Plain Plane

Design and Planning

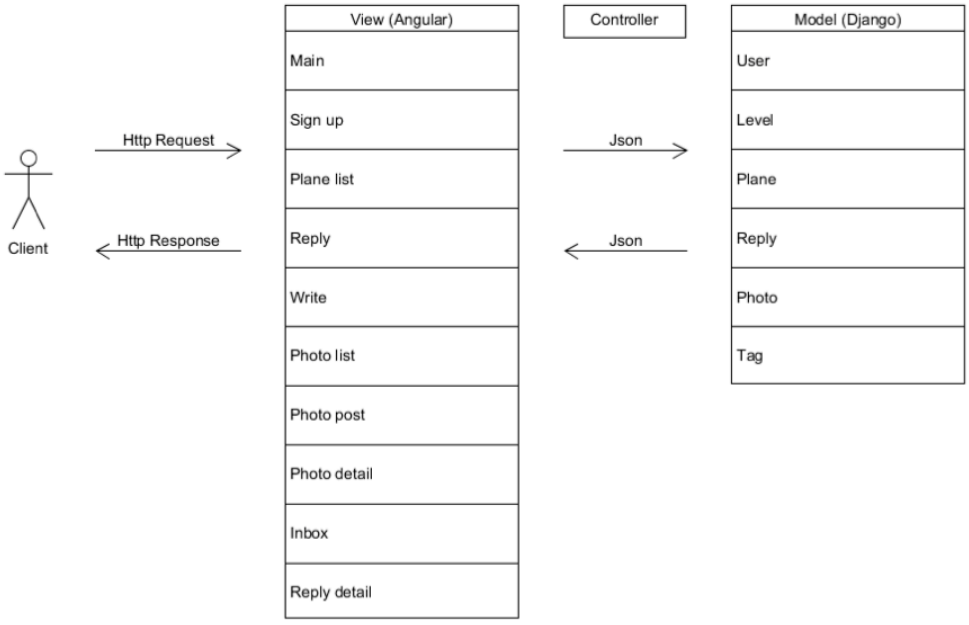
Rev. 1.0 2017-11-6 – initial version

Members

강민지, 원종훈, 박용훈, 이현종

System Architecture

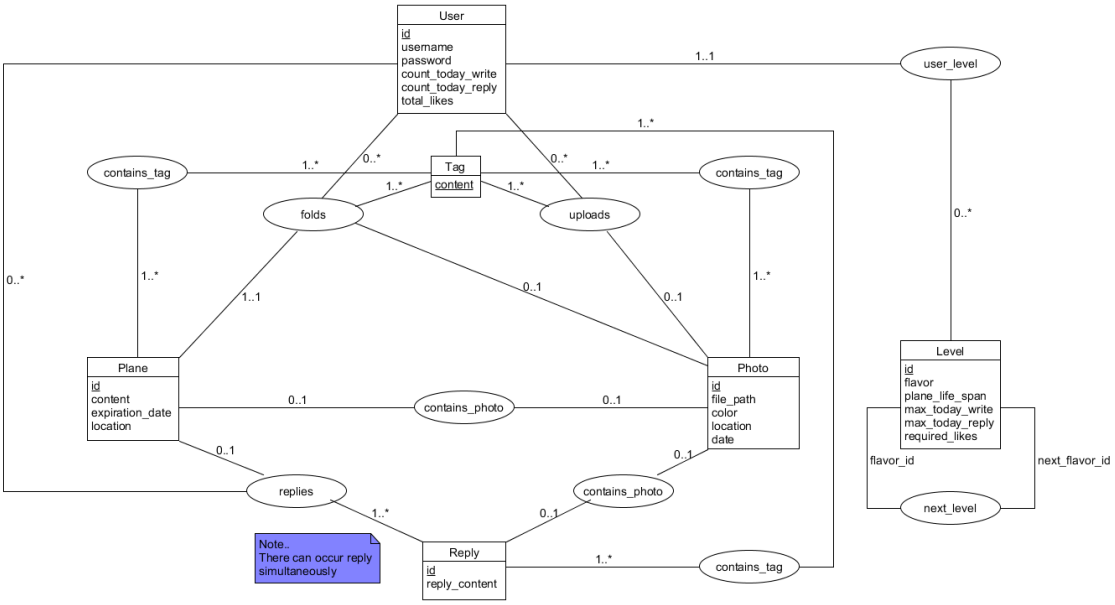
Here is MVC of Plain Plane.



There are 10 views and 6 models. Controller will be described further in Design Details part.

