Plain Plane

Final Report

Members

2015-10028 강민지, 2015-15894 원종훈, 2015-16535 박용훈, 2015-18634 이현종

Abstract

All those modern people who are living tough days, may want to share their honest feelings and thoughts, and comfort each other. General social media, however, where people inevitably expose and are exposed to glamorous lives, doesn't meet their needs. This was the starting point of this totally new SNS service: "Plain Plane".

"Plain Plane" provides a service for people to share honest feelings and thoughts anonymously. Here is the catchphrase:

"People plain about their feelings and thoughts on a plain paper,
fold up a paper plane with it, fuel up with plain yogurt, and fly it to the sky."

Introduction

The service consists of three essential parts: writing new content, replying to the writings, and sharing pictures of sky.

- Writing new content (for registered users only): Users can write about their feelings and thoughts anonymously and leave hashtags of the content. Lifespan of the plane depends on user level (flavor of yogurt). When there is a reply, users evaluate it, which affects the user level of the replier.
- Replying to the writings (for registered users only): Users can pick up the plane based on the radar (location based) or random plane list. Those who agreed to use geolocation service can use radar, and planes flown by location-agreed users are exposed on radar. When using radar, users choose distance: 5km, 25km, and 100km, then planes flown within the distance are shown. In random plane list, planes are shown randomly. Both feature supports refreshing the plane

list. Users decide which plane to pick up and reply by looking at the hashtags. Lifespan of replies are permanent.

- Sharing pictures of sky (appreciating for all users, uploading for registered users only): Users can view all photos randomly or search them by color (red, orange, yellow, green, blue, purple). Registered users can upload new photos of sky with hashtags. Only sky images can be uploaded, and non-sky images are blocked by visual recognition algorithm. Lifespan of photos are permanent.

And here is some additional information of the service:

- Flavor of the yogurt indicates user level. The number of writing and replying per day, and the lifespan of the plane vary based on the flavor.

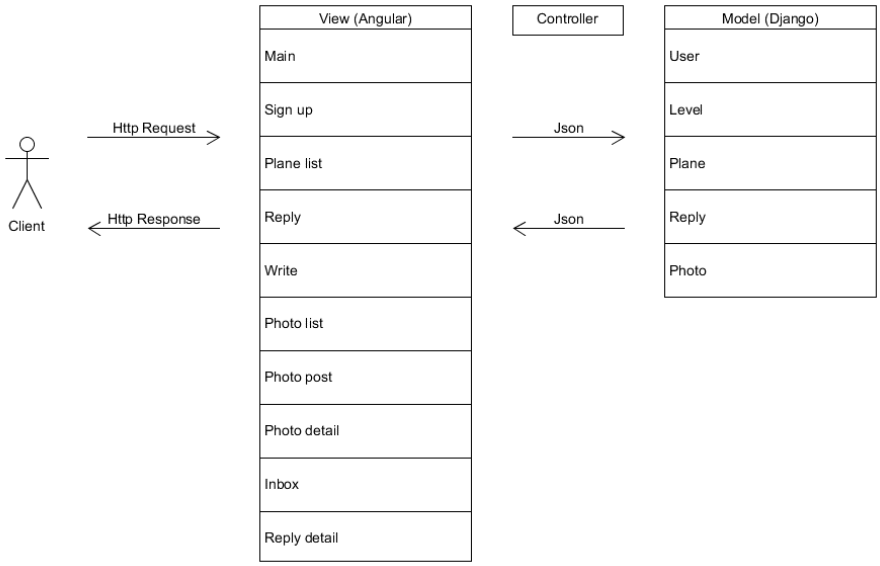
	Plain	Strawberry	Mango	Melon	Blueberry	Jasmine
Write	3	4	5	6	7	777
Reply	4	8	16	32	64	777
Lifetime(day)	2	3	5	8	13	30
Min Likes	0	5	20	50	100	1000

  Depending on evaluation users get(Like, Report), flavor can be promoted or degraded. More than 10 Reports will make user level become soy sauce yogurt, and the user will be banished from the service. These constraints make users write, pick, and reply sincerely.

