



TestCloud Suite — Всё, что нужно QA — в одном облаке

TestCloud Suite – это интегрированная облачная платформа для команд качества (QA), объединяющая в одном месте все необходимые инструменты тестирования. Она сочетает возможности менеджера тест-кейсов, баг-трекера, среды для API-тестирования и хаба автоматизации UI-тестов. Цель платформы – предоставить единую экосистему для QA-команды, устраняя разрозненность инструментов и повышая эффективность процесса обеспечения качества.

Проблема, которую решает продукт

QA-инженерам зачастую приходится использовать разрозненные инструменты: таблицы Excel/Google для тест-кейсов, отдельный баг-трекер для дефектов, отдельные средства для API и UI тестирования. Такой разнобой затрудняет поддержание единого пространства тестовой документации и замедляет работу команды ¹. Отсутствие интеграции ведет к дублированию данных, сложности в отслеживании покрытий и метрик, а также к дополнительным усилиям по синхронизации между системами. **TestCloud Suite** решает эту проблему, объединив **всё, что нужно QA, в одном облаке**: вся информация о тестировании хранится централизованно, что создает единое рабочее пространство для QA-команды ². Это повышает прозрачность процесса, облегчает трассируемость (например, от требования до тест-кейса и бага) и ускоряет цикл обратной связи между тестировщиками и разработчиками.

Основные боли, которые устраняет TestCloud Suite: - **Разрозненность инструментов:** больше не нужно переключаться между множеством приложений – все модули (тест-кейсы, баги, авто-тесты) доступны в единой платформе. - **Потеря трассируемости:** платформа автоматически связывает тест-кейсы, результаты прогона и баг-репорты, обеспечивая полную прослеживаемость. Например, из отчета о тестовом прогоне можно сразу перейти к связанным дефектам. - **Инэффективность и ошибки интеграции:** единая система устраняет проблемы интеграции между отдельными сервисами (Test Case Management + Bug Tracking и т.д.), уменьшая риск человеческих ошибок при переносе данных и ускоряя работу команды ¹. - **Ограниченнная видимость метрик:** TestCloud Suite предоставляет сквозную аналитику по качеству – от статуса тестирования до показателей дефектов – в реальном времени, что затруднительно при разрозненных инструментах.

Модули платформы TestCloud Suite

Платформа состоит из нескольких модулей, каждый из которых отвечает за определенный аспект процесса тестирования. В совокупности они покрывают полный цикл QA – от планирования тестов до отслеживания дефектов и запуска автоматических сценариев. Ниже подробно описаны основные модули:

Test Manager (управление тест-кейсами)

Test Manager – модуль управления тестовой документацией. Он позволяет создавать и организовывать тест-кейсы (тестовые сценарии) и тестовые наборы: - **Создание и иерархия тест-кейсов:** QA-инженеры могут создавать тест-кейсы с шагами, ожидаемыми результатами и приоритетами. Тест-кейсы можно группировать по модулям или функциям приложения, присваивать метки и формировать иерархию (папки или разделы для удобства навигации). - **Тестовые наборы/планы:** поддерживается объединение тест-кейсов в **тестовые наборы** (Test Suites) или планы для выполнения определенного цикла тестирования. Это может быть регрессионный набор или набор для конкретного релиза. - **Управление тестовыми циклами:** модуль позволяет планировать **прогоны тестирования** (Test Runs) – выбирать набор тест-кейсов и запускать их выполнение (вручную или автоматически). Каждый прогон фиксируется с указанием версии приложения, окружения и ответственного исполнителя. - **Отслеживание статуса и покрытие требований:** в реальном времени виден прогресс выполнения тест-кейсов (сколько пройдено, провалено, заблокировано). Поддерживается привязка тест-кейсов к требованиям или user stories, что дает метрику покрытия требований тестами ³. - **Импорт/экспорт и интеграция:** модуль поддерживает импорт существующих тест-кейсов из CSV/Excel, что облегчает миграцию с других инструментов. Реализована интеграция с баг-трекером платформы: при провале шага тест-кейса тестировщик может сразу создать баг, ссылка на который сохранится в результатах прогона. - **Различные виды тестов:** Test Manager пригоден как для мануальных сценариев, так и для хранения ссылок на автоматизированные тесты. Например, для автотестов можно сохранять ссылки на исходный код или ID сценария, чтобы объединить отчеты.

Bug Tracker (трекер дефектов)

Bug Tracker – встроенная система отслеживания ошибок и дефектов, тесно интегрированная с остальными модулями: - **Регистрация багов:** тестировщики и другие участники команды могут заводить баг-репорты с указанием описания проблемы, шагов воспроизведения, приоритетов и вложением скриншотов/логов. Поля бага настраиваемые (тип, степень влияния, компонент и т.д.). - **Статусы и workflow:** баг-трекер поддерживает типовой жизненный цикл дефекта – Новые, В работе, На проверке, Закрыты и т.д. Можно настраивать workflow под процессы команды (например, добавлять статус “На анализе” или “Отклонено”). При изменении статуса автоматически отправляются уведомления заинтересованным участникам. - **Связь с тестами:** ключевое отличие – баги связаны с результатами тестов. Если дефект найден во время прогона тестов, система фиксирует связь бага с конкретным тест-кейсом и прогоном. Это позволяет в будущем быстро проверить, покрыт ли баг автотестами и в каком регрессе он был обнаружен. Платформа поддерживает заведение дефекта прямо из окна тест-рана ⁴ – при провале теста инженер одним кликом создает баг, а все данные (шаги, ожидание, фактический результат) прикрепляются автоматически. - **Поиск и фильтрация:** удобный поиск по багам (по ключевым словам, статусам, ответственным). Фильтры позволяют сформировать, например, список открытых критических багов перед релизом или все баги, найденные в определенном модуле приложения. - **Управление приоритетами и распределение:** менеджеры могут назначать ответственного за исправление (разработчика), устанавливать приоритет и сроки. Система поддерживает рассылку уведомлений и комментарии в баг-репортах для обсуждения проблемы. - **Отчеты по дефектам:** модуль предоставляет базовые отчеты, такие как количество открытых/закрытых багов по проекту, среднее время жизни дефекта, дефекты по приоритетам и др. Эти метрики интегрируются и в общий дашборд качества.

