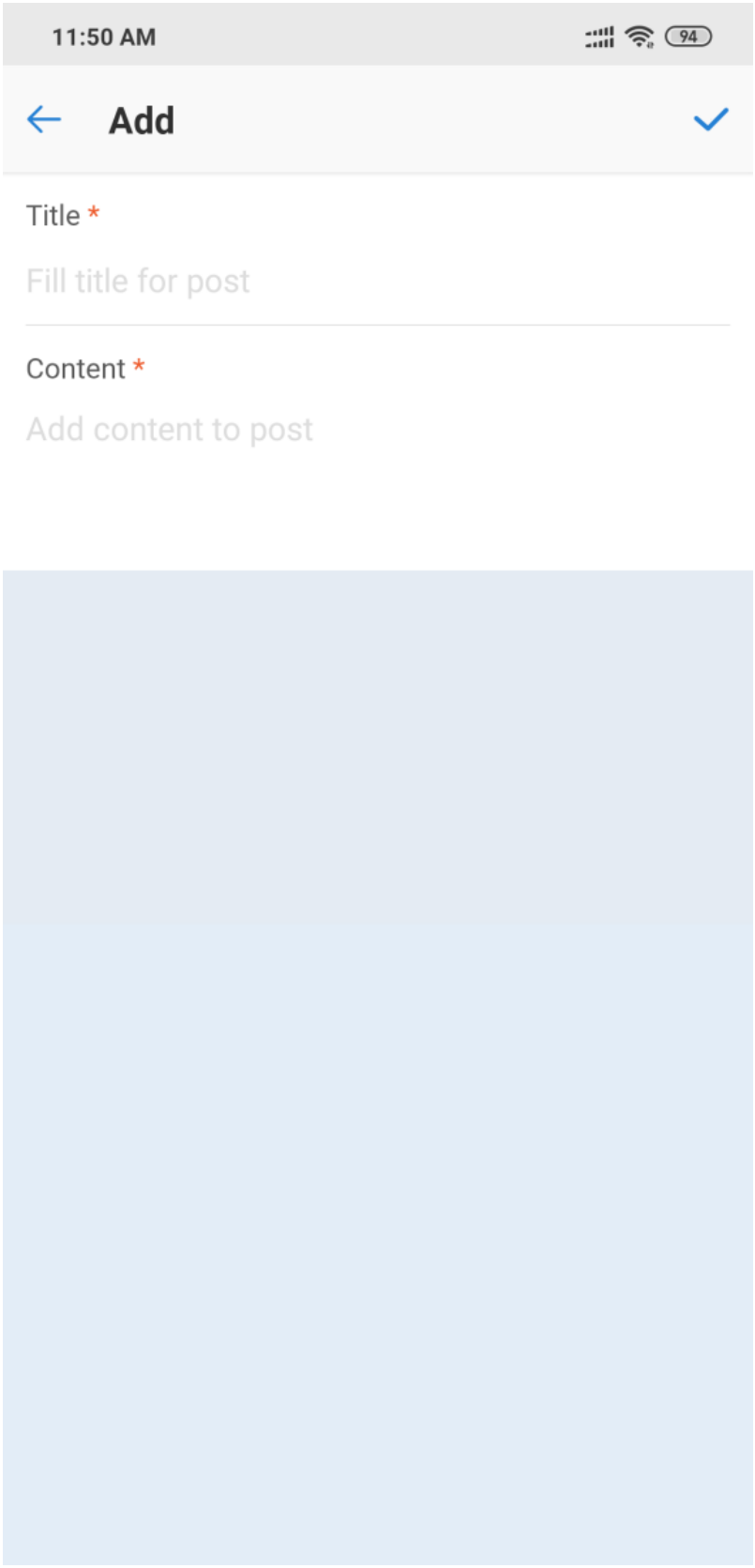
Tutorial 3: Implement Add new button and Create Post form

In order for users to create a new post, we need to add a button and navigate a user to the Create form.