- Users are required to pass reCaptcha when logging in or signing up.

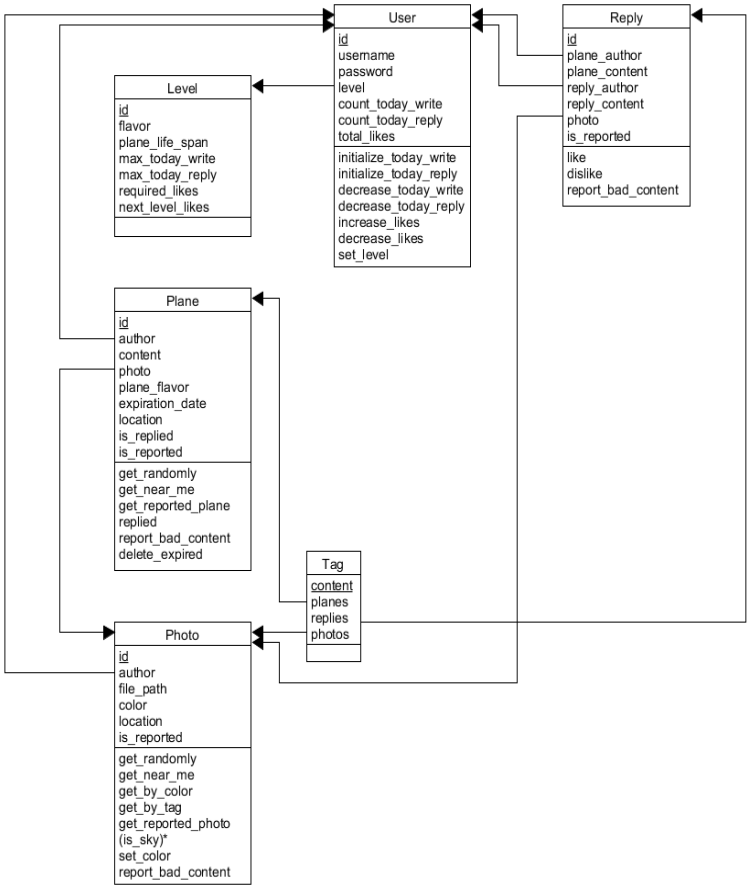
- There is optional email field when signing up. If users fill in the field, the users are required to pass email verification, and they can find password using the email. Initialized password can be changed in inbox page.

Design



Here is MVC of Plain Plane. There are 10 views and 5 models.

