PHP Package Development

ဒီအပိုင်းမှာတော့ PHP အတွက် package တစ်ခုဖန်တီးပါမယ်။

Quiz Api Client

Quiz Api က linux တို့၊ docker တို့၊ php တို့ စတဲ့ tech နဲ့ ပတ်သတ်တာတွေကို muliple choice question တွေထုတ်ပေးတဲ့ api တစ်ခုပါ။ https://quizapi.io/ မှာ အကောင့်ဖွင့်ပြီး api token တစ်ခု create လုပ်ဖို့လိုပါတယ်။

Setup Composer Project

Project တစ်ခု create လုပ်ဖို့အတွက်

```
mkdir quiz-api-client
cd quiz-api-client
composer init
```

ဆိုပြီး run ပါမယ်။

```
Package name (<vendor>/<name>) [pyaesoneaung/quiz-api-client]:
```

Package name ကို `pyaesoneaung/quiz-api-client` ဆိုပြီး ထည့်ပါမယ်။ vendor မှာထည့်တာက vendor folder ကိုဖွင့်လိုက်ရင် စစတွေ့မယ့် folder name ပါ။ အများအားဖြင့် author နာမည်ကိုပဲထည့်ပါတယ်။ name မှာထည့်တာက package name ပါ။

```
Description []:
```

Description က ကိုယ်ကြိုက်တာထည့်လို့ရပါတယ်။

```
Author [Pyae Sone Aung <pyaesoneaung.code@gmail.com>, n to skip]:
```

Author ကလည်း သူပေးတဲ့ format နဲ့ ကြိုက်တာထည့်လို့ရပါတယ်။ ဘာမှမထည့်ချင်ရင် `n` နဲ့ skip လို့ရပါတယ်။

```
Minimum Stability []:
```

Minimum Stability က လိုအပ်တဲ့ dependency တွေကို ဘယ် version တွေ သွင်းမလဲဆိုတာ သတ်မှတ်တာပါ။ ဒီနေရာမှာ `dev` လို့ထည့်ပါမယ်။

```
Package Type (e.g. library, project, metapackage, composer-plugin)
[]:
```

ဒီနေရာမှာ `library` လို့ပဲထည့်ရပါမယ်။

```
License []:
```

License အကြောင်းတွေသိရင် ထည့်လိုက်ပါ။ မသိရင် ကျော်သွားလို့ရပါတယ်။

```
Define your dependencies.

Would you like to define your dependencies (require) interactively
[yes]?
```

no ဆိုပြီး ကျော်သွားလိုက်ပါ။

```
Would you like to define your dev dependencies (require-dev)
interactively [yes]?
```

no ဆိုပြီး ကျော်သွားလိုက်ပါ။

```
Add PSR-4 autoload mapping? Maps namespace
"Pyaesoneaung\QuizApiClient" to the entered relative path. [src/, n
to skip]:
```

enter နှိပ်ပေးလိုက်ပါ။

```
Do you confirm generation [yes]?
```

enter နှိပ်ပေးလိုက်ပါ။

အဲ့ဒါဆိုရင် အခုလို folder structure ရပါပြီ။

```
├── src
├── vendor
└── composer.json
```

composer.json မှာ အခုလိုပြင်ပါမယ်။

```
{
    "name": "pyaesoneaung/quiz-api-client",
    "description": "Quiz Api Client",
    "type": "library",
    "license": "MIT",
    "autoload": {
        "psr-4": {
-           "Pyaesoneaung\\QuizApiClient\\": "src/"
+           "PyaeSoneAung\\QuizApiClient\\": "src/"
        }
    },
    "authors": [
        {
            "name": "Pyae Sone Aung",
            "email": "pyaesoneaungrgn@gmail.com"
        }
    ],
    "minimum-stability": "dev",
+   "prefer-stable": true,
    "require": {}
}
```

"prefer-stable": true က **"minimum-stability": "dev"** ဖြစ်နေရင်တောင်မှပဲ dependency တွေထဲက stable အဖြစ်ဆုံး version ကိုပဲ ဦးစားပေးသွင်းမယ်ဆိုလိုတာပါ။

"PyaeSoneAung\QuizApiClient\": "src/" အဲ့အပိုင်းက ကျတော်တို့ src အောက်မှာ အသစ်ရေးမယ့် class တွေက **"PyaeSoneAung\QuizApiClient"** namespace အောက်ကသွားမယ်လို့ သတ်မှတ်လိုက်တာပါ။

ဘာမှမရေးခင် အရင်ဆုံး ကိုယ့် package က PHP version ဘယ်လောက်အနိမ့်ဆုံးလိုမယ်ဆိုတာ သတ်မှတ်ဖို့လိုပါတယ်။ ကိုယ်က modern syntax တွေသုံးချင်ရင် php-8.1 လောက်ကစသင့်ပါတယ်။

```
composer require php:^8.1
```

ဒါဆိုရင် composer.json မှာ

```
"require": {
    "php": "^8.1"
}
```

ဆိုပြီးတိုးသွားမှာပါ။ ဒီနေရာမှာ ^ လေးက 8.1 အထက်လို့ဆိုလိုတာ ကိုယ့်စက်မှာ php-8.2 သုံးနေရင်လည်း အဆင်ပြေမှာပါ။ တကယ်လို့ ကိုယ့်စက်က php-8.0 (သို့) php-7.2 ဖြစ်နေရင် သွင်းရမှာမဟုတ်ဘူး။ အဲ့ကျရင် အခုလိုပြင်ရေးဖို့လိုပါတယ်။

```
"require": {
    "php": "^7.2|^8.0"
}
```

`>=7.2` ဆိုလည်းရပါတယ်။ ကျတော်ကတော့ အဲ့လိုပေးတာ သိပ်သဘောမကျပါဘူး။ အဲ့လိုပေးလိုက်ရင် php-9.* ထွက်ခဲ့ရင်လည်း သွင်းလို့ရနေမှာ။ ကိုယ့် package က php-9.* မှာ support ပေးမပေးဆိုတာက ကြိုမသိနိုင်လို့ php-9.* ထွက်ခဲ့ရင် ကိုယ်တိုင် test လုပ်ပြီးမှ **"php": "^7.2|^8.0|^9.0"** ဆိုပြီး ထပ်ထည့်ပြီး version တစ်ခု release လုပ်တာက ပိုကောင်းပါတယ်။

PHP သွင်းပြီးသွားရင်တော့ Quiz api ခေါ်ဖို့ အတွက် Guzzle ကို သွင်းပါမယ်။

```
composer require guzzlehttp/guzzle:^7.0
```

ဒါဆိုရင် composer.json ရဲ့ require block မှာ အခုလိုတိုးသွားမှာပါ။

```
"require": {
    "php": "^8.1",
    "guzzlehttp/guzzle": "^7.0"
}
```

Api Integration

Quiz Api ကို integrate လုပ်ဖို့ အခုလိုရေးပါမယ်။

```
├── src
│   ├── Concerns
│   │   ├── BuildBaseClient.php
│   │   ├── CanSendGetRequest.php
│   ├── Resources
│   │   ├── QuestionResource.php
│   ├── QuizApi.php
├── vendor
└── composer.json
```

src/Concerns/BuildBaseClient.php

```php
<?php

namespace PyaeSoneAung\QuizApiClient\Concerns;

use GuzzleHttp\Client;

trait BuildBaseClient
{
    public function buildClient(): Client
    {
        return new Client([
            'base_uri' => 'https://quizapi.io',
            'headers' => [
                'X-Api-Key' => $this->apiKey,
            ],
        ]);
    }
}
```

src/Concerns/CanSendGetRequest.php

```php
<?php

namespace PyaeSoneAung\QuizApiClient\Concerns;

use GuzzleHttp\Client;
use Psr\Http\Message\ResponseInterface;

trait CanSendGetRequest
{
    public function get(Client $apiClient, string $url):
ResponseInterface
    {
        return $apiClient->request('GET', $url);
    }
}
```

src/Resources/QuestionResource.php

```php
<?php

namespace PyaeSoneAung\QuizApiClient\Resources;

use Pyaesoneaung\QuizApiClient\QuizApi;

class QuestionResource
{
    public function __construct(
        private readonly QuizApi $service,
    ) {
    }

    public function get(): array
    {
        $body = $this->service->get(
            apiClient: $this->service->buildClient(),
            url: '/api/v1/questions',
        )->getBody();

        return json_decode($body, true);
    }
}
```

src/QuizApi.php

```php
<?php

namespace PyaeSoneAung\QuizApiClient;

use PyaeSoneAung\QuizApiClient\Concerns\BuildBaseClient;
use PyaeSoneAung\QuizApiClient\Concerns\CanSendGetRequest;
use PyaeSoneAung\QuizApiClient\Resources\QuestionResource;

class QuizApi
{
    use BuildBaseClient;
    use CanSendGetRequest;

    public function __construct(
        private readonly string $apiKey
    ) {
    }

    public function questions(): QuestionResource
    {
        return new QuestionResource(
            service: $this
        );
    }
}
```

ကျတော် အရင်ကရေးဖူးတဲ့ Proper Way for Api Integration ထဲက ပုံစံအတိုင်းရေးထားတာပါ။ အသေးစိတ်ကို အခဲ့မှာ ဖတ်လို့ရပါတယ်။

ဒါဆိုရင် Quiz api ကို ဒီလိုခေါ်လို့ရပါပြီ။

```php
use PyaeSoneAung\QuizApiClient\QuizApi;

(new QuizApi($apiKey))->questions()->get();
```

ဒီလို syntax နဲ့ api ခေါ်ချင်လို့ အပေါ်ကလိုမျိုးရေးထားတာပါ။ Laravel ရေးနေကျဆိုရင် ဒီ syntax ကို မြင်တာနဲ့ QuizApi ကနေ question တွေအကုန် သွားခေါ်တယ်ဆိုတာ အလိုလိုခံစားမိမှာပါ။

ရေးပြီးတဲ့ code တွေ အလုပ်လုပ်မလုပ် စမ်းဖို့အတွက် root folder မှာပဲ `playground.php` ဆိုပြီး file တစ်ခု create လုပ်ပါမယ်။

`playground.php`

```php
<?php

require __DIR__.'/vendor/autoload.php';

use PyaeSoneAung\QuizApiClient\QuizApi;

$apiKey = 'real-api-key-here';
$data = (new QuizApi($apiKey))->questions()->get();
var_dump($data);
```

ပြီးရင်တော့ terminal ကနေ playground.php ကို run ပါမယ်။

```
php playground.php
```

ဒါဆိုရင် `$data` မှာ QuizApi ရဲ့ response array ဆိုရမှာပါ။

`require __DIR__.'/vendor/autoload.php';` က ကျတော်တို့ `composer.json` မှာ သတ်မှတ်ထားတဲ့ Namespace ကို resolve လုပ်တာတို့၊ vendor folder ထဲက package တွေကို သုံးလို့ရအောင်တို့ကို လုပ်ပေးတာပါ။

Testing

ကျတော်တို့ package အလုပ်လုပ်မလုပ် စမ်းဖို့အတွက် tests တွေရှိဖို့လိုပါတယ်။ Test ရေးဖို့ အတွက် `Pest PHP` ကို install လုပ်ပါမယ်။

```
composer require pestphp/pest --dev --with-all-dependencies
```

ဒီနေရာမှာ `--dev` က development လုပ်တဲ့အချိန်မှာပဲ သွင်းဖို့လိုတယ်လို့ ဆိုလိုတာပါ။ Package ကို release လုပ်ပြီးလို့ သူများတွေ install လုပ်တဲ့အချိန်မှာဆို `guzzlehttp/guzzle` ကိုပဲ dependency အနေနဲ့ သွင်းသွားမှာပါ။

ပြီးရင်တော့

```
./vendor/bin/pest --init
```

ဆိုပြီး run ပါမယ်။ ဒါဆိုရင် `tests` folder နဲ့ `phpunit.xml` file ကို generate လုပ်သွားပေးမှာပါ။ ဒါဆိုရင်

```
./vendor/bin/pest
```

ဆိုပြီး test တွေ run လို့ရပါပြီ။

`tests` folder ထဲမှာ Feature နဲ့ Unit ဆိုပြီး folder ၂ခု ရှိပါတယ်။ Package တွေမှာတော့ Feature သပ်သပ်၊ Unit သပ်သပ်ဆိုပြီး test တွေရေးတာထက် အပြင်မှာ class တစ်ခု create လုပ်ပြီးရေးတာမျိုးက ပိုများပါတယ်။ Test ရေးဖို့ အတွက် Feature နဲ့ Unit folder နှစ်ခုလုံးကို ဖျက်လိုက်ပါမယ်။ ပြီးရင် `Pest.php` မှာ အခုလိုပြင်ပါမယ်။

```php
<?php

use GuzzleHttp\Client;
use GuzzleHttp\Handler\MockHandler;
use GuzzleHttp\HandlerStack;
use GuzzleHttp\Psr7\Response;
use PyaeSoneAung\QuizApiClient\QuizApi;

function quizApi()
{
    return new QuizApi('foo');
}

function mockClient()
{
    $mock = new MockHandler([
        new Response(status: 200, body: json_encode(['foo' =>
'bar'])),
    ]);
    $handlerStack = HandlerStack::create($mock);

    return new Client(['handler' => $handlerStack]);
}
```

`Pest.php` မှာ ရေးတဲ့ helper function တွေကို test ရဲ့ ကြိုက်တဲ့နေရာက ခေါ်သုံးလို့ရပါတယ်။

`quizApi()` ကိုခေါ်ရင် `QuizApi` object return ပြန်မှာပါ။ ဒီနေရာမှာ api key ထည့်ဖို့မလိုပါဘူး။ ကျတော်တို့ code က Quiz Api ကို api သွားခေါ်နိုင်လားဆိုတာပဲ test ဖို့လိုတာပါ။ Quiz Api ကို key အစစ်ထည့်ပြီး တကယ်သွားခေါ်ဖို့ မလိုပါဘူး။ ဥပမာ တခြား developer တစ်ယောက် ဒီ

code တွေ အလုပ်လုပ်မလုပ် စမ်းဖို့အတွက် သူကိုယ်တိုင် အကောင့်ဖွင့်ပြီး api token ယူပြီး စမ်းနေဖို့မလိုပါဘူး။

`mockClient()` က guzzle client ကို mock လုပ်ဖို့ပါ။ mock လုပ်ထားတဲ့ client ကိုသုံးရင် api တကယ်သွားမခေါ်ဘဲ 200 response ကို ပြန်ရမှာပါ။ အဲ့အတွက် `MockHandler` ကို create လုပ်ပြီး `Client` မှာ pass ပေးထားတာပါ။

ပြီးရင်တော့ tests folder မှာပဲ `QuizApiTest.php` ဆိုပြီး class တစ်ခု create လုပ်ပါမယ်။

`tests/QuizApiTest.php`

```php
<?php

use GuzzleHttp\Client;
use Psr\Http\Message\ResponseInterface;
use PyaeSoneAung\QuizApiClient\Resources\QuestionResource;

it('can build client', function () {
    expect(quizApi()->buildClient())->toBeInstanceOf(Client::class);
});

it('can send get request', function () {
    expect(quizApi()->get(mockClient(), '/foo'))-
>toBeInstanceOf(ResponseInterface::class);
});

it('can return QuestionResource', function () {
    expect(quizApi()->questions())-
>toBeInstanceOf(QuestionResource::class);
});
```

`can build client` က `buildClient` function က guzzle client ကို create လုပ်နိုင်လား test တာပါ။ `toBeInstanceOf()` က `expect` ရဲ့ return ပြန်တဲ့ class က toBeInstanceOf() မှာထည့်ထားတဲ့ calss ဖြစ်ရမယ်လိုဆိုလိုတာပါ။

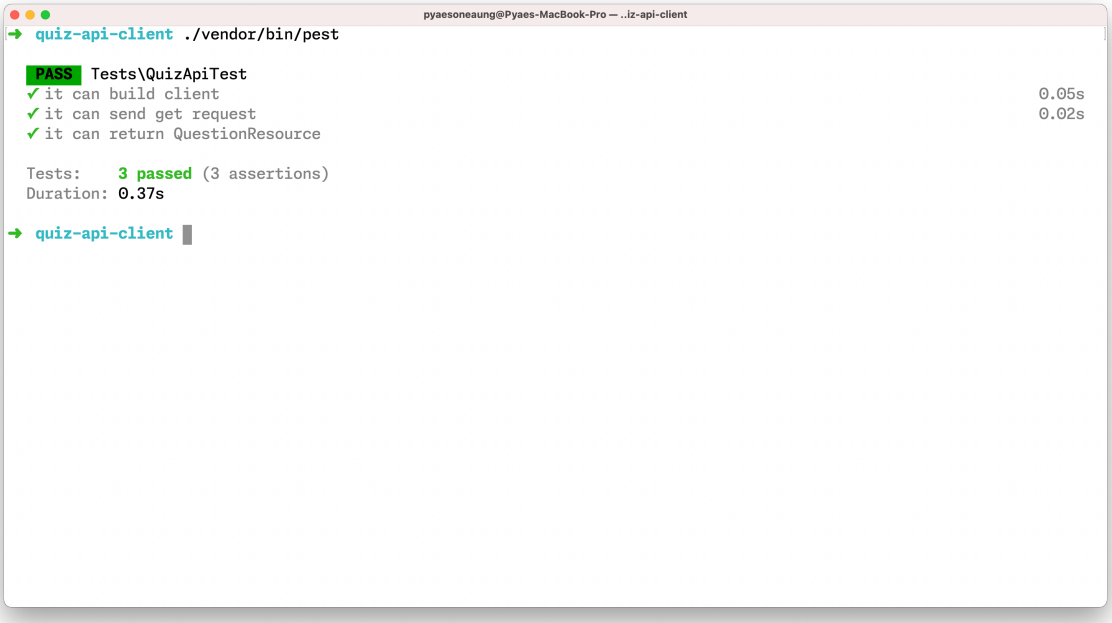
`can send get request` က GET method နဲ့ api ခေါ်လို့ရလား test တာပါ။ `get()` အတွက် လိုအပ်တဲ့ api client နေရာမှာ `mockClient()` ကိုသုံးထားပါတယ်။