Step 1: Add a button to the app's home screen

Extends the **PostApi::getAppSetting** method in the previous tutorial as follows
PostApi.php

```
/**
@param $param
@return MobileApp
*/
public function getAppSetting($param)
{
        $app = new MobileApp('post', [
                'title' => 'Posts',
                'home_view' => 'menu',
                'main_resource' => new PostResource([])
                ]);
        // Add create post button on App's home screen
        $app->addSetting('home.header_buttons', [
                'post' => [
                        [
                                'icon' => 'plus',
                                'action' => Screen::ACTION_ADD,
                                'params' => [
                                'resource_name' => 'post',
                                'module_name' => 'post'
                        ]
                ]
        ]);
        return $app;
}
```

Step 2: Create the creation form

Native Mobile App allows creating forms via API. The form API returns the structure of a form in JSON and base on the structure Mobile App automatic create a screen with form, fields and drive submit action.
The **GeneralForm** class as a *base class* helps build forms and handle submission, validation... You can create a Post form as following code
PostForm.php

```
<?php
namespace Apps\Posts\Api\Form;
use Apps\Core_MobileApi\Api\Form\GeneralForm;
use Apps\Core_MobileApi\Api\Form\Type\SubmitType;
use Apps\Core_MobileApi\Api\Form\Type\TextareaType;
use Apps\Core_MobileApi\Api\Form\Type\TextType;
use Apps\Core_MobileApi\Api\Form\Validator\StringLengthValidator;
class PostForm extends GeneralForm
{
/**
Override build form to generate form
@return mixed
@internal param array $data
*/
public function buildForm()
{
        $this
        ->addField('title', TextType::class, [
                'label' => 'title',
                'placeholder' => 'fill_title_for_post',
                'required' => true
                ], [new StringLengthValidator(5, 100)])
        ->addField('text', TextareaType::class, [
                'label' => 'content',
                'placeholder' => 'add_content_to_post',
                'required' => true
                ], [new StringLengthValidator(5)])
        ->addField('submit', SubmitType::class, [
                'label' => 'submit',
                'value' => 1
                ]);
        }
}
```

In the above code, we have added 3 fields *Title*, *Text* and *Submit* button to a form. The field type decides which native component will be used to generate the form element on the Mobile app.
The Phpfox Mobile App has a lot of *pre-defined form elements.*

Then, need to implement *Get Form API* of the current resource. Back to **PostApi** class and implement **PostApi::form()** method
PostApi.php

```
<?php
namespace Apps\Posts\Api\Service;
use Apps\Posts\Api\Form\PostForm;
use Apps\Posts\Api\Resource\PostResource;
class PostApi extends AbstractResourceApi implements MobileAppSettingInterface
{
/.../
        public function form($params = [])
        {
                $form = $this->createForm(PostForm::class, [
                        'title' => "Add a new post",
                        'action' => UrlUtility::makeApiUrl('post'),
                        'method' => 'post'
                ]);
                return $this->success($form->getFormStructure());
        }
}
```

Now check the *Get Form API*with API URL: **"url/restful_api/mobile/post/form?access_token=token"** and the API response should be returned as below
API response with form's structure

