МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Специальность: 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения»

Специализация: «Эксплуатация вычислительных машин, комплексов, систем и сетей специального назначения»

ДИПЛОМНЫЙ ПРОЕКТ

на тему

КРОССПЛАТФОРМЕННОЕ ПРИЛОЖЕНИЕ НА БАЗЕ ТЕХНОЛОГИИ ВКОНТАКТЕ *MINI APPS*

Студент _____ 10.06.2022 Лисов А.А.
(подпись, дата)

Руководитель _____ 10.06.2022 Бычков А.С.
(подпись, дата)

Нормоконтролёр _____ 10.06.2022 Бычков А.С.
(подпись, дата)

Рецензент нач. отдела, ф-л ПАО «Россети Волга» - «Пензаэнерго»
_____ _____ 10.06.22 Севастьянов А.В.
(должность, место работы) (подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 2.06.22 № 13)

Заведующий кафедрой _____ М.А. Митрохин
(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 17.06.2022 № 8)

Секретарь ГЭК _____ Е.И. Гурин
(подпись)

Пенза 2022

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«26» апреля 2022 г.

ЗАДАНИЕ
НА ДИПЛОМНЫЙ ПРОЕКТ СПЕЦИАЛИСТА

1. Студент Лисов Андрей Алексеевич, гр.17ВО2 факультета вычислительной техники.

2.Тема работы: «Кроссплатформенное приложение на базе технологии ВКонтакте *Mini Apps*»

Тема утверждена приказом ПГУ №0357/о от "26" апреля 2022 г.

3. Руководитель работы к.т.н., доцент каф. ВТ ПГУ Бычков А.С.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Приложение предназначено для снятия психологической нагрузки операторов автоматизированных систем. В качестве базовых технологий для разработки использовать библиотеку создания пользовательских интерфейсов *React JS*, библиотеку управления состоянием веб приложений *MobX*, набор визуальных компонентов *VKUI* и *API* платформы ВКонтакте *Mini Apps*.

5. Перечень подлежащих разработке вопросов
Требуется произвести анализ предметной области, разработать дизайн приложения, разработать алгоритм работы экранов приложения, разработать логику переходов между экранами, разработать структуру и принцип хранения данных пользователя, отладить работу приложения. Приложение должно предоставлять пользователю возможность выбрать элемент (или несколько элементов) из набора, выдавать данные в соответствии со сделанным выбором, предоставлять возможность сохранить полученную информацию, предоставлять информацию о каждом элементе, предоставлять возможность просматривать сохраненные данные и конфигурировать свой интерфейс (цвета элементов и канваса, логику переходов между экранами и тему всего приложения). Хранить пользовательские настройки и сохраненные данные необходимо в *Storage* (хранилище данных предоставляемое всем пользователям ВКонтакте *Mini Apps*).

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	24	26.04.2022 – 03.05.2022	
2.	Выбор средств и технологий разработки	10	04.05.2022 – 7.05.2022	
3.	Проектирование дизайна приложения	12	8.05.2022 – 10.05.2022	
4.	Разработка архитектуры и логики приложения	12	11.05.2022 – 14.04.2022	
5.	Написание кода приложения	15	15.04.2022 – 19.05.2022	
6.	Отладка и тестирование кода приложения	12	20.05.2022 – 23.05.2022	
8.	Оформление пояснительной записки	15	24.05.2022 – 28.05.2022	

Дата выдачи задания "26" апреля 2022 г.

Руководитель работы специалиста _____ 26.04.22 Бычков А.С.
(подпись, дата)

Задание к исполнению принял студент _____ 26.04.22 Лисов А.А.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ 16.06.2022 Фионова Л.Р.
(подпись, дата)

Реферат

Работа содержит 182 листа, 28 рисунков, 9 таблиц, 11 источников и 3 приложений.

КРОССПЛАТФОРМЕННОЕ ПРИЛОЖЕНИЕ, ВЕБ ПРИЛОЖЕНИЕ, *REACT JS*, ВКОНТАКТЕ *MINI APPS*, *VKUI*, СНИЖЕНИЕ СТРЕССА, ЭМОЦИОНАЛЬНОЕ ВЫГОРАНИЕ.

Цель дипломного проекта - разработка кроссплатформенного приложения на базе технологии ВКонтакте *Mini Apps* для снятия психологической нагрузки операторов автоматизированных систем.

Объектом разработки является клиентское приложение для снятия психологической нагрузки операторов автоматизированных систем на базе платформы ВКонтакте *Mini Apps*.

В результате проделанной работы разработано приложение, полностью соответствующее поставленной задаче, а именно: разработано клиентское приложение позволяющее выбрать элемент (или несколько элементов) из набора, выдающее данные в соответствии со сделанным выбором, предоставляющее возможность сохранить полученную информацию, предоставляющее информацию о каждом элементе, предоставляющее возможность просматривать сохраненные данные и конфигурировать собственный интерфейс (цвета элементов и канваса, логику переходов между экранами и тему всего приложения). Все пользовательские настройки и сохраненные данные хранятся в *Storage*. Приложение было размещено на платформе ВКонтакте *Mini Apps*.

Содержание

Перечень сокращений и обозначений 5

Введение 6

1 Кроссплатформенное приложение на базе платформы ВКонтакте Mini Apps для снижения эмоционального напряжения 9

1.1 Назначение, область применения и принципы работы с приложением . 9

1.1.1 Основные концепции приложения 11

1.2 Проектирование приложения 12

1.2.1 Роутинг приложения 12

1.2.2 Состояние приложения 13

1.2.3 Вспомогательные функции приложения 25

1.2.3.1 Функция *dragAndDrop* 25

1.2.3.2 Функция *getPosition* 30

1.2.4 Сервисы приложения 31

1.2.4.1 Сервис *loadPredictions* 31

1.2.4.2 Сервис *savePrediction* 33

1.2.5 Основные модули приложения 34

1.2.5.1 Планировка приложения 35

1.2.5.2 Панель «Мой профиль» 36

1.2.5.3 Панель «Мои вопросы» 38

1.2.5.4 Панели «Результат» и «Ответ» 40

1.2.5.5 Панель «Расклады» 44

1.2.5.6 Основная панель 45

1.2.5.7 Панель «Описание» 48

1.2.5.8 Панель информации об элементе 49

1.2.5.9 Панель «Настройки» 50

1.2.5.10 Панель «Ошибка» 54

2 Используемые технологии разработки 57

2.1 Языки программирования *JavaScript* (*TypeScript*) 57
2.2 Одностраничные приложения 58
2.3 Сборщик *Webpack* 59
2.4 Библиотека *React JS* 59
2.5 Библиотека *MobX* 62
3 Экономическое обоснование дипломного проекта 64
3.1 Описание организации 64
3.2 Анализ объекта исследования 65
3.3 Определение экономической эффективности 66
3.4 Методы повышения экономической эффективности 70
4 Безопасность жизнедеятельности 72
4.1 Микроклимат рабочего помещения 73
4.2 Освещение рабочего места оператора 73
4.3 Эргономические нормы при работе с компьютером 78
4.3.1 Размещение компьютеров в помещении 79
Заключение 80
Приложение А – *UML*-диаграммы 82
Приложение Б – Листинг программы 88
Приложение В – Презентация 174

Перечень сокращений и обозначений

API (*Application Programming Interface*) – программный интерфейс приложения

HTML (*Hyper Text Markup Language*) – язык гипертекстовой разметки

IT (*Information Technology*) – информационные технологии

JS – язык программирования JavaScript

MobX – библиотека управления состоянием приложения

React JS – библиотека разработки пользовательских интерфейсов веб приложений

UML (*Unified Modeling Language*) – унифицирован язык моделирования

Webpack – сборщик модулей веб приложений

ВКонтакте *Mini Apps* – открытая платформа для публикации мини приложений

ПО – программное обеспечение

ЭВМ – электронно-вычислительная машина

Введение

Сфера информационных технологий — одно из самых динамично развивающихся направлений. Огромное количество специалистов задействовано в разработке и использовании программного обеспечения. Для большинства операторов автоматизированных систем характерна многочасовая работа над поставленной задачей без перерывов на иные виды активности. Первые несколько месяцев организм оператора не испытывает неудобств, но со временем такой режим работы может привести к серьезным проблемам со здоровьем и не только физическим, но и психологическим. В процессе работы оператор задействует аналитическое мышление, систематизирует информацию, устанавливает причинно-следственные связи, сравнивает и так далее. Эти процессы задействуют лобную долю мозга, значительно нагружая нервную систему. По уровню психических процессов мышление делится на аналитическое и интуитивное. Американский психолог Даниэль Канеман отмечал, что интуиция — это способность автоматической выработки решений без длительных логических рассуждений или доказательств [1].

Даже кратковременный перерыв в работе, позволяющий переключиться с аналитического на интуитивное мышление, способствует значительному повышению эффективности работы оператора. Для этой цели создано множество приложений, позволяющих ненадолго отвлечься от дел, переключив внимание на менее сложную активность. Примером таких приложений является всем известные пасьянс «Паук», «Сапер», «Зума» и т.п.

Приложение, разработанное в ходе данной дипломной, работы является актуальным для пользователей, которые много часов беспрерывно проводят, работая за компьютером. Благодаря простоте использования и удобному современному интерфейсу, данное приложение будет полезным для кратковременного переключения внимания оператора с рабочих активностей на развлекательные.

Объектом исследования является борьба с эмоциональным выгоранием операторов автоматизированных систем с помощью современных технологий.

Предметом исследования является разработка кроссплатформенных веб приложений на базе реактивных технологий разработки и технологии ВКонтакте *Mini Apps* для борьбы с эмоциональным выгоранием операторов автоматизированных систем.

Целью работы является создание приложения для платформы ВКонтакте *Mini Apps* для снятия психологического напряжения оператора автоматизированных систем.

Задачи, необходимые для достижения цели работы:

- изучить причины эмоционального выгорания операторов автоматизированных систем;
- спроектировать приложение, предназначенное для снятия психологического напряжения операторов автоматизированных систем;
- разработать дизайн приложения с использованием компонентов из набора *VKUI*;
- разработать приложение с использованием реактивных технологий веб разработки (с помощью библиотек *React JS* и *MobX*) и *API* ВКонтакте;
- разместить приложение на платформе ВКонтакте *Mini Apps*.

Предполагается, что после разработки и публикации приложения значительно упростится процесс снятия психологического напряжения операторов автоматизированных систем, являющихся пользователями социальной сети ВКонтакте.

Практическая значимость данной работы заключается в том, что для операторов автоматизированных систем, являющихся пользователями ВКонтакте теперь существует приложение, предоставляющее доступный и удобный способ снятия психологического напряжения. Кроме того, разработка подобных приложений – один из самых эффективных способов

развития экосистемы сервисов ВКонтакте. Это особенно актуально сегодня, когда, в связи с внешнеполитической обстановкой, были заблокированы многие популярные западные социальные сети и был взят курс на информационное импортозамещение.

Введение раскрывает актуальность, определяет объект, предмет, цель, задачи и практическую значимость дипломной работы.

1 Кроссплатформенное приложение на базе технологии ВКонтакте *Mini Apps* для снижения эмоционального напряжения

1.1 Назначение, область применения и принципы работы с приложением

Целью данной дипломной работы является разработка кроссплатформенного приложения на базе технологии ВКонтакте *Mini Apps* для снятия эмоционального напряжения операторов автоматизированных систем путем гадания на рунах.

В 2019 году Всемирная организация здравоохранения определила выгорание как «синдром, концептуализированный как результат хронического стресса на рабочем месте, с которым не удалось справиться».

Согласно этому определению, люди, страдающие выгоранием, испытывают чувство истощения, увеличение умственной дистанции, демотивацию и цинизм по отношению к своей работе, а также снижение профессиональной эффективности. Также важно отметить, что эмоциональное выгорание по определению ВОЗ вызывается исключительно стрессорами на рабочем месте.

Факторы, ведущие к выгоранию, делятся на две категории: внутренние и внешние. Внутренние факторы присущи человеку, переживающему выгорание и обычно связанны с его личностными чертами. Вот несколько примеров внутренних черт, которые могут вызвать «активное выгорание»:

- завышенные ожидания от себя;
- сильная потребность в признании;
- потребность угождать другим людям, подавляя при этом свои собственные потребности;
- стремление к полному контролю и невозможность делегировать;
- чрезмерное посвящение себя работе;

- отношение к работе как к единственно значимой деятельности и замена работой социальной жизни [2].

Разработанное приложение ориентировано на борьбу с тремя последними факторами – стремление полностью контролировать ситуацию на проекте, чрезмерное посвящение себя работе и отношение к работе как к единственно значимой деятельности. Основная идея заключается в переключении внимания оператора с рабочего процесса на отвлеченное занятие. Для этой цели создано уже немало приложений. Как правило они предоставляют пользователю возможность отвлечься и расслабиться за счет медитативных практик, таких как глубокое дыхание и прослушивание успокаивающих звуков (звуки природы или специальные мелодии для медитации). Однако в процессе работы под давлением горящих сроков и завышенных ожиданий оператор зачастую не находит в себе сил, чтобы прерваться на прослушивание музыки или для выполнения дыхательной гимнастики. Полностью погруженный в работу, он стремительно, и сам того не замечая, истощает ресурсы своей нервной системы. Аналитическое мышление перестает работать, снижается эффективность труда, но ощущение, что решение проблемы будет вот-вот достигнуто заставляет оператора продолжать, не делая перерывов. Итогом такой работы редко бывает хороший результат. Как правило, она заканчивается только головными болями и депрессивным состоянием, которые приводят к бессоннице, потере аппетита и еще большему снижению продуктивности. Получается замкнутый круг, который в конечном итоге приводит к полной потере мотивации и эмоциональному выгоранию оператора.

Основная идея разработанного приложения заключается в том, что работа с ним требует минимального количества времени оператора, но при этом является увлекательным процессом, способным привлечь его внимание и подтолкнуть к более частому использованию приложения. Таким образом,

отвлечься на кратковременное взаимодействие с данным приложением намного проще, чем оторваться от работы для медитации.

Так как человеческие интересы различаются, невозможно создать универсальное приложение, привлекательное для всех операторов автоматизированных систем. Единственно возможный путь решения проблемы эмоционального выгорания для всех операторов автоматизированных систем – разработка большого количества приложений с различными тематиками. Логика работы с данными приложениями не должна требовать от оператора нагрузки аналитического мышления. Приложение должно быть простым, доступным на любых платформах и интересным в использовании. Платформа ВКонтакте *Mini Apps* прекрасно подходит для реализации этой идеи.

В качестве тематики для первого приложения с опорой на собственные интересы было принято решение использовать работу с рунами. Это интересный процесс, не требующий умственного напряжения оператора и позволяющий отвлечься от мыслей о рабочем процессе.

1.1.1 Основные концепции приложения

Для реализации данной задачи было разработано веб приложение. Веб приложения обладают рядом преимуществ перед обычными настольными и мобильными приложениями. Это возможность использовать одно приложение на различных платформах без необходимости внесения серьезных изменений в его структуру, возможность пользоваться приложением без установки – приложение доступно в браузере и в приложение ВКонтакте, а также быстрое распространение приложения по сети – для того, чтобы начать пользоваться приложением достаточно перейти по нужной ссылке.

Традиционно веб разработка используется для создания информационных сайтов, призванных познакомить пользователя с ассортиментом магазина или с новой научной статьей, однако с развитием

технологий разработки, появилась возможность создания быстрых и динамичных приложений, работающих в браузере.

1.2 Проектирование приложения

1.2.1 Роутинг приложения

Приложение построено на базе компонентов *VKUI*. Данная библиотека наполнена компонентами, дизайн которых соответствует дизайну основного приложения ВКонтакте. Благодаря этому, приложения, разработанные с помощью данного набора, выглядят как часть главного приложения ВКонтакте. Библиотека не предоставляет компонентов для реализации декларативного роутинга в приложении, чтобы не диктовать разработчикам способы его организации.

Для переключения между экранами можно воспользоваться различными подходами. Наиболее простой способ для этого – выделить поле в хранилище состояния приложения и вносить в него идентификатор текущего экрана. Этот подход легко реализуется и легко масштабируется при увеличении количества экранов, но имеет ряд недостатков. Одной из главных проблем данного подхода является то, что при работе приложения на мобильном устройстве кнопка «Назад», (существующая почти на всех устройствах на базе операционной системы *Android*), будет приводить к выходу из приложения, тогда как ожидаемое поведение при ее нажатии – переход на предыдущий экран. Кроме того, использование такого подхода лишает разработчика возможности пользоваться преимуществами традиционного роутинга, который автоматически сохраняет историю переходов между страницами, а также позволяет передавать параметры страницы в адресной строке. Для этих целей командой разработчиков ВКонтакте была создана библиотека *@happysanta/router*. Данная библиотека заточена под работу с ВКонтакте *Mini Apps* и позволяет, используя декларативные методы роутинга организовать переходы между экранами,

панелями и модальными окнами приложения [3]. Для данного приложения используются только возможности перехода между экранами и панелями, так как количество модальных окон невелико и взаимодействие с ними может быть без труда организовано с помощью хранилища состояния.

Архитектурные правила построения приложений, установленные набором компонентов *VKUI,* требуют размещение в каждом экране хотя бы одной панели.

1.2.2 Состояние приложения

Состояние приложения управляется с помощью библиотеки *MobX*. Данная библиотека позволяет очень гибко управлять состоянием и предоставляет различные варианты для его организации. Функция библиотеки *makeObservable* принимает в качестве параметра любой объект и конвертирует его свойства в наблюдаемые с помощью механизма аннотаций (разработчик может сам указать какие свойства объекта необходимо сделать наблюдаемыми, а какие нет). Механизм работы наблюдаемых свойств заключается в проксировании обращений к ним. Таким образом, когда код приложения обращается к наблюдаемому свойству, меняя его значение, компоненты, которые обращаются к этому же свойству для получения его значения будут уведомлены о его изменении и отреагируют на него определенным образом. Для взаимодействия этого механизма с компонентами *React JS*, *MobX* предоставляет функцию высшего порядка observer, которая наблюдает за свойствами, передаваемыми компоненту, и выполняет перерисовку компонента, если значения свойств являются наблюдаемыми и были подвергнуты изменению.

Наиболее масштабируемым вариантом организации хранилища состояния на мой взгляд является использование класса. Класс легко поддерживать даже при значительном увеличении его размеров, а также механизм наследования классов удобнее механизма наследования объектов,

кроме того, при необходимости, можно переиспользовать класс для других модулей приложения.

Так как размер приложения не очень велик, для управления состоянием подойдет паттерн проектирования «Одиночка» («*Singletone*»). Принцип данного подхода к разработке заключается в централизованном хранении всех данных приложения в одном объекте. Преимуществами использования данного паттерна является его простота (нет необходимости реализовывать сложные механизмы взаимодействия между модулями приложения, так как все их данные хранятся в одном месте) и повышение надежности приложения (благодаря наличию единственного источника истины) [4]. При необходимости хранилище всегда можно разбить на меньшие классы и обеспечить их взаимодействие внутри компонентов или прямо в основном классе хранилища (путем передачи объекта главного класса в экземпляр другого через конструктор или путем создания меньших классов внутри конструктора основного класса).

Поля класса хранилища состояния приложения описаны в таблице 1.

UML-диаграмма класса состояния приложения приведена в приложении А.3.

Таблица 1 – Поля класса состояния приложения

Название поля	Тип (интерфейс) поля	Описание поля
user	*{* *id: number;* *first_name: string;* *photo_200: string;* *}*	объект, содержащий информацию о пользователе, загружаемую из пользовательского профиля ВКонтакте; содержит идентификатор

Название поля	Тип (интерфейс) поля	Описание поля
		пользователя, его имя и ссылку на его фотографию; наблюдаемое поле;
isMobile	*boolean*	флаг, определяющий текущий размер экрана; наблюдаемое поле;
isMobilePlatform	*boolean*	флаг, определяющий, используется ли на текущей платформе мышь или тачпад; наблюдаемое поле;
predictionPage	*number*	номер страницы списка сохраненных вопросов на экране «Мои вопросы»; наблюдаемое поле;
predictionTitle	*string*	поле для временного хранения вопроса; наблюдаемое поле;
appLoading	*{* *isSettingsLoading: boolean;* *isPredictionsLoading: boolean;* *isUserProfileLoading: boolean;* *isPredictionSavingLoading: boolean;*	объект, хранящий состояние загрузки различных модулей приложения: загрузка настроек приложения, загрузка сохраненных вопросов, загрузка данных пользователя,

Название поля	Тип (интерфейс) поля	Описание поля
	isPredictionDeletingLoading: boolean *}*	сохранение вопроса, удаление вопроса; наблюдаемое поле;
settings	*{* *runeColor: string;* *canvasColor: string;* *shouldShowResult: boolean;* *isDarkTheme: boolean;* *shouldHideRuneLabel: boolean* *}*	объект настроек; содержит информацию о цвете элемента, цвете полотна, выбранной теме, логики отображения элементов при игре и логики перехода к экрану «Результат» после выбора; наблюдаемое поле;
tempSettings	*{* *runeColor: string;* *canvasColor: string;* *shouldShowResult: boolean;* *isDarkTheme: boolean;* *shouldHideRuneLabel: boolean* *}*	объект временных настроек приложения, использующий тот же интерфейс, что и поле *settings*, и применяющийся для предпоказа настроек пользователя на экране «Настройки»; наблюдаемое поле;
isCanvasRefreshed	*boolean*	вспомогательное поле логического типа, хранящее информацию о состоянии обновления

Название поля	Тип (интерфейс) поля	Описание поля
		канваса; необходимо для того, чтобы при повторном переходе к основной панели элементы размещались по позициям без анимации, так как их позиции остались прежними после первого распределения; наблюдаемое поле;
openedRunes	*string[]*	массив с идентификаторами открытых элементов; наполняется при выборе элемента на основном экране; наблюдаемое поле;
currentPrediction	*{ date: number; title: string; runes: string[] }*	объект, содержащий выбранный на панели «Мои вопросы» вопрос; содержит дату вопроса, сам вопрос и элементы, выбранные при игре; наблюдаемое поле;
currentRune	*string*	элемент, выбранный на панели «Описание»; наблюдаемое поле;

Название поля	Тип (интерфейс) поля	Описание поля
closedRunes	*string[]*	массив элементов, сброшенных после выбора элементов; наблюдаемое поле;
randomRunes	*{* *name: string;* *isInverted: boolean;* *order: number;* *x: number;* *y: number;* *rotation: number* *}[]*	набор элементов со случайно заданными координатами (*x*, *y*) и поворотом (*rotation*), а также с заданным порядком *order*, определяющем размещение элементов по оси *Z* и свойством *isInverted* (определяющим положение элемента); наблюдаемое поле;
wasResultShown	*boolean*	вспомогательное поле для определения логики взаимодействия основной панели и модального окна подтверждения перехода к панели «Результат»; наблюдаемое поле;
isPredictionSaved	*boolean*	поле, определяющее был ли сохранен вопрос и используемое для

Название поля	Тип (интерфейс) поля	Описание поля
		предотвращения повторного сохранения вопроса; наблюдаемое поле;
wereSettingsSaved	*boolean*	определяет были ли сохранены настройки, используется для состояния блокировки кнопок «Сохранить» и «Отмена» на панели «Настройки»; наблюдаемое поле;
wereSettingsChanged	*boolean*	определяет были ли изменены настройки, используется для состояния блокировки кнопок «Сохранить» и «Отмена» на панели «Настройки»; наблюдаемое поле;
isDefaultSettings	*boolean*	определяет соответствие настроек настройкам по умолчанию, используется для состояния блокировки кнопки «Сбросить»; наблюдаемое поле;
userPredictions	*{*	массив сохраненных

Название поля	Тип (интерфейс) поля	Описание поля
	date: number; *title: string;* *runes: string[]* *key: string* *}[]*	вопросов, загруженных из *Storage*; наблюдаемое поле; интерфейс расширяет интерфейс поля *currentPrediction*, дополняя его свойством *key*, содержащим ключ вопроса в *Storage*; наблюдаемое поле;
canvas	*HTMLDivElement*	*HTML* элемент полотна, необходимый для расчета позиций элементов;
divinationType	*string*	тип расклада («1», «3» или «4»); наблюдаемое поле;

Методы класса состояния приложения описаны в таблице 2.

Таблица 2 – методы класса состояния приложения

Название	Параметры	Назначение метода
setRandomRune	*newRune: {* *x: number;* *y: number;* *name: string* *}*	изменяет поля «*x*» и «*y*» объекта с полем name равным *newRune.name* в массиве *randomRunes* на значения *newRune.x* и *newRune.y*
openRune	*rune: string*	заносит значение rune в

Название	Параметры	Назначение метода
		массив *openedRunes*, если там такого значения нет и извлекает значение rune из массива *closedRunes*, если оно там есть
openPrediction	*prediction: {* *date: number;* *title: string;* *runes: string[]* *}*	меняет значение поля *currentPrediction* на значение параметра prediction
mergeRandomRunes	*runes: {* *name: string;* *isInverted:* *boolean;* *order: number;* *x: number;* *y: number;* *rotation:* *number* *}[]*	заменяет элементы в массиве *randomRunes* на переданные в параметре runes
closeRune	*rune: string*	заносит значение rune в массив *closedRunes*, если там такого значения нет и извлекает значение rune из массива *openedRunes*, если оно там есть и вызывает метод *mergeRandomRunes*

Название	Параметры	Назначение метода
getRandomRunes	*runes: string[]*	определяет случайные позиции элементов для переданного набора в пределах полотна (канваса)
refresh	*isForce: boolean*	выполняет перемешивание элементов
setIsPredictionSaved	*isSaved: boolean*	заносит в поле *isPredictionSaved* значение параметра *isSaved*
setWasResultShown	*wasShown: boolean*	заносит в поле *wasResultShown* значение параметра *wasShown*
setPredictionPage	*page: number*	заносит в поле *predictionPage* значение параметра *page*
loadUserPredctionsPage	*page: number*	выполняет вызов сервиса загрузки страницы сохраненных вопросов и возвращает true при удачном завершении и false при ошибке
loadUserPredictions		выполняет вызов метода *loadUserPredctionsPage* до тех пор, пока он не выполнится без ошибок

Название	Параметры	Назначение метода
loadUserInfo		загружает информацию о пользователе
deleteUserPrediction	*predictionKey: string*	удаляет сохраненный вопрос по ключу
setPredictionTitle	*title: string*	заносит в поле *predictionTitle* значение параметра *title*
setActiveModal	*modal?: string*	заносит в поле *activeModal* значение параметра *modal*
setCurrentRune	*rune: string*	заносит в поле *currentRune* значение параметра *rune*
setDivinationType	*type: string*	заносит в поле *divinationType* значение параметра *type*
saveSettings		сохраняет обновленные настройки приложения в *Storage* и обновляет значение полей *isDefaultSettings*, *wereSettingsChanged*, *wereSettingsSaved* и *settings*
setTempSettings	*value: string |* *{* *h: number;* *s: number;*	устанавливает значение поля объекта *tempSettings*, заданного параметром *field* в

Название	Параметры	Назначение метода
	l: number; *} |* *boolean,* *field: string*	соответствии со значением параметра *value*
resetAllSettings		сбрасывает пользовательские настройки приложения к настройкам по умолчанию и обновляет значение полей *isDefaultSettings*, *wereSettingsChanged*, *wereSettingsSaved* и *tempSettings*
resetTempSettings		сбрасывает пользовательские настройки приложения к предыдущему состоянию и обновляет значение полей *isDefaultSettings*, *wereSettingsChanged*, *wereSettingsSaved* и *tempSettings*
reloadAll		выполняет повторную загрузку всех внешних ресурсов, вызывая методы *loadUserInfo*, *loadUserSettings* и

Название	Параметры	Назначение метода
		loadUserPredictions
saveUserPrediction	*predictionProps: {* *runes: string [];* *title: string;* *}*	сохраняет вопрос пользователя в *Storage*

1.2.3 Вспомогательные функции приложения

1.2.3.1 Функция *dragAndDrop*

Назначение функции – перемещение *HTML* блоков по экрану путем перетаскивания с помощью мыши или тачпада. На вход функция получает объект с интерфейсом:

- *element*: *HTMLElement* – перемещаемый *HTML* элемент

- *container*: *HTMLDivElement* – блок, внутри которого выполняется перемещение элемента;

- *setIsMoving*: *(isMoving: boolean) => void* – функция, которая будет вызываться при перемещении элемента – нужна, чтобы различать событие клика и событие поднятия клавиши мыши (или пальца от тачпада) после перетаскивания элемента;

- *setRandomRune*: *(x: number, y: number) => void* – функция, которая будет вызываться после перемещения элемента для обновления его координат.

Схема программы функции *dragAndDrop* приведена на рисунке 1.

Как видно из схемы программы, представленной на рисунке 1 функция dragAndDrop добавляет обработчики событий к перетаскиваемому элементу. Обработчик нажатия на элемент срабатывает при начале его перетаскивания и нужен для инициализации обработчика перемещения указателя мыши.

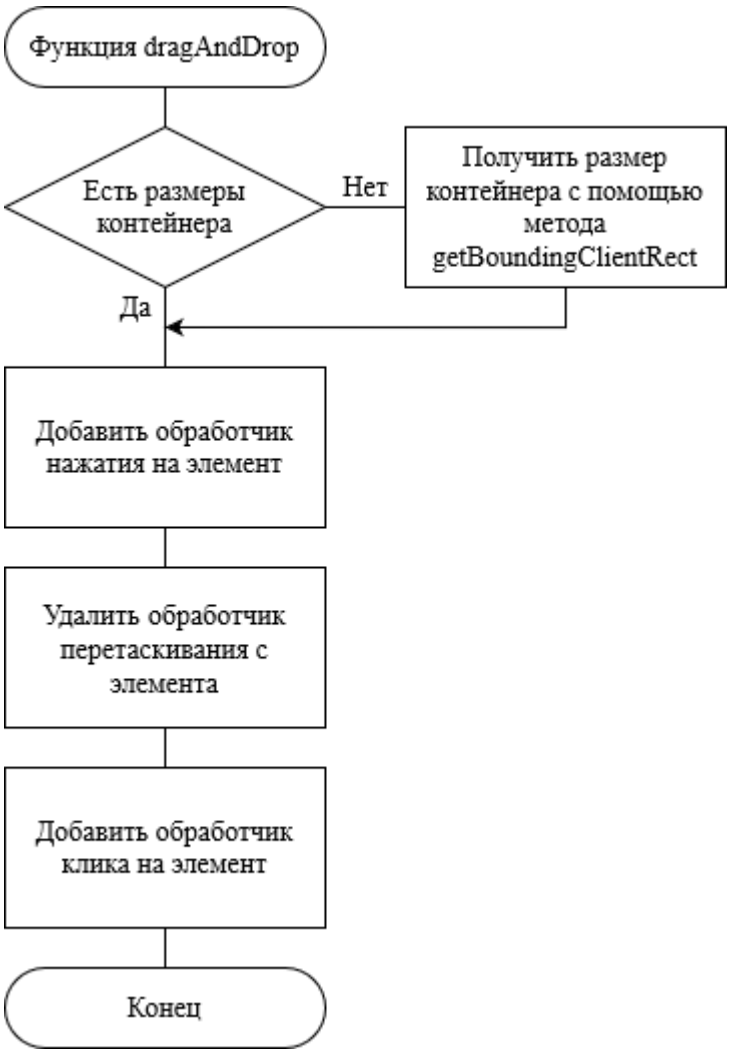


Рисунок 1 – Схема программы функции *dragAndDrop*

Схема программы обработчика нажатия на элемент представлена на рисунке 2.

Рисунок 2 – Схема программы обработчика нажатия на элемент

Обработчик клика на элемент срабатывает после окончания перетаскивания и нужен для предотвращения поведения по умолчанию при клике на элемент. Схема программы обработчика представлена на рисунке 3.

Рисунок 3 – Схема программы обработчика клика по элементу

Обработчик нажатия на элемент добавляет ко всей веб странице обработчик перемещения мыши. Целевым объектом обработчика является весь веб документ, так как при перемещении мыши пользователь может выйти за границы контейнера, внутри которого происходит перемещение, однако это не должно приводить к отключению обработчика – элемент должен по-прежнему перемещаться внутри контейнера, но только по оси, координаты элемента на которой, не достигли координат границы контейнера. Схема программы обработчика перемещения элемента отражена на рисунке 4.

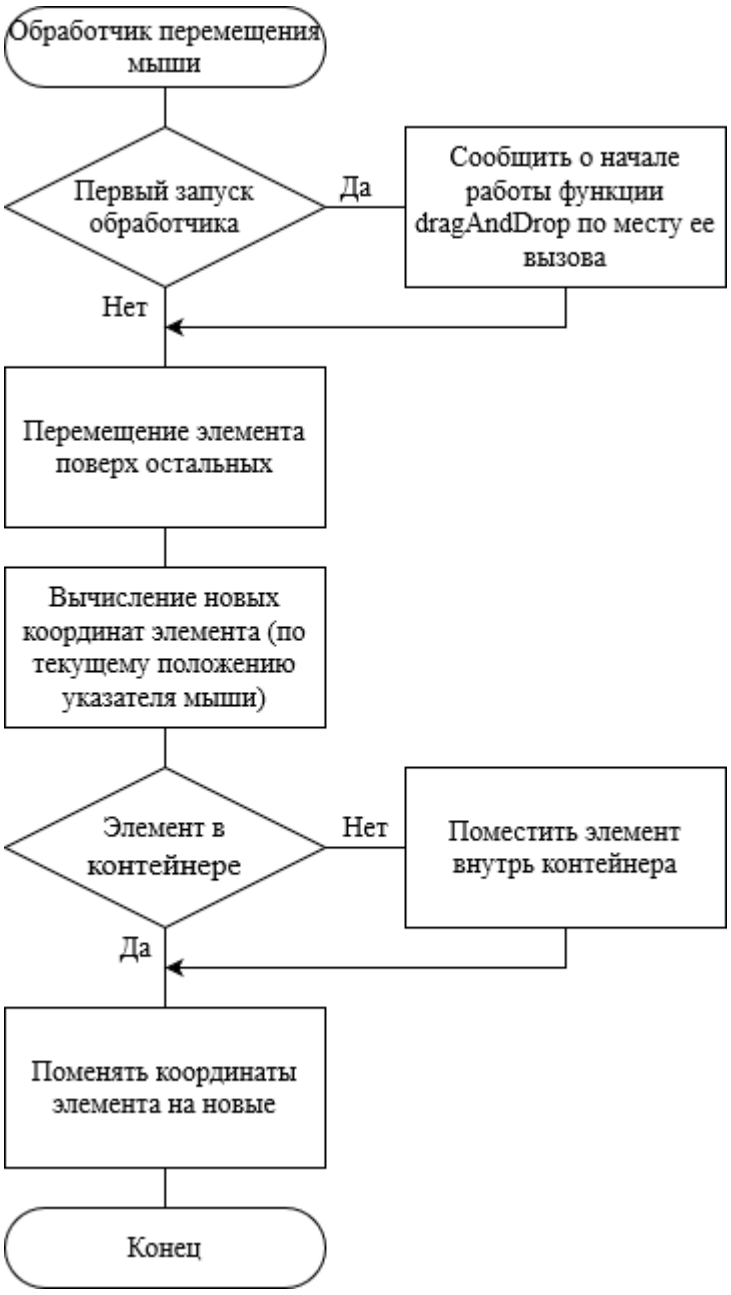


Рисунок 4 – Схема программы обработчика перемещения курсора мыши

Также к документу добавляется обработчик отжимания кнопки мыши, нужный для прекращения перемещения элемента. Благодаря привязке обработчика ко всей веб странице, перемещение будет остановлено независимо от того, где пользователь отожмет кнопку мыши. Схема программы обработчика отжимания кнопки мыши отражена на рисунке 5.

Рисунок 5 – Схема программы обработчика отжимания клавиши мыши

1.2.3.2 Функция *getPosition*

Назначение функции – получение случайно сгенерированных координат и угла вращения для элемента в пределах заданного контейнера. На вход функции передается число *offset*, определяющий отступ от границ контейнера, и объект *container*, содержащий поля *width*: *number* и *height*: number, определяющие размеры контейнера.

Схема программы функции *getPosition* приведена на рисунке 6.

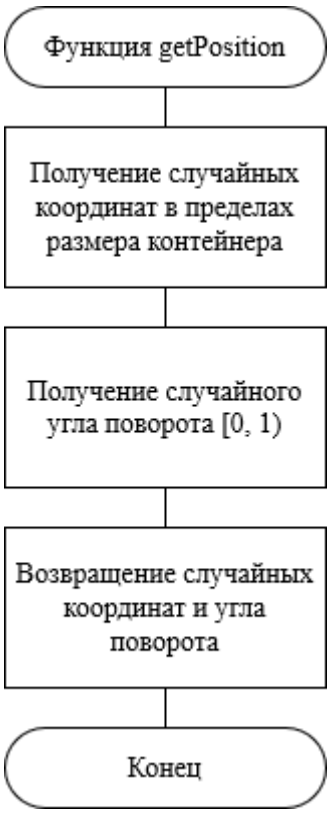


Рисунок 6 – Схема программы функции *getPosition*

1.2.4 Сервисы приложения

Сервисы – это функции, задача которых обеспечить взаимодействие с серверным *API* (в данном случае с *API* ВКонтакте *Mini Apps*).

1.2.4.1 Сервис *loadPredictions*

Данный сервис используется для загрузки сохраненных вопросов пользователя. В качестве параметра он получает номер страницы, которую необходимо загрузить (*page: number*). Так как время взаимодействия с сервером вызывает самые длительные задержки в веб приложениях, необходимо стремиться к максимальному его сокращению. Так как для хранения всех ключей в *Storage* используется общая коллекция нет никакой возможности получить данные конкретных сущностей (сохраненных вопросов в данном случае). Чтобы не загружать все сохраненные данные сервис сначала получает все доступные в *Storage* ключи. Для ключей сохраненных вопросов был определен формат *PREDICTION_<Дата сохранения>*. Из полученного списка выбираются

ключи, соответствующие данному формату. Затем они сортируются по убыванию даты и из них выбирается десять ключей с учетом смещения от начала списка, зависящего от номера страницы. По полученным в результате ключам запрашиваются значения, которые возвращаются из сервиса.

Схема программы функции представлена на рисунке 7.

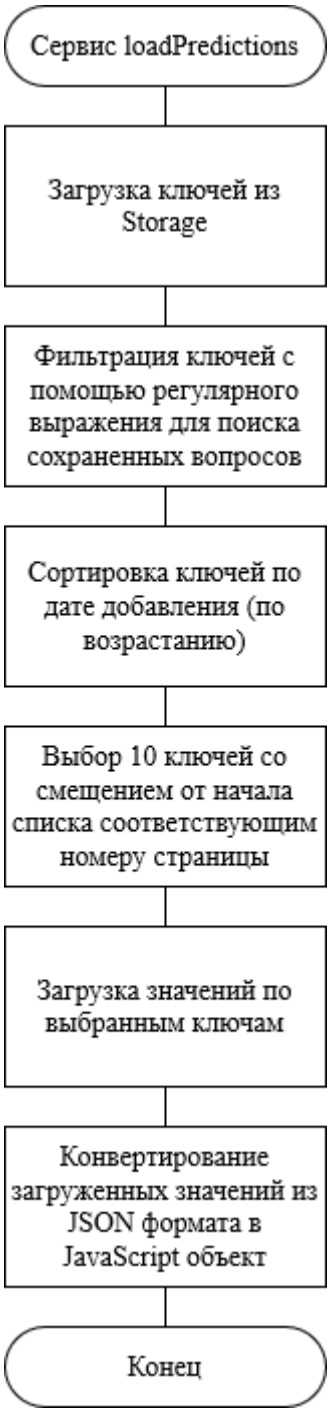


Рисунок 7 – Схема программы сервиса *loadPrediction*

1.2.4.2 Сервис *savePrediction*

Данный сервис используется для сохранения вопросов пользователя. В качестве параметра он получает объект с данными вопроса, который необходимо сохранить. *UML*-диаграмма интерфейса объекта параметров сервиса представлена на рисунке 8.

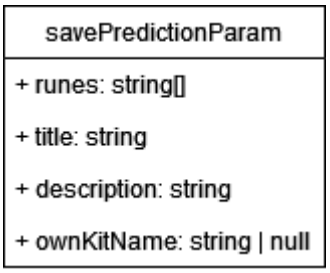


Рисунок 8 – *UML*-диаграмма класса параметров сервиса *savePrediction*

Так как количество переменных хранящихся в *Storage* ограничено, перед сохранением сервис проверяет текущее количество сохраненных вопросов и сохраняет новый только в случае, если оно меньше 800. Схема программы сервиса приведена на рисунке 9.

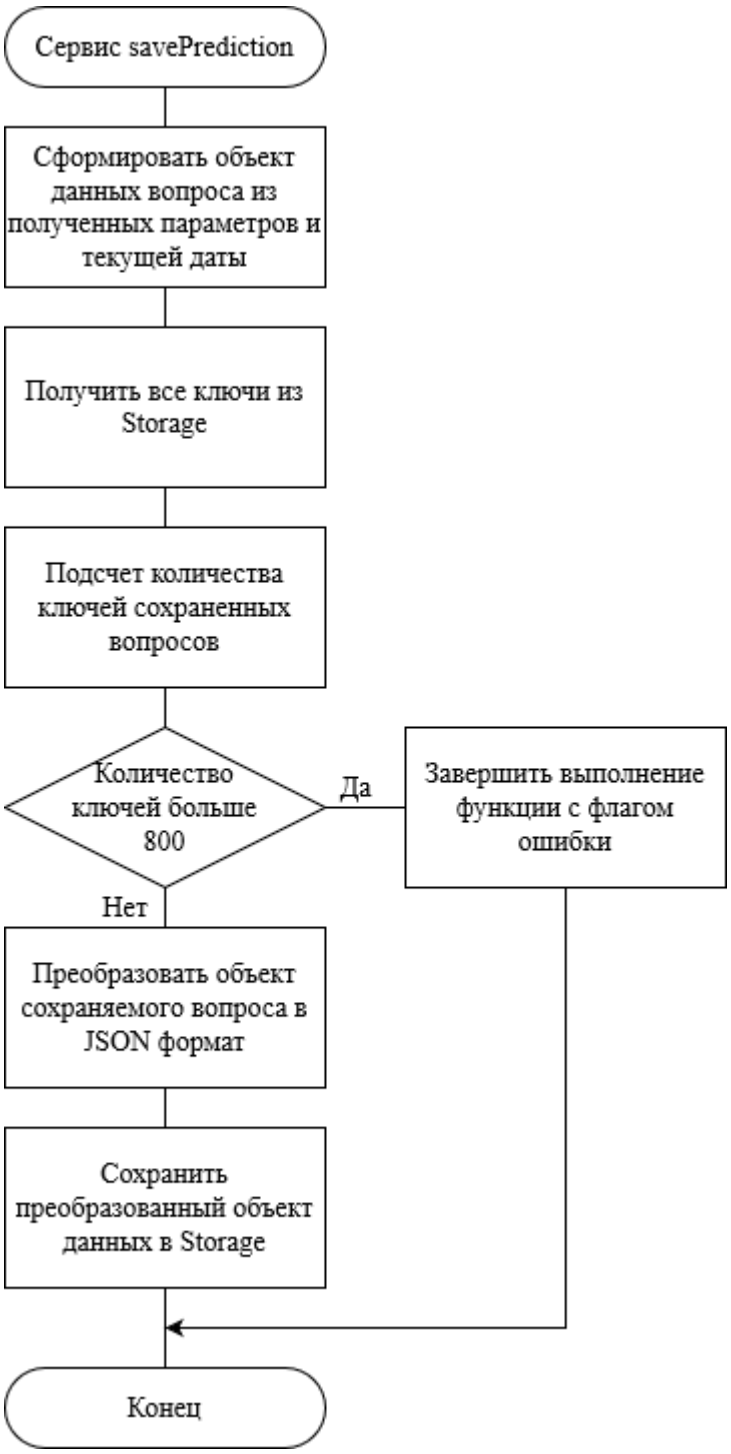


Рисунок 9 – Схема программы сервиса *savePrediction*

1.2.5 Основные модули приложения

Для взаимодействия с *API* ВКонтакте и *API* устройства используется библиотека *BridgePlus*. Она является оберткой над официальной библиотекой *VKBridge* и предоставляет декларативный интерфейс для работы с ВКонтакте *API* и со *Storage*. Для избегания проблем с кодировкой, значения *Storage*

должны храниться в формате *BASE64*. Для обеспечения этого, а также для повышения качества типизации методов библиотеки, был написан класс-адаптер *Bridge*, предоставляющий методы-обертки для некоторых методов работы со *Storage*. Класс содержит два специальных метода *_stringToBase64* и *_base64ToString* для конвертации строки из формата *UTF-8* в формат *BASE64* и наоборот, соответственно. При сохранении нового значения в *Storage* (с помощью метода *storageSet*), оно конвертируется в *BASE64* с помощью метода *_stringToBase64*, а при получении данных из *Storage* (с помощью методов *storageGet* и *storageGetKey*) значение конвертируется из *BASE64* в *UTF-8* с помощью метода *_base64ToString*. Также для удобства работы с библиотекой класс *Bridge* содержит метода *storageDelete*, который вызывает метод *storageSet* с пустым значением (способ удаления ключей из *Storage*) [5].

Приложения, разработанные для платформы ВКонтакте *Mini Apps*, состоят из экранов (*View*), панелей и модальных окон. Экраны – основные части приложения, панели используются для навигации внутри экрана, а модальные окна – для запроса пользовательского подтверждения или для выбора конфигураций, например фильтров или сортировки для таблицы [6].

UML-диаграмма последовательностей, описывающая возможные сценарии взаимодействия с приложением приведена в приложении А.1.

UML-диаграмма развертывания, описывающая архитектуру исполнения системы, приведена в приложении А.2.

UML-диаграмма прецедентов, описывающая варианты использования приложения приведена в приложении А.4.

UML-диаграмма компонентов, используемых при разработке приложения приведена в приложении А.5.

1.2.5.1 Планировка приложения

Для планировки приложения используется компонент *AdaptiveLayout*. Он содержит компоненты навигации, позволяющие перемещаться между

экранами приложения и непосредственно сами экраны, а также панели. Навигация приложения реализована с помощью компонента *Epic* (рисунок 10) и библиотеки *@happysanta/router*. Данная библиотека предоставляет хуки *useRouter* и *useLocation*. *useRouter* возвращает объект роутера, содержащий методы *pushPage* – для перехода на другую страницу и *popPage* – для перехода назад. Каждая панель в данном приложении (кроме панели «Мой профиль») содержит стрелку «Назад», при нажатии на которую выполняется метод *popPage,* осуществляющий переход на предыдущую панель. Каждая кнопка на панели навигации (рисунок 10) выполняет вызов метода *pushPage* с адресом страницы для перехода. Кнопки «Профиль», «Расклады», «Играть», «Описание» и «Настройки» выполняют переход на панели «Мой профиль», «Выбор расклада», «Играть», «Описание» и «Настройки» соответственно.

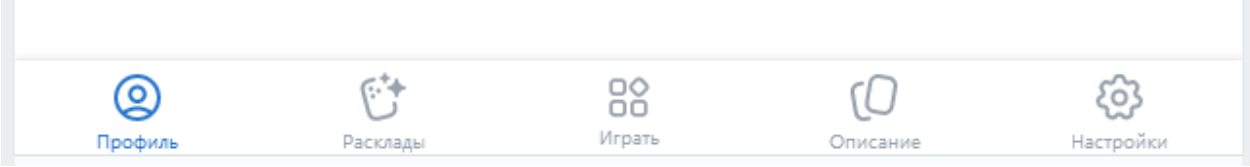


Рисунок 10 – Основная навигация приложения

1.2.5.2 Панель «Мой профиль»

Панель приведена на рисунке 11. Она является частью экрана «Профиль», является основной в приложении и открывается при его запуске. После загрузки приложения, выполняется запрос для получения данных пользователя ВКонтакте с помощью метода *loadUserInfo* и запрос для получения сохраненных вопросов с помощью метода *loadUserPredictions*.

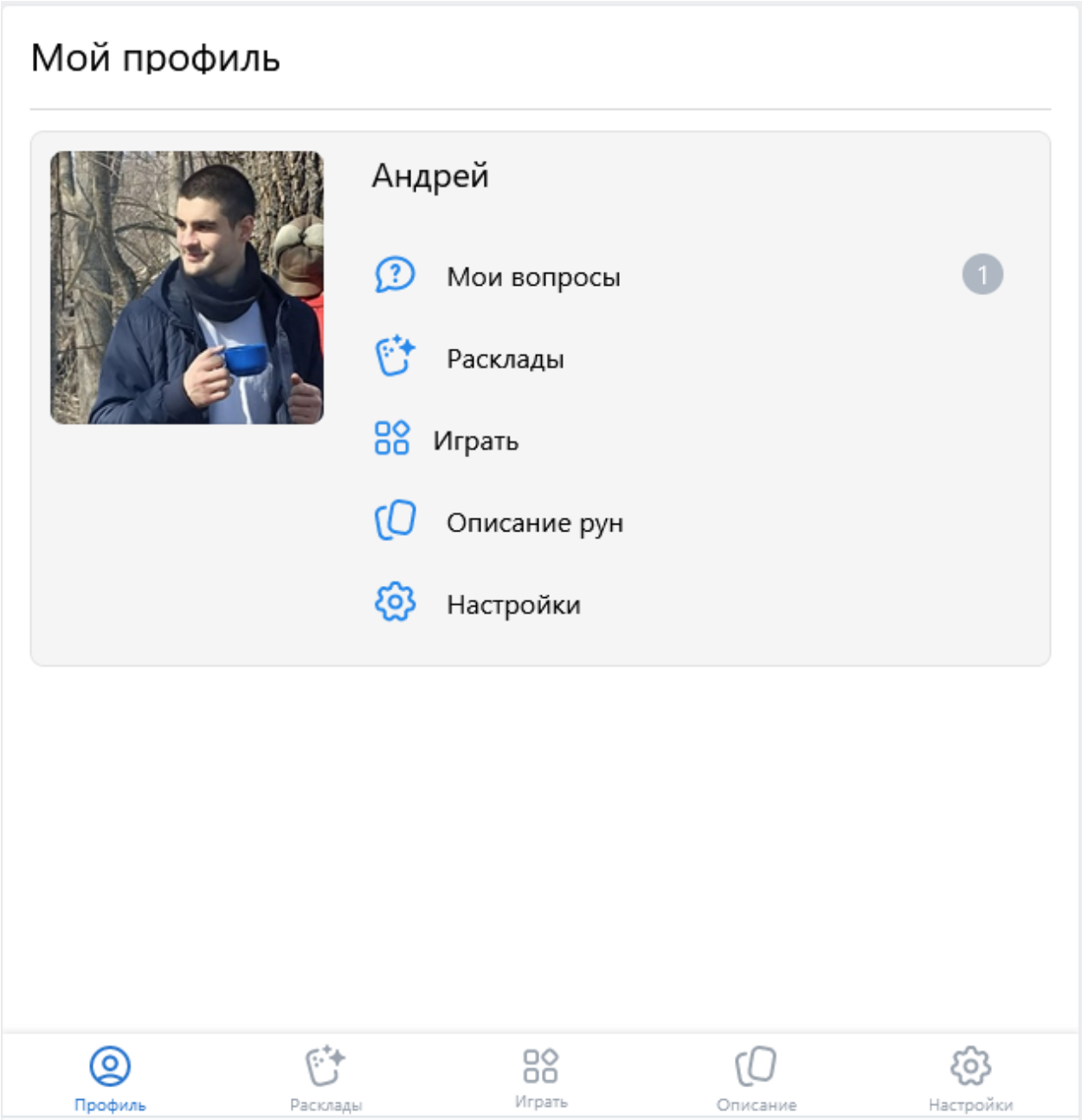


Рисунок 11 – Панель «Мой профиль»

Полученные данные сохраняются в состоянии приложения и остаются там на всем протяжении его работы. Форма данных приведена в таблице 3.

Таблица 3 – Форма данных пользователя

Название поля	Тип поля
Имя пользователя	*string*
Ссылка на фотографию пользователя	*string*

Панель «Мой профиль» состоит из двух блоков:

- блок информации о пользователе, содержащий его фотографию и имя;

- блок навигации, содержащий меню для перехода на другие экраны приложения.

Блок навигации состоит из пяти пунктов:

- «Мои вопросы», содержащий количество сохраненных вопросов (рисунок 12) и переводящий на панель «Мои вопросы»;

- «Расклады», переводящий на панель «Выбор расклада»;

- «Играть», переводящий на панель «Играть»;

- «Описание рун», переводящий на панель «Описание»;

- «Настройки», переводящий на панель «Настройки»;

Четыре последних пункта аналогичны соответствующим пунктам на основной панели навигации и нужны для удобства пользователя.

Рисунок 12 – Пункт меню «Мои вопросы»

1.2.5.3 Панель «Мои вопросы»

Панель является частью экрана «Профиль» и содержит список сохраненных вопросов (рисунок 13) из хранилища состояния приложения и кнопку «Обновить», нажатие на которую выполняет повторный запрос данных из *Storage* путем вызова метода *loadUserPredictions*. Вопросы загружаются при загрузке панели «Мой профиль», так как длина списка отображается в пункте меню «Мои вопросы» Список представляется постранично по 10 элементов на странице. Данные для панели хранятся в

Storage, предоставляемом *API* ВКонтакте. Если количество сохраненных вопросов меньше или равно 10 компонент перехода по страницам не отображается.

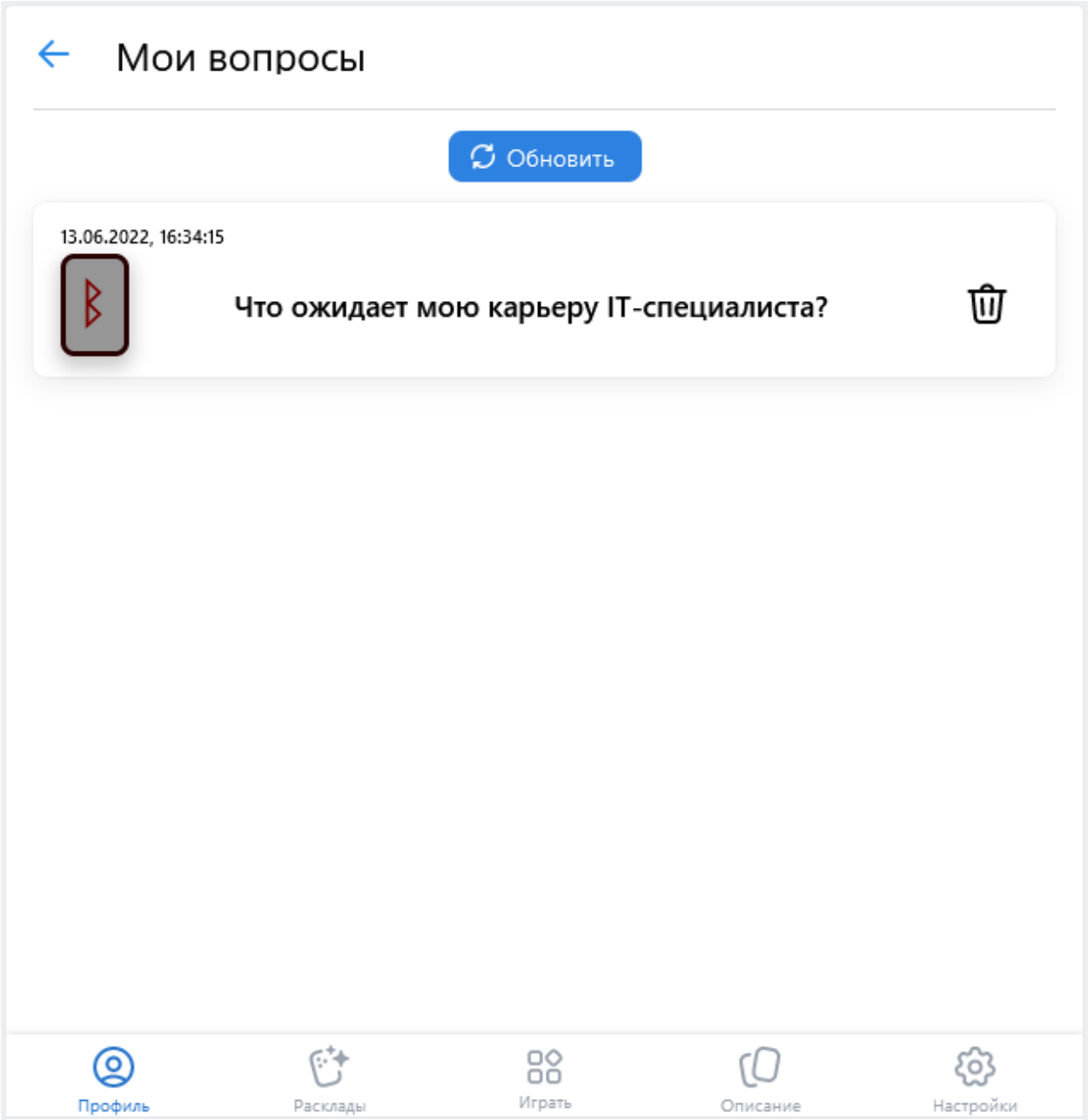


Рисунок 13 – Панель «Мои вопросы»

Если список сохраненных вопросов пуст, панель отображает текст «Нет сохраненных вопросов!» и кнопку «Обновить» (рисунок 14). При переходе на другую страницу вызывается метод *setPredictionPage*,

устанавливающий поле *predictionPage* в хранилище состояния приложения, а затем метод *loadUserPredictions*, загружающий сохраненные вопросы.

При нажатии на элемент списка сохраненных вопросов выполняется переход на панель «Ответ» (рисунок 16).

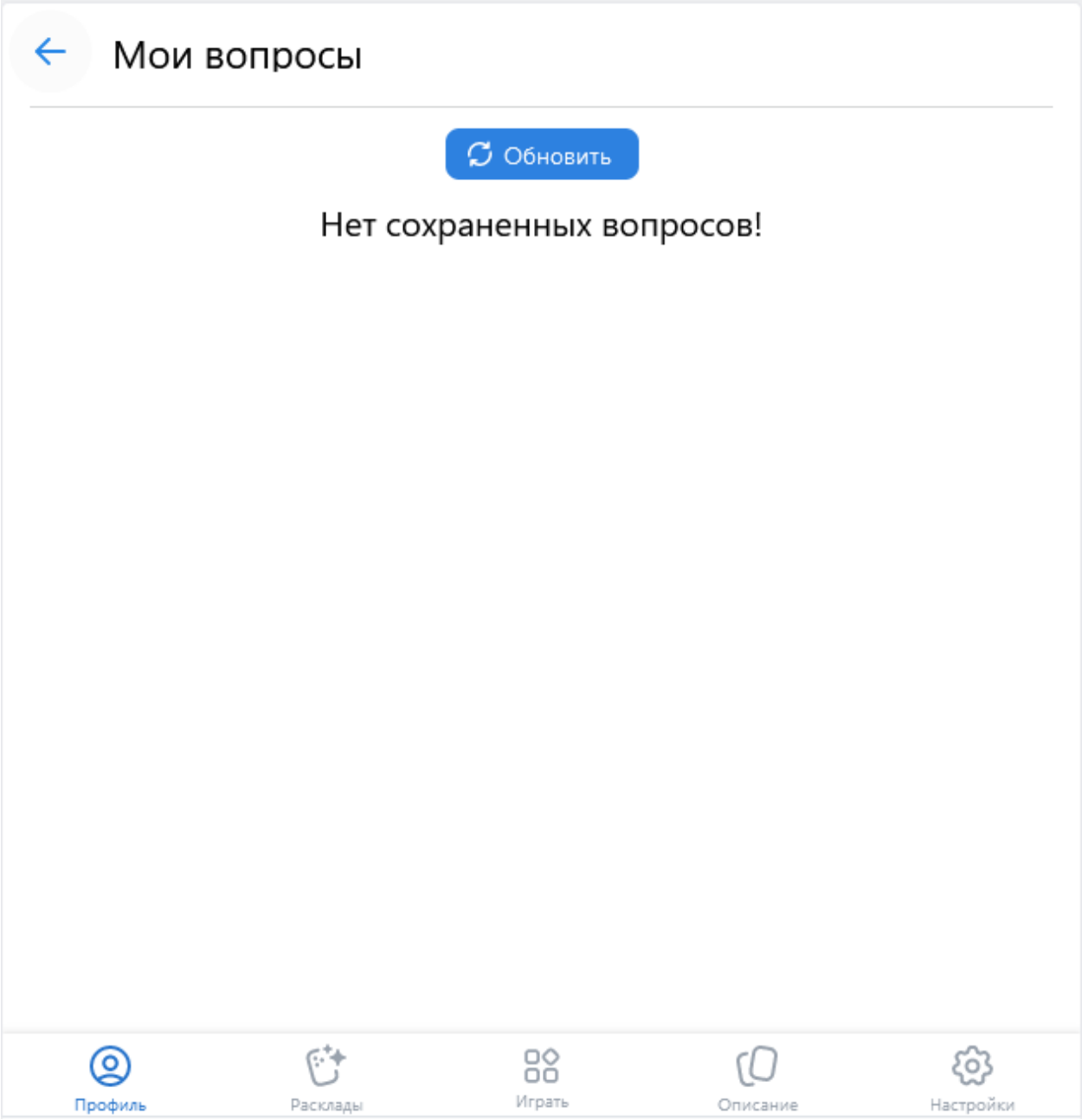


Рисунок 14 – Панель «Мои вопросы» с пустым списком

1.2.5.4 Панели «Результат» и «Ответ»

Данные панели используют один и тот же компонент для отображения содержимого. Панель «Результат» (рисунок 15) используется для

демонстрации ответа на заданный при выборе элемента вопрос, а панель «Ответ» (рисунок 16) – для демонстрации ответа на сохраненный вопрос.

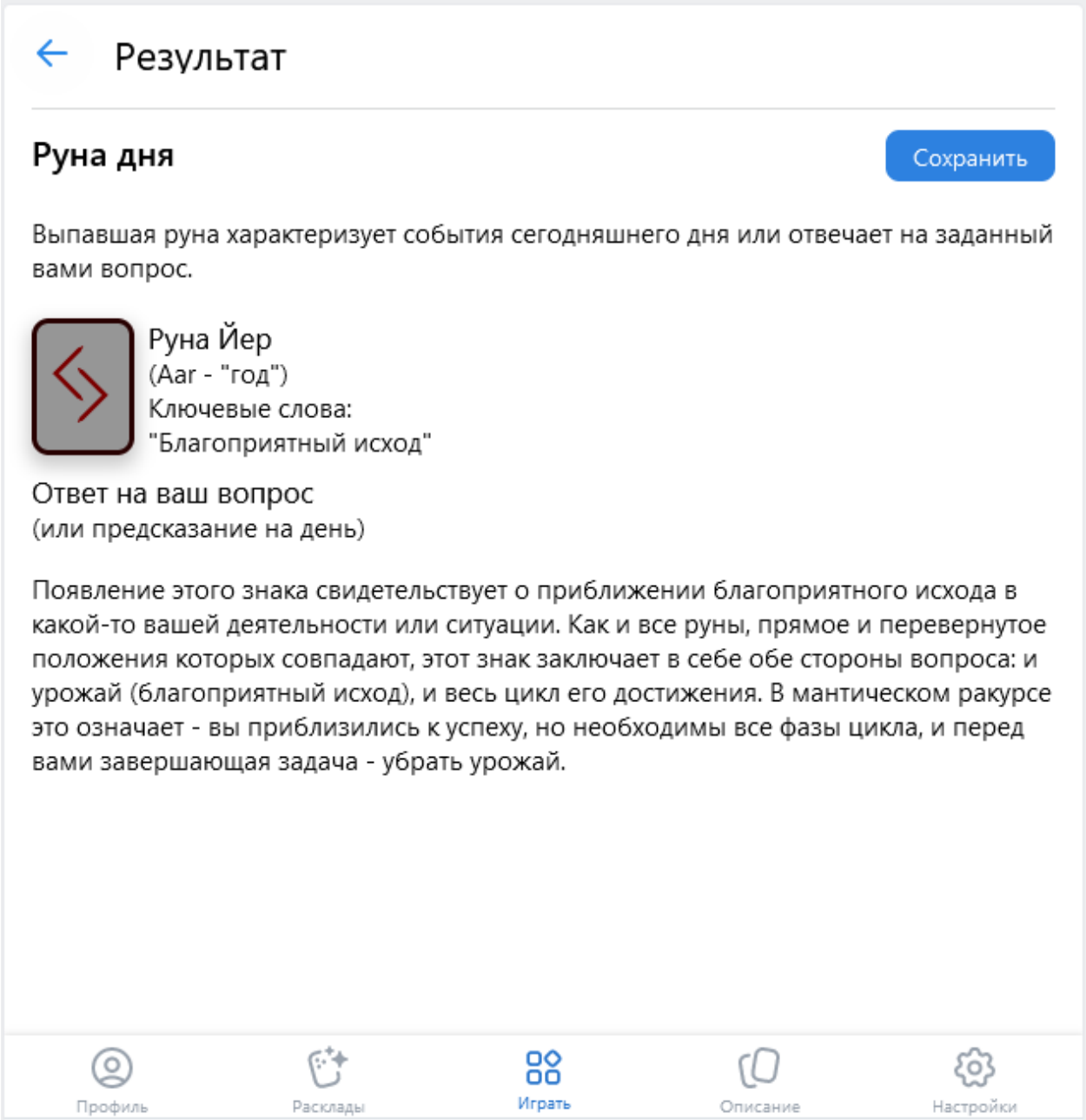


Рисунок 15 – Панель «Результат»

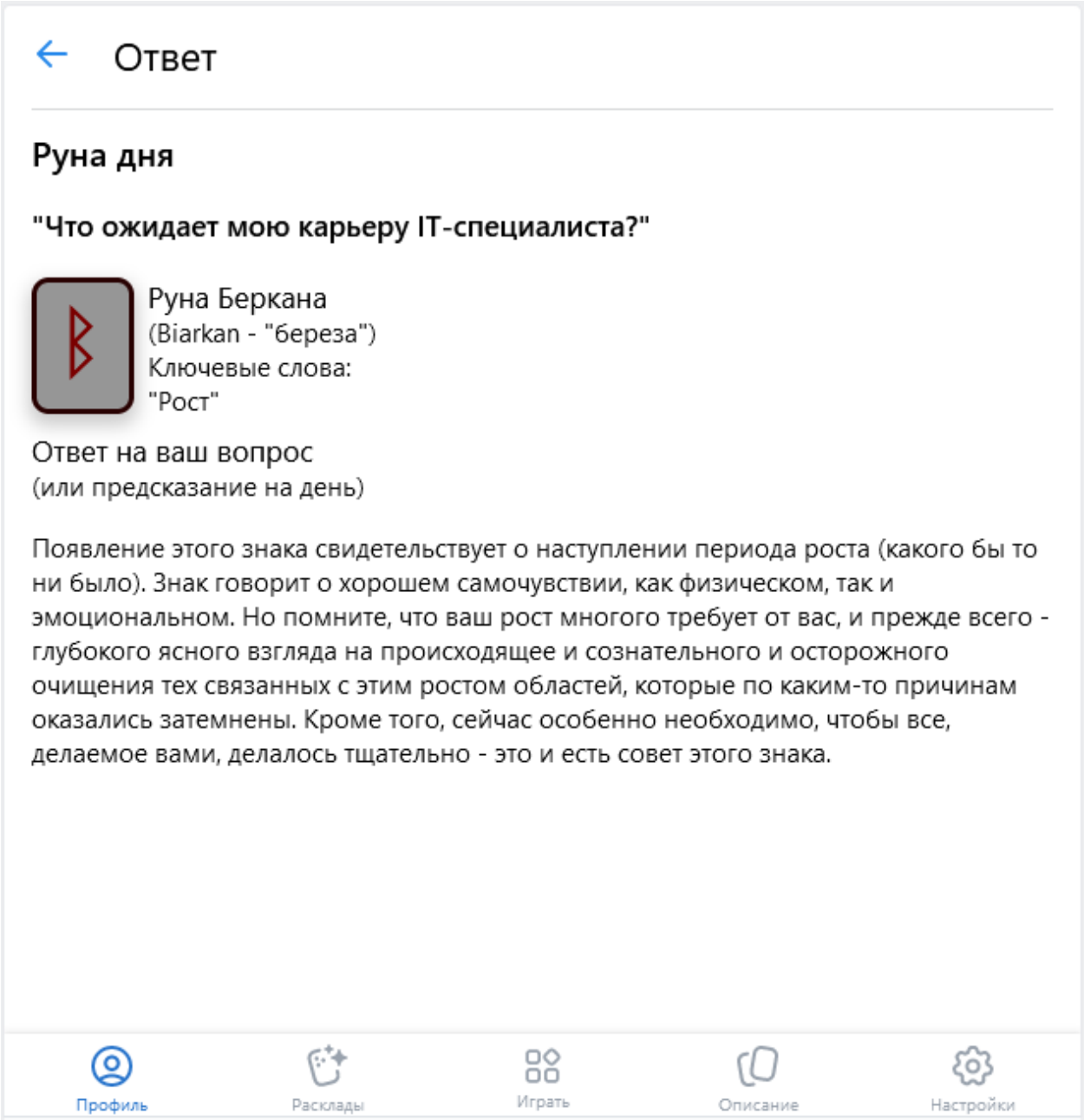


Рисунок 16 – Панель «Ответ»

В отличии от панели «Результат», панель «Ответ» содержит текст сохраненного вопроса, не показывает описание выбранного способа игры и содержит кнопку «Сохранить» (или заблокированную кнопку «Сохранено», если результат уже был сохранен), при нажатии на которую открывается модальное окно «Заметка к предсказанию», позволяющее сохранить вопрос (рисунок 17).

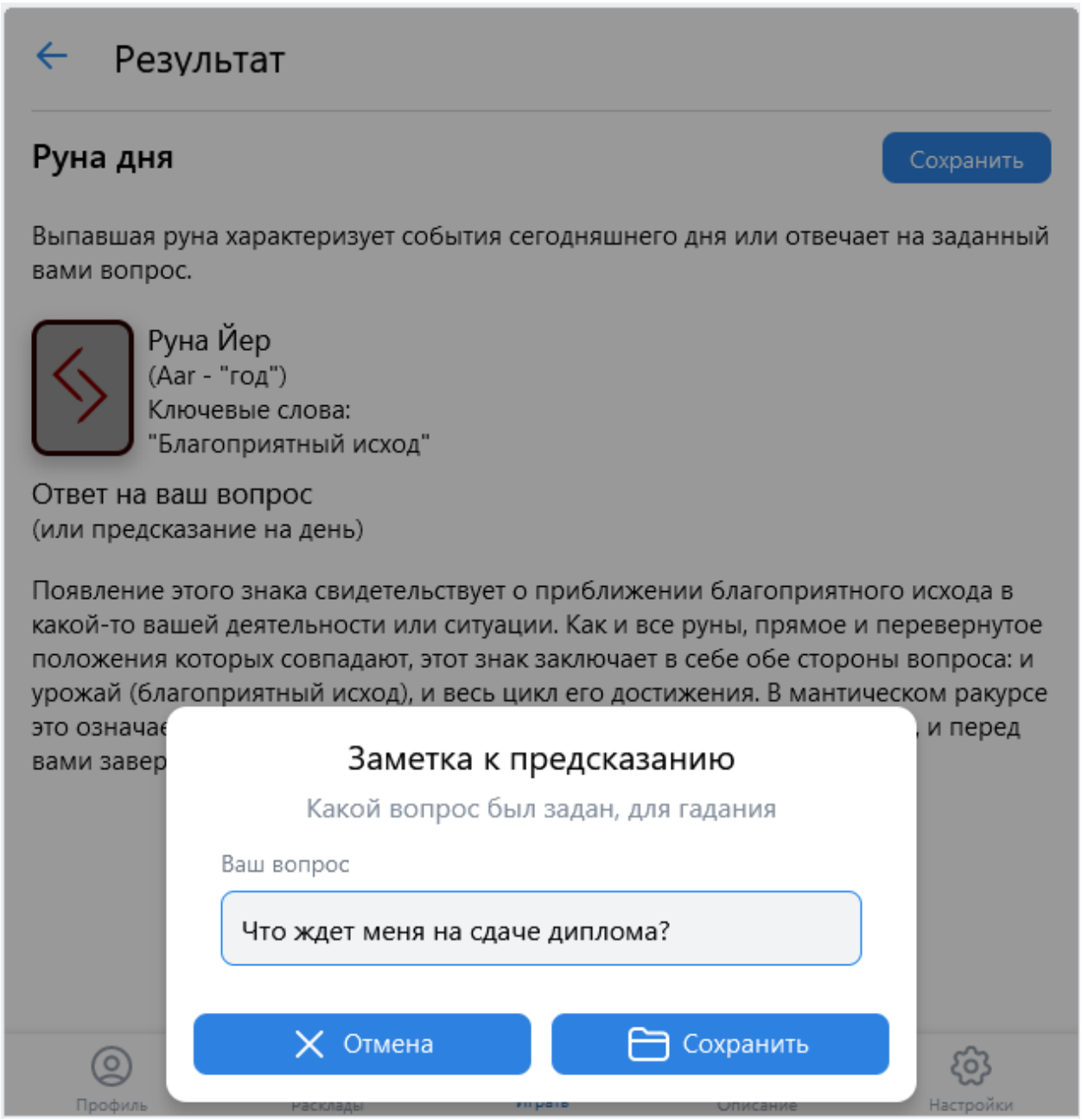


Рисунок 17 – Панель «Результат» (в процессе сохранения)

Компонент, используемый для отрисовки содержимого панелей, работает с данными раскладов. Расклады называются по количеству используемых в них элементов – «1», «3» или «4» для данного приложения.

Каждый расклад описывается объектом, соответствующим интерфейсу:

- *name: string* – название расклада;

- *description?: string* – описание расклада;

- *stages: Stage[]* – массив стадий, необходимых для трактовки результата.

Каждая стадия соответствует интерфейсу *Stage*, который содержит следующие поля:

- *name: string* – название стадии;

- *description: string* – описание стадии.

Таким образом, при необходимости можно добавить любое количество других раскладов, соответствующих общей схеме.

Также компонент результата использует данные об элементе, собранные в объект со следующим интерфейсом:

- *name: string* – название;

- *meaning: string* – значение в прямом положении (при игре);

- *invertedMeaning: string* – значение в обратном положении (при игре);

- *description: string* – описание;

- *transcription: string* – произношение на скандинавском языке;

- *keyWords: string[]* – массив ключевых слов, для в прямом положении;

- *invertedKeyWords: string[]* – массив ключевых слов, для в обратном положении.

1.2.5.5 Панель «Расклады»

Данная панель является вспомогательной и позволяет пользователю, выбрав необходимый расклад, перейти на основную панель. Панель «Выбор расклада» приведена на рисунке 18. При нажатии на карточку расклада (или на кнопку «Выбрать» внутри карточки) происходит вызов метода *setDivinationType* с типом расклада в качестве параметра. После этого выполняется переход на основную панель с помощью метода роутера *pushPage*.

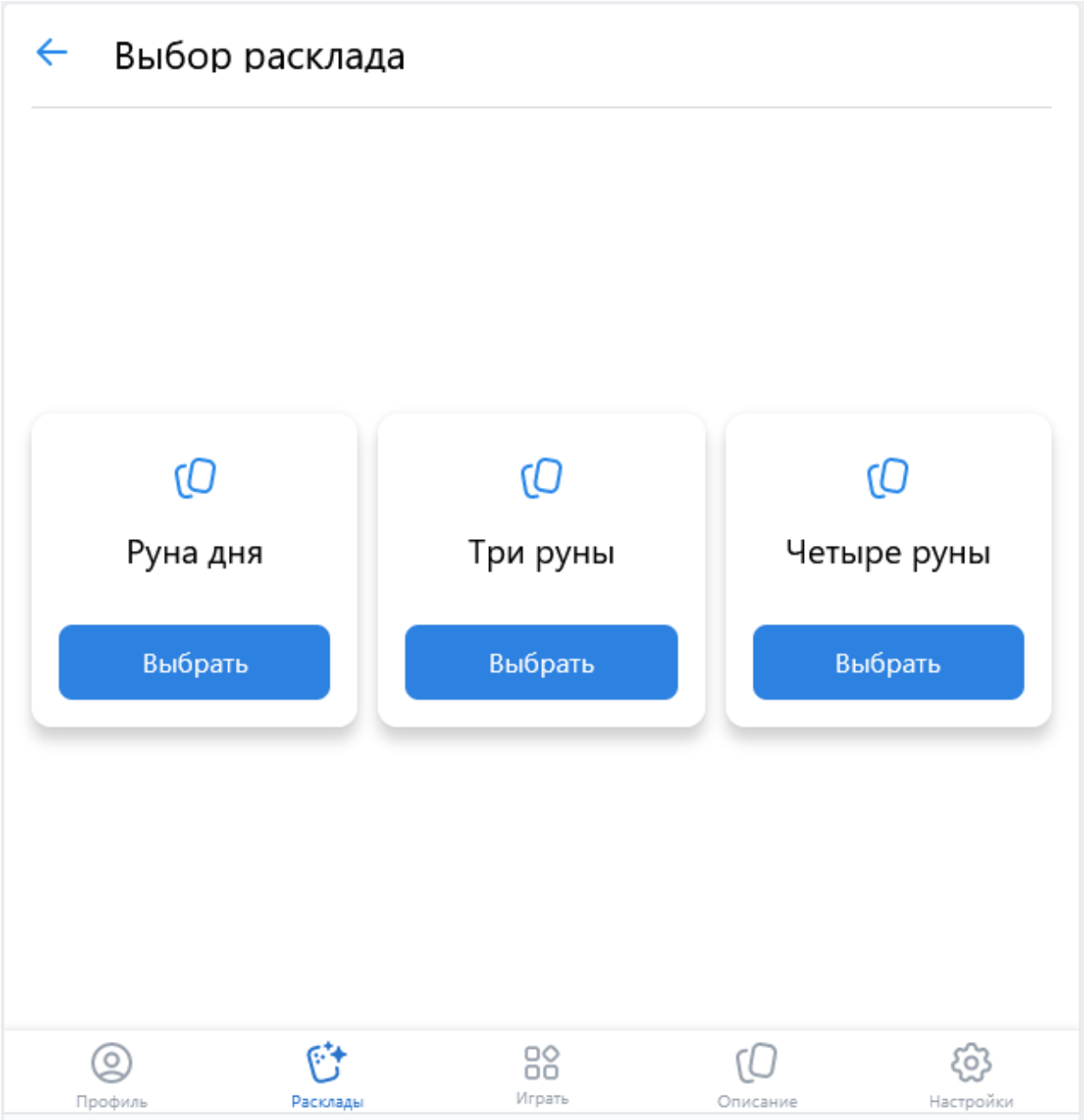


Рисунок 18 – Панель «Выбор расклада»

1.2.5.6 Основная панель

Панель позволяет взаимодействовать с набором элементов – перемешивать, перемещать отдельно взятый элемент, выбрать элемент (рисунок 19). При выборе элемента выполняется переход на экран «Результат» (рисунок 15). Основная панель состоит из двух блоков:

- блок управления содержит кнопку «Смешать» и набор слотов, в которые попадает элемент после его выбора;

- блок работы с элементами, позволяющий перемещать элементы по экрану и выбирать их.

Рисунок 19 – Основная панель

Кнопка смешать служит для переопределения координат и углов вращения каждого элемента (при нажатии вызывается метод *refresh* с параметром *isForced* равным *true* – для перемешивания элементов с анимацией).

Слоты элементов нужны для информирования пользователя о количестве выбранных им элементов (от одной до четырех в зависимости от выбранного расклада).

Каждый компонент элемента содержит специальную обертку, позволяющую перемещать элемент по экрану. Для этой цели с использованием браузерного *API* реализованы алгоритмы перетаскивания элементов для настольных и мобильных устройств (функции *dragAndDrop* и *dragAndDropMobile*, соответственно).

В зависимости от выбранных настроек переход к панели «Результат» выполняется сразу после выбора необходимого количества элементов или сначала появляется модальное окно, позволяющее пользователю перейти к панели «Результат» или выполнить сброс выбранных элементов, чтобы начать заново (рисунок 20).

Рисунок 20 – Основная панель с модальным окном подтверждения

1.2.5.7 Панель «Описание»

Панель содержит список существующих элементов (рисунок 21) и нужна для перехода на панель «Сведения о руне». Является частью экрана «Описание».

Рисунок 21 – Панель «Описание»

1.2.5.8 Панель информации об элементе

Состоит из блока с названием и изображением элемента и блока с его описанием (рисунок 22). Нужна для знакомства с каждым элементом. Является частью экрана «Описание».

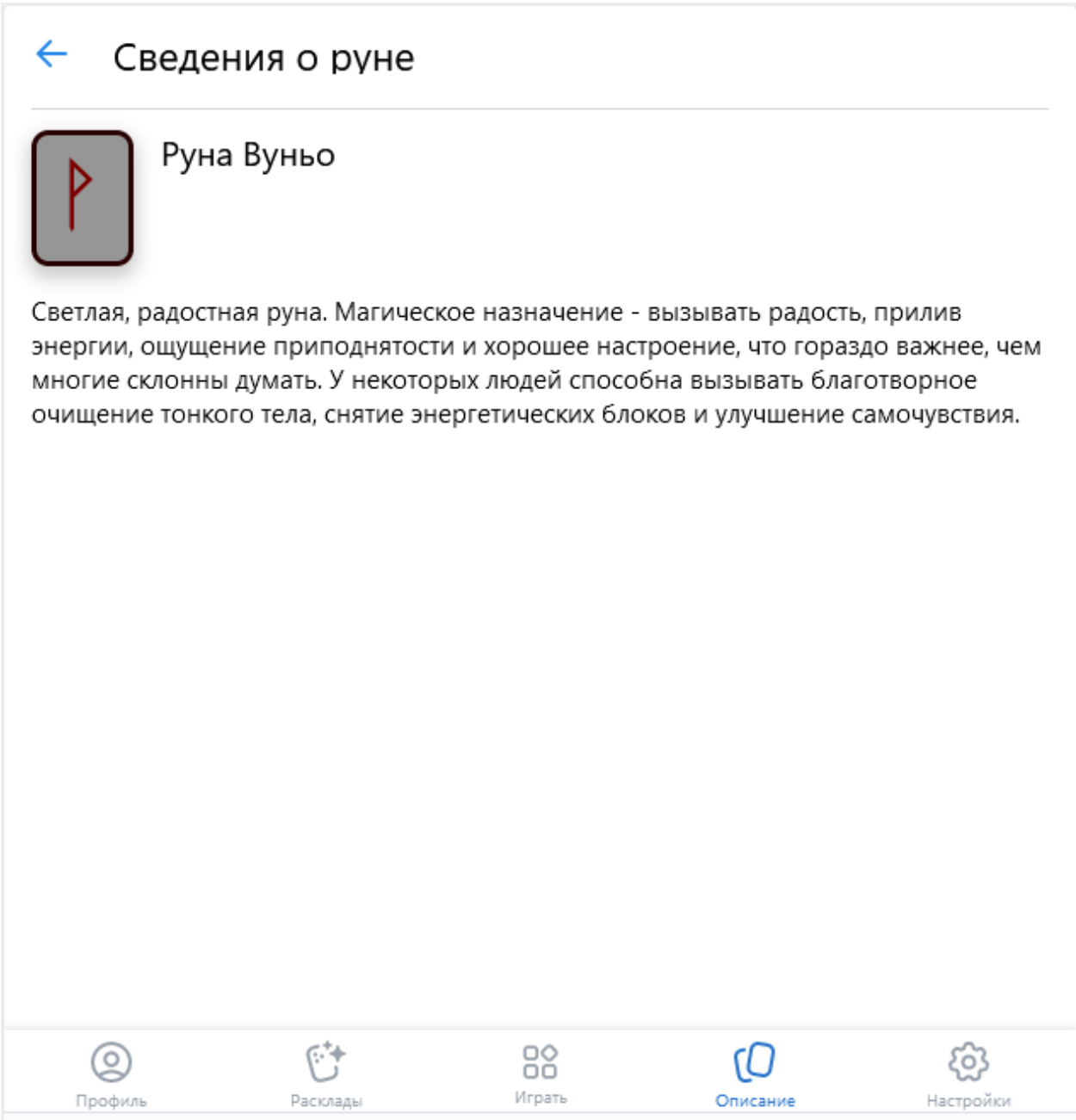


Рисунок 22 – Панель «Сведения о руне»

1.2.5.9 Панель «Настройки»

Панель состоит из четырех блоков:

- блок предпоказа, содержащий компоненты, на характеристики которых влияют настройки приложения;

- блок настройки цветовой гаммы;

- блок настройки темы и логики работы приложения;

- блок управления с кнопками «Сохранить», «Отмена» и «Сбросить».

Блок предпоказа отражает изменения цветовой гаммы, вносимые пользователем в блоке настройки цветовой гаммы. Значения цветов берутся из поля *tempSettings* в хранилище состояния приложения.

Блок настройки цветовой гаммы содержит пять контроллеров цвета, отвечающих за цвет плашки элемента, цвет обводки элемента, цвет рунического символа, цвет полотна (канваса) и цвет обводки канваса. Значения цветов для контроллеров берутся из поля *tempSettings* в хранилище состояния приложения. При изменении цвета контроллера вызывается метод *setTempSettings*, принимающий значение и название поля в объекте *tempSettings*.

Блок настройки темы и логики работы приложения содержит переключатель «Тема», отвечающий за выбор темы приложения (светлая/темная), переключатель «Скрывать руну» – выбранные элементы будут перевернуты пустой стороной при попадании в слот) и переключатель «Результат сразу», определяющий поведение основной панели при выборе необходимого количества элементов (если переключатель в положении включено, переход на панель «Результат» будет выполняться без подтверждения. Иначе на основной панели появится модальное окно, позволяющее пользователю сбросить выбранные элементы или перейти к панели «Результат» (рисунок 20).

Переключатель темы – единственная настройка, которая будет работать без сохранения. Тема приложения определяется в компоненте из библиотеки *VKUI – ConfigProvider*. Данный компонент принимает пропс appearance, который определяет тему приложения. Так как оценить удобство работы с темной или светлой темой возможно только после просмотра всех панелей приложения, значение темы берется из поля хранилища состояния *tempSettings*, а значит выбранная тема будет применена ко всему приложению без сохранения настроек.

Все остальные пункты настроек работают только в пределах панели настроек и не распространяются на остальное приложение до нажатия на кнопку «Сохранить». При нажатии на эту кнопку выполняется вызов метода *saveSettings*, который трансформирует объект *tempSettings* в *JSON* формат, сохраняет обновленные настройки (этот *JSON*) в *Storage*, а затем копирует значения *tempSettings* в settings, которые управляют логикой и внешним видом всего приложения. После сохранения настроек, кнопки «Сохранить» и «Отмена» блокируется до внесения новых изменений. При внесении изменений эти кнопки снова становятся доступными. Кнопка «Сбросить» доступна если текущие настройки отличаются от настроек по умолчанию. При нажатии на нее, настройки сбрасываются, кнопка блокируется, а кнопки «Сохранить» и «Отмена» становятся активными. Кнопка «Отмена» выполняет откат измененных настроек к действующим в данный момент (*settings*) и доступна, если новые настройки отличаются от текущих.

Панель «Настройки» приведена на рисунке 23. На рисунке 24 приведена основная панель в темной теме с обновленными пользовательскими настройками.

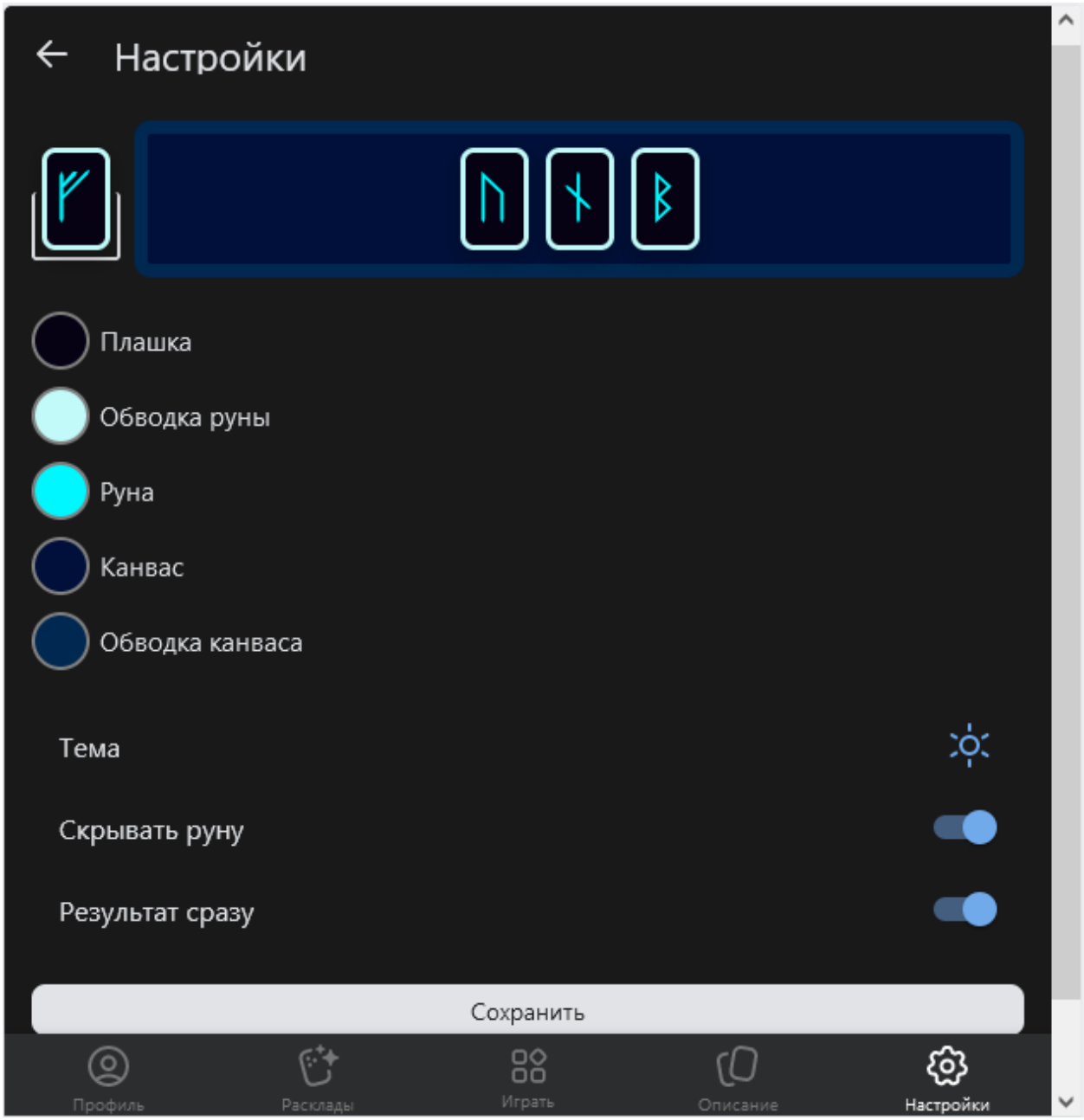


Рисунок 23 – Панель «Настройки»

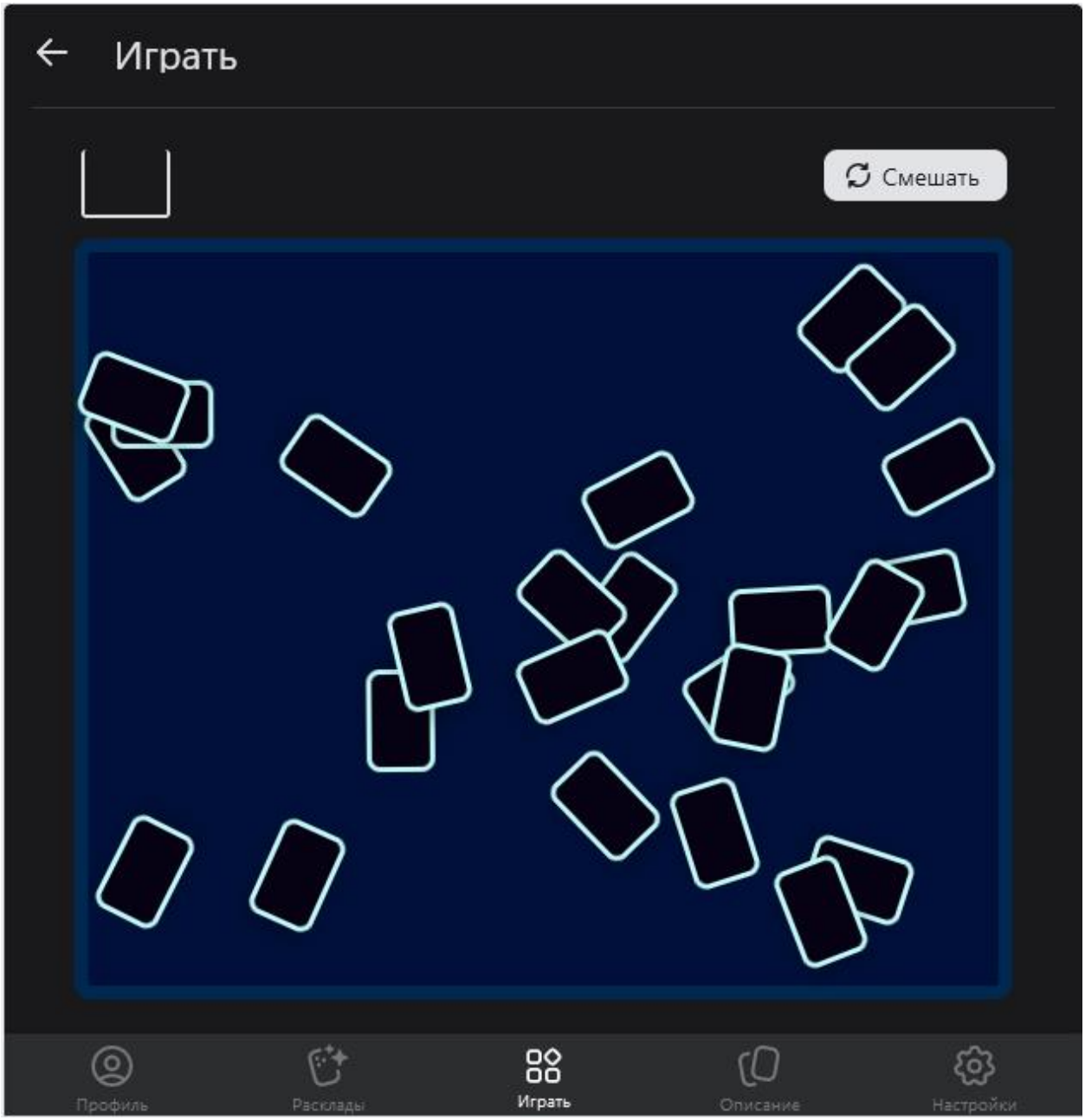


Рисунок 24 – Основная панель с пользовательскими настройками

1.2.5.10 Панель «Ошибка»

Как правило, при работе веб приложения ошибки происходят при взаимодействии с сервером, например, при попытке пользователя обратиться к недоступному ресурсу или при некорректном заполнении и отправке формы. Иногда проблема возникает в работе самого сервера, что тоже приводит к ошибке. О том, что операция, запрашиваемая пользователем, окончилась неудачей необходимо как-то сообщать. Для этого могут

использоваться модальные окна или тосты с информацией об ошибке, а также отдельные страницы для отображения ошибки. В данном приложении нет сервера, однако оно взаимодействует с ВКонтакте *API*, что в ряде случаев может приводить к ошибкам, например, пустому ответу на запрос сохраненных вопросов. Сервис выполнения запроса обрабатывает возможные исключения, но едва ли возможно учесть все возможные проблемы. Чтобы обезопасить пользователя от «падения» приложения был создан экран для обработки ошибок (рисунок 25). На экране существует единственная кнопка «Обновить данные». Нажатие на нее выполняет вызов метода *reloadAll* из хранилища состояния приложения, который выполняет обновление всех данных, полученных от ВКонтакте *API*. После обновления выполняется переход на страницу пользователя (с помощью метода *router.pushPage*). При этом, как видно на рисунке 25, страница ошибки не блокирует основную навигацию приложения. Это значит, что пользователь по-прежнему может переключаться между экранами, но в случае ошибки снова будет перенаправлен на страницу ошибки.

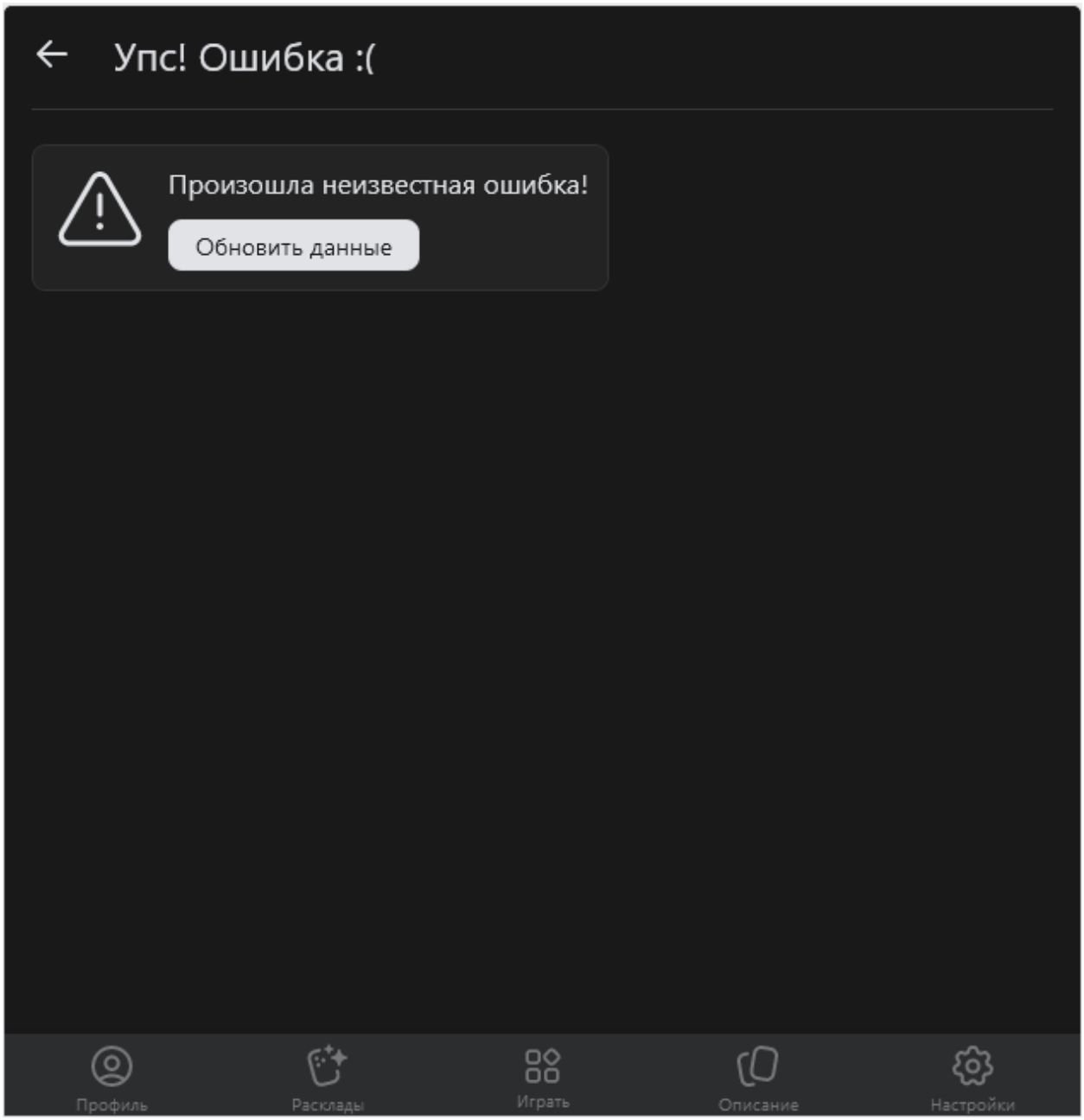


Рисунок 25 – Экран обработки ошибки.

2 Используемые технологии разработки

2.1 Языки программирования *JavaScript* (*TypeScript*)

Разработка веб приложений осуществляется благодаря языку программирования *JavaScript*. Этот язык используется во всех современных браузерах с целью добавления динамики на статичные шаблоны веб страниц. Благодаря доступу к браузерному *API*, с помощью *JavaScript* можно получить полный контроль над содержимым веб страницы – добавлять, удалять и изменять все элементы, размещенные на ней. Данный язык динамично развивается и сегодня может выполняться на любой платформе с установленным на ней движком *JavaScript* [7].

*Я*зык *JavaScript* является реализацией спецификации *ECMAScript*. В языке используется динамическая типизация – типы данных определяются движком *JavaScript* в процессе выполнения кода. Отсутствие строгой типизации имеет ряд преимуществ, но и не лишено недостатков. Основным преимуществом динамической типизации является скорость разработки – программисту не приходится описывать типы данных вручную, что экономит его время. С другой стороны, при росте приложения и использовании в нем сложных структур данных, возрастает сложность разработки. Программист должен помнить какие структуры данных он использует в коде, чтобы применять к ним соответствующие операции. Например, распространенной ошибкой, вызванной отсутствием строгой типизации является некорректная работа со строками и числами – *JavaScript* позволяет складывать числа и конкатенировать строки, но, при попытке сложить строку с числом результатом будет конкатенация, даже если строка состоит из цифр. Таким образом, при попытке сложить числа 5 и строки «5» результатом будет строка «55», что зачастую не является ожидаемым. Для решения этой и многих других проблем, связанных с отсутствием строгой типизации, разработчиками *Microsoft* был создан специальный синтаксис для *JavaScript*

– *TypeScript*. Используя этот синтаксис, разработчик может определять типы структур данных, используемых в разрабатываемом приложении. В случае ошибки типизации компилятор *TypeScript* укажет на ошибку. Приятным бонусом при использовании *TypeScript* являются подсказки редактора – при работе с любыми структурами данных редактор кода подскажет разработчику, какие поля и методы определены в структуре. Эта возможность есть практически у всех современных редакторов, таких как (*Visual Studio Code* и *WebStorm*) [8].

Браузеры не умеют выполнять код, написанный на *TypeScript*. Для решения этой проблемы существует компилятор *TypeScript*, который конвертирует типизированный код в обычный *JavaScript* код.

Приложение, разработанное в ходе данной дипломной работы написано на *TypeScript*.

2.2 Одностраничные приложения

Одностраничное приложение – веб приложение, полностью или почти полностью написанное на *JavaScript*. В таком приложении есть всего один файл разметки, который содержит только инструкции для загрузки основного скрипта программы. Этот скрипт, выполняясь браузером, динамически генерирует содержимое страницы веб сайта. При необходимости перехода на другую страницу, приложение запрашивает с сервера не шаблон новой страницы, а только данные, необходимые для ее отрисовки. При получении новых данных приложение генерирует другую страницу в соответствии с полученной информацией. Размер данных, как правило, значительно меньше размера целой страницы, поэтому запрос для их получения выполняется гораздо быстрее, чем запрос для получения целой страницы (это очень важно для пользовательского опыта, так как процесс взаимодействия клиента с сервером, как правило является наиболее продолжительным). Кроме того, сам размер одностраничного приложения обычно гораздо меньше размера

традиционного многостраничного сайта, что обеспечивает более высокую скорость первоначальной загрузки содержимого сайта.

Приложение, разработанное в ходе данной дипломной работы сделано по принципам одностраничных приложений.

2.3 Сборщик *Webpack*

Webpack — это сборщик модулей. Он анализирует модули приложения, создает граф зависимостей, затем собирает модули в правильном порядке в один или несколько скриптов, на который может ссылаться файл с разметкой. Обычно, при создании приложения на *JavaScript*, код разделяется на несколько частей (модулей). Затем в файле с разметкой необходимо указать путь до каждого скрипта. Это не только утомительно, но и подвержено ошибкам. Важно не только не забыть про какой-нибудь скрипт, но и расположить их в правильном порядке. Кроме того, если при разработке используются библиотеки или препроцессоры, синтаксис которых отличается от чистого *JavaScript* и *CSS*, то код, написанный с помощью этих инструментов, должен быть скомпилирован в понятный для браузера формат (программный код должен компилироваться в *JavaScript*, а стили компонентов – в *CSS*). *Webpack* позволяет без труда решить эту задачу, при подключении к нему необходимых модулей. Webpack может работать в режиме разработки и в режиме выпуска. В режиме разработки *Webpack* компилирует неоптимизированный код, который можно сразу запустить для выполнения отладки приложения, а в режиме выпуска генерирует оптимизированный и сжатый код, максимально уменьшая размер скрипта [9].

Webpack используется для сборки модулей разрабатываемого в данной работе приложения.

2.4 Библиотека *React JS*

Библиотека *React JS* – библиотека создания пользовательских

интерфейсов. Она полностью написана на *JavaScript* и предоставляет удобный *API* для разработки одностраничных приложений. Работа с *React JS* требует настройки *Webpack*, так как эта библиотека предоставляет особое расширение для синтаксиса *JavaScript – JSX* – расширение языка *JavaScript*, напоминающее язык шаблонов *HTML*.

У *React JS* существует несколько конкурентов. Одни из самых популярных среди – фреймворки *Angular JS* и *Vue JS*. Я остановил свой выбор на технологии *React JS*, так как:

- *React JS* – библиотека (*Angular JS* и *Vue JS* – фреймворки), поэтому размер программного кода *React* меньше;

- синтаксис *JSX* можно использовать только с *React JS*;

- *React JS* проще в использовании, чем *Angular JS* и дает больше императивных возможностей работы с *DOM*, чем *Vue JS*;

- так как приложение разрабатывается для платформы ВКонтакте *Mini Apps* использование *React JS* предпочтительно, потому что библиотека визуальных компонентов ВКонтакте *VKUI* написана на *React JS* и может быть использована только с этой библиотекой.

React исходит из принципа, что логика рендеринга неразрывно связана с прочей логикой пользовательского интерфейса: с тем, как обрабатываются события, как состояние изменяется во времени и как данные готовятся к отображению. Вместо того, чтобы искусственно разделить технологии, помещая разметку и логику в разные файлы, *React* разделяет ответственность с помощью слабо связанных единиц, называемых «компоненты», которые содержат и разметку, и логику. Такой подход к организации файлов называется компонентным. *React*-компонент – это функция, которая должна возвращать особый объект, содержащий в себе информацию об изменении разметки страницы или класс с обязательным методом render, который также должен возвращать объект с разметкой. Параметры, передаваемые в компонент, принято называть «*props*» (пропсы) - сокращение от слова

«*properties*» (свойства). Каждый компонент имеет жизненный цикл, состоящий из событий:

- монтирование компонента – при отрисовке компонента в *DOM* дереве;

- обновление компонента – происходит при обновлении родительского компонента или пропсов или состояния текущего компонента;

- размонтирование – выполняется перед удалением компонента из дерева *DOM*.

Для работы с состоянием и событиями жизненного цикла компонентов в *React JS* существует *Hooks API*. *Hook* (хук) – специальная функция, позволяющая использовать возможности *React* в функциональных компонентах. Так для работы с состоянием компонента существует хук *useState*. Эта функция в качестве параметра принимает начальное значение состояния или функцию, возвращающую начальное значение и возвращает массив, первый элемент которого – значение состояния, а второй – функция, которую можно использовать для изменения состояния. Менять состояние необходимо только с помощью данной функции, иначе компонент не будет перерисовываться после изменения [10].

Для работы с событиями жизненного цикла, существует хук *useEffect*. Данный хук принимает два параметра – функцию-обработчик события и массив зависимостей. Если функция-обработчик возвращает другую функцию, то эта функция будет выполнена перед следующим событием жизненного цикла. Массив зависимостей определяет условия срабатывания обработчика. Если его содержимое изменяется в ходе события жизненного цикла, функция-обработчик выполняется. В противном случае, ничего не происходит [10].

Другим фундаментальным хуком является *useContext*. Этот хук тесно связан с *React Context API*, который используется для определения контекста компонентов. Существует специальная функция *createContext*, которая

возвращает объект контекста, содержащий компонент *Provider* (провайдер), который используется в качестве обертки для других компонентов и принимает только один пропс – *value*. Все компоненты, находящиеся внутри провайдера, имеют возможность, используя хук *useContext* и передать ему в качестве параметра объект контекста. Тогда хук вернет value, которое было передано компоненту провайдер в качестве пропса. Этот механизм позволяет компонентам с легкостью получать данные, которые находятся значительно выше в дереве компонентов и избежать передачи необходимого значения через всю цепочку компонентов. В разработанном приложение *Context API* используется для предоставления всем компонентам доступа к хранилищу состояния [10].

2.5 Библиотека *MobX*

MobX – библиотека управления состоянием приложения. Состояние приложение определяет его внешний вид и поведение. В библиотеке используется паттерн «Наблюдатель» («*Observer*»). Компоненты приложения подписываются на изменения состояния и оповещаются о том, что оно произошло. При этом происходит перерисовка компонента с обновленным значением состояния. *MobX* легко интегрируется с *React JS*.

MobX работает по принципу паттерна проектирования «Наблюдатель» в соответствии со схемой на рисунке 26. Пользовательское событие (например, клик) вызывает действие (*action*). Действие — это любой фрагмент кода, который изменяет состояние. Использование действий помогает структурировать код и предотвращает непреднамеренное изменение состояния. Изменение состояния приложения приводит к пересчету вычисляемых значений (которые зависят от состояния). После выполнения пересчета, выполняются сторонние эффекты, которые были подписаны на изменение вычисляемых значений. К таким эффектам

относится, например перерисовка компонентов приложения, а также любая другая функция, зависящая от изменения состояния [11].

Рисунок 26 – Схема работы библиотеки *MobX*.

Другой популярной библиотекой управления состоянием для веб приложений и для библиотеки *React JS*, в частности, является *Redux*. Ключевым отличием *Redux* от *MobX* является способ работы с состоянием. *Redux* использует подход иммутабельных данных, а *MobX* – наоборот призывает к изменению состояния с помощью мутирования данных, при этом разработчик по-прежнему имеет возможность работать с иммутабельными данными, если это необходимо. *Redux*, также как и *MobX* использует действия (*actions*) для обновления состояния, но, для обеспечения иммутабельности вводит дополнительное понятие – редьюсер (*reducer*). Редьюсер – это чистая функция, которая на основе входящих данных должна изменить определенный фрагмент состояния приложения. Эта функция выступает промежуточным звеном между действием и изменением приложения. Написание редьюсеров и действий для каждого из них – утомительная работа, которой можно легко избежать при работе с *MobX*. Другим немаловажным преимуществом этой библиотеки является отсутствие ограничений на количество хранилищ состояния. *Redux* использует паттерн проектирования «Одиночка» и вынуждает разработчика использовать только одно хранилище состояния для всего приложения. Используя *MobX*, можно создавать любое количество хранилищ и легко обеспечивать их взаимодействие.

3 Экономическое обоснование дипломного проекта

Разработка веб приложений сегодня одно из самых актуальных направлений в сфере информационных технологий. Повышенный интерес к веб приложением объясняется простотой их использования (нет необходимости установки, доступно в браузере) и широкой доступностью. В зависимости от основного профиля предприятия (фирмы или компании), занимающейся разработкой программного продукта необходимо определить порядок найма и финансирования команды разработчиков. Если у предприятия нет собственной команды разработчиков дешевле будет обратиться в компанию, занимающуюся заказной разработкой программного обеспечения, чем нанимать и содержать собственную команду. Иначе, выгоднее будет заниматься продуктовой разработкой с помощью собственных разработчиков.

3.1 Описание организации

Компания ООО «Джетлин» занимается заказной и продуктовой разработкой, внедрением и поддержкой программного обеспечения. Персонал компании насчитывает более 50 человек. Среди них – разработчики программного обеспечения, системные администраторы, проектные менеджеры, бизнес-аналитики, тестировщики программного обеспечения, дизайнеры, сотрудники отдела кадров, бухгалтерии и другие. Компания существует с 2011 года. Показатели выручки и стоимости компании сегодня достигают 104 миллиона рублей и 50.6 миллионов рублей, соответственно. Из-за пандемии и общей экономической нестабильности 2020-2021 годов, рост выручки, активно продолжавшийся с 2018 года, был приостановлен в 2020 году. Для повышения объема выручки компания «Джетлин» может предпринимать следующие шаги:

- поиск новых партнеров для сотрудничества с целью продажи специалистов для заказной разработки программного обеспечения;

- разработка собственных программных продуктов.

В этом году особенно затруднительна работа по второму направлению, так как, несмотря на все меры, принятые государством для поддержки *IT*-специалистов в нашей стране, многие из них приняли решение о переезде за границу, а количества оставшихся не хватает для организации команды разработчиков.

3.2 Анализ объекта исследования

Сегодня активно развивается экосистема сервисов ВКонтакте. Одна из популярных платформ этой компании – ВКонтакте *Mini Apps*. Это открытая платформа для разработки встраиваемых кросс-платформенных мини-приложений, которые можно запустить внутри ВКонтакте — в *Android, iOS*, версии для компьютера (*vk.com*) и мобильной версии m.vk.com. Мини-приложения можно публиковать в каталогах приложений, сообществах и виджетах ВКонтакте, а также на других популярных площадках: в Одноклассниках, в мобильной версии браузера *Atom* для *Android* и в мобильном приложении Почта *Mail.ru*.

Компания ВКонтакте активно поддерживает разработчиков мини приложений для платформы ВКонтакте *Mini Apps*, устраивая различные конкурсы с денежными призами и всячески помогая разработчикам в создании и публикации их приложений. Приложения, размещенные на платформе ВКонтакте, получают повышенный спрос со стороны пользователей, что способствует их продвижению и монетизации. Разработанное приложение способно не только привлечь внимание пользователей к процессу работы с ним, но и к другим продуктам компании, что будет способствовать повышении их популярности и экономической выгоды. Из-за широкой доступности и распространения в России социальной сети ВКонтакте, борьба за внимание ее пользователей становится актуальной задачей для всех российских *IT* компаний.

Разработка приложения требует анализа предметной области и постановки задач для разработчиков.

3.3 Определение экономической эффективности

Для определения экономической эффективности необходимо определить размеры затрат на создание проекта и размеры предполагаемой прибыли от его реализации и публикации.

Расчет стоимости часа работы команды приведен в таблице 4.

Таблица 4 – Расходы на реализацию программного продукта

Расходы	Процент участия в разработке	Стоимость работы	Затраты в час
Проектирование	35%	90000	279
Разработка	100%	50000	625
Тестирование	50%	60000	379
Нагрузочное тестирование	5%	50000	31
Внедрение	5%	50000	31
Документация	5%	60000	38
Накладные расходы			272
Итоговая стоимость часа работы			1655

Для разработки веб приложения необходимо разработать серверную и клиентскую часть. Определение сроков, необходимых для разработки каждой из частей зависит от многих факторов:

- степень профессиональной подготовки и опыт работы участников команды;

- сложность работы с используемыми технологиями;

- степень детализации поставленных задач (положительно влияет на скорость разработки, но увеличивает срок проектирования и планирования);

- определенность в видении конечного результата;

- степень финансирования проекта и размер команды.

Так как созданием продукта занимаются люди, большую роль при определении сроков играет человеческий фактор. При попытке оценить время, требуемое для реализации поставленной задачи, разработчик как правило дает оптимистичный прогноз, оценивая задачу поверхностно и не представляя всех возможных сложностей, возникающих при ее реализации. При такой оценке зачастую уложиться в срок не получается, потому что в процессе работы возникают не замеченные ранее проблемы, снижающие скорость разработки и повышающие психологическое напряжение разработчика. Менеджеры проекта должны учитывать эту проблему при определении сроков сдачи проекта.

Технические расчеты времени на разработку программного продукта ведутся на основании ГОСТа 19.102-77 «Стадии разработки» и складываются из следующих временных затрат:

- *Тпо* — подготовительное описание задачи;

- *То* — описание задачи;

- *Тбс* — подготовка блок-схем алгоритмов;

- *Тн* — написание программного обеспечения;

- *Тд* — документирование продукта.

Общее время на создание продукта вычисляется по формуле:

$$T = Tпо + To + Tбс + Tн + Tд. \quad (1)$$

Время на подготовительное описание задачи (*Тпо*) определяется по фактически затраченному времени.

Время на описание задачи (*То*) рассчитывается по формуле:

$$To = Q * B / 50 * K, \quad (2)$$

где Q – условное число команд,

B – коэффициент вероятных изменений в задаче (берется из интервала 1,2-1,5),

К – коэффициент квалификации разработчика, зависящий от стажа.

Время на подготовку блок схем алгоритмов (*Тбс*) рассчитывается по формуле:

$$Тбс = Q / 50 * К. \quad (3)$$

Время на написание программного кода (*Тн*) рассчитывается по формуле:

$$Тн = Q - 1.5 / 50 * К. \quad (4)$$

Время на документирование продукта (*Тд*) берется равным 40 чел/часов.

Условное количество команд *Q* определяется по формуле:

$$Q = q * c, \quad (5)$$

где *q* – число команд,

c – коэффициент сложности задачи (1.25-2)

Определим значения коэффициентов следующим образом:

- *Тпо* (время на подготовительное описание задачи) в данном случае составило около 80 часов;

- *В* (коэффициент вероятных изменений в задаче) приближенно равен *1.3*, так как проект динамично развивается, что способствует увеличению количества требований к нему и, как следствие вероятность внесения изменений;

- *К* (коэффициент стажа разработчика) определяется в соответствии с таблицей 5 и равен *0.8*, так как стаж разработчиков менее двух лет.

- *q* (число команд) около 3200;

- *c* (коэффициент сложности задачи) *1.4*, так как для разработки необходимо познакомиться с набором компонентов *VKUI* и *API* платформы ВКонтакте *Mini Apps*.

Таблица 5– Коэффициенты стажа разработчика

Стаж разработчика	Коэффициент стажа
до двух лет	0.8
от двух до трех лет	1.0
от трех до пяти лет	1.1-1.2
от пяти до семи лет	1.3-1.4
свыше семи лет	1.5-1.6

Расчет значений коэффициентов приведен в таблице 6.

Таблица 6 – Расчет времени для разработки приложения

Коэффициент	Выражение для расчета	Результат
Q	*3200*1.4*	*4480*
To	*4480 * 1.3 / 50 * 0.8*	*93.184*
Тбс	*4480 / 50 * 0.8*	*71.68*
Тн	*4480 – 1.5 / 50 * 0.8*	*71.656*
T	*80 + 91.184 + 71.68 + 71.656 + 40*	*354.52*

Исходя из результатов, полученных в таблице 6, время, необходимое для работки программного продукта составляет 354.52 чел/час.

Стоимость разработки проекта рассчитывается по формуле:

$$S = T * R, \quad (6)$$

где *T* – приблизительное время разработки,

R – стоимость часа работы команды.

Для данного проекта стоимость разработки равна:

*S = 354.52 * 1655 = 586730.6 (рублей).*

Стоимость рекламы и продвижения проекта была включена в расчет затрат на его реализацию, так как решением этих вопросов будут заниматься специалисты, занимающиеся аналитикой и проектированием данного программного продукта.

3.4 Методы повышения экономической эффективности

Повышение экономической эффективности возможно за счет снижения затрат на разработку. Так как большая часть сотрудников компании «Джетлин» задействована в заказной разработке, размер команды для создания собственного продукта сильно ограничен. Решением данной проблемы может стать разработка приложения без серверной части с использованием функционала, предоставляемого платформой ВКонтакте *Mini Apps*. Таким образом, команда будет состоять только из разработчиков клиентской части приложения, тестировщиков, бизнес-аналитиков и дизайнеров. Благодаря использованию платформы ВКонтакте *Mini Apps*, также отсутствует необходимость привлечения в проект специалистов по организации непрерывной интеграции, так как платформа ВКонтакте позволяет без труда реализовать доставку программного продукта на собственные сервера.

Обновленные расходы на реализацию приведены в таблице 7.

Таблица 7 – Расходы на реализацию программного продукта

Расходы	Процент участия в разработке	Стоимость работы	Затраты в час
Проектирование	35%	90000	279
Разработка	100%	25000	312
Тестирование	50%	30000	189
Документация	5%	60000	38
Накладные расходы			136
Итоговая стоимость часа работы			954

Исходя из итоговой стоимости часа работы, полученной в таблице 7, стоимость разработки проекта составляет:

*S = 354.52 * 954= 338212.08 (рублей).*

Таким образом, упрощение концепции приложения позволило сэкономить 586730.6 - 338212.08 = 248518.52 рублей, что составляет приблизительно 42.3 процента от изначально полученной стоимости проекта.

4 Безопасность жизнедеятельности

Рабочее место - место, где работник должен находиться в течение рабочего времени и где он выполняет работу в режиме и условиях, предусмотренных нормативно-технической документацией. Действующее законодательство предполагает право работника на безопасные условия труда (ст.212 ТК РФ).

Обеспечение безопасной жизнедеятельности программиста в значительной степени зависит от правильной оценки опасных и вредных производственных факторов.

Вредный производственный фактор – производственный фактор, воздействие которого на работающего, в определённых условиях, приводит к заболеванию или снижению работоспособности. Опасный производственный фактор - производственный фактор, воздействие которого на работающего, в определённых условиях, приводит к травме или другому внезапному ухудшению здоровья. Вредный производственный фактор, в зависимости от интенсивности и продолжительности воздействия, может стать опасным.

На программиста негативное действие могут оказывать следующие вредные и опасные факторы:

- плохой микроклимат помещения: повышенная и пониженная температура воздуха; чрезмерная запыленность и загазованность воздуха; повышенная и пониженная влажность воздуха;
- недостаточная освещенность рабочего места;
- превышающий допустимые нормы шум;
- повышенный уровень ионизирующего излучения;
- повышенный уровень электромагнитных полей;
- повышенный уровень статического электричества;
- опасность поражения электрическим током;
- блеклость экрана дисплея;
- нарушение эргономических норм при работе с компьютером и др.

В связи с длительной монотонной сидячей работой у оператора могут развиться:

- боли в спине;
- хроническая венозная недостаточность (из-за статической нагрузки);
- повышенное внутриглазное давление и общее падение зрения.

Правильная организация эргономики рабочего места оператора может снизить риски развития перечисленных патологий.

4.1 Микроклимат рабочего помещения

Микроклимат помещения – это совокупность температуры, влажности и скорости потока воздуха в помещении. Воздействие этих факторов может оказывать существенное влияние на самочувствие оператора автоматизированных систем и снижать его продуктивность. Среднее значение для температуры воздуха в помещении – плюс 22 градуса, для влажности – 46 процентов. Минимальная температура воздуха – плюс 19 градусов, максимальная – плюс 25. Помещения, в которых работают операторы автоматизированных систем, как правило содержат большое количество ЭВМ, которые при работе выделяют значительное количество тепла. Для поддержания оптимальной температуры в таких помещениях должна быть организована система вентиляции и кондиционирования в теплое время года, а зимой – водяное отопление.

4.2 Освещение рабочего места оператора

В большей степени интенсивность освещения офисных помещений подбирается из характера зрительной работы сотрудников. Чем она сложнее – тем ярче освещение должно быть на столе служащего. Работа программиста связана с очень высоким классом точности разрешения: размер наименьшего объекта различения (пиксела) составляет 0.28 мм. Поэтому такая работа относится ко 2-му разряду зрительной работы (очень высокой

точности). В зависимости от контраста объекта с фоном и характеристики фона различают подразряды зрительной работы. Для разряда 2 существуют четыре подразряда (2а-2г). Работу пользователя ЭВМ можно отнести к подразрядам 2в и 2г. В таблице 8 приведено соответствие характеристик зрительной работы и необходимой освещенности.

Таблица 8 - Соответствие характеристик зрительной работы и освещенности

Характеристика зрительной работы	Наименьший размер различения, мм	Разряд и подразряд зрительной работы		Освещенность, ЛК, при искусственном освещении
Наивысшей точности	менее 0.15	1	А	1500
			Б	1250
			В	750
			Г	400
Очень высокой точности	от 0.15 до 0.3	2	А	1250
			Б	750
			В	500
			Г	300
Высокой точности	свыше 0.3 до 0.5	3	А	500
			Б, В	300
			Г	200
Средней точности	свыше 0.5 до 1	4	А	300
			Б, В	200
			Г	150
Малой точности	свыше 1 до 5	5	А	200
			Б, В	150
			Г	100
Грубая	более 5	6	-	150

Характеристика зрительной работы	Наименьший размер различения, мм	Разряд и подразряд зрительной работы		Освещенность, ЛК, при искусственном освещении
Работа со светящимися материалами и изделиями в горячих цехах	более 0.5	7	-	200
Общее наблюдение за ходом производственного процесса	-	8	постоянное	75
			периодическое	около 40

Согласно СНиП 23-05-95 рекомендуемой считается освещенность рабочих помещений для работы с видеотерминалами 500 ЛК при большом контрасте фона с объектом различения и темным фоном, оптимальной - 750 ЛК. Скачки яркости при смене полей зрения должны быть минимальными, т.е. интенсивность освещения поверхности, где находятся рукописи и документы, не должна превышать яркости экрана дисплея.

Для проверки освещенности рабочего места оператора автоматизированной системы необходимо провести расчет. Исходные данные для расчета:

- длина помещения 5 метров;
- ширина помещения 4 метров;
- высота потолка 3 метра.

Расчет проводится для типового растрового светильника с четырьмя линейными люминесцентными лампами, мощностью 18 Вт каждая. Световой поток каждой лампы – 900 ЛК.

При расчете необходимо учитывать цвет и степень отражаемости всех поверхностей в помещении. Соответствие типа поверхности коэффициенту отражения приведено в таблице 9.

Таблица 9 - Соответствие типа поверхности коэффициенту отражения

Отражающая поверхность	Коэффициент отражения, %
Плоскость с белой поверхностью	80
Плоскость со светлой поверхностью	50
Плоскость с серой поверхностью	30
Плоскость с темной поверхностью	10

Потолки и стены в помещении белые, пол серый, соответственно коэффициенты отражения составляют 80, 80 и 30 процентов.

За норму освещенности возьмем значение 500 ЛК (согласно СНиП 23-05-95).

Индекс помещения рассчитывается по формуле, представленной ниже:

$$Ип = S / ((h1 - h2) * (а + б)), \quad (8)$$

где *S* – площадь помещения,

h1 – высота потолка,

h2 – высота стола,

а – длина помещения,

б – ширина помещения.

В нашем случае:

$$Ип = (5 * 4) / ((3 – 0,8) * (5 + 4)) = 1.01.$$

Исходя из определенных выше коэффициентов отражения и вычисленного индекса помещения, коэффициент использования будет равен 65, в соответствии с таблицей на рисунке 27.

	Отражение поверхностей								
Индекс помещения	Потолок	80	80	80	70	50	50	30	0
	Стены	80	50	30	50	50	30	30	0
	пол	30	30	10	20	10	10	10	0
	0,6	53	38	32	37	35	31	31	27
	0,8	60	45	38	44	41	38	37	34
	1	65	51	43	49	46	43	42	38
	1,25	70	57	49	54	51	48	47	44
	2	76	66	56	61	57	55	54	51
	2,5	78	70	59	64	60	58	57	54
	3	80	73	62	67	62	60	59	57
	4	81	76	64	69	63	62	61	58
	5	82	78	65	70	65	64	62	60

Рисунок 27 – Таблица коэффициентов использования

Теперь по формуле вычисления количества светильников для помещения:

$$Ксв. = (E * S * 100) / (U * Кл * СПл), \quad (9)$$

где E – норма освещенности рабочего места,

S – площадь помещения,

U – коэффициент использования,

$Кл$ – количество ламп в светильнике,

$СПл$ – световой поток лампы.

Вычислим:

$$Ксв = (500 * 20 * 100) / (65 * 4 * 900) = 5 (штук).$$

Таким образом, для нормального освещения рабочего места оператора автоматизированной системы достаточно 5 растровых светильников. Рабочее место, за которым происходило выполнение данной дипломной работы соответствует этому требованию.

4.3 Эргономические нормы при работе с компьютером

Существует ряд правил для операторов автоматизированных систем, соблюдая которые можно избежать проблем со здоровьем, вызванных неправильной организацией рабочего места и рабочего процесса. Вот некоторые из них:

- работать за компьютером необходимо сидя, не сутулясь, в правильно отрегулированном кресле – спинка кресла должна поддерживать спину в области чуть выше поясницы и не должна сковывать движения;

- плечи следует держать расслабленными, руки – согнутыми в локтях под прямым углом, чтобы избежать нарушения кровообращения.

- голову необходимо держать прямо или чуть наклонив вниз; расстояние от монитора до глаз не должно быть меньше 40 сантиметров и больше 75 сантиметров;

- сидеть следует на мягкой подушке, чтобы не затруднять кровообращение в ногах;

- ноги должны стоять на полу всей стопой и быть согнуты в коленях под прямым углом – это наиболее комфортное положение;

- клавиатуру следует держать на поверхности стола;

Правильное положение тела при работе за компьютером отражено на рисунке 28.

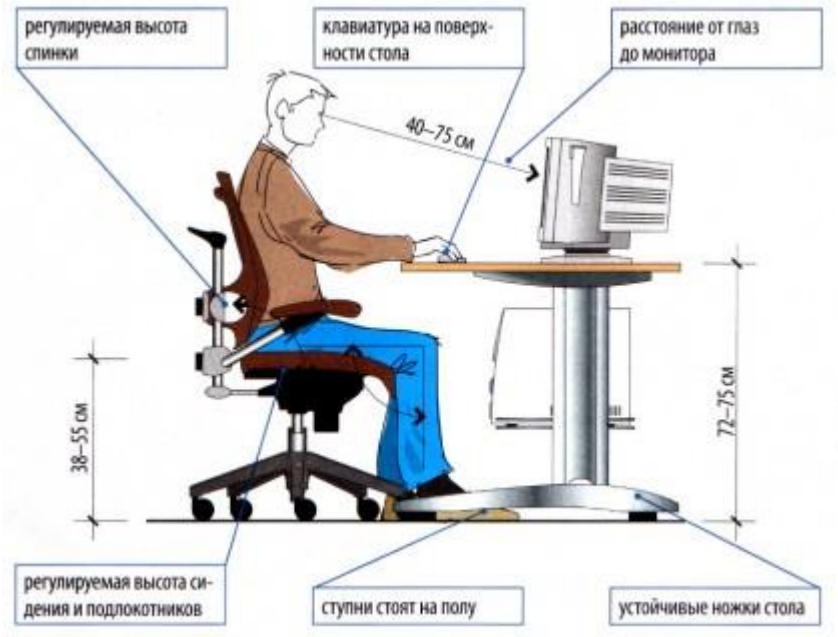


Рисунок 28 – Правильная посадка за компьютером

4.3.1 Размещение компьютеров в помещении

Помимо эргономичной организации персонального рабочего места оператора, необходимо правильно организовать размещение компьютеров в помещении:

- рабочий стол с компьютером не должен быть расположен вблизи силового кабеля или трансформатора;

- помещение, в котором размещены компьютеры, должно быть заземлено в соответствии с техническими требованиями по эксплуатации;

- рабочий стол должен располагаться боком по отношению к световым проемам так, чтобы естественный свет падал слева от компьютера.

Заключение

В ходе выполнения дипломной работы было разработано приложение, предназначенное для снятия психологической нагрузки операторов автоматизированных систем, путем переключения аналитического мышления на интуитивное с помощью работы с рунами. В качестве базовых технологий для разработки использовалась библиотека создания пользовательских интерфейсов *React JS*, библиотека управления состоянием веб приложений *MobX*, набор визуальных компонентов *VKUI* и *API* платформы ВКонтакте *Mini Apps*.

В ходе выполнения дипломной работы были разработаны схемы алгоритмов функций приложения, *UML*-диаграммы классов, последовательностей, компонентов, прецедентов и развертывания, а также полностью разработано и отлажено программное обеспечение.

Графическая часть дипломной работы содержит следующие *UML*-диаграммы:

- диаграмма последовательностей – 1 лист формата А1;
- диаграмма классов – 1 лист формата А1;
- диаграмма развертывания – 1 лист формата А1;
- диаграмма прецедентов – 1 лист формата А1;
- диаграмма компонентов – 1 лист формата А1.

Список использованных источников

1. Даниэль Канеман. Думай медленно… решай быстро. – М.: АСТ, Москва, 2013 – 11 с.

2. Эмоциональное выгорание разработчиков ПО. [Электронный ресурс]: Режим доступа: *https://codesubmit.io/blog/developer-burnout/*

3. Сложная навигация для приложений ВКонтакте *Mini Apps*. [Электронный ресурс]: Режим доступа: *https://vk.com/@-197036977-slozhnaya-navigaciya-na-vkui*

4. Паттерн проектирования «Одиночка». [Электронный ресурс]: Режим доступа: *https://refactoring.guru/ru/design-patterns/singleton*

5. Обзор *API* для работы с *VK Bridge Storage*. [Электронный ресурс]: Режим доступа: *https://dev.vk.com/bridge/getting-started#%D0%A1%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%20Storage*

6. Организация структуры приложений ВКонтакте *Mini Apps*. [Электронный ресурс]: Режим доступа: *https://vkcom.github.io/VKUI/#/Structure*

7. Вводная документация по языку программирования *JavaScript*. [Электронный ресурс]: Режим доступа: *https://developer.mozilla.org/ru/docs/Web/JavaScript*

8. Настольная книга для пользователей языка программирования *TypeScript*. [Электронный ресурс]: Режим доступа: *https://www.typescriptlang.org/docs/handbook/2/basic-types.html*

9. Документация по конфигурированию сборщика *Webpack*. [Электронный ресурс]: Режим доступа: *https://webpack.js.org/concepts/*

10. Обзор хуков библиотеки *React JS*. [Электронный ресурс]: Режим доступа: *https://ru.reactjs.org/docs/hooks-reference.html*

11. Базовые концепции библиотеки *MobX* [Электронный ресурс]: Режим доступа: *https://mobx.js.org/the-gist-of-mobx.html*

Приложение А

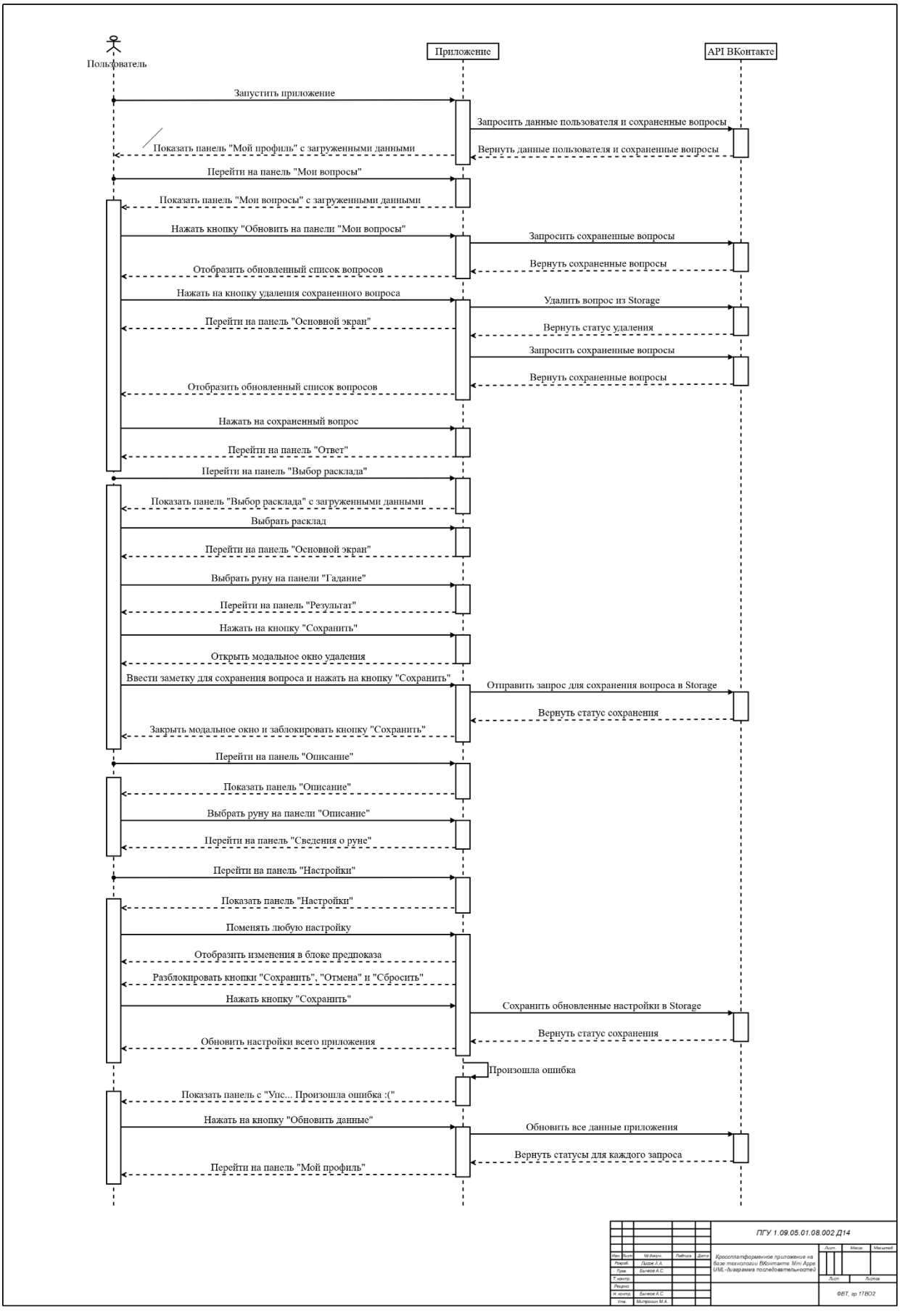
(Обязательное)

Кроссплатформенное приложение на базе технологии

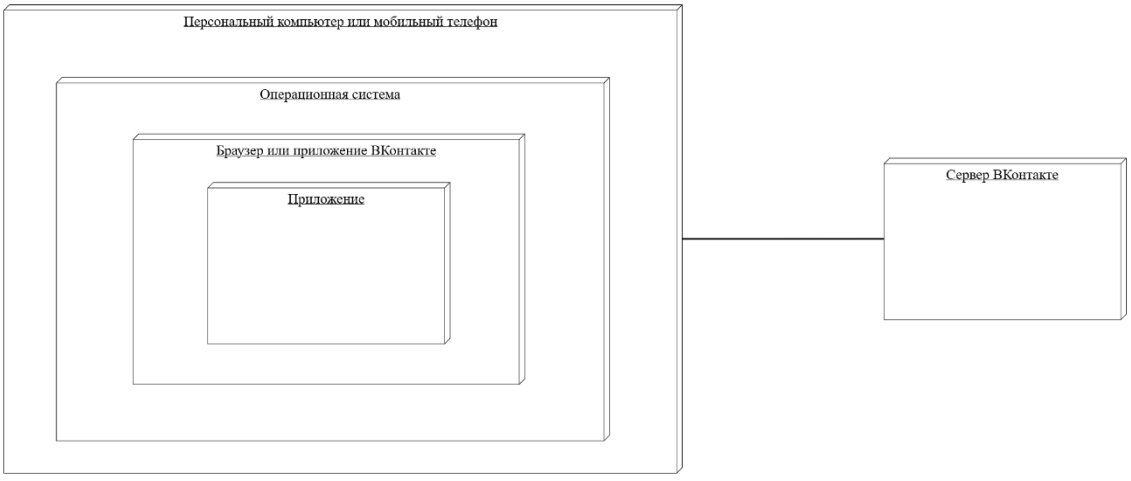
ВКонтакте *Mini Apps*

UML-диаграммы

Приложение А.1 - Диаграмма последовательностей

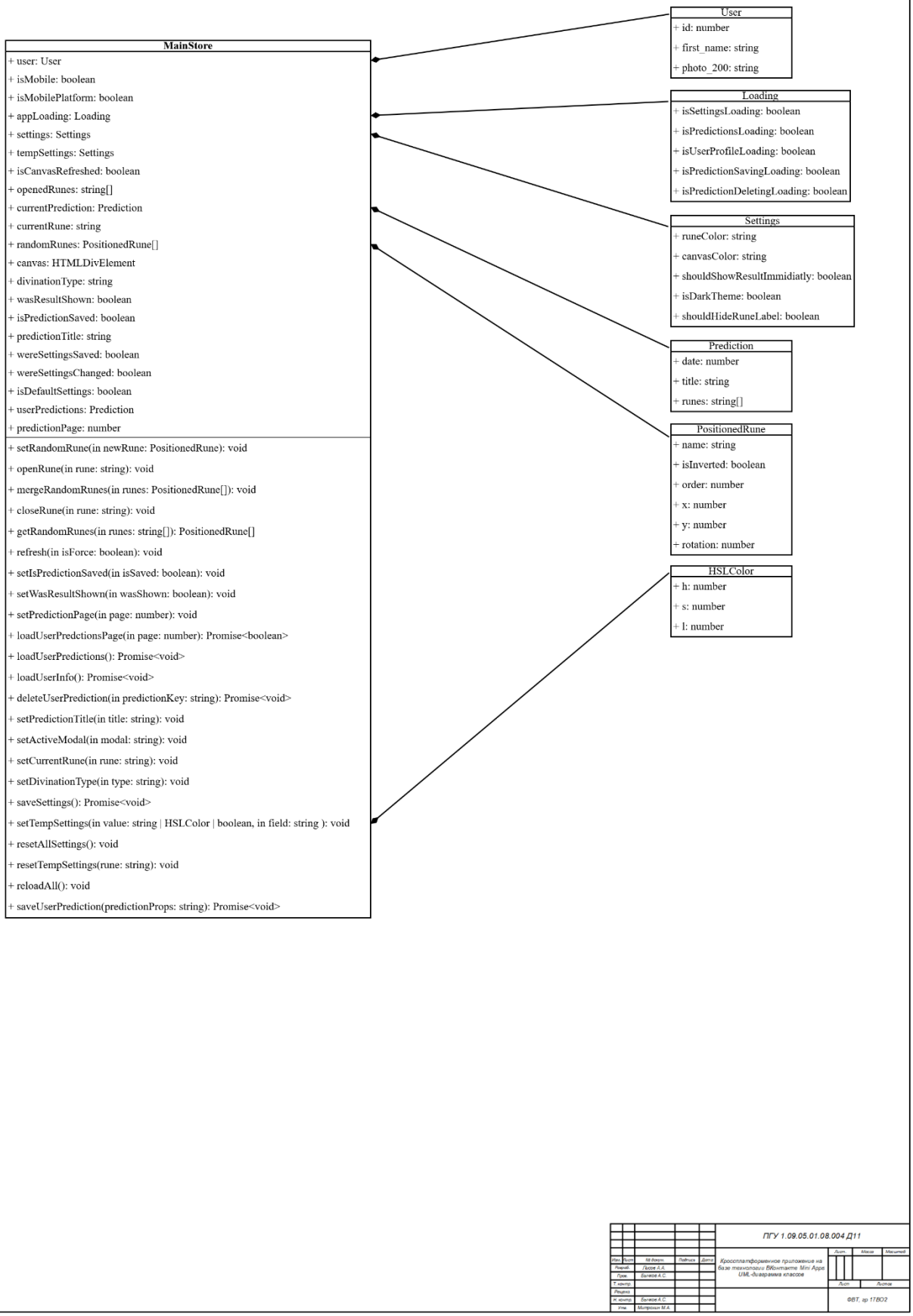


Приложение А.2 - Диаграмма развертывания



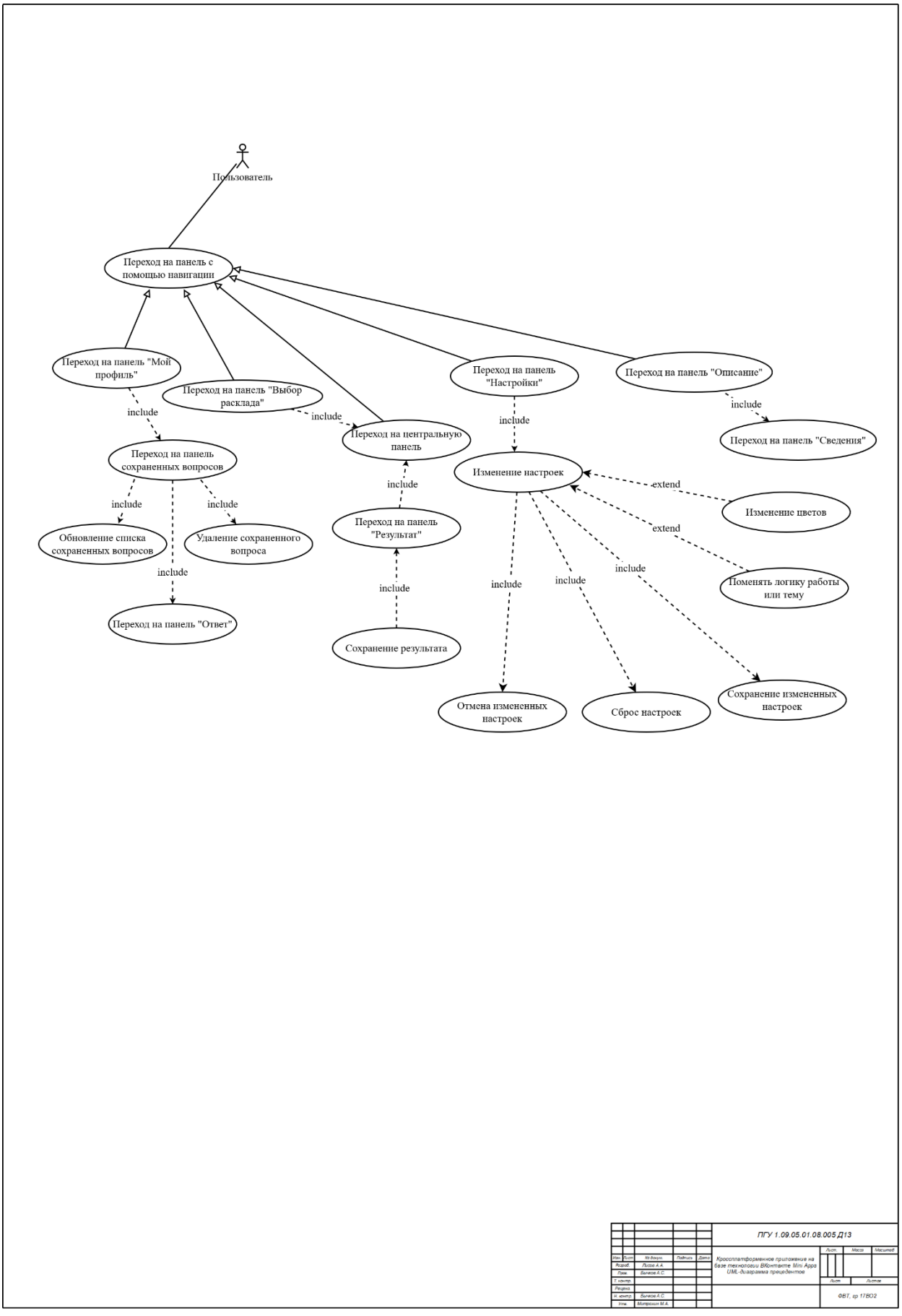
								ПГУ 1.09.05.01.08.003 Д19	
Изм.	Лист	№ докум.	Подпись	Дата					
Разраб.		Попов А.А.			Кросплатформенное приложение на базе технологии ВКонтакте Mini Apps UML-диаграмма развертывания	Лит.	Масса	Масштаб	
Пров.		Быков А.С.							
Т. контр.							Лист	Листов	
Реценз.									
Н. контр.		Быков А.С.					ФВТ, гр 17ВО2		
Утв.		Митрохин М.А.							

Приложение А.3 - Диаграмма классов



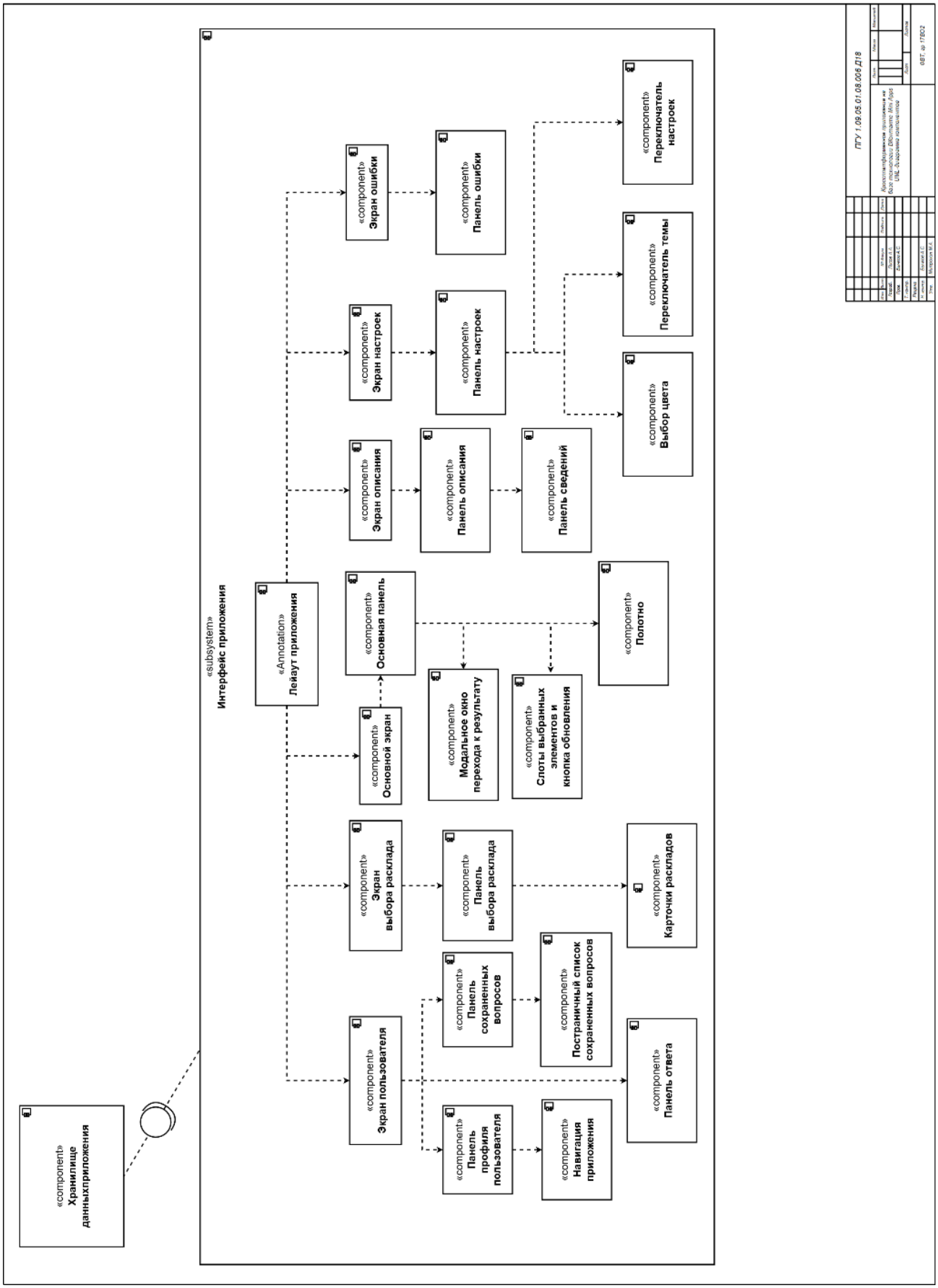
						ПГУ 1.09.05.01.08.004 Д11	
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Гарин А.А.			Кроссплатформенное приложение на базе технологии ВКонтакте Mini Apps. UML-диаграмма классов	Лит.	Масса	Масштаб
Пров.		Балаев А.С.						
Т. контр.						Лист	Листов	
Рецензент								
Н. контр.		Балаев А.С.				ФВТ, гр 17ВО2		
Утв.		Моторин М.А.						

Приложение А.4 - Диаграмма прецедентов



					ПГУ 1.090501.08.001 ПЗ	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.5 - Диаграмма компонентов



Приложение Б

(обязательное)

Кроссплатформенное приложение на базе технологии

ВКонтакте *Mini Apps*

Листинг программы

Файл src\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { configure } from 'mobx';
import { Bridge } from './modules/Bridge';

import App from './App';
import { MainStoreProvider } from './store/MainStoreContext';
import { RouterContext, Router } from '@happysanta/router';
import { PROFILE_PAGE, routes } from './routes';
import '@vkontakte/vkui/dist/vkui.css';
import '@vkontakte/vkui/dist/unstable.css';
import './reset.css';
import './index.css';
import { ErrorBoundary } from './components/ErrorBoundary';

Bridge.init();

configure({
  computedRequiresReaction: true,
  reactionRequiresObservable: true,
  observableRequiresReaction: true,
});

const root = document.getElementById('root') as HTMLElement;

const router = new Router(routes);
router.start();
router.pushPage(PROFILE_PAGE);

const app = (
  <RouterContext.Provider value={router}>
    <MainStoreProvider>
      <ErrorBoundary>
        <App />
```

```
      </ErrorBoundary>
    </MainStoreProvider>
  </RouterContext.Provider>
);

ReactDOM.render(app, root);
```

Файл src\components\AdaptiveLayout\index.tsx

```
import {
  Icon28SmartphoneStarsOutline,
  Icon28Cards2Outline,
  Icon24ServicesOutline,
  Icon28SettingsOutline,
  Icon28Profile,
} from '@vkontakte/icons';
import {
  Group,
  Panel,
  PanelHeader,
  SplitLayout,
  Tabbar,
  TabbarItem,
  View,
  ViewWidth,
  useAdaptivity,
  PanelHeaderBack,
} from '@vkontakte/vkui';
import { Epic } from
'@vkontakte/vkui/dist/components/Epic/Epic';
import { observer } from 'mobx-react-lite';
import { useMainStore } from '../../store/MainStoreContext';
import { DivinationTypeView } from
'../../views/DivinationTypeView';
import { Prediction } from '../../views/Prediction';
import { PredictionsList } from '../../views/PredictionsList';
```

```
import { Profile } from '../../views/Profile';
import { RuneDetail } from '../../views/RuneDetail';
import { RunesCanvas } from '../../views/RunesCanvas';
import { RunesList } from '../../views/RunesList';
import { Settings } from '../../views/Settings';
import { useLocation } from './useLocation';
import { useRouter } from '@happysanta/router';
import { Modals } from '../Modals';
import {
  CANVAS_PAGE,
  CANVAS_PANEL,
  CANVAS_VIEW,
  DESCRIPTIONS_PAGE,
  DESCRIPTIONS_PANEL,
  DESCRIPTIONS_VIEW,
  ERROR_PANEL,
  ERROR_VIEW,
  PREDICTIONS_PANEL,
  PREDICTION_PANEL,
  PROFILE_PAGE,
  PROFILE_PANEL,
  PROFILE_VIEW,
  RUNE_DETAIL_PANEL,
  SETTINGS_PAGE,
  SETTINGS_PANEL,
  SETTINGS_VIEW,
  TYPES_CHOOSING_PAGE,
  TYPES_CHOOSING_PANEL,
  TYPES_CHOOSING_VIEW,
} from '../../routes';
import { ErrorView } from '../../views/ErrorView';
import { ErrorBoundary } from '../ErrorBoundary';

export const AdaptiveLayout = observer(() => {
  const router = useRouter();
```

```
  const { getViewActivePanel, location } = useLocation();
  const store = useMainStore();

  const activeStory = location.getViewId();
  const adaptivityProps = useAdaptivity();

  const isDesktop = adaptivityProps.viewWidth >=
ViewWidth.TABLET;
  store.isMobile = !isDesktop;

  return (
    <SplitLayout modal={<Modals />} style={{ justifyContent:
'center' }}>
      <Epic
        {...adaptivityProps}
        activeStory={activeStory}
        tabbar={
          <Tabbar>
            <TabbarItem
              onClick={() => router.pushPage(PROFILE_PAGE)}
              selected={activeStory === PROFILE_VIEW}
              data-story="profile"
              text="Профиль"
            >
              <Icon28Profile />
            </TabbarItem>

            <TabbarItem
              onClick={() =>
router.pushPage(TYPES_CHOOSING_PAGE)}
              selected={activeStory === TYPES_CHOOSING_VIEW}
              data-story="main"
              text="Расклады"
            >
              <Icon28SmartphoneStarsOutline />
```

```
            </TabbarItem>

            <TabbarItem
              onClick={() => router.pushPage(CANVAS_PAGE)}
              selected={activeStory === CANVAS_VIEW}
              data-story="canvas"
              text="Играть"
            >
              <Icon24ServicesOutline />
            </TabbarItem>
            <TabbarItem
              onClick={() => router.pushPage(DESCRIPTIONS_PAGE)}
              selected={activeStory === DESCRIPTIONS_VIEW}
              data-story="runes-list"
              text="Описание"
            >
              <Icon28Cards2Outline />
            </TabbarItem>

            <TabbarItem
              onClick={() => router.pushPage(SETTINGS_PAGE)}
              selected={activeStory === SETTINGS_VIEW}
              data-story="settings"
              text="Настройки"
            >
              <Icon28SettingsOutline />
            </TabbarItem>
          </Tabbar>
        }
      >
        <View id={PROFILE_VIEW}
activePanel={getViewActivePanel(PROFILE_VIEW)}>
          <Panel id={PROFILE_PANEL}>
            <PanelHeader>Мой профиль</PanelHeader>
            <ErrorBoundary>
```

```
              <Profile />
            </ErrorBoundary>
          </Panel>

          <Panel id={PREDICTIONS_PANEL}>
            <PanelHeader left={<PanelHeaderBack onClick={() =>
router.popPage()} />}>
              Мои вопросы
            </PanelHeader>
            <ErrorBoundary>
              <PredictionsList />
            </ErrorBoundary>
          </Panel>

          <Panel id={PREDICTION_PANEL}>
            <PanelHeader left={<PanelHeaderBack onClick={() =>
router.popPage()} />}>
              Ответ
            </PanelHeader>
            <ErrorBoundary>
              <Prediction isNewPrediction={false} />
            </ErrorBoundary>
          </Panel>
        </View>

        <View
          id={TYPES_CHOOSING_VIEW}
          activePanel={getViewActivePanel(TYPES_CHOOSING_VIEW)}
        >
          <Panel id={TYPES_CHOOSING_PANEL}>
            <PanelHeader left={<PanelHeaderBack onClick={() =>
router.popPage()} />}>
              Выбор расклада
            </PanelHeader>
            <ErrorBoundary>
```

```
              <DivinationTypeView />
            </ErrorBoundary>
          </Panel>
        </View>

        <View id={CANVAS_VIEW}
activePanel={getViewActivePanel(CANVAS_VIEW)}>
          <Panel id={CANVAS_PANEL}>
            <PanelHeader left={<PanelHeaderBack onClick={() =>
router.popPage()} />}>
              Играть
            </PanelHeader>
            <ErrorBoundary>
              <RunesCanvas />
            </ErrorBoundary>
          </Panel>

          <Panel id={PREDICTION_PANEL}>
            <PanelHeader left={<PanelHeaderBack onClick={() =>
router.popPage()} />}>
              Результат
            </PanelHeader>
            <ErrorBoundary>
              <Prediction isNewPrediction />
            </ErrorBoundary>
          </Panel>
        </View>

        <View id={DESCRIPTIONS_VIEW}
activePanel={getViewActivePanel(DESCRIPTIONS_VIEW)}>
          <Panel id={DESCRIPTIONS_PANEL}>
            <PanelHeader left={<PanelHeaderBack onClick={() =>
router.popPage()} />}>
              Описание
            </PanelHeader>
```

```
            <ErrorBoundary>
              <RunesList />
            </ErrorBoundary>
          </Panel>

          <Panel id={RUNE_DETAIL_PANEL}>
            <PanelHeader left={<PanelHeaderBack onClick={() =>
router.popPage()} />}>
              Сведения о руне
            </PanelHeader>
            <ErrorBoundary>
              <RuneDetail />
            </ErrorBoundary>
          </Panel>
        </View>

        <View id={SETTINGS_VIEW}
activePanel={getViewActivePanel(SETTINGS_VIEW)}>
          <Panel id={SETTINGS_PANEL}>
            <PanelHeader left={<PanelHeaderBack onClick={() =>
router.popPage()} />}>
              Настройки
            </PanelHeader>
            <ErrorBoundary>
              <Settings />
            </ErrorBoundary>
          </Panel>
        </View>

        <View id={ERROR_VIEW}
activePanel={getViewActivePanel(ERROR_VIEW)}>
          <Panel id={ERROR_PANEL}>
            <PanelHeader left={<PanelHeaderBack onClick={() =>
router.popPage()} />}>
              {'Упс! Ошибка :('}
```

```
            </PanelHeader>
            <Group>
              <ErrorView />
            </Group>
          </Panel>
        </View>
      </Epic>
    </SplitLayout>
  );
});
```

Файл components\AdaptiveLayout\useLocation.ts

```
import { useCallback } from 'react';
import { useLocation as useHappySantaLocation } from
'@happysanta/router';

export const useLocation = () => {
  const location = useHappySantaLocation();

  const getViewActivePanel = useCallback(
    (viewId: string) => location.getViewActivePanel(viewId) ||
'',
    [location]
  );

  return { location, getViewActivePanel };
};
```

Файл src\components\ErrorBoundary\index.tsx

```
import { RouterProps, withRouter } from '@happysanta/router';
import { Component, ReactNode } from 'react';
import { ERROR_PAGE } from '../../routes';

interface Props extends RouterProps {
  children: ReactNode;
```

```
}

class ErrorBoundaryComponent extends Component<Props, {
hasError: boolean }> {
  constructor(props: Props) {
    super(props);
    this.state = { hasError: false };
  }

  componentDidCatch() {
    this.setState({ hasError: true });
  }

  render() {
    if (this.state.hasError) {
      this.props.router.pushPage(ERROR_PAGE);
      return null;
    }

    return this.props.children;
  }
}

export const ErrorBoundary: any =
withRouter(ErrorBoundaryComponent);
```

Файл components\Modals\hooks\usePredictionResult.ts

```
import { useMainStore } from '../../../store/MainStoreContext';

export const usePredictionResult = () => {
  const { closeAllRunes } = useMainStore();

  return { closeAllRunes };
};
```

Файл components\Modals\hooks\usePredictionSaving.ts

```
import { useMainStore } from
'./../../../store/MainStoreContext';

export const usePredictionSaving = () => {
  const {
    openedRunes,
    predictionTitle,
    isPredictionSaved,
    setActiveModal,
    setIsPredictionSaved,
    setPredictionTitle,
    saveUserPrediction,
  } = useMainStore();

  const handlePredictionSaving = async () => {
    setIsPredictionSaved(true);
    const result = await saveUserPrediction({ runes:
openedRunes, title: predictionTitle });

    if (result === 1) {
      alert('Ne vishlo');
    }

    setActiveModal();
    setPredictionTitle('');
  };

  return { isPredictionSaved, predictionTitle,
setPredictionTitle, handlePredictionSaving };
};
```

Файл src\components\Modals\index.tsx

```
import {
  Icon28ClearDataOutline,
```

```
  Icon28StarsOutline,
  Icon28CancelOutline,
  Icon28FolderOutline,
} from '@vkontakte/icons';
import { Button, FormItem, Input, ModalCard, ModalRoot,
PanelSpinner } from '@vkontakte/vkui';
import { observer } from 'mobx-react-lite';
import { useRouter } from '@happysanta/router';
import { useMainStore } from '../../store/MainStoreContext';
import { usePredictionResult } from
'./hooks/usePredictionResult';
import { usePredictionSaving } from
'./hooks/usePredictionSaving';
import { NEW_PREDICION_PAGE } from '../../routes';

export const Modals = observer(() => {
  const { appLoading, activeModal, setActiveModal } =
useMainStore();
  const { closeAllRunes } = usePredictionResult();
  const { isPredictionSaved, predictionTitle,
setPredictionTitle, handlePredictionSaving } =
    usePredictionSaving();
  const router = useRouter();

  return (
    <ModalRoot activeModal={activeModal}>
      <ModalCard
        id="prediction-result"
        header="Выбор сделан!"
        subheader="Предсказание готово"
        actions={[
          <Button before={<Icon28ClearDataOutline />} size="m"
onClick={closeAllRunes}>
            Заново
          </Button>,
```

```
      <Button
        before={<Icon28StarsOutline />}
        size="m"
        onClick={() => router.pushPage(NEW_PREDICION_PAGE)}
      >
        Смотреть
      </Button>,
    ]}
  ></ModalCard>

  <ModalCard
    id="prediction-saving"
    header="Заметка к предсказанию"
    subheader="Какой вопрос был задан, для гадания"
    actions={[
      <Button
        before={<Icon28CancelOutline />}
        size="m"
        onClick={() => setActiveModal()}
        key="cancel"
      >
        Отмена
      </Button>,

      <Button
        before={<Icon28FolderOutline />}
        disabled={isPredictionSaved}
        size="m"
        onClick={handlePredictionSaving}
        key="save"
      >
        Сохранить
      </Button>,
    ]}
```

```
      >
        <FormItem top="Ваш вопрос">
          {appLoading.isPredictionSavingLoading ? (
            <PanelSpinner />
          ) : (
            <Input
              value={predictionTitle}
              onChange={({ target }) =>
setPredictionTitle(target.value)}
              type="text"
            />
          )}
        </FormItem>
      </ModalCard>
    </ModalRoot>
  );
});
```

Файл src\components\Rune\index.tsx

```
import { classNames as cn } from '@vkontakte/vkui';
import { observer } from 'mobx-react-lite';
import { useState } from 'react';
import { RuneColor, RuneKeyFull } from '../../types';
import { getHslCssColor } from '../../helpers/getHslCssColor';
import styles from './style.module.css';

interface Props {
  className?: string;
  name: RuneKeyFull;
  isOpen?: boolean;
  color?: RuneColor;
  size?: 'sm' | 'md' | 'lg' | 'xl';
  shouldInvertLabel?: boolean;
  isDisabled?: boolean;
  hasHover?: boolean;
```

```
  onClick?: (rune: RuneKeyFull, isInverted: boolean) => void;
}

export const Rune = ({
  name,
  shouldInvertLabel = false,
  isOpen,
  color,
  className,
  size = 'md',
  hasHover = true,
  isDisabled,
  onClick,
}: Props) => {
  const [isHovered, setIsHovered] = useState(false);
  const isInverted = name === name.toUpperCase();
  const label = shouldInvertLabel ? name : name.toLowerCase();

  return (
    <button
      onMouseEnter={() => hasHover && setIsHovered(true)}
      onMouseLeave={() => hasHover && setIsHovered(false)}
      style={{
        ...(color
          ? {
              backgroundColor: getHslCssColor({
                hslColor: color.body,
                lightnessDiff: -5,
                isHovered: hasHover && isHovered,
              }),
              color: getHslCssColor({
                hslColor: color.label,
                lightnessDiff: 5,
                isHovered: hasHover && isHovered,
              }),
```

```
              borderColor: getHslCssColor({
                hslColor: color.border,
                lightnessDiff: -5,
                isHovered: hasHover && isHovered,
              }),
            }
          : {}),
      }}
      onClick={() => !isDisabled && onClick?.(name, isInverted)}
      className={cn(
        styles.wrapper,

        {
          [styles.smallSize]: size === 'sm',
          [styles.mediumSize]: size === 'md',
          [styles.largeSize]: size === 'lg',
          [styles.xtraLargeSize]: size === 'xl',
          [styles.withHover]: hasHover && !isDisabled,
          [styles.disabledRune]: isDisabled,
        },
        className
      )}
    >
      {isOpen ? label : ''}
    </button>
  );
};

export const RuneObserved = observer(Rune);
```

Файл src\components\SpinnerWithBackdrop\index.tsx

```
import type { FC } from 'react';
import React, { useState, useEffect } from 'react';
```

```
import { moveIntoViewportCenter } from
'./moveIntoViewportCenter';
import { useDisabledTabIndex } from './useDisabledTabIndex';
import { Spinner } from '@vkontakte/vkui';
import styles from './style.module.css';
import { classNames as cn } from '@vkontakte/vkjs';

interface Props {
  isLoading?: boolean;
  className?: string;
  spinnerClassName?: string;
  opacBackdrop?: boolean;
  size?: 'small' | 'regular' | 'large' | 'medium';
}

export const SpinnerWithBackdrop: FC<Props> = ({
  children,
  isLoading = false,
  size,
  opacBackdrop = false,
  className,
  spinnerClassName,
}) => {
  const [container, setContainer] = useState<HTMLDivElement |
null>(null);
  const [spinComponent, setSpinComponent] =
useState<HTMLDivElement | null>(null);

  useEffect(() => {
    if (isLoading && container && spinComponent) {
      moveIntoViewportCenter({ container, childComponent:
spinComponent });
    }
  }, [isLoading, size, container, spinComponent]);
```

```
  useDisabledTabIndex(container, isLoading);

  return (
    <div
      className={cn(styles.root, className)}
      ref={(ref) => setContainer(ref)}
      data-testid="spinner-with-backdrop"
    >
      {isLoading && (
        <div ref={(ref) => setSpinComponent(ref)}
className={styles.spin}>
          <Spinner size={size} className={spinnerClassName} />
        </div>
      )}
      <div
        className={cn({
          [styles.spinContainerBlur]: isLoading,
          [styles.opacBackdrop]: isLoading && opacBackdrop,
        })}
      >
        {children}
      </div>
    </div>
  );
};
```

Файл components\SpinnerWithBackdrop\moveIntoViewportCenter.ts

```
const getViewportSize = (documentViewportSize: number,
startCoord: number, endCoord: number) => {
  let viewportSize = documentViewportSize;

  if (startCoord > 0) {
    viewportSize -= startCoord;
  }
  if (endCoord < documentViewportSize) {
```

```
    viewportSize -= documentViewportSize - endCoord;
  }

  return viewportSize;
};

const getCoord = (
  containerSize: number,
  containerViewportSize: number,
  childSize: number,
  offset: number
) => {
  let coord = containerViewportSize / 2 + offset - childSize /
2;

  if (coord < 0) {
    coord = 0;
  }
  if (offset >= containerSize - childSize) {
    coord = containerSize - childSize;
  }

  return coord;
};

interface Params {
  container: HTMLDivElement | null;
  childComponent: HTMLDivElement | null;
  prevChildSize?: number;
  prevContainerTop?: number;
  prevContainerWidth?: number;
  prevDocumenHeight?: number;
}

export const moveIntoViewportCenter = ({
```

```
    container,
    childComponent,
    prevChildSize,
    prevContainerTop,
    prevContainerWidth,
    prevDocumenHeight,
  }: Params) => {
    if (!container || !childComponent) {
      return null;
    }

    const childSize =
  childComponent.getBoundingClientRect().width;
    if (prevChildSize && childSize !== prevChildSize) {
      return null;
    }

    const {
      top: containerTop,
      bottom: containerBottom,
      height: containerHeight,
      width: containerWidth,
    } = container.getBoundingClientRect();

    const documentViewportHeight: number =
  document.documentElement.clientHeight;

    if (
      prevContainerTop === containerTop &&
      prevContainerWidth === containerWidth &&
      documentViewportHeight === prevDocumenHeight &&
      prevChildSize === childSize
    ) {
      return requestAnimationFrame(() => {
        moveIntoViewportCenter({
```

```
          container,
          childComponent,
          prevChildSize: childSize,
          prevContainerTop,
          prevContainerWidth,
          prevDocumenHeight: documentViewportHeight,
        });
      });
    }

    const contanerViewportHeight = getViewportSize(
      documentViewportHeight,
      containerTop,
      containerBottom
    );

    const topOffset = containerTop < 0 ? -containerTop : 0;
    const top = getCoord(containerHeight, contanerViewportHeight,
childSize, topOffset);
    const left = containerWidth / 2 - childSize / 2;

    childComponent.style.transform = `translate(${left}px,
${top}px)`;
    if (childComponent.style.display !== 'block') {
      childComponent.style.display = 'block';
    }

    return requestAnimationFrame(() => {
      moveIntoViewportCenter({
        container,
        childComponent,
        prevChildSize: childSize,
        prevContainerTop: containerTop,
        prevContainerWidth: containerWidth,
        prevDocumenHeight: documentViewportHeight,
```

```
    });
  });
};
```

Файл components\SpinnerWithBackdrop\useDisabledTabIndex.ts

```
import { useEffect, useRef } from 'react';

export const useDisabledTabIndex = (container: HTMLElement |
null, isLoading: boolean) => {
  const tabIndexes = useRef(new Map());

  useEffect(() => {
    if (!container) {
      return;
    }

    const interactiveElements: HTMLElement[] = Array.from(
      container.querySelectorAll('button, input, select,
textarea, [tabindex]')
    );

    if (isLoading) {
      return interactiveElements.forEach((el) => {
        tabIndexes.current.set(el, el.tabIndex);
        el.tabIndex = -1;
      });
    }

    if (tabIndexes.current.size === 0) {
      return;
    }

    interactiveElements.forEach((el) => {
      const tabIndex = tabIndexes.current.get(el);
      el.tabIndex = tabIndex;
```

```
      tabIndexes.current.delete(el);
    });
  }, [container, isLoading]);

  return tabIndexes.current;
};
```

Файл constants\index.ts

```
import { Divination } from '../types';

export const divinationsData: Divination = {
  '1': {
    name: 'Руна дня',
    description:
      'Выпавшая руна характеризует события сегодняшнего дня или
отвечает на заданный вами вопрос.',
    stages: [{ name: 'Ответ на ваш вопрос', description: 'или
предсказание на день' }],
  },

  '3': {
    name: 'Три руны',
    stages: [
      {
        name: '1. Сейчас',
        description: 'текущая ситуация',
      },
      {
        name: '2. Будет',
        description: 'возможное развитие ситуации',
      },
      {
        name: '3. Действие',
        description: 'оптимальный вариант действий',
      },
```

```
    ],
  },

  '4': {
    name: 'Четыре руны',
    stages: [
      {
        name: '1. Сейчас',
        description: 'текущая ситуация',
      },
      {
        name: '2. Позитивные силы',
        description: 'позитивные силы, развивающие ситуацию',
      },
      {
        name: '3. Негативные силы',
        description: 'враждебные силы',
      },
      {
        name: '4. Развитие ситуации',
        description: 'возможное развитие ситуации, будущее,
которое может наступить.',
      },
    ],
  },
};

export const PREDICTIONS_PAGE_SIZE = 10;
```

Файл helpers\dragAndDrop.ts

```
interface Params {
  element: HTMLElement;
  container?: HTMLDivElement;
  setIsMoving: (isMoving: boolean) => void;
  setRandomRune: (x: number, y: number) => void;
```

```
}

let cachedWidth: number;
let cachedHeight: number;

export const dragAndDrop = ({ element, container, setIsMoving,
setRandomRune }: Params) => {
  let isDragging = false;
  let order = 29;
  let resultX: number;
  let resultY: number;
  let containerWidth: number = 100000;
  let containerHeight: number = 100000;
  const borderOffset = 14;

  if (typeof cachedWidth === 'number' && typeof cachedHeight ===
'number') {
    containerWidth = cachedWidth;
    containerHeight = cachedHeight;
  } else {
    if (container) {
      const { width, height } =
container.getBoundingClientRect();
      containerWidth = width;
      containerHeight = height;
      cachedWidth = width;
      cachedHeight = height;
    }
  }

  element.addEventListener('mousedown', function (event) {
    const { left: startX, top: startY, width, height } =
element.getBoundingClientRect();
    const size = (height + width) / 2;
```

```
    const [offsetX, offsetY, turn] =
element.style.transform.match(/-?\d+\.?\d*/g)!;
    const shiftX = event.clientX - startX;
    const shiftY = event.clientY - startY;
    const transition = element.style.transition;

    function moveAt(e: MouseEvent) {
      if (!isDragging) {
        setIsMoving(true);
      }

      isDragging = true;
      element.style.zIndex = '1001';
      let x = Number(offsetX) + e.x - startX - shiftX;
      let y = Number(offsetY) + e.y - startY - shiftY;

      if (x < borderOffset) {
        x = borderOffset;
      }
      if (x > containerWidth - size - borderOffset) {
        x = containerWidth - size - borderOffset;
      }

      if (y < 4) {
        y = 4;
      }
      if (y > containerHeight - height - borderOffset) {
        y = containerHeight - height - borderOffset;
      }

      element.style.transform = `translate(${x}px, ${y}px)
rotate(${turn}turn)`;

      resultX = x;
      resultY = y;
```

```
      element.style.transition = 'none';
    }

    document.addEventListener('mousemove', moveAt);

    document.addEventListener(
      'mouseup',
      () => {
        document.removeEventListener('mousemove', moveAt);
        element.style.transition = transition;
        element.style.zIndex = `${order}`;
        order += 1;
        setRandomRune(resultX, resultY);
        element.onmouseup = null;
      },
      { once: true }
    );
  });

  element.ondragstart = function () {
    return false;
  };

  element.onclick = (e) => {
    if (isDragging) {
      e.preventDefault();
      e.stopPropagation();
      isDragging = false;
      setIsMoving(false);
    }
  };
};
```

Файл helpers\dragAndDropMobile.ts

```
interface Params {
```

```
  element: HTMLElement;
  container?: HTMLDivElement;
  setIsMoving: (p: boolean) => void;
  setRandomRune: (x: number, y: number) => void;
}

let cachedWidth: number;
let cachedHeight: number;

export const dragAndDropMobile = ({ element, container,
setIsMoving, setRandomRune }: Params) => {
  let isDragging = false;
  let order = 29;
  let resultX: number;
  let resultY: number;
  let containerWidth: number = 100000;
  let containerHeight: number = 100000;
  const borderOffset = 14;

  if (typeof cachedWidth === 'number' && typeof cachedHeight ===
'number') {
    containerWidth = cachedWidth;
    containerHeight = cachedHeight;
  } else {
    if (container) {
      const { width, height } =
container.getBoundingClientRect();
      containerWidth = width;
      containerHeight = height;
      cachedWidth = width;
      cachedHeight = height;
    }
  }

  element.addEventListener('touchstart', (event) => {
```

```
    const touchStartEvent = event.targetTouches[0];
    const { left: startX, top: startY, width, height } =
element.getBoundingClientRect();
    const size = (height + width) / 2;
    const [offsetX, offsetY, turn] =
element.style.transform.match(/-?\d+\.?\d*/g)!;
    const shiftX = touchStartEvent.clientX - startX;
    const shiftY = touchStartEvent.clientY - startY;
    const transition = element.style.transition;

    function moveAt(event: TouchEvent) {
      const touchMoveEvent = event.targetTouches[0];
      if (!isDragging) {
        setIsMoving(true);
      }

      isDragging = true;
      element.style.zIndex = '1001';
      let x = Number(offsetX) + touchMoveEvent.clientX - startX
- shiftX;
      let y = Number(offsetY) + touchMoveEvent.clientY - startY
- shiftY;

      if (x < borderOffset) {
        x = borderOffset;
      }
      if (x > containerWidth - size - borderOffset) {
        x = containerWidth - size - borderOffset;
      }

      if (y < 4) {
        y = 4;
      }
      if (y > containerHeight - height - borderOffset) {
        y = containerHeight - height - borderOffset;
```

```
      }

      element.style.transform = `translate(${x}px, ${y}px)
rotate(${turn}turn)`;

      resultX = x;
      resultY = y;
      element.style.transition = 'none';
    }

    element.addEventListener('touchmove', moveAt);

    document.addEventListener(
      'touchend',
      () => {
        element.removeEventListener('touchmove', moveAt);
        element.style.transition = transition;
        element.style.zIndex = `${order}`;
        order += 1;
        setRandomRune(resultX, resultY);
        element.onmouseup = null;
      },
      { once: true }
    );
  });

  element.ondragstart = function () {
    return false;
  };

  element.onclick = (e) => {
    if (isDragging) {
      e.preventDefault();
      e.stopPropagation();
      isDragging = false;
```

```
      setIsMoving(false);
    }
  };
};
```

Файл helpers\filterRunes.ts

```
import { RuneKey, RuneKeyFull } from '../types';

export const filterRunes = (runeToExclude: RuneKey |
RuneKeyFull, runesKit: RuneKeyFull[]) => {
  return runesKit.filter(
    (r) => r !== runeToExclude.toLowerCase() && r !==
runeToExclude.toUpperCase()
  );
};
```

Файл helpers\findRune.ts

```
import { RuneKey, RuneKeyFull } from '../types';

export const findRune = (rune: RuneKey | RuneKeyFull, runesKit:
RuneKeyFull[]) => {
  return runesKit.find((r) => r === rune.toLowerCase() || r ===
rune.toUpperCase());
};
```

Файл helpers\getHslCssColor.ts

```
import { HslColor } from 'react-colorful';

interface Params {
  hslColor?: HslColor;
  lightnessDiff?: number;
  isHovered?: boolean;
}

export const getHslCssColor = ({ hslColor, lightnessDiff = 0,
isHovered = false }: Params) =>
```

```
  hslColor
    ? `hsl(${hslColor.h}, ${hslColor.s}%, ${isHovered ?
hslColor.l + lightnessDiff : hslColor.l}%)`
    : '';
```

Файл helpers\getPosition.ts

```
export type Size = { width: number; height: number };
export type Coord = { x: number; y: number };
type CoordDeg = { rotation: number };

export const getPosition = (offset: number, container: Size) =>
{
  const position: Coord & CoordDeg = { x: 0, y: 0, rotation: 0
};

  position.x = Math.random() * (container.width - offset) + 5;
  position.y = Math.random() * (container.height - offset);
  position.rotation = Math.random();

  return position;
};
```

Файл src\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { configure } from 'mobx';
import { Bridge } from './modules/Bridge';

import App from './App';
import { MainStoreProvider } from './store/MainStoreContext';
import { RouterContext, Router } from '@happysanta/router';
import { PROFILE_PAGE, routes } from './routes';
import '@vkontakte/vkui/dist/vkui.css';
import '@vkontakte/vkui/dist/unstable.css';
import './reset.css';
import './index.css';
```

```
import { ErrorBoundary } from './components/ErrorBoundary';

Bridge.init();

configure({
  computedRequiresReaction: true,
  reactionRequiresObservable: true,
  observableRequiresReaction: true,
});

const root = document.getElementById('root') as HTMLElement;

const router = new Router(routes);
router.start();
router.pushPage(PROFILE_PAGE);

const app = (
  <RouterContext.Provider value={router}>
    <MainStoreProvider>
      <ErrorBoundary>
        <App />
      </ErrorBoundary>
    </MainStoreProvider>
  </RouterContext.Provider>
);

ReactDOM.render(app, root);
```

Файл modules\Bridge.ts

```
import { BridgePlus } from '@happysanta/bridge-plus';
import { VKStorageKeys } from '../types';

window.Buffer = window.Buffer || require('buffer').Buffer;

export class Bridge extends BridgePlus {
```

```
  static async storageGetKey(key: VKStorageKeys, defaultValue?:
string) {
    const result = await super.storageGetKey(key, defaultValue);
    return this._base64ToString(result);
  }

  static storageGetKeys(count?: number, offset?: number) {
    return super.storageGetKeys(count, offset) as Promise<{
keys: VKStorageKeys[] }>;
  }

  static async storageGet(keys: VKStorageKeys[]) {
    const result = await super.storageGet(keys);
    return {
      keys: result.keys.map(({ key, value }) => ({
        key,
        value: this._base64ToString(value),
      })) as {
        key: VKStorageKeys;
        value: string;
      }[],
    };
  }

  static storageSet(key: VKStorageKeys, value: string) {
    return super.storageSet(key, this._stringToBase64(value));
  }

  static storageDelete(key: VKStorageKeys) {
    return super.storageSet(key, '');
  }

  static _stringToBase64(str: string) {
    return Buffer.from(str, 'utf8').toString('base64');
  }
```

```
  static _base64ToString(base64Str: string) {
    return Buffer.from(base64Str, 'base64').toString('utf8');
  }
}
```

Файл \routes.ts

```
import { Page } from '@happysanta/router';

export const PROFILE_PAGE = '/profile';
export const PROFILE_VIEW = 'profile';
export const PROFILE_PANEL = 'profile-panel';

export const PREDICTIONS_PAGE = `${PROFILE_PAGE}/predictions`;
export const PREDICTIONS_PANEL = 'predictions-list-panel';

export const CANVAS_PAGE = '/canvas';
export const CANVAS_VIEW = 'canvas';
export const CANVAS_PANEL = 'canvas-panel';

export const NEW_PREDICION_PAGE = `/new-prediction`;
export const STORED_PREDICION_PAGE = `/stored-prediction`;
export const PREDICTION_PANEL = 'prediction-panel';

export const TYPES_CHOOSING_PAGE = '/types-choosing';
export const TYPES_CHOOSING_VIEW = 'types-choosing';
export const TYPES_CHOOSING_PANEL = 'types-choosing-panel';

export const DESCRIPTIONS_PAGE = '/descriptions';
export const DESCRIPTIONS_VIEW = 'descriptions';
export const DESCRIPTIONS_PANEL = 'descriptions-panel';

export const RUNE_DETAIL_PAGE = `${DESCRIPTIONS_PAGE}/rune-
detail`;
export const RUNE_DETAIL_PANEL = 'rune-detail-panel';
```

```
export const SETTINGS_PAGE = '/settings';
export const SETTINGS_VIEW = 'settings';
export const SETTINGS_PANEL = 'settings-panel';

export const ERROR_PAGE = '/error-page';
export const ERROR_VIEW = 'error';
export const ERROR_PANEL = 'error-panel';

export const routes = {
  '/': new Page(PROFILE_PANEL, PROFILE_VIEW),

  [PROFILE_PAGE]: new Page(PROFILE_PANEL, PROFILE_VIEW),

  [PREDICTIONS_PAGE]: new Page(PREDICTIONS_PANEL, PROFILE_VIEW),

  [CANVAS_PAGE]: new Page(CANVAS_PANEL, CANVAS_VIEW),

  [NEW_PREDICION_PAGE]: new Page(PREDICTION_PANEL, CANVAS_VIEW),

  [STORED_PREDICION_PAGE]: new Page(PREDICTION_PANEL,
PROFILE_VIEW),

  [TYPES_CHOOSING_PAGE]: new Page(TYPES_CHOOSING_PANEL,
TYPES_CHOOSING_VIEW),

  [DESCRIPTIONS_PAGE]: new Page(DESCRIPTIONS_PANEL,
DESCRIPTIONS_VIEW),

  [RUNE_DETAIL_PAGE]: new Page(RUNE_DETAIL_PANEL,
DESCRIPTIONS_VIEW),

  [SETTINGS_PAGE]: new Page(SETTINGS_PANEL, SETTINGS_VIEW),

  [ERROR_PAGE]: new Page(ERROR_PANEL, ERROR_VIEW),
```

```
};
```

Файл services\loadPredictions.ts

```
import { PredictionKey, PredictionWithKey } from './../types';
import { Bridge } from '../modules/Bridge';
import { PREDICTIONS_PAGE_SIZE } from '../constants';

export const loadPredictions = async (
  page: number
): Promise<{ predictions: PredictionWithKey[]; predictionsCount:
number }> => {
  try {
    const storageKeys = (await Bridge.storageGetKeys(1000,
0)).keys;

    const predictionKeys = storageKeys
      .filter((key) => /PREDICTION_\d+$/.test(key))
      .reverse() as PredictionKey[];

    const offset = (page - 1) * PREDICTIONS_PAGE_SIZE;
    const predictions = (
      await Bridge.storageGet(
        predictionKeys.slice(offset, offset +
PREDICTIONS_PAGE_SIZE)
      )
    ).keys.filter(({ value }) => Boolean(value));

    return {
      predictions: predictions
        .filter(({ value }) => Boolean(value))
        .map(({ key, value }) => ({ ...JSON.parse(value), key }
as PredictionWithKey)),
      predictionsCount: predictionKeys.length,
    };
  } catch (error) {
```

```
    console.error(error);
    return { predictions: [], predictionsCount: 0 };
  }
};
```

Файл services\savePrediction.ts

```
import { Bridge } from '../modules/Bridge';
import { Prediction, RuneKeyFull, VKStorageKeys } from
'../types';

interface Params {
  title: string;
  runes: RuneKeyFull[];
  description?: string;
  ownKitName?: string | null;
}

export const savePrediction = async ({ runes, title,
description, ownKitName = null }: Params) => {
  try {
    const predictionDate = Date.now();

    const newPrediction: Prediction = {
      runes,
      date: predictionDate,
      description,
      title,
      ownKitName,
    };

    const storageKeys = await Bridge.storageGetKeys();

    const predictionsCount =
storageKeys.keys.reduce<number>((acc, key) => {
      if (key.startsWith('PREDICTION_')) {
```

```
        return acc + 1;
      }

      return acc;
    }, 0);

    if (predictionsCount > 800) {
      return Promise.resolve(1);
    }

    const newPredictionKey: VKStorageKeys =
`PREDICTION_${predictionDate}`;

    return Bridge.storageSet(newPredictionKey,
JSON.stringify(newPrediction));
  } catch (error) {
    console.error(error);
  }
};
```

Файл store\MainStore.ts

```
import { UserInfo } from '@vkontakte/vk-bridge';
import { action, makeObservable, observable, runInAction } from
'mobx';
import { HslColor } from 'react-colorful';
import { filterRunes } from '../helpers/filterRunes';
import { findRune } from '../helpers/findRune';
import { Coord, getPosition } from '../helpers/getPosition';
import { Bridge } from '../modules/Bridge';
import { runes } from '../runes';
import { loadPredictions } from '../services/loadPredictions';
import { savePrediction } from '../services/savePrediction';
import {
  PositionedRune,
  RuneKeyFull,
```

```
  RuneKey,
  DivinationRunesCount,
  Setting,
  SettingField,
  RuneColor,
  CanvasColor,
  ModalName,
  PredictionWithKey,
  Prediction,
  AppLoading,
  PredictionKey,
} from '../types';

const defaultSettings = JSON.stringify({
  canRefreshByShake: false,
  canvasColor: { body: '#3a1d11', border: '#706d6b' },
  runeColor: {
    body: { h: 0, s: 0, l: 59 },
    border: { h: 0, s: 95, l: 8 },
    label: { h: 0, s: 100, l: 24 },
    slot: '#000000',
  },
  shouldShowResultImmidiatly: false,
  shouldHideRuneLabel: true,
  isDarkTheme: false,
} as Setting);

const initLoading: AppLoading = {
  isSettingsLoading: false,
  isPredictionsLoading: false,
  isUserProfileLoading: false,
  isPredictionSavingLoading: false,
  isPredictionDeletingLoading: false,
};
class MainStore {
```

```
  user?: UserInfo;
  appLoading: AppLoading = initLoading;
  isCanvasRefreshed = false;
  settings: Setting = JSON.parse(defaultSettings);
  tempSettings: Setting = JSON.parse(defaultSettings);
  activeModal?: ModalName;
  openedRunes: RuneKeyFull[] = [];
  currentPrediction?: Prediction;
  closedRunes: RuneKeyFull[] = [...runes];
  currentRune?: RuneKeyFull;
  randomRunes: PositionedRune[] = [];
  canvas?: HTMLDivElement;
  divinationType: DivinationRunesCount = '1';
  wasResultShown = false;
  isPredictionSaved = false;
  predictionTitle = '';

  wereSettingsSaved = true;
  wereSettingsChanged = false;
  isDefaultSettings = true;

  userPredictions?: PredictionWithKey[];
  predictionPage = 1;

  isMobile = false;
  isMobilePlatform = false;

  constructor() {
    this.loadUserSettings();

    makeObservable(this, {
      user: observable.ref,
      randomRunes: observable.ref,
      openedRunes: observable.ref,
      currentPrediction: observable.ref,
```

```
        isCanvasRefreshed: observable,
        isPredictionSaved: observable,
        appLoading: observable,

        wereSettingsSaved: observable,
        wereSettingsChanged: observable,
        isDefaultSettings: observable,

        setRandomRune: action.bound,
        openRune: action.bound,
        openPrediction: action.bound,
        closeRune: action.bound,
        closeAllRunes: action.bound,
        refresh: action.bound,
        setIsPredictionSaved: action.bound,

        wasResultShown: observable,
        setWasResultShown: action.bound,

        userPredictions: observable.ref,
        predictionPage: observable,
        setPredictionPage: action.bound,
        loadUserPredictions: action.bound,
        loadUserPredctionsPage: action.bound,
        loadUserInfo: action.bound,
        deleteUserPrediction: action.bound,
        saveUserPrediction: action.bound,

        predictionTitle: observable,
        setPredictionTitle: action.bound,

        activeModal: observable,
        setActiveModal: action.bound,

        currentRune: observable,
```

```
      setCurrectRune: action.bound,

      divinationType: observable,
      setDivinationType: action.bound,

      settings: observable.ref,
      tempSettings: observable.deep,
      setTempSettings: action.bound,
      saveSettings: action.bound,
      resetAllSettings: action.bound,
      resetTempSettings: action.bound,

      isMobilePlatform: observable,
      reloadAll: action.bound,
    });

    this.initPlatform();
  }

  initPlatform() {
    document.addEventListener(
      'touchstart',
      () => {
        runInAction(() => {
          this.isMobilePlatform = true;
        });
      },
      { once: true }
    );
  }

  loadUserSettings = async () => {
    const userSettingsData = await
Bridge.storageGetKey('USER_SETTINGS');
```

```
    runInAction(() => {
      const defaultSettingsObject = JSON.parse(defaultSettings);
      const userSettings = userSettingsData
        ? JSON.parse(userSettingsData)
        : defaultSettingsObject;
      this.settings = userSettings;
      this.tempSettings = { ...defaultSettingsObject,
...userSettings };

      this.isDefaultSettings = userSettingsData ===
defaultSettings;
    });
  };

  setIsPredictionSaved(isSaved: boolean) {
    this.isPredictionSaved = isSaved;
  }

  setCurrectRune(rune: RuneKey) {
    this.currentRune = rune;
  }

  setDivinationType(type: DivinationRunesCount) {
    this.divinationType = type;
  }

  async loadUserInfo() {
    this.appLoading.isUserProfileLoading = true;
    this.user = await Bridge.getUserInfo();

    runInAction(() => {
      this.appLoading.isUserProfileLoading = true;
    });
  }
```

```
  async loadUserPredctionsPage(page: number) {
    this.appLoading.isPredictionsLoading = true;
    const { predictions, predictionsCount } = await
loadPredictions(page);

    if (predictionsCount > 0 && predictions.length === 0) {
      return false;
    }

    runInAction(() => {
      this.appLoading.isPredictionsLoading = false;

      this.userPredictions = predictions;
    });

    return true;
  }

  async loadUserPredictions() {
    while (!(await
this.loadUserPredctionsPage(this.predictionPage))) {
      runInAction(() => (this.predictionPage -= 1));
    }
  }

  async deleteUserPrediction(predictionKey: PredictionKey) {
    this.appLoading.isPredictionDeletingLoading = true;
    await Bridge.storageDelete(predictionKey);
    this.loadUserPredictions();

    runInAction(() => {
      this.appLoading.isPredictionDeletingLoading = false;
    });
  }
```

```
  async saveUserPrediction(predictionProps: {
    runes: RuneKeyFull[];
    title: string;
    description?: string;
  }) {
    this.appLoading.isPredictionSavingLoading = true;
    const result = await savePrediction(predictionProps);
    this.loadUserPredictions();

    runInAction(() => {
      this.appLoading.isPredictionSavingLoading = false;
    });

    return result;
  }

  setPredictionTitle(title: string) {
    this.predictionTitle = title;
  }

  setPredictionPage(page: number) {
    this.predictionPage = page;
  }

  setTempSettings(value: string | boolean | HslColor, field:
SettingField) {
    this.wereSettingsChanged = true;
    this.wereSettingsSaved = false;

    if (field in this.tempSettings) {
      return (this.tempSettings[
        field as keyof Omit<Setting, 'runeColor' |
'canvasColor'>
      ] = value as boolean);
    }
```

```
    const [rootField, colorField] = field.split('.');
    if (rootField === 'rune') {
      return (this.tempSettings.runeColor[colorField as keyof
RuneColor] =
        value as HslColor);
    }

    if (rootField === 'canvas') {
      this.tempSettings.canvasColor[colorField as keyof
CanvasColor] = value as string;
    }
  }

  saveSettings() {
    if (this.isDefaultSettings) {
      this.wereSettingsChanged = false;
      this.wereSettingsSaved = true;
      this.settings = JSON.parse(defaultSettings);

      return Bridge.storageDelete('USER_SETTINGS');
    }

    const newSettingsJson = JSON.stringify(this.tempSettings);

    Bridge.storageSet('USER_SETTINGS', newSettingsJson);

    this.isDefaultSettings = newSettingsJson ===
defaultSettings;
    this.wereSettingsChanged = false;
    this.wereSettingsSaved = true;

    this.settings =
JSON.parse(JSON.stringify(this.tempSettings));
  }
```

```
  resetTempSettings() {
    this.wereSettingsChanged = false;
    this.wereSettingsSaved = true;

    const currentSettings = JSON.stringify(this.settings);

    this.tempSettings = JSON.parse(currentSettings);
    this.isDefaultSettings = currentSettings ===
defaultSettings;
  }

  resetAllSettings() {
    this.wereSettingsChanged = true;
    this.wereSettingsSaved = false;
    this.isDefaultSettings = true;

    this.tempSettings = JSON.parse(defaultSettings);
  }

  openRune(rune: RuneKeyFull) {
    if (!findRune(rune, this.openedRunes)) {
      this.openedRunes = [...this.openedRunes, rune];
    }
    this.closedRunes = filterRunes(rune, this.closedRunes);
  }

  openPrediction(prediction: Prediction) {
    this.currentPrediction = prediction;
  }

  closeRune(rune: RuneKeyFull) {
    if (!findRune(rune, this.closedRunes)) {
      this.closedRunes = [...this.closedRunes, rune];
    }
```

```
    this.openedRunes = filterRunes(rune, this.openedRunes);
    this.mergeRandomRunes(this.getRandomRunes([rune]));
    this.isCanvasRefreshed = true;
  }

  closeAllRunes() {
    this.closedRunes = [...runes];
    const returnedRunes = this.getRandomRunes(this.openedRunes);
    this.mergeRandomRunes(returnedRunes);
    this.openedRunes = [];
    this.isCanvasRefreshed = true;
    this.wasResultShown = false;
    this.isPredictionSaved = false;
  }

  setRandomRune(newRune: { x: number; y: number; name: RuneKey
}) {
    this.randomRunes = this.randomRunes.map((rune) =>
      rune.name === newRune.name ? { ...rune, ...newRune } :
rune
    );
  }

  refresh(isForce = false) {
    if (!isForce && this.randomRunes.length > 0) {
      this.isCanvasRefreshed = false;
      return;
    }

    this.randomRunes = this.getRandomRunes(this.closedRunes);

    this.isCanvasRefreshed = true;
  }

  getRandomRunes(runes: RuneKeyFull[]) {
```

```
    const runesKit = [...runes];
    const existedPositions: Coord[] = [];

    this.canvas = document.getElementById('runes-canvas') as
HTMLDivElement;

    const canvasSize = this.canvas.getBoundingClientRect();
    const runeSize: number = this.isMobile ? 75 : 55;

    return runes
      .map((r) => {
        const rune = r.toLowerCase() as RuneKey;
        const length = runesKit.length;
        runesKit.pop();
        const newPosition = getPosition(runeSize, {
          width: canvasSize.width,
          height: canvasSize.height,
        });
        existedPositions.push(newPosition);

        return {
          name: rune,
          order: Math.floor(Math.random() * length),
          isInverted: newPosition.rotation > 0.25 &&
newPosition.rotation <= 0.75,
          ...newPosition,
        };
      })
      .sort((a, b) => a.order - b.order);
  }

  mergeRandomRunes(runes: PositionedRune[]) {
    this.randomRunes =
this.randomRunes.reduce<PositionedRune[]>((acc, rune) => {
```

```
      const newValue = runes.find(({ name }) => name ===
rune.name);
      if (newValue) {
        return acc.concat(newValue);
      }

      return acc.concat(rune);
    }, []);
  }

  setWasResultShown(wasShown: boolean) {
    this.wasResultShown = wasShown;
  }

  setActiveModal(modal?: ModalName) {
    this.activeModal = modal;
  }

  reloadAll() {
    return Promise.all([
      this.loadUserInfo(),
      this.loadUserSettings(),
      this.loadUserPredictions(),
    ]);
  }
}

export const mainStore = new MainStore();
```

Файл src\views\DivinationTypeView\index.tsx

```
import { useMainStore } from '../../store/MainStoreContext';
import { DivinationRunesCount } from '../../types';
import { Button, CardGrid, ModalCardBase } from
'@vkontakte/vkui';
import { Icon28Cards2Outline } from '@vkontakte/icons';
```

```
import { useRouter } from '@happysanta/router';
import styles from './style.module.css';
import './style.css';
import { CANVAS_PAGE } from '../../routes';

export const DivinationTypeView = () => {
  const { closeAllRunes, setDivinationType } = useMainStore();
  const router = useRouter();

  const handleClick =
    (type: DivinationRunesCount) => (e:
React.MouseEvent<HTMLDivElement, MouseEvent>) => {
      e.stopPropagation();
      setDivinationType(type);
      router.pushPage(CANVAS_PAGE);
      setTimeout(() => {
        closeAllRunes();
      }, 16);
    };

  return (
    <div className={styles.wrapper}>
      <CardGrid className={styles.cardGrid} size="l">
        <ModalCardBase
          style={{ padding: 0 }}
          className={styles.runeWrapper}
          onClick={handleClick('1')}
          icon={<Icon28Cards2Outline />}
          header="Руна дня"
          actions={
            <Button size="l" onClick={handleClick('1')}>
              Играть
            </Button>
          }
        ></ModalCardBase>
```

```
        <ModalCardBase
          style={{ padding: 0 }}
          className={styles.runeWrapper}
          onClick={handleClick('3')}
          icon={<Icon28Cards2Outline />}
          header="Три руны"
          actions={
            <Button size="l" onClick={handleClick('3')}>
              Играть
            </Button>
          }
        ></ModalCardBase>

        <ModalCardBase
          style={{ padding: 0 }}
          className={styles.runeWrapper}
          onClick={handleClick('4')}
          icon={<Icon28Cards2Outline />}
          header="Четыре руны"
          actions={
            <Button size="l" onClick={handleClick('4')}>
              Играть
            </Button>
          }
        ></ModalCardBase>
      </CardGrid>
    </div>
  );
};
```

Файл src\views\ErrorView\index.tsx

```
import { useRouter } from '@happysanta/router';
import { Banner, Button } from '@vkontakte/vkui';
import { observer } from 'mobx-react-lite';
```

```
import { useCallback, useState } from 'react';
import { SpinnerWithBackdrop } from
'../../components/SpinnerWithBackdrop';
import { PROFILE_PAGE } from '../../routes';
import { useMainStore } from '../../store/MainStoreContext';
import { Icon56ErrorTriangleOutline } from '@vkontakte/icons';

export const ErrorView = observer(() => {
  const router = useRouter();
  const { reloadAll } = useMainStore();
  const [isLoading, setIsLoading] = useState(false);

  const handleReload = useCallback(async () => {
    setIsLoading(true);
    await reloadAll();
    return router.pushPage(PROFILE_PAGE);
  }, [router, reloadAll]);

  return (
    <SpinnerWithBackdrop isLoading={isLoading}>
      <Banner
        before={<Icon56ErrorTriangleOutline />}
        header="Произошла неизвестная ошибка!"
        actions={<Button onClick={handleReload}>Обновить
данные</Button>}
      />
    </SpinnerWithBackdrop>
  );
});
```

Файл src\views\Prediction\index.tsx

```
import { ReactElement } from 'react';
import { observer } from 'mobx-react-lite';
import { Button, ButtonProps, Div, Text, Title } from
'@vkontakte/vkui';
```

```
import { Rune } from '../../components/Rune';
import { divinationsData } from '../../constants';
import { useMainStore } from '../../store/MainStoreContext';
import runesData from '../../assets/data/runes.json';
import { DivinationRunesCount, RuneData, RuneKey, RuneKeyFull }
from '../../types';
import { RunesEnum } from '../../runes';
import styles from './style.module.css';

interface SaveButtonProps {
  mode: ButtonProps['mode'];
  label: string;
  before?: ReactElement;
}

export const Prediction = observer(
  ({ isNewPrediction }: { isNewPrediction: boolean }) => {
    const {
      openedRunes,
      currentPrediction,
      divinationType,
      settings,
      isPredictionSaved,
      setActiveModal,
    } = useMainStore();

    const actualRunes: RuneKeyFull[] = isNewPrediction
      ? openedRunes
      : (currentPrediction?.runes as RuneKeyFull[]);

    const divination = isNewPrediction
      ? divinationsData[divinationType]
      : divinationsData[String(actualRunes.length) as
DivinationRunesCount];
```

```
    const handleSave = () => {
      setActiveModal('prediction-saving');
    };

    const saveButtonProperties: SaveButtonProps =
isPredictionSaved
      ? { label: 'Сохранено', mode: 'secondary' }
      : { label: 'Сохранить', mode: 'primary' };

    return (
      <Div className={styles.wrapper}>
        <div className={styles.title}>
          <Title level="2">{divination.name}</Title>

          {isNewPrediction && (
            <Button
              mode={saveButtonProperties.mode}
              disabled={isPredictionSaved}
              onClick={handleSave}
            >
              {saveButtonProperties.label}
            </Button>
          )}
        </div>

        {currentPrediction?.title && !isNewPrediction && (
          <Title
level="3">{`"${currentPrediction.title}"`}</Title>
        )}

        {isNewPrediction && divination.description && (
          <Text weight="medium">{divination.description}</Text>
        )}

        {divination.stages.map((stage, index) => {
```

```
          const rune = actualRunes[index];
          const runeData = (runesData as RuneData)[
            RunesEnum[rune.toLocaleLowerCase() as RuneKey]
          ];
          const isInverted = rune === rune.toUpperCase() &&
'invertedMeaning' in runeData;
          const keyWords = isInverted ?
runeData.invertedKeyWords : runeData.keyWords;
          const transcription = runeData.transcription
            ? ` (${runeData.transcription})`
            : '';

          return (
            <div className={styles.stageWrapper}
key={stage.name}>
              <div className={styles.header}>
                <Rune
                  size="lg"
                  name={rune}
                  hasHover={false}
                  color={settings.runeColor}
                  isOpen
                  shouldInvertLabel
                />

                <div>
                  <Title level="3" weight="2">{`Руна
${runeData.name}${
                    isInverted ? ' перевернутая' : ''
                  }`}</Title>

                  {transcription && <Text
weight="regular">{transcription}</Text>}

                  {keyWords && (
```

```
                    <div>
                      <Text weight="regular">Ключевые
слова:</Text>
                      <Text weight="regular">
                        {keyWords.map((word) =>
`"${word}"`).join(', ')}
                      </Text>
                    </div>
                  )}
                </div>
              </div>

              <div className={styles.stageInfo}>
                <Title level="3" weight="2">
                  {stage.name}
                </Title>
                <Text
weight="regular">({stage.description})</Text>
              </div>

              <Text className={styles.description}
weight="medium">
                {isInverted ? runeData.invertedMeaning :
runeData.meaning}
              </Text>
            </div>
          );
        })}
      </Div>
    );
  }
);
```

Файл src\views\PredictionsList\components\Pagination\index.tsx

```
import { Pagination as VkPagination, useAdaptivity, ViewWidth }
from '@vkontakte/vkui';
import { useCallback } from 'react';
import { PREDICTIONS_PAGE_SIZE } from '../../../../constants';
import { observer } from 'mobx-react-lite';
import { useMainStore } from
'../../../../store/MainStoreContext';

export const Pagination = observer(() => {
  const { predictionPage, userPredictions, loadUserPredictions,
setPredictionPage } =
    useMainStore();
  const { viewWidth } = useAdaptivity();
  const total = userPredictions?.length || 0;

  const handleChange = useCallback(
    (newPage: number) => {
      setPredictionPage(newPage);
      loadUserPredictions();
    },
    [setPredictionPage, loadUserPredictions]
  );
  const pageCount = Math.ceil(total / PREDICTIONS_PAGE_SIZE);

  return pageCount > 1 ? (
    <VkPagination
      currentPage={predictionPage}
      onChange={handleChange}
      totalPages={pageCount}
      siblingCount={viewWidth === ViewWidth.SMALL_MOBILE ? 0 :
1}
    />
  ) : (
    <div />
  );
```

```
});
```

Файл src\views\PredictionsList\components\PredictionCard\index.tsx

```
import { Icon28DeleteOutline } from '@vkontakte/icons';
import { ContentCard, IconButton, Title } from
'@vkontakte/vkui';
import { observer } from 'mobx-react-lite';
import { useCallback } from 'react';
import { Rune } from '../../../../components/Rune';
import { Bridge } from '../../../../modules/Bridge';
import { useMainStore } from
'../../../../store/MainStoreContext';
import { PredictionWithKey } from '../../../../types';
import { useRouter } from '@happysanta/router';
import styles from './style.module.css';
import { STORED_PREDICION_PAGE } from '../../../../routes';

interface Props {
  prediction: PredictionWithKey;
}

export const PredictionCard = observer(({ prediction }: Props)
=> {
  const { settings, loadUserPredictions, openPrediction } =
useMainStore();
  const router = useRouter();
  const handleDeleting = useCallback(
    async (e: React.MouseEvent<HTMLElement, MouseEvent>) => {
      e.stopPropagation();
      await Bridge.storageDelete(prediction.key);
      loadUserPredictions();
    },
    [loadUserPredictions, prediction.key]
  );
```

```
return (
  <ContentCard
    className={styles.wrapper}
    header={
      <div className={styles.titleWrapper}>
        <div className={styles.runesWrapper}>
          {prediction.runes.map((rune) => (
            <Rune
              color={settings.runeColor}
              name={rune}
              size={prediction.runes.length > 1 ? 'sm' : 'md'}
              shouldInvertLabel
              isOpen
              key={rune}
            />
          ))}
        </div>

        <Title className={styles.title} level="3">
          {prediction.title}
        </Title>
        <IconButton onClick={handleDeleting}>
          <Icon28DeleteOutline />
        </IconButton>
      </div>
    }
    subtitle={new Date(prediction.date).toLocaleString('ru')}
    onClick={() => {
      openPrediction(prediction);
      router.pushPage(STORED_PREDICION_PAGE);
    }}
    hasHover
    hasActive
  >
    Card
```

```
    </ContentCard>
  );
});
```

Файл src\views\PredictionsList\index.tsx

```
import { Icon16Sync } from '@vkontakte/icons';
import { Button, CardGrid, Div, Title } from '@vkontakte/vkui';
import { observer } from 'mobx-react-lite';
import { SpinnerWithBackdrop } from
'../../components/SpinnerWithBackdrop';
import { useMainStore } from '../../store/MainStoreContext';
import { Pagination } from './components/Pagination';
import { PredictionCard } from './components/PredictionCard';
import styles from './style.module.css';

export const PredictionsList = observer(() => {
  const { appLoading, userPredictions, loadUserPredictions } =
useMainStore();

  return (
    <SpinnerWithBackdrop
      size="large"
      isLoading={
        appLoading.isPredictionsLoading ||
appLoading.isPredictionDeletingLoading
      }
    >
      <Div className={styles.wrapper}>
        <Button before={<Icon16Sync />}
onClick={loadUserPredictions}>
          Обновить
        </Button>

        {userPredictions?.length ? (
          <>
```

```
            <CardGrid className={styles.cardGrid} size="l">
              {userPredictions.map((prediction) => (
                <PredictionCard prediction={prediction}
key={prediction.date} />
              ))}
            </CardGrid>

            <Pagination />
          </>
        ) : (
          <Title weight="regular" level="2">
            Нет сохраненных вопросов!
          </Title>
        )}
      </Div>
    </SpinnerWithBackdrop>
  );
});
```

Файл src\views\Profile\index.tsx

```
import { Avatar, Banner, Title } from '@vkontakte/vkui';
import { observer } from 'mobx-react-lite';
import { useEffect } from 'react';
import { useMainStore } from '../../store/MainStoreContext';
import { ProfileNavigation } from './ProfileNavigation';
import styles from './style.module.css';

export const Profile = observer(() => {
  const { user, userPredictions, loadUserInfo,
loadUserPredictions } = useMainStore();

  useEffect(() => {
    if (!user) {
      loadUserInfo();
    }
```

```
  }, [user, loadUserInfo]);

  useEffect(() => {
    if (!userPredictions?.length) {
      loadUserPredictions();
    }
  }, [userPredictions?.length, loadUserPredictions]);

  return (
    <div>
      <Banner
        className={styles.banner}
        header={
          <Title weight="regular" className={styles.title}
level="2">
            {user?.first_name}
          </Title>
        }
        before={
          <div className={styles.avatarWrapper}>
            <Avatar mode="image" src={user?.photo_200}
size={160} />
          </div>
        }
        actions={<ProfileNavigation />}
      />
    </div>
  );
});
```

Файл src\views\Profile\ProfileNavigation\index.tsx

```
import {
  Icon28SettingsOutline,
  Icon28QuestionOutline,
  Icon28Cards2Outline,
```

```
  Icon24ServicesOutline,
  Icon28SmartphoneStarsOutline,
} from '@vkontakte/icons';
import { Cell, Counter, Group, Spinner } from '@vkontakte/vkui';
import { observer } from 'mobx-react-lite';
import { useMainStore } from '../../../store/MainStoreContext';
import { useRouter } from '@happysanta/router';
import {
  CANVAS_PAGE,
  DESCRIPTIONS_PAGE,
  PREDICTIONS_PAGE,
  SETTINGS_PAGE,
  TYPES_CHOOSING_PAGE,
} from '../../../routes';

export const ProfileNavigation = observer(() => {
  const router = useRouter();
  const { appLoading, userPredictions } = useMainStore();

  return (
    <Group>
      <Cell
        onClick={() => router.pushPage(PREDICTIONS_PAGE)}
        before={<Icon28QuestionOutline />}
        after={
          appLoading.isPredictionsLoading ? (
            <Spinner />
          ) : (
            <Counter>{userPredictions?.length}</Counter>
          )
        }
      >
        Мои вопросы
      </Cell>
```

```
      <Cell
        before={<Icon28SmartphoneStarsOutline />}
        onClick={() => router.pushPage(TYPES_CHOOSING_PAGE)}
      >
        Расклады
      </Cell>

      <Cell
        before={<Icon24ServicesOutline />}
        onClick={() => router.pushPage(CANVAS_PAGE)}
      >
        Играть
      </Cell>

      <Cell
        before={<Icon28Cards2Outline />}
        onClick={() => router.pushPage(DESCRIPTIONS_PAGE)}
      >
        Описание рун
      </Cell>

      <Cell
        before={<Icon28SettingsOutline />}
        onClick={() => router.pushPage(SETTINGS_PAGE)}
      >
        Настройки
      </Cell>
    </Group>
  );
});
```

Файл src\views\RuneDetail\index.tsx

```
import { Text, Title, Headline, Div } from '@vkontakte/vkui';
import { observer } from 'mobx-react-lite';
import { useMemo } from 'react';
```

```
import runesData from '../../assets/data/runes.json';
import { RunesEnum } from '../../runes';
import { useMainStore } from '../../store/MainStoreContext';
import { RuneData, RuneKey } from '../../types';
import { Rune } from '../../components/Rune';
import styles from './style.module.css';

export const RuneDetail = observer(() => {
  const { currentRune, settings } = useMainStore();

  const content = useMemo(() => {
    if (!currentRune) {
      return (
        <div>
          <Text weight="medium">Вы еще не выбрали руну!</Text>
        </div>
      );
    }

    const runeKey = currentRune.toLowerCase() as RuneKey;
    const runeData = (runesData as
RuneData)[RunesEnum[runeKey]];
    const isInverted = currentRune === currentRune.toUpperCase()
&& 'invertedMeaning' in runeData;
    const transcription = runeData.transcription ? `
(${runeData.transcription})` : '';

    return (
      <Div>
        <div className={styles.mainContent}>
          <Rune
            color={settings.runeColor}
            size="lg"
            shouldInvertLabel
            isOpen
```

```
            name={currentRune}
            hasHover={false}
          />

          <div>
            <Title level="2" weight="2">{`Руна
${runeData.name}${
              isInverted ? ' перевернутая' : ''
            }`}</Title>

            {transcription && <Headline
weight="regular">{transcription}</Headline>}
          </div>
        </div>
        <Text className={styles.description} weight="medium">
          {runeData.description}
        </Text>
      </Div>
    );
  }, [currentRune, settings]);

  return content;
});
```

Файл src\views\RunesCanvas\components\FloatPosition\index.tsx

```
import React, { useEffect, useRef, useState } from 'react';
import { classNames as cn } from '@vkontakte/vkui';
import { FC } from 'react';
import { dragAndDrop } from '../../../../helpers/dragAndDrop';
import styles from './style.module.css';
import { useMainStore } from
'../../../../store/MainStoreContext';
import { RuneKey } from '../../../../types';
import { dragAndDropMobile } from
'../../../../helpers/dragAndDropMobile';
```

```
interface Props {
  name: RuneKey;
  x: number;
  y: number;
  order: number;
  rotation: number;
  isRefreshed: boolean;
}

export const FloatPosition: FC<Props> = React.memo(
  ({ children, x, y, rotation, isRefreshed, order, name }) => {
    const [isMoving, setIsMoving] = useState(false);
    const { canvas, isMobilePlatform, setRandomRune } =
useMainStore();
    const ref = useRef<HTMLDivElement>(null);

    useEffect(() => {
      setTimeout(() => {
        if (ref.current) {
          ref.current.style.transform = `translate(${x}px,
${y}px) rotate(${rotation}turn)`;
        }
      }, 16);
    }, [x, y, rotation]);

    useEffect(() => {
      const dNdHandler = isMobilePlatform ? dragAndDropMobile :
dragAndDrop;

      dNdHandler({
        element: ref.current!,
        container: canvas,
        setIsMoving,
```

```
        setRandomRune: (x: number, y: number) => setRandomRune({
x, y, name }),
      });
    }, [name, isMobilePlatform, canvas, setRandomRune]);

    return (
      <div
        ref={ref}
        className={cn({ [styles.wrapper]: !isMoving },
styles[`layer${order}`])}
        style={{
          transition: isRefreshed ? 'transform 1s' : 'none',
          position: 'absolute',
          top: 0,
          left: 0,
        }}
      >
        {children}
      </div>
    );
  }
);
```

Файл src\views\RunesCanvas\components\RunesPocket\index.tsx

```
import { Rune } from '../../../../components/Rune';
import { useMainStore } from
'../../../../store/MainStoreContext';
import { observer } from 'mobx-react-lite';
import { RuneKeyFull } from '../../../../types';
import { ReactElement } from 'react';
import styles from './style.module.css';

interface Params {
  count: number;
  openedRunes: RuneKeyFull[];
```

```
  renderRune: (name: RuneKeyFull) => ReactElement;
  slotColor: string;
}

const getSlots = ({ count, openedRunes, renderRune, slotColor }:
Params) => {
  const slots = [];

  for (let i = 0; i < count; i++) {
    slots.push(
      <div style={{ borderColor: slotColor }}
className={styles.slot} key={i}>
        {openedRunes[i] && renderRune(openedRunes[i])}
      </div>
    );
  }

  return slots;
};

export const RunesPocket = observer(() => {
  const { tempSettings, settings, divinationType, openedRunes }
= useMainStore();

  const slots = getSlots({
    count: Number(divinationType),
    openedRunes,
    renderRune: (name: RuneKeyFull) => {
      return (
        <Rune
          name={name}
          color={settings.runeColor}
          isOpen={!settings.shouldHideRuneLabel}
          hasHover={!settings.shouldHideRuneLabel}
          shouldInvertLabel
```

```
          key={name}
        />
      );
    },
    slotColor: tempSettings.isDarkTheme ? '#e1e3e6' : '#2d81e0',
  });

  return (
    <div className={styles.runesPocket}>
      <div className={styles.runesWrapper}>{slots}</div>
    </div>
  );
});
```

Файл src\views\RunesCanvas\index.tsx

```
import { useCallback, useEffect } from 'react';
import { Rune } from '../../components/Rune';
import { Icon16Sync } from '@vkontakte/icons';
import { observer } from 'mobx-react-lite';
import { useMainStore } from '../../store/MainStoreContext';
import { Button } from '@vkontakte/vkui';
import { FloatPosition } from './components/FloatPosition';
import { RunesPocket } from './components/RunesPocket';
import { RuneKeyFull } from '../../types';
import { findRune } from '../../helpers/findRune';
import { useRouter } from '@happysanta/router';
import { NEW_PREDICION_PAGE } from '../../routes';
import styles from './style.module.css';

export const RunesCanvas = observer(() => {
  const router = useRouter();
  const {
    isCanvasRefreshed,
    randomRunes,
    openedRunes,
```

```
    divinationType,
    wasResultShown,
    settings,
    setWasResultShown,
    setActiveModal,
    openRune,
    refresh,
  } = useMainStore();

  const handleWindowResize = useCallback(() => {
    refresh(true);
  }, [refresh]);

  useEffect(() => {
    window.addEventListener('resize', handleWindowResize);
    refresh();
    return () => {
      window.removeEventListener('resize', handleWindowResize);
      setActiveModal();
    };
  }, [refresh, handleWindowResize, setActiveModal]);

  const runesComponents = randomRunes.map(
    ({ name, isInverted, x, y, rotation, order }) => {
      const isOpen = findRune(name, openedRunes);

      if (isOpen) {
        return null;
      }

      const runeName = isInverted ? (name.toUpperCase() as
RuneKeyFull) : name;

      return (
        <FloatPosition
```

```
            x={x}
            y={y}
            rotation={rotation}
            isRefreshed={isCanvasRefreshed}
            order={order}
            name={name}
            key={name}
          >
            <Rune
              color={settings.runeColor}
              isDisabled={Number(divinationType) ===
openedRunes.length}
              onClick={() => openRune(runeName)}
              name={runeName}
            />
          </FloatPosition>
        );
      }
    );

    useEffect(() => {
      if (!(openedRunes.length === Number(divinationType))) {
        return setActiveModal();
      }

      if (settings.shouldShowResultImmidiatly && !wasResultShown)
{
        router.pushPage(NEW_PREDICION_PAGE);
        return () => setWasResultShown(true);
      }

      setActiveModal('prediction-result');
    }, [
      router,
      openedRunes.length,
```

```
    divinationType,
    settings,
    wasResultShown,
    setWasResultShown,
    setActiveModal,
  ]);

  return (
    <div className={styles.viewWrapper}>
      <div className={styles.toolbar}>
        <RunesPocket />

        <Button
          before={<Icon16Sync />}
          className={styles.refreshButton}
          onClick={() => refresh(true)}
          size="s"
        >
          Смешать
        </Button>
      </div>

      <div
        style={{
          backgroundColor: settings.canvasColor.body,
          borderColor: settings.canvasColor.border,
        }}
        id="runes-canvas"
        className={styles.wrapper}
      >
        {runesComponents}
      </div>
    </div>
  );
});
```

Файл src\views\RunesList\index.tsx

```
import { Div, Headline } from '@vkontakte/vkui';
import { Rune } from '../../components/Rune';
import { runes, RunesEnum } from '../../runes';
import runesData from '../../assets/data/runes.json';
import { useMainStore } from '../../store/MainStoreContext';
import { observer } from 'mobx-react-lite';
import { useRouter } from '@happysanta/router';
import styles from './style.module.css';
import { RUNE_DETAIL_PAGE } from '../../routes';

export const RunesList = observer(() => {
  const router = useRouter();
  const { settings, setCurrectRune } = useMainStore();

  return (
    <Div className={styles.wrapper}>
      {runes.map((r) => (
        <div className={styles.runeCard} key={r}>
          <Rune
            color={settings.runeColor}
            size="xl"
            onClick={() => {
              setCurrectRune(r);
              router.pushPage(RUNE_DETAIL_PAGE);
            }}
            name={r}
            isOpen
          />
          <Headline
weight="medium">{runesData[RunesEnum[r]].name}</Headline>
        </div>
      ))}
    </Div>
  );
```

```
});
```

Файл src\views\Settings\components\ColorPicker\index.tsx

```
import { HexColorPicker, HslColorPicker, HslColor } from 'react-
colorful';
import { Dropdown } from '@vkontakte/vkui/dist/unstable';
import { ReactNode, useState } from 'react';
import { classNames as cn, Text } from '@vkontakte/vkui';
import { getHslCssColor } from
'../../../../helpers/getHslCssColor';
import { observer } from 'mobx-react-lite';
import styles from './style.module.css';

interface CommonProps {
  label?: ReactNode;
}

type hexChangeHandler = (newColor: string) => void;
type hslChangeHandler = (newColor: HslColor) => void;

interface HexColorProps {
  hslColor?: never;
  color?: string;
  onChange: hexChangeHandler;
}
interface HslColorProps {
  hslColor?: HslColor;
  color?: never;
  onChange: hslChangeHandler;
}

type Props = CommonProps & (HexColorProps | HslColorProps);

export const ColorPicker = observer((props: Props) => {
  const [isDropdownOpen, setIsDropdownOpen] = useState(false);
```

```
  const color = 'color' in props ? props.color : undefined;
  const hslColor = 'hslColor' in props ? props.hslColor :
undefined;
  const { label, onChange } = props;

  return (
    <Dropdown
      action="click"
      shown={isDropdownOpen}
      onShownChange={setIsDropdownOpen}
      placement="auto"
      content={
        color ? (
          <HexColorPicker
            className={styles.picker}
            color={color}
            onChange={onChange as hexChangeHandler}
          />
        ) : (
          <HslColorPicker
            className={styles.picker}
            color={hslColor}
            onChange={onChange as hslChangeHandler}
          />
        )
      }
    >
      <div className={styles.wrapper}>
        {label && (
          <Text className={styles.label} weight="regular">
            {label}
          </Text>
        )}
        <div
```

```
          className={cn(styles.coloredCircle, {
[styles.hoveredColoredCircle]: isDropdownOpen })}
          style={{ backgroundColor: color ? color :
getHslCssColor({ hslColor }) }}
        />
      </div>
    </Dropdown>
  );
});
```

Файл src\views\Settings\components\SettingToggle\index.tsx

```
import { SimpleCell, Switch } from '@vkontakte/vkui';
import { observer } from 'mobx-react-lite';
import { useMainStore } from
'../../../../store/MainStoreContext';
import { BooleanSettingField } from '../../../../types';

interface Props {
  label: string;
  fieldName: BooleanSettingField;
}

export const SettingToggle = observer(({ label, fieldName }:
Props) => {
  const { tempSettings, setTempSettings } = useMainStore();

  return (
    <SimpleCell
      Component="label"
      after={
        <Switch
          onChange={(event) =>
setTempSettings(event.target.checked, fieldName)}
          checked={tempSettings[fieldName]}
        />
```

```
      }
    >
      {label}
    </SimpleCell>
  );
});
```

Файл src\views\Settings\components\ThemePicker\index.tsx

```
import { Icon28MoonOutline, Icon28SunOutline } from
'@vkontakte/icons';
import { IconButton } from '@vkontakte/vkui';
import { observer } from 'mobx-react-lite';
import { useMainStore } from
'../../../../store/MainStoreContext';

export const ThemePicker = observer(() => {
  const { tempSettings, setTempSettings } = useMainStore();

  const isDark = tempSettings.isDarkTheme;

  return (
    <IconButton
      onClick={(e) => {
        e.preventDefault();
        setTempSettings(!isDark, 'isDarkTheme');
      }}
    >
      {isDark ? <Icon28SunOutline /> : <Icon28MoonOutline />}
    </IconButton>
  );
});
```

Файл src\views\Settings\index.tsx

```
import { Button, ButtonGroup, Div, SimpleCell, useAdaptivity,
ViewWidth } from '@vkontakte/vkui';
```

```
import { observer } from 'mobx-react-lite';
import { RuneObserved } from '../../components/Rune';
import { useMainStore } from '../../store/MainStoreContext';
import { ColorPicker } from './components/ColorPicker';
import { ThemePicker } from './components/ThemePicker';
import { SettingToggle } from './components/SettingToggle';
import styles from './style.module.css';

export const Settings = observer(() => {
  const {
    tempSettings,
    wereSettingsChanged,
    wereSettingsSaved,
    isDefaultSettings,
    setTempSettings,
    saveSettings,
    resetTempSettings,
    resetAllSettings,
  } = useMainStore();

  const { viewWidth } = useAdaptivity();
  const isMobile = viewWidth < ViewWidth.SMALL_TABLET;

  return (
    <Div>
      <div className={styles.previewWrapper}>
        <div
          style={{ borderColor: tempSettings.isDarkTheme ?
'#e1e3e6' : '#2d81e0' }}
          className={styles.slot}
        >
          <RuneObserved name="f" isOpen
color={tempSettings.runeColor} />
        </div>
```

```
        <div
          style={{
            backgroundColor: tempSettings.canvasColor.body,
            borderColor: tempSettings.canvasColor.border,
          }}
          className={styles.canvas}
        >
          <RuneObserved name="u" isOpen
color={tempSettings.runeColor} />

          <RuneObserved name="n" isOpen
color={tempSettings.runeColor} />

          <RuneObserved name="b" isOpen
color={tempSettings.runeColor} />
        </div>
      </div>

      <div className={styles.runeColorPickers}>
        <ColorPicker
          label="Плашка"
          hslColor={tempSettings.runeColor.body}
          onChange={(hslColor) => setTempSettings(hslColor,
'rune.body')}
        />

        <ColorPicker
          label="Обводка руны"
          hslColor={tempSettings.runeColor.border}
          onChange={(hslColor) => setTempSettings(hslColor,
'rune.border')}
        />

        <ColorPicker
          label="Руна"
```

```
          hslColor={tempSettings.runeColor.label}
          onChange={(hslColor) => setTempSettings(hslColor,
'rune.label')}
        />

        <ColorPicker
          label="Канвас"
          color={tempSettings.canvasColor.body}
          onChange={(color) => setTempSettings(color,
'canvas.body')}
        />

        <ColorPicker
          label="Обводка канваса"
          color={tempSettings.canvasColor.border}
          onChange={(color) => setTempSettings(color,
'canvas.border')}
        />
      </div>

      <div className={styles.switchControllers}>
        <SimpleCell Component="label" after={<ThemePicker />}>
          Тема
        </SimpleCell>

        <SettingToggle label="Скрывать руну"
fieldName="shouldHideRuneLabel" />

        <SettingToggle label="Результат сразу"
fieldName="shouldShowResultImmidiatly" />

        {/* {platform === 'android' ||
          (platform === 'ios' && (
            <SimpleCell
              Component="label"
```

```
              after={<Switch
checked={tempSettings.shouldShowResultImmidiatly} />}
            >
              Трясти для перемешки
            </SimpleCell>
          ))} */}
      </div>

      <div className={styles.actionButtons}>
        {isMobile ? (
          <ButtonGroup mode="vertical" stretched>
            <Button disabled={wereSettingsSaved} stretched
onClick={saveSettings}>
              Сохранить
            </Button>

            <ButtonGroup stretched>
              <Button
                disabled={isDefaultSettings}
                mode="tertiary"
                stretched
                onClick={resetAllSettings}
              >
                Сбросить
              </Button>
              <Button
                disabled={!wereSettingsChanged}
                mode="secondary"
                stretched
                onClick={resetTempSettings}
              >
                Отмена
              </Button>
            </ButtonGroup>
          </ButtonGroup>
```

```
          ) : (
            <ButtonGroup>
              <Button
                disabled={isDefaultSettings}
                mode="tertiary"
                stretched
                onClick={resetAllSettings}
              >
                Сбросить
              </Button>
              <Button
                disabled={!wereSettingsChanged}
                mode="secondary"
                stretched
                onClick={resetTempSettings}
              >
                Отмена
              </Button>
              <Button disabled={wereSettingsSaved} stretched
onClick={saveSettings}>
                Сохранить
              </Button>
            </ButtonGroup>
          )}
        </div>
      </Div>
    );
});
```

Приложение В

(обязательное)

Кроссплатформенное приложение на базе технологии

ВКонтакте *Mini Apps*

Презентация

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ДИПЛОМНЫЙ ПРОЕКТ НА ТЕМУ

КРОССПЛАТФОРМЕННОЕ ПРИЛОЖЕНИЕ ДЛЯ ПЛАТФОРМЫ ВКОНТАКТЕ MINI APPS

- Студент – Лисов А.А.
- Руководитель – Бычков А.С.

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

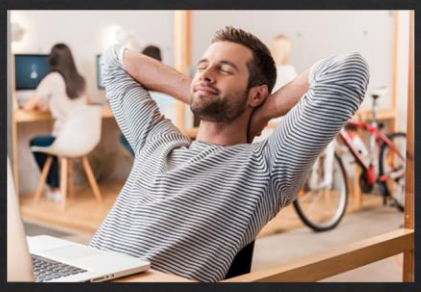
Эмоциональное выгорание

- Эмоциональное выгорание – это процесс психологического истощения нервной системы человека

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Разработчики выполняют тяжелую работу

- На первый взгляд, кажется, что работа оператора автоматизированных систем не требует большого напряжения



- Однако, на самом деле каждый разработчик способен рассказать о собственном опыте знакомства с синдромом эмоционального выгорания

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Как бороться с проблемой?

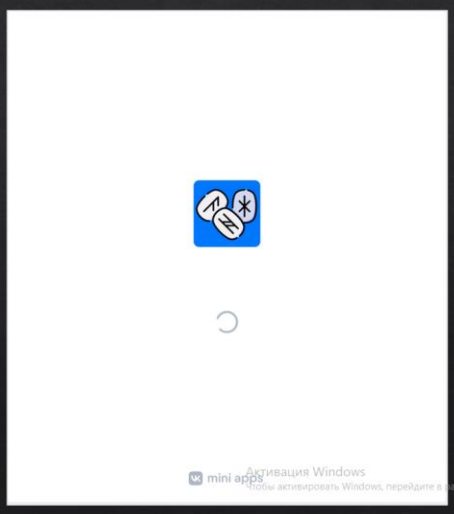
- Чтобы справиться с эмоциональным выгоранием достаточно делать в работе кратковременные перерывы

перейдите в раздел "Параметры".

Приложение для перерыва

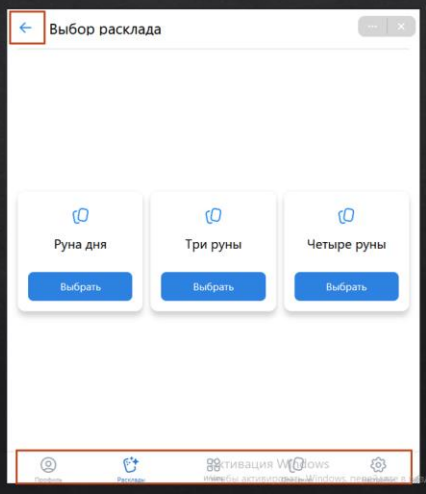
- Для организации небольшого перерыва в работе оператора было разработано приложение для платформы ВКонтакте Mini Apps
- Уже доступно по ссылке:

  https://vk.com/app8157146_149100237



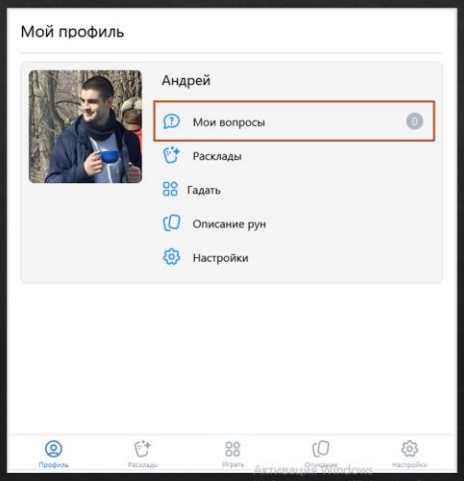
Панели приложения

- Приложение состоит из нескольких панелей.
- Перемещение между ними организуется с помощью навигации в нижней части экрана
- И с помощью стрелок «Назад» в верхнем левом углу панелей



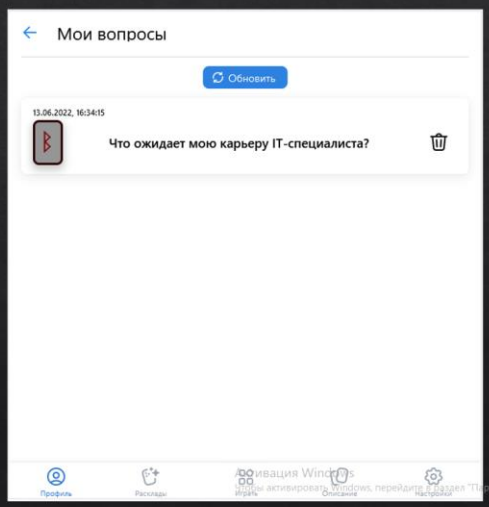
Панель «Мой профиль»

- Основная панель приложения.
- Содержит второй блок навигации, дублирующий основную с дополненным пунктом «Мои вопросы»



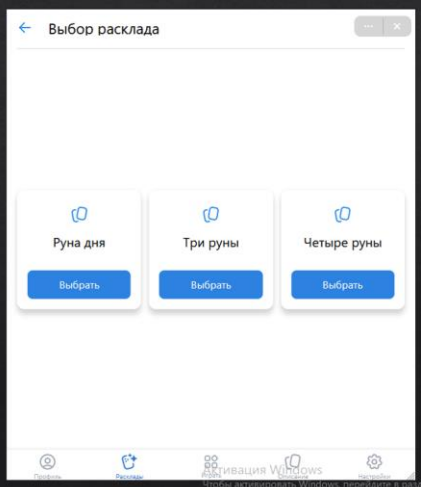
Панель «Мои вопросы»

- Панель «Мои вопросы» нужна для отображения списка сохраненных вопросов
- Позволяет перейти к полученному ответу и удалить сохраненный вопрос



Панель «Выбор расклада»

- Панель определяет поведение основной панели
- Позволяет задать количество выбираемых элементов
- При нажатии на карточку расклада выполняется переход на основную панель

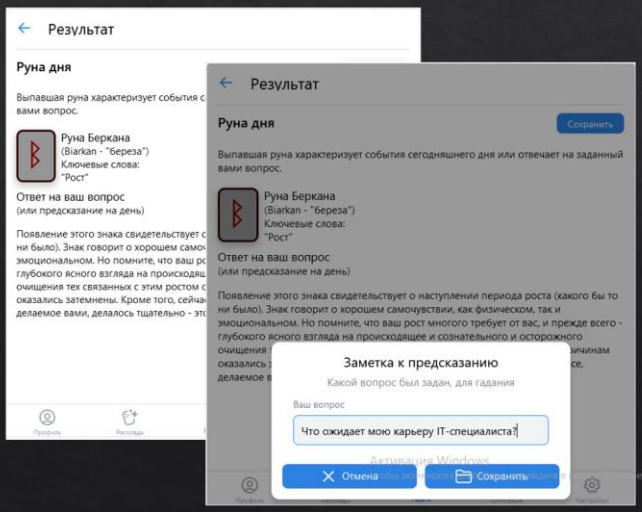


Основная панель

- Позволяет выбрать элемент из случайным образом перемешенного набора
- После выбора элемента осуществляется переход на панель результат
- Набор можно заново перемешать нажатием на кнопку «Смешать»
- Каждый элемент можно перемещать в пределах полотна

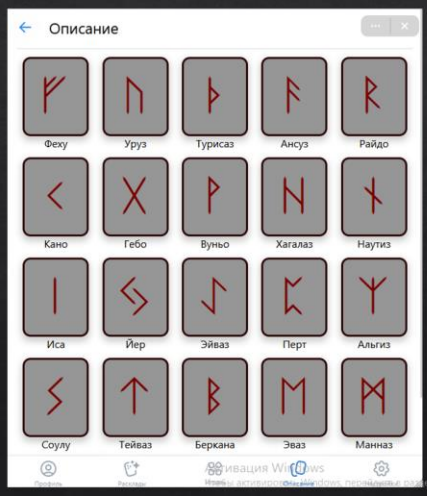
Панель «Результат»

- Содержит информацию о значении выбранного на основной панели элемента
- Позволяет пользователю сохранить полученный ответ с собственной заметкой
- Зависит от заданного расклада



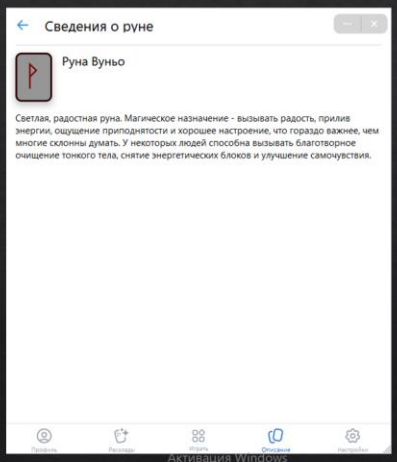
Панель «Описание»

- Информационная панель
- Содержит список используемых в приложении элементов
- Клик по элементу открывает панель сведений об элементе



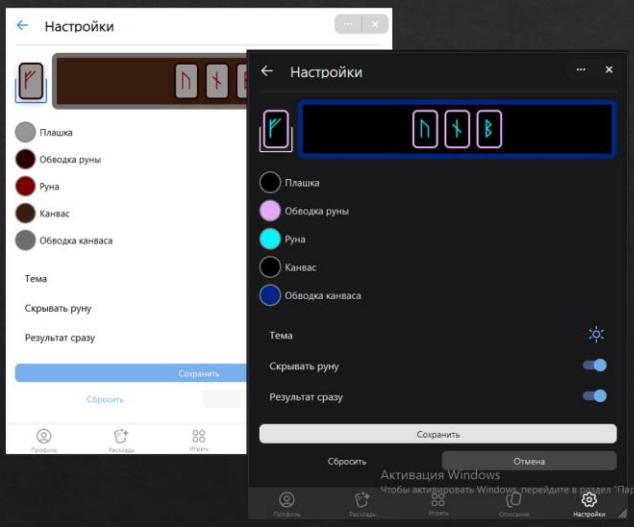
Панель сведений об элементе

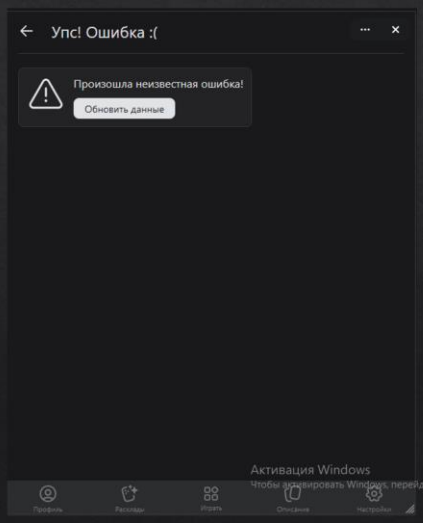
- Здесь содержится общая информация о выбранном на панели «Описание» элементе



Панель «Настройки»

- Интерактивная настройка элементов приложения
- Определение логики переходов между основной панелью и панелью «Результат»
- Выбор темы приложения



А если что-то пойдет не так?

- Панель отображения ошибки не позволит приложению упасть
- У пользователя остается возможность взаимодействия с приложением даже после ошибки
- Кнопка «Обновить данные» повторно загружает все данные пользователя