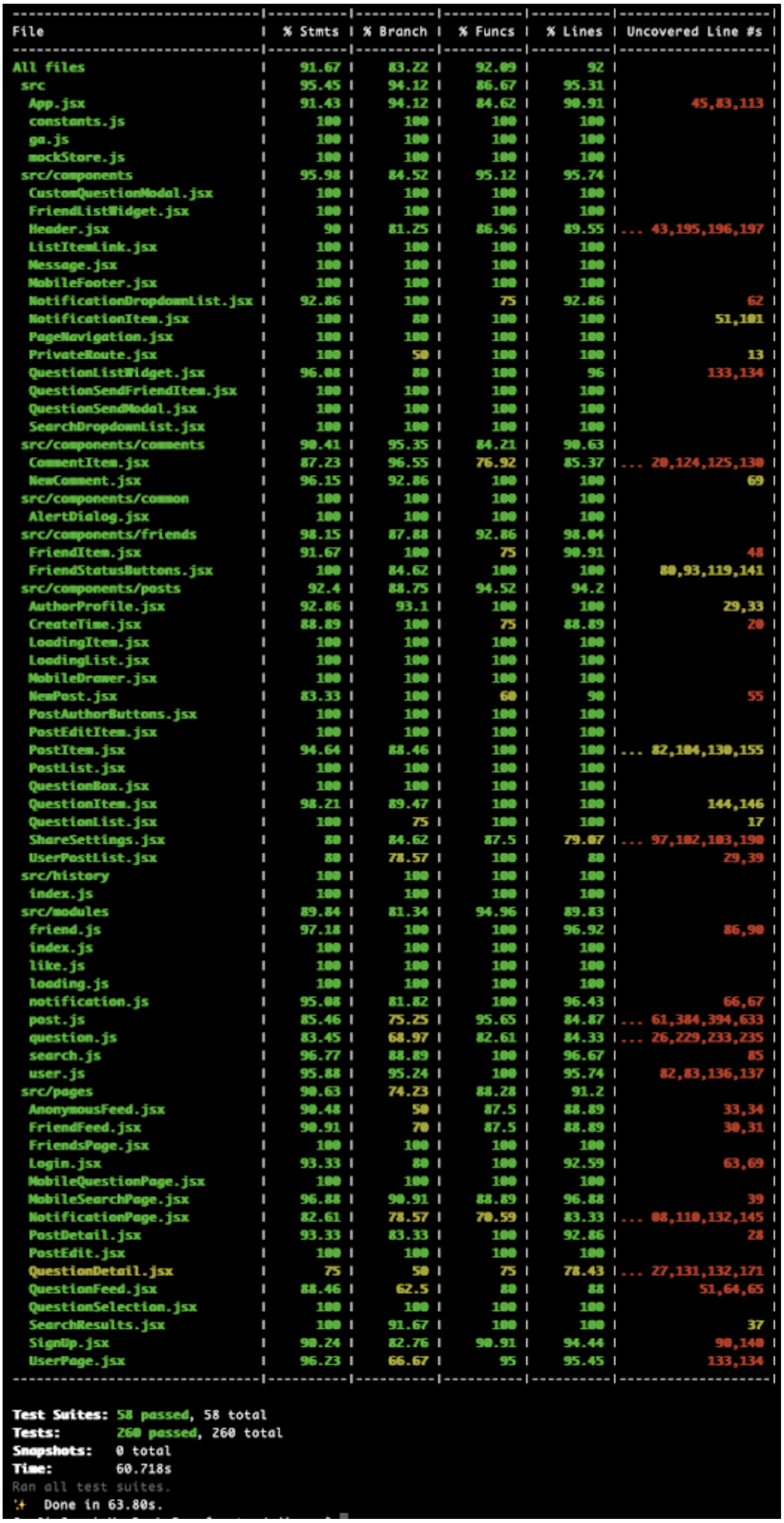
Furthermore, Pylint was used to perform minor error checks in addition to errors when building or compiling. In conducting every test we implemented, we managed a Pylint score of 10.00 out of 10.00.



