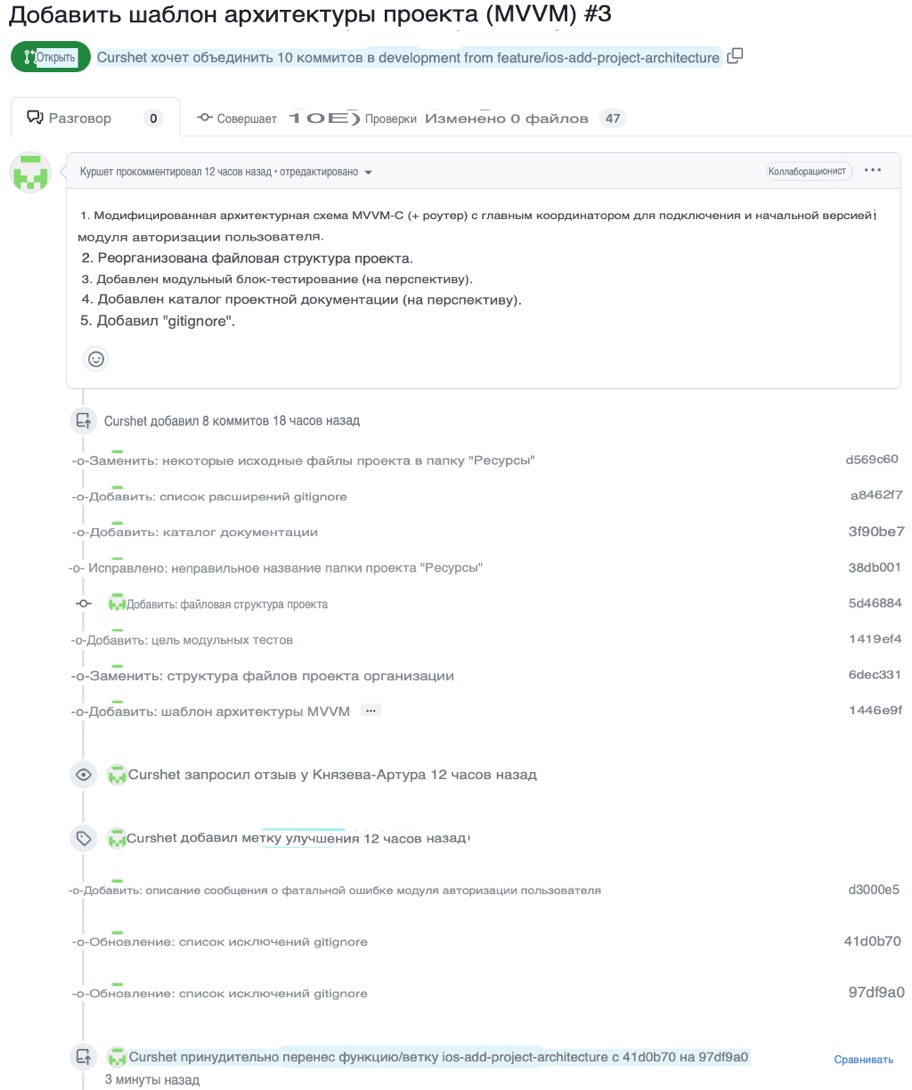
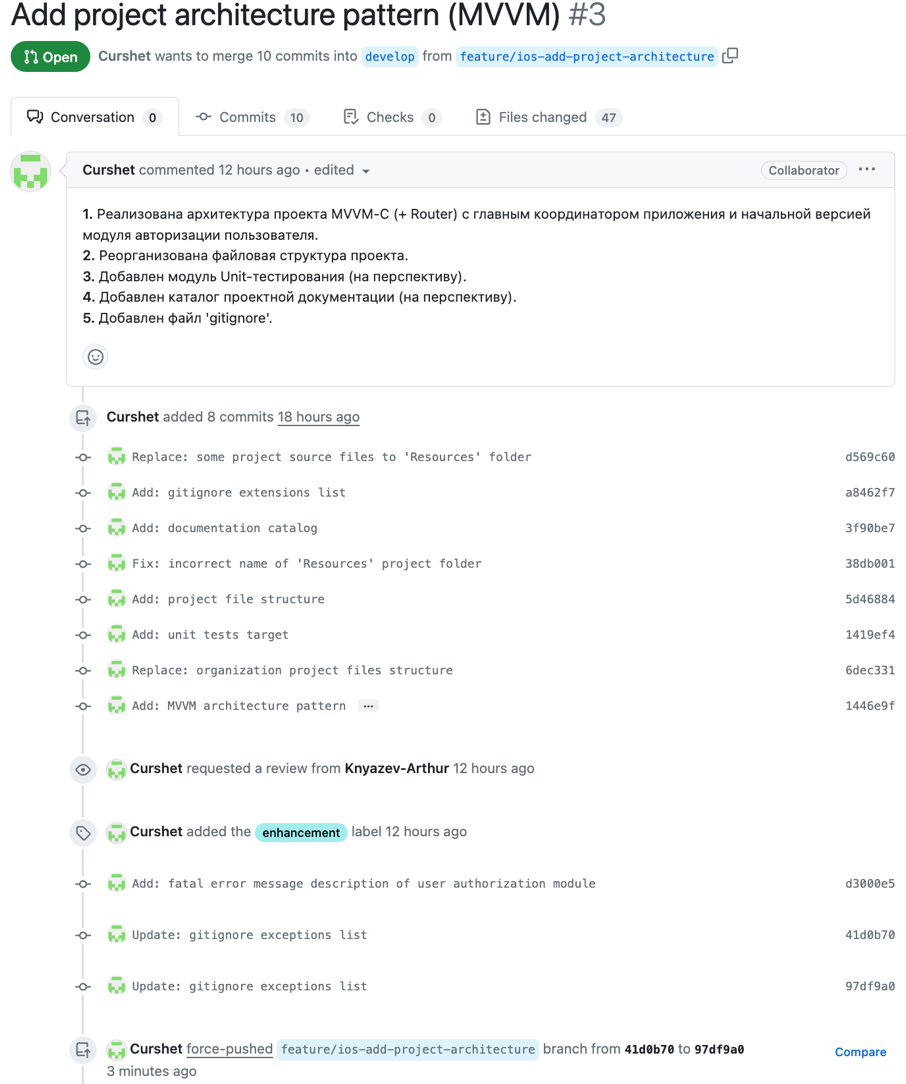
**Правила оформления запросов на слияние (pull request/PR) веток проекта**

