

Continuous Integration Testing Report

Continuous Integration (CI) in GitHub is a development practice where code changes are automatically tested and verified as part of the software development process. It involves integrating code changes into a shared repository frequently, typically several times a day. Each integration triggers automated builds and tests, ensuring that the codebase remains in a functional state at all times.

Here we used GitHub Actions as an integration tool .

Steps:

1. Defining CI Workflows: Create a `.github/workflows` directory in our repository and define YAML files to define our CI workflows. These files typically specify the events that trigger the workflow (e.g., pushes, pull requests) and the sequence of steps to be executed.

2. Setting Up Jobs and Steps: Within our workflow file, define one or more jobs, each representing a set of related tasks to be performed. Each job consists of a series of steps, which are individual units of work, such as checking out code, installing dependencies, running tests, and deploying artifacts.

3. Configuring Triggers: Configure triggers for our workflows to specify when they should be executed. This could include triggers based on specific branches, tags, or types of events (e.g., `pull_request`, `push`).

4. Using Actions and Services: Leverage the vast ecosystem of GitHub Actions and third-party actions to perform common tasks in our workflows. These actions encapsulate reusable logic and can be easily integrated into your workflows.

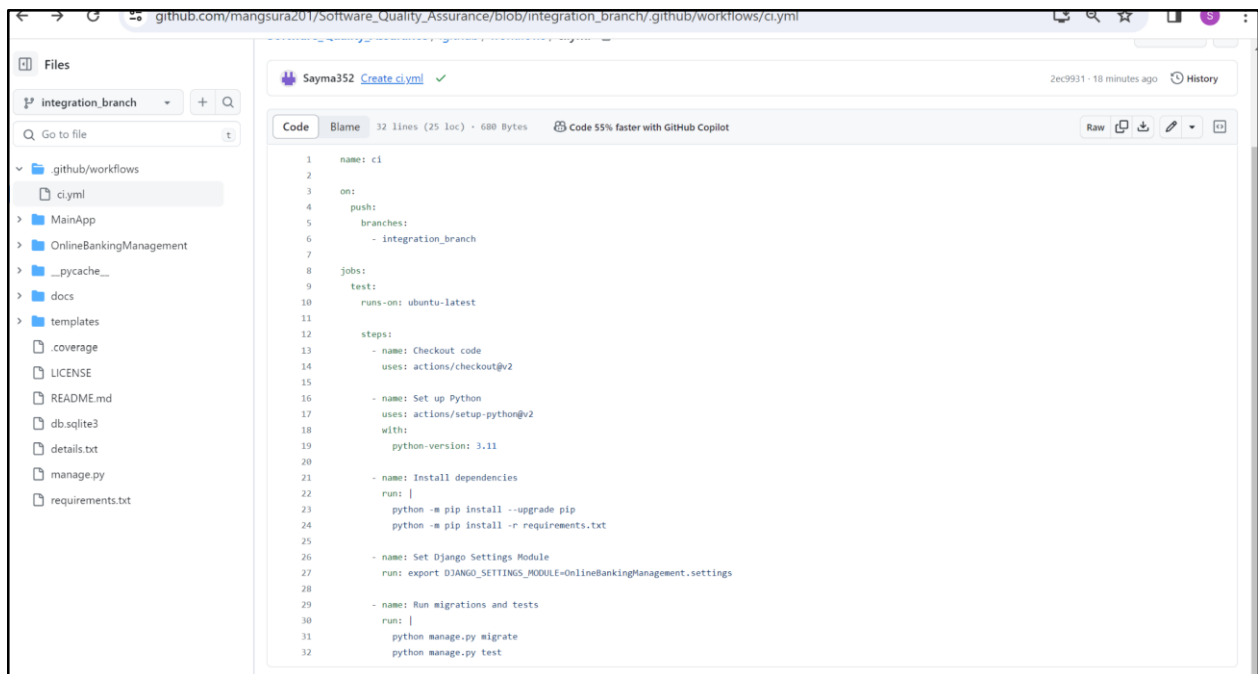
5. Running Tests: Use your CI workflows to run automated tests on our codebase whenever changes are made.

6. Monitoring and Notifications: Monitor the status of our CI workflows directly within GitHub. GitHub provides detailed logs and status indicators for each workflow run, allowing us to quickly identify any issues. We can also configure notifications to alert of workflow failures or other important events.

7. Integrating with External Services: GitHub Actions can integrate with external services, such as cloud providers, code quality tools, or deployment platforms, to automate additional tasks in our CI process. This enables us to create end-to-end CI/CD pipelines tailored to our specific requirements.

At first we tried to test each branch individually. Since we have dependencies among branches we weren't able to test each branch successfully.

But we are successfully able to test the “integration_branch” where we merged all the branches. Here is the snapshot -



```
1 name: ci
2
3 on:
4   push:
5     branches:
6       - integration_branch
7
8 jobs:
9   test:
10    runs-on: ubuntu-latest
11
12    steps:
13      - name: Checkout code
14        uses: actions/checkout@v2
15
16      - name: Set up Python
17        uses: actions/setup-python@v2
18        with:
19          python-version: 3.11
20
21      - name: Install dependencies
22        run: |
23          python -m pip install --upgrade pip
24          python -m pip install -r requirements.txt
25
26      - name: Set Django Settings Module
27        run: export DJANGO_SETTINGS_MODULE=OnlineBankingManagement.settings
28
29      - name: Run migrations and tests
30        run: |
31          python manage.py migrate
32          python manage.py test
```

3 workflow runs		Event ▾	Status ▾	Branch ▾	Actor ▾
✓	Create ci.yml ci #3: Commit 2ec9931 pushed by Sayma352			integration_branch	2 minutes ago 17s

← ci

✓ Create ci.yml #3

Re-run all jobs



Summary

Jobs

✓ test

Run details

Usage

Workflow file

test

succeeded 15 minutes ago in 8s

Beta

Give feedback

Search logs



>	✓ Set up job	0s
>	✓ Checkout code	1s
>	✓ Set up Python	0s
>	✓ Install dependencies	4s
>	✓ Set Django Settings Module	0s
>	✓ Run migrations and tests	1s
>	✓ Post Set up Python	0s
>	✓ Post Checkout code	0s
>	✓ Complete job	0s