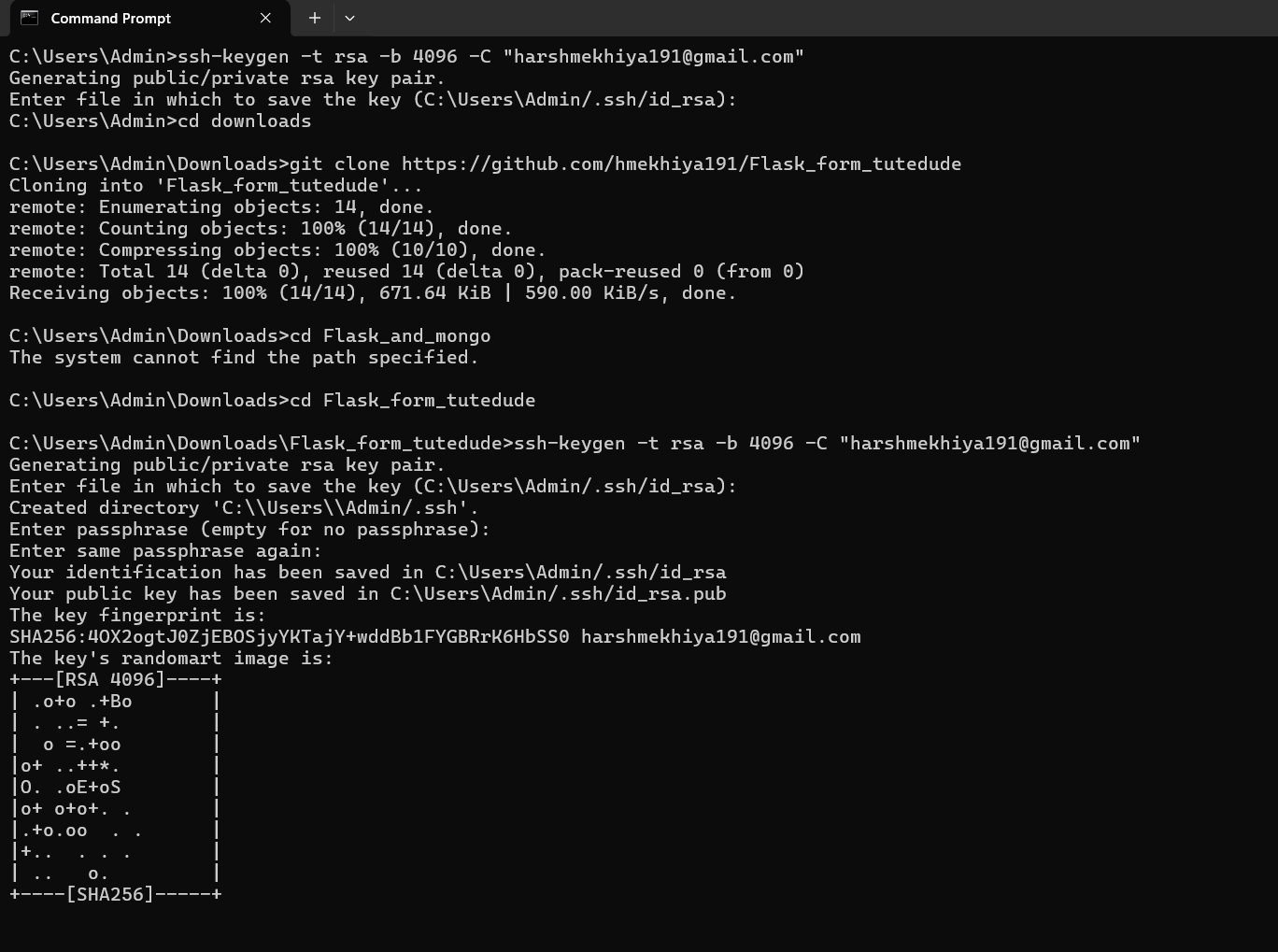
**Assignment: 4 Git\_Hub**

Name: Harsh M. Mekhiya   
email: [harshmekhiya191@gmail.com](mailto:harshmekhiya191@gmail.com)

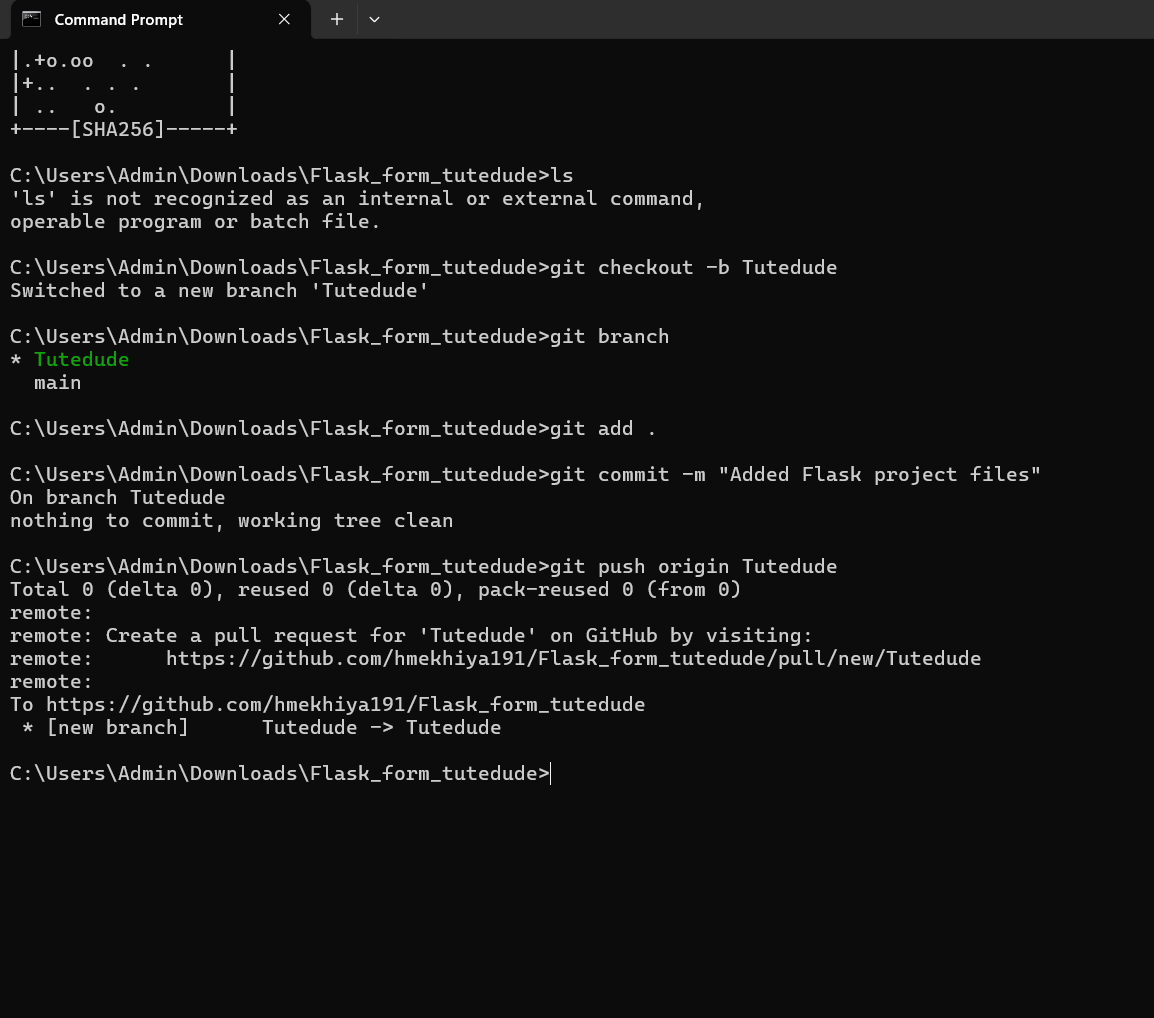
**Tasks-1**

**1. Create a new GitHub repository.**

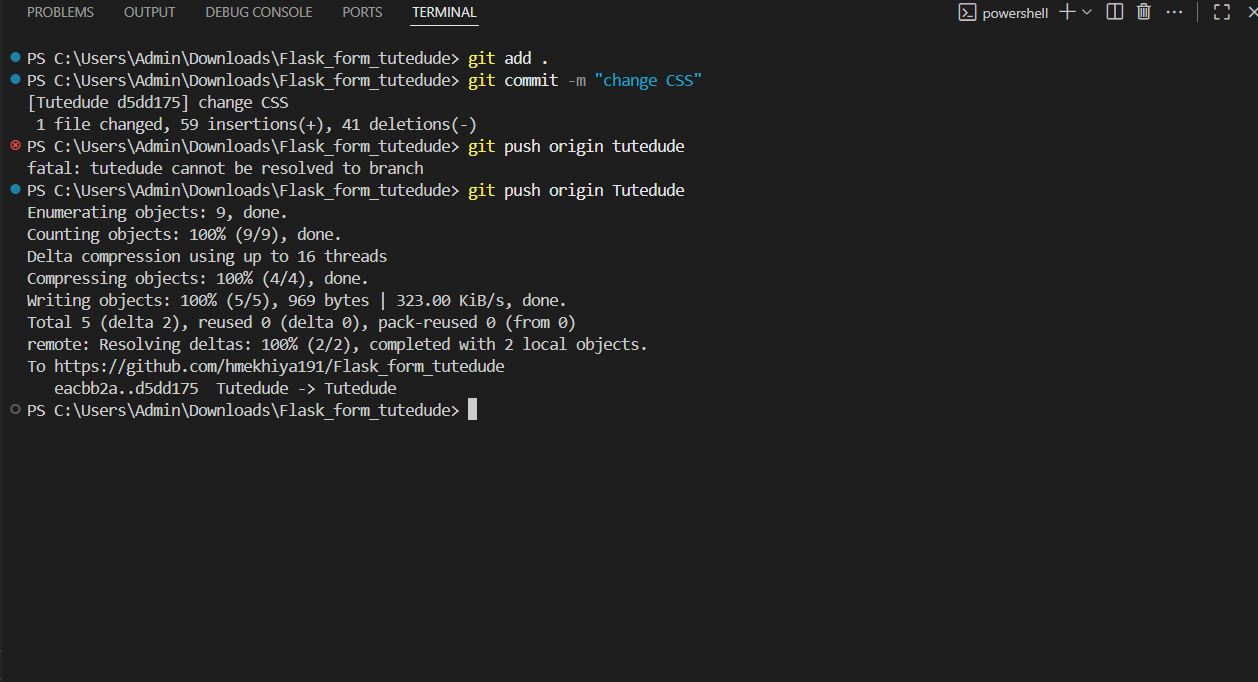
* **Clone the repository to your local machine using SSH (generate an SSH key if needed, add the public key to your GitHub account).**

