1. Create a new GitHub repository.

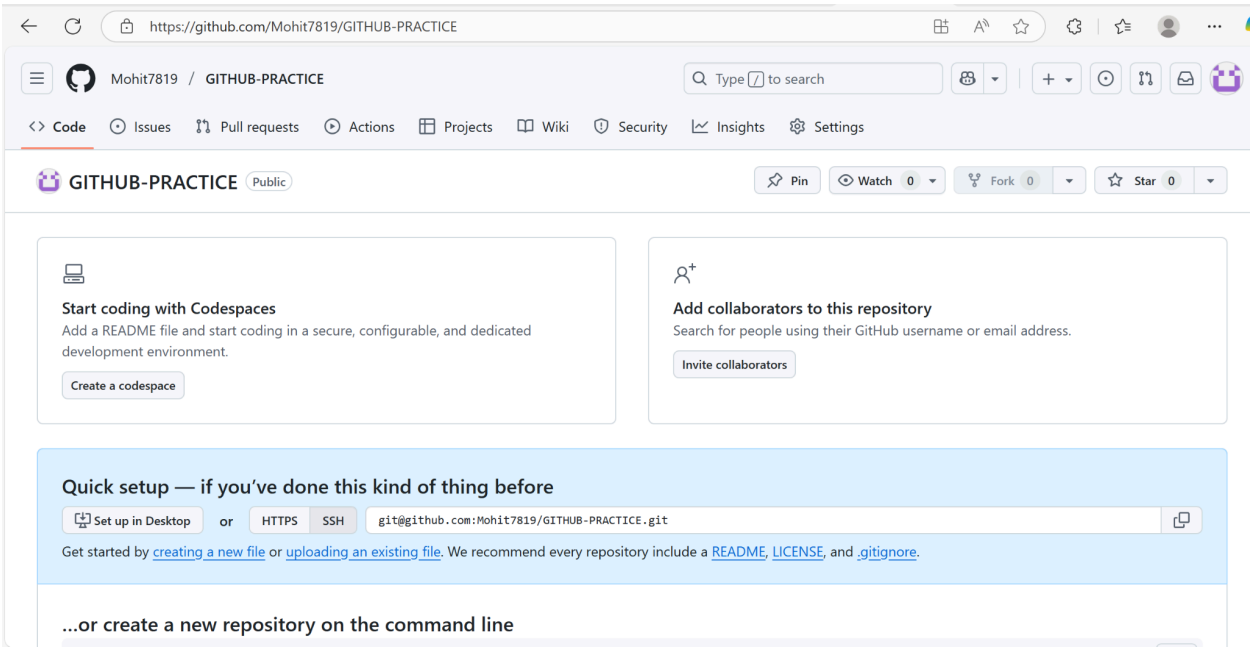
- Clone the repository to your local machine using SSH (generate an SSH key if needed, add the public key to your GitHub account).
- Create a new branch named after your username (e.g., `Tutedude`).
- Add your Flask project files to this branch.
- Commit the changes and merge the branch into the `main` branch.

🔹 Step 1: Create a New GitHub Repository

- I created a new repository named GITHUB-PRACTICE.

URL (SSH):

```
git@github.com:Mohit7819/GITHUB-PRACTICE.git
```

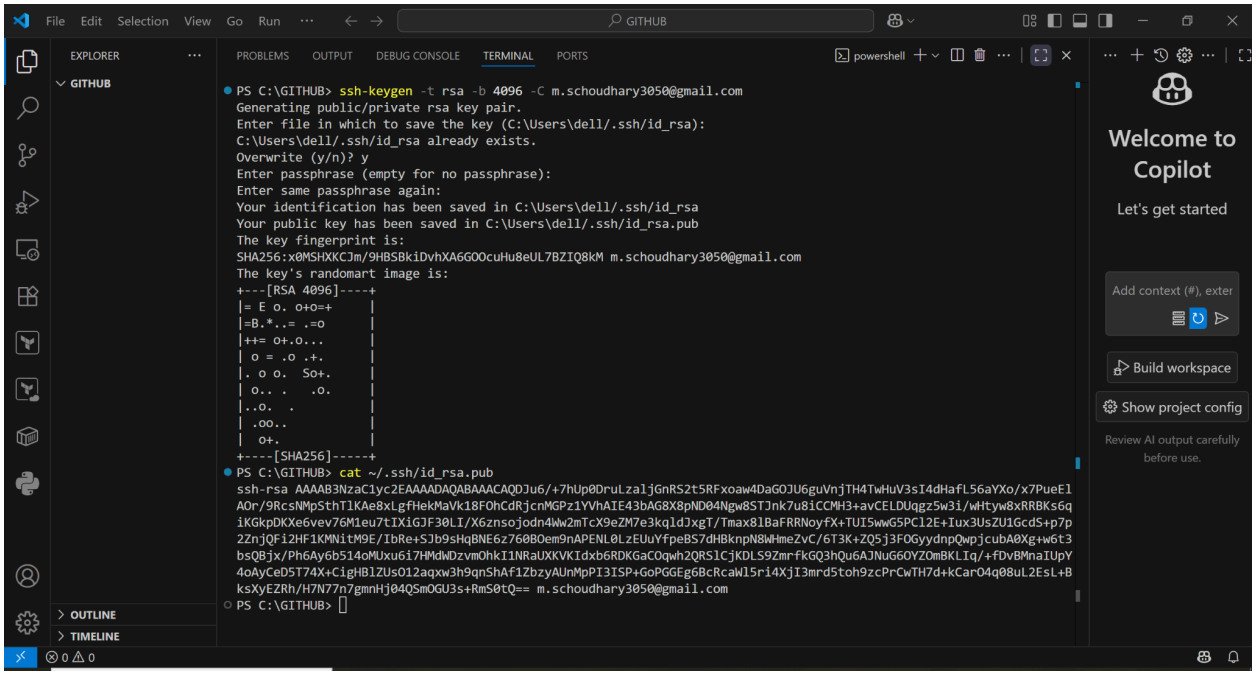


🔹 Step 2: Generate SSH Key and Add to GitHub

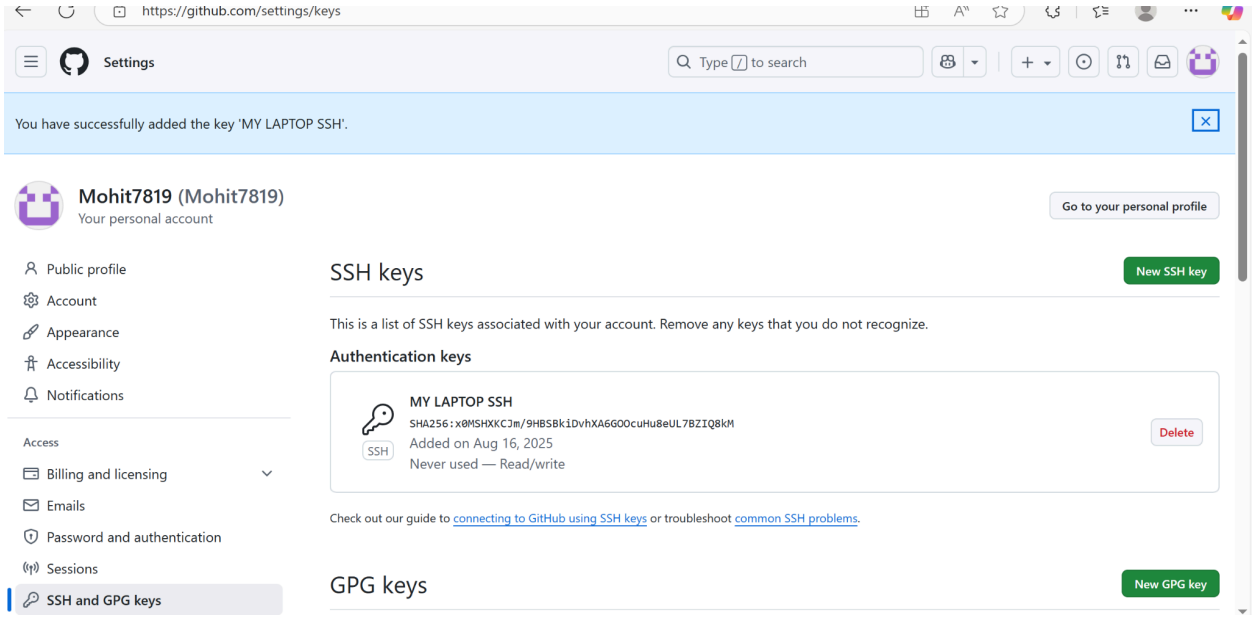
```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

- Copied the public key:

```
cat ssh.pub
```



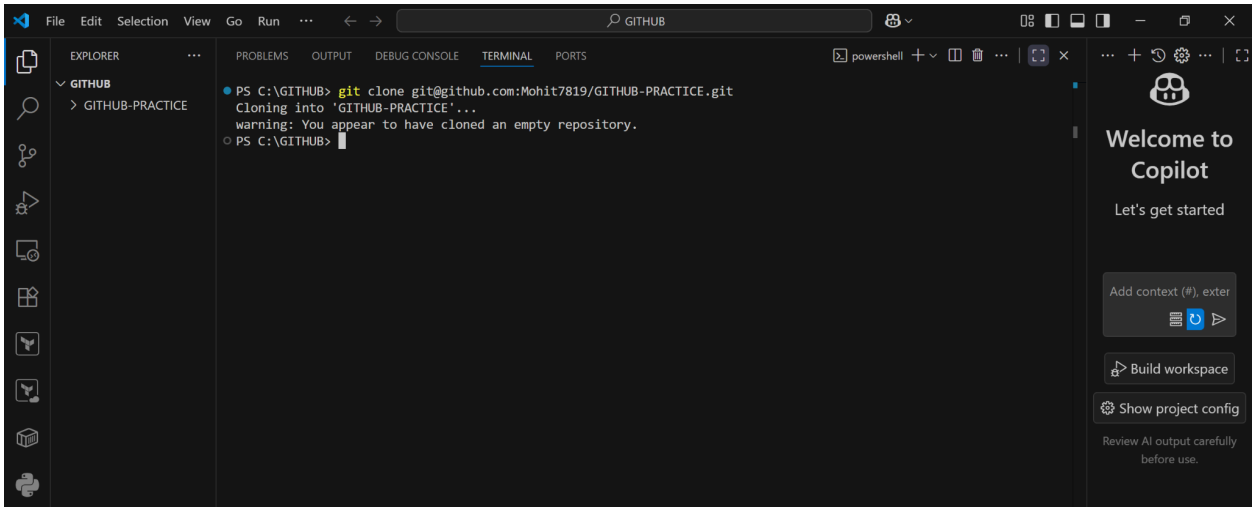
- Added it to GitHub under **Settings > SSH and GPG keys**



🔹 Step 3: Clone the Repository via SSH

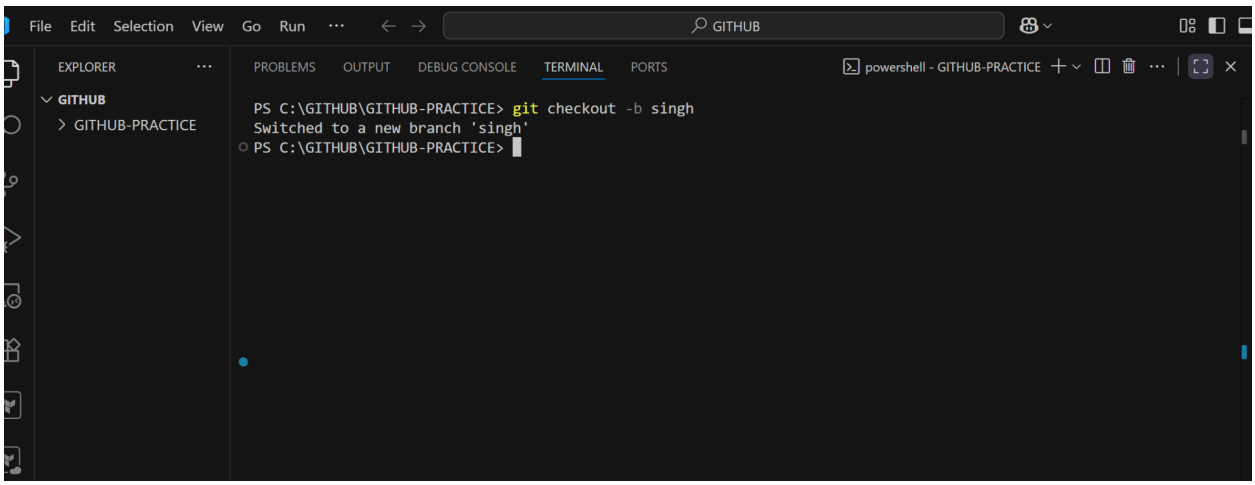
```
git clone git@github.com:Mohit7819/GITHUB-PRACTICE.git
```

```
cd GITHUB-PRACTICE
```



🔹 Step 4: Create a New Branch (Named After Your Username)

```
git checkout -b your-username
```

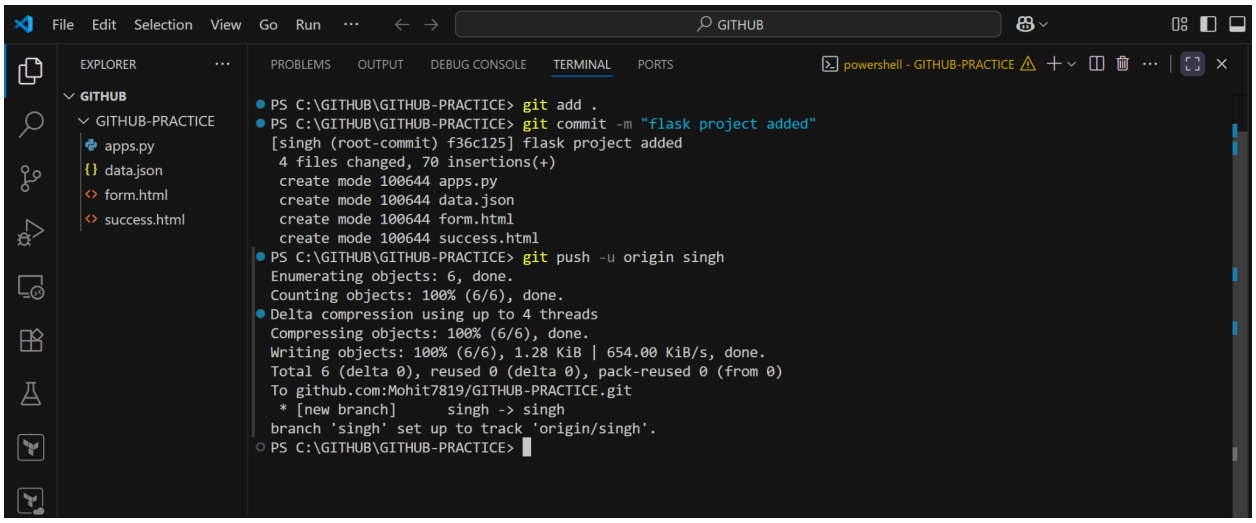


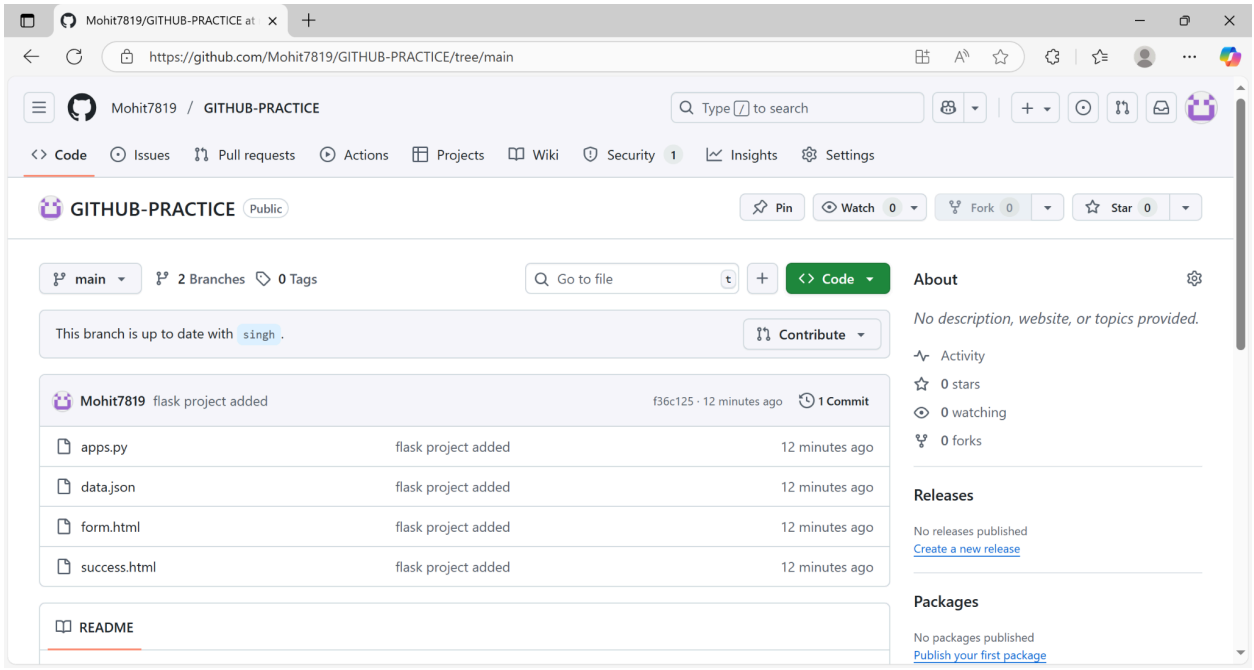
🔹 Step 5: Add Your Flask Project Files

- Copy your Flask project files into the GITHUB_PRACTICE folder.

🔹 Step 6: Add, Commit,Push and Merge Changes

```
git add .

git commit -m "Flask project added"

git push -u origin singh

git checkout main

git merge singh
```





2. Create a new branch named `<your_name>_new` (e.g., `Tutedude_new`).

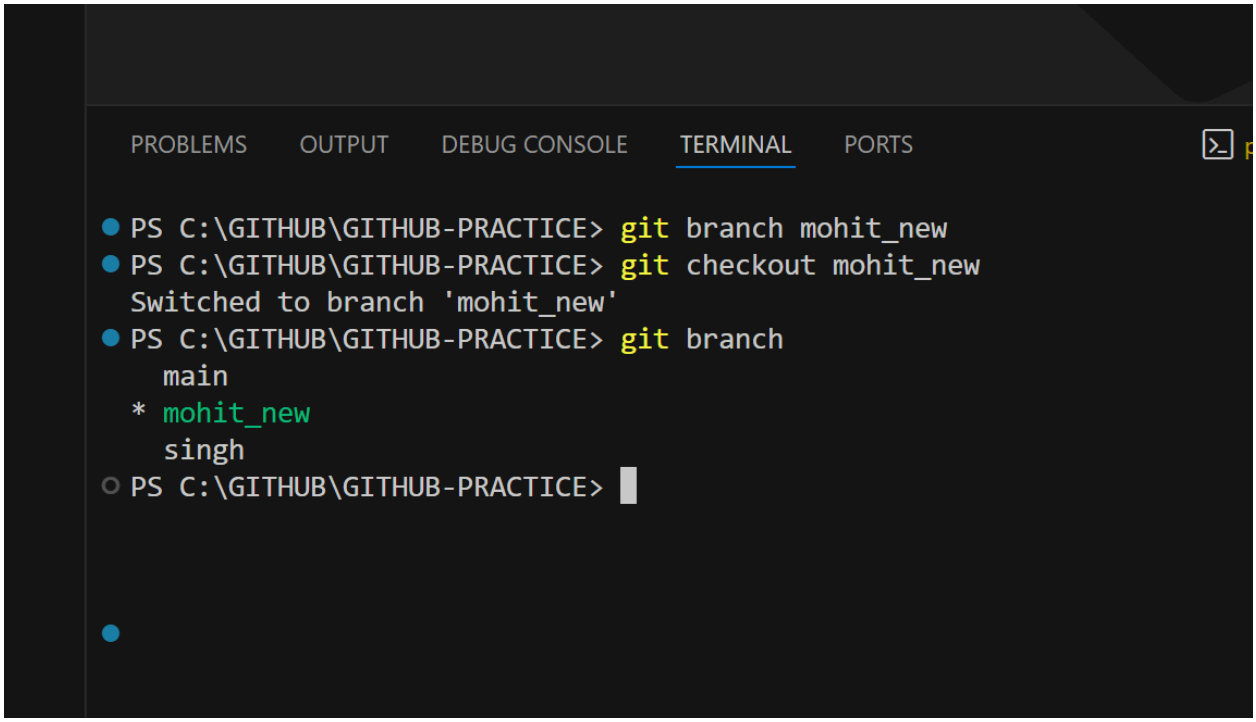
- Update the content of the JSON file used for the `/api` route in this branch.
- Merge the `<your_name>_new` branch into the `main` branch.
- If there are conflicts during the merge, resolve them by accepting the changes from the `<your_name>_new` branch.
- Add the resolved changes to the staging area, commit them, and push the updates to the remote repository.

🔹 Step 1: Create a New Branch Named `<your_name>_new`

```
git checkout -b yourname_new
```

OR

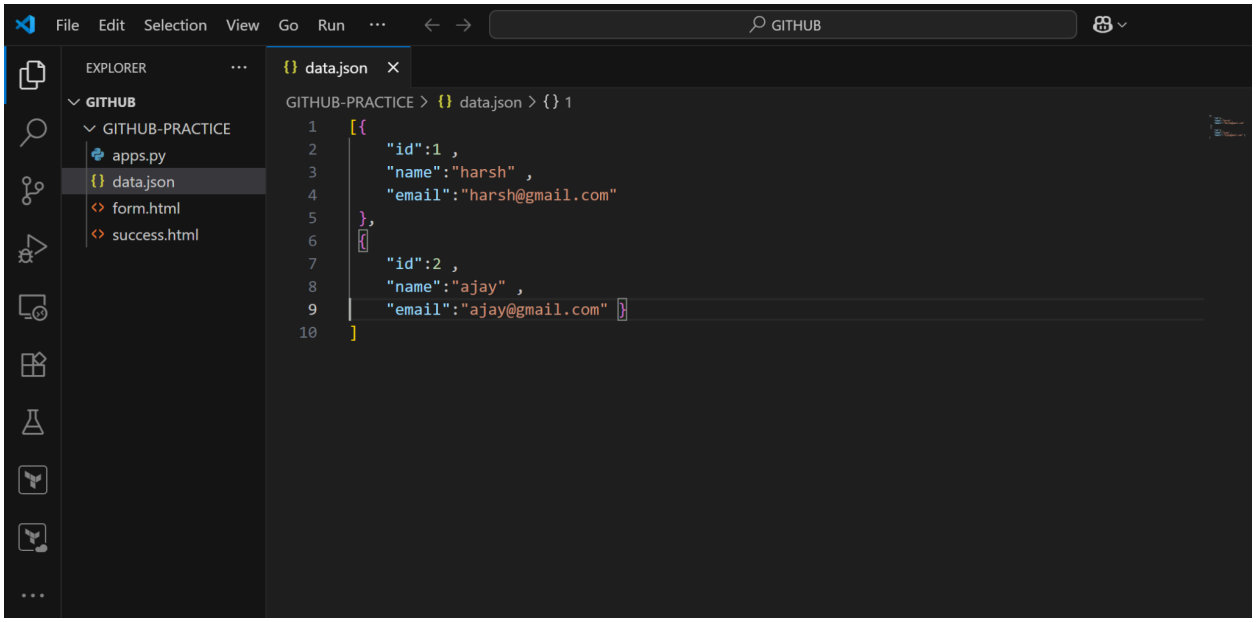
```
git branch mohit_new

git checkout mohit_new

git branch
```



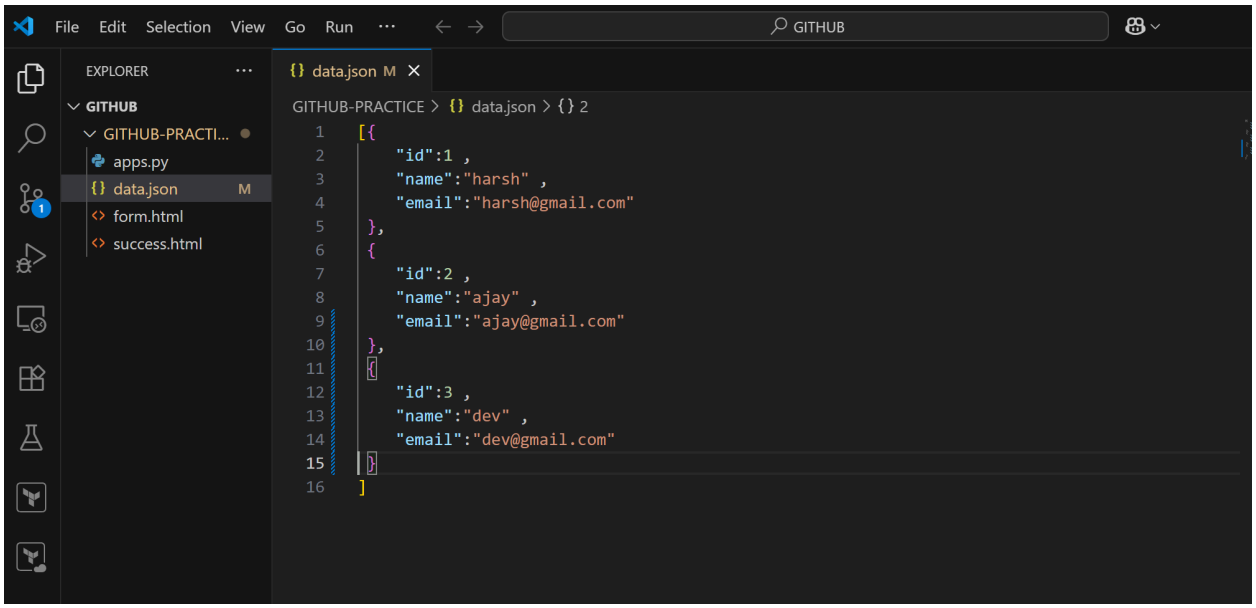
🔹 Step 2: Update the JSON File Used in `/api` Route

Let's say your Flask app has a `data.json` file used in the `/api` route. Open it and update the content.
For example:

Before (`data.json`):

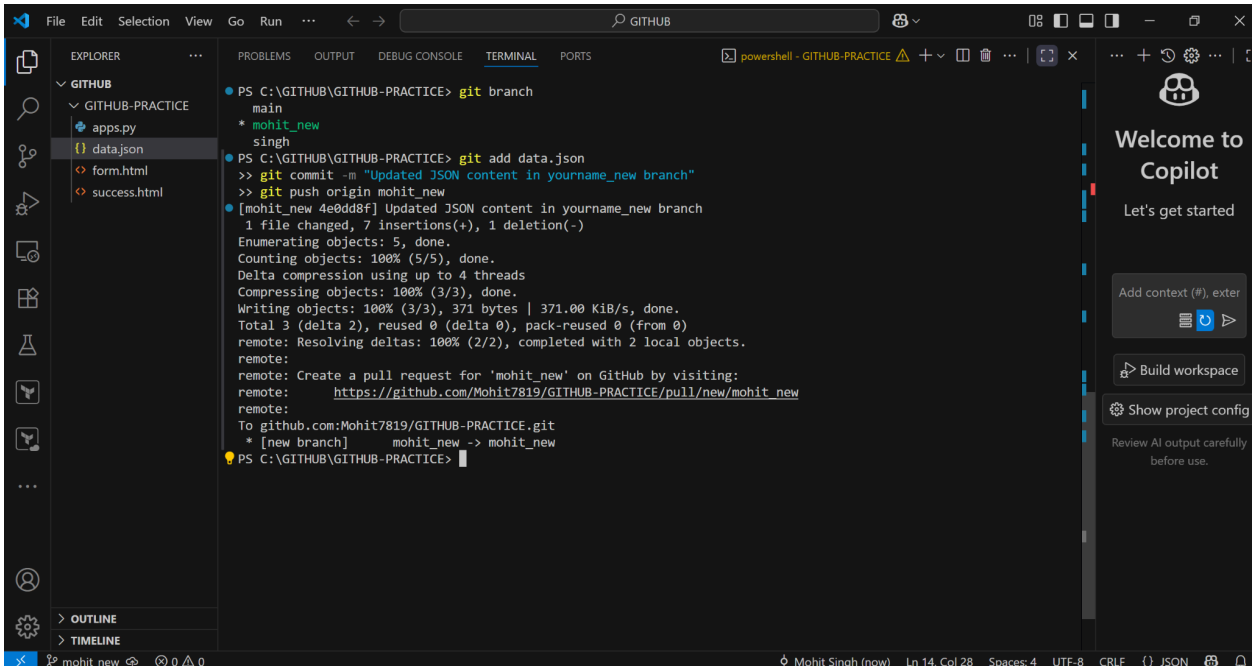


After:



🔹 Step 3: Add, Commit, and Push the Changes

```
git add data.json

git commit -m "Updated JSON content in yourname_new branch"

git push origin mohit_new
```



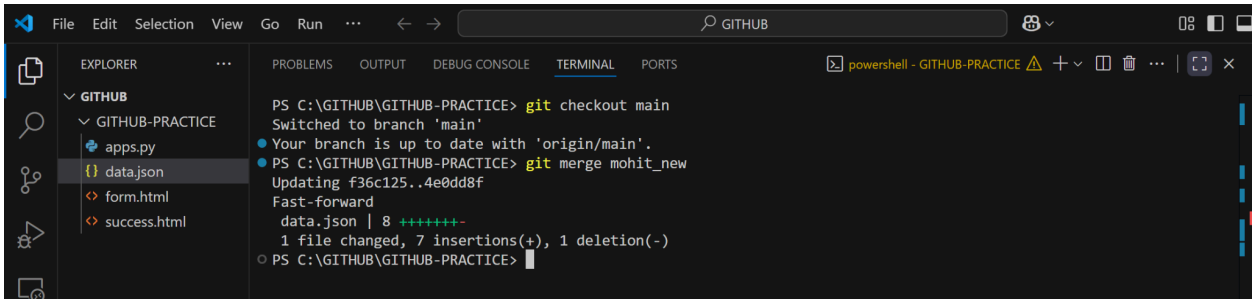
🔹 Step 4: Merge `mohit_new` Branch into `main`

First, switch back to the `main` branch:

```
git checkout main
```

Then, merge:

```
git merge mohit_new
```



🔹 Step 5: Push Final Changes to Remote

```
git push origin main
```

```
PS C:\GITHUB\GITHUB-PRACTICE> git push origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Mohit7819/GITHUB-PRACTICE.git
   f36c125..4e0dd8f  main -> main
PS C:\GITHUB\GITHUB-PRACTICE>
```

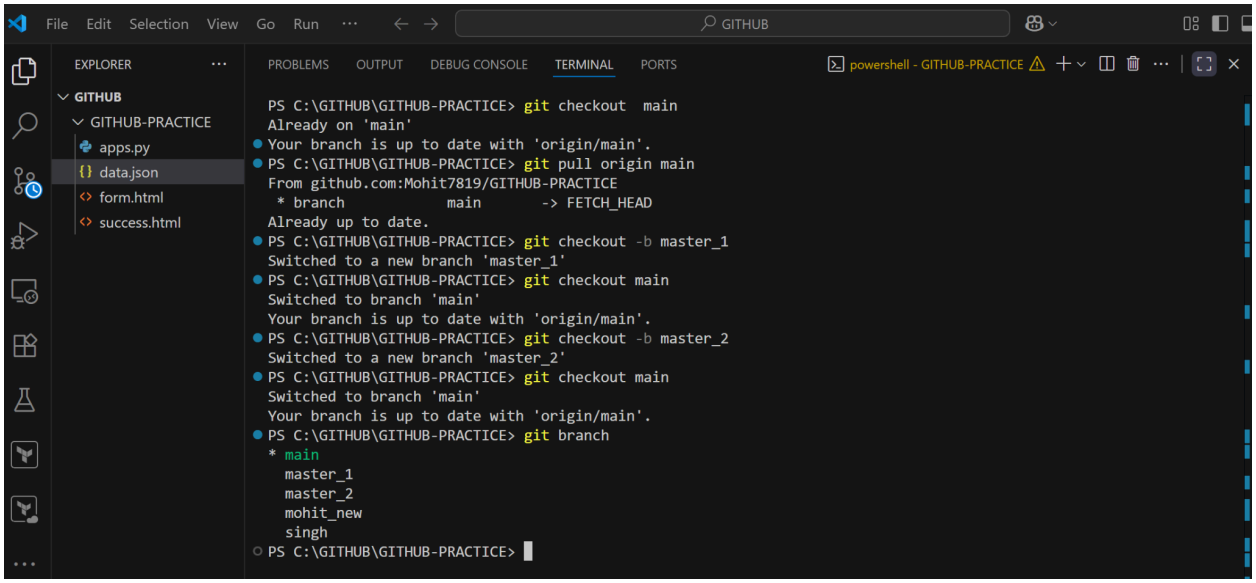
3. **Branch Creation**:

- Create two branches: `master_1` and `master_2` from the `main` branch.
- **Feature Development in** `master_1`:
- In the `master_1` branch, create a **To-Do Page** in the frontend.
    - The page should contain a form with the following fields:
        - **Item Name**
        - **Item Description**
- **Backend API in** `master_2`:
- In the `master_2` branch, create a backend route named `/submittodoitem`.
- This route will:
    - Accept `itemName` and `itemDescription` via a POST request.
    - Store these details in a MongoDB database.
- **Merging Changes**:
- Merge the changes from both `master_1` and `master_2` into the `main` branch.

🔹 Step 1: Create Branches `master_1` and `master_2` from `main`

```
# Make sure you're on main and up to date

git checkout main

git pull origin main


# Create and switch to master_1

git checkout -b master_1
```

```
# Switch back to main

git checkout main


# Create and switch to master_2

git checkout -b master_2
```

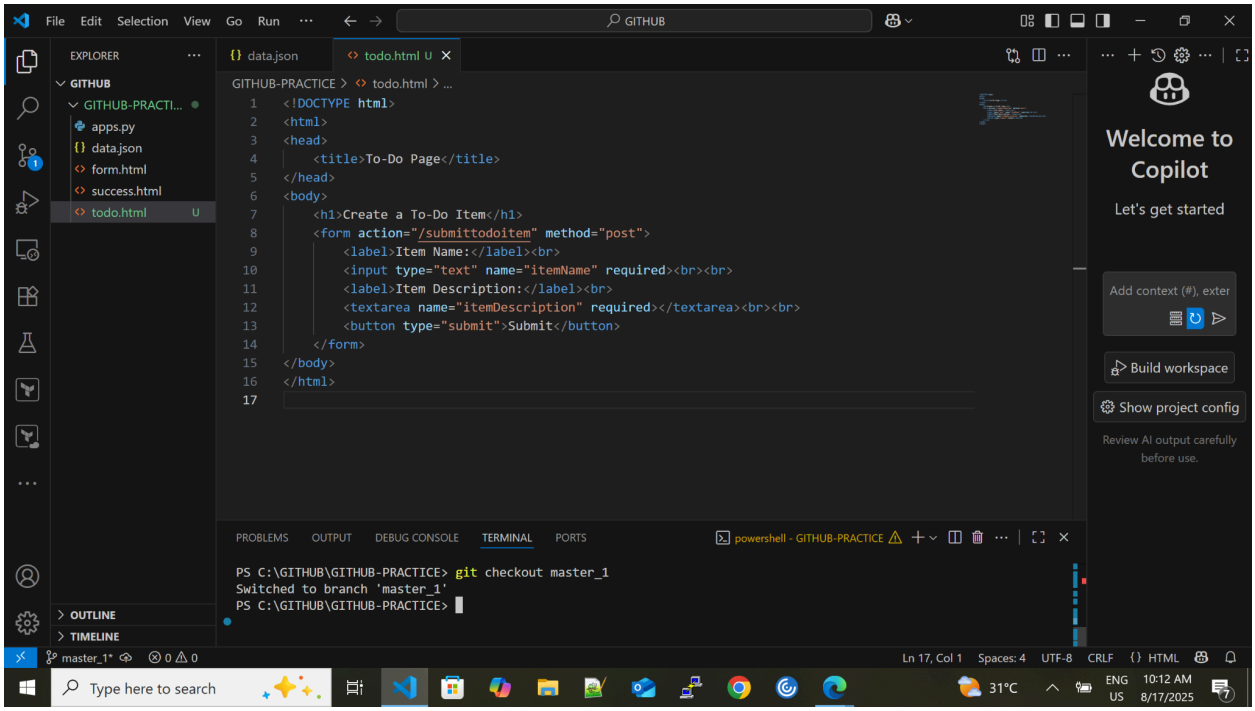


🔸 Part A: Feature Development in `master_1` (Frontend To-Do Page)

1. Switch to `master_1` branch if not already:

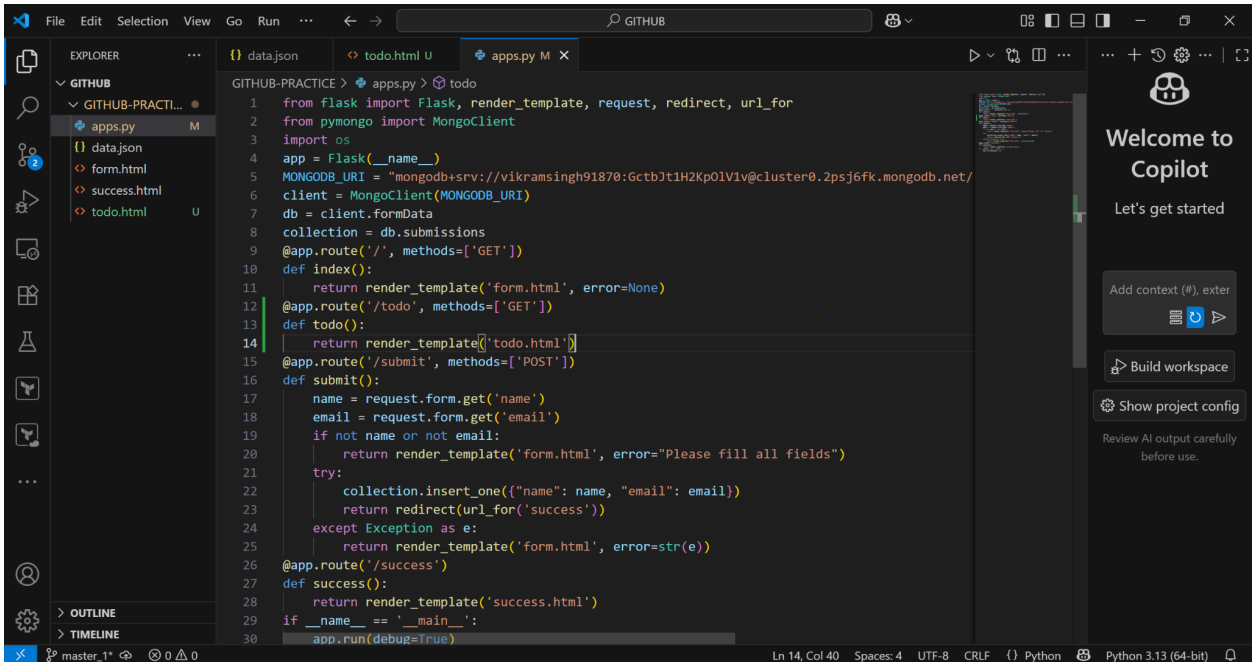
```
git checkout master_1
```

2. Create a file `todo.html` :



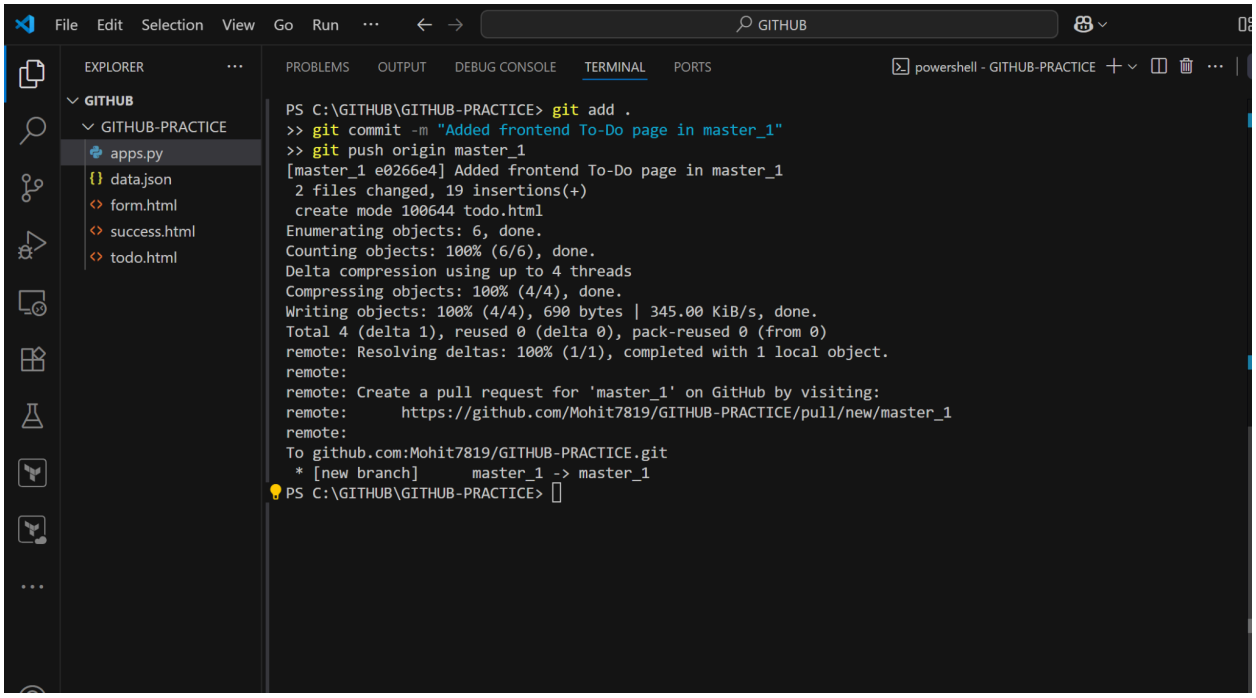
3. In your Flask app route (e.g., `apps.py`):

```
@app.route('/todo', methods=['GET'])
def todo():
    return render_template('todo.html')
```



4. Add, commit, and push:

```
git add .

git commit -m "Added frontend To-Do page in master_1"

git push origin master_1
```

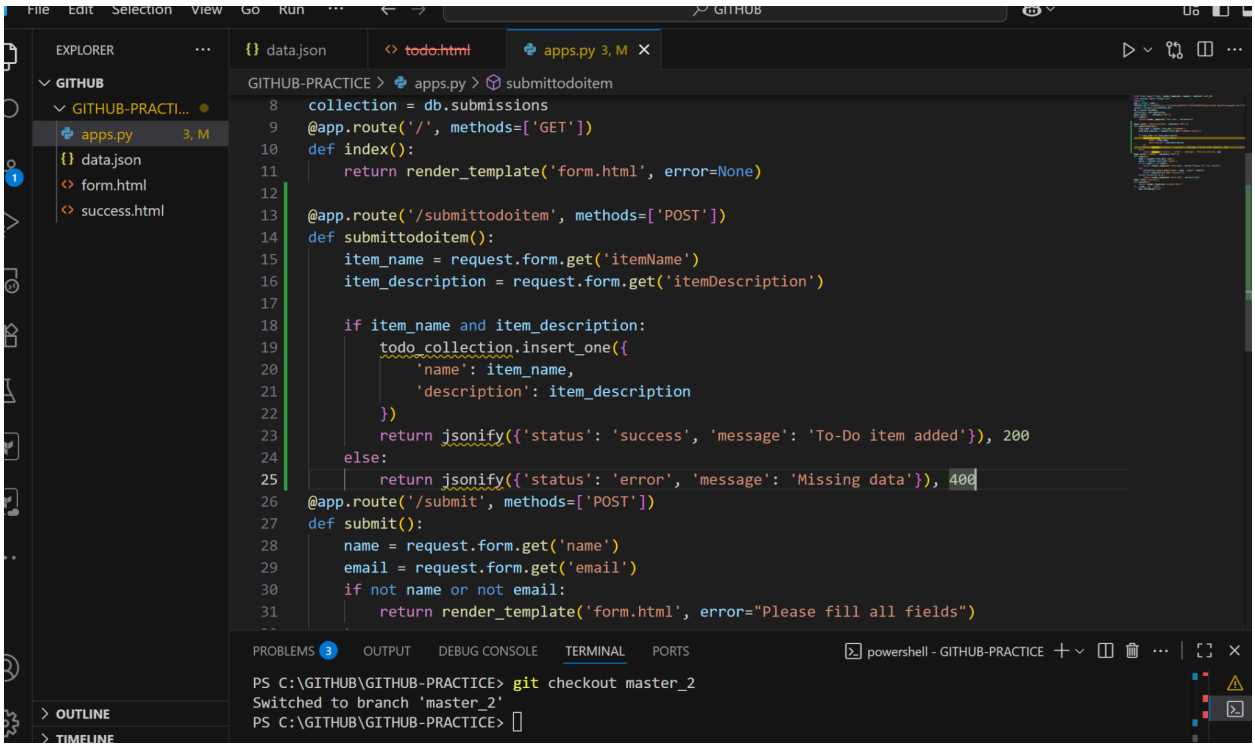


🔸 Part B: Backend API in master_2 (MongoDB Integration)

1. Switch to master_2 branch:

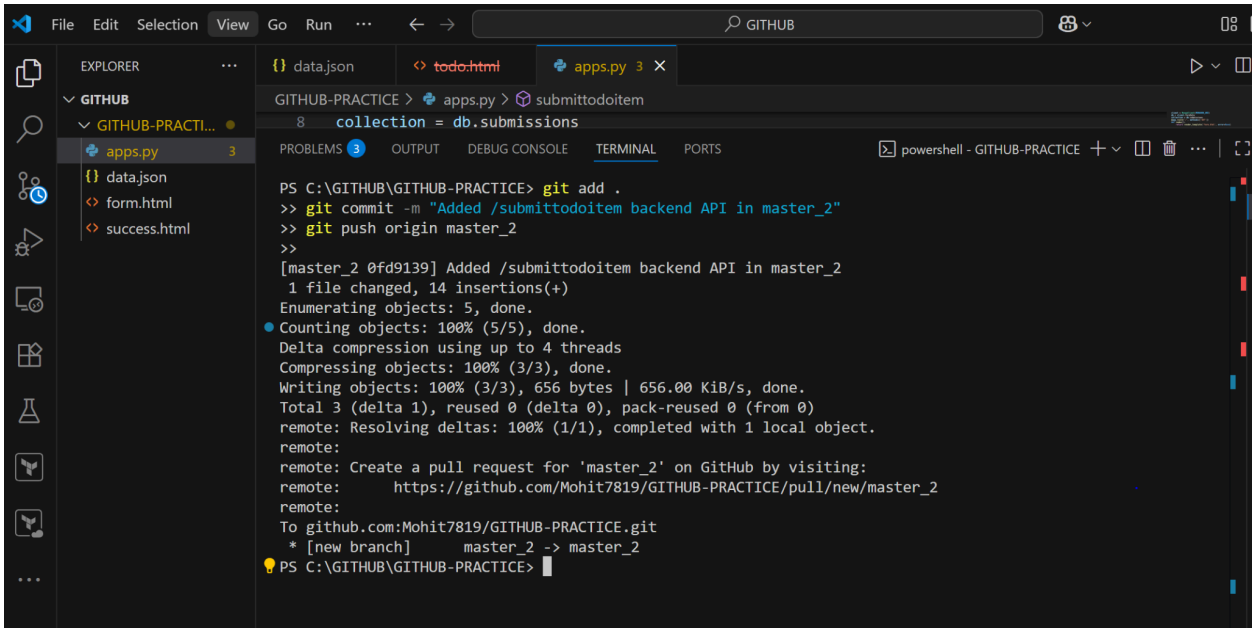
```
git checkout master_2
```

2. Add the /submittodoitem route to apps.py



Add, commit, and push:

```
git add .
git commit -m "Added /submittodoitem backend API in master_2"
git push origin master_2
```



🔹 Step 4: Merge master_1 and master_2 into main

1. First, switch to main:

```
git checkout main
```

2. Merge master_1:

```
git merge master_1
```

✅ Resolve any conflicts (if any), commit the resolved version:

```
git add .
git commit -m "Merged master_1 into main"
```

3. Then merge `master_2`:

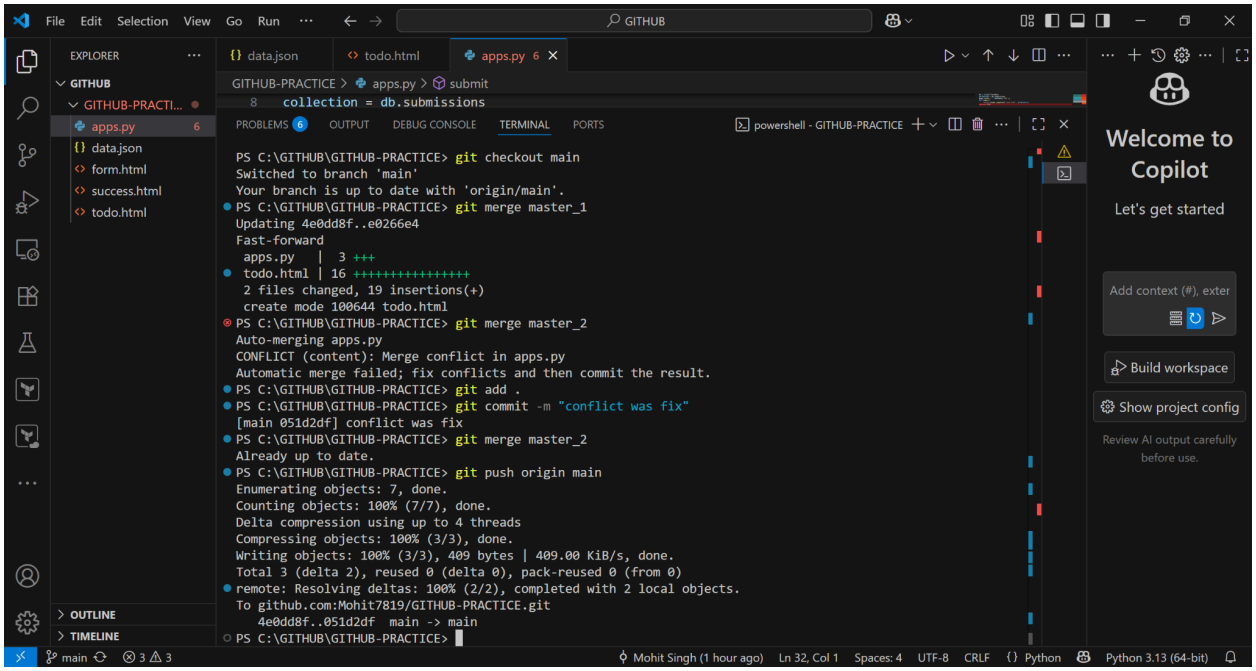
```
git merge master_2
```

✅ Resolve conflicts (if any), commit:

```
git add .
git commit -m "conflict was fix"
```

4. Push final merged code to GitHub:

```
git push origin main
```



4. **Enhancing the To-Do Form in** `master_1`:

- In the `master_1` branch, add the following fields to the To-Do form:
  - **Item ID**
  - **Item UUID**
  - **Item Hash**
- **Committing in Sequence**:
- Add and commit each field separately in the following order:
  - **First commit**: Add **Item ID** field.
  - **Second commit**: Add **Item UUID** field.
  - **Third commit**: Add **Item Hash** field.
- **Merging to** `main`:
- Merge the `master_1` branch into the `main` branch.
- **Git Reset and Commit Deletion**:
- In the `main` branch, use **Git Reset** to roll back to the commit where only the **Item ID** field was added.
- Use `git reset --soft` to ensure changes remain staged.
- Re-commit this state to the `main` branch.
- Merge this updated state to the `main` branch.
- **Rebasing Changes**:
- Rebase the updated changes in the `main` branch to the `master_1` branch.
  Clarification:
  - During rebasing, **preserve individual commits** to maintain the commit history for each change (i.e., do not squash commits).
  - Use `git rebase main master_1` to integrate changes from the `main` branch back into the `master_1` branch.

🔹 Step 1: Add Fields to To-Do Form in `master_1`

Make sure you're in the correct branch:

```
git checkout master_1
```

🔸 First Commit: Add Item ID Field

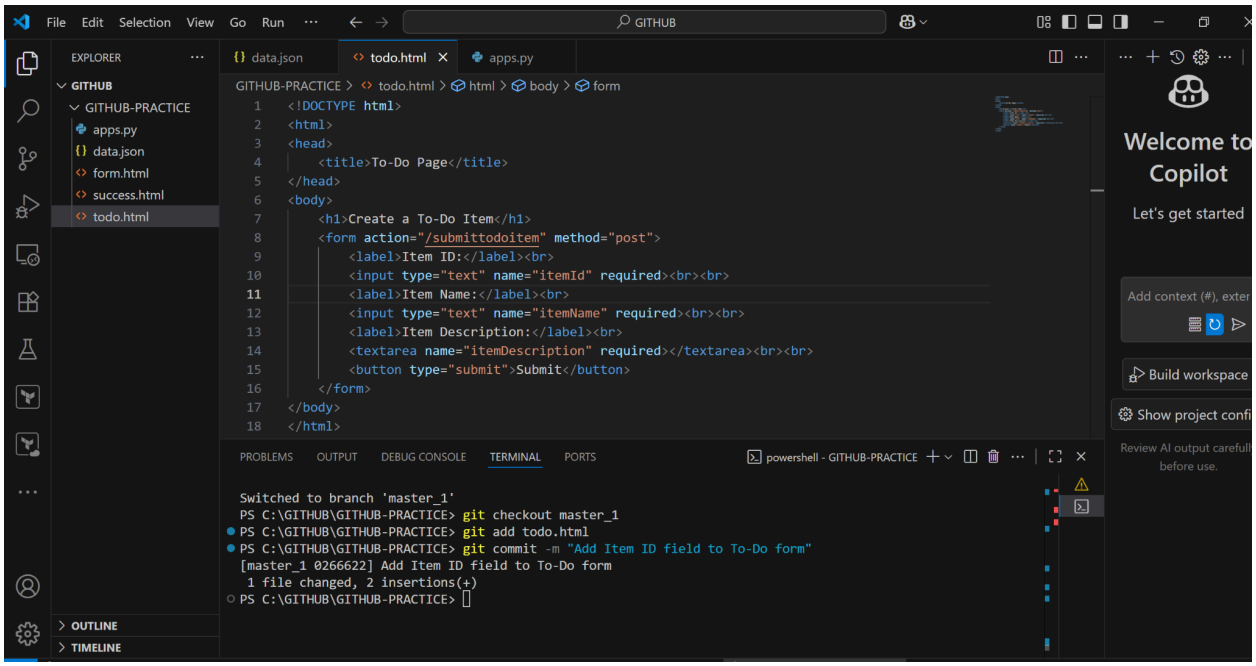
Update `todo.html`:

```
<label>Item ID:</label><br>

<input type="text" name="itemId" required><br><br>
```

Then commit:

```
git add todo.html
```

```
git commit -m "Add Item ID field to To-Do form"
```



🔸 **Second Commit: Add Item UUID Field**

Add to `todo.html`:

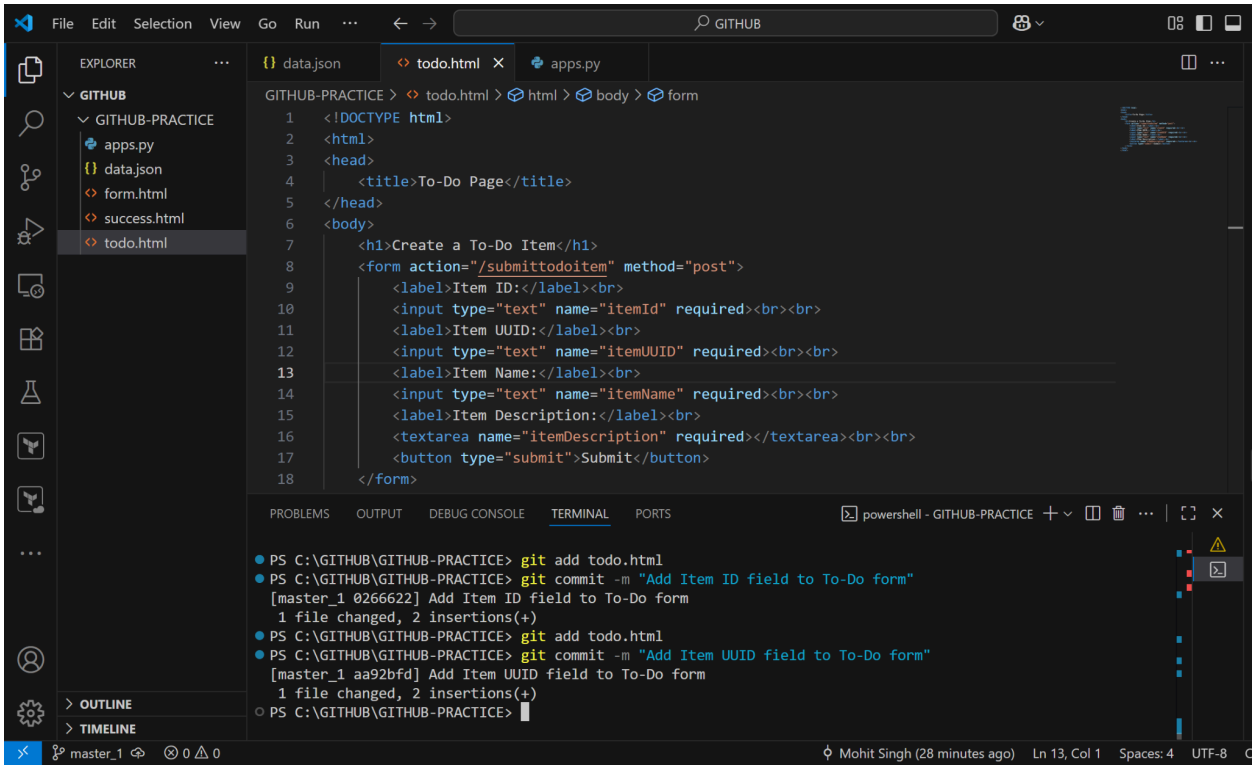
```
<label>Item UUID:</label><br>
```