`can return QuestionResource` ဒီဟာကတော့ရှင်းပါတယ်။ `quizApi()->questions()` လို့ခေါ်ရင် QuestionResource ရမရစစ်တာပါ။

`QuizApiTest.php` ထဲက test တွေက `BuildBaseClient.php`၊ `CanSendGetRequest.php` နဲ့ `QuizApi.php` ထဲက function တွေအကုန် cover ဖြစ်ပါပြီ။

```
./vendor/bin/pest
```

ဆိုပြီး test ကို run ကြည့်လို့ရပါပြီ။



ပြီးရင်တော့ ကျန်နေသေးတဲ့ `QuestionResource` class အတွက် test ရေးဖို့ package တစ်ခု သွင်းဖို့လိုပါတယ်။

```
composer require mockery/mockery --dev
```

ပြီးရင်တော့ `tests/Resources/QuestionResourceTest.php` ဆိုပြီး class တစ်ခု create လုပ်ပါမယ်။

`QuestionResourceTest.php`

```php
<?php

use PyaeSoneAung\QuizApiClient\QuizApi;
use PyaeSoneAung\QuizApiClient\Resources\QuestionResource;

it('can get questions', function () {
    $client = Mockery::mock(QuizApi::class);
    $client->shouldReceive('buildClient')->andReturnUsing(
        fn () => mockClient()
    );
    $client->shouldReceive('get')->andReturnUsing(
        fn () => mockClient()->get('/foo')
    );

    expect((new QuestionResource($client))->get())->toBeArray();
});
```

`can get questions` မှာ mockery package က `Mockery::mock()` ကို သုံးပြီး client တစ်ခု create လုပ်ပါတယ်။ ဒီနေရာမှာ `QuizApi::class` က `"PyaeSoneAung\QuizApiClient\QuizApi"` string ဖြစ်ပါတယ်။ ဒီနေရာမှာ ကိုယ်ကြိုက်တဲ့ string passs လို့ရပါတယ်။ mock name ကို unique ဖြစ်အောင်လို့နဲ့ `QuizApi` class ကို mock လုပ်မှန်းသိသာအောင် `Mockery::mock(QuizApi::class)` ဆိုပြီးသုံးထားတာပါ။

```php
$client->shouldReceive('buildClient')->andReturnUsing(
    fn () => mockClient()
);
```

shouldReceive က mock လုပ်ထားတဲ့ `$client` မှာ `buildClient` function ရှိတယ်လို့ သတ်မှတ်တာပါ။ `andReturnUsing()` အဲ့ function ကိုခေါ်ရင် mock လုပ်ထားတဲ့ guzzle client return ပြန်မယ်လို့ သတ်မှတ်တာပါ။

```php
$client->shouldReceive('get')->andReturnUsing(
    fn () => mockClient()->get('/foo')
);
```

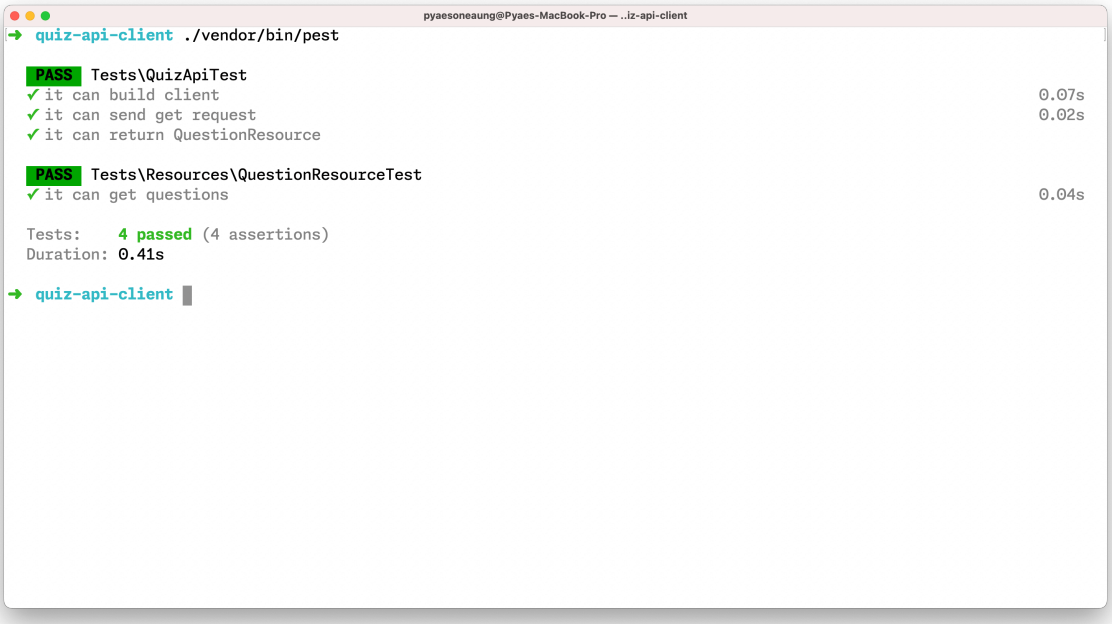
ဒီဟာလဲ အပေါ်ကသဘောပါပဲ။

```php
expect((new QuestionResource($client))->get())->toBeArray();
```

`(new QuestionResource($client))->get()` ကို ခေါ်ရင် return ပြန်တာက array ဖြစ်ရ မယ်လို့ ဆိုလိုတာပါ။

```
./vendor/bin/pest
```

ဆိုပြီး test ကို run ကြည့်လို့ရပါပြီ။



Continuous Integration

CI ဆိုတာကတော့ ကျတော်တို့ code တွေမှာ changes တွေဖြစ်တိုင်း build လုပ်တာတွေ၊ tests run တာတွေကို auto လုပ်ပေးတာပါ။ ဒီနေရာမှာ CI platform တစ်ခုဖြစ်တဲ့ GitHub Actions ကို သုံးပြီး tests တွေကို auto run အောင် setup လုပ်ပါမယ်။

အဲ့အတွက် `.github/workflows/run-tests.yml` ဆိုပြီး file တစ်ခု create လုပ်ပါမယ်။

`run-tests.yml`

```yaml
name: Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ${{ matrix.os }}
    strategy:
      fail-fast: true
      matrix:
        os: [ubuntu-latest]
        php: [8.1, 8.2, 8.3]

    name: P${{ matrix.php }} - ${{ matrix.os }}

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Setup PHP
        uses: shivammathur/setup-php@v2
        with:
          php-version: ${{ matrix.php }}
          coverage: none

      - name: Install dependencies
        run: composer install --prefer-dist --no-interaction --
optimize-autoloader

      - name: Execute tests
        run: ./vendor/bin/pest
```

အခဲ့ထဲက အရေးကြီးတာ တစ်ချို့ကို ရှင်းပြချင်ပါတယ်။

```yaml
runs-on: ${{ matrix.os }}
  strategy:
    fail-fast: true
    matrix:
      os: [ubuntu-latest]
      php: [8.1, 8.2, 8.3]

  name: P${{ matrix.php }} - ${{ matrix.os }}
```

သူက ubuntu latest version မှာ php-8.1 က နေ 8.3 ထိ tests ၃မျိုး run မယ်လို့ သတ်မှတ်တာပါ။ တကယ်လို့ windows latest version မှာ run ချင်ရင် `os: [ubuntu-latest, windows-`

`latest]` ဆိုပြီး ပြင်ရေးလို့ရပါတယ်။ အခဲ့တာဆိုရင် `php-8.1 - ubuntu` `php-8.2 - ubuntu` `php-8.3 - ubuntu` `php-8.1 - windows` `php-8.2 - windows` `php-8.3 - windows` အခဲ့လို tests ၆မျိုး run သွားမှာပါ။ ကျတော်ကတော့ windows ကို ထည့်မရေးပါဘူး test လုပ်ရင် ကြာလို့ပါ။