```
(swpp-env) (base) gimjaewon@gimjaewon-ui-MacBookPro adoorback % pylint **/*.py --load-plugins pylint_django

------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

B. Frontend

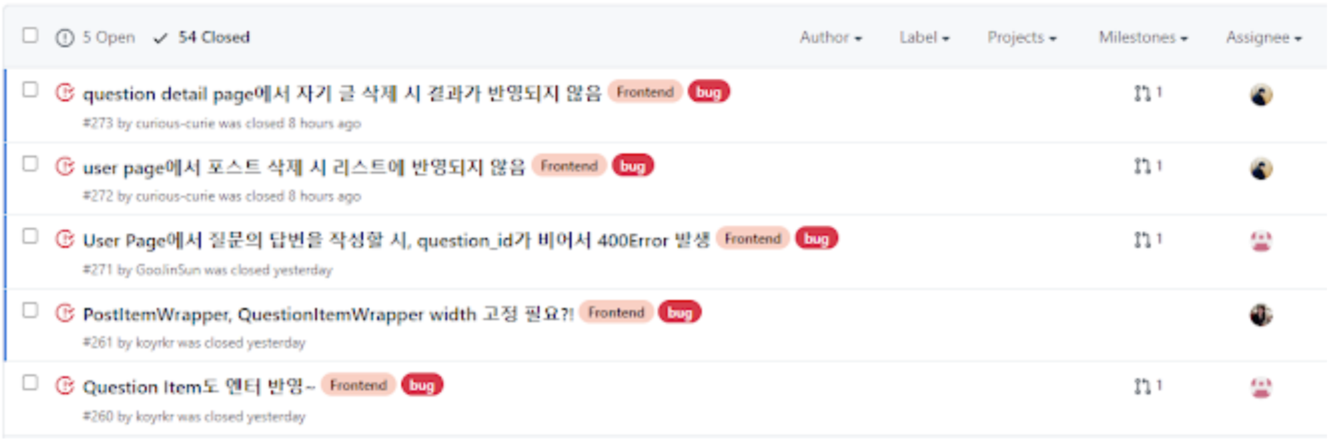
We mainly used Jest and Enzyme to test all the components, pages and modules including reducers on implemented actions and reducer state that changes. Whenever a certain component, page, and module were created, the test file (*.test.jsx, *.test.js) was created together to proceed in a kind of TDD(Test-driven Development) method. It was difficult to create a test case corresponding to infinite scrolling, but we tried to supplement the coverage as much as possible in other parts except for that. That is why the coverage of the branching part is relatively low.



```
-------------------------------|----------|----------|----------|----------|--------------------|
File                           |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s  |
-------------------------------|----------|----------|----------|----------|--------------------|
All files                      |    91.67 |    83.22 |    92.09 |       92 |                    |
 src                           |    95.45 |    94.12 |    86.67 |    95.31 |                    |
  App.jsx                      |    91.43 |    94.12 |    84.62 |    90.91 |          45,83,113 |
  constants.js                 |      100 |      100 |      100 |      100 |                    |
  ga.js                        |      100 |      100 |      100 |      100 |                    |
  mockStore.js                 |      100 |      100 |      100 |      100 |                    |
 src/components                |    95.98 |    84.52 |    95.12 |    95.74 |                    |
  CustomQuestionModal.jsx      |      100 |      100 |      100 |      100 |                    |
  FriendListWidget.jsx         |      100 |      100 |      100 |      100 |                    |
  Header.jsx                   |       90 |    81.25 |    86.96 |    89.55 |... 43,195,196,197 |
  ListItemLink.jsx             |      100 |      100 |      100 |      100 |                    |
  Message.jsx                  |      100 |      100 |      100 |      100 |                    |
  MobileFooter.jsx             |      100 |      100 |      100 |      100 |                    |
  NotificationDropdownList.jsx |    92.86 |      100 |       75 |    92.86 |                 62 |