Model

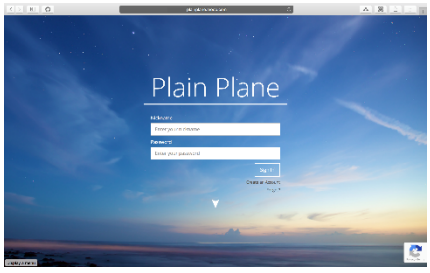
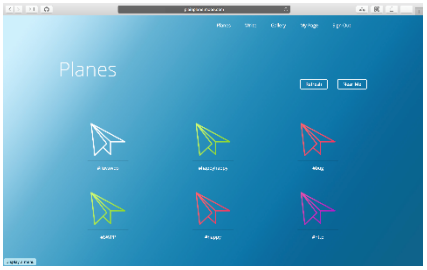
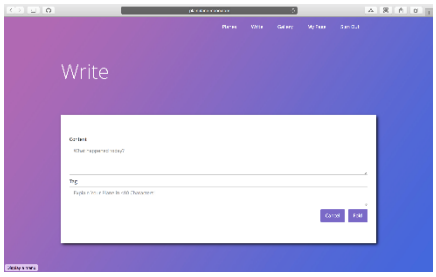
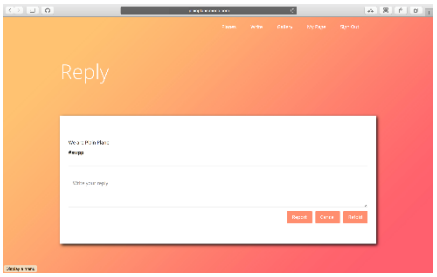
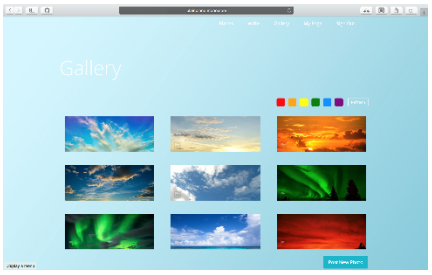
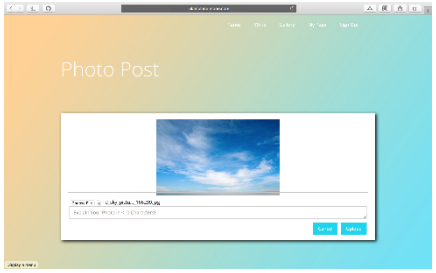
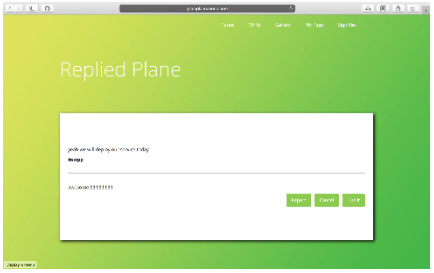
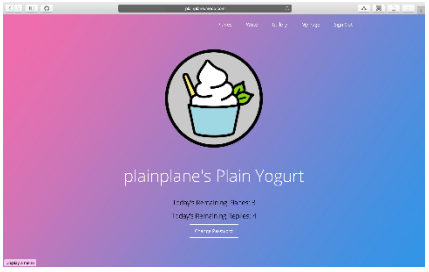


The service is to delete expired or replied Plane objects for database optimization, so Reply relation does not reference Plane relation. The "is_sky" method for Photo model classifies the picture is sky. We used 'Watson API' to classify it. It checks score of sky and ad and then fill out ads.

Since a tag is just for explaining plane and photo briefly, we did not made Tag model. We add 'tag' attributes to plane, photo and reply model.