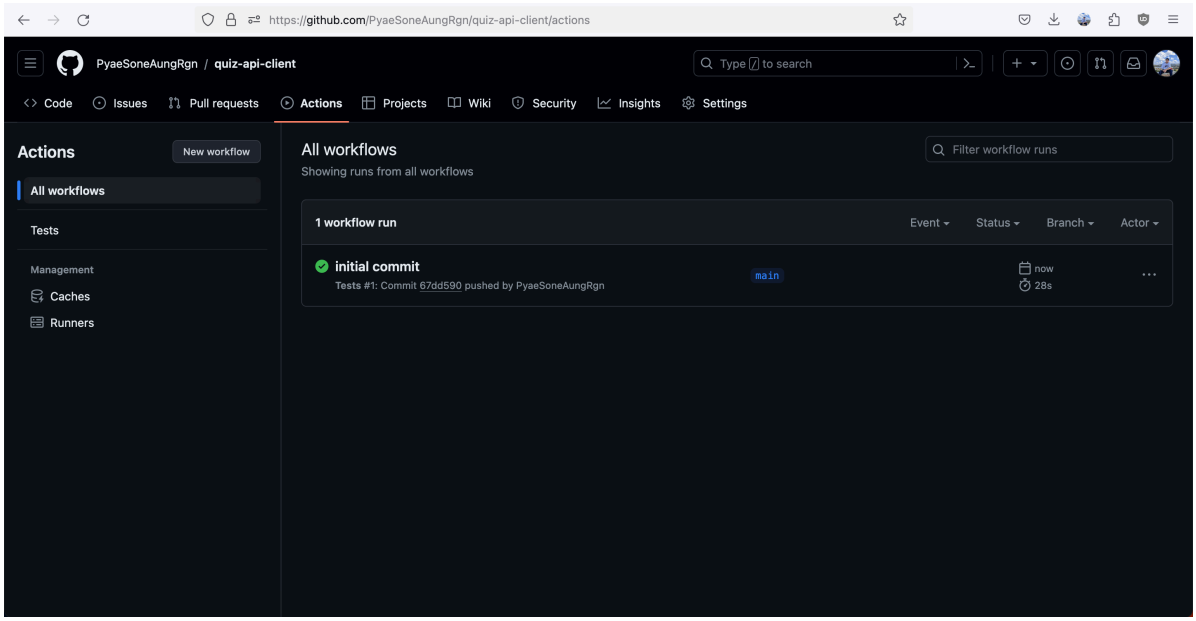
ဒါဆိုရင် `git init` လုပ်ပြီး github ပေါ်တင်လို့ရပါပြီ။ `git init` မလုပ်ခင် `.gitignore` ဆိုပြီး file တစ်ခု create လုပ်ပြီး အခုလိုရေးပါမယ်။

```
vendor
composer.lock
playground.php
```

vendor folder ရယ်၊ composer.lock ရယ်၊ playground.php file ကို git ထဲမထည့်ဘူးသတ်မှတ်တာပါ။ ဒီနေရာမှာ `playground.php` မှာ တကယ့် api key အစစ်ကို ထည့်ပြီး စမ်းထားတာရှိပါတယ်။ အခဲ့လို key တွေ ထည့်ပြီးစမ်းတဲ့ file မျိုးကို git ထဲ မထည့်မိဖို့က အရမ်းအရေးကြီးပါတယ်။

```
git init
git add .
git commit -m "initial commit"
```

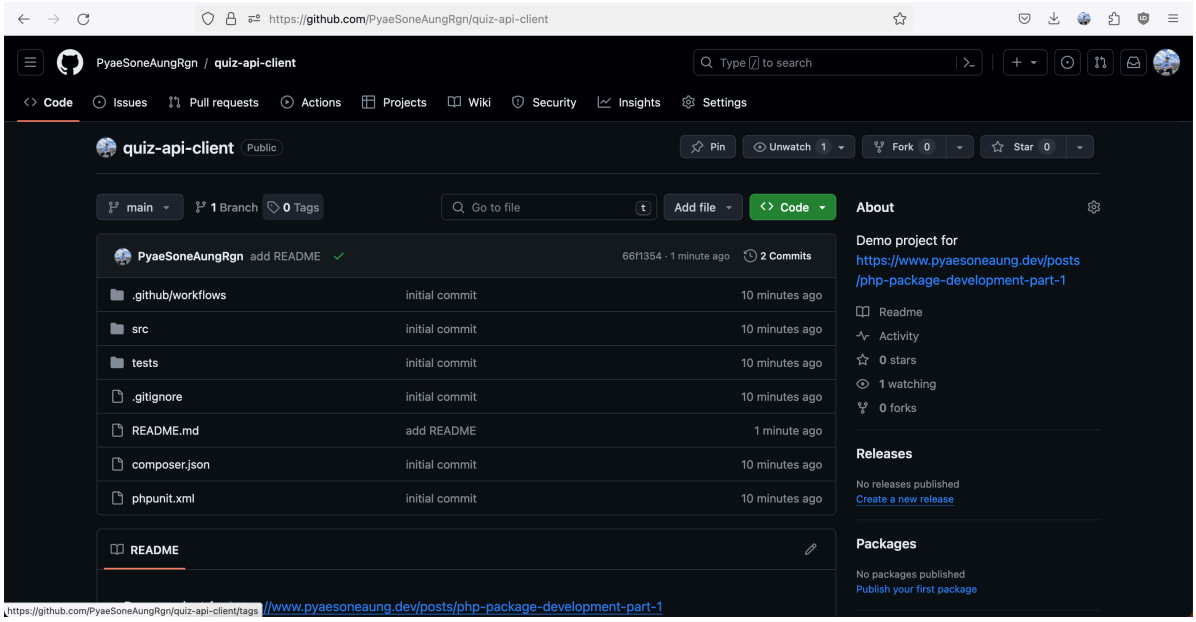
ပြီးရင်တော့ GitHub မှာ repo တစ်ခု create လုပ်ပြီး push ပါမယ်။



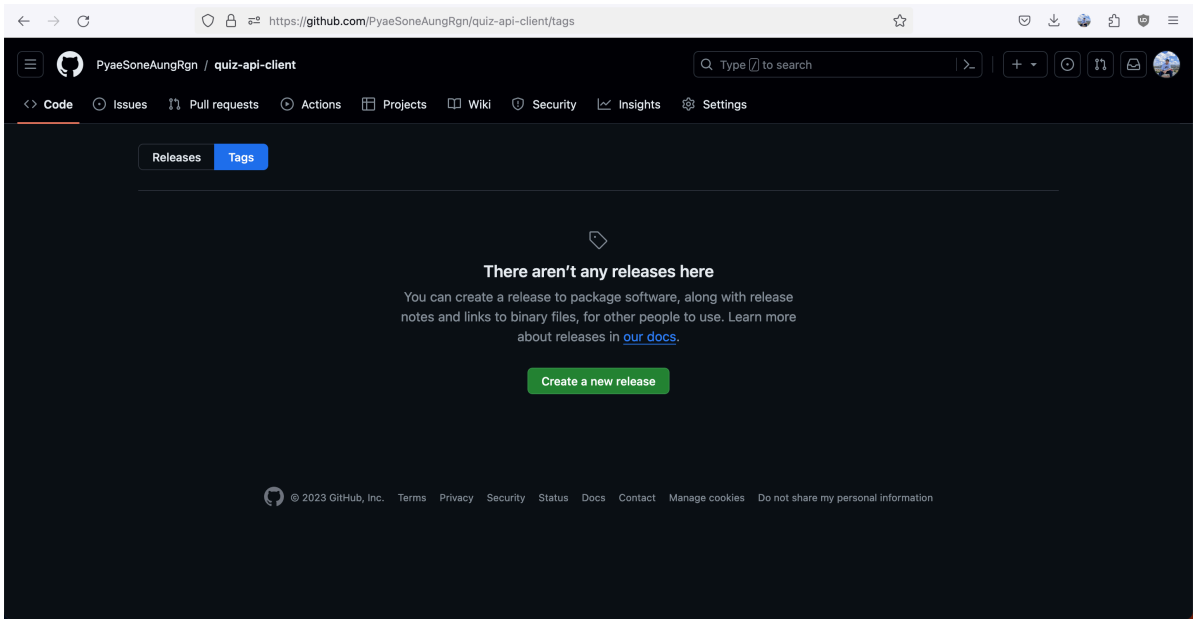
ဒါဆိုရင် GitHub Repo ထဲက Actions tab မှာ အခုလို test တွေ success ဖြစ်နေတာကိုတွေ့မှာပါ။

