СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

В настоящей пояснительной записке применяют следующие обозначения и сокращения:

API	– Application Programming Interfaces
BLL	– Business Logic Layer
DAL	– Data Access Layer
MVC	– Model View Controller
UML	– Unified Modeling Language
БД	– база данных
ПО	– программное обеспечение
ПП	– программный продукт
СУБД	– система управления базами данных

ВВЕДЕНИЕ

Современный мир невозможен без денег. Проблема состоит в том, что не каждый человек может с уверенностью сказать, что хорошо умеет распоряжаться деньгами, знает в какое русло их направить и как заставить их «работать».

Финансовое образование необходимо всем категориям граждан. И одним из основных направлений должно быть обучение детей и студентов. Так как именно повышение финансовой грамотности этой возрастной категории поможет заложить фундамент для дальнейшего развития навыков планирования бюджета и сбережений, поможет в решении проблем финансирования и подготовит к самостоятельной жизни.

Одними из лучших решений по ненавязчивому обучению являются геймификация и совмещение «приятного с полезным» то есть предложение полезного приложения, которое помимо основной функции будет включать в себя и дополнительный функционал. В нашем случае основным функционалом будет инструмент планирования бюджета, а дополнительными инструментами будет постановщик целей и выдача рекомендаций на основе анализа введенных доходов и расходов.

Актуальность данной тематики не подлежит сомнению. Параллельно с повышением уровня финансовой грамотности населения развиваются и рынки финансовых услуг, сокращаются издержки денежного обращения, в финансовую систему вовлекаются возрастающие сбережения населения, создаются условия для развития страховых рынков, население получает знание принципов и инструментов финансового рынка, снижаются риски подверженности паники на потребительском и финансовом рынках.

Существует достаточно большое количество приложений по планированию бюджета (monefy, дребеденьги, CoinKeeper и тому подобные) и все они имеют свои преимущества и недостатки. Но, ни в одном из приложений не реализован метод «приятное с полезным». Ни одно из проанализированных приложений не предлагают свои варианты по оптимизации расходов и доходов. Ни одно из приложений не учит правильно обращаться с деньгами. Учитывая эти моменты, мое приложение не имеет полноценных аналогов на сегодняшний день.

Несомненно, мое приложение не является панацеей от всех бед, но оно будет выполнять ряд функций, которые помогут подать основы финансовой грамотности населению в простой и интересной форме.

Настоящий документ описывает анализ, проектирование, прототипирование и разработку мобильного приложения «финансовый

помощник студента» на платформе Android.

Основными целями приложения являются повышение уровня знаний студентов и предоставление удобного инструмента планирования бюджета.

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ. ПОСТАНОВКА ЗАДАЧИ

1.1 Постановка задачи

В рамках данного проекта поставлена задача разработать мобильное приложение, которое упростит и сделает более удобным процесс планирования бюджета, а так же, будет выполнять функции финансового советника и обучать пользователя основам финансовой грамотности.

При разработке программного обеспечения были поставлены следующие задачи:

- разработать мобильное приложение, способное собирать и анализировать финансовую информацию пользователя;
- обеспечить гибкость и расширяемость приложения;
- сделать упор на дизайн и юзабилити для удобства пользования приложением.

1.2 Анализ предметной области

Для выявления потребностей целевой аудитории был приведен социологический опрос. В процессе анализа результатов была выявлена актуальная тематика для приложения – планирование бюджета т.к. большинство респондентов хоть и планируют бюджет, но не используют специализированные программы из-за их неудобства.

Для того, что бы удостовериться в отсутствии достойных аналогов, обладающих всем необходимым функционалом и включающих функции обучения, был проведен выборочный анализ нескольких приложений из топов андроид-магазинов.

В настоящее время существует ряд приложений, которые предоставляют возможность планировать бюджет. Ниже представлены примеры некоторых из них, с описанием положительных и отрицательных моментов.

Monefy

Мобильное приложение для контроля финансов. В настоящее время, данное приложение является одним из лидеров по количеству пользователей и по их оценке. Данное приложение привлекает удобным и лаконичным дизайном, простотой и понятным интерфейсом, поддерживает создание и редактирование неограниченного количества категорий доходов и расходов. Имеет синхронизацию с dropbox. Большой плюс приложения - поддержка множества валют. Приложение дорабатывается и в настоящее время.

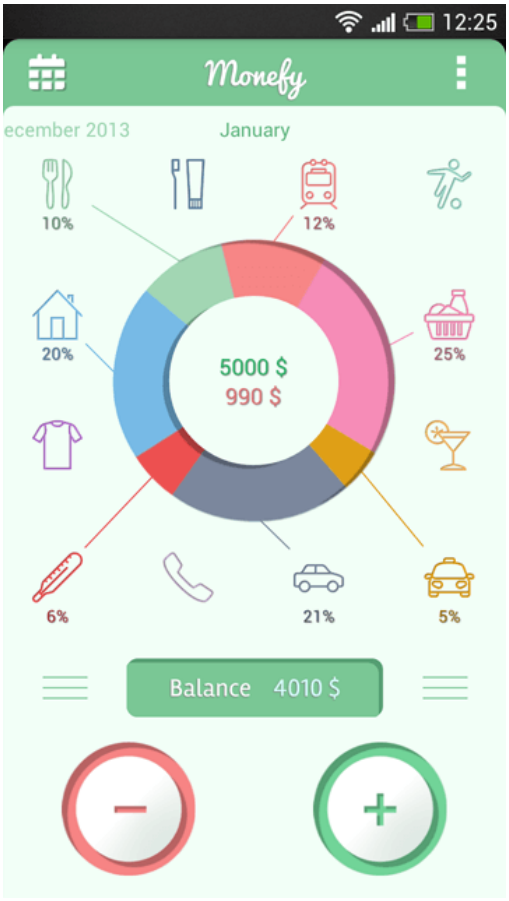

Рисунок 1.1 – Скриншот приложения «Monefy»

Плюсы приложения:

- выбор отчетного периода (день, неделя, месяц, год);
- выбор валюты;
- просмотр общей статистики;
- просмотр статистики трат по категориям;
- возможность добавить свою категорию;
- добавление дохода в три категории (зарплата, депозит, сбережения).

Минусы приложения:

- нет конверсии валют;
- нет возможности добавить ежемесячные платежи/ежемесячные финансовые поступления;
- нет графика трат по дням;
- нет предупреждений об уходе в минус;
- нет возможности записать предстоящие траты.

Учет расходов. Дребеденьги

Простое приложение для учета личных финансов. В настоящее время, данное приложение предоставляет наиболее широкий функционал среди конкурентов статистики на сайте разработчиков. В платном расширении

имеется возможность сбора затрат всей семьи, что позволяет эффективно планировать семейный бюджет. Так же, в качестве одной из особенностей приложения, можно выявить сканер чеков (что ускоряет процесс ввода данных) и настройку получения отчетов по общей статистике и уведомлений о выходе за лимит.

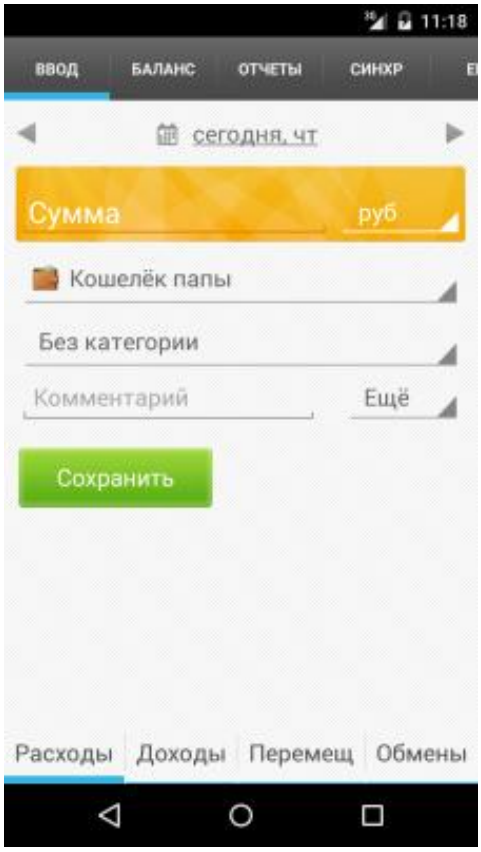

Рисунок 1.2 – Скриншот приложения «Дребеденьги»

Плюсы приложения:

- автоматический ввод расходов (есть возможность обработки смс от банка);
- многопользовательский режим;
- возможность формирования списка покупок;
- синхронизация с сайтом;
- возможность записи об обменных операциях;
- возможность вносить данные по фотографии с чека.

Минусы приложения:

- несколько устаревший дизайн;
- не дружественный интерфейс;
- предоставление полной статистики только на сайте.

CoinKeeper

Рисунок 1.3 – Скриншот приложения «CoinKeeper»

Плюсы приложения:
- вся необходимая информация содержится на главном экране;
- обучение при первом запуске;
- возможность добавления новых категорий;
- три области (счета, доходы, расходы);
- дружественный интерфейс;
- возможность задавать лимит на категорию;
- напоминание о периодических расходах.

Минусы приложения:
- функционал значительно слабее, чем у аналогов;
- нет реализации нескольких валют;
- не совсем понятный вывод статистики;
- тормозит на топовых смартфонах.

Приложение несколько уступает по функционалу предыдущему, но является лучшим в своей категории в плане юзабилити, дизайна и подхода к реализации.

Мой бюджет-план

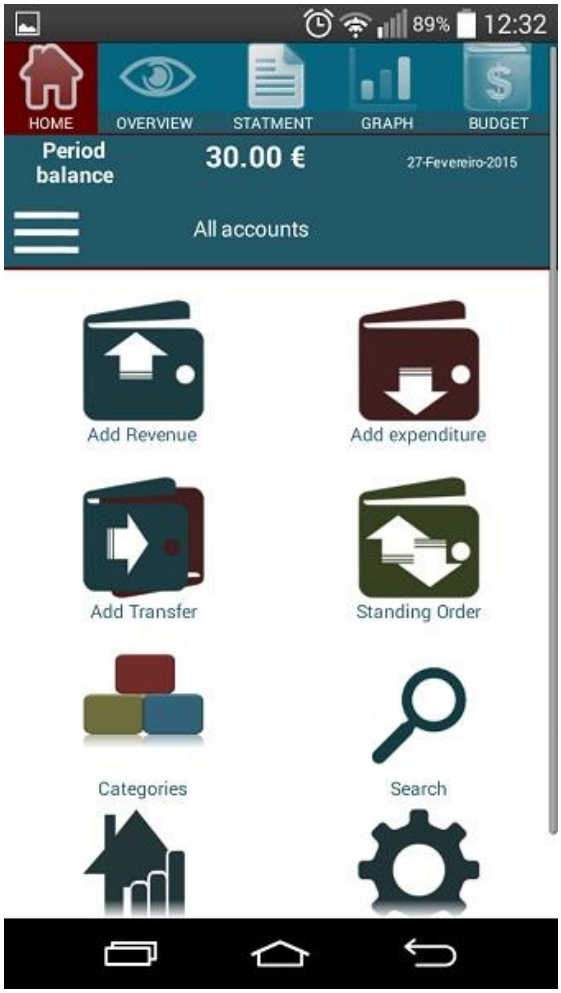

Рисунок 1.4 – Скриншот приложения «Мой бюджет-план»

Плюсы приложения:
- добавление регулярных платежей и доходов;
- наличие кредитного калькулятора;
- обзор всех доходов и расходов;
- возможность поиска определенных расходов.

Минусы приложения:
- устаревший дизайн;
- отсутствие стандартных категорий;
- нечитабельный отчет;
- нет возможности добавить больше 10 млн бел. рублей;
- неудобный интерфейс;
- приложение тормозит даже на топовых смартфонах.

1.3 Анализ конечного потребителя

В рамках дипломного проекта был проведен социологический опрос целевой аудитории (т.е. студентов) с целью выявления основных потребностей в ПО, связанного с финансовой грамотностью. Опросник содержал различные вопросы, призванные выявить наиболее интересные для целевой аудитории тематики, наиболее популярные услуги и отношения выбранной группы к тому или иному финансовому инструменту

Данный опрос прошло 602 человека, что дает нам право опираться на его результаты. Ниже представлены графики, отображающие ответы респондентов.

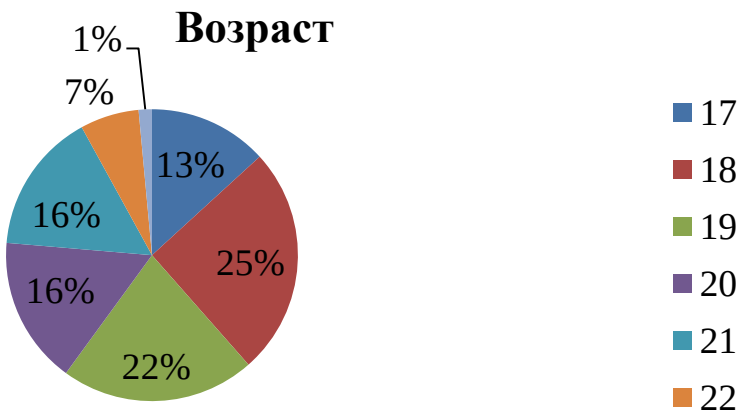

Рисунок 1.5 – График «Возраст»

Как мы видим на рисунке 1.5, подавляющее большинство интервьюеров – это возрастная группа 18- 21 года, т.е. студенты второго – четвертого курсов. Студенты именно этих курсов и являются потенциальными пользователями моего приложения.