Model

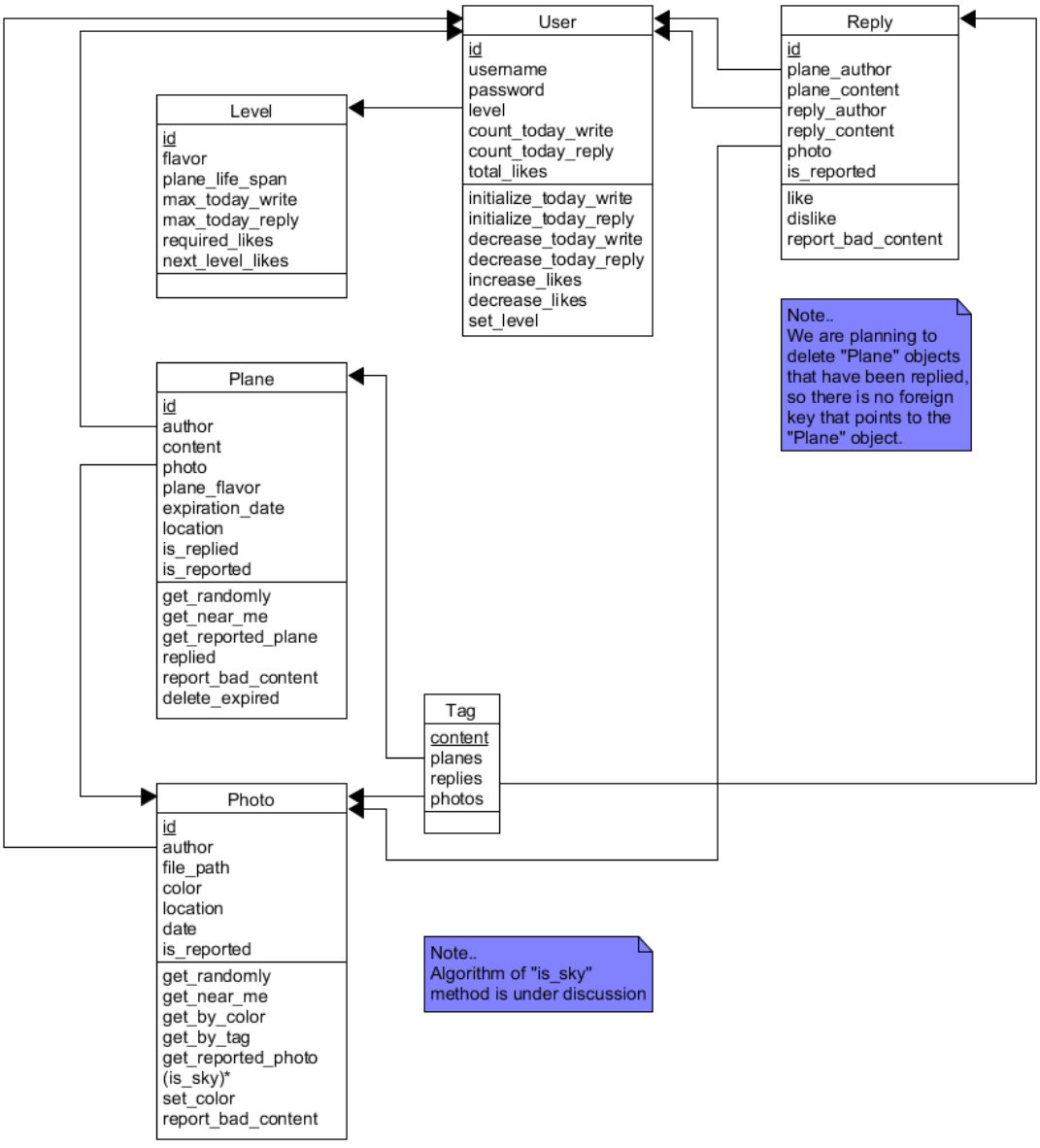
Here is E-R (Entity Relationship) diagram for model design.



Rectangle stands for entity set, oval stands for relationship set, and numbers next to line represent mapping cardinality constraints (min..max). Entity attributes are listed inside entity rectangle, and underlined attribute indicates primary key.

In principle, the service should not allow multiple replies to the same plane, but there can occur reply simultaneously in accident. Therefore, mapping cardinality between Plane entity and Reply entity is 1..*, not 1..1.

