*Федеральное государственное бюджетное образовательное учреждение высшего образования*

*«Владимирский государственный университет*

*имени Александра Григорьевича и Николая Григорьевича Столетовых»*

*Кафедра информационных систем и программной инженерии*

***ПОЯСНИТЕЛЬНАЯ ЗАПИСКА***

***к курсовому проекту по дисциплине   
"Технологии разработки мобильных приложений"***

***на тему***

*Разработка мобильной игры кликера с возможностью покупки улучшений*

*Выполнил: Грачев Д. А.*

*Принял: ассистент кафедры*

*ИСПИ Петрова А.И.*

*Владимир, 2023*

**Аннотация**

В данном курсовом проекте производилось проектирование и разработка мобильной игры кликера с возможностью покупки улучшений. Проект состоит из 3 этапов, включающих в себя описание предметной области, моделирование структуры объектов предметной области и их взаимодействия на концептуальном уровне, реализацию программной системы, содержащую серверную часть системы и интерфейс пользователя.

Разработка мобильного приложения для курсового проекта велась на языке программирования JavaScript с использованием библиотеки React-Native. Для реализации авторизации и хранения данных использовалась платформа Firebase, в которой были подключены компоненты «Authentication» и «Cloud Firestore».

This course project involved the design and development of a mobile clicker game with the ability to purchase improvements. The project consists of 3 stages, including a description of the subject area, modeling the structure of objects in the subject area and their interaction at the conceptual level, implementation of a software system containing the server part of the system and the user interface.

The development of a mobile application for the course project was carried out in the JavaScript programming language using the React-Native library. To implement authorization and data storage, the Firebase platform was used, in which the “Authentication” and “Cloud Firestore” components were connected.

Курсовой проект представлен на 27 страницах, рисунков – 16, таблиц – 2, использованных источников – 6, приложений – 1.

**Содержание**

[ВВЕДЕНИЕ 4](#_Toc154639158)

[1 Описание предметной области 5](#_Toc154639159)

[1.1. Описание предметной области 5](#_Toc154639160)

[1.2. Требования в системе 5](#_Toc154639161)

[1.3. Стек технологий 5](#_Toc154639162)

[1.4. Обзор аналогов 6](#_Toc154639163)

[1.5. Словарь предметной области 7](#_Toc154639164)

[1.6. Вывод 7](#_Toc154639165)

[2. проектирование системы 8](#_Toc154639166)

[2.1. Диаграмма прецендентов 8](#_Toc154639167)

[2.2. Спецификации прецендентов 8](#_Toc154639168)

[2.3. Диаграмма классов 9](#_Toc154639169)

[2.4. Диаграмма развертывания 10](#_Toc154639170)

[2.5. Схема навигации приложения 11](#_Toc154639171)

[2.6. Общие принципы организации системы 11](#_Toc154639172)

[2.7. Макет приложения 12](#_Toc154639173)

[3 Разработка системы 15](#_Toc154639174)

[3.1. Хранение данных 15](#_Toc154639175)

[3.2. Мобильное приложение 16](#_Toc154639176)

[заключение 23](#_Toc154639177)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 24](#_Toc154639178)

[ПРИЛОЖЕНИЕ А 25](#_Toc154639179)

# ВВЕДЕНИЕ

В ходе выполнения курсового проекта требуется спроектировать и разработать мобильную игру кликер с возможностью покупки улучшений.

Цель работы: изучение процесса разработки мобильных приложений на примере мобильной игры кликера с возможностью покупки улучшений.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать хранилище данных, в котором будет находиться информация о различных сущностях системы
2. Разработать интерфейс, с которым будет взаимодействовать пользователь

# Описание предметной области

* 1. Описание предметной области

В сфере развлечений для программистов необходимо создать мобильную игру-кликер, предназначенную для развлечения и отдыха. Данная игра обеспечивает:

* + Возможность кратковременного ежедневного входа в игру без надобности постоянного контроля за происходящим в ней
  + Возможность покупать себе помощников
  + Авторизацию через Google
  + Сохранение данных между устройствами и между перезапусками
  1. Требования в системе

Функциональные требования:

* Войти в аккаунт
* Выйти из аккаунта
* Сохранить прогресс
* Загрузить прогресс
* Покупка помощников

Нефункциональные требования:

* Все возможности приложения должны быть доступны максимум в два нажатия
* Звуковое сопровождение игровых действий
* Использование Firebase в качестве хранилища данных
  1. Стек технологий
     1. JavaScript (ECMAScript2023)
     2. React-Native
     3. React
     4. MobX
     5. Metro
     6. Firebase
  2. Обзор аналогов

В качестве аналогов разрабатываемой игры были выбраны одни из самых популярных представителей жанра кликеры.

«Cookie Clicker» – это классический кликер, где игроки начинают с клика по печенькам для их накопления. Постепенно можно улучшать производство и развивать свой "пекарский бизнес", накапливая огромное количество печенек. «Cookie Clicker» является образцом жанра и привлекает своей простотой, позволяя игрокам просто кликать и собирать печеньки. Разработчики регулярно добавляют новые элементы геймплея и улучшения, поддерживая интерес игроков. Однако ограниченная механика и монотонность геймплея могут привести к быстрому насыщению.

«Clicker Heroes» - «Clicker Heroes» предлагает динамичный геймплей, где игроки сражаются с монстрами, кликая по ним. Со временем происходит автоматизация кликов и развитие героев, придающее стратегический элемент в борьбе за преодоление всё более сложных уровней. Игра предлагает возможность автоматизации кликов и развития персонажей, что придает динамичность и глубину, и возможность улучшения навыков и наносимого урона создает стратегический элемент. Новички могут столкнуться с изначальной сложностью в понимании всех механик.

«AdVenture Capitalist» – это смесь кликера и тайкун-симулятора, где игроки начинают с небольшого бизнеса и развивают его, инвестируя в различные отрасли. Цель – стать миллиардером и доминировать в мировом бизнесе. Игра включает в себя элементы тайкунов и экономической стратегии, где игроки управляют своим бизнесом и инвестициями. Легкий старт и возможность автоматизации процессов делают игру привлекательной для широкой аудитории. Некоторые игроки могут почувствовать ограничение в разнообразии механик и стратегий.

Для разработки игры обратим внимание на «Clicker Heroes» за его динамичность и систему прокачки персонажей. Также можно взять во внимание «AdVenture Capitalist» для интеграции экономических элементов и «Cookie Clicker» как образец простой и забавной механики.

* 1. Словарь предметной области
     1. Псевдокод– случайный набор символов, отображаемый на экране во время игры
     2. Помощники – элементы игры, которые можно приобрести за игровую валюту для того, чтобы ускорить свой прогресс. Они будут приносить прибыль пассивно, без вашего участия.
     3. Клик – нажатие на экран, элемент геймплея
  2. Вывод

Определена предметная область и цель разработки. Выявлены функциональные и нефункциональные требования, а также составлен стек технологий, на основе которых будет разрабатываться система

# проектирование системы

* 1. Диаграмма прецендентов

На данной диаграмме отображены все преценденты и ее актеры. Каждый прецендент имеет своего инициатора в виде актера.

На представленном ниже рисунке показана диаграмма прецендентов (рис. 1).

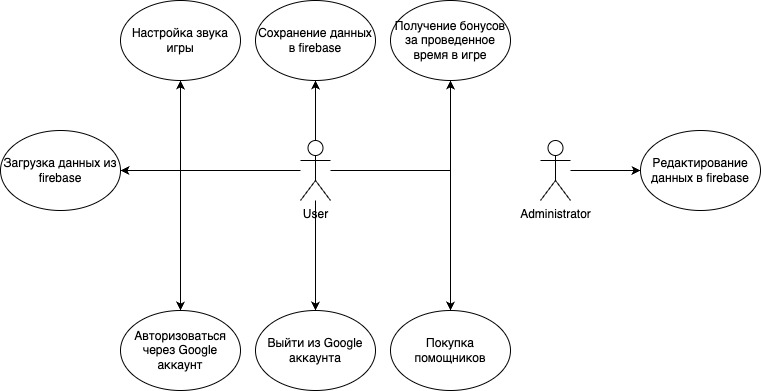


Рисунок 1. Диаграмма прецендентов

* 1. Спецификации прецендентов

Таблица 1. Спецификация прецендента "Авторизоваться через Google аккаунт"

|  |  |
| --- | --- |
| Раздела спецификации прецендента | Описание |
| Название | Авторизоваться через Google аккаунт |
| Актер | User |
| Предусловия | Пользователь открывает приложение и в нем открывает меню настроек; пользователь заранее не войден ни в один из аккаунтов |
| Основной поток | Пользователь нажимает кнопку входа через Google, заполняет данные для входа в свой аккаунт |
| Альтернативный поток | В случае, если пользователь уже войден в аккаунт у него будет отсутствовать кнопка входа и присутствовать кнопка выхода |
| Постусловие | После входа в аккаунт у пользователя будет доступна кнопка выхода из аккаунта и функционал сохранения и подгрузки данных из Firebase |

Таблица 2. Спецификация прецендента "Загрузка данных из Firebase"

|  |  |
| --- | --- |
| Раздел спецификации прецендента | Описание |
| Название | Загрузка данных из Firebase |
| Актер | User |
| Предусловие | Пользователь вошел в свой аккаунт, перешел к настройкам, нажал кнопку получения данных |
| Основной поток | Данные получаются из Firebase и сохраняются в состояние приложения |
| Альтернативный поток | Данные не были получены из Firebase, состояние приложения не изменяется |
| Постусловие | Пользователь продолжает игру с сохранением своего старого прогресса |

* 1. Диаграмма классов

Диаграммы классов могут быть двух уровней. Первый – концептуальный; используется для изображения модели данных будущей системы. Второй – уровень реализации приближен к структуре классов готовой системы. На нем изображаются классы системы, которые тем или иным образом должны реализовывать функциональность системы, прецеденты, модель данных и работу с данными.

На представленном ниже рисунке показана диаграмма классов уровня реализации(рис. 2).

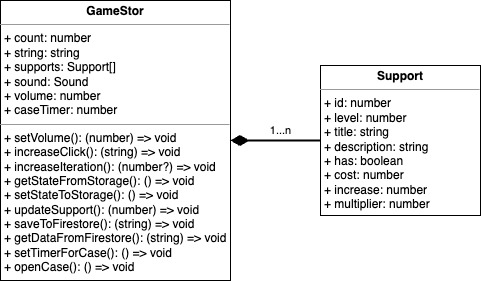


Рисунок 2. Диаграмма классов

Сущности программной системы были спроектированы учитываю опыт разработки приложения и с учетом необходимых данных:

1. count – состояние приложения, хранящее в себе количество нажатий(килков)
2. string – состояние приложения, хранящее в себе отображаемую строку
3. supports – состояние приложения, хранящее в себе информацию о помощниках. Класс Support хранит информацию о различных ценах помощника, его уровень, информацию для отображения и информацию о его улучшениях
4. sound – состояние приложения, хранящее в себе информацию о воспроизводимом звуке
5. volume – состояние приложения, хранящее в себе громкость звука
6. caseTimer – состояние приложения, хранящее в себе оставшееся количество секунд до возможности открыть сундук
   1. Диаграмма развертывания

Диаграмма развертывания – это тип UML-диаграммы, которая показывает архитектуру исполнения системы, включая такие узлы, как аппаратные или программные среды исполнения, а также промежуточное программное обеспечение, соединяющее их.

Приложение разворачивается на мобильном устройстве пользователя и взаимодействует с базой данных, которая находится на серверах Google.

Диаграмма развертывания представлена на рисунке 3.

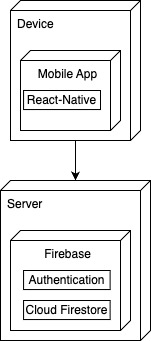


Рисунок 3. Диаграмма развертывания

* 1. Схема навигации приложения

На схеме навигации описываются возможные пути перехода между страницами в приложении.

В данном случае схема навигации получается довольно простая в виду того, что в приложении содержится малое количество страниц, так как весь функционал сосредоточен на кликах на основном окне игры. (рис. 4)

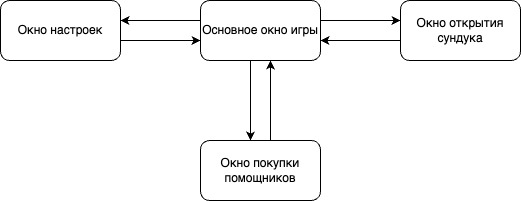


Рисунок 4. Схема навигации

* 1. Общие принципы организации системы

Разрабатываемая система представляет собой мобильное приложение, построенное на базе библиотеки React-Native, на языке JavaScript. Данная библиотека предоставляет возможность написания кроссплатформенного приложения. Такое приложение можно написать один раз, а после запускать на разных платформах.

Приложение разделено на несколько составных частей: часть представления – это компоненты интерфейса, в которых происходит отображение данных на экран, а так же считывание действий пользователя, часть бизнес-логики – это компоненты хранилищ данных, в которых находятся все данные связанные с приложением и необходимые для их работы, а так же функции для работы с этими данными.

* 1. Макет приложения

Для разработки было создано несколько макетов страниц приложения, на которых были отражены различные состояния приложения и все элементы интерфейса.



Рисунок 5. Основное окно игры

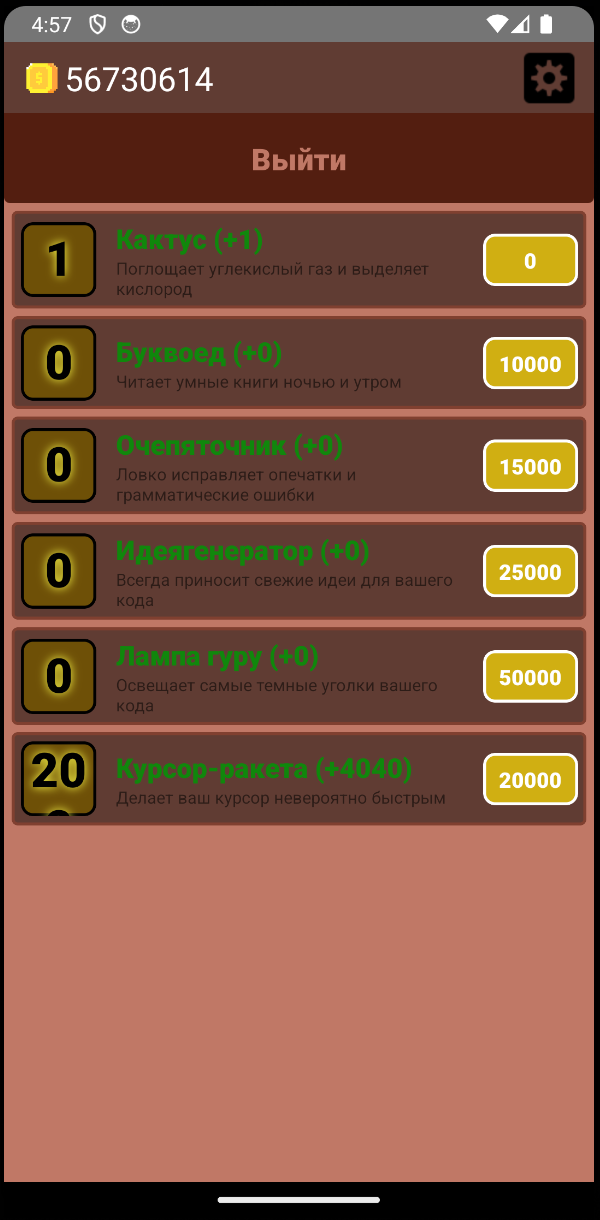


Рисунок 6. Окно покупки помощников



Рисунок 7. Окно открытия сундука

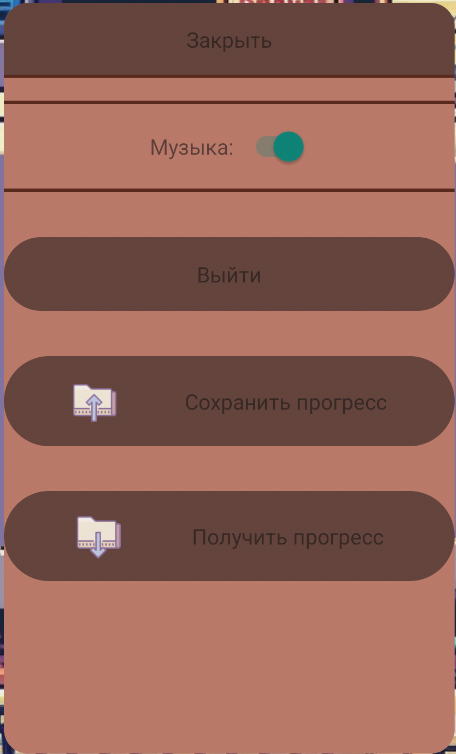


Рисунок 8. Всплывающее окно настроек

# Разработка системы

* 1. Хранение данных

В качестве системы хранения данных была выбрана документо-ориентированная база данных Firebase – Cloud Firestore. Данный выбор был сделан на основе сочетания качеств простоты использования и скорости подключения данный системы. Также данный сервис предоставляет готовое решение для реализации авторизации пользователей посредством Google аккаунта.

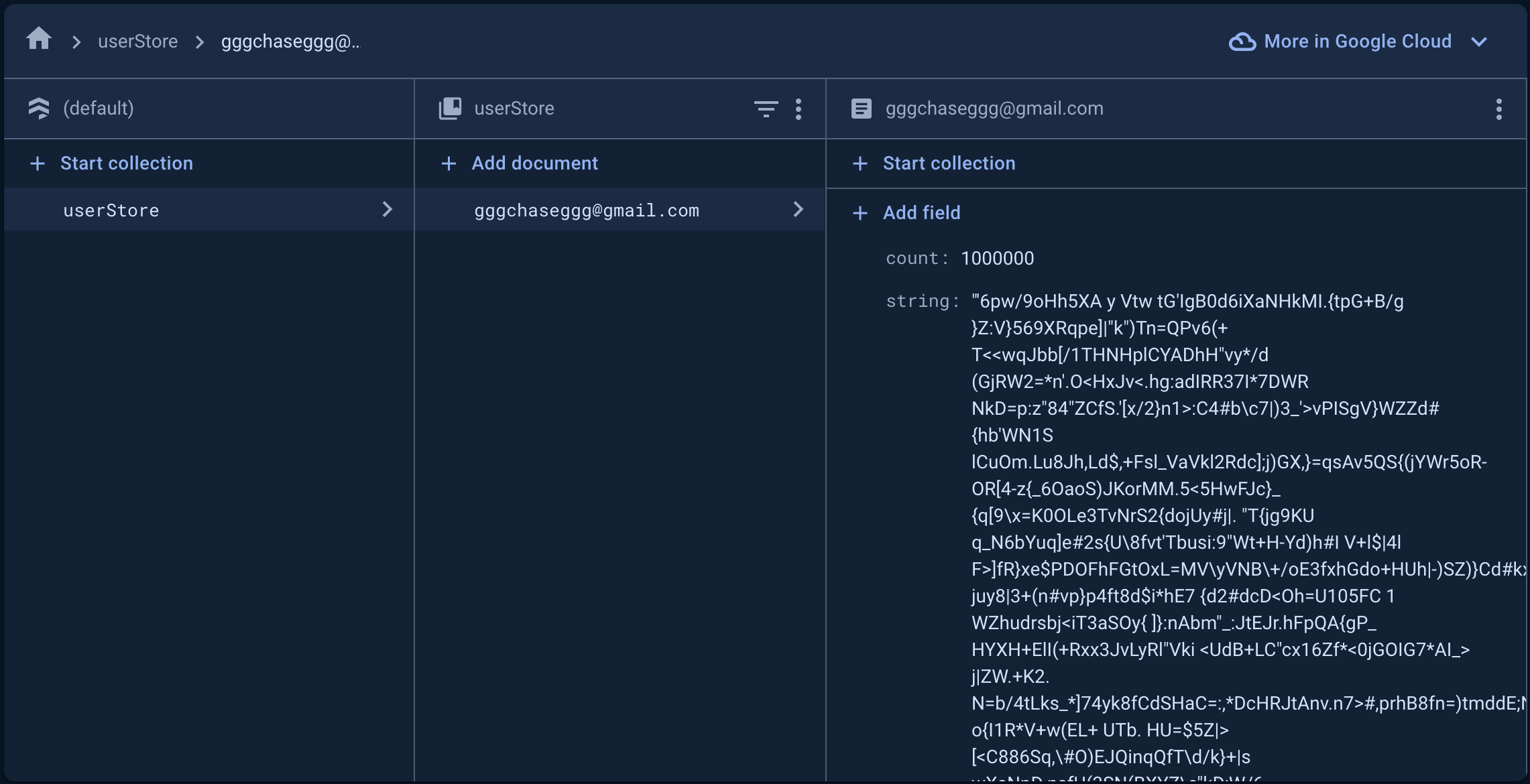


Рисунок 9. Схема базы данных

База данных состоит из коллекций и документов внутри них. В данном случае была создана коллекция «userStore», которая хранит в себе документы, ключами для которых являются почтовые адреса пользователей. Значениями же данных документов являются состояния приложений пользователя, а именно количество денег, строка напечатанных символов и информация о помощниках.

Также в Firebase был использован сервис «Authentication», который позволяет быстро производить авторизацию пользователей. Для подключения Fairbase к приложению необходимо указать токен подключения, это делается достаточно просто и указано в инструкциях для разработчиков.

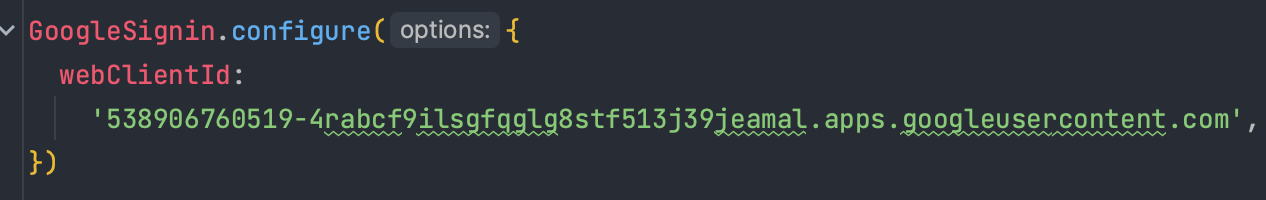


Рисунок 10. Подключение Firebase

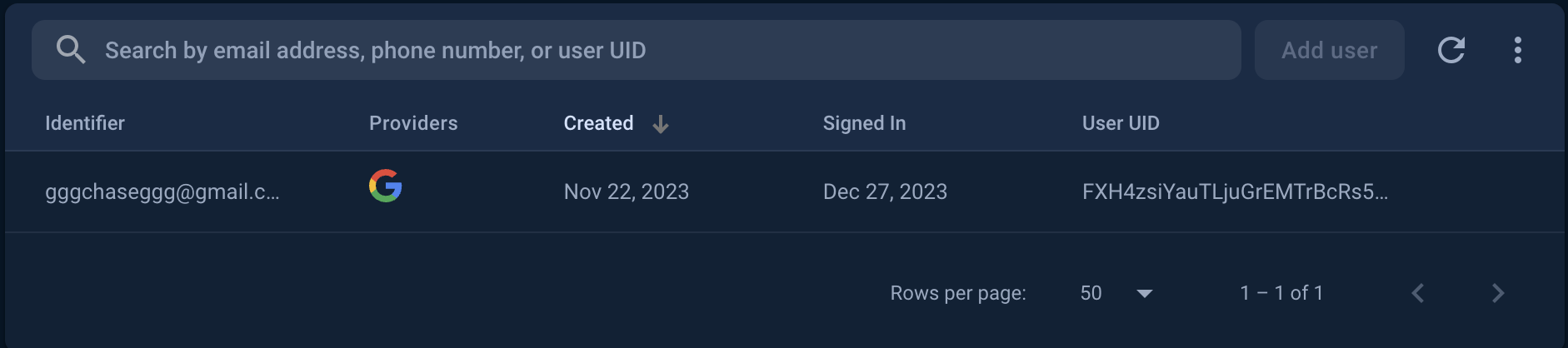


Рисунок 11. Таблица авторизированных аккаунтов

* 1. Мобильное приложение

Для реализации пользовательского интерфейса была выбрана библиотека React-Native – javascript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов, которая разрабатывается и поддерживается запрещенной в России организацией.

* + 1. Главное окно приложения

Когда пользователь откроет приложение, он увидит его главное окно(рис. 11). На нем есть три блока.

Верхний блок(а) отображает количество заработанных денег слева и кнопку открытия настроек. Для реактивного отображения количества денег была использована библиотека Mobx. Данная библиотека позволяет отслеживать состояния свойств приложения и обновлять необходимые компоненты в тот момент, когда изменяются свойства, которые они используют. В данном случае, при каждом обновлении количества денег, хедер будет перерисовываться и отображать актуальное количество.

Нижний блок(б) служит навигационной панелью. И хоть он не является навигационной панелью в действительности, он выполняет функцию отображения различных модальных окон, в данном случае это окно для покупки и улучшения помощников и окно для открытия сундука.

Третий блок – это средний блок, основной использующийся для игры, сюда пользователь должен нажимать, чтобы зарабатывать деньги и здесь же отображается псевдокод, состоящий из случайных символов.



Рисунок 12. Главное окно приложения

* + 1. Меню настроек

Меню настроек представляет собой модальное окно, которое появляется при нажатии на шестеренку в правом верхнем углу. Если пользователь не авторизирован (рис. 14), ему будет предложено пройти авторизацию через Google аккаунт. Если пользователь авторизован (рис. 13), то ему доступны функции сохранения и загрузки прогресса с сервера. За это отвечают кнопки «Сохранить прогресс» и «Получить прогресс» соответственно.

Листинг функций отправки и получения прогресса:

public saveToFirestore = (email: string) => {  
 firestore().collection('userStore').doc(email).set({  
 count: this.count,  
 string: this.string,  
 supports: this.supports,  
 })  
}  
  
public getDataFromFirestore = async (email: string) => {  
 const data = (  
 await firestore().collection<GameState>('userStore').doc(email).get()  
 ).data()  
  
 runInAction(() => {  
 if (data) {  
 this.count = data.count  
 this.string = data.string  
 this.supports = data.supports  
 }  
 })  
}

В первой функции в состояние приложения сохраняется в Firebase в коллекцию с названием «userStore» по ключу равным почте.

Во второй функции происходит тоже самое, только наоборот: из Firebase, из коллекции «userStore», по ключу равному почте достаются данные и присваиваются в состояние приложения.

Полный код MobX класса с хранением состояния приложения можно увидеть в приложении А.

Также в меню настроек можно выключить звук у приложения, тогда не будет слышно нажатий клавиатуры, при клике на экран.

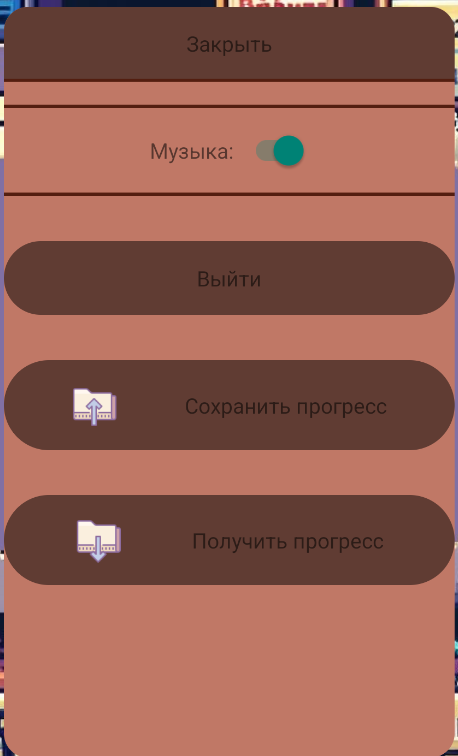


Рисунок 13. Меню настроек авторизованного пользователя

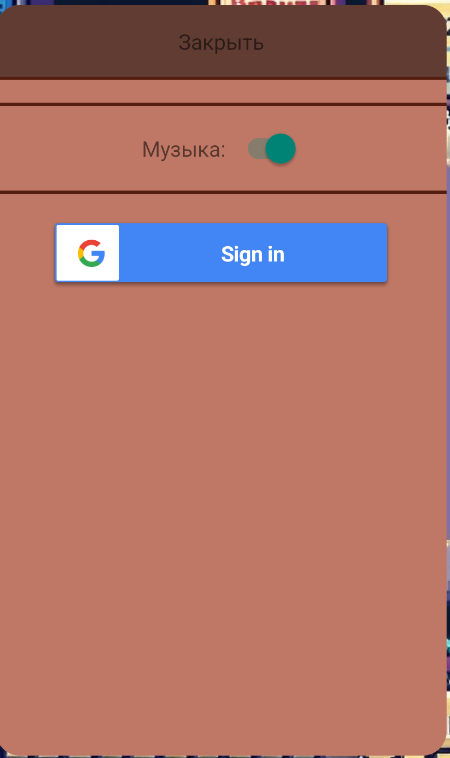


Рисунок 14. Меню настроек не авторизованного пользователя

* + 1. Меню покупки помощников

При нажатии кнопки «Открыть магазин» всплывет меню покупки помощников(рис. 15), где можно увидеть список всех помощников, их описание и уровень, до которого улучшен данный помощник.

Листинг элемента списка помощников:

const SupportItem: FC<SupportItemProps> = ({support}) => {  
 const {count} = gameStore  
  
 const onCostClick = () => {  
 if (  
 (support.has && support.cost / 5 > count) ||  
 (!support.has && support.cost > count)  
 ) {  
 return  
 }  
 gameStore.updateSupport(support.id)  
 }  
  
 return (  
 <View style={style.container}>  
 <View style={style.infoContainer}>  
 <Text style={style.level}>{support.level}</Text>  
 <View style={style.textContainer}>  
 <Text style={style.title}>  
 {support.title} (+{support.increase \* support.level})  
 </Text>  
 <Text style={style.description}>{support.description}</Text>  
 </View>  
 <Pressable style={style.chestContainer} onPress={onCostClick}>  
 <Text style={style.costText}>  
 {!support.has ? support.cost : support.cost / 5}  
 </Text>  
 </Pressable>  
 </View>  
 </View>  
 )  
}  
  
export default observer(SupportItem)

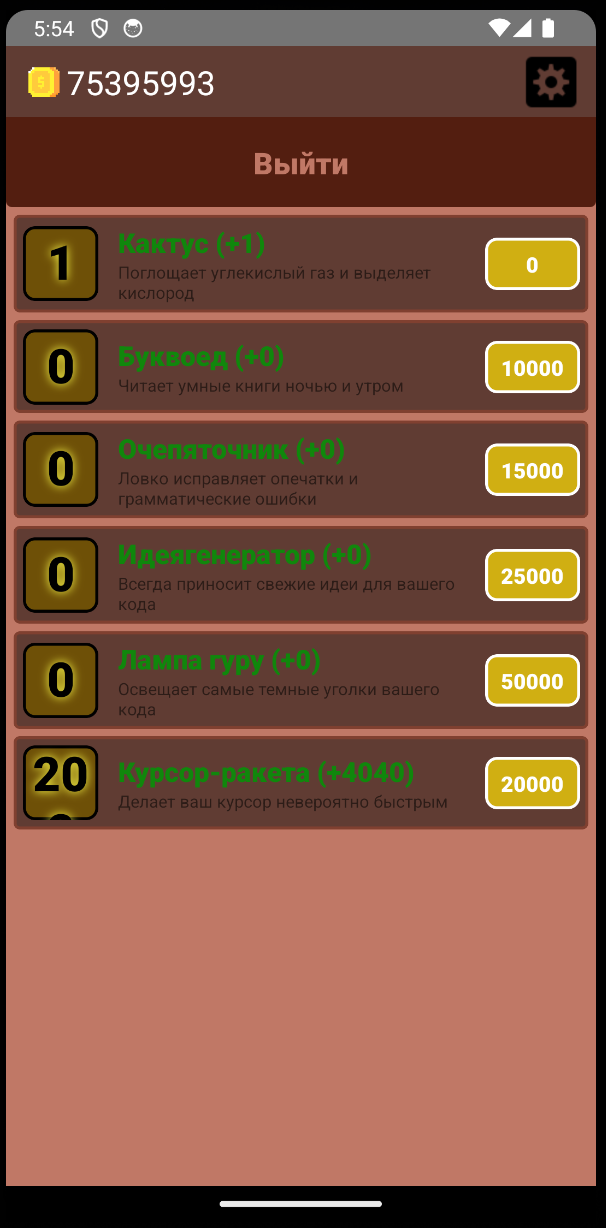


Рисунок 15. Меню покупки помощников

Если помощник еще не куплен, его уровень составляет 0 и справа указана цена его покупки. Когда пользователь покупает помощника, рядом с ним начинает показываться количество получаемых денег и цена покупки меняется на цену улучшения (цена улучшения для всех помощников составляет одну пятую от стоимости покупки)

* + 1. Меню бонусов

При нажатии кнопки «Бонусы» всплывет меню с бонусами. На данный момент из бонусов реализован сундук с деньгами. Через каждый установленный промежуток времени открывается сундук, нажав на который, можно получить дополнительные деньги.

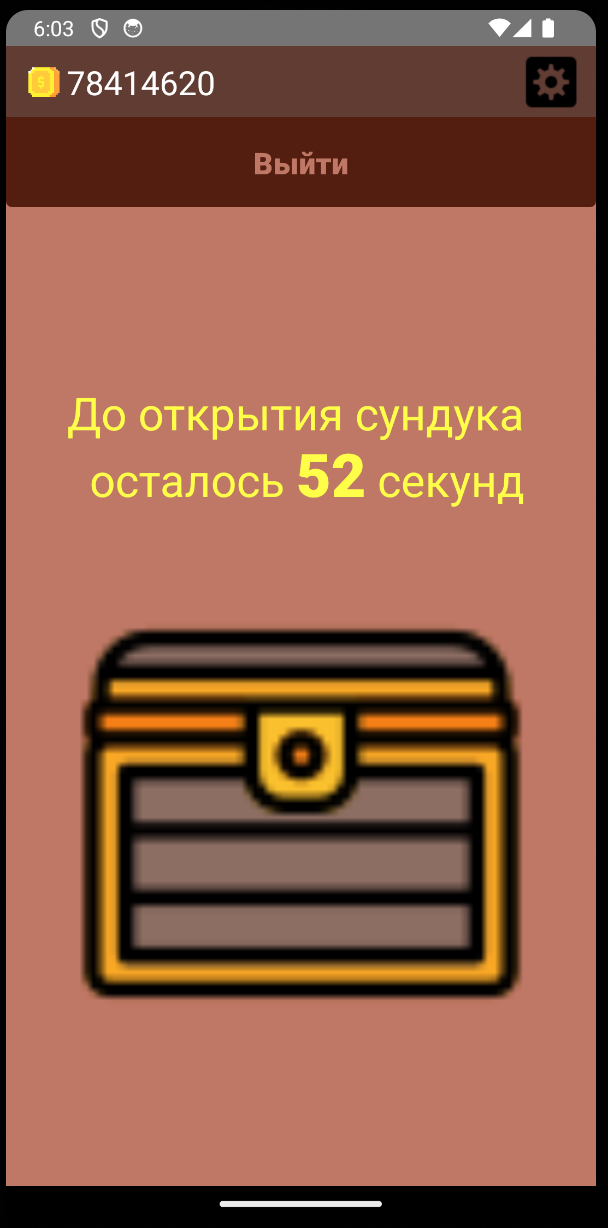


Рисунок 16. Меню бонусов

Листинг меню бонусов:

const CaseModal: FC<CaseModalProps> = ({visible, toggleVisible}) => {  
 const {caseTimer} = gameStore  
  
 const openChest = () => {  
 gameStore.openCase()  
 }  
  
 return (  
 <Modal visible={visible} animationType={'slide'} transparent={true}>  
 <View style={style.container}>  
 <Header toggleIsShopOpen={toggleVisible} />  
 <View style={style.chestContainer}>  
 {caseTimer === 0 ? (  
 <Text style={style.text}>Откройте сундук</Text>  
 ) : (  
 <Text style={style.text}>  
 До открытия сундука {'\n'} осталось{' '}  
 <Text style={style.seconds}>{caseTimer}</Text> секунд  
 </Text>  
 )}  
 <Pressable onPress={openChest}>  
 <Image  
 source={caseTimer === 0 ? Chest : CloseChest}  
 style={{width: 300, height: 300}}  
 />  
 </Pressable>  
 </View>  
 </View>  
 </Modal>  
 )  
}  
  
export default observer(CaseModal)

Открытие сундука по таймеру реализовано с помощью функции setTimeout языка JavaScript:

public setTimerForCase = () => {  
 setTimeout(() => {  
 runInAction(() => {  
 this.caseTimer -= 1  
 })  
  
 if (this.caseTimer === 0) {  
 return  
 }  
 this.setTimerForCase()  
 }, 1000)  
}

# заключение

В ходе выполнения курсового проекта была спроектирована и разработана мобильная игра кликер с возможностью покупки улучшений.

В процессе разработки данного курсового проекта были освоены такие технологии как React – построение пользовательского интерфейса, React-Native – создание кросплатформенных приложений, Firebase – облачная NoSQL база данных.

Было разработано мобильное приложение, в котором используется MobX для хранения состояния приложения на одном устройстве и Firebase для синхронизации и сохранения состояния между устройствами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Липаев В.В. Проектирование программных систем. М.: Высш. шк, 1990.
2. Буч Г. Объектно-ориентированное проектирование / Пер. с анг. Конкорд, 1996.
3. Майерс Г. Надежность программного обеспечения. М.: Мир, 1980
4. Firebase Documentation [Электронный ресурс]: Google. – Режим доступа: <https://firebase.google.com/docs?hl=ru>
5. React Documentation [Электронный ресурс]: Meta. – Режим доступа: <https://ru.reactjs.org/docs/getting-started.html>
6. React Native Documentation [Электронный ресурс]: Meta – Режим доступа: <https://reactnative.dev/>

# ПРИЛОЖЕНИЕ А

**ИСХОДНЫЙ КОД MOBX КЛАССА С СОСТОЯНИЕМ ПРИЛОЖЕНИЯ**

import {makeAutoObservable, runInAction} from 'mobx'

import AsyncStorage from '@react-native-async-storage/async-storage'

import {GameState} from '../models'

import {defaultSupports} from './supports.data'

import {generateRandomCharacter} from '../view/Main/Main.utils'

import Sound from 'react-native-sound'

import firestore from '@react-native-firebase/firestore'

import {Alert} from 'react-native'

class GameStore {

public count = 0

public string = ''

public supports = defaultSupports

public sound = new Sound('click')

public volume = 1

public caseTimer = 60

constructor() {

makeAutoObservable(this)

}

public setVolume = (volume: number) => {

this.volume = volume

}

public increaseClick = (character: string) => {

let increaseCount = 1

this.supports.forEach((support) => {

if (!support.has) return

increaseCount \*= support.multiplier

})

this.sound.setVolume(this.volume).play()

runInAction(() => {

this.count += increaseCount

this.string += character

})

}

public increaseIteration = (time: number = 1) => {

let increaseCount = 0

let countSupports = 0

this.supports.forEach((support) => {

if (!support.has) return

countSupports += 1

increaseCount += support.increase \* support.level

})

if (countSupports === 0) return

const seconds = ~~((new Date().getTime() - time) / 1000)

if (time !== 1 && seconds > 0) {

Alert.alert(

'Пассивный заработок',

`За время отсутствия в игре выы получили ${

increaseCount \* seconds

} денег`,

)

}

runInAction(() => {

this.count += increaseCount \* (time !== 1 && seconds > 0 ? seconds : 1)

this.string += generateRandomCharacter(countSupports)

})

}

public getStateFromStorage = async () => {

const gameState: GameState = JSON.parse(

(await AsyncStorage.getItem('game-state')) ?? '{}',

)

runInAction(() => {

this.count = gameState.count ? gameState.count : 0

this.string = gameState.string ? gameState.string : ''

this.supports = gameState.supports ? gameState.supports : defaultSupports

})

this.increaseIteration(gameState.lastTime)

}

public setStateToStorage = async () => {

const gameState = JSON.stringify({

count: this.count,

string: this.string,

supports: this.supports,

lastTime: new Date().getTime(),

})

await AsyncStorage.setItem('game-state', gameState)

}

public updateSupport = (id: number) => {

this.supports = this.supports.map((item) => {

if (item.id !== id) return item

if (!item.has) {

this.count -= item.cost

return {

...item,

level: 1,

has: true,

}

}

this.count -= item.cost / 5

return {

...item,

level: ++item.level,

}

})

}

public saveToFirestore = (email: string) => {

firestore().collection('userStore').doc(email).set({

count: this.count,

string: this.string,

supports: this.supports,

})

}

public getDataFromFirestore = async (email: string) => {

const data = (

await firestore().collection<GameState>('userStore').doc(email).get()

).data()

runInAction(() => {

if (data) {

this.count = data.count

this.string = data.string

this.supports = data.supports

}

})

}

public setTimerForCase = () => {

setTimeout(() => {

runInAction(() => {

this.caseTimer -= 1

})

if (this.caseTimer === 0) {

return

}

this.setTimerForCase()

}, 1000)

}

public openCase = () => {

this.caseTimer = 60

const coins = ~~(Math.random() \* 1000)

Alert.alert(

'Заначка в кармане',

`Вы порылись в кармане зимней куртки и нашли ${coins} рублей`,

)

this.setTimerForCase()

}

}

export const gameStore = new GameStore()