  NotificationItem.jsx         |      100 |       80 |      100 |      100 |             51,101 |
  PageNavigation.jsx           |      100 |      100 |      100 |      100 |                    |
  PrivateRoute.jsx             |      100 |       50 |      100 |      100 |                 13 |
  QuestionListWidget.jsx       |    96.08 |       80 |      100 |       96 |            133,134 |
  QuestionSendFriendItem.jsx   |      100 |      100 |      100 |      100 |                    |
  QuestionSendModal.jsx        |      100 |      100 |      100 |      100 |                    |
  SearchDropdownList.jsx       |      100 |      100 |      100 |      100 |                    |
 src/components/comments       |    90.41 |    95.35 |    84.21 |    90.63 |                    |
  CommentItem.jsx              |    87.23 |    96.55 |    76.92 |    85.37 |... 20,124,125,130 |
  NewComment.jsx               |    96.15 |    92.86 |      100 |      100 |                 69 |
 src/components/common         |      100 |      100 |      100 |      100 |                    |
  AlertDialog.jsx              |      100 |      100 |      100 |      100 |                    |
 src/components/friends        |    98.15 |    87.88 |    92.86 |    98.04 |                    |
  FriendItem.jsx               |    91.67 |      100 |       75 |    90.91 |                 48 |
  FriendStatusButtons.jsx      |      100 |    84.62 |      100 |      100 |     80,93,119,141 |
 src/components/posts          |     92.4 |    88.75 |    94.52 |     94.2 |                    |
  AuthorProfile.jsx            |    92.86 |     93.1 |      100 |      100 |              29,33 |
  CreateTime.jsx               |    88.89 |      100 |       75 |    88.89 |                 20 |
  LoadingItem.jsx              |      100 |      100 |      100 |      100 |                    |
  LoadingList.jsx              |      100 |      100 |      100 |      100 |                    |
  MobileDrawer.jsx             |      100 |      100 |      100 |      100 |                    |
  NewPost.jsx                  |    83.33 |      100 |       60 |       90 |                 55 |
  PostAuthorButtons.jsx        |      100 |      100 |      100 |      100 |                    |
  PostEditItem.jsx             |      100 |      100 |      100 |      100 |                    |
  PostItem.jsx                 |    94.64 |    88.46 |      100 |      100 |... 82,104,130,155 |
