Continuous Integration using GitHub Actions

Continuous integration (CI) is a software practice that requires frequently committing code to a shared repository. Committing code more often detects errors sooner and reduces the amount of code a developer needs to debug when finding the source of an error. Frequent code updates also make it easier to merge changes from different members of a software development team. This is great for developers, who can spend more time writing code and less time debugging errors or resolving merge conflicts.

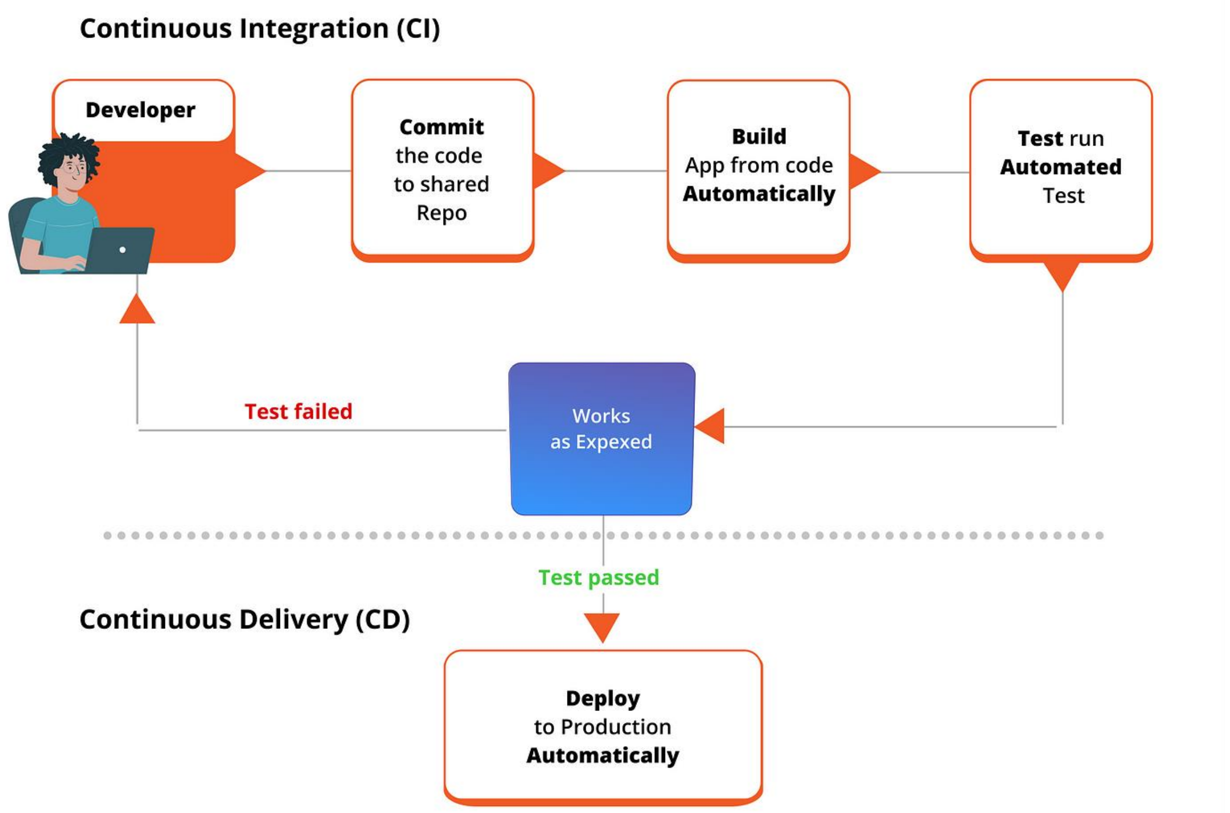
When we commit code to our repository, we can continuously build and test the code to make sure that the commit doesn't introduce errors. Our tests can include code linters (which check style formatting), security checks, code coverage, functional tests, and other custom checks.

Building and testing your code requires a server. You can build and test updates locally before pushing code to a repository, or you can use a CI server that checks for new code commits in a repository.

CI using **GitHub Actions** offers workflows that can build the code in our repository and run our tests. Workflows can run on GitHub-hosted virtual machines, or on machines that we host ourselves.

We can configure our CI workflow to run when a GitHub event occurs (for example, when new code is pushed to repository), on a set schedule, or when an external event occurs using the repository dispatch webhook.

