

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Профиль Вычислительные машины, комплексы, системы и сети

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ
РАБОТА БАКАЛАВРА**

на тему

Смарт-органайзер для ОС Андроид

Студент


(подпись, дата)

Соловьев Антон Алексеевич

Руководитель


(подпись, дата)

Егоров В.Ю.

Нормоконтролёр


(подпись, дата)

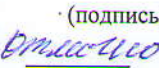
Кучин А.В.

Работа допущена к защите (протокол зас. кафедры от 03.06.2020 № 12)

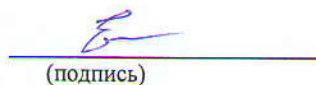
Заведующий кафедрой



Митрохин М.А.

Работа защищена с отметкой  (подпись) (протокол заседания ГАК от 17.06.20 № 13)

Секретарь ГЭК


(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2020

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
Вычислительная техника

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«30» апреля 2020г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Соловьев Антон Алексеевич гр. 16BV1 факультета вычислительной техники.
2. Тема работы «Смарт-органайзер для ОС Андроид».
Тема утверждена приказом ПГУ №0335а/о от "30" апреля 2020 г.
3. Руководитель работы к.т.н., доцент кафедры ВТ ПГУ Егоров В.Ю.
4. Задание на работу (назначение разработки, исходные данные и т.п.).
Назначение: мобильное приложение под операционную систему Android, предназначенное для организации личного времени, составления списков задач, формирования привычек, отслеживания продуктивности.
5. Перечень подлежащих разработке вопросов.
Требуется произвести анализ предметной области, определить наиболее актуальные и подходящие методы и инструменты для разработки мобильного приложения, подобрать способ организации серверной стороны приложения.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10	01.05.20	
2.	Выбор стека технологий для приложения	20	04.05.20	
3.	Разработка интерфейсных компонентов	30	10.05.20	
4.	Разработка бизнес-логики приложения	30	17.05.20	
5.	Тестирование и отладка	10	24.05.20	
6.	Оформление выпускной работы	30	28.05.20	

Дата выдачи задания "30" апреля 2020 г.

Руководитель бакалаврской работы  16.06.2020 Егоров В.Ю.
(подпись, дата)

Задание к исполнению принял студент  16.06.2020 Соловьев А.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа 149 листов, 28 рисунков, 7 источников, 2 приложения.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, СМАРТ ОРГАНАЙЗЕР, ANDROID, REACT NATIVE, ОРГАНИЗАЦИЯ ЛИЧНОГО ВРЕМЕНИ, СОСТАВЛЕНИЕ ЗАДАЧ.

Цель работы — разработать мобильное приложение под ОС Android, позволяющее осуществлять организацию личного времени путем составления списков задач.

В результате проделанной работы реализован проект, удовлетворяющий поставленной задаче, а именно, разработано мобильное приложение для планирования задач.

Данное приложение рекомендуется использовать частным лицам для организации своего распорядка дня, фокусирования на определенных целях, отслеживания собственной продуктивности.

					ПГУ 2.090301.35.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Соловьев А.А.			Смарт-органайзер для ОС Android. Пояснительная записка.	Лит.	Лист
Провер.		Егоров В.Ю.					Листов
Н. Контр.		Кучин А.В.					2
							149
Утверд.		Митрохин М.А.				ФВТ, каф. ВТ гр. 16ВВ1	

Содержание

Реферат	2
Введение	5
1. Анализ предметной области.....	6
1.1. Постановка задачи.....	6
1.2. Актуальность.....	6
1.3. Обзор аналогов	8
1.4. Стек технологий для реализации приложения	10
1.4.1. Анализ требований.....	10
1.4.2. Выбор актуального метода разработки.....	10
1.4.3. React Native	13
1.4.4. React Native Navigation от Wix.....	15
1.4.5. React Native Gesture Handler.....	16
1.4.6. Firebase	17
1.4.7. Realm база данных	18
2. Реализация мобильного приложения	19
2.1. Описание работы приложения	19
2.2. Особенности разработки интерфейса.....	22
2.2.1. Основные понятия для построения интерфейса	22
2.2.2. Реализация интерфейса главного экрана	26
2.2.3. Реализация интерфейса бокового меню	28
2.2.4. Реализация интерфейса создания новой задачи.....	30
2.3. Реализация анимации	34
2.4. Работа с базой данных	36
2.5. Реализация серверной стороны.....	39
2.6. Тестирование приложения.....	41
2.6.1. Тестирование аутентификации	41
2.6.2. Тестирование создания задачи	43
2.6.3. Тестирование редактирования задачи	44

2.6.4.	Тестирование взаимодействия потоков	45
3.	Руководство пользователя	47
3.1.	Авторизация	47
3.2.	Главный экран.....	47
3.3.	Работа с задачами	49
3.4.	Особенности работы со списками	50
3.5.	Доступные настройки	52
4.	Безопасность жизнедеятельности	53
4.1.	Влияние мобильных устройств на организм	53
4.2.	Рекомендуемые меры предосторожности.....	53
	Заключение.....	55
	Список использованных источников.....	56
	Приложение А. Листинги мобильного приложения	57
	Приложение А.1. Components	57
	Приложения А.2. Screens	103
	Приложения А.3. Sources.....	130
	Приложения А.4. Styles	137
	Приложения А.5. Index.js	138
	Приложение Б. Презентация.....	142

Введение

На момент выполнения данной выпускной квалификационной работы мобильная разработки продолжает активно развиваться, мобильные устройства на базе двух ключевых операционных систем (Android от Google и iOS от Apple) с каждым годом становятся все более распространёнными среди населения Земли, их количество продолжает расти, хоть и не так стремительно, как это было раньше. По данным за начало 2020 года смартфонами владеет 45% населения Земли, в развитых странах Европы и Америки этот показатель достигает 80%. Мобильные устройства играют всё большую роль в жизни современного человека, предоставляют возможность получить необходимую информацию и удобным способом обработать ее, существенно упрощают многие процессы жизни.

При этом вместе с ростом количества мобильных устройств изменяются и подходы к разработке мобильных приложений для них. Если раньше для написания приложения приходилось вести разработку на определенных языках программирования под соответствующие платформы, то теперь все большую популярность набирает разработка на кроссплатформенных решениях, таких как React Native от Facebook, Flutter от Google, Xamarin от Microsoft.

Как мы можем видеть, в разработке кроссплатформенных приложений заинтересованы практически все основные IT-компании, задающие тренды разработки для мобильного рынка. Ведь у такого подхода несомненно есть основное ключевое преимущество – сокращение времени разработки, что в свою очередь позволяет существенно сэкономить при выходе на несколько платформ одновременно. И в дальнейшем использовать сэкономленные средства для развития продукта, основываясь на отзывах реальных пользователей.

1. Анализ предметной области

1.1. Постановка задачи

Требуется разработать мобильное приложение под операционную систему Android, предоставляющее возможность эффективно организовывать личное время пользователя, упрощать отслеживание как глобальных целей, так и мелкой рутины.

Мобильное приложение должно обеспечивать следующий функционал:

- авторизация пользователя;
- отображение списка задач на заданный интервал времени с учетом оставшегося свободного времени;
- создание задач с добавлением к ним определенных свойств, таких как необходимое время для исполнения, срок выполнения, привязанность к определенному списку, возможность повторения задачи каждую неделю или каждый день;
- создание и удаление списков задач для их последующего наполнения задачами;
- возможность выполнения, редактирования, удаления задач из каждого списка;
- автоматическое группирования задач в зависимости от их свойств;
- настройка профиля пользователя.

Приложение должно обладать интуитивно понятным графическим интерфейсом и адекватной производительностью на большинстве современных устройств.

1.2. Актуальность

Ежедневно человек сталкивается с огромным количеством поступающей информации, большая часть этой информации из социальных сетей, средств массовой информации является по большому счету бесполезным

					ПГУ 2.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		6

информационным шумом. Тем не менее источникам этого «шума» удастся захватывать наше внимание и отвлекать нас от приоритетных задач в течение дня. Мозг человека идет по наименьшему сопротивлению и при условиях выбора между работой и развлекательным контентом практически всегда делает выбор в пользу второго. Это существенным образом усугубляет проблему прокрастинации.

Держа все задания в уме, мы откладываем неприятные и рутинные дела на последний момент или же вовсе забываем о них до тех пор, пока это не выльется в серьезные проблемы. Задачи, выполненные в спешке и в последний день, вряд ли будут в лучшую сторону отличаться качеством. При этом с каждым днем усиливается чувство собственной нереализованности, падает самооценка, это в свою очередь дополнительно вызывает неуверенность.

Большую часть вышеописанных проблем поможет решить планирование своей деятельности в течение дня, в идеальном случае в течение нескольких недель (хотя бы ключевые цели).

Современному человеку необходим современный подход для решения проблемы. Мобильное приложение для планирования задач поможет:

- не забыть о всех заданиях на день;
- четко осознавать план действий на ближайшее время;
- даже примерный записанный план на каждый день позволит помнить о негативных последствиях в случае задач на следующий день;
- получить чувство удовлетворения после выполнения всех задач на день, в следствие чего повышается уверенность в себе;
- двигаться к глобальным целям путем дробления их на мелкие подзадачи.

Особо это актуально для тех, кто большую часть времени работает из дома. В условиях дома отвлекающих факторов еще больше и без четкого плана действий работать продуктивно весьма проблематично. Я сам сталкиваюсь с

такого рода, поэтому и было принято решения написать свой планировщик, конкретно под свои задачи.

Проблемы, описанные в предыдущем пункте, касаются подавляющего большинства людей в той или иной степени, на рынке мобильных приложений под обе операционные системы (iOS и Android) существует огромное количество решений. Большинство из этих решений обладает неплохим количеством скачиваний и активных пользователей. Для подкрепления данных слов приведем примеры на рисунке ниже:

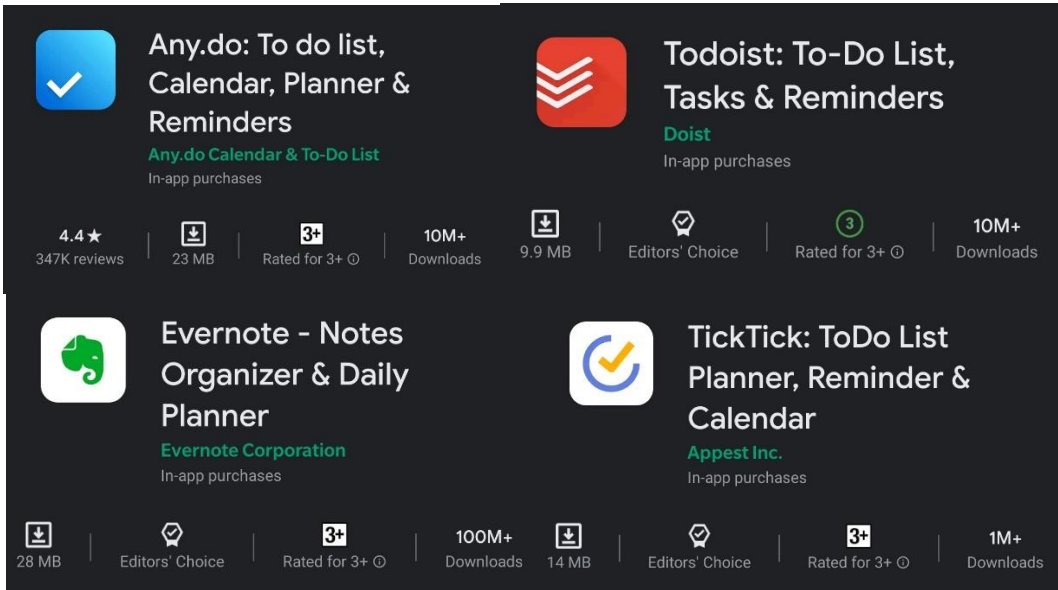


Рисунок 1.1. Количество загрузок у todo приложений.

1.3. Обзор аналогов

Большое количество приложений может говорить не только об актуальности проблемы среди населения, но и о сложности создания идеального решения. Стоит отметить, что приложения такого плана появляются практически каждый месяц. У людей есть различные требования к мобильным приложениям такого рода, кому-то необходим широкий набор функционала для досконального расписания своих планов, анализа собственной производительности в течение периода времени. Других же напротив отпугивает такое количество функций, их большой набор может существенно увеличить

время составления плана и со временем превратит его в еще одну неприятную задачу.

Давайте перейдем к рассмотрению основных аналогов. Базовый функционал у всех одинаковый и состоит из составления задач с определенными свойствами, формирования этих задач в соответствующие списки. Также сходства проявляются и в графическом интерфейсе, пример интерфейса приведен на рисунке 1.2.

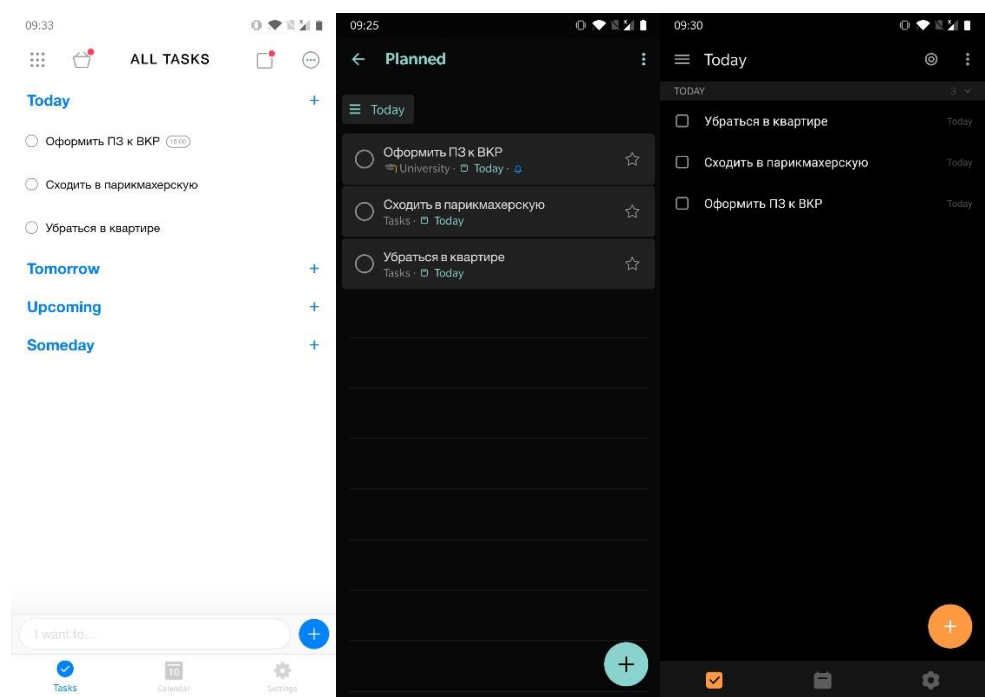


Рисунок 1.2. Пример интерфейса todo приложений.

На данном рисунке слева направо представлены приложения Any.do (10+ млн. скачиваний), Microsoft To Do (5+ млн. скачиваний) и TickTick (1+ млн. скачиваний). Да, можно сказать, что интерфейс у них весьма схож, но различия проявляются в нюансах, которые и заставляют пользователей склоняться к определенным вариантам.

Эти нюансы и вызвали у меня необходимость написания собственного аналога. Мне не хватало функционала указания времени исполнения конкретной задачи и автоматического подсчета оставшегося на день свободного времени. Человек склонен переоценивать свои возможности и планировать на день большее количество задач, чем он действительно способен выполнить. В

результате чего невыполнение всего списка запланированных задач приводит опять же чувству вины, нереализованности.

1.4. Стек технологий для реализации приложения

1.4.1. Анализ требований

Перед тем как приступить к разработке любого программного обеспечения необходимо проанализировать требования к конечному продукту и на основе этих данных уже определиться со стеком технологий. В рамках выпускной квалификационной работы приложение должно быть преимущественно написано под ОС Android, тем не менее в дальнейшем хотелось бы без существенных дополнительных затрат перенести приложение на другие популярные платформы, в частности iOS от Apple и web. И на текущий момент существует два варианта: нативная или кроссплатформенная разработка. Стоит пояснить, что под нативной подразумевается разработка с применением оригинальных языков и средств для конкретной операционной системы. В случае с iOS от Apple – это будет Swift и xCode, если рассматривать ОС Android от Google – то java/kotlin и Android Studio.

При кроссплатформенной разработке мы используем специализированные языки и средства, позволяющие писать практически идентичный код под обе операционные системы одновременно. Несомненно, у этих двух подходов есть свои преимущества, так и минусы, и далее мы рассмотрим их.

1.4.2. Выбор актуального метода разработки

Мы не будем уделять внимание минусам и плюсам языков и средств разработки под конкретные операционные системы, а взглянем на общую картину нативной разработки.

Ключевыми плюсами нативного подхода к разработке являются скорость работы и отклика интерфейсов написанных приложений, в связи с глубокой интеграцией с ОС. Тем не менее с развитием кроссплатформенных решений и

					ПГУ 2.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		10

увеличением производительности мобильных устройств, этот плюс становится все менее существенным. Глубокая интеграция с операционной системой также позволяет и удобно работать с различными функциями ОС, опрашивать датчики устройства, оперировать родными элементами интерфейса, такими как списки, меню, кнопки, навигация. Родные элементы интерфейса более привычны и удобны пользователю приложения.

Поддерживать «нативный» код все же проще, чем кроссплатформенный, обновление систем реже приводит к необходимому обновлению приложения. Отладка приложения на нативных инструментах гораздо удобнее, ведь кроссплатформенные среды лишь транслируют запросы к системным вызовам, это в свою очередь может вызывать недопонимание и снижает быстроедействие. Также стоит отметить огромное комьюнити, сформировавшееся вокруг нативной разработки, скорее всего, проблемы, с которыми вы столкнетесь в процессе программирования, уже были решены до вас, и вы с легкостью сможете найти ответ на форумах.

Но несмотря на эти плюсы, у нативного подхода имеется существенный недостаток в виде стоимости разработки, если вы планируете выход сразу на несколько платформ. В таком случае, приходится либо увеличивать штат разработчиков, либо существенно увеличивать время разработки, ведь продукт придется писать несколько раз под каждую платформу, используя разные языки и среды разработки. Этот недостаток и послужил развитию кроссплатформы.

При кроссплатформенном подходе нет необходимости поддерживать полностью уникальный для каждой операционной системы код, продукт на нескольких ОС будет работать на одном языке. Подобные решения стремительно развиваются из года в год, но не тем не менее до сих пор не избавились от существенных недостатков, которые мешают стать данному подходу «серебряной пулей».

В первую очередь, это производительность и удобство интерфейса. Да, кроссплатформенные решения прекрасно работают и существенно сокращают время разработки в случае небольших и простых проектов, быстрой сборки

					ПГУ 2.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		11

MVP. Но если речь заходит о крупном проекте, тесно взаимодействующем с операционной системой, требующем повышенного внимания к безопасности и стабильности, то здесь уже возникают проблемы. Дело в том, что в своем подавляющем большинстве, кроссплатформенное решение выступает в роли абстракции для разработчика, позволяющей писать ему единый код под несколько ОС, и в дальнейшем именно не прямое «общение» приложения с ОС через абстракцию вызывает проблемы с производительностью. Единый код значит и единый интерфейс для всех ОС, это лишает приложения особенностей UX каждой операционной системы и вполне может вызывать недопонимание работы интерфейса у пользователей.

Android и iOS часто обновляются, внедряются новые возможности, расширяется функционал, но это зачастую может быть недоступно разработчикам, использующим кроссплатформу, трудно поддерживать совместимость приложений с ограниченными инструментами, потребуется дополнительное время.

В целом, большинство проблем можно решить интегрированием нативных модулей в приложение и разделением интерфейса под каждую операционную систему. Но при большом количестве нативных модулей и различий в интерфейсе под платформы стоит задуматься о необходимости использования кроссплатформенного подхода.

Рассмотрим популярность самых актуальных фреймворков среди людей на рисунке 1.3. за последние 5 лет (2015-2020):

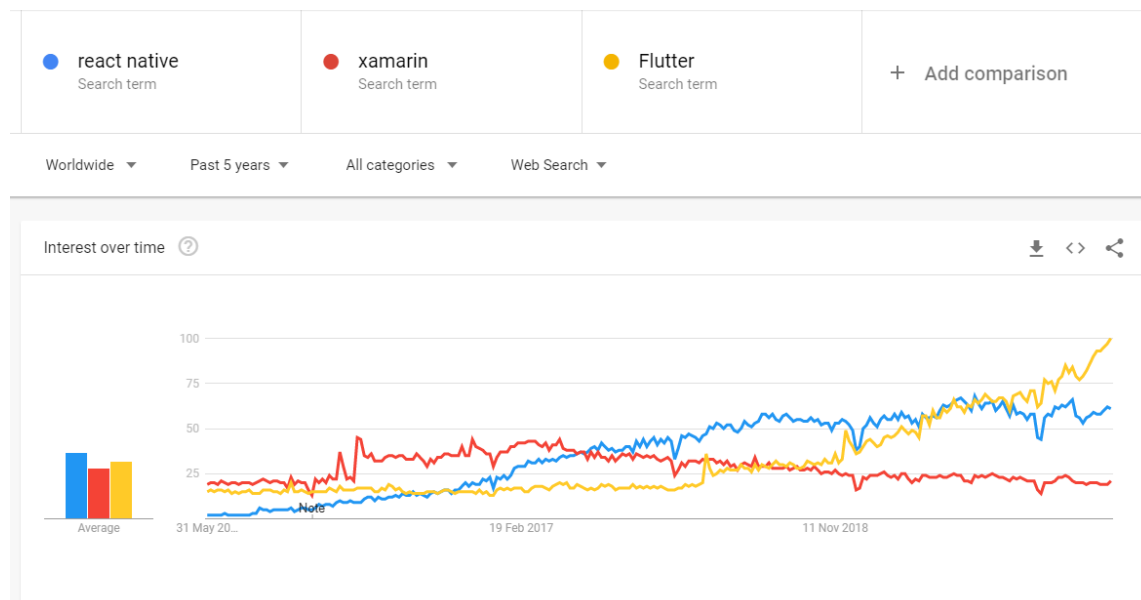


Рисунок 1.3. Популярность фреймворков за последние 5 лет.

Flutter от Google является самым «молодым» решением и тем не менее, как мы видим, в последний год ему удалось все же завоевать популярность среди людей.

Для разработки данной ВКР воспользуемся кроссплатформенным подходом и возьмем в качестве фреймворка React Native от Facebook, на это есть ряд веских причин, которые мы рассмотрим в следующем разделе.

1.4.3. React Native

React Native – фреймворк для кроссплатформенной разработки от Facebook на JavaScript, с вытекающими из этого языка плюсами и минусами.

Из плюсов можно отметить, что приложения, написанные при помощи React Native, работают практически также быстро, как и нативные. Это достигается тем, что примитивы RN транслируются в нативные компоненты интерфейса каждой операционной системы. По итогу при запуске приложения мы имеем 2 потока, один основной – полностью нативный, отображающий интерфейс приложения и второй JS поток, отвечающий за всю бизнес-логику. Для общения между потоками используется мост, позволяющим потокам максимально оптимизированным образом взаимодействовать, не блокировать

друг друга и приводить сообщения в необходимый формат. Общая схема взаимодействия представлена на рисунке 1.4:

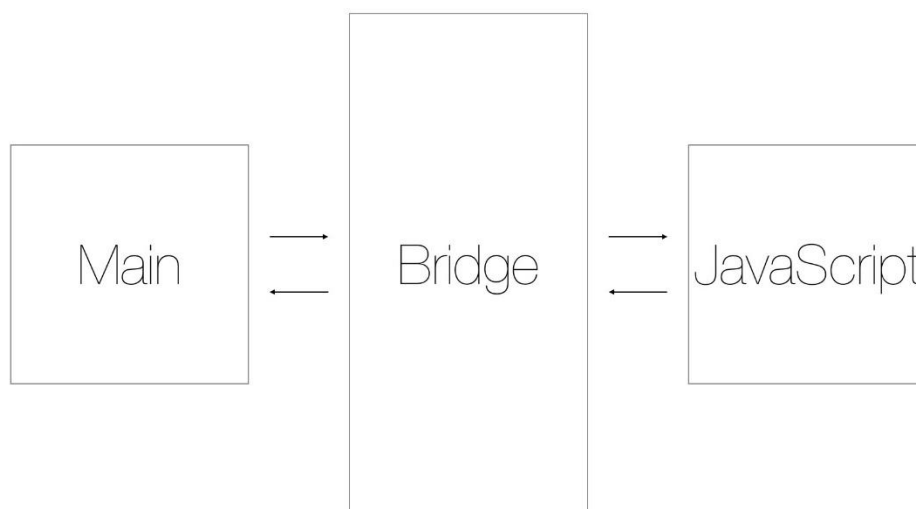


Рисунок 1.4. Общая схема взаимодействия потоков.

К тому же я знаком с JavaScript и React Native это, пожалуй, лучшее для меня решение, позволяющее писать приложения, состоящие практически на 95% из кроссплатформенного кода. К тому же React Native имеет такой привычный для веба hot-reload, позволяющий сразу же видеть вносимые изменения, не перезагружая при этом приложение. Таким образом, выбирая React Native в качестве фреймворка для кроссплатформенной разработки мы выбираем и JavaScript в качестве основного языка программирования.

Вокруг React Native уже успело сформироваться большое сообщество программистов, появились всевозможные готовые решения, компоненты. Многие крупные компании активно внедряли его в свои проекты, из наиболее известных можно отметить Airbnb, Discord, Skype, Instagram, Facebook, Tesla, Oculus, Uber Eats.

Тем не менее у RN также есть и недостатки, которые обязательно следует учитывать, один из них как раз связан с использованием моста, который может выступать «узким горлышком» в приложении. Происходит это в случае частого обновления интерфейса в приложении, поэтому могут возникнуть сложности при использовании сложных анимаций в больших проектах.

Для полноценного применения React Native необходимо использовать готовые модули от сообщества разработчиков, рассмотрим ряд таких модулей.

1.4.4. React Native Navigation от Wix.

Практически любому мобильному приложению нужна навигация и в своей официальной документации разработчики React Native рекомендуют два возможных решения: простое JavaScript решение от сообщества (React Navigation / RN) и React Native Navigation (RNN) от разработчиков Wix, предоставляющее производительность, схожую с родными для операционной системы компонентами навигации.

Оба решения имеют хорошую документацию по кейсам использования и API, тем не менее документация React Navigation подробнее и доступнее. Но все же выбор оставляю именно на React Native Navigation от Wix, в его пользу говорит существенный прирост производительности при переходе на него с RN. Установка модуля RNN проста и выполняется парой команд в консоли (необходим пакетный менеджер npm):

```
$ npm install --save react-native-navigation
```

```
$ react-native link react-native-navigation
```

После установки модуля мы можем перейти к его использованию в проекте, для этого импортируем его в index.js, зарегистрируем каждый «экран» приложения и установим стек навигации. Реальный пример из ВКР:

```
import { Navigation } from "react-native-navigation";  
Navigation.registerComponent('loginScreen', () => Login);  
Navigation.registerComponent('taskViewerScreen', () =>  
  gestureHandlerRootHOC(TaskViewer));  
...  
Navigation.events().registerAppLaunchedListener(() => {  
  Navigation.setRoot({  
    root: {
```

```

    stack: {
      children: [
        {
          component: {
            name: 'loginScreen'
          }
        }
      ]
    }
  }
});

```

1.4.5. React Native Gesture Handler

Взаимодействие с современным приложением сложно представить без обработки определенных жестов со стороны пользователя. Для упрощения их обработки воспользуемся готовым решением от сообщества – React Native Gesture Handler, который нацелен на замену встроенной в React Native системы отслеживания жестов. Также мотивацией к замене встроенной системы послужили ее ограничения по производительности.

Если кратко, то React Native Gesture Handler предоставляет следующие возможности:

- простой механизм отслеживания ключевых жестов, таких как свайпы, повороты, различного рода нажатия;
- плавная реализация жестов благодаря использованию Animated Native Driver, взаимодействия будут отзывчивыми даже если JS поток перегружен;
- декларативный подход к отслеживанию жестов.

Для установка данного модуля необходимо выполнить несколько команд в консоли:

```

$ npm install --save react-native-gesture-handler
$ react-native link react-native-gesture-handler

```

После успешной установки мы можем использовать модуль в нашем проекте, необходимо только импортировать в необходимый для использования файл:

```
import { gestureHandlerRootHOC } from 'react-native-gesture-handler';
```

Также стоит учитывать особенности работы данной библиотеки с компонентами React Native Navigation: нам необходимо «обвернуть» каждую регистрацию экрана приложения в специальный метод:

```
Navigation.registerComponent('optionsScreen', () =>  
  gestureHandlerRootHOC(Options));
```

Более подробно взаимодействие с жестами описано в разделе 2.2. «Особенности разработки интерфейса».

1.4.6. Firebase

Роль серверной части приложения будет выполнять набор инструментов Firebase от Google. Использование Firebase позволяет нам не писать ни строчки для бэкэнд стороны приложения, при этом мы получаем быстрый механизм авторизации, поддержка от Google придает дополнительной уверенности в стабильности работы. Бесплатный тариф более чем покрывает наши студенческие потребности, а платный не сковывает при масштабировании проекта.

Но это если касаться только авторизации, в случае хранения и синхронизации данных пользователей Firebase уже бьет по карману при большом количестве запросов к нему. Для сокращения количества запросов к Firebase мы будем использовать локальную базу данных Realm. В результате синхронизация будет происходить не при каждом изменении данных, а через определенный интервал времени.

1.4.7. Realm база данных

Как уже было сказано в предыдущем пункте мы не станем использовать облачную Firebase базу данных от Google из-за ее высокой стоимости при масштабировании проекта. По этой причине нам стоит выбрать локальную базу данных.

В качестве локальной базы данных выступит Realm DB, которая имеет открытый исходный код и является объектно-ориентированной базой данных, имеет собственный механизм и не просто полагается на хранилища значений ключей. Объектно-ориентированная модель делает его почти в 10 раз производительнее, чем SQLite базы, и избавляет вас от выполнения множества запросов, что является обычным явлением для типичной базы данных SQL.

Помимо этого, Realm не использует традиционный способ чтения данных из базы, который приводит к ненужному копированию в объекты необходимого языка, а избегает этого, отображая все данные в памяти, используя B + деревья, и всякий раз, когда данные запрашиваются, просто вычисляет смещение, считывает из отображенной области памяти и возвращает исходное значение. Это объясняет то, почему разработчики предпочитают Realm при работе с большим объемом данных.

Кроме того, использование Realm DB является еще и безопасным, для каждой платформы он использует соответствующий метод шифрования данных. Например, для Android используется AES-256, для iOS CommonCrypto библиотека.

Установка, как мы уже привыкли, не занимает много времени и выполняется в пару команд:

```
$ npm install --save realm
```

```
$ react-native link realm
```

2. Реализация мобильного приложения

2.1. Описание работы приложения

Общая схема большинства React Native приложений представлена на рисунке 2.1.

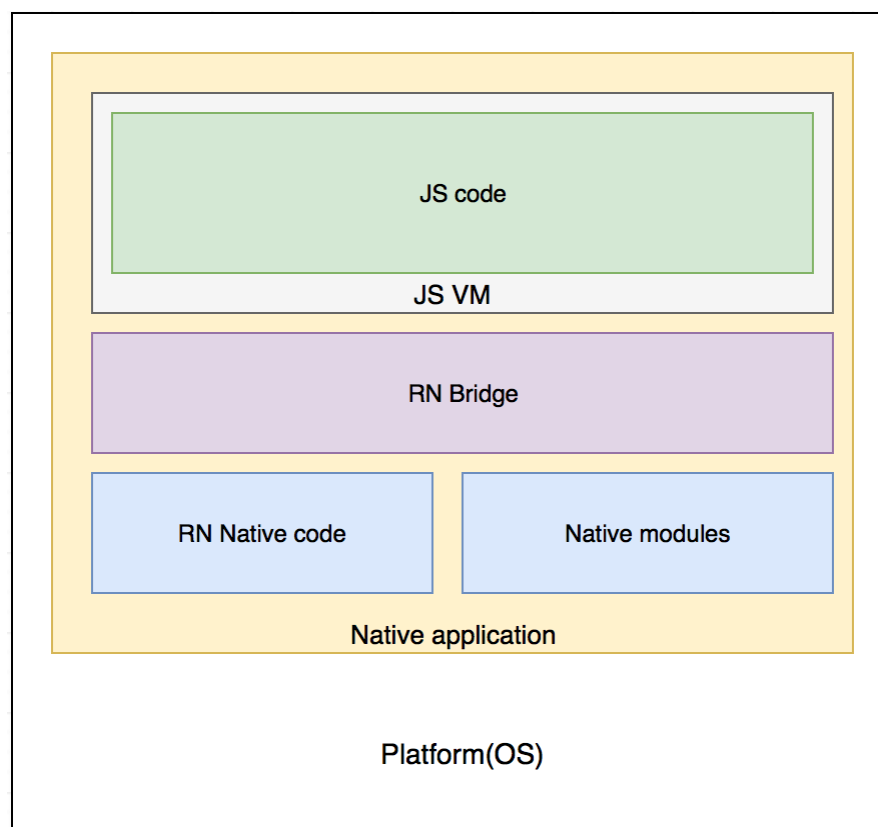


Рисунок 2.1. Общая схема React Native приложений

В операционной системе запущено нативное приложение. В нем на низком уровне работают среда выполнения React Native и код нативных модулей, написанных разработчиком приложения (или автором библиотек для React Native). На уровень выше уже работает React Native Bridge – промежуточное звено между нативным кодом и js, служащее посредником между двумя потоками, и позволяет им «общаться» друг с другом, избегая серьезных проблем с производительностью, а именно общаться асинхронно. Это в свою очередь приводит к стабильной работе, потому что потоки никогда не смогут заблокировать друг-друга. Сам JavaScript код исполняется внутри JS VM, чью

роль исполняет JavaScriptCore. На iOS она предоставляется системой, на Android же приложение подтягивает ее в виде библиотеки.

Рассмотрим подробнее структуру конкретно JavaScript части приложения на рисунке 2.2:

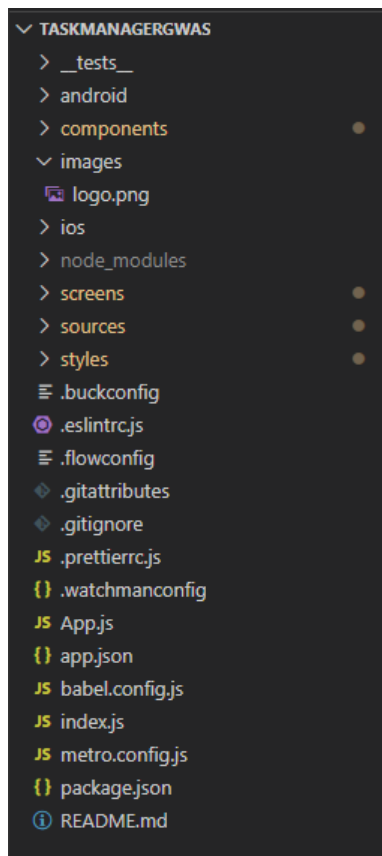


Рисунок 2.2. Структура проекта в Visual Studio Code.

Для понимания распишем основные файлы и папки проекта:

- index.js – основной стартовый файл приложения необходимый для регистрации основных экранов приложения;
- package.json отвечает за установку всех зависимостей и библиотек, использующихся в приложении;
- папка node_modules необходима для хранения всех модулей, библиотек в приложении, каждый установленный сторонний модуль хранится там (как пример, там находится React Native Gesture Handler, Realm DB, RNN);
- папка screens содержит в себе реализацию всех экранов приложения;

- папка components служит для хранения всех написанных в процессе разработки компонентов;
- папка sources необходима для хранения векторных ресурсов, например, всех svg иконок, использованных в приложении;
- папка images хранит в себе всю использующуюся растровую графику;
- папки ios и android содержат нативную реализацию приложения под соответствующие названию платформы;
- папка styles хранит в себе js файлы, определяющие основные стили, базовые для большинства компонентов приложения;

Такой формат содержания проекта помогает не запутаться в процессе разработки, когда решение становится все больше по объему, увеличивается количество написанных компонентов, используемых ресурсов.

Каждый компонент и экран приложения мы рассмотрим более подробно в последующих разделах.

Исходя из постановки задачи на ВКР, общая функциональность и поведение системы может быть представлено в виде следующей UML-диаграммы вариантов использования на рисунке 2.3:

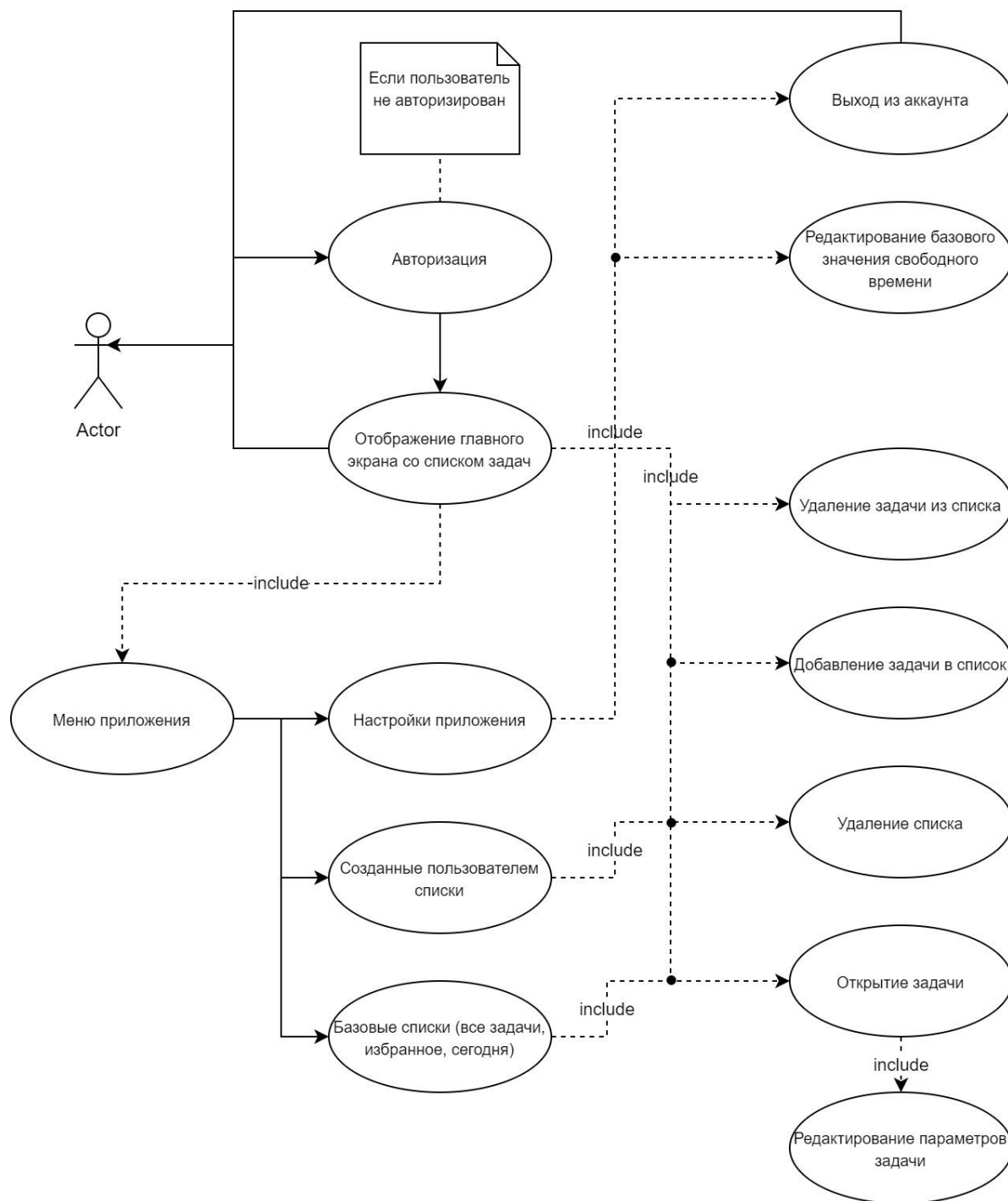


Рисунок 2.3. UML-диаграмма вариантов использования.

2.2. Особенности разработки интерфейса

2.2.1. Основные понятия для построения интерфейса

React Native использует декларативный подход к программированию, в большей степени это значит, что написание интерфейса строится на

компонентах, реализующих различные элементы интерфейса, контроллы, экраны. React Native уже имеет в своем распоряжении «примитивные» базовые компоненты, такие как текстовые блоки, кнопки, контейнеры для изображений, списки, являющиеся своего рода «оберткой» над нативными Android / iOS view. То есть в процессе выполнения компоненты React Native обращаются к своим нативным «аналогам» под каждую платформу. Естественно для проектирования интерфейса собственного приложения необходимо писать «кастомные» компоненты, строящиеся с использованием базовых.

Стоит также понимать, что компоненты в React Native написаны, используя JSX, который является синтаксическим сахаром для JavaScript, позволяющий писать вызов компонентов в таком весьма привычном для веб-разработчиков виде:

```
<Text>Hello, I am your cat!</Text>
```

Две ключевые вещи, которые нам следует знать при разработке компонентов интерфейсов – это props, state.

Props (сокращение от properties – свойства) необходимы для кастомизации наших компонентов при вызове. Предоставляя компоненту props в виде аргументов, мы получаем возможность менять изначальное отображение компонента. При этом props реализован таким образом, что процесс передачи параметров очень схож с установлением атрибутов для html тэгов. Для лучшего понимания посмотрим пример непосредственно из нашего приложения.

Для вызова компонента необходимо написать его «тег» и указать атрибуты:

```
<FABButton onPress={addTask} show={!showAdding}>
```

```
  <AddIcon color={colors.white} />
```

```
</FABButton>
```

Теперь рассмотрим, как компонент использует переданные ему атрибуты, для этого откроем его реализацию в fabButton.js (полную реализацию можно посмотреть в приложении А.1):

```
export class FABButton extends PureComponent {
```

```

    constructor(props) {
      super(props);
      this.state = {
        show: this.props.show,
        pressButton: this.props.onPress,
      }
    }
    render() {
      return (this.props.show ?
        <View style={styles.wrapper}>
          <View style={styles.base}>
            <TouchableNativeFeedback          onPress={()          =>
this.state.pressButton()} background={TouchableNativeFeedback.Ripple('rgb(30, 38,
66)', true)}>
              <View style={styles.body}>
                {this.props.children}
              ...
            )
          }
        )
      }
    }
  }

```

В коде объект props передается в качестве аргумента в конструктор компонента (это строго необходимо):

```

constructor(props) {
  super(props);

```

В последующем используется при помощи обращения к полям объекта:

this.props.show – отвечает за показ кнопки в интерфейсе;

this.props.children – используется для коллбэка дочернего элемента, а именно вызова векторной иконки;

this.props.onPress – передает указатель на функцию, которая будет исполнена по нажатию на кнопку.

Графическое представление данного компонента представлено на рисунке 2.4:

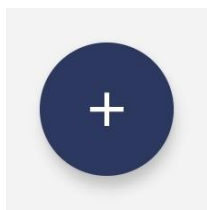


Рисунок 2.4. Графическое представление FABbutton.

Стоит также отметить, что каждый базовый компонент React Native (например, View) имеет атрибут style, необходимый для настройки стилей компонента. Опять же, процесс назначения стилей очень схож с тем, как это происходит при web-разработке, но вместо привычного CSS используется схожий Yoga Layout.

Для удобного объявления стилей в одном месте необходимо использовать объект StyleSheet, который создается методом *StyleSheet.create()*. Наглядный пример из ВКР, в качестве компонента используется все тот же FABbutton:

```
const styles = StyleSheet.create({
  wrapper: {
    bottom: 0,
    right: 0,
    zIndex: 1,
    position: 'absolute',
  },
  base: {
    height: 56,
    width: 56,
    backgroundColor: colors.primary,
    elevation: 4,
    borderRadius: 32,
    margin: 16,
```

```

        justifyContent: 'center',
        alignItems: 'center'
    },
    ...
  })

```

State (состояние) служит в качестве своего рода «памяти» компонента, позволяющей ему менять свое отображение и поведение в течении взаимодействия с пользователем. Так как мы уже проходили курс по паттернам программирования и сталкивались с «состоянием», не будем уделять большое количество времени на его использование.

Важно только понимать, что при написании логики работы стоит как можно реже обращаться к интерфейсу и вызывать его обновление, ведь как мы знаем, в React Native это является слабым звеном, сказывающемся на производительности проекта. Каждое обновление компонента происходит именно при любом изменении state. По этой причине при написании собственных компонентов мы используем наследование от PureComponent, это позволяет обновлять компоненты только при различии нового значения state со старым.

В последующих разделах мы более подробно рассмотрим взаимодействие с интерфейсом на примере ключевых компонентов.

2.2.2. Реализация интерфейса главного экрана

Высокая производительность и плавность приложений на React Native обусловлена тем, что пользователь по сути взаимодействует с нативными компонентами интерфейса ОС. Для наглядности рассмотрим диаграмму деятельности главного экрана на рисунке 2.5:

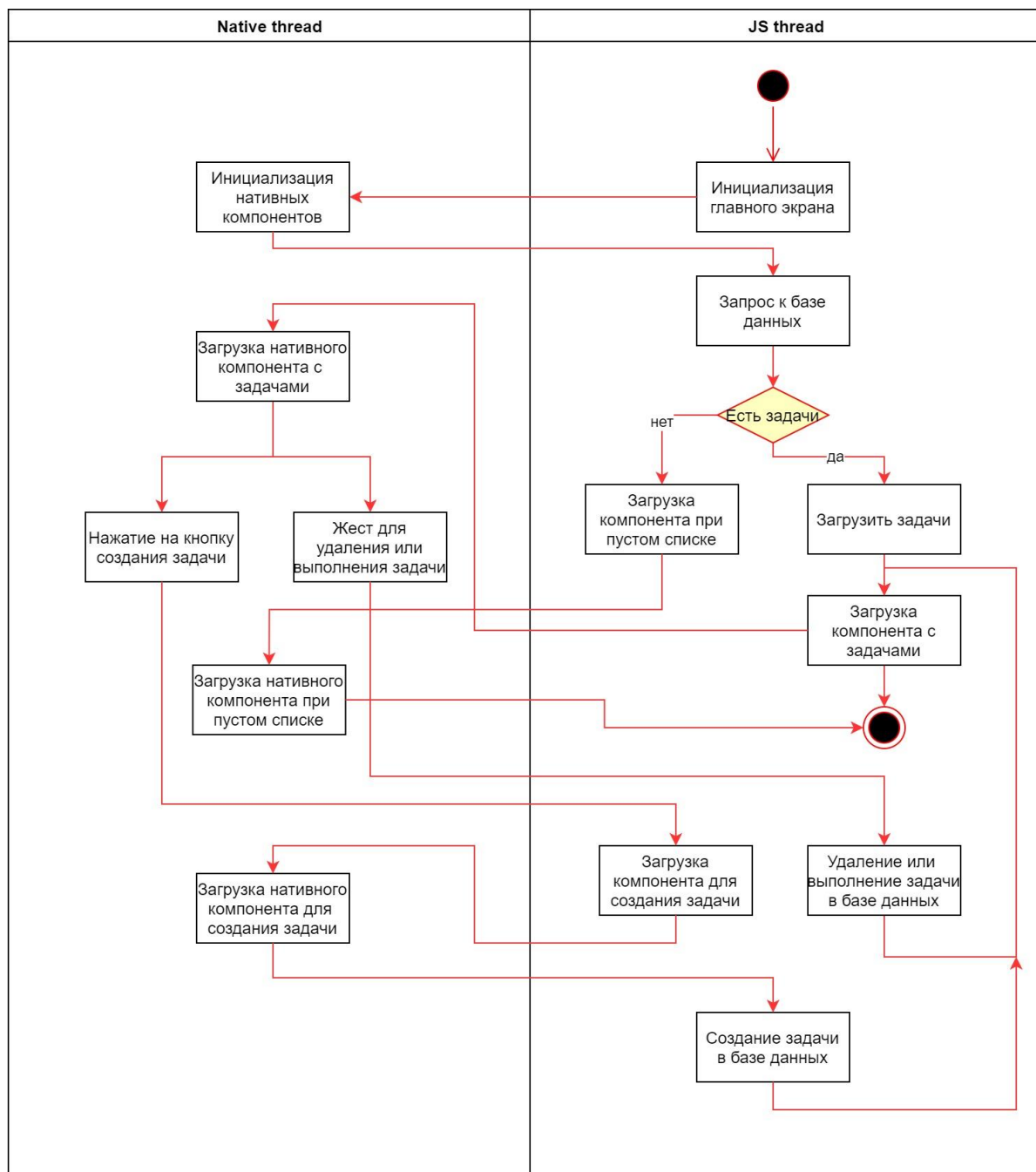


Рисунок 2.5. UML-диаграмма деятельности главного экрана.

Отличительной особенностью реализации интерфейсов экранов приложения является то, что в отличие от остальных компонентов они написаны не как классы, а как функции:

```

export const Main = props => {
  const [resultData, setResultData] = useState();
  const [todayList, setTodayList] = useState();
  const [showAdding, setShowingAdding] = useState();

```

```
const [update, setUpdate] = useState();
```

В таком случае для использования состояний необходимо использовать хук `useState()`, props передаются просто в качестве аргумента в функцию. Для выполнения определенных функций при загрузке и обновлении состояния компонента (главного экрана) необходимо использовать хук `useEffect()`.

Рассмотрим применение данного хука на конкретном примере:

```
useEffect(() => {
    DataBase.queryAllDataBase().then(dataBase => {
        setResultData((Array.from(dataBase)));
        return (dataBase);
    }).then(database => {
        Array.from(database).length == 0 ? DataBase.startNewDataBase() :
null;
    }).then(() => {
        DataBase.queryTodayTasks().then(dataBase => {
            setTodayList((JSON.parse(JSON.stringify(Array.from(dataBase)))));
        })
    });
    Navigation.updateProps('sideMenuId', {
        mainScreenHandler: setUpdate,
    })
}, [showAdding, update])
```

Как мы можем видеть при загрузке компонента происходит запрос на промисах к базе данных на наличие записей и обновляются props бокового меню приложения, в котором отображаются все списки задач.

2.2.3. Реализация интерфейса бокового меню

Диаграмма деятельности представлена на рисунке 2.6:

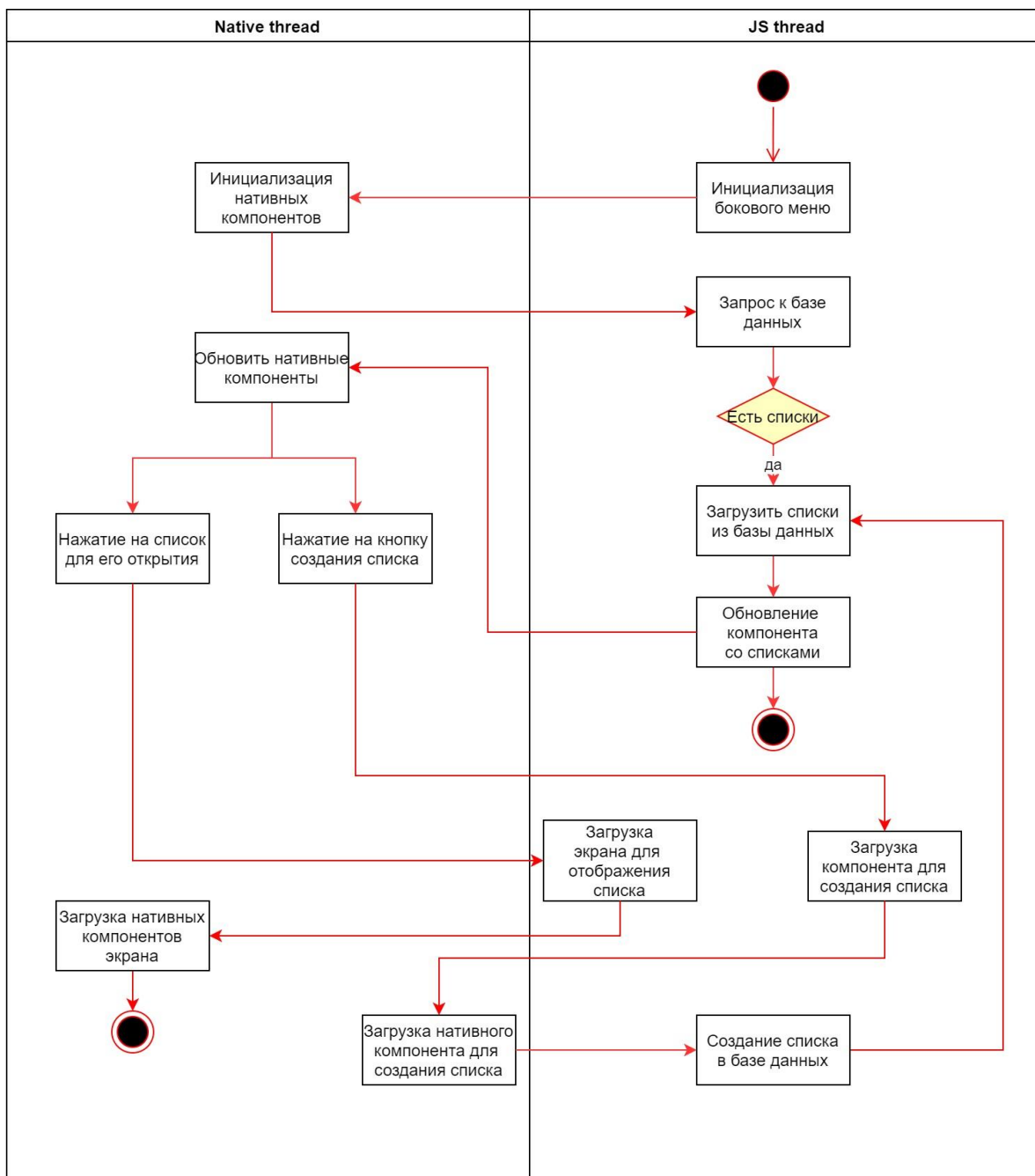


Рисунок 2.6. UML-диаграмма деятельности бокового меню.

В боковом меню аналогично главному экрану используются хуки создания состояний *useState()*, определения функций при инициализации, обновлении состояний экрана *useEffect()*.

Из особенностей необходимо отметить функцию работы со стеком навигации React Native Navigation от Wix. Рассмотрим конкретный пример:

```
function navigateTo(listName, componentName = 'listViewerScreen') {
  Navigation.setStackRoot(props.componentId, {
```

```

sideMenu: {
  left: {
    component: {
      id: 'sideMenuId',
      name: 'sideMenu',
      passProps: {
        mainScreenHandler: props.mainScreenHandler,
        ...
      },
    },
    center: {
      stack: {
        children: [{
          component: {
            name: componentName,
            passProps: {
              listName: listName,
              ...
            }
          }
        ]
      }
    }
  }
}

```

Функция принимает в качестве параметров имя списка (*listName*), который необходимо открыть, и название компонента-экрана, использующегося для отображения этого списка (*listViewerScreen*). Суть работы функции в установке нового стека навигации при нажатии на список.

2.2.4. Реализация интерфейса создания новой задачи.

Диаграмма деятельности представлена на рисунке 2.7:

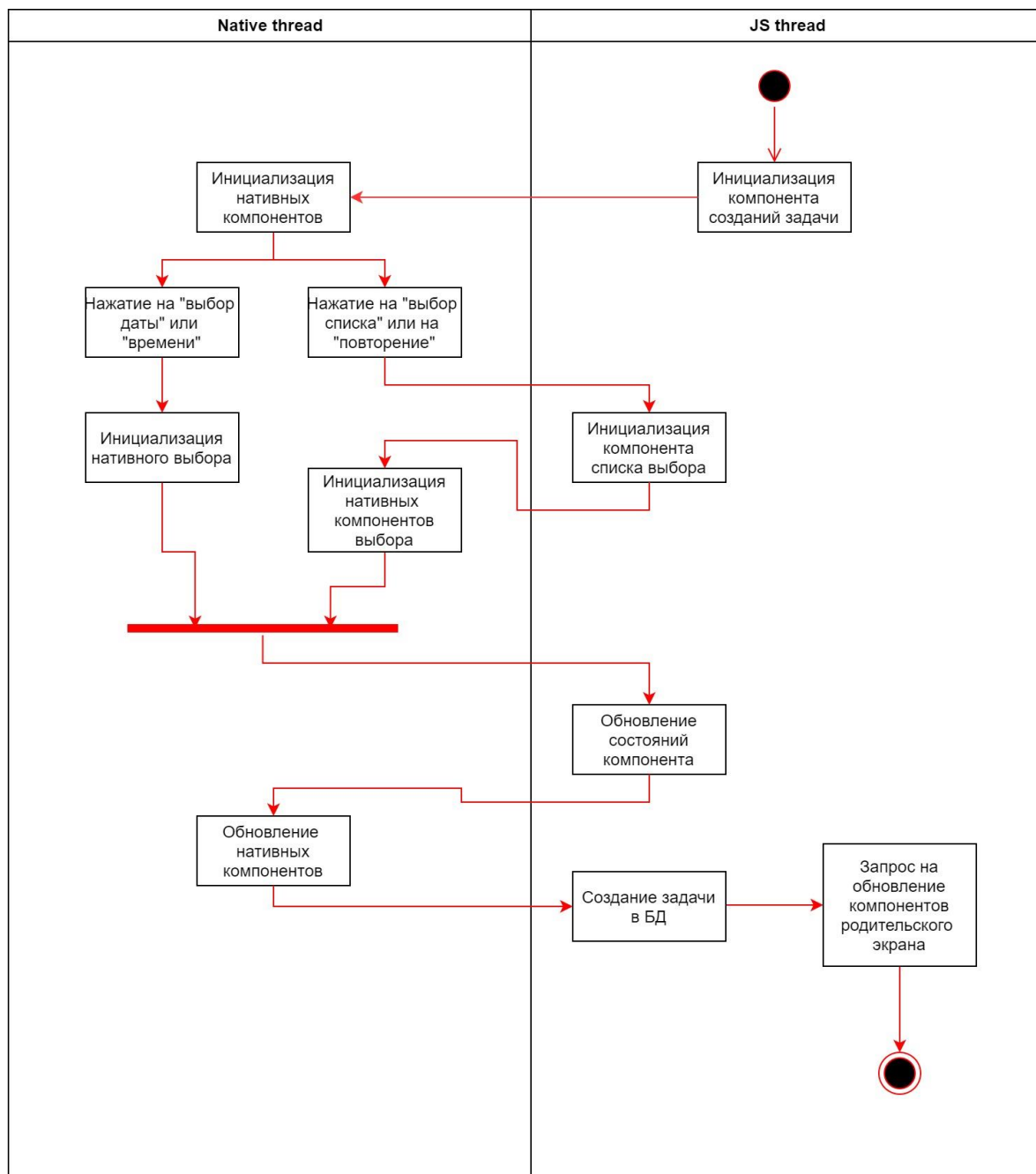


Рисунок 2.7. UML-диаграмма деятельности создания задачи.

Данный компонент вызывается с главного экрана или с любого экрана отображения выбранного списка задач, служит для задания параметров новой задачи и обращения к базе данных для ее добавления.

Стоит отметить, что в отличие компонентов, которые служат в качестве экранов приложения, данный компонент выполнен с использованием класса, унаследованного от PureComponent React Native. Как уже было сказано ранее, PureComponent служит для уменьшения количества обновлений и

соответственно уменьшения обращений к нативным компонентам. Такое поведение приводит к положительным изменениям производительности компонента.

Если при реализации компонентов в качестве функций мы использовали хук для определения поведения при инициализации и обновлении, то в классе это уже не требуется, благодаря встроенным методам:

UNSAFE_componentWillMount() – вызывается сразу же при инициализации компонента;

UNSAFE_componentWillUnmount() – вызывается при закрытии компонента.

Рассмотрим на конкретном примере из ВКР:

```
export class NewTask extends PureComponent {
  constructor(props) {
    super(props);
    this.handleBackButtonClick = this.handleBackButtonClick.bind(this);
    ...
  }
  UNSAFE_componentWillMount() {
    ...
    BackHandler.addEventListener('hardwareBackPress',
this.handleBackButtonClick);
  }
  UNSAFE_componentWillUnmount() {
    BackHandler.removeEventListener('hardwareBackPress',
this.handleBackButtonClick);
  }
  handleBackButtonClick() {
    this.state.parentHandler(false);
    return true;
  }
}
```

Мы используем описанные встроенные методы для обработки нажатия системной кнопки «назад», для этого необходимо связать handler с текущим контекстом при помощи:

```
this.handleBackButtonClick = this.handleBackButtonClick.bind(this);
```

Затем начать «прослушивать» кнопку «назад» после того, как компонент был инициализирован (в качестве аргумента передается указатель на функцию, выполняющуюся при нажатии кнопки):

```
BackHandler.addEventListener('hardwareBackPress',  
this.handleBackButtonClick);
```

Кнопка назад прослушивается не только в данном компоненте приложения, но и в ряде других, по этой причине нам необходимо закрыть EventListener после закрытия компонента:

```
BackHandler.removeEventListener('hardwareBackPress',  
this.handleBackButtonClick);
```

Стоит также отметить, что при использовании класса в качестве реализации компонента требуется помещать «возврат» интерфейса в специальный метод render():

```
render() {  
  return (this.props.show ?  
    <View style={styles.wrapper}>  
      <View style={styles.base}>  
        ...  
      </View>  
    </View> : null  
  )  
}
```

2.3. Реализация анимации

Сложно себе представить современное мобильное приложение без использования даже примитивных анимаций. React Native, конечно, же позволяет реализовывать анимации различной сложности, для этого он использует специальный Animated API, представляющий набор методов, объектов для применения анимации к базовым компонентам, таким как View, Text, Image, ScrollView, FlatList и SectionList.

При этом анимации гибко настраиваются пользовательскими и встроенными функциями замедления, задержки, затухания и т.д. Например, объект Animated предоставляет несколько типов анимаций, наиболее часто используемым среди разработчиков является *Animated.timing()*. Если вы хотите создать двухсекундную анимацию плавного перемещения объекта на позицию, то реализация будет выглядеть следующим образом:

```
Animated.timing(this.state.xPosition, {  
  toValue: 100,  
  easing: Easing.back(),  
  duration: 2000  
}).start();
```

В данном примере:

- toValue отвечает за значение к которому приходит объект при перемещении;
- easing предоставляет возможность использовать различные механизмы выполнения анимации по кривой во времени;
- duration указывает продолжительность времени анимации.

Анимации отлично дополняют обработку пользовательских жестов, делают приложение более интуитивно понятным для человека. Для обработки жестов в данной ВКР была использована библиотека React Native Gesture Handler. Ее использование позволяет нам существенно упростить процесс обработки жестов и избежать при этом потери производительности, ведь

разработчики сразу позаботились о том, чтобы минимизировать сообщения между нативным и JavaScript потоком.

React Native Gesture Handler также, как и React Native, использует декларативный подход и предоставляет нам набор компонентов для отслеживания жестов, к ним относятся следующие компоненты: Buttons, Swipeable, Touchables, DrawerLayout. В рамках ВКР используется Swipeable, рассмотрим конкретный пример (компонент отображения задачи в списке listItem):

```
...
render() {
  return (
    <View style={styles.base}>
      <Swipeable
        ref='SwipeableTask'
        containerStyle={{borderRadius: 4, elevation: 12, },
this.props.isCompleted == true ? completedTask.body : null}}
        renderLeftActions={CompleteTaskBody}
        renderRightActions={DeleteTaskBody}
        onSwipeableRightOpen={() => this.deleteTask()}
        onSwipeableLeftOpen={() => this.completeTask()}
      >
        <TouchableWithoutFeedback          onPress={()          =>
this.showTaskSettings()}>
          <View style={[styles.body, this.props.isCompleted == true ?
completedTask.body : null]}>
            <View>
              <Text style={[styles.taskTitle, this.props.isCompleted ==
true ? completedTask.taskTitle : null]}>
                {this.props.taskTitle}
              </Text>
            </View>
          </View>
        </TouchableWithoutFeedback>
      </Swipeable>
    </View>
  )
}
```

```

        <View style={styles.options}>
            {this.props.listName ?
                <View style={styles.optionsWrapper}>
                    <Text
style={styles.optionsText}>{this.props.listName}</Text>
                </View>
                : null}
            ...

```

Как можем видеть, компонент обворачивается в `Swipeable` и тем самым у нас появляется возможность отслеживать жесты «свайпа» в правую и левую стороны. Для дальнейшей работы устанавливаются следующие атрибуты:

`renderLeftActions` и `renderRightActions` — здесь устанавливаются компоненты, которые будут отображаться при сдвиге задачи в соответствующую сторону;

`onSwipeableRightOpen` и `onSwipeableLeftOpen` необходимы для передачи указателя функции, которые будут выполняться при полном сдвиге задачи в соответствующую сторону.

2.4. Работа с базой данных

Как уже было сказано ранее в разделе выбора стека технологий, в данной работе используется no-sql база данных `Realm`. `Realm` предлагает внушительный API для работы, тем не менее для комфортного использования в процессе разработки возникла необходимость применения паттерна «фасад», позволяющего получить еще более простой и удобный способ работы с БД.

`Realm` является объектно-ориентированной базой данных, поэтому для начала нужно определить структуру объектов, она представлена на рисунке 2.8:

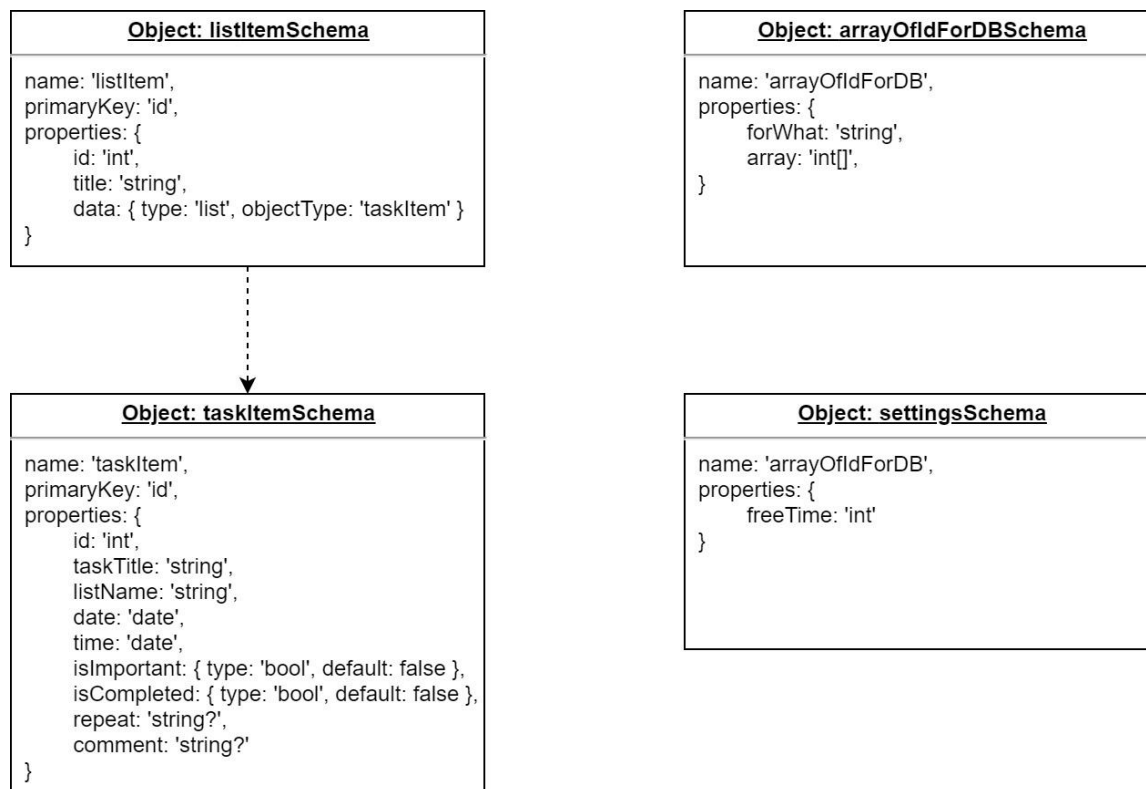


Рисунок 2.8. Диаграмма объектов БД.

Данные объекты используются для наполнения базы данных, рассмотрим поля listItemSchema:

- name указывает имя, которые потом используется в качестве типа в других объектах БД;
- primaryKey указывает на ключ для удобной работы с БД;
- properties.id – уникальный идентификатор списка;
- properties.title – наименование списка;
- properties.data – хранит ссылки на вложенный объекты, сами задачи.

Перейдем к обзору taskItemSchema, который служит для определения объекта задачи в БД:

- properties.id – уникальный идентификатор;
- properties.taskTitle – наименование задачи;
- properties.listName – наименование списка, к которому задача закреплена;
- properties.date – дата исполнения задачи;

- properties.time – время необходимое для выполнения задачи, на основе этого поля рассчитывается свободное на день время;
- properties.isImportant – является ли задача важной;
- properties.isCompleted – является ли задача выполненной;
- properties.repeat – необходимо ли повторять задачу и с каким интервалом;
- properties.comment – содержит текст заметки к каждой задаче.

Стоит также упомянуть и arrayOfIdForDBSchema, необходимый для корректного создания и удаления задач:

- properties.forWhat указывает для какого именно типа объекта стоит хранить массив ключей;
- properties.array хранит сам массив идентификаторов.

Не стоит и забывать о settingsSchema, служащей для хранения настроек приложения, в рамках ВКР настраивается только базовое количество свободного времени.

Теперь посмотрим представленную на рисунке 2.9 UML-диаграмму получившегося класса-фасада для упрощения работы с базой данных. Сразу же стоит уточнить, что поле Realm является singleton объектом, предоставляющий доступ к базе данных.

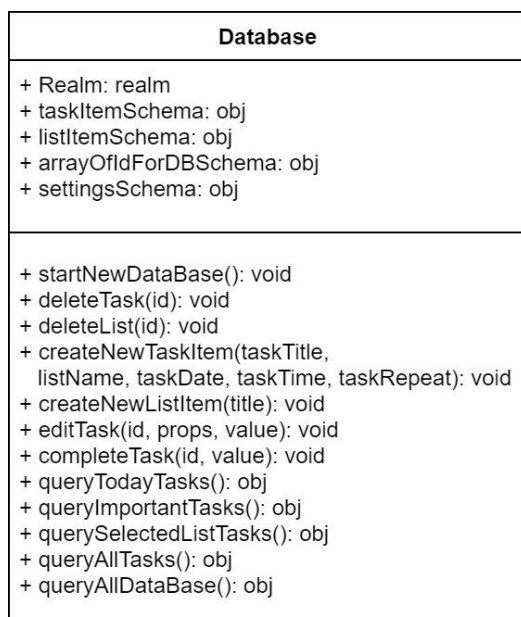


Рисунок 2.9. UML-диаграмма класса для работы с Realm DB.

Методы класса имеют говорящие названия, с легкостью дающие понять, что именно предоставляет метод. Особенность работы с Realm DB также в том, что каждый метод работает, используя механизм JavaScript промисов. Конкретный пример из ВКР:

```
export const EditTask = (id, props, value) => new Promise((resolve, reject) =>
{
    Realm.open({
        schema: [taskItemSchema, listItemSchema, arrayOfIdForDBSchema],
        deleteRealmIfMigrationNeeded: true
    }).then(realm => {
        realm.write(() => {
            realm.create('taskItem', { id: id, [props]: value }, 'modified');
            resolve()
        });
    }).catch((error) => reject(error));
})
```

2.5. Реализация серверной стороны

Для решения серверной стороны приложения мы используем готовый набор инструментов Firebase от Google, позволяющий нам с легкостью использовать механизмы аутентификация пользователя через email, пароль и большинство социальных сетей. В рамках выпускной квалификационной работы мы воспользуемся только входом при помощи Google аккаунта, который имеется у всех Android пользователей, что очень удобно.

Для начала работы с Firebase необходимо установить официально рекомендуемую библиотеку для работы с ним React Native Firebase. Процесс установки не так прост, как в случае с уже ранее используемыми библиотеками, для начала выполняем установки через npm:

```
$ npm install --save @react-native-firebase/app
```

После необходимо зарегистрировать приложение в консоли firebase и сгенерировать AndroidManifest.xml файл для проекта и «связать» его. Производить редактирование нативных android файлов лучше стоит при помощи Android Studio, аналогично нужно поступать и при работе с iOS – воспользоваться Xcode.

Добавляем плагин google сервисов в зависимости /android/build.gradle файла:

```
buildscript {  
  dependencies {  
    // ... other dependencies  
    classpath 'com.google.gms:google-services:4.2.0'  
    // Add me --- ^  
  }  
}
```

Также добавляем плагин и в /android/app/build.gradle:

```
apply plugin: 'com.google.gms.google-services'
```

После установки всех зависимостей мы готовы использовать Firebase в своем проекте. Для использования аутентификации включаем ее в консоли Firebase, как это сделать подробно описано в официальной документации к Firebase и React Native Firebase.

Для вызова нативного компонента выбора аккаунта по нажатию на кнопку входа при помощи Google мы используем компонент GoogleSignIn:

```
GoogleSignin.configure({  
  webClientId: 'secretID',  
});  
  
export class GoogleLogin extends PureComponent {  
  async onGoogleButtonPress() {  
    // Получение пользовательского ID токена  
    const { idToken } = await GoogleSignin.signIn();  
    // Создание при помощи токена учетных данных Google
```

```

const googleCredential = auth.GoogleAuthProvider.credential(idToken);
// Вход с использованием этих учетных данных
return auth().signInWithCredential(googleCredential);
}
render() {
  return (
    <TouchableNativeFeedback onPress={() => this.onGoogleButtonPress()}>
      ...
    </TouchableNativeFeedback>
  );
}
}

```

2.6. Тестирование приложения

В результате тестирования полученного приложения мы должны убедиться в его корректной работе, соответствии между ожидаемым и реальным поведением.

2.6.1. Тестирование аутентификации

При начальном запуске приложения нас встречает экран аутентификации, для проверки его корректной работы необходимо нажать кнопку входа, выбрать соответствующий аккаунт. На рисунке 2.10 отображен процесс выбора аккаунта:

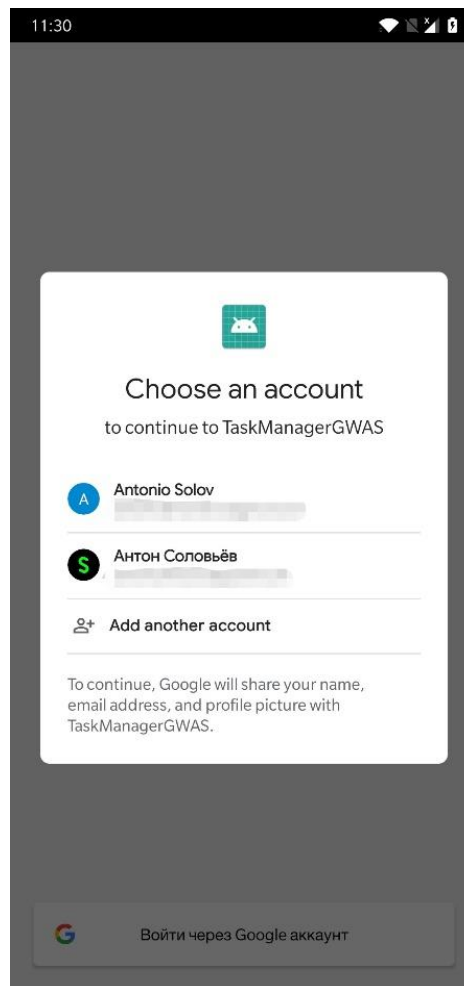


Рисунок 2.10. Тестирование: выбор аккаунта.

После происходит успешный вход в приложение, и отображается главный экран, а в консоли Firebase появляется запись о входе. Демонстрация Firebase консоли представлена на рисунке 2.11.

Search by email address, phone number or user UID					Add user	↺	⋮
Identifier	Providers	Created	Signed In	User UID ↑			
antonio****ov@gmail.com		27 May 2020	27 May 2020	9a*****51UHqPgYn7Z2			
*****@gmail.com		21 Apr 2020	1 May 2020	a21o*****ktcMYOIPASOscKI73			
Rows per page: 50					1-2 of 2	⏪	⏩

Рисунок 2.11. Тестирование: firebase консоль.

2.6.2. Тестирование создания задачи

Перейдем к созданию задач, программа должна учитывать введенные свойства при создании задачи и корректно отображать их в существующих списках. Процесс ввода задачи с определенными свойствами отображен на рисунке 2.12:



Рисунок 2.12. Тестирование: создание новой задачи.

На рисунке 2.13 мы можем наблюдать, что задача была создана в соответствии с заданными свойствами и отображается в необходимом списке. Также можно заметить, что отображение задачи учитывается и в расчете свободного на день времени.

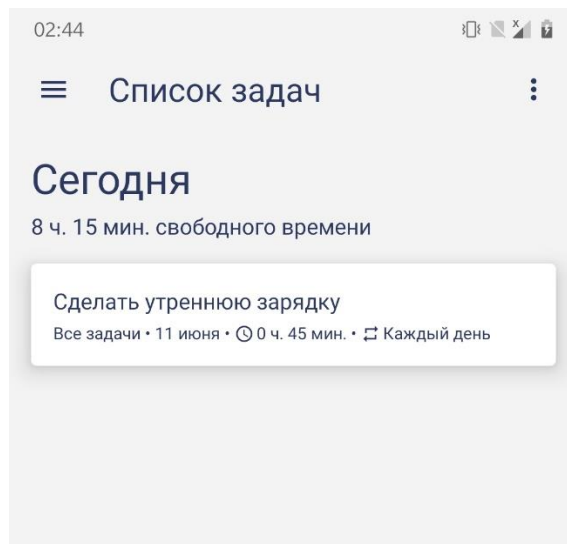


Рисунок 2.13. Тестирование: корректное отображение новой задачи.

2.6.3. Тестирование редактирования задачи

Редактирование уже созданных задач является основным функционалом приложения, существенно упрощая работу для пользователя. По этой причине крайне необходимо, чтобы данный функционал работал корректным образом. Отображение экрана свойств созданной задачи находится на рисунке 2.14:

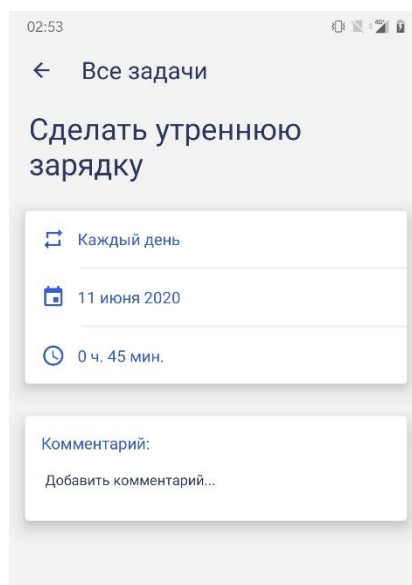


Рисунок 2.14. Тестирование: экран свойств задачи.

Изменим свойство времени исполнения с сегодняшней даты на следующий день и убедимся в том, что задача пропадет из списка на главном экране. Посмотрев на рисунок 2.15, можем убедиться в правильности работы данного

функционала, задача действительно пропала и появится только при достижении следующего дня.

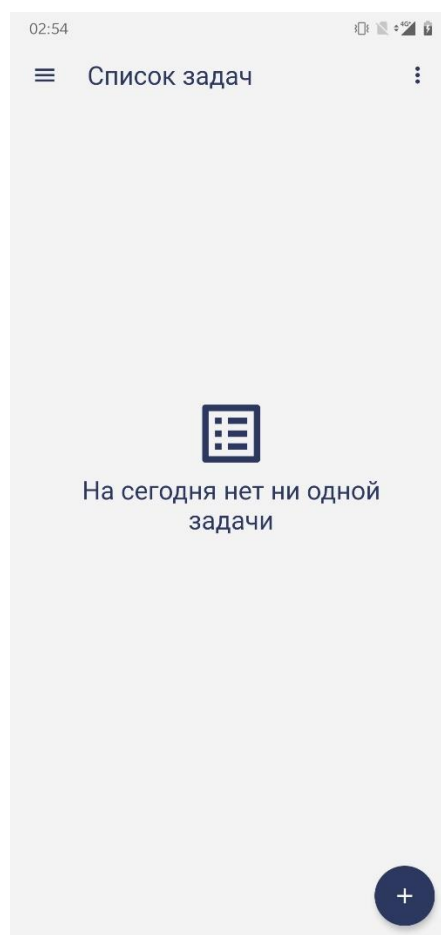


Рисунок 2.15. Тестирование: пустой главный экран.

2.6.4. Тестирование взаимодействия потоков

Как уже было ранее неоднократно сказано, взаимодействие между нативным и JavaScript потоками через bridge является узким местом в работе приложения и может вызвать нежелательную потерю производительности интерфейса.

По этой причине необходимо следить за количеством обновлений компонентов. Для слежения будем использовать простой вывод сообщения с именем компонента в консоль при каждом его обновлении. В идеальном случае каждое взаимодействие пользователя с интерфейсом должно приводить только

к одному обновлению каждого компонента. На рисунке 2.16 отображена консоль, в которой мы можем наблюдать выходы наших сообщений.

```

node "C:\Graduate Work\TaskManagerGWAS\node_modules\react-native\scripts\cli.js" start
[Thu Jun 11 2020 02:41:57.765] LOG Main Screen updated
[Thu Jun 11 2020 02:41:57.852] LOG SideMenu updated
[Thu Jun 11 2020 02:41:57.869] LOG Main Screen updated
[Thu Jun 11 2020 02:41:57.871] LOG Main Screen updated
[Thu Jun 11 2020 02:42:11.110] LOG 0 ч. 45 мин.
[Thu Jun 11 2020 02:42:13.280] LOG Running "listPicker" with {"initialProps":{"componentId":"Component22"},"rootTag":41}
[Thu Jun 11 2020 02:42:13.990] LOG
[Thu Jun 11 2020 02:42:28.895] LOG 2020-06-10T21:45:04.769Z
[Thu Jun 11 2020 02:42:28.902] LOG 2020-06-10T23:40:05.964Z
[Thu Jun 11 2020 02:42:28.904] LOG Сделать утреннюю зарядку
[Thu Jun 11 2020 02:42:28.905] LOG Main Screen updated
[Thu Jun 11 2020 02:42:28.919] LOG Все задачи
[Thu Jun 11 2020 02:42:28.919] LOG 8
[Thu Jun 11 2020 02:42:28.920] LOG {}
[Thu Jun 11 2020 02:42:28.940] LOG SideMenu updated
[Thu Jun 11 2020 02:42:28.942] LOG Main Screen updated
[Thu Jun 11 2020 02:42:28.950] LOG Main Screen updated
[Thu Jun 11 2020 02:43:15.426] LOG Main Screen updated
[Thu Jun 11 2020 02:43:15.449] LOG SideMenu updated
[Thu Jun 11 2020 02:43:15.467] LOG Main Screen updated
[Thu Jun 11 2020 02:43:15.472] LOG Main Screen updated
[Thu Jun 11 2020 02:53:43.926] LOG Running "taskViewerScreen" with {"initialProps":{"componentId":"Component25"},"rootTag":51}
[Thu Jun 11 2020 02:53:43.929] LOG
[Thu Jun 11 2020 02:54:23.637] LOG Task Viewer updated
[Thu Jun 11 2020 02:54:23.681] LOG Main Screen updated
[Thu Jun 11 2020 02:54:23.681] LOG SideMenu updated
[Thu Jun 11 2020 02:54:23.688] LOG Main Screen updated
[Thu Jun 11 2020 02:54:23.700] LOG Main Screen updated
  
```

Рисунок 2.16. Тестирование: консоль работы приложения.

Как можно заметить существенных проблем с обновлением при работе не возникает, производительность приложения не падала.

3. Руководство пользователя

Приложение состоит из нескольких экранов, в руководстве пользователей описаны их функционал, рекомендуемые варианты использования.

3.1. Авторизация

Экран авторизации приложения встречает пользователя при первом запуске приложения, служит для входа или регистрации пользователя. Его графическое представление находится на рисунке: 3.1:

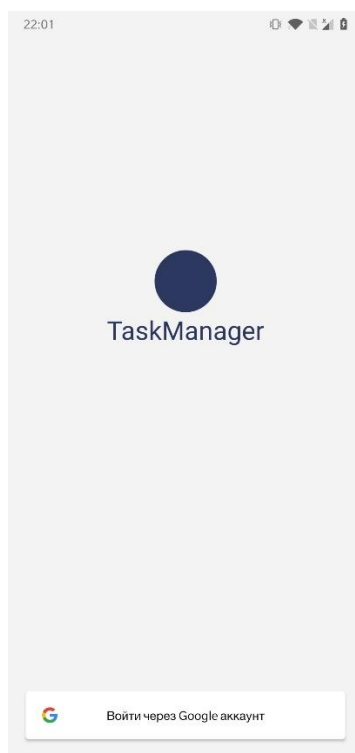


Рисунок 3.1. Экран авторизации.

В случае если пользователь производил вход в приложение ранее, его встречает главный экран.

3.2. Главный экран.

Главный экран отображает список задач на день, количество свободного времени на день с учетом задач. Изображен на рисунке 3.2.

					ПГУ 2.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		47



Рисунок 3.2. Главный экран со списком задач на день.

Позволяет перейти с него к редактированию существующих задач или создать новую. Также стоит отметить, что по нажатию на кнопку меню в верхнем левом углу, открывается боковое меню со списком задач. Вариант главного экрана в случае отсутствия задач на день представлен на рисунке 3.3.

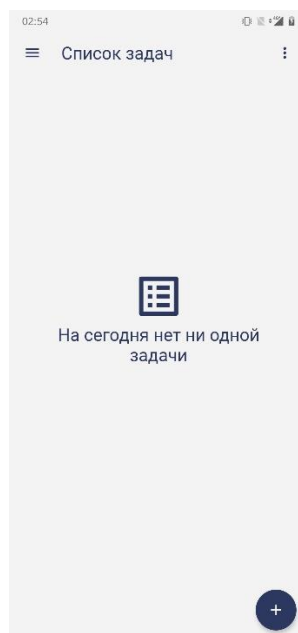


Рисунок 3.3. Главный экран с пустым списком задач.

3.3. Работа с задачами

Вы можете создать задачи с любого экрана их отображения нажав на кнопку в правом нижнем углу. После нажатия откроется компонент создания задачи, представленный на рисунке 3.4.

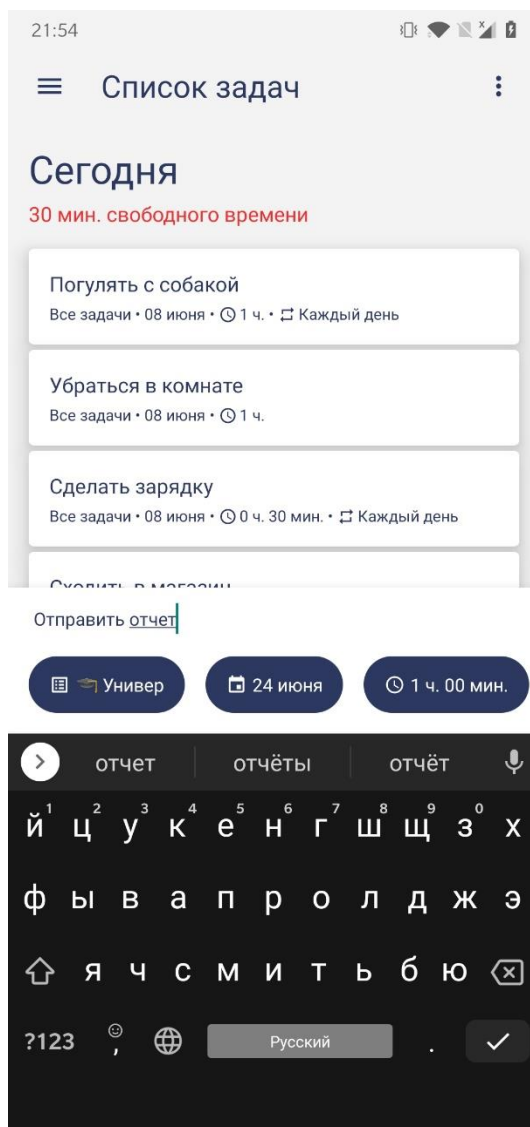


Рисунок 3.4. Создание новой задачи.

После создания задач, они распределяются по спискам в зависимости от назначенных им свойств. По жесту «свайпа» в правую или левую сторону выполняется выполнение и удаление задачи соответственно. При нажатии на задачу откроется экран ее редактирования, который изображен на рисунке 3.5. По долгому нажатию на задачу она добавляется в список важных задач.