  PostList.jsx                 |      100 |      100 |      100 |      100 |                    |
  QuestionBox.jsx              |      100 |      100 |      100 |      100 |                    |
  QuestionItem.jsx             |    98.21 |    89.47 |      100 |      100 |            144,146 |
  QuestionList.jsx             |      100 |       75 |      100 |      100 |                 17 |
  ShareSettings.jsx            |       80 |    84.62 |     87.5 |    79.07 |... 97,102,103,190 |
  UserPostList.jsx             |       80 |    78.57 |      100 |       80 |              29,39 |
 src/history                   |      100 |      100 |      100 |      100 |                    |
  index.js                     |      100 |      100 |      100 |      100 |                    |
 src/modules                   |    89.84 |    81.34 |    94.96 |    89.83 |                    |
  friend.js                    |    97.18 |      100 |      100 |    96.92 |              86,90 |
  index.js                     |      100 |      100 |      100 |      100 |                    |
  like.js                      |      100 |      100 |      100 |      100 |                    |
  loading.js                   |      100 |      100 |      100 |      100 |                    |
  notification.js              |    95.08 |    81.82 |      100 |    96.43 |              66,67 |
  post.js                      |    85.46 |    75.25 |    95.65 |    84.87 |... 61,384,394,633 |
  question.js                  |    83.45 |    68.97 |    82.61 |    84.33 |... 26,229,233,235 |
  search.js                    |    96.77 |    88.89 |      100 |    96.67 |                 85 |
  user.js                      |    95.88 |    95.24 |      100 |    95.74 |     82,83,136,137 |
 src/pages                     |    90.63 |    74.23 |    88.28 |     91.2 |                    |
  AnonymousFeed.jsx            |    90.48 |       50 |     87.5 |    88.89 |              33,34 |
  FriendFeed.jsx               |    90.91 |       70 |     87.5 |    88.89 |              30,31 |
  FriendsPage.jsx              |      100 |      100 |      100 |      100 |                    |
  Login.jsx                    |    93.33 |       80 |      100 |    92.59 |              63,69 |
  MobileQuestionPage.jsx       |      100 |      100 |      100 |      100 |                    |
  MobileSearchPage.jsx         |    96.88 |    90.91 |    88.89 |    96.88 |                 39 |
