**Configure Docker:**

Generate a token from:

<https://hub.docker.com/settings/security>

paste it in your GitHub projects Secrets actions with secret name: DOKERHUB\_TOKEN:

<https://github.com/yugants/recipe-app-api/settings/secrets/actions>

create another secret with: DOCKERHUB\_USER

and save docker username in it.

Create a requirements.txt and type versions of Django and djangorestframework

.dockerignore – ignored by docker

.gitignore – ignored by git

**DockerFile:**

The Dockerfile is used to build our image, which contains a mini Linux Operating System with all the dependencies needed to run our project.

In Dockerfile we performed:

* Copy operation from local to docker
* Created a virtual env. on docker
* Created a user for docker named “django-user”
* Set the python env. path

Create an app folder in main DIR and run:

docker build .

**Docker-compose:**

We are initializing docker compose for our project, after configuring run:

docker-compose build

**Linting:**

That’s right! Linting is used to ensure code is formatted correctly. It highlights issues like invalid tab spacing and line lengths.

We will use “flake8” for linting

Now make changes in docker-compose, Dockerfile and create .flake8

To check if flake8 is installed correctly or not type:

docker-compose run --rm app sh -c "flake8"

**Create Project:**

docker-compose run --rm app sh -c "django-admin startproject app ."

**Run Server in Docker:**

docker-compose up

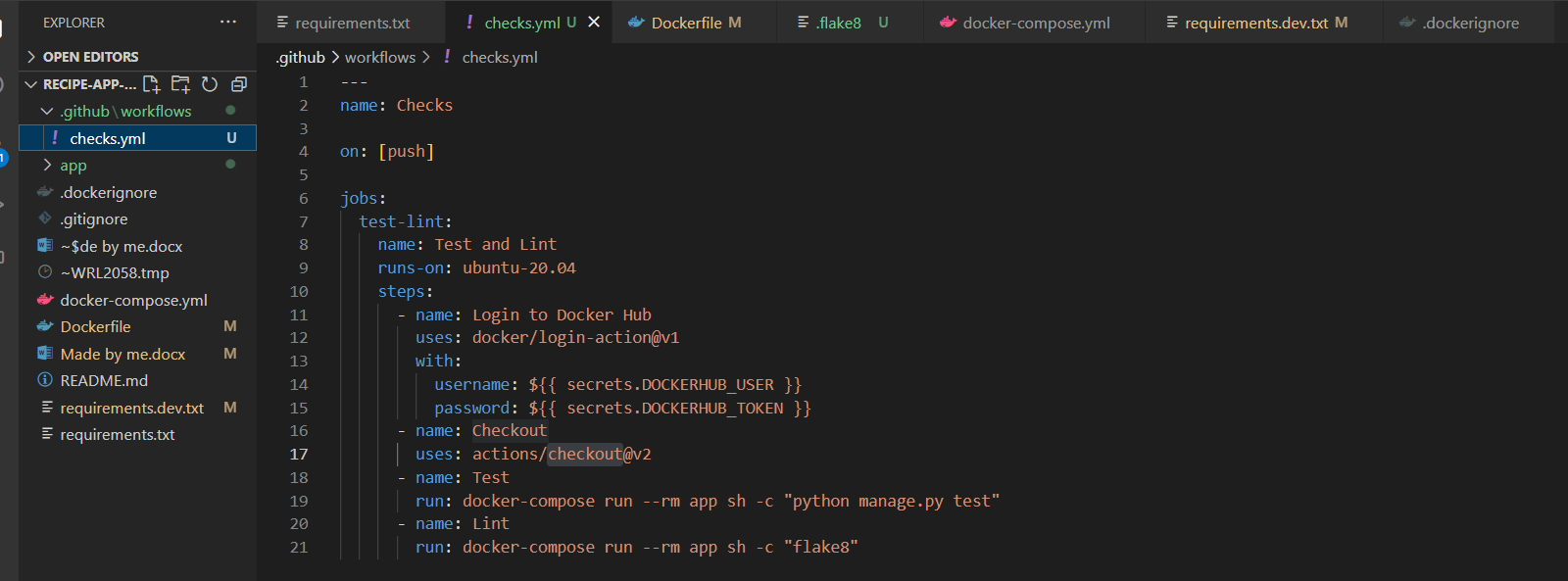
**GitHub Actions:**

* Automation tool
* Similar to Jenkins, GitLab CI/CD
* Run jobs when code changes
* Automate tasks

Our trigger for CI/CD is push operation on github.