API Test Workspace (среда тестирования API)

API Test Workspace – модуль, предназначенный для тестирования API (REST/SOAP/GraphQL) без необходимости покидать платформу: - **Конструктор API-запросов:** пользователи могут создавать коллекции API-тестов, аналогично Postman. Предоставляется UI для настройки HTTP-запросов (метод, URL, заголовки, тело, параметры) и определения проверок (assertions) на ответ – например, код ответа 200, содержание определенного поля в JSON и т.д. - **Хранение и версионирование:** каждый API-тест хранится в проекте и может быть объединен в коллекции или сценарии последовательных запросов. Поддерживается параметризация – возможность задавать переменные (например, baseURL, токены) и окружения (Dev, Staging, Prod), чтобы повторно использовать тесты против разных сервисов. - **Запуск и результаты:** API-тесты можно запускать вручную из UI или автоматически в рамках тест-рунов. Платформа отправляет запросы и фиксирует результаты (ответной код, время ответа, результат проверки assertions). Результаты представляются в удобном виде: показывается каждый запрос, его ответ и прошли ли проверки. - **Интеграция с Test Manager:** API-тесты могут быть связаны с тест-кейсами в Test Manager. Например, ручной тест-кейс “API отвечает корректно на запрос X” может иметь ссылку на автоматический API-тест. Это позволяет отразить статус автотеста прямо в отчете тест-рана. - **Библиотека и повторное использование:** Workspace позволяет хранить часто используемые настройки (шаблоны запросов, авторизационные токены). Реализовано хранение коллекций – наборов API-тестов, которые можно запускать пачкой (аналог Postman Collection Runner). В будущем планируется совместимость с форматами Postman Collections для импорта/экспорта. - **Мониторинг и расписание:** (MVP) В базовой версии предусмотрен ручной запуск. В перспективе модуль позволит планировать автоматические запуски API-тестов по расписанию (например, nightly builds) и мониторить доступность API.

UI Automation Hub (хаб автоматизации UI)

UI Automation Hub – модуль, обеспечивающий запуск и управление автоматизированными UI-тестами (web и мобильных приложений) в облаке: - **Интеграция с фреймворками:** Hub не привязан жестко к конкретному инструменту автоматизации – поддерживается интеграция с популярными фреймворками (Selenium/WebDriver, Cypress, Playwright, Appium и др.). Авто-тесты разрабатываются инженерами как обычно, а платформа предоставляет инфраструктуру для их удаленного запуска. - **Облачные агенты выполнения:** TestCloud Suite включает **агентов** – исполняющие узлы, на которых запускаются автотесты. Агенты могут быть развернуты в облаке платформы или установлены on-premise (например, в инфраструктуре клиента для тестирования внутренних систем). Через UI Automation Hub пользователи отправляют задания на выполнение тестов, которые ставятся в очередь и распределяются на доступные агенты. - **Поддержка разных сред:** Авто-хаб позволяет конфигурировать окружения запуска – например, разные браузеры (Chrome, Firefox, Safari) или разные устройства/эмulateры. Платформа интегрируется с облачными сервисами браузеров и устройств либо использует собственный Selenium Grid. QA-инженер может выбрать, в каком окружении выполнить тест-сьют. - **Сбор результатов и логов:** при выполнении автотеста агент отправляет результаты обратно в платформу. В UI отображается статус прогона (в процессе/завершен), и по завершении собираются подробные логи: какие тесты прошли/провалились, скриншоты экранов при ошибках, логи консоли, время выполнения каждого шага. Эти артефакты сохраняются в **Объектном хранилище** платформы и привязываются к сущности **Test Run**. - **Управление тест-сьютами:** через UI Automation Hub можно запускать как отдельные тесты, так и целые сьюты (наборы) автотестов. Реализовано распределение нагрузки: например, если нужно запустить 1000 тестов, несколько агентов разделят этот набор между собой (параллелизация). - **Обратная связь и алерты:** результаты автотестов автоматически обновляют дашборд качества. При падении критических тестов могут отправляться уведомления (например, в Slack или почтой) ответственным за продукт.

Разработчики получают быстрый фидбек о регрессиях. - **Ручное триггерение и CI/CD:** MVP предоставляет веб-интерфейс для ручного запуска наборов автотестов. Кроме того, есть API для интеграции с CI/CD: например, после деплоя нового билда в тестовое окружение можно программно вызвать запуск соответствующего тестового прогона через API Hub. В следующих версиях планируется готовый плагин для Jenkins/GitLab CI, упрощающий эту интеграцию.

Reporting & Analytics (аналитика и отчётность)

Reporting & Analytics – модуль отчетности, объединяющий ключевые метрики качества и прогресса тестирования в наглядных дашбордах: - **Дашборд качества:** на главной панели проекта отображаются основные показатели: процент пройденных/проваленных тестов в текущем цикле, число открытых дефектов (с разбивкой по приоритетам), покрытие требований тестами и т.д. Менеджмент одним взглядом видит статус качества перед релизом. - **Стандартные отчеты:** платформа генерирует отчеты по различным разрезам. Например, **Test Run Report** – детальная сводка прогона (сколько тестов прошло/упало, какие баги заведены) ⁵; **Defect Density** – отчет о дефектах (количество дефектов на функцию или на 100 тест-кейсов); **Coverage Report** – соответствие протестированных функций заявленным требованиям. Отчеты можно просматривать в веб-интерфейсе или выгружать в PDF для рассылки. - **Метрики продуктивности и качества:** встроены метрики для оценки эффективности QA-процесса ⁵. Среди них: *процент автоматизации* (доля тест-кейсов, покрытых автотестами), *скорость тестирования* (например, среднее время прогона регрессии), *метрики дефектов* – среднее время на исправление багов, коэффициент переоткрытых дефектов, *метрики команды* – количество тестов, проводимых одним инженером в единицу времени, и др. Эти показатели позволяют выявлять узкие места и улучшать процессы. - **Кастомизация и экспорты:** пользователь может настраивать виджеты на дашборде – например, добавлять график «Дефекты по приоритетам за последние 4 недели» или «Тренд покрытия требований». Все данные доступны через API, что позволяет строить кастомные отчеты вне системы. Также поддерживается интеграция с BI-инструментами: например, выгрузка сырых данных о тестах и багах в формате CSV для анализа в Tableau/PowerBI. - **Уведомления и рассылки:** модуль отчетности умеет по расписанию отправлять выбранные отчеты заинтересованным лицам. Например, еженедельный отчет о качестве продукта – с кратким суммари: сколько новых багов, сколько закрыто, текущий процент прохождения тестов и т.д. Это особенно полезно для информирования стейкхолдеров и руководства, которые могут не заходить в систему ежедневно.