  NotificationPage.jsx         |    82.61 |    78.57 |    70.59 |    83.33 |... 08,110,132,145 |
  PostDetail.jsx               |    93.33 |    83.33 |      100 |    92.86 |                 28 |
  PostEdit.jsx                 |      100 |      100 |      100 |      100 |                    |
  QuestionDetail.jsx           |       75 |       50 |       75 |    78.43 |... 27,131,132,171 |
  QuestionFeed.jsx             |    88.46 |     62.5 |       80 |       88 |           51,64,65 |
  QuestionSelection.jsx        |      100 |      100 |      100 |      100 |                    |
  SearchResults.jsx            |      100 |    91.67 |      100 |      100 |                 37 |
  SignUp.jsx                   |    90.24 |    82.76 |    90.91 |    94.44 |             90,140 |
  UserPage.jsx                 |    96.23 |    66.67 |       95 |    95.45 |            133,134 |
-------------------------------|----------|----------|----------|----------|--------------------|

Test Suites: 58 passed, 58 total
Tests:       260 passed, 260 total
Snapshots:   0 total
Time:        60.718s
Ran all test suites.
✨  Done in 63.80s.
```

C. E2E testing per page

By allocating pages to each team member, we tried to conduct E2E testing as meticulously as possible. When a bug occurs, the bug is posted on the issue, and when the Bug Assignee resolves the bug, the issue is closed. For a more efficient work method, issues were posted with labels such as Frontend, Backend, Bug, and Enhancement like below.



5 Open ✓ 54 Closed — Author ▾ Label ▾ Projects ▾ Milestones ▾ Assignee ▾

question detail page에서 자기 글 삭제 시 결과가 반영되지 않음 Frontend bug
#273 by curious-curie was closed 8 hours ago — 1

user page에서 포스트 삭제 시 리스트에 반영되지 않음 Frontend bug
#272 by curious-curie was closed 8 hours ago — 1

User Page에서 질문의 답변을 작성할 시, question_id가 비어서 400Error 발생 Frontend bug
#271 by GoolinSun was closed yesterday — 1

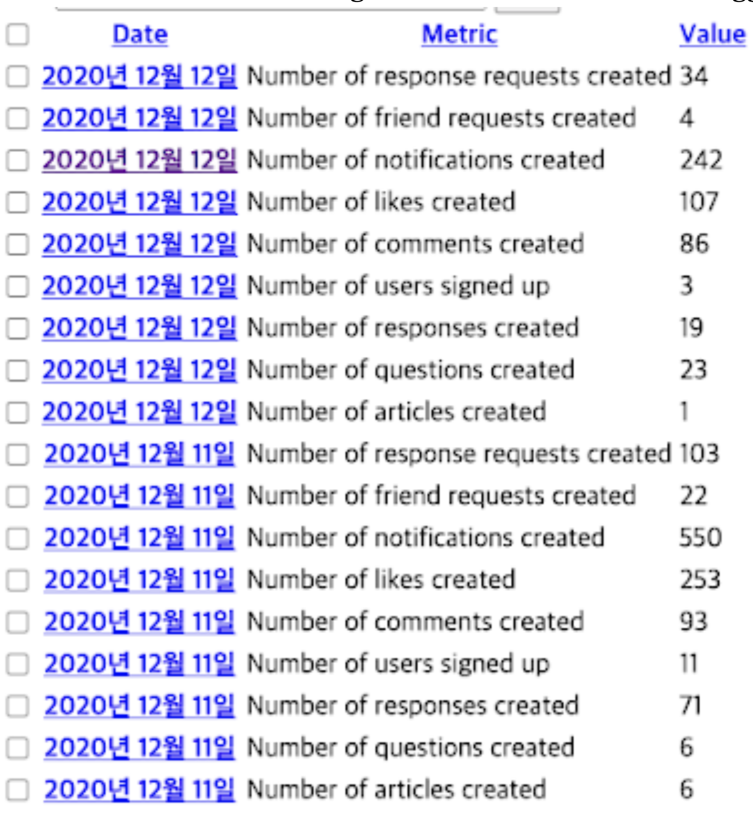
PostItemWrapper, QuestionItemWrapper width 고정 필요?! Frontend bug
#261 by koyrkr was closed yesterday

Question Item도 엔터 반영~ Frontend bug
#260 by koyrkr was closed yesterday — 1

V. Other Features

A. Model Tracking : trackstat

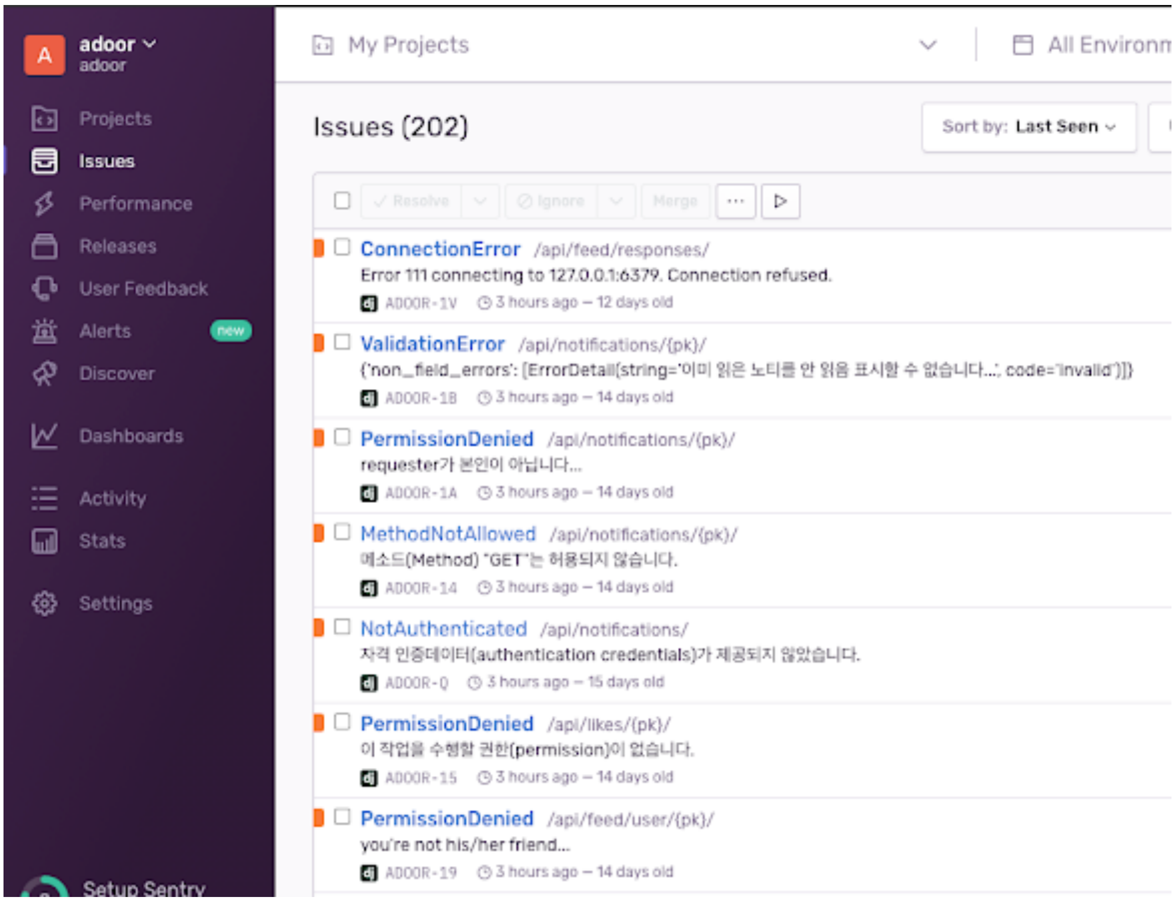
For each model, the number of models created during the day is tracked and logged. It produces statistics in a generic and structural fashion and organize the results of various aggregate queries.