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

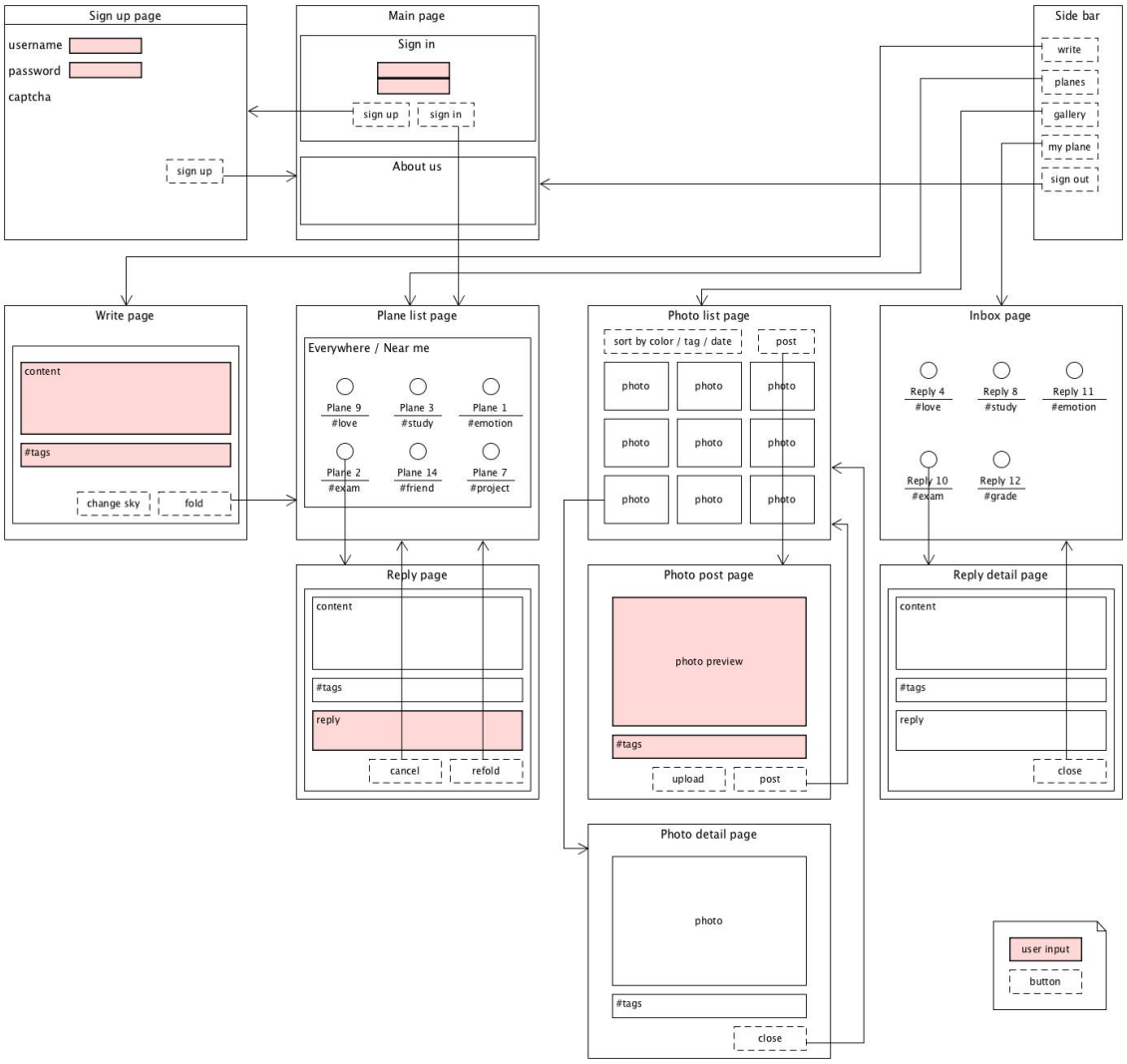


Rectangle stands for relation schema. Schema attributes are listed inside relation rectangle, followed by methods for Django models. Underlined attribute indicates primary key, and arrow represents foreign key constraints. Since Django saves whole object for foreign key, this diagram also follows the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

The service is to delete expired or replied Plane objects for database optimization, so Reply relation does not reference Plane relation. However, this is still ongoing issue and can be modified. Also, algorithm of “is_sky” method for Photo model is under discussion.

View

Here is user interface for view design.



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

1. Sign up page (`/sign_up`)
   - Sign up a new user
   - Get `nickname`, `password` as user inputs
   - Check if the user is robot by CAPTCHA API

2. Main page (`/`)
   - Sign in
   - About us: Describe about the Plane Plain service

3. Write page (`/plane/create`)
   - Create a plane.
   - Get `content`, `tags` of plane as user inputs
   - If the user clicks 'Change sky' button, change the background that the user uploads.
   - If the user clicks 'Fold' button, create the plane and increase `count_today_write`

4. Plane list page (`/plane`)
   - Show planes
   - There are 2 taps 'Everywhere' and 'Near me'
   - 'Everywhere' tap gets planes randomly and shows them.
   - 'Near me' tap gets planes that are close to the location of the user and shows them.
   - If the user clicks a plane, navigate to the Reply page

5. Reply page (`/plane/:id`)
   - Show the content, the tags of the selected plane and make the user reply to the plane.
   - By accessing this page, increase `count_today_reply`
   - Get `reply_content` as input
   - If the user clicks 'Cancel' button, navigate to the Plane list page
   - If the user clicks 'Refold' button, save the reply and navigate to the Plane list page

6. Photo list page (`/photo`)
   - Show photos
   - Sort photos by color / tag / date as the user clicks each option. (default: date)
   - If the user clicks 'Post' button, navigate to the Photo post page.
   - If the user clicks a photo, navigate to the Photo detail page.

