Реферат

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Марковский А.Г.*

Пров.

Годун А.В.

Консульт.

Годун А.В.

Н. контр.

Рыжанкова А.С.

Утв.

Пацей Н.В.

Реферат

Лит.

Листов

1

*74417049, 2021*

Пояснительная записка содержит 65 страниц, 33 рисунка, 13 таблиц, 12  источников, 6 приложений, 19 листингов.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, КРИПТОВАЛЮТНЫЙ КОШЕЛЕК, ПЛАТФОРМА РАЗРАБОТКИ IOS, SWIFT, ПАТТЕРН ПРОЕКТИРОВАНИЯ VIPER, BLOCKCHAIN, IDE XCODE

Объектом дипломного проекта является разработка мобильного приложения, выполняющего функции криптовалютного кошелька.

Цель дипломного проекта – спроектировать и разработать мобильное приложение для платформы iOS, представляющее из себя криптокошелек.

В разработке дипломного проекта был использован язык программирования Swift, протокол обмена данными HTTP.

Экономическая эффективность проекта заключаются в программной структуре проекта, благодаря которой, на ее основе могут быть разработаны подобные приложения в более сжатые сроки.

Пояснительная записка состоит из введения, шести разделов и заключения.

Во введении представлена общая информация о текущем состоянии в сфере рассматриваемой темы.

В первом разделе представлены результаты аналитического обзора предметной области и технологий разработки.

Во втором разделе представлен процесс проектирования архитектуры мобильного приложения.

В третьем разделе продемонстрирован процесс разработки мобильного приложения.

В четвертом разделе проведено тестирование мобильного приложения и представлены его результаты.

В пятом разделе представлено руководство пользователя мобильного приложения.

В шестом разделе представлены результаты себестоимости и отпускной цены разработанного мобильного приложения.

В заключении представлены итоги дипломного проекта и задачи, которые были решены в ходе разработки мобильного приложения.

Abstract

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Марковский А.Г.*

Пров.

Годун А.В.

Консульт.

Годун А.В.

Н. контр.

Рыжанкова А.С.

Утв.

Пацей Н.В.

Abstract

Лит.

Листов

1

*74417049, 2021*

The explanatory note of the diploma project contains 65 pages of explanatory note, 33 illustrations, 13 tables, 12 sources used, 6 appendices, 19 listings.

MOBILE APPLICATION, CRYPTOCURRENCY WALLET, IOS DEVELOPMENT PLATFORM, SWIFT, VIPER PATTERN DESIGN, BLOCKCHAIN, IDE XCODE

The object of the diploma project is to develop a mobile application that performs the functions of the cryptocurrency wallet.

The purpose of the graduation project is to design and develop an mobile application for the IOS platform, which represents the cryptococheries.

In the development of the graduation project, the Swift programming language was used, the HTTP data exchange protocol.

The economic efficiency of the project is to the project program structure, due to which, on its basis, such applications can be developed in more compressed time.

The explanatory note consists of an introduction, six sections, conclusion.

The introduction provides general information about the current state of affairs in the field of the topic under consideration.

The first section presents the results of an analytical review of the subject area and development technologies.

The second section introduces the process of designing a mobile application architecture.

In the third segment of the software development process.

The fourth section contains the results of software testing.

The fifth section provides a user guide for the mobile application.

The sixth section presents the results of the cost and selling price of the developed software tool.

In conclusion, the results of the diploma project and the tasks that were solved in the course of software development.

**Содержание**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Марковский А.Г.*

Пров.

Годун А.В.

Консульт.

Годун А.В.

Н. контр.

Рыжанкова А.С.

Утв.

Пацей Н.В.

Содержание

Лит.

Листов

2

*74417049, 2021*

[Введение 7](#_Toc73668574)

[1 Аналитический обзор литературы и постановка задачи 8](#_Toc73668575)

[1.1 Исследование предметной области 8](#_Toc73668576)

[1.1.1 Требования к хранению приватных данных пользователя 8](#_Toc73668577)

[1.1.2 Способы защиты действий, требующих подтверждения 9](#_Toc73668578)

[1.2 Обзор аналогов разрабатываемого мобильного приложения 10](#_Toc73668579)

[1.2.1 Мобильное приложение «*Coins*» 10](#_Toc73668580)

[1.2.2 Мобильное приложение «*TrustWallet*» 13](#_Toc73668581)

[1.3 Постановка задачи 15](#_Toc73668582)

[1.4 Выводы по разделу 16](#_Toc73668583)

[2 Проектирование мобильного приложения 17](#_Toc73668584)

[2.1 Обзор используемых технологий 17](#_Toc73668585)

[2.1.1 Платформа *iOS* 17](#_Toc73668586)

[2.1.2 Язык программирования *Swift* 18](#_Toc73668587)

[2.1.3 Интегрированная среда разработки *Xcode* 19](#_Toc73668588)

[2.1.4 Формат представления данных *JSON* 19](#_Toc73668589)

[2.2 Общая архитектура мобильного приложения 20](#_Toc73668590)

[2.3 Архитектура экранов мобильного приложения. *VIPER* модули. 23](#_Toc73668591)

[2.4 Хранение данных в мобильном приложении 24](#_Toc73668592)

[2.5 Выводы по разделу 26](#_Toc73668593)

[3 Разработка мобильного приложения 27](#_Toc73668594)

[3.1 *VIPER*-модули 27](#_Toc73668595)

[3.2 Сервисы мобильного приложения 28](#_Toc73668596)

[3.2.1 Сервисы *MAGWallet* проекта-модуля 30](#_Toc73668597)

[3.2.2 Сервисы *BTCModule*, *ETHModule*, *LTCModule* проектов-модулей 30](#_Toc73668598)

[3.2.3 Сервисы *SharedFilesModule* проекта-модуля 30](#_Toc73668599)

[3.3 *Core*-компоненты мобильного приложения 31](#_Toc73668600)

[3.3.1 *Core*-компоненты *MAGWallet* проекта-модуля 32](#_Toc73668601)

[3.3.2 *Core*-компоненты *BTCModule*, *ETHModule*, *LTCModule* модулей 32](#_Toc73668602)

[3.3.3 *Core*-компонент *ProtectedStorage* проекта-модуля *SharedFilesModule* 32](#_Toc73668603)

[3.4 *Event*-прокси мобильного приложения 33](#_Toc73668604)

[3.4.1 *Event*-прокси *MAGWallet* проекта-модуля 35](#_Toc73668605)

[3.4.2 *Event*-прокси *BTCModule*, *ETHModule*, *LTCModule* проектов 35](#_Toc73668606)

[3.4.3 *Event*-прокси *SharedFilesModule* проекта-модуля 35](#_Toc73668607)

[3.5 Хранение данных в мобильном приложении 36](#_Toc73668608)

[3.6 Функция автоблокировки мобильного приложения 36](#_Toc73668609)

[3.7 Реализация мультиаккаунтности в мобильном приложении 38](#_Toc73668610)

[3.8 Реализация валидации суммы транзакции 38](#_Toc73668611)

[3.9 Выводы по разделу 40](#_Toc73668612)

[4 Тестирование мобильного приложения 41](#_Toc73668613)

[4.1 Unit тестирование 41](#_Toc73668614)

[4.2 Ручное тестирование 44](#_Toc73668615)

[4.2.1 Первоначальная настройка мобильного приложения 44](#_Toc73668616)

[4.2.2 Импорт криптовалютного кошелька 45](#_Toc73668617)

[4.2.3 Формирование транзакции 46](#_Toc73668618)

[4.2.4 Блокировка продвижения по сценарию использования 47](#_Toc73668619)

[4.3 Выводы по разделу 47](#_Toc73668620)

[5 Руководство пользователя 48](#_Toc73668621)

[5.1 Первоначальная настройка мобильного приложения 48](#_Toc73668622)

[5.2 Кошелек. Список и отправка транзакций 51](#_Toc73668623)

[5.3 Настройки мобильного приложения 53](#_Toc73668624)

[5.3.1 Экран настроек *Accounts* 54](#_Toc73668625)

[5.3.2 Экран настроек *Autoblock* 55](#_Toc73668626)

[5.3.3 Экран настроек *Change* *Network* 55](#_Toc73668627)

[5.4 Дополнительные возможности мобильного приложения 55](#_Toc73668628)

[5.5 Выводы по разделу 57](#_Toc73668629)

[6 Экономическое обоснование цены мобильного приложения 58](#_Toc73668630)

[6.1 Общая характеристика разрабатываемого мобильного приложения 58](#_Toc73668631)

[6.2 Исходные данные и маркетинговый анализ 58](#_Toc73668632)

[6.3 Методика обоснования цены 59](#_Toc73668633)

[6.3.1 Объем мобильного приложения 60](#_Toc73668634)

[6.3.2 Основная заработная плата 61](#_Toc73668635)

[6.3.3 Дополнительная заработная плата 61](#_Toc73668636)

[6.3.4 Отчисления в Фонд социальной защиты населения 61](#_Toc73668637)

[6.3.5 Расходы на материалы 62](#_Toc73668638)

[6.3.6 Расходы на оплату машинного времени 62](#_Toc73668639)

[6.3.7 Прочие прямые затраты 62](#_Toc73668640)

[6.3.8 Накладные расходы 63](#_Toc73668641)

[6.3.9 Сумма расходов на разработку мобильного приложения 63](#_Toc73668642)

[6.3.10 Расходы на сопровождение и адаптацию 63](#_Toc73668643)

[6.3.11 Полная себестоимость 63](#_Toc73668644)

[6.3.12 Определение цены, оценка эффективности 64](#_Toc73668645)

[6.4 Вывод по разделу 64](#_Toc73668646)

[Заключение 66](#_Toc73668647)

[Список использованных источников 67](#_Toc73668648)

[ПРИЛОЖЕНИЕ А Диаграмма вариантов использования 68](#_Toc73668649)

[ПРИЛОЖЕНИЕ Б Диаграмма классов BTCAuth сервиса 69](#_Toc73668650)

[ПРИЛОЖЕНИЕ В Блок-схема процесса автоблокировки приложения 70](#_Toc73668651)

[ПРИЛОЖЕНИЕ Г Блок-схема процесса создания Bitcoin криптокошелька 71](#_Toc73668652)

[ПРИЛОЖЕНИЕ Д Скриншоты работы мобильного приложения 72](#_Toc73668653)

[ПРИЛОЖЕНИЕ Е Блок-схема процесса создания Bitcoin транзакции 73](#_Toc73668654)

# **Введение**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Марковский А.Г.*

Пров.

Годун А.В.

Консульт.

Годун А.В.

Н. контр.

Рыжанкова А.С.

Утв.

Пацей Н.В.

Введение

Лит.

Листов

1

*74417049, 2021*

С развитием электронных систем неоднократно возникали идеи создать электронный аналог наличных денег. Но проблемой становилась потенциальная возможность двойного расходования одних и тех же средств. При оплате наличными двойного расходования не возникает из-за того, что оплата сопровождается передачей денег, но электронным системам присуща возможность копирования состояния, что позволяет произвести несколько платежей в разных направлениях. Проблема решалась лишь с помощью доверенных посредников, которые ведут учет платежей и гарантируют операции исключительно в рамках наличия средств.

Технология криптовалют изначально нацелена на отсутствие доверенного узла за счет искусственного усложнения внесения изменений в реестр истории операций. Отсутствие какого-либо администратора приводит к тому, что никто не может воздействовать на транзакции участников сети.

Вышеописанная особенность подтолкнула общество к изучению и использованию различных систем, работающих с криптовалютами: бирж, кошельков, хранилищ приватных данных и других. В связи с повышение спроса на такие системы, появилась необходимость создания удобных и быстрых интерфейсов доступа к повседневным процессам.

Исходя из этого, целью дипломного проекта является разработка мобильного приложения, представляющее из себя криптокошелек для платформы *iOS* со следующими функциональными возможностями:

* приложение должно позволять создавать или импортировать криптокошельки *Bitcoin*, *Ethereum*, *Litecoin* криптовалют;
* функция отправки транзакций поддерживаемых криптовалют;
* возможность просмотра истории транзакций каждого кошелька;
* возможность добавление нескольких кошельков какой-либо поддерживаемой криптовалюты (мультиаккаунтность);
* возможность выбора рабочей сети между *mainnet* и *testnet*;
* функция сканирования *QR*-кодов, содержащих адреса крипто-кошельков, для дальнейшего создания транзакции;
* функция генерации *QR*-кода, содержащего адрес криптокошелька пользователя;
* возможность выбора времени автоблокировки приложения;
* функция защиты значимых операций пин-кодом.

# **1 Аналитический обзор литературы и постановка задачи**

ФИО

Подпись

Дата

Лист

1

БГТУ 01.00.ПЗ

Разраб.

*Марковский А.Г.*

Пров.

Годун А.В.

Консульт.

Годун А.В.

Н. контр.

Рыжанкова А.С.

Утв.

Пацей Н.В.

1 Аналитический обзор литературы и постановка задачи

Лит.

Листов

9

*74417049, 2021*

Для создания мобильного приложения с уникальным дизайном, удобным и простым для понимания интерфейсом, всем необходимым техническим функционалом, нужно тщательно изучить уже представленные на рынке аналоги. Это даст возможность выделить успешные дизайнерские решения, организовать грамотную структуру страниц, определить необходимые модули, отметить все достоинства и недостатки, что в результате поможет разработать лучший вариант продукта для конечного клиента.

Одними из важнейших аспектов при оценке аналогов были функциональность в связке с простой восприятия интерфейса, уровень безопасности и заботы о данных пользователя, так как это определяет дальнейшее отношение пользователей к системе

С появлением криптовалют стало необходимо наличие надежного места, где эти новые цифровые активы можно хранить. На данный момент у пользователей есть три варианта:

* хранить криптовалюту непосредственно на бирже, где она была куплена. Однако, известно, что криптовалютные платформы закрываются, замораживают средства пользователей или подвергаются взлому;
* купить аппаратный кошелек;
* использовать онлайн-кошелек, чтобы активы всегда были доступны и защищены от хакеров. Для инвесторов самый удобный способ хранения цифровых активов – это онлайн-кошельки, поскольку они позволяют получать доступ к своим активам в любое время и из любого места. Сегодня есть множество сервисов, предлагающих инвесторам возможность хранить свои криптовалюты [1].
  1. Исследование предметной области
     1. **Требования к хранению приватных данных пользователя**

Безопасность приложений – один из самых важных аспектов разработки программного обеспечения. Пользователи приложений надеются, что информация, которую они предоставляют, надежно защищена, поэтому нельзя так просто предоставлять кому-либо конфиденциальную информацию.

