TOMSK POLYTECHNIC UNIVERSITY ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Инженерная школа информационных технологий и робототехники
Направление подготовки 09.04.02 Информационные системы и технологии
Отделение информационных технологий

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Тема работы
Разработка программной библиотеки синхронизации данных мобильных приложений для offline-работы

УДК 004.428:004.057.5

Студент

Группа	ФИО	Подпись	Дата
8ИМ02	Тюндеров Кирилл Вадимович		

Руководитель ВКР

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Савельев Алексей Олегович	к.т.н		

Консультант

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Технический директор ООО «МЦЦ-Томск»	Дробинский Артур Владимирович			

КОНСУЛЬТАНТЫ ПО РАЗДЕЛАМ:

По разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОСГН ШБИП	Былкова Татьяна Васильевна	канд.экон.наук		

По разделу «Социальная ответственность»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Профессор ООД ШБИП	Федоренко Ольга Юрьевна	д-р мед. наук		

По разделу на английском языке

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Старший преподаватель ОИЯ	Пичугова Инна Леонидовна			

ДОПУСТИТЬ К ЗАЩИТЕ:

Руководитель ООП	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Савельев Алексей Олегович	к.т.н		

Томск – 2022 г.

ПЛАНИРУЕМЫЕ РЕЗУЛЬТАТЫ ОБУЧЕНИЯ

Код компетенции	Наименование компетенции
	Универсальные компетенции
УК(У)-1	Способен осуществлять критический анализ проблемных ситуаций на основе системного подхода, вырабатывать стратегию действий
УК(У)-2	Способен управлять проектом на всех этапах его жизненного цикла
УК(У)-3	Способен организовывать и руководить работой команды, вырабатывая командную стратегию для достижения поставленной цели
УК(У)-4	Способен применять современные коммуникативные технологии, в том числе на иностранном(-ых) языке(-ах), для академического и профессионального взаимодействия
УК(У)-5	Способен анализировать и учитывать разнообразие культур в процессе межкультурного взаимодействия
УК(У)-6	Способен определять и реализовывать приоритеты собственной деятельности и способы ее совершенствования на основе самооценки
	Общепрофессиональные компетенции
ОПК(У)-1	Способен самостоятельно приобретать, развивать и применять математические, естественно-научные, социально-экономические и профессиональные знания для решения нестандартных задач, в том числе в новой или незнакомой среде и в междисциплинарном контексте
ОПК(У)-2	Способен разрабатывать оригинальные алгоритмы и программные средства, в том числе с использованием современных интеллектуальных технологий, для решения профессиональных задач
ОПК(У)-3	Способен анализировать профессиональную информацию, выделять в ней главное, структурировать, оформлять и представлять в виде аналитических обзоров с обоснованными выводами и рекомендациями
ОПК(У)-4	Способен применять на практике новые научные принципы и методы исследований
ОПК(У)-5	Способен разрабатывать и модернизировать программное и аппаратное обеспечение информационных и автоматизированных систем
ОПК(У)-6	Способен использовать методы и средства системной инженерии в области получения, передачи, хранения, переработки и представления информации посредством информационных технологий

ОПК(У)-7	Способен разрабатывать и применять математические модели процессов и объектов при решении задач анализа и синтеза распределенных информационных систем и систем поддержки принятия решений
ОПК(У)-8	Способен осуществлять эффективное управление разработкой программных средств и проектов
	Профессиональные компетенции
ПК(У)-1	Способен управлять программно-техническими, технологическими и человеческими ресурсами
ПК(У)-2	Способен управлять развитием баз данных
ПК(У)-3	Способен управлять работами по сопровождению и проектами создания (модификации) информационных систем, автоматизирующих задачи организационного управления и бизнес-процессы.
ПК(У)-4	Способен проектировать и организовывать учебный процесс по образовательным программам с использованием современных образовательных технологий.
ПК(У)-5	Способен осуществлять руководство разработкой комплексных проектов на всех стадиях и этапах выполнения работ.

TOMSK POLYTECHNIC UNIVERSITY ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Инженерная школа информационных технологий и робототехники
Направление подготовки 09.04.02 Информационные системы и технологии
Отделение информационных технологий

УТВЕРЖДАЮ:
Руководитель ООП
_____ _____ А.О. Савельев
(Подпись) (Дата) (ФИО)

ЗАДАНИЕ
на выполнение выпускной квалификационной работы

В форме:

Магистерской диссертации

(бакалаврской работы, дипломного проекта/работы, магистерской диссертации)

Студенту:

Группа	ФИО
8ИМ02	Тюндеров Кирилл Вадимович

Тема работы:

Разработка программной библиотеки синхронизации данных мобильных приложений для offline-работы	
Утверждена приказом директора (дата, номер)	Приказ №103-16/с от 13.04.2022

Срок сдачи студентом выполненной работы:	23.06.2022

ТЕХНИЧЕСКОЕ ЗАДАНИЕ:

Исходные данные к работе (наименование объекта исследования или проектирования; производительность или нагрузка; режим работы (непрерывный, периодический, циклический и т. д.); вид сырья или материал изделия; требования к продукту, изделию или процессу; особые требования к особенностям функционирования (эксплуатации) объекта или изделия в плане безопасности эксплуатации, влияния на окружающую среду, энергозатратам; экономический анализ и т. д.).	Объектом разработки является программная библиотека, предоставляющая функционал по организации offline-работы мобильного приложения. Платформы разработки: мобильное устройство. Требование к функционалу: предоставление бесперебойной работы приложению в необходимых местах функционала. Ресурсы мобильного устройства ограничены.

Перечень подлежащих исследованию, проектированию и разработке вопросов *(аналитический обзор по литературным источникамс целью выяснения достижений мировой науки техники в рассматриваемой области; постановка задачи исследования, проектирования, конструирования; содержание процедуры исследования, проектирования, конструирования; обсуждение результатов выполненной работы; наименование дополнительных разделов, подлежащих разработке; заключение по работе).*	1. Обзор и анализ существующих решений. 2. Разработка программной библиотеки по предоставлению инструментов организации offline-работы для использования разработчиками мобильных приложений. 3. Описание раздела финансового менеджмента, ресурсоэффективности и ресурсосбережения. 4. Описание раздела социальной ответственности.
Перечень графического материала *(с точным указанием обязательных чертежей)*	1. Блок-схемы алгоритмов. 2. Презентация в формате *.pptx.
Консультанты по разделам выпускной квалификационной работы *(с указанием разделов)*	
Финансовый менеджмент, ресурсоэффективность и ресурсосбережение	Былкова Татьяна Васильевна, Доцент ОСГН, ШБИП
Социальная ответственность	Федоренко Ольга Юрьевна, Профессор ООД, ШБИП
Английский язык	Пичугова Инна Леонидовна, Старший преподаватель ОИЯ
Названия разделов, которые должны быть написаны на русском и иностранномязыках:	
4 Разработка концепта программного решения 4.1 Структура 4.2 Реализация программного решения на сервере 4.2.1 Сервис по получению схемы базы данных 4.2.2 Сервис по управлению транзакциями на сервере	

Дата выдачи задания на выполнение выпускной квалификационной работы по линейному графику	

Задание выдал руководитель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Савельев Алексей Олегович	к.т.н		

Задание принял студент:

Группа	ФИО	Подпись	Дата
8ИМ02	Тюндеров Кирилл Вадимович		

TOMSK POLYTECHNIC UNIVERSITY ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Инженерная школа информационных технологий и робототехники
Направление подготовки 09.04.02 Информационные системы и технологии
Отделение информационных технологий

Форма представления работы:

Магистерской диссертации

(бакалаврской работы, дипломного проекта/работы, магистерской диссертации)

КАЛЕНДАРНЫЙ РЕЙТИНГ – ПЛАН

Выполнения выпускной квалификационной работы

Срок сдачи студентом выполненной работы:	

Дата контроля	Название раздела (модуля) /вид работы (исследования)	Максимальныйбалл раздела (модуля)
	Основная часть	75
	Финансовый менеджмент, ресурсоэффективность и ресурсосбережение	15
	Социальная ответственность	10

СОСТАВИЛ:
Руководитель ВКР

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Савельев Алексей Олегович	к.т.н		

СОГЛАСОВАННО:
Руководитель ООП

Руководитель ООП	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Савельев Алексей Олегович	к.т.н		

ЗАДАНИЕ ДЛЯ РАЗДЕЛА «ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ»

Студенту:

Группа	ФИО
8ИМ02	Тюндеров Кирилл Вадимович

Школа	ИШИТР	Отделение школы (НОЦ)	ОИТ
Уровень образования	Магистратура	Направление/специальность	09.04.02 Информационные системы и технологии

Исходные данные к разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»:	
1. *Стоимость ресурсов научного исследования (НИ): материально-технических, энергетических, финансовых, информационных и человеческих*	Стоимость ресурсов определялась по средней рыночной стоимости, и в соответствии с окладами сотрудников организации.
2. *Нормы и нормативы расходования ресурсов*	30% районный коэффициент, коэффициент дополнительной заработной платы 12%; накладные расходы 16 %.
3. *Используемая система налогообложения, ставки налогов, отчислений, дисконтирования и кредитования*	30% отчисления во внебюджетные фонды
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. *Оценка коммерческого и инновационного потенциала НТИ*	Провести предпроектный анализ
2. *Разработка устава научно-технического проекта*	Представить Устав научного проекта магистерской работы
3. *Планирование процесса управления НТИ: структура и график проведения, бюджет, риски и организация закупок*	Разработать план управления НТИ
4. *Определение ресурсной, финансовой, экономической эффективности*	Рассчитать сравнительную эффективность исследования
Перечень графического материала *(с точным указанием обязательных чертежей):*	
1. *«Портрет» потребителя результатов НТИ* 2. *Сегментирование рынка* 3. *Оценка конкурентоспособности технических решений* 4. *Матрица SWOT* 5. *График проведения и бюджет НТИ* 6. *Оценка ресурсной, финансовой и экономической эффективности НТИ* 7. *Потенциальные риски*	

Дата выдачи задания для раздела по линейному графику	

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОСГН, ШБИП	Былкова Татьяна Васильевна	канд.экон.наук		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8ИМ02	Тюндеров Кирилл Вадимович		

ЗАДАНИЕ ДЛЯ РАЗДЕЛА «СОЦИАЛЬНАЯ ОТВЕТСТВЕННОСТЬ»

Студенту:

Группа	ФИО
8ИМ02	Тюндеров Кирилл Вадимович

Школа	ИШИТР	Отделение	ОИТ
Уровень образования	Магистратура	Направление/специальность	09.04.02 Информационные системы и технологии

Тема ВКР:

Разработка программной библиотеки синхронизации данных мобильных приложений для offline-работы	
Исходные данные к разделу «Социальная ответственность»:	
1. Характеристика объекта исследования (вещество, материал, прибор, алгоритм, методика, рабочая зона) и области его применения	Разработана программная библиотека, предоставляющая набор инструментов для организации офлайн работы мобильного приложения
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. Правовые и организационные вопросы обеспечения безопасности при разработке проектного решения: – организационные мероприятия при компоновке рабочей зоны.	– Трудовой кодекс Российской Федерации (ТК РФ). – ГОСТ 12.2.032-78 ССБТ. Рабочее место при выполнении работ сидя. Общие эргономические требования. – ГОСТ 21889-76. Система «человек-машина». Кресло человека-оператора. Общие эргономические требования. – ГОСТ 12.1.045-84 ССБТ. Система стандартов безопасности труда. Электростатические поля. Допустимые уровни на рабочих местах и требования к проведению контроля. – ГОСТ 12.1.003-2014 Система стандартов безопасности труда (ССБТ). Шум. Общие требования безопасности. – ГОСТ 12.1.029-80 Система стандартов безопасности труда (ССБТ). Средства и методы защиты от шума. – СанПиН 1.2.3685-21 Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания. – СП 60.13330.2020 Отопление, вентиляция и кондиционирование воздуха СНиП 41-01-2003 (с Поправкой). – СП 52.13330.2016 Естественное и искусственное освещение. Актуализированная редакция СНиП 23-05-95.
2. Производственная безопасность при разработке проектного решения:	Вредные производственные факторы: – Наличие электромагнитных полей промышленных частот. – Нарушение микроклимата.

– Анализ выявленных вредных и опасных факторов – Расчет уровня опасного или вредного производственного фактора	– Отсутствие или недостатки необходимого искусственного освещения. – Повышенная пульсация светового потока. – Умственное перенапряжение. Опасные факторы: – Электрический ток. Средства коллективной защиты: – Вентиляция и очистка воздуха. – Кондиционирование воздуха. – Отопление. – Осветительные приборы; – Защитные заземления и зануления; – Изолирующие устройства и покрытия; Индивидуальные средства защиты не требуются. Расчет искусственного освещения.
3. Экологическая безопасность при разработке проектного решения: – Указать, какое воздействия на селитебную зону, атмосферу, гидросферу и литосферу оказывает процесс разработки.	Воздействие на селитебную зону: отсутствует. Воздействие на литосферу: при неправильной утилизации ноутбука, монитора и телефонов, макулатуры. Воздействие на гидросферу: отсутствует. Воздействие на атмосферу: отсутствует.
4. Безопасность в чрезвычайных ситуациях при разработке проектного решения: – Перечислить возможные ЧС при разработке проектного решения. – Указать наиболее типичную ЧС из перечисленных.	Возможные ЧС: пожар, ураган, гроза. Наиболее типичная ЧС: пожар.

Дата выдачи задания для раздела по линейному графику	

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Профессор ООД ШБИП	Федоренко Ольга Юрьевна	д-р мед. наук		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8ИМ02	Тюндеров Кирилл Вадимович		

Реферат

Выпускная квалификационная работа содержит 116 страниц, 32 рисунка, 21 таблицу, список используемых источников содержит 39 наименований.

Ключевые слова: offline-режим, мобильное приложение, react-native, бесперебойная работа, программное обеспечение.

Цель работы – реализовать программное решение, синхронизирующее клиентскую и серверную базы данных, позволяющее управлять необходимыми приложению данными как в онлайн-, так и в офлайн-режимах.

В ходе работы был проведён анализ существующих решений по предоставлению бесперебойной работы мобильных приложений в контексте доступа и работы с данными. На основе выявленных недостатков и преимуществ подходов была реализована программная библиотека, предназначенная для использования разработчиками мобильных приложений. Данная библиотека предоставляет инструментарий по гибкой организации offline-работы приложения, также совместной работы с данными приложения нескольких пользователей в одно время.

Программная библиотека состоит из 2 частей: клиентской и серверной. Разработка велась на языках Typescript и C# на основе технологий react-native и Asp.Net Core.

Содержание

ПОСТАНОВКА ПРОБЛЕМЫ

В настоящее время для работы с данными в мобильных приложениях повсеместно используется интернет-соединение. В такой модели всё управление, изменение и получение данных идёт через запросы к серверному приложению. Для того, чтобы изменить данные отправляется запрос на сервер, который в свою очередь чаще всего отвечает измененными данными. То же происходит и в приложениях с одновременной работой нескольких пользователей, чаще всего для этого используются веб-сокеты, по которым на сервер отправляются пакеты с изменениями, а сами изменения в данных применяются на сервере и рассылаются пользователям. И всё бы ничего – получать данные с сервера и изменять их запросами и особо не заботиться о доступности интернет-соединения, но возникают ситуации, когда подключения к сети нет, в таком случае приложение становится бесполезным. Данная проблема возникает у работников складов, магазинов, также, когда необходима работа приложения в дороге и сельской местности.

В некоторых местах предоставить интернет-соединение сложно, например, в больших складских помещениях или в старых зданиях с толстыми стенами, либо в зданиях, находящихся в удалённой местности. Потребность в работе с данными может возникнуть в любой ситуации – будь то регистрация пациента или же доступ к истории болезни пациента, его динамике терапии или же получение информации о товаре непосредственно на складе или при выезде куда-то загород.

Чтобы приложения могли работать в любом состоянии, вводится поддержка офлайн-режима, который подразумевает возможность работы без запросов на сервер. Это может быть реализовано различными способами. В таком режиме функционал приложения может быть ограничен, но основные функции, для которых необходима бесперебойная работа, должны быть доступны пользователю.

Порядок работы приложения в офлайн-режиме обусловлен функционалом самого приложения. В общем случае, должны присутствовать возможности как чтения, так и записи, а при подключении к сети, сообщать серверу об изменениях.

Таким образом, в некоторых ситуациях возникает необходимость поддержки бесперебойного доступа к работе с данными, что сделать не всегда просто.

ТРЕБОВАНИЯ К РЕШЕНИЮ

Предполагаемое решение должно отвечать некоторым требованиям:

- обеспечение доступа к чтению данных приложения вне зависимости от наличия интернет-соединения у пользователя;
- обеспечение возможности изменения данных приложения вне зависимости от наличия интернет-соединения у пользователя;
- организация загрузки данных для дальнейшего использования;
- синхронизация изменений с данными на сервере;
- встраиваемость решения в уже существующие приложения (как серверные, так и мобильные), то есть, не должно быть необходимости строить приложение на основе потенциального решения с нуля;
- совместимость с мобильными приложениями, написанными на React [1] и серверными приложениями, использующими Asp.Net Core [2].

1 Варианты решения проблемы

Любая проблема может иметь разные варианты решения, рассмотрим несколько касаемо организации бесперебойной работы приложения.

1.1 Расширение WIFI-сети

Самый прямой вариант обеспечения бесперебойности работы клиент-серверного приложения – обеспечить доступ в интернет повсеместно на площади, это может достигаться WIFI-ретрансляторами [3], например. Данное устройство позволяет принять WIFI-сигнал и передать его далее, тем самым усиливая его для пользователя. Также можно организовать так называемую mesh-сеть [4], считающуюся для приемника сигнала как одна большая WIFI-сеть. И таких устройств необходимо некоторое множество. Это значит, что для них нужна дополнительная сеть питания, а также в случае выхода из строя каких-то ретрансляторов, существует опасность лишения доступа в интернет в остальном секторе, что получал сигнал на ретрансляцию с вышедшего из строя устройства. К тому же, качество сигнала может сильно снижаться после нескольких ретрансляций.

Рисунок 1 – WIFI-ретранслятор

Таким образом, можно выделить преимущества и недостатки метода. Преимущества:

- не требуется поддержка офлайн-режима от приложения, что ведёт за собой удешевление разработки собственного решения или не накладывает ограничения на выбор существующего решения;
- простота реализации.

Недостатки:

- зависимость количества устройств от площади или плана здания. Если здание состоит из комнат и имеет не самые тонкие стены, сигнал WIFI будет затухать быстрее, а значит радиус покрытия меньше, и, как результат, увеличение количества ретрансляторов;
- необходимость настройки новых ретрансляторов;
- зависимость ретрансляторов от друг друга. В таком случае, при выходе из строя устройства, либо его перезагрузки, все зависимые ретрансляторы

утрачивают функциональность. При отсутствии сигнала на одном ретрансляторе, например, потребуется последовательная перезагрузка всех зависимых ретрансляторов. [5] В противном случае, к каждому устройству необходимо будет подвести Ethernet;

– после нескольких точек ретрансляции качество интернет-сигнала может ощутимо снижаться.

1.2 Кэширование запросов

Иным рабочим вариантом является кэширование запросов. Здесь источником данных остаётся сервер, при этом, когда клиент получает ответ на тот или иной запрос, он сохраняет этот ответ локально совместно с данными запроса. Далее, при переходе в офлайн-режим, при совершении запросов будут возвращены уже сохраненные варианты ответов для конкретных запросов с определенным набором динамических параметров (существуют решения, сильно упрощающие реализацию данного подхода). При совершении запросов на изменение, создаются локальные пакеты с данными для запроса, которые отправятся с восстановлением доступа в интернет. И такая система является удобной, понятной в реализации, но есть нюансы. Запрос на получение данных может иметь какие-либо динамические параметры, например, поиск среди какой-либо коллекции записей, отвечающих введенным условиям пользователя. Здесь возникает проблема: сохраненного запроса с такими параметрами может не быть, следовательно, унифицировать получаемые запросы достаточно сложно, ведь детерминированной зависимости между данными запроса и ответом нет, интерполировать данные считается невозможным. Где в таком случае брать данные? Что показывать пользователю? Для запросов на изменение ситуация не лучше: из-за отсутствия логики по изменению данных на клиенте, так как всё происходит на сервере, мы, совершая то или иное изменение, добавляем данные о запросе в некий буфер, потому что не можем сказать, как именно будут выглядеть данные приложения, особенно, если от изменяемой сущности зависят и другие

данные. В таких случаях пользователь не увидит результата своих действий, а вероятнее всего, запись будет помечена, как инвалидированная или устаревшая и с ней нельзя будет работать в полной мере.

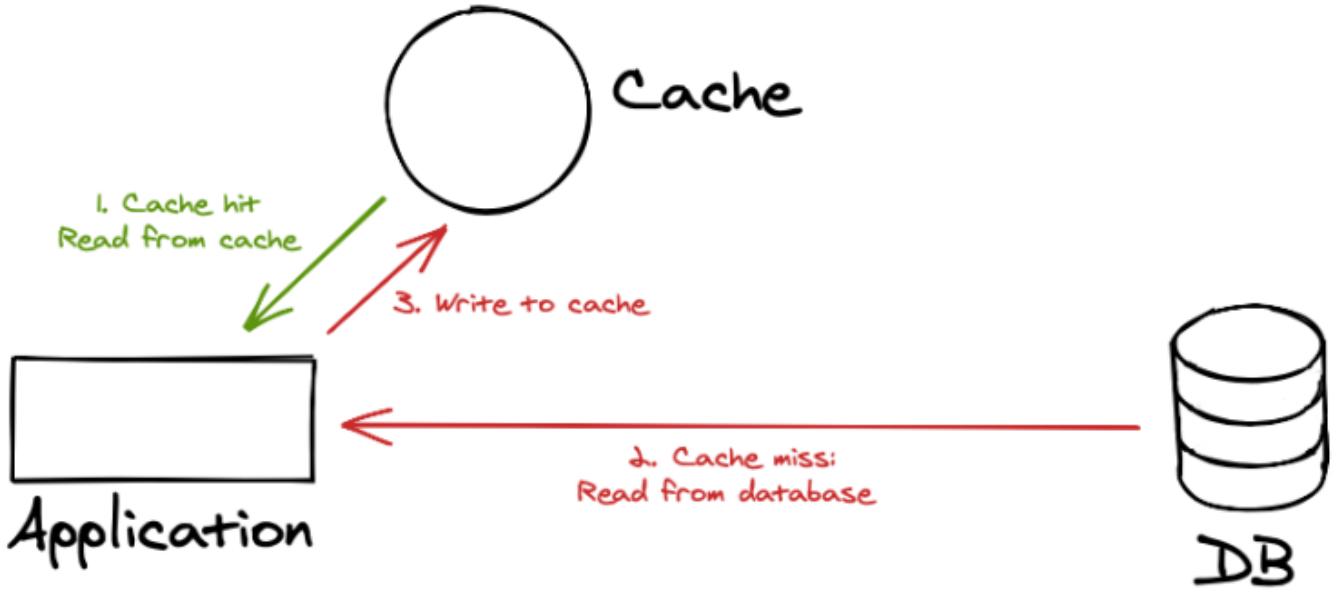


Рисунок 2 – Общая схема кэширования

Данный метод имеет ряд преимуществ:

– порядок работы и получения данных не меняется. Все данные запрашиваются с сервера с последующей работой с кэшем;
– сравнительно простая реализация кэширования запросов при использовании специальных библиотек (например, react-query [6]) по сравнению с другими вариантами решения проблемы.

А также ряд недостатков:

– чаще всего для реализации сохранения записей «ключ-значение» с возможностью восстановления после перезапуска приложения используется AsyncStorage [7], тут имеются некоторые ограничения [8] – данные не должны превышать суммарно 6 Мб, а также за раз читаемые данные не должны превышать 2 Мб. Это достаточно высокие пороги и их можно ещё увеличить, но всё же наличие ограничений означает, что в каких-то случаях использование AsyncStorage будет невозможным;

– невозможность кэширования запросов с динамическими параметрами. Конечно, можно бесконечно перебирать все возможные варианты запросов и сохранять ответы, но это нагрузка на сеть, сервер и само приложение, что всё равно не даст желаемых результатов, а ресурсов клиента будет потрачено большое количество, что является плохим решением;
– неприменимость созданных изменений к имеющимся данным, что ощущается как игнорирование действий пользователя. К тому же, управление и хранение неотправленных данных происходит отдельно для каждого из предметов изменений и полностью лежит на разработчике приложения, что потенциально производит больше багов, чем использование соответствующей библиотеки для разделения ответственности за функционал.

1.3 Синхронизация баз данных

Крайний вариант поддержания бесперебойности работы приложения, в плане доступа к данным, является перенос источника данных для приложения с сервера на какое-либо локальное хранилище. Доступ к такому хранилищу зависит только от состояния устройства пользователя. Данное хранилище заполняется данными с сервера при наличии интернет-соединения. Данные могут храниться в любом удобном формате и структуре. Если это данные какой-то одной сущности, то структура данных может быть тривиальной, что сильно упростит работу. Если приложение обеспечивает бесперебойную работу с большим количеством сущностей, использующих связи и прочее, то структура может быть устроена по подобию базы, находящейся на сервере, в таком случае это тоже упрощение порядка работы с данными.