GitHub runs CI tests and provides the results of each test in the pull request, so we can see whether the change in our branch introduces an error. When all CI tests in a workflow pass, the changes we pushed are ready to be reviewed by a team member or merged. When a test fails, one of our changes may have caused the failure.


Continuous Integration (CI)
Developer
Commit the code to shared Repo
Build App from code Automatically
Test run Automated Test
Works as Expexed
Test failed
Test passed
Continuous Delivery (CD)
Deploy to Production Automatically

Key advantages of using GitHub Actions for CI/CD pipelines

But first, let's talk through some of the benefits to using GitHub Actions—because let's be honest, there are a lot of other tools out there. Let me unpack the four big benefits that I've come across:

1. **CI/CD pipeline set-up is simple:** GitHub Actions is made by and for developers, so you don't need dedicated resources to set up and maintain your pipeline. There's no need to manually configure and set up CI/CD. You don't have to set up webhooks, you don't have to buy hardware, reserve some instances out there, keep them up to date, do security patches, or spool down idle machines. You just drop one file in your repo, and it works.

2. **Respond to any webhook on GitHub:** Since GitHub Actions is fully integrated with GitHub, you can set any webhook as an event trigger for an automation or CI/CD pipeline. This includes things like pull requests, issues, and comments, but it also includes webhooks

from any app you have integrated into your GitHub repository. Let's say you're going to use any one of the many tools that are out there to run part of your development pipeline. With GitHub Actions, you can trigger CI/CD workflows and pipelines of webhooks from these apps (even something simple, like a chat app message, if you've integrated your chat app into your GitHub repository, of course).

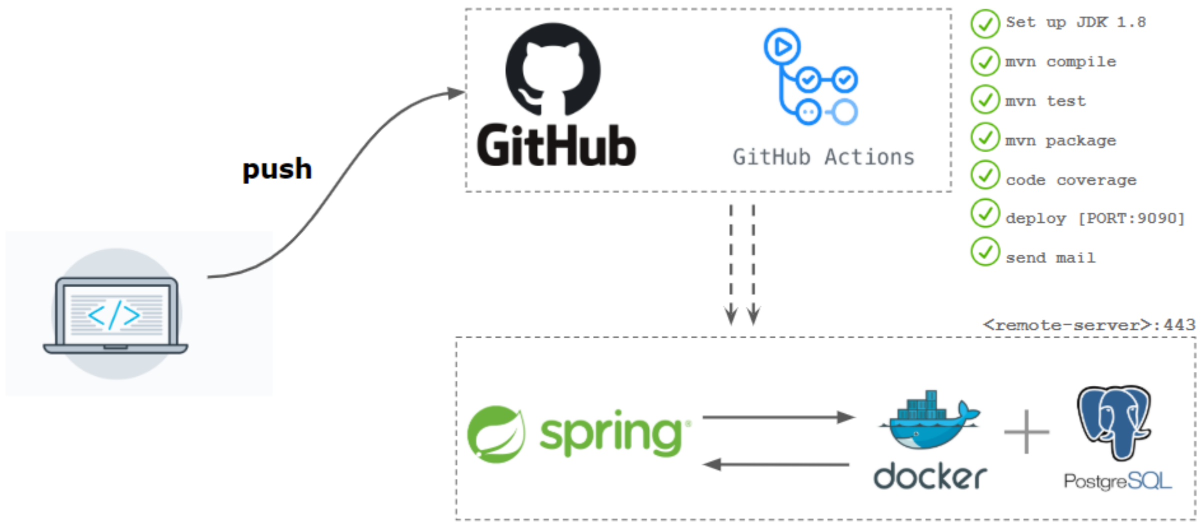
3. **Community-powered, reusable workflows:** You can share your workflows publicly with the wider GitHub community or access pre-built CI/CD workflows in the GitHub Marketplace (there are more than 11,000 available actions!). Did I mention every action is reusable just by referencing its name? Yes, that too.

4. **Support for any platform, any language, and any cloud:** GitHub Actions is platform agnostic, language agnostic, and cloud agnostic. That means you can use it with whatever technology you choose.

