### **Git Branching Strategy for Your Team**

Since your team consists of **four roles**, we will implement a **Git Feature Branch Workflow** with a **main** and **develop** branch structure to avoid conflicts.

### **Branching Structure**

1. **Main Branch (main)**
   * This is the final, stable version of the project. **No one directly commits here.**
   * Merges into main happen only after thorough review & testing.
2. **Development Branch (develop)**
   * This is where all new features are merged before reaching main.
   * Each member works on their own **feature branch** and merges into develop.
3. **Feature Branches (feature/{task-name})**
   * Each team member creates a new branch when working on a feature.
   * Example:
     + feature/backend-auth
     + feature/ui-homepage
     + feature/db-setup
     + feature/data-collection
4. **Hotfix Branches (hotfix/{bug-name})**
   * Used to fix urgent issues in production.

### **How Each Team Member Should Work on Git Branches**

| **Role** | **Branch Naming** | **Responsibilities** |
| --- | --- | --- |
| **Scrum Master & Backend (you)** | feature/backend-\* | API, authentication, server logic |
| **UI/UX & Frontend** | feature/ui-\* | Frontend design, Tailwind CSS |
| **Database Admin** | feature/db-\* | MongoDB setup, CRUD operations |
| **Data Researcher** | feature/data-\* | Collects & adds place details |

### **How to Use Git Branches from Linux Terminal**

#### **1. Clone the Repository (If Not Cloned Yet)**

git clone <repo-url>

cd <repo-folder>

#### **2. Create and Switch to a Feature Branch**

git checkout -b feature/backend-auth

#### **3. Make Changes & Commit**

git add .

git commit -m "Implemented authentication system"

#### **4. Push the Feature Branch to GitHub**

git pu origin feature/backend-auth

#### **5. Switch Back to Develop Branch**

git checkout develop

#### **6. Merge Your Feature Branch into Develop**

git merge feature/backend-auth

#### **7. Delete the Feature Branch (If Merged Successfully)**

git branch -d feature/backend-auth

### **Merging Process**

1. **Before merging into develop**, the team must review each other's code via **pull requests (PRs)** on GitHub.
2. Once develop is stable, it gets merged into main after final testing.

### **Best Practices**

**Always pull the latest changes before starting work:**   
  
git pull origin develop

* **Use meaningful commit messages.**
* **Never commit directly to main or develop.**
* **Always test your feature branch before merging into develop.**

### **Final Thoughts**

This structure ensures that your **main** branch stays clean and **develop** is always in sync with stable new features. Let me know if you need more details!

##### **You said:**

So if for example me and the person responsible for the ui and frontend start to work on our project one day, and we both start correctly by pulling the code and then developing our features in our feature branches. My colleague finishes lets say a page and some ui components and pushes the changes to the branch they created, on the other hand i pulled the code from the main branch when my colleague hadn't created the new page and ui components ( as she did the same ), and now i implemented some auth backend feature and pushed the code to my feature branch, after testing our new features we decide that they are okay to push to develop branch and merged the feature branches with the develop branch, does my colleague get my backend feature and do i get the new ui frontend feature when we pull the code from the develop branch

###### **ChatGPT said:**

Yes, once you **both merge** your feature branches into the develop branch, you and your colleague will have access to each other's changes when you pull from develop.

### **Step-by-Step Breakdown of What Happens**

1. **You Both Start Working**
   * You and your colleague pull the latest develop branch.
   * You create feature/backend-auth, and your colleague creates feature/ui-homepage.
   * You work on backend auth, and your colleague works on UI.
2. **Your Colleague Finishes First**
   * They push feature/ui-homepage to GitHub.
   * They merge it into develop.
   * At this point, **develop** now has the new UI components.
3. **You Finish Your Backend Work Later**
   * Your local branch feature/backend-auth **does not yet have** the new UI because you created your branch before it was merged into develop.
   * You push feature/backend-auth to GitHub.
   * You then merge it into develop.
   * Now, **develop** has both the UI and backend features.
4. **How You Both Get Each Other’s Features**
   * If your colleague pulls develop now, they will **get your backend feature**.
   * If you pull develop, you will **get their UI components**.

### **What Should You Do to Avoid Conflicts?**

If your colleague merges their feature/ui-homepage into develop **before** you finish, you should:

**Before merging your feature, pull the latest develop into your branch**:  
  
   
  
git checkout feature/backend-auth

git pull origin develop

* + This updates your branch with the latest UI changes.
  + You can fix any conflicts if necessary.

**Then, merge your branch into develop as usual**:  
  
   
  
git checkout develop

git merge feature/backend-auth

git push origin develop

Now, you **both** have the latest changes without conflicts.

##### **You said:**

How to create develop branch

###### **ChatGPT said:**

To create and set up the develop branch, follow these steps in your Linux terminal:

### **1. Check Your Current Branch (Make Sure You’re on main)**

git branch

* This will show the current branches.

If you're not on main, switch to it:  
   
  
git checkout main

### **2. Create the develop Branch**

git branch develop

### **3. Switch to the develop Branch**

git checkout develop

### **4. Push develop to GitHub (So Others Can Use It)**

git push -u origin develop

Now, your team can pull and work from the develop branch instead of main.