Github Project

Question:

Case Study: Get/Developing a Simple Application

Scenario:

You are part of a small development team tasked with managing git tasks for a simple java/*.net/python/ruby application.

The application should allow users to:

Get/Create a simple application.

Requirements:

The application, the code in it should be different for each student.

Project Setup:

Create a new Git repository for the project.

Initialize the repository with a README.md file explaining the project.

Create a .gitignore file to exclude unnecessary files and directories from version control

(e.g. *.class files, *.jar files etc.).

Initial Development:

Create a new branch for the initial development (feature/initial-development).

Get/Develop the core functionality of the application.

Make multiple commits

Commit your changes regularly with meaningful commit messages.

Push your changes to your remote repository (e.g., GitHub).

Collaboration:

Create a pull request to merge your feature/initial-development branch into the main branch.

Simulate a code review by requesting feedback from a classmate (or an imaginary reviewer) and addressing their comments.

Implement merge using different strategies like

Rebase your branch onto the main branch before merging to maintain a clean linear history.

Fast-Forward

3 way commit

With conflict fix the conflict

Please ensure to push all changes to github so that we have traceability.

Ensure all changes are done with Verified user commit (signatures)

Enhancements:

Create a new branch (feature-test).

Develop and test the new feature.

Create a pull request for the feature-test branch.

Bug Fixes:

Create a new branch (bugfix/issue-1) to fix a bug you introduced in a previous commit.

Fix the bug and test thoroughly.

Create a pull request for the bugfix/issue-1 branch.

Version Control:

Create a tag (e.g., v1.0) to mark the initial release of the application.

Create a new branch (feature/new-ui) for a major UI/UX redesign.

Git LFS (Optional):

If your application involves large files (e.g., images, audio), experiment with Git LFS to store them more efficiently.

GitHub Administration (if applicable):

If working in a team, explore GitHub's team and organization features.

Experiment with different access control levels for team members.

Create a project board to track the progress of the project.

Git Hooks (Optional):

Implement a pre-commit hook to check for code style violations (e.g., using a linter).

Implement a post-receive hook to notify the team of new commits (e.g., via email or Slack).

Deliverables:

A well-structured Git repository with a clear commit history.

A working to-do list application with the required features.

A well-documented project with a README file.

A report summarizing the project, including the challenges faced and the lessons learned.

Assessment:

Code quality and readability.

Git usage and best practices (branching, merging, rebasing, tagging).

Collaboration and communication skills.

Understanding of Git concepts and commands.

Ability to solve problems and troubleshoot issues.

Project Link:

https://github.com/Gowrisankar5877/ToDoList