**For Configuring CI/CD:**

Make a folder in main file: .github/workflows -> in it create a file -> checks.yml:



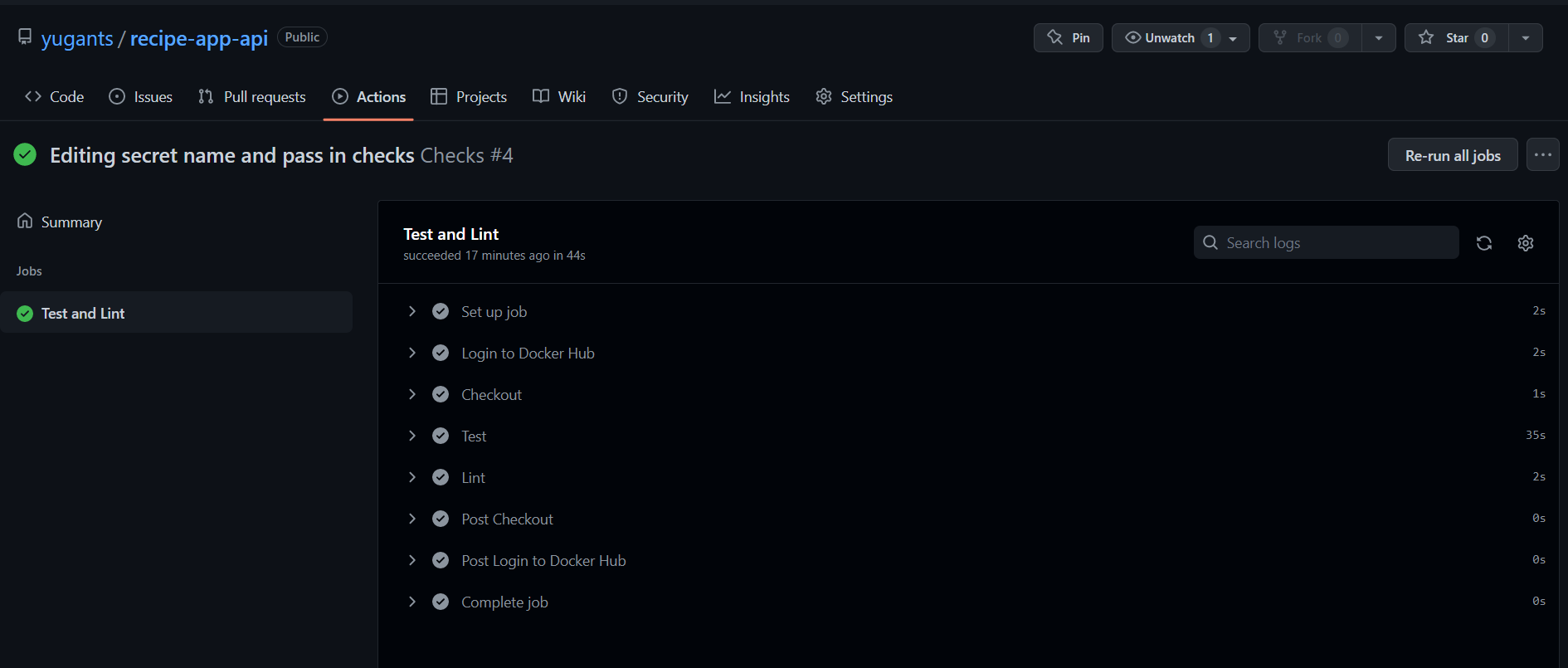
Then run the in shell to check:

docker-compose run --rm app sh -c "python manage.py test"

We are making a CI/CD pipeline in above SS:

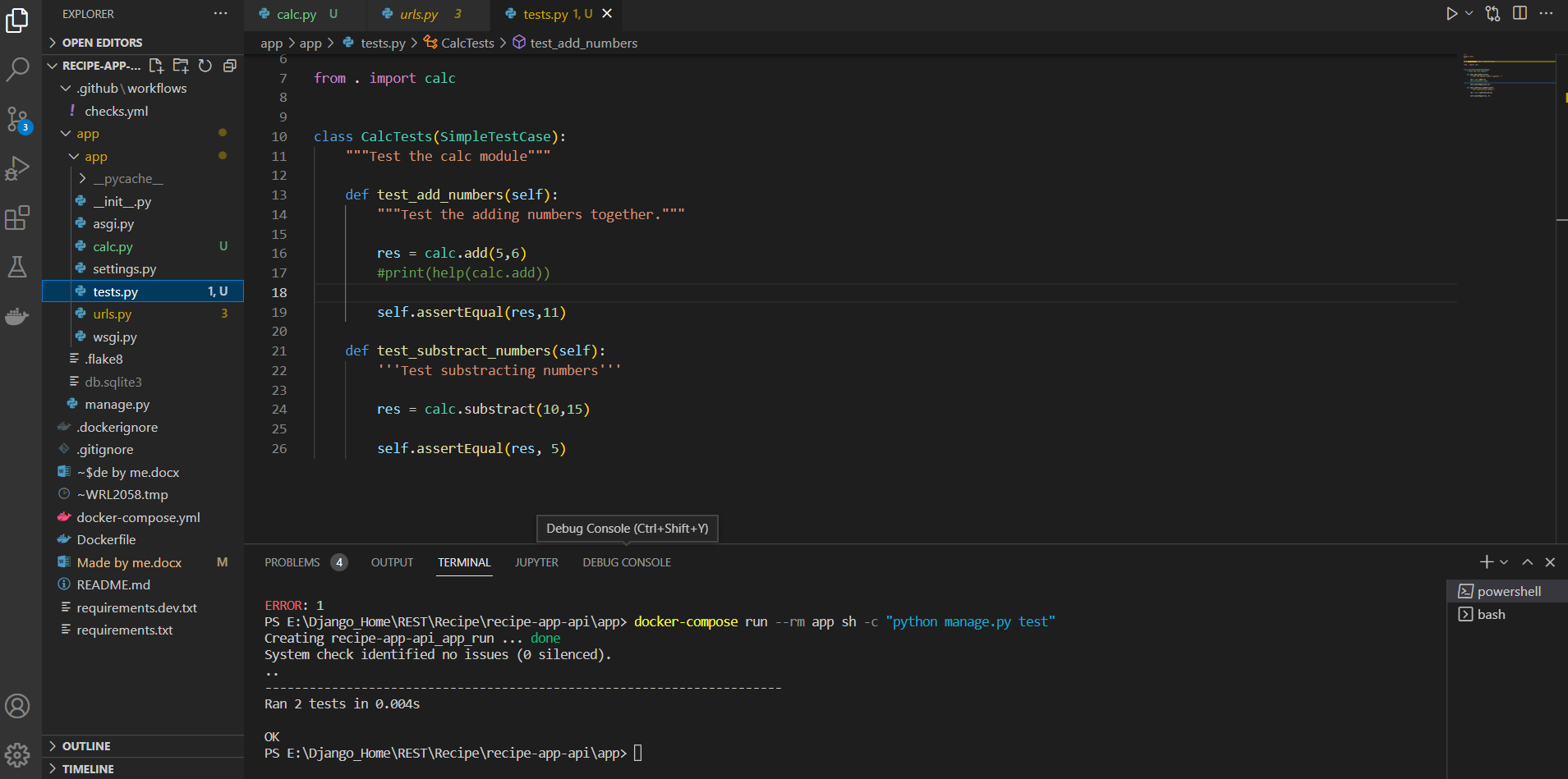
on: execution method

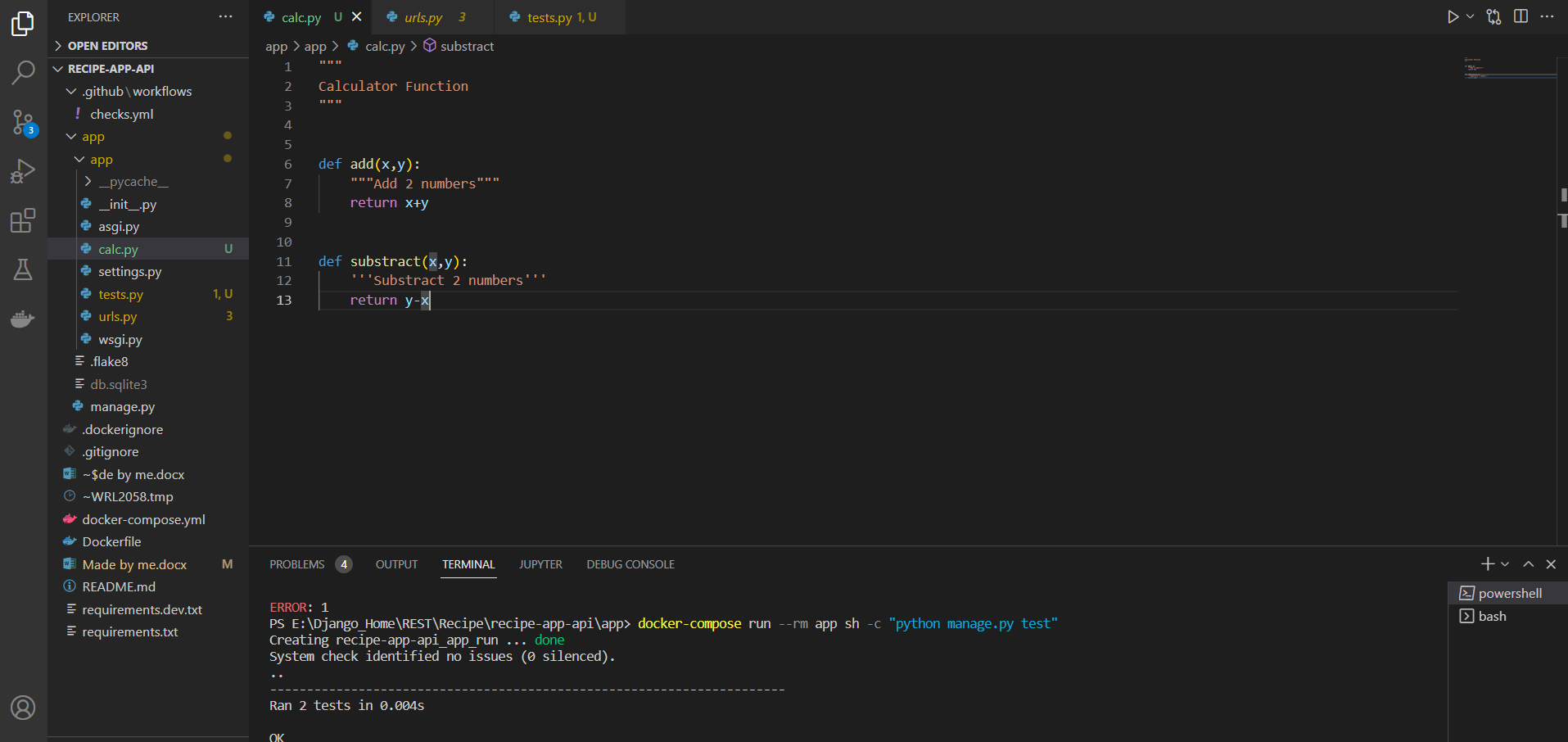
steps: we have-



**Tests in Django:**

Create a file named tests.py or a folder named test or create a dir named tests and in it define an \_\_init\_\_() file, start files in module with test prefix and also start function name with test\_, we cannot use both of them either tests/ or tests.py

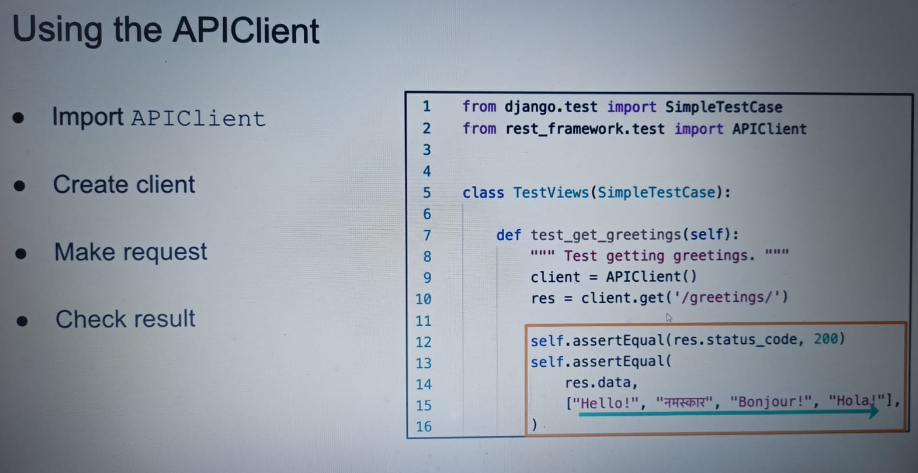




## What Is Mocking?

A [mock object](https://en.wikipedia.org/wiki/Mock_object) substitutes and imitates a real object within a [testing environment](https://realpython.com/python-testing/). It is a versatile and powerful tool for [improving the quality of your tests](https://realpython.com/python-cli-testing/#mocks). One reason to use Python mock objects is to control your code’s behaviour during testing. For example, if your code makes [HTTP requests](https://realpython.com/python-requests/) to external services, then your tests execute predictably only so far as the services are behaving as you expected. Sometimes, a temporary change in the behaviour of these external services can cause intermittent failures within your test suite. Because of this, it would be better for you to test your code in a controlled environment. [Replacing the actual request with a mock object](https://realpython.com/testing-third-party-apis-with-mocks/) would allow you to simulate external service outages and successful responses in a predictable way.

**API Client:**



Postgres Configuration in docker-compose:

