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.

Answer :-

1. Initial Setup and Repository Operations

A. Create GitHub Repo & Clone

1. Go to GitHub and create a **new repository** (e.g., flask-todo-app).
2. Open terminal on your local machine:

bash
CopyEdit
ssh-keygen -t ed25519 -C "your_email@example.com"

3. Add the public key (~/.ssh/id_ed25519.pub) to GitHub:
   GitHub > Settings > SSH and GPG keys > New SSH key
4. Clone the repository:

bash
CopyEdit
git clone git@github.com:your_username/flask-todo-app.git
cd flask-todo-app

B. Create Branch & Push Project

bash
CopyEdit
git checkout -b your_username # Replace with your GitHub username
Copy your Flask project files into this directory
git add .
git commit -m "Add initial Flask project"
git checkout main
git merge your_username
git push origin main

---

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.

Answer :-

2. JSON API Branch Update and Merge
A. Create and Update Branch
bash
CopyEdit
git checkout -b your_username_new
Modify the JSON file used in /api route
git add your_json_file.json
git commit -m "Update JSON data for /api route"
git checkout main
git merge your_username_new
B. Conflict Resolution (if needed)
If there are **merge conflicts**:
bash
CopyEdit
Open the conflicting files and manually resolve them
Accept the changes from `your_username_new`

git add .
git commit -m "Resolve merge conflict by accepting your_username_new changes"
git push origin main

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.

Answer :-

3. To-Do Feature with Branches

A. Create master_1 and master_2

bash
CopyEdit
git checkout main
git checkout -b master_1
git checkout main
git checkout -b master_2

B. In master_1: Create To-Do Page (frontend)

Add HTML/JS code to create a form with:

- Item Name
- Item Description

bash
CopyEdit
git add .
git commit -m "Add To-Do frontend form with name and description"

C. In master_2: Create /submittodoitem Route (backend)

Example Flask code:

python
CopyEdit

```
from flask import Flask, request
from pymongo import MongoClient

app = Flask(__name__)
client = MongoClient("mongodb://localhost:27017/")
db = client["todo_db"]
collection = db["todo_items"]

@app.route("/submittodoitem", methods=["POST"])
def submit_item():
    data = request.get_json()
    collection.insert_one({
        "itemName": data["itemName"],
        "itemDescription": data["itemDescription"]
    })
    return {"status": "success"}, 200
```

bash
CopyEdit
git add .

git commit -m "Add /submittodoitem backend route with MongoDB integration"

✅ **D. Merge Both to main**

bash
CopyEdit
git checkout main
git merge master_1
git merge master_2
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.

Answer :-

4. Enhancing and Resetting To-Do Form
A. In master_1: Add Fields Sequentially
bash

CopyEdit
git checkout master_1

1. Add Item ID field
Modify frontend form
git add .
git commit -m "Add Item ID field to form"

2. Add Item UUID field
Add UUID field to form
git add .
git commit -m "Add Item UUID field to form"

3. Add Item Hash field
Add hash field to form
git add .
git commit -m "Add Item Hash field to form"

B. Merge master_1 to main

bash
CopyEdit
git checkout main
git merge master_1
git push origin main

C. Reset to Commit with Only Item ID

bash
CopyEdit
git log # Find the commit ID for "Add Item ID field"
git reset --soft <commit_id> # Keeps changes staged
git commit -m "Rollback to Item ID field only"
git push origin main --force

D. Rebase Changes Back to master_1

bash
CopyEdit
git checkout master_1
git rebase main # preserve commit history; don’t squash
If conflicts occur, resolve and run:
git add .
git rebase --continue
