УДК 004.4’2:004.55

Решетняк Максим Александрович
магистрант
ФГБОУ ВО «Кубанский государственный университет»
Россия, г. Краснодар

АРХИТЕКТУРА И ПРОЕКТИРОВАНИЕ АДАПТИВНЫХ ИНТЕРФЕЙСНЫХ МОДУЛЕЙ ДЛЯ CRM-СИСТЕМ (НА ПРИМЕРЕ AMOCRM)

Аннотация: В работе рассмотрена проблема проектирования адаптивных интерфейсных модулей, встраиваемых в CRM-системы. На примере платформы amoCRM проанализированы ограничения, накладываемые средой исполнения виджетов, и предложена слоистая архитектура клиентского приложения, включающая интеграционный, логический и интерфейсный уровни. Описаны механизмы централизованного кэширования справочных данных, нормализации неоднородных событий и адаптации интерфейса под различные устройства и темы оформления. Сформулированы принципы, которые могут быть перенесены на другие CRM-платформы.

***Ключевые слова*:** CRM, виджет, amoCRM, адаптивный интерфейс, архитектура, React, API, кэширование, нормализация данных, кроссплатформенность.

Введение

Современные CRM-системы давно вышли за рамки простых клиентских баз. Сегодня это сложные многомодульные платформы, объединяющие продажи, маркетинг, складскую логистику и аналитику. Однако ни одна, даже самая развитая, CRM не способна «из коробки» закрыть все специфические потребности

конкретного бизнеса. Решением становятся встраиваемые интерфейсные модули — виджеты, которые расширяют функциональность системы, не затрагивая её ядро.

Платформа amoCRM реализует этот подход через механизм виджетов, подключаемых в заданных областях интерфейса, включая карточки сделок. Разработчик получает формализованную модель встраивания: манифест с описанием виджета, колбэки жизненного цикла, шаблонизатор Twig и REST API для работы с данными [1]. Но, несмотря на наличие SDK, проектирование качественного модуля остаётся нетривиальной задачей. Виджет должен быть одновременно «родным» для CRM по визуальному стилю, достаточно быстрым, чтобы не нарушать рабочий ритм пользователя, и при этом корректно отображаться на разных устройствах — от десктопа до смартфона.

Анализ существующих решений показывает, что многие из них либо ориентированы на одну узкую функцию, либо выполнены как внешние приложения без глубокой интеграции с контекстом сделки. Вопросы построения целостной архитектуры модульного адаптивного виджета, способного объединить аналитику, историю и операционные действия, остаются проработанными недостаточно.

Цель данной статьи — предложить и обосновать архитектурный подход к созданию адаптивных интегрируемых фронтенд-модулей для CRM-систем на примере amoCRM, а также описать ключевые технические решения, обеспечивающие устойчивость, производительность и удобство сопровождения таких модулей.

Анализ среды исполнения виджетов amoCRM

Прежде чем проектировать архитектуру, необходимо понять, в каких условиях будет работать модуль. amoCRM предоставляет разработчику виджетов жёстко определённую, но хорошо документированную модель встраивания [1]. Центральными элементами здесь выступают:

- Манифест — JSON-файл, в котором задаются название виджета, области подключения, параметры настроек и пути к основным ресурсам [2].

- Стандартные колбэки (init, render, bind_actions, onSave) — они определяют, как виджет инициализируется, отрисовывается и взаимодействует с платформой [1].
- Шаблонизатор Twig — рекомендованный механизм для отделения HTML-разметки от исполняемой логики, предотвращающий смешение бизнес-логики и представления. В документации amoCRM прямо указано, что шаблоны лучше отделять от логики виджета и размещать в отдельных twig-файлах, чтобы код не превращался в нагромождённое месиво [3].
- REST API v4 — единообразный интерфейс доступа к сущностям CRM (сделкам, контактам, событиям, статусам воронок и т.д.) [4].

Из этого вытекает несколько архитектурных ограничений. Во-первых, виджет не контролирует всю страницу — он встраивается в конкретную область интерфейса (например, карточку сделки). Во-вторых, получение идентификатора текущей сущности должно быть реализовано с учётом возможных отказов основного механизма: помимо системного контекста, часто приходится прибегать к резервному поиску через DOM-атрибуты [1]. В-третьих, интерфейс модуля обязан быть визуально согласован с активной темой CRM и корректно реагировать на изменение размеров контейнера.

Эти ограничения носят не только технический, но и методологический характер. Они подталкивают к модульному, слоистому построению приложения, при котором интеграционная «обвязка» отделена от бизнес-логики и пользовательского интерфейса.

Требования к архитектуре интегрируемого фронтенд-модуля

С учётом сказанного можно сформулировать ключевые требования к архитектуре разрабатываемого виджета:

1. Совместимость с платформенной моделью. Модуль должен естественно вписываться в жизненный цикл CRM, использовать штатные колбэки и не конфликтовать со стилями системы.

2. Разделение ответственности. Интеграционный слой (работа с API), слой обработки данных и интерфейсный слой должны быть изолированы друг от друга, чтобы изменение одного из них не вызывало каскадных правок.
3. Производительность и минимизация запросов. Данные, которые редко меняются в пределах сессии (названия статусов, имена пользователей, справочная информация), должны кэшироваться. Это снижает нагрузку на API и ускоряет повторные открытия интерфейса.
4. Устойчивость к неоднородным данным. События, поступающие от API, могут иметь различную внутреннюю структуру. Архитектура должна предусматривать нормализацию таких данных до их попадания в интерфейс.
5. Адаптивность и тематическая согласованность. Внешний вид виджета должен динамически подстраиваться под тип устройства (мобильное, планшетное, десктопное) и текущую тему оформления CRM (светлую или тёмную).
6. Удобство сопровождения и расширения. Добавление новых функций или перенос модуля на другую сущность CRM не должны требовать переписывания всей архитектуры.

Предлагаемая архитектура

Опираясь на перечисленные требования, архитектура виджета может быть спроектирована как многоуровневая система. В ней выделяются три основных горизонтальных слоя, над которыми находится компонентный интерфейсный каркас.

1. Интеграционный слой отвечает за все взаимодействия с внешним миром: запросы к API, обработку колбэков CRM, монтирование интерфейсного приложения в DOM-контейнер, созданный через шаблонизатор платформы.
2. Слой обработки данных выполняет нормализацию, преобразование и расчёт производных показателей. Именно здесь сырые ответы API превращаются в структуры, пригодные для отображения.

3. Интерфейсный слой строится на компонентной библиотеке (например, React) и включает переиспользуемые элементы (карточки, таблицы, вкладки, уведомления), которые получают уже подготовленные данные через свойства и не содержат собственной логики запросов.

Дополнительно выделяются сквозные сервисы: менеджер кэша для хранения справочной информации, менеджер темы для определения текущей визуальной схемы и детектор устройства для идентификации типа экрана. Эти сервисы используются на разных уровнях, но их логика централизована.

Такое разделение позволяет свести связанность между частями системы к минимуму. Замена или обновление API, доработка аналитических расчётов или смена фронтенд-библиотеки затрагивают лишь соответствующий слой. На рисунке 1 представлена схема предложенный архитектуры:

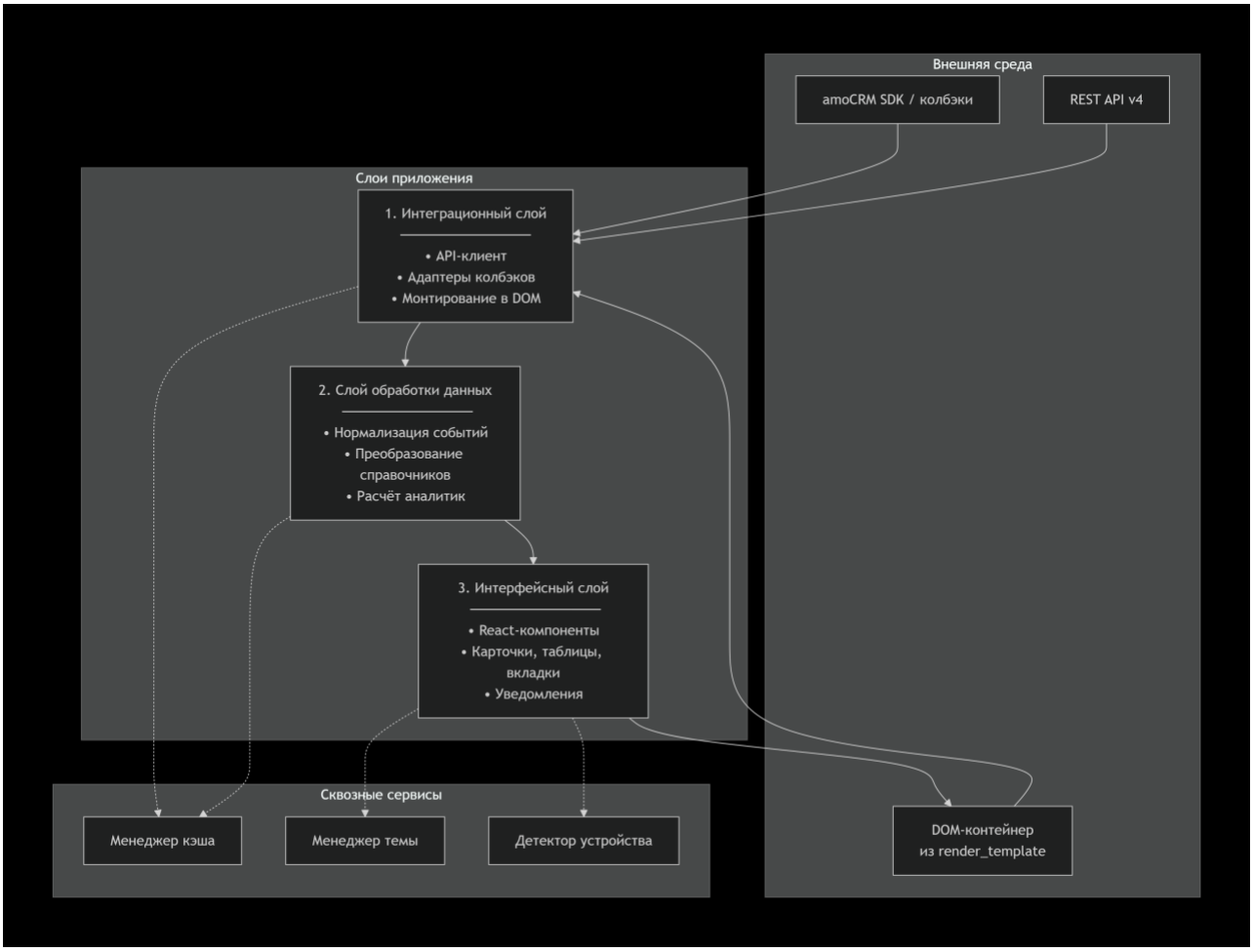

Рисунок 1 – Схема предложенной архитектуры.

Интеграционный слой строится на принципе «единого транспортного уровня». Все запросы к API инкапсулированы в отдельных асинхронных функциях, каждая из которых возвращает только необходимые данные. Благодаря этому интерфейсные компоненты не знают ни о структуре URL, ни о формате ответов сервера.

Централизованная загрузка контекста сделки выполняется одной точкой входа, которая последовательно запрашивает нужные сущности и агрегирует их в единый объект. Это исключает дублирование запросов и позволяет централизованно управлять индикаторами загрузки и обработкой ошибок. Документация amoCRM API v4 предоставляет для этого полный набор методов, включая получение сделки по ID, фильтрацию событий и работу со справочниками статусов [4].

Одной из наиболее критичных для надёжности задач является обработка событий смены статусов. Данные, получаемые через API, могут размещать идентификаторы старого и нового статусов в разных полях в зависимости от версии или контекста вызова. Жёсткая привязка к одному формату сделала бы модуль хрупким.

В архитектуре предусматривается нормализующий слой, который перебирает возможные пути извлечения идентификаторов и приводит их к единому внутреннему представлению. После извлечения идентификаторы преобразуются в читаемые названия статусов с использованием закэшированного справочника. Такой подход гарантирует, что интерфейс всегда работает с единообразной моделью данных, а при изменении формата API правки потребуются только в одном месте.

Значительная часть данных, необходимых для отображения, редко меняется в течение рабочей сессии: названия статусов и воронок, имена пользователей,

сведения о компаниях. Повторная загрузка этих справочников при каждом открытии виджета создавала бы избыточную нагрузку на API и увеличивала время отклика.

Для решения этой проблемы в архитектуру вводится менеджер кэша. Он хранит уже загруженные справочные данные в памяти и предоставляет методы для их чтения, записи и сброса. Интеграционные функции, прежде чем выполнить запрос к API, проверяют наличие данных в кэше и, если они там есть, сразу возвращают готовый результат. При смене текущей сделки кэш не сбрасывается целиком — перезапрашиваются только данные, непосредственно связанные с новой сущностью. Это позволяет поддерживать высокую отзывчивость интерфейса при многократных открытиях виджета.

Заключение

В статье представлен архитектурный подход к построению адаптивных интерфейсных модулей для CRM-систем, рассмотренный на примере amoCRM. Предложенная слоистая структура с разделением на интеграционный, логический и интерфейсный уровни обеспечивает устойчивость к изменениям платформенного API, упрощает сопровождение кода и позволяет независимо развивать как аналитические функции, так и визуальное представление.

Ключевыми техническими решениями, повышающими практическую ценность архитектуры, являются: централизованное кэширование справочных данных, нормализация неоднородных событий смены статусов, а также встроенные механизмы адаптации интерфейса под тип устройства и тему оформления CRM. Данные решения могут быть использованы в качестве методической основы при разработке аналогичных виджетов для различных CRM-платформ.

Список литературы:

1. amoCRM. Механика работы виджета [Электронный ресурс] // Официальная документация amoCRM. — URL:

https://www.amocrm.ru/developers/content/web_sdk/mechanics (дата обращения: 10.05.2026).
2. amoCRM. Структура виджета [Электронный ресурс] // Официальная документация amoCRM. — URL: https://www.amocrm.ru/developers/content/integrations/structure (дата обращения: 10.05.2026).
3. amoCRM. JS-Виджет [Электронный ресурс] // Официальная документация amoCRM. — URL: https://www.amocrm.ru/developers/content/integrations/script_js (дата обращения: 12.05.2026).
4. amoCRM. Сделки (API v4) [Электронный ресурс] // Официальная документация amoCRM. — URL: https://www.amocrm.ru/developers/content/crm_platform/leads-api (дата обращения: 14.05.2026).