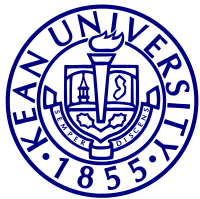
**SEE U**

**Old Code Review**



Hang Xie 1025910

Jiasheng Yu 1025957

Rui Chen 1025567

Jinhang Wu 1025891

**Version: (1.0)** **Date: (05/15/2020)**

**Table Content**

[**Design 3**](#_Toc40521877)

[**Functionality 3**](#_Toc40521878)

[**Post moments 3**](#_Toc40521879)

[**Exhibiting moments 6**](#_Toc40521880)

[**Likes posts 7**](#_Toc40521881)

[**Delete posts 9**](#_Toc40521882)

[**Ask for the reply from servlet 10**](#_Toc40521883)

[**Complexity 12**](#_Toc40521884)

[**Tests 12**](#_Toc40521885)

[**Naming 13**](#_Toc40521886)

[**Comments 13**](#_Toc40521887)

[**Style 13**](#_Toc40521888)

[**Consistency 14**](#_Toc40521889)

[**Documentation 14**](#_Toc40521890)

[**Good Things 14**](#_Toc40521891)

# 

# Design

I think the original design for posting moments, exhibiting moments, like posts and deleting posts is proper. The interactions are meaningful and the changes are codebase. Honestly, this application can be run on both Android and IOS devices. Because our group cannot implement the Flutter environment properly, so we think this is not the good time for us to add new functions. Moreover, the original code has nearly accomplished all the functions.

However, because we changed the IDE and developmental environment, we have to change the design of sending the data request to the back-end.

# Functionality

## Post moments

I think this function is developed successfully. Although our group cannot implement the code under Flutter developmental environment, the original group gives us the .apk file and we installed it. From the perspective of programmers, the code is not friendly to us but from the perspective of users, the application can get high ranks because they don’t need to see the source code.

The edge case we met is the developmental environment issue. Our group cannot implement the environment properly so we cannot test the code but the mobile application works well.

Because we cannot revise, we want to explain some details of the original source code to let you have a brief idea about this function.

This function was developed at “publish.dart”. The library “image\_picker” is used to access users’ photos and videos on their mobile phones. The below code is one of the key parts of the function which is called the StoreConnector. This generic object can protect the data if the app is closed accidentally.

StoreConnector<AppState, \_ViewModel>(

      converter: (store) => \_ViewModel(

            type: store.state.publish.type,

            text: store.state.publish.text,

            images: store.state.publish.images,

            videos: store.state.publish.videos,

          ),

**Figure 1: StoreConnector code**

Another import component in this function is PopupMenuButton. This one can help users change the type of moments they want to post. switchType() method is responsible for this function and text, image, and video types are defined.

PopupMenuButton<String>(

                  onSelected: (value) => \_bodyKey.currentState.switchType(

                        PostType.values.firstWhere(

                          (v) => v.toString() == value,

                        ),

                      ),

                  initialValue: PostType.image.toString(),

                  itemBuilder: (context) => PostType.values

                      .skip(1)

                      .map<PopupMenuEntry<String>>((v) => PopupMenuItem<String>(

                            value: v.toString(),

                            child: Text(PostEntity.typeNames[v]),

                          ))

                      .toList(),

**Figure 2: PopupMenuButton code**



**Figure 3: PostType Defination**

Next, I want to talk about the \_removeFile method. This method can dynamically decide the moment type and you can click the right corner cross to delete it.