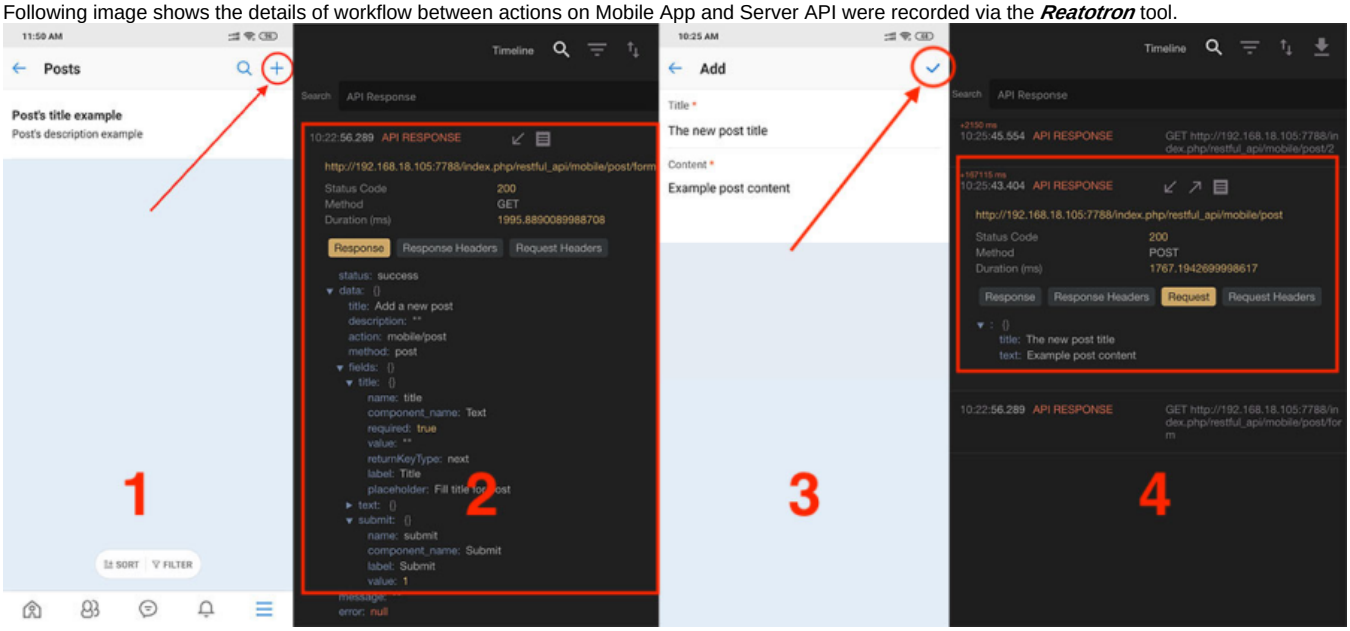
```
{
        "status": "success",
        "data": {
                "title": "Add a new post",
                "description": "",
                "action": "mobile/post",
                "method": "post",
                "fields": {
                        "title": {
                                "name": "title",
                                "component_name": "Text",
                                "required": true,
                                "value": "",
                                "returnKeyType": "next",
                                "label": "Title",
                                "placeholder": "Fill title for post"
                        },
                        "text": {
                                "name": "text",
                                "component_name": "TextArea",
                                "required": true,
                                "value": "",
                                "returnKeyType": "default",
                                "label": "Content",
                                "placeholder": "Add content to post"
                        },
                        "submit": {
                                "name": "submit",
                                "component_name": "Submit",
                                "label": "Submit",
                                "value": 1
                        }
                }
        },
        "message": "",
        "error": null
}
```

Step 3: Handle form submission

Above form's structure has defined the action and the method, it routes the form's submission to the corresponding API. Then the handling code can be implemented in **PostApi::create()** method.
PostApi.php

```
<?php
/* PostApi.php code */
/**
Create new document
*
@param $params
@return mixed
@throws \Exception
*/
function create($params)
{
        // Create PostForm instance
        $form = $this->createForm(PostForm::class);
        // Get submitted values and validate
        if (($values = $form->getValues()) && $form->isValid()) {
                // TODO: Your business code to save the values as a new post to database
                $newPostId = 2;
                // Return success and needed information to communicate with Native Mobile App
                return $this->success([
                        'id' => $newPostId,
                        'resource_name' => 'post',
                        'module_name' => 'post'
                ]);
        }
        return $this->validationParamsError($form->getInvalidFields());
}
```

Following image shows the details of workflow between actions on Mobile App and Server API were recorded via the ***Reatotron*** tool.



Step 4: Handling update the post

Similar to the workflow of creating a post, the code of the API form structure can be reused and the API call from the Mobile App will add ID parameter */post/form/post_id*. The Edit Post link will be defined in the action menu of each post.
The handling Update submission can be implemented in **PostApi::update()** method.