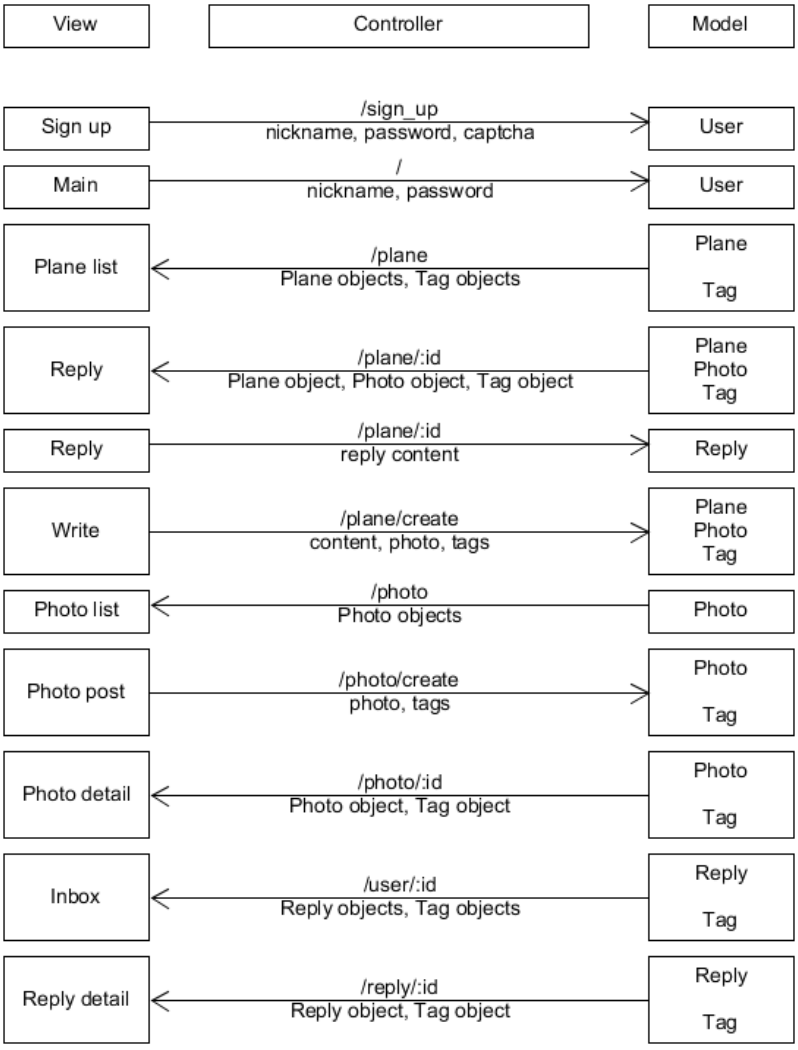
View

My page is added compare to Sprint 2. In my page, users can confirm their level and see replies of their planes. Users can also verify daily remaining planes, the number of write planes, and daily remaining replies. Below pictures are final view of our pages. Planes page, Gallery page, Write Page, and My page can be accessed by navigation bar.

Main Page(`/`) 	Sign Up (`/sign_up`) 	Planes Page(`/planes`) 
Write page (`/write`) 	Reply Page (`/plane/:id`) 	Gallery Page (`/gallery`) 
Photo Detail page (‘gallery/:id`) 	Reply Detail Page(`/reply/:id`) 	My Page (`/my_page`) 

Web-flow	When
Main page → Sign up	A user want to sign up
Main page ↔ Planes	A user sign in
Planes ↔ Reply	A user clicks the plane
Write → Planes	A user folds the plane
Gallery ↔ Photo detail	A user clicks the photo from gallery
My page ↔ Reply detail	A user clicks the replied plane from my page
* → Main page	A user sign out

Controller

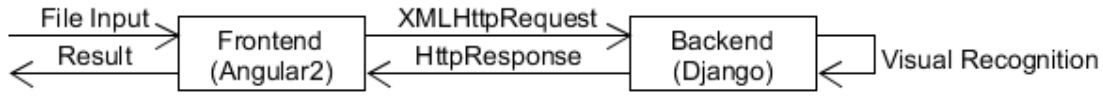


Left side is view part (frontend) and right side is model part (backend). Left-to-right arrow represents http request with user inputs from view, and reverse direction represents http response with data from model. Above the arrow, there is an API that controller uses to transfer JSON data below the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

Implementation

Gallery

The core part of the gallery implementation is sending files through request and receive the result of visual recognition algorithm by response.



When there is a file input, frontend first wraps the file with Observable, and checks size limit (2MB), then sends the file with "author_id" and "tag" through XMLHttpRequest(XHR). Observable waits for response from backend.

Backend restores the file from XHR using chunks, then test whether it is sky or not. Visual recognition algorithm is implemented by IBM Watson. Algorithm uses custom classifier "Sky Detection" which consists of "Sky" class and "Ad" class. The classifier is trained to detect whether the image is sky or not, and whether it is advertisement or not (especially, ads of which backgrounds are sky can also be detected). Minimum threshold for sky is 0.4 and maximum threshold for ad is 0.1.

After testing, backend sends HttpResponse with proper status. If the image is sky, the response status is 204 (No Content). Unless, the response status is 406 (Not Acceptable) and uploaded image is removed.

Observable detects state change and determines further action depending on XHR status code. If the code is 204, router navigates to gallery page. Unless, it alerts that uploaded image is not acceptable.