How to build a CI/CD pipeline with GitHub Actions

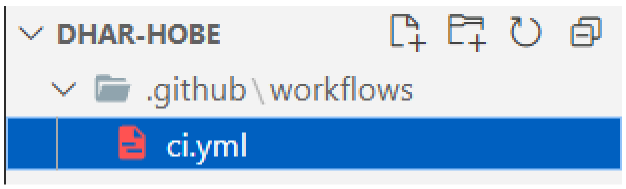
Before we dive in, here are a few quick notes:

Be clear about what a CI/CD pipeline is and should do. This is a simple note, but important. A CI pipeline runs when code changes and should make sure all of our changes work with the rest of the code when it's integrated. It should also compile our code, run tests, and check that it's functional. A CD pipeline goes one step further and deploys the built code into production.



Step 1: Set Up GitHub Actions Workflow

Create a ***.github/workflows*** directory in the root of your Django project. Inside the workflows directory, create a YAML file, for example, ci.yml. This file will define your GitHub Actions workflow.



Code to trigger CI pipeline

```yaml
name: Django CI

on:
  push:
    branches:
      - rent_sanzida

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Repository
        uses: actions/checkout@v2

      - name: Set Up Python
        uses: actions/setup-python@v2
        with:
          python-version: 3.x

      - name: Install Dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt

      - name: Run Tests
        run: |
          cd BharaHobe
          python manage.py test
```

Step 2: Generate requirements.txt file

generate requirements.txt file (if not present) in the project folder using

```
pip freeze requirements.txt
```

In the requirements.txt file, keep the library or package names that are needed for the corresponding project.

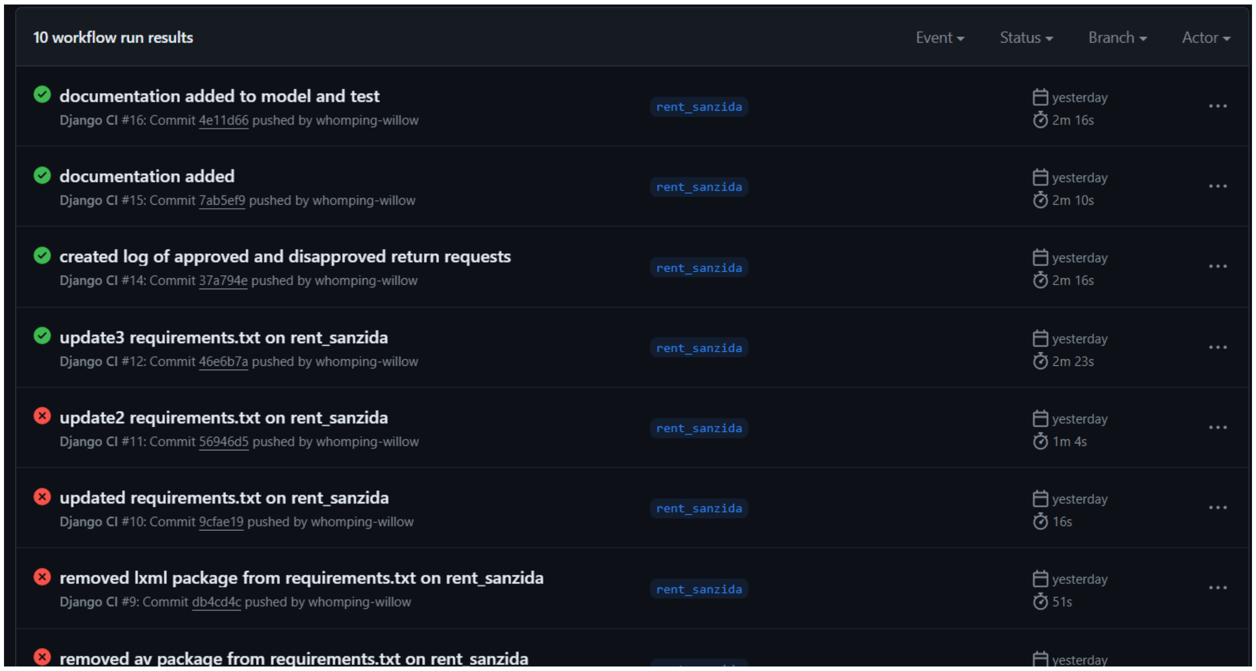
Step 3: Commit and Push Changes

Add the new workflow file and any changes you made to include the monthly and advance payment tests.

```
git add .
git commit -m "commit message"
git push origin <branch-name>
```

