КРОСС-ПЛАТФОРМЕННОЕ МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ КОМПЛЕКСНОГО УПРАВЛЕНИЯ МЕРОПРИЯТИЯМИ

Савинич Т.М., студент

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Куприянова Д.В. – маг. техн. наук, старший преподаватель

В данной работе рассматривается проектирование и программная реализация мобильного приложения для автоматизации управления мероприятиями, предназначенного для организаторов событий различного масштаба – от частных торжеств до корпоративных конференций. Основной акцент сделан на построении модульной клиент-серверной архитектуры с разделением ответственности между слоями бизнес-логики, авторизации и уведомлений, а также на обеспечении масштабируемости системы через гибкую модель подписок и серверную проверку пользовательских лимитов.

Современный рынок цифровых инструментов для организации мероприятий характеризуется фрагментированностью решений: управление гостями, задачами, командой и коммуникациями зачастую осуществляется в разрозненных приложениях, что порождает проблему дублирования данных и потери контекста. Традиционные подходы, основанные на использовании табличных редакторов, мессенджеров и бумажных чек-листов, не способны обеспечить единое информационное пространство мероприятия: централизованный учёт гостей с их статусами подтверждения, контроль выполнения задач с назначением ответственных и сроков, а также оперативное информирование всех участников команды о текущем состоянии подготовки. Недостаточная интеграция таких инструментов приводит к снижению эффективности организационного процесса, увеличению количества ошибок и потере критически важной информации на этапах планирования. В связи с этим актуальной научно-практической задачей является создание архитектуры мобильного приложения, способной объединить все аспекты управления мероприятием в едином контексте и обеспечить оперативное взаимодействие между участниками в режиме реального времени.

Анализ существующих аналогов (Eventbrite, Trello, Google Sheets в связке с мессенджерами) показал, что ни одно из представленных решений не обеспечивает полного покрытия жизненного цикла мероприятия в рамках единого мобильного приложения. Платформы продажи билетов ориентированы на публичные мероприятия и не предоставляют инструментов для внутренней координации команды. Универсальные менеджеры задач лишены доменной специфики: понятий «гость», «приглашение», «рассадка» или «шаблон мероприятия». Таким образом, разработка специализированного решения, объединяющего функции управления задачами, гостевыми списками, командной работой и коммуникациями в контексте конкретного мероприятия, является обоснованной.

Разрабатываемое решение основано на клиент-серверной архитектуре с применением REST API и токенной авторизации, что позволяет строго разделить логику представления и обработки данных.

Серверный слой (REST API). Центральным узлом обработки данных выступает сервер на основе платформы Node.js с фреймворком Express.js [1]. Каждое взаимодействие мобильного клиента с системой (создание мероприятия, управление задачами, отправка приглашений, обмен сообщениями) инкапсулируется в HTTP-запрос и направляется на соответствующий маршрут API. Серверная часть содержит 44 файла маршрутов, покрывающих все функциональные модули приложения, и набор вспомогательных утилит (usersHelper, tasksHelper, emailHelper, paymentHelper), инкапсулирующих повторно используемую бизнес-логику. Это позволяет реализовать чёткое разделение ответственности: сервер выполняет проверку входных данных, проверку прав доступа через middleware-слой и бизнес-логику, а клиент отвечает исключительно за представление и пользовательское взаимодействие. Использование пула соединений MySQL2 в режиме Promise обеспечивает асинхронную обработку запросов к базе данных без блокировки основного потока событий (Event Loop) Node.js, что критично при одновременной работе множества пользователей. Middleware-компонент authMiddleware выполняет верификацию JWT-токена при каждом защищённом запросе, извлекая идентификатор пользователя из поля sub токена и помещая его в объект запроса для дальнейшей обработки маршрутами.

Реляционное ядро (MySQL). Для моделирования предметной области применяется реляционная база данных MySQL, структура которой отражает доменную модель системы управления мероприятиями. Пользователи, мероприятия, задачи, гости, приглашения и шаблоны представляются в виде связанных таблиц с внешними ключами, обеспечивающими ссылочную целостность. Ключевым элементом архитектуры данных является связующая таблица user_event_mapping, реализующая ролевую модель доступа: каждому участнику мероприятия назначается роль с определённым набором прав (управление гостями, создание и назначение задач, приглашение новых членов команды, отправка сообщений). Данная модель позволяет одному пользователю участвовать в нескольких мероприятиях с различными ролями, а одному мероприятию – иметь несколько организаторов с дифференцированными правами доступа. В отличие от неструктурированных и документо-

ориентированных хранилищ, реляционная модель обеспечивает целостность данных через механизмы транзакций и каскадных ограничений (ON DELETE CASCADE, ON UPDATE CASCADE), что критично для системы, где несколько организаторов параллельно вносят изменения в одно мероприятие. Подключение к базе данных организовано через модуль config/db.js, формирующий пул соединений с параметрами, вынесенными в конфигурационный файл, что упрощает переключение между средами разработки и продакшена.