\_removeFile(File file) {

    if (widget.vm.type == PostType.image) {

      widget.store.dispatch(PublishRemoveImageAction(

        image: file.path,

      ));

    } else if (widget.vm.type == PostType.video) {

      widget.store.dispatch(PublishRemoveVideoAction(

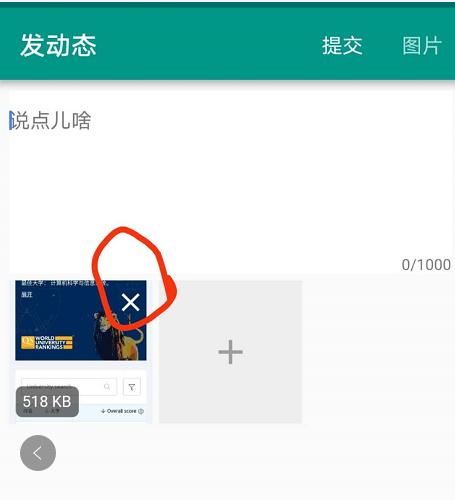
        video: file.path,

      ));

    }

  }

**Figure 4: removeFile code**



**Figure 5: Remove the selected file**

Also, in the old source old, the maximum photos are restricted within 6.

 if (widget.vm.images.length < 6) {

          children.add(GestureDetector(

            onTap: Feedback.wrapForTap(\_addFile, context),

            child: Container(

              width: width,

              height: height,

              color: WgTheme.greyLight,

              child: Center(

                child: Icon(

                  Icons.add,

                  color: WgTheme.greyNormal,

                  size: 32,

                ),

              ),

            ),

          ));

        }

**Figure 6: The code for restriction of the number of images**

## Exhibiting moments

For the same reason, I want to show you some functions built by the original code.

CircleAvatar() function is used to display the user’s avatar. Users can choose the avatar on their own or use the default one.

  child: CircleAvatar(

                    radius: 15,

                    backgroundImage: vm.creator.avatar == ''

                        ? null

                        : CachedNetworkImageProvider(vm.creator.avatar),

                    child: vm.creator.avatar == '' ? Icon(Icons.person) : null,

                  ),

**Figure 7: The code for user avatar**

This part is for username, "5" means the largest digit that can be shown next to the avatar.

If the number of username digits beyond 5, the exceeded part will be shown like “….”

 padding: EdgeInsets.all(5),

                      child: Text(

                        vm.creator.username,

                        overflow: TextOverflow.ellipsis,

                        style: TextStyle(

                          color: Theme.of(context).accentColor,

                          fontSize: WgTheme.fontSizeLarge,

**Figure 8: The code for username**

The below part is for showing the time.

  Text(

            widget.post.createdAt.toString().substring(0, 16),

            style: TextStyle(color: Theme.of(context).hintColor),

          ),

**Figure 9: The code for showing the post time**

The following part is for image or video display.

  return Wrap(

          spacing: margin,

          runSpacing: margin,

          children: images

              .asMap()

              .entries

              .map<Widget>((entry) => GestureDetector(

                  onTap: Feedback.wrapForTap(

                    () => Navigator.of(context).push(MaterialPageRoute(

                          builder: (context) => ImagesPlayerPage(

                                images: images

                                    .map<ImageEntity>((image) => image.original)

                                    .toList(),

                                initialIndex: entry.key,

                              ),

                        )),

                    context,

                  ),

**Figure 10: The code for showing the videos or images**

## Likes posts

Once the likes button is hit, a heart icon will become solid. “widget.post.isLiked” is used to check the status of likes button. When the user clicks the likes button again, the liked will be removed and this is what “unlikePost” do.

 widget.post.isLiked

                  ? GestureDetector(

                      onTap: Feedback.wrapForTap(

                        () => \_unlikePost(context, vm),

                        context,

                      ),

                      child: Container(

                        padding: EdgeInsets.all(5),

                        child: Icon(

                          Icons.favorite,

                          size: 20,

                          color: WgTheme.redLight,

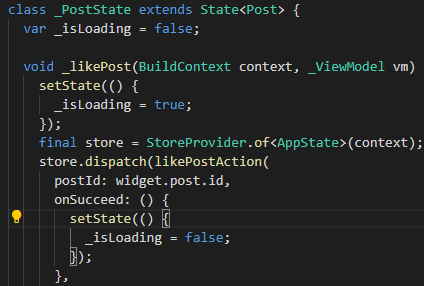
                        ),

                      ),

                    )

**Figure 11: The code for showing the likes button**

Once the moment is liked, an asynchronous method “\_likePost” is called. And the application will show you the loading information.