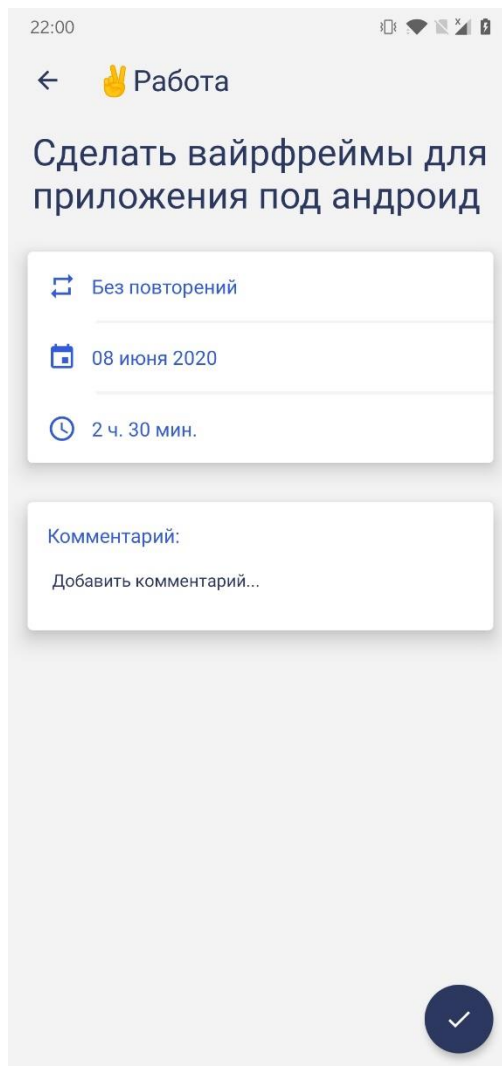


Рисунок 3.5. Экран редактирования задачи.

3.4. Особенности работы со списками

В приложении есть три базовых списка:

- «все задачи» — который отображает все созданные задачи вне зависимости от их свойств;
- «сегодня», отображающий задачи, назначенные на текущий день;
- «важные» — в этот список попадают задачи, которые пользователь отметил важными для себя, в других списках эти задачи попадают на самый верх отображения.

Все три базовых списка отображаются в боковом меню приложения, которое изображено на рисунке 3.6.

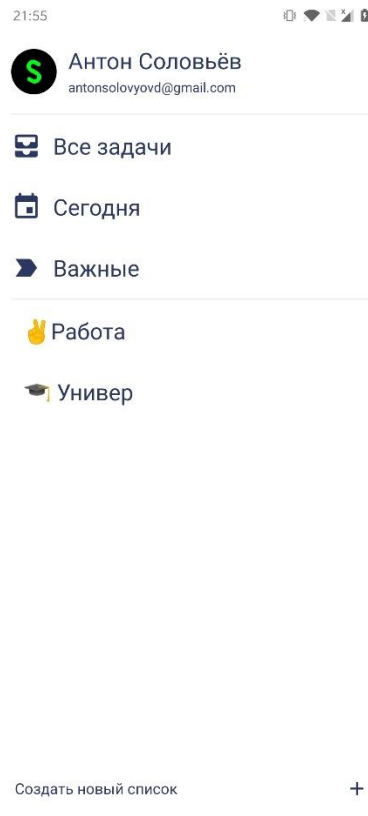


Рисунок 3.6. Боковое меню со списками задач.

Также пользователь имеет возможность создавать собственные списки задач. Для этого необходимо внизу меню нажать на кнопку создания списка. Откроется создание, как на рисунке 3.7.

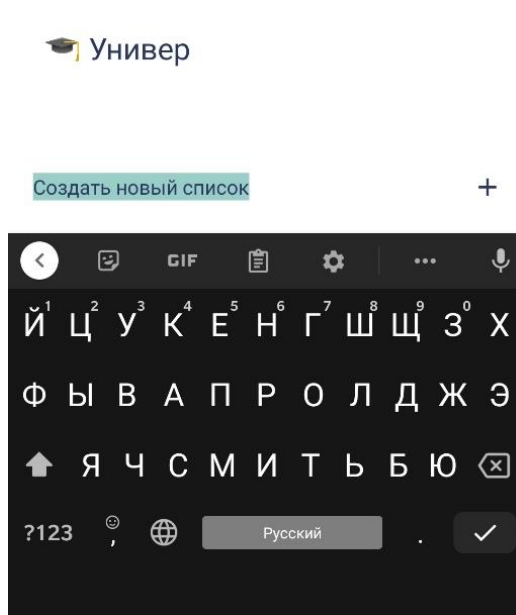


Рисунок 3.7. Создание нового списка.

3.5. Доступные настройки

Получить доступ к экрану настроек приложения можно получить из бокового меню нажатием на аккаунт пользователя. Настройки представлены на рисунке 3.8. В рамках ВКР настраивается только базовое количество свободного времени на день. С этого же экрана можно произвести выход из аккаунта.

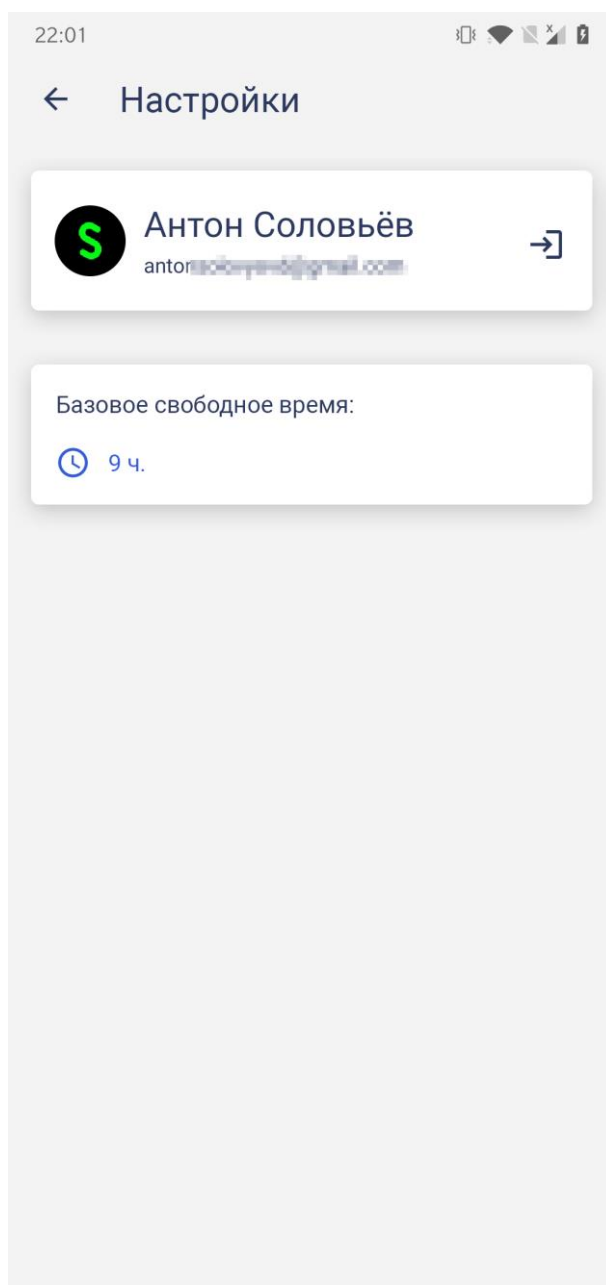


Рисунок 3.8. Экран настроек.

4. Безопасность жизнедеятельности

Современные мобильные устройства являются вполне безопасными, проходят всевозможные сертификации соответствия локальным и международным стандартам производства такого типа техники. Тем не менее все же имеют ряд нежелательных факторов влияния на организм человека.

4.1. Влияние мобильных устройств на организм

Среди факторов воздействия мобильных устройств можно отметить следующие виды:

- психофизиологические факторы (устомление, монотонность работы, статические перегрузки и др.);
- электромагнитное излучение, исходящее от техники;
- опасность поражения электричеством.

С каждым годом устройства становятся не только все более производительными, но и безопасными. Соответственно воздействие вышеописанных факторов на человека снижается. Но все же важно соблюдать определенные меры предосторожности.

4.2. Рекомендуемые меры предосторожности

Для снижения влияния психофизиологических факторов следует:

- избегать устройств с высоким показателем ШИМ у дисплея;
- делать перерывы хотя бы каждые 20-30 минут работы;
- следить за временем использования устройства в течение дня;
- занимать удобную позу при работе.

Также не стоит покупать в свое пользование мобильные устройства, не прошедшие европейские и отечественные сертификации. Это поможет избежать негативного влияния электромагнитного излучения, снизить опасность поражения электричеством при неисправной работе устройства.

Для еще большего снижения влияния излучения следует:

- снизить чрезмерно длительное общение по мобильному телефону, максимум 15 мин;
- не вести интенсивные переговоры находясь в транспорте;
- не разговаривать в зоне неустойчивого приема;
- не класть мобильный телефон рядом с местом сна.

					ПГУ 2.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		54

Заключение

В ходе выполнения выпускной квалификационной работы была проанализирована предметная область, исследованы ключевые аналоги по теме работы, изучены современные методы разработки программного обеспечения для мобильных устройств, разработано мобильное приложение под операционную систему Android «Смарт-органайзер» с использованием кроссплатформенного подхода и применением React Native. Полученный продукт, предназначен для эффективного планирования личного времени, формирования привычек. Программа прошла экспериментальную проверку на настоящем физическом устройстве. Применение разработанного приложения позволяет действительно бороться с негативными последствиями прокрастинации, способствует повышению продуктивности в течение дня.

В будущем планируется дополнить существующий функционал приложения, доработать релиз под другие платформы, такие как iOS и web. Также следует повысить стабильность и производительность работы.

Список использованных источников

1. Крупнейшее мировое сообщество разработчиков [Электронный ресурс]: <https://stackoverflow.com/>
2. Официальная документация по React Native [Электронный ресурс]: <https://reactnative.dev/>
3. Официальная документация по работе с Realm [Электронный ресурс]: <https://realm.io/docs>
4. Официальная документация по использованию Firebase в своем проекте [Электронный ресурс]: <https://firebase.google.com/docs>
5. Официальная документация по использованию React Native Navigation от Wix [Электронный ресурс]: <https://wix.github.io/react-native-navigation/docs/before-you-start/>
6. Официальная документация по использованию React Native Gesture Handler [Электронный ресурс]: <https://docs.swmansion.com/react-native-gesture-handler/>
7. Официальная документация по использованию библиотеки day.js [Электронный ресурс]: <https://day.js.org/docs/ru/installation/installation>

Приложение А. Листинги мобильного приложения

Приложение А.1. Components

Файл database.js:

```
const Realm = require('realm');

export const taskItemSchema = {
  name: 'taskItem',
  primaryKey: 'id',
  properties: {
    id: 'int',
    taskTitle: 'string',
    listName: 'string',
    date: 'date',
    time: 'date',
    isImportant: { type: 'bool', default: false },
    isCompleted: { type: 'bool', default: false },
    repeat: 'string?',
    comment: 'string?'
  }
}

export const listItemSchema = {
  name: 'listItem',
  primaryKey: 'id',
  properties: {
    id: 'int',
    title: 'string',
    data: { type: 'list', objectType: 'taskItem' }
  }
}

export const arrayOfIdForDBSchema = {
```

```

name: 'arrayOfIdForDB',
properties: {
  forWhat: 'string',
  array: 'int[]',
}
}

```

```

export const startNewDataBase = () => new Promise((resolve, reject) => {
  Realm.open({
    schema: [taskItemSchema, listItemSchema, arrayOfIdForDBSchema],
    deleteRealmIfMigrationNeeded: true
  }).then(realm => {
    realm.write(() => {
      if (realm.objects('arrayOfIdForDB').length == 0) {
        realm.create('arrayOfIdForDB', { forWhat: 'forLists', array: [] })
        realm.create('arrayOfIdForDB', { forWhat: 'forTasks', array: [] })
        console.log(realm.objects('arrayOfIdForDB'));
      };
      resolve()
    });
  }).catch((error) => reject(error));
})

```

```

export const deleteTask = (id) => new Promise((resolve, reject) => {
  Realm.open({
    schema: [taskItemSchema, listItemSchema, arrayOfIdForDBSchema],
    deleteRealmIfMigrationNeeded: true
  }).then(realm => {
    realm.write(() => {
      let arrayOfId = realm.objects('arrayOfIdForDB');

```

```

        arrayOfId = arrayOfId.filtered('forWhat = "forTasks"');
        arrayOfId[0].array.push(id);
        console.log(arrayOfId[0].array);
        console.log(realm.objectForPrimaryKey('taskItem', id));
        realm.delete(realm.objectForPrimaryKey('taskItem', id));
    })
}).catch((error) => reject(error));
})

export const deleteList = id => new Promise((resolve, reject) => {
    Realm.open({
        schema: [taskItemSchema, listItemSchema, arrayOfIdForDBSchema],
        deleteRealmIfMigrationNeeded: true
    }).then(realm => {
        realm.write(() => {
            console.log(id);
            let arrayOfId = realm.objects('arrayOfIdForDB');
            let actualList = realm.objectForPrimaryKey('listItem', id);
            arrayOfId = arrayOfId.filtered('forWhat = "forLists"');
            arrayOfId[0].array.push(id);
            console.log(arrayOfId[0].array);
            if (actualList.data.length !== 0) {
                for (let i = 0; i < actualList.data.length; i++) {
                    console.log(actualList.data[i].id);
                    deleteTask(actualList.data[i].id);
                }
            }
            console.log(actualList.data.length);
            realm.delete(realm.objectForPrimaryKey('listItem', id));
            resolve();
        });
    });
});

```

```

    })
  }).catch((error) => reject(error));
})

```

```

export const createNewTaskItem = (taskTitle, listName, taskDate, taskTime,
taskRepeat = 'default') => new Promise((resolve, reject) => {
  Realm.open({
    schema: [taskItemSchema, listItemSchema, arrayOfIdForDBSchema],
    deleteRealmIfMigrationNeeded: true
  }).then(realm => {
    realm.write(() => {
      let freeId;
      let arrayOfId = realm.objects('arrayOfIdForDB');
      arrayOfId = arrayOfId.filtered('forWhat = "forTasks"');
      if (arrayOfId[0].array.length == 0) {
        freeId = realm.objects('taskItem').length;
      }
      else { freeId = arrayOfId[0].array.pop(); }
      if (listName == 'Все задачи') {
        console.log(listName);
        realm.create('taskItem', {
          id: freeId,
          taskTitle: taskTitle,
          listName: listName,
          date: taskDate,
          time: taskTime,
          repeat: taskRepeat,
        })
      }
      else {

```

```

    let actualList = realm.objects('listItem');
    actualList = actualList.filtered('title = "' + listName + '"');
    actualList[0].data.push({
      id: freeId,
      taskTitle: taskTitle,
      listName: listName,
      date: taskDate,
      time: taskTime,
      repeat: taskRepeat,
    });
    console.log(actualList[0]);
  }
  console.log(freeId);
  console.log(arrayOfId[0].array);
  resolve()
})
}).catch((error) => reject(error));
})

export const createNewListItem = title => new Promise((resolve, reject) => {
  Realm.open({
    schema: [taskItemSchema, listItemSchema, arrayOfIdForDBSchema],
    deleteRealmIfMigrationNeeded: true
  }).then(realm => {
    realm.write(() => {
      let freeId;
      let arrayOfId = realm.objects('arrayOfIdForDB');
      arrayOfId = arrayOfId.filtered('forWhat = "forLists"');
      if (arrayOfId[0].array.length == 0) {
        freeId = realm.objects('listItem').length;

```

```

    }
    else { freeId = arrayOfId[0].array.pop(); }
    realm.create('listItem', { id: freeId, title: title, data: [] });
    console.log(realm.objects('listItem'));
    resolve()
  });
}).catch((error) => reject(error));
})

export const EditTask = (id, props, value) => new Promise((resolve, reject) =>
{
  Realm.open({
    schema: [taskItemSchema, listItemSchema, arrayOfIdForDBSchema],
    deleteRealmIfMigrationNeeded: true
  }).then(realm => {
    realm.write(() => {
      realm.create('taskItem', { id: id, [props]: value }, 'modified');
      resolve()
    });
  }).catch((error) => reject(error));
})

export const completeTask = (id, value = true) => new Promise((resolve, reject)
=> {
  Realm.open({
    schema: [taskItemSchema, listItemSchema, arrayOfIdForDBSchema],
    deleteRealmIfMigrationNeeded: true
  }).then(realm => {
    realm.write(() => {
      realm.create('taskItem', { id: id, isCompleted: value }, 'modified');

```

```

        resolve()
    });
}).catch((error) => reject(error));
})

export const queryTodayTasks = () => new Promise((resolve, reject) => {
    Realm.open({
        schema: [taskItemSchema, listItemSchema, arrayOfIdForDBSchema],
        deleteRealmIfMigrationNeeded: true
    }).then(realm => {
        var startDate = new Date();
        startDate.setHours(0, 0, 0, 0);
        var endDate = new Date();
        endDate.setHours(23, 59, 59, 999);

        let dataBase = realm.objects('taskItem');
        dataBase = dataBase.filtered('date >= $0 && date <= $1', startDate,
endDate)

        dataBase = dataBase.sorted('isCompleted');
        resolve(dataBase);
    }).catch((error) => reject(error));
})

```

```

export const queryImportantTasks = () => new Promise((resolve, reject) => {
    Realm.open({
        schema: [taskItemSchema, listItemSchema, arrayOfIdForDBSchema],
        deleteRealmIfMigrationNeeded: true
    }).then(realm => {
        let dataBase = realm.objects('taskItem');
        dataBase = dataBase.filtered('isImportant == true');
    });
});

```

```

        dataBase = dataBase.sorted('isCompleted');
        resolve(dataBase);
    }).catch((error) => reject(error));
})

```

```

export const querySelectedListTasks = listName => new Promise((resolve,
reject) => {
    Realm.open({
        schema: [taskItemSchema, listItemSchema, arrayOfIdForDBSchema],
        deleteRealmIfMigrationNeeded: true
    }).then(realm => {
        let dataBase = realm.objects('taskItem');
        dataBase = dataBase.filtered('listName == $0', listName);
        dataBase = dataBase.sorted('isCompleted');
        resolve(dataBase);
    }).catch((error) => reject(error));
})

```

```

export const queryAllTasks = () => new Promise((resolve, reject) => {
    Realm.open({
        schema: [taskItemSchema, listItemSchema, arrayOfIdForDBSchema],
        deleteRealmIfMigrationNeeded: true
    }).then(realm => {
        let dataBase = realm.objects('taskItem');
        dataBase = dataBase.sorted('isCompleted');
        resolve(dataBase);
    }).catch((error) => reject(error));
})

```

```

export const queryAllDataBase = () => new Promise((resolve, reject) => {

```

```

Realm.open({
  schema: [taskItemSchema, listItemSchema, arrayOfIdForDBSchema],
  deleteRealmIfMigrationNeeded: true
}).then(realm => {
  /* realm.write(()=> {
    realm.delete(realm.objectForPrimaryKey('listItem', 0));
  })
  console.log(realm.objectForPrimaryKey('listItem', 0)); */
  let dataBase = realm.objects('listItem');
  resolve(dataBase);
}).catch((error) => reject(error));
})

```

Файл editingProps.js:

```

import React, { PureComponent } from 'react';
import { StyleSheet, View, TouchableNativeFeedback, Text, Keyboard } from
'react-native';

import { Navigation } from "react-native-navigation";
import DateTimePicker from '@react-native-community/datetimepicker';
import { colors } from '../styles/colors';
import * as Typography from '../styles/typography';
import * as DataBase from '../components/database';
import dayjs from 'dayjs';
import { TextInput } from 'react-native-gesture-handler';
import { FABButton } from '../fabButton';
import { DoneIcon } from '../sources/icons';
require('dayjs/locale/ru');

export class CommentSection extends PureComponent {
  constructor(props) {

```

```

super(props);
//this.showAcceptButton = this.showAcceptButton.bind(this);
this.state = {
  currentText: this.props.content,
  mainScreenHandler: this.props.mainScreenHandler,
  parentHandler: this.props.parentHandler,
  showAcceptButton: false,
}
}

```

```

EditComment(value) {
  console.log(value.nativeEvent.text);
  this.setState({ currentText: value.nativeEvent.text });
  DataBase.EditTask(this.props.id, 'comment', value.nativeEvent.text);
  this.setState({ showAcceptButton: false });
  this.state.mainScreenHandler(new Date());
}

```

```

showAcceptButton() {
  this.setState({ showAcceptButton: true });
  this.state.parentHandler({ showCompleteTask: false })
}

```

```

render() {
  return (
    <View style={styles.wrapper}>
      <View style={[styles.body, { flexDirection: 'column', alignItems:
'flex-start' }]}>
        <Text style={{ ...Typography.h3, color: colors.highlight
}}>Комментарий:</Text>

```

```

<View style={{
  height: 2,
  backgroundColor: colors.background,
  marginLeft: 56,
}} />
<TextInput style={{ ...Typography.h4 }}
  ref='commentInput'
  defaultValue={this.state.currentText}
  selectTextOnFocus
  multiline
  onFocus={() => this.showAcceptButton()}
  onEndEditing={value => this.EditComment(value)}
/>
</View>
<FABButton  onPress={() =>  this.refs.commentInput.blur()}
show={this.state.showAcceptButton}>
  <DoneIcon width={24} height={24} color={colors.white} />
</FABButton>
</View>
)
}
}

```

```

export class EditRepeatProps extends PureComponent {
  constructor(props) {
    super(props);
    this.state = {
      mainScreenHandler: this.props.mainScreenHandler,
      currentMode: this.props.content,
    }
  }

```

```

    }

    onSelected = (selectedItem) => {
      this.setState({ currentMode: selectedItem });
      DataBase.EditTask(this.props.id, 'repeat', selectedItem);
      this.state.mainScreenHandler(new Date());
    }

    showList() {
      Navigation.showOverlay({
        component: {
          name: 'listPicker',
          options: {
            overlay: {
              interceptTouchOutside: true,
            },
            statusBar: {
              backgroundColor: '#f3f3f3',
              style: 'dark',
            },
          },
          passProps: {
            data: ['Без повторений', 'Каждый день', 'Каждую неделю'],
            title: 'Повторять задачу',
            parentHandler: newState => this.setState(newState),
            onSelected: this.onSelected,
          }
        },
      });
    }
  }

```

```

render() {
  return (
    <View style={styles.wrapper}>
      <TouchableNativeFeedback onPress={() => this.showList()}>
        <View style={styles.body}>
          {this.props.children}
          <Text style={styles.text}>
            {this.state.currentMode == 'default' ? 'Без повторения' :
this.state.currentMode}
          </Text>
        </View>
      </TouchableNativeFeedback>
    </View>
  )
}
}

```

```

export class EditDateTimeProps extends PureComponent {
  constructor(props) {
    super(props);
    this.state = {
      show: false,
      mainScreenHandler: this.props.mainScreenHandler,
      currentDate: this.props.type == 'date' ?
dayjs(this.props.content).locale('ru').format('DD MMM YYYY') :
dayjs(this.props.content).locale('ru').format('мм мин.') != '00 мин.' ?
      dayjs(this.props.content).locale('ru').format('Н ч. мм мин.')
      : dayjs(this.props.content).locale('ru').format('Н ч.'),
    }
  }
}

