Gitflow Strategy for the Personal Finance Tracker App

This document explains how to implement the **Gitflow strategy** for the **Personal Finance Tracker App** project. Gitflow is a branching model that provides a robust workflow for managing development, features, and releases. It is especially helpful for a team of three working collaboratively on a large project.

Overview of Gitflow Strategy

Gitflow uses multiple branches to manage different stages of development. It helps to maintain clean code, track features, manage releases, and handle bug fixes effectively.

Key Branches in Gitflow

1. `main`: The stable production branch containing the latest release-ready code.
2. `develop`: The main development branch where the team integrates new features and changes.
3. `feature`: Temporary branches created to develop new features.
4. `release`: Used to prepare code for production releases.
5. `hotfix`: Used to fix urgent issues in the `main` branch after a release.

Gitflow Branching Structure

Here's how Gitflow branches can be structured for the project:

```
personal-finance-tracker/
│
├── main
│
├── develop
│   ├── feature/data-management
```

```
│   ├── feature/budget-management
│   ├── feature/data-analysis
│   └── feature/visualization
│
└── release
```

Step-by-Step Guide to Gitflow for the Project

1. Initial Setup

1. **Clone the Repository**:
   - Each team member should clone the repository to their local machine:

     ```
     git clone <https://github.com/your-username/personal-fi
     nance-tracker.git>
     cd personal-finance-tracker
     ```

2. **Create the `develop` Branch**:
   - The `develop` branch will be created from the `main` branch:

     ```
     git checkout -b develop
     git push origin develop
     ```

3. **Protect the `main` and `develop` Branches**:
   - Set up branch protection rules on GitHub to prevent direct pushes to these branches.
   - This ensures that changes are merged through pull requests (PRs) only.

---

2. Feature Development

1. **Create a Feature Branch**:
   - Each new feature should be developed in a separate branch created from `develop`.

- Example: For the data management module, create a branch called `feature/data-management`:

```
git checkout -b feature/data-management develop
```

2. **Work on the Feature**:
   - Develop and test the new feature in the feature branch.
   - For example, implement functions in `data_management.py` to handle loading, adding, editing, and deleting transactions.
3. **Commit Changes**:
   - Commit changes regularly, providing clear commit messages:

```
git add .
git commit -m "Add transaction loading and adding funct
ions to data management module"
```

4. **Push the Feature Branch**:
   - Push the feature branch to GitHub:

```
git push origin feature/data-management
```

5. **Create a Pull Request (PR) to `develop`**:
   - Once the feature is complete, create a pull request to merge it into the `develop` branch.
   - Team members should review the PR, suggest improvements, and approve it before merging.
6. **Merge into `develop`**:
   - After the PR is approved, merge it into the `develop` branch:

```
git checkout develop
git merge feature/data-management
```

```
git push origin develop
```

---

3. Preparing a Release

1. **Create a Release Branch**:
   - When the `develop` branch is stable and ready for release, create a `release` branch from `develop`.
   - Example:

```
git checkout -b release/1.0 develop
git push origin release/1.0
```

2. **Testing and Final Adjustments**:
   - In the `release` branch, perform final testing, bug fixes, and adjustments.
   - Only minor changes (e.g., documentation updates, final UI tweaks) should be made in this branch.
3. **Merge into `main` and `develop`**:
   - Once the release is ready, merge the `release` branch into both `main` and `develop`:

```
git checkout main
git merge release/1.0
git push origin main

git checkout develop
git merge release/1.0
git push origin develop
```

4. **Tag the Release**:
   - Tag the release in the `main` branch for easy reference:

```
git tag -a v1.0 -m "Release version 1.0"
git push origin v1.0
```

---

4. Hotfixes

1. **Create a Hotfix Branch**:
   - If a critical bug is found in production, create a `hotfix` branch from `main` to address it:

     ```
     git checkout -b hotfix/critical-bug main
     ```

2. **Fix the Bug**:
   - Make the necessary changes to fix the bug in the `hotfix` branch.

3. **Merge into `main` and `develop`**:
   - Once the bug is fixed, merge the `hotfix` branch into both `main` and `develop` to ensure consistency:

     ```
     git checkout main
     git merge hotfix/critical-bug
     git push origin main

     git checkout develop
     git merge hotfix/critical-bug
     git push origin develop
     ```

4. **Tag the Hotfix**:
   - Tag the hotfix in the `main` branch:

     ```
     git tag -a v1.0.1 -m "Hotfix for critical bug"
     git push origin v1.0.1
     ```

Example Workflow for the Personal Finance Tracker App

1. **Starting Development**:
   - Create a `develop` branch from `main`.
   - Team members create feature branches for different modules:
     - `feature/data-management`
     - `feature/budget-management`
     - `feature/data-analysis`
     - `feature/visualization`
2. **Adding a New Feature**:
   - A team member creates a `feature/data-management` branch to add transaction management functions.
   - Once complete, the feature is tested and merged into `develop` via a pull request.
3. **Preparing for Release**:
   - After all planned features are merged into `develop`, a `release/1.0` branch is created.
   - Final testing and adjustments are made, then merged into both `main` and `develop`.
4. **Fixing a Critical Bug**:
   - If a critical bug is found after release, a `hotfix/critical-bug` branch is created from `main`.
   - After the bug is fixed, the branch is merged into both `main` and `develop`, ensuring consistency.

Summary

- **Main Branches**: `main`, `develop`, `feature`, `release`, `hotfix`.
- **Feature Development**: Develop new features in separate branches from `develop`.

- **Release Preparation**: Create a `release` branch for final testing and adjustments.
- **Hotfixes**: Use a `hotfix` branch to fix critical bugs in production.