# 3 РАЗРАБОТКА ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ ОТСЛЕЖИВАНИЯ И ВИЗУАЛИЗАЦИИ ДЕЙСТВИЙ ПОЛЬЗОВАТЕЛЕЙ В ИГРАБЕЛЬНОЙ РЕКЛАМЕ

## 3.1 Структура программного комплекса

Структура программного комплекса организована с учетом принципов надежности, масштабируемости и модульности, что позволяет эффективно отслеживать действия пользователей в играбельной рекламе и визуализировать их в реальном времени.

Ключевым элементом является легковесный сервис, который отвечает исключительно за прием трекинг-запросов. Его проектирование ориентировано на максимальную стабильность и отказоустойчивость, поэтому он не содержит ни дополнительного функционала, ни избыточных зависимостей, что гарантирует возможность фиксировать каждый входящий запрос в любой момент времени без риска потери данных.

Код, реализующий функциональность трекинг-сервиса, интегрируется непосредственно в монолитный дашборд, что упрощает архитектуру и обеспечивает тесную связь с другими компонентами системы. Архитектура дашборда организована как единое серверное приложение, код которого состоит из множества отдельных сервисов, каждый из которых отвечает за обслуживание эндпоинтов, обработку событий, поступающих с клиентской стороны, выполнение специализированного кода при запуске системы и выполнение регулярных задач, необходимых для поддержания актуальности и целостности данных. Такая структура позволяет изолировать бизнес-логику, связанную с различными функциональными возможностями, а также централизовать управление процессами внутри комплекса.

Взаимодействие между серверной и клиентской частями реализовано посредством постоянного *websocket*-соединения, что обеспечивает двустороннюю связь и мгновенное обновление информации. Это позволяет клиентской части получать данные в режиме реального времени, что является критически важным для оперативного мониторинга и анализа действий пользователей. Соединение по протоколу *websocket* обеспечивает минимальную задержку и высокую производительность при обмене данными, что особенно актуально в условиях интенсивной работы системы трекинга [16].

Клиентская часть дашборда оснащена функционалом для настройки параметров трекинга, визуализации собранных данных и формирования отчетов. Здесь пользователю предоставляется возможность не только наблюдать за поступающей информацией, но и самостоятельно управлять параметрами, влияющими на сбор данных, а также загружать подробные отчеты для последующего анализа. Такой подход позволяет обеспечить

гибкость и адаптивность системы под различные требования и сценарии использования.

Кроме того, сервер дашборда выполняет задачу по сборке финальных билдов играбельной рекламы. В процессе сборки, если включена опция трекинга, в билд автоматически интегрируется специальный код, который отвечает за отправку трекинг-запросов непосредственно на легковесный сервис. Это позволяет обеспечить непрерывную цепочку обработки данных от момента взаимодействия пользователя с рекламой до отображения информации в дашборде, гарантируя, что все действия фиксируются и могут быть проанализированы в последующем.

В общих чертах структура программного комплекса представлена на рисунке 3.1

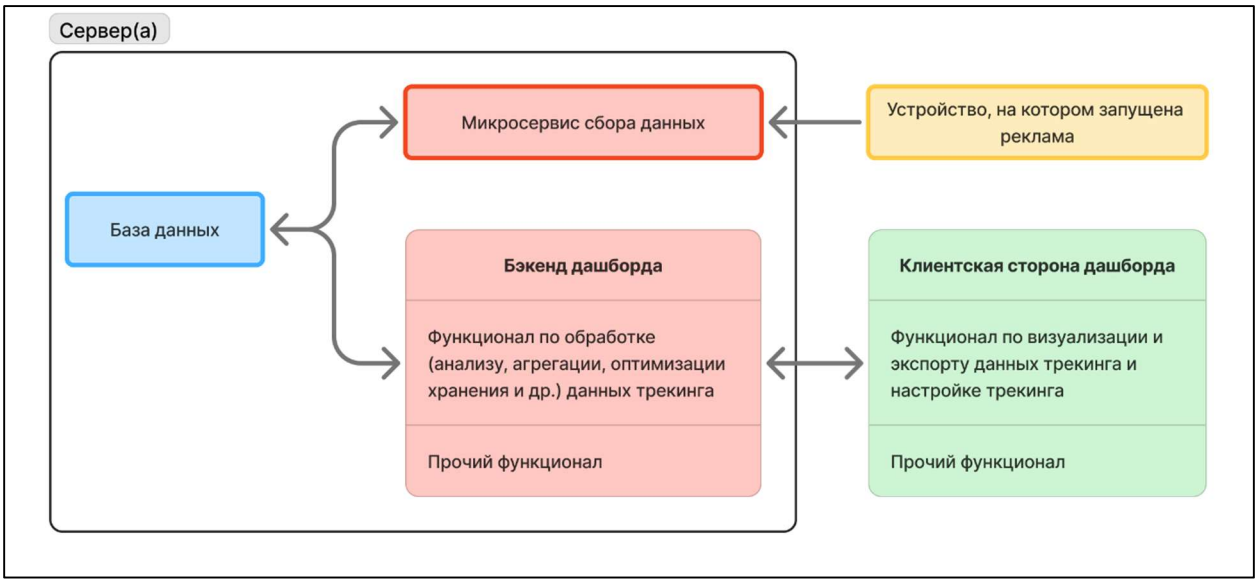


Рисунок 3.1 – Структурная схема программного комплекса

Структурная схема программного комплекса представлена на чертеже ГУИР 110902.001.

Таким образом, комплексная структура программного обеспечения объединяет высоко оптимизированный легковесный сервис, монолитный дашборд с модульной серверной архитектурой, эффективное *websocket*-соединение для обмена данными в реальном времени и клиентскую часть с широкими возможностями настройки и анализа. Это обеспечивает надежный, масштабируемый и гибкий инструмент для отслеживания и визуализации действий пользователей в играбельной рекламе, удовлетворяя как технические, так и аналитические требования конечных пользователей системы.

### 3.2 Алгоритм работы программного комплекса

Работа программного комплекса включает в себя реализацию ряда алгоритмов, рассмотрим наиболее значимые из них.

Алгоритм работы встроенного клиентского кода:

1 Произошло отслеживаемое событие: в момент, когда в системе фиксируется одно из заданных событий, например, загрузка плеебла, взаимодействие пользователя с интерфейсом приложения, истечение таймаута, возникновение ошибки, этот факт становится отправной точкой для дальнейшей обработки. Такое событие может быть инициировано как самим пользователем при нажатии кнопки или скролле, так и внутренним кодом рекламного приложения, автоматически генерирующим событие в ответ на предустановленные условия.

2 Проверяется, инициализирован ли модуль трекинга: после детектирования события система выполняет проверку состояния модуля аналитики, чтобы убедиться, что все необходимые компоненты инициализированы и готовы к сбору данных. В случае отсутствия инициализации дальнейшие шаги по обработке события приостанавливаются до восстановления корректного состояния модуля.

3 Генерируются и сохраняются уникальные идентификаторы сессии и времени старта: если модуль трекинга еще не был инициализирован, код автоматически создает уникальные идентификаторы для текущей сессии и фиксирует отметку времени старта работы пользователя, что позволяет различать и корректно обрабатывать данные разных запусков приложения.

4 При необходимости создается и сохраняется уникальный идентификатор пользователя: после генерации идентификаторов сессии система проверяет, присвоен ли текущему пользователю уникальный ключ, и в случае его отсутствия создает новый. Это необходимо для долгосрочного отслеживания поведения пользователей и проведения качественной аналитики.

5 Определяется, требуется ли получение геолокации: после инициализации пользовательских и сессионных метаданных код проверяет настройки трекинга и устанавливает, нужно ли собирать данные о местоположении пользователя в рамках данной сессии или события.

6 Если геолокация требуется, отправляется запрос и приём событий приостанавливается: при необходимости сбора координат приложение формирует запрос к геолокационному сервису устройства, при этом прием дальнейших пользовательских событий временно ставится на паузу во избежание потери данных и некорректных тайминг-марок.

7 После получения ответа о геолокации возобновляется приём событий: независимо от успешного или неуспешного получения координат система отмечает статус «геолокация получена» и снимает паузу, после чего

продолжает прием и обработку других событий согласно установленным правилам.

8 Устанавливается ожидание появления элемента в поле зрения пользователя: если в настройках указано отслеживать отображение конкретного элемента интерфейса, код переходит в режим слежения за видимостью, где оно ожидается до тех пор, пока целевой объект не станет видимым в области просмотра пользователя.

9 После появления элемента в поле зрения возобновляется передача накопленных событий: как только целевой элемент отображается пользователю, система выключает режим ожидания и отправляет все события, накопленные в буфере за время ожидания, на сервер аналитики.

10 Подключаются внешние библиотеки аналитики через динамическую загрузку: для более сложных сценариев отслеживания производится динамическая загрузка необходимых скриптов сторонних аналитических провайдеров, их конфигурация и инициализация, что позволяет расширять функционал без пересборки основного приложения.

11 Подписываются обработчики на все типы событий в окружении: код регистрирует слушателей для всех целевых типов событий, будь то события среды MRAID, внутренних шин обмена сообщениями или других интерфейсов, что обеспечивает всесторонний мониторинг поведения приложения.

12 При каждом событии формируется объект с параметрами: имя события, дополнительная информация, временные метки и счетчики: каждый раз при возникновении события система создает детализированный объект, включающий название события, контекстные данные, отметки времени и при необходимости счетчики повторных триггеров.

13 Для каждого целевого URL проверяются условия отправки: на этом этапе алгоритм определяет для каждого адреса получателя, требуется ли отправлять только первое событие в серии или все подряд, исходя из настроек кампаний и ограничений по частоте.

14 Если целевой элемент не видим, событие помещается в буфер ожидания: при отсутствии видимости отслеживаемого объекта событие не отправляется немедленно, а добавляется в промежуточное хранилище до наступления условия видимости.

15 Если элемент видим или видимость не требуется, событие отправляется немедленно: как только выполняются условия видимости или они изначально не обязательны, данные отправляются на целевой URL без задержек, обеспечивая своевременный учет действий пользователя.

16 Для глобального трекинга проверяются исключения и ограничения по числу отправок: перед каждой отправкой алгоритм оценивает правила глобального трекинга, исключения и квоты, чтобы не превышать лимиты запросов и не создавать избыточный сетевой трафик.

17 Каждое событие отправляется к провайдеру через XHR, загрузку изображения или динамический скрипт: в зависимости от настроек и возможностей среды события передаются методом XHR-запроса, тегом  или подключением скрипта, что делает систему максимально гибкой в различных окружениях.

18 Нативные события накапливаются в пакет и отправляются по расписанию: события, генерируемые нативными средствами платформы, собираются в пакеты и передаются на сервер периодически, согласно заданному интервалу, чтобы оптимизировать нагрузку и сократить число соединений.

19 При возникновении ошибки приложения событие об ошибке регистрируется и добавляется в пакет нативного трекинга: если во время работы приложения происходит исключение или ошибка, информация о ней фиксируется и включается в следующий отправляемый пакет вместе с другими событиями.

20 После отправки пакет очищается; в случае неудачи данные сохраняются для повторной попытки: после успешной передачи накопленные события удаляются из буфера, а при возникновении сбоя сохраняются для повторной отправки, что гарантирует надежность доставки данных.

21 Перед отправкой каждого запроса URL обогащается актуальными параметрами: перед непосредственным формированием сетевого запроса адрес целевого сервиса дополняется актуальными значениями — временем, именем события, идентификаторами и другими параметрами, необходимыми для корректной обработки на стороне сервера.

Алгоритм работы сервиса приёма и сохранения запросов на сервере:

1 При старте приложения регистрируются обработчики для приёма одиночных и пакетных событий: в момент запуска серверного приложения создаются маршруты и функции-обработчики, способные принимать как одиночные запросы, так и пакеты накопленных данных от клиентов.

2 При поступлении запроса извлекаются параметры контекста и полезная нагрузка: после получения HTTP-запроса система анализирует заголовки и тело, извлекая идентификаторы проекта, версию SDK и саму полезную нагрузку — события и связанные с ними метаданные.

3 Выполняется валидация обязательных параметров: система проверяет наличие всех необходимых полей и их корректность, и если данные не соответствуют ожидаемым форматам или отсутствуют ключевые параметры, формирует ответ об ошибке с подробным описанием проблемы.

4 Для каждого события подготавливается запись: объединяются контекстные данные и параметры события в одну сущность, готовую к записи в хранилище, что упрощает последующие операции агрегации и анализа.

5 Записи событий сохраняются в хранилище асинхронно: после подготовки объектов они передаются в очередь или непосредственно в базу

данных асинхронно, чтобы не блокировать основной поток обработки входящих запросов и обеспечить высокую пропускную способность.

6 При необходимости формируется и сохраняется запись информации о пользователе: если событие содержит новые данные о пользователе или меняет его атрибуты, сервер дополнительно создает или обновляет соответствующий профиль в пользовательском хранилище.

7 Ошибки при сохранении логируются и влияют на итоговый код ответа: при возникновении проблем с записью данных система фиксирует их в логах и возвращает клиенту соответствующий HTTP-код ошибки, информируя о сбоях на сервере.

8 Ожидается завершение всех операций записи: перед формированием окончательного ответа система дожидается выполнения всех фоновых задач по сохранению событий и профилей пользователей, что гарантирует консистентность данных.

9 По результатам всех операций формируется и отправляется клиенту итоговый статус выполнения: на основании успешности или неудачи сохранения всех записей сервер собирает сводный результат и возвращает его в виде JSON-объекта, завершив рабочий цикл обработки запроса.

Алгоритм агрегирующего сервиса для дашборда:

1 Запускается процедура агрегации данных: по расписанию или вручную инициируется процесс, который готовит сводные отчёты по накопленным событиям за разные периоды времени.

2 Выбираются временные интервалы и определяется их длительность: в зависимости от типа агрегации (по часам, дням, месяцам) система рассчитывает границы каждого интервала и продолжительность, необходимую для корректного подсчёта метрик.

3 Для каждого интервала определяется точка последней обработки: система считывает в базе метку об окончании предыдущей агрегации, чтобы не обрабатывать одни и те же события повторно и обеспечить инкрементальный режим работы.

4 Последовательно запрашиваются порции неподготовленных данных с учётом лимита и смещения: для уменьшения нагрузки на базу и ограничения по объёму данных проводятся выборки частями с использованием параметров лимита и оффсета.

5 При отсутствии новых записей переходит к следующему интервалу: если выборка вернула пустой результат, алгоритм завершает обработку текущего периода и начинается агрегация по следующему.

6 Для каждой записи вычисляется начало её интервала и обновляется счётчик в агрегированной таблице: анализируя временные метки событий, система определяет, к какому интервалу они относятся, и увеличивает соответствующие агрегированные значения.

7 После обработки всех записей интервала происходит переход к следующему: по завершении обхода всех частей данных алгоритм фиксирует успешное завершение обработки данного периода и переходит к следующим интервалам по нарастающей.

8 По завершении агрегации по всем интервалам процедура окончательно завершается: после обработки всех заданных периодов сохраняются контрольные метки и освобождаются ресурсы, алгоритм считается завершенным.

Алгоритм обработки данных при скачивании отчёта:

1 Запускается обработка запроса на скачивание отчёта: при получении клиентом команды на формирование отчета активируется соответствующая служба, готовая сформировать выгрузку данных.

2 Проверяются права доступа и корректность запрошенной версии: система удостоверяется, имеет ли пользователь необходимые права для получения отчёта и существует ли требуемая версия данных.

3 Извлекаются параметры диапазона дат и формат отчёта: из запроса считываются границы периода и требуемый формат выгрузки — сырые данные или агрегированные по дням.

4 В зависимости от формата выбирается путь обработки: для сырых данных формируется последовательная выборка порций, а для агрегированных — единоразовый запрос к готовой таблице агрегатов.

5 Для сырого формата устанавливаются заголовки и начинается поэтапная выборка данных: клиенту отправляются HTTP-заголовки, после чего в теле ответа пошагово передаются участки данных порциями.

6 Каждая порция преобразуется в строку нужного формата и сразу отправляется клиенту: при получении очередного блока записей они конвертируются (CSV, JSON или другой формат) и сразу же передаются по открытому соединению.

7 После обработки всех порций соединение корректно закрывается: по завершении передачи последней части данных система отправляет сигнал об окончании и закрывает соединение без ошибок.

8 Для формата «дни» выполняется единоразовый запрос всех агрегированных записей за период: при выборе агрегированного формата вся информация за указанный диапазон извлекается одной большой выборкой.

9 Собирается структура таблицы в памяти: на основе полученных агрегатов формируется двумерный массив или структура с заголовками столбцов, метками дат и значениями счетчиков.

10 Генерируется итоговый файл и отправляется целиком: после подготовки полного набора данных файл конвертируется в запрошенный формат (Excel, CSV, PDF) и пересылается клиенту одной партией.

11 По завершении обеспечивается корректное завершение передачи: после успешной отправки система фиксирует статус операции и закрывает все вспомогательные ресурсы, гарантируя целостность и полноту отчёта.

Алгоритм периодической проверки трекинга:

1 Запускается процедура проверки доступности сервиса: по таймеру или по расписанию активируется модуль мониторинга, готовый проверить работоспособность трекинг-сервиса.

2 Выполняется запрос к тестовому эндпоинту: система отправляет контрольный HTTP-запрос или пинг к специальному тестовому URL для проверки доступности основного API трекинга.

3 Анализируется результат: оценивается ответ сервера — успешный код или ошибки соединения и таймауты — для определения текущего состояния.

4 При недоступности генерируется запись об ошибке: если тестовый запрос завершился ошибкой, модуль создает лог-запись с деталями сбоя для последующего анализа.

5 Сравнивается текущее состояние с предыдущим: новая информация сверяется с последним известным статусом, чтобы определить изменение состояния сервиса.

6 При изменении статуса отправляются уведомления администраторам: в случае перехода из доступного состояния в недоступное или обратно система рассылает письма или сообщения в мессенджеры ответственным специалистам.

7 Для каждого активного соединения с клиентом отправляется обновлённый статус: если существуют открытые соединения с клиентскими приложениями, они получают уведомление о текущем состоянии трекинга.

8 Процедура проверки повторяется через заданный интервал времени: по окончании всех шагов алгоритм ожидает указанное время и запускает все проверки заново для поддержания постоянного мониторинга.

### 3.3 Структура базы данных программного комплекса

Структура базы данных должна обеспечивать логичное хранение данных трекинга и доступ к ним, а также возможности по оптимизации хранения данных и мониторинг ошибок.

Структурная схема базы данных программного комплекса представлена на рисунке 3.2.