```

```

    }

    onChange = (event, selectedDate) => {
      let formatDate;
      switch (this.props.type) {
        case 'time':
          formatDate = dayjs(selectedDate).locale('ru').format('H ч. мм мин. ');
          this.setState({ currentDate: formatDate, show: false });
          DataBase.EditTask(this.props.id, 'time', selectedDate);
          this.state.mainScreenHandler(new Date());
          break;
        case 'date':
          formatDate = dayjs(selectedDate).locale('ru').format('DD MMM
YYYYY');
          this.setState({ currentDate: formatDate, show: false });
          DataBase.EditTask(this.props.id, 'date', selectedDate);
          this.state.mainScreenHandler(new Date());
          break;
      }
    }

    render() {
      return (
        <View style={styles.wrapper}>
          <TouchableNativeFeedback onPress={() => this.setState({ show: true
        }}}>

          <View style={styles.body}>
            {this.props.children}
            <Text style={styles.text}>
              {this.state.currentDate}

```

```

        </Text>
      </View>
    </TouchableNativeFeedback>
    { this.state.show ?
      <DateTimePicker
        testID="dateTimePicker"
        timeZoneOffsetInMinutes={0}
        mode={this.props.type}
        onChange={this.onChange}
        value={new Date()}
        is24Hour={true}
        display="default"
      /> : null}
    </View>
  )
}
}

```

```

const styles = StyleSheet.create({
  wrapper: {
  },
  base: {
    marginHorizontal: 16,
    marginBottom: 4
  },
  body: {
    padding: 16,
    borderRadius: 4,
    flexDirection: 'row',
    alignItems: 'center',

```

```

    },
    text: {
      ...Typography.h3,
      color: colors.highlight,
      marginLeft: 12,
    }
  })

```

Файл **emptyListView.js**:

```

import React, { PureComponent } from 'react';
import { StyleSheet, View, Text, Dimensions } from 'react-native';
import * as Typography from '../styles/typography';

```

```

export class EmptyListView extends PureComponent {
  constructor(props) {
    super(props);
    this.state = {}
  }

  render() {
    return(
      <View style={styles.base}>
        {this.props.showIcon ? this.props.children : null}
        <Text style={styles.text}>{this.props.title}</Text>
      </View>
    )
  }
}

```

```

const styles = StyleSheet.create({

```

```

    wrapper: {
    },
    base: {
      height: Dimensions.get('window').height,
      justifyContent: 'center',
      alignItems: 'center',
      paddingHorizontal: 48,
      paddingBottom: '40%',
    },
    text: {
      ...Typography.h2,
      textAlign: 'center',
    }
  })

```

Файл fabButton.js:

```

import React, { PureComponent } from 'react';
import { StyleSheet, View, TouchableNativeFeedback } from 'react-native';
import { colors } from '../styles/colors';

```

```

export class FABButton extends PureComponent {
  constructor(props, icon) {
    super(props);
    this.state = {
      show: this.props.show,
      pressButton: this.props.onPress,
    }
  }

  render() {

```

```

return (this.props.show ?
  <View style={styles.wrapper}>
    <View style={styles.base}>
      <TouchableNativeFeedback
        onPress={() =>
this.state.pressButton()} background={TouchableNativeFeedback.Ripple('rgb(30, 38,
66)', true)}>
        <View style={styles.body}>
          {this.props.children}
        </View>
      </TouchableNativeFeedback>
    </View>
  </View> : null
)
}
}

```

```

const styles = StyleSheet.create({
  wrapper: {
    bottom: 0,
    right: 0,
    zIndex: 1,
    position: 'absolute',
  },
  base: {
    height: 56,
    width: 56,
    backgroundColor: colors.primary,
    elevation: 4,
    borderRadius: 32,
    margin: 16,
  },
});

```

```

        justifyContent: 'center',
        alignItems: 'center'
    },
    body: {
        height: 56,
        width: 56,
        margin: 16,
        justifyContent: 'center',
        alignItems: 'center'
    }
})

```

Файл listItem.js:

```

import React, { PureComponent } from 'react';
import Swipeable from 'react-native-gesture-handler/Swipeable';
import dayjs from 'dayjs';
require('dayjs/locale/ru');

```

```

import {
    StyleSheet,
    View,
    Text,
    TouchableWithoutFeedback,
} from 'react-native';
import * as DataBase from '../components/database';

```

```

import { h5, h3, h4 } from '../styles/typography';
import { colors } from '../styles/colors';

```

```

import { Navigation } from "react-native-navigation";

```

```

import { TimeIcon, RepeatIcon } from '../sources/icons';

const DeleteTaskBody = (progress, dragX) => {
  return (
    <View style={deleteViewTaskStyles.body}>
      <Text style={deleteViewTaskStyles.text}>Удалить задачу</Text>
    </View>
  )
}

const CompleteTaskBody = (progress, dragX) => {
  return (
    <View style={completeViewTaskStyles.body}>
      <Text style={completeViewTaskStyles.text}>Выполнить
задачу</Text>
    </View>
  )
}

export class ListItem extends PureComponent {
  constructor(props) {
    super(props);
    this.state = {
      parentHandler: this.props.parentHandler,
    }
  }

  deleteTask() {
    DataBase.deleteTask(this.props.taskId);
    this.state.parentHandler(new Date());
  }
}

```

```
}
```

```
completeTask() {
```

```
    DataBase.completeTask(this.props.taskId, !this.props.isCompleted);
```

```
    this.state.parentHandler(new Date());
```

```
    this.refs.SwipeableTask.close();
```

```
}
```

```
showTaskSettings() {
```

```
    Navigation.showModal({
```

```
        component: {
```

```
            name: 'taskViewerScreen',
```

```
            options: {
```

```
                overlay: {
```

```
                    interceptTouchOutside: true,
```

```
                },
```

```
                statusBar: {
```

```
                    backgroundColor: '#f3f3f3',
```

```
                    style: 'dark',
```

```
                },
```

```
                animations: {
```

```
                    showModal: {
```

```
                        translationX: {
```

```
                            from:
```

```
                            native').Dimensions.get('window').width,
```

```
                            to: 0,
```

```
                            duration: 300
```

```
                        },
```

```
                    },
```

```
                    dismissModal: {
```

```
                        require('react-
```

					ПГУ 2.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		77

```

        translationX: {
            from: 0,
            to: require('react-native').Dimensions.get('window').width,
            duration: 300
        }
    }
},
passProps: {
    parentHandler: this.state.parentHandler,
    id: this.props.taskId,
    listName: this.props.listName,
    taskTitle: this.props.taskTitle,
    taskDate: this.props.taskDate,
    taskTime: this.props.taskTime,
    taskRepeat: this.props.taskRepeat,
    taskComment: this.props.taskComment,
    isCompleted: this.props.isCompleted
},
}
});
}

render() {
    return (
        <View style={styles.base}>
            <Swipeable
                ref='SwipeableTask'
                containerStyle={[{ borderRadius: 4, elevation: 12, },
this.props.isCompleted == true ? completedTask.body : null]}

```

					ПГУ 2.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		78

```

renderLeftActions={ CompleteTaskBody }
renderRightActions={ DeleteTaskBody }
onSwipeableRightOpen={() => this.deleteTask()}
onSwipeableLeftOpen={() => this.completeTask()}
>
<TouchableWithoutFeedback      onPress={()      =>
this.showTaskSettings()}>
  <View style={[styles.body, this.props.isCompleted == true ?
completedTask.body : null]}>
    <View>
      <Text style={[styles.taskTitle, this.props.isCompleted ==
true ? completedTask.taskTitle : null]}>
        {this.props.taskTitle}
      </Text>
      <View style={styles.options}>
        {this.props.listName ?
          <View style={styles.optionsWrapper}>
            <Text
style={styles.optionsText}>{this.props.listName}</Text>
          </View>
          : null}
        {this.props.taskDate ?
          <View style={styles.optionsWrapper}>
            <Text      style={styles.optionsText}>      •
{dayjs(this.props.taskDate).locale('ru').format('DD MMM')}</Text>
          </View>
          : null}
        {this.props.taskTime ?
          <View style={styles.optionsWrapper}>
            <Text style={styles.optionsText}>•</Text>

```

```

<View style={{ paddingRight: 2, paddingTop: 1 }}>
  <TimeIcon      width={14}      height={14}
color={colors.primary} />
</View>
<Text style={styles.optionsText}>{
  dayjs(this.props.taskTime).locale('ru').format('mm мин.') != '00 мин.' ?
  dayjs(this.props.taskTime).locale('ru').format('H ч. mm мин.')
  :
  dayjs(this.props.taskTime).locale('ru').format('H ч.')
  }</Text>
</View>
: null}
{this.props.taskRepeat != 'Без повторений' ?
  <View style={styles.optionsWrapper}>
    <Text style={styles.optionsText}> • </Text>
    <View style={{ paddingRight: 2, paddingTop: 1 }}>
      <RepeatIcon      width={14}      height={14}
color={colors.primary} />
    </View>
    <View style={styles.optionsWrapper}>
      <Text
style={styles.optionsText}> {this.props.taskRepeat} </Text>
    </View>
  </View>
: null}
</View>
</View>
</View>

```

```

        </TouchableWithoutFeedback>
      </Swipeable>
    </View>
  )
}
}

```

```

const styles = StyleSheet.create({
  base: {
    marginLeft: 16,
    marginRight: 16,
    marginBottom: 4
  },
  body: {
    backgroundColor: colors.white,
    padding: 16,
    borderRadius: 4,
  },
  options: {
    flexDirection: 'row',
    flexWrap: 'wrap',
    paddingTop: 4,
  },
  taskTitle: {
    ...h3,
  },
  optionsWrapper: {
    alignItems: 'center',
    flexDirection: 'row',
  },

```

```

optionsText: {
  ...h5,
},
})

```

```

const completedTask = StyleSheet.create({
  body: {
    elevation: 0,
    backgroundColor: colors.white,
  },
  taskTitle: {
    textDecorationLine: 'line-through',
  },
  optionsText: {
    textDecorationLine: 'line-through',
  },
})

```

```

const completeViewTaskStyles = StyleSheet.create({
  body: {
    backgroundColor: colors.primary,
    flex: 1,
    justifyContent: 'center',
    paddingLeft: 16
  },
  text: {
    ...h5,
    color: colors.white,
  }
})

```

```
const deleteViewTaskStyles = StyleSheet.create({
  body: {
    backgroundColor: colors.red,
    flex: 1,
    justifyContent: 'center',
    alignItems: 'flex-end',
    paddingRight: 16
  },
  text: {
    ...h5,
    color: colors.white,
  }
})
```

Файл listPicker.js:

```
import React, { PureComponent } from 'react';
import { StyleSheet, View, TouchableNativeFeedback, Text } from 'react-native';
import { colors } from '../styles/colors';
import { Navigation } from "react-native-navigation";
import * as Typography from '../styles/typography';
import { mainStyles } from '../styles/styles';

export class ListPicker extends PureComponent {
  constructor(props) {
    super(props);
    this.state = {
      listData: this.props.data,
      parentHandler: this.props.parentHandler,
    }
  }
}
```

```

    }
  }

  onItemSelected(selectedItem) {
    setTimeout(() => Navigation.dismissOverlay(this.props.componentId),
100);

    console.log(selectedItem);
    this.props.onSelected(selectedItem);
  }

  render() {
    return (
      <View style={styles.wrapper}>
        <View style={styles.body}>
          <View style={styles.listItemBody}>
            <Text style={{ ...Typography.h4, alignSelf: 'center'
}}>{this.props.title}</Text>
          </View>
          {this.state.listData.map(data => {
            return <TouchableNativeFeedback onPress={() =>
this.onItemSelected(data)}>
              <View style={styles.listItemBody}>
                <Text style={{ ...Typography.h3 }}>{data}</Text>
              </View>
            </TouchableNativeFeedback>
          })}
        </View>
      </View>
    )
  }

```

```

    }

    const styles = StyleSheet.create({
      wrapper: {
        position: 'absolute',
        left: 0,
        right: 0,
        bottom: 0,
      },
      body: {
        backgroundColor: colors.white,
        paddingVertical: 8,
        elevation: mainStyles.elevation,
      },
      listItemBody: {
        paddingVertical: 16,
        paddingLeft: 16,
      }
    })

```

Файл listTitle.js:

```

import React, { PureComponent } from 'react';

import {
  StyleSheet,
  View,
  Text,
} from 'react-native';

import { h1, h3 } from '../styles/typography';
import { colors } from '../styles/colors';

```

```

export class ListTitle extends PureComponent {
  render() {
    if (this.props.time == undefined) {
      return (
        <View style={styles.body}>
          <Text style={{ ...h1 }}>{this.props.title}</Text>
        </View>
      )
    }
    return (
      <View style={styles.body}>
        <Text style={{ ...h1 }}>{this.props.title}</Text>
        <Text style={{ ...h3, color: colors.primary, paddingTop: 4
}}>{this.props.time} мин. свободного времени</Text>
      </View>
    )
  }
}

```

```

const styles = StyleSheet.create({
  body: {
    padding: 16,
  }
})

```

Файл newList.js:

```

import React, { PureComponent } from 'react';
import { StyleSheet, View, TextInput, BackHandler, TouchableNativeFeedback
} from 'react-native';
import { colors } from '../styles/colors';

```

```

import { AddIcon } from '../sources/icons';
import * as Typography from '../styles/typography';
import * as DataBase from '../components/database';

export class NewList extends PureComponent {
  constructor(props) {
    super(props);
    this.handleBackButtonClick = this.handleBackButtonClick.bind(this);
    this.state = {
      text: 'Создать новый список',
      parentHandler: this.props.parentHandler,
      mainScreenHandler: this.props.mainScreenHandler,
    }
  }

  handleBackButtonClick() {
    this.refs.TextInput.blur();
    return true;
  }

  addList() {
    if (this.refs.TextInput.isFocused() == true) {
      DataBase.createNewListItem(this.state.text);
      this.state.parentHandler(new Date());
      this.state.mainScreenHandler(new Date());
      console.log("updateMainScreen");
      this.refs.TextInput.blur();
    }
    else {
      this.refs.TextInput.focus();
    }
  }
}

```

```

    }
  }

  onChangeText(text) {
    this.setState({ text: text });
  }

  render() {
    return (
      <View style={styles.wrapper}>
        <View style={styles.base}>
          <TextInput style={{ ...Typography.h3 }}
            ref='TextInput'
            defaultValue={this.state.text}
            autoFocus={false}
            selectTextOnFocus
            onFocus={() =>
              BackHandler.addEventListener('hardwareBackPress', this.handleBackButtonClick)}
            onBlur={() => {
              this.setState({ text: 'Создать новый список' });
              BackHandler.removeEventListener('hardwareBackPress',
this.handleBackButtonClick);
            }}
            onChangeText={(text) => this.changeText(text)}
            onSubmitEditing={() => this.addList()}
          />
          <TouchableNativeFeedback
            onPress={() => this.addList()}
            background={TouchableNativeFeedback.Ripple(colors.primary,
true)}>

```

```

        <View style={styles.iconButton}>
          <AddIcon />
        </View>
      </TouchableNativeFeedback>
    </View>
  </View>
)
}
}

```

```

const styles = StyleSheet.create({
  wrapper: {
    position: 'absolute',
    bottom: 0,
    left: 0,
    right: 0,
  },
  base: {
    backgroundColor: colors.white,
    paddingLeft: 16,
    paddingRight: 10,
    paddingVertical: 12,
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
  },
  iconButton: {
    height: 40,
    width: 40,
    justifyContent: 'center',

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        alignItems: 'center',
    },
})

```

Файл newTask.js:

```

import React, { PureComponent } from 'react';
import { StyleSheet, View, TextInput, ScrollView, BackHandler,
TouchableNativeFeedback } from 'react-native';
import { colors } from '../styles/colors';
import { mainStyles } from '../styles/styles';
import * as Typography from '../styles/typography';
import * as DataBase from '../components/database';
import * as NewTaskProps from './newTaskProps';
import { DateIcon, TimeIcon, ListIcon, RepeatIcon } from '../sources/icons';

export class NewTask extends PureComponent {
  constructor(props) {
    super(props);
    this.handleBackButtonClick = this.handleBackButtonClick.bind(this);
    this.state = {
      parentHandler: this.props.parentHandler,
      date: new Date(),
      time: new Date(),
      repeat: 'Без повторений',
      listName: 'Все задачи',
    }
  }

  CreateTask(taskTitle) {
    console.log(this.state.time);
  }

```

```

        console.log(this.state.date);

        DataBase.createNewTaskItem(taskTitle.nativeEvent.text,
this.state.listName, this.state.date, this.state.time, this.state.repeat);

        console.log(taskTitle.nativeEvent.text);

        let defaultTime = new Date();
        defaultTime.setHours(1, 0, 0);
        this.setState({ time: defaultTime });
        this.state.parentHandler(false);
    }

```

```

UNSAFE_componentWillMount() {
    let defaultTime = new Date();
    defaultTime.setHours(1, 0, 0);
    this.setState({ time: defaultTime });
    BackHandler.addEventListener('hardwareBackPress',
this.handleBackButtonClick);
}

```

```

UNSAFE_componentWillUnmount() {
    BackHandler.removeEventListener('hardwareBackPress',
this.handleBackButtonClick);
}

```

```

handleBackButtonClick() {
    this.state.parentHandler(false);
    return true;
}

```

```

render() {

```

```
return (this.props.show ?
```

```
  <View style={styles.wrapper}>
```

```
    <View style={styles.base}>
```

```
      <View style={{ paddingLeft: 16 }}>
```

```
        <TextInput style={{ ...Typography.h4 }}>
```

```
          ref='TextInput'
```

```
          defaultValue={'Добавить новую задачу'}
```

```
          autoFocus={true}
```

```
          selectTextOnFocus
```

```
          onSubmitEditing={value => this.CreateTask(value)}
```

```
        />
```

```
      </View>
```

```
    <ScrollView
```

```
      horizontal={true}
```

```
    disableScrollViewPanResponder={true}>
```

```
      <NewTaskProps.NewTaskListPicker
```

```
        data={this.props.data}
```

```
        textInputSetFocus={() => this.refs.TextInput.focus()}
```

```
        parentHandler={newState => this.setState(newState)}
```

```
      >
```

```
        <ListIcon width={16} height={16} color={colors.white} />
```

```
      </NewTaskProps.NewTaskListPicker>
```

```
      <NewTaskProps.NewTaskDateTimeProps
```

```
        type={'date'}
```

```
        textInputSetFocus={() => this.refs.TextInput.focus()}
```

```
        parentHandler={newState => this.setState(newState)}
```

```
      >
```

```
        <DateIcon width={16} height={16} color={colors.white} />
```

```
      </NewTaskProps.NewTaskDateTimeProps>
```

```
      <NewTaskProps.NewTaskDateTimeProps
```

```
        type={'time'}
```

```

        textInputSetFocus={() => this.refs.TextInput.focus()}
        parentHandler={newState => this.setState(newState)}
      >
        <TimeIcon width={16} height={16} color={colors.white} />
      </NewTaskProps.NewTaskDateTimeProps>
      <NewTaskProps.NewTaskRepeatPicker
        textInputSetFocus={() => this.refs.TextInput.focus()}
        parentHandler={newState => this.setState(newState)} />
    </ScrollView>
  </View>
</View> : null
)
}
}

```

```

const styles = StyleSheet.create({
  wrapper: {
    position: 'absolute',
    bottom: 0,
    right: 0,
    left: 0,
  },
  base: {
    backgroundColor: colors.white,
    elevation: mainStyles.elevation,
  }
})

```

Файл newTaskProps.js:

```
import React, { PureComponent } from 'react';
```

```

import { StyleSheet, View, TouchableNativeFeedback, Text, Keyboard } from
'react-native';

import { Navigation } from "react-native-navigation";
import DateTimePicker from '@react-native-community/datetimepicker';
import { colors } from '../styles/colors';
import * as Typography from '../styles/typography';
import dayjs from 'dayjs';
import { RepeatIcon } from '../sources/icons';
require('dayjs/locale/ru');

export class NewTaskRepeatPicker extends PureComponent {
  constructor(props) {
    super(props);
    this.state = {
      parentHandler: this.props.parentHandler,
      currentDate: 'Повторение',
    }
  }

  onSelected = (selectedItem) => {
    this.setState({ currentDate: selectedItem });
    this.state.parentHandler({ repeat: selectedItem });
    this.props.textInputSetFocus();
  }

  showList() {
    Navigation.showOverlay({
      component: {
        name: 'listPicker',
        options: {

```

```

        overlay: {
            interceptTouchOutside: true,
        },
        statusBar: {
            backgroundColor: '#f3f3f3',
            style: 'dark',
        },
    },
    passProps: {
        data: ['Каждый день', 'Каждую неделю'],
        title: 'Повторять задачу',
        parentHandler: newState => this.setState(newState),
        onSelect: this.onSelected,
    }
    }
    });
}

render() {
    return (
        <View style={styles.wrapper}>
            <View style={styles.rippleWrapper}>
                <TouchableNativeFeedback
                    useForeground={true}
                    onPress={() => this.showList()}
                    background={TouchableNativeFeedback.Ripple('rgb(30, 38,
66)', true)}>
                    <View style={[styles.base, {
                        backgroundColor: this.state.currentDate == 'Повторение' ?
colors.white : colors.primary,

```

```

paddingLeft: this.state.currentDate === 'Повторение' ? 0 : 16,
marginRight: 16,
}}}>
<RepeatIcon width={16} height={16}
color={this.state.currentDate === 'Повторение' ? colors.primary : colors.white} />
<Text style={[styles.text, {
color: this.state.currentDate === 'Повторение' ?
colors.primary : colors.white,
}]}>{this.state.currentDate}</Text>
</View>
</TouchableNativeFeedback>
</View>
</View>
)
}
}

```

```

export class NewTaskDateTimeProps extends PureComponent {
  constructor(props) {
    super(props);
    this.state = {
      show: false,
      parentHandler: this.props.parentHandler,
      currentDate: this.props.type === 'date' ? 'Сегодня' : '1 ч.',
    }
  }
}

```

```

onChange = (event, selectedDate) => {
  let formatDate;

```

```

switch (this.props.type) {
  case 'time':
    formatDate = dayjs(selectedDate).locale('ru').format('H ч. мм мин. ');
    console.log(formatDate);
    this.setState({ currentDate: formatDate, show: false });
    this.state.parentHandler({ [this.props.type]: selectedDate });
    this.props.textInputSetFocus();
    break;
  case 'date':
    formatDate = dayjs(selectedDate).locale('ru').format('DD MMMM');
    console.log(formatDate);
    this.setState({ currentDate: formatDate, show: false });
    this.state.parentHandler({ [this.props.type]: selectedDate });
    this.props.textInputSetFocus();
    break;
}
}