```
<input type="text" name="itemUUID" required><br><br>
```

Then commit:

```
git add todo.html
```

```
git commit -m "Add Item UUID field to To-Do form"
```



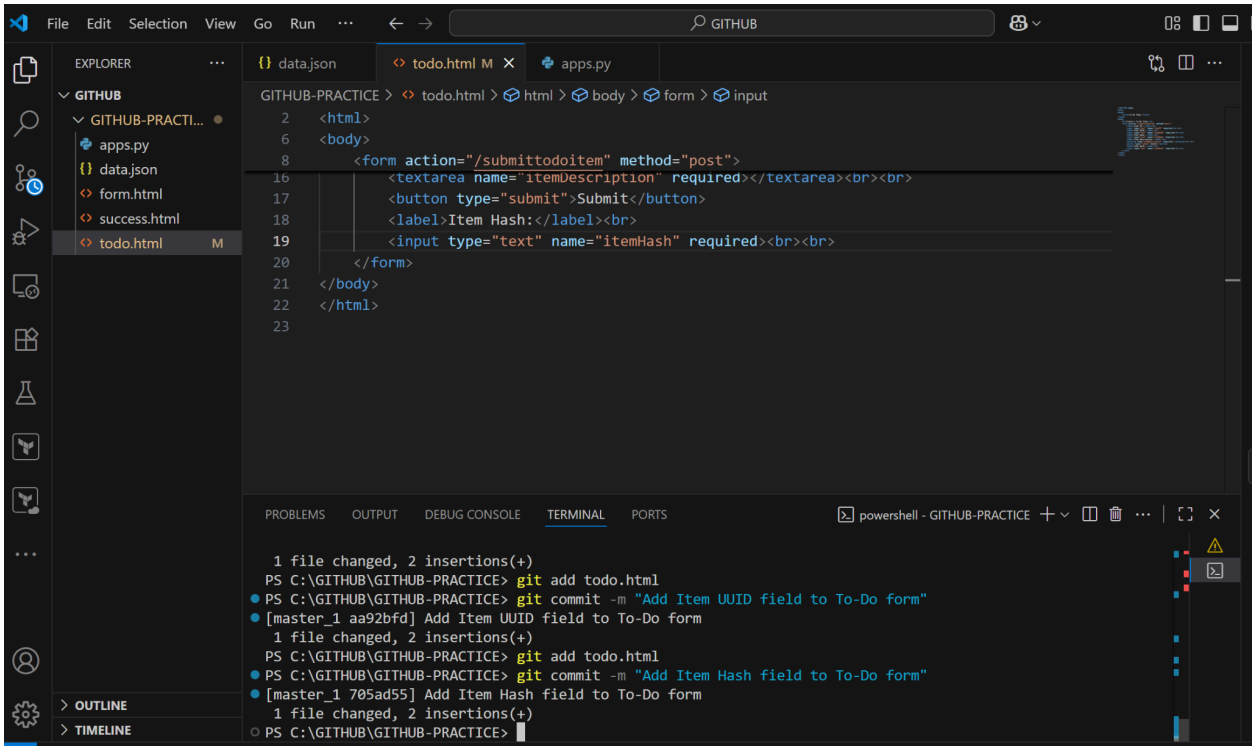
🔸 **Third Commit: Add Item Hash Field**

Add to `todo.html`:

```
<label>Item Hash:</label><br>
<input type="text" name="itemHash" required><br><br>
```

Then commit:

```
git add todo.html
git commit -m "Add Item Hash field to To-Do form"
```



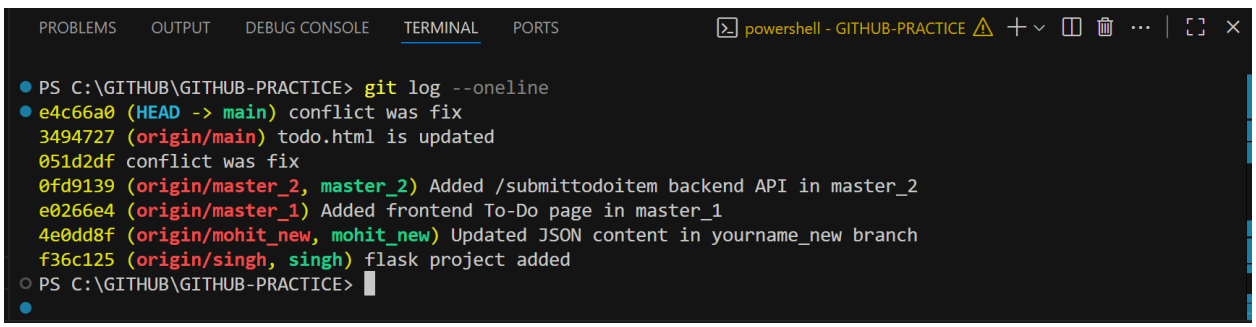
🔹 Step 2: Merge master_1 to main

```
git checkout main

git merge master_1

git push origin main
```

🔹 Step 3: Git Reset in main to Keep Only First Commit

1. View commit history:

```
git log --oneline
```

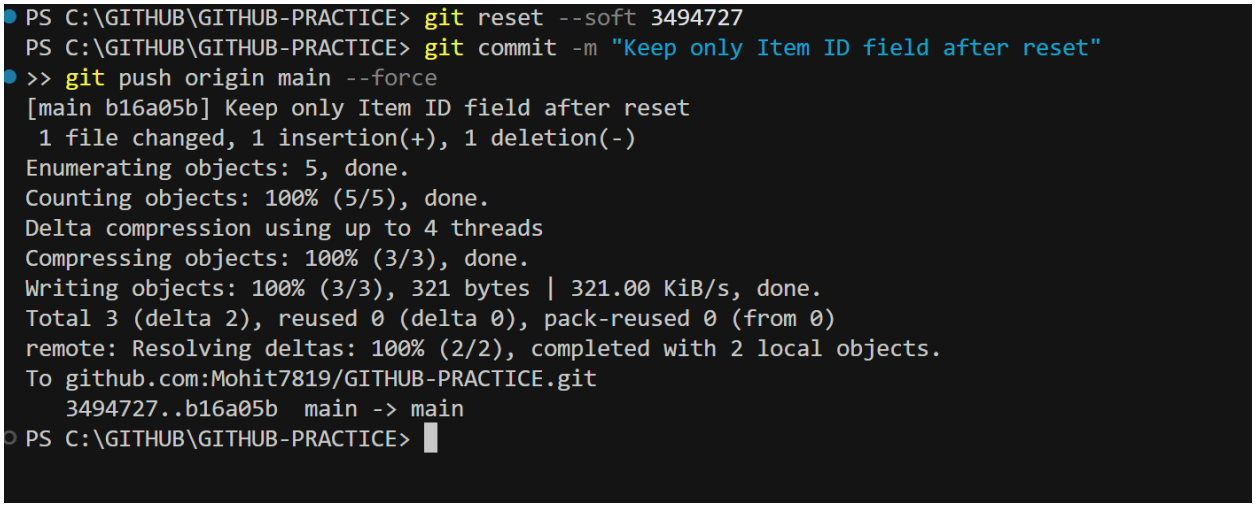


2. Reset to that commit (keep changes staged):

```
git reset --soft 3494727
```

3. Recommit only the Item ID state:

```
git commit -m "Keep only Item ID field after reset"

git push origin main --force
```



🔹 Step 4: Rebase `main` into `master_1` (Preserve Commit History)

Switch to `master_1`:

```
git checkout master_1
```

Then rebase:

```
git rebase main
```

💡 During rebase:

- You may see interactive options in the editor.
- Just **keep all commits as-is** to preserve the history.
- If conflicts arise, resolve them, then continue:

```
PS C:\GITHUB\GITHUB-PRACTICE> git checkout master_1
Switched to branch 'master_1'
PS C:\GITHUB\GITHUB-PRACTICE> git branch
  main
* master_1
  master_2
  mohit_new
  singh
PS C:\GITHUB\GITHUB-PRACTICE> git rebase main
CONFLICT (content): Merge conflict in todo.html
error: could not apply 0266622... Add Item ID field to To-Do form
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
hint: Disable this message with "git config set advice.mergeConflict false"
Could not apply 0266622... # Add Item ID field to To-Do form
PS C:\GITHUB\GITHUB-PRACTICE> git rebase main
Current branch master_1 is up to date.
PS C:\GITHUB\GITHUB-PRACTICE>
```

`git add .`

```
git rebase --continue
```