Слой авторизации и безопасности. Система авторизации построена на базе JSON Web Tokens (JWT) и реализует стандартную схему «access token + refresh token». При первичной аутентификации пользователь передаёт учётные данные (email и пароль), пароль верифицируется с помощью алгоритма bcrypt, после чего сервер генерирует подписанный JWT-токен с использованием секретного ключа, хранящегося в конфигурационном файле. Каждый последующий запрос к защищённым эндпоинтам проходит через middleware authMiddleware, который декодирует токен, проверяет его корректность и срок действия, а затем извлекает идентификатор пользователя для дальнейшей авторизации на уровне бизнес-логики. На мобильном клиенте управление сессиями реализовано через модуль SessionManager, использующий React Native Keychain для безопасного хранения токенов в защищённом хранилище операционной системы (Keychain на iOS, Keystore на Android), что предотвращает компрометацию учётных данных при несанкционированном доступе к файловой системе устройства [2].

Слой уведомлений (Push Notifications). Для обеспечения оперативного информирования пользователей о значимых событиях применяется сервис Firebase Cloud Messaging (FCM) [3]. Серверный модуль FirebaseNotificationManager инкапсулирует логику формирования и отправки push-уведомлений. При возникновении триггерных событий (новое приглашение в команду, изменение статуса задачи, новое сообщение в чате мероприятия, подтверждение присутствия гостем) серверная часть формирует структурированное push-уведомление с указанием типа события и идентификатора связанной сущности, после чего направляет его на зарегистрированные FCM-токены устройств целевых пользователей. На стороне мобильного клиента модуль NotificationHandler управляет регистрацией FCM-токена, обработкой входящих уведомлений в режимах foreground и background, а также навигацией к соответствующему экрану приложения при нажатии на уведомление. Библиотека Notifee обеспечивает отображение локальных уведомлений с расширенным функционалом: кастомные каналы уведомлений, счётчик непрочитанных сообщений (badge count) и группировка уведомлений по мероприятиям. Помимо push-уведомлений, система реализует email-оповещения через модуль emailHelper с использованием библиотеки Nodemailer и SMTP-протокола, что обеспечивает доставку приглашений и важных уведомлений пользователям, временно не имеющим доступа к мобильному приложению.

Слой монетизации и масштабирования. Система предусматривает три тарифных плана подписки – Basic, Advanced и Premium, – управляемых через сервис RevenueCat с нативной интеграцией в App Store и Google Play. Каждый тарифный план определяет лимиты использования ключевых сущностей системы: количество одновременных мероприятий (от 5 до 100), задач на мероприятие (от 100 до 10 000), гостей (от 50 до 5 000) и членов команды (от 5 до 500). Middleware-компонент checkUserLimits осуществляет двухуровневую проверку лимитов на стороне сервера: первый уровень проверяет соответствие базовому тарифу, второй – премиальному. При превышении лимитов сервер возвращает структурированный ответ с кодом ошибки и информацией о текущем тарифе, позволяя клиенту отобразить пользователю предложение об обновлении подписки. Обработка платежей за разовые услуги выполняется через платёжный шлюз BePaid с поддержкой мультивалютности (USD, BYN, RUB, EUR), логика которого инкапсулирована в модуле paymentHelper. Такой подход позволяет гибко масштабировать функциональные возможности приложения без изменения базовой архитектуры и кодовой базы.

Слой шаблонов и повторного использования. Важной архитектурной особенностью системы является модуль шаблонов, позволяющий пользователям сохранять структуру мероприятия (набор задач, категории гостей, роли команды) в качестве повторно используемого шаблона. При создании нового мероприятия на основе шаблона система автоматически генерирует все связанные сущности, что существенно сокращает время подготовки для организаторов, регулярно проводящих однотипные мероприятия. Данный механизм реализован через серверные маршруты, копирующие структуру шаблона с генерацией новых идентификаторов и привязкой к текущему пользователю.