```

```

render() {
  return (
    <View style={styles.wrapper}>
      <View style={styles.rippleWrapper}>
        <TouchableNativeFeedback
          useForeground={true}
          onPress={() => {
            Keyboard.dismiss();
            this.setState({ show: true }) }}
          background={TouchableNativeFeedback.Ripple('rgb(30, 38, 66)', true)}>
          <View style={styles.base}>

```

```

        { this.props.children }
        <Text style={styles.text}>{ this.state.currentDate } </Text>
      </View>
    </TouchableNativeFeedback>
  </View>
  { this.state.show ?
    <DateTimePicker
      testID="dateTimePicker"
      timeZoneOffsetInMinutes={0}
      mode={ this.props.type }
      onChange={ this.onChange }
      value={ new Date() }
      is24Hour={ true }
      display="default"
    /> : null }
  </View>
)
}
}

```

```

export class NewTaskListPicker extends PureComponent {
  constructor(props) {
    super(props);
    this.showList = this.showList.bind(this);
    this.state = {
      parentHandler: this.props.parentHandler,
      currentList: 'Все задачи',
    }
  }
}

```

```

onSelected = (selectedItem) => {
  this.setState({ currentList: selectedItem });
  this.state.parentHandler({ listName: selectedItem });
  this.props.textInputSetFocus();
  this.props.data.map(data => {
    console.log(data.title);
  })
}

```

```

showList() {
  Keyboard.dismiss();
  let listArray = [];
  this.props.data.map(data => {
    listArray.push(data.title);
  });
  if (listArray == 0) {
    this.props.textInputSetFocus();
    return;
  }
  Navigation.showOverlay({
    component: {
      name: 'listPicker',
      options: {
        overlay: {
          interceptTouchOutside: true,
        },
        statusBar: {
          backgroundColor: '#f3f3f3',
          style: 'dark',
        },
      },
    },
  });
}

```

```

    },
    passProps: {
      data: listArray,
      title: 'Выберите список',
      parentHandler: newState => this.setState(newState),
      onSelected: this.onSelected,
    }
  }
});
}

render() {
  return (
    <View style={styles.wrapper}>
      <View style={styles.rippleWrapper}>
        <TouchableNativeFeedback
          useForeground={true}
          onPress={() => this.showList()}
          background={TouchableNativeFeedback.Ripple('rgb(30, 38,
66)', true)}>
          <View style={styles.base}>
            {this.props.children}
            <Text style={styles.text}>{this.state.currentList}</Text>
          </View>
        </TouchableNativeFeedback>
      </View>
    </View>
  )
}
}

```

```

const styles = StyleSheet.create({
  wrapper: {
    marginLeft: 16,
    paddingBottom: 16
  },
  rippleWrapper: {
    overflow: 'hidden',
    borderRadius: 32,
    alignSelf: 'flex-start',
    marginTop: 4,
  },
  base: {
    backgroundColor: colors.primary,
    paddingHorizontal: 16,
    paddingVertical: 10,
    alignSelf: 'flex-start',
    flexDirection: 'row',
    alignItems: 'center',
    borderRadius: 32,
  },
  text: {
    ...Typography.h4,
    color: colors.white,
    marginLeft: 4,
  }
})

```

Файл googleLoginButton.js:

```

import React, { PureComponent } from 'react';

```

					ПГУ 2.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		101

```
import auth from '@react-native-firebase/auth';
import { GoogleSignin } from '@react-native-community/google-signin';
import { Animated, Text, StyleSheet, View, TouchableNativeFeedback } from
'react-native';
```

```
import { GoogleIcon } from '../sources/icons';
```

```
GoogleSignin.configure({
  webClientId: 'secret code',
});
```

```
export class GoogleLogin extends PureComponent {
  async onGoogleButtonPress() {
    // Get the users ID token
    const { idToken } = await GoogleSignin.signIn();
    // Create a Google credential with the token
    const googleCredential = auth.GoogleAuthProvider.credential(idToken);
    // Sign-in the user with the credential
    return auth().signInWithCredential(googleCredential);
  }
}
```

```
render() {
  return (
    <TouchableNativeFeedback onPress={() => this.onGoogleButtonPress()}>
      <Animated.View style={Buttons.Body}>
        <GoogleIcon style={{ flex: 1, position: 'absolute', margin: 16 }} />
        <Text style={Buttons.text, { flex: 1, textAlign: 'center' }}>Войти через
Google аккаунт</Text>
      </Animated.View>
    </TouchableNativeFeedback>
  );
}
```

```

    );
  }
}

const Buttons = StyleSheet.create({
  Body: {
    height: 56,
    width: "90%",
    flexDirection: 'row',
    alignItems: "center",
    borderRadius: 4,
    backgroundColor: "#fff",
    elevation: 2,
  },
  text: {
    fontSize: 14,
    fontFamily: 'Roboto',
  }
});

```

Приложения А.2. Screens

Файл mainScreen.js:

```

import React, { useState, useEffect, PureComponent } from 'react';
import { Navigation } from "react-native-navigation";
import {
  StyleSheet,
  View,
  Text,
  SafeAreaView,

```

```

    TouchableWithoutFeedback,
    TouchableNativeFeedback,
    FlatList,
  } from 'react-native';
import { ListTitle } from '../components/listTitle'
import { ListItem } from '../components/listItem';
import { colors } from '../styles/colors';
import { h2 } from '../styles/typography'
import { MenuIcon, MoreIcon, AddIcon, ListIcon } from '../sources/icons';
import * as DataBase from '../components/database';
import { FABButton } from '../components/fabButton';
import { NewTask } from '../components/newTask';
import { EmptyListView } from '../components/emptyListView';

export const Main = (props) => {
  const [resultData, setResultData] = useState();
  const [todayList, setTodayList] = useState();
  const [showAdding, setShowingAdding] = useState();
  const [update, setUpdate] = useState();

  useEffect(() => {
    DataBase.queryAllDataBase().then(dataBase => {
      setResultData((Array.from(dataBase)));
      return (dataBase);
    }).then(database => {
      Array.from(database).length == 0 ? DataBase.startNewDataBase() : null;
    }).then(() => {
      DataBase.queryTodayTasks().then(dataBase => {
        setTodayList((JSON.parse(JSON.stringify(Array.from(dataBase)))));
      })
    })
  })
}

```

```

    });
    Navigation.updateProps('sideMenuId', {
      mainScreenHandler: setUpdate,
    })
  }, [showAdding, update])

function addTask() {
  setShowingAdding(true);
}

return (
  <SafeAreaView styles={{ flex: 1 }}>
    <View style={styles.body}>
      <View style={styles.topBar}>
        <View style={styles.leftTopBar}>
          <TouchableNativeFeedback onPress={() => {
            Navigation.mergeOptions(props.componentId, {
              sideMenu: {
                left: {
                  visible: true,
                },
              },
            )});
          }}
          background={TouchableNativeFeedback.Ripple(colors.primary, true)}>
            <View style={styles.topBarButton}>
              <MenuIcon />
            </View>
          </TouchableNativeFeedback>
          <Text style={styles.topBarTitle}>Список задач</Text>

```

```

</View>
<TouchableWithoutFeedback>
  <View style={styles.topBarButton}>
    <MoreIcon />
  </View>
</TouchableWithoutFeedback>
</View>

{ console.log('Main Screen updated')}

{todayList == 0 ?
  <EmptyListView showIcon={true} title={'На сегодня нет ни
одной задачи'} >
    <ListItemIcon width={72} height={72} color={colors.primary}/>
  </EmptyListView>
:
<FlatList
  data={todayList}
  renderItem={({ item }) => (
    <ListItem
      parentHandler={setUpdate}
      taskId={item.id}
      taskTitle={item.taskTitle}
      listName={item.listName}
      taskDate={item.date}
      taskTime={item.time}
      taskRepeat={item.repeat}
      taskComment={item.comment}
      isCompleted={item.isCompleted}
    />

```

```

    })
    keyExtractor={item => item.id.toString()}
    ListFooterComponent={() => <View style={{ height: 32
}}></View>}

    ListHeaderComponent={() => (
      <ListTitle
        title={'Сегодня'}
        time={'8 ч. 15'}
      />
    )}
  />
}

<FABButton onPress={addTask} show={!showAdding}>
  <AddIcon color={colors.white} />
</FABButton>

  <NewTask data={resultData} parentHandler={setShowingAdding}
show={showAdding} />
</View>
</SafeAreaView>
)
}

const styles = StyleSheet.create({
  body: {
    backgroundColor: colors.background,
    height: "100%",
  },
  topBar: {

```

```

      height: 56,
      flexDirection: 'row',
      padding: 16,
      justifyContent: 'space-between',
      alignItems: 'center',
    },
    leftTopBar: {
      flexDirection: 'row',
      alignItems: 'center',
    },
    topBarTitle: {
      ...h2,
      paddingLeft: 24,
    },
    topBarButton: {
      height: 32,
      width: 32,
      justifyContent: 'center',
      alignItems: 'center'
    },
  },
})

```

Файл **listViewer.js**:

```

import React, { useState, useEffect } from 'react';
import { Navigation } from "react-native-navigation";
import {
  StyleSheet,
  View,
  Text,
  SafeAreaView,

```

```

    TouchableWithoutFeedback,
    TouchableNativeFeedback,
    FlatList,
  } from 'react-native';
import { ListTitle } from '../components/listTitle'
import { ListItem } from '../components/listItem';
import { colors } from '../styles/colors';
import { h2 } from '../styles/typography'
import { MenuIcon, MoreIcon, AddIcon } from '../sources/icons';
import * as DataBase from '../components/database';
import { FABButton } from '../components/fabButton';
import { NewTask } from '../components/newTask';
import { EmptyListView } from '../components/emptyListView';

export const ListViewer = (props) => {
  const [resultData, setResultData] = useState();
  const [List, setList] = useState();
  const [showAdding, setShowingAdding] = useState();
  const [update, setUpdate] = useState();

  useEffect(() => {
    DataBase.queryAllDataBase().then(dataBase => {
      setResultData((Array.from(dataBase)));
      return (dataBase);
    }).then(database => {
      Array.from(database).length == 0 ? DataBase.startNewDataBase() : null;
    }).then(() => {

      if (props.listName == 'Все задачи') {
        DataBase.queryAllTasks().then(dataBase => {

```

```

        setList((JSON.parse(JSON.stringify(Array.from(dataBase)))));
    })
}
if (props.listName === 'Важные') {
    DataBase.queryImportantTasks().then(dataBase => {
        setList((JSON.parse(JSON.stringify(Array.from(dataBase)))));
    })
}
if (props.listName !== 'Все задачи' && props.listName !== 'Важные') {
    DataBase.querySelectedListTasks(props.listName).then(dataBase =>
{
        setList((JSON.parse(JSON.stringify(Array.from(dataBase)))));
    })
}
});
}, [showAdding, update])

function addTask() {
    setShowingAdding(true);
}

return (
    <SafeAreaView styles={{ flex: 1 }}>
        <View style={styles.body}>
            <View style={styles.topBar}>
                <View style={styles.leftTopBar}>
                    <TouchableNativeFeedback onPress={() => {
                        Navigation.mergeOptions(props.componentId, {
                            sideMenu: {
                                left: {

```

```

        visible: true,

        },

    },

    });

}}

background={TouchableNativeFeedback.Ripple(colors.primary, true)}>
    <View style={styles.topBarButton}>
        <MenuIcon />
    </View>
</TouchableNativeFeedback>
<Text style={styles.topBarTitle}>Список задач</Text>
</View>
<TouchableWithoutFeedback>
    <View style={styles.topBarButton}>
        <MoreIcon />
    </View>
</TouchableWithoutFeedback>
</View>

{console.log('List Viewer updated')}

{List == 0 ?
    <EmptyListView title={'Вы не создали еще ни одной задачи'} />
    :
    <FlatList
        data={List}
        renderItem={({ item }) => (
            <ListItem
                parentHandler={setUpdate}
                taskId={item.id}

```

```

        taskTitle={item.taskTitle}
        listName={item.listName}
        taskDate={item.date}
        taskTime={item.time}
        taskComment={item.comment}
        taskRepeat={item.repeat}
        isCompleted={item.isCompleted}
    />
  ))
  keyExtractor={item => item.id.toString()}
  ListFooterComponent={() => <View style={{ height: 32
}}></View>}

  ListHeaderComponent={() => (
    <ListTitle
      title={props.listName}
    />
  )}
/>
}

<FABButton onPress={addTask} show={!showAdding}>
  <AddIcon color={colors.white} />
</FABButton>

  <NewTask data={resultData} parentHandler={setShowingAdding}
show={showAdding} />
</View>
</SafeAreaView>
)
}

```

```

const styles = StyleSheet.create({
  body: {
    backgroundColor: colors.background,
    height: "100%",
  },
  topBar: {
    height: 56,
    flexDirection: 'row',
    padding: 16,
    justifyContent: 'space-between',
    alignItems: 'center',
  },
  leftTopBar: {
    flexDirection: 'row',
    alignItems: 'center',
  },
  topBarTitle: {
    ...h2,
    paddingLeft: 24,
  },
  topBarButton: {
    height: 32,
    width: 32,
    justifyContent: 'center',
    alignItems: 'center'
  },
});

```

Файл loginScreen.js:

```

import React, { useState, useEffect } from 'react';

```

					ПГУ 2.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		113

```

import {
  StyleSheet,
  View,
  Text,
  Image,
  SafeAreaView,
  Dimensions,
} from 'react-native';
import { colors } from '../styles/colors';
import { h1 } from '../styles/typography';
import auth from '@react-native-firebase/auth';
import { GoogleLogin } from './googleLoginButton';
import { Navigation } from "react-native-navigation";

export const Login = (props) => {
  const [initializing, setInitializing] = useState(true);
  const [user, setUser] = useState();

  function onAuthStateChanged(user) {
    setUser(user);
    if (initializing) setInitializing(false);
  }

  useEffect(() => {
    const subscriber = auth().onAuthStateChanged(onAuthStateChanged);
    return subscriber;
  }, []);

  if (initializing) return null;

  if (!user) {

```

```

return (
  <SafeAreaView>
    <View style={styles.body}>
      <View style={styles.header}>
        <Image style={styles.logo}
          source={require("/Graduate
Work/TaskManagerGWAS/images/logo.png")}
        />
        <Text style={styles.logoname}>TaskManager</Text>
      </View>
      <GoogleLogin />
    </View>
  </SafeAreaView>
);
}

```

```

if (user) Navigation.setStackRoot(props.componentId, {
  sideMenu: {
    left: {
      component: {
        id: 'sideMenuId',
        name: 'sideMenu',
        passProps: {
          mainScreenHandler: 'lol',
        },
      },
    },
    center: {
      stack: {
        children: [{

```

```

        component: {
            name: 'mainScreen'
        }
    }]
}
},
options: {
    sideMenu: {
        left: {
            width: Dimensions.get('window').width,
        },
    }
}
},
});

return (null)
}

const styles = StyleSheet.create({
    body: {
        height: "100%",
        alignItems: "center",
        justifyContent: 'space-between',
        paddingBottom: 24,
        backgroundColor: colors.background
    },
    header: {
        marginTop: '60%'
    },
},

```

```

    logo: {
      height: 72,
      width: 72,
      alignSelf: "center",
    },
    logoname: {
      ...h1,
      textAlign: "center",
    },
  },
})

```

Файл sideMenu.js:

```

import React, { useState, useEffect, PureComponent } from 'react';
import {
  StyleSheet,
  View,
  Text,
  TouchableNativeFeedback,
  Image,
  Dimensions,
} from 'react-native';
import * as DataBase from '../components/database';
import { colors } from '../styles/colors';
import { h2, h3, h4 } from '../styles/typography';
import auth from '@react-native-firebase/auth';
import { GoogleSignin } from '@react-native-community/google-signin';
import { Navigation } from "react-native-navigation";
import { NewList } from '../components/newList';
import { ImportantIcon, DateIcon, AllTasksIcon } from '../sources/icons';

```

```

const Separator = () => {
  return (
    <View style={{
      height: 2,
      backgroundColor: colors.background,
      marginLeft: 16,
    }} />
  )
}

export const SideMenu = (props) => {
  const [listArray, setlistArray] = useState();
  const [update, setUpdate] = useState();
  const user = auth().currentUser;

  useEffect(() => {
    DataBase.queryAllDataBase().then(dataBase => {
      setlistArray((Array.from(dataBase)));
      return (Array.from(dataBase));
    });
  }, [update])

  function deleteList(id) {
    console.log(id);
    DataBase.deleteList(id);
    props.mainScreenHandler(new Date());
    setUpdate(new Date());
  }

  function navigateTo(listName, componentName = 'listViewerScreen') {

```

```

Navigation.setStackRoot(props.componentId, {
  sideMenu: {
    left: {
      component: {
        id: 'sideMenuId',
        name: 'sideMenu',
        passProps: {
          mainScreenHandler: props.mainScreenHandler,
        }
      },
    },
  },
  center: {
    stack: {
      children: [{
        component: {
          name: componentName,
          passProps: {
            listName: listName,
          }
        }
      }]
    }
  },
  options: {
    sideMenu: {
      left: {
        width: Dimensions.get('window').width,
      },
    }
  }
}

```

```

    },
  });
}

if (typeof (listArray) === 'undefined') {
  return (
    <View style={styles.body}>
      <Text>undefined</Text>
    </View>
  )
}
else {
  return (
    <View style={styles.body}>
      <TouchableNativeFeedback onLongPress={() => {
        Navigation.setStackRoot(props.componentId, {
          component: {
            name: 'loginScreen'
          }
        });
        GoogleSignin.signOut();
        auth().signOut();
      }}>
        <View style={styles.userBody}>
          {console.log('SideMenu updated')}
          <Image
            style={styles.logo}
            source={{ uri: user.photoURL, }}
          />
          <View style={styles.userInfoContainer}>

```

```

        <Text style={styles.userName}>{user.displayName}</Text>
        <Text style={styles.userEmail}>{user.email}</Text>
    </View>
</View>
</TouchableNativeFeedback>

<Separator />

<View>
    <TouchableNativeFeedback onPress={() => navigateTo('Все
задачи')}}>
        <View style={styles.listItem}>
            <AllTasksIcon width={32} height={32}
color={colors.primary} />
            <Text style={styles.listItemText}>Все задачи</Text>
        </View>
    </TouchableNativeFeedback>
    <TouchableNativeFeedback onPress={() => navigateTo('Сегодня',
'mainScreen')}}>
        <View style={styles.listItem}>
            <DateIcon width={32} height={32} color={colors.primary}
/>
            <Text style={styles.listItemText}>Сегодня</Text>
        </View>
    </TouchableNativeFeedback>
    <TouchableNativeFeedback onPress={() =>
navigateTo('Важные')}}>
        <View style={styles.listItem}>
            <ImportantIcon width={32} height={32}
color={colors.primary} />

```

```

        <Text style={styles.listItemText}>Важные</Text>
      </View>
    </TouchableNativeFeedback>
  </View>

  <Separator />

  <NewList      mainScreenHandler={props.mainScreenHandler}
parentHandler={setUpdate} />

  <View>
    {listArray.map(data => {
      return (
        <TouchableNativeFeedback
          onPress={() => navigateTo(data.title)}
          onLongPress={() => { deleteList(data.id) }}
        >
          <View style={styles.listItem}>
            <Text style={styles.listItemText}>{data.title}</Text>
          </View>
        </TouchableNativeFeedback>
      )
    })}
  </View>
</View>

)
}

return (null)
}

```

```

const styles = StyleSheet.create({
  body: {
    backgroundColor: colors.white,
    height: '100%',
  },
  listItem: {
    flexDirection: 'row',
    alignItems: 'center',
    paddingHorizontal: 16,
    paddingVertical: 16,
  },
  listItemText: {
    ...h2,
    marginLeft: 12,
  },
  userBody: {
    paddingHorizontal: 16,
    paddingVertical: 16,
    flexDirection: 'row',
  },
  userInfoContainer: {
    paddingLeft: 12,
    justifyContent: 'center'
  },
  userName: {
    ...h2,
  },
  userEmail: {
    ...h4,
  },

```

```

    },
    logo: {
      width: 48,
      height: 48,
      borderRadius: 48,
      marginTop: 4
    },
  },
))

```

Файл **taksViewer.js**:

```

import React, { useState, useEffect, useRef } from 'react';
import { Navigation } from "react-native-navigation";
import {
  StyleSheet,
  View,
  Text,
  SafeAreaView,
  TouchableNativeFeedback,
  BackHandler,
} from 'react-native';
import { colors } from '../styles/colors';
import { h2, h1 } from '../styles/typography'
import { BackIcon, DateIcon, AddIcon, TimeIcon, RepeatIcon, DoneIcon }
from '../sources/icons';
import * as DataBase from '../components/database';
import { FABButton } from '../components/fabButton';
import { EditDateTimeProps, EditRepeatProps, CommentSection } from
'../components/editingProps';
import { mainStyles } from '../styles/styles';
import { TextInput } from 'react-native-gesture-handler';

```

```

const Separator = () => {
  return (
    <View style={{
      height: 2,
      backgroundColor: colors.background,
      marginLeft: 56,
    }} />
  )
}

export const TaskViewer = (props) => {
  const [state, setState] = useState({
    mainScreenHandler: props.parentHandler,
    date: props.taskDate,
    time: props.taskTime,
    repeat: props.taskRepeat,
    listName: props.listName,
    comment: props.taskComment,
    isCompleted: props.isCompleted,
    showCompleteTask: props.isCompleted ? false : true,
  });
  const [commentIsEditing, setCommentIsEditing] = useState(false);
  const [update, setUpdate] = useState();
  const inputRef = useRef();

  useEffect(() => {
    }, [update])

```

```

function updateTaskTitle(value) {
    DataBase.EditTask(props.id, 'taskTitle', value.nativeEvent.text);
    state.mainScreenHandler(new Date());
}

function handleBackButtonClick() {
    inputRef.current.blur();
    BackHandler.removeListener('hardwareBackPress',
handleBackButtonClick)
    return true;
}

function completeTask() {
    DataBase.completeTask(props.id, !props.isCompleted);
    setState({ isCompleted: !props.isCompleted });
    state.mainScreenHandler(new Date());
}

return (
    <SafeAreaView styles={{ flex: 1 }}>
        <View style={styles.body}>
            <View style={styles.topBar}>
                <View style={styles.leftTopBar}>
                    <TouchableNativeFeedback onPress={() => {
                        Navigation.dismissModal(props.componentId);
                    }}
background={TouchableNativeFeedback.Ripple(colors.primary, true)}>
                        <View style={styles.topBarButton}>
                            <BackIcon />
                        </View>