Date	Metric	Value
2020년 12월 12일	Number of response requests created	34
2020년 12월 12일	Number of friend requests created	4
2020년 12월 12일	Number of notifications created	242
2020년 12월 12일	Number of likes created	107
2020년 12월 12일	Number of comments created	86
2020년 12월 12일	Number of users signed up	3
2020년 12월 12일	Number of responses created	19
2020년 12월 12일	Number of questions created	23
2020년 12월 12일	Number of articles created	1
2020년 12월 11일	Number of response requests created	103
2020년 12월 11일	Number of friend requests created	22
2020년 12월 11일	Number of notifications created	550
2020년 12월 11일	Number of likes created	253
2020년 12월 11일	Number of comments created	93
2020년 12월 11일	Number of users signed up	11
2020년 12월 11일	Number of responses created	71
2020년 12월 11일	Number of questions created	6
2020년 12월 11일	Number of articles created	6

B. Error Tracking : sentry

We established a software development process that can be managed continuously by collecting user error logs that occur in real time, using Sentry - error monitoring tool. Both backend and frontend errors and failed HTTP responses in API are all tracked and logged and managed.



Issues (202)

ConnectionError /api/feed/responses/
Error 111 connecting to 127.0.0.1:6379. Connection refused.
ADOOR-1V 3 hours ago – 12 days old

ValidationError /api/notifications/{pk}/
{'non_field_errors': [ErrorDetail(string='이미 읽은 노티를 안 읽음 표시할 수 없습니다...', code='invalid')]}
ADOOR-1B 3 hours ago – 14 days old

PermissionDenied /api/notifications/{pk}/
requester가 본인이 아닙니다...
ADOOR-1A 3 hours ago – 14 days old

MethodNotAllowed /api/notifications/{pk}/
메소드(Method) "GET"는 허용되지 않습니다.
ADOOR-14 3 hours ago – 14 days old

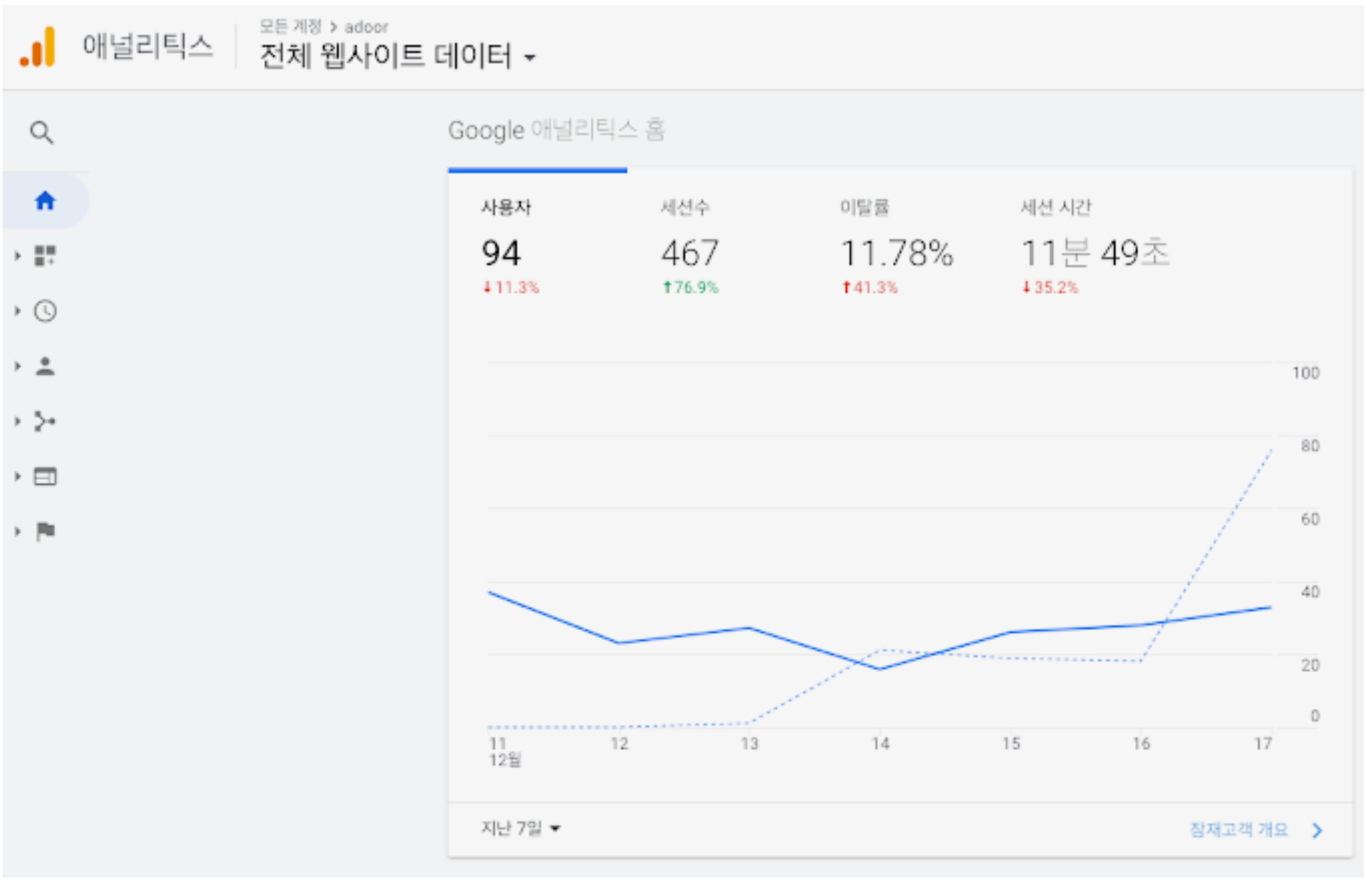
NotAuthenticated /api/notifications/
자격 인증데이터(authentication credentials)가 제공되지 않았습니다.
ADOOR-Q 3 hours ago – 15 days old

PermissionDenied /api/likes/{pk}/
이 작업을 수행할 권한(permission)이 없습니다.
ADOOR-15 3 hours ago – 14 days old

PermissionDenied /api/feed/user/{pk}/
you're not his/her friend...
ADOOR-19 3 hours ago – 14 days old

C. User Tracking : google analytics

We set up universal analytics & tracking code at the front-end side. We track pageviews and user behavior flow, and produce statistics provided by google analytics. We can also track sources and medium of users' inflow. We can analyze statistics such as the user's page path and how long the user stayed on each page using statistics in google analytics.