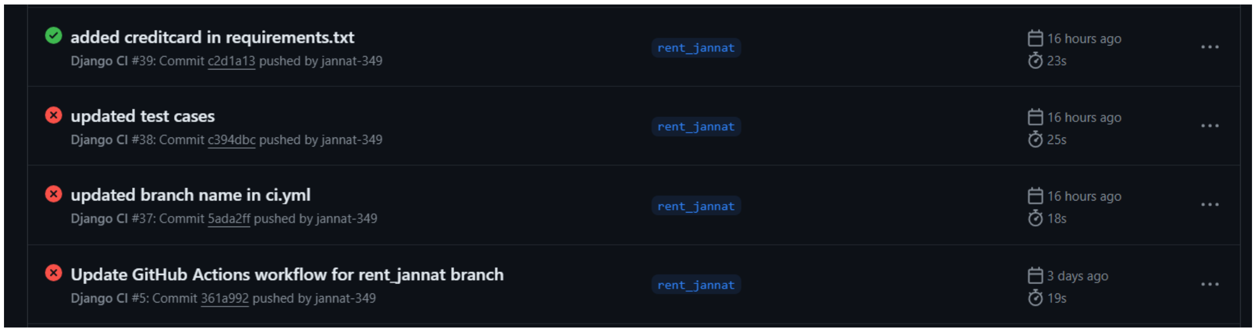
Step 4: Monitor GitHub Actions

Go to your GitHub repository and navigate to the "Actions" tab. You should see the GitHub Actions workflow running. If everything is set up correctly, the workflow will run your tests on each push to the corresponding branch.



10 workflow run results

✓	documentation added to model and test Django CI #16: Commit 4e11d66 pushed by whomping-willow	rent_sanzida	yesterday	2m 16s
✓	documentation added Django CI #15: Commit 7ab5ef9 pushed by whomping-willow	rent_sanzida	yesterday	2m 10s
✓	created log of approved and disapproved return requests Django CI #14: Commit 37a794e pushed by whomping-willow	rent_sanzida	yesterday	2m 16s
✓	update3 requirements.txt on rent_sanzida Django CI #12: Commit 46e6b7a pushed by whomping-willow	rent_sanzida	yesterday	2m 23s
✗	update2 requirements.txt on rent_sanzida Django CI #11: Commit 56946d5 pushed by whomping-willow	rent_sanzida	yesterday	1m 4s
✗	updated requirements.txt on rent_sanzida Django CI #10: Commit 9cfae19 pushed by whomping-willow	rent_sanzida	yesterday	16s
✗	removed lxml package from requirements.txt on rent_sanzida Django CI #9: Commit db4cd4c pushed by whomping-willow	rent_sanzida	yesterday	51s
✗	removed av package from requirements.txt on rent_sanzida		yesterday	

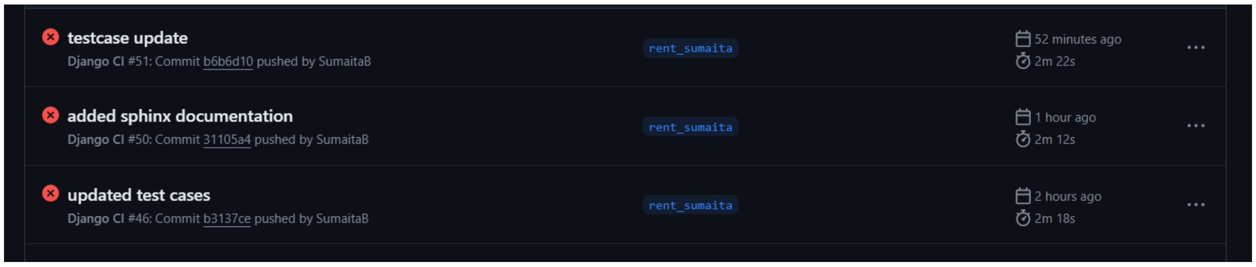


added creditcard in requirements.txt
Django CI #39: Commit c2d1a13 pushed by jannat-349 · rent_jannat · 16 hours ago · 23s

updated test cases
Django CI #38: Commit c394dbc pushed by jannat-349 · rent_jannat · 16 hours ago · 25s

updated branch name in ci.yml
Django CI #37: Commit 5ada2ff pushed by jannat-349 · rent_jannat · 16 hours ago · 18s