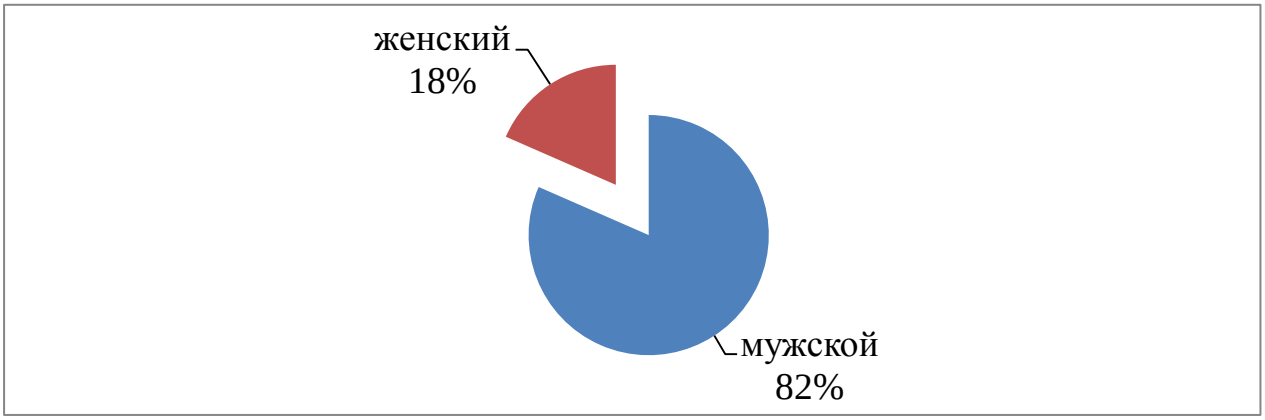

Рисунок 1.6 – График «Пол»

Из рисунка 1.6 видно, что основные пользователи – мужчины. При разработке приложения это будет учитываться на стадии дизайна. Но, для привлечения и женской части можно реализовать дизайн в нейтральных цветах, либо добавить возможность смены цветового оформления мобильного приложения.

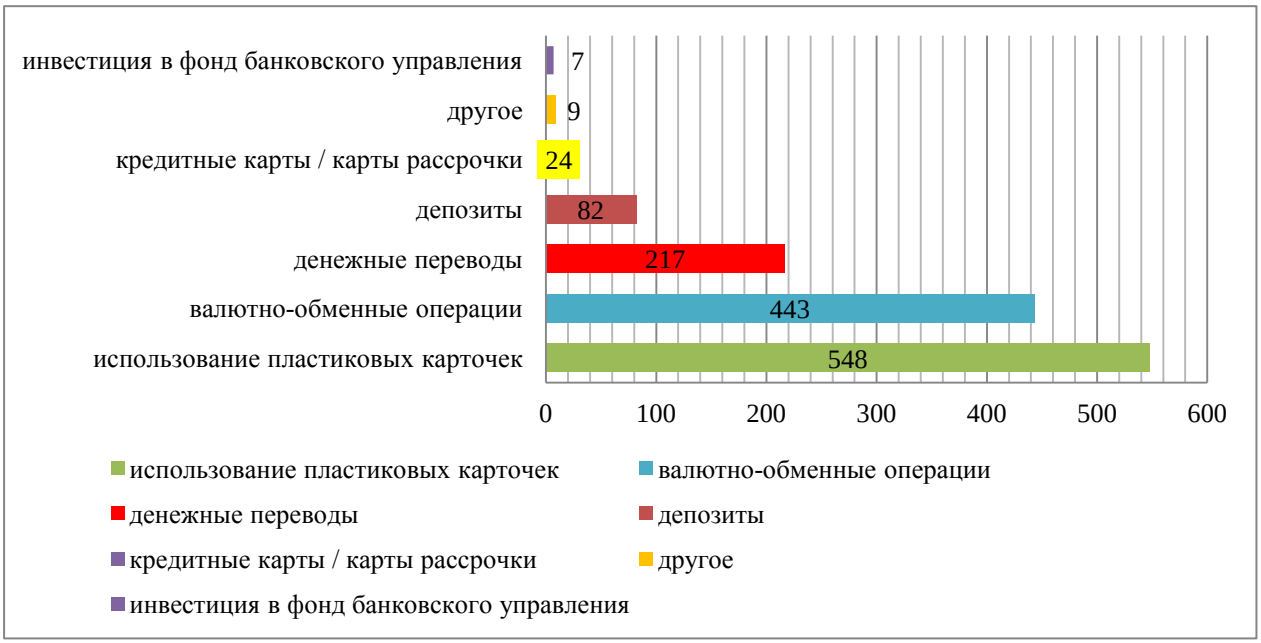

Рисунок 1.7 – График «Популярность банковских услуг»

На рисунке 1.7 продемонстрирован график популярности банковских услуг среди студентов второго – четвертого курсов. Данная информация будет полезна при выборе основного направления приложения.

Следующий график, изображенный на рисунке 1.8, показывает популярность банковских инструментов и сервисов среди выбранной целевой аудитории.

Как мы можем видеть на графике – подавляющее большинство респондентов являются пользователями интернет – банкинга, а значит тематика онлайн-платежей может быть интересна данной аудитории.

Так же, из графика можно понять, что инструмент просмотра курсов валют, тоже может быть включен в приложение о повышении финансовой грамотности.

Но, учитывая что сервисов по предоставлению такой информации достаточно, можно принять к сведению интерес к курсам валют и добавить статьи по этой тематике в раздел «Обучение».

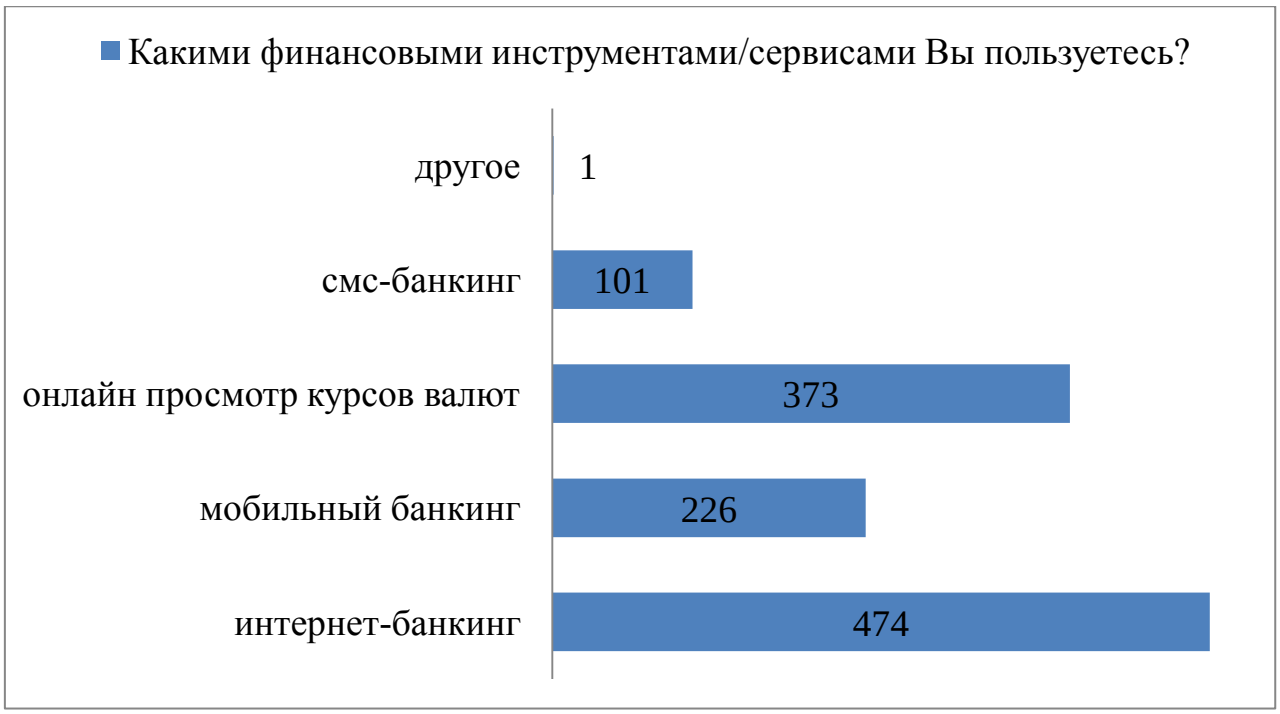

Рисунок 1.8 – График «Популярность финансовых инструментов»

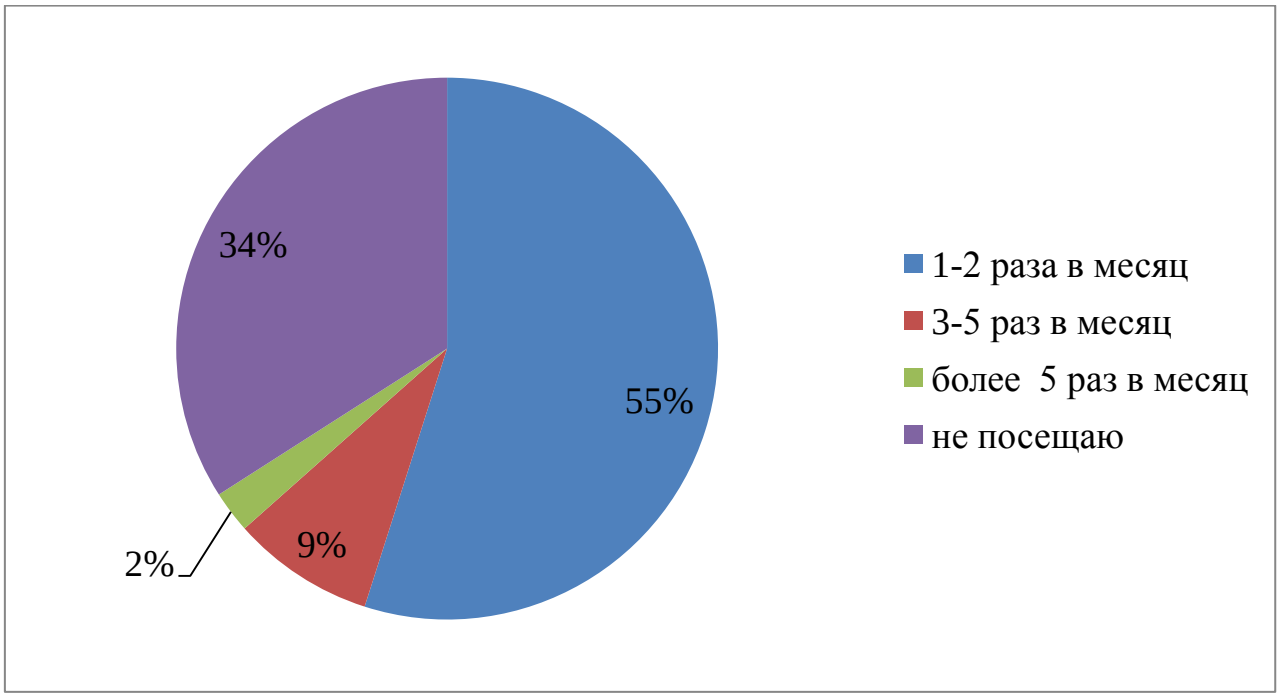

Рисунок 1.9 – График «Частота посещения учреждений банков»

Исходя из графиков, изображенных на рисунке 1.7-1.9, можно сделать вывод о том, что использование пластиковых карточек среди студентов набирает популярность. Это утверждение подкрепляется тем фактом, что 34% респондентов не посещают банковских учреждений и 55% посещают их

достаточно редко. Это особенность также можно учесть в разрабатываемом приложении.

Следующий вопрос был направлен на сбор информации о приоритетах целевой аудитории. В вопросе предлагалось ответить на вопрос «Если бы у вас были лишние деньги, на что бы вы их потратили?». Результаты опроса можно увидеть на рисунке 1.10.

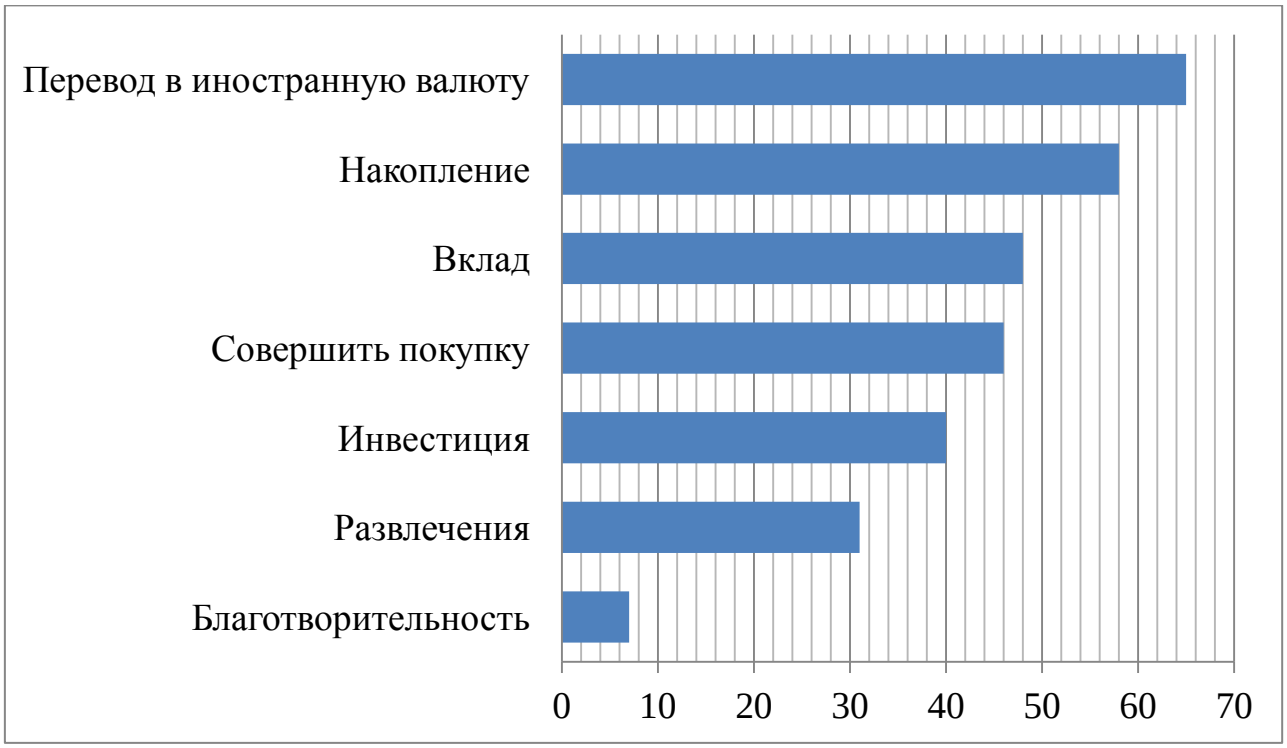

Рисунок 1.10 – График «Приоритеты»

Данные варианты ответов в достаточной степени повлияли на выбор тематики моего приложения в целом и отдельного раздела в частности.

Как мы видим, большинство опрошенных перевели бы деньги в иностранную валюту, либо просто откладывали бы их. Это говорит о достаточно низком уровне финансовой грамотности, неуверенности в современной экономике и банковских системах.

В то же время, количество ответом «Вклад» и «Инвестиция» оказались выше ожидаемого, из чего можно сделать вывод о том, что данные тематики все же отчасти знакомы и интересны целевой аудитории.

Следующая группа вопросов и ответов на них стала основополагающей, при выборе тематики моего приложения.

Рисунок 1.11 визуализирует ответы на вопрос «планируете ли вы свой бюджет?»

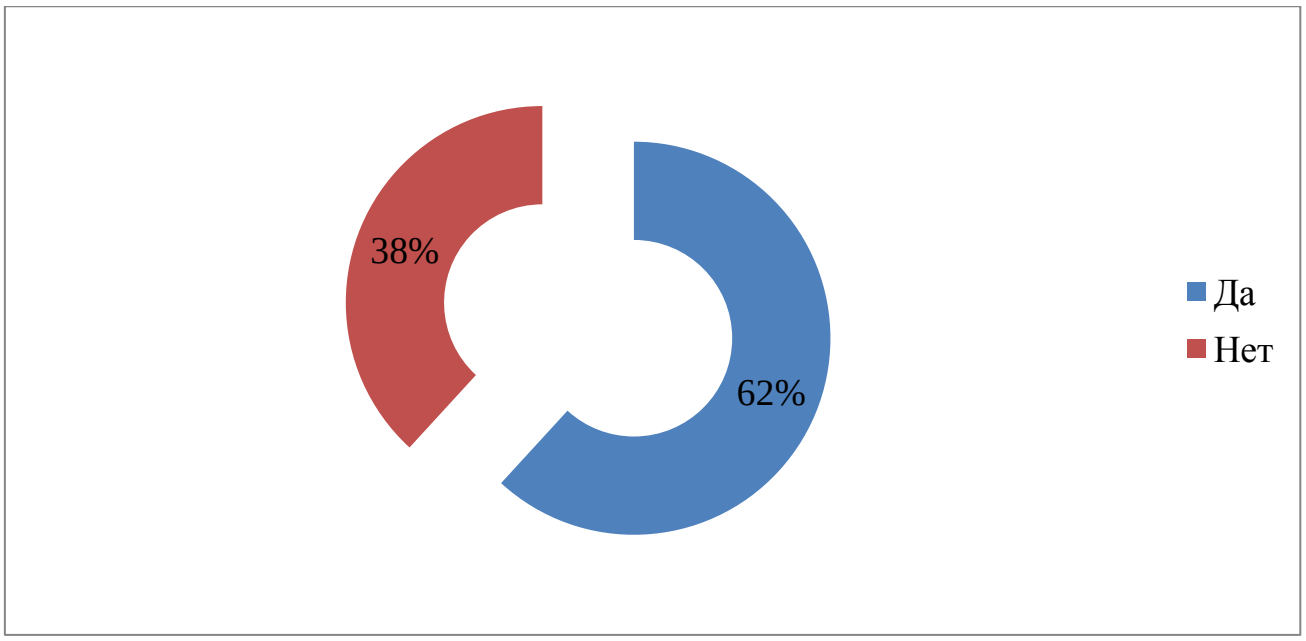

Рисунок 1.11 – График «Планируете ли вы свой бюджет?»

Несмотря на большой процент положительно ответивших респондентов, была выявлена проблема – отсутствие удобных инструментов планирования бюджета, что приведено на рисунке 1.12.

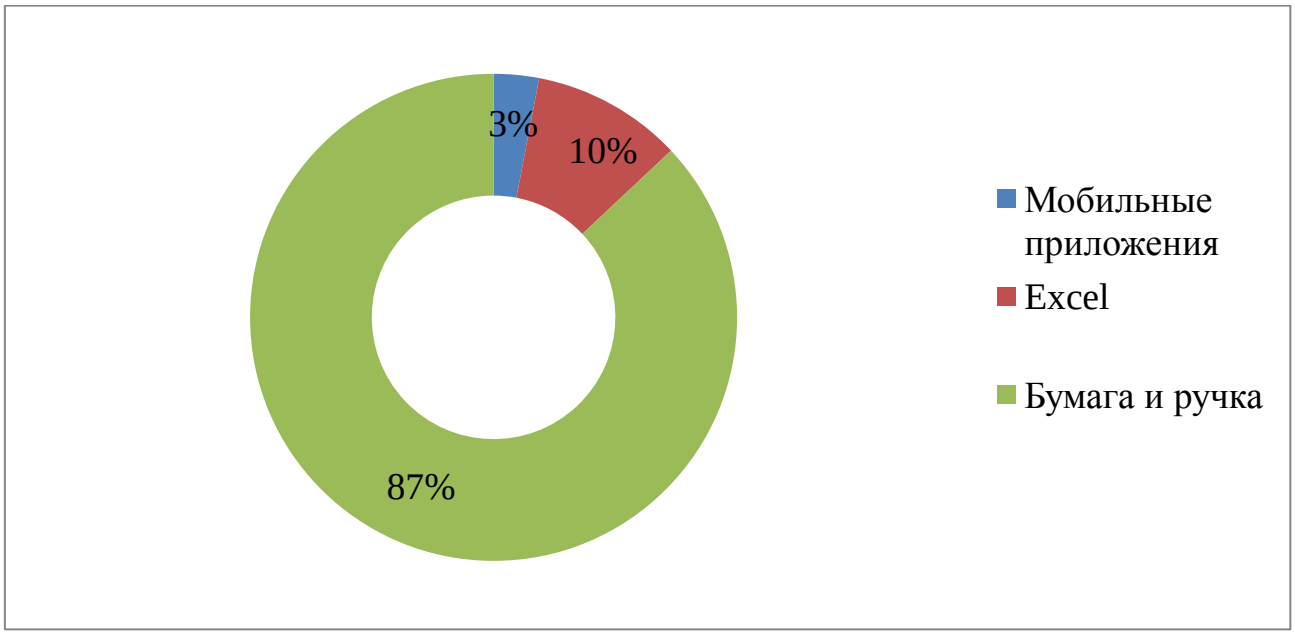

Рисунок 1.12 – График «Инструменты планирования бюджета»

Т.е. в век информационных технологий подавляющее большинство студентов IT специальностей все еще используют банальный ручной расчет для планирования бюджета. Из этого можно сделать следующий вывод – или до

сих пор не существует удобных приложений для планирования бюджета, подходящего данной целевой аудитории, или приложения есть, но в настоящий момент они не распространены из-за того, что целевая аудитория о них просто не знает. В любом из двух случаев мое приложение будет полезным и актуальным.

Таким образом, мы подобрали «полезный» функционал для дипломного проекта, но, приложение должно содержать еще и обучающую часть. Следующие несколько вопросов помогут в некоторой степени выбрать правильную стратегию подачи информации.

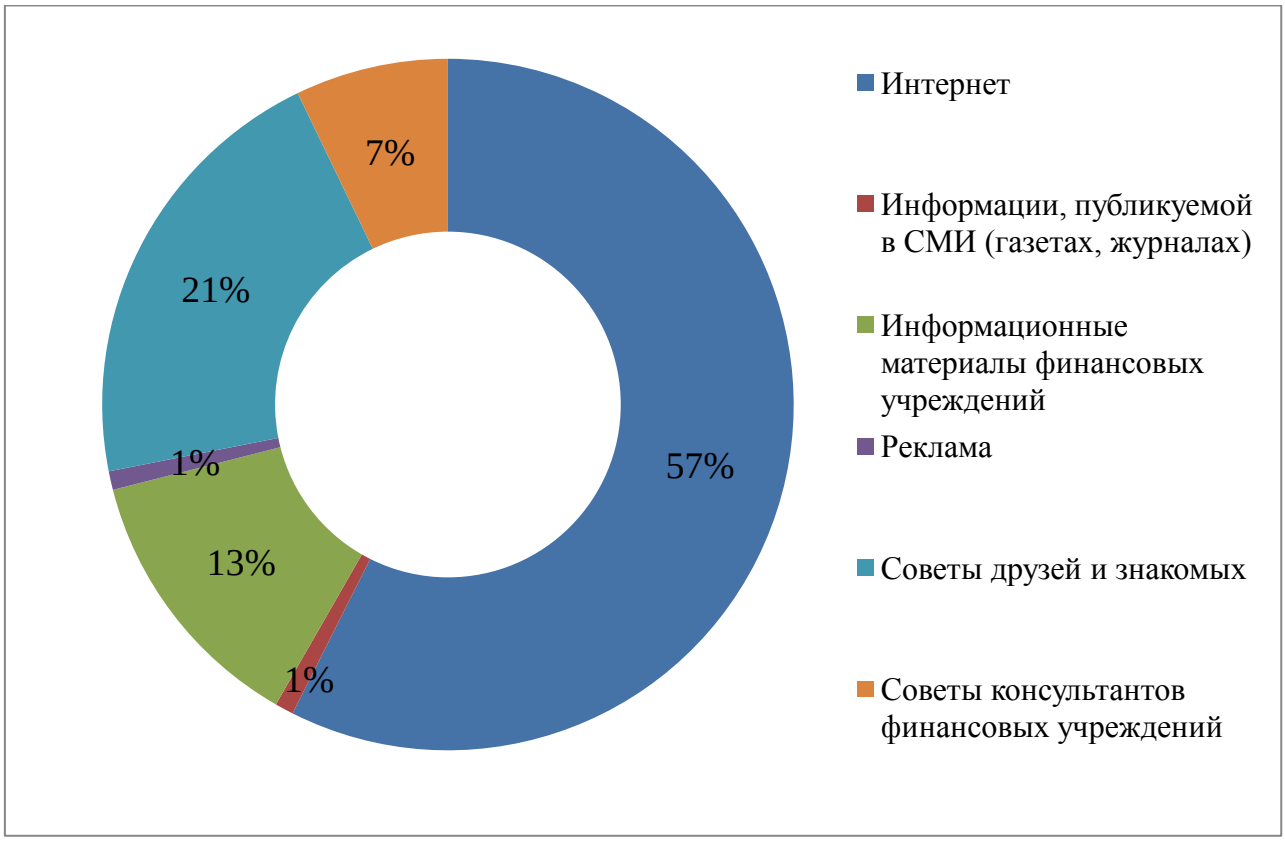

Рисунок 1.13 – График «Какие источники информации вы предпочитаете при выборе финансовых услуг»

Из рисунка 1.13 очевидно, что целевая аудитория доверяет информации из интернета, соответственно и для обучения можно использовать информацию из этого источника. Последний вопрос заключается в том – какую информацию необходимо давать пользователям. Несомненно, вся информация из программы Национального Банка Республики Беларусь будет полезной. Но, далеко не факт что она будет интересной. Ниже представлены самые популярные тематики из ответов на вопрос «что бы вы хотели узнать об управлении финансами»:

- вклады;

- как сохранить деньги;
- как управлять деньгами;
- как размножать деньги;
- как оптимизировать расходы;
- инвестиции;
- эффективное планирование бюджета;
- поиск работы.

Мое приложение отвечает на все эти вопросы. Оно помогает грамотно оптимизировать расходы, помогает сохранить деньги и спланировать бюджет. Несомненно, такие приложения уже существуют. Но, главная особенность моей программы – анализ статистических показателей и предложение пользователю именно той информации, которая будет ему актуальна и интересна в настоящий момент. Вторая особенность приложения заключается в постановке целей. На эту мысль меня натолкнуло разнообразие ответов на вопрос по приоритетам (рисунок 1.10).

1.4 Перспективы развития приложения

В перспективе, данное приложение может быть расширено. К приложению могут быть добавлены такие функции как:

1. Финансовый органайзер. Приложение будет напоминать пользователю о необходимых платежах (оплата обучения, общежития, интернета);
2. Поиск вакансии. Основная цель приложения – помочь не только сэкономить деньги, но и помочь их заработать. Теоретически, можно передавать приложению данные с сайтов по поиску работы и, на основании заполненной пользователем формы предлагать различные варианты;
3. Благодаря партнерским программам, возможно подключить рекомендации статей не только по финансовой тематике, но и по теме саморазвития, интересов, хобби в зависимости от выбранной цели. Таким образом, данное приложение можно превратить в коммерческое и работать по лидам.

1.5 Подготовка и анализ требований. Составление спецификации

Одним из самых важных этапов при разработке является сбор и организация требований. Хорошо описанные требования существенно снижают риски при проектировании архитектуры, разработке и тестировании программного продукта.

1.5.1 Функциональные требования

Разрабатываемая система предназначена для анализа доходов и расходов, а так же, для повышения уровня финансового образования целевой аудитории. Пользователь приложения имеет возможность:

- вносить данные о доходах;
- вносить данные о расходах;
- вносить ограничения по расходам;
- получать визуальное отображение расходов/доходов по каждой тематике;
- вносить данные о цели;
- получать отображение информации о процентах достижения целей;
- получать общую статистическую информацию о финансовом состоянии;
- получать рекомендации статей к прочтению;
- получать оповещения при выходе за ограничения по расходам.

1.5.2 Нефункциональные требования

Для использования системы достаточно иметь смартфон на платформе Android с версией операционной системы 2.3 и выше.

Для использования всех функций приложения необходима возможность подключения к сети Интернет.

У пользователей системы не должно уходить много времени на изучение или привыкание к пользовательскому интерфейсу. Планируемый срок обучения до 10 минут. Интерфейс приложения должен быть интуитивно понятен целевой аудитории.

2 ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

2.1 OS Android

В рамках данного проекта, приложение разрабатывается под операционную систему Android.

Android – одна из операционных систем нового поколения, созданных для работы с аппаратным обеспечением современных мобильных устройств. На сегодняшний день Windows Mobile, Apple iPhone и Palm Pre предлагают достаточно мощные и более простые в использовании среды разработки мобильных приложений. Однако в отличие от Android это запатентованные операционные системы, в которых в определенных случаях приоритет отдается встроенному ПО, а не приложениям сторонних программистов.

Кроме того, эти операционные системы ограничивают возможности взаимодействия приложений с данными телефона, а также ограничивают или контролируют процесс распространения сторонних приложений, созданных для данных платформ [1].

Android дает новые возможности для мобильных приложений, предлагая открытую среду разработки, построенную на открытом ядре Linux. У всех приложений есть доступ к аппаратным средствам устройства, для чего используются специальные серии API-библиотек. Кроме того, здесь включена полная и контролируемая поддержка взаимодействия приложений.

На платформе Android все программы имеют одинаковый статус. Сторонние приложения написаны на том же API, что и встроенное ПО, при этом во всех программах одинаковое время исполнения.

Как это часто бывает в IT многие вещи не могут быть объяснены в отрыве от истории возникновения конкретного программного обеспечения. Вот почему мы должны обратиться к истокам ОС Android.

Разработка ОС Android была начата в 2003 молодой компанией Android Inc. В 2005 году эта компания была куплена Google. Главные особенности архитектуры Android были определены именно в этот период. Это заслуга не только Android Inc; архитектурные концепции и финансовые ресурсы Google оказали решающее влияние на архитектуру Android.

2003-2005 года были ознаменованы повышенным вниманием к AJAX приложениям. Это оказало основополагающее влияние на архитектуру Android: во многих аспектах она ближе к архитектуре типичного AJAX приложения, нежели к десктопному GUI приложению, написанному на Java, C#, C++, VB и т. п.

Это придумал кто-то из Google в тот период, когда насыщенные

интернет-приложения (Rich Internet Applications, RIA) в духе Google Docs 10 или Gmail считались решением всех проблем. Эту идею нельзя назвать ни плохой, ни хорошей. Просто Android-приложения очень сильно отличаются от десктопных [2].

Влияние архитектурной философии Eclipse заметно в выборе принципа реализации GUI, который больше похож на SWT, нежели на Swing. В стандартах оформления кода Android присутствует «венгерская нотация», рождѐнная в стенах MS. Можно предположить, что тот, кто писал эти стандарты, ранее занимался разработкой под Windows.

2.2 Программный стек Android

Программный стек Android состоит из элементов, показанных на рис. 2.1. Их подробное описание приводится ниже.

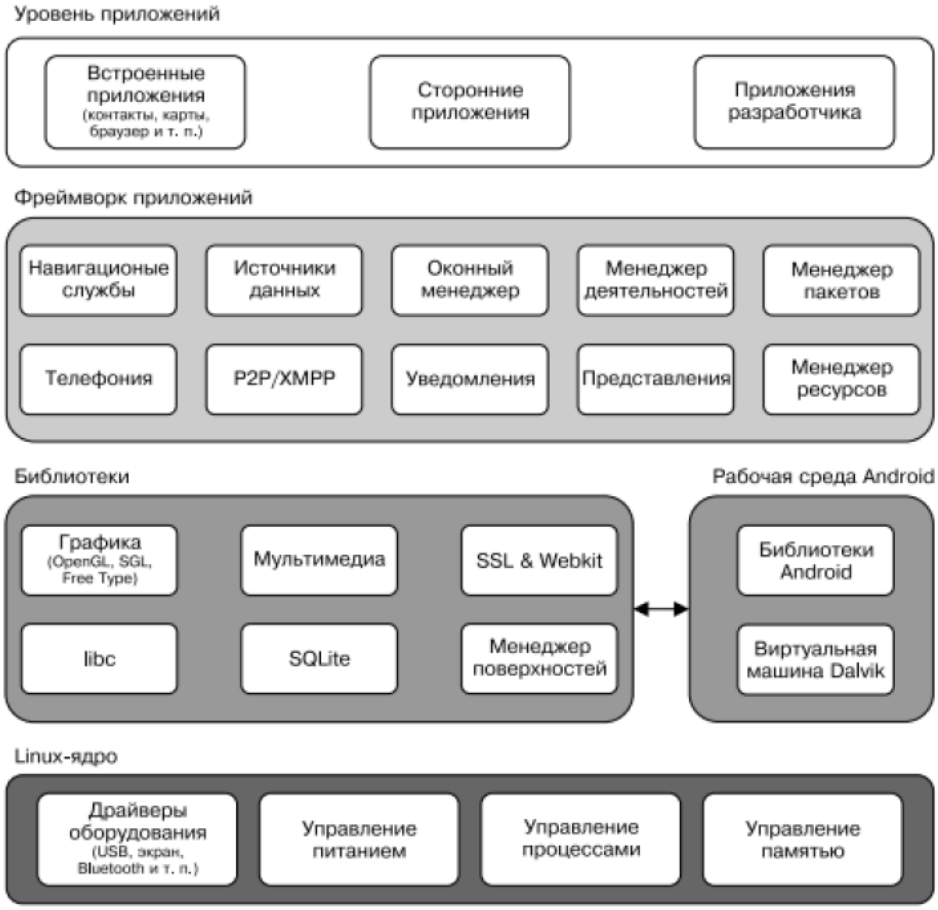

Рисунок 2.1– Программный стек Андроид

Упрощенно их можно представить как комбинацию ядра Linux и набора библиотек C/C++, которые доступны в фреймворке приложения. Последний

обеспечивает управление и функционирование рабочей среды и приложений.

Работу системных служб (драйверы устройств, управление процессами и памятью, питанием, безопасность, сетевые службы) обеспечивает ядро Linux версии 2.6. Оно также отвечает за уровень абстракции между аппаратной начинкой и остальной частью программного стека.

Android включает разнообразные системные библиотеки C/C++ (например, SSL и libc), которые работают поверх ядра. Среди них можно выделить:

1. Библиотеку для работы с мультимедиа, которая обеспечивает проигрывание аудио- и видеофайлов;

2. Менеджер интерфейса, отвечающий за управление отображением;

3. Графические библиотеки, такие как SQL и OpenGL, для работы с 2D- и 3D-графикой;

4. Библиотеку SQLite, обеспечивающую работу встроенных баз данных;

5. SSL и WebKit для работы встроенного веб-браузера и обеспечения интернет-безопасности.

Особенным телефон на платформе Android делает не столько мобильная версия ОС Linux, сколько рабочая среда Android. Она включает в себя библиотеки ядра и виртуальную машину Dalvik и обеспечивает функционирование программ, а вместе с библиотеками формирует основу фреймворка приложений:

1. Библиотеки ядра. Хотя приложения для Android разрабатываются на языке Java, Dalvik — это не виртуальная Java-машина. Библиотеки ядра Android обеспечивают основную функциональность библиотек ядра Java, а также присущий Android уникальный функционал;

2. Виртуальная машина Dalvik. Dalvik — это виртуальная машина на основе регистров, которая оптимизирована таким образом, чтобы на устройстве можно было запускать несколько приложений одновременно. В ее основе ядро Linux, которое обеспечивает работу потоков и низкоуровневое управление памятью;

3. Фреймворк приложений. Фреймворк включает набор классов, которые используются для разработки приложений. Он также предоставляет обобщенные абстрактные классы для доступа к оборудованию и обеспечивает управление пользовательским интерфейсом и ресурсами приложения;

4. Уровень приложений. Все программы, как встроенные, так и сторонние, разрабатываются на уровне приложений с использованием одних и тех же библиотек API. Уровень приложений функционирует внутри рабочей среды Android, используя классы и службы, открытые для доступа на этом уровне.

2.2.1 Уровень Linux

Если рассматривать Linux на высоком уровне, то это комбинация ядра (без которого нельзя обойтись) и множества других, необязательных частей. Можно даже запустить одно ядро, без чего бы то ни было ещѐ. Так, Google вынуждена в любом случае использовать ядро Linux как часть ОС Android. Кроме того, были рассмотрены необязательные части и из них выбрано самое необходимое. Например, были добавлены сетевой фаервол IPTables и оболочка Ash. Любопытно, что добавили именно Ash, а не Bash, несмотря на то, что последний на порядок мощнее; вероятно, это решение было основано на том, что Ash менее требователен к ресурсам.

Разработчики Android модифицировали ядро Linux, добавив поддержку железа, используемого в мобильных устройствах и, чаще всего, недоступного на компьютерах.

Выбор Linux в качестве основы оказал огромное влияние на все аспекты ОС Android. Сборка Android, по сути, есть вариация процесса сборки Linux. Код Android находится под управлением git (инструмент, разработанный для управления кодом Linux).

2.2.2 Уровень инфрастуктуры приложения

Несмотря на некоторое сходство Apple iOS и Android ОС, существуют значительные отличия между архитектурными решениями на инфраструктурном уровне обоих ОС.

Apple решила использовать Objective-C как язык программирования и среду выполнения приложения iOS. Objective-C выглядит более или менее естественным выбором для ОС, в основе которой лежит Free BSD. Можно рассматривать Objective-C как обычный C++ с пользовательским препроцессором, который добавляет некоторые специфические лингвистические конструкции. Почему же нельзя использовать стандартный C++, на котором написана Free BSD? Причина в том, что Apple старается все делать в своем, ―эппловском стиле.

Основная идея в том, что приложения iOS написаны более или менее на том же языке, что и стоящая за ними ОС. Android-приложения сильно отличаются в этом смысле. Они написаны на Java, а это совсем другая технология, нежели C++ (хотя синтаксис и унаследован от C++).

Основная причина состоит в необходимости одному и тому же приложению работать на различном аппаратном обеспечении. Эта проблема имеет место лишь для ОС Android; у ребят из Apple такой проблемы нет. iOS

работает только на оборудовании собственного производства, и Apple полностью контролирует весь процесс. Для Android же все наоборот: Google не контролирует производителей аппаратных средств. Например, ОС Android работает на процессорах с архитектурой x86, ARM и Atom. На бинарном уровне эти архитектуры несовместимы.

Если бы архитекторы ОС Android выбрали тот же путь, что и архитекторы из Apple, разработчики приложений под Android были бы вынуждены распространять несколько версий одного и того же приложения одновременно. Это стало бы серьезной проблемой, которая могла бы привести к краху всего проекта Android.

Для того чтобы одно и то же приложение могло работать на разном аппаратном обеспечении, компания Google использовала контейнер-ориентированную архитектуру (container-based architecture). В такой архитектуре двоичный код выполняется программным контейнером и изолируется от деталей конкретного аппаратного обеспечения. Примеры всем знакомы – Java и C#. В обоих языках двоичный код не зависит от специфики аппаратного обеспечения и выполняется виртуальной машиной.

Конечно, есть и другой способ достигнуть независимости от аппаратного обеспечения на уровне двоичного кода. Как один из вариантов, можно использовать эмулятор аппаратного обеспечения, так же известный как QEMU. Он позволяет эмулировать, например, устройство с процессором ARM на платформе x86 и так далее. Google могла бы использовать C++ как язык для разработки приложений внутри эмуляторов. Действительно, Google использует такой подход в своих эмуляторах Android, которые построены на основе QEMU.

Очень хорошо, что они не пошли по такому пути, поскольку тогда кому-то пришлось бы запускать ОС на эмуляторе, требующем намного больше ресурсов, и