Branching strategy

The branching model.

- `production` **branch:** This is the source of truth for the **current live website**. It is only used for urgent bug fixes ("break fix") and very minor code changes to the production site.
- `main` **branch:** This is the primary development branch for the **entire website redesign**. All new feature work and redesign efforts happen here. This branch will eventually replace `production` as the live branch.

Developer Workflow: Step-by-Step Instructions

Scenario 1: A Redesign Feature (Default Workflow)

When: For all work related to the new website redesign.

Step-by-Step Instructions:

1. **Get the latest `main` code:**

```
git checkout main
git pull origin main
```

2. **Create your feature branch:**

```
git checkout -b feature/new-user-dashboard
```

3. **Develop the feature:** Write the code, tests, and commit your work.

```
# ...work on the redesign feature...
git add .
git commit -m "Feat: Build new user dashboard page"
git push origin feature/new-user-dashboard
```

4. **Create a Pull Request (PR) to `main`:**
    - **Base Branch:** `main`
    - **Compare Branch:** `feature/new-user-dashboard`
    - Get it reviewed, merge it, and deploy `main` to the redesign's test (dev-web) environment.

This corrected workflow provides a clear separation of concerns while establishing the critical process of manually porting essential fixes from the legacy site to the new redesign codebase.

Scenario 2: A Production Hot fix or Minor Change (Prod Site)

When: An urgent bug is found on the live site, or a tiny, essential change is needed.

Step-by-Step Instructions:

1. **Get the latest `production` code:** Always start here for a hot fix.

```
git checkout production
git pull origin production
```

2. **Create your hot fix branch:**

```
git checkout -b hotfix/api-gateway-error
```

3. **Fix the bug:** Make the necessary changes to the prod code.

```
# ...write code and test...
git add .
git commit -m "Fix: Resolves API gateway timeout"
git push origin hotfix/api-gateway-error
```

4. **Create two Pull Requests (PR #1 and #2)**
    - **PR #1 to `production`**
        - **Base Branch:** `production`

    - **Compare Branch:** `hotfix/api-gateway-error`
- **PR #2 to `main` (see step #5 for detailed steps) - This is the back-merge PR.**
    - **Base Branch:** `hotfix/api-gateway-error` / (`production` for explicit back-merge)
    - Branch name: `chore/api-gateway-error`
    - **Merge (commits) into:** `main`

5. **CRITICAL NEXT STEP: Port the Fix to `main` (The Redesign Branch) - Back merging.**
    - The same bug likely exists in the new codebase. You must now apply the same logic to the `main` branch.
    - **Developer Responsibility:** The developer who made the hot fix is responsible for porting/back merging the code fix it to `main`.
    - **Method: Create the Main/Redesign PR (PR #2)**
        1. Update the `production` branch.

```
git checkout production
git pull
```

        2. Check out a new branch from the PR1 branch:

```
git checkout hotfix/api-gateway-error
git checkout -b chore/api-gateway-error
git merge production
git push
```

        3. (Optional) Use `git cherry-pick` to apply the commit:

```
# This will try to apply the exact change from that commit to your
# current branch
git cherry-pick a1b2c3d #optional
git push origin chore/api-gateway-fix
```

4. Create PR# 2. With **Base Branch (commits into):** `main` .
4. **Resolve Conflicts:** It is very likely you will have merge conflicts if the surrounding code has changed in the redesign. You must manually resolve these conflicts to ensure the fix works with the new code.

PR Reviews

- Get both PRs reviewed, merge it to `production` and `main` branch.
- Ensure the review comments are addressed and available in both PRs, and the branch/PR not deleted before making the changes available.

Example flow