Исследования, проведенные над множеством *iOS* приложений, показывают, что во многих допускается одна и та же ошибка – конфиденциальная информация хранится там, где ее быть не должно.

В *AppStore* были найдены приложения, которые хранят данные в *UserDefaults*, такие как: токены, активные и возобновляемые подписки, число доступных денег и другое.

Все эти данные можно получить и использовать со злым умыслом, начиная от управления платными подписками в приложении и заканчивая взломом на сетевом уровне.

Если разработчик решает хранить данные личного характера в *UserDefaults*, то он подвергает ее риску. *UserDefaults* является хранилищем, где данные хранятся в файле со списком свойств, который находится внутри папки «Настройки» в приложении. Данные сохраняются в приложении без шифрования. В *UserDefaults* следует хранить только небольшой объем информации, такой как настройки внутри приложения, то есть данные, которые не являются конфиденциальными для пользователя [2].

Реализация хэширования вручную может оказаться чересчур сложной, поэтому в разрабатываемом программном средстве допустимо воспользоваться библиотекой «*CryptoSwift*». В ней собрано множество стандартных надежных алгоритмов шифрования, применяемых в *Swift*.

Никогда не стоить пренебрегать безопасностью разрабатываемого приложения. Понимание разницы и последствий при хранении конфиденциальных данных в *UserDefaults* или *Keychain* является очень важным аспектом разработки под мобильную платформу *iOS*.

* + 1. **Способы защиты действий, требующих подтверждения**

В программных средствах, работающих с приватными данными, такими как, например, валютные активы пользователя, все значимые действия должны быть подтверждены непосредственно перед их программным выполнением для того, чтобы удостовериться в том, что именно владелец этих данных производит операции над ними и для того, чтобы дать пользователю понимание важности предстоящего действия, которое, возможно, в дальнейшем нельзя будет обратить.

В данный момент есть много разнообразных способов решения задачи об подтверждении действий:

* пин-код или пин-пароль;
* одноразовые пароли на бумаге;
* отправка временного кода или ссылки на адрес электронной почты;
* отправка одноразового пароля по *SMS*;
* *OTP*-токены;
* биометрическое сканирование.

Одноразовые пароли на бумаге, как правило, используют онлайн-банки. Можно, например, получить в офисе кредитной организации карту со стираемым покрытием поверх полей с кодами или просто распечатать набор паролей в банкомате, а система дистанционного обслуживания при совершении транзакции на сайте будет просить ввести одноразовый набор символов под определенным номером.

Отправка пароля по электронной почте – это максимально упрощенный вариант защиты. Для аутентификации пользователя нужно знать всего лишь его адрес. Соответственно, уровень защиты таким методом ниже, чем в других сценариях, особенно если используется один пароль для входа в почту и на целевой ресурс.

Еще одним способом двухфакторной аутентификации является отправка одноразового пароля по *SMS*. Однако коды в *SMS*-сообщениях тоже не очень надежный вариант. Во-первых, такой пароль можно подсмотреть при помощи уведомлений на экране блокировки смартфона. Во-вторых, сообщение может быть перехвачено вредоносной программным обеспечением. Распространен также вид мошенничества, при котором путем обмана или сговора в салоне сотовой связи можно получить новую *SIM*-карту с нужным номером, и *SMS*-сообщения будут приходить на нее, а телефон жертвы даже не сможет подключиться к сети.

Более надежным является аппаратный *OTP*-токен. Он представляет собой устройство в виде брелка с дисплеем и кнопкой. В памяти устройства заранее программируется определенное количество паролей. При нажатии на кнопку коды отображаются на дисплее [3].

Пин-код или пин-пароль устанавливается самим пользователем в основном при регистрации и используется в дальнейшем для подтверждения операций. Недостатки данного способа очевидны – код или пароль не является одноразовым, может совпадать с паролями других сервисов, а также может быть крайне простым, что не вызовет сложностей у злоумышленников при попытке взлома. Однако у пин-кода есть и преимущество – простота и скорость ввода, что может быть допустимо, в зависимости от разрабатываемого программного средства.

Биометрическое сканирование предполагает непосредственный физический контакт владельца с устройством, с которого производится операция, что, в большинстве случаев, является довольно надежным способом защиты.

* 1. Обзор аналогов разрабатываемого мобильного приложения
     1. **Мобильное приложение «Coins»**

*Coins* – это мобильное приложение, которое позволяет пользователям отправлять, получать, хранить, инвестировать и изучать криптовалюты.

Одна из самых популярных функций, которая выделяет его среди конкурентов, это возможность покупки криптовалют напрямую через *Apple Pay*.

*Coins* очень серьезно относится к безопасности – он защищен биометрическими технологиями и компанией по кибербезопасности *Grey* *Wizard*.

В кошельке используется кроссплатформенная библиотека *trust-wallet-core*, которая реализует низкоуровневую функциональность криптографического кошелька для всех поддерживаемых блокчейнов.

Другая особенность *Coins* в том, что с пользователей взымается очень низкая комиссия за проведение транзакций. Независимо от того, совершает ли пользователь транзакции с *Bitcoin*, *Ethereum* или *Litecoin* [4].

Версия криптокошелька для *iOS* доступна для загрузки из *AppStore*. Что касается версии *Android*, она готова к бета-тестированию.

Логически приложение разделено на пять разделов, где каждый выполняет свою отдельную функцию для пользователя. Раздел «Сегодня» отображает данные об общем состоянии криптовалютного рынка, значимые новости и публикации, а также лидеров роста стоимости за последние сутки (рисунок 1.1).



Рисунок 1.1 – Раздел приложения «Сегодня»

Раздел «Монеты» выводит рыночную информацию о каждой монете. Данные могут быть отсортированы по множеству различных параметров, что дает пользователю возможность самому контролировать и анализировать информацию, выводимую на экран. Снимок экрана раздела приведен на рисунке 1.2.



Рисунок 1.2 – Раздел приложения «Монеты»

Раздел «Мои монеты» позволяет взаимодействовать пользователю с его личными активами – управлять текущими счетами, создавать новые, отправлять и принимать транзакции. Снимок экрана раздела приведен на рисунке 1.3.



Рисунок 1.3 – Раздел приложения «Мои монеты»

Из достоинств приложения можно выделить приятный дизайн в стиле нативных *iOS* приложений, высокий уровень безопасности, который выражается в максимальном сокрытии приватных данных от глаз, невозможность производить операции со счетом до сохранения мнемоника в надежном месте, возможность использования биометрического функционала устройства для защиты входа и основных операций.

Явным минусом приложения является отсутствие возможности добавить несколько адресов одной валюты, что накладывает ограничения на пользователя, так как ему необходимо искать другой способ сделать это, если возникнет такая необходимость, что, скорее всего, повлечет за собой переход пользователя на платформу конкурентов и, как следствие, финансовые убытки.

**1.2.2 Мобильное приложение «TrustWallet»**

*TrustWallet* – официальный криптокошелек *Binance*, одной из крупнейших в мире бирж криптовалют и насчитывающий более 400 тысяч пользователей в месяц.

Он предлагает пользователям простое мобильное приложение для управления их токенами и криптовалютами, в то же время позволяет полностью контролировать личные ключи. Ключи хранятся локально и защищены от любой потенциальной атаки множеством уровней безопасности, гарантируя, что никто и никогда несанкционированно не получит к ним доступ [4].

Особенность кошелька в том, что он отлично работает с протоколами *Binance* *DEX* и *Kyber* *Network*, что позволяет совершать мгновенные сделки на децентрализованной бирже. Безопасность данных пользователей действительно внушительна для относительно простого криптокошелька, поэтому даже сама компания-разработчик не получит доступ к личной информации.

К особенностям приложения можно отнести:

* закрытые ключи пользователей хранятся на их собственных устройствах;
* для быстрого восстановления доступа предусмотрено средство резервного копирования;
* встроенный браузер *Web3* позволяет взаимодействовать с децентрализованными приложениями;
* возможность покупки и продажи монет осуществляется через подключение к *Binance* *DEX*;
* кошелек анонимный, что позволяет пользоваться им, не предоставляя личных данных;
* открытый исходный код и регулярные аудиты безопасности от сертифицированных специалистов;
* возможность импортировать в *TrustWallet* большинство других крипто-кошельков.

У *TrustWallet* есть встроенный веб-браузер, который позволяет легко и безопасно пользоваться веб-приложениями. Кошелек совместим со многими криптовалютами и протоколами, в том числе *ETH*, *BTC*, *XRP*, *TRX*, *XLM* и *BNB*. Кроме того, он предлагает полную поддержку любых токенов *ERC-20* в сети *Ethereum* и *BEP2* в блокчейне *Binance*. Официальные версии кошелька доступны для *Android* и *iOS* в *Google* *Play* и *App* *Store*. Пользователи *MacOS* уже сейчас могут пользоваться *desktop* версией кошелька. Версии для *Windows* и *Linux* находятся в разработке.

Логически приложение разделено на два основных раздела и один вспомогательный. Раздел «*Wallet*» отвечает за взаимодействие пользователя с его криптоактивами. Раздел содержит возможность просмотра баланса адресов каждой добавленной монеты с предоставлением информации об актуальной стоимости токена и процентом суточного изменения цены за единицу криптовалюты, отображение истории транзакций с детализацией, функции отправки транзакции и запроса средств путем генерации *QR*-кода с закодированными в него данными об адресе пользователя и сумме предполагаемой транзакции. Снимок экрана раздела приведен на рисунке 1.4.



Рисунок 1.4 – Раздел приложения «*Wallet*»

Следующим основным разделом приложения является раздел «*DEX*», в котором находится функциональность по обмену одних монет на другие, что исключает затраты на перевод монет в базовые электронные валюты и обратно в криптовалюты, а также возможность покупки криптовалют через стандартные банковские системы через систему оплаты.

*TrustWallet* подключается непосредственно к *Binance* *DEX*, чтобы дать пользователям возможность торговать активами. Внешне все выглядит как традиционная криптобиржа с книгой заказов, где можно создавать свои заказы на покупку и продажу. Следует помнить, что торговля на бирже требует сетевой комиссии, поскольку деньги переводятся напрямую с кошелька на кошелек.

Продажа осуществляется точно так же, необходимо переключиться на раздел «Продать» в поле создания заказа.

Основной проблемой в подобных разделах является сложность интерфейса для тривиальных пользователей. В *TrustWallet*, в свою очередь, раздел выполнен лаконично – без лишней нагруженности, которая может отпугивать. Снимок экрана раздела «*DEX*» приведен на рисунке 1.5.



Рисунок 1.5 – Раздел приложения «*DEX*»

*TrustWallet* предоставляет простой способ безопасно хранить такие активы, где токены связаны с публичным адресом и защищены средствами блокчейна.

* 1. Постановка задачи

В рамках дипломного проекта необходимо разработать мобильное приложение для взаимодействия пользователя с его криптоактивами.

Основные требования к проекту:

* приложение должно позволять создавать или импортировать криптокошельки *Bitcoin*, *Ethereum*, *Litecoin* криптовалют;
* возможность отправки транзакций поддерживаемых криптовалют между различными адресами;
* возможность просмотра истории транзакций каждого кошелька;
* возможность добавления нескольких кошельков какой-либо поддерживаемой криптовалюты (мультиаккаунтность);
* возможность выбора рабочей сети между *mainnet* и *testnet*;
* функция сканирования *QR*-кодов, содержащих адреса крипто-кошельков, для дальнейшего создания транзакции;
* функция генерации *QR*-кода адреса криптокошелька пользователя;
* возможность выбора времени автоблокировки приложения;
* функция защиты значимых операций пин-кодом.

На основе диаграммы вариантов использования, представленной на рисунке 1.6, а также в приложении А, требуется реализовать необходимые сценарии использования мобильного приложения пользователем.

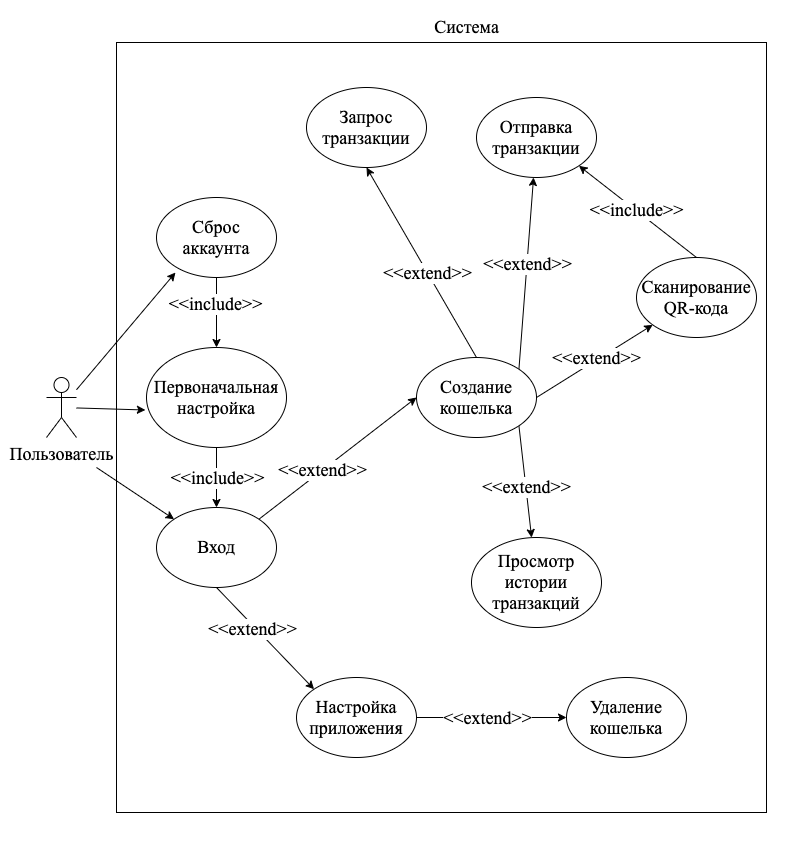


Рисунок 1.6 – Диаграмма вариантов использования мобильного приложения

* 1. Выводы по разделу

Проанализировав существующие аналоги, можно сказать, что на сегодняшний день существует некоторое количество мобильных приложений, предоставляющих функционал по доступу к криптовалютным активам в той или иной мере, но имеющих значительные недостатки, серьезно влияющие на пользовательский опыт. В разрабатываемом мобильно приложении дипломного проекта они должны быть решены.