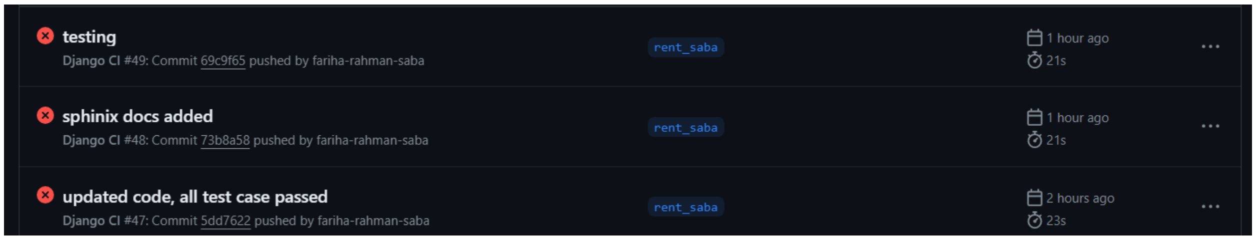
Update GitHub Actions workflow for rent_jannat branch
Django CI #5: Commit 361a992 pushed by jannat-349 · rent_jannat · 3 days ago · 19s



testcase update
Django CI #51: Commit b6b6d10 pushed by SumaitaB · rent_sumaita · 52 minutes ago · 2m 22s

added sphinx documentation
Django CI #50: Commit 31105a4 pushed by SumaitaB · rent_sumaita · 1 hour ago · 2m 12s

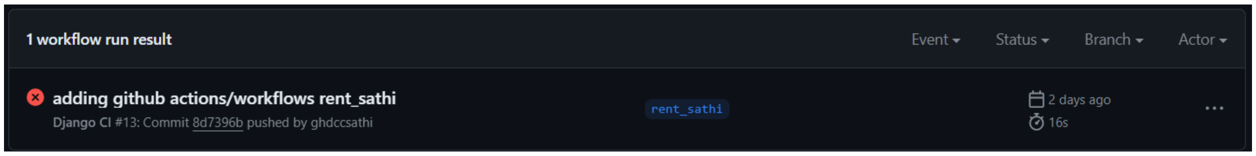
updated test cases
Django CI #46: Commit b3137ce pushed by SumaitaB · rent_sumaita · 2 hours ago · 2m 18s



testing
Django CI #49: Commit 69c9f65 pushed by fariha-rahman-saba · rent_saba · 1 hour ago · 21s

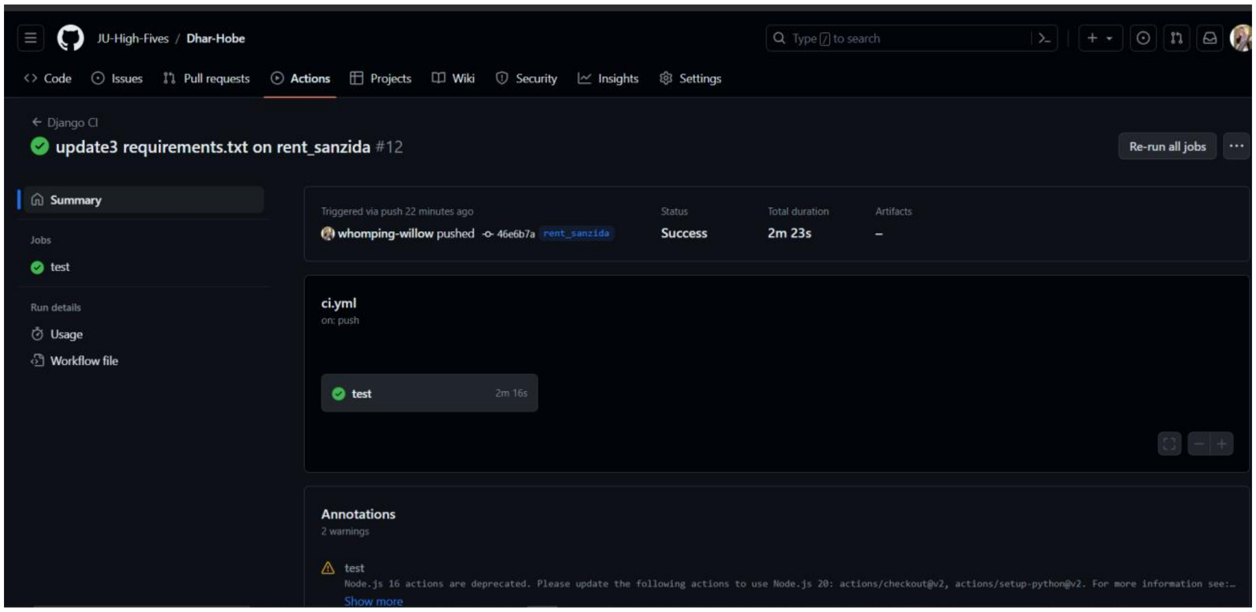
sphinix docs added
Django CI #48: Commit 73b8a58 pushed by fariha-rahman-saba · rent_saba · 1 hour ago · 21s

