Для контроля версий был выбран подход Feature Branching.

Данный подход к управлению версиями в разработке программного обеспечения предполагает, что каждая новая функциональность разрабатывается в отдельной ветке кода. После завершения разработки функциональности ветка сливается обратно в основную ветку (main).

Обоснование, почему для нашего проекта подходит данный подход.

1. **Изоляция функциональности:**
   * Каждая функциональность разрабатывается в изолированном пространстве, что предотвращает конфликты между кодом различных функциональностей.
   * В отличие от других подходов, таких как Trunk-Based Development, где все изменения происходят в единой ветке, Feature Branching позволяет избежать проблем, связанных с одновременными изменениями в различных частях кодовой базы.
2. **Удобный формат для параллельной разработки:**
   * Разработчики могут параллельно работать над разными фичами, что ускоряет процесс разработки.
   * Так как в рамках нашего проекта ограниченное количество разработчиков, данный поход позволяет разделить функциональности между разработчиками, что ускорит время на создание продукта
3. **Более простое внедрение новых функциональностей:**
   * Ветки функциональностей могут быть созданы и удалены по мере необходимости, что упрощает внедрение новых функциональностей.

*Структура репозитория*

Постоянные ветки

|  |  |  |
| --- | --- | --- |
| Наименование | Назначение | Примечание |
| main | Сборка релизов  Используется для деплоя в продакшн | Официальная история проекта  Содержит только стабильные коммиты |

Временные ветки

|  |  |  |
| --- | --- | --- |
| Наименование | Назначение | Примечание |
| feature/database-recipes | Добавление функциональности работы с БД для рецептов | Включает перенос рецептов в БД и настройку взаимодействия с БД. |
| feature/user-data-db | Добавление функциональности сохранения данных о пользователе в БД | Реализует механизм сохранения и управления данными о пользователях в базе данных. |
| feature/bot-skeleton | Построение основного скелета бота и обработка запросов | Включает в себя начальную структуру бота и базовую обработку запросов от пользователей. |
| hotfix/[название-ветки] | Решение критических проблем уже выпущенной версии программного продукта |  |

*CodeReview*

Разработчик, завершив работу в своей временной ветке, создает Pull Request для интеграции его изменений в основную ветку (main).

Член команды, отвечающий за разработку, в рамках созданного созданного Pull Request внимательно изучает изменения. Он оставляет комментарии, задает вопросы, указывает на потенциальные улучшения или проблемы в коде. Происходит обсуждение изменения, вносимые в код, и разрабатывают пути их улучшения.

При выявлении ошибок, неточностей, разработчик вносит необходимые изменения в код в ответ на комментарии рецензентов. После внесения изменений разработчик запросит повторный Code Review. Рецензенты могут проверить, были ли учтены их предложения и как влияют внесенные изменения на код.

После успешного завершения Code Review и устранения всех замечаний рецензентов Pull Request утверждается, изменения могут быть интегрированы в основную ветку.

*Типовые сценарии работы*

*- Внесение изменений или добавление новой функциональности*

1. **Создание временной ветки:**
   * Разработчик создает временную ветку для работы над задачей. Название ветки может соответствовать типу задачи или ее описанию.

git checkout -b feature/[название-ветки]

1. **Обновление локального репозитория:**
   * Разработчик обновляет локальный репозиторий, чтобы убедиться, что он актуален.

git pull origin [имя-ветки]

1. **Работа над задачей:**
   * Разработчик вносит необходимые изменения, разрабатывает новую функциональность, или решает задачу в рамках своей временной ветки. Каждый коммит снабжается комментарием, описывающим внесенные изменения.

git add .

git commit -m "Описание изменений"

1. **Фиксация изменений и отправка на сервер:**
   * Разработчик фиксирует изменения и отправляет их на удаленный сервер.

git push origin feature/[название-ветки]

1. **Создание Pull Request:**
   * Разработчик создает Pull Request на GitHub, указывая ветки для сравнения (свою временную ветку и ветку **main**).
2. **Code Review:**
   * Член команды, отвечающий за разработку, проводит Code Review. В случае выявления замечаний разработчик вносит изменения и повторно отправляет код на рассмотрение.
3. **Утверждение Pull Request:**
   * После успешного завершения Code Review и устранения всех замечаний, член команды утверждает Pull Request.
4. **Слияние в основную ветку:**
   * После утверждения разработчик сливает свою временную ветку в основную ветку проекта (**main**).

git checkout main

git pull origin main

git merge --no-ff feature/[название-ветки]

git push origin main

1. **Очистка локальной среды:**
   * Разработчик удаляет локальную временную ветку после успешного слияния.

git branch -d feature/[название-ветки]

- *Обнаружение критической проблемы в текущей стабильной версии бота*

Применяется аналогичный сценарий с некоторыми уточнениями

1. Временная ветка для работы над исправлениями: hotfix/[название-ветки]
2. После утверждения Pull Request и слияния веток, происходит обязательное распространение изменений на другие активные ветки проекта

git checkout feature/[название-ветки]

git pull origin feature/[название-ветки]

git merge --no-ff hotfix/[название-ветки]

git push origin feature/[название-ветки]