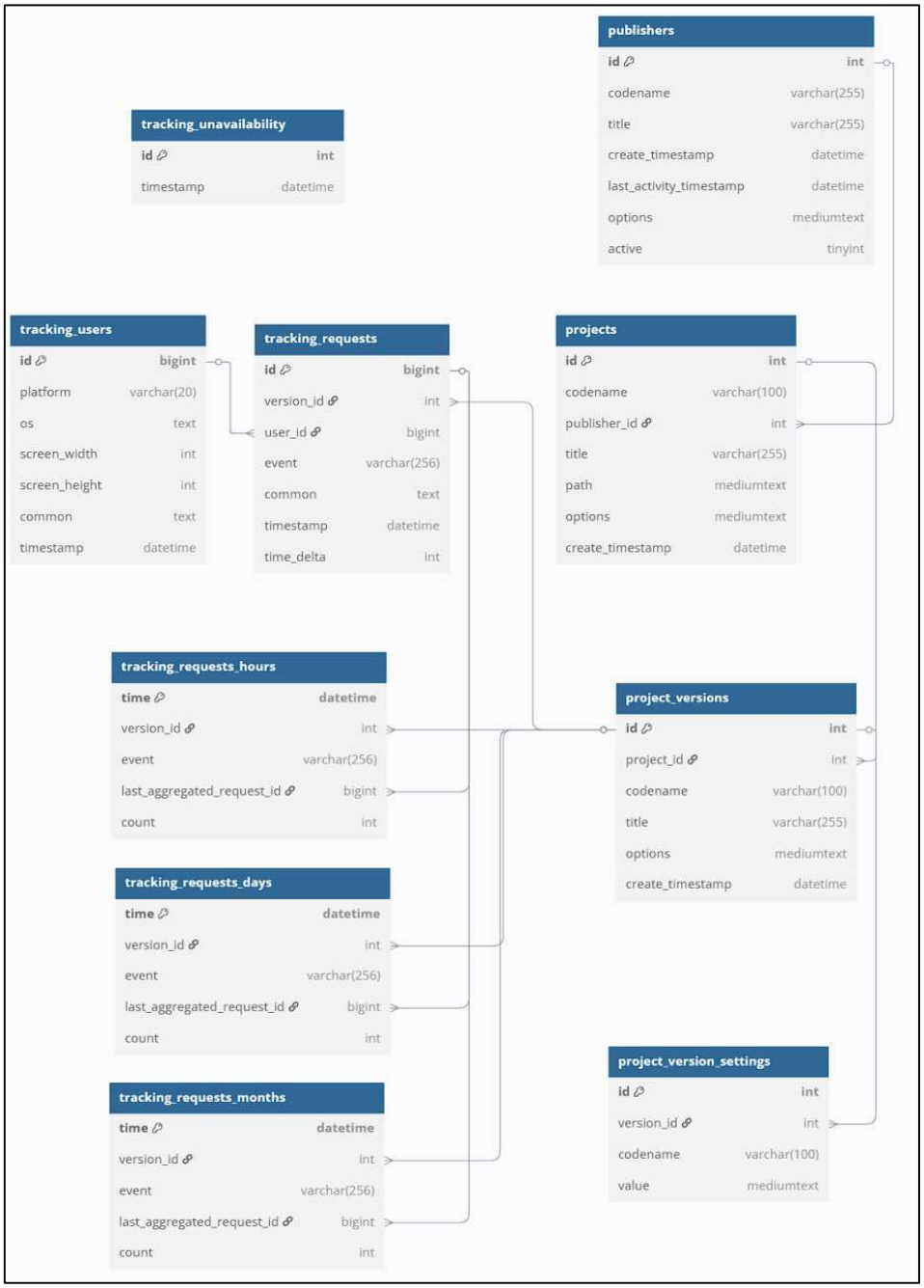


Рисунок 3.2 – Структурная схема базы данных программного комплекса

Структура базы данных также представлена на чертеже ГУИР 110902.002 ППД.

Опишем все таблицы и их поля.

Поле id в таблице tracking_users представляет собой уникальный идентификатор пользователя и служит основным ключом для связывания записей о пользователе с другими сущностями, поле platform указывает на среду выполнения (например, web, android или ios) и позволяет сегментировать данные по типу клиентского приложения, поле os содержит

точное наименование операционной системы на устройстве пользователя, а поля screen_width и screen_height фиксируют текущие размеры экрана в пикселях, что важно для адаптивного рендеринга интерфейса; поле common хранит любые дополнительные сведения в свободном формате (например, язык интерфейса или версия браузера), а timestamp отмечает момент создания записи, что помогает анализировать историю изменений и определять актуальность информации.

В таблице tracking_requests поле id является уникальным идентификатором каждой записи о событии, поле version_id связывает запрос с определённой версией проекта или приложения, user_id выступает внешним ключом на tracking_users.id и помогает восстановить контекст пользователя, поле event описывает тип события (например, открытие экрана или нажатие на кнопку), поле common даёт возможность хранить произвольные дополнительные параметры события (например, координаты нажатия или имя элемента интерфейса), а timestamp фиксирует точное время регистрации запроса, что позволяет отслеживать динамику пользовательских действий во времени.

Таблица tracking_requests_hours агрегирует данные по часам: поле time указывает на начало часового интервала (например, «2025-05-24 13:00:00»), version_id связывает агрегат с конкретным выпуском приложения, event описывает тип события, по которому ведётся подсчёт, last_aggregated_request_id хранит идентификатор последнего включённого в расчёт запроса, а поле count отражает общее число зафиксированных событий данного типа за указанный часовой интервал.

Таблицы tracking_requests_days и tracking_requests_months аналогично таблице tracking_requests_hours содержат поля time, version_id, event, last_aggregated_request_id и count, но их интервал привязан соответственно к суткам и календарному месяцу, что даёт возможность получать суммарные показатели по более длительным периодам и упрощает построение отчётов с нужным уровнем детализации.

В таблице tracking_unavailability поле id является уникальным ключом записи, а поле timestamp фиксирует момент, когда было зафиксировано событие недоступности или ошибки в системе; эти данные помогают анализировать периоды сбоев в работе сервиса и рассчитывать среднее время безотказной работы.

Таблица projects хранит сведения о каждом проекте: поле id выполняет роль основного ключа, codename содержит кодовое обозначение проекта, publisher_id выступает внешним ключом на таблицу издателей и связывает проект с владельцем, поле title хранит человекочитаемое название, а поля path и options определяют директорию проекта и его конфигурационные параметры; поле create_timestamp фиксирует дату и время создания проекта в системе.