7. Photo post page (`/photo/create`)
   - Create a photo.
   - If the user clicks 'Upload' button, change the photo preview to the photo that the user uploads.
   - If the user clicks 'Post' button, create the photo and increase `count_today_write`

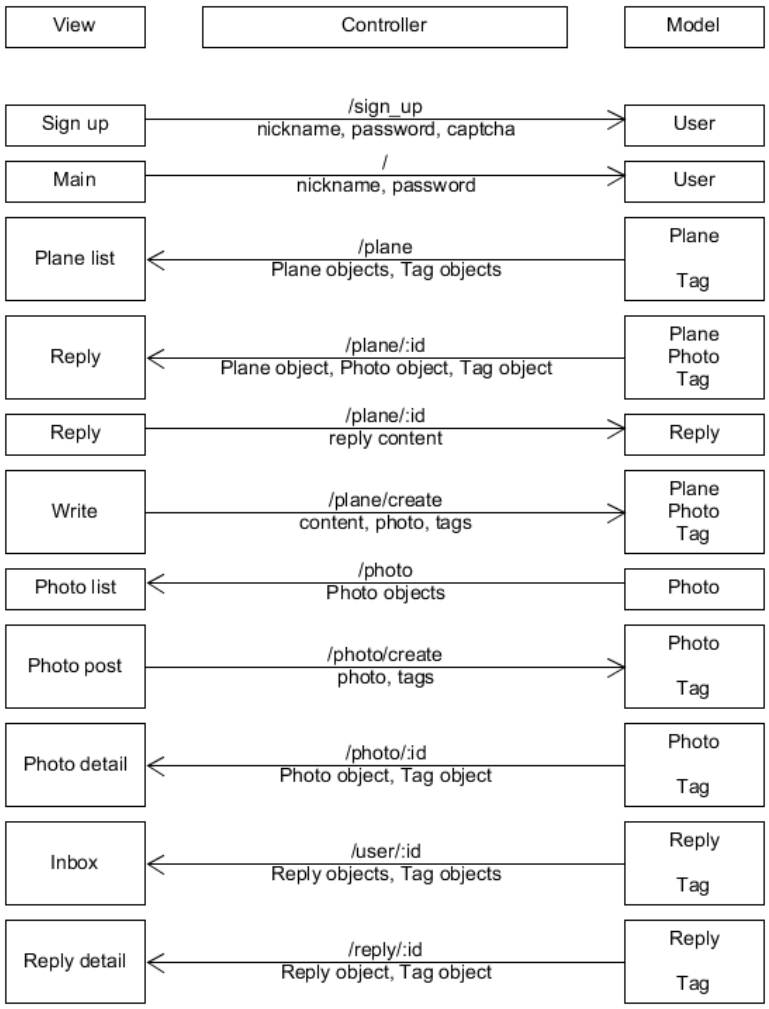
8. Photo detail page (`/photo/:id`)
   - Show the selected photo and the tags
   - If the user clicks 'Close' button, navigate to the Photo list page.

9. Inbox page (`/user/:username`)
   - Show replies that the user received.
   - If the user clicks a reply, navigate to the Reply detail page

10. Reply detail page (`/reply/:id`)
    - Show the selected Reply
    - Show `plane_content`, `tags` of original plane and the `reply_content`
    - If the user clicks 'Close' button, navigate to the Inbox page.

Controller

Here is controller design.