updated code, all test case passed
Django CI #47: Commit 5dd7622 pushed by fariha-rahman-saba · rent_saba · 2 hours ago · 23s



1 workflow run result

adding github actions/workflows rent_sathi
Django CI #13: Commit 8d7396b pushed by ghdccsathi · rent_sathi · 2 days ago · 16s



update3 requirements.txt on rent_sanzida #12

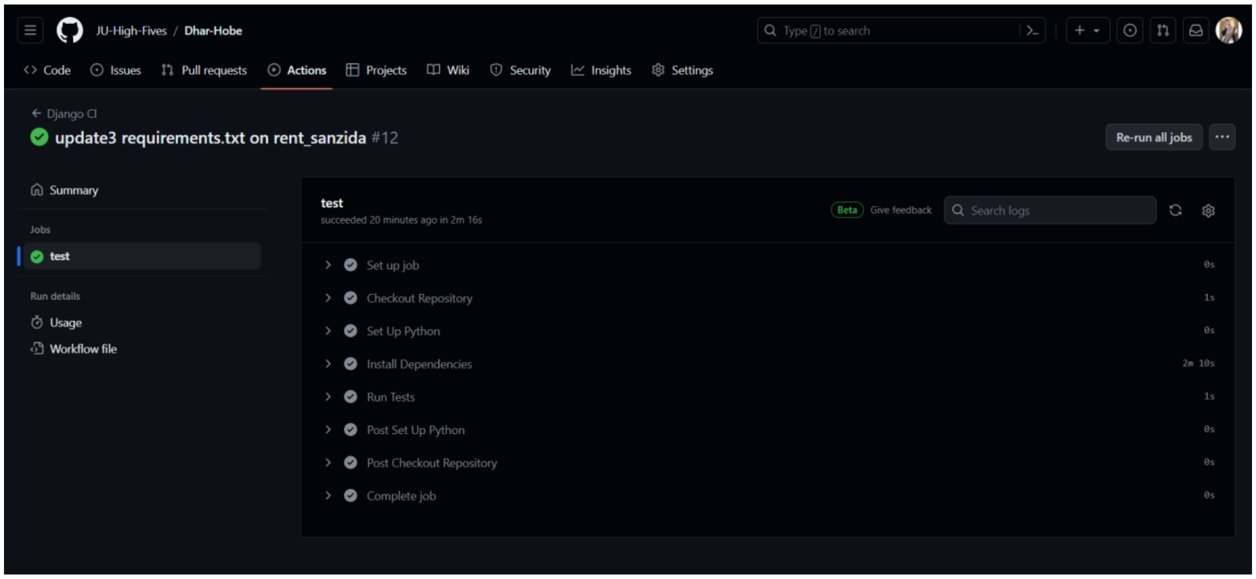
Jobs: test

Triggered via push 22 minutes ago
whomping-willow pushed 46e6b7a rent_sanzida
Status: Success
Total duration: 2m 23s
Artifacts: –

ci.yml
on: push
test 2m 16s

Annotations
2 warnings

test
```
Node.js 16 actions are deprecated. Please update the following actions to use Node.js 20: actions/checkout@v2, actions/setup-python@v2. For more information see:…
```
Show more



update3 requirements.txt on rent_sanzida #12

test
succeeded 20 minutes ago in 2m 16s

```
Set up job                 0s
Checkout Repository        1s
Set Up Python              0s
Install Dependencies       2m 10s
Run Tests                  1s
Post Set Up Python         0s
Post Checkout Repository   0s
Complete job               0s
```

References:

1. https://docs.github.com/en/actions/automating-builds-and-tests/about-continuous-integration
2. https://github.blog/2022-02-02-build-ci-cd-pipeline-github-actions-four-steps/
3. https://medium.com/nimbella/ci-cd-pipeline-with-github-actions-71abd144ddb4