Release

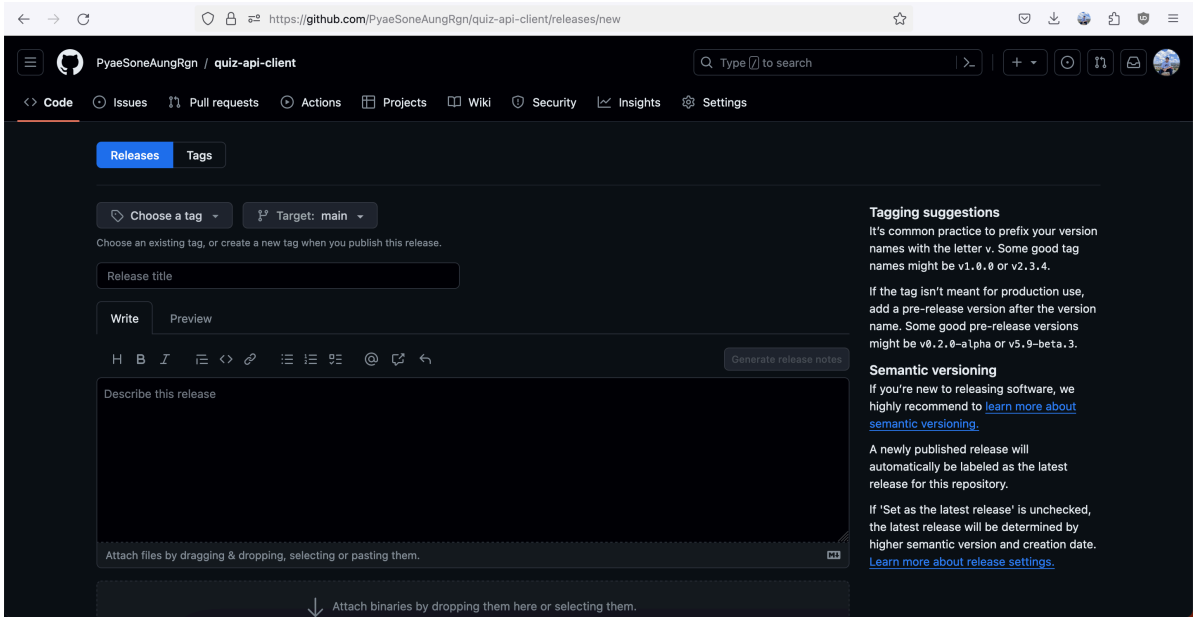
Release လုပ်ဖို့အတွက် GitHub ကိုသွားပြီး အခုလိုလုပ်ပါမယ်။



`0 tags` ကို နှိပ်ပါမယ်။

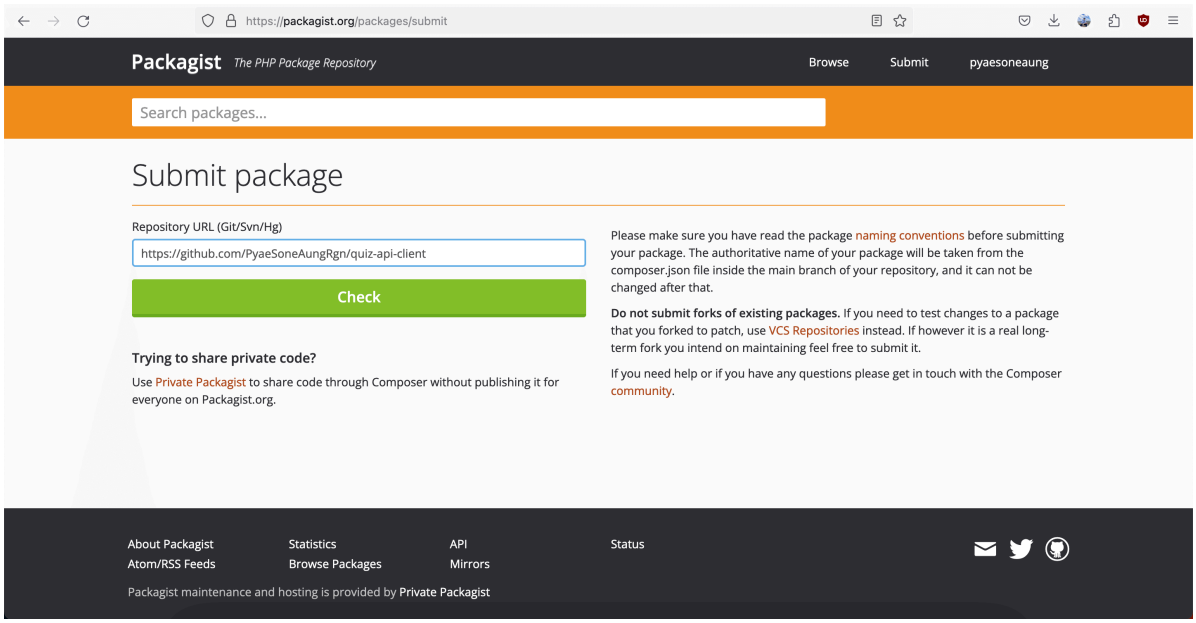


`Create a new release`



Choose a tag မှာ `v1.0.0` ဆိုပြီး tag တစ်ခု create လုပ်ပါမယ်။ Title နဲ့ description မှာ အဆင်ပြေတာ ရေးလို့ရပါတယ်။

ပြီးရင်တော့ packagist.org မှာ submit သွားလုပ်ပါမယ်။



ဒါပြီးရင်တော့

```
composer require pyaesoneaung/quiz-api-client
```

ဆိုပြီးတော့ ကျတော်တို့ package ကို composer ကနေ install လုပ်လို့ရပါပြီ။

နောက်အပိုင်းမှာတော့ Laravel အတွက် package development ကို ဆက်ပြီး knowledge sharing လုပ်သွားပါမယ်။ Laravel ဆိုရင်တော့ အခုလို object create လုပ်ပြီး constructor မှာ api key ထည့်တာတွေကို framework level မှာ dependency injection လုပ်ပြီး facades ကနေပဲ အလွယ်တကူ ခေါ်သုံးလို့ရအောင် ဖန်တီးလို့ရပါတယ်။ Source code ကို ဒီမှာ ကြည့်လို့ရပါတယ်။

Laravel Package Development

ဒီအပိုင်းမှာတော့ Laravel အတွက် package တစ်ခုဖန်တီးပါမယ်။

Requirements

- PHP Package Development

Setup Composer Project

Project တစ်ခု create လုပ်ဖို့အတွက်

```
mkdir quiz-api
cd quiz-api
composer init
```

ဆိုပြီး run ပါမယ်။

အဲ့ဒါဆိုရင် အခုလို folder structure ရပါပြီ။

```
├── src
├── vendor
└── composer.json
```

`composer.json` မှာ အခုလိုပြင်ပါမယ်။

```
{
    "name": "pyaesoneaung/quiz-api",
    "description": "Quiz Api Client",
    "type": "library",
    "license": "MIT",
    "autoload": {
        "psr-4": {
-           "Pyaesoneaung\\QuizApi\\": "src/"
+           "PyaeSoneAung\\QuizApi\\": "src/"
        }
    },
    "authors": [
        {
            "name": "Pyae Sone Aung",
            "email": "pyaesoneaungrgn@gmail.com"
        }
    ],
    "minimum-stability": "dev",
+   "prefer-stable": true,
    "require": {}
}
```

ဘာမှမရေးခင် အရင်ဆုံး ကိုယ့် package က Laravel version ဘယ်လောက်အနိမ့်ဆုံးလိုမယ်ဆိုတာ သတ်မှတ်ဖို့လိုပါတယ်။

`composer.json` မှာ အခုလိုပြင်ပါမယ်။

```
"require": {
  "illuminate/contracts": "^10.0|^11.0",
  "pyaesoneaung/quiz-api-client": "^1.0"
},
"require-dev": {
  "orchestra/testbench": "^9.0|^10.0",
}
```

"illuminate/contracts": "^10.0|^11.0" က Laravel 10 နဲ့ 11 မှာ သွင်းလို့ရမယ်လို့ ဆိုလိုတာပါ။

"pyaesoneaung/quiz-api-client": "^1.0" က ကျတော်တို့ PHP Package Development မှာ ရေးထားတဲ့ package ကို install လုပ်မယ်ဆိုလိုတာပါ။

"orchestra/testbench": "^9.0|^10.0" က test ရေးဖို့ အတွက်ပါ။ ပြီးတော့ testbench မှာ development လုပ်နေတဲ့ အချိန်မှာလိုအပ်တဲ့ ServiceProvider တို့၊ Facade တို့ အပြင်တခြား Laravel ရဲ့ core class တွေကို တခါထဲသွင်းသွားတဲ့ အတွက် text editor တွေရဲ့ auto complete

feature ကိုလည်း အသုံးပြုနိုင်မှာပါ။ ^9.0 က Laravel 10 အတွက်ဖြစ်ပြီး ^10.0 က Laravel 11 အတွက်ပါ။ အသေးစိတ်ကိုတော့ packages.tools မှာ ကြည့်လို့ရပါတယ်။

Setup Config

QuizApi ကို ခေါ်သုံးဖို့ အတွက် api key ရှိဖို့လိုပါတယ်။ Api key ကို config ကနေ တစ်ဆင့် `.env` file ကနေ `QUIZ_API_KEY` ဆိုတဲ့ key နဲ့ ယူပါမယ်။ အဲ့အတွက် project root folder မှာပဲ `config` ဆိုပြီး folder တစ်ခု create လုပ်ပါမယ်။ ပြီးရင် အဲ့ folder အောက်မှာပဲ `quiz-api.php` ဆိုပြီး php file တစ်ခု create လုပ်ပါမယ်။

```
├── config
│   ├── quiz-api.php
├── src
├── vendor
└── composer.json
```