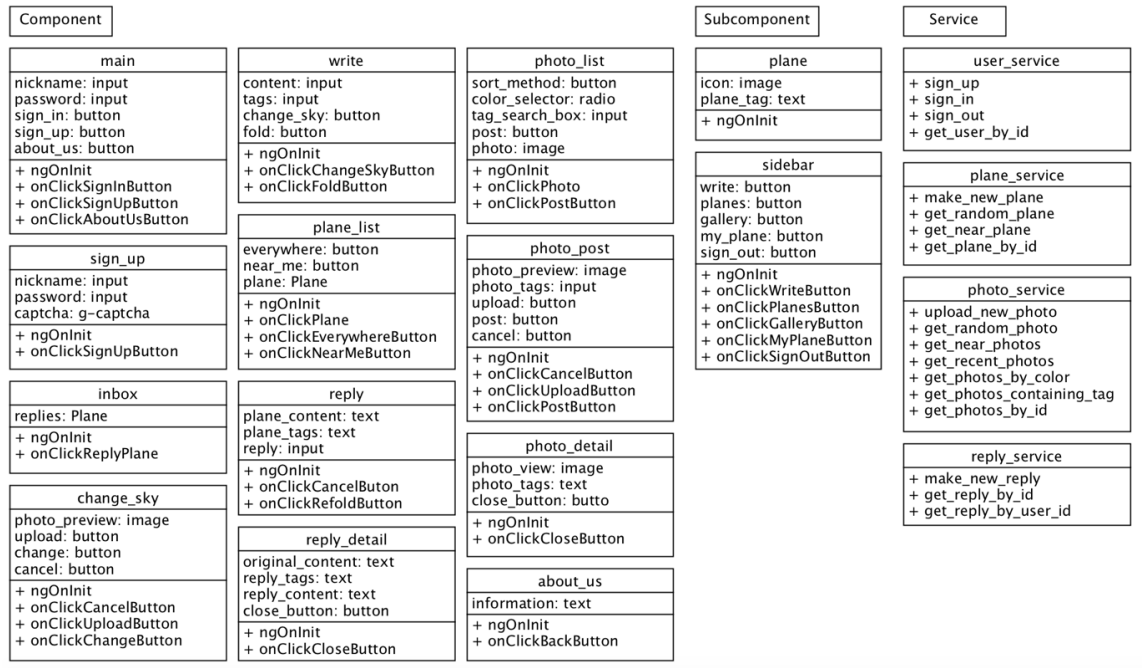
Left side is view part (frontend) and right side is model part (backend). Left-to-right arrow represents http request with user inputs from view, and right-to-left arrow represents http response with data from model. Above the arrow, there is an API that controller uses to transfer JSON data below the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

Design Details

Frontend Design

Frontend Components

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



Frontend Algorithms

Components

1. ngOnInit

- Check the user is logged in by calling backend api, and redirect to main page if user is not signed in.

2. main

- onClickSignInButton(username: string, password: string): Call backend signin api. If sign-in is accepted, redirect to the plane_list page. If sign-in is rejected, alert the user.
- onClickSignUpButton(): Redirect to sign_up page.
- onClickAboutUsButton(): Redirect to about_us page.

3. sign_up

- onClickSignUpButton(username: string, password: string): Make a new user. If username is already used, alert the user to use other username. If sign up is successful, redirect to the main page.

4. about_us
- onClickBackButton(): Redirect to main page

5. inbox
- ngOnInit(): Call getReplyById of ReplyService and show the reply planes.
- onClickReplyPlane(replyPlaneId: number): Get reply content by using replyPlaneId, and show it to the user.

6. write
- onClickChangeSkyButton(): Show change_sky page.
- onClickFoldButton(content: string, tags: string): Make new plane. If failed, alert the user. If success, Decrement the writing limit by one, and redirect to plane_list page.

7. change_sky
- onClickCancelButton(): Redirect to write page.
- onClickUploadButton() -> photo_id: Call System photo upload api. Make new photo database, and return the photo_id. Redirect to change_sky page with the uploaded photos preview shown.
- onClickChangeButton() -> photo_id: Redirect to the write page with write page background changed using the photo_id return value.

8. plane_list
- ngOnInit(): Depending on the current tab status(near or everywhere), get near planes and show them. Show only whose 'replied' attribute is false.
- onClickPlane(planeId: number): Open reply page with the given planeId. Decrement the today opening number limit by one.
- onClickEverywhereButton(): Change tab to everywhere. Call the getRandomPlane service function and show those planes. Show only whose 'replied' attribute is false.
- onClickNearMeButton(): Change tab to near me. Call the getNearPlane service function an show those planes. Show only whose 'replied' attribute is false.

9. reply
- ngOnInit(): Show the plane content using the given planeId.
- onClickCancelButton(): Redirect to plane_list page.
- onClickRefoldButton(content: string): Make a new Reply data. Set the plane's lifetime till tomorrow, and plane's 'replied' attribute to true.

10. reply_detail
- ngOnInit(): Show reply contents by using given replyPlaneId.
- onClickCloseButton(): Redirect to inbox page.

11. photo_list
- ngOnInit(): Using getRecentPhotos method, show preview of photos.
- onClickPhoto(photoId: number): Show photo_detail page.
- onClickPostButton(): Show photo_post page.

12. photo_post
- onClickCancelButton(): Redirect to photo_list page.
- onClickUploadButton() -> photo_id: Call System photo upload api. Make new photo database, and return the photo_id. Redirect to photo_post page with the uploaded photos preview shown.
- onClickPostButton(): Redirect to photo_list page. Decrement the writing limit by one.

13. photo_detail
- ngOnInit(): Using the given PhotoId, show the photo.
- onClickCloseButton(): Redirect to photo_list page.

Subcomponents
1. plane
- ngOnInit(): Show the plane image with tags.

2. sidebar
- onClickWriteButton(): Redirect to write page.
- onClickPlanesButton(): Redirect to plane_list page.
- onClickGalleryButton(): Redirect to photo_list page.
- onClickMyPlaneButton(): Redirect to inbox page.
- onClickSignOutButton(): Call signOut API< and Redirect to main page.

Services
1. UserService
- signUp(username: string, password: string) -> Promise<{status: boolean, reason: enum}>: Call Backend signUp API, and return the result.
- signIn(username: string, password: string) -> Promise<boolean>: Call backend signin api, and return the result.
- signOut(): Call signout api.
- getUserById(userId: number) -> Promise<User>: Call backend api, and return the user.

2. PlaneService
- makeNewPlane(content: string, tags: string[]) -> Promise<{status: boolean, reason: enum}>: Call

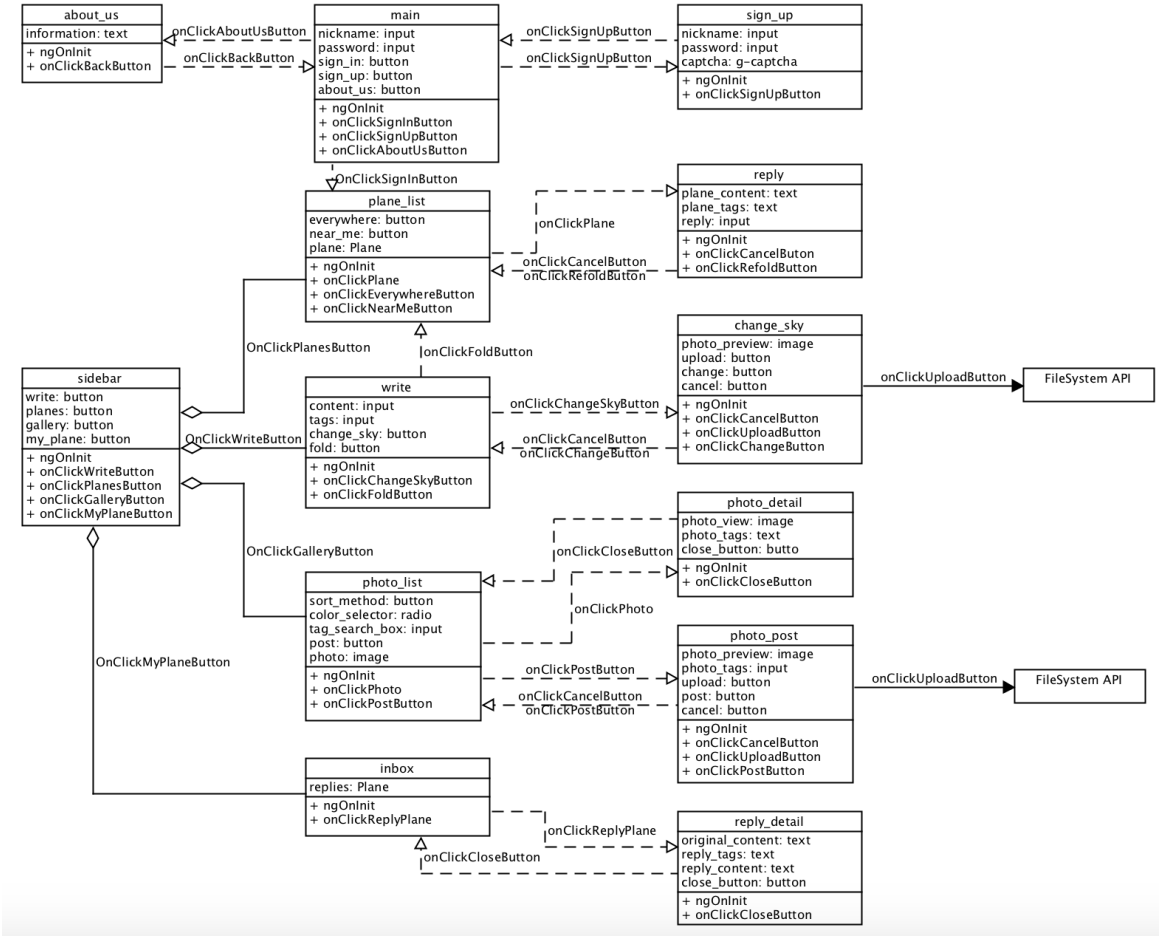
backend api to make new plane, and return the result.
- getRandomPlane(planesCount: number) -> Promise<Plane[]>: Call backend api to get specific number of planes.
- getNearPlane(userId: number, planesCount: number) -> Promise<Plane[]>: Call backend api to get specific number of planes near the user.
- getPlaneById(planeId: number) -> Promise<Plane>: Call backend api to get a specific plane.