Техническая реализация. Серверная часть системы реализована на платформе Node.js с применением фреймворка Express.js, обеспечивающего высокую скорость маршрутизации HTTP-запросов и поддержку цепочек middleware-обработчиков. Сервер запускается на порту 3001 с настроенной политикой CORS для обработки кросс-доменных запросов от мобильного клиента. Авторизация реализована через JWT-токены (библиотека jsonwebtoken) с хешированием паролей алгоритмом bcrypt. Для обработки загрузки медиафайлов (аватары пользователей, изображения мероприятий) используется библиотека Multer с настраиваемыми ограничениями по размеру и типу файла, а для последующей оптимизации изображений – библиотека Sharp, выполняющая компрессию и масштабирование без потери качества. Статические файлы загрузок обслуживаются через

Express.static из директории /uploads. Поддержка мультиязычности интерфейса (русский и английский языки) реализована через библиотеку i18n, определяющую язык пользователя по заголовку Accept-Language и подставляющую соответствующие строковые ресурсы в серверные ответы. Развёртывание серверной части осуществляется через CI/CD-пайплайн на основе GitHub Actions с автоматической доставкой по протоколу FTP на хостинг CloudLinux, что обеспечивает непрерывную интеграцию и сокращает время доставки обновлений до минут.

Клиентское приложение реализовано на фреймворке React Native версии 0.72.14, что позволяет использовать единую кодовую базу JavaScript/TypeScript для платформ iOS и Android, обеспечивая при этом нативный пользовательский опыт и доступ к аппаратным средствам устройства. Навигация построена на React Navigation 6.x с комбинацией Stack-навигатора для иерархических экранов и BottomTab-навигатора для основных разделов приложения. Глобальное состояние текущего мероприятия управляется через React Context API (EventContext), обеспечивая доступ к данным события из всех связанных экранов без необходимости передачи параметров через пропсы. Для хранения несущественных данных сессии используется AsyncStorage, а учётные данные (JWT-токены) защищаются через React Native Keychain с аппаратным шифрованием. Сетевой слой клиента реализован через библиотеку Axios с централизованной конфигурацией базового URL и интерцепторами для автоматического добавления токена авторизации к каждому запросу. Для улучшения пользовательского опыта применяются анимированные индикаторы загрузки на основе Lottie, визуализация статистики через react-native-chart-kit, а также интерактивные модальные панели с использованием @gorhom/bottom-sheet. Модуль Urls.js содержит 52 определения API-эндпоинтов, обеспечивая централизованное управление адресами серверных ресурсов и упрощая миграцию между средами.

Результаты и выводы. Предложенный подход к построению системы управления мероприятиями позволяет достичь следующих результатов:

– сокращение времени организации мероприятия за счёт объединения управления задачами, гостями, командой и коммуникациями в едином приложении с общим контекстом события, а также за счёт механизма шаблонов, позволяющего повторно использовать структуру ранее проведённых мероприятий;

– повышение координации между организаторами благодаря ролевой модели доступа с дифференцированными правами, встроенному мессенджеру с push-уведомлениями и системе отслеживания статусов задач в реальном времени, что исключает информационные потери на этапе подготовки;

– обеспечение масштабируемости платформы через тарифную систему подписок с двухуровневой серверной проверкой лимитов, позволяющую обслуживать как единичные мероприятия с десятками гостей, так и масштабные проекты с тысячами участников и сотнями задач без изменения архитектуры системы;

– снижение порога входа для организаторов за счёт кроссплатформенности мобильного приложения (iOS и Android из единой кодовой базы), мультиязычного интерфейса и интуитивной навигации с контекстной привязкой всех действий к выбранному мероприятию.

Таким образом, применение модульной клиент-серверной архитектуры с разделением ответственности между REST API на базе Node.js/Express.js, реляционной базой данных MySQL с ролевой моделью доступа и сервисом push-уведомлений Firebase Cloud Messaging является эффективным решением для создания современных интеллектуальных систем управления мероприятиями.

Разработанная архитектура позволяет автоматизировать процесс планирования и координации мероприятий, снизить нагрузку на организаторов за счёт делегирования задач и автоматического контроля их выполнения, обеспечить безопасность пользовательских данных через JWT-авторизацию и аппаратное шифрование учётных данных, а также повысить вовлечённость команды посредством предоставления актуальной и контекстно-зависимой информации о ходе подготовки события через многоканальную систему уведомлений.

Список использованных источников:

1. Express.js Documentation [Электронный ресурс]. — Режим доступа: https://expressjs.com. – Дата доступа: 10.03.2026.

2. React Native Documentation [Электронный ресурс]. – Режим доступа: https://reactnative.dev. – Дата доступа: 15.03.2026.

3. Firebase Cloud Messaging [Электронный ресурс]. — Режим доступа: https://firebase.google.com/docs/cloud-messaging. – Дата доступа: 20.03.2026.