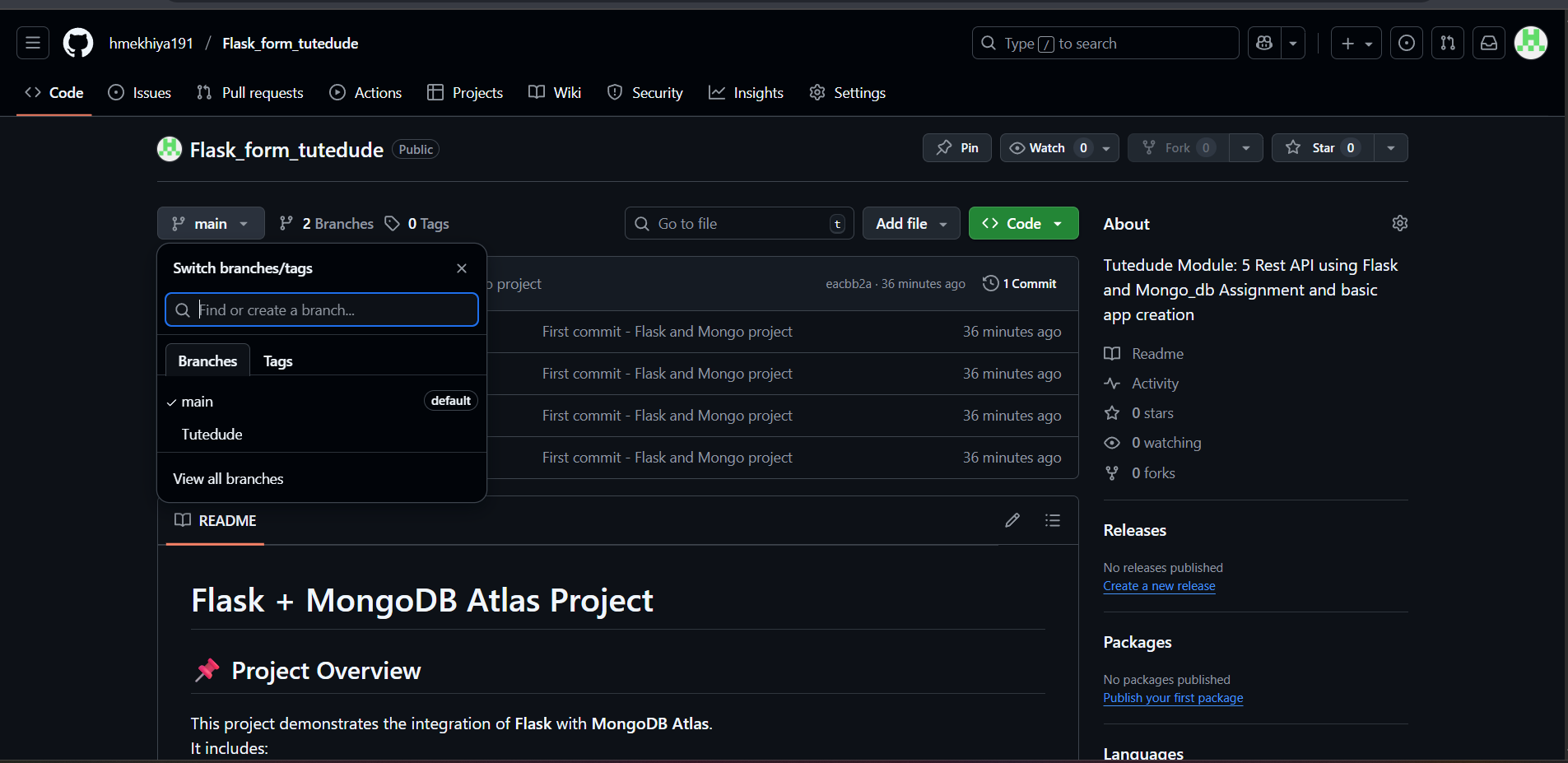


* **Create a new branch named after your username (e.g., Tutedude).**

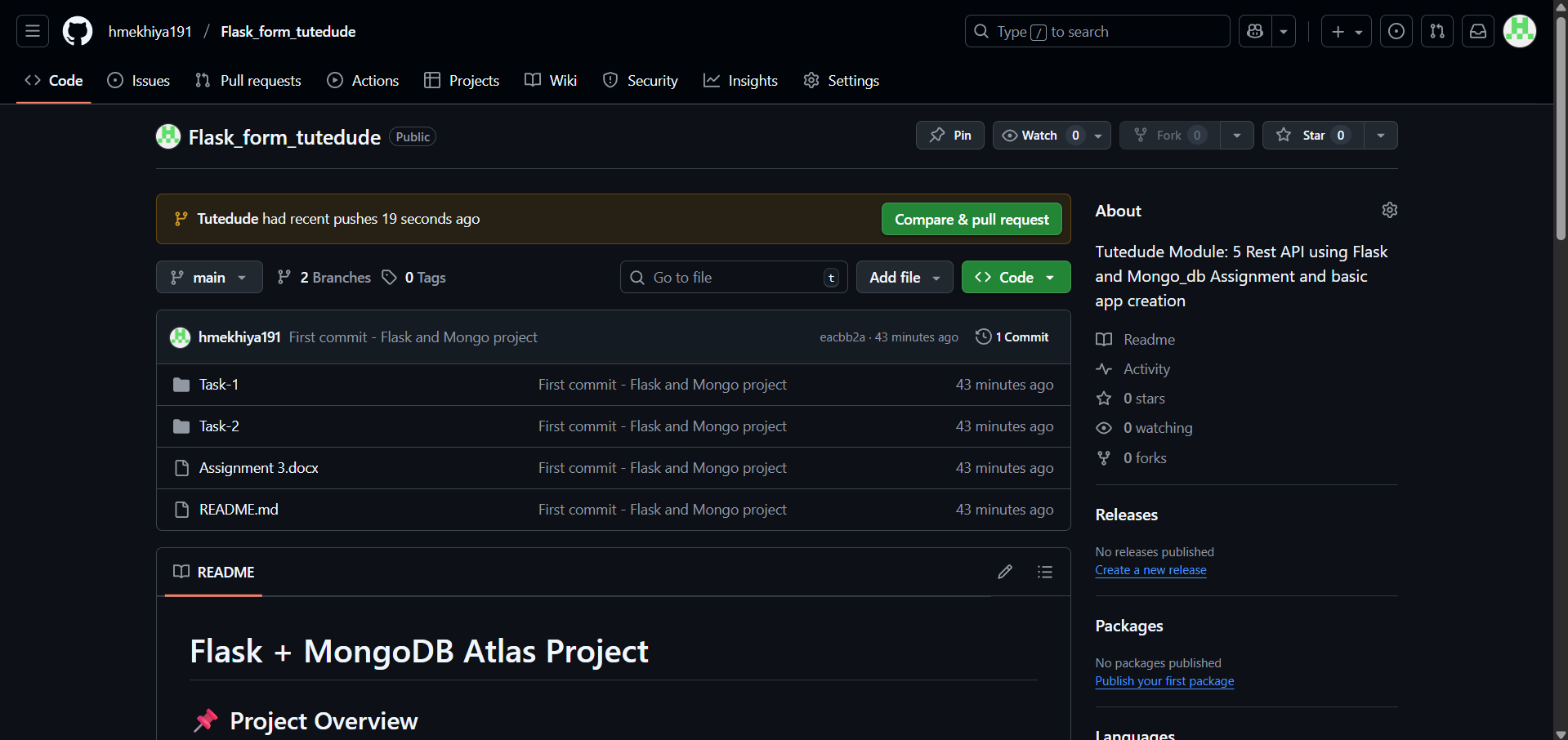


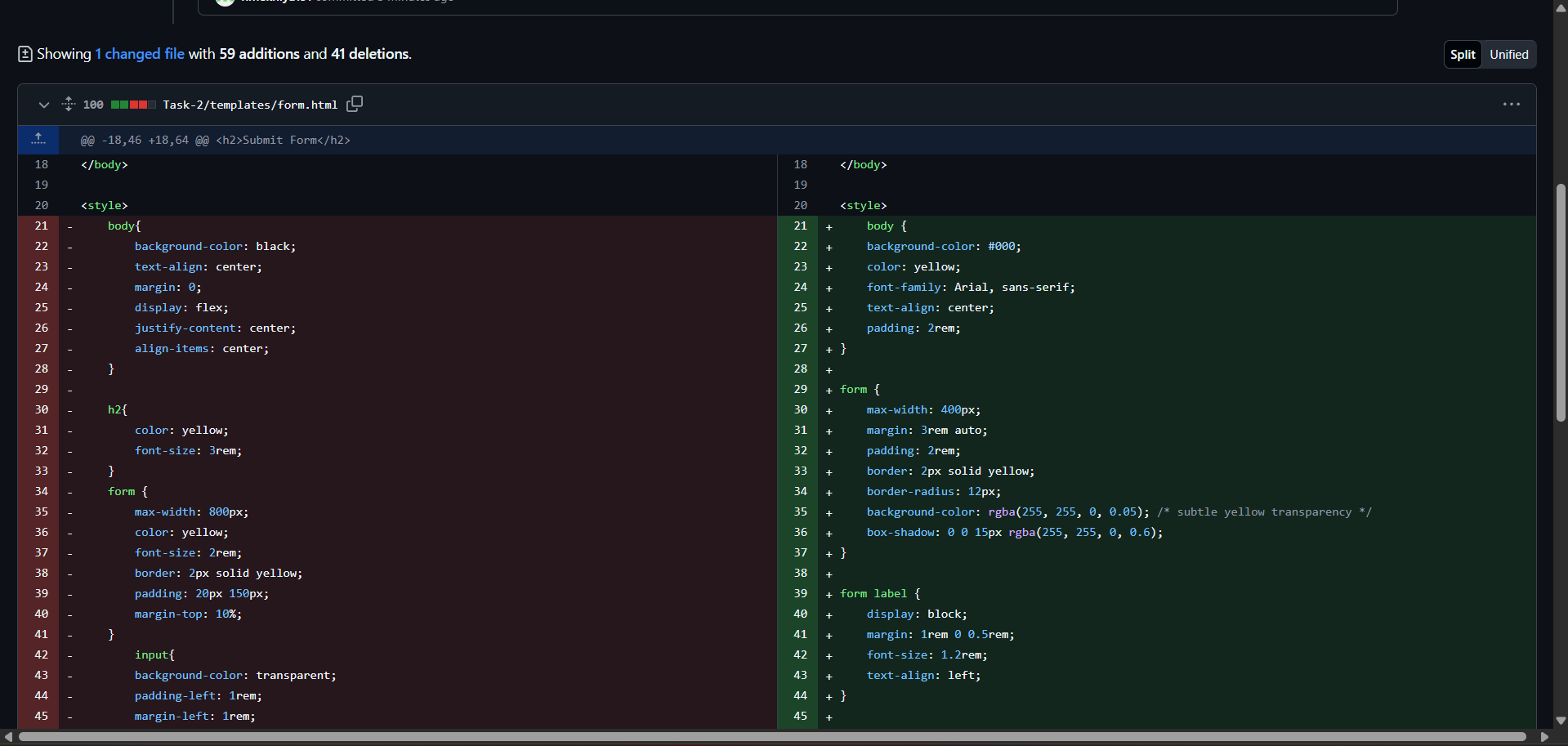
* **Add your Flask project files to this branch.**





* **Commit the changes and merge the branch into the main branch.**

****

****

****

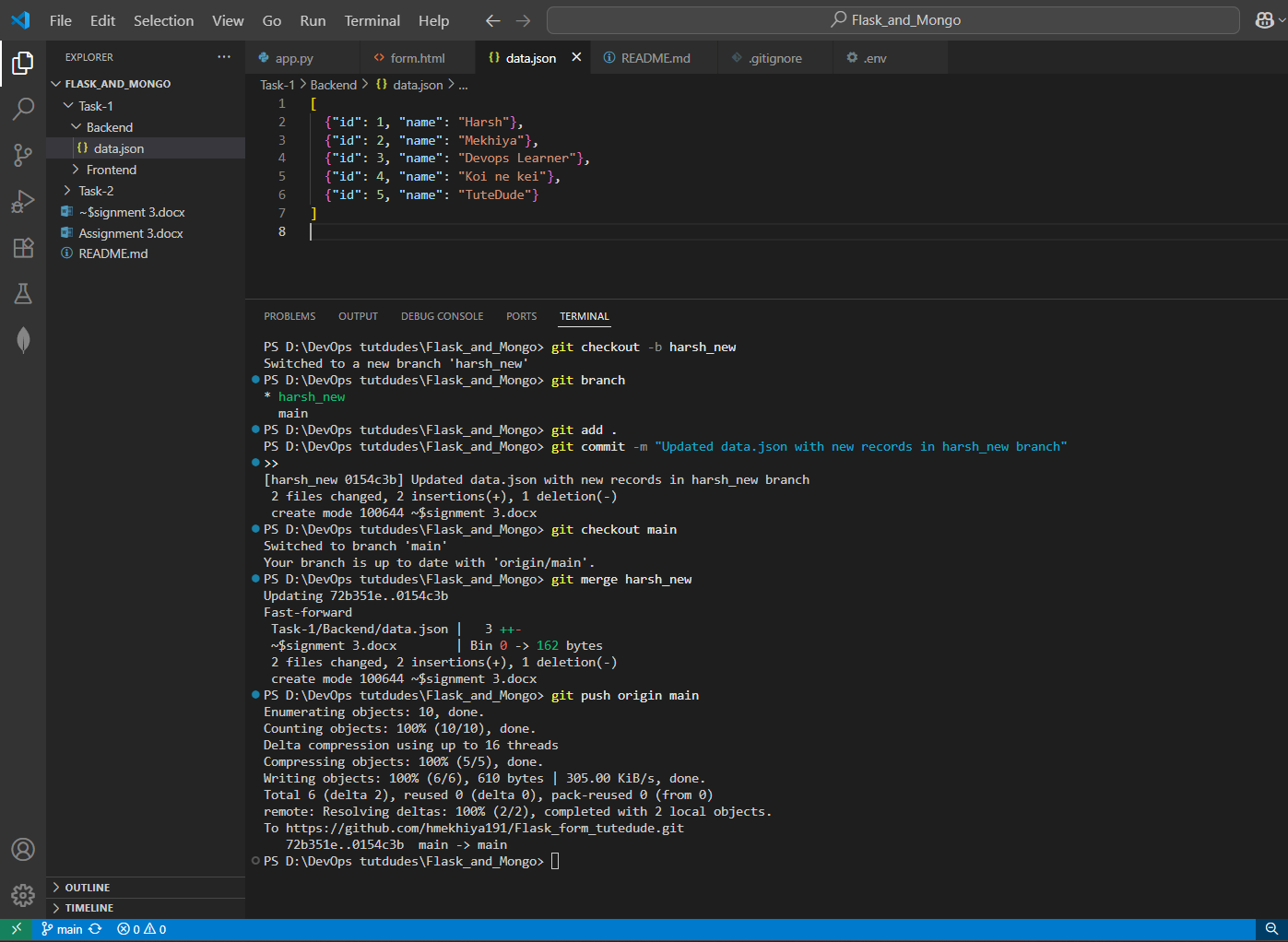
**Explanation:**

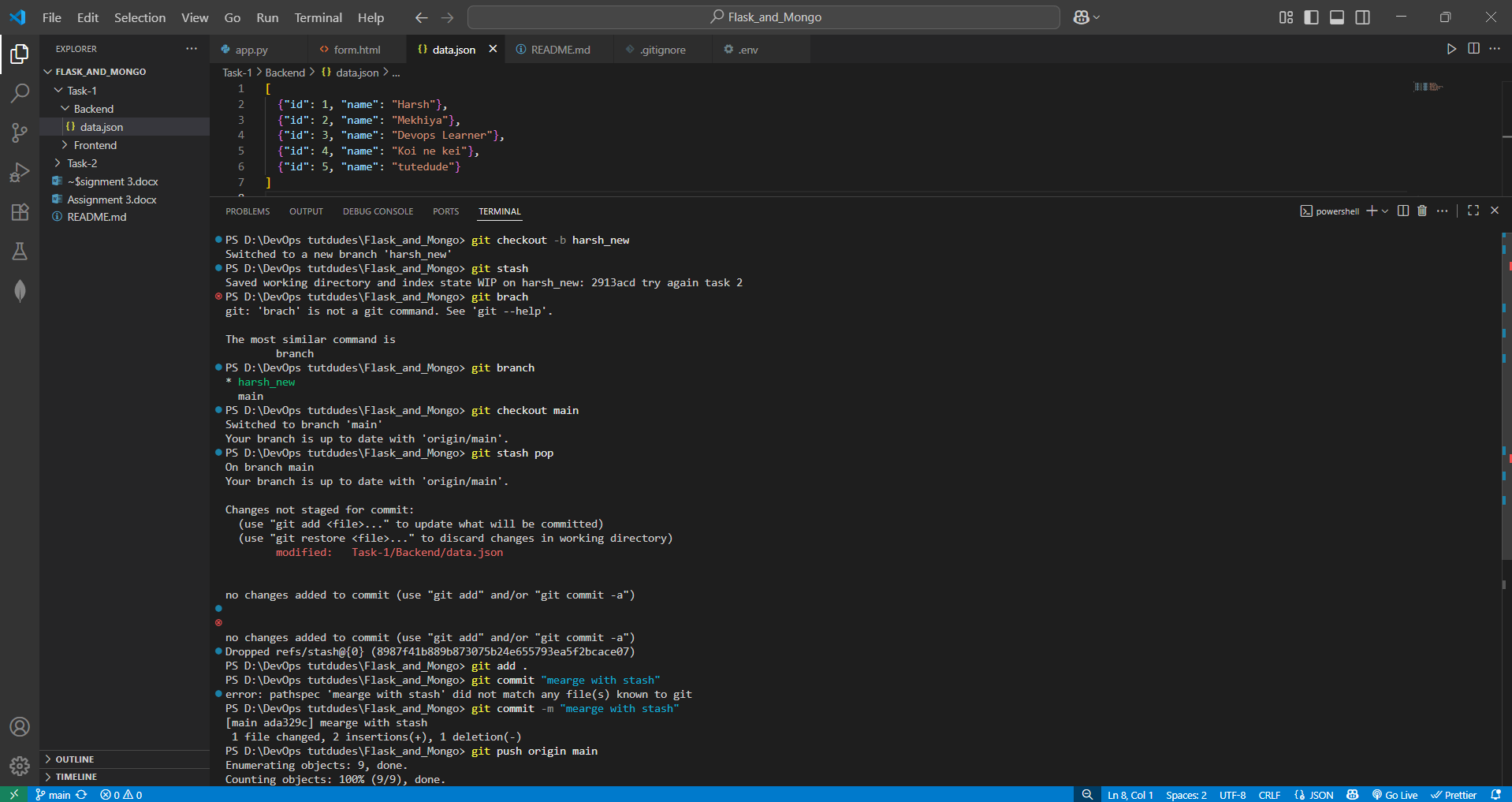
I created a new GitHub repository named **Flask\_form\_tutedude** and set up SSH authentication for secure communication. For this, I generated an SSH key on my local system, added the public key to my GitHub account, and verified the connection. Then, I cloned the repository to my local machine using the SSH URL and created a new branch named **Tutedude** to keep my changes organized. After copying my Flask project files into the repository folder, I staged and committed them, and then pushed the branch to GitHub. Finally, I created a pull request on GitHub and merged the **Tutedude** branch into the **main** branch, making my Flask project available in the main repository. This process helped me understand how to securely connect GitHub with SSH, manage branches, and merge changes into the main project.