3. PhotoService
- uploadNewPhoto() -> Promise<{status: boolean, reason: enum, photoId: number}>: Call backend api to upload new photo, and return the result.
- getRandomPhoto() -> Promise<string>: Call backend api to get a random photo link.
- getNearPhotos(userId: number, photosCount: number) -> Promise<string[]>: Call backend api to get specific number of photo links near the user.
- getRecentPhotos(photosCount: number) -> Promise<string[]>: Call backend api to get specific number of photo links, sorted randomly.
- getPhotosByColor(color: enum, photosCount: number) -> Promise<string[]>: Call backend api to get specific number of photo links, sorted by color.
- getPhotosContainingTag(tag: string, photosCount: number) -> Promise<string[]>: Call backend api to get specific number of photo links, containing tag string.
- getPhotosById(photoId: number) -> Promise<string>: Call backend api to get specific photo by photo id.

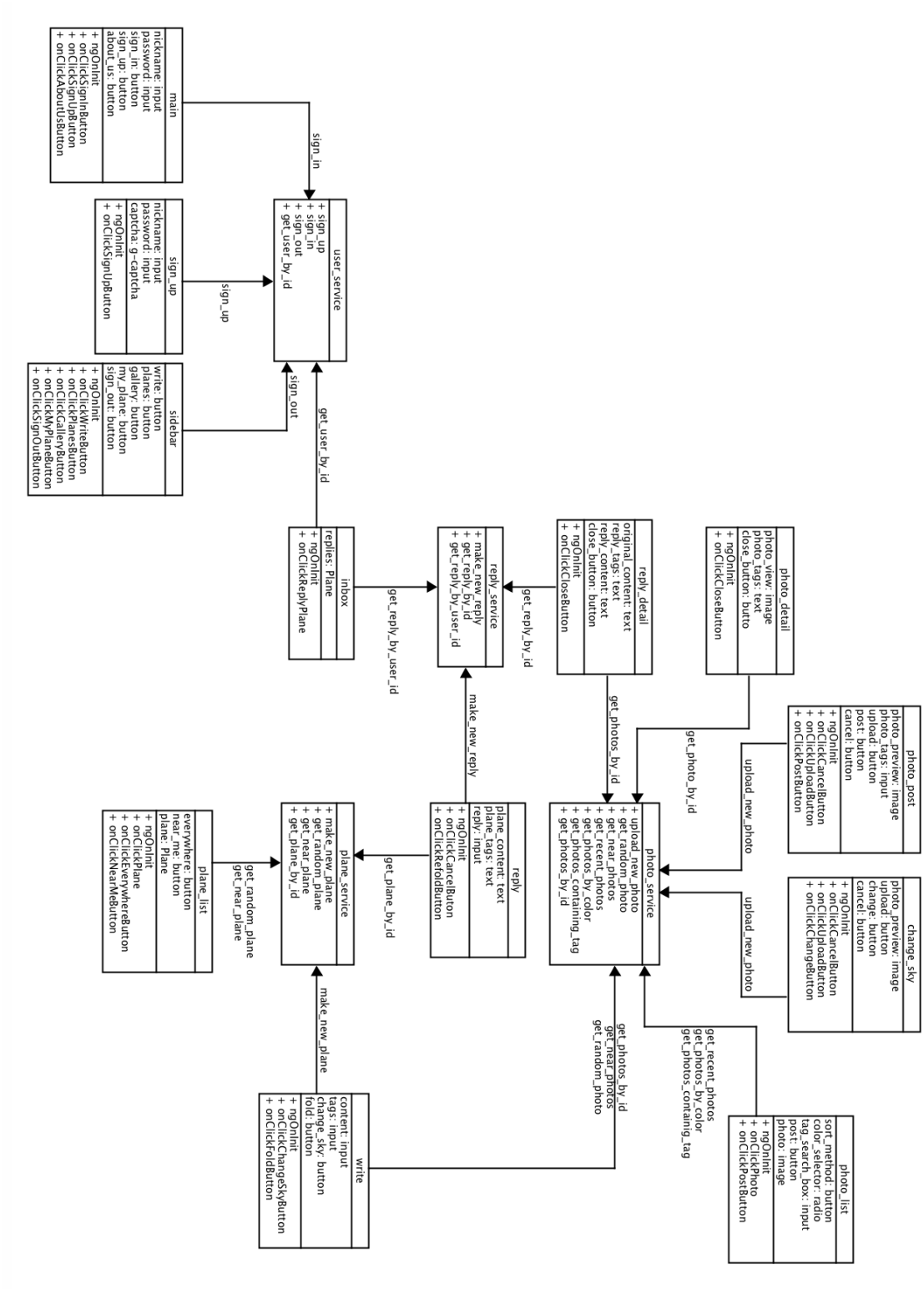
4. ReplyService
- makeNewReply(original_content: string, content: string) -> Promise<{status: boolean, reason: enum}>: Call backend api to make a new reply, and return the result.
- getReplyById(replyId: number) -> Promise<Reply>: Call backend api to get specific reply by id.
- getReplyByUserId(userId: number) -> Promise<Reply[]>: Call backend api to get replies specified by user id.