В разделе составлена постановка задачи, включающая обязательный набор функций и предполагаемый технологический стек разработки. По результатам исследования были выделены основные требования к хранению конфиденциальных данных пользователя и изучены основные способы защиты выполнения значимых действий. У создаваемого мобильного приложения было обнаружено несколько аналогов, но анализ показывает то, что ни в одном из них нет полного комплекса всех предполагаемых возможностей.

Суммируя все вышеперечисленное можно сказать, что работа проделанная по аналитическому обзору литературы является крепким фундаментом для разрабатываемого программного средства. На его основе можно преступать разработке дизайна пользовательского интерфейса приложения, проектированию архитектуры приложения и программной его реализации.

# **2 Проектирование мобильного приложения**

ФИО

Подпись

Дата

Лист

1

БГТУ 02.00.ПЗ

Разраб.

*Марковский А.Г.*

Пров.

Годун А.В.

Консульт.

Годун А.В.

Н. контр.

Рыжанкова А.С.

Утв.

Пацей Н.В.

2 Проектрование мобильного приложения

Лит.

Листов

10

*74417049, 2021*

## 2.1 Обзор используемых технологий

Проектирование программного средства начинается с важнейшего этапа – определения технологий, которые будут применены при разработке системы. Именно понимание достоинств и недостатков тех или иных технологий, алгоритмов, паттернов и подходов проектирования обеспечит надежную, тестируемую и масштабируемую архитектуру в будущем. Далее будут рассмотрены технологии, примененные в разработке данного мобильного приложения.

### **2.1.1 Платформа iOS**

На данный момент сфера мобильной разработки поделена между двумя основными платформами – *iOS* и *Android*. В текущих условиях мобильное приложение будет разрабатываться лишь для платформы *iOS*, так как, опираясь на исследования, она является более безопасной из-за того, что имеет закрытую файловую организацию операционной системы, т.е. пользователь не имеет доступа к файлам устройства, что неоспоримый плюс, когда дело касается разработки приложения для взаимодействия с валютными активами пользователя, где на первое место ставится защищенность данных пользователя.

*iOS* – популярная мобильная операционная система для смартфонов, электронных планшетов, проигрывателей и некоторых других устройств, разрабатываемая и выпускаемая американской компания *Apple* [5], которая является лидеров в области разработки программного обеспечения и производства носимых устройств.

В *iOS* на базовом уровне используется ядро *XNU*, а основой ядра *XNU*, в свою очередь, является простое микроядро *Mach*, которое запускает монолитное ядро как отдельный процесс. За счет этого достигается некоторая гибкость – на одном экземпляре *Mach* можно запустить несколько экземпляров монолитного ядра, но также это связано с потерями производительности из-за большего количества переключений контекста и необходимости отображения или копирования содержимого сообщений между адресными пространствами [6].

Ядро *iOS* почти идентично ядру настольной операционной системы *Apple macOS*. Начиная с самой первой версии, операционная система работает только на планшетных компьютерах и смартфонах с процессорами архитектуры *ARM*, а с недавнего времени, мобильный приложения можно запускать и на компьютерах *Mac*, оснащенных процессорами *M1*.

На данный момент актуальной версией операционной системы является версия 14.6.

### **2.1.2 Язык программирования Swift**

*Swift* – компилируемый язык программирования общего назначения, созданный компанией *Apple* в первую очередь для разработчиков программного обеспечения для платформ *iOS* и *macOS*, поддерживающий так же разработку серверов. *Swift* работает с фреймворками *Cocoa* и *Cocoa Touch*, совместим с основной кодовой базой *Apple*, написанной на языке *Objective-C* [7].

Язык программирования задумывался как более легкий для чтения и устойчивый к ошибкам программиста язык, нежели предшествовавший ему *Objective-C*. Программы компилируются при помощи *LLVM*, входящей в интегрированную среду разработки *Xсode*.

*Swift* позаимствовал довольно многое из *Objective-C*, однако он определяется не указателями, а типами переменных, которые обрабатывает компилятор. По аналогичному принципу работают многие скриптовые языки. В то же время, он предоставляет разработчикам многие функции, которые прежде были доступны в *C++* и *Java*, такие как определяемые наименования, обобщения и перегрузка операторов. *Swift* может использовать *runtime* *Objective-C* и *С*, что делает возможным использование этих языков в рамках одной программы.

Часть функций языка выполняется гораздо быстрее по сравнению с другими языками программирования. Например, сортировка комплексных объектов выполняется в 3,9 раз быстрее, чем в *Python*, и почти в 1,5 раза быстрее, чем в *Objective-C*.

*Apple* разделила код *Swift* на несколько открытых репозиториев. Компилятор и стандартная библиотека:

* «*Swift*» – основной репозиторий, который содержит исходный код для компилятора, стандартную библиотеку и *SourceKit*;
* «*Swift-Evolution*» – документы, относящиеся к продолжающемуся развитию языка, включая цели для предстоящих выпусков, предложения для изменений и расширений.

Библиотеки ядра:

* «*Swift corelibs-foundation*» – исходный код библиотеки *Foundation*, которая предоставляет общую функциональность для всех приложений;
* «*Swift corelibs-libdispatch*» – исходный код для библиотеки *libdispatch*, предоставляющая примитивы параллелизма для работы на многоядерном аппаратном обеспечении;
* «*Swift corelibs-xctest*» – исходный код библиотеки *XCTest*, которая обеспечивает фундаментальную инфраструктуру тестирования для приложений и библиотек.

*Swift* опирается на несколько других проектов с открытым кодом, поэтому имеет некоторые клонированные репозитории:

* «*Swift llvm*» – исходный код *LLVM* с частью *Swift* дополнений;
* «*Swift clang*» – исходный код для *Clang*, модернизированный под *Swift*;
* «*Swift lldb*» – исходный код *LLDB* для отладки программ.

На данный момент актуальной версией языка программирования является версия 5.4.

### **2.1.3 Интегрированная среда разработки Xcode**

*Xcode* – это бесплатная интегрированная среда разработки, предназначенная для разработки приложений самого различного масштаба преимущественно экосистемы *Apple* – носимых устройств *iPad*, *iPhone* и компьютеров *Mac*.

*Xсode* предоставляет инструменты для управления всем потоком операций разработки – для создания приложения, его тестирования, оптимизации и публикации в магазин приложений *App Store* [8].

Для удобства создания новых проектов присутствует большой выбор готовых шаблонов для разработки под все устройства. Помимо шаблонов для разработки приложений, есть выбор для создания различных библиотек.

*IDE* «*Xcode*» включает в себя:

* интегрированный редактор кода;
* средство «*Interface Builder*» использующееся для создания графических интерфейсов;
* управление ресурсами приложения;
* тестирование и отладку в окне рабочей области;
* набор компиляторов «*GNU Compiler Collection*» и поддерживает языки *C*, *C++*, *Objective-C*, *Objective-C++*, *Swift*, *Java*, *AppleScript*, *Python* и *Ruby*;
* часть документации для разработчика от *Apple*.

На момент написания дипломного проекта *IDE* «*Xcode*» представлена в 12 версиях, а актуальной является версия 12.5.

### **2.1.4 Формат представления данных JSON**

За представление структурированных данных отвечает стандартный текстовый формат под названием *JSON*, аббревиатура которого расшифровывается как «*JavaScript Object Notation*».

*JSON* – текстовый формат данных, используемый практически во всех скриптовых языках программирования, однако его истоки находятся у *JavaScript*. Он имеет сходство с буквенным синтаксисом данного языка программирования, но может использоваться отдельно от него.

Многие среды разработки отлично справляются с его чтением и генерированием. *JSON* находится в состоянии строки, поэтому позволяет передавать информацию по сети. Он преобразуется в объект *JavaScript*, чтобы пользователь мог прочитать эти данные. Осуществляется это методами языка программирования, но сам *JSON* методов не имеет, только свойства [9].

Пользователь может сохранить текстовый файл *JSON* в его собственном расширении «*.json*», где содержимое будет отображаться как текст. *MIME* типом для формата является «*application/json*».

*JSON* используется для обмена данными, которые являются структурированными и хранятся в файле или в строке кода. Числа, строки или любые другие объекты отображаются в виде текста, поэтому пользователь обеспечивает простое и надежное хранение информации.

Формат обладает рядом преимуществ, которые и сделали его популярным:

* не занимает много памяти, является компактным в написании и быстро компилируется;
* создание текстового содержимого понятно человеку, просто в реализации, а чтение со стороны среды разработки не вызывает проблем;
* структура преобразуется для чтения на любых языках программирования;
* практически все языки имеют соответствующие библиотеки или другие инструменты для чтения данныхформата.

## 2.2 Общая архитектура мобильного приложения

Архитектура разрабатываемого мобильного приложения представляет собой несколько проектов-модулей, каждому из которых отведена своя задача, образуя иерархию зависимостей между собой. Общая структура мобильного приложения представлена на рисунке 2.1.

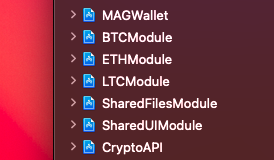


Рисунок 2.1 – Общая структура мобильного приложения

Модуль *SharedUIModule* является проектом-модулем, инкапсулирующим в себе программные объекты, а именно классы, структуры и ресурсы, связанные с пользовательским интерфейсом, которые, как следует из названия модуля, являются общими для всех остальных модулей проекта. Такими компонентами являются классы нестандартных элементов интерфейса, цвета, изображения и другое, что используется во всех частях приложения.

Такой подход позволяет вынести общие элементы в отдельное место, где, при необходимости, они могут быть изменены единожды, но, при этом, изменения коснутся всех областей, где они были применены. Структура модуля продемонстрирована на рисунке 2.2.

Задача проекта-модуля *SharedFilesModule* схожа с задачей *SharedUIModule*, но, если в случае *SharedUIModule* общие элементы относились к пользовательскому интерфейсу приложения, то в данном случае такие компоненты связаны к бизнес-логике приложения.

Модуль объединяет в себе различные классы, структуры, перечисления, расширения классов, интерфейсы и модели, доступ к которым может быть получен другими проектами-модулями для выполнения определенной логики. Такими компонентами являются, например, классы некоторых *core*-компонентов, сервисов, сущностей-помощников, перечисления глобальных ошибок и некоторые другие.

Исходя из соображений разделения ответственностей и зависимостей, взаимодействие между модулями *SharedFilesModule* и *SharedUIModule* недопустимо, так как их области их задач существенно разнятся.



Рисунок 2.2 – Структура *SharedUIModule* модуля

Структура проекта-модуля *SharedFilesModule* отображена на рисунке 2.3.



Рисунок 2.3 – Структура *SharedFilesModule* модуля

*BTCModule*, *ETHModule*, *LTCModule* – модули *Bitcoin*, *Ethereum* и *Litecoin* криптовалют соответственно. В данных модулях находится *UI* слой, бизнес-логика и, также, тесты к соответствующей валюте.

Данные модули организованы идентично, так как построены по одному шаблону. Выбранный подход позволяет реализовать поддержку других криптовалют путем тривиального добавления уже разработанного на стороне проекта-модуля необходимой валюты в данный проект.

Иерархию взаимодействия модулей выстроена следующим образом – любой из этих модулей может включать в себя модули *SharedFilesModule* и *SharedUIModule*, а также *CryptoAPI* модуль, который будет рассмотрен далее.

Модуль *CryptoAPI* является разработанной библиотекой, выступающей в роли обертки над публичным *API* доступа в блокчейн сеть и необходим для выполнения запросов. *CryptoAPI* модуль носит вспомогательную функцию и может использоваться в *BTCModule*, *ETHModule*, *LTCModule* и *MAGWallet* модулях. Физическая структура модуля изображена на рисунке 2.4.



Рисунок 2.4 – Физическая структура *CryptoAPI* модуля

Основой приложения, которая объединяет в себе и взаимодействует со всеми вышеперечисленными модулями, является модуль *MAGWallet*, отвечающий за жизненный цикл приложения, построение экранов и навигацию между ними, следит за некоторыми возможностями устройства, такой как, например, доступ в Интернет.

## 2.3 Архитектура экранов мобильного приложения. VIPER модули.

Архитектура экранов мобильного приложения в рамках дипломного проекта основана на архитектурном паттерне «*VIPER*». В данный момент паттерн лидирует в сфере разработки под мобильную платформу *iOS*, так как решает известные проблемы архитектуры *MVC*, предложенной *Apple* как эталон.

Основный недостаток *MVC* в *iOS* – массивность каждого из компонентов архитектуры при построении сложных экранов, что затрудняет читаемость, легкость восприятия, тестирование и масштабирование кодовой базы [10].

*VIPER*, в свою очередь, решает возникающую проблему, разделяя каждый модуль на, как минимум, пяти частей:

* v*iewController*;
* *presenter*;
* *interactor*;
* *router*;
* *protocols*.

Детальнее будет рассмотрена каждая составляющую модуля. Пример структуры модуля изображен на рисунке 2.5.



Рисунок 2.5 – Пример физической структуры *VIPER*-модуля

*ViewController* – класс-наследник класса *UIViewController* библиотеки *UIKit,* отвечающий подготовку и логику отображения элементов пользовательского интерфейса и данных приходящих из вне, настройку локализации. Дополнительной задачей класса является обработка и передача *UI* событий, таких как, например, нажатия кнопок, расположенных на данном экране.