Since XHR follows same-origin policy, there was Cross-Origin Resourcing Sharing (CORS) issue when sending HttpResponse to frontend (Django doesn't handle CORS automatically regarding XHR). The problem was resolved by installing django-cors-headers application.

Testing

Unit Test

The unit tests cover all components and modules independently. Jasmine and Karma are used for frontend testing and Python unit test is used for backend testing. We achieved 90.2% of code coverage for frontend and 87.9% of code coverage for backend.

The initial goal of code coverage was over 90% for both frontend and backend, but we could not achieve 90% for backend. This is because it is difficult to test external APIs or file transfer. We will explain why some components have low coverages.

In frontend, Photo Post component has 64.52% of code coverage because it contains file transfer. Also, Plane Near Me component has 65.00% because it contains geolocation API.

All files

90.2% Statements `902/1000` **75.51%** Branches `299/396` **89.5%** Functions `213/238` **89.43%** Lines `812/908`

File	Statements		Branches		Functions		Lines	
src	100%	16/16	100%	0/0	100%	1/1	100%	16/16
src/app	98.53%	67/68	100%	0/0	85.71%	6/7	98.33%	59/60
src/app/change-password	100%	50/50	93.55%	29/31	100%	7/7	100%	47/47
src/app/find-password	98.53%	67/68	92.68%	38/41	100%	9/9	98.46%	64/65
src/app/main-page	100%	49/49	89.47%	17/19	100%	10/10	100%	46/46
src/app/models	92.59%	200/216	73.68%	28/38	87.5%	70/80	91.92%	182/198
src/app/my-page	100%	44/44	70.59%	12/17	100%	14/14	100%	40/40
src/app/navigation-bar	96%	24/25	70%	7/10	100%	8/8	95.45%	21/22
src/app/photo-detail	100%	25/25	73.33%	11/15	100%	7/7	100%	22/22
src/app/photo-list	90.63%	29/32	60%	6/10	90.91%	10/11	89.29%	25/28
src/app/photo-post	64.52%	40/62	53.57%	15/28	90.91%	10/11	60.71%	34/56
src/app/planes	85%	34/40	71.43%	15/21	81.82%	9/11	82.86%	29/35
src/app/planes-near-me	65%	26/40	64.71%	11/17	45.45%	5/11	65.71%	23/35
src/app/replied-plane	85%	34/40	64%	16/25	100%	10/10	83.33%	30/36
src/app/reply	78.85%	41/52	67.5%	27/40	75%	9/12	76.6%	36/47
src/app/sign-up	90.32%	84/93	83.05%	49/59	100%	12/12	89.89%	80/89
src/app/write	85.19%	46/54	72%	18/25	91.67%	11/12	84%	42/50
src/testing	100%	18/18	100%	0/0	100%	4/4	100%	11/11
src/testing/models	100%	8/8	100%	0/0	100%	1/1	100%	5/5

In backend, *gallery/classifier* has 33% of code coverage because the code uses Watson classifier API to classify whether the photo is of sky. *gallery/viewsets* has 60% because it contains file upload so that complicated file handling files is needed. Both *user/view* and *user/viewset* contain RECAPCHA API to verify users so they have 44%, 48% of code coverage each.