Роли пользователей и разграничение прав

В TestCloud Suite предусмотрена ролевая модель доступа, позволяющая разграничить права разных категорий пользователей. Основные **роли** и их возможности: - **Администратор платформы:** полные права на уровне организации/аккаунта. Админ управляет пользователями (приглашение новых, назначение ролей), настраивает глобальные параметры (например, интеграцию с внешними инструментами, биллинг). Администратор видит все проекты и может редактировать настройки проектов. - **Менеджер QA / Тест-менеджер:** отвечает за управление тестированием внутри одного или нескольких проектов. Может создавать и удалять проекты, управлять настройками проекта. В рамках проекта менеджер QA имеет все привилегии: создание/редактирование тест-кейсов и наборов, планирование тест-ранов, просмотр всех дефектов, распределение задач по инженерам. Также менеджеру доступны аналитические отчеты для принятия решений. - **QA-инженер / Тестировщик:** основной пользователь системы, выполняющий тесты. Тестировщик может создавать и редактировать тест-кейсы (если ему делегировано), запускать прогоны тестов, отмечать результаты (пометить тест как пройденный/не пройденный), заводить баги. Он видит только проекты, к которым ему предоставлен доступ, и в этих проектах – только функционал, относящийся к тестированию. Например, тестировщик не

может изменять настройки проекта или удалять чужие тест-кейсы без соответствующих прав. - **Разработчик:** в платформе может иметь роль наблюдателя или участника для работы с баг-трекером. Разработчику предоставляется доступ к модулю баг-трекера в пределах проекта – он может просматривать баг-репорты, комментировать их, переводить в статус “Исправлено”. При необходимости разработчику могут быть даны права запускать определенные автотесты (например, прогнать регресс перед выкаткой фикса). Но, как правило, разработчики не меняют тест-кейсы и не помечают результаты тестов. - **Просмотр / Гость:** для стейкхолдеров, клиентов или менеджеров проектов может предоставляться роль с правами только на чтение. Такие пользователи могут просматривать дашборды, отчеты, состояние тестирования и дефектов, но не вносить изменений. Это удобно, чтобы, например, заказчик мог отслеживать качество продукта в реальном времени, не рискуя случайно что-то отредактировать.

Гибкая настройка прав позволяет адаптироваться под процессы разных команд. Например, можно настроить роль “Team Lead” с комбинированными правами менеджера и тестировщика в рамках одного проекта. Каждое действие (создание/удаление теста, переход бага в определенный статус, запуск авто-теста) контролируется системой прав. Благодаря этому обеспечиваются безопасность и упорядоченность: каждый пользователь видит и делает только то, что ему положено.

Ключевые метрики качества

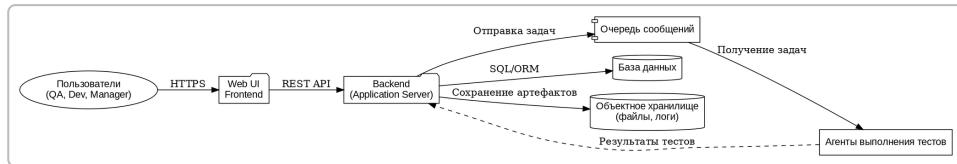
TestCloud Suite ориентирован не только на управление тестами, но и на измерение качества продукта и эффективности тестирования. Платформа автоматически собирает и вычисляет различные **метрики качества**, доступные в отчетах: - **Прогресс тестирования:** процент выполненных тест-кейсов из запланированных. Отчет “Progress” показывает, сколько тестов пройдено успешно, провалено или заблокировано в рамках прогона или релиза. Эта метрика дает понимание, готова ли функция к релизу. - **Покрытие тестами (Test Coverage):** сколько требований или функциональных областей покрыто тест-кейсами и/или автоматизированными тестами. Требования можно отмечать как покрытые, если на них есть хотя бы один связанный тест-кейс. Дашборд показывает, если есть непокрытые требования, что может сигнализировать о рисках. - **Показатели дефектов:** - *Density* (плотность дефектов): количество багов на определенную единицу (на тысячу строк кода или на функциональный модуль, или на 10 тест-кейсов) – помогает определить наиболее проблемные области приложения. - *Defect Leakage*: процент дефектов, ушедших в продакшн (нашедшихся после релиза) относительно дефектов, найденных до релиза. Чем ниже этот показатель, тем эффективнее тестирование до выпуска. - **Среднее время жизни дефекта:** время от регистрации бага до его закрытия – отражает оперативность исправления. - **Процент переоткрытых дефектов:** сколько багов возвращается на доработку после первоначального фикса (показатель качества исправлений). - **Эффективность тестирования:** - **Автоматизация:** доля автоматизированных тест-кейсов. Например, “70% регрессионных тестов автоматизировано”. - **Пропускная способность команды:** число тест-кейсов, выполняемых одним тестировщиком за день/спринт. Позволяет оценить загрузку и эффективность инженеров. - **Время цикла тестирования:** среднее время от начала тестирования фичи до полного завершения, включая фиксы багов. Сокращение этого цикла говорит об ускорении выпуска продукта. - **Качество тестов:** - *Flakiness Rate*: процент нестабильных автотестов (которые дают то пасс, то фейл без кода изменений) – метрика, помогающая улучшать надежность тестовой базы. - *Requirement Traceability*: (при наличии модуля требований) – отчет о том, насколько требования покрыты тестами и как много дефектов на каждое требование. Это связывает качество непосредственно с бизнес-требованиями.

Многие из указанных метрик отображаются в **модуле Reporting & Analytics**. Например, в разделе *Reporting* доступны метрики прогресса, результатов тестирования, статистики по

проектам и продуктивности команды⁵. Анализ этих показателей позволяет постоянно улучшать процесс QA: находить узкие места, обосновывать необходимость автоматизации, демонстрировать ценность тестирования для бизнеса.

Кроме того, TestCloud Suite собирает технические метрики автотестов (покрытие кода, время выполнения тестовых наборов, использование ресурсов), которые помогут DevOps-инженерам оптимизировать pipelines. Все метрики обновляются автоматически по мере работы команды, избавляя от необходимости ручного ведения сводных таблиц.

Архитектурная схема



Архитектура платформы TestCloud Suite: фронтенд взаимодействует с бекеном через API, бекенд хранит данные в базе и файловом хранилище, а для выполнения автотестов использует очередь сообщений и агентов.

На схеме выше представлена высокуюровневая архитектура TestCloud Suite. **Frontend** – это веб-интерфейс (одностраничное веб-приложение), через который пользователи взаимодействуют с системой. Он обращается к **Backend** через REST API. Бекенд реализован как набор сервисов (микросервисы либо модульные компоненты монолита), обеспечивающих работу модулей платформы: Test Manager, Bug Tracker, API Workspace, Automation Hub и т.д. Основные компоненты и взаимодействия:

- **Web UI (Frontend)**: отвечает за отображение интерфейса пользователям. Реализован с использованием современных веб-технологий (например, React/Vue). Взаимодействует с бекеном по HTTPS (REST API) для всех операций – загрузка проектов, тест-кейсов, отправка результатов, создание багов и пр. Фронтенд обеспечивает реактивный опыт: обновление данных (например, статуса тест-рана) происходит в реальном времени посредством WebSocket/Long Polling от сервера, чтобы тестировщики сразу видели актуальную информацию.
- **Backend (Application Server)**: серверное приложение, содержащее бизнес-логику всех модулей. Написан на выбранном стеке (например, Node.js/Express или Java/Spring Boot, в MVP – любой удобный для быстрой разработки). Основные функции бекенда:
 - Обработка API-запросов от фронтенда (авторизация, проверка прав, выполнение команд – добавление тест-кейса, поиск багов и т.п.).
 - Управление данными: сохранение и получение информации из базы данных (через ORM или SQL-запросы) – тест-кейсы, баги, пользователи, результаты и т.д.
 - Постановка задач на выполнение автотестов: когда пользователь запускает автоматизированный тест-ран, бекенд формирует задание (содержит, какие тесты запустить, на каких агентах/браузерах) и помещает его в очередь сообщений.
 - Обработка результатов от агентов: бекенд принимает обратные сигналы от агентов (по API или через очередь) о завершении тестов, затем сохраняет результаты и логи, обновляет статус тест-рана.
- **База данных**: централизованное хранилище структурированных данных (реляционная СУБД, например PostgreSQL). В БД хранятся сущности платформы: проекты, пользователи, тест-кейсы, шаги, результаты, дефекты, комментарии и т.д. Использование транзакционной БД обеспечивает надежность хранения и целостность данных (например, ссылки тест-ранов на тест-кейсы, связи багов и проектов).
- **Объектное хранилище файлов**: для больших объектов (скриншоты, вложения, логи тестов) используется объектное хранилище – например, AWS S3 либо собственный сервис хранения файлов. Платформа сохраняет туда все артефакты тестирования: прикрепленные к багам файлы, журналы выполнения автотестов, экспортированные отчеты. В базе данных хранятся только ссылки на

объекты в этом хранилище. Это обеспечивает масштабируемость хранения и быстрый доступ к большим данным. - **Очередь сообщений:** компонент для асинхронного взаимодействия между бекеном и агентами. Реализован, например, на RabbitMQ или Apache Kafka. Когда требуется выполнить автоматические тесты, бекенд публикует сообщение-задачу в очередь (с подробностями: какие тесты, где запускать). Агенты, подписанные на эту очередь, получают задачу (pull) и приступают к выполнению. Очередь обеспечивает декуплинг: бекенд не ждет выполнения тестов, а может продолжать обработку других запросов, и множество агентов могут получать задачи параллельно. - **Агенты выполнения тестов:** отдельные процессы или узлы, которые выполняют автоматизированные тесты. Агент подключается к бекенду (проходит аутентификацию) и подписывается на задания из очереди для определенного проекта/типа тестов. Получив задачу, агент запускает нужные тест-скрипты (например, запускает Selenium тесты) у себя в окружении. По завершении агент отправляет результаты обратно – либо напрямую на REST API бекенда, либо публикует сообщение в ту же/другую очередь. На схеме показано, что агент получает задачи из очереди и отправляет результаты тестов обратно в бекенд (пунктирной стрелкой). Агенты могут масштабироваться горизонтально – добавление новых агентов увеличивает пропускную способность запуска автотестов.

Архитектура рассчитана на облачное развертывание: фронтенд – статически распространяемый на CDN, бекенд и очередь – в облачном сервисе (можно в контейнерах Kubernetes), база и хранилище – управляемые сервисы (AWS RDS, S3 или аналоги). Такое построение обеспечивает масштабируемость (можно отдельно масштабировать бекенд для API-запросов и пул агентов для тест-нагрузки) и отказоустойчивость.

Безопасность данных обеспечивается разграничением по проектам (например, токены доступа агентов ограничены конкретным проектом), шифрованием соединений и хранением резервных копий БД.

ER-диаграмма с основными сущностями

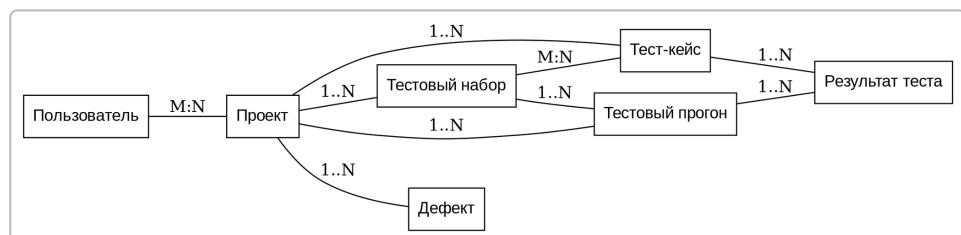


Диаграмма сущностей (ER-модель) платформы TestCloud Suite: показаны основные объекты системы и связи между ними.

На ER-диаграмме выше отражена логическая модель данных TestCloud Suite. Главные сущности и их взаимосвязи следующие: - **Проект:** центральная сущность, объединяющая работу команды по конкретному продукту или компоненту. **Проект** содержит в себе все другие объекты: тест-кейсы, тестовые наборы, прогоны и дефекты, связанные с данным проектом. У одного проекта может быть множество тест-кейсов, наборов, прогонов и багов (отношение 1..N). Пользователи также привязаны к проектам (через роли), при этом один пользователь может состоять в нескольких проектах, а каждый проект обычно имеет нескольких участников (отношение M:N между Пользователем и Проектом). - **Пользователь:** представляет аккаунт участника команды. У пользователя есть атрибуты (имя, email, и т.д.) и назначенная роль в контексте каждого проекта (как описано в разделе ролей). Пользователь может иметь доступ к нескольким проектам, и один проект включает нескольких пользователей (см. связь M:N с Проектом на диаграмме). Ролевые