D. Chrome Extension

To allow for easier access and provide real-time notifications, we published a chrome extension for users. The app can be installed via the following link: [Chrome Extension] Link

VI. Difficulties & The Lessons Learned

A. API granularity (재원)

Due to some poor API granularity choices, most API calls required tens of exceptions cases and branches which not only slowed the calls down but also made the serializers and views extremely complex. The situation was only aggravated as we continued to add new features such as anonymous comments (which are not equivalent to private comments) and anonymous likes, all of which had to be protected with very specific and precise permissions. A better way to deal with this issue would have been to break the APIs down to smaller ones but due to time constraints, we decided to leave the API structure as is, though it may be a little inefficient.

B. Backend Model Refactoring

Our initial model design consisted of 14 different models with complex structure involving inheritance and polymorphism. Though the design offered a few merits, the code was a little too complex and messy. We then decided to refine our model design, and refactored our code. The sheer amount of time and effort required to implement and refactor the complex model structures with 100% test coverage and seeding was a challenge in itself. In accordance with the changed and refactored model relationship, the API or axios design that was written in Sprint 2 was also modified as well.

C. Efficient Infinite Scroll using Intersection Observer

In general, for infinite scroll implementation, the window.addEventListener function is used to continuously subscribe to the scrolling behavior of a window and to compute the scrolling position.

However, these methods result in quite large inefficiencies, in scroll events, where the 'offsetTop' is used to determine the current height value, each time a layout is drawn to bring the correct value. This causes browser performance degradation because it regenerates the render tree.

You also need to control the number of unnecessary function calls because continuous scrolling causes scroll events to occur several times a second. Therefore, it is necessary to implement and use methods such as debounce and throttle, which were difficult because of the need to use different modules or increase the number of codes.

