ou’ve completed Step 2 (pushing the repository to GitHub). Let’s continue step-by-step with the remaining setup for the CI/CD pipeline and testing automation.

**Step 3: Verify the GitHub Repository**

1. **Check Repository on GitHub**:
   * Visit your GitHub account.
   * Verify that your LMS\_Project repository has been created and that all files are visible.
2. **Verify the .github/workflows/ci.yml File**:
   * Ensure that the workflow file for the CI pipeline exists in:

bash

Copy code

.github/workflows/ci.yml

* + The file should match this content:

yaml

Copy code

name: CI Pipeline

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

test:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Python

uses: actions/setup-python@v2

with:

python-version: '3.9'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

- name: Run migrations

run: python manage.py migrate

- name: Run tests

run: python manage.py test

1. **Push the Workflow File**: If the workflow file hasn’t been pushed yet, do so:

bash

Copy code

git add .github/workflows/ci.yml

git commit -m "Add CI pipeline with GitHub Actions"

git push origin main

**Step 4: Trigger the Workflow**

1. **Trigger the CI Pipeline**:
   * Push a change to the main branch to trigger the CI pipeline.

bash

Copy code

git commit --allow-empty -m "Trigger CI pipeline"

git push origin main

1. **Check Workflow Status**:
   * On GitHub, go to the **Actions** tab in your repository.
   * Verify that the pipeline is running.
2. **Monitor Results**:
   * Confirm that all steps (e.g., dependency installation, migrations, and tests) complete successfully.

**Step 5: Write Unit Tests**

1. **Create a Test File**: Add a new file under tests/ (if it doesn’t exist):

bash

Copy code

mkdir -p courses/tests

nano courses/tests/test\_scorm\_playback.py

1. **Write a Basic Test**: Paste the following content:

python

Copy code

from django.test import TestCase

from courses.models import SCORM, Course

class SCORMPlaybackTestCase(TestCase):

def setUp(self):

self.course = Course.objects.create(title="Test Course", description="Test Description")

self.scorm = SCORM.objects.create(

title="Test SCORM",

course=self.course,

launch\_url="/course\_content/scorm/1/test/index.html"

)

def test\_scorm\_launch\_url(self):

response = self.client.get(self.scorm.launch\_url)

self.assertEqual(response.status\_code, 200)

1. **Run the Test Locally**:

bash

Copy code

python manage.py test

1. **Push the Test**:

bash

Copy code

git add courses/tests/test\_scorm\_playback.py

git commit -m "Add SCORM playback test"

git push origin main

1. **Verify the Workflow**:
   * On GitHub, check the **Actions** tab to see if the test runs successfully in the pipeline.

**Step 6: Automate Testing for Runtime Tracking**

1. **Create Another Test File**:

bash

Copy code

nano courses/tests/test\_runtime\_tracking.py

1. **Write a Test for Runtime Tracking**:

python

Copy code

from django.test import TestCase

from courses.models import SCORM, Course

import json

class SCORMRuntimeTrackingTestCase(TestCase):

def setUp(self):

self.course = Course.objects.create(title="Test Course", description="Test Description")

self.scorm = SCORM.objects.create(

title="Test SCORM",

course=self.course,

launch\_url="/course\_content/scorm/1/test/index.html"

)

def test\_runtime\_tracking\_update(self):

runtime\_data = {

"progress": 50,

"score": 85,

"completion\_status": "in\_progress"

}

response = self.client.post(

f"/courses/scorm/runtime/update/{self.course.id}/",

data=json.dumps(runtime\_data),

content\_type="application/json"

)

self.assertEqual(response.status\_code, 200)

self.assertEqual(response.json()["status"], "success")

1. **Run the Test Locally**:

bash

Copy code

python manage.py test

1. **Push the Test**:

bash

Copy code

git add courses/tests/test\_runtime\_tracking.py

git commit -m "Add runtime tracking test"

git push origin main

1. **Monitor the Pipeline**:
   * Verify that the runtime tracking test runs successfully in GitHub Actions.

**Step 7: Set Up Docker for Development**

1. **Create a Dockerfile**:

bash

Copy code

nano Dockerfile

Paste the following:

dockerfile

Copy code

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

COPY . .

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]

1. **Create a docker-compose.yml**:

bash

Copy code

nano docker-compose.yml

Paste the following:

yaml

Copy code

version: '3.8'

services:

web:

build: .

ports:

- "8000:8000"

volumes:

- .:/app

environment:

- DEBUG=True

1. **Run the Application**:

bash

Copy code

docker-compose up

1. **Verify the Application**:
   * Visit: http://127.0.0.1:8000

**Next Steps**

* Let me know where you’d like to begin or if you encounter any issues during the process.
* I’ll guide you through each step and verify that everything is working as intended.

**You said:**

harikrishnan@haris-mbp LMS\_Project % .github/workflows/ci.yml zsh: no such file or directory: .github/workflows/ci.yml harikrishnan@haris-mbp LMS\_Project %

**ChatGPT said:**

ChatGPT

The error indicates that the .github/workflows/ci.yml file does not exist. Let’s create it step by step:

**Step 1: Create the Directory Structure**

Ensure the .github/workflows/ directory exists:

bash

Copy code

mkdir -p .github/workflows

**Step 2: Create the Workflow File**

Create the ci.yml file:

bash

Copy code

nano .github/workflows/ci.yml

Paste the following content into the file:

yaml

Copy code

name: CI Pipeline

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

test:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Python

uses: actions/setup-python@v2

with:

python-version: '3.9'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

- name: Run migrations

run: python manage.py migrate

- name: Run tests

run: python manage.py test

Save and exit the file (Ctrl + O, Enter, Ctrl + X).

**Step 3: Commit and Push the Workflow**

Add the workflow to Git and push it to GitHub:

bash

Copy code

git add .github/workflows/ci.yml

git commit -m "Add CI pipeline with GitHub Actions"

git push origin main

**Step 4: Verify on GitHub**

1. Go to your repository on GitHub.
2. Click the **Actions** tab.
3. Verify that the pipeline is listed and runs on the next push or pull request.