* для описания PR используется русский язык
* название PR кратко и точно отражает основную цель/задачу, которую он решает
* описание PR содержит перечень и описание выполненных задач
* автор PR запрашивает просмотр кода - ревью (review) от других участников проекта

**Правила оформления коммитов при размещении проекта**

* для описания коммитов используется английский язык
* каждый коммит начинается с глагола в повелительном наклонении, например «Add», «Replace», «Fix»
* описание коммитов кратко и точно отражает суть внесенных изменений
* в некоторых случаях используются метки для дополнительной категоризации, как «enhancement»

**Пример**



**Стандартные метки в Git**

|  |  |
| --- | --- |
| Метка | Описание |
| bug | неожиданная проблема или непреднамеренное поведение в коде |
| documentation | необходимость улучшения или добавления документации |
| duplicate | указывает на то, что есть похожие вопросы, pull request’ы или обсуждения |
| good first issue | вопрос подходит для новых участников |
| help wanted | помощь мейнтейнеру (специалист сопровождения свободного ПО) по вопросу или pull request’у |
| Invalid | вопрос, pull request или обсуждение больше не актуальны |
| question | по вопросу, pull request’у или обсуждению требуется больше информации |
| wontfix | работа над вопросом, pull request’ом или обсуждением не будет продолжена |
| enhancement | улучшение существующей функциональности |