Thus, instead of using this event subscription method, we implemented infinite scrolling using the web API, intersection observer. This is an API that asynchronously observes where the target element intersects the parent or the viewport of the target's parent or parent element, and eventually subscribes to whether the target element is exposed to the screen.

Intersection Observer's simple operating principles are as follows:

1. Create an 'Intersection Observer' object and deliver 'Callback Function' and 'option'.
2. Add 'Target Element' to subscribe from 'Intersection Observer' to 'observe'.
3. If the Target Element is exposed or excluded from the screen by the proportion defined as options.threshold, add it to the entry array and call callback function.
4. The callback function checks the received entry array and uses the isInterception property to check for exposure.
5. If you no longer need to subscribe to 'Target Element', you can remove it from the IntersectionObserver with ‘unobserve’.

Infinite scrolling has been used for almost all of the post list pages - home feed, anonymous feed, user page, notification feed, etc. Using this intersection observer, if the end of the list was exposed to the lower part of the screen, the next page API URL was called to add the following posts. There was a version-specific compatibility problem with intersection observer in Safari, which was resolved by installing polyfill. It was memorable because we could solve the problem using asynchronous web APIs without using the most common but inefficient methods.

D. Difficulty in implementing auto-login feature using CSRF tokens and JWT tokens

Features that should retain logged in status even when the browser or browser tab is turned off had to be implemented using csrf cookie and jwt token without saving the user's personal information on browser storage. There were several issues related to this feature.

If jwt refresh token remains in the cookie, there was a bug that kept the wrong cookie and redirected to the unauthorized home. This issue has been modified if the cookie is not valid, loading reducer handling the all api calls let user redirect to login page before redirect to home feed.

VII. Conclusion(재원)

Our service, adoor, aims to complement existing social media and better online social interactions by encouraging more diverse modes and depths of interactions. Our design welcomes users with friendly design and our Frontend and Backend implementations provides secure and comfortable user experience. We hope to further enhance this service by implementing easily customizable friend groups, enhancing API granularity, and developing a mobile application.

VIII. References

A. Sites referenced

- **Frontend**
    - React/Redux Tips: Better Way to Handle Loading Flags in Your Reducers : https://medium.com/stashaway-engineering/react-redux-tips-better-way-to-handle-loading-flags-in-your-reducers-afda42a804c6

- **Frontend Testing**
    - UI testing library docs : https://testing-library.com/docs/
    - Testing React Function Components with Hooks using Enzyme : https://medium.com/@acesmndr/testing-react-functional-components-with-hooks-using-enzyme-f732124d320a
    - jest-styled-components : https://github.com/styled-components/jest-styled-components

- **Authentication**
    - JWT vs. CSRF : https://mygumi.tistory.com/375
    - JWT Token : https://simpleisbetterthancomplex.com/tutorial/2018/12/19/how-to-use-jwt-authentication-with-django-rest-framework.html
    - Hackernoon - React Authentication : https://hackernoon.com/110percent-complete-jwt-authentication-with-django-and-react-2020-iejq34ta
    - Secret Key : https://dean-kim.github.io/django/2018/05/14/Django-Settings-Secret-Key.html

B. Used Framework & Libraries

- React : https://reactjs.org/
- Redux : https://redux.js.org/
- Jest : https://jestjs.io/en/
- Enzyme : https://enzymejs.github.io/enzyme/
- Django Rest Framework : https://www.django-rest-framework.org/
- Django Test Plus : https://django-test-plus.readthedocs.io/en/latest/
- Django Unit Test : https://docs.python.org/3.7/library/unittest.html
- Celery : https://docs.celeryproject.org/en/stable/django/first-steps-with-django.html
- Honepot : https://pypi.org/project/django-admin-honeypot/
- Pandas : https://pandas.pydata.org/
- PostgreSQL : https://www.postgresql.org/
- SQLite : https://www.sqlite.org/index.html

- Nginx : https://www.nginx.com/
- Uwsgi : https://github.com/unbit/uwsgi