`quiz-api.php` မှာ အခုလိုရေးပါမယ်။

```php
<?php

return [
    'api_key' => env('QUIZ_API_KEY'),
];
```

ဒီ config file ကို Laravel framework က သိဖို့အတွက် ServiceProvider တစ်ခု create လုပ်ပြီး register လုပ်ပေးဖို့လိုပါတယ်။

`src` folder အောက်မှာ `QuizApiServiceProvider.php` ဆိုပြီး file တစ်ခု create လုပ်ပါမယ်။

```
├── config
│   ├── quiz-api.php
├── src
│   ├── QuizApiServiceProvider.php
├── vendor
└── composer.json
```

`QuizApiServiceProvider.php` မှာ အခုလိုရေးပါမယ်။

```php
<?php

namespace PyaeSoneAung\QuizApi;

use Illuminate\Support\ServiceProvider;

class QuizApiServiceProvider extends ServiceProvider
{
    public function boot(): void
    {
        $this->mergeConfigFrom(
            __DIR__ . '/../config/quiz-api.php',
            'quiz-api'
        );
    }
}
```

ပြီးရင် QuizApiServiceProvider ကို `composer.json` မှာ register လုပ်ပါမယ်။

```json
"extra": {
  "laravel": {
    "providers": ["PyaeSoneAung\\QuizApi\\QuizApiServiceProvider"]
  }
}
```

ဒါဆိုရင် `config('quiz-api.api_key')` ဆိုပြီး ခေါ်သုံးလို့ရပါပြီ။

Service Container

Laravel Service Container ကို အကြမ်းဖျင်း နည်းနည်း ရှင်းပြချင်ပါတယ်။ Service Container မှာ အဓိက ၂ ပိုင်း ရှိပါတယ်။ Class တွေရဲ့ dependency တွေကို manage လုပ်တာနဲ့ dependency injection လုပ်ပေးတာပါ။

Binding

Dependency ကို manage လုပ်တယ်ဆိုတာက ဥပမာ ApiClient class တစ်ခုရဲ့ constructor မှာ $endpoint နဲ့ $apiKey လိုတယ်ဆိုရင် config ထဲက ဘယ် key တွေကို pass လိုက်ပါဆိုပြီး

သတ်မှတ်တာကို ဆိုလိုတာပါ။

```
$this->app->bind('api-client', function () {
    return new ApiClient('https://example.com', 'example-key');
});
```

ဒါဆိုရင် **'api-client'** ကို resolve လုပ်ရင် ApiClient ရဲ့ constructor မှာ `https://example.com` နဲ့ `example-key` ကို laravel က pass ပေးသွားမှာပါ။ တကယ့် real world project မှာတော့ အပေါ်က code လို ရိုးရိုး မ bind ဘဲ singleton method ကို သုံးပြီး bind ပါတယ်။

```
$this->app->singleton('api-client', function () {
    return new ApiClient('https://example.com', 'example-key');
});
```

ဘာကွာလဲဆိုရင် singleton method က တစ်ခါပဲ resolve လုပ်ပေးမှာ။ `app('api-client')` ဆိုပြီး object တစ်ခါဆောက်ပြီးရင် နောက်တစ်ခါ နောက်တစ်နေရာမှာ `app('api-client')` လို့ ခေါ်လည်း object အသစ်မဆောက်ဘဲ ပထမ ဆောက်ထားတဲ့ object ပဲ return ပြန်မှာပါ။

Resolving

`app()` ဆိုတာက dependency တွေကို reslove ဖို့သုံးတာပါ။ `app('api-client')` လို့ ခေါ်လိုက်ရင် service container မှာ bind ထားတဲ့အတိုင်း ApiClient ရဲ့ constructor မှာ $endpoint နဲ့ $apiKey pass ပြီး create လုပ်ထားတဲ့ object ကိုရမှာပါ။

QuizApi Integration

QuizApi ကို object ဆောက်ပြီး ခေါ်သုံးဖို့ အတွက် QuizApiServiceProvider မှာ အခုလိုရေးပါမယ်။

```php
<?php

namespace PyaeSoneAung\QuizApi;

use Illuminate\Support\ServiceProvider;
use PyaeSoneAung\QuizApiClient\QuizApi;

class QuizApiServiceProvider extends ServiceProvider
{
    public function boot(): void
    {
        $this->mergeConfigFrom(
            __DIR__ . '/../config/quiz-api.php',
            'quiz-api'
        );

        $this->app->singleton(QuizApi::class, function () {
            return new QuizApi(config('quiz-api.api_key'));
        });
    }
}
```

ဒါဆိုရင် `app(QuizApi::class)` လို့ခေါ်ရင် api key ထည့်ပြီးသား QuizApi object ကို ရမှာပါ။ ဒီနေရာမှာ `QuizApi::class` က **"PyaeSoneAung\QuizApiClient\QuizApi"** ဆိုတဲ့ string ကိုဆိုလိုတာပါ။ `app('PyaeSoneAung\QuizApiClient\QuizApi')` လို့ခေါ်လည်း QuizApi object ကိုရမှာပါ။ ဒီနေရာမှာ `QuizApi::class` လို့ သုံးတာက dependency resolve လုပ်တဲ့ အချိန် တခြား package တွေနဲ့ နာမည်တူမျိုး မဖြစ်အောင်လို့ပါ။

ဒါပေမဲ့ Laravel ကနေ

```php
use PyaeSoneAung\QuizApiClient\QuizApi;

app(QuizApi::class)->questions()->get();
```

ဆိုပြီး ခေါ်သုံးမှာ မဟုတ်ပါဘူး။ အခုလို ရှုပ်ထွေးတဲ့ object ဆောက်တာတွေကို Laravel Facade အောက်မှာ hide ပြီး Laravel ကနေ အခုလိုခေါ် သုံးမှာပါ။

```php
use QuizApi;

QuizApi::questions()->get();
```

Facade ကို create လုပ်ဖို့ အတွက် `src/Facades` folder အောက်မှာ `QuizApi.php` ဆိုပြီး file တစ်ခု create လုပ်ပါမယ်။

`QuizApi.php`

```php
<?php

namespace PyaeSoneAung\QuizApi\Facades;

use Illuminate\Support\Facades\Facade;

class QuizApi extends Facade
{
    protected static function getFacadeAccessor(): string
    {
        return \PyaeSoneAung\QuizApiClient\QuizApi::class;
    }
}
```

`getFacadeAccessor()` function က `app()` ထဲမှာ solve လုပ်မယ့် နာမည်ကို return ပြန်ပေးတာပါ။ ဒီနေရာမှာ **"PyaeSoneAung\QuizApiClient\QuizApi"** string ကို return ပြန်ပေးမှာပါ။

ပြီးရင်တော့ `composer.json` မှာ QuizApi Facade ကို register လုပ်ပါမယ်။

```json
"extra": {
  "laravel": {
    "providers": ["PyaeSoneAung\\QuizApi\\QuizApiServiceProvider"],
    "aliases": {
      "QuizApi": "PyaeSoneAung\\QuizApi\\Facades\\QuizApi"
    }
  }
}
```

ဒါဆိုရင်တော့ `QuizApi::questions()->get()` ဆိုပြီး Laravel app ကနေ ခေါ်သုံးလို့ရပါပြီ။

Testing

ကျတော်တို့ package အလုပ်လုပ်မလုပ် စမ်းဖို့အတွက် tests တွေရှိဖို့လိုပါတယ်။ Test ရေးဖို့ အတွက် `Pest PHP` ကို install လုပ်ပါမယ်။

```
composer require pestphp/pest --dev --with-all-dependencies
```

ပြီးရင်တော့

```
./vendor/bin/pest --init
```

ဆိုပြီး run ပါမယ်။ ဒါဆိုရင် tests folder နဲ့ phpunit.xml file ကို generate လုပ်သွားပေးမှာပါ။ ဒါဆိုရင်

```
./vendor/bin/pest
```

ဆိုပြီး test တွေ run လို့ရပါပြီ။

`tests` folder ထဲမှာ Feature နဲ့ Unit ဆိုပြီး folder ၂ခု ရှိပါတယ်။ Package တွေမှာတော့ Feature သပ်သပ်၊ Unit သပ်သပ်ဆိုပြီး test တွေရေးတာထက် အပြင်မှာ class တစ်ခု create လုပ်ပြီးရေးတာမျိုးက ပိုများပါတယ်။ Test ရေးဖို့ အတွက် Feature နဲ့ Unit folder နှစ်ခုလုံးကို ဖျက်လိုက်ပါမယ်။ ပြီးရင် `TestCase.php` ရေးဖို့ အတွက် `composer.json` မှာ အခုလိုပြင်ပါမယ်။

```
"autoload-dev": {
  "psr-4": {
    "PyaeSoneAung\\QuizApi\\Tests\\": "tests"
  }
}
```

`tests` folder ထဲက class တွေက `PyaeSoneAung\QuizApi\Tests` ဆိုတဲ့ namespace အောက်မှာ ရှိတယ်ဆိုပြီး သတ်မှတ်လိုက်လိုက်တာပါ။