**Правила создания проекта**

* код проекта организовывается по функциональности или модулям. Например, по отдельным папкам для различных модулей, таких как «Main», «Map», «Profile»
* создается файл README
* создается файл gitignore, в котором указываются названия или расширения файлов, не учитываемых при отправки в удаленный репозиторий Git
* создается документация
* добавляется локализация проекта
* добавляются файлы тестирования

**Правила работы с проектом**

* реализация новой/отдельной задачи выполняется в новой/отдельной ветке
* для внесения изменений в основную ветку используются pull requests
* перед внесением изменений в основную ветку код проходит процесс ревью
* ветки в репозитории следуют определенной номенклатуре. Например, в схеме «feature/ios-add-project-architecture», где «feature» - тип ветки, «ios- платформа, «add-project-architecture» - краткое описание задачи
* разработка проекта реализуется на нескольких ветках (branch, параллельных линиях разработки), чтобы при допущении ошибок в одной версии проекта, не пострадала другая его версия.

**Содержание файла README**

* название проекта
* описание проекта - краткое описание того, что делает приложение и для кого оно предназначено
* требования - минимальные требования к версии iOS, устройствам, наличие определенных библиотек или зависимостей
* установка - инструкции по установке приложения. Информацию о том, как установить зависимости через CocoaPods, Carthage или Swift Package Manager, а также как загрузить исходный код из репозитория.
* Использование - информация о запуске, основных функциях и возможностях, а также о том, как пользователи могут взаимодействовать с приложением
* архитектура проекта - краткое описание используемой архитектуры проекта
* зависимости - указание сторонних библиотек, фреймворков или инструментов, которые используются в проекте
* лицензия - информация о лицензии, в соответствии с которой распространяется приложение. Это может быть MIT, Apache, GPL и т. д.
* авторы - имена авторов или команду разработчиков, которые работали над проектом
* ссылки на документацию, репозиторий проекта, багтрекер, обсуждения или другие полезные ресурсы
* версии – текущая и предыдущее версии
* примеры кода - если есть какие-то особенности или сложные моменты в приложении, можно предоставить примеры кода для их иллюстрации

**Gitflow**

Стандартная модель ветвления, включающая в себя 5 веток:

* master – главная ветка, в которую сливаются все остальные. должна содержать исключительно рабочую версию проекта
* develop – ветка для разработки, служащая основой для функциональных веток. должная содержать последние разработки
* feature - ветка для разработки новых функций. создается от ветки develop и сливается обратно в нее, когда функция завершена
* hotfix - ветка для быстрого исправления ошибок в рабочей версии проекта. создается от ветки master и сливается обратно в нее и develop, когда ошибки исправлены
* release - ветка для подготовки нового продуктового релиза. создается от ветки develop и сливается обратно в нее и master, когда релиз готов

**Процесс запуска приложения**

Этапы запуска приложения от момента запуска до момента, когда пользователь видит и может взаимодействовать с интерфейсом приложения, могут включать в себя:

Инициализация приложения:

* Когда пользователь нажимает на значок приложения на устройстве, система iOS запускает ваше приложение. В этом этапе система инициализирует необходимые ресурсы и окружение для запуска приложения.

Main-файл:

* Система ищет файл main.swift или @UIApplicationMain атрибут в вашем проекте. Это основная точка входа для вашего приложения.

Загрузка главного интерфейса:

* Ваше приложение загружает главный интерфейс, который обычно определен в Main.storyboard или MainInterface.storyboard.

Загрузка информации о конфигурации:

* Приложение может загрузить информацию о своей конфигурации из Info.plist, такую как иконки приложения, описания, права доступа и другие настройки.

Запуск цикла жизни приложения:

* Когда главный интерфейс загружен, запускается цикл жизни приложения. Это включает в себя вызов методов делегата приложения, таких как application(\_:didFinishLaunchingWithOptions:), applicationWillResignActive, applicationDidEnterBackground и другие, в зависимости от текущего состояния приложения.