Такую работу приложения можно охарактеризовать как работу по синхронизации баз данных на сервере и клиенте. При таком подходе приложение локально имеет все необходимые данные для работы желаемого функционала, здесь же ничего не мешает изменять данные и отображать валидные данные пользователю,

так как источником данных выступает локальное хранилище, даже при наличии интернет-соединения. При использовании решения подобного плана необходим перенос или дублирование логики работы с данными с сервера на клиент, это может внести некоторые неудобства при изменении логики. В такой модели при внесении изменений в данные, события записи сохраняются в то же локальное хранилище для того, чтобы при появлении сети, отправить эту информацию на сервер и синхронизовать данные в базах.

Для обладания высокой скоростью работы с данными, при поддержке динамических атрибутов поиска, хорошо подходит SQLite база данных, которая практически не имеет ограничений по использованию [9]. Реализация работы с таким хранилищем требует динамического построения запросов, что является сложной задачей, но выполнимой. Также присутствуют и другие варианты локального хранилища, у всех есть свои достоинства и недостатки, выбор зависит от приложения и требования к порядку работы с хранилищем, его интенсивностью. Данный метод сложнее остальных в реализации, но он покрывает все возможные нужды при работе в офлайн-режиме приложения. И подобное решение существует.

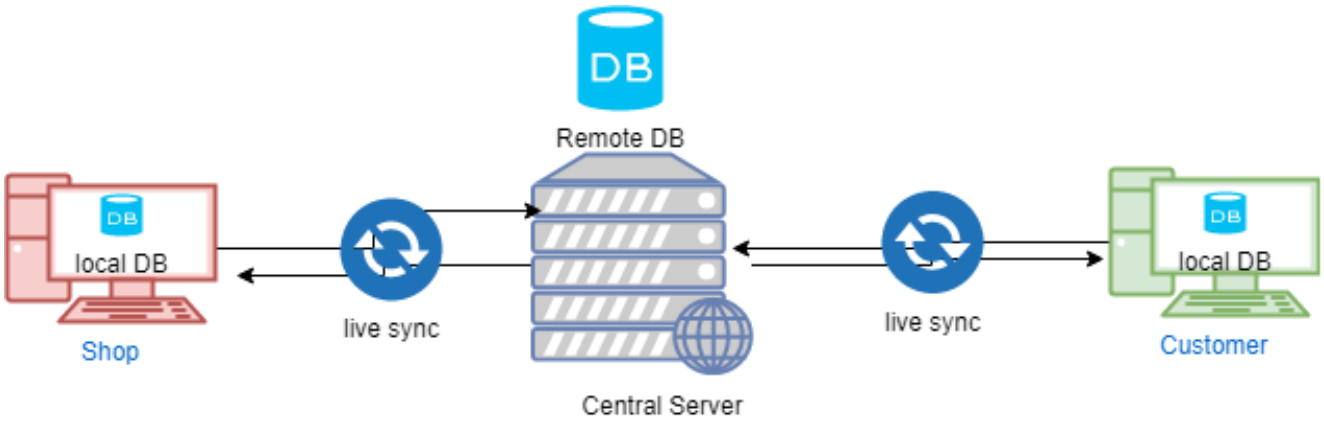


Рисунок 3 – Общая схема синхронизации баз данных

Подход синхронизации баз данных имеет несколько преимуществ:

– свобода по работе с данными. Так как в локальной базе имеются синхронизованные данные, ограничений по получению и изменению данных нет;

– возможность использовать данные в онлайн-режиме, то есть изменения, внесенные другим пользователем, будут применяться при синхронизации ко всем остальным клиентам;
– практически отсутствуют ограничения использования локального хранилища;
– при переносе источника данных на локальную базу, унифицируется доступ в онлайн- и офлайн-режимах к записям.

Также имеются у подхода и недостатки:

– сложность реализации. Основную сложность составляют некоторые случаи синхронизации с сервером, а также доступ к связанным данным и сохранение реляционной структуры базы;
– необходимость переноса или дублирования логики по работе с данными;
– тратятся ресурсы клиента – память при хранении и вычислительные мощности при выполнении логики получения нужных данных, что приводит к увеличению расхода заряда батареи, повышению температуры и снижение свободного места в памяти мобильного устройства.

1.4 Оценочная карта

Таблица 1 – Оценочная карта подходов организации бесперебойного доступа к данным

Подход	**Возможности работы с данными**	**Простота реализации**	**Робастность**	**Итого**
Расширение WIFI-сети	10	8	2	20
Кэширование запросов	6	7	4	17
Синхронизация баз данных	10	5	8	23

В ходе рассмотрения возможных подходов к решению проблемы бесперебойной работы с данными, представленному в таблице Таблица 1, основным критерием выступала работа с данными. В этом плане синхронизация баз данных предоставляет наиболее широкий спектр возможностей, тогда как кэширование запросов подходит только, если необходимых запросов конечное небольшое множество. Также синхронизация баз данных имеет высокий бал безотказной работы, то есть приложение с таким подходом будет работать в заметно большем количестве условий, что увеличивает целевую аудиторию.

Данный анализ позволяет определить задачу на разработку.

2 Задачи

2.1 Глобальная задача

Реализовать программное решение, синхронизирующее клиентскую и серверную базы данных, позволяющее управлять необходимыми приложению данными как в онлайн-, так и в офлайн-режимах.

Задача будет решаться на базе мобильного приложения, использующего React-native в качестве основного фреймворка, и серверного приложения, работающего на ASP.NET Core 6. База данных на сервере, подлежащая синхронизации, является реляционной и работает на СУБД PostgreSQL, выступающей самой продвинутой СУБД с открытым кодом [10].

Оценим возможный набор подзадач, предстоящих к выполнению в ходе разработки программного решения.

2.2 Локальные задачи

- Определить технологию локального хранилища.
- Реализовать экспорт структуры базы данных сервера.
- Реализовать разворачивания структуры базы в локальном хранилище.
- Реализовать доступ к данным из локального хранилища.
- Реализовать заполнение локального хранилища данными с сервера.
- Реализовать возможность изменения данных из локального хранилища.
- Реализовать создание сущностей, несущих в себе информацию об изменениях локальных данных.
- Организовать передачу данных между сервером и приложением.
- Реализовать синхронизацию данных с сервером.
- Разработать порядок разрешения конфликтов при синхронизации баз данных.
- Реализовать поддержку конфигурации синхронизации.

3 Аналог

Существует платный сервис Realm [11], разработанный компанией MongoDB [12]. Он представляет собой целую систему по работе с MongoDB [12] базами, синхронизацией, и многим сложным полезным функционалом, сильно упрощающим жизнь большой компании с большими данными.

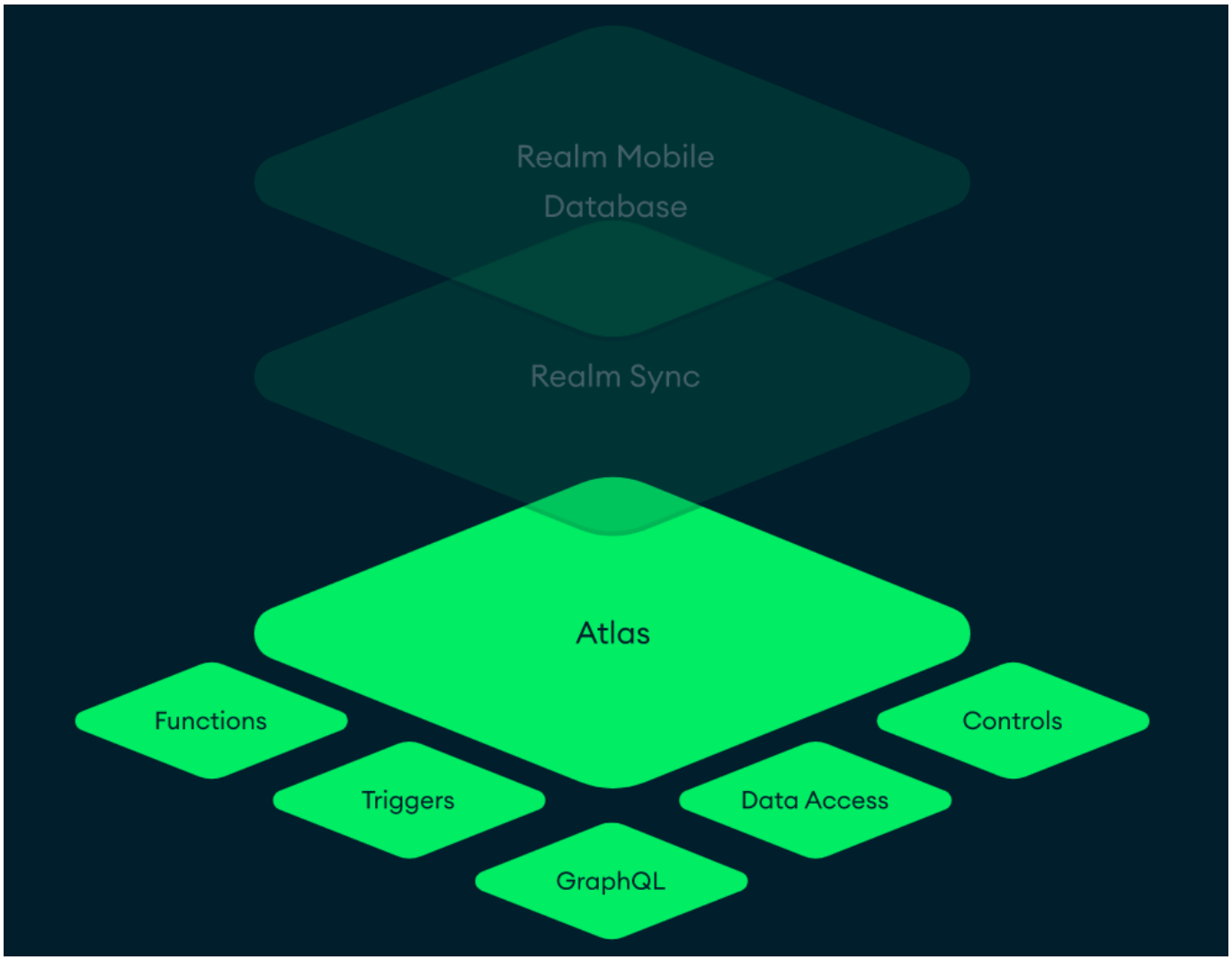


Рисунок 4 – Структурная схема устройства Realm [11]

Realm [11] включает в себя сервис Realm Sync [13], отвечающий за синхронизацию баз данных, работающих на MongoDB [12]. Сервис представлен в нескольких версиях библиотеки для разных платформ и языков программирования.

4 Разработка концепта программного решения

4.1 Структура

Для решения поставленной проблемы необходимо предоставить сервисы как на клиентской (мобильное приложение), так и на серверной части. Рассмотрим серверную часть, определив основные функциональные компоненты, необходимые для работы сервиса. Все детали реализации будут рассмотрены позже.

Прежде всего, необходимо предоставить мобильному приложению структуру базы данных. Для этого достаточно будет сервиса, получающего контекст базы через dependency injection [14]. Этот сервис будет по требованию создавать необходимую структуру данных и отдавать её запросчику.

Созданную сервисом структуру необходимо как-то доставить клиенту. В этом случае можно использовать контроллер, который сможет обработать запрос на получение схемы базы. По dependency injection [14] контроллер получает вышеописанный сервис, который после выполнения соответствующего метода отдаст желаемую схему.

Другой немаловажной частью функционала является сама синхронизация. То есть необходим сервис, способный применить те изменения, что пришли с клиентской стороны. Примем, что единичная запись, атомарно несущая в себе информацию об одной из операций изменения данных называется транзакцией. Также этот сервис должен отвечать за выдачу несинхронизированных транзакций для каждого клиента.

Чтобы синхронизация работала, необходимо обеспечить обмен информацией между клиентом и сервисом, отвечающим за транзакции. При том необходима поддержка возможности инициации передачи данных со стороны сервера, то есть необходимо устойчивое соединение, например на основе технологии websocket [15]. Дабы установить такое соединение, а также принимать и отправлять сообщения, потребуется сервис, ответственный за всю работу с websockets [15].

По описанным составным частям серверного решения составим структурную схему приложения и представим в рисунке Рисунок 5.

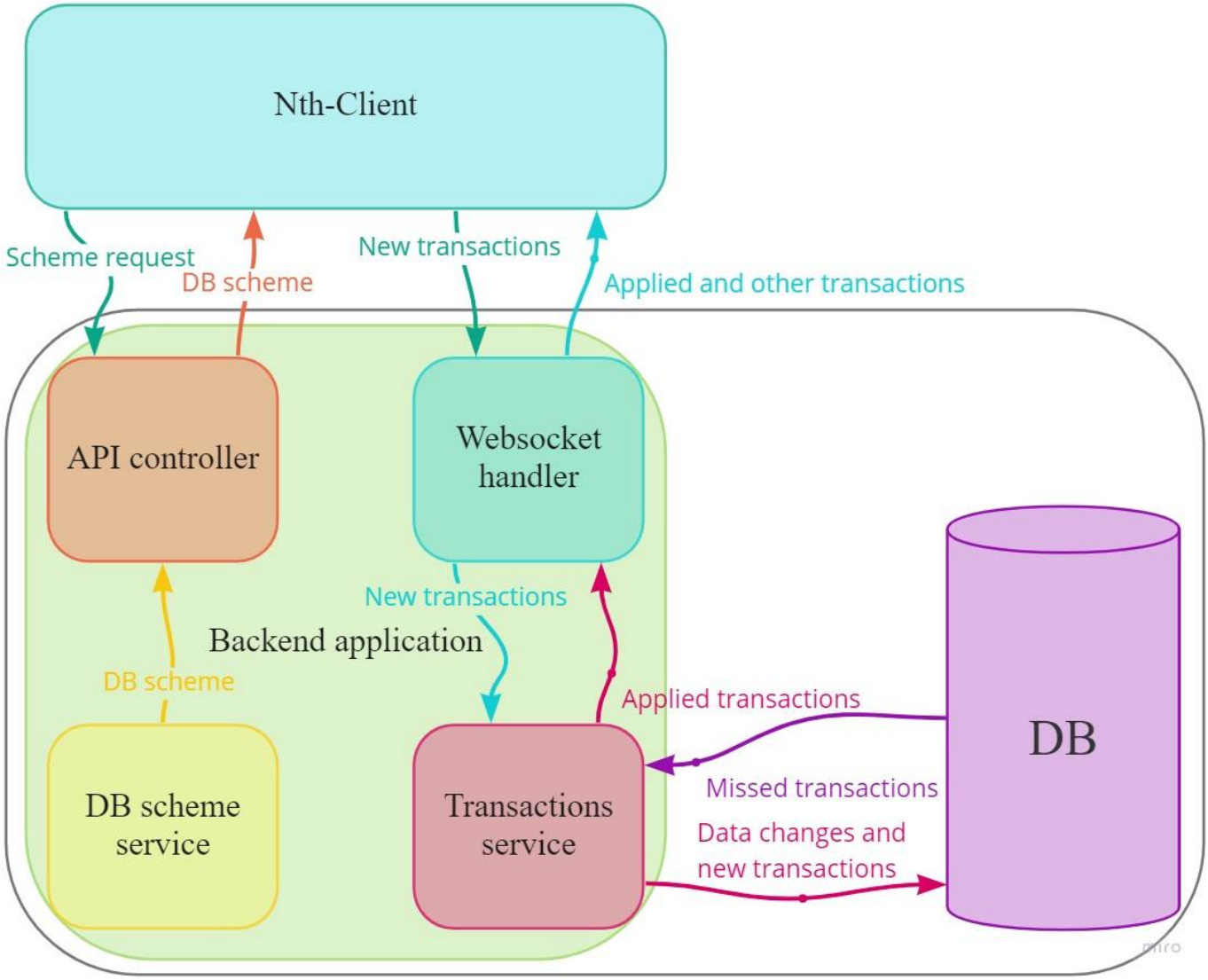


Рисунок 5 – Функциональная схема серверной части программного решения

Рассмотрим подобным образом клиентскую часть.

Исходя из подхода, который заключается в переносе источника данных в локальную область, необходимо само локальное хранилище и сервис по работе с ним. Данный сервис должен отвечать за получение записей, настройку запроса (сортировка, фильтрация) и операции изменения данных. Также должна быть возможность хранения и представления данных в структуре, диктуемой схемой с

сервера. Соответственно, сервису необходимо обладать функционалом по применению структуры к хранилищу.

Так как синхронизация баз происходит посредством транзакций, их необходимо как-то создавать. Логичнее всего иметь для этого прослойку между отданными пользователю данными, соответственно и операциями изменения, и сервисом по работе с локальным хранилищем. В этой прослойке должны будут обрабатываться изменения и записываться в транзакции, после чего они будут сохраняться в локальное хранилище. Затем они должны будут отправлены на сервер и синхронизированы с серверной базой. Выполняться это будет следующим сервисом.

Как было уже ранее определено в серверной части, клиент и сервер должны будут установить websocket соединение. Для этого удобно иметь отдельный сервис, отвечающий за подключение, прием сообщений от сервера и отправку на сервер.

Когда сервер присылает транзакции, их необходимо применить к локальному хранилищу. Это достаточно обособленный и сложный функционал, который может быть тоже вынесен в отдельный сервис.

Так как в структуре клиентской части задумано несколько отдельных самостоятельных сервисов, то необходим еще один – это сервис-контроллер. Этот сервис будет выступать как агрегатор, который может перенаправлять потоки данных от одного сервиса к другому, тем самым достигая желаемого результата по части синхронизации баз данных.

Составим функциональную схему клиентской части приложения.

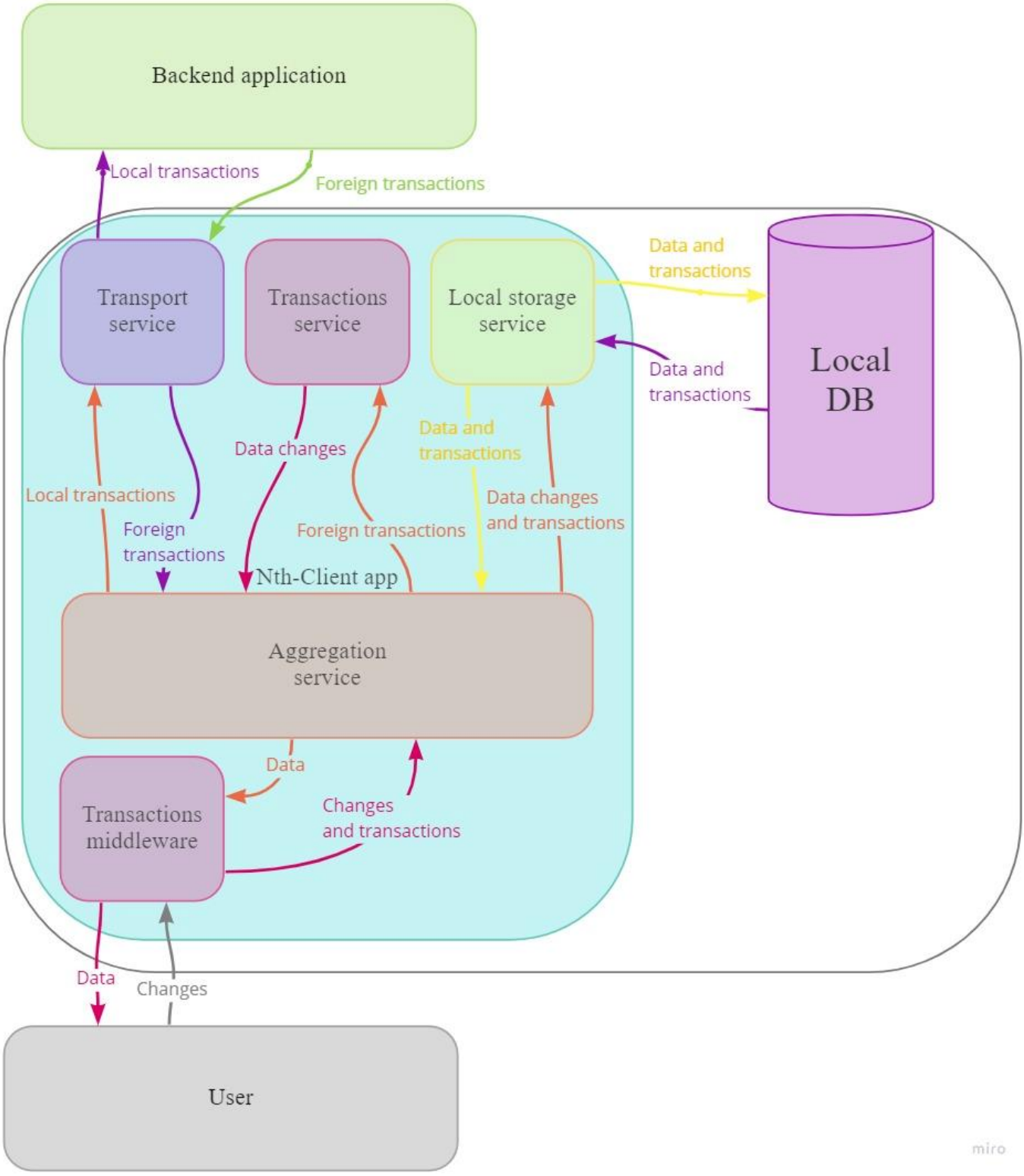


Рисунок 6 – Функциональная схема мобильной части программного решения

Определив все части разрабатываемого решения, можно переходить к их реализации.

4.2 Реализация программного решения на сервере

Рассмотрим реализацию в порядке рассмотрения структуры программы.

4.2.1 Сервис по получению схемы базы данных

Определим необходимые данные для клиента и, заодно, их структуру. Самое важное – описание списка таблиц. Здесь каждая схема таблицы должна содержать:

- имя таблицы;
- полное имя класса сущности, хранящейся в базе;
- имя сборки, в которой находится класс представляемый таблицей класс;
- список атрибутов, где каждый имеет:
    - имя атрибута;
    - схема, состоящая из:
        - тип атрибута (возможно, необходимо будет сделать конвертацию типов СУБД PostgreSQL в типы для хранилища на мобильном устройстве);
        - формат, если требуется для спецификации простого типа, например, только data (DateOnly), только время (TimeOnly) или дата со временем и указанием часового пояса (DateTime);
        - ссылка на детализацию типа, если он сложный, например, Enum;
    - флаг – является ли уникальным;
    - флаг – является ли ненулевым;
    - флаг – является ли первичным;
- список первичных ключей (список – для возможной поддержки составных ключей), в котором каждый элемент будет включать в себя:
    - имя атрибута;
- список связей таблицы, в котором каждая связь имеет следующие поля:
    - имя связанной таблицы;

- список связанных атрибутов таблицы;
- список атрибутов связанной таблицы;
- тип связи;
- какое ограничение применено на обновление связанных данных.

В случае со сложным типом, таким как Enum, необходимо предоставить к перечню описанных таблиц перечень схем таких сложных структур. Подобный подход используется при составлении swagger definition [16]. Определим структуру элементов коллекции схем:

- имя типа
- флаг – отличаются ли значения Enum от стандартной последовательности;
- словарь значений, где ключ – читаемое представление элемента Enum.

В дальнейшем, используя эту информацию, можно сгенерировать файл, который статически сможет использовать разработчик при работе с базой. То есть, это позволит автоматически типизировать всю работу с сущностями.

Для выполнения функционала по составлению данных структур, был создан и зарегистрирован сервис. Опишем его основной метод – получения структуры базы данных и составление выше объявленной структуры.