**Figure 12: The asynchronous method “\_likePost” to react to the operation of the like**

The storeConnector in pages/post/posts\_liked.dart, shows different users’ liked moments. If some users don’t like any moment before, it will return null.

return StoreConnector<AppState, \_ViewModel>(

      converter: (store) => \_ViewModel(

            userId: userId,

            postsLiked: (store.state.post.postsLiked[userId.toString()] ?? [])

                .map<PostEntity>((v) => store.state.post.posts[v.toString()])

                .toList(),

          ),

**Figure 13: StoreConnecttor code**

\_scrollListener can help user scroll down the list.

void \_scrollListener() {

    if (\_scrollController.position.pixels ==

        \_scrollController.position.maxScrollExtent) {

      \_loadPostsLiked();

    }

  }

**Figure 14: \_scrollListener code**

\_refresh method helps the user refresh the moments when they scroll down the page. The completer.future object will be sent to RefreshIndicator and the ListView will show the new posts.

  Future<Null> \_refresh() {

    final completer = Completer<Null>();

    \_loadPostsLiked(

      more: false,

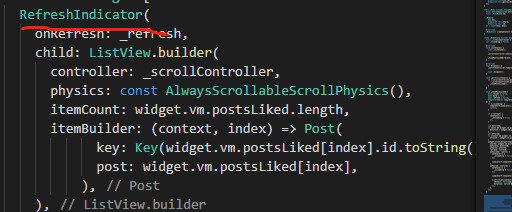
      refresh: true,

      completer: completer,

    );

    return completer.future;

  }



**Figure 15: The refresh method**

## Delete posts

The following function is used to display the delete model. We want to mention that only if the creatorID == user.id, the delete model can be visible.

Visibility(

                visible: widget.post.creatorId == vm.user.id,

                child: GestureDetector(

                  onTap: Feedback.wrapForTap(

                    () => \_deletePost(context, vm),

                    context,

                  ),

                  child: Container(

                    padding: EdgeInsets.symmetric(horizontal: 10, vertical: 5),

                    child: Icon(

                      Icons.delete\_outline,

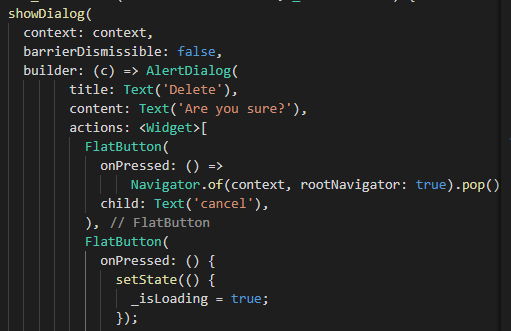
                      size: 20,

                      color: Theme.of(context).hintColor,

                    ),

**Figure 16: The visible method**

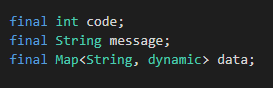
Once the \_deletePost method is called, it will go to the method body.



**Figure 17: \_deletePost method body**

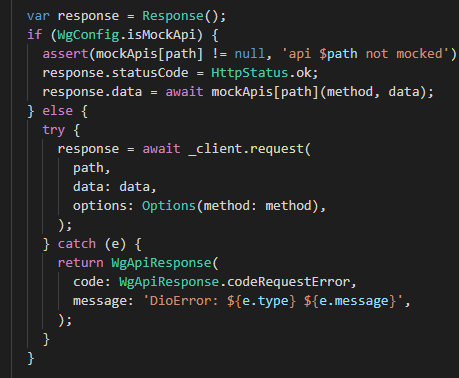
## Ask for the reply from servlet

The basic part of the back-end response is designed as the class "WgApiResponse”. Integer code is for status code, the message is used for description, and data is for business data.



**Figure 18: Data fields for back-end response**

When the original group test the code the above part of the below code was called. When the apk is export, the “try” part is used.



**Figure 19: response part**

The below methods are predefined when the requests are sent. It will automatically decide which method should be invoked.