**Tasks- 2**

**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.**

****

****

**Explanation:**

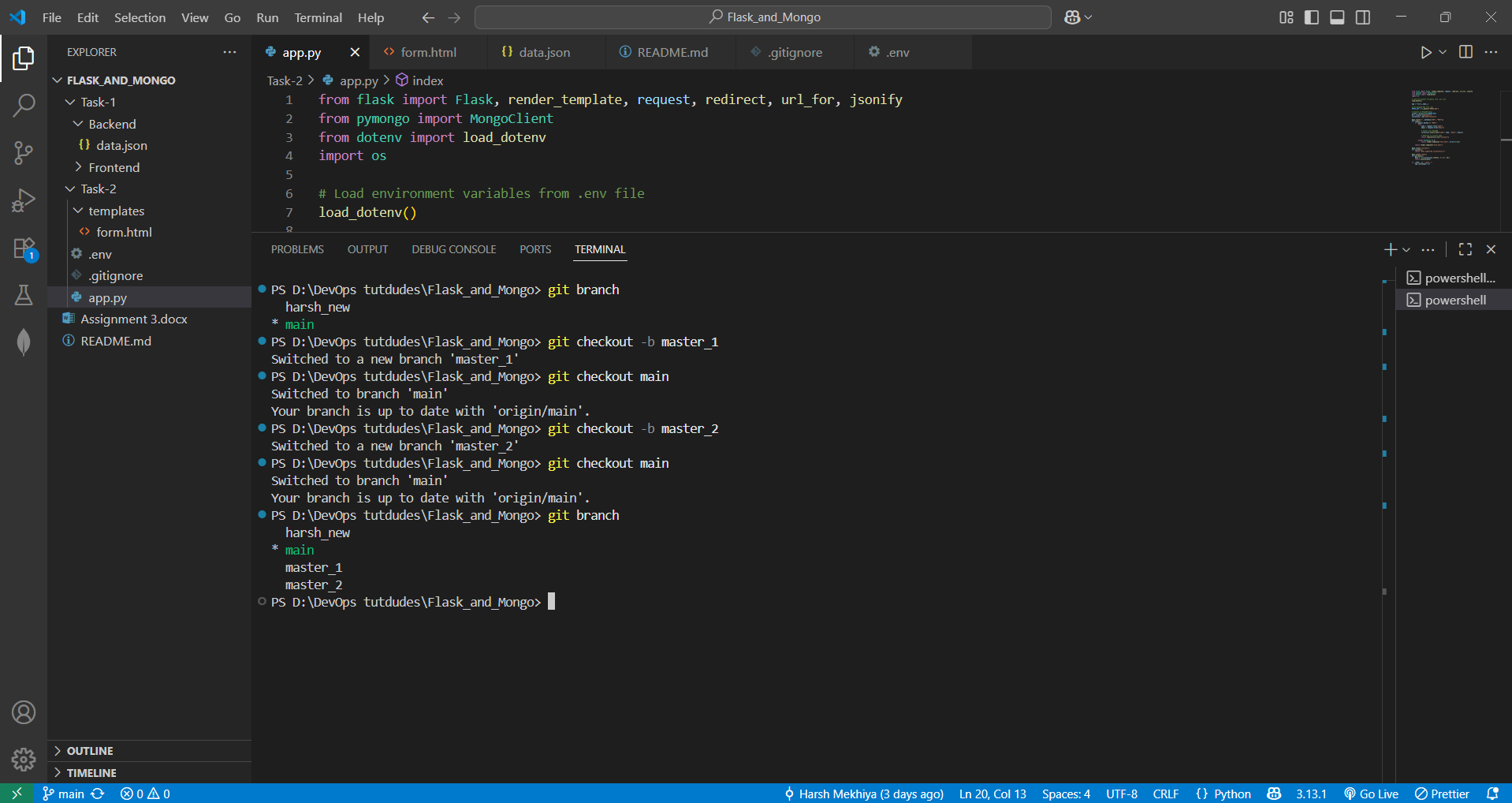
In this task, I created a new branch named <yourname>\_new from the main branch. Inside this branch, I updated the data.json file with new content. After committing the changes, I switched back to the main branch and merged the harsh\_new branch. If there were any conflicts, they would be resolved by editing the file and committing the changes again. Finally, I pushed the updated main branch to GitHub so that the changes are reflected in the remote repository.

Also In this exercise, I learned how to use **git stash** to temporarily save changes without committing them. First, I made modifications in the data.json file on a branch, then used git stash to store them safely. After that, I switched to the main branch without any errors. Later, I restored the stashed changes using git stash pop, staged them, and committed to the repository. This process is useful when I want to switch branches but keep my ongoing work without committing it immediately.

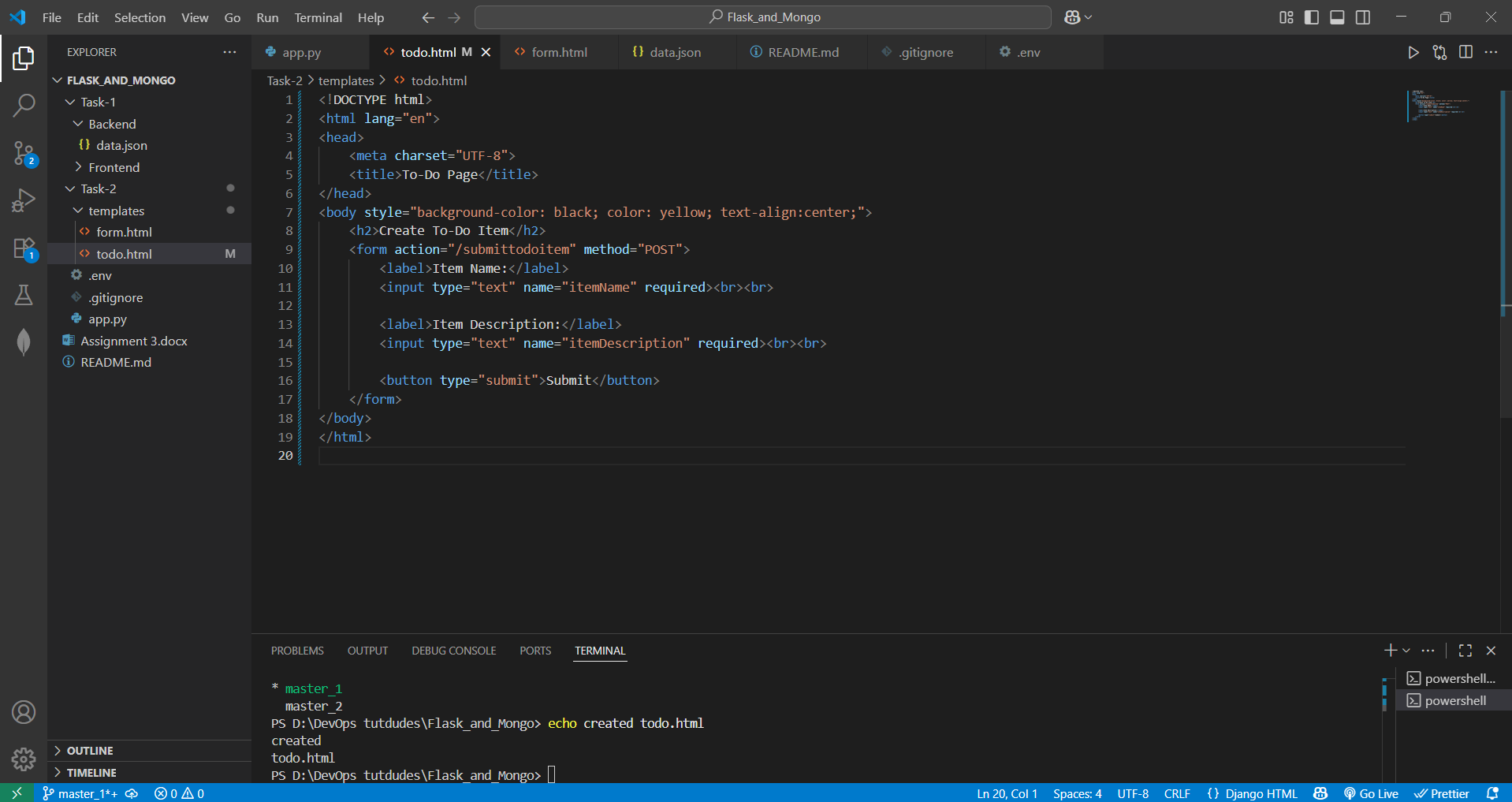
**Task:**

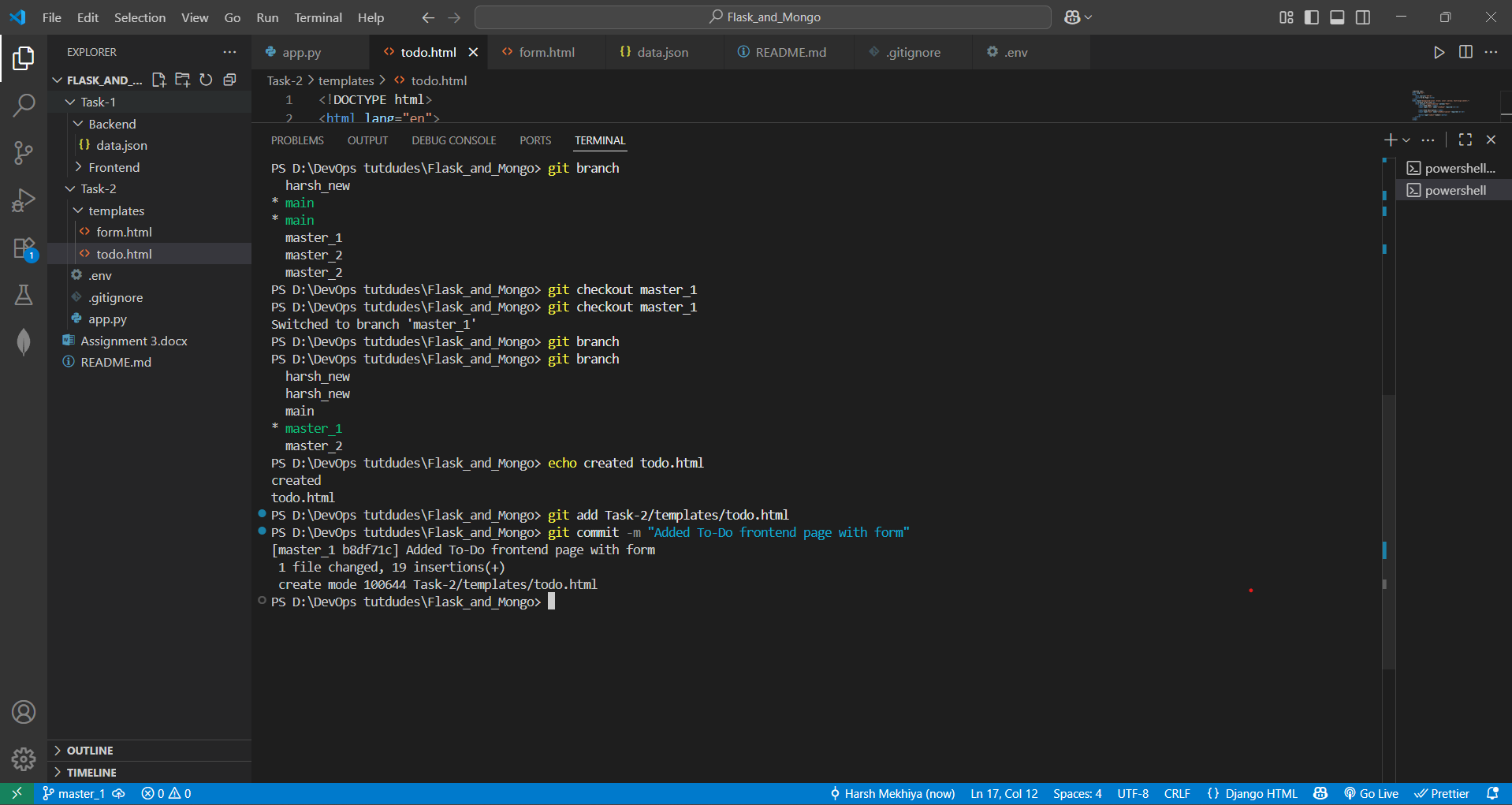
**3. Branch Creation:**

* **Create two branches: master\_1 and master\_2 from the main branch.**

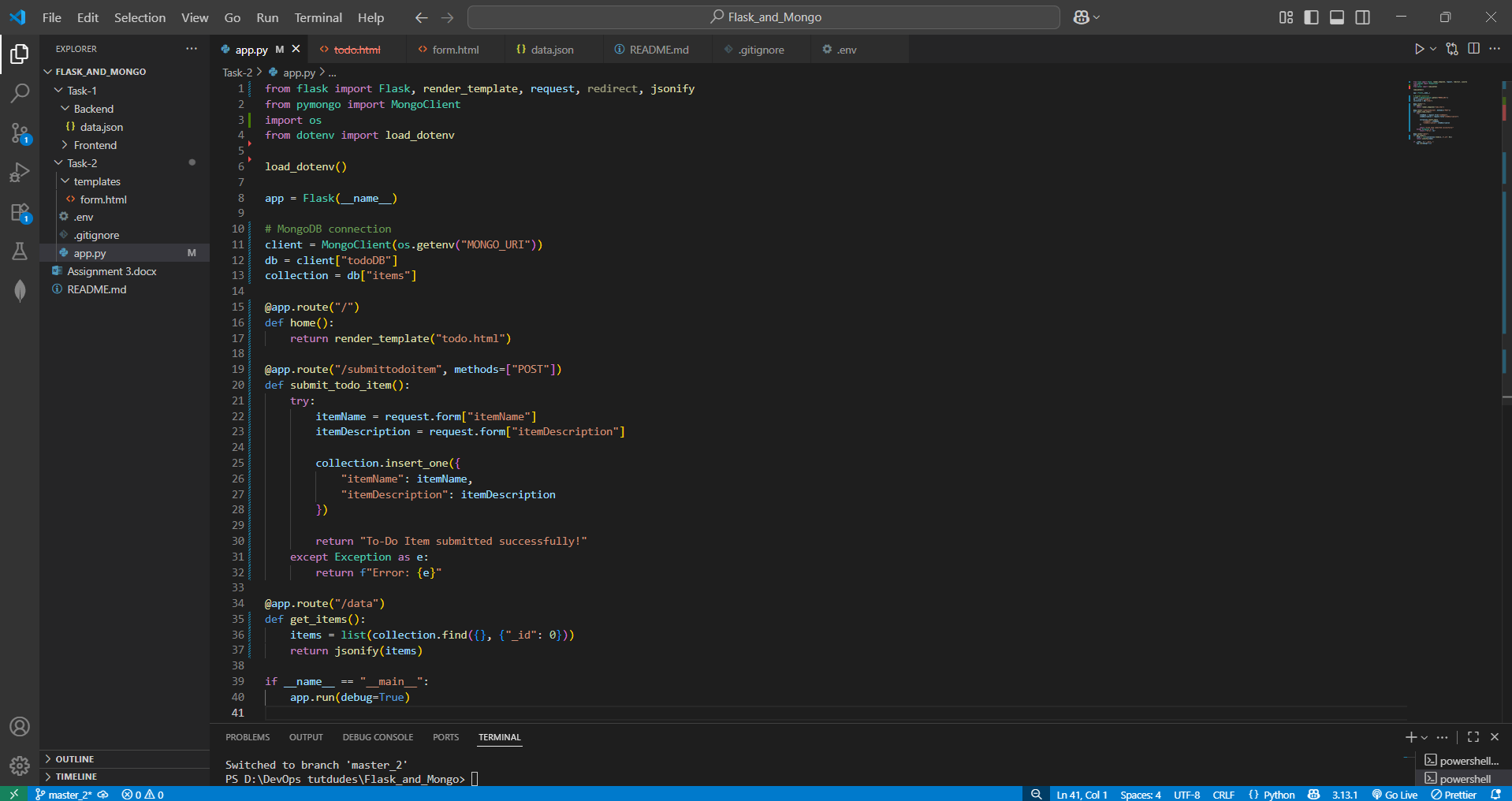
****

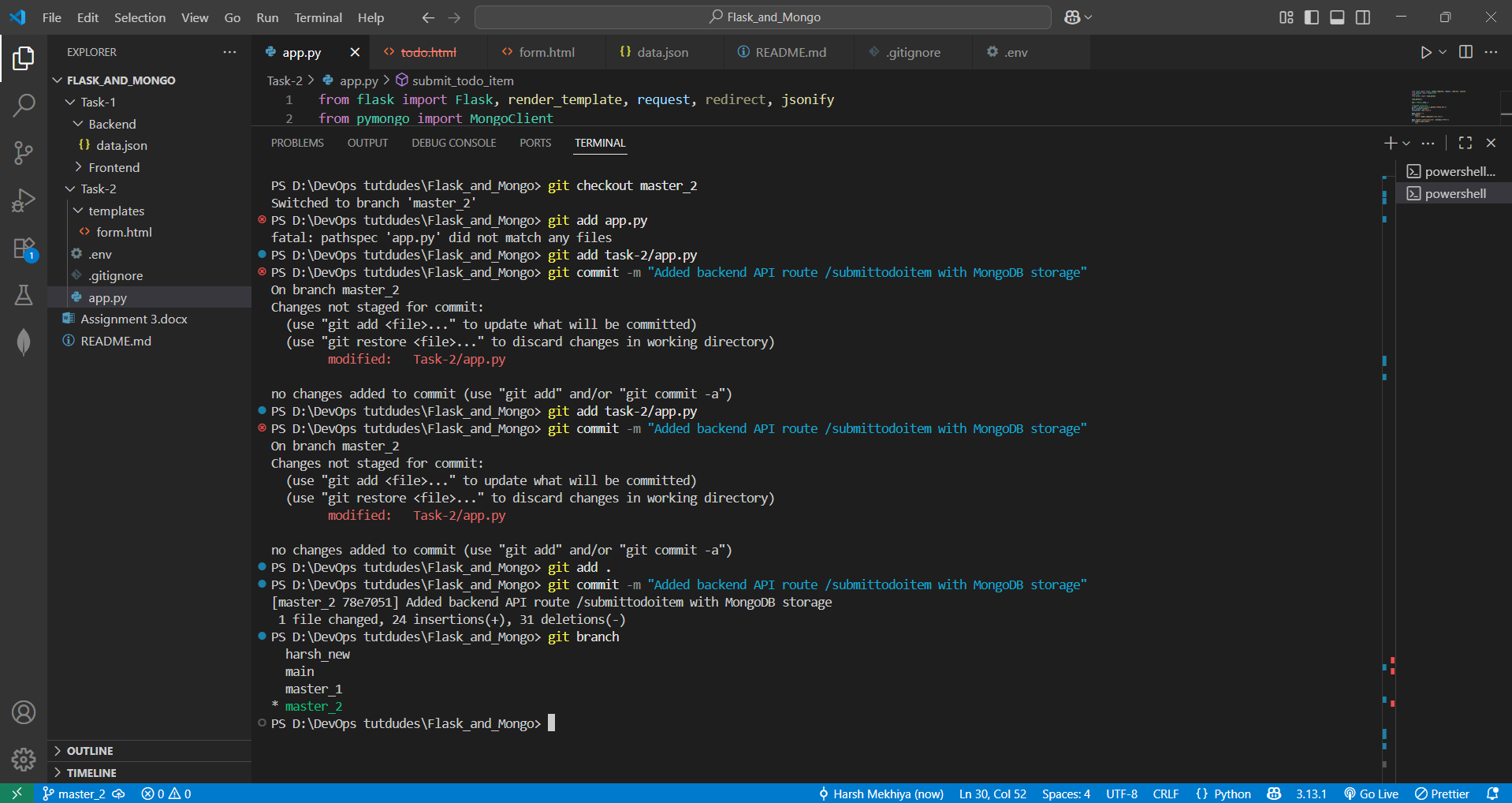
* **Feature Development in master\_1:**
* **In the master\_1 branch, create a To-Do Page in the frontend.**
  + **The page should contain a form with the following fields:**
    - **Item Name**
    - **Item Description**

****

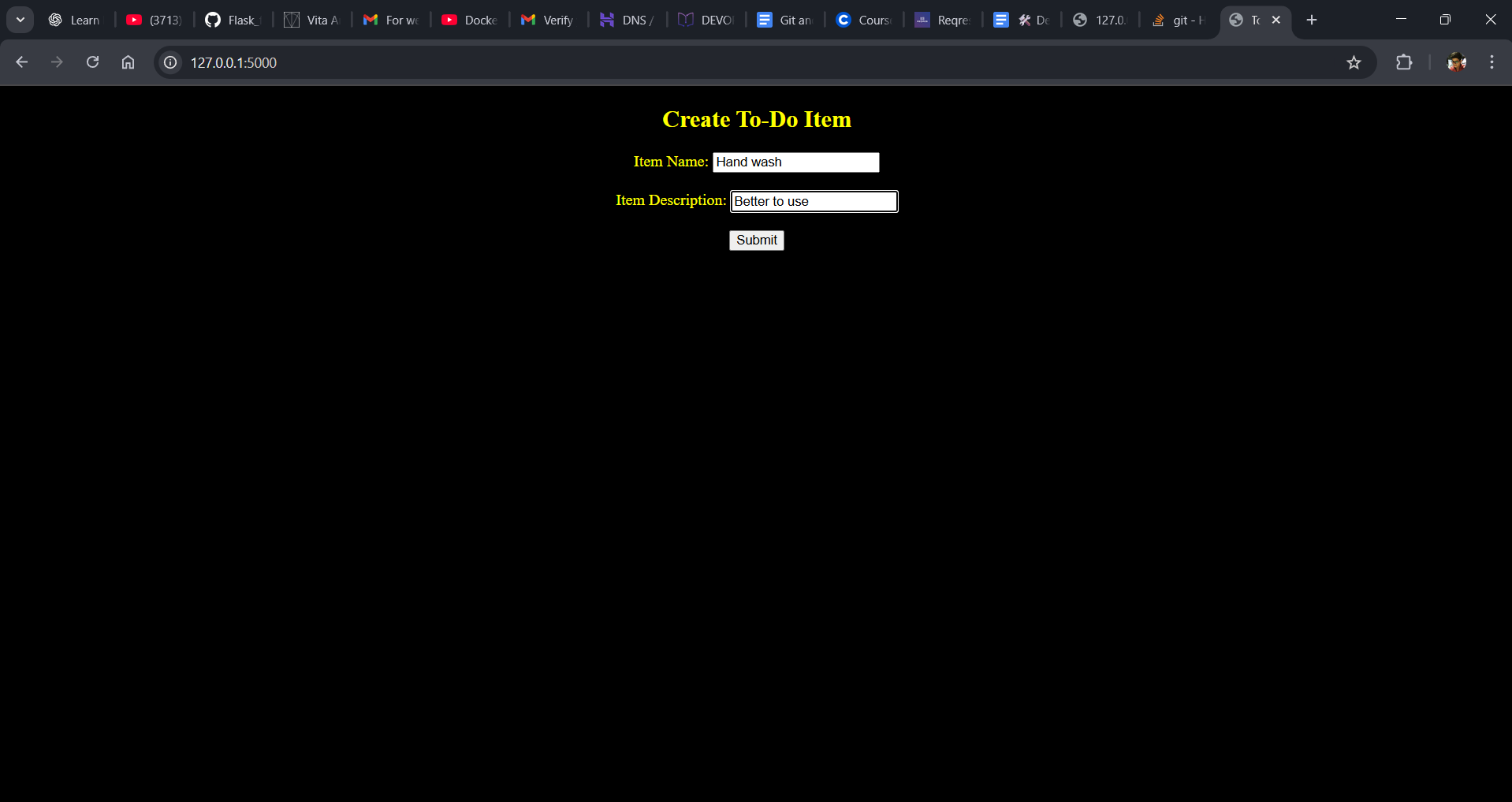
****

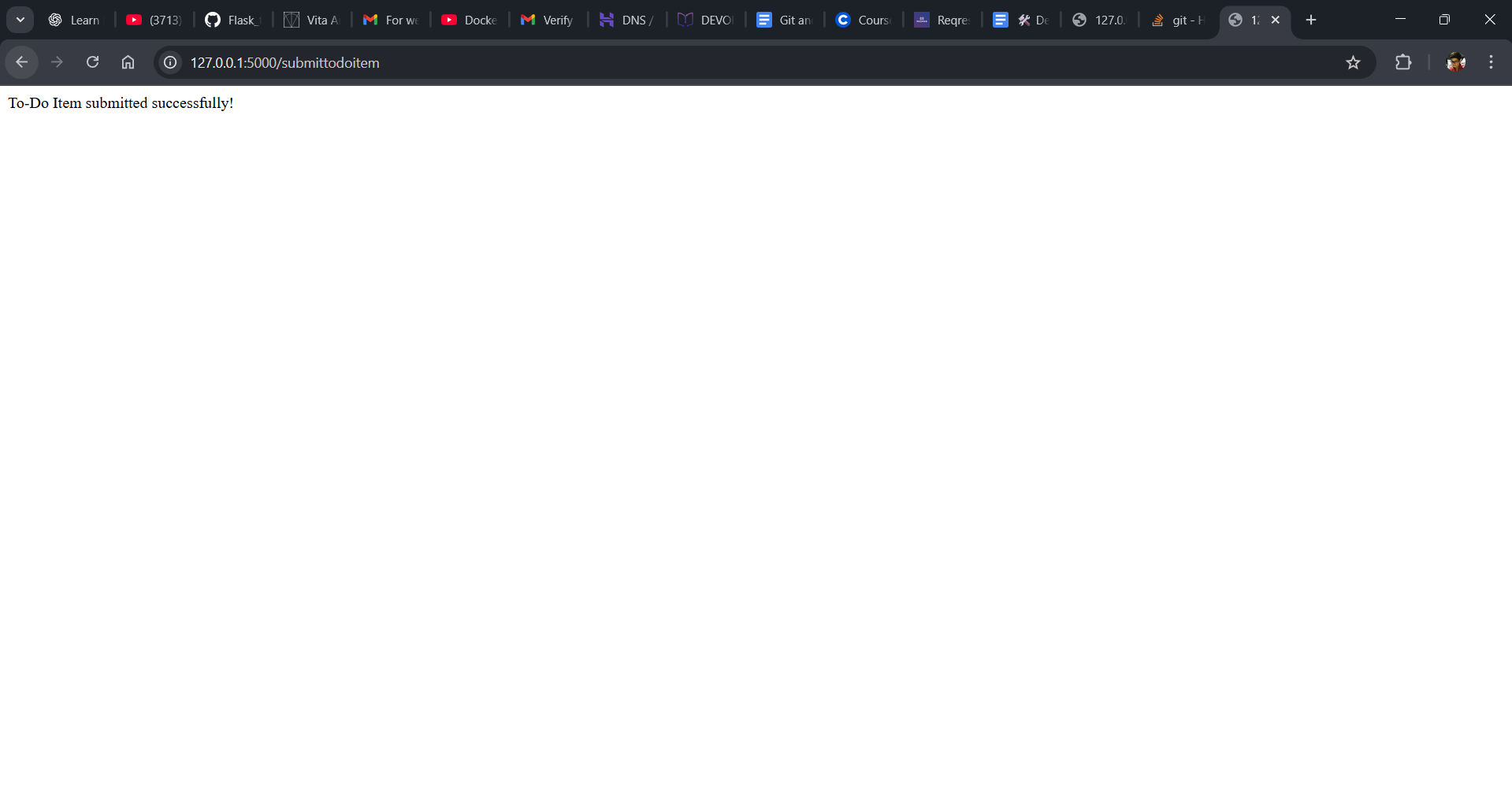
* **Backend API in master\_2:**
* **In the master\_2 branch, create a backend route named /submittodoitem.**

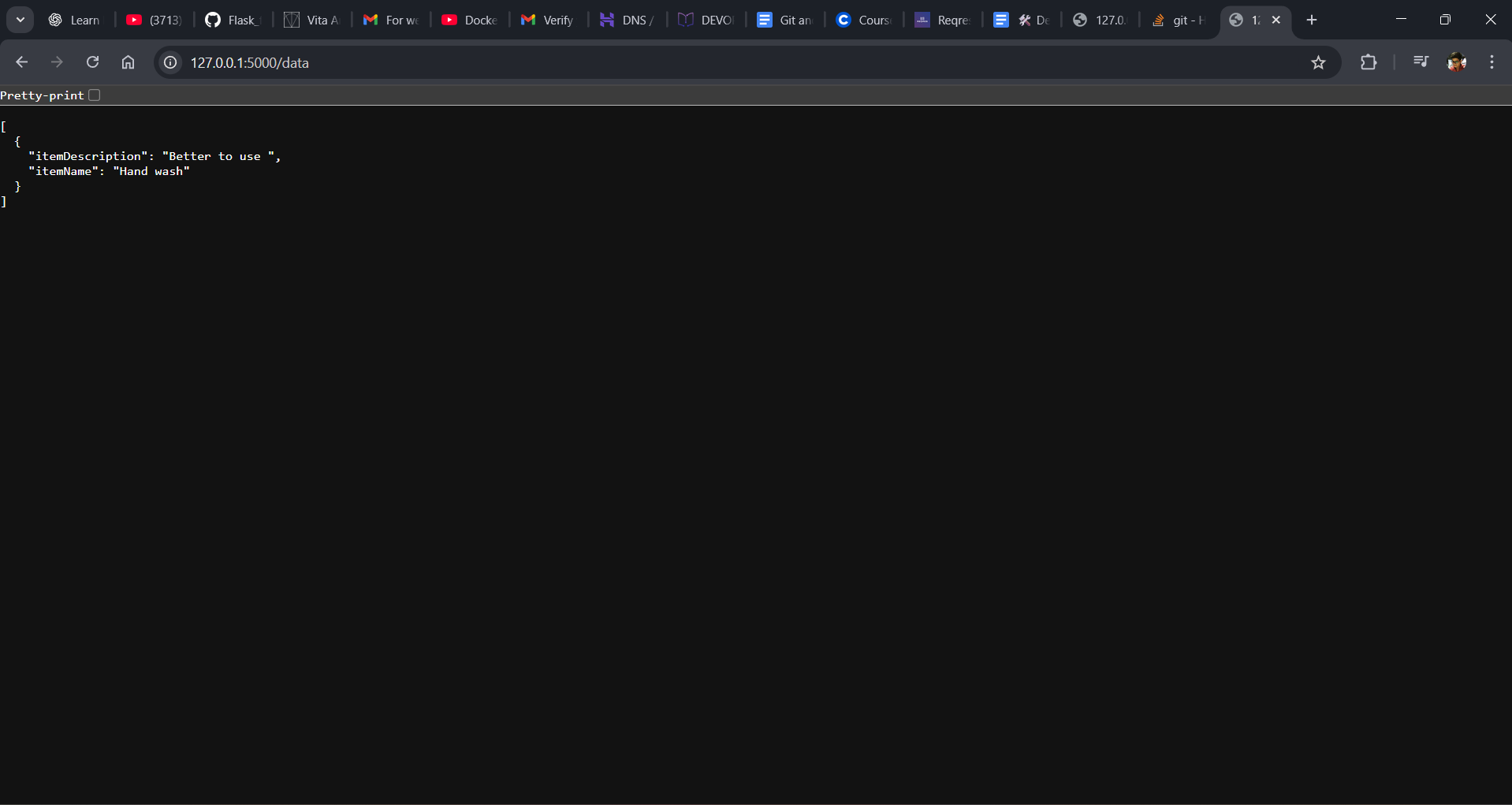
****

****

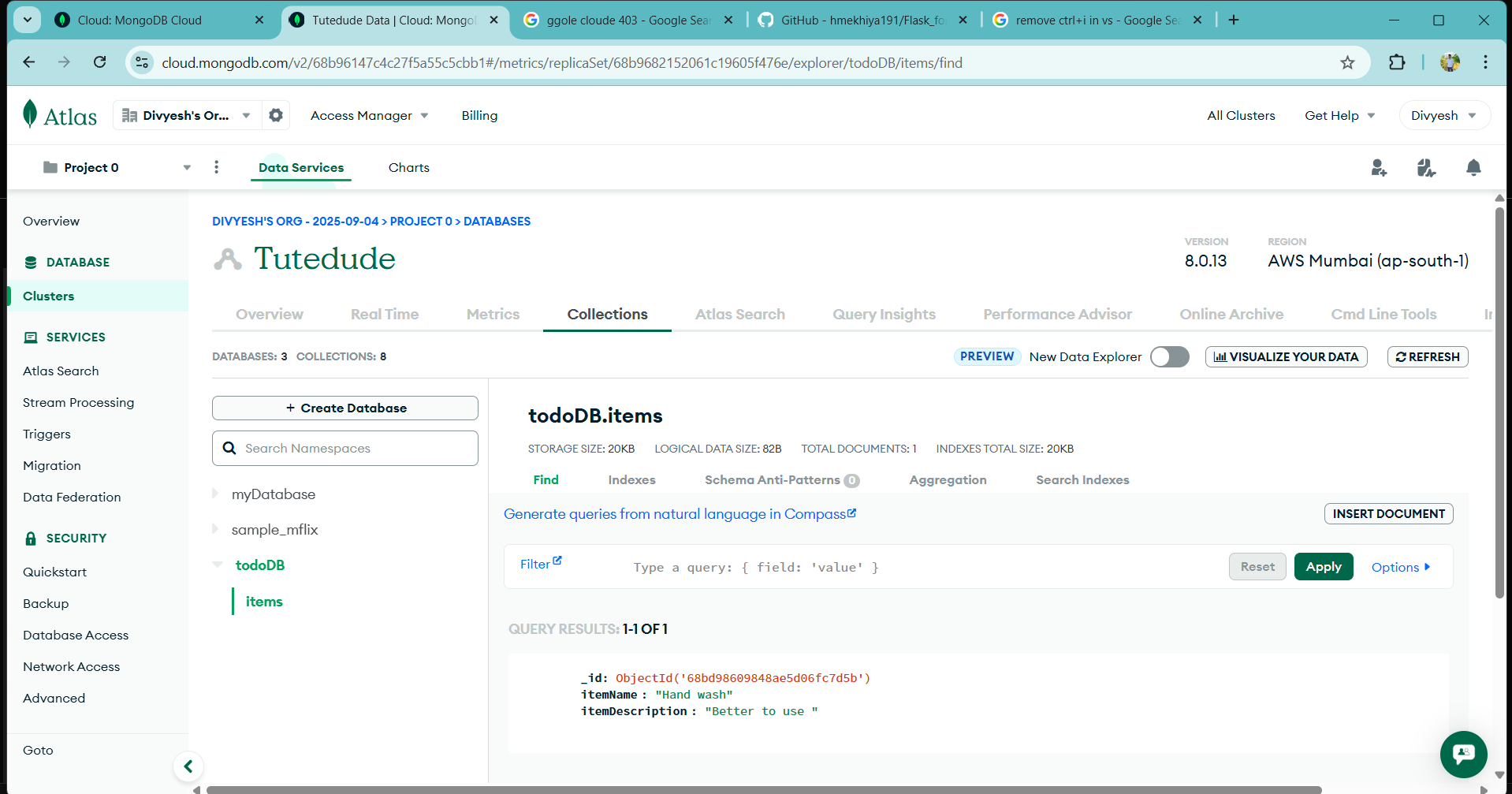
* **This route will:**
  + **Accept itemName and itemDescription via a POST request.**
  + **Store these details in a MongoDB database.**

****

**Click on submit and re directs to this page **

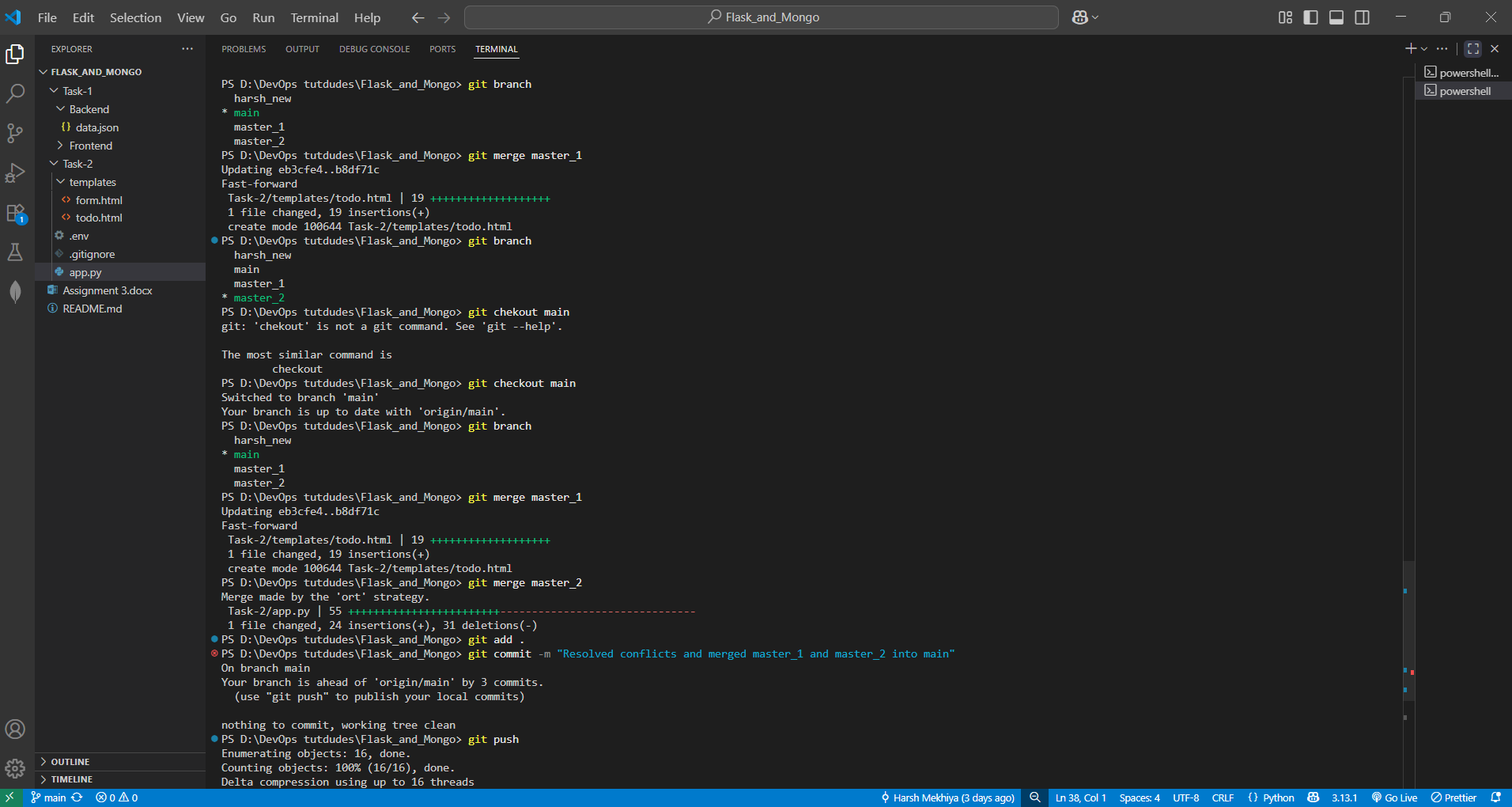
****

Data is stored in mongodb and shown with **“localhost/data”** Json file



* **Merging Changes:**

**Merge the changes from both master\_1 and master\_2 into the main branch.**

****

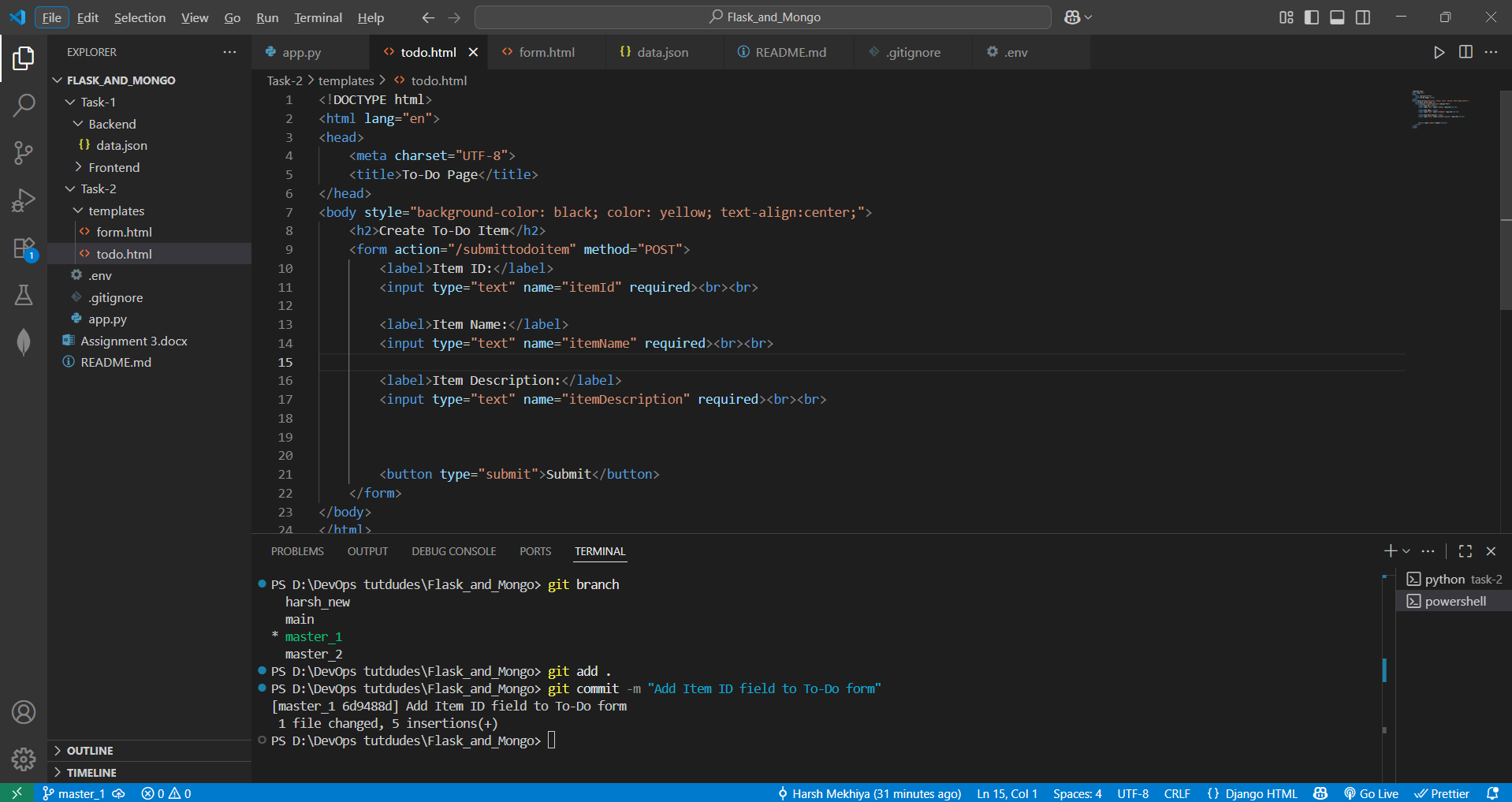
**Explanation:**

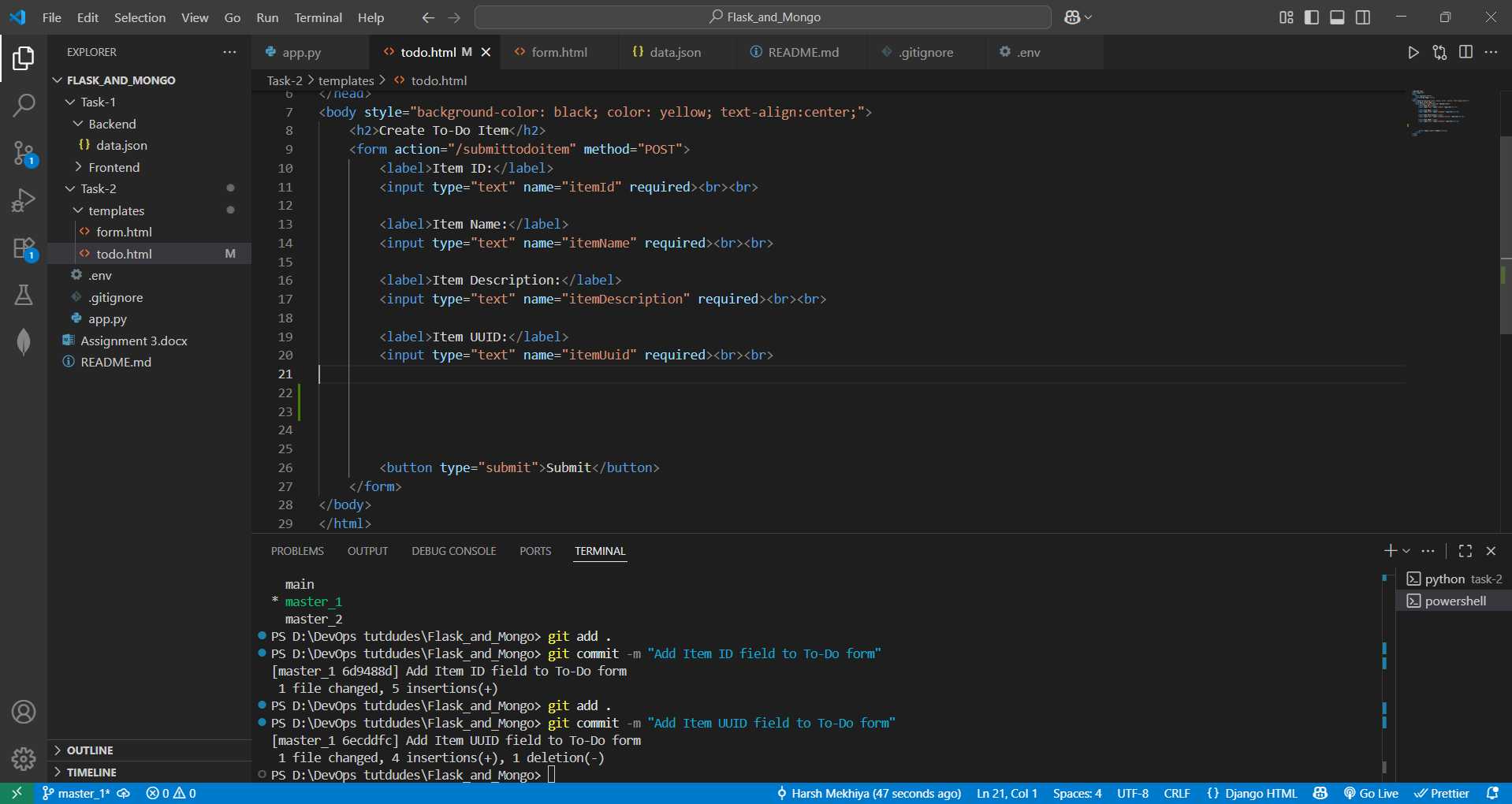
In this task, I first created two new branches from the main branch: master\_1 for frontend development and master\_2 for backend development. In the master\_1 branch, I worked on building a simple To-Do page on the frontend that contains a form with fields for entering an item name and an item description. This form acts as the user interface for submitting tasks. Next, I switched to the master\_2 branch, where I created a Flask backend route called /submittodoitem. This route accepts POST requests, retrieves the item name and item description from the form submission, and stores the data in a MongoDB database. Once both branches were completed, I returned to the main branch and merged the changes from master\_1 and master\_2 so that the main branch now contains both the frontend To-Do page and the backend API functionality. During merging, if any conflicts occurred, they needed to be resolved by reviewing the differences and keeping the required changes before committing the final merged version. This exercise helped me practice managing frontend and backend development separately using branches, and then combining them effectively into the main branch.

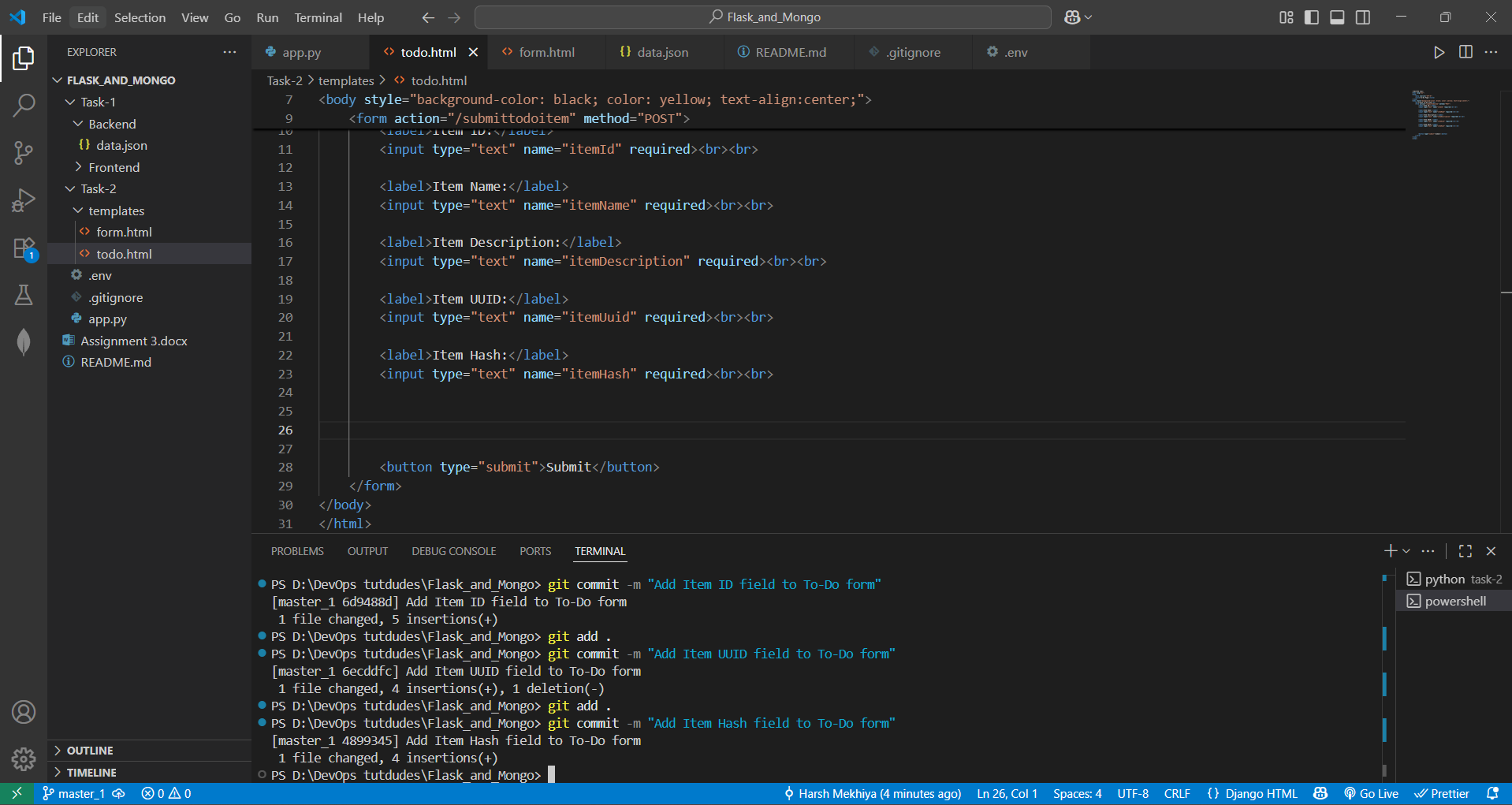
**Task:**

**4. Enhancing the To-Do Form in master\_1:**

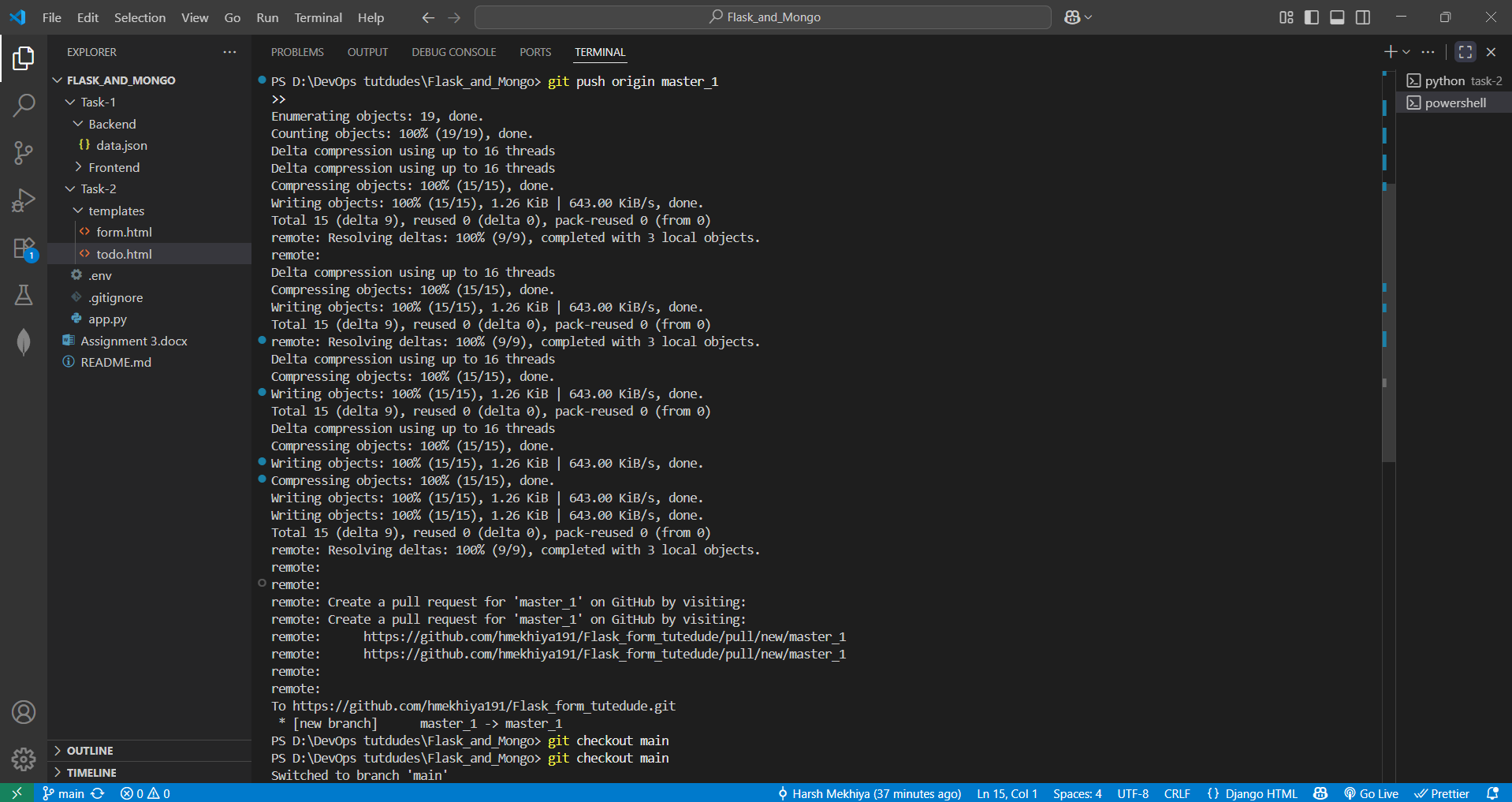
* **In the master\_1 branch, add the following fields to the To-Do form:**
  + **Item ID**
  + **Item UUID**
  + **Item Hash**
* **Committing in Sequence:**
* **Add and commit each field separately in the following order:**
  + **First commit: Add Item ID field.**
  + **Second commit: Add Item UUID field.**
  + **Third commit: Add Item Hash field.**

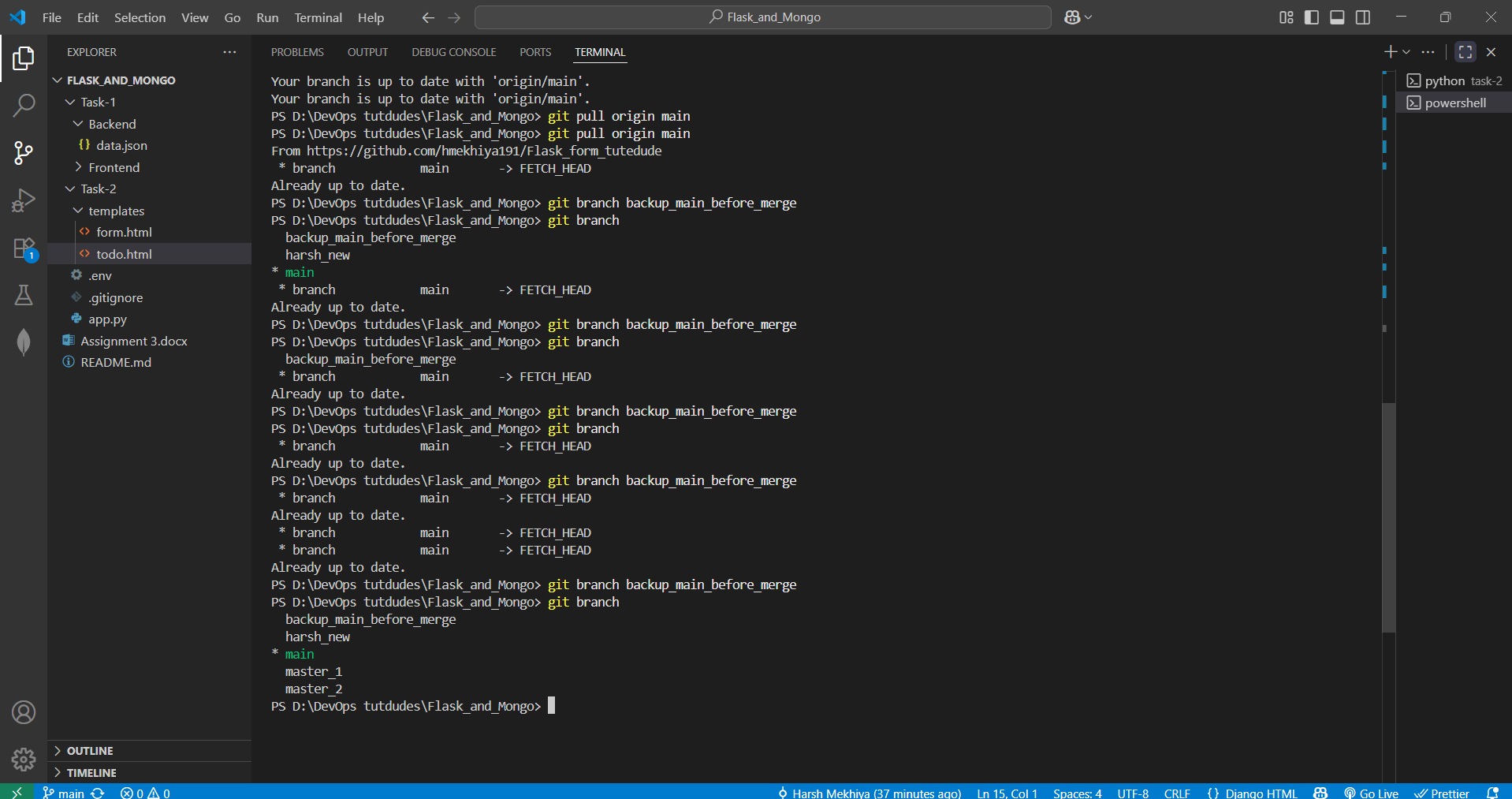
****

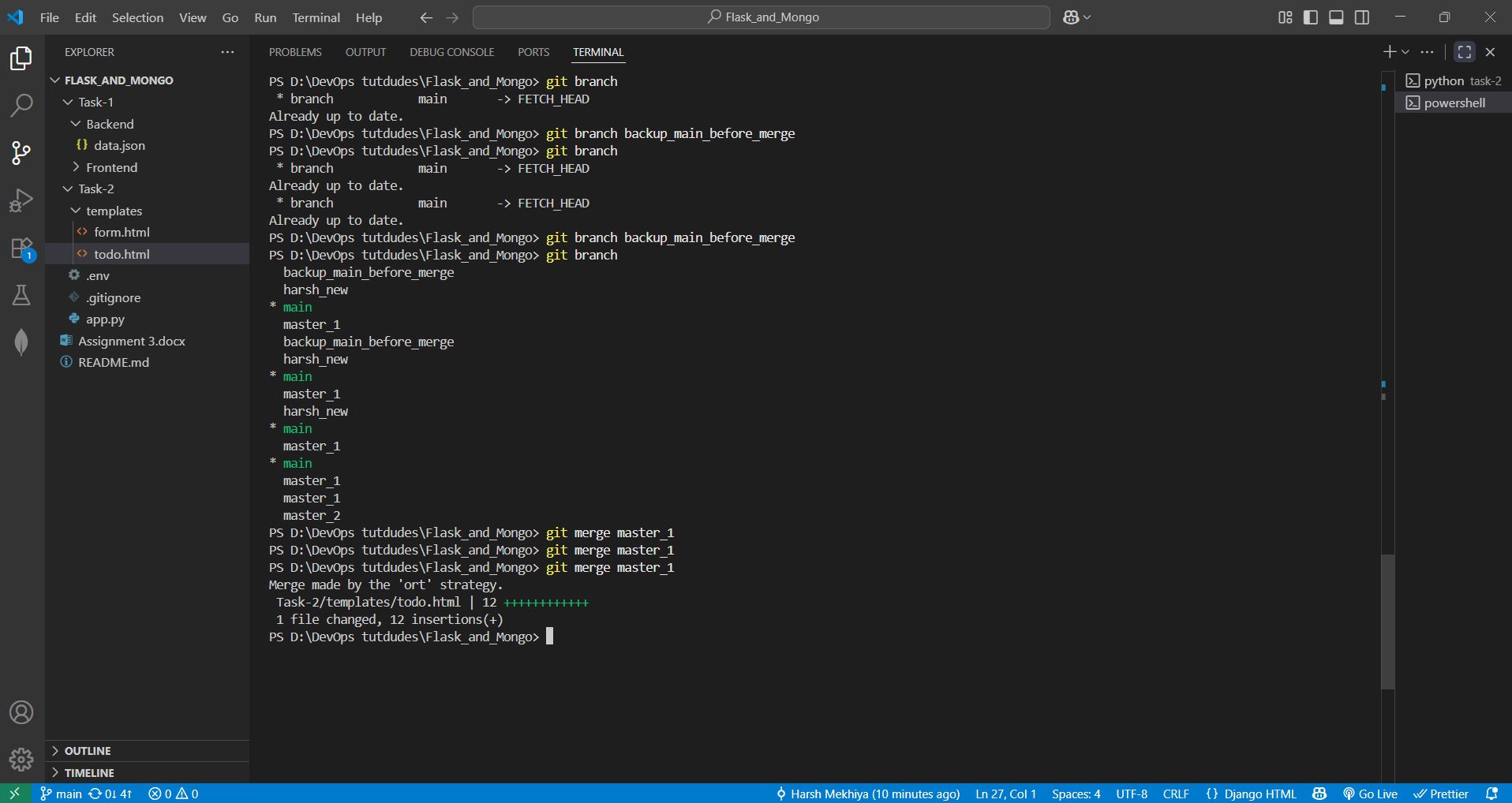
****

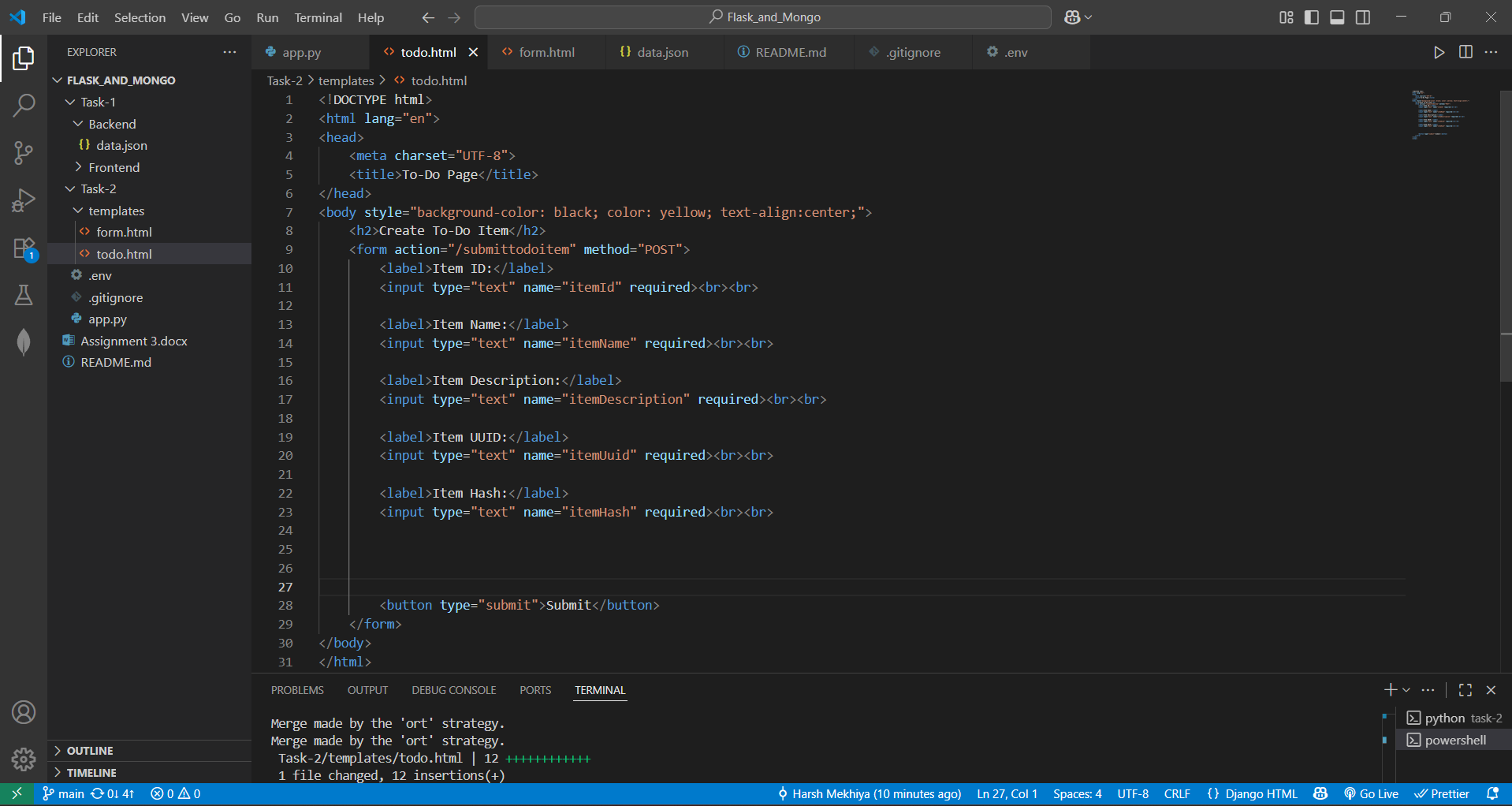
****

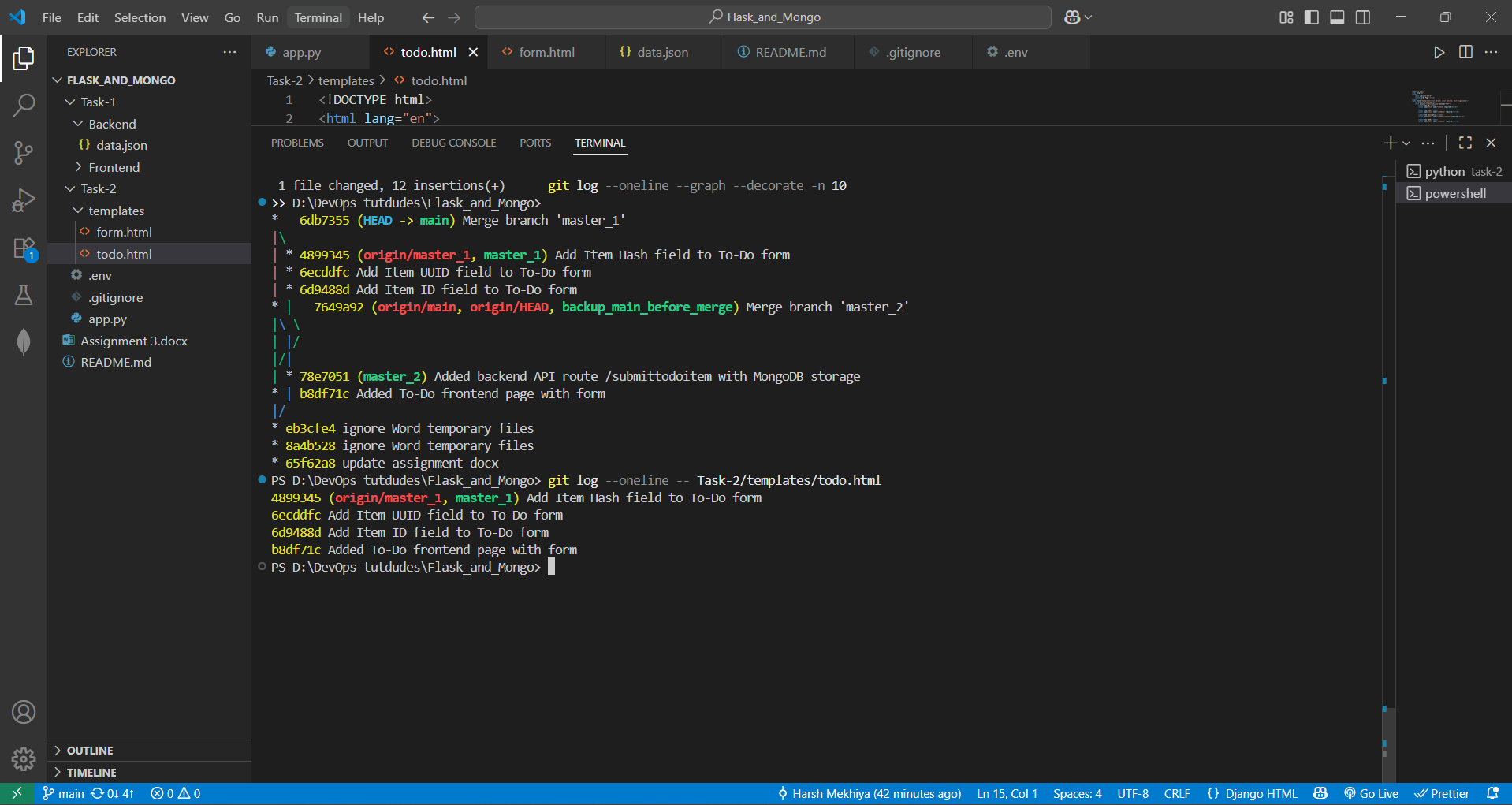
* **Merging to main:**
* **Merge the master\_1 branch into the main branch.**

****

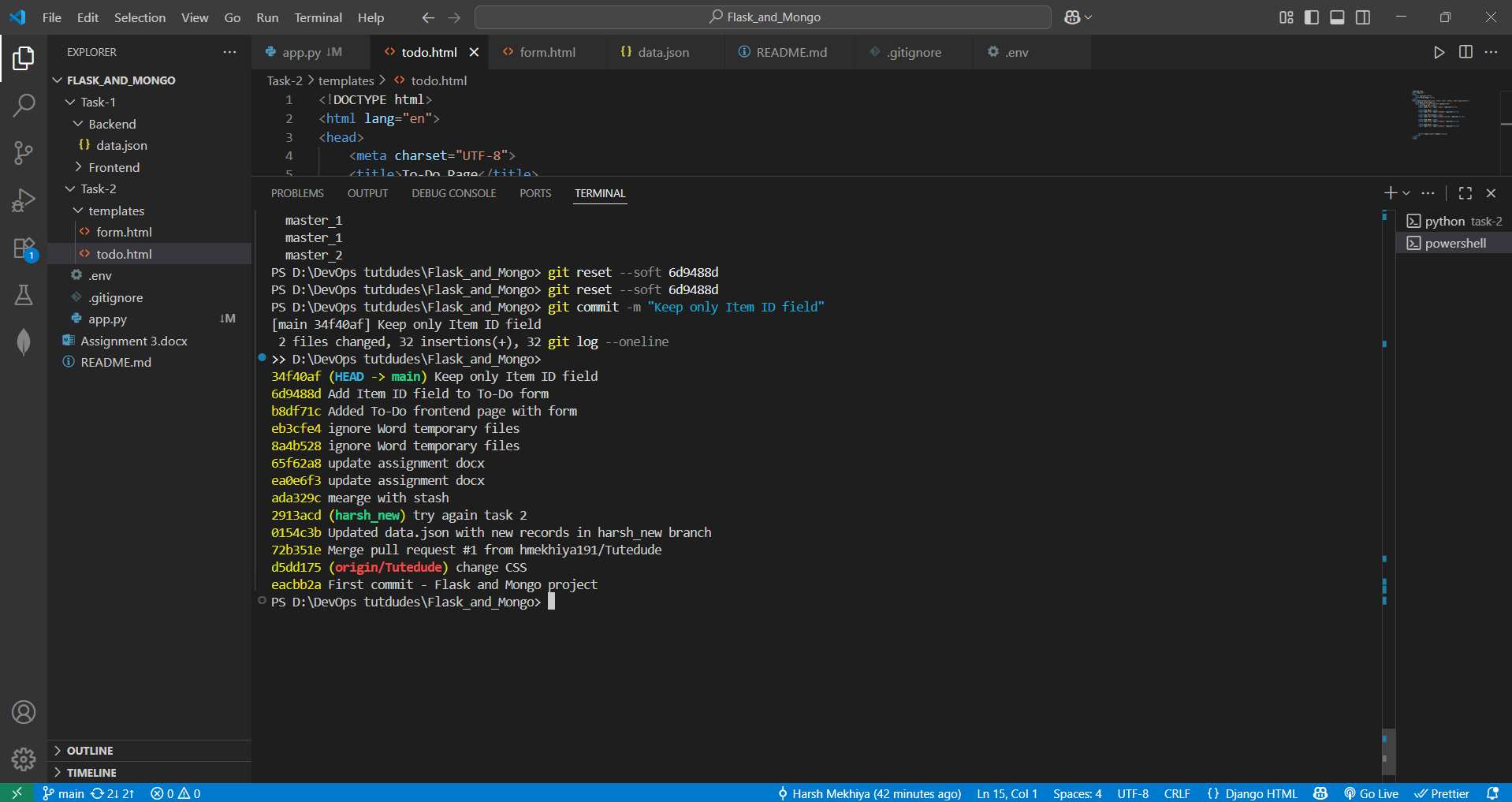
****

****

****

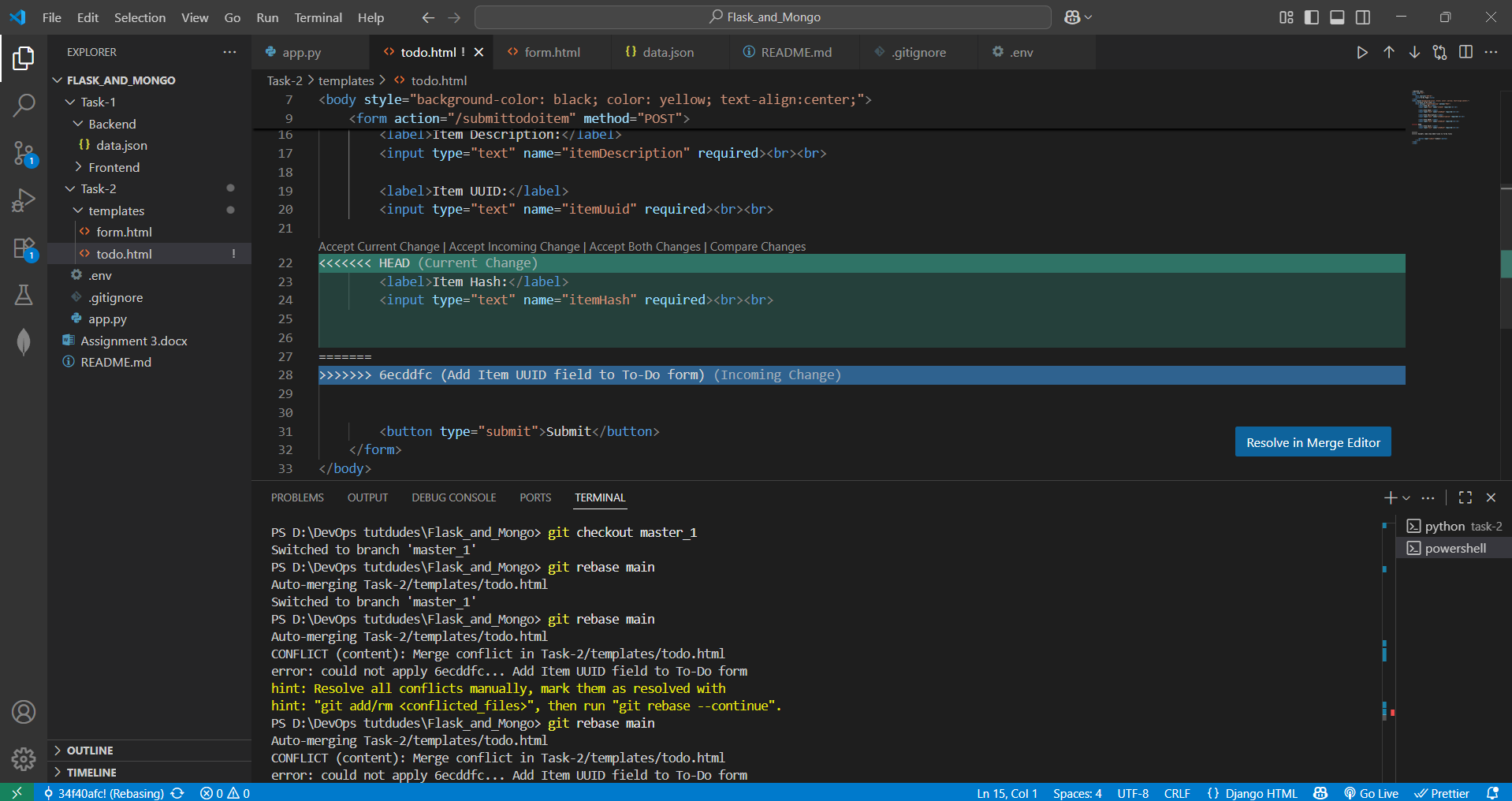
****

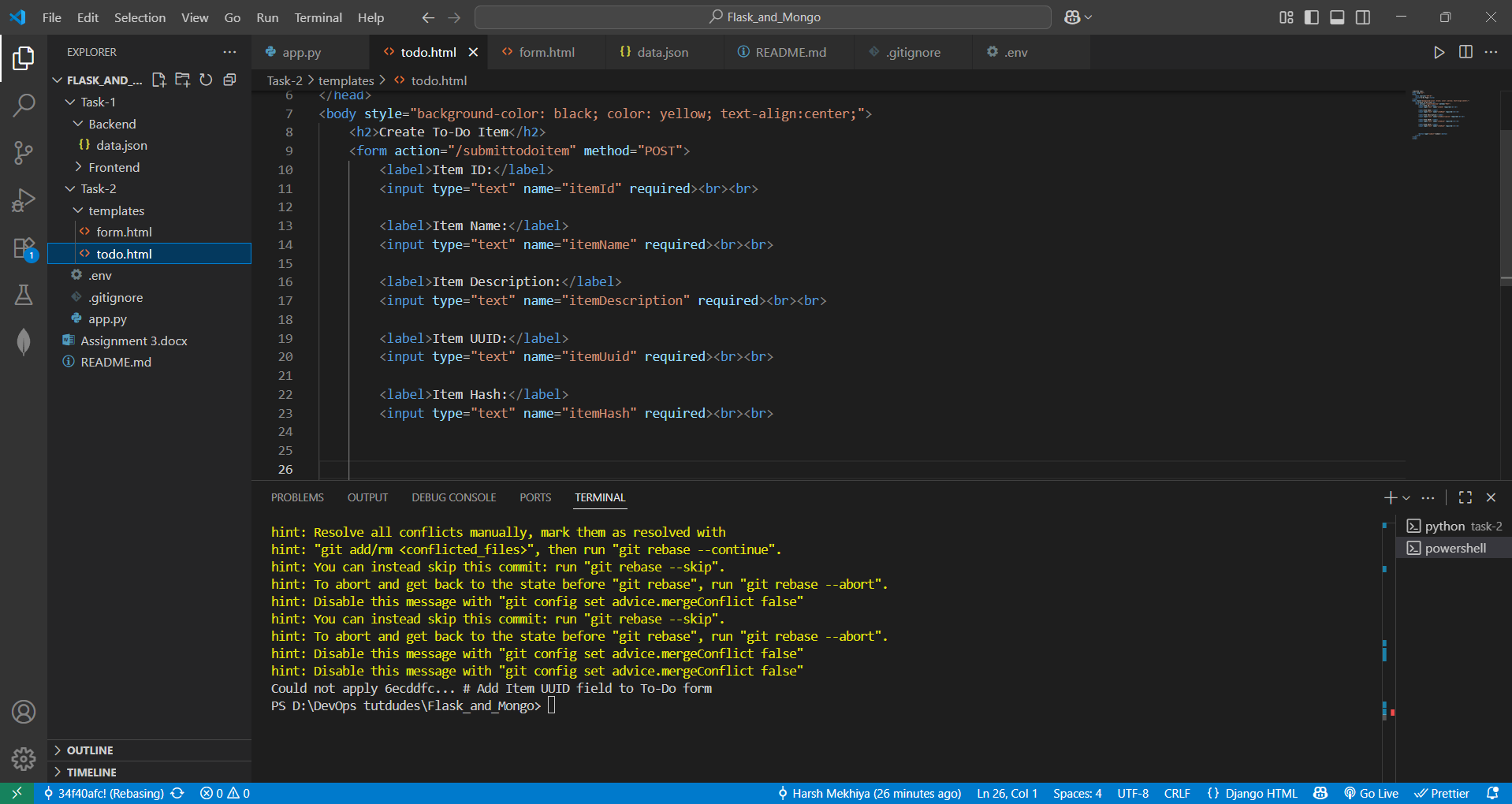
* **Git Reset and Commit Deletion**
* **In the main branch, use Git Reset to roll back to the commit where only the Item ID field was added.**
* **Use git reset --soft to ensure changes remain staged.**

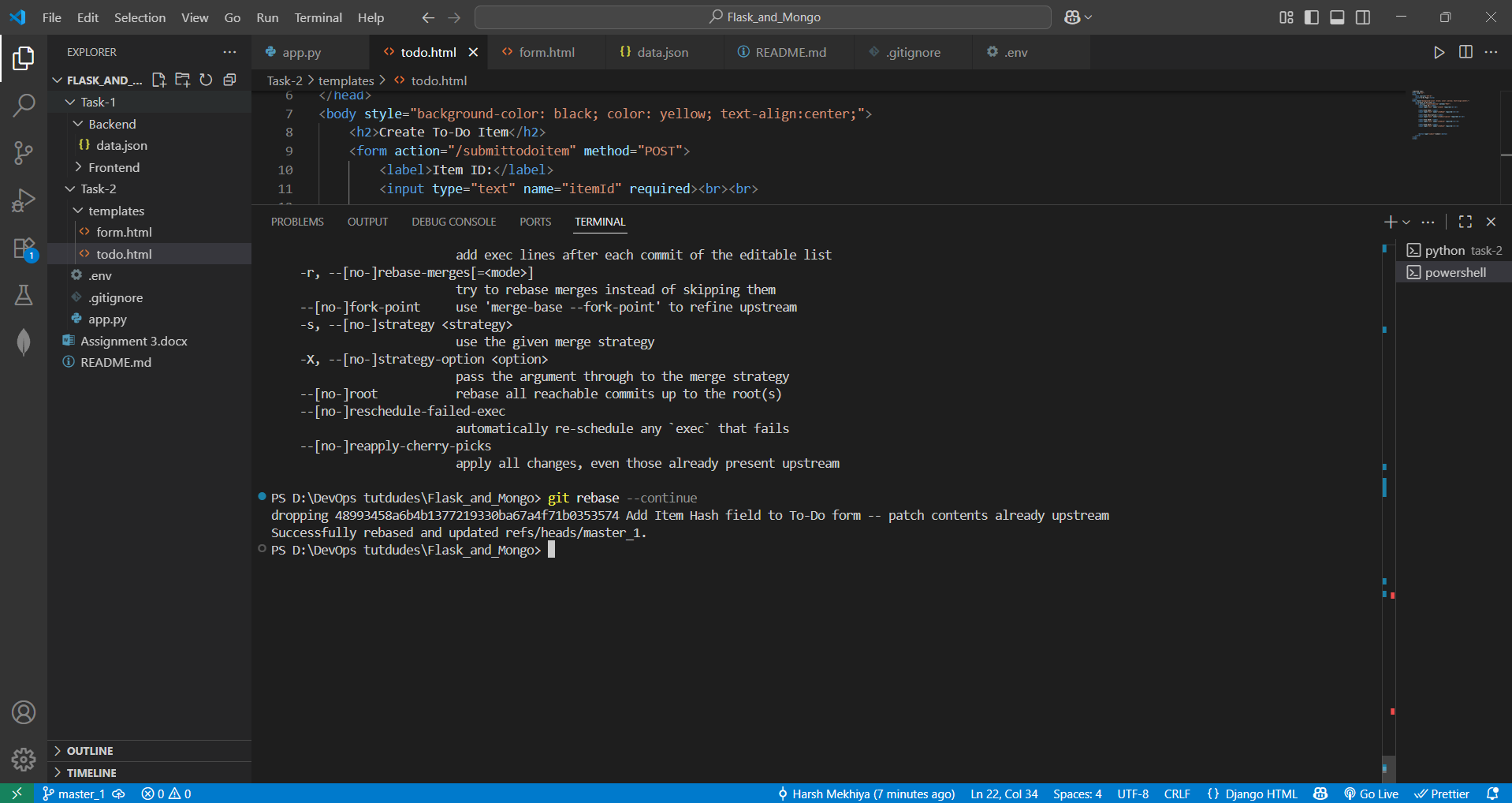
****

* **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.**

****

****

****

**Explanation:**

In this task, I was working with two branches: main and master\_1. The main branch only contained the commit where I added the *Item ID* field, while the master\_1 branch had three sequential commits for *Item ID*, *Item UUID*, and *Item Hash*. When I rebased master\_1 onto main, Git attempted to replay the commits from master\_1 on top of main. During this process, I encountered a merge conflict in the todo.html file because both branches had made changes to the same section of code. Git marked the conflict with special symbols (<<<<<<<, =======, >>>>>>>), and I had to manually edit the file to resolve it. I chose to keep the required field(s), removed the conflict markers, and then staged the file using git add. After that, I ran git rebase --continue to proceed with the rebase. I repeated this process until all conflicts were resolved and the rebase finished successfully. As a result, the master\_1 branch now has a clean commit history with separate commits for each field (Item ID, Item UUID, and Item Hash), while the main branch preserves only the desired changes.