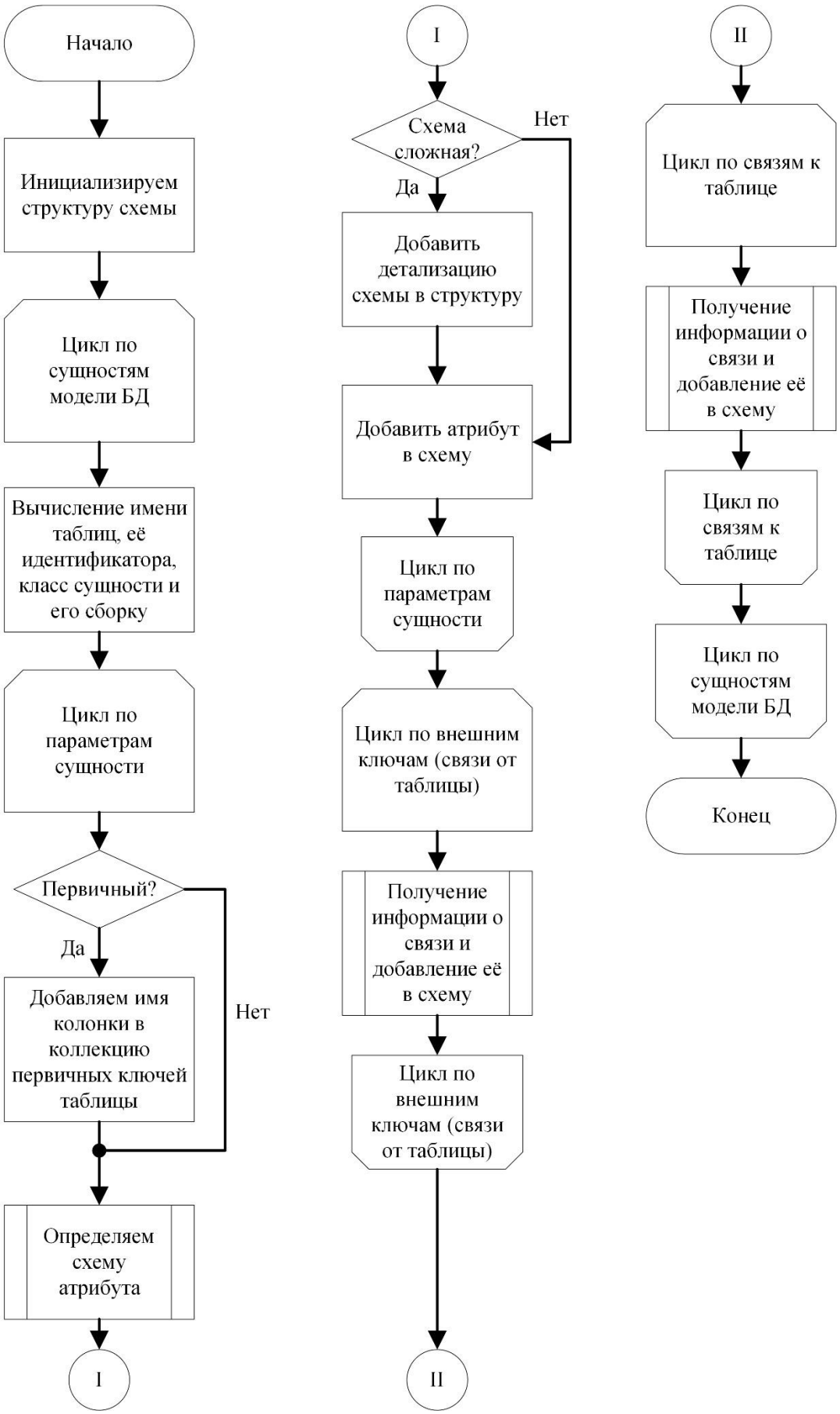


Рисунок 7 – Блок-схема получения структуры базы данных

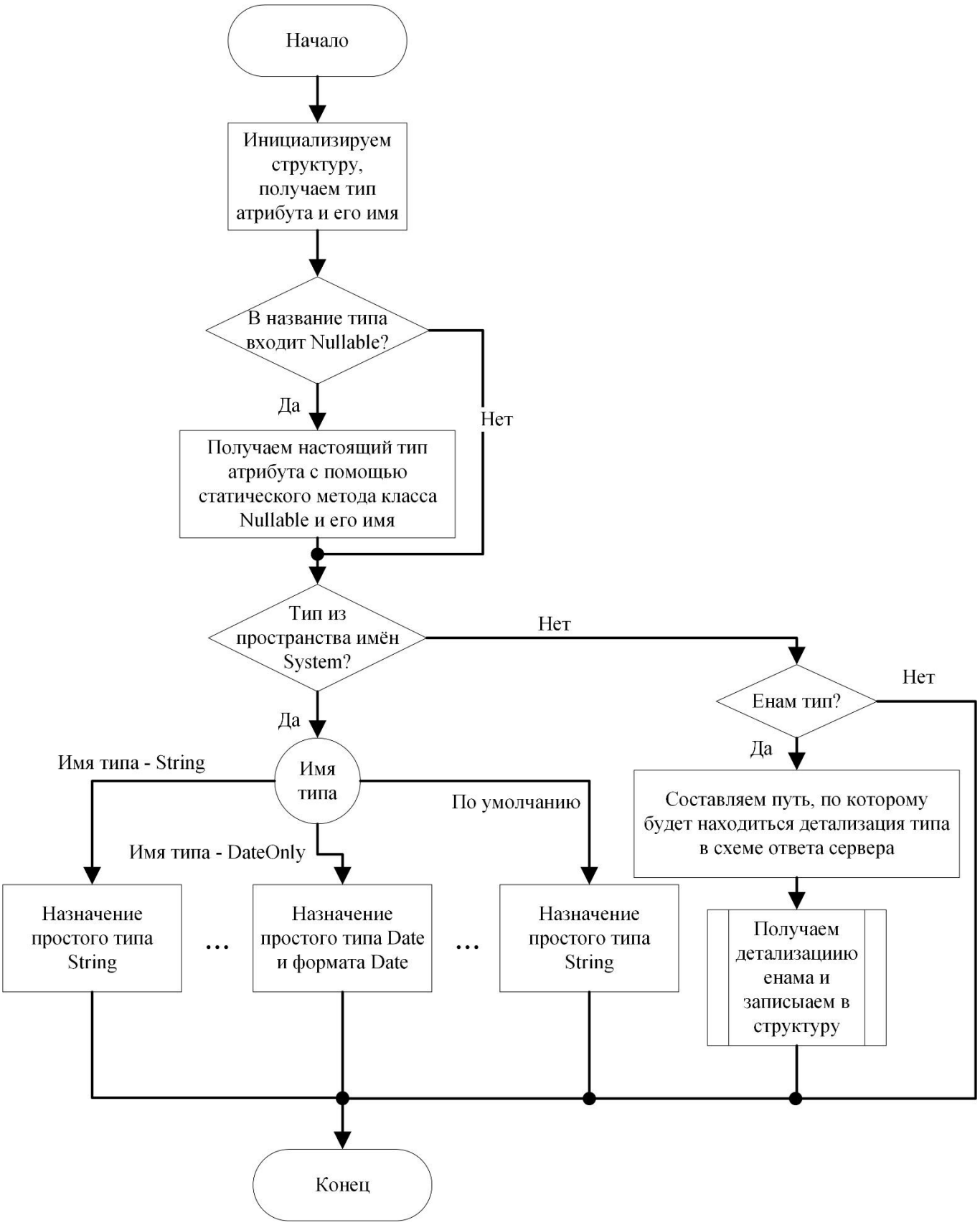


Рисунок 8 – Блок-схема получения схемы атрибута

Данный метод вызывается из контроллера по GET-запросу. Для получения схемы и правильного определения типов атрибутов программа была запущена в отладочном режиме и остановлена на сущности с самым широким по типизации набором параметров. Для каждого параметра было проведено исследование с целью поиска опоры для дальнейшего разделения типов и их обработки. Таким образом, получилось вычленить следующую информацию (все выписанные флаги здесь имеют истинное значение, а состав набора выписанных флагов был обусловлен собственным чувством пригодности того или иного флага):

- Int32 – целочисленный формат данных;
    - флаг – IsPrimitive;
    - флаг – IsValueType;
    - флаг – IsTypeDefinition;
- String – строковый тип данных;
    - флаг – IsTypeDefinition;
- DateTime – тип даты и времени с указанием часового пояса;
    - флаг – IsValueType;
    - флаг – IsTypeDefinition;
- DateOnly – тип даты без времени;
    - флаг – IsValueType;
    - флаг – IsTypeDefinition;
- Enum – специальный тип объекта с именованными полями, имеющими численные значения;
    - флаг – IsEnum;
    - флаг – IsValueType;
    - флаг – IsTypeDefinition.

При анализе полученной информации, стало ясно, что эффективно разделить типы по их данным не представляется возможным, но все же полезная информация имеется. С помощью флага IsEnum можно отделить группу пользовательских типов, что является удобным решением, ведь в другой группе остаются только системные

типы. Это значит, что, проверяя на принадлежности системному пространству имён, можно без проблем опираться на название типа.

Иной вопрос – получение данных пользовательского Enum. Для определения полезной информации было проделано подобное исследование. Ниже представлены выводы по получению полезной информации.

– На основе типа Enum значения в момент выполнения серверной программы можно получить коллекцию значений класса. Для этого используется операция:

$$\text{enumValues} = \text{enumType}.\,\text{GetEnumValues}(),$$

где $enumType$ – Clr [17] тип значения, несущего в себе один из полей класса Enum.

– Чтобы правильно определить значения класса, можно использовать классический цикл for. Таким образом, для каждого элемента будет определен последовательный целочисленный член, что отражает стандартное поведение такого типа, как Enum. При этом в каждой итерации цикла можно получить настоящее значение элемента класса через трансформацию имени поля Enum в значение через подлежащий тип. Выполняется это за счет операции:

$$\text{name} = \text{enumValues}.\,\text{GetValue(i)},$$

$$\text{intValue} = \text{Convert}.\,\text{ChangeType}\big(\text{name}, \text{Enum}.\,\text{GetUnderlyingType}(\text{enumType})\big),$$

где $name$ – строковое представление поля Enum;

$enumType$ – Clr [17] тип значения, несущего в себе один из полей класса Enum;

intValue – численное значение поля класса Enum.

И если они в какой-то итерации разнятся, то можно записать в схему, что значения в Enum нестандартные. На это необходимо будет опираться клиенту при составлении типизации базы, о которой говорилось ранее.

– Записываются ключ-значение в соответствующий словарь. Здесь ключом выступает строковое представление значения, а значением пары является численное представление.

С реализацией получения схемы базы на этом этапе всё уже понятно. Следующий немаловажный этап – сервис, отвечающий за websockets и связанный с ним сервис транзакций.

4.2.2 Сервис по управлению транзакциями на сервере

Для Asp.Net Core [2] компания Microsoft предоставляет пакет SignalR [18], работающий в основном на websockets [15]. Данный пакет предоставляет возможность обмена данными с указанием конкретных обработчиков, которые должны будут вызваны на клиентской части.

На сервере вся работа с SignalR [18] ведется через специальные контроллеры – хабы. Каждому хабу назначается свой путь, по которому подключаются клиенты. Внутри хаба реализованы асинхронные методы, и клиент может вызывать их, используя названия методов хаба. При этом порядок работы с данными запроса остаётся таким же, как и при работе с обычным контроллером, то же касается и dependency injection [14].

В хабе должен присутствовать самый важный метод, отвечающий за синхронизацию транзакций, то есть их приём и вызов сервиса, ответственного за работу с транзакциями, а затем еще и отправку остальным клиентам новых транзакций.

Прежде, чем перейти к реализации, рассмотрим – что должна в себя включать транзакция, чтобы иметь возможность применения как на стороне сервера, так и на стороне клиента.

Прежде всего – название таблицы, то есть куда сохранять в конечном итоге. Необходима графа изменений, которая нужна не только при конкретных изменениях, но и при создании она может включать в себя весь контент записи. В случае удаления, графа может включать пустое значение. Для быстрого и корректного доступа к идентификатору затронутой записи, эту информацию можно вынести в отдельное

поле. При этом, когда составляется транзакция для создания записи пусть туда дублируется информация с идентификатором из графы изменений.

В момент реализации и апробации возникла проблема с определением класса, с которым необходимо было работать, то есть создавать экземпляры, определять содержимое класса и применять изменения, после чего было принято решение добавить для каждой таблицы полное имя класса, включающее версию, уникальный идентификатор, а также имя сборки, включающей в себя этот класс. Имя сборки необходимо для поддержки использование класса, хранящегося вне текущего решения. Это может быть как структурный приём организации серверного приложения, так и использование класса со сторонней библиотеки. Эти данные были включены в схему каждой таблицы, собиравшейся сервисом, описанным выше. Таким образом, для мобильного приложения и мобильной базы данная информация является бесполезной, она нужна для того, чтобы вложить её в транзакцию при отправке на сервер, чтобы уже в момент синхронизации взялся именно тот класс, что нужен.

Транзакции еще необходимо назначить какое именно она в себе несет изменение, то есть что-то из: изменение, создание, удаление. Также удобно было бы иметь даты создания и синхронизации. Таким образом, структура транзакции:

- идентификатор транзакции;
- имя измененной таблицы;
- имя класса сущности (в полном формате);
- имя сборки, включающей в себя класс сущности;
- словарь изменений, внесенных в таблицу;
- тип изменения – изменение, удаление, создание;
- идентификатор записи, к которой относится транзакция;
- дата создания транзакции;
- дата синхронизации с базой на сервере.

Рассмотрим сервис по применению транзакций, непосредственно. Основным методом здесь выступает получение транзакций и применение их.

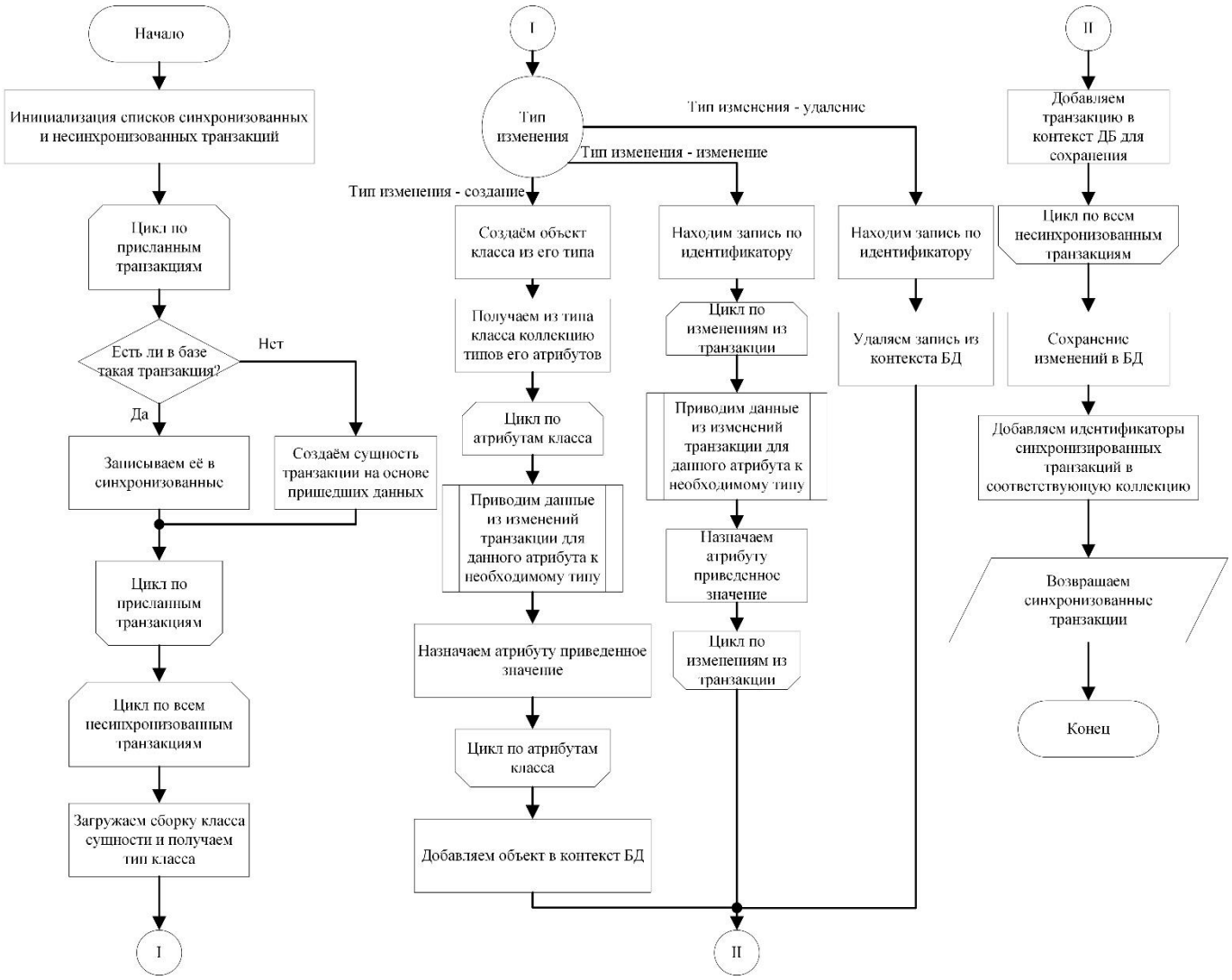


Рисунок 9 – Блок-схема метода сервиса по применению транзакций

Для начала, надо разделить на примененные уже транзакции и новые. Конечно, клиент вряд ли пришлёт уже синхронизованные транзакции, но в любом случае, это надо обработать. Далее для каждой транзакции, требующей синхронизации, загружается сборка с тем классом, что указан, в сущности, этой транзакции. Это дешевая операция, потому что загрузка файла происходит только первый раз, во все остальные разы возвращается уже вычисленное значение. Из загруженной сборки получается Clr [17] тип класса, который, в свою очередь, позволяет получить информацию о содержимом класса, его структуре, а также инстанциировать. Затем, в зависимости от того, какой тип изменения был совершен выполняются действия:

– при создании – инстанциируется объект класса с помощью системного класса Activator [19], из типа класса достаются его поля и после каждое поле задаётся данными транзакции. Для того, чтобы entity framework [20] обнаружил изменения и перенес их в базу, необходимо добавить объект в контекст соответствующим методом Add;
– при изменении – находится объект, соответствующий типу класса и имеющий идентификатор из транзакции, это можно сделать с помощью метода Find. Затем итерируются изменения из транзакции, для каждого из них получается тип поля класса, который позволяет назначить данное поле в экземпляре класса. Данный объект можно не добавлять в контекст базы данных, потому как метод Find возвращает привязанный к контексту объект;
– при удалении – находим объект точно так же, как и при изменении и удаляем объект из контекста.

На данном этапе все необходимые изменения, что несет транзакция, перенесены в базу. Теперь можно её и сохранить. Также все применённые транзакции собираются и отправляются остальным подключенным клиентам, а тому клиенту, который синхронизировал эти транзакции, отправляются идентификаторы, чтобы уведомить его об успешности.

В данном методе очень важен функционал приведения данных к нужному типу, ведь C# – один из языков программирования, имеющих типизацию во время выполнения, а это значит, что при неправильном типе данных, несовместимым с выделенной областью памяти, будет вызвано исключение.

Другой немаловажный функционал данного сервиса – нахождение списка транзакций для клиента, необходимых для актуализации его базы. Для этого можно переопределить родительский метод хаба, позволяющий выполнять код при подключении клиента. Примем, что при составлении адреса подключения, он будет добавлять параметр запроса, имеющий значение идентификатора последней сохраненной транзакции. В таком случае, хаб вызовет метод сервиса транзакций, который вернет список синхронизованных на сервере, начиная с данной.

Параметры запроса не приходят в хаб напрямую как аргументы метода, как это было бы с методом контроллера, поэтому, чтобы работать с ними, необходимо обратиться к контексту запроса и там найти необходимы параметр с определенным именем, отвечающим за последний синхронизованный идентификатор.

Рисунок 10 – Блок-схема метода сервиса по работе с транзакциями, вызываемого при подключении клиента к хабу

Также этот механизм можно использовать как временное решение по скачиванию данных с сервера при первом старте. В таком случае, идентификатора передано не будет и транзакции просто возьмутся все. Это решение временное, так как по мере работы приложения, транзакции будут сильно разрастаться, а если их чистить, теряется смысл вообще такого функционала, да и может приводить к ошибкам, когда какие-то данные для клиента были утеряны. Далее планируется выделение отдельного контроллера для реализации скачивания данных без транзакций при первом подключении.

4.3 Реализация программного решения на клиенте

4.3.1 Локальное хранилище

В качестве локального хранилища на мобильном устройстве могут выступать:

- SQLite;
- Realm;
- AsyncStorage.

От выбора хранилища зависит реализация сервиса по работе с ним. Так, например, у Realm [11] есть свой интерфейс ORM [21], тогда как для SQLite на момент разработки ничего актуального готового не было. Для AsyncStorage [7] не так важен способ хранения, так как хранилище построено по принципу ключ-значение.

В случае с SQLite, необходимо реализовать генератор запросов для внесения изменений в базу, а также интерфейс по построению запросов на получение данных с фильтрацией и сортировкой. Другими словами, реализовать свою ORM [21] для SQLite [22].

Для Realm вместе с хранилищем предоставляется готовый интерфейс по работе с получением данных, с их записью и изменением. К тому же Realm [11] предоставляет возможность использования подписок на изменение данных. Такой функционал может решить проблему необходимости промежуточного слоя, который

отвечает за создание транзакций. Потому как этот данных в такого рода подписках достаточно для создания транзакций.

AsyncStorage [7] не позволяет по своему устройству модифицировать запрос к данным, то есть, полученную коллекцию данных необходимо будет итерировать непосредственно в коде, это сильно снизит производительность приложения.

Составим оценочную карту и представим ее в таблице Таблица 2.

Таблица 2 – Оценочная карта технологии локального хранилища

Хранилище	Возможности запросов	Предоставленный интерфейс	Вместимость	Качество поддержки разработчиком	Итог
SQLite [22]	10	6	10	5	31
Realm [11]	8	10	10	10	38
AsyncStorage [7]	3	4	7	8	22

По результатам, полученным в таблице выше, наилучшим вариантом выступает Realm [11] как локальное хранилище данных на мобильном устройстве. Основным преимуществом данной технологии является предоставляемый интерфейс, включающий в себя весь механизм построения запросов с помощью кода. К тому же, библиотека реализована на c++ и хранит данные с индексацией в виде b-деревьев [23], что позволяет достигать высокой производительности.

На основе данной технологии будет строиться вся дальнейшая работа. Рассмотрим сервис по работе с данными.

4.3.2 Сервис по работе с хранилищем

Чтобы хранилище работало, необходимо ему передать схему. Такая схема привязывается к предоставляемому пути (где хранится около 5 файлов для каждого специфицированного пути в приложении). Удобно, в таком случае, иметь сервис,

который будет отвечать за схему базы, а также за открытие подключения к базе и переподключение.

Схема должна быть составлена для локальной базы в формате, определенном Realm [11], так в сервисе определяется подходящий тип на основе предоставленного структурой базы с сервера. Также для реализации реляционности необходимо добавить свойства навигации. Для таких полей также добавляется поддержка обратного соединения. Это значит, что при попытке получения связанной записи, будет выдан точно тот же объект, что и вернулся бы для простого получения данной записи по идентификатору.

В общем, в данном сервисе реализовано приведение схемы с сервера в формат схемы для локальной базы. При этом, для реализации функционала приложения необходима еще одна таблица, которой нет на сервере, и она не может быть получена из присланной схемы. Данная таблица содержит только идентификатор последней синхронизированной транзакции с сервером.

Для подключения к базе был реализован метод, проверяющий активность соединения и его наличие. Так при открытом соединении будет возвращен ранее открытый объект базы.

Данный сервис также включает в себя надстройку над транзакциями записи в базу. Realm [11] требует, чтобы все изменения проходили только через его транзакции, которые он добавляет к себе в очередь на выполнение. Но иногда, возникают ошибки по типу: невозможно добавить транзакцию в очередь так как база уже в транзакции. Для таких случаев у объекта базы присутствует свойство, позволяющее проверить - в транзакции ли она находится. Здесь, если транзакция уже выполняется, просто выполняется переданная функция, изменяющая данные, иначе же, добавляется транзакция с такой же функцией в очередь.

4.3.3 Сервис по работе с данными

Необходимо было предоставить интерфейс как надстройку над Realm [11], чтобы сохранить весь функционал запросов. Как уже отмечалось ранее, Realm [11] имеет удобный интерфейс работы с данными.

Лучше всего предоставлять разработчику инструмент с привычным для данного стека технологий интерфейсом. В react [1] принято использовать хуки [24] для динамических данных. В таком случае, можно объединить подходы Realm [11] и хуков [24] из react.

Был создан универсальный хук, позволяющий получить данные с таким же функционалом, как и оригинальный метод Realm [11], но при этом внутри он имеет подписку на изменение предоставляемых данных. Эта подписка необходима для создания транзакций, там вычисляется разница данных. Как и везде, есть некоторые нюансы, усложняющие реализацию. Рассмотрим ряд особенностей и соответствующих решений, принятых при реализации библиотеки:

– подписка на обновление данных срабатывает уже после применения изменений, а Realm [11] так устроен, что выданные данные локально изменяются моментально. Другими словами, данные до изменений взять неоткуда. А они нужны для вычисления изменений и записи их в транзакции. Таким образом, было принято решение о создании дублирующей базы с точно такой же схемой, только не включающей в себя таблицы последней синхронизованной транзакции и сами транзакции. Идея такова, что с каждым изменением основной базы, предназначенной для отображения и работы с данными, мы получаем все те записи, что были затронуты, но уже из дублирующей базы. После чего анализируем разницу и создаём транзакцию. После чего, как изменения уже вычислены, эти изменения применяются и к дублирующим данным, так они будут обновляться вместе с основными данными. Здесь есть некоторые сложности. Заключаются они в том, что при удалении, как уже сказано было ранее, Realm [11] все изменения моментально доводит до всех используемых приложением данных, теряются какие-либо записи об этих объектах. Это значит, что нельзя наверняка определить с каким

идентификатором была удалена запись. Для такого рода изменений Realm [11] предоставляет индексы тех записей, что были удалены из коллекции, на изменения в которой была сделана подписка. В этих случаях мы пытаемся получить эти записи из базы-дублёра. Это будет успешно в том случае, если все изменения будут проходить только через созданную библиотеку. Так, для разработчика вводится правило, что компонент, собирающийся изменять данные, должен быть подписан хуками на все возможные для изменения таблицы;

- при добавлении фильтрации для получения данных с хука, и подписываясь при на изменения этой коллекции, изменения приходят некорректные. Например, если продукты отфильтрованы по цвету «красный», будет возвращена коллекция продуктов только со значением «красный» в поле цвет. Но если для каких-либо записей значение «красный» поменять на другое, например, «зеленый» - то Realm [11] воспримет это как удаление записи для текущей коллекции и в обработчик изменений будет прислан идентификатор будто запись удалена. То же работает и с изменением для существующей записи значений цвета на «красный», только в этом случае запись будет числиться как добавленная. Чтобы избежать такое нечеткое поведение с изменениями данных, было принято решение подписываться только на «чистую» коллекцию объектов, то есть до применения всех возможных модификаторов выдачи. Таким образом, все приходящие данные об изменениях всегда будут означать одно и то же и будут отражать реальные операции удаления, добавления и модификации;
- при применении транзакций, пришедших с сервера, в сервисе, который будет описан далее, может быть так, что в приложении присутствуют активные подписки на таблицы, на которые нацелены транзакции. При их применении будут вызваны все те функции, которые вызываются при обычном изменении. Это означает, что для этих изменений будут снова созданы транзакции с теми же данными об изменениях, что были приняты с сервера. Решено это было

регистрацией подписок. Так, при инициализации этот хук регистрируется в сервисе по работе с транзакциями, реализация там будет описана далее, но здесь отлавливаются эти изменения и транзакции не создаются.

Таким образом, хук позволяет получить данные, настроить выборку и при этом будут отловлены все изменения данных таблицы, на которую подписан хук. Будут созданы соответствующие транзакции и при изменении данных в базе, будет вызван рендер компонента, чтобы отобразить изменения. Транзакции будут сохраняться локально, а только затем отправляться на сервер для синхронизации. Так делается для поддержки офлайн-режима. Таким образом, если пользователь не имеет возможности подключиться к серверу, транзакции будут сохранены локально и отправятся как будет возможность. При этом хранить уже синхронизованные транзакции нет необходимости. Также для того, чтобы отлавливать изменения в связанных данных, не отображаемых на экране, добавлена возможность отключения вызова рендера при изменении данных конкретного хука, то есть таблицы, это повысит производительность при внесении изменений в данные.

4.3.4 Сервис по работе с транзакциями

Основной целью данного сервиса выступает применение транзакций, полученных от других пользователей с сервера, локально. Когда с хаба приходит на клиент список транзакций, подлежащих синхронизации, вызывается метод сервиса. Также при получении транзакций с сервера, локально могут быть неотправленные изменения, к тому же, это происходит каждый раз, когда пользователь подключается к серверу. То есть при подключении к хабу клиенту запрещено отправлять транзакции до тех пор, пока хаб не пришлет ответ по поводу восстановления всех изменений, что он пропустил за время отсутствия подключения. В этом случае необходимо обработать локальные транзакции, чтобы не навредить данным сервиса и другим пользователям.

Рассмотрим ряд решений, касаемо данного сервиса:

– синхронизация клиентской и серверной баз происходит посредством сравнения списка транзакций и восстановлением их в двунаправленном порядке. Подобный подход позволит офлайн вносить изменения в данные, сразу отображая изменения, а по подключению к серверу, отправлять недостающие на сервере транзакции (локально совершенные операции). Отправленные транзакции применятся к базе на сервере, а так как список транзакций сервера пополнится, остальные пользователи тоже получат изменения, только в обратном порядке. То есть, один пользователь добавляет операции в базу на сервере, теперь другие пользователи, синхронизующие базу, получают эти изменения от сервера и применяют их к своим данным, например, по websocket-соединению, или периодически запрашивая обновления. Здесь может возникнуть вопрос с уникальностью идентификаторов записей в синхронизируемых сущностях, но решается это требованием использования строкового уникального идентификатора для записей, например, GUID;
– при одновременной работе нескольких пользователей офлайн с последующей синхронизацией, может возникнуть ситуация наложения изменений для одной записи. Чтобы конфликт можно было определить, транзакция имеет соответствующие данные – изменяемая сущность, произведённая операция, затронутая запись в сущности (транзакции атомарны по отношению к записям), список затронутых колонок. В таком случае, решено отдавать приоритет транзакции, пришедшей с сервера, это значит, изменения первого синхронизирующего будут выше по приоритету. Для пользователя, чьи транзакции к данной записи не имеют транзакций, пришедших для синхронизации ранее, ситуация простая и заключается в применении новых операций в локальную базу. Проблемы появляются, когда у пользователя имеются локальные изменения, которые затрагивают не только конфликтные поля, но и другие атрибуты, поэтому просто опустить их не получится и необходимо было их обработать, чтобы при синхронизации, они не внесли

неправильные изменения как на сервере, так и у других клиентов. Далее будет рассмотрен порядок решения конфликта на основе подхода, где сервер всегда прав, а локальные транзакции перерабатываются в бесконфликтные.

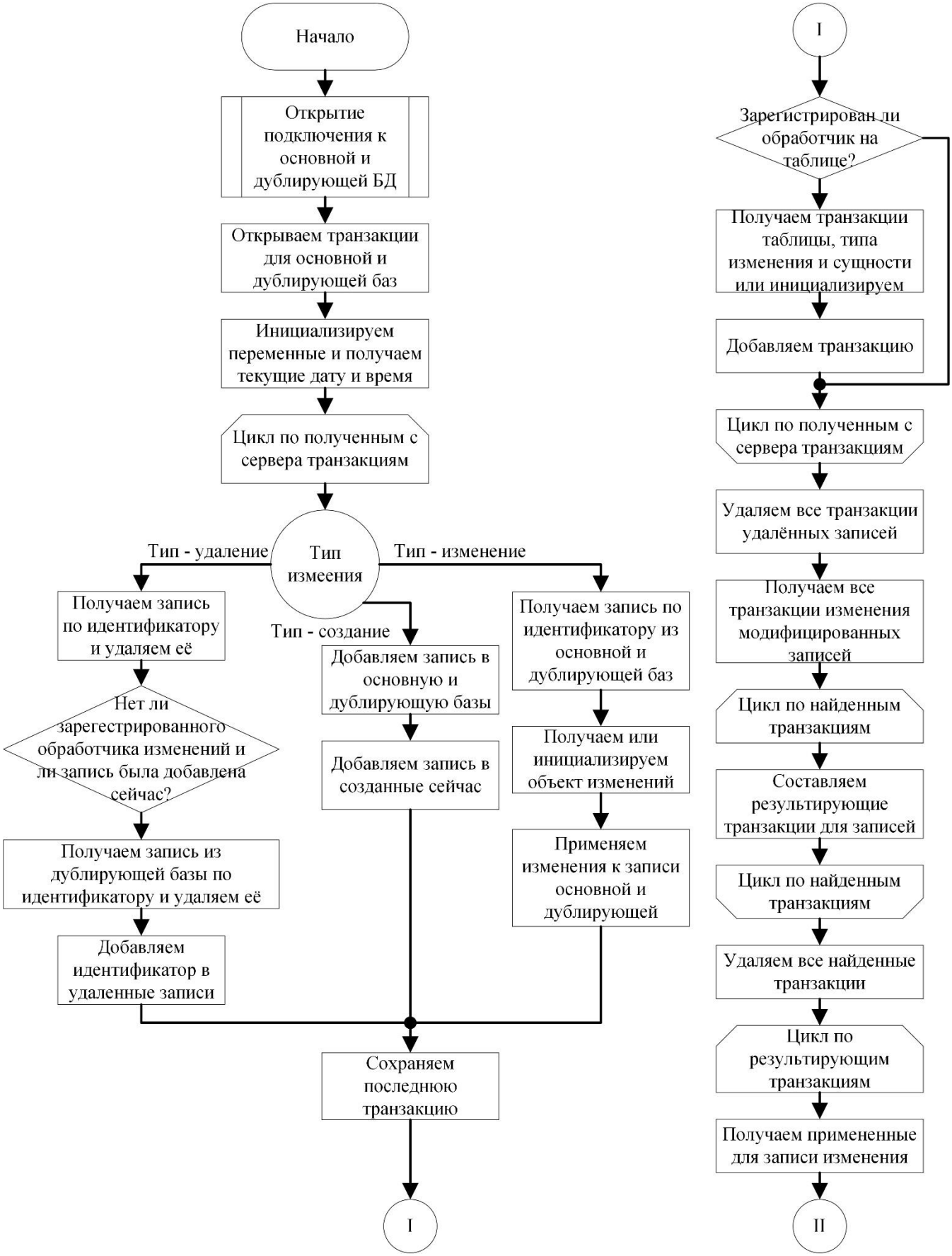


Рисунок 11 – 1я часть блок-схемы метода по применению транзакций с сервера

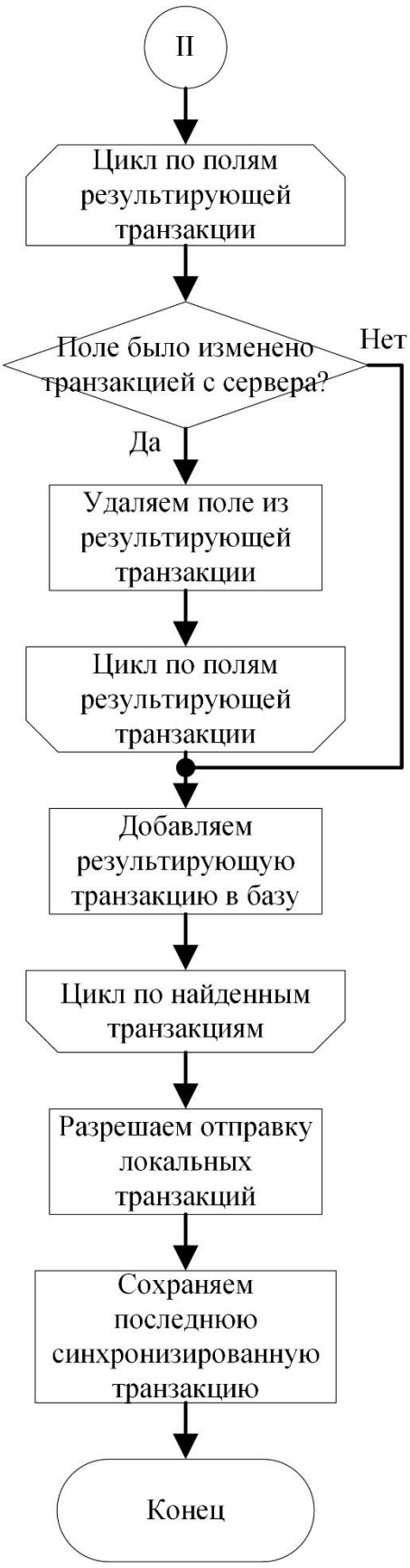


Рисунок 12 – 2я часть блок-схемы по применению транзакций с сервера

Помимо того, что в методе применяются изменения к основным данных, необходимо ещё применить эти изменения к дублирующим данным, для поддержания правильного их состояния, ведь при дальнейших изменениях в основных данных стандартным путём, разница будет вычислена неверно. Здесь есть очень важный нюанс, о котором говорилось ранее, что нельзя получить доступ к данным после их удаления, никоим образом. А для того, чтобы в рассмотренном хуке отловить изменение касаемо удаления с сервера, необходимо знать идентификатор удаленной записи, который можно получить из дублирующих данных. В таком случае нельзя применять операцию удаления для дублирующих данных непосредственно в сервисе после применения к основным. Решением будет успешное определение изменения от транзакции в хуке и применение удаления к дублирующим данным там. Это разносит применение изменений в дублирующую базу в 2 места, но это жертва, на которую пришлось пойти. Остальные изменения применяются при синхронизации. Также может получиться так, что локально присутствуют не синхронизованные транзакции, с которыми может возникнуть конфликт. Здесь было принято решение отдавать приоритет изменениям с сервера, а только затем синхронизировать оставшиеся локальные изменения с сервером. Для этого необходимо будет проанализировать пришедшие транзакции и затем менять локальные. Рассмотрим – как это было реализовано.

Итерируются транзакции, и в зависимости от типа изменения совершаются действия:

- при удалении нам необходимо собрать все идентификаторы записей, ведь, если сервер прав, а с сервера пришло то, что запись удалена, то и локально, помимо самой записи, все транзакции для этой записи необходимо стереть. Здесь же удаляется запись и из дублирующих данных в случае, если нет обработчика изменений для данной таблицы или мы сами не добавили ранее эту запись транзакцией, а следом удаляем;
- при добавлении вносятся изменения сразу в основную и дублирующую базы, также запоминается какие записи мы именно добавили, чтобы в случае чего

при удалении узнать записи, которые не будут обработаны подпиской на изменения (так как они внутри одной транзакции были добавлены и удалены);

– при изменении вместе с тем, как модифицируются данные основной и дублирующей баз, для конкретных записей складывается их результирующее изменение, для каждой свой результат.

Затем происходит удаление транзакций для уже удалённых записей. После этого получаем все транзакции изменений, которые были сделаны над записями, затронутыми текущей синхронизацией в плане модификации. Это те транзакции, с которыми необходимо будет работать.

Чтобы достичь желаемого – не нарушить концепцию, где сервер прав, можно выделить 2 подхода:

– для каждой транзакции убрать те изменения, которые конфликтуют с серверными;

– собрать все транзакции в результирующую, у неё удалить конфликтные поля и удалить все транзакции, добавив только результирующую.

Был выбран 2й подход из-за большей производительности. Так, проходятся все транзакции, полученные ранее для измененных записей, и для каждой записи составляется новая, результирующая, транзакция. Затем, удаляются все полученные транзакции, кроме только что созданных. Теперь они итерируются и для каждой получаются изменения с сервера, которые были собраны в результирующем виде ранее и из новой транзакции удаляются все поля изменений, присутствующих в изменениях с сервера. На этом этапе новые транзакции готовы, так что они добавляются в базу.

В конце данного метода, отвечающего за синхронизацию с сервером, сохраняется идентификатор последней синхронизованной транзакции, чтобы при дальнейшем подключении отсчет пропущенных транзакций начинался с неё. На этом синхронизация с сервером завершена.

4.3.5 Сервис по работе с хабом сервера

Данный сервис необходим для приёма сообщений с сервера и отправки на него. Клиент реализован в сервисе с помощью библиотеки "@microsoft/signalr" [25], специально предоставляемой Microsoft для работы с хабами signalr [18].

У сервера был заранее задан путь до хаба, по нему подключается клиент. Так как это не совсем обычный запрос, тут не получится передать что-то в body или header, а передать идентификатор последней записи надо. В таком случае, рекомендуется добавлять к адресу хаба параметры запроса, работа с которыми уже была рассмотрена ранее в серверной части. Изменение данного параметра запроса происходит во время разрыва соединения для того, чтобы при его восстановлении получить актуальные изменения.

Совместно с сервисом по работе с локальной базой была организована отправка локальных транзакций с применением ограничения на частоту отправки. Здесь используется уже известный механизм подписок на изменения в данных сущности, здесь – на изменения в транзакциях. Нам важны операции добавления и изменения, реакции на остальные изменения можно опустить. Каждое изменение применяется функция, ограниченная по частоте, это значит, что функция будет выполняться не чаще, чем задано разработчиком и выполняться она будет с аргументами последней попытки вызова на момент выполнения. Таким образом, часто добавляемые транзакции будут накапливаться и отправляться группой за раз. Также стоит отметить, что отправка возможна только после первого ответа сервера, чтобы обработать возможные конфликты. Также отправка осуществляется группами до 30 транзакций, если в одно время для отправки подготовлено более 30 транзакций, то сверх данного порога отправятся в следующий вызов метода. Это необходимо для уменьшения задержки получения изменений с сервера.

При отправке транзакций, создаются специальные объекты, куда включается информация о классах сущностей и их сборках, которые уже необходимы серверу для корректного применения изменений. Метод хаба, вызываемый на сервере,

возвращает идентификаторы синхронизованных транзакций, которые необходимо удалить на клиенте, ведь они уже синхронизованы и больше не нужны. Также сохраняется и последняя синхронизованная транзакция, даже при том, что она была создана нами.

4.4 Тестирование и апробация

Разработка велась на мобильном устройстве непосредственно с имитацией других пользователей. Имитация реализовывалась временным методом хаба, генерирующим транзакции и отсылающим их на клиент. Также тестировалось локально создание транзакций, их отправка и применение. Стоит отметить также, что на период тестирования удаление транзакций отключалось для упрощения отладки программы. Также во всем тестировании данные записывались и производились только от клиента, не включая имитацию транзакций других пользователей. Все данные в базе сервера принадлежат только клиенту.

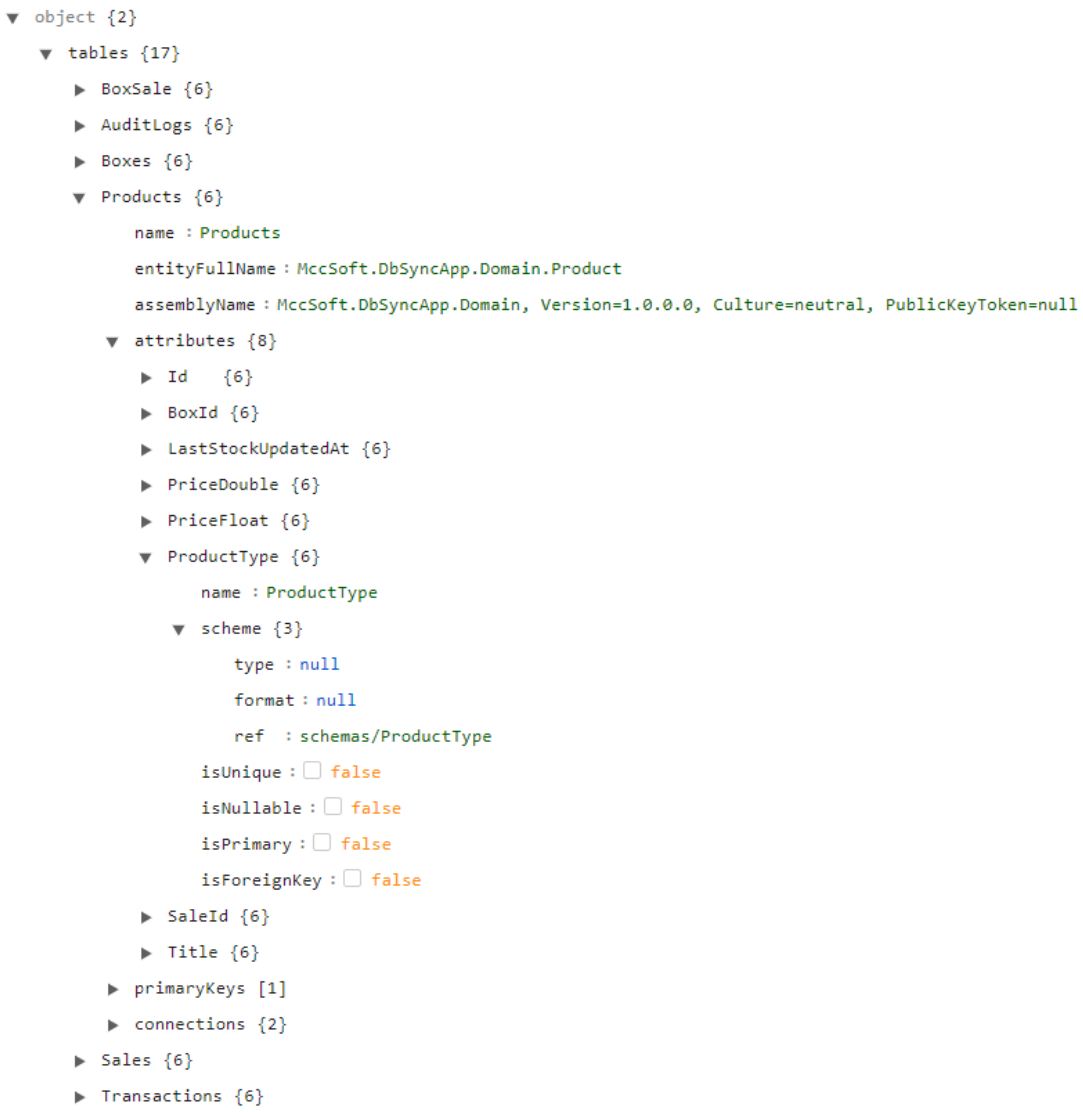


Рисунок 13 – Пример экспорта структуры базы данных сервером

На рисунке Рисунок 13 изображен снимок экрана, показывающий пример ответа сервера по пути получения схемы базы. Тут представлен список таблиц, у каждой описан класс сущности, название, связи и колонки.

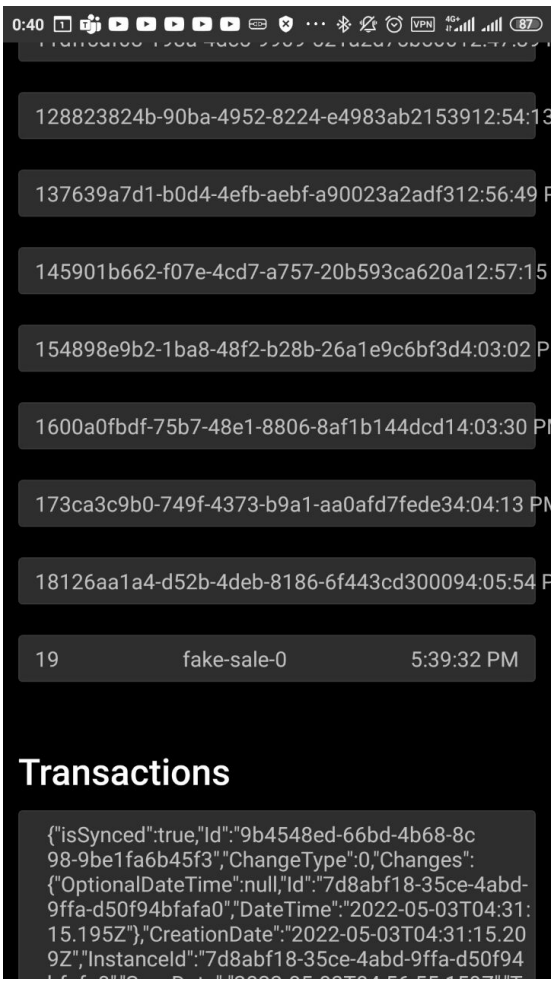


Рисунок 14 – Снимок экрана телефона с данными

На рисунке Рисунок 14 изображены записи из локальной базы мобильного устройства. Также под номером 19 присутствует синхронизированная запись с сервера.

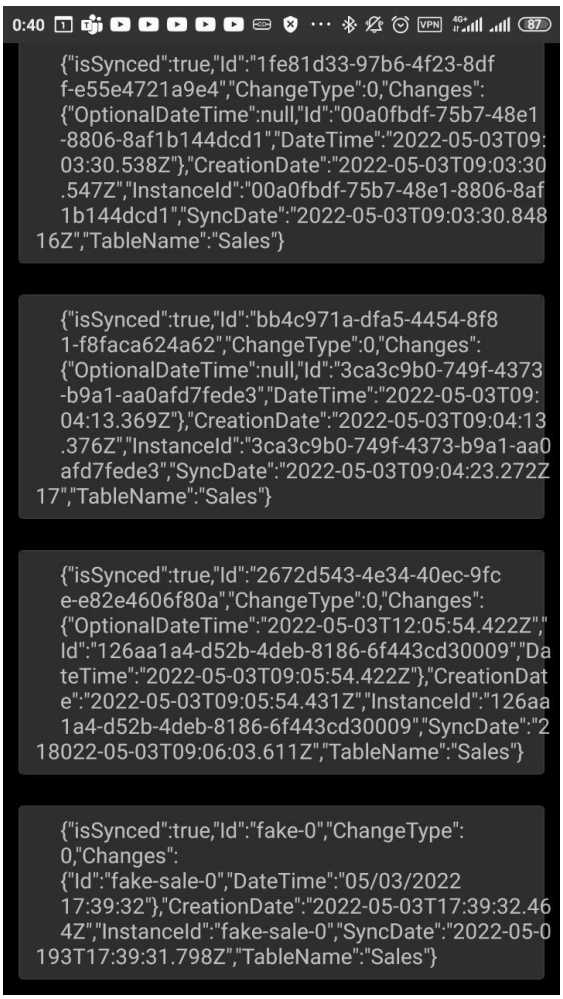
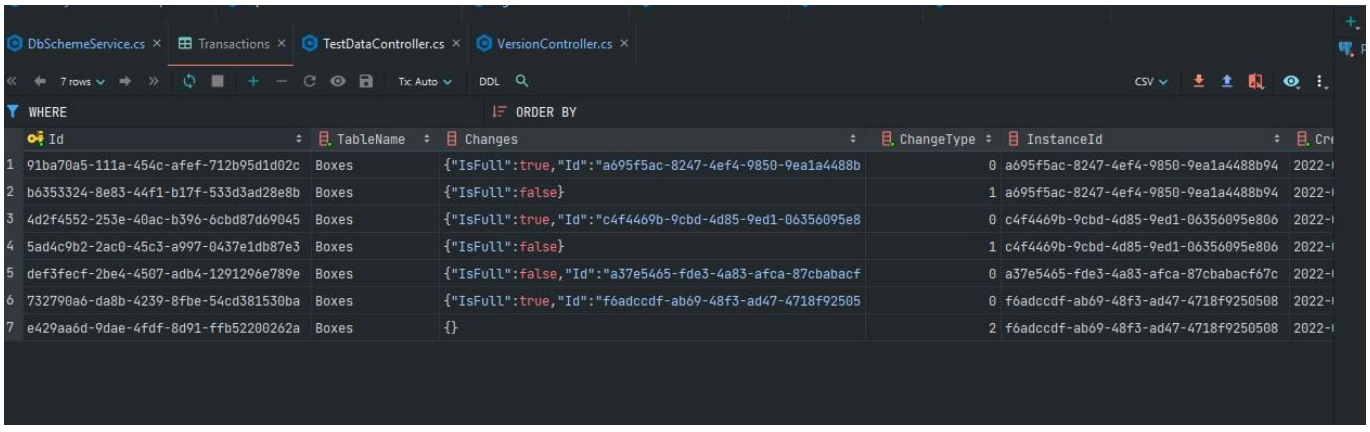


Рисунок 15 – Снимок экрана телефона с транзакциями

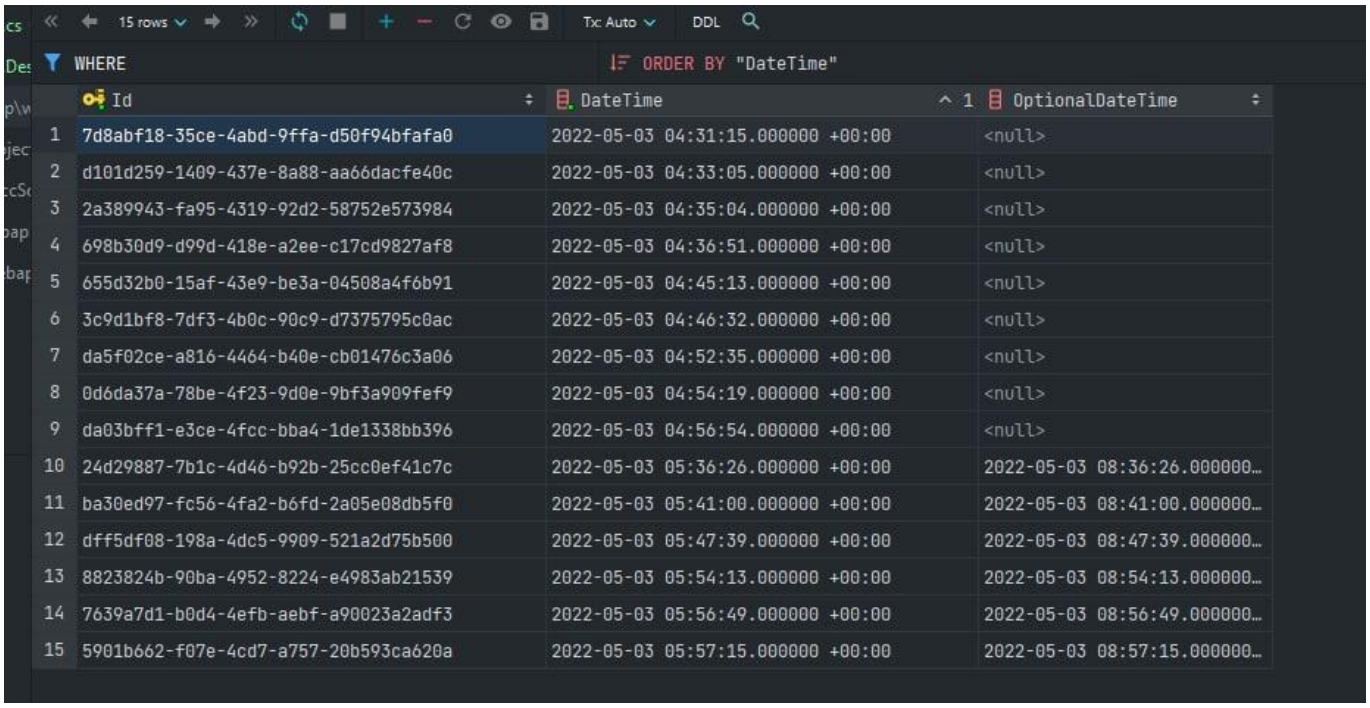
На рисунке Рисунок 15 изображены транзакции, примененные к локальной базе данных. Также крайней снизу является транзакция, пришедшая с сервера.



	Id	TableName	Changes	ChangeType	InstanceId	Cr
1	91ba70a5-111a-454c-afef-712b95d1d02c	Boxes	{"IsFull":true,"Id":"a695f5ac-8247-4ef4-9850-9ea1a4488b	0	a695f5ac-8247-4ef4-9850-9ea1a4488b94	2022-I
2	b6353324-8e83-44f1-b17f-533d3ad28e8b	Boxes	{"IsFull":false}	1	a695f5ac-8247-4ef4-9850-9ea1a4488b94	2022-I
3	4d2f4552-253e-40ac-b396-6cbd87d69045	Boxes	{"IsFull":true,"Id":"c4f4469b-9cbd-4d85-9ed1-06356095e8	0	c4f4469b-9cbd-4d85-9ed1-06356095e806	2022-I
4	5ad4c9b2-2ac0-45c3-a997-0437e1db87e3	Boxes	{"IsFull":false}	1	c4f4469b-9cbd-4d85-9ed1-06356095e806	2022-I
5	def3fecf-2be4-4507-adb4-1291296e789e	Boxes	{"IsFull":false,"Id":"a37e5465-fde3-4a83-afca-87cbabacf	0	a37e5465-fde3-4a83-afca-87cbabacf67c	2022-I
6	732790a6-da8b-4239-8fbe-54cd381530ba	Boxes	{"IsFull":true,"Id":"f6adccdf-ab69-48f3-ad47-4718f92505	0	f6adccdf-ab69-48f3-ad47-4718f9250508	2022-I
7	e429aa6d-9dae-4fdf-8d91-ffb52200262a	Boxes	{}	2	f6adccdf-ab69-48f3-ad47-4718f9250508	2022-I

Рисунок 16 – Синхронизированные транзакции на сервере

На рисунке Рисунок 16 изображены транзакции в базе на сервере, синхронизованные с клиента. Здесь можно заметить организацию контента транзакций, зависящую от типа изменения в транзакции.



	Id	DateTime	OptionalDateTime
1	7d8abf18-35ce-4abd-9ffa-d50f94bfafa0	2022-05-03 04:31:15.000000 +00:00	<null>
2	d101d259-1409-437e-8a88-aa66dacfe40c	2022-05-03 04:33:05.000000 +00:00	<null>
3	2a389943-fa95-4319-92d2-58752e573984	2022-05-03 04:35:04.000000 +00:00	<null>
4	698b30d9-d99d-418e-a2ee-c17cd9827af8	2022-05-03 04:36:51.000000 +00:00	<null>
5	655d32b0-15af-43e9-be3a-04508a4f6b91	2022-05-03 04:45:13.000000 +00:00	<null>
6	3c9d1bf8-7df3-4b0c-90c9-d7375795c0ac	2022-05-03 04:46:32.000000 +00:00	<null>
7	da5f02ce-a816-4464-b40e-cb01476c3a06	2022-05-03 04:52:35.000000 +00:00	<null>
8	0d6da37a-78be-4f23-9d0e-9bf3a909fef9	2022-05-03 04:54:19.000000 +00:00	<null>
9	da03bff1-e3ce-4fcc-bba4-1de1338bb396	2022-05-03 04:56:54.000000 +00:00	<null>
10	24d29887-7b1c-4d46-b92b-25cc0ef41c7c	2022-05-03 05:36:26.000000 +00:00	2022-05-03 08:36:26.000000…
11	ba30ed97-fc56-4fa2-b6fd-2a05e08db5f0	2022-05-03 05:41:00.000000 +00:00	2022-05-03 08:41:00.000000…
12	dff5df08-198a-4dc5-9909-521a2d75b500	2022-05-03 05:47:39.000000 +00:00	2022-05-03 08:47:39.000000…
13	8823824b-90ba-4952-8224-e4983ab21539	2022-05-03 05:54:13.000000 +00:00	2022-05-03 08:54:13.000000…
14	7639a7d1-b0d4-4efb-aebf-a90023a2adf3	2022-05-03 05:56:49.000000 +00:00	2022-05-03 08:56:49.000000…
15	5901b662-f07e-4cd7-a757-20b593ca620a	2022-05-03 05:57:15.000000 +00:00	2022-05-03 08:57:15.000000…

Рисунок 17 – Снимок экрана с синхронизованными данными на сервере с клиента

На рисунке Рисунок 17 представлена выдача данных из базы сервера, синхронизированных с клиента.

```
export const schemeConfig: DbSchemeConfig = {
  Sales: {
    properties: {
      OptionalDateTime: true,
      DateTime: true,
      Id: true,
    },
  },
  Boxes: {
    properties: {
      Id: true,
      IsFull: true,
    },
  },
  BoxSale: {
    properties: {
      BoxesId: true,
      SalesId: true,
    },
  },
  Products: {
    properties: {
      BoxId: true,
      Id: true,
      ProductType: true,
      SaleId: true,
      Title: true,
    },
  },
};
```

Рисунок 18 – Снимок экрана с примером конфигурации синхронизации базы

На рисунке Рисунок 18 изображен пример конфигурации синхронизации изменений, он может быть представлен в таком, подробном виде, где можно управлять чувствительностью к изменениям конкретных полей, а может быть в кратком – перечислении таблиц, которые полностью будут включаться в синхронизацию.

4.5 Сравнение с аналогом – Realm Sync

Сравним по имеющимся данным предлагаемое решение и Realm [11].

Таблица 3 – Сравнение предлагаемого решения и Realm Sync [13]

Критерий	**Realm Sync [13]**	**Разработанное решение**
Поддержка табличных баз данных	-	+
Поддержка документных баз данных	+	-
Политика распространения	платно	бесплатно
Страна организации, предоставляющей сервис	Америка	Россия
Платформы, для которых представлен сервис	4	2
Поддержка react-native	+	+
Автоматическое разрешение конфликтов синхронизации	+	+
Дополнительный функционал	Интеграция с Atlas	-

По составленному сравнению, можно сделать вывод: если для решения задачи устраивает документная база данных (коей является MongoDB [12]), размер оплаты Realm и готовы к рискам работы с зарубежными компаниями, Realm может дать широкий функционал. В то же время, предлагаемое решение не включает ничего лишнего для приложения, при этом реализуя все важные функции бесплатно для разработчиков и охватывая при этом круг технологий шире, к которым применима

библиотека. Также преимуществом предлагаемого решения является разворачивание на практически любом существующем проекте, тогда как Realm может быть развернут только с MongoDB [12].

4.6 Вывод по разработанной библиотеке по организации оффлайн-работы приложения

В ходе реализации библиотеки и её тестирования была достигнута цель и выполнены следующие задачи:

- локальное хранилище для мобильного устройства было определено путём сравнительного анализа, где фаворитом стала технология Realm [11];
- был реализован экспорт структуры базы данных, составляемой по запросу клиента и показанной на рисунке Рисунок 13. При этом реализованные сервис и контроллер являются полностью самостоятельными и не требуют перестройки приложения, а лишь регистрации сервиса и контроллера при старте сервера;
- реализовано разворачивание структуры базы на мобильном устройстве и работа с ней. Также реализован механизм по созданию, применению и изменению транзакций, что показывают рисунки 19 и 20;
- сервер и клиент обмениваются транзакциями с их применением и разрешением конфликтов, о чем свидетельствуют снимки экрана на рисунках 21 – 22;
- была реализована возможность настройки синхронизации в 2 форматах – подробном, с указанием чувствительных и нечувствительных к изменениям полей, и краткой, с указанием названий таблиц. Пример такой конфигурации изображен на рисунке Рисунок 18;
- также был проработан порядок разрешения конфликтов, основанный на принципе – сервер всегда прав.

Базы данных синхронизируются при имеющемся интернет-соединении, а, при его отсутствии, транзакции хранятся локально, при этом данные имеют измененный вид.

Серверная часть формирует схему базы данных, которую в будущем удобно будет использовать для генерации схем таблиц, что позволит типизировать базу с сервера для мобильного приложения. Также сервер поддерживает websocket-соединение [15], необходимое для обмена транзакциями между клиентами и сервером. Транзакции применяются, используя рефлексию, встроенную в C#.

Решение на клиентской части имеет интерфейс для получения данных и их изменения. Вся работа по синхронизации и обращению транзакций происходит автоматически и эффективно по времени и памяти.

5 Финансовый менеджмент, ресурсоэффективность и ресурсосбережение

5.1 Предпроектный анализ

Целевым рынком разработанной программной библиотеки является область разработки мобильных приложений, а именно разработчики, создающие приложения с поддержкой бесперебойной работы.

Данный рынок сегментирован по нескольким направлениям: тип предоставления бесперебойной работы и платное или бесплатное решение. Основными конкурентами на рынке являются:

- realm sync;
- react-query;
- самостоятельный монтаж оборудования.

Последнее представляет собой опцию, когда заказчик сам себя обеспечивает оборудованием для предоставления бесперебойной работы, либо нанимает кого-то, кто может настроить сеть, но это всё включено в данный пункт.

Составим карту сегментирования рынка.

Таблица 4 – Карта сегментирования рынка

		Финансовый основа применения	
		Бесплатная	Платная
Тип бесперебойной работы	Расширение сети		
	Кэширование запросов		
	Синхронизация баз данных		

Самостоятельный монтаж оборудования – ; React-query – ; Realm Sync – .

В таблице Таблица 4 отображена карта сегментации рынка по предоставлению решения бесперебойной работы приложения. По представленным сегментам можно выделить наиболее благоприятный свободный пласт – решение по синхронизации баз данных, распространяемое на бесплатной основе.

Проведем анализ решения по методологии SWOT. Для этого определим все сильные, слабые стороны. Распишем возможности и угрозы. На пересечениях определенных черт найдем возможные действия или пути решения, в случае угроз.

Таблица 5 – SWOT-матрица

	Сильные стороны: – автоматическое разрешение конфликтов; – поддержка реляционности; – автоматическое создание структуры базы на клиенте; – автоматическое применение транзакций; – конфигурация синхронизации; – возможна генерация типизации базы; – автоматическое составление структуры баз данных на сервере; – принцип применения транзакций «сервер – всегда прав»; – hook-интерфейс доступа к данным с включенной функцией обновления изменившихся данных.	**Слабые стороны:** – отсутствие документации; – отсутствие авто-тестов; – не все крайние случаи проработаны при рассинхронизации данных; – сложность интеграции с другими библиотеками, работающими с Realm; – неоптимальная структура кода.
Возможности: – обретение популярности как библиотеки; – заинтересованность нуждающихся в данном функционале компаний.	– Больше приложений смогут работать без особых вложений и усилий. – Развитие функционала за счет вовлеченности заказчика.	– Улучшение документации. – Улучшение общего качества кода. – Усовершенствование подхода разрешения особых случаев. – Переработка структуры решения.

Продолжение таблицы Таблица 5 – SWOT-матрица

	Сильные стороны: – автоматическое разрешение конфликтов; – поддержка реляционности; – автоматическое создание структуры базы на клиенте; – автоматическое применение транзакций; – конфигурация синхронизации; – возможна генерация типизации базы; – автоматическое составление структуры баз данных на сервере; – принцип применения транзакций «сервер – всегда прав»; – hook-интерфейс доступа к данным с включенной функцией обновления изменившихся данных.	**Слабые стороны:** – отсутствие документации; – отсутствие авто-тестов; – не все крайние случаи проработаны при рассинхронизации данных; – сложность интеграции с другими библиотеками, работающими с Realm; – неоптимальная структура кода.
Угрозы: – неспособность корректно работать на проектах с высокой частотой изменений в базе данных от разных пользователей.	– Добавление очередей на выполнение и улучшение производительности для сглаживания задержки, которая может появиться при решении более фундаментальных проблем.	– Переработка подхода с учетом полученной информации об ошибках и несоответствии ожиданий с результатом.

Разработанная библиотека будет отпускаться на бесплатной основе, поэтому коммерциализация не предполагается.

5.2 Инициация проекта

Устав научного проекта магистерской работы:

1. Цели и результат проекта.

Таблица 6 – Цели и результат проекта

Цель проекта:	Реализовать программное решение, синхронизирующее клиентскую и серверную базы данных, позволяющее управлять необходимыми приложению данными как в онлайн-, так и в офлайн-режимах.
Ожидаемый результат проекта:	Программная библиотека, предоставляющая функционал по организации работы с данными для мобильного приложения в онлайн и офлайн режимах.
Критерии приёмки результата проекта:	«Чистое» приложение с интегрированной разработанной библиотекой является работоспособным вне зависимости от наличия интернета.
Требования к результату проекта:	**Требования**
	Обеспечение доступа к чтению данных приложения вне зависимости от наличия интернет-соединения у пользователя.
	Обеспечение возможности изменения данных приложения вне зависимости от наличия интернет-соединения у пользователя.
	Организация загрузки данных для дальнейшего использования.

Таблица 6 – Цели и результат проекта

Требования к результату проекта:	**Требования**
	Синхронизация изменений с данными на сервере.
	Встраиваемость решения в уже существующие приложения (как серверные, так и мобильные), то есть, не должно быть необходимости строить приложение на основе потенциального решения с нуля.
	Совместимость с мобильными приложениями, написанными на React и серверными приложениями, использующими Asp.Net Core.

2. Организационная структура проекта. В рабочую группу по разработке библиотеки вошли 3 специалиста: руководитель проекта, эксперт и исполнитель. Разработка велась поэтапно с разным задействованием ролей команды. Распишем рабочую группу в таблице Таблица 7.

Таблица 7 – Рабочая группа проекта

ФИО, место работы, должность	**Роль в проекте**	**Функции**	**Трудозатраты, час**
Савельев Алексей Олегович, ТПУ, доцент ОИТ	Руководитель проекта (Р)	Координация действий исполнителя, определение порядка работы и помощь в постановке цели и задач	128

Продолжение таблицы Таблица 7 – Рабочая группа проекта

ФИО, место работы, должность	Роль в проекте	Функции	Трудозатраты, час
Дробинский Артур Владимирович, ООО «МЦЦ-Томск», технический директор	Эксперт проекта (Э)	Разрешение спорных в концепции моментов, консультирование по недостающим исполнителю компетенциям	144
Тюндеров Кирилл Вадимович, ООО «МЦЦ-Томск», инженер-программист	Исполнитель проекта (И)	Реализация проекта, решение поставленных задач	944
Итого			1216

5.3 Планирование управления научно-техническим проектом

5.3.1 План проекта

Определим ключевые этапы проекта и представим их в виде таблицы Таблица 8.

Таблица 8 – План проекта

№	Название	Длительность, дни	Дата начала	Дата окончания	Состав участников
1	Определение проблематики и её анализ	5	29.01.2022	02.02.2022	Р, И
2	Формулирование требований к решению	6	03.02.2022	08.02.2022	Р, И
3	Определение целей и задач	3	09.02.2022	11.02.2022	Р, И
4	Поиск и обработка информации по техническим подходам	14	12.02.2022	25.02.2022	И
5	Разработка серверной части по получению структуры базы данных	10	26.02.2022	08.03.2022	Э, И
6	Разработка клиентской части по организации работы с локальным хранилищем	8	09.03.2022	16.03.2022	И
7	Разработка клиентской части по работе с транзакциями	14	17.03.2022	31.03.2022	И
8	Разработка серверной части по работе с транзакциями и SignalR	10	1.04.2022	10.04.2022	Э, И

Продолжение таблицы Таблица 8 – План проекта

№	Название	Длительность, дни	Дата начала	Дата окончания	Состав участников
9	Разработка клиентской части по работе с хабом на сервере	8	11.04.2022	18.04.2022	И
10	Разработка клиентской части по применению транзакций	18	19.04.2022	07.05.2022	Э, И
11	Отладка и тестирование	4	08.05.2022	11.05.2022	И
12	Оформление отчетной записки и подготовка графических материалов	17	12.05.2022	28.05.2022	Р, И
13	Подведение итогов	1	29.05.2022	29.05.2022	Р, Э, И

Отобразим план разработки с помощью диаграммы Ганта для лучшего отображения этапов работы и задействованных членов команды. Запишем его в таблицу Таблица 9.

Таблица 9 – Календарный план-график проекта

№	Вид работ	Исполнители	$T_к$, кал. дни	Продолжительность выполнения работ												
				Янв.	Февраль			Март			Апрель			Май		
				3	1	2	3	1	2	3	1	2	3	1	2	3
1	Определение проблематики и её анализ	Р, И	5													
2	Формулирование требований к решению	Р, И	6													
3	Определение целей и задач	Р, И	3													
4	Поиск и обработка информации по техническим подходам	И	14													
5	Разработка серверной части по получению структуры базы данных	Э, И	10													
6	Разработка клиентской части по организации работы с локальным хранилищем	И	8													
7	Разработка клиентской части по работе с транзакциями	И	14													

Продолжение таблицы Таблица 9 – Календарный план-график проекта

№	Вид работ	Исполнители	$T_к$, кал. дни	Продолжительность выполнения работ												
				Янв.	Февраль			Март			Апрель			Май		
				3	1	2	3	1	2	3	1	2	3	1	2	3
8	Разработка серверной части по работе с транзакциями и SignalR	Э, И	10													
9	Разработка клиентской части по работе с хабом на сервере	И	8													
10	Разработка клиентской части по применению транзакций	Э, И	18													
11	Отладка и тестирование	И	4													
12	Оформление отчетной записки и подготовка графических материалов	Р, И	17													
13	Подведение итогов	Р, Э, И	1													

– Исполнитель проекта; – Руководитель проекта; – Эксперт проекта.

5.3.2 Бюджет научного исследования

Представим группировку затрат по статьям в таблице Таблица 10.

Таблица 10 – Группировка затрат по статьям

Статья расходов	Сумма, ₽	
	Исп. 1	Исп. 2
Сырье, материалы и программные лицензии	14 601,4	2915,4
Специальное оборудование	20 200	–
Заработная плата	448 362	279 310
отчисления во внебюджетные фонды	134 508	83 793
Прочие расходы	61 767	36 602
Итого	679 438,4	402 620,4

5.3.3 Сырье, материалы и программные лицензии

Для работы с клиентской и серверной частями необходимы программы, позволяющие быстро и удобно разрабатывать функционал. В данной работе использовались программы от JetBrains:

- WebStorm для клиентской части;
- Rider для серверной части.

Обе программы отпускаются по годовой подписке и стоят 59 $ и 139 $, соответственно. По нынешнему курсу, ежегодные подписки могут быть оформлены за 3 482 ₽ и 8 204 ₽, соответственно.

Также необходимо оформление документации и прочие канцелярские расходы, здесь бюджет можно ограничить 2 000 ₽. Вместе с документацией возникают и транспортно-заготовительные расходы, составляющие примерно 200 ₽.

В данную статью также входит потраченная электроэнергия. Расчет $C_{эл}$ ведется по рабочим дням инженера. Так как разработка велась на ноутбуке и дополнительном мониторе, их мощность легко найти на блоках питания. Разработка велась в 10 корпусе ТПУ, поэтому тариф за электроэнергию в расчетах использован соответствующий, он составляет $6{,}59\,\frac{₽}{кВт \cdot ч}$.

Таблица 11 – Затраты на электроэнергию

Оборудование	Время работы оборудования, час	Потребляемая мощность, кВт	Затраты $C_{эл}$, ₽
Ноутбук	944	0,09	559,9
Монитор	944	0,025	155,5
Итого			715,4

Чтобы производить подобную разработку не обязательно покупать лицензии именно таких программ. На рынке существуют и бесплатные варианты, представляющие тот же самый базовый функционал. Например:

- VS Code – бесплатный редактор кода для клиентской части приложения;

– Visual studio – бесплатная интегрированная среда разработки для серверной части.

При использовании других программ у опытного программиста не возникнет штрафов по работоспособности, то есть время разработки увеличено не будет, поэтому затраты на электроэнергию остаются те же.

5.3.4 Специальное оборудование

Для независимой разработки и тестирования, рекомендуется приобрести 2 любых смартфона, при этом можно купить на вторичном рынке, так как дальнейшее использование не предполагается. На данную статью закладывается 20 000 ₽. Также следует сюда включить транспортные расходы или доставку – 200 ₽.

Иным вариантом работы является использование собственного смартфона и эмулятора в Android Studio. Таким образом, можно избежать покупки ненужных смартфонов. Эмулятор на android может быть выбран любым и только отвечать требованиям поддержки react-native, то есть версия системы не должна быть ниже Android 5.0 (Lollipop).

5.3.5 Заработная плата

Определим заработную плату участников проекта. Она строится из 2 составляющих:

$$С_{зп} = З_{осн} + З_{доп},$$

где $З_{осн}$ – основная заработная плата работника, ₽;

$З_{доп}$ – дополнительная заработная плата участника, ₽.

Рассчитаем основную заработную плату для участников проекта.

$$З_{осн} = З_{дн} \cdot T_{раб},$$

где $T_{раб}$ – продолжительность работ, раб. дни;

$З_{дн}$ – среднедневная заработная плата работника, ₽.

Здесь $З_{дн}$ рассчитывается по формуле:

$$З_{дн} = \frac{З_м \cdot М}{F_д},$$

где $З_м$ – месячный должностной оклад работника, ₽;

М – количество месяцев без отпусков;

$F_д$ – действительный годовой фонд рабочего времени, раб. дни.

Определим фонд рабочего времени на 2022 год.

Таблица 12 – Фонд рабочего времени для участников проекта

Работник	Дней в году	Выходные и праздничные дни	Рабочие дни	Отпуск	Фонд рабочего времени
Руководитель	365	66	299	48	251
Исполнитель		66	299	48	251
Эксперт		118	247	24	223

Рассчитаем месячный оклад для руководителя и исполнителя. Это можно сделать по формуле:

$$З_м = З_б \cdot (k_{пр} + k_д) \cdot k_р,$$

где $З_б$ – базовый оклад сотрудника, ₽;

$k_{пр}$ – премиальный коэффициент;

$k_д$ – коэффициент доплат;

$k_р$ – районный коэффициент.

Таблица 13 – Среднемесячная заработная плата членов команды

Сотрудник	$З_б$	$k_{пр} + k_д$	$k_р$	$З_м$
Руководитель	37 700	1,3	1,3	63 713
Исполнитель	23 800	1,22	1,3	37 747
Опытный эксперт	200 000	–	–	200 000
Эксперт	30 000	–	–	30 000

Для минимизации затрат можно договориться о помощи с экспертом с немного ниже опытом, но с подобными компетенциями.

Рассчитаем среднедневную зарплату по ранее описанной формуле.

Таблица 14 – Среднедневная заработная плата членов команды

Сотрудник	$З_м$	М	$F_д$	$З_{дн}$
Руководитель	63 713	10,4	251	2 640
Исполнитель	37 747	10,4	251	1 564
Опытный эксперт	200 000	11,2	223	10 045
Эксперт	30 000	11,2	223	1 507

Определим основную заработную плату.

Таблица 15 – Основная заработная плата членов команды

Сотрудник	$З_{дн}$	$T_{раб}$	$З_{осн}$
Руководитель	2 640	16	42 240
Исполнитель	1 564	118	184 552
Опытный эксперт	10 045	18	180 810
Эксперт	1 507	18	27 126

Для общей заработной платы также необходимо рассчитать дополнительную плату сотруднику. В данном случае, она составляет 10% от основной заработной платы. Теперь можно рассчитать общую заработную плату.

Таблица 16 – Заработная плата сотрудников

Сотрудник	$З_{осн}$	$З_{доп}$	$С_{зп}$
Руководитель	42 240	4 224	46 464
Исполнитель	184 552	18 455	203 007
Опытный эксперт	180 810	18 081	198 891
Эксперт	27 126	2 713	29 839
Итого с опытным экспертом			448 362
Итого с экспертом			279 310

5.3.6 Отчисления во внебюджетные фонды

30% отчисления во внебюджетные фонды:

$$C_{\text{соц}} = C_{\text{зп}} \cdot 0{,}3 = 134\,508 \text{ ₽ при работе с опытным экспертом;}$$
$$C_{\text{соц}} = C_{\text{зп}} \cdot 0{,}3 = 83\,793 \text{ ₽ при работе с экспертом.}$$

5.3.7 Прочие расходы

В данной статье идёт расчет на непредвиденные расходы и на то, что не было учтено ранее. Величина статьи выбрана равной 10% от всех вышеприведенных затрат.

$$C_{\text{пр}} = 0.1 \sum_i C_i = 61\,767 \text{ ₽ при работе с опытным экспертом;}$$

$$C_{\text{пр}} = 0.1 \sum_i C_i = 36\,602 \text{ ₽ при работе с экспертом,}$$

где C_i – вышеописанные статьи расходов.

5.4 Организационная структура

Для реализации такого проекта хорошо подходит проектная модель построения процессов. Это обусловлено тем, что используются достаточно новые технологии, сама разработка имеет высокую сложность. Также клиентская часть и серверная, а также разные части внутри названных также сильно связаны и работают как единая система. Изобразим проектную структуру проекта на рисунке 19.

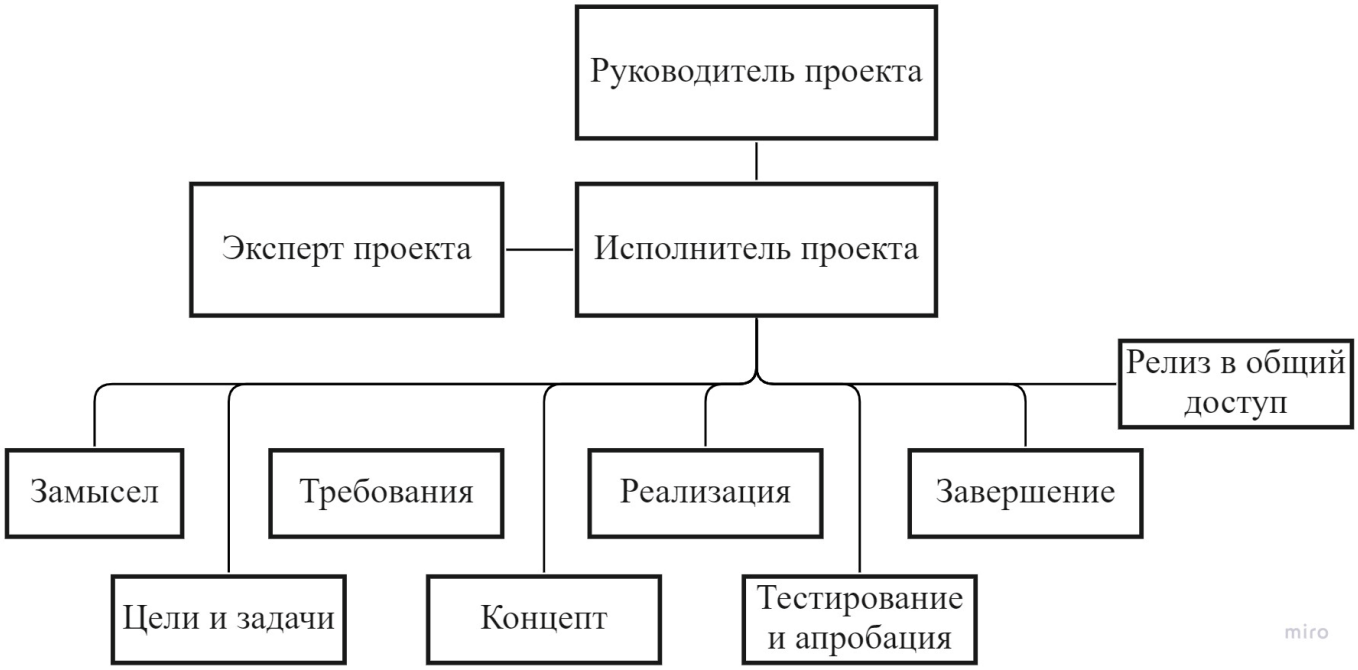


Рисунок 19 – Проектная структура

В данном проекте, по такой структуре организации, на протяжении всех этапов исполнитель является еще и ответственным лицом. Он отвечает за каждый этап, руководитель проекта в этом случае является утверждающим лицом, а эксперт – согласующим.

Всё перемещение информации происходит от исполнителя. Исполнитель информирует о затруднениях и успехах руководителю и эксперту, при этом получая необходимую информацию, но всё общение начинается с исполнителя.

Как и любой проект, имеются риски. Самым главным риском здесь является вероятные технические сложности в работе с большими данными, с большой нагрузкой. Данный риск требует лишь более детальной проработки особых случаев.

5.5 Оценка сравнительной эффективности исследования

Рассчитаем интегральный показатель для обоих вариантов затрат по проекту. Для этого можно использовать формулу:

$$I_{\text{ф}}^{p,a} = \frac{\Phi_{pi}}{\Phi_{max}},$$

где $I_{\text{ф}}^{p,a}$ – интегральный финансовый показатель разработки / аналога;

Φ_{pi} – стоимость i-го варианта разработки;

Φ_{max} – максимальная стоимость исполнения.

Так по формуле показатели для решения и аналога, соответственно:

$$I_{\text{ф}}^{p} = 1 - \text{финансовый показатель разработки;}$$

$$I_{\text{ф}}^{a} = 0{,}59 - \text{финансовый интегральный показатель аналога.}$$

Составим оценочную карту решения и аналога исполнения и представим в таблице Таблица 17. Определим баллы по позициям и их веса.

Таблица 17 – Оценочная карта вариантов исполнения описываемой разработки

Критерии	Весовой коэффициент критерия	Текущий проект	Аналог
Комфорт при разработке	0,3	5	2
Надежность оборудования	0,1	4	5
Достоверность тестирования	0,3	5	3
Нахождение правильного решения в спорном моменте	0,2	5	4
Энергосбережение	0,1	3	5
Итого	1		

Рассчитаем интегральные показатели ресурсоэффективности описываемого решения и аналога. Для этого просуммируем оценки по критерием, масштабированные по соответствующим критериям весам.

$$I_{m}^{p} = 4{,}7 - \text{интегральный показатель для решения;}$$

$$I_{m}^{a} = 3{,}3 - \text{интегральный показатель для аналога.}$$

С помощью полученных интегральных финансовых и ресурсоэффективных показателей можно получить общие показатели эффективности:

$$I^{p}_{\text{финр}} = \frac{I^{p}_{m}}{I^{p}_{\text{ф}}} = 4{,}7 - \text{интегральный показатель эффективности разработки;}$$

$$I^{a}_{\text{финр}} = \frac{I^{a}_{m}}{I^{a}_{\text{ф}}} = 5{,}59 - \text{интегральный показатель эффективности аналога.}$$

Теперь можно определить сравнительную эффективность разработки:

$$\text{Э}_{\text{ср}} = \frac{I^{p}_{\text{финр}}}{I^{a}_{\text{финр}}} = 0{,}84.$$

Приведем все полученные показатели в таблице Таблица 18.

Таблица 18 – Сравнительная эффективность разработки

Показатель	Аналог	Разработка
Интегральный финансовый показатель разработки	0,59	1
Интегральный показатель ресурсоэффективности разработки	3,3	4,7
Интегральный показатель эффективности	5,59	4,7
Сравнительная эффективность вариантов исполнения	1,2	

Аналогично описанное исполнение проекта выигрывает фактическое из-за более высокой финансовой эффективности, даже при том, что ресурсоэффективно сильно проигрывает. Финансово наибольшую разницу имеет заработная плата эксперта. Таким образом, если эти подходы совместить, то есть использовать ресурсы и сырье из проведенного варианта разработки, а эксперта найти как из аналогичного варианта. В таком случае, эффективность будет максимальная.

5.6 Вывод по разделу финансового менеджмента, ресурсоэффективности и ресурсосбережению

В ходе проведения оценки финансового аспекта НИР, был разработан календарный план выполнения проекта. Таким образом, общее время разработки программной библиотеки по организации offline-работы мобильного приложения заняло 118 рабочих дней.

Также была рассчитана смета проекта, было учтено несколько вариантов исполнения, различающихся командой, программными средствами и оборудованием, по совершённым расчетам, цена разработки составила 679 438,4 ₽ и 402 620,4 ₽.

На основании полученных расчетов затрат был проведён сравнительный анализ эффективности разработки, которая составила $Э_{ср} = 0{,}84$.

Также была определена наиболее подходящая структура организации работы в команде по реализации данного проекта – проектная форма организации. Также рассмотрен порядок обмена информацией между членами команды.

6 Социальная ответственность

Данная ВКР направлена на разработку библиотеки, предоставляющей функционал по организации офлайн-работы приложения. При этом библиотека будет отпускаться с открытым исходным кодом бесплатно. Библиотека направлена на разработчиков мобильных приложений, которые, встраивая её в своё решение, с небольшими усилиями получат корректную работу в офлайн режиме. Библиотека состоит из 2 частей – клиентской и серверной. Также библиотека позволяет работать нескольким пользователям одновременно с одними данными, при этом сразу получая обновления от других пользователей.

Приложения с поддержкой офлайн режима востребованы, когда приложение должно работать вне зависимости от наличия сети или скорости соединения. Приложение, в котором будет использоваться данная библиотека, может быть востребована работниками складов, медицинскому персоналу, выезжающим по вызовам, путешественникам и т.д.

Для разработки данной библиотеки хорошо подходит стандартное компьютерное рабочее место.

6.1 Правовые и организационные вопросы обеспечения безопасности

Основной направленностью ВКР является возможность использования решения в качестве библиотеки. Интеграция в мобильное приложение подобного рода библиотек проводится программистом. Для программиста рабочей зоной является его рабочее место.

6.1.1 Специальные правовые нормы правового законодательства

Для требуемого программиста, работающего непосредственно в офисе организации (в общем случае), в Трудовом кодексе Российской Федерации [26]

специальных правовых норм не предусмотрено. Таким образом, на программистов распространяются только общие нормы трудового права, такие как:

- основные права и обязанности работника (часть I ТК РФ [26]);
- ответственность сторон при социальном партнерстве (часть II ТК РФ [26]);
- минимальный размер оплаты труда (ст. 133 ТК РФ [26]);
- максимальная продолжительность рабочей недели (ч. 2 ст. 91 ТК РФ [26]);
- минимальная продолжительность ежегодного отпуска (ч.1 ст. 115 ТК РФ [26]);
- охрана труда и т. д.

6.1.2 Организационные мероприятия по компоновке рабочей зоны

Рабочая зона должна быть сформирована в соответствии со стандартом ГОСТ 12.2.032-78 ССБТ [27]. Рассмотрим основные положения по организации рабочей зоны.

6.1.3 Размерные характеристики рабочего места

Рабочее пространство должно быть обеспечено с учетом выполнения трудовых операций (написание и отладки кода программ) в пределах зоны моторной досягаемости работника. Несмотря на то, что для работы программисту не требуется большое пространство в вертикальной плоскости, все же, следует обеспечить просторное место работы по всем плоскостям. Средняя зона моторной досягаемости работника в сидячем положении представлена на рисунках 20 и 21 [27].

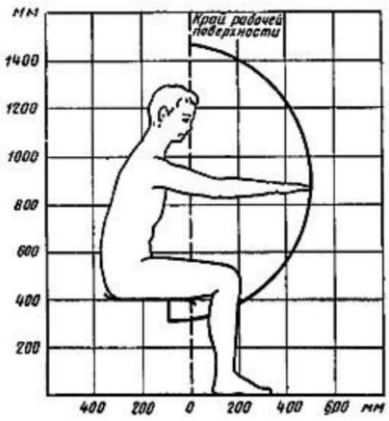


Рисунок 20 – Зона моторной досягаемости в вертикальной плоскости

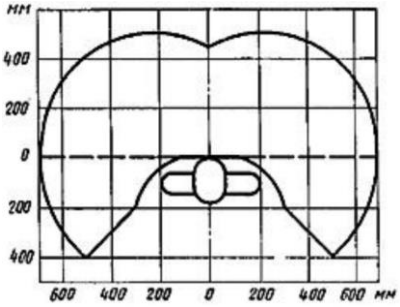


Рисунок 21 – Зона моторной досягаемости в горизонтальной плоскости

Так как программирование гендерно-нейтральный вид деятельности, рабочее место должно быть обустроено с учетом общих средних антропометрических параметров женщин и мужчин [27]. Используя средние данные о росте программистов, следует по номограмме, изображенной на рисунке 22, выбрать и организовать высоту рабочей поверхности, пространство для ног и либо выбрать высоту рабочего сидения, либо использовать кресло с возможностью регулирования высоты с конструкцией, соответствующей требованиям ГОСТ 21889-76 [28].

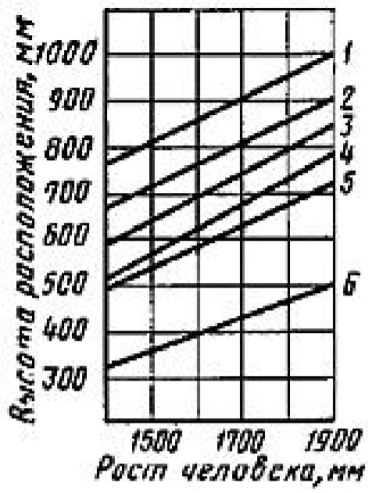


Рисунок 22 – Номограмма зависимости высоты рабочей поверхности для разных видов работ (1-4), пространства для ног (5) и высоты рабочего сидения (6) от роста человека

6.1.4 Требования к размещению органов управления

Основными органами управления для программиста являются клавиатура и мышь. Они размещаются с таким расчетом, чтобы не было пересечения рук, и чтобы располагались горизонтально в зонах 1 и 2, обозначенных на рисунке 23 [27].

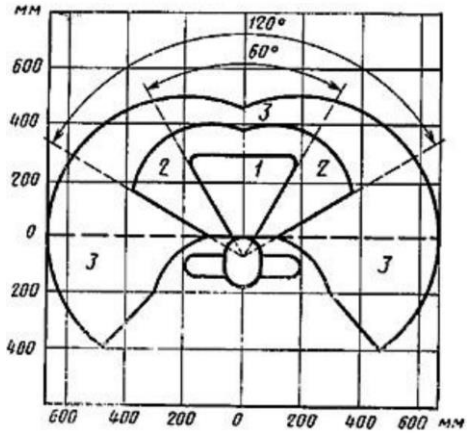


Рисунок 23 – Зоны для выполнения ручных операций и размещения органов управления

6.1.5 Требования к размещению средств отображения информации

Расположение монитора должно соответствовать общим требования к размещению средств отображения информации, представленных в ГОСТ 22269-76 [29].

Нормаль экрана монитора должна быть соосна с нормальной линией взгляда человека ($-15°$ от горизонтальной линии взгляда).

6.2 Производственная безопасность

Неблагоприятные условия, длительная работа при которых вызывает болезни или снижение работоспособности, называют вредными производственными факторами. При увеличении уровня и длительности воздействия вредные производственные факторы могут стать опасными.

Опасные производственные факторы приводят работающего к травмам или к внезапным ухудшениям здоровья.

Определим возможные опасные и вредные факторы для проектируемой рабочей зоны при разработке и эксплуатации.

Таблица 19 – Возможные опасные и вредные факторы

Факторы (ГОСТ 12.0.003-2015)	Разработка	Нормативные документы
1. Электромагнитные поля	+	СанПиН 1.2.3685-21 "Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания"[30]
2. Нарушение микроклимата помещения	+	– СП 60.13330.2020 Отопление, вентиляция и кондиционирование воздуха СНиП 41-01-2003 (с Поправкой). [31] – ГОСТ 12.1.005-88 ССБТ. Общие санитарно-гигиенические требования к воздуху рабочей зоны. [32]

Продолжение таблицы Таблица 19– Возможные опасные и вредные факторы

Факторы (ГОСТ 12.0.003-2015)	Разработка	Нормативные документы
3. Отсутствие или недостатки необходимого искусственного освещения	+	СП 52.13330.2016 Естественное и искусственное освещение. Актуализированная редакция СНиП 23-05-95. [33]
4. Повышенная пульсация светового потока	+	СП 52.13330.2016 Естественное и искусственное освещение. Актуализированная редакция СНиП 23-05-95. [33]
5. Умственное перенапряжение	+	Трудовой кодекс Российской Федерации (ТК РФ); [26]
6. Электрический ток	+	– ГОСТ 12.1.030-81 ССБТ. Электробезопасность. Защитное заземление, зануление. [34] – ГОСТ 12.1.038-82 ССБТ. Электробезопасность. Предельно допустимые уровни напряжений прикосновения и токов. [35]

Проведем анализ представленных в таблице Таблица 19 факторов.

6.3 Анализ опасных и вредных производственных факторов

1. Электромагнитные поля.

   По большей части, основными источниками электромагнитного излучения являются системный блок и монитор.

   При долговременном и сильном воздействии электромагнитного поля на организм человека, нарушаются сердечно-сосудистая система, психофизическое состояние, начинает болеть голова, ухудшается самочувствие и чувствуется слабость во всём организме. При этом происходит сильное изменение электрической активности мозга.

При небольшом, но постоянном излучении происходит накапливание электромагнитных воздействий, что приводит к снижению иммунитета, частым стрессам и повышенной усталости.

По СанПиН 1.2.3685-21 "Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания" [30] ПДУ электромагнитного поля (ЭП) частотой 50 Гц на рабочем месте - 5 кВ/м.

Для снижения излучения рекомендуется устанавливать оборудование на расстоянии от работника и использовать блоки питания / мониторы достаточной мощности, также соблюдать график работы и делать перерывы.

2. Нарушение микроклимата помещений.

Микроклимат рабочих помещений – состояние внутренней среды помещений, которые определяются показателями температуры, влажности, скорости движения воздуха и теплового излучения. Эти условия могут влиять на самочувствие, работоспособность, здоровье и производительность труда сотрудника как в положительную, так и в отрицательную сторону.

Оптимальные и допустимые условия микроклимата при работе с компьютерами устанавливаются НД ГОСТ 12.1.005-88 ССБТ. Общие санитарно-гигиенические требования к воздуху рабочей зоны. [32].

Вся трудовая деятельность разбита на категории по интенсивности энергозатрат организма в ккал/ч (Вт). Таким образом, работа сидя, сопровождающаяся незначительными физическими нагрузками, попадает в категорию Iа – работа с интенсивностью энергозатрат до 120 ккал/ч (до 139 Вт).

Допустимый уровень микроклимата достигается вентиляцией и отоплением помещения. Если обычные средства не позволяют достичь допустимых условий микроклимата (из-за большого размера, например) или надо установить оптимальные показатели, необходимы дополнительные меры по защите работников от возможного перегревания и охлаждения. Например,

система кондиционирования; индивидуальные средства защиты от повышенной или пониженной температуры; смена помещения или разбиение на смены и др. Оптимальные и допустимые условия микроклимата представлены в таблицах

Таблица 20 и Таблица 21, соответственно.

Таблица 20 – Оптимальные условия микроклимата для Iа категории

Период года	Температура, °С	Относительная влажность, %	Скорость движения воздуха, м/с
Холодный	(22 – 24)	(40 – 60)	0,1
Теплый	(23 – 25)	(40 – 60)	0,1

Таблица 21 – Допустимые нормы условий микроклимата для Iа категории

Период года	Температура, °С		Относительная влажность, %	Скорость движения воздуха, м/с	
	Диапазон ниже оптимальных величин	Диапазон выше оптимальных величин		Для диапазона температур ниже оптимальных величин, не более	Для диапазона температур выше оптимальных величин, не более
Холодный	(18 – 21)	(25 – 26)	(40 – 75)	0,1	0,1
Теплый	(20 – 22)	(28 – 30)	(40 – 55)	0,1	0,2

3. Отсутствие или недостатки необходимого искусственного освещения.

Под освещением в рабочем помещении понимают количество световой энергии, получаемой, распределяемой и используемой для обеспечения благоприятных условий видимости. Недостаток или переизбыток освещения может нести вред для зрения человека, а также ухудшать настроение и самочувствие, из-за чего снижается эффективность труда.

Всё оборудование, обеспечивающие освещение, должно соответствовать нормативным требованиям СП 52.13330.2016 Естественное и искусственное

освещение. Актуализированная редакция СНиП 23-05-95. [33]. Согласно этим требованиям, нормируемый показатель искусственного освещения в помещениях для работы малой точности (объект различия 1-5 мм) равен 300 лк.

В случае, если показатель освещенности не является нормальными, следует уменьшить освещенность путём отключения части оборудования или замены на менее мощное, либо добавить путём установки индивидуальный средств освещения; замены оборудования или выполнения рабочего помещения в светлых тонах, что обеспечит лучшее отражение света и повысит освещенность.

Произведем расчет общего равномерного искусственного освещения.

Для освещения будет использоваться светильник: открытый двухламповый светильник типа ОДО (1230х266х158 мм) с двумя лампами ЛТБ мощности 40 Вт и световым потоком 2850 лм.

В этом случае при высоте помещения 4 м и высоте свеса 0,5 м, высота подвеса равна:

$$h_{\text{п}} = 4 - 0{,}5 = 3{,}5 \text{ м}.$$

Данный показатель попадает под предельно допустимые нормы.

Средняя высота рабочего стола составляет 0,75 м. С учетом этого и ранее вычисленной высоты можно получить расчетную высоту:

$$h = 3{,}5 - 0{,}75 = 2{,}75 \text{ м}.$$

Рассчитаем расстояние между рядами светильников L и от светильника до стенки $\frac{L}{3}$, при этом примем коэффициент расположения светильников для данного типа ламп 1,2.

$$L = 2{,}75 \cdot 1{,}2 = 3{,}3 \text{ м};$$

$$\frac{L}{3} = 1{,}1 \text{ м}.$$

Примем расстояние между светильниками в ряду 0,47 м. Теперь рассчитаем количество рядов светильников и сколько их в ряду.

$$n_{\text{ряд}} = \frac{7 - 2{,}2}{3{,}3} + 1 = 2{,}45;$$

$$n_{\text{св}} = \frac{8 - 2{,}2}{1{,}23 + 0{,}47} = 3{,}4.$$

Так как эти значения должны быть целочисленными, примем 2 и 3, соответственно. Тогда общее количество светильников равно 6, соответственно, ламп – 12.

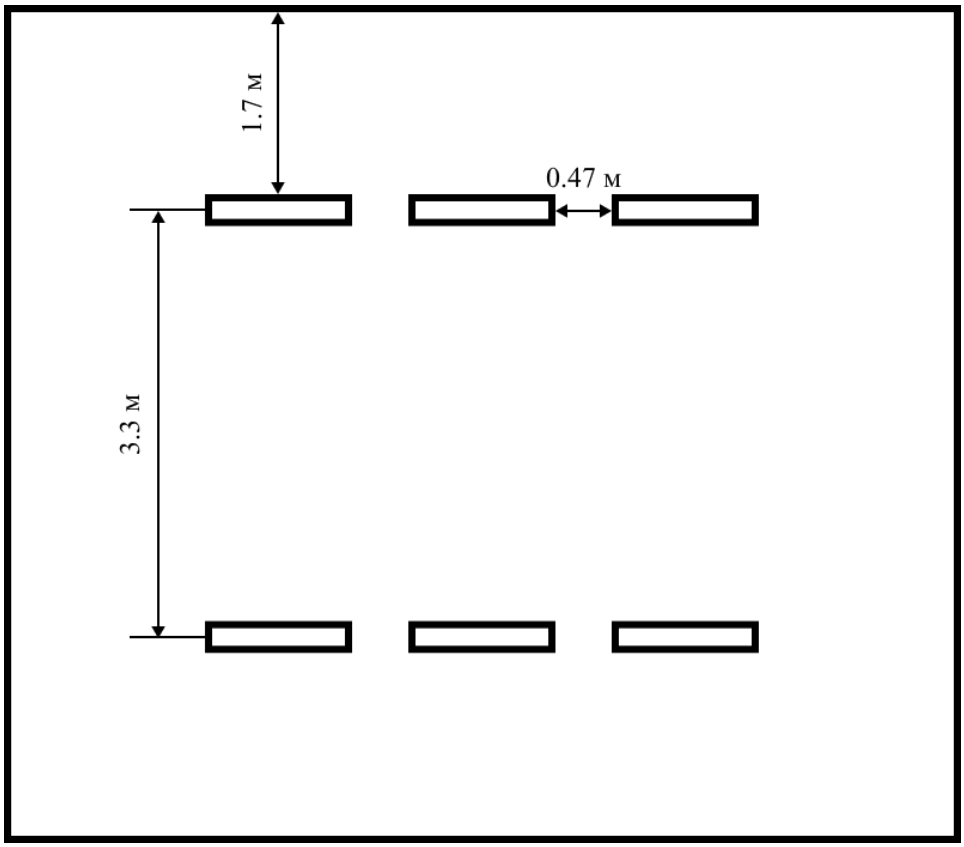


Рисунок 24 – План помещения с расстановкой светильников

В данном помещении свежепобеленный потолок и оклеенные светлыми обоями стены, поэтому коэффициенты отражения потолка и стен, соответственно, равны $\rho_{\text{п}} = 70\%$ и $\rho_{\text{ст}} = 30\%$. Индекс помещения:

$$i = \frac{56}{2{,}75 \cdot (7 + 8)} = 1{,}36.$$

Тогда коэффициент использования светильника равен 51%.

Для офисного помещения коэффициент запыленности составляет 1,5.

Определим необходимый световой поток лампы при нормативной освещенности в 300 лк.

$$\Phi = \frac{300 \cdot 56 \cdot 1.5 \cdot 1.1}{12 \cdot 0{,}51} = 4529 \text{ лм}$$

Для выбранной лампы данное превышение составляет почти в 2 раза. Таким образом, изменим количество рядов светильников, увеличив их на один, то есть теперь оно равно 3. При этом количество ламп равно 18. Тогда план помещения будет тоже изменен.

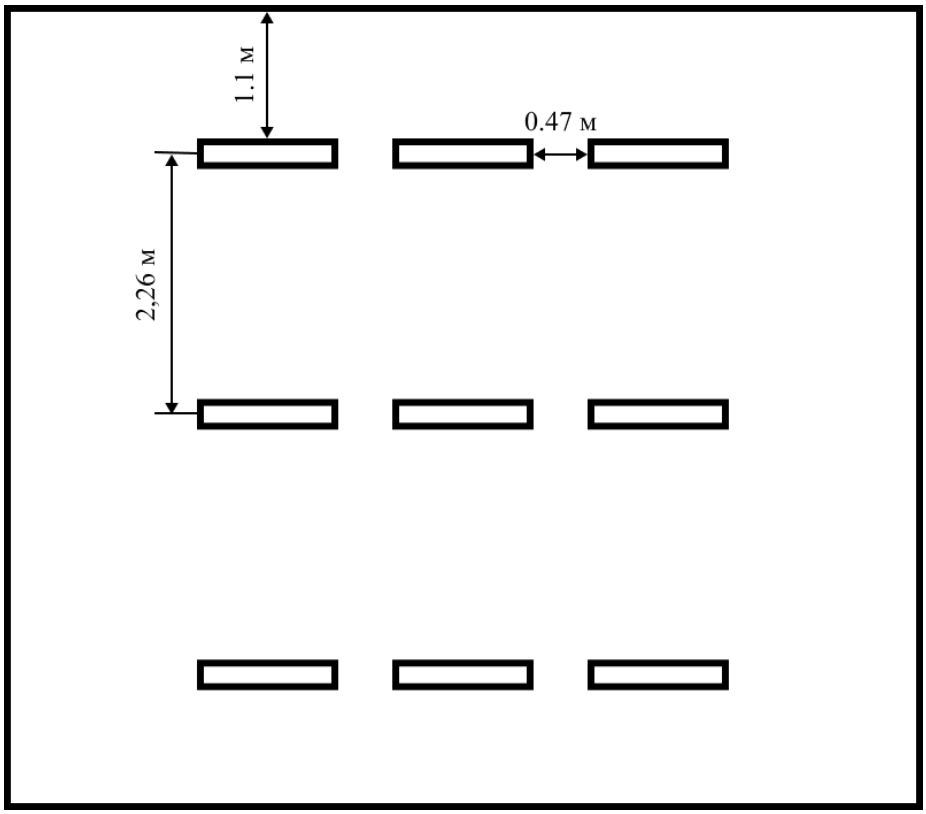


Рисунок 25 – План помещения с измененным расположением светильников

Рассчитаем заново световой поток лампы.

$$\Phi = \frac{300 \cdot 56 \cdot 1.5 \cdot 1.1}{18 \cdot 0{,}51} = 3019 \text{ лм.}$$

Произведем проверку полученного значение с выбранной ЛТБ лампой.

$$\Delta = \frac{|2850 - 3019|}{2850} \cdot 100\% = 5{,}9\%.$$

Полученное отклонение находится в пределах нормы.

При этом номинальная мощность осветительной системы составит:

$$P = 18 \cdot 40 = 720 \text{ Вт.}$$

4. Повышенная пульсация светового потока.

   При колебаниях в электрической сети до 300 Гц возможна пульсация светового потока осветительных приборов. Частая пульсация может оказать негативное влияние на общую и зрительную работоспособность программиста. Соблюдение норм коэффициента пульсации освещенности позволяет предотвратить отрицательное влияние фликера, стробоскопического эффекта и снизить зрительное и общее утомление человека.

   Эти нормы регламентированы в СП 52.13330.2016 Естественное и искусственное освещение. Актуализированная редакция СНиП 23-05-95. [33], составляют 20% для данного типа работы.

5. Умственное перенапряжение.

   Часто при долгой работе за компьютером у человека появляется переутомление, стресс, эмоциональное выгорание и умственное перенапряжение.

   Если рабочее место организовано неправильно, работоспособность сотрудника будет снижена, а утомляемость возрастет.

   Оптимальный режим труда, обусловленный Трудовым кодексом Российской Федерации (ТК РФ): при работе с данными, совершении операций на мониторе, чтении информации с экрана непрерывная продолжительность работы за компьютером не должна превышать 4 часов. Через каждый час работы необходимо делать перерыв на 5-10 минут, а через два часа – на 15 минут.

   Чтобы предотвратить переутомление, соблюдать график труда с определенными перерывами, в периоды отдыха уходить с рабочего места (желательно в другую комнату), создать уютную атмосферу в рабочем

помещении и позволить персонализировать рабочие место и процесс под удобства работника.

6. Электрический ток.

   Удар электрическим током может наступать в случае неисправности оборудования или ошибках в организации электрической сети рабочего помещения. В зависимости от тяжести удара, могут возникать следующие последствия: ожоги, боль, нарушения сердечного и дыхательного ритмов, головокружение, нарушение зрения, сознания, иногда возбуждение, ретроградная амнезия, разрывы мышц при их судорожном сокращении, также возможны компрессионные и отрывные переломы костей.

   Работу по электробезопасности регламентируют нормативные документы: ГОСТ 12.1.030-81 ССБТ. Электробезопасность. Защитное заземление, зануление [34] и ГОСТ 12.1.038-82 ССБТ. Электробезопасность. Предельно допустимые уровни напряжений прикосновения и токов [35].

   Во избежания ударов электрическим током необходимо следовать технике безопасности при работе с электронными устройствами и использовать надежное сетевое оборудование.

6.4 Экологическая безопасность

При разработке программной библиотеки некоторые устройства могут выйти из строя, в таком случае их необходимо утилизировать. При неправильной утилизации электроники и макулатуры идёт воздействие на экологическое состояние литосферы.

Требования к охране почв от загрязнения указаны в НД ГОСТ 17.4.3.04-85 Охрана природы (ССОП). Почвы. Общие требования к контролю и охране от загрязнения [36]. Здесь выдвигается требование к охране почв, то есть, любой воздействующей на неё деятельности, что необходима утилизация и захоронение

выбросов, сбросов, отходов, стоков и осадков сточных вод с соблюдением мер по предотвращению загрязнения почв.

Порядок утилизации отходов и обращения с ними регламентируют такие нормативные документы, как: ГОСТ Р 57740-2017 Ресурсосбережение. Обращение с отходами. Требования к приему, сортировке и упаковыванию опасных твердых коммунальных отходов [37], ГОСТ Р 53692-2009 Ресурсосбережение. Обращение с отходами. Этапы технологического цикла отходов [38]. Поэтому, в случае с макулатурой, необходимо собрать всю использованную и ненужную бумагу и пр. и доставить в специальный центр по переработке отходов. В случае с электроникой (ноутбуки, телефоны и мониторы), их необходимо будет сдать на утилизацию бытовой техники. Подобные организации несложно найти в своём или ближайшем городе. Также, подобные услуги предоставляют распространенные сети магазинов, такие как, «Эльдорадо».

Воздействие на остальные экологические зоны отсутствует.

6.5 Безопасность в чрезвычайных ситуациях

В период разработки программного решения возможно возникновение следующих техногенных и природных ЧС: пожар от неисправной электроники, ураган и гроза. Наиболее вероятным из них является пожар. Он может произойти от выхода из строя адаптеров питания ноутбука и монитора, также при разрушении батареи ноутбука тоже может возникнуть возгорание.

Чтобы предупредить возникновение такого рода пожара, необходимо периодически проверять адаптеры питания на температуру нагрева, она должна быть в допустимых устройством пределах. Поможет также использование сетевых фильтров и автоматов отключения. Также стоит обращать внимание на целостность проводов и конструкции, а также на искрение в момент подключения к сети.

Данная ЧС регламентирована федеральным законом от 22.07.2008 N 123-ФЗ (ред. от 30.04.2021) "Технический регламент о требованиях пожарной безопасности" [39].

Возможный пожар в данном случае будет относиться к категории В (пожары горючих жидкостей или плавящихся твердых веществ и материалов) или Е (пожары горючих веществ и материалов электроустановок, находящихся под напряжением).

При возникновении данного пожара:

1. Попытаться обесточить устройство, если возможно.
2. Воспользоваться индивидуальными средствами тушения, то есть, огнетушителем.
3. Покинуть помещение согласно плану эвакуации, пример представлен на рисунке 26.
4. Вызвать пожарных, при этом рассказав о том, что горит и где.

Рисунок 26 – План эвакуации здания, где происходила разработка проекта

6.6 Вывод по разделу социальной ответственности

В ходе разработки данного раздела были выделены опасные и вредные факторы, возможные к возникновению при разработке программной библиотеки. На основании каждого был проведен анализ и были сопоставлены регулирующие нормативные документы.

Также рассмотрена компоновка рабочего места программиста и произведен расчет с последующим проектированием системы искусственного освещения рабочего помещения. Отклонение спроектированной и рассчитанной составляет 5,9%, что является допустимым.

Выполнение данной работы попадает под категорию Iа и проходит в помещениях класса Д, то есть офисные здания без взрывоопасных веществ и при невысокой температуре.

Также было рассмотрено влияние данной разработки на литосферу и предложены возможные варианты утилизации твердых отходов.

СПИСОК ИСТОЧНИКОВ

1. React – https://reactjs.org/;
2. Overview to ASP.NET Core – https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0;
3. Что такое Wi-Fi репитер (повторитель), как он работает, и что значит роутер в режиме репитера? – https://help-wifi.com/poleznoe-i-interesnoe/chto-takoe-wi-fi-repetir-povtoritel-kak-on-rabotaet-i-chto-znachit-router-v-rezhime-repitera/;
4. Что такое Mesh Wi-Fi-система и как ее сделать – https://www.it-world.ru/tech/practice/154593.html;
5. Два репитера в одной Wi-Fi сети. Как подключить несколько усилителей к одному роутеру? – https://help-wifi.com/poleznoe-i-interesnoe/dva-repitera-v-odnoj-wi-fi-seti-kak-podklyuchit-neskolko-usilitelej-k-odnomu-routeru/;
6. React-query – https://react-query.tanstack.com/;
7. AsyncStorage Usage – https://react-native-async-storage.github.io/async-storage/docs/usage;
8. Known storage limits – https://react-native-async-storage.github.io/async-storage/docs/limits/;
9. Limits In SQLite – https://www.sqlite.org/limits.html;
10. PostgreSql – https://www.postgresql.org/;
11. MongoDB Realm. Build, deploy, and scale apps with ease. – https://www.mongodb.com/realm;
12. MongoDB. Build faster. Build smarter. – https://www.mongodb.com/
13. Realm Sync. For reactive, run-anywhere mobile apps. – https://www.mongodb.com/realm/mobile/sync;
14. Dependency injection in ASP.NET Core – https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0;
15. The WebSocket API (WebSockets) – https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API;
16. Basic Structure – https://swagger.io/docs/specification/2-0/basic-structure/;

17. Common Language Runtime (CLR) overview – https://docs.microsoft.com/en-us/dotnet/standard/clr;
18. Учебник. Начало работы с ASP.NET Core SignalR с использованием TypeScript и Webpack. – https://docs.microsoft.com/ru-ru/aspnet/core/tutorials/signalr-typescript-webpack?view=aspnetcore-6.0&tabs=visual-studio;
19. Activator.CreateInstance Method – https://docs.microsoft.com/en-us/dotnet/api/system.activator.createinstance?view=net-6.0#system-activator-createinstance(system-string-system-string);
20. Entity Framework Core – https://docs.microsoft.com/en-us/ef/core/;
21. ORM – Википедия – https://ru.wikipedia.org/wiki/ORM
22. What Is SQLite? – https://www.sqlite.org/index.html;
23. B-дерево – https://ru.wikipedia.org/wiki/B-дерево;
24. Введение в хуки – https://ru.reactjs.org/docs/hooks-intro.html;
25. @microsoft/signalr – https://www.npmjs.com/package/@microsoft/signalr;
26. Кодекс Российской Федерации "Трудовой кодекс Российской Федерации" от 30.12.2001 № 197-ФЗ // Российская газета. 2001 г. № 256. с изм. и допол. в ред. от 16.12.2019;
27. Межгосударственный стандарт "ГОСТ 12.2.032-78 ССБТ. Рабочее место при выполнении работ сидя. Общие эргономические требования" от 26 апреля 1978 № 1102 // Государственный комитет стандартов Совета Министров СССР. 1979 г;
28. Государственный стандарт Союза ССР "ГОСТ 21889-76. Система "Человек-машина". Кресло человека-оператора. Общие эргономические требования" от 25.05.76 № 1283 // Постановление Государственного комитета стандартов Совета Министров СССР. 1977 г. с изм. и допол. в ред. от март 1993 г;
29. ГОСТ 22269-76 Система «человек-машина». Рабочее место оператора. Взаимное расположение элементов рабочего места. Общие эргономические требования. от 22 декабря 1976 г. № 2798 // Постановление Государственного комитета стандартов Совета Министров СССР. 1978 г;

30. СанПиН 1.2.3685-21 "Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания";
31. СП 60.13330.2020 Отопление, вентиляция и кондиционирование воздуха СНиП 41-01-2003 (с Поправкой).
32. МЕЖГОСУДАРСТВЕННЫЙ СТАНДАРТ "Действующий ГОСТ 12.1.005-88 Система стандартов безопасности труда (ССБТ). Общие санитарно-гигиенические требования к воздуху рабочей зоны" от 29.09.88 № 3388 // Постановление Государственного комитета СССР по стандартам. 1989 г. с изм. и допол. в ред. от июнь 2000 г;
33. СП 52.13330.2016 Естественное и искусственное освещение. Актуализированная редакция СНиП 23-05-95.
34. ГОСТ 12.1.030-81 ССБТ. Электробезопасность. Защитное заземление, зануление.
35. ГОСТ 12.1.038-82 ССБТ. Электробезопасность. Предельно допустимые уровни напряжений прикосновения и токов.
36. ГОСТ 17.4.3.04-85 Охрана природы (ССОП). Почвы. Общие требования к контролю и охране от загрязнения.
37. ГОСТ Р 57740-2017 Ресурсосбережение. Обращение с отходами. Требования к приему, сортировке и упаковыванию опасных твердых коммунальных отходов.
38. ГОСТ Р 53692-2009 Ресурсосбережение. Обращение с отходами. Этапы технологического цикла отходов.
39. Федеральный закон от 22.07.2008 N 123-ФЗ (ред. от 30.04.2021) "Технический регламент о требованиях пожарной безопасности".

Приложение I

(справочное)

Software solution development

Студент

Группа	ФИО	Подпись	Дата
8ИМ02	Тюндеров Кирилл Вадимович		

Руководитель ВКР

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Савельев Алексей Олегович	к.т.н.		

Консультант-лингвист отделения иностранных языков ШБИП

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Старший преподаватель ОИЯ	Пичугова Инна Леонидовна			

7 Decision design

7.1 Structure

To solve the issue mentioned earlier we need to provide server and client library parts to achieve the goal. Let us take a look at a server part and define its main components, which are mandatory if you need your library to work properly. The whole implementation will be detailed later.

First of all, mobile application needs database structure loading. To meet this need we are going to add separate service that gets database context via dependency injection. This service creates database structure in data transfer object with JSON format on demand and returns it to caller.

To deliver this data to client we can use api-controller which can work with client database structure requests. The controller gets database structure service via dependency injection and runs its method. When this method is finished, the controller returns database schema to caller (client).

Other functionality is database synchronization. There should be service which is able to apply changes from mobile application to database. Let us define that object which contains information only about one change for single record called transaction. This service is also responsible for taking unsynchronized transactions to every client.

To make synchronization work, we need to setup information exchange between handheld application and synchronization service on the server. In this case we need to have a possibility to send data to mobile application. We need to setup a stable duplex connection, for example, we can use websocket technology. To make a good design, we need to separate service to do all websockets staff. It will receive and send messages and setup connection with mobile application.

Backend application structure is shown in Figure 1. There are all described parts presented.

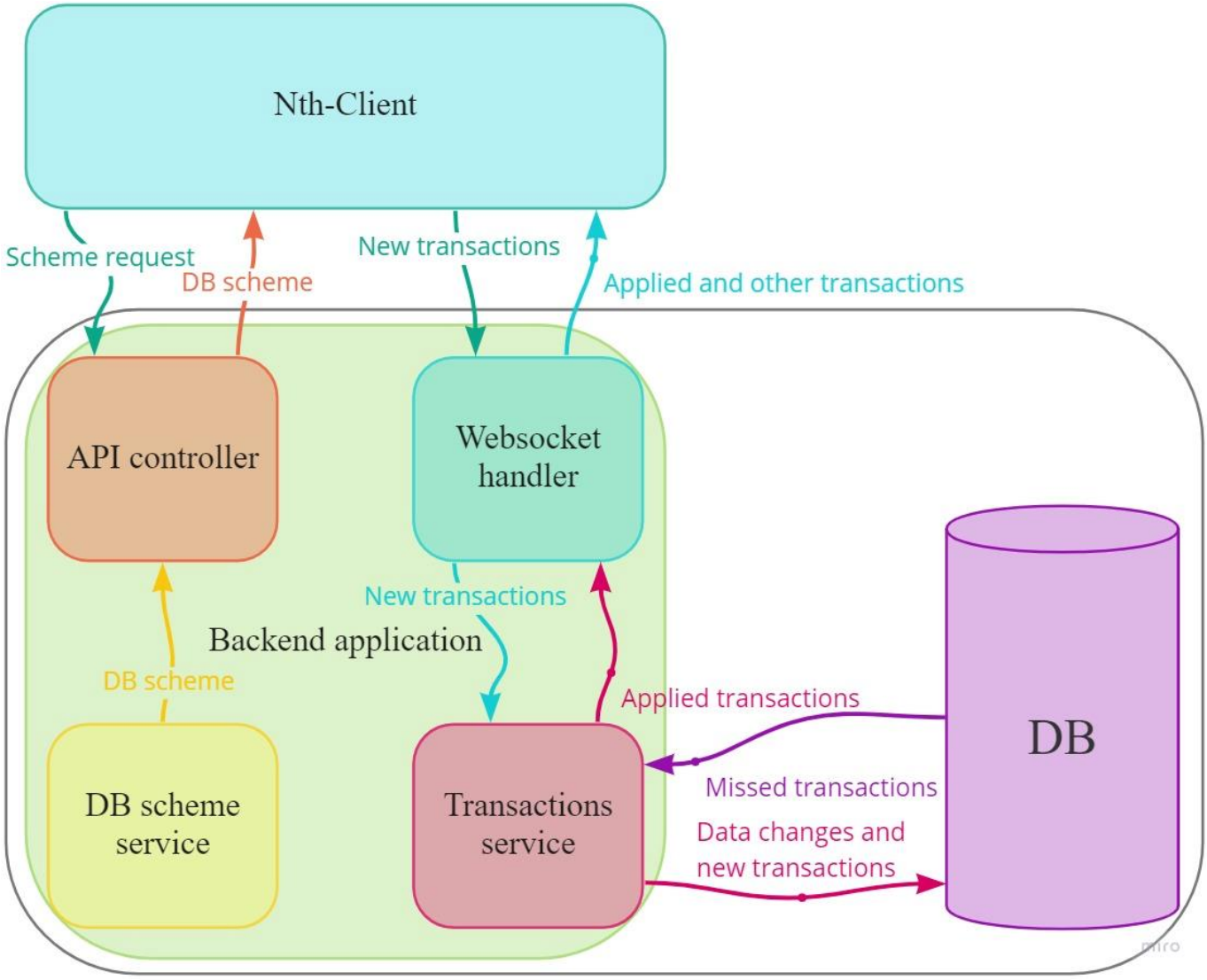


Figure 1 – Server-side program functional scheme

Let us define client-side structure with the same way.

If we are using data-source transition approach, we need some local storage and related service. This approach assumes to get all data from local storage, not from server (but local storage could be updated with server). Local storage service is responsible for getting records from local database, request configuring (sorting and filtering) and data modification operations. Also, there should be a possibility to store and present data with provided schema from the server. Due to it, the service should be able to apply this database schema to local one.

Database synchronization is developed with transactions, so we need to create them somehow. It is better to have some middleware between user data representation, modify

operations and local storage service. The middleware works with user data and writes changeset in transactions. Then these transactions are saved to local storage. The next step is transmitting them to the server and applying transactions to server-side database. This will be handled by the next service.

As it was designed earlier in server-side application, a client and the server setup websocket connection. And it is clear to have separate service for this functionality. This mobile service connects to server websocket part, sends and receives transactions.

When mobile application gets transactions from server, it should apply them to local database. There is separate and really complicated functional, so it can be moved to its own service.

Due to client-side structure comprises several separate independent services, there is a need to have aggregation one. It will connect all of them and control data stream – redirect data from one service to another. It allows us to reach the main goal – database synchronization.

Let us build client-side functional scheme and represent it in Figure 2.

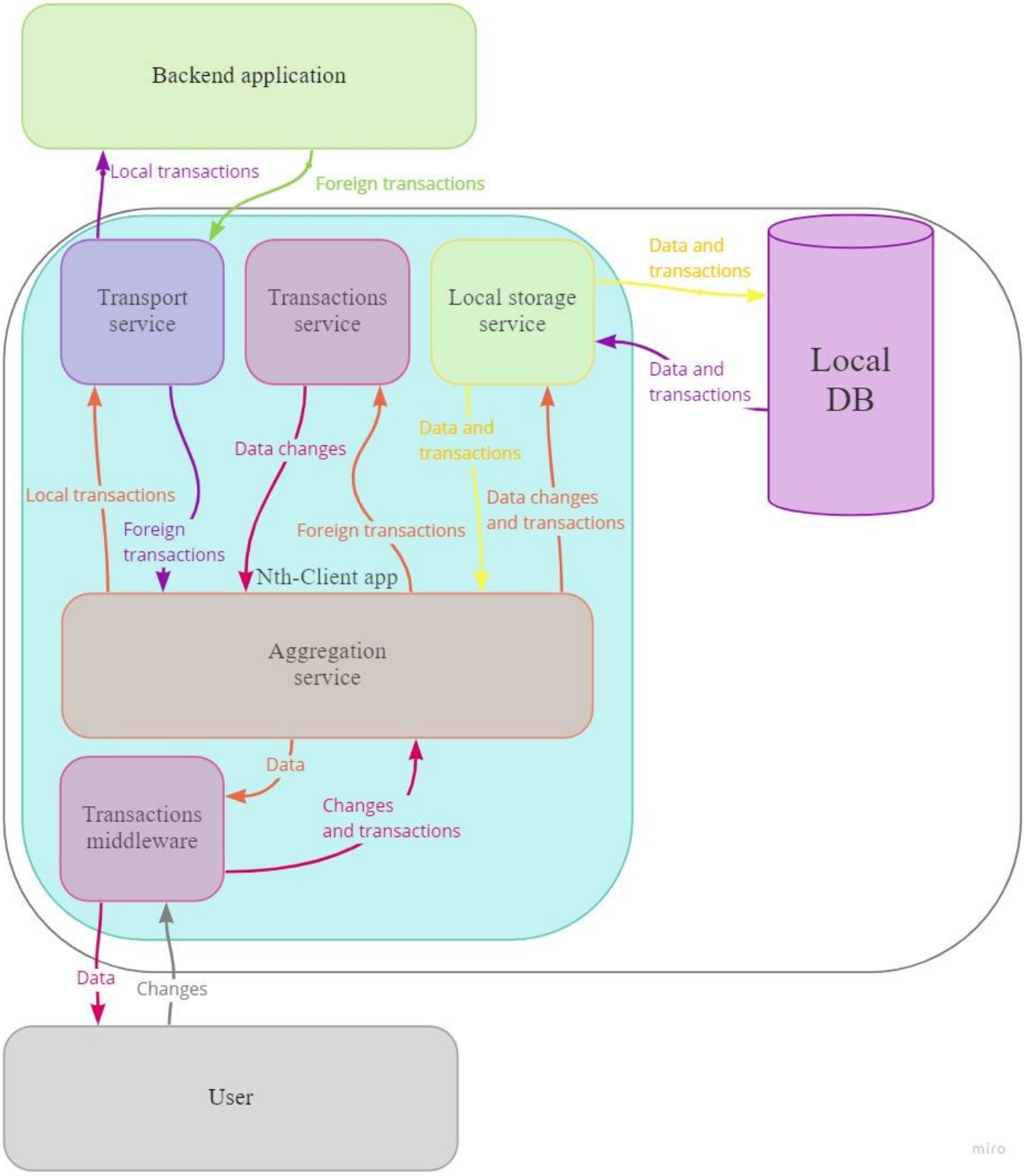


Figure 2 – Mobile application functional scheme

The structure design for both of parts is done (server and mobile parts), so we can move to its implementation.

7.2 Backend application implementation

Development will be shown with the design order.

7.2.1 Database scheme structure

It is necessary to define mandatory data for client and its format. The main part is tables list in database. So, every table should have these properties:

- table name;
- entire class name of entity, which is stored in database;
- solution name which contains entity class;
- attribute-s list. Each attribute has:
    - attribute name;
    - scheme:
        - — attribute type (probably, it is necessary to convert PostgreSQL column type to mobile local storage type);
        - — format, if there is simple data, for example DateOnly or DateTime;
        - — type details link, if there is complicated data, for example, Enum;
    - «is unique» flag;
    - «is not null» flag;
    - «is primary» flag;
- primary keys list (we use list to support union primary keys), there every item in the list is only attribute name;
- table relations list, where each item has:
    - related table name;
    - own attributes;
    - external attributes;
    - relation type;
    - update constraints.

In case of complicated types (such is Enum), we need to provide additional data. This data would contain type details. The same way is used for swagger definition. Let us define type details data transfer object structure:

- type name;
- «are values default» flag (common Enum values – natural digits sequence);
- values dictionary. There are its labels and its values.

Further, this information will be used for type generation. When the developer uses this database structure with api calls, he knows schema only on runtime, but it is very convenient to have typing in development stage. It can automatically define entity types and simplify work with them.

There was a service developed, which is responsible for making database scheme. Let us describe its main method in the flowchart. The method is used to build database scheme and additional typing structure.

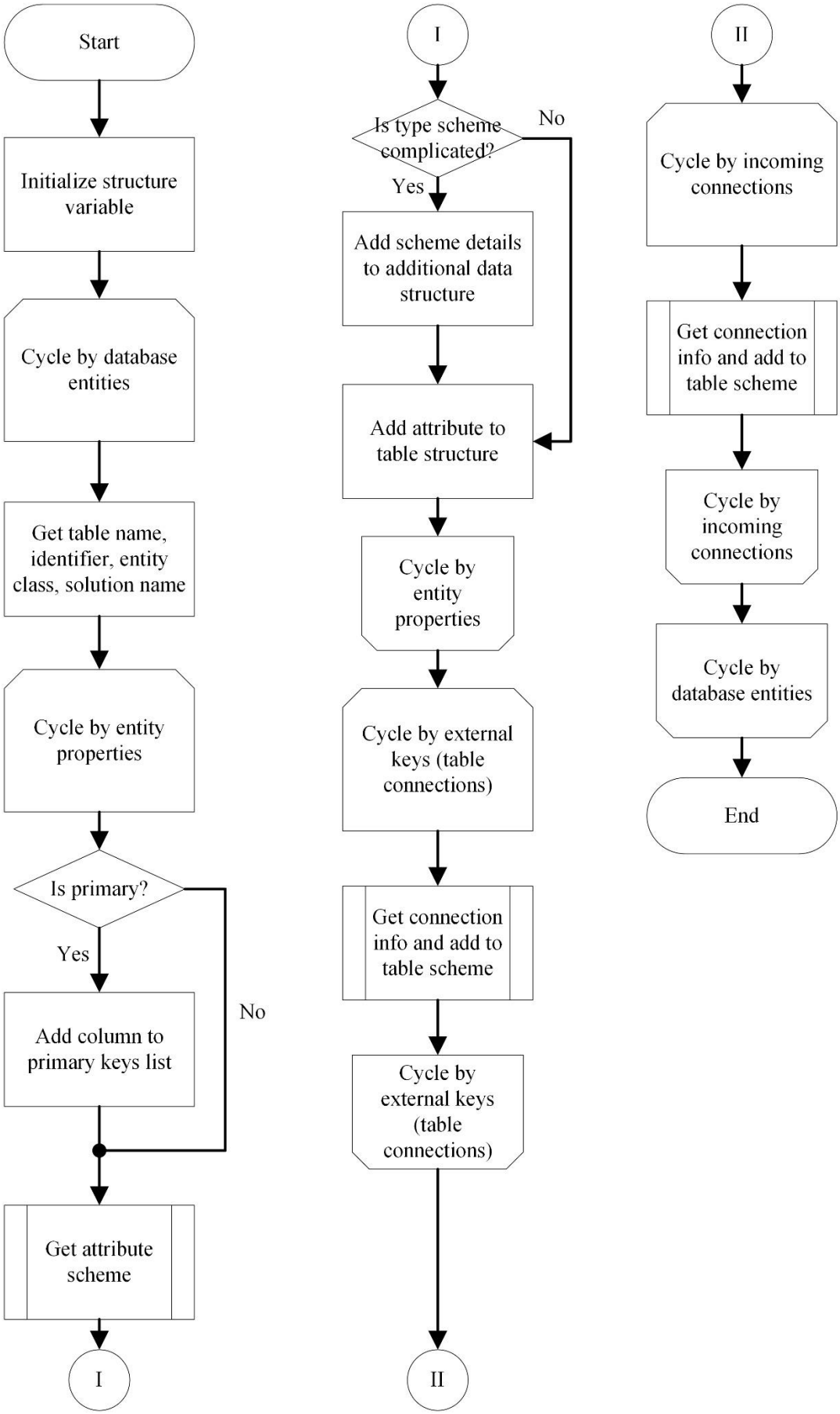


Figure 3 – A flowchart of getting database scheme method algorithm

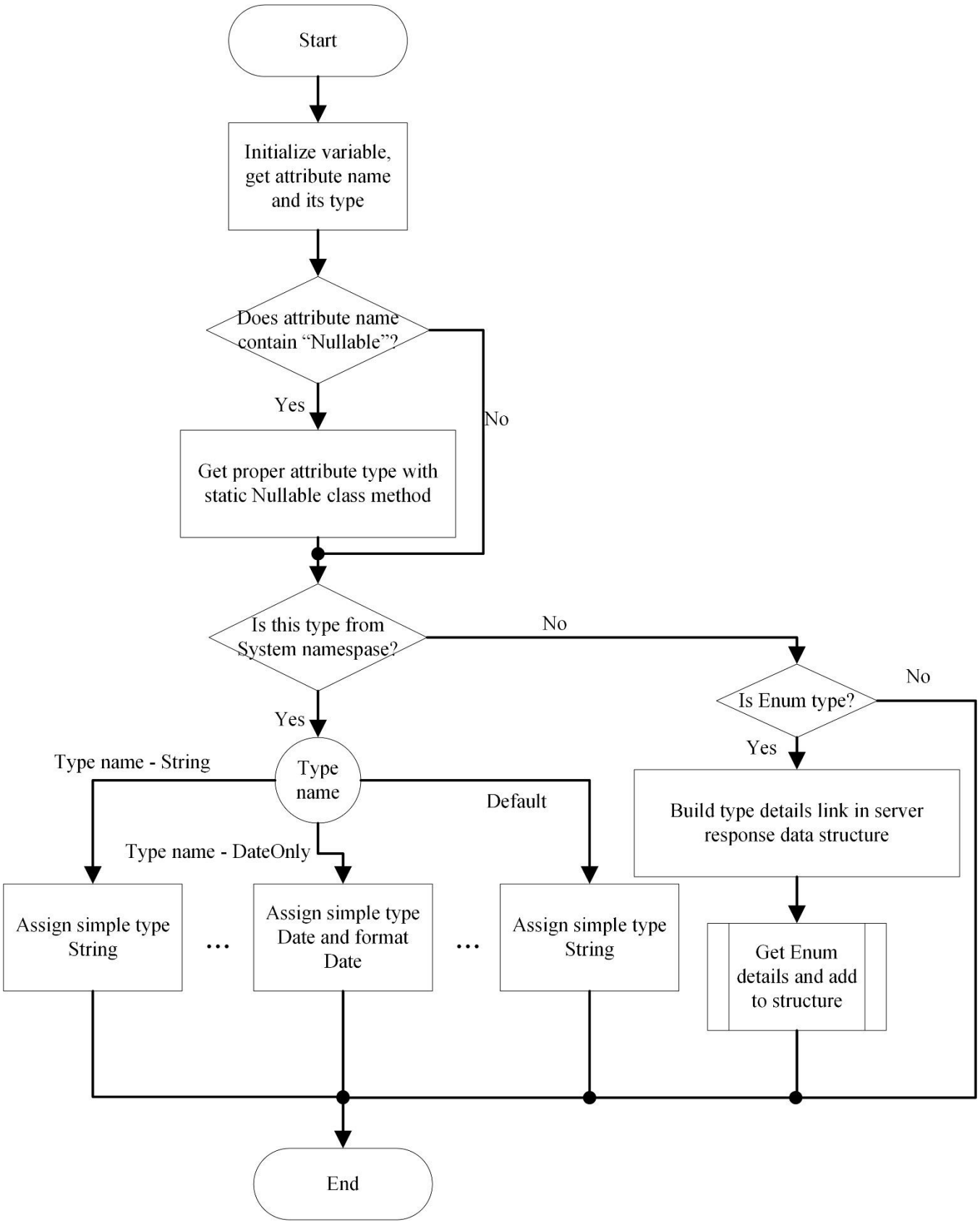


Figure 4 – A flowchart of getting attribute scheme method algorithm

This method can be called by api controller GET-request. To get schema and finding a proper way for defining attribute types the backend application was paused in debug mode. At this moment there was the widest entity with a large set of different types of attributes. There was some kind of research to find type feature that could help to distinguish them to groups and process. So, there is information, which was got by this research (all of these flags have true value, and these flags collections were filtered by usability further):

- Int32 – integer numbers format;
    - flag – IsPrimitive;
    - flag – IsValueType;
    - flag – IsTypeDefinition;
- String – string data type;
    - flag – IsTypeDefinition;
- DateTime – date type with time and defined time zone;
    - flag – IsValueType;
    - flag – IsTypeDefinition;
- DateOnly – date type without time;
    - flag – IsValueType;
    - flag – IsTypeDefinition;
- Enum – special class with named properties, which have number values;
    - flag – IsEnum;
    - flag – IsValueType;
    - flag – IsTypeDefinition.

It is clear, that with this data – any separation or defining what is what is impossible. But there is some useful information. We ca use flag «IsEnum» to know that there is complicated user's attribute type. Moreover, it is convenient, because the rest of attributes is System-based. It means that if we check whether type's namespace is "System" or not and check only type name (such as String or Int32).

The other question is how to get user's Enum values. To extract useful information, the research like the described above was conducted. So, there are several statements regarding this need (based on research results).

- We can get all Enum values, based on Enum attribute type in backend application runtime. To do this we should run:

$$\mathrm{enumValues} = \mathrm{enumType.GetEnumValues}(),$$

  where $enumType$ – Clr [17] Enum attribute type, which is one of Enum values.

- To get proper Enum class values, «for» cycle could be used. So, every class item will have iteration number, this sequence represents the default Enum behavior. Also, the exact value can be got in each iteration via transforming property name through underlying Enum type to its value. It can be made by:

$$\mathrm{name} = \mathrm{enumValues.GetValue(i)},$$

$$\mathrm{intValue} = \mathrm{Convert.ChangeType}\big(\mathrm{name}, \mathrm{Enum.GetUnderlyingType(enumType)}\big),$$

  where $name$ – Enum string property label;

  $enumType$ – Clr [17] Enum value type;

  $\mathrm{intValue}$ – Enum property number value.

  If some of values differ with iteration number, we can note that there are special values configured in Enum. It could help on typing generation, some kind of optimization.

- We set key-value pair to dictionary. There key – string Enum property label, value – its number value.

Database scheme service implementation is clear on this stage. Next important parts are websocket and transactions services.

7.2.2 Transactions backend service implementation

Microsoft company has developed and provided SignalR library for Asp.Net Core. This library works on websockets mostly (there are several other ways if websockets are

unavailable). This package gives a possibility for data exchange with some specific handler. This handler will be called with the provided arguments on client side.

To work with SignalR, server-side application has special controllers registered – hubs. Every hub has its own path (url), which is used by clients to establish websocket-connection. There are several asynchronous methods inside the hub which would be called by connected client. To call one of them, mobile applications should provide name of handler-method. Arguments deserialization and dependency injection are the same like in common controller.

The most important method in the hub is synchronization handler which receives transactions from handheld application and calls related backend service to apply them. Then it sends the applied transactions to other connected clients.

Before we go into service implementation, let us define what is included in transaction. It should be applicable on server and mobile sides.

First of all, it should contain affected table name. There is also change field need, which can be used not only for change operation. In case of insertion this field will contain entire record, on deletion – it will be empty. Also, the good decision is to make instance identifier like a separate field. This field allows us to get instance identifier quickly to find its mirror-record in the database. When insertion happens, we write all instance data to change-field and duplicate its identifier to corresponding property.

There was an issue with proper entity class determination while testing intermediate solution. This class was required to instantiate new entities, get class fields content and apply changes to the found records. Special technical information has been added to solve class determination issue: entire class name with its version and identifier, solution name (if entity class is stored under other project solution, such as external library or your own structural solution, we cannot access it without exact solution knowledge). This extra data was added for every table in a server response (database scheme dto). This extra technical information is useless for handheld application and further it can be simplified, but now it is used for backend application synchronization.

Transaction should also have information about change type. There could be insertion, update and delete change operation. It is convenient to have creation and synchronization date. It allows us to get transactions in a proper order. So, the transaction structure is as follows:

- transaction identifier;
- affected table name;
- entire entity class name;
- entity class’s solution;
- applied changes dictionary;
- change type;
- affected instance identifier;
- creation date;
- synchronization date.

Let us dig deeper into service for transactions applying. The main method there is function which is responsible for getting transactions and applying them to server database.

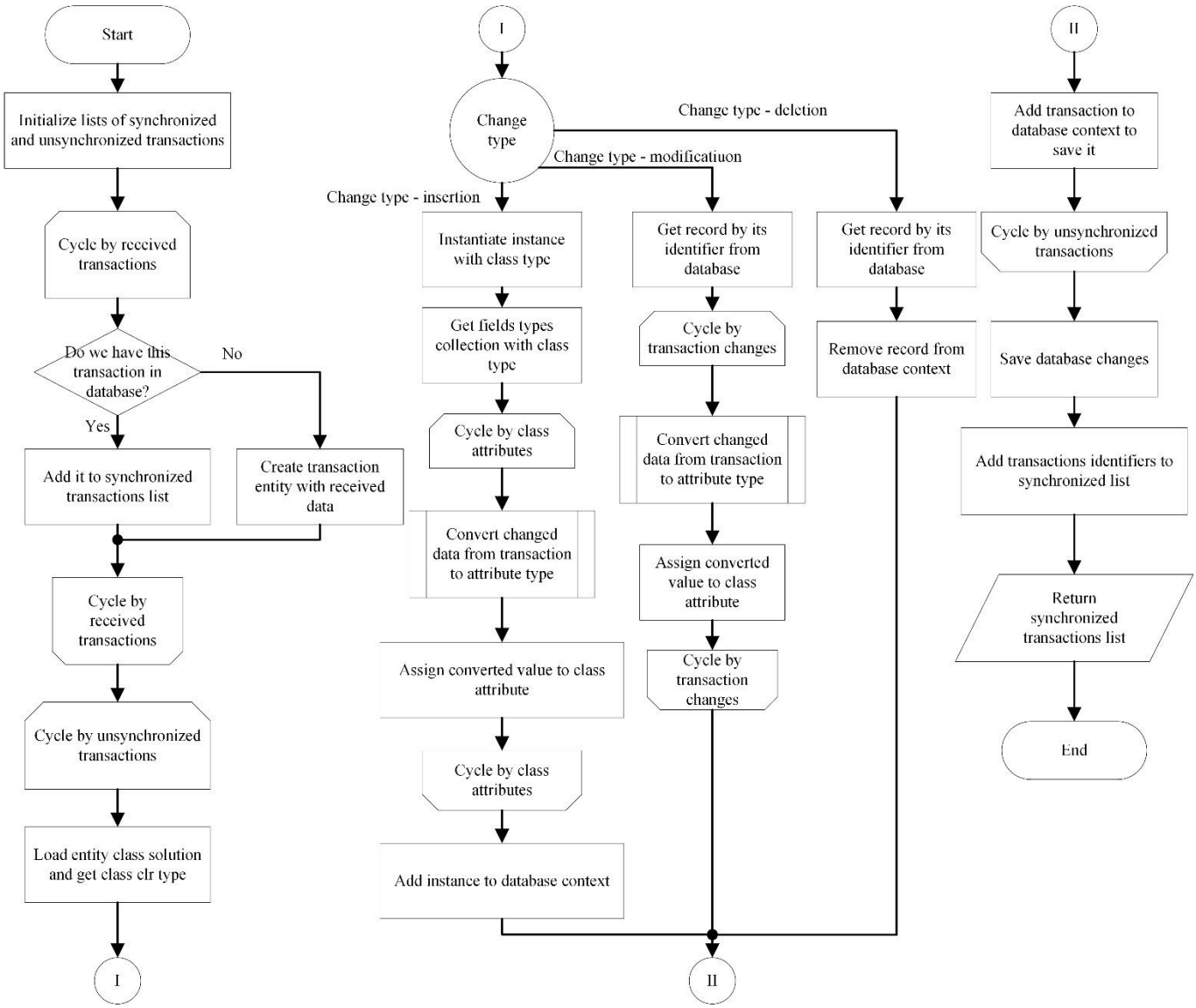


Figure 5 – A flowchart of applying transactions method algorithm

Firstly, we split up transactions in synchronized and unsynchronized groups. Of course, this case is unlikely happened event, but it is better to handle this case. Then we load a solution with entity class for every table from unsynchronized transaction. This load is cheap, because the file is loaded once, then there will be cached version returned. We get clr class type from loaded solution, which provides class content information, its structure. Also, this clr type can be used to create new class instances. The next step depends on change type, so we check it and do the following:

– on insertion – we create class object with system class Activator, then we get entity fields from its class type and fill them with data. Finally, we use Add method to attach created instance to database context to make it visible for EF Core ORM;

- on modification – we find record in server database by identifier form transaction and class clr type. We can do this with Find database context method. Then we get field clr type for each changed attribute in transaction changes. This type is used for new value assignment. Due to getting this attached instance via Find context method, we can miss adding this object to database context;
- on deletion – find instance with the same way and just detach it from context, EF core will delete in from the database.

For now, we can assume that all changes from new transaction are recorded to the server database. We need to save all context detected changes to make it real for the database. All applied transactions are grouped in one list and are going to be delivered to other connected clients. We return all applied transaction identifiers to client, who started this synchronization, to notify him about operation result.

There is one small, but important part – data type casting. C# is one of programming languages that has typing in runtime. It means that if we try to assign some data with wrong type to some field or variable, it will cause exception and program will crash. So, proper casting is a very important part.

Another important feature of this service is missing transactions determination. It is related to each client. We should provide each of them with an individual list of missing transactions which were applied to server database when this client was disconnected. To do this we can extend hub on-connected handler. To get this transactions list we should know – what transaction was the last synchronized for this client. Due to connection hub request is not a common http request, it does not have headers and request body, we can only configure a path, or to be more precisely, query parameters. Let us make agreement, that client will place its last synchronized transaction identifier.

Query parameters from path are not parsed automatically, like we can expect in api-controller. To work with them, we should get request context and extract data from it with a special query parameter name.

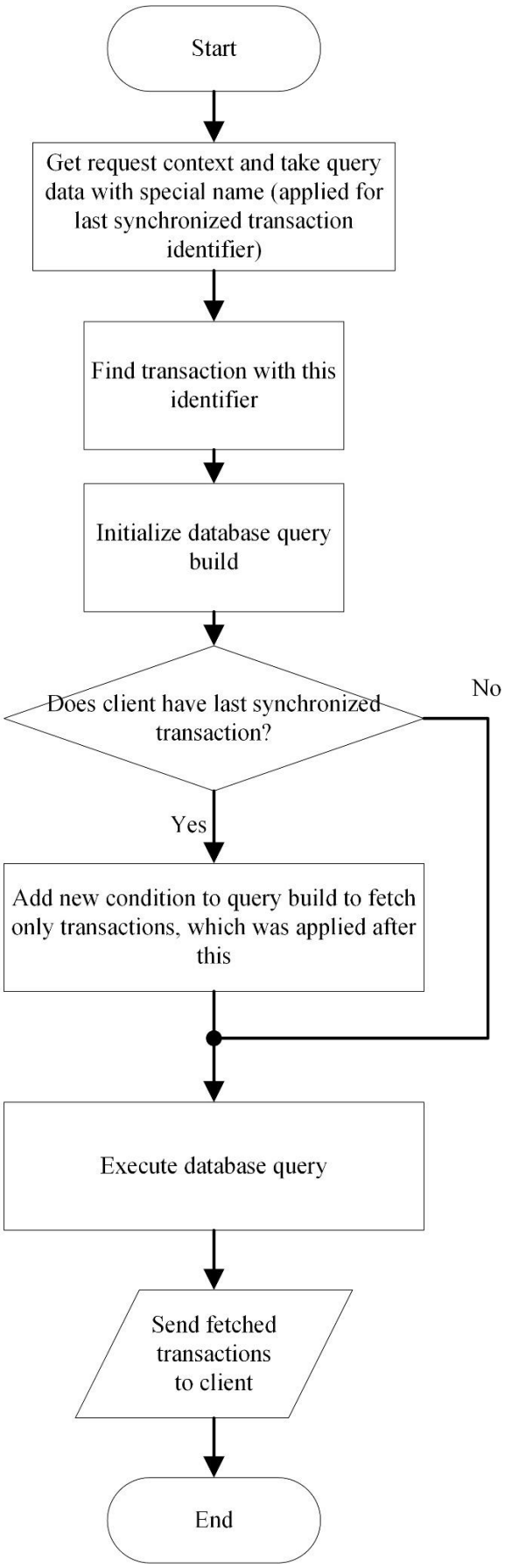


Figure 6 – A flowchart of hub connection handler algorithm

This feature is used in a case when there is a new client and he does not have any data or transactions. He will just receive all transactions from the server. Sure, there is not the best solution and it will be replaced with api-call further.