`TestCase.php`

```php
<?php

namespace PyaeSoneAung\QuizApi\Tests;

use Illuminate\Support\Facades\Config;
use PyaeSoneAung\QuizApi\QuizApiServiceProvider;
use Orchestra\Testbench\TestCase as Orchestra;

class TestCase extends Orchestra
{
    protected function setUp(): void
    {
        parent::setUp();

        Config::set('quiz-api.api_key', 'foo');
    }

    protected function getPackageProviders($app)
    {
        return [
            QuizApiServiceProvider::class,
        ];
    }
}
```

`setUp()` function မှာက testing လုပ်တဲ့ အချိန်မှာသုံးမယ့် api key ကို သတ်မှတ်လိုက်တာပါ။

`getPackageProviders()` function မှာက `QuizApiServiceProvider` ကို testing မှာ bootstrap လုပ်မယ်ဆိုပြီး သတ်မှတ်လိုက်တာပါ။

ပြီးရင် `Pest.php` မှာ အခုလိုရေးပါမယ်။

`Pest.php`

```php
<?php

use PyaeSoneAung\QuizApi\Tests\TestCase;

uses(TestCase::class)->in(__DIR__);
```

`tests` folder အောက်က test တွေကို run ရင် `TestCase` class ကို သုံးမယ်ဆိုပြီး သတ်မှတ်လိုက်တာပါ။ ပြီးရင်တော့ `QuizApiTest.php` ဆိုပြီး file တစ်ခု create လုပ်ပြီး အခုလိုရေးပါမယ်။

QuizApiTest.php

```php
<?php

use PyaeSoneAung\QuizApi\Facades\QuizApi;
use PyaeSoneAung\QuizApiClient\QuizApi as QuizApiClient;
use PyaeSoneAung\QuizApiClient\Resources\QuestionResource;

it('can get QuizApi client', function () {
    expect(app(QuizApiClient::class))-
>toBeInstanceOf(QuizApiClient::class);
});

it('can get question resource', function () {
    expect(QuizApi::questions())-
>toBeInstanceOf(QuestionResource::class);
});
```

`can get QuizApi client` က ကျတော်ဆို့ `app()` ဆိုပြီး ခေါ်သုံးတဲ့နေရာမှာ QuizApi client object ပြန်ရလားဆိုပြီး စစ်တာပါ။

`can get question resource` က `QuizApi::questions()` ဆိုပြီး facade ကို ခေါ်သုံးလို့ ရလားဆိုပြီး စစ်တာပါ။

ဒါဆိုရင်

```
./vendor/bin/pest
```

ဆိုပြီး test ကို run ကြည့်လို့ရပါပြီ။

Continuous Integration

CI အတွက် `.github/workflows/run-tests.yml` ဆိုပြီး file တစ်ခု create လုပ်ပါမယ်။

run-tests.yml

```yaml
name: run-tests

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ${{ matrix.os }}
    strategy:
      fail-fast: true
      matrix:
        os: [ubuntu-latest]
        php: [8.2]
        laravel: [10.*]
        stability: [prefer-stable]
        include:
          - laravel: 10.*
            testbench: 8.*
            carbon: ^2.63

    name: P${{ matrix.php }} - L${{ matrix.laravel }} - ${{
matrix.stability }} - ${{ matrix.os }}

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup PHP
        uses: shivammathur/setup-php@v2
        with:
          php-version: ${{ matrix.php }}
          extensions: dom, curl, libxml, mbstring, zip, pcntl, pdo,
sqlite, pdo_sqlite, bcmath, soap, intl, gd, exif, iconv, imagick,
fileinfo
          coverage: none

      - name: Setup problem matchers
        run: |
          echo "::add-matcher::${{ runner.tool_cache }}/php.json"
          echo "::add-matcher::${{ runner.tool_cache
}}/phpunit.json"

      - name: Install dependencies
        run: |
```

```
          composer require "laravel/framework:${{ matrix.laravel }}"
"orchestra/testbench:${{ matrix.testbench }}" "nesbot/carbon:${{
matrix.carbon }}" --no-interaction --no-update
          composer update --${{ matrix.stability }} --prefer-dist --
no-interaction

      - name: List Installed Dependencies
        run: composer show -D

      - name: Execute tests
        run: vendor/bin/pest --ci
```

ဒီနေရာမှာ include အပိုင်းကို အနည်းငယ်ရှင်းပြချင်ပါတယ်။

```
include:
  - laravel: 10.*
    testbench: 8.*
    carbon: ^2.63
```

Laravel 10 ကို test ရင် testbench ကို version 8.* ပဲ သုံးရမယ်လို့ သတ်မှတ်တာပါ။ Laravel 9 ဆိုရင် testbench: 7.*၊ Laravel 11 ဆိုရင် 9.* ကို သုံးမယ်ဆိုပြီး သတ်မှတ်ဖို့လိုပါတယ်။ အခု စာရေးနေတဲ့ အချိန်မှာတော့ Laravel 11 မထွက်သေးတဲ့ အတွက် Laravel 10 အတွက်ပဲ CI ရေးထားတာပါ။

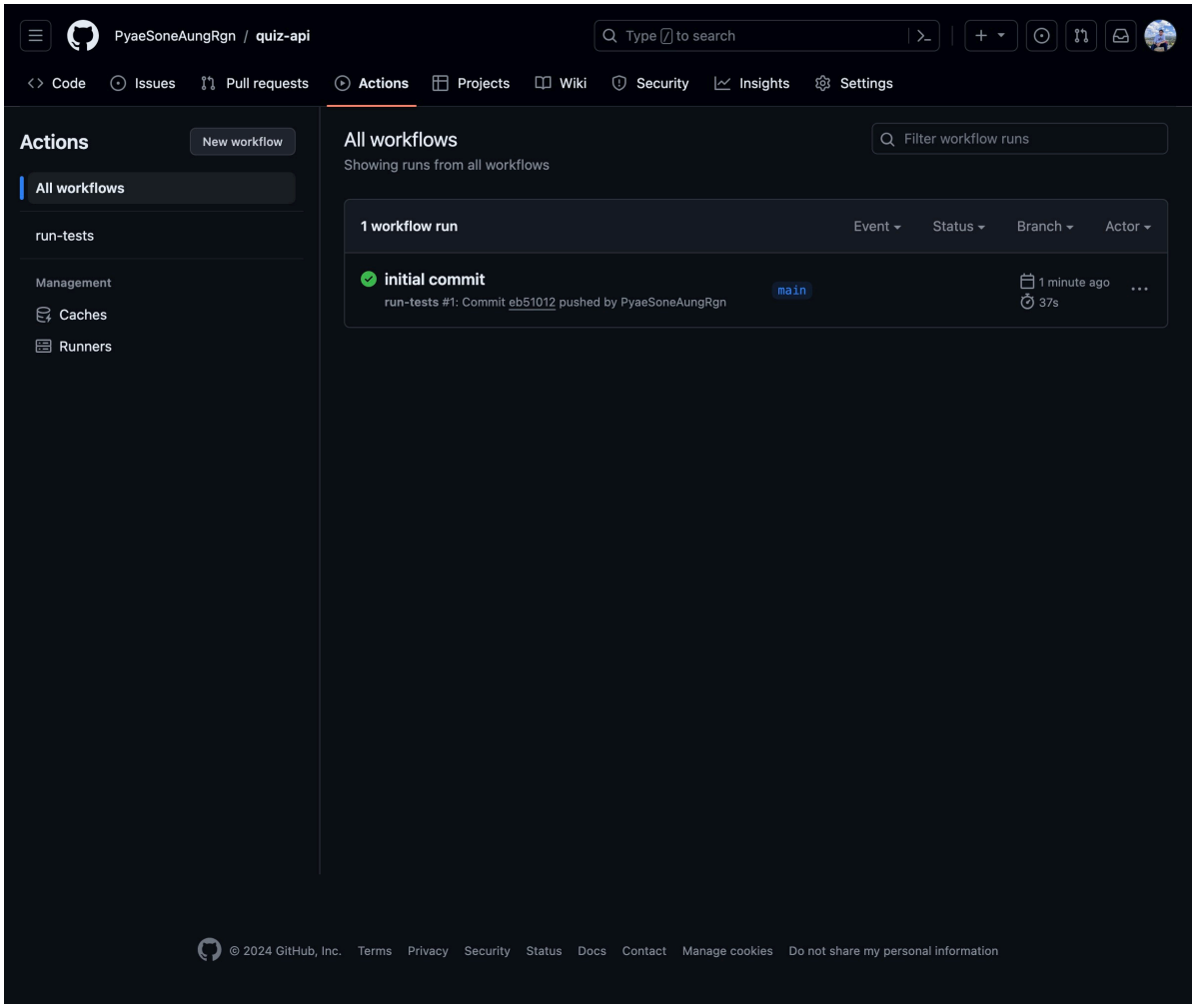
ဒါဆိုရင် `git init` လုပ်ပြီး GitHub ပေါ်တင်လို့ရပါပြီ။ `git init` မလုပ်ခင် `.gitignore` ဆိုပြီး file တစ်ခု create လုပ်ပြီး အခုလိုရေးပါမယ်။

```
vendor
composer.lock
```

vendor folder ရယ်၊ composer.lock ရယ် ကို git ထဲမထည့်ဘူး သတ်မှတ်တာပါ။

```
git init
git add .
git commit -m "initial commit"
```