В таблице project_versions поле id является уникальным идентификатором версии, project_id ссылается на родительский проект в таблице projects, поле codename содержит кодовое имя выпуска, title хранит название версии, поле options сохраняет дополнительные настройки конкретного выпуска, а create_timestamp отмечает время создания версии для возможности отката или сравнения между релизами.

Таблица project_version_settings отвечает за хранение параметров конфигурации каждой версии: поле id — уникальный ключ записи, version_id — внешний ключ на project_versions.id, поле codename задаёт название или ключ настройки, а поле value содержит её значение в свободном формате, позволяя сохранять разнообразные параметры без изменения структуры базы.

Далее опишем все сущности, хранящиеся в базе данных

Структура сущности tracking_unavailability представлена в таблице 3.1.

Таблица 3.1. Структура сущности tracking_unavailability

| Название поля | Тип данных | Описание |
|---|---|---|
| id (PK) | INT | Идентификатор записи |
| timestamp | DATETIME | Время записи |

Структура сущности publishers представлена в таблице 3.2.

Таблица 3.2. Структура сущности publishers

| Название поля | Тип данных | Описание |
|---|---|---|
| id (PK) | INT | Идентификатор паблишера |
| codename | VARCHAR(255) | Кодовое имя паблишера |
| title | VARCHAR(255) | Отображаемое имя паблишера |
| create_timestamp | DATETIME | Время создания паблишера |
| last_activity_timestamp | DATETIME | Время последней активности паблишера |
| options | MEDIUMTEXT | Опции паблишера |
| active | TINYINT | Статус активности паблишера |

Структура сущности tracking_users представлена в таблице 3.3.

Таблица 3.3. Структура сущности tracking_users

| Название поля | Тип данных | Описание |
|---|---|---|
| id (PK) | BIGINT | Идентификатор пользователя |

Продолжение таблицы 3.3.

| | | |
|---|---|---|
| platform | VARCHAR(20) | Название платформы (релиза) |
| os | TEXT | Название операционной системы |
| screen_width | INT | Ширина экрана |
| screen_height | INT | Высота экрана |
| common | TEXT | Дополнительная информация |
| timestamp | DATETIME | Время создания пользователя |

Структура сущности tracking_requests представлена в таблице 3.4.

Таблица 3.4. Структура сущности tracking_requests

| Название поля | Тип данных | Описание |
|---|---|---|
| id (PK) | BIGINT | Идентификатор запроса |
| version_id (FK) | INT | Идентификатор версии |
| user_id (FK) | BIGINT | Идентификатор пользователя |
| event | VARCHAR(256) | Название события |
| common | TEXT | Дополнительная информация |
| timestamp | DATETIME | Время запроса |
| time_delta | INT | Время, прошедшее с запуска плеебла |

Структура сущности projects представлена в таблице 3.5.

Таблица 3.5. Структура сущности projects

| Название поля | Тип данных | Описание |
|---|---|---|
| id (PK) | INT | Идентификатор проекта |
| codename | VARCHAR(100) | Кодовое имя проекта |
| publisher_id (FK) | INT | Идентификатор паблишера |
| title | VARCHAR(255) | Отображаемое имя проекта |
| path | MEDIUMTEXT | Путь к папке с файлами проектп |
| options | MEDIUMTEXT | Опции проекта |
| create_timestamp | DATETIME | Время создания проекта |

Структура сущности tracking_requests_hours представлена в таблице 3.6.

Таблица 3.6. Структура сущности tracking_requests_hours

| Название поля | Тип данных | Описание |
|---|---|---|
| time (PK) | DATETIME | Время начала интервала |
| version_id (FK) | INT | Идентификатор версии |
| event | VARCHAR(256) | Название события |
| last_aggregated_id (FK) | BIGINT | Идентификатор последнего подсчитанного события |
| count | INT | Количество сосчитанных событий |

Структура сущности tracking_requests_days представлена в таблице 3.7.

Таблица 3.7. Структура сущности tracking_requests_days

| Название поля | Тип данных | Описание |
|---|---|---|
| time (PK) | DATETIME | Время начала интервала |
| version_id (FK) | INT | Идентификатор версии |
| event | VARCHAR(256) | Название события |
| last_aggregated_id (FK) | BIGINT | Идентификатор последнего подсчитанного события |
| count | INT | Количество сосчитанных событий |

Структура сущности tracking_requests_months представлена в таблице 3.8.