Отображение интерфейса:

* После инициализации приложение отображает свой интерфейс на экране устройства. Это может включать в себя отображение начального экрана, загрузочных экранов или любой другой пользовательский интерфейс.

Взаимодействие с пользователем:

* После загрузки интерфейса пользователь может взаимодействовать с приложением, выполняя различные действия, такие как нажатие на кнопки, ввод текста и т. д.

Обработка событий и запросов:

* Приложение обрабатывает действия пользователя, обновляет интерфейс и выполняет необходимые операции, включая обработку сетевых запросов, загрузку данных из базы данных и т. д.

**Сборка объектов**

Создание экземпляров классов:

* Первый шаг в сборке объектов - создание экземпляров классов, которые будут использоваться в вашем приложении. Это может быть экземпляр представления (UIView), контроллера (UIViewController), модели данных (Model) и т. д.
* Обычно создание экземпляров классов происходит с использованием ключевого слова init или с использованием фабричных методов.

Инициализация объектов:

* После создания экземпляров классов они должны быть проинициализированы, чтобы установить начальное состояние и поведение.
* В инициализаторах (конструкторах) объектов происходит установка значений свойств, подписка на уведомления, установка делегатов и другие необходимые действия.

Установка взаимосвязей:

* Многие объекты в вашем приложении могут быть связаны между собой. Например, представление может ссылаться на контроллер, контроллер может иметь ссылки на модели данных, а сервисы могут быть доступны через инъекцию зависимостей.
* Взаимосвязи между объектами обеспечивают согласованное взаимодействие между ними в рамках архитектурного паттерна вашего приложения.

Жизненный цикл объектов:

* После создания и инициализации объектов они начинают свой жизненный цикл, который может включать в себя различные состояния, такие как активное, неактивное, удаленное и т. д.
* Каждый объект в вашем приложении имеет свой собственный жизненный цикл, и понимание этого цикла помогает эффективно управлять ресурсами и обеспечивать правильное поведение приложения.

Уничтожение объектов:

* Когда объект больше не нужен, он должен быть уничтожен, чтобы освободить ресурсы и избежать утечек памяти.
* Обычно это происходит автоматически при помощи сборщика мусора (Garbage Collector) в языках программирования с автоматическим управлением памятью, таких как Swift и Objective-C, или при помощи ARC (Automatic Reference Counting).

**Основные аспекты Code Style**

Форматирование кода:

* Использование правильного форматирования кода помогает повысить его читаемость и понимание.
* Это включает в себя правильное выравнивание, использование отступов, расстановку фигурных скобок и т. д.

Именование переменных и функций:

* Понятные и описательные имена переменных и функций делают код более понятным и самодокументируемым.
* Имена должны быть осмысленными и отражать назначение объекта или операции, которую они представляют.

Структура файлов:

* Организация файлов в проекте также важна для его понимания и навигации.
* Файлы должны быть логически сгруппированы по функциональности и расположены в соответствующих папках и подпапках.

Комментарии и документация:

* Комментарии и документация помогают понять цели и особенности кода.
* Важно писать чистые и информативные комментарии, которые объясняют сложные или нетривиальные части кода.

Использование констант:

* Использование констант для хранения значений, которые могут измениться в будущем, делает код более гибким и легко поддерживаемым.
* Это помогает избежать магических чисел и строк в коде.

Управление ошибками:

* Обработка и управление ошибками должны быть единообразными и предсказуемыми во всем коде.
* Использование стандартных соглашений и паттернов помогает обеспечить надежную обработку ошибок.

Стиль написания кода:

* Консистентность в стиле написания кода важна для улучшения его читаемости и поддержки.
* Это включает в себя использование одного стиля скобок, выравнивание операторов, использование или отсутствие пустых строк и т. д.

Проверка стиля кода:

* Использование инструментов для автоматической проверки стиля кода помогает обнаружить и исправить нарушения соглашений.
* Это может быть реализовано с помощью статических анализаторов кода или интегрированных сред разработки.