Frontend Relations

Here are the relations between components.



Here are the relations between services and components. (rotated for readability)



Backend Design

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

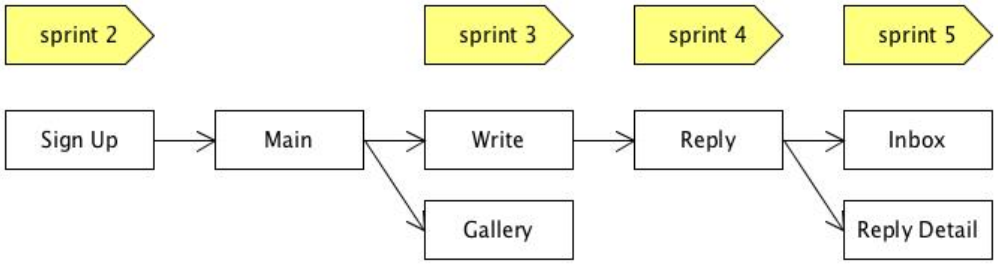
Detailed specifications of RESTful APIs are as following:

Model	API	GET	POST	PUT	DELETE
User	`/signup`	X	Create new user	X	**X**
	`/signin`	X	Log in	X	**X**
	`/signout`	Log out	X	X	**X**
	`/user/:id`	Get specific user	X	X	**X**
Plane	`/plane`	X	Create new plane	X	**X**
	`/plane/random`	Get plane list by random	X	X	**X**
	`/plane/location`	Get plane list by location	X	X	**X**
	`/plane/:id`	Get specified plane	X	Set reply flag	**X**
Photo	`/photo`	X	Create new photo	X	**X**
	`/photo/random`	Get photo list randomly	X	X	**X**
	`/photo/location`	Get photo list by location	X	X	**X**
	`/photo/color`	Get photo list by color	X	X	**X**
	`/photo/:tag`	Get photo list by tag	X	X	**X**
	`/photo/:id`	Get specified photo	X	X	**X**
Reply	`/reply`	X	Create new reply	X	**X**
	`/reply/:user_id`	Get reply list by user id	X	X	**X**
	`/reply/:id`	Get specified reply	X	Set seen/archive flag	**X**

Implementation Plan

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

The dependencies between tasks plays an important part in determining the order of development. We will develop 'Sign up' page and 'Main' page in sprint 2 which create the user and sign the user in. Then 'Write', 'Gallery', 'Reply', 'Inbox', 'Reply Detail' will be developed in order according to their dependencies.



The division is according to the difficulties of tasks and the estimates of the time for tasks. First, the difficulties of tasks are expressed as numbers between 1 and 5. Second, the estimates of the time for tasks are in minutes.

Page	**Feature**	**Difficulties (1-5)**	**Time (mins)**	**Sprints**	**Person**	**Challenge**
Main	Go to Sign up page	1	20	2	원종훈	
Main	Sign in and go to Reply page	2	60	2	원종훈	
Main	Go to About Us Page	1	30	2	강민지	
Sign up	Sign up and go to Main page	2	60	2	원종훈	**CAPTCHA API**
Write	Change the sky image	3	180	3	이현종, 박용훈	**upload photo and change the background**
Write	Fold the Plane	2	60	3	강민지	
Gallery	See Photos of sky	4	180	3	이현종, 박용훈	**sort photos by color, tag, date**
Gallery	Upload Photo	2	120	3	강민지	**upload photo and change the preview**
Reply	See planes from 'Everywhere' and 'Near me'	5	240	4	강민지, 원종훈	**get planes by random/location**
Reply	Open and Reply the Plane	3	180	4	이현종, 박용훈	
Inbox	See Replies	3	120	5	원종훈	
Reply detail	See the Reply	2	60	5	이현종	

Testing Plan

Unit Testing

Every components and modules should be tested. In each sprint, we would test implemented modules by following frameworks. We expect the code coverage is over 90%.

- Angular2: Jasmine & Karma
- Django: Python unit test

Functional Testing

Every APIs should be tested. As in hw2 and hw3, we will use following frameworks and mock data to test. In sprint 3, we would cover test RESTful API about authentication, plane, and photo. In sprint 4, we would test API about reply.

- Angular2: Jasmine & Karma
- Django: Python unit test

Acceptance & Integration Testing

Since Cucumber automatically maps user stories into tests for user and provides testing without a human in the loop to perform the actions, we would use Cucumber for acceptance testing. We already wrote user stories by Gherkin in Sprint 1. We would test them in sprint 5. For integration testing, we will use Travis CI.

- Acceptance Testing: Cucumber
- Integration Testing: Travis CI