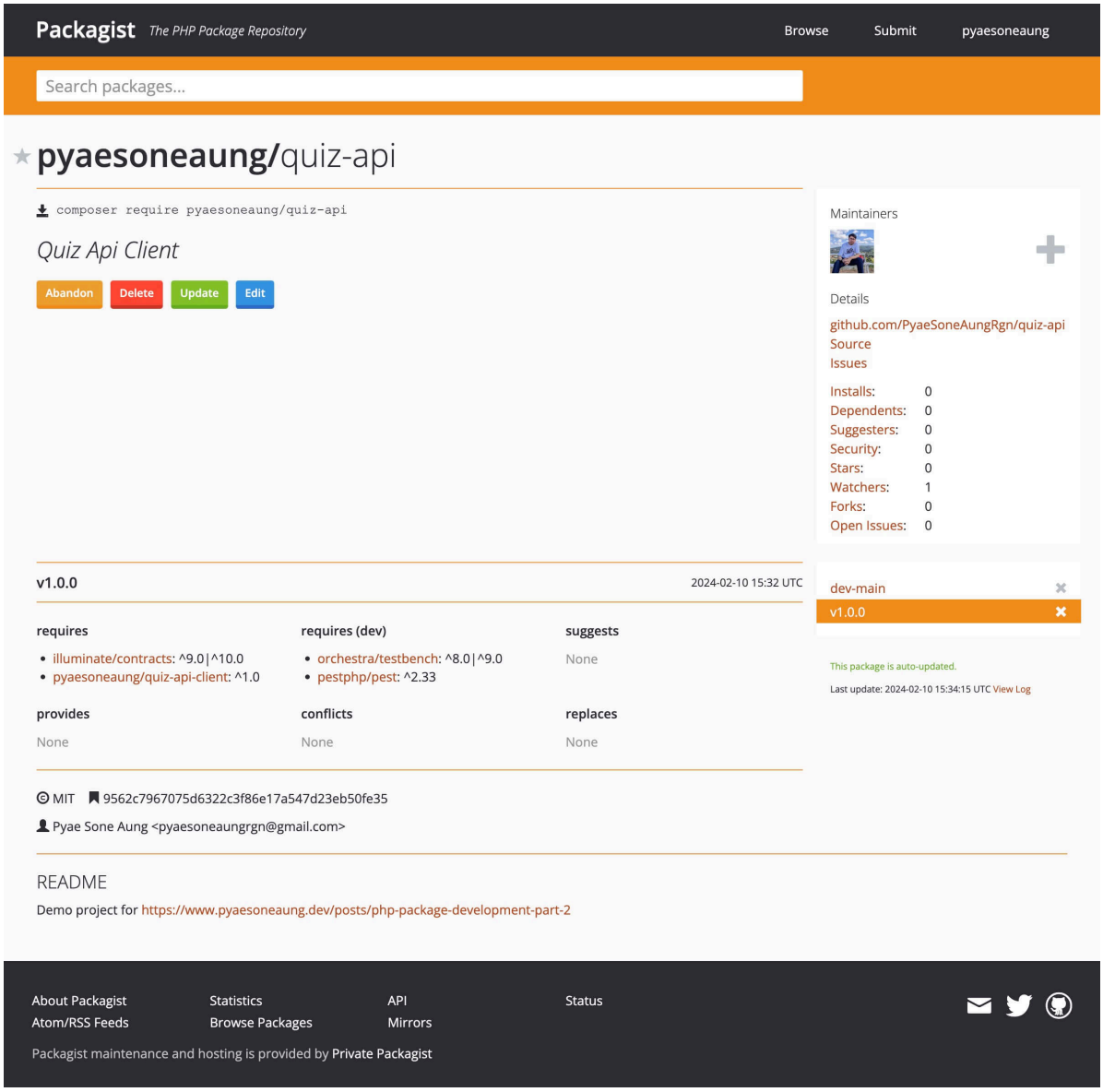
ပြီးရင်တော့ GitHub မှာ repo တစ်ခု create လုပ်ပြီး push ပါမယ်။



ဒါဆိုရင် GitHub Repo ထဲက Actions tab မှာ အခုလို test တွေ success ဖြစ်နေတာကိုတွေ့မှာပါ။

Release

GitHub မှာ `v1.0.0` ကို release လုပ်ပြီး packagist.org မှာ submit သွားလုပ်ပါမယ်။



ဒါပြီးရင်တော့

```
composer require pyaesoneaung/quiz-api
```

ဆိုပြီးတော့ ကျတော်တို့ package ကို composer ကနေ install လုပ်လို့ရပါပြီ။

နောက်အပိုင်းမှာတော့ package တစ်ခုကို အခုလို အစဆုံးရေးနေတာမျိုး မဟုတ်ဘဲ template တွေအသုံးပြုပြီး ဖန်တီးတာရယ်၊ spatie ရဲ့ package development tool package အကြောင်းရယ်၊ documentation အကြောင်းတွေကို ဆက်ပြီး knowledge sharing လုပ်သွားပါမယ်။ Source code ကို ဒီမှာ ကြည့်လို့ရပါတယ်။

Package Development Tools

ဒီအပိုင်းမှာတော့ package development အတွက် အသုံးဝင်တဲ့ tool တွေကို ပြောပြသွားပါမယ်။

Requirements

- PHP Package Development
- Laravel Package Development

Documentation

ကျတော်အတွက် documentation က package တစ်ခုအတွက် အရေးကြီးဆုံးအပိုင်းလို့ ခံယူထားပါတယ်။ ကိုယ်ရေးတဲ့ package က ရိုးရိုးရှင်းရှင်းပဲဆိုရင် `README.md` file မှာ installation နဲ့ usage လောက်ရေးရုံနဲ့ အဆင်ပြေပါတယ်။ တကယ်လို့ ကိုယ်ရေးတဲ့ package က feature တွေ အများကြီးပါတယ်ဆိုရင် static site generator တစ်ခုခုကိုသုံးပြီး ရေးသင့်ပါတယ်။

- VitePress
- Docsify
- VuePress
- JIGSAW

ဒါတွေကတော့ ကျတော်ကိုယ်တိုင်သုံးဖူးပြီး အဆင်ပြေတဲ့ static site generator တွေပါ။ Documentation site တစ်ခုဖန်တီးတဲ့ အကြောင်းကို Build a Modern Documentation Site မှာ ကျတော် အသေးစိတ်ရေးထားပါတယ်။

Package Skeleton

Package တစ်ခုရေးမယ်ဆိုရင် `composer init` လုပ်တာတွေ၊ `src` folder တွေ၊ `composer.json` တွေကို အစအဆုံး create လုပ်နေတာထက် package skeleton ကိုသုံးလိုက်

တာက ပိုပြီး အလုပ်တွင်ပါတယ်။ Package skeleton တွေမှာက development တွေလိုအပ်တဲ့ tests တွေ၊ folder structure တွေ၊ project setup အတွက်လိုတာတွေ ပါပြီးသားဖြစ်တဲ့အတွက် business logic ကိုတန်းရေးလို့ ရပါတယ်။

- package-skeleton-php
- package-skeleton-laravel

Laravel Package Tools

spatie/laravel-package-tools က Laravel package ရေးတဲ့အခါမှာ migration တို့၊ config တို့၊ view တို့ လိုအပ်ရင် ServiceProvider ကနေ အလွယ်တကူချိတ်လို့ရအောင် ဖန်တီးထားတဲ့ package တစ်ခုပါ။

Laravel News

ကျတော်တို့ ရေးထားတဲ့ package တွေကို Laravel News မှာ အကောင့်ဖွင့်ပြီး submit လုပ်လို့ရပါတယ်။ သူတို့ approve ပေးလိုက်ရင် scoial platoform အမျိုးမျိုးမှာ ကိုယ့် package ကို share ပေးသွားမှာပါ။

Made With Laravel မှာလည်း submit လုပ်လို့ရတယ်။

Learning Resources

Spatie ကရေးတဲ့ package တွေရဲ့ source code တွေကို လေ့လာထားရင်ကို package development အတွက် တော်တော် များများကို သိနေမှာပါ။

Package Development - ဒါကတော့ package development အတွက် Laravel ရဲ့ official documentation ပါ။

Strategies for making Laravel packages customizable - ဒါကတော့ package တွေကို customizable ဖြစ်အောင် ဘယ်လိုရေးရမလဲဆိုတာကို ရှင်းပြထားတဲ့ article တစ်ခုပါ။