Version Control Processes and Documentation

1. Branching Strategy

We will use a Git Flow-inspired branching model:

- `main` (or `master`): Code that can be or has been deployed into production.
- `develop`: Working version of the application, branched from `main`.
- `feature/*`: Different features of the application in development.

Process:

1. Branch `develop` from `main`.
2. Create feature branches from `develop`.
3. Work on features in their respective branches.
4. When a feature is complete and tested, create a pull request to merge into `develop`.
5. When `develop` is tested and approved, merge it into `main` for production deployment.

Branch Lifecycle:

- `main`: Long-lived branch representing the production-ready code.
- `develop`: Long-lived branch hosting the actively worked on version.
- `feature/*`: Short-lived branches deleted after merging into `develop`.

2. Commit Guidelines

- Use of clear, descriptive commit messages
- Start with a capitalised, imperative verb (e.g., "Add", "Fix", "Update")
- The first line is under 50 characters
- More details in the commit body if necessary

Example:

```
Add user authentication to Dining Services API

- Implement JWT-based authentication
- Create login and logout endpoints
- Add middleware for protected routes
```

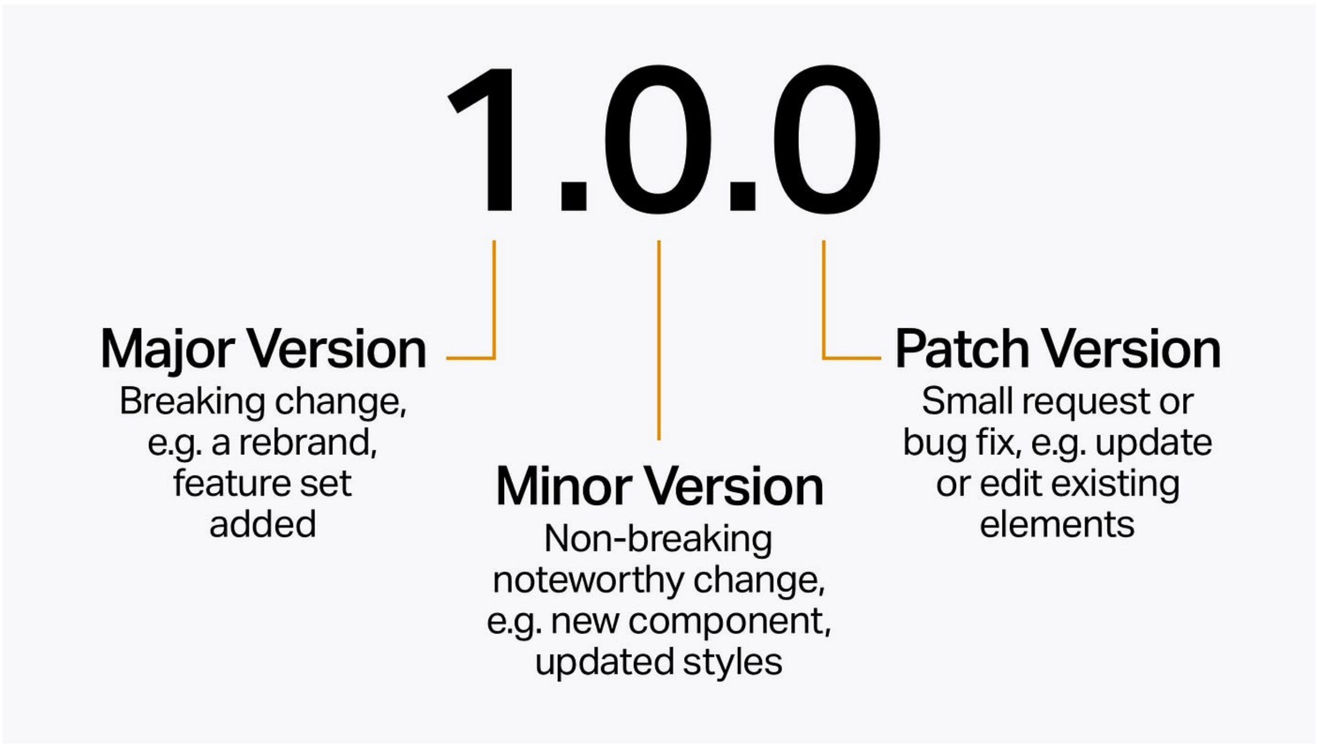
3. Code Review Process

1. Create a pull request (PR) for each completed feature
2. Address all comments and get approval before merging
3. After merging, delete the feature branch

4. Versioning

We will use Semantic Versioning for our project. Semantic Versioning is a formal convention for specifying compatibility using a three-part version number: MAJOR.MINOR.PATCH.

Example: 1.2.3 (Major.Minor.Patch)


1.0.0
Major Version
Breaking change, e.g. a rebrand, feature set added
Minor Version
Non-breaking noteworthy change, e.g. new component, updated styles
Patch Version
Small request or bug fix, e.g. update or edit existing elements

5. Documentation

README.md

We will Maintain an up-to-date README.md in the root of the repository, including:

- Project overview
- Setup instructions
- Key features
- Contributing guidelines

API Documentation

Swagger/OpenAPI for API documentation:

- Document all endpoints, request/response formats
- Keep documentation in sync with code changes

CHANGELOG.md

Maintain a CHANGELOG.md file to track version changes:

- List all notable changes for each version
- Categorize changes (Added, Changed, Deprecated, Removed, Fixed, Security)

6. Continuous Integration/Continuous Deployment (CI/CD)

- Set up automated testing for all branches
- Configure automated deployments to staging environments for `main` branch

7. Issue Tracking

We will use GitHub Issues:

- Track bugs, feature requests, and tasks
- Link issues to relevant commits and pull requests

8. Release Process

1. Ensure all desired features are merged into `develop`
2. Perform final testing on `develop`
3. Update version numbers and CHANGELOG.md
4. Create a pull request to merge `develop` into `main`
5. After approval, merge into `main`
6. Tag the release in `main`
7. Deploy to production