Составляющая молуля «*Interactor*» выполняет основную логику получения, обработки и подготовки данных для передачи в *ViewController* для дальнейшего их отображения на экране. *Interactor* держит зависимости на сервисы, *core*-компоненты и различные прокси-сервисы, реагируя на события приложения. Например, при потере соединения с Интернетом в *Interactor* экрана приходит соответствующее событие. Отвечая на него, генерируется команда отобразить ошибку на экране.

*Router* необходим *VIPER*-модулю для сборки других модулей и обеспечения навигации в случаях, когда один экран порождает другой.

*Presenter* является связующим звеном между *ViewController*, *Interactor* и *Router*, передавая данные и вызывая методы компонентов.

*Protocols* содержит в себе протоколы, определяющие интерфейс взаимодействия между различными частями *VIPER*-модуля.

Рассмотрим пример, при котором пользователь нажимает кнопку «Загрузить данные» на экране. *ViewController* сообщает *Presenter* о том, что кнопка была нажата, *Presenter* реагирует на это, уведомляя *Interactor* о необходимости обращения в соответствующий сервис для загрузки данных. После того, как сервис выполнил запрос и отдал данные, *Interactor* обрабатывает их, создавая *view*-модели, которые передаются в *Presenter*. *Presenter*, в свою очередь, транслирует данные в *ViewController*, где тот отображает их, например, в виде списка транзакций. Предположим, нажимая на выбранный элемент списка, пользователь желает перейти в экран деталей некой транзакции. *ViewController*, обработав нажатие, сообщает об этом в *Presenter*, где тот обращается к *Router*, чтобы получить новый *VIPER*-модуль для отображения.

Общая структура взаимодействия компонентов модуля представлена на рисунке 2.6.

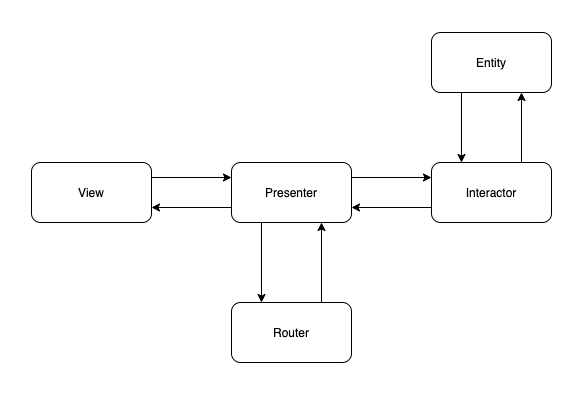


Рисунок 2.6 – Схема взаимодействия компонентов *VIPER*-модуля

## 2.4 Хранение данных в мобильном приложении

Классическим решением хранения данных в программных средствах, является использование систем управления базами данных.

Такой подход позволяет удобно хранить большие объемы структурированных данных, взаимодействовать с ними посредствам запросов.

Однако, в разработке мобильных приложений, данный подход требуется не всегда. В основном, приложения, которые взаимодействует с *API*, расположенном на удаленном сервере или при работе с децентрализованными системами, данные могут храниться в специальных хранилищах мобильной операционной системы, данные в которых имею вид «ключ – значение». Хранилища подобной структуры способствуют увеличению скорости извлечения и записи данных, что является неоспоримым плюсом для мобильных приложений, в которых важна скорость отклика на действия пользователя.

В мобильном приложении, разрабатываемом в рамках дипломного проекта, будет использоваться два вида описанных выше хранилищ – *UserDefaults*, данные в котором хранятся в незашифрованном виде в файле, привязанном к приложению, и хранилище *KeyChain*, данные в котором надежно зашифровываются операционной системой.

Описание данных, ключей, типов, хранящихся в каждом из представленных хранилищ представлены в таблице 2.1.

Таблица 2.1 – Сохраняемые данные мобильного приложения

|  |  |  |  |
| --- | --- | --- | --- |
| Сохраняемый объект | Наименование ключа | Тип данных значения | Хранилище |
| Хэш значение пин‑кода приложения | *kPinHash* | *String* | *KeyChain* |
| Значение выбранного таймаута автоблокировки приложения | *kAutoblockTime* | *String* | *UserDefauls* |
| Значение, определяющее, включен ли вход в приложение посредствам биометрических данных | *kisFingerprintOn* | *String* | *UserDefauls* |
| Публичные данные криптокошельков | *kWalletsPublicDataMap* | *Generic* | *UserDefauls* |
| Биометрические данные *TouchId* или *FaceId* для входа в приложения посредствам биометрии | *kFingerprint* | *String* | *KeyChain* |
| Мнемоник криптокошелька | Зашифрованный публичны адрес кошелька + *kseedKey* | *String* | *KeyChain* |

## 2.5 Выводы по разделу

Для реализации мобильного приложения будут использоваться следующие технологии:

* платформа *iOS*;
* в качестве основного языка программирования будет выступать язык *Swift*;
* дополнительный язык программирования, который в некоторых случаях может использоваться как вспомогательный, является *Objective-C*;
* в качестве архитектуры мобильного приложения будет использоваться *VIPER* паттерн, что позволит удобно организовать код приложения;
* форматом передаваемых данных по сети станет формат *JSON*.

В дополнение к вышеперечисленному, для разработки приложения были отобраны вспомогательные библиотеки, технологии, утилиты и подходы:

* стандартная библиотека *Foundation*, содержащая базовые типы данных и методы работы с ними*;*
* стандартная библиотека *UIKit*, предоставляющая базовый набор элементов пользовательского интерфейса*;*
* библиотека *Swinject* для организации инъекции зависимостей;
* библиотека взаимодействия с ресурсами приложения *R.swift*;
* библиотека *SnapKit*, упрощающая определение отношений между элементами пользовательского интерфейса;
* библиотека поддержки анимаций в формате *JSON Lottie*;
* библиотека генерации и считывания *QR*-кодов *QRCodeReader;*
* класс *ReachabilitySwift*, отслеживающий доступ устройства в сеть Интернет*;*
* библиотека *CryptoSwift*, предоставляющая набор криптографических алгоритмов;
* система контроля версий *GIT*.

Данный набор технологий в полной мере обеспечивает беспрепятственную реализацию мобильного приложения по предоставлению пользователю доступа к его криптоактивам.

# **3 Разработка мобильного приложения**

ФИО

Подпись

Дата

Лист

1

БГТУ 03.00.ПЗ

Разраб.

*Марковский А.Г.*

Пров.

Годун А.В.

Консульт.

Годун А.В.

Н. контр.

Рыжанкова А.С.

Утв.

Пацей Н.В.

3 Разработка мобильного приложения

Лит.

Листов

14

*74417049, 2021*

В разрабатываемом мобильном приложении можно выделить несколько основных категорий компонентов, которые взаимодействую между собой и выстраивают работоспособную систему. Такими являются следующие компоненты:

* *viper*-модули;
* сервисы;
* *core*-компоненты;
* *event*-прокси;

Каждый их перечисленных компонентов выполняет свою собственную задачу, которые будут описаны далее.

## 3.1 VIPER-модули

*VIPER*-модули представляют из себя целые экраны приложения или отдельные его части, если функционал какой-либо логической области экрана является слишком сложным, чтобы быть помещенным в один модуль.

Из немногих недостатков данного архитектурного паттерна можно выделить лишь большое количество файлов и базового кода в каждом из модулей. За время разработки программного средства был создан 51 *VIPER*-модуль, давая в сумме минимум 357 файлов, что довольно много для относительно небольшого проекта.

Структура и назначения составных частей модуля была описана в разделе 2.3 настоящей пояснительной записки.

Относясь к одному сценария использования, несколько *VIPER*-модулей могут логически объединяться, образуя так называемый *flow* мобильного приложения, между которыми может происходить переключение. Например, после *flow* авторизации следует «*Main» flow*, в котором происходит взаимодействие с элементами главного экрана. Изображение, демонстрирующее такой подход при разработке программного средства приведено на рисунке 3.1

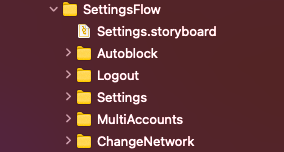


Рисунок 3.1 – Объединение модулей в *flow*

Пример нетривиального экрана, разбитого на несколько самостоятельных модулей продемонстрирован на рисунке 3.2.



Рисунок 3.2 – Пример экрана, состоящего из нескольких модулей

Правильное разбиение экранов на небольшие самостоятельные модули является основным принципом в разработке качественного приложения.

## 3.2 Сервисы мобильного приложения

Сервисы представляют собой сущности, выполняющие определенную область задач посредствам обращения к *core*-компонентам или *network*-адаптерам, в последующем обрабатывая полученные данные и производя манипуляции над ними. Пример, интерфейс сервиса авторизации для криптовалюты *Bitcoin* представлен в листинге 3.1, а также на диаграмме классов в приложении Б.

|  |
| --- |
| protocol BTCAuthService: BTCNetworkConfigurable {  func verifySeed(\_ seed: String) -> Bool  func obtainPrivateKey(seed: String) throws -> String  func createWallet(seed: String) throws -> BTCWallet  func getAllWallets() throws -> [BTCWallet]  func getCurrentWallet() throws -> BTCWallet  func isWalletExist(\_ wallet: BTCWallet) -> Bool  func saveWallet(\_ wallet: BTCWallet, makeCurrent: Bool) throws  func selectWallet(\_ wallet: BTCWallet) throws  func deleteWallet(\_ wallet: BTCWallet) throws  } |

Листинг 3.1 – Интерфейс *BTCAuth* сервиса

В основном, с сервисами взаимодействует *Interactor* какого-либо *VIPER‑*модуля приложения.

Рассмотрим подробнее имплементацию интерфейса *BTCAuthService* в листинге 3.2.

|  |
| --- |
| class BTCAuthServiceImp: BTCAuthService {  var btcCoreComponent: BTCCoreComponent!  var storageService: PublicDataService!  var currentNetwork: BTCNetworkType! |

Рисунок 3.2 – Зависимости *BTCAuthService*

Из листинга 3.2 следует, что сервис *BTCAuthService* держит зависимости на некоторые другие сущности, а именно на *Bitcoin* *core*-компонент, *storage* сервис, задача которого в сохранении публичной информации, и хранит свойство *currentNetwork*, определяющее тип текущей сети. Эти зависимости необходимы сервису для реализации отведенной ему логики. Листинг 3.3 показывает пример того, как сервис реализует собственную логику взаимодействуя с другими компонентами.

|  |
| --- |
| var hasWallets: Bool {  let wallets = try? loadWalletsForNetwork()  return !wallets.isEmpty  }    func getSeed() throws -> String {  do {  let seed = try btcCoreComponent.generateSeed()  return seed.joined(separator: " ")  } catch {  throw BTCWalletAuthError.invalidSeed  }  }  func verifySeed(\_ seed: String) -> Bool {  let seed = seed.components(separatedBy: " ")  let isValid = btcCoreComponent.verifySeed(seed)  return isValid  }  func obtainPrivateKey(seed: String) throws -> String {  let seed = seed.components(separatedBy: " ")  do {  let privateKey = try btcCoreComponent.getPrivateKey(seed: seed, networkType: currentNetwork)  return privateKey  } catch {  throw BTCWalletAuthError.invalidKey  }  } |

Листинг 3.3 – Взаимодействие сервиса с другими компонентами

### **3.2.1 Сервисы MAGWallet проекта-модуля**

В таблице 3.1 представлено перечисление сервисов *MAGWallet* проекта-модуля и краткое описание задач, которые они выполняют.

Таблица 3.1 – Сервисы *MAGWallet* проекта

|  |  |
| --- | --- |
| Название сервиса | Назначение сервиса |
| *SensitiveDataService* | Обеспечивает сохранение, удаление, получение и перешифровку приватных данных пользователя |
| *ReachabilityService* | Слушает текущее состояние системы по подключению к сети Интернет, генерирует глобальные уведомления при изменении состояния доступа |
| *AuthService* | Общий сервис аутентификации, сохраняющий, изменяющий и проверяющий пин-код при его вводе |

### **3.2.2 Сервисы BTCModule, ETHModule, LTCModule проектов-модулей**

Описание сервисов *BTCModule*, *ETHModule*, *LTCModule* проектов-модулей на примере *BTCModule* приведено в таблице 3.2.

Таблица 3.2 – Сервисы *BTCModule* проекта

|  |  |
| --- | --- |
| Название сервиса | Назначение сервиса |
| *BTCTransferService* | Содержит логику отправки транзакции и обработки результата, а также проверяет адреса на валидность |
| *BTCUpdateService* | Обновляет балансы криптокошельков *Bitcoin* криптовалюты |
| *BTCAuthService* | Сервис аутентификации *Bitcoin*. Выполняет операции с мнемоником кошелька. Создает, сохраняет и удаляет кошельки соответствующей валюты. Хранит в себе кошелек, выбранный основным |

### **3.2.3 Сервисы SharedFilesModule проекта-модуля**

*SharedFilesModule* проект-модуль имеет два сервиса, назначение которых описано в таблице 3.3.

Таблица 3.3 – Сервисы *SharedFilesModule* проекта

|  |  |
| --- | --- |
| Название сервиса | Назначение сервиса |
| *PublicDataService* | Сервис, задача которого заключается взаимодействии с хранилищем публичной информации для ее получения, сохранения или удаления |

Окончание таблицы 3.3

|  |  |
| --- | --- |
| *FingerprintAccessService* | Занимается взаимодействием с биометрическими возможностями устройства, проверкой доступности биометрического функционала |

## 3.3 Core-компоненты мобильного приложения

*Core*-компонентами являются небольшие сущности, в подавляющем большинстве классы, выполняющие узконаправленную задачу. В основном, *core*‑компоненты практически не имеют собственных зависимостей, а сами являются составными частями чего-либо, т.е. зависимостями, например, для сервисов приложения. Сервисы могут иметь достаточно много зависимостей на *core*-компоненты, но в этим и заключается их суть – они небольшие по объему, простые для понимания их задачи и легко тестируются. Пример интерфейса *core*-компонента отображен в листинге 3.4.

|  |
| --- |
| protocol BTCCoreComponent {  func generateSeed() throws -> [String]  func generateKey(seed: [String], networkType: BTCNetworkType) throws-> BTCKey  func verifySeed(\_ seed: [String]) -> Bool  func getAddress(seed: [String], networkType: BTCNetworkType) throws-> String  func getPrivateKey(seed: [String], networkType: BTCNetworkType) throws -> String    func calculateTransactionFee(  amount: Int,  feePerKb: Int,  unspentOutputs: [BTCTransactionOutput],  networkType: BTCNetworkType  ) throws -> Int    func createTransaction(  privateKey: BTCKey,  outputs: [BTCTransactionOutput],  changeAddress: BTCAddress,  toAddress: BTCAddress,  value: BTCAmount,  fee: BTCAmount,  checkSignature: Bool  ) throws -> BTCTransaction?    func getTransactionHex(transaction: BTCTransaction) -> String?  } |

Листинг 3.4 – Пример интерфейса *core*-компонента

*Core*-компоненты каждого из проектов-модулей будут рассмотрены далее.

### **3.3.1 Core-компоненты MAGWallet проекта-модуля**

Единственным *core*-компонентом модуля является *CryptoCoreComponent*, который выполняет криптографические функции – шифрование, дешифрование и вычисление хэша переданной строки. Программный интерфейс *CryptoCoreComponent* представлен в листинге 3.5.

|  |
| --- |
| protocol CryptoCoreComponent {  func hash(string: String) -> String  func encrypt(string: String, salt: String) throws -> String  func decrypt(hash: String, salt: String) throws -> String  } |

Листинг 3.5 – Пример интерфейса *core*-компонента

### **3.3.2 Core-компоненты BTCModule, ETHModule, LTCModule модулей**

Описание *core*-компонентов *BTCModule*, *ETHModule*, *LTCModule* проектов‑модулей на примере *BTCModule* приведено в таблице 3.4.

Таблица 3.4 – *Сore*-компоненты *BTCModule* проекта

|  |  |
| --- | --- |
| Название *core*-компонента | Назначение *core*-компонента |
| *BTCCoreComponent* | Занимается функционалом, связанным непосредственно с криптовалютой – генерация мнемоников, вывод приватных ключей, верификация мнемоника, получение адреса кошелька и некоторых других |
| *QRCodeCoreComponent* | Генерирует изображения *QR*-кода на основе входной строки требуемого размера |

### **3.3.3 Core-компонент ProtectedStorageCore проекта-модуля SharedFilesModule**

*ProtectedStorageCore*является компонентом по работе с данными вида «ключ‑значение», их сохранением, получением и удалением в зашифрованном системном хранилище *KeyChain*. Исходный код компонента приведен в листинге 3.6.

|  |
| --- |
| public final class ProtectedStorage: KeyValueStoring {  let keychain = Keychain(service: Bundle.main.bundleIdentifier!).accessibility(.whenUnlockedThisDeviceOnly)  public func set(key: String, value: Data) {  do {  try keychain.set(value, key: key)  } catch let error {  print("error: \(error)")  }  }  public func get(key: String) -> String? {  do { |

let value = try keychain.get(key)

return value

} catch let error {

print("error: \(error)")

return nil

}

}

public func remove(key: String) {

do {

try keychain.remove(key)

} catch let error {

print("error: \(error)")

}

}

}

Листинг 3.6 – Исходный код *core*-компонента *ProtectedStorageCore*

## 3.4 Event-прокси мобильного приложения

*Event*-прокси представляют из себя набор объединенных интерфейсов, имплементируя которые некоторая сущность, например *Interactor* *VIPER*-модуля, может выступать в роли издателя событий либо быть тем, кто реагирует на них. Пример программной реализации *event*-прокси отображен в листинге 3.7.

|  |
| --- |
| protocol ReachabilityEventDelegate: AnyObject {  func reachabilityChanged(newConnection: ReachabilityConnection)  }  protocol ReachabilityEventDelegateHandler: AnyObject {  var delegate: ReachabilityEventDelegate? { get set }  }  protocol ReachabilityEventActionHandler {  func actionReachabilityChanged(newConnection: ReachabilityConnection)  }  protocol ReachabilityEventProxy: ReachabilityEventActionHandler, ReachabilityEventDelegateHandler {  } |

Листинг 3.7 – Пример программной реализации *event*-прокси

Приведем пример на основе прослушки статуса доступа в сеть Интернет. Имеется сервис, задача которого – следить за изменением статуса доступа устройства в сеть Интернет исходный код которого приведен в листинге 3.8.

|  |
| --- |
| class ReachabilityServiceImp: ReachabilityService {  var reachability: Reachability!  var reachabilityActionHandler: ReachabilityEventActionHandler! |

var connection: ReachabilityConnection {

return mapReachabilityStatus(connection: reachability.connection)

}

func configure() {

reachability = try? Reachability()

reachability.whenReachable = { [weak self] reachability in

guard let self = self else {

return

}

let mappedConnection = self.mapReachabilityStatus(connection: reachability.connection)

self.reachabilityActionHandler.actionReachabilityChanged(newConnection: mappedConnection)

}

reachability.whenUnreachable = { [weak self] reachability in

guard let self = self else {

return

}

let mappedConnection = self.mapReachabilityStatus(connection: reachability.connection)

self.reachabilityActionHandler.actionReachabilityChanged(newConnection: mappedConnection)

}

try? reachability.startNotifier()

}

}

Листинг 3.8 – Исходный код сервиса по прослушке статуса доступа в Интернет

Также, имеется экран приложения, который при обрыве связи должен произвести определенные действия. В таком случае сервис выступает в роли *ActionHandler*, генерируя событие о том, что статус сети был изменен, заставляя всех делегатов отреагировать. Экраны приложения выступают в роли *EventDelegate* (листинг 3.9), которые подписываются на события *ActionHandler*, добавляются в его массив делегатов, что позволяет выстроить модель взаимодействия «один ко многим».

|  |
| --- |
| extension MainRoutingInteractor: ReachabilityEventDelegate {  func reachabilityChanged(newConnection: ReachabilityConnection) {  switch newConnection {  case .reachable:  updateRemoteNotifications()  print("Reachable connection")    case .unreachable:  print("Unreachable connection")  }  }  } |

Листинг 3.9 – Исходный код экран приложения, реагирующего на событие

### **3.4.1 Event-прокси MAGWallet проекта-модуля**

В таблице 3.5 приведены *event*-прокси сущности, разработанные в рамках *MAGWallet* проекта-модуля.

Таблица 3.5 – *Event*-прокси *MAGWallet* проекта

|  |  |
| --- | --- |
| Название *event*-прокси | Назначение *event*-прокси |
| *AuthEventProxy* | Генерирует события, когда пользователь совершает вход в кошелек или выходит из него |
| *ReachabilityEventProxy* | Следит за статусом доступа в Интернет устройства, генерируя события при его изменении |

### **3.4.2 Event-прокси BTCModule, ETHModule, LTCModule проектов**

Описание назначений *event*-прокси сущностей, разработанные в рамках *BTCModule*, *ETHModule* и *LTCModule* проектов-модулей описаны в таблице 3.6 на примере *BTCModule*.

Таблица 3.6 – *Event*-прокси *BTCModule* проекта

|  |  |
| --- | --- |
| Название *event*-прокси | Назначение *event*-прокси |
| *BTCUpdateEventProxy* | Генерирует и реагирует на события изменения баланса *Bitcoin* кошелька |
| *AuthEventProxy* | Создает событие, когда в приложение добавляется новый *Bitcoin* кошелек |

### **3.4.3 Event-прокси SharedFilesModule проекта-модуля**

В таблице 3.7 приведены *event*-прокси компоненты, разработанные в рамках *SharedFilesModule*проекта-модуля. Данные *event*-прокси будут использоваться повсеместно в разработке мобильного приложения.

Таблица 3.7 – *Event*-прокси *SharedFilesModule*проекта

|  |  |
| --- | --- |
| Название *event*-прокси | Назначение *event*-прокси |
| *SensitiveDataEventProxy* | Генерирует события, когда пользователь хочет произвести действия над приватными данными. Например, при попытке отправки транзакции показывается экран ввода пин-кода |
| *SnackBarsEventProxy* | Создает события, связанные с показом *popup*‑уведомлений с определенным содержимым пользователю |
| *SessionExpireEventProxy* | *Event*-прокси для реализации автоблокировки приложения при определенном времени бездействия пользователя |

## 3.5 Хранение данных в мобильном приложении

Как описывалось в разделе 2.4 данной пояснительной записки, данные в разрабатываемом мобильном приложении сохранятся в виде «ключ – значение» в двух типах хранилищ – зашифрованном и незашифрованном. Пример реализации взаимодействия с хранилищем при получении данных о сохраненном значении времени автоблокировки приложения приведен в листинге 3.10. Здесь *AutoblockInteractor* включает в себе зависимость на *sharedStorage*, которая представляет из себя *core*-компонент по работе с хранилищем. Далее, для того чтобы извлечь данные, необходимо вызвать метод «*get» core*-компонента и передать в качестве входного параметра соответствующий ключ, вынесенный в константные значения приложения.

|  |
| --- |
| class AutoblockInteractor {  weak var output: AutoblockInteractorOutput!  var sharedStorage: StorageCore!  var preventExpireActionHandler: SessionExpirePreventActionHandler!    func getCurrenctAutoblockTime() -> BlockTime {  if let currentTimeString = sharedStorage.get(key: SharedFilesModule.Constants.SharedKeys.autoblockTime),  let state = BlockTime(rawValue: currentTimeString) {  return state  } else {  return SharedFilesModule.Constants.Settings.defautTimeoutForSessionExpiration  }  } |

Листинг 3.10 – Получение данных из хранилища

Сами классы хранилища реализуют интерфейс *StorageCore*, который определяет методы для получения, сохранения и удаления данных их хранилища. Интерфейс *StorageCore* приведен в листинке 3.11.

|  |
| --- |
| public protocol StorageCore {  var storage: KeyValueStoring { get set }    func get<T>(key: String, type: T.Type) -> T? where T: Decodable  func get(key: String) -> String?  func set<T: Codable>(key: String, value: T) throws  func remove(key: String)  } |

Листинг 3.11 – Интерфейс взаимодействия с хранилищем *StorageCore*

## 3.6 Функция автоблокировки мобильного приложения

Данная функция предоставляет возможность выбрать один из вариантов

времени автоблокировки приложения при бездействии пользователя или выбрать опцию, которая позволит не блокировать мобильное приложение вовсе. По истечению выбранного значения времени приложение отобразит экран ввода пин‑кода, после введения которого пользователь может продолжить взаимодействие с приложением с того места, на котором оно было заблокировано.

Реализация описанной функции заключается в наложении невидимого слоя поверх всех остальных экранов мобильного приложения. Чтобы реализовать счет времени в мобильном приложении присутствует глобальный таймер, который сбрасывается при любом касании экрана пользователем. При истечении таймера, отведенный под эту задачу *event*‑прокси генерирует событие, на которое реагирует невидимый слой, отображая там экран ввода пин‑кода и убирает его только в том случае, если пользователь ввел верный, установленный при первоначальной настройке, пароль. Исходный код глобального таймера представлен в листинге 3.12.

|  |
| --- |
| final class RepeatingTimer {  private lazy var timer: DispatchSourceTimer = {  let aTimer = DispatchSource.makeTimerSource()  aTimer.schedule(deadline: .now() + expireTime)  aTimer.setEventHandler { [weak self] in  self?.eventHandler?()  }  return aTimer  }()    var eventHandler: (() -> Void)?  var expireTime: Double = 10    private enum State {  case suspended  case resumed  }  } |

Листинг 3.12 – Глобальный таймер автоблокировки приложения

Исходный код *event*‑прокси, генерирующего событие, на которое реагирует невидимый слой приведен в листинге 3.13. Блок‑схема процесса автоблокировки представлена в приложении В.

|  |
| --- |
| public protocol SessionTimeoutDelegate: class {  func sessionActivityTimeout()  }  public protocol SessionTimeoutDelegateHandler: class {  var timeoutDelegate: SessionTimeoutDelegate? { get set }  }  public protocol SessionExpirePreventActionHandler {  func startMonitoring()  func stopMonitoring()  func expirationTimeDidChange() |

func setLocalExpireTime(time: BlockTime)

func removeLocalExpireTime()

func isSessionExpire() -> Bool

}

public protocol SessionExpireEventProxy: SessionExpirePreventActionHandler,

SessionTimeoutDelegateHandler {}

Листинг 3.13 – *Event*‑прокси автоблокировки приложения

## 3.7 Реализация мультиаккаунтности в мобильном приложении

Исходя из проведенного анализа аналогов был выделен один из основных их недостатков – отсутствие возможности добавления нескольких кошельков единой криптовалюты. Реализация данного функционала представляет собой сохранение всех кошельков в хранилище в виде массива, а также его извлечение и последующую сортировку по типу выбранной сети. Исходный код методов сохранения и извлечения криптокошельков представлен в листинге 3.14.

|  |
| --- |
| private func saveWallets(\_ wallets: [BTCWallet]) throws {  try storageService.setPublicData(key: Constants.AuthConstants.savedWalletsKey, data: wallets)  }  private func loadWalletsForNetwork() throws -> [BTCWallet] {  do {  let wallets = try loadWallets()  return wallets.filter { $0.network == currentNetwork }  } catch let error {  throw error  }  }  private func loadWallets() throws -> [BTCWallet] {  guard let wallets = try? storageService.obtainPublicData(key: Constants.AuthConstants.savedWalletsKey,  type: [BTCWallet].self)  return wallets  } |

Листинг 3.14 – Исходный код методов сохранения и извлечения криптокошельков

## 3.8 Реализация валидации суммы транзакции

При отправке транзакции пользователю необходимо ввести сумму операции.

После того как введено значение, его необходимо проверить, чтобы оно не было больше суммы транзакции и значения комиссии за проведение транзакции. В случае, если баланса кошелька недостаточно, пользователю должна быть отображена ошибка, говоря о том, что введенное значение неверно. Для реализации данного алгоритма был разработан метод, результатом работы которого является логическое значение, символизирующее о том, хватает ли средств для проведения операции.

Исходный код разработанного метода приведен в листинге 3.15.

|  |
| --- |
| private func isEnoughAmount(address: String, amount: String, fee: String) throws -> Bool {  guard let amountNumber = BTCAmount(amount), let feeNumber = BTCAmount(fee) else {  return false  }  var isEnough = false    networkAdapter.balance(address: address) { [weak self] result in  switch result {  case .success(let balance):  if let balanceNumber = BTCAmount(balance.balance) {  isEnough = balanceNumber >= amountNumber + feeNumber  }    case .failure(let error):  print(error)  }  self?.semaphore.signal()  }  semaphore.wait()    return isEnough  } |

Листинг 3.15 – Исходный код метода валидации суммы транзакции

Для вычисления необходимой суммы баланса для проведения транзакции, необходимо вычислить значение комиссии. Исходный код получения значения комиссии за килобайт данных представлен в листинге 3.16.

|  |
| --- |
| private func getFeePerKb() throws -> Int {  var feePerKbResponse: String?  networkAdapter.getFeePerKb { [weak self] result in  switch result {  case .success(let fee):  feePerKbResponse = fee    case .failure(let error):  print(error)  }  }  let feeString = feePerKbResponse  let value = BigDecimalNumber(feeString)  let decimal = BigDecimalNumber(Constants.BTCConstants.BTCDecimalString)  return = Int(value.powerOfTen(decimal)) } |

Листинг 3.16 – Исходный код получения значения комиссии за килобайт данных

## 3.9 Выводы по разделу

В данном разделе была описана программная реализация всех ключевых сущностей и подходов, использованных в рамках этапа разработки, которые необходимы для достижения целей, перечисленных в разделе постановки задачи.

Описание *VIPER*-модулей было, а также логика построения экранов на их основе, объединение в f*low* представлено в разделе «*VIPER*-модули».

Созданные на этапе разработки сервисы, их задачи, способы взаимодействия и логика зависимостей подробно описана в разделе «Сервисы».

Разработанные *core*-компоненты приложения подробно описаны в разделе «*Сore*‑компоненты».

Паттерн *event*-прокси, логика его работы, а также список назначений каждого из них описаны в разделе «*Event*-прокси». Были представлены и описаны участки кода в виде листингов, раскрывающие логику отдельных компонентов приложения, а также способ взаимодействия компонентов между собой.

Также, в разделе была описана реализация некоторых возможностей приложения, а именно:

* хранение данных;
* автоблокировка приложения;
* мультиаккаунтность.

Данные возможности были отмечены как задачи, требующие решения и реализации.

Итогом выполнения разработки дипломного проекта стало программное средство, представляющее из себя мобильное приложений для платформы *iOS*, цель которого – предоставление доступа и функций управления криптоактивами пользователя.

# **4 Тестирование мобильного приложения**

ФИО

Подпись

Дата

Лист

1

БГТУ 04.00.ПЗ

Разраб.

*Марковский А.Г.*

Пров.

Годун А.В.

Консульт.

Годун А.В.

Н. контр.

Рыжанкова А.С.

Утв.

Пацей Н.В.

4 Тестирование мобильного приложения

Лит.

Листов

7

*74417049, 2021*

Тестирование – одна из самых значимых частей разработки программного средства. Статистика в сфере *IT* показывает, что при создании программного обеспечения все больше времени отводится тестированию. Именно в процессе тестирования находят подавляющее большинство проблем в работе системы, к которым пользователи с течением времени становятся все менее лояльными.

Перед командой тестирования становится важная задача – необходимо протестировать максимальное число вариантов использования программного средства пользователем, чтобы исключить возможные ошибки в процессе работы приложения.

Среди использованных подходов к тестированию можно отметить *unit* и *manual* тестирование, назначения которых будут описаны в данном разделе.

## 4.1 Unit тестирование

*Unit* тестирование, или как оно еще называется, модульное тестирование, направлено на проверку отдельных модулей исходного кода, которое позволяет проверять не привело ли какое-либо изменение кода к появлению ошибок. В случае появления ошибки, такой вид тестирования позволяет достаточно быстро их обнаружить и устранить, поскольку затрагивает только один модуль.

Для тестирования мобильных приложений *Apple* предоставляет мощные стандартные средства – утилиту тестирования в *IDE «Xcode»* и библиотеку «*XCTest»*, включенную в стандарт языка программирования *Swift*, отвечающие всем современным требованиям написания *unit*-тестов.

В данном проекте тестами были покрыты подавляющее большинство сервисов и *core*-компонентов приложения, задействованных в основной логике приложения.

Всего был разработано 38 unit-тестов проверки логики приложения, разработанной в течение дипломного проекта. Описание основных тестовых методов и модулей, которые они тестируют, представлено в таблице 4.1 на примере *Bitcoin* подмодуля.

Таблица 4.1 – Описание юнит-тестов

|  |  |  |
| --- | --- | --- |
| Модуль | Название теста | Описание |
| *Bitcoin* сервис аутентификации и авторизации | *testCreateWalletTestnet* | Проверка генерации адреса кошелька из мнемоника для *testnet* сети |

Окончание таблицы 4.1.

|  |  |  |
| --- | --- | --- |
| Модуль | Название теста | Описание |
| *Bitcoin* *сore*-компонент | *testGetPrivateKey* | Проверка извлечения приватного ключа кошелька из мнемоника |
| *Bitcoin* *network*-адаптер | *testGetTestnetBalance* | Проверка возвращения баланса кошелька |
| *Bitcoin* cервис транзакций | *testVerifyAddress* | Проверка на валидность адреса получателя, указанного в транзакции |
| *Bitcoin* сервис обновления данных | *testObtainTestnetWalletHistory* | Проверка получения истории транзакций кошелька |
| *QR-code* *core*-компонент | *testGenerateLargeImage* | Проверка генерации *QR*-кода на основе переданной строки |
| Сервис данных | *testStorePin* | Проверка сохранения и извеления пин-кода |
| Сервис криптографии | *testEncryptPasswordAndDecryptWithSameSalt* | Проверка шифрования и дешифрования пин-кода |
| Сервис приватных данных | *testGetSensativeDataWrongPass* | Проверка сохранения данных с верным паролем и попытка получить их с неверным |

Исходный код теста метода *getPrivateKey* модуля *BTСoreComponent* представлен в листинге 4.1.

|  |
| --- |
| func testGetPrivateKey() {  //arrange  let seed = ["exhibit", "assume", "foil", "setup", "annual", "token", "roof", "spray", "fuel", "laugh", "typical", "leave"]  let expectedPrivateKey = "417a8fa6c6e54e31a5fe79644189a95310ead631b10c7b805d54ea05a6211ee8"    do {  //act  let privateKey = try btcCore.getPrivateKey(seed: seed, networkType: .mainnet)  //assert  XCTAssertEqual(privateKey, expectedPrivateKey)  } catch {  //assert  XCTFail()  }  } |

Листинг 4.1 – Исходный код теста метода *getPrivateKey*

Также был протестирован метод смены пин‑кода *Auth* сервиса. Исходный код теста представлен в листинге 4.2.

|  |
| --- |
| func testChangeVerifyPin() {  //arrange  let pin = "111111"  let newPin = "222222"    //act  try? facade.storePin(pin: pin)  try? facade.changePin(newPin: newPin, oldPin: pin)  let result = facade.verify(pin: newPin)    //assert  XCTAssertTrue(result)  } |

Листинг 4.2 – Исходный код *Unit*-теста смены пин‑кода

Одной из самых важных функций мобильного приложения является сохранение конфиденциальных данных в надежном виде. Данная логика реализована в *SensativeDataService*. Для того, чтобы быть уверенным в том, что методы данного сервиса работают корректно, к ним также были написаны Unit‑тесты. Исходный код теста, сценарий которого заключается в попытке получения конфиденциальных данных с неверным паролем приведен в листинге 4.3.

|  |
| --- |
| func testGetSensativeDataWrongPass() {  //arrange  let value = "someValue"  let key = "someKey"  let pass = "somePassword"  let wrongPass = "wrongPass"    //act  do {  try service.setSensitiveData(pass: pass, key: key, data: value)  let result = try service.obtainSensitiveData(pass: wrongPass, key: key)  //assert  let result = try service.obtainSensitiveData(pass: wrongPass)  //assert  XCTAssertNotEqual(result, value)  } catch let error {  //assert  XCTFail(error.localizedDescription)  }  } |

Листинг 4.3 – Исходный код теста с получением конфиденциальных данных

Таким образом, в результате тестирования, были протестированы все значимые сервисы и *core*-компоненты приложения. Исход выполнения тестов продемонстрирован на рисунке 4.1.

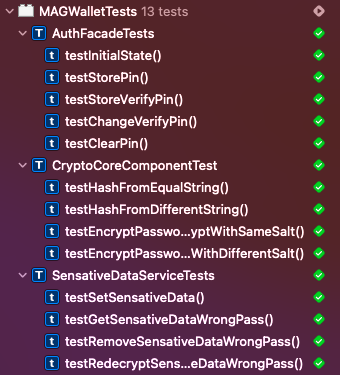


Рисунок 4.1 – Результат выполнения тестов

## 4.2 Ручное тестирование

Любое новое приложение должно быть проверено вручную, прежде чем его можно будет автоматизировать. Ручное тестирование требует больше усилий, но необходимо для проверки возможности автоматизации.При помощи ручного тестирования были проверены основные и наиболее используемые сценарии, а именно, сценарий входа в приложение, импорта криптовалютного кошелька и отправки транзакции.

### **4.2.1 Первоначальная настройка мобильного приложения**

При первом запуске приложению пользователю необходимо придумать и ввести пин-код, а затем подтвердить его повторным входом. Если повторный, подтверждающий, ввод будет содержать ошибку, система покажет это и потребует ввести его заново, как продемонстрировано на рисунке 4.2. Данный пин-код будет использоваться мобильным приложением в случаях, когда необходимо войти в приложение после его закрытия либо в сценарии, когда пользователь совершает операции, связанные с приватными данными, как, например, отправка транзакции.



Рисунок 4.2 – Ошибка подтверждающего ввода пин-кода

### **4.2.2 Импорт криптовалютного кошелька**

Импортируя кошелек в приложение, пользователю предоставляется поле для ввода мнемоника – набора слов, который идентифицирует кошелек. Слова, которые доступны для ввода, предопределены в количестве 256. В случае, если пользователь вводит менее требуемых 12 слов или вводит слово, не в ходящее в набор, то система презентует ошибку, сообщающую о том, что введенный мнемоник является не валидным. Снимок экрана с отображением ошибки представлен на рисунке 4.3.



Рисунок 4.3 – Ошибка ввода мнемоника при импорте кошелька

### **4.2.3 Формирование транзакции**

Отправляя транзакцию, заполняются следующие необходимые поля:

* адрес получателя;
* сумма транзакции в криптовалюте либо в USD валюте, которая будет сконвертирована в значение токена по актуальному курсу.

Каждое из вышеперечисленных полей имеет собственную валидацию, которая не позволит ввести некорректные значения и произвести транзакцию.

При неверном вводе адреса получателя, пользователь получит уведомление об ошибке, изображенное на рисунке 4.4.



Рисунок 4.4 – Ошибка ввода адреса получателя

Сумма транзакции, указываемая пользователем, определенно, не может быть свыше количества валюты, имеющейся на балансе криптокошелька. Указывая превышающую сумму транзакции, отображается валидационная ошибка, сообщающая об этом (рисунок 4.5).



Рисунок 4.5 – Ошибка ввода суммы транзакции

### **4.2.4 Блокировка продвижения по сценарию использования**

Ожидая определенных действий, таких как, например, ввод данных в поля формирования транзакции, система препятствует дальнейшее продвижение пользователя по сценарию использования приложения, деактивирует кнопки и выделяя их по средства *UX* методик – делая визуально нейтральными. Пример недоступной к нажатию кнопки изображен на рисунке 4.6.



Рисунок 4.6 – Блокировка кнопки системой

## 4.3 Выводы по разделу

В данном разделе предоставлены результаты тестирования мобильного приложения, разработанного для предоставления пользователям контроля и управления их криптоактивами.

На этапе тестирования использовалось два подхода к тестированию программного средства, а именно, модульное и ручное. Объединение данных видов тестирования гарантируют стабильно надежную работу приложения на всех этапах жизненного цикла – как при основной разработке, так и при поддержке, когда новый вводимый в приложение функционал не должен препятствовать работе старого. Приложение допускается к публикации и использованию, только в том случае, если все тестовые сценарии важного по приоритету функционала пройдены успешно.

Разработанные тесты сервисов и *core*-компонентов покрывают весь необходимы базовый функционал приложения.

В проект была добавлена обработка данных полей ввода, с целью полностью исключить некорректные значения и, следовательно, избежать ошибок работы системы. Также, приложение блокирует кнопки, что препятствует пользователю продвигаться по сценарию использования до момента ввода валидных значений.

Полученные результаты при модульном и ручном тестировании показали, что программное средство имеет высокий уровень отказоустойчивости и работает корректно.

# **5 Руководство пользователя**

ФИО

Подпись

Дата

Лист

1

БГТУ 05.00.ПЗ

Разраб.

*Марковский А.Г.*

Пров.

Годун А.В.

Консульт.

Годун А.В.

Н. контр.

Рыжанкова А.С.

Утв.

Пацей Н.В.

5 Руководство пользователя

Лит.

Листов

10

*74417049, 2021*

Каждое разработанное приложение должно иметь инструкцию пользователя, для того чтобы клиент приложения мог начать работу с приложением сразу без каких-либо сложностей. Инструкция должна быть написана простым языком в сопровождении графических материалов, которые будут показывать, что именно должно происходить, выполнив то или иное действие.

5.1 Первоначальная настройка мобильного приложения. Добавление кошелька

При первом входе в приложение перед пользователем становится задача установки и подтверждение шестизначного пин-кода приложения, который будет использоваться в дальнейшем для входа и ввод которого потребуется для подтверждения значимых действий в приложении. Снимок экрана с первоначальной настройкой приложения представлен на рисунке 5.1.



Рисунок 5.1 – Первоначальная настройка приложения

Далее, пользователь переходит на главный экран приложения, на котором отображены токены, кошельки которых могут быть созданы или импортированы путем нажатия кнопки «*Add Account*» интересующей криптовалюты.

В верхнем правом углу экрана располагается кнопка, ведущая в настройки приложения. Внешний вид главного экрана приложения изображен на рисунке 5.2.



Рисунок 5.2 – Главный экран приложения

Нажимая на кнопку «*Add account*» пользователь перенаправляется на экран, изображенный на рисунке 5.3.



Рисунок 5.3 – Выбор способа добавления кошелька

Данный экран предоставляет возможность выбора между функцией импорта и создания криптокошелька с нуля. Выбирая «*Import Brainkey*» пользователь переходит на экран ввода мнемоника – набора двенадцати слов, который идентифицирует конкретный кошелек. Снимок экрана импорта представлен на рисунке 5.4.



Рисунок 5.4 – Экран ввода мнемоника при функции импорта

В случает, если на экране выбора функции добавления кошелька пользователь выбирает «*Create New Address*», то появляется экран, на котором отображается сгенерированный случайным образом мнемоник. Пользователь имеет возможность скопировать данный набор слов в буфер обмена устройства, чтобы, в дальнейшем, сохранить его в надежном для себя месте и продолжить добавление (рисунок 5.5). Блок‑схема процесса создания кошелька представлена в приложении Г.



Рисунок 5.5 – Экран генерации мнемоника при функции создания

5.2 Кошелек. Список и отправка транзакций

После завершения процесса добавления кошелька, данный кошелек будет отображен на главном экране приложения.

Выбирая определенный добавленный кошелек, производится переход в детали кошелька, где отображаются следующие данные и функциональные кнопки:

* текущий баланс и эквивалент в *USD* валюте;
* адрес текущего кошелька и кнопка копирования адреса в буфер обмена устройства;
* история транзакций в виде списка с возможностью перехода к деталям каждой транзакции;
* кнопка настроек данного кошелька;
* кнопка, переводящая к сканированию *QR*-кода;
* кнопка «*Request*», нажатие на которую отображает на экране *QR*-код текущего кошелька, который может быть сканирован другими устройствами;
* кнопка «*Send*» перенаправляет пользователя на экран создания транзакции.

Снимок экрана деталей кошелька представлен на рисунке 5.6, а также в приложении Д.



Рисунок 5.6 – Детали кошелька

Детали выбранной транзакции представляют собой экран, на котором отображается расширенная информация о транзакции, включающая в себя сумму операции и взысканную комиссию, ее дату и время, а также некоторую техническую информацию – статус транзакции, блок, в котором она расположена в сети блокчейна, адреса с которого и на который была произведена операция. Внешний вид одной из транзакций изображен на рисунке 5.7.



Рисунок 5.7 – Детали транзакции

Возвращаясь к экрану деталей, у пользователя есть возможность отправки транзакций со своего кошелька на любой иной при помощи простого экрана с полями ввода необходимой для совершения операции информации. Пользователю необходимо указать адрес, на который он желает произвести перевод либо сканировать адрес в виде *QR*-кода при помощи кнопки сканирования в верхнем правом углу экрана. Следующим полем является сумма транзакции, которая может быть указана либо в валюте токена, либо в валюте *USD*. Имеются вспомогательные кнопки «*Send All*», при нажатии на которые в поле суммы транзакции будет введено значение баланса кошелька, с которого производится операция. Последнее, нижнее поле является информативным и отображает сумму комиссии за транзакцию, которая будет списана со счета отправителя. Данный экран представлен на рисунке 5.8. Блок‑схема процесса создания транзакции представлена в приложении Е.



Рисунок 5.8 – Экран формирования и отправки транзакции

Следующим экраном, заслуживающим рассмотрения, является экран настроек.

5.3 Настройки мобильного приложения

На экране настроек располагаются настройки как самого приложения, так и настройки кошельков. Внешний вид экрана настроек приложения представлен на рисунке 5.9. Из доступных, имеется функция смены пин-кода приложения, изменение времени автоблокировки приложения при бездействии, возможность изменять рабочую сеть для каждой из поддерживаемых криптовалют, выход из текущего аккаунта.



Рисунок 5.9 – Экран настроек приложения

### **5.3.1 Экран настроек Accounts**

Нажатие на пункт настроек «*Accounts*» ведет пользователя на экран управления всеми добавленными кошельками, где он может выбрать тот, что будет использоваться по умолчанию, удалить некоторые из них по желанию или добавить новые адреса, если в этом есть необходимость (рисунок 5.10).



Рисунок 5.10 – Экран настроек «*Accounts*»

### **5.3.2 Экран настроек Autoblock**

Данный экран настроек предоставляет возможность выбрать один из вариантов времени автоблокировки приложения при бездействии пользователя или выбрать опцию, которая позволит не блокировать мобильное приложение вовсе. По истечению выбранного значения времени приложение отобразит экран ввода пин‑кода, после введения которого, пользователь может продолжить взаимодействие с приложением с того места, на котором оно было заблокировано. Экран выбора времени таймаута мобильного приложения изображен на рисунке 5.11.

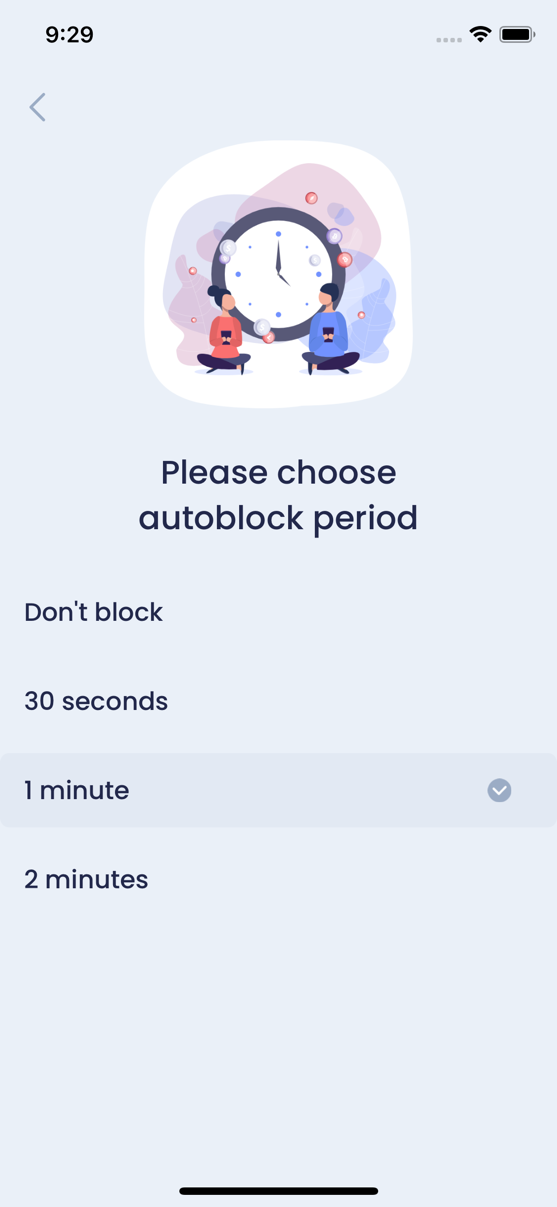


Рисунок 5.11 – Экран настроек автоблокировки мобильного приложения

### **5.3.3 Экран настроек Change Network**

Экран настроек «*Change Network*» позволяет пользователю выбрать рабочую сеть каждой из поддерживаемых криптовалют. Данная настройка наделяет пользователя возможностью проводить операции в тестовой сети, которая не привязана к реальным активам, что позволяет изучить мобильное приложение и протестировать его перед использованием с действительными криптокошельками на которых хранятся средства.

5.4 Дополнительные возможности мобильного приложения

Последний экран, который стоит рассмотреть, это экран сканирования *QR‑*кодов, который может стать крайне удобной возможностью в ситуациях, когда необходимо быстро ввести адрес кошелька, не набирая текст вручную. При первом запуске приложения у пользователя запрашивается разрешение на использование камеры устройства, на котором установлено приложение. Далее, пользователь может воспользоваться функцией сканирования. Изображение экрана сканирования *QR*-кодов представлено на рисунке 5.12.



Рисунок 5.12 – Экран сканирования *QR-*кодов

В случае, если пользователь запретил использование камеры устройства, каждый раз при переходе на экран сканирования он будет получать уведомление о том, что для совершения дальнейших действий нужно соответствующее разрешение. Навигационные кнопки внизу экрана предлагают либо вернуться на предыдущий экран, либо перейти в настройки устройства для выдачи разрешения. Уведомление об отсутствии разрешение изображено на рисунке 5.13.



Рисунок 5.13 – Уведомление об отсутствии разрешения

5.5 Выводы по разделу

В данном разделе была представлена инструкция для пользователя по работе с мобильным приложение. Она предоставляет собой характеристику основных возможностей, которые пользователь может совершить в рамках взаимодействия с приложением, а также описание последовательности действий при желании совершения тех или иных действий.

Пользователю доступна возможность простого создания или импорта криптокошельков, отслеживание их истории транзакций, а также пересылки средств на другие криптокошельки.

Исходя из простоты описания раздела можно сделать вывод о том, что, интерфейс приложения является удобным и интуитивно понятным для конечного пользователя, а значит велика вероятность того, что клиент не примет решения о смене системы для хранения собственных криптоактивов. В разработанном мобильном приложении предусмотрены различные уведомления об ошибках и критических ситуаций, а также информативные всплывающие окна в пользу улучшения обратной связи и понимания происходящего на экране.

# **6 Экономическое обоснование цены мобильного приложения**

ФИО

Подпись

Дата

Лист

1

БГТУ 06.00.ПЗ

Разраб.

*Марковский А.Г.*

Пров.

Годун А.В.

Консульт.

Годун А.В.

Н. контр.

Рыжанкова А.С.

Утв.

Пацей Н.В.

6 Экономическое обоснование цены мобильного приложения

Лит.

Листов

8

*74417049, 2021*

## 6.1 Общая характеристика разрабатываемого мобильного приложения

Во время разработки дипломного проекта использовалась мобильная платформа *iOS*. Разработанное программное средство представляет собой криптовалютный кошелек для доступа пользователя к его криптоактивам и взаимодействия с ними. Приложение будет распростроняться на платной основе в магазине приложений *App Store*.

Данный раздел служит для определения затрат, произведенных на всех стадиях разработки программного средства.

## 6.2 Исходные данные и маркетинговый анализ

Исходные данные для расчета приведены в таблице 6.1.

Таблица 6.1 – Исходные данные для расчета

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателя | Единица измерения | Условные обозначения | Норматив |
| Численность разработчиков | чел. |  | 1 |
| Норматив дополнительной заработной платы | % | Ндз | 15 |
| Ставка отчислений в Фонд социальной защиты населения | % | Нфсзн | 34 |
| Ставка отчислений в БРУСП «Белгосстрах» | % | Нбгс | 0,4 |
| Цена одного машино-часа | руб. | Смч | 0,06 |
| Норматив прочих затрат | % | Нпз | 29,5 |
| Норматив накладных расходов | % | Нобп*,* обх | 198 |
| Норматив расходов на сопровождение и адаптацию | % | Нрса | 17 |
| Ставка НДС | % | Нндс | 20 |

В ходе проведения маркетингового анализа, была выявлена стоимость разработки программного продукта криптовалютного кошелька на основе аналогов, приведенных в таблице 6.2. Данный анализ предоставит понимание того, насколько рентабельной будет являться разработка мобильного приложения, представленного в рамках дипломного проекта.

Таблица 6.2 – Стоимость разработки аналогичных программных средств

|  |  |  |
| --- | --- | --- |
| Наименование приложения | Описание приложения | Стоимость разработки, руб. |
| Мобильное приложение «*Coins*» | Криптовалютный кошелек | 40 000 |
| Мобильное приложение «*TrustWallet*» | Криптовалютный кошелек, хранилище ключей и биржа | 50 000 |

Общая цена разработки аналогичного продукта составляет 40 000 рублей. Также стоит учитывать аккаунт *Apple* разработчика, которая отдельно стоит 250 рублей. Таким образом, общая стоимость разработки данного программного средства, выбранного в качестве базы сравнения составит 40 250 рублей.

## 6.3 Методика обоснования цены

В современных рыночных экономических условиях программное средство (ПС) выступает преимущественно в виде продукции организаций, представляющей собой функционально завершенные и имеющие товарный вид ПС, реализуемые покупателям по рыночным отпускным ценам. Все завершенные разработки ПС являются научно-технической продукцией.

Широкое применение вычислительных технологий требует постоянного обновления и совершенствования ПС. Выбор эффективных проектов ПС связан с их экономической оценкой и расчетом экономического эффекта, который может определяться как у разработчика, так и у пользователя.

У разработчика экономический эффект выступает в виде чистой прибыли от реализации ПС, остающейся в распоряжении организации, а у пользователя – в виде экономии трудовых, материальных и финансовых ресурсов, получаемой за счет:

* снижения трудоемкости расчетов и алгоритмизации программирования и отладки программ;
* сокращения расходов на оплату машинного времени и других ресурсов на отладку программ;
* снижения расходов на материалы;
* ускорение ввода в эксплуатацию новых систем;
* улучшения показателей основной деятельности в результате использования ПС.

Стоимостная оценка ПС у разработчиков предполагает определение затрат, что включает следующие статьи:

* заработная плата исполнителей – основная и дополнительная;
* отчисления в фонд социальной защиты населения;
* отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний;
* расходы на оплату машинного времени;
* прочие прямые затраты.

На основании затрат рассчитывается себестоимость и отпускная цена ПС.

### **6.3.1 Объем мобильного приложения**

Для оценки объема программного средства, все его функции классифицируются с использованием специального каталога функций, который определяет их объем. Общий объем программного средства *V*o, вычисляется как сумма объемов *V*i каждой из *n* его функций (формула 6.1).

|  |  |  |
| --- | --- | --- |
|  |  | (6.1) |

В таблице 6.3 представлены функции, присутствующие в рассматриваемом программном средстве и соответствующий им объем в условных машино-командах.

Таблица 6.3 – Содержание и объем функций в программном средстве

|  |  |  |
| --- | --- | --- |
| Номер функции | Содержание функции | Объем, условных  машино-команд |
| 101 | Организация ввода информации | 700 |
| 102 | Контроль, предварительная обработка и ввод информации | 450 |
| 111 | Управление вводом/выводом | 1200 |
| 202 | Взаимодействие между компонентами системы | 2400 |
| 401 | Взаимодействие с базой данных | 150 |
| 402 | Вспомогательные методы | 330 |
| 506 | Обработка ошибочных и сбойных ситуаций | 960 |
| 707 | Графический вывод результатов | 2300 |
|  | Итого: | 8490 |

Опираясь на данные таблицы 6.3, можно определить объем программного средства, разработанного в ходе дипломного проектирования:

*V*o = 700 + 450 + 1200 + 2400 + 150 + 330 + 960 + 2300 = 8490 (условных машино-команд).

Уточненный объем программного средства *V*o*/* равен произведению объема программного средства *V*o на коэффициент изменения скорости обработки информации Кск (формула 6.2).

|  |  |  |
| --- | --- | --- |
|  | *V*o*/ = V*o*·* Кск. | (6.2) |

Исходя из вычисленного объема программного средства, можно определить уточненный объем программного средства:

*V*o*/* = 8490 ⋅ 0,6 = 5094 (условных машино-команд).

### **6.3.2 Основная заработная плата**

Для определения величины основной заработной платы, было проведено исследование величин заработных плат для специалистов в сфере мобильной разработки на *iOS*. Источником данных служили открытые веб-порталы, различные форумы, официальная отчетность, а также общий средний уровень заработка в сфере информационных технологий. Итогом изучения и анализы полученных данных, стала информация о том, что средняя месячная заработная плата для позиций *junior*/*middle* составляет 1500 рублей.

Проект разрабатывался одним человеком на протяжении трех месяцев. Таким образом, основная заработная плата будет рассчитываться по формуле (6.3):

|  |  |
| --- | --- |
| , | (6.3) |

где Соз – основная заработная плата, руб.;

Траз – время раработки, месяцев;

Краз – количество разработчиков, человек;

Сзп – средняя месячная заработная плата.

|  |  |
| --- | --- |
| руб. |  |

### **6.3.3 Дополнительная заработная плата**

Законодательство о труде предусматривает наличие выплат, которые определяются по нормативу в процентах к основной заработной плате по формуле (6.4):

|  |  |
| --- | --- |
|  | (6.4) |

где Соз – основная заработная плата, руб.;

Ндз – норматив дополнительной заработной платы, %.

руб.

### **6.3.4 Отчисления в Фонд социальной защиты населения**

Отчисления в Фонд социальной защиты населения (ФСЗН) определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей и вычисляются по формуле (6.5):

|  |  |
| --- | --- |
|  | (6.5) |

где Соз – основная заработная плата, руб.;

Сдз – дополнительная заработная плата на конкретное ПС, руб.;

Нфсзн – норматив отчислений в Фонд социальной защиты населения, %.

Отчисления в БРУСП «Белгосстрах» вычисляются по формуле 6.6:

|  |  |
| --- | --- |
|  | (6.6) |

руб.

руб.

### **6.3.5 Расходы на материалы**

Сумма расходов на материалы СМ определяется как произведение нормы расхода материалов в расчете на сто строк исходного кода НМ на уточненный объем программного средства *V*o*/* (формула 6.7).

|  |  |  |
| --- | --- | --- |
|  | . | (6.7) |

Учитывая, что норма расхода материалов в расчете на сто строк исходного кода равен 0,46 руб., можно определить сумму расходов на материалы:

СМ = 0,46 ⋅ 5094 / 100 = 23,43 руб.

### **6.3.6 Расходы на оплату машинного времени**

Сумма расходов на оплату машинного времени Смв определяется как произведение стоимости одного машино-часа Смч на уточненный объем программного средства *V*o*/* и на норматив расхода машинного времени на отладку ста строк исходного кода НМВ (формула 6.8).

|  |  |  |
| --- | --- | --- |
|  | . | (6.8) |

Учитывая, что норматив машинного времени на отладку ста строк исходного кода равен 12, можно определить сумму расходов на оплату машинного времени:

Смв =0,06 ⋅ 5094 ⋅ 15 / 100 = 45,85 руб.

### **6.3.7 Прочие прямые затраты**

Сумма прочих затрат Спз определяется как произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив прочих затрат в целом по организации Нпз (формула 6.9).

|  |  |  |
| --- | --- | --- |
|  |  | (6.9) |

Спз = 4500 ⋅ 29,5 / 100 = 1327,5 руб.

### **6.3.8 Накладные расходы**

Сумма накладныхрасходовСобп,обх – произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив накладных расходов в целом по организации Нобп,обх (формула 6.10).

|  |  |  |
| --- | --- | --- |
|  |  | (6.10) |

Все данные необходимые для вычисления есть, поэтому можно определить сумму накладных расходов:

Собп,обх = ⋅ 198 / 100 = руб.

### **6.3.9 Сумма расходов на разработку мобильного приложения**

Сумма расходов на разработку программного средства Ср определяется как сумма основной и дополнительной заработных плат исполнителей на конкретное программное средство, отчислений на социальные нужды, расходов на материалы, расходов на оплату машинного времени, суммы прочих затрат и суммы накладных расходов (формула 6.11).

|  |  |  |
| --- | --- | --- |
|  | Ср = Соз + Сдз + Сфсзн + Сбгс + См + Смв + Спз + Собп,обх, | (6.11) |

Ср =  + + + 20,7 + 23,43 + 45,85 + 1327,5 + 8910 = 17 261,98 руб.

### **6.3.10 Расходы на сопровождение и адаптацию**

Сумма расходов на сопровождение и адаптацию программного средства Срса определяется как произведение суммы расходов на разработки на норматив расходов на сопровождение и адаптацию Нрса (формула 6.12).

|  |  |  |
| --- | --- | --- |
|  | , | (6.12) |

Срса = 17 261,98 ⋅ 17 / 100 = 2934,54 руб.

### **6.3.11 Полная себестоимость**

Полная себестоимость Сп определяется как сумма двух элементов: суммы расходов на разработку Ср и суммы расходов на сопровождение и адаптацию программного средства Срса (формула 6.13).

|  |  |  |
| --- | --- | --- |
|  | , | (6.13) |

Сп = 17261,98 + 2934,54 = 20 196,52 руб.

Полная себестоимость программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе.

### **6.3.12 Определение цены, оценка эффективности**

При расчете прибыли будем опираться на среднюю рыночную цену, по которой можно разработать данное программное обеспечение. Средняя рыночная цена составляет 40250 руб.

Прибыль рассчитывается по формуле (6.14):

|  |  |  |
| --- | --- | --- |
|  | (6.14) | (6.13) |

где Ппс – прибыль от реализации программного средства, руб.;

Цр – средняя рыночная цена продукта, руб.;

Сп – полная себестоимость программного средства, руб.;

Уровень рентабельности разработанного программного средства определяется по формуле 6.15:

|  |  |
| --- | --- |
| , | (6.15) |

где Урент – уровень рентабельности программного средства, %

Сп – полная себестоимость программного средства, руб.;

Ппс – прибыль от реализации программного средства, руб.

## 6.4 Вывод по разделу

В таблице 6.4 представлены результаты расчетов для основных показателей данного раздела в краткой форме.

Таблица 6.4 – Результаты расчетов

|  |  |
| --- | --- |
| Наименование показателя | Значение |
| Время разработки, мес. | 3 |
| Количество программистов, чел. | 1 |
| Зарплата с отчислениями, руб. | 6955,2 |
| Расходы на материалы, оплату машинного времени, прочие, руб | 1396,78 |
| Накладные расходы, руб | 8910 |
| Себестоимость разработки программного средства, руб. | 17 261,98 |
| Расходы на сопровождение и адаптацию, руб. | 2934,54 |
| Полная себестоимость, руб. | 20 196,52 |

Окончание таблицы 6.4

|  |  |
| --- | --- |
| Наименование показателя | Значение |
| Цена аналога, руб. | 40 250 |
| Прибыль от реализации, руб. | 13 345,15 |
| Рентабельность разработки, % | 66,07 |

Необходимость разработки программного средства обусловлена развитием криптовалютных активов ростом интереса людей к ним. Все большее количество пользователей нуждаются в простом и понятном доступе к криптовалюте через устройство, которое находится всегда с ними – мобильный телефон. Чтобы решить данную проблему было разработано программное средство, которое обеспечивает доступ к криптовалюте, а также предоставляет необходимый набор операций над ними.

Разработка программного средства, осуществляемая одним программистом в течение трех месяцев, при заданных условиях обойдется компании в 20 196,52 руб. Реализации данного программного средства по среднерыночной цене в 40 250 руб., принесет прибыль компании в сумме 13 345,15 рублей. Уровень рентабельности составит 66,07%.

# **Заключение**

с

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Марковский А.Г.*

Пров.

Годун А.В.

Консульт.

Годун А.В.

Н. контр.

Рыжанкова А.С.

Утв.

Пацей Н.В.

Заключение

Лит.

Листов

1

*74417049, 2021*

В ходе дипломного проектирования проанализированы существующие мобильные приложения, применяемые для предоставления пользователям доступа к их криптоактивам с возможностью выполнять и отслеживать транзакции, выявлены их достоинства и недостатки. Произведенный анализ позволил определить проблемы анализируемых программных продуктов и найти пути их решения, чтобы применить их в рамках разработки собственного проекта.

Рассмотрены основные термины и определения, которые помогают прояснить необходимые технические аспекты и дают общее представление о работе технологии *blockchain*.

Рассмотрены различные библиотеки, которые могут быть использованы при разработке программного средства, описаны их достоинства и недостатки. На основе рассмотренных технологий был сделан и обоснован выбор тех библиотек, которые использовались в разработке приложения.

Спроектирована общая архитектура приложения. Описан принцип взаимодействия между компонентами приложения, а также архитектура каждого отдельного модуля.

Рассмотрен процесс разработки мобильного приложения: даны примеры различных элементов приложения, подробно описано их назначение, а также принцип взаимодействия

Разработано мобильное приложение, представляющее собой приложение для платформы *iOS*, назначением которого является предоставление пользователю доступа к его криптоактивам и проведение операций над ними и имеет следующий функционал:

* приложение должно позволять создавать или импортировать криптокошельки *Bitcoin*, *Ethereum*, *Litecoin* криптовалют;
* функция отправки транзакций поддерживаемых криптовалют;
* возможность просмотра истории транзакций каждого кошелька;
* возможность добавление нескольких кошельков какой-либо поддерживаемой криптовалюты (мультиаккаунтность);
* возможность выбора рабочей сети между *mainnet* и *testnet*;
* функция сканирования *QR*-кодов, содержащих адреса крипто-кошельков, для дальнейшего создания транзакции;
* функция генерации *QR*-кода адреса криптокошелька пользователя;
* возможность выбора времени автоблокировки приложения;
* функция защиты значимых операций пин-кодом.

В результате проделанной работы все обозначенные цели достигнуты, а задачи по реализации выполнены в полном объеме.

# **Список использованных источников**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Марковский А.Г.*

Пров.

Годун А.В.

Консульт.

Годун А.В.

Н. контр.

Рыжанкова А.С.

Утв.

Пацей Н.В.

Список использованных источников

Лит.

Листов

1

*74417049, 2021*

1. Как хранить криптовалюту [Электронныйресурс]. / RBC – Режимдоступа: https://www.rbc.ru/crypto/news/5f59fc719a794769c28debef. – Дата доступа: 27.03.2021.
2. Безопасности приложений [Электронныйресурс]. / habr – Режимдоступа: https://habr.com/ru/company/redmadrobot/blog/349272. – Дата доступа: 28.03.2021.
3. Обзор видов 2FA [Электронныйресурс]. / anti-malware – Режимдоступа: https://www.anti-malware.ru/analytics/Market\_Analysis/commercial-2FA-applications. – Дата доступа: 01.04.2021.
4. Топ мобильных криптокошельков [Электронныйресурс]. / ihodl – Режимдоступа: https://ru.ihodl.com/analytics/2020-03-03/top-3-mobilnyh-kriptokoshelkov. – Дата доступа: 15.04.2021.
5. Платформа iOS[Электронныйресурс]. / wikipedia – Режимдоступа: https://ru.wikipedia.org/wiki/IOS. – Дата доступа: 29.03.2021.
6. Гибридное ядро XNU [Электронный ресурс]. / wikipedia – Режим доступа: https://ru.wikipedia.org/wiki/XNU. – Дата доступа: 02.04.2021.
7. Swift – язык пограммирования [Электронный ресурс]. / wikipedia – Режим доступа: https://ru.wikipedia.org/wiki/Swift. – Дата доступа: 03.04.2021.
8. IDE Xcode[Электронный ресурс]. / wikipedia – Режим доступа: https://ru.wikipedia.org/wiki/Xcode. – Дата доступа: 10.04.2021.
9. Что такое формат JSON [Электронный ресурс]. / timeweb – Режим доступа: https://timeweb.com/ru/community/articles/format-json. – Дата доступа: 13.04.2021.
10. Разбор архитектуры VIPER[Электронныйресурс]. / habr –Режим доступа: https://habr.com/ru/post/358412/. –Дата доступа: 16. 04. 2021.
11. Ручное тестирование приложений [Электронный ресурс]. / habr – Режим доступа: https://habr.com/ru/company/skillbox/blog/418889. – Дата доступа: 07.05.2021.
12. Калькулятор стоимости ПО [Электронный ресурс]. / appcraft – Режим доступа: https://appcraft.pro/blog/kalkulyator-stoimosti-mobilnogo-prilozheniya/. – Дата доступа: 16.05.2021.

# **ПРИЛОЖЕНИЕ А**

Диаграмма вариантов использования

# **ПРИЛОЖЕНИЕ Б**

Диаграмма классов *BTCAuth* сервиса

# **ПРИЛОЖЕНИЕ В**

Блок-схема процесса автоблокировки мобильного приложения

# **ПРИЛОЖЕНИЕ Г**

Блок-схема процесса создания *Bitcoin* криптокошелька

# **ПРИЛОЖЕНИЕ Д**

Скриншоты работы мобильного приложения

# **ПРИЛОЖЕНИЕ Е**

Блок-схема процесса создания *Bitcoin* транзакции