Module ↓	*statements*	*missing*	*excluded*	*coverage*
backend/__init__.py	0	0	0	100%
backend/settings.py	29	0	0	100%
backend/urls.py	6	0	0	100%
backend/wsgi.py	4	4	0	0%
gallery/__init__.py	0	0	0	100%
gallery/admin.py	3	0	0	100%
gallery/classifier.py	18	12	0	33%
gallery/color_picker.py	33	1	0	97%
gallery/migrations/0001_initial.py	8	0	0	100%
gallery/migrations/__init__.py	0	0	0	100%
gallery/models.py	15	2	0	87%
gallery/serializer.py	7	0	0	100%
gallery/tests.py	78	0	0	100%
gallery/urls.py	6	0	0	100%
gallery/viewsets.py	50	20	0	60%
level/__init__.py	0	0	0	100%
level/admin.py	3	0	0	100%
level/migrations/0001_initial.py	6	0	0	100%
level/migrations/__init__.py	0	0	0	100%
level/models.py	24	0	0	100%
level/tests.py	36	0	0	100%
manage.py	13	6	0	54%
plane/__init__.py	0	0	0	100%
plane/admin.py	3	0	0	100%
plane/migrations/0001_initial.py	8	0	0	100%
plane/migrations/0002_auto_20171205_2047.py	6	0	0	100%
plane/migrations/0003_auto_20171218_0714.py	6	0	0	100%
plane/migrations/__init__.py	0	0	0	100%
plane/models.py	30	0	0	100%
plane/tests.py	137	0	0	100%
plane/urls.py	6	0	0	100%
plane/viewsets.py	133	4	0	97%
reply/__init__.py	0	0	0	100%
reply/admin.py	3	0	0	100%
reply/migrations/0001_initial.py	7	0	0	100%
reply/migrations/__init__.py	0	0	0	100%
reply/models.py	10	0	0	100%
reply/tests.py	107	0	0	100%
reply/urls.py	6	0	0	100%
reply/viewsets.py	82	0	0	100%
user/__init__.py	0	0	0	100%
user/admin.py	3	0	0	100%
user/migrations/0001_initial.py	9	0	0	100%
user/migrations/0002_auto_20171205_2047.py	6	0	0	100%
user/migrations/0003_auto_20171218_0322.py	6	0	0	100%
user/migrations/__init__.py	0	0	0	100%
user/models.py	35	0	0	100%
user/tests.py	111	8	0	93%
user/tokens.py	6	1	0	83%
user/urls.py	7	0	0	100%
user/views.py	18	10	0	44%
user/viewsets.py	145	75	0	48%
Total	**1229**	**143**	**0**	**88%**

Integration Test

The integrations test uses Travis CI and the code coverage is 86%.

Future Works

Developing Visual Recognition Algorithm

Currently, to determine whether the given photo by user is true sky image or not, we are using IBM's Watson Developer Cloud API. However, Watson's free plan service has many limitations, like daily serving limit, photo size limit, etc.

We are planning to train our own neural network using TensorFlow. By training our own local serving model, we can overcome the problems of Watson API listed above.

Yogurt Vending Machine

As our service's profit model, we are planning to add 'Yogurt Vending Machine' service. Using PayPal API, users can donate to our service in the form of buying new yogurt flavor in the vending machine.

Deploy in HTTPS

Our service fully implemented location-based services using geolocation API. However, almost every modern browser blocks geolocation API invocation when the service is deployed using HTTP connection. Therefore, location-based services are now supported in very limited condition.

We are planning to deploy our services in HTTPS so that every user can fully use location-based service in moderate conditions.

Lessons Learned

The biggest lesson from this project is "collaboration".

Deploying service requires magnificent exquisiteness. Thinking out idea is quite easy, but to connect an idea to actual service, there must present preceding conditions such as team's coordination, balance of work, specific planning, etc. It becomes more important for large-scale projects. Developing software together is like processing big data. While processing data, we need lots of strategies to solve a large single problem: how to divide data into small unit, how to process each unit data, and how to combine them. Likely, in team project, we need several strategies, like dividing work regarding each teammates' characteristics. We should define the given problem precisely, make detailed schedule based on that, and run that schedule properly. Also, we should handle various problems that we meet during sprints.

So far, we did a lot of personal programming projects, so the team project which started from planning to implementing was unfamiliar to us. The team project was stressful and difficult, but it was really valuable experience to realize what is needed for successful teamwork with meaningful outcome. Although the project was smaller scale than real projects in companies, we feel like finishing warming-up for projects we would encounter in future.

Thanks for passionate professor and TAs. We won't forget attending this class!