Future<WgApiResponse> get(String path, {Map<String, dynamic> data}) async {

    return request('GET', path, data: data);

  }

  Future<WgApiResponse> post(String path, {Map<String, dynamic> data}) async {

    return request('POST', path, data: data);

  }

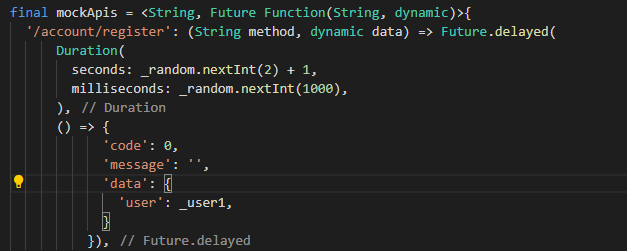
  Future<WgApiResponse> postForm(String path, {FormData data}) async {

    return request('POST', path, data: data);

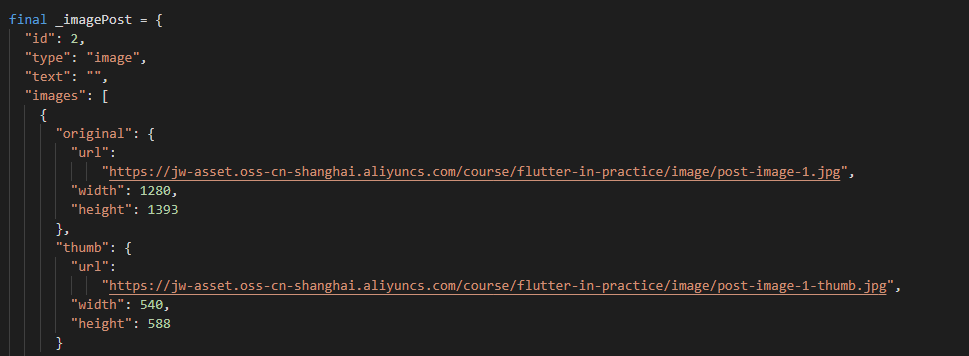
  }

**Figure 20: Encapsulation method**

The basic structure of one method “register”. Future.delayed() is used to simulate the time that real user needs to wait when they register.



**Figure 21: Encapsulated API structure**



**Figure 22: Test example of image post**

# Complexity

We should tell the truth that the Flutter framework is really complex but the old group has tried best to make each function developed in the easiest way.

As I mentioned before, our group cannot implement the environment properly so it’s likely to introduce bugs when we try to call or modify this code.

I don’t think this project is over-engineering. All the functions and codes are needed. So far, the only issue is our group cannot revise the code. No other issues are found here.

# Tests

We perform α test, β test, integration test, and end-to-end test. The application performs well but we need to connect our Tumblr account.

# Naming

All variables are named formally.

# Comments

The comments are not good. Although there are many English comments, very few comments explain why the codes are needed somewhere. I think lots of comments are not necessary, such as the below one:



**Figure 23: Comments are not necessary**

Actually, we hope the comments can provide some explanation about what does some functions do and what is the meaning of some functions?

# Style

We think the code style of Flutter they used is based on Google standard.

# Consistency

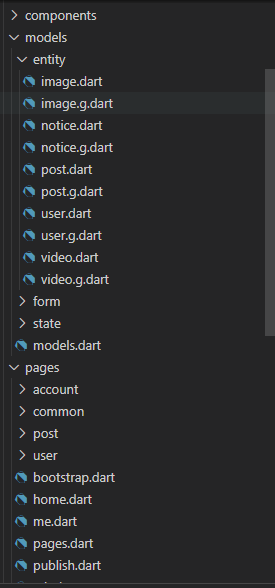
The style guide is based on <https://flutter.dev/docs/resources/design-docs>

# Documentation

For related documents please see <https://github.com/Sven97/SeeU>.

# Good Things

I think the original group has already tried their best to develop the application. I think they used all possible libraries to implement and this is the best thing. The structure is also the best I have seen from this course. Flutter is a really difficult framework and I think what they have done is the best among all groups. Deploy the Flutter environment is also tough, so I think they surprise me with the project integrity. We can run the application and it works well, although we cannot test their code. All functions are valid.



**Figure 24: the structure of the Flutter project**