```

```

        </TouchableNativeFeedback>
        <Text style={styles.topBarTitle}>{props.listName}</Text>
    </View>
</View>

{console.log('Task Viewer updated')}

<View style={styles.taskTitleBody}>
    <TextInput
        ref={inputRef}
        style={[styles.taskTitle, state.isCompleted ? {
textDecorationLine: 'line-through' } : null]}
        defaultValue={props.taskTitle}
        selectTextOnFocus
        multiline
        onFocus={() =>
BackHandler.addEventListener('hardwareBackPress', handleBackButtonClick)}
        onEndEditing={value => updateTaskTitle(value)}
    />
</View>

<View style={styles.editingPropsWrapper}>
    <EditRepeatProps id={props.id}
mainScreenHandler={state.mainScreenHandler} content={state.repeat}>
        <RepeatIcon width={24} height={24} color={colors.highlight}
    />

    </EditRepeatProps>

    <Separator />

```

```

        <EditDateTimeProps          type={'date'}          id={props.id}
mainScreenHandler={state.mainScreenHandler} content={state.time}>
        <DateIcon width={24} height={24} color={colors.highlight} />
    </EditDateTimeProps>

    <Separator />

    <EditDateTimeProps          type={'time'}          id={props.id}
mainScreenHandler={state.mainScreenHandler} content={state.time}>
        <TimeIcon width={24} height={24} color={colors.highlight} />
    </EditDateTimeProps>
</View>

<View style={styles.editingPropsWrapper}>
    <CommentSection    id={props.id}    parentHandler={setState}
mainScreenHandler={state.mainScreenHandler} content={state.comment != null ?
state.comment : 'Добавить комментарий...'} />
</View>

    <FABButton                                onPress={completeTask}
show={state.showCompleteTask}>
        <DoneIcon width={24} height={24} color={colors.white} />
    </FABButton>

</View>
</SafeAreaView>
)
}

const styles = StyleSheet.create({
    body: {

```

```

        backgroundColor: colors.background,
        height: "100%",
    },
    topBar: {
        height: 56,
        flexDirection: 'row',
        padding: 16,
        justifyContent: 'space-between',
        alignItems: 'center',
    },
    leftTopBar: {
        flexDirection: 'row',
        alignItems: 'center',
    },
    topBarTitle: {
        ...h2,
        paddingLeft: 24,
    },
    topBarButton: {
        height: 32,
        width: 32,
        justifyContent: 'center',
        alignItems: 'center'
    },
    taskTitleBody: {
        paddingHorizontal: 16,
    },
    taskTitle: {
        ...h1,
    },

```

```

editingPropsWrapper: {
  marginVertical: 16,
  marginHorizontal: 16,
  elevation: mainStyles.elevation,
  backgroundColor: colors.white,
  borderRadius: mainStyles.borderRadius,
}
})

```

Приложения А.3. Sources

Файл icons.js:

```

import React from "react";
import Svg, { Path } from "react-native-svg";
import { colors } from '../styles/colors';

export function GoogleIcon(props) {
  return (
    <Svg width={24} height={24} viewBox="0 0 24 24" fill="none"
    {...props}>
      <Path
        fillRule="evenodd"
        clipRule="evenodd"
        d="M20.64 12.205c0-.639-.057-1.252-.164-1.841H12v3.481h4.844a4.14 4.14 0 01-1.796 2.716v2.259h2.908c1.702-1.567 2.684-3.875 2.684-6.615z"
        fill="#4285F4"
      />
      <Path
        fillRule="evenodd"

```

```

        clipRule="evenodd"
        d="M12 21c2.43 0 4.467-.806 5.956-2.181-2.908-2.259c-.806.54-
1.837.86-3.048.86-2.344 0-4.328-1.584-5.036-3.711H3.957v2.332A8.997 8.997 0
0012 21z"
        fill="#34A853"
    />
    <Path
        fillRule="evenodd"
        clipRule="evenodd"
        d="M6.964 13.71A5.41 5.41 0 016.682 12c0-.593.102-1.17.282-
1.71V7.958H3.957A8.996 8.996 0 003 12c0 1.452.348 2.827.957 4.042l3.007-2.332z"
        fill="#FBBC05"
    />
    <Path
        fillRule="evenodd"
        clipRule="evenodd"
        d="M12 6.58c1.321 0 2.508.454 3.44 1.345l2.582-2.58C16.463 3.891
14.426 3 12 3a8.997 8.997 0 00-8.043 4.958l3.007 2.332C7.672 8.163 9.656 6.58 12
6.58z"
        fill="#EA4335"
    />
</Svg>
)
}

export function MenuIcon(props) {
    return (
        <Svg width={24} height={24} fill="none" viewBox="0 0 24 24"
        {...props}>

```

```

        <Path fill={props.color ? props.color : colors.primary} d="M3 18h18v-
2H3v2zm0-5h18v-2H3v2zm0-7v2h18V6H3z" />
      </Svg>
    )
  }

export function MoreIcon(props) {
  return (
    <Svg width={12} height={24} fill="none" viewBox="0 0 12 24"
    {...props}>
      <Path
        fill={props.color ? props.color : colors.primary}
        d="M6 8c1.1 0 2-.9 2-2s-.9-2-2-2-.9-2 2-.9 2 2 2zm0 2c-1.1 0-2 .9-2
2s.9 2 2 2-.9 2-2-.9-2-2-2zm0 6c-1.1 0-2 .9-2 2s.9 2 2 2-.9 2-2-.9-2-2-2z"
      />
    </Svg>
  )
}

export function AddIcon(props) {
  return (
    <Svg width={24} height={24} viewBox="0 0 24 24" fill="none"
    {...props}>
      <Path
        fillRule="evenodd"
        clipRule="evenodd"
        d="M11 5v6H5v2h6v6h2v-6h6v-2h-6V5h-2z"
        fill={props.color ? props.color : colors.primary}
      />
    </Svg>
  )
}

```

```

    )
  }

  export function BackIcon(props) {
    return (
      <Svg width={24} height={24} viewBox="0 0 24 24" fill="none"
{...props}>
        <Path
          fillRule="evenodd"
          clipRule="evenodd"
          d="M20 11H7.8l5.6-5.6L12 4l-8 8 8 8 1.4-1.4L7.8 13H20v-2z"
          fill={props.color ? props.color : colors.primary}
        />
      </Svg>
    )
  }

```

```

  export function DateIcon(props) {
    return (
      <Svg height={props.height} viewBox="0 0 24 24" width={props.width}
{...props}>
        <Path d="M0 0h24v24H0z" fill="none" />
        <Path
          d="M17 12h-5v5h5v-5zM16 1v2H8V1H6v2H5c-1.11 0-1.99.9-1.99
2L3 19a2 2 0 002 2h14c1.1 0 2-.9 2-2V5c0-1.1-.9-2-2-2h-1V1h-2zm3
18H5V8h14v11z"
          fill={props.color ? props.color : colors.primary}
        />
      </Svg>
    )
  }

```

```

    }

    export function TimeIcon(props) {
      return (
        <Svg height={props.height} viewBox="0 0 24 24" width={props.width}
        {...props}>
          <Path d="M0 0h24v24H0z" fill="none" />
          <Path
            d="M11.99 2C6.47 2 2 6.48 2 12s4.47 10 9.99 10C17.52 22 22 17.52
            22 12S17.52 2 11.99 2zM12 20c-4.42 0-8-3.58-8-8s3.58-8 8-8 3.58 8 8-3.58 8-8 8z"
            fill={props.color ? props.color : colors.primary}
          />
          <Path
            d="M12.5 7H11v6l5.25 3.15.75-1.23-4.5-2.67z"
            fill={props.color ? props.color : colors.primary}
          />
        </Svg>
      )
    }

```

```

    export function ListIcon(props) {
      return (
        <Svg height={props.height} viewBox="0 0 24 24" width={props.width}
        {...props}>
          <Path d="M0 0h24v24H0z" fill="none" />
          <Path
            d="M19 5v14H5V5h14m1.1-2H3.9c-.5 0-.9.4-.9.9v16.2c0
            .4.4.9.9h16.2c.4 0 .9-.5.9-.9V3.9c0-.5-.5-.9-.9-.9zM11 7h6v2h-6V7zm0 4h6v2h-6v-
            2zm0 4h6v2h-6zM7 7h2v2H7zm0 4h2v2H7zm0 4h2v2H7z"
            fill={props.color ? props.color : colors.primary} />

```

```

        </Svg>
    )
}

export function RepeatIcon(props) {
    return (
        <Svg height={props.height} viewBox="0 0 24 24" width={props.width}
        {...props}>
            <Path d="M0 0h24v24H0z" fill="none" />
            <Path
                d="M7 7h10v3l4-4-4-4v3H5v6h2V7zm10 10H7v-3l-4 4 4 4v-3h12v-
                6h-2v4z"
                fill={props.color ? props.color : colors.primary}
            />
        </Svg>
    )
}

```

```

export function DoneIcon(props) {
    return (
        <Svg height={props.height} viewBox="0 0 24 24" width={props.width}
        {...props}>
            <Path d="M0 0h24v24H0z" fill="none" />
            <Path
                d="M9 16.2L4.8 12l-1.4 1.4L9 19 21 7l-1.4-1.4L9 16.2z"
                fill={props.color ? props.color : colors.primary}
            />
        </Svg>
    )
}

```

```

export function ImportantIcon(props) {
  return (
    <Svg height={props.height} viewBox="0 0 24 24" width={props.width}
    {...props}>
      <Path d="M0 0h24v24H0V0z" fill="none" />
      <Path
        d="M3.5 18.99l11 .01c.67 0 1.27-.33 1.63-.84L20.5 12l-4.37-6.16c-
        .36-.51-.96-.84-1.63-.84l-11 .01L8.34 12 3.5 18.99z"
        fill={props.color ? props.color : colors.primary}
      />
    </Svg>
  )
}

```

```

export function AllTasksIcon(props) {
  return (
    <Svg height={props.height} viewBox="0 0 24 24" width={props.width}
    {...props}>
      <Path d="M0 0h24v24H0V0z" fill="none" />
      <Path
        d="M19 3H5c-1.1 0-2 .9-2 2v7c0 1.1 2 2 2h14c1.1 0 2-.9 2-2V5c0-
        1.1-.9-2-2-2zm0 6h-4c0 1.62-1.38 3-3 3s-3-1.38-3-3H5V5h14v4zm-4 7h6v3c0 1.1-.9
        2-2 2H5c-1.1 0-2-.9-2-2v-3h6c0 1.66 1.34 3 3 3s3-1.34 3-3z"
        fill={props.color ? props.color : colors.primary}
      />
    </Svg>
  )
}

```

Приложения А.4. Styles

Файл colors.js:

```
export const colors = {  
  background: '#f3f3f3',  
  primary: 'rgb(45, 56, 96)',  
  highlight: '#345add',  
  white: '#fff',  
  red: '#f32f2f',  
};
```

Файл styles.js:

```
export const mainStyles = {  
  elevation: 12,  
  borderRadius: 4,  
}
```

Файл typography.js:

```
import { colors } from './colors';  
  
export const h1 = {  
  fontSize: 30,  
  fontFamily: 'Roboto',  
  color: colors.primary,  
}  
  
export const h2 = {  
  fontSize: 24,  
  fontFamily: 'Roboto',  
  color: colors.primary,  
}  
  
export const h3 = {  
  fontSize: 16,  
  fontFamily: 'Roboto',
```

```

        color: colors.primary
    }
export const h4 = {
    fontSize: 14,
    lineHeight: 24,
    fontFamily: 'Roboto',
    color: colors.primary
}
export const h5 = {
    fontSize: 12,
    fontFamily: 'Roboto',
    color: colors.primary
}

```

Приложения A.5. Index.js

```

/**
 * @format
 */

import { gestureHandlerRootHOC } from 'react-native-gesture-handler';
import { Navigation } from "react-native-navigation";
import { Login } from './screens/loginScreen';
import { Main } from './screens/mainScreen';
import { ListPicker } from './components/listPicker';
import { SideMenu } from './screens/sideMenu';
import { ListView } from './screens/listViewer';
import { TaskViewer } from './screens/taskViewer';
import { Options } from './screens/taskViewer';

Login.options = {
    topBar: {

```

```

        visible: false
      },
      statusBar: {
        backgroundColor: '#f3f3f3',
        style: 'dark',
      },
    }
  }

```

```

TaskViewer.options = {
  ...Login.options,
}

```

```

Options.options = {
  ...Login.options,
}

```

```

ListViewer.options = {
  ...Login.options,
  animations: {
    push: {
      content: {
        translationX: {
          from: require('react-native').Dimensions.get('window').width,
          to: 0,
          duration: 300
        }
      }
    }
  }
}

```

```

    }

    Main.options = {
        ...ListViewer.options,
    }

    SideMenu.options = {
        ...Login.options,
        statusBar: {
            backgroundColor: '#fff',
            style: 'dark',
        }
    }
}

Navigation.registerComponent('loginScreen', () => Login);
Navigation.registerComponent('taskViewerScreen', () =>
    gestureHandlerRootHOC(TaskViewer));
Navigation.registerComponent('optionsScreen', () =>
    gestureHandlerRootHOC(Options));
Navigation.registerComponent('listViewerScreen', () =>
    gestureHandlerRootHOC(ListViewer));
Navigation.registerComponent('sideMenu', () =>
    gestureHandlerRootHOC(SideMenu));
Navigation.registerComponent('mainScreen', () =>
    gestureHandlerRootHOC(Main));
Navigation.registerComponent('listPicker', () => ListPicker);
Navigation.events().registerAppLaunchedListener(() => {
    Navigation.setRoot({
        root: {
            stack: {

```

```
children: [  
  {  
    component: {  
      name: 'loginScreen'  
    }  
  }  
]  
}  
}  
});  
});
```

					ПГУ 2.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		141

Приложение Б. Презентация

ТЕМА РАБОТЫ:

«Смарт-органайзер для ОС Андроид»

Выполнил:
студент 16ВВ1 Соловьев А.А.

Научный руководитель:
к.т.н., доцент каф. ВТ ПГУ Егоров В.Ю.

Постановка задачи:

Необходимо реализовать мобильное приложение для операционной системы Android, позволяющее эффективно организовывать личное время, формировать привычки.

Приложение должно иметь следующий функционал:

- Авторизация через Google аккаунт.
- Создание задач с определенным набором свойств.
- Создание списков задач.
- Выполнение, удаление и установка приоритета задачи
- Редактирование списков и параметров задач.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.35.001 ПЗ

Лист

142

Проблема:

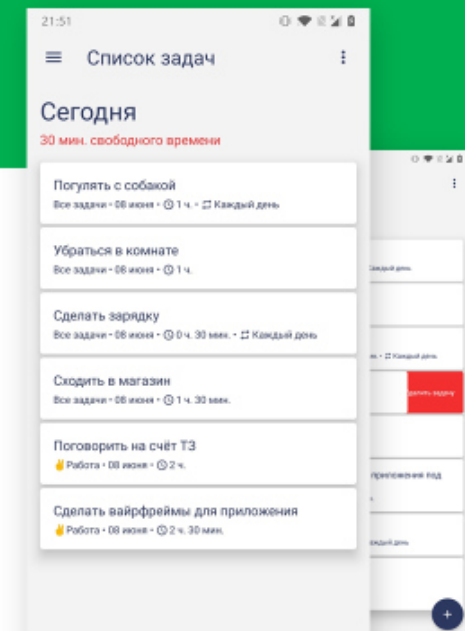
Удержание списка дел в уме даже всего лишь на один день приводит к таким нежелательным последствиям, как:

- забывание некоторых задач;
- потери времени в связи с отсутствием четкого списка дел, мозг старается не вспоминать о неприятных, рутинных делах;
- чувство собственной нереализованности;
- неорганизованность.

Решение:

Решением является смарт-органайзер для мобильного устройства. Он позволит вам:

- эффективно организовать рабочий день;
- не забыть о мелких, нежеланных делах;
- получить удовлетворение от успешного выполнения всех задач;
- планомерно двигаться к глобальным целям путем выполнения подзадач.



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.35.001 ПЗ

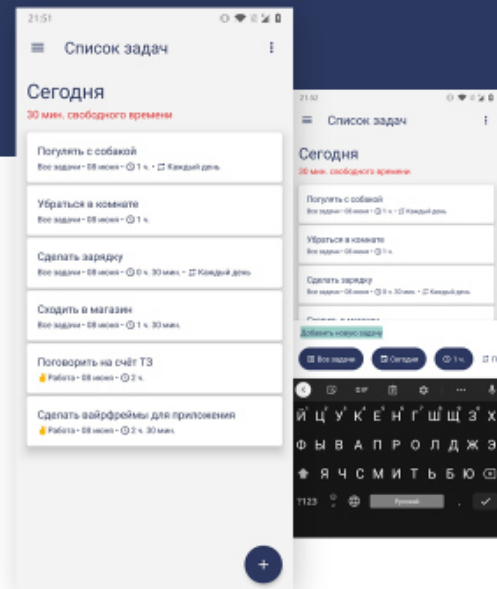
Лист

143

Главный экран:

Главный экран встречает авторизованного пользователя и служит для:

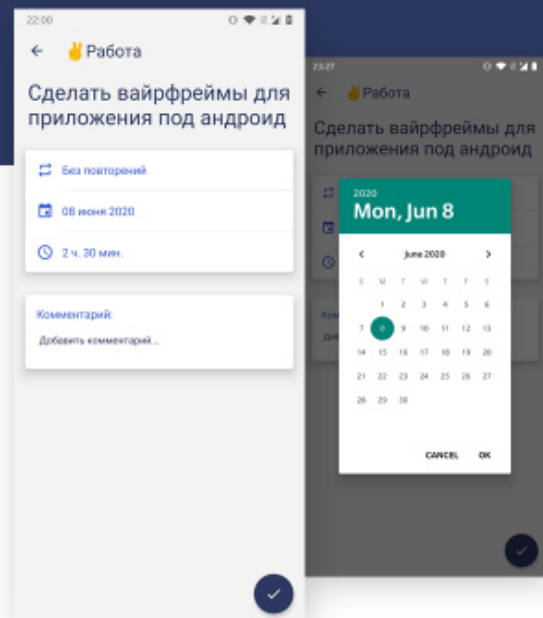
- отображения всех задач на сегодня;
- показа кол-ва свободного на день времени;
- доступа к редактированию задач;
- доступа к созданию задач;
- доступа к меню списков.



Экран настройки задачи:

Открывается при нажатии на задачу в списке и позволяет:

- редактировать свойства задачи;
- добавить заметку к задаче;
- отметить задачу, как выполненную, путем нажатия на кнопку в правом нижнем углу.



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.35.001 ПЗ

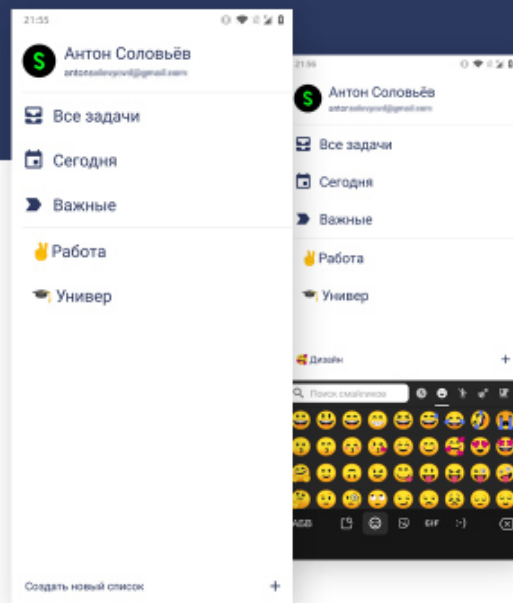
Лист

144

Меню списков задач:

Открывается по нажатию на привычную иконку навигации в левом верхнем углу, либо свайпом.

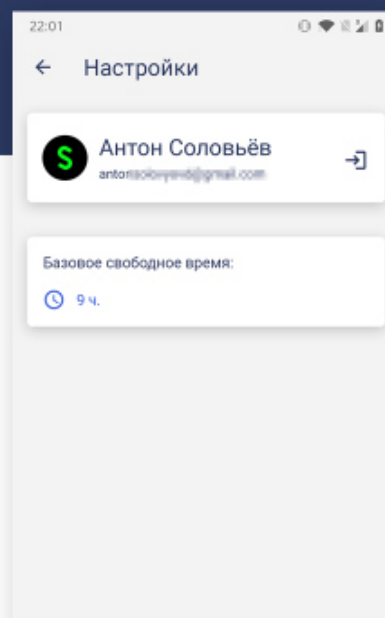
- Отображает все списки задач.
- Позволяет создавать новые списки.
- Открывает необходимый список по нажатию на него.



Экран настроек:

Открывается по нажатию на пользователя в меню и позволяет:

- выйти из аккаунта;
- редактировать базовое кол-во свободного времени;



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.35.001 ПЗ

Лист

145

Реализация мобильного приложения:



Язык программирования: JavaScript



Среда разработки: Visual Studio Code



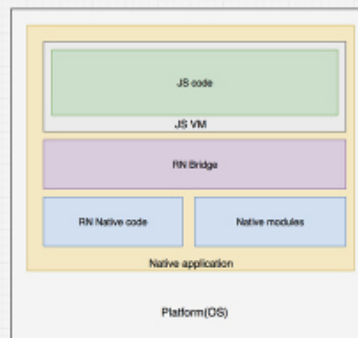
React Native — фреймворк для кроссплатформенной разработки



Firebase: для реализации авторизации и хранения данных пользователей.



Realm DataBase — быстрая open-source no-sql база данных.



Общая структура приложения на React Native

Модули для React Native:



React Native Gesture Handler
— API для упрощения работы с жестами.



React Native Navigation — API для работы с маршрутизацией, рекомендованное разработчиками React Native.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.35.001 ПЗ

Лист

146

Заключение:

Разработано мобильное приложение «Смарт-Органайзер» для ОС Android, предназначенное для организации личного времени, формирования привычек, структурирования рутинных задач.

В будущем планируется повысить производительность и стабильность работы приложения, расширить функционал работы с задачами, доработать интерфейс с точки зрения удобства использования.

Хоть приложение и написано преимущественно под Android, React Native позволяет с легкостью доработать его к релизу на устройства от Apple на базе iOS.

Спасибо за внимание!

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.35.001 ПЗ

Лист

147