Таблица 3.8. Структура сущности tracking_requests_months

| Название поля | Тип данных | Описание |
|---|---|---|
| time (PK) | DATETIME | Время начала интервала |
| version_id (FK) | INT | Идентификатор версии |
| event | VARCHAR(256) | Название события |
| last_aggregated_id (FK) | BIGINT | Идентификатор последнего подсчитанного события |
| count | INT | Количество сосчитанных событий |

Структура сущности project_versions представлена в таблице 3.9.

Таблица 3.9. Структура сущности project_versions

| Название поля | Тип данных | Описание |
|---|---|---|
| id (PK) | INT | Идентификатор версии проекта |
| project_id (FK) | INT | Идентификатор проекта |
| codename | VARCHAR(100) | Кодовое название версии |
| title | VARCHAR(255) | Отображаемое название версии |
| options | MEDIUMTEXT | Опции |
| create_timestamp | DATETIME | Время создания версии |

Структура сущности project_version_settings представлена в таблице 3.10.

Таблица 3.10. Структура сущности project_version_settings

| Название поля | Тип данных | Описание |
|---|---|---|
| id (PK) | INT | Идентификатор настройки |
| version_id (FK) | INT | Идентификатор версии проекта |
| codename | VARCHAR(100) | Кодовое название настройки |
| value | MEDIUMTEXT | Значение настройки |

### 3.4. Выводы и оценка результатов разработки

Структура программного комплекса для отслеживания действий пользователей в играбельной рекламе выстроена на основе тщательно продуманных архитектурных решений, ориентированных на устойчивость к нагрузкам и возможность масштабирования. Основным компонентом является лёгковесный сервис трекинга, задача которого заключается исключительно в сборе и передаче данных о взаимодействиях конечного пользователя с рекламным контентом. Благодаря минимальному числу внешних зависимостей и отсутствию избыточных модулей этот сервис сохраняет высокую отказоустойчивость и может обрабатывать непрерывный поток запросов без деградации производительности даже при экстремальных нагрузках или сбоях отдельных сегментов сети.

Код трекинг-сервиса и компоненты, отвечающие за агрегацию и хранение данных, тесно интегрированы в монолитный дашборд, обеспечивая единую точку управления и упрощая сопровождение всего проекта. Такой подход позволяет разработчикам и операторам системы в реальном времени отслеживать ключевые метрики и метаданные, выявлять узкие места и оперативно вносить необходимые изменения в конфигурацию. При этом для обмена сообщениями между клиентской частью и сервером задействовано постоянное WebSocket-соединение, которое гарантирует минимальные

задержки при передаче данных и снижает накладные расходы протоколов HTTP-запросов, что особенно важно при высокоинтенсивном обмене событиями.

Клиентская библиотека трекинга, внедряемая в билд рекламного приложения, предлагает расширенный набор функций для гибкой настройки параметров сбора данных: здесь предусмотрена возможность тонкой фильтрации типов событий, определения условий срабатывания триггеров и динамического управления буферизацией сообщений. Кроме того, пользовательский интерфейс дашборда снабжён инструментами для визуализации накопленных данных — графиками, таблицами и тепловыми картами — позволяющими аналитикам получать детальную картину поведения аудитории и быстро формировать отчёты на основании полученной информации.

Не менее важным элементом решения является продуманная структура баз данных, спроектированная с учётом требований к высокой скорости записи большого количества мелких событий и способности выполнять гибкую агрегацию статистики по различным временным срезам и категориям. Партиционирование таблиц по времени и версиям проекта, а также индексация ключевых полей обеспечивают как быстрый доступ к свежим данным, так и эффективное выполнение запросов для построения отчётов за произвольные интервалы. Дополнительные атрибуты событий, включая метки времени и расширенные поля «common», позволяют при необходимости проводить глубокий анализ и сегментировать данные по самым тонким признакам.

Непрерывное обновление дашборда через WebSocket-связь даёт возможность оперативно отображать самые актуальные сведения о действиях пользователей, сохраняя высокую точность и адаптивность настроек мониторинга. В итоге разработанный комплекс представляет собой надёжный и гибкий инструмент, который не только удовлетворяет техническим требованиям промышленных систем аналитики, но и легко адаптируется под новые сценарии использования, обеспечивая долговременное улучшение качества рекламных кампаний на основании детальных данных о поведении аудитории.