GITHUB BRANCH AND DESIGN

To implement a branching strategy in GitHub that aligns with the development, test, QA, UAT, and production stages, you can adopt a structure that leverages Git's branching capabilities to manage code through different stages of development and deployment. This strategy helps in maintaining a clean and organized codebase, facilitating collaboration, and ensuring that code changes are thoroughly tested before reaching production. Here's how you can set up such a branching strategy:

1. Main Branch

* Purpose: This branch should always represent the production-ready state of your application. It is the default branch for your repository and serves as the base for creating feature branches.
* Access: Only merge code into the main branch after it has been thoroughly tested and approved in the UAT environment.

2. Develop Branch

* Purpose: This branch is used for active development. Developers create feature branches off of develop and merge them back into develop once their work is completed and tested.
* Access: All developers should have access to this branch to work on new features or bug fixes.

3. Feature Branches

* Purpose: These branches are created off the develop branch for each new feature or bug fix. They are used to isolate work until it is ready for integration.
* Naming Convention: Use descriptive names, such as feature/add-new-payment-types.
* Merging: Once a feature is complete and tested, it should be merged back into the develop branch.

4. Release Branches

* Purpose: These branches are created from the develop branch when the team is ready to start preparing a new production release. They allow for final bug fixes and preparation of release documentation.
* Naming Convention: Use version numbers or release names, such as release/1.0.1.
* Merging: Once the release is finalized, it should be merged into the main branch and tagged with the release version.

5. Hotfix Branches

* Purpose: These branches are used to quickly patch production releases. They are created from the main branch and merged back into both the main and develop branches once the fix is applied.
* Naming Convention: Use descriptive names, such as hotfix/fix-critical-bug.
* Merging: After the hotfix is applied, it should be merged back into both the main and develop branches.

6. QA and UAT Branches

* Purpose: These branches are used for testing. QA (Quality Assurance) tests the features in a more controlled environment, while UAT (User Acceptance Testing) involves end-users testing the application in a real-world scenario.
* Access: Only QA and UAT teams should have access to these branches.

7. Merging and Pull Requests

* Use pull requests to merge changes from feature, release, and hotfix branches into the develop and main branches. This ensures that code changes are reviewed and approved before they are integrated.
* Enforce branch protection rules on your main and develop branches to prevent direct pushes and ensure that pull requests are reviewed.

This branching strategy helps in organizing your development workflow, ensuring that code changes are thoroughly tested and approved at each stage before reaching production. It also supports collaboration among team members by allowing them to work on different features simultaneously without affecting the main codebase.