права определяют, какие действия пользователь может выполнять с объектами проекта. - **Тест-кейс**: единица тестовой работы, описывающая проверку какого-либо функционального требования или сценария. Тест-кейс включает описание шагов, ожидаемый результат и прочие атрибуты (приоритет, тип теста – manual/auto, и др.). В системе каждый тест-кейс привязан к одному проекту. Дополнительно тест-кейс может входить в один или несколько *тестовых наборов* (отношение многие-ко-многим между *Тестовым набором* и *Тест-кейсом* – один тест-кейс может использоваться в нескольких наборах, и один набор включает много тест-кейсов). - **Тестовый набор (Test Suite)**: логическое объединение группы тест-кейсов. Обычно соответствует либо функциональному блоку, либо циклу тестирования (например, "Smoke Tests", "Regression Suite"). Набор принадлежит проекту и содержит определенный список тест-кейсов. С тестовыми наборами тесно связаны прогоны: как правило, *Тестовый прогон* запускается для выполнения всех тест-кейсов, входящих в выбранный тестовый набор. Один набор может быть выполнен много раз (несколько прогонов), поэтому между *Тестовым набором* и *Тестовым прогоном* связь 1..N (набор — один, прогонов этого набора — много). - **Тестовый прогон (Test Run)**: сущность, представляющая факт запуска и выполнения набора тестов в конкретное время. Прогон связан с проектом (например, "Regression Run #5 для проекта X") и, дополнительно, с тестовым набором, который выполняется в рамках прогона. В каждом прогоне фиксируется: кто запускал, когда, какие тест-кейсы входили, какое окружение/версия, и собираются результаты выполнения. У одного проекта может быть множество прогонов тестирования (отношение 1..N). - **Результат теста (Test Result)**: результаты выполнения конкретного тест-кейса в рамках конкретного прогона. Это зависимая сущность, связанная одновременно с *Тестовым прогоном* и *Тест-кейсом*. В ходе каждого прогона для каждого включенного в него тест-кейса формируется запись результата: статус (Passed/Failed/Blocked), время выполнения, отметка об исполнителе (для ручных тестов) или ссылка на лог выполнения (для автотестов). Отношение между прогоном и результатами – один прогон имеет много результатов (1..N), и каждый результат относится ровно к одному прогону; аналогично, один тест-кейс может фигурировать во многих результатах (со временем, по мере повторных запусков), поэтому связь *Тест-кейс* – *Результат теста* также 1..N. **Test Result** обеспечивает детализацию: позволяют отследить историю прохождения каждого тест-кейса в разных запусках. - **Дефект (баг)**: объект баг-трекера, описывающий найденную проблему. Дефект принадлежит проекту (в рамках которого обнаружен). Отношение *Проект* – *Дефект* равно 1..N (в проекте может быть много багов). Прямая связь дефекта с тест-кейсами не обязательна (баг может быть найден по ходу ручного exploratory-тестирования вне конкретного тест-кейса). Однако платформа позволяет устанавливать связь дефекта с результатом теста или конкретным кейсом, если баг выявлен при выполнении этого теста. Такая связь (опциональная) на диаграмме не показана, чтобы не усложнять схему, но в данных она может храниться (например, ID связанного Test Result или Test Case внутри объекта Defect). Это дает возможность быстро проверять, покрыт ли баг соответствующим тестом после исправления (например, через обратную ссылку: баг -> тест-кейс -> автотест).

В совокупности, эта модель данных обеспечивает хранение всех QA-артефактов и их взаимосвязей. Например, можно проследить цепочку: *Дефект* связан с *Результатом теста* (непрошедшим), который относится к определенному *Тест-кейсу* и *Прогону*, а тест-кейс входит в *Тестовый набор*. Это отражает трассируемость от обнаруженной проблемы к конкретному тесту и набору, в котором она выявлена.

Модель легко расширяется: можно добавить сущность "**Требование**" и связать ее с тест-кейсами (для реализации traceability matrix), либо сущность "**Задача**" для интеграции с task-трекерами разработчиков. Однако уже указанных сущностей достаточно для MVP, покрывающего основные потребности QA-команды.

Дорожная карта MVP (90 дней)

Запуск TestCloud Suite планируется через 90 дней (около 12 недель) интенсивной разработки. Ниже приведена подробная **дорожная карта MVP** по неделям, разбитая на этапы:

1. **Неделя 1:** Формирование команды и настройка процессов. Анализ требований и детализация спецификации MVP. Проектирование архитектуры (логической и физической) – выбор технологий для фронтенда и бекенда, выбор СУБД, очереди, набора инструментов для агентов. Разворачивание базовой инфраструктуры: репозиторий кода, CI/CD pipeline (для деплоя dev-версий), базовое облачное окружение. **Результат недели:** готовый план работ, техническое задание, настроенная среда разработки.
2. **Неделя 2:** Реализация базового каркаса приложения. Создание проекта бекенда с шаблоном API и подключением к базе данных. Создание проекта фронтенда с базовым шаблоном (например, каркас React-приложения). Реализация модуля аутентификации и управления пользователями: регистрация/логин, разграничение ролей (пока жестко заданных). Проверка схемы БД, миграции. **Результат:** можно войти в систему под разными ролями, создан первый “проект” в системе вручную через базу.
3. **Неделя 3:** Разработка модуля **Test Manager** – создание и редактирование тест-кейсов. Бекенд: CRUD API для тест-кейсов (создать, получить, обновить, удалить), сущности “TestCase” и “TestSuite” в БД. Фронтенд: интерфейс для списка тест-кейсов проекта, форма создания/редактирования тест-кейса. Добавление базовых полей (название, шаги, ожидаемый результат, приоритет). Реализация иерархии папок/секций для организации тест-кейсов (минимально – поле Section и фильтр по нему). **Результат:** в UI можно заводить тест-кейсы и они сохраняются в базе, отображаются в списке.
4. **Неделя 4:** Разработка функционала **Test Suite** и **Test Run**. Бекенд: API для создания тестовых наборов, привязка тест-кейсов к набору (многие-ко-многим), API для запуска тест-рана (создание сущности TestRun, связанной с TestSuite или произвольным списком тест-кейсов). Фронтенд: интерфейс для управления наборами – создание набора, добавление в него тест-кейсов из существующих; экран запуска прогона – выбор набора и запуск. Заглушка: пока тест-ран выполняется мгновенно и помечает все тесты как “не пройденные” или “пройденные” для имитации (реальную логику выполним позже). **Результат:** пользователь может сгруппировать тесты в набор и инициировать тестовый прогон (пусть даже без фактического исполнения шагов) – в системе фиксируется запись TestRun с привязанными TestResult (статусы можно проставить вручную для имитации).
5. **Неделя 5:** Разработка **Bug Tracker (часть 1)** – базовые операции с дефектами. Бекенд: сущность Defect в БД, API для создания бага, обновления статуса, получения списка/фильтрации. Реализация базового workflow: поля статус (New/Open, In Progress, Resolved, Closed), приоритет, описание, комментарии. Фронтенд: интерфейс подачи баг-репорта – форма с указанием всех полей и прикреплением файла (реализуем загрузку файла в объектное хранилище через бекенд). Страница списка багов проекта с фильтрацией по статусу и поиском. **Результат:** тестировщик через UI может создать баг, приложить скриншот; баг сохраняется и отображается в списке. Можно менять статус (например, разработчик помечает “Исправлено”).
6. **Неделя 6: Интеграция баг-трекера с тест-ранами.** Доработка: при провале тест-кейса в прогоне – возможность сразу создать баг. Реализация на UI: на экране прогона рядом с провальным тестом кнопка “Завести баг”; по нажатию открывается форма, часть полей заполнена (например, название дефекта = название тест-кейса + причина провала). Бекенд: связывание баг-репорта с тест-результатом (новое поле result_id в баге). **Результат:** UX улучшен – тестировщик экономит время, а система хранит связь дефекта с результатом теста. Также на этой неделе завершается интеграция Authentication/Authorization для всех новых API (чтобы права учитывались в багах, тестах и т.д.).

- 7. Неделя 7:** Начало работы над **API Test Workspace**. Реализация упрощенной версии: пользователю предоставляется возможность ввести URL и параметры запроса и выполнить его. Бекенд: подключение библиотеки для HTTP-запросов (например, Axios или native http client) – API endpoint “/api-test/execute” принимает описание запроса (метод, URL, headers, body) и возвращает ответ. Фронтенд: страница “API Workspace” с формой ввода запроса и областью вывода результата (HTTP-код, headers, тело ответа текстом). **Результат:** MVP функциональность – пользователь может вручную выполнять единичные API-запросы и видеть ответ, что закладывает основу для полноценного API-тестирования.
- 8. Неделя 8:** Развитие **API Test Workspace** – сохранение запросов и проверок. Добавление: возможность сохранить настроенный API-запрос как тест-case (в сущность TestCase с пометкой тип=API). Хранение expected result: например, пользователь может задать проверку “код ответа == 200”. Бекенд: расширение модели TestCase для хранения API-полей (или новая сущность ApiTest, связанная с TestCase). Фронтенд: доработка UI – возможность сохранить текущий запрос как тест (с именем, описанием) и добавление секции “API Tests” на странице тест-кейсов. **Результат:** платформа уже может выступать аналогом Postman Collections – можно сохранять и переиспользовать API-тесты.
- 9. Неделя 9:** Начало работы над **UI Automation Hub** – подготовка инфраструктуры агентов. Настройка окружения для запуска агентов: пишется простой скрипт-агент (например, на Python), который может брать задания из очереди. Разворачивание RabbitMQ (если еще не сделано) или использование встроенной очереди на базе БД (для MVP, чтобы не усложнять). Реализация: бекенд получает запрос на запуск автотестов (например, специальный тип TestRun с флагом automated), помещает сообщение в очередь “test_runs”. Параллельно пишется упрощенный агент, который постоянно опрашивает очередь (или таблицу заданий), и при появлении задания мгновенно возвращает “прохождение” или “падение” фиктивного теста. **Результат:** создан каркас системы автотестов – пусть пока агент ничего реального не выполняет, но цепочка коммуникации бекенд -> очередь -> агент -> бекенд налажена и проверена на простом примере.
- 10. Неделя 10: UI Automation Hub (продолжение)** – интеграция с реальным фреймворком тестирования. Решаем, что для MVP будем поддерживать, например, Selenium для web-тестов. Доработка агента: агент может получать команду запустить тесты, и запускает скрипт Selenium (например, набор тестов, написанных на Python/Java, которые находятся в репозитории). В MVP можно ограничиться одним простым скриптом-заглушкой, эмулирующим тест (открыть браузер, зайти на страницу и вернуть pass). Бекенд/Фронтенд: добавление возможности загрузить тест-скрипт или указать репозиторий, откуда агент заберет автотесты – это сложно для 1 спринта, поэтому в MVP делаем упрощенно: агент “зашит” с одним тестом для демонстрации. **Результат:** демонстрационный прогон автотеста: пользователь нажимает “Запустить автотесты” в UI Automation Hub, агент выполняет скрипт (например, открывает test page) и возвращает успех, в UI отображается зеленый статус.
- 11. Неделя 11: Reporting & Analytics** – сбор и отображение метрик. Реализация на бекенде: функции расчета основных показателей (число тестов пройдено/не пройдено в рамках прогона, количество багов в статусах, и т.д.). Создание API endpoints для получения агрегированных данных (например, GET /reports/summary?project_id=X). Фронтенд: разработка дашборда проекта – несколько виджетов с метриками (используем готовые графики или простые числовые индикаторы). Например, “Open Bugs: 5”, “Test Pass Rate: 90%” и график тренда pass rate за последние 5 прогонов. **Результат:** на странице проекта отображается простой дашборд с ключевой статистикой, обновляемой в реальном времени или по запросу.

12. Неделя 12: Завершение MVP, тестирование и подготовка к запуску. Эта неделя отводится на:

- **Тестирование продукта:** команда вручную тестирует все сценарии работы платформы, выявляет баги и стабилизирует систему. Исправляются критические ошибки.
- **Оптимизация и UX-полировка:** улучшается удобство интерфейса (подсказки, загрузка индикаторов, мелкие исправления в верстке). Проверяется поддержка разных браузеров.
- **Безопасность:** проверка правильности разграничения доступа (пользователь не может получить данные чужого проекта через прямой запрос), настройка CORS, HTTPS.
- **Документация:** подготовка краткой документации для пользователей MVP – как пользоваться основными модулями. Возможно, создание нескольких минут видео-демо для демонстрации продукта заинтересованным сторонам.
- **Деплой на тестовый контур:** разворачивание MVP-версии в облаке (стейджинг) и внутреннее бета-тестирование командой.

13. Неделя 13 (резерв): Непредвиденные задачи, исправления по итогам внутреннего тестирования. Подготовка презентационных материалов (слайды, техническое описание) для представления инвесторам/партнерам вместе с демо системы. После устранения последних шероховатостей – выкатывание версии **MVP 1.0** для ограниченного круга пользователей или пилотного проекта.

Этот план рассчитан на агрессивные сроки, но реалистичен благодаря определению четкого скоупа MVP: только основные функции, минимум сложных интеграций. Менеджмент будет строго приоритизировать задачи, чтобы за 3 месяца получить рабочий продукт, готовый показать ценность идеи. Некоторые возможности (полноценная поддержка множества фреймворков, масштабируемость под тысячи тестов и т.д.) отложены на пост-MVP этап (см. следующий раздел), чтобы сфокусироваться на главном.

Модели монетизации и тарифные планы

TestCloud Suite планируется распространять по модели SaaS (Software as a Service) с гибкими тарифными планами. Монетизация будет сочетать **подписку** на использование платформы и дополнительные платные опции для продвинутых возможностей. Ключевые подходы монетизации: - **Подписка по уровню функциональности и размера команды:** предлагаются несколько тарифных планов (от бесплатного базового до корпоративного). Стоимость может зависеть от количества пользователей (инженеров) и предела ресурсов (например, объема хранения, числа запускаемых тестов в месяц). - **Free Trial / Freemium:** для привлечения аудитории предоставляется ограниченный бесплатный план – идеально подходит для маленьких команд или знакомства с продуктом. Бесплатный план может включать все основные модули, но с ограничениями (например, до 5 пользователей, 1 проект, ограниченный объем памяти и без доступа к расширенной аналитике). - **Оплата за пользователя или за проект:** коммерческие планы могут тарифицироваться **per user per month** (например, N долларов за одного активного пользователя в месяц) либо пакетами (до X пользователей за фиксированную цену). Альтернативно или дополнительно – ограничение по числу проектов. - **Usage-based billing (по потреблению):** за использование инфраструктуры автотестов может взиматься плата по факту: например, каждому платному плану дается определенный лимит часов выполнения тестов на облачных агентах, а свыше лимита – почасовая оплата. Так крупные клиенты платят за действительно используемые ресурсы (вычислительные мощности для запуска тестов). -

Enterprise лицензия: для крупных организаций предлагается тариф **Enterprise** с индивидуальными условиями: возможность **on-premise** установки (самостоятельный хостинг платформы в инфраструктуре клиента), расширенная поддержка (выделенный менеджер, SLA на ответы 24/7), кастомизация под нужды клиента. Этот план монетизируется по модели ежегодной корпоративной лицензии или договора. - **Маркетплейс интеграций и дополнений:** в перспективе возможен каталог расширений (например, подключение к дополнительным сервисам AI для анализа тестов) – такие расширения могут продаваться отдельно, либо входить в более дорогие тарифы. - **Консалтинг и обучение:** дополнительный источник – платные услуги по внедрению TestCloud Suite в процессы заказчика, обучение команды, поддержка миграции данных. Хотя продукт SaaS, некоторые клиенты могут оплачивать помощь экспертов для оптимального использования платформы.

Ниже приведена таблица с предложенными тарифными планами (пока ориентировочно, для запуска MVP будут актуальны первые два):

Тарифный план	Возможности	Стоимость
Базовый (Free)	<p>Для небольших команд: до 5 пользователей, 1 проект. Ограничение на число тест-кейсов (например, 200) и багов (100) в базе.</p> <p>Облачные агенты доступны 10 часов в месяц. Базовые отчеты и поддержка через сообщество. Подходит для старта работы с платформой.</p>	\$0 (бесплатно)
Профессиональный	<p>Для растущих команд: до 20 пользователей, до 5 проектов. Неограниченное количество тест-кейсов и багов. Полный доступ ко всем модулям (API, UI Automation). Облачные агенты: 100 часов/мес включено.</p> <p>Расширенные отчеты и дашборды.</p> <p>Стандартная техподдержка (рабочие дни, 8/5).</p>	<p>~\$20 за пользователя в месяц (или пакет 10 пользователей за \$200/мес)</p>
Корпоративный	<p>Для организаций и предприятий: неограниченно пользователей и проектов.</p> <p>Возможность локального развёртывания (on-premise) или выделенного облака.</p> <p>Неограниченные ресурсы агентов (либо договаривается отдельно). Приоритетная поддержка (24/7), персональный менеджер успеха. Дополнительные функции безопасности (SSO, шифрование на стороне клиента) и кастомизация под процессы компании.</p>	Индивидуально (Enterprise договор)

Примечание: Цены указаны условно для примера и подлежат уточнению. Монетизация будет корректироваться после анализа использования MVP и ценности для клиентов. Важная задача – обеспечить привлекательность начальных планов для привлечения QA-сообщества (низкий порог входа), но при этом иметь пути масштабирования монетизации на крупных корпоративных клиентов.

В рамках MVP, акцент делается на отработке ценности продукта, поэтому возможен запуск бесплатной бета-программы для первых пользователей. Это поможет собрать обратную связь. В дальнейшем конверсия на платные планы будет стимулироваться уникальными преимуществами (например, больше параллельных запусков, интеграция с Jira/AzureDevOps в платном плане и т.д.). Также планируется партнерская программа и специальные условия для образовательных учреждений (например, бесплатная лицензия для учебных целей), чтобы повысить популярность платформы.

Будущие модули и направления развития

После успешного MVP и сбора отзывов мы наметили несколько перспективных направлений развития TestCloud Suite. Эти **будущие модули и улучшения** позволят усилить конкурентные преимущества и расширить охват платформы:

- **Модуль Requirements Management:** добавление возможности управлять требованиями и user stories прямо в платформе. Это позволит построить полноценную трассируемость “Requirement → Test Cases → Defects”. Пользователи смогут создавать требования, линковать их с тест-кейсами и отслеживать статус каждого требования (сколько тестов покрыто, есть ли открытые баги). Такой модуль особенно полезен для команд, работающих по стандартам вроде ISO/IEC/IEEE, требующим матрицу трассируемости.
- **Performance & Load Testing:** интеграция инструментов нагрузочного тестирования. Планируется модуль, где QA-инженер может задавать сценарии нагрузки (например, с помощью JMeter, Locust) и запускать их в облаке. Платформа будет визуализировать метрики производительности (черезрут, response time percentiles, etc) и выявлять деградации между версиями. Это сделает TestCloud Suite единым окном для всех видов тестирования, включая performance-тесты.
- **AI-помощник и смарт-автоматизация:** использование методов искусственного интеллекта для ускорения и улучшения тестирования. Например, **AI Test Generation** – автогенерация черновиков тест-кейсов на основе описания функционала или спецификаций (LLM, обученная на тест-дизайне); **Smart Test Maintenance** – система, которая анализирует частые места падения автотестов и подсказывает, где тест нестабилен; **Root Cause Analysis** – ИИ, который по логам автотеста и коду приложения пытается предположить возможную причину бага. Также AI можно использовать для интеллектуального приоритезации тест-наборов перед релизом (подсказывает, какие тесты запустить в первую очередь на основании статистики изменений кода).
- **Mobile Test Hub:** развитие UI Automation Hub в сторону мобильных платформ. Интеграция с облачными фермами устройств (например, BrowserStack, AWS Device Farm) либо собственный модуль управления устройствами. Это позволит запускать автоматические тесты мобильных приложений (Android, iOS) прямо из TestCloud Suite, с аналогичным сбором результатов (скриншоты, видео, логи). Также, возможно, добавить эмулятор в браузере для ручного прогона мобильных приложений и записи сценариев.
- **Более глубокая CI/CD интеграция:** выпуск официальных плагинов для CI-систем (Jenkins, GitLab CI, Azure DevOps). Цель – сделать подключение TestCloud Suite в конвейер максимально простым: например, после деплоя вызывается шаг “Run TestCloud tests” с параметром тестового набора, а результаты автоматически публикуются обратно в pipeline (можно прервать pipeline, если критических тестов > 0). Это повысит привлекательность платформы для DevOps-культур.
- **Расширяемость и маркетплейс:** предоставление публичного API и SDK для разработки плагинов к платформе. Например, сторонние разработчики смогут добавлять новые виджеты отчёtnости, интеграции с экзотическими системами (ALM, тест-менеджмент

специфичный для домена) и продавать их через Marketplace. Это создаст вокруг TestCloud Suite экосистему расширений и ускорит внедрение в узких отраслях.

- **Улучшения UX на основе телеметрии:** планируется сбор анонимной телеметрии использования (с согласия клиентов) – какие функции используются чаще, где пользователи тратят больше времени. На основе этих данных будет проводится улучшение пользовательского опыта: упрощение часто используемых потоков (например, массовый импорт тест-кейсов, быстрые действия типа “клонировать тест-ран”), добавление недостающих горячих клавиш, оптимизация скорости работы интерфейса при больших объемах данных.
- **Более тонкая аналитика качества:** развитие модуля Reporting. Добавление новых отчетов, например “Предиктивная аналитика”: прогноз вероятности того, что релиз пройдет успешно, на основе текущих метрик (количество открытых багов, тренды). Реализация более гибкого конструктора отчетов, чтобы пользователи сами комбинировали метрики и строили графики под свои вопросы.
- **Повышение масштабируемости и отказоустойчивости:** архитектурные улучшения – переход на микросервисы (если MVP был монолитом), возможность горизонтального масштабирования всех компонентов. Реализация multi-tenant архитектуры для эффективного изолированного обслуживания множества клиентов. Введение механизмов резервирования агентов (если один падает, задания автоматом перенаправляются на другой) и авто-скейлинга агентной фермы под текущий объем тестов.
- **Локализация и настройки под отрасли:** перевод интерфейса на другие языки (английский – изначально, возможно поддержка китайского, испанского и др. для выхода на глобальный рынок). Специальные модули под определенные домены: например, поддержка GxP Compliance для фарма (аудит-лог каждого изменения, электронные подписи на тест-результатах), или интеграция с аппаратными тестовыми стендаами в промышленности.

Данные направления будут реализовываться поэтапно, исходя из приоритетов, установленных первыми клиентами и стратегическим видением развития. TestCloud Suite позиционируется как **живущая платформа**, которая будет постоянно улучшаться, чтобы оставаться на острие технологий тестирования.

Ключевые отличия от существующих решений

На рынке существуют как отдельные инструменты для тест-менеджмента и баг-трекинга, так и комплексные решения. Однако TestCloud Suite выгодно отличается по ряду параметров:

- **Единая платформа vs. набор разрозненных инструментов:** В отличие от связки разнородных систем (например, сочетание Jira для багов + TestRail для тест-кейсов + Jenkins для автотестов), TestCloud Suite предоставляет все эти возможности в одном продукте. Это устраняет проблемы интеграции и делает опыт пользователя более цельным ¹. QA-команда работает в одном окне, без постоянного переключения контекста, а данные (тесты, баги, результаты) сразу взаимосвязаны.
- **Глубокая интеграция модулей:** Даже существующие комплексные решения не всегда обеспечивают по-настоящему тесную связь между тестами и багами. В TestCloud Suite все компоненты изначально спроектированы для взаимодействия. Например, баг, заведенный из проваленного теста, автоматически содержит шаги воспроизведения и ссылку на тест-кейс – это не нужно настраивать дополнительно, как в сторонних интеграциях ⁴. Dashboard мгновенно отражает новые баги во время прогона, а запуск автотеста легко инициируется прямо из тест-рана.

- **Облачное исполнение тестов:** Многие конкуренты фокусируются либо только на менеджменте (TestRail, Qase и др.), либо только на запуске тестов (Selenium Grid, BrowserStack). Наша платформа объединяет эти подходы: хранение и планирование тестов и их выполнение. Встроенная облачная инфраструктура для автотестов означает, что клиенту не нужно поднимать свой Jenkins или покупать отдельный сервис для запуска тестов – TestCloud Suite “из коробки” позволяет гонять авто-сьюты на нужных браузерах и устройствах.
- **Доступность и скорость внедрения:** Благодаря облачной модели, начать использовать TestCloud Suite можно очень быстро – не требуется сложной установки на серверах клиента (кроме optionalных локальных агентов для специфичных нужд). Интерфейс ориентирован на удобство: знакомые всем понятия (тест-кейсы, баги, Suites) представлены интуитивно. Порог входа ниже, чем у тяжеловесных enterprise-систем (типа Micro Focus ALM/Quality Center), при сопоставимом охвате функциональности.
- **Стоимость владения:** За счёт объединения функциональности нескольких продуктов, TestCloud Suite экономичнее для организации. Вместо покупки лицензий на баг-трекер, test management tool, отдельный хостинг для CI – достаточно одной подписки. Кроме того, модель оплаты “по пользователям” или “по использованию” гибко масштабируется – команда платит только за нужные ресурсы. Для малого бизнеса есть бесплатный план, тогда как многие конкуренты вообще не предлагают бесплатных опций либо имеют ограничения на интеграцию.
- **Современные технологии и AI:** Платформа изначально разрабатывается с учётом новейших трендов. Например, планируемая интеграция AI-ассистентов для генерации тестов или анализа логов – это то, чего нет (или реализовано слабо) в традиционных инструментах. Также акцент на поддержке современных методологий: CI/CD, DevOps, Shift-left testing. TestCloud Suite легко вписывается в Agile-процессы, позволяя тестировщикам создавать тесты одновременно с разработкой и автоматически запускать их при каждом коммите.
- **Ориентация на командную работу:** Мы уделяем внимание коллaborации – совместное редактирование тест-кейсов, упоминания пользователей (@mention) в комментариях к багам, интеграция с мессенджерами (Slack, Microsoft Teams) для оповещений. Многие существующие решения сфокусированы либо на индивидуальной работе тестировщика, либо на отчетности для менеджера. Мы же стараемся учесть интересы всех ролей: и тестировщика (удобство заведения багов и тестов), и разработчика (понятные отчеты о проблемах, минимум лишних логинов в чужие системы), и руководителя (метрики, графики).
- **Локализация и поддержка русского языка:** На данный момент ряд популярных инструментов не имеют русскоязычного интерфейса или поддержки, что затрудняет их использование некоторыми командами. TestCloud Suite изначально доступен на русском (и английском) языке, с локализацией терминологии, а служба поддержки ориентируется в том числе на локальный рынок. Это преимущество для компаний из СНГ и Восточной Европы.

В совокупности, **TestCloud Suite** стремится занять нишу универсальной облачной QA-платформы. В отличие от традиционных решений, продукт предлагает более тесную интеграцию, простоту использования и готовность к будущим технологическим вызовам. Это дает нам уверенность конкурировать как с громоздкими enterprise-системами, так и с узкоспециализированными инструментами, предлагая лучшее из обоих миров в одном пакете.

1 2 3 4 5 Test Case Management Tool: как правильно сделать выбор и не пожалеть об этом / Хабр

<https://habr.com/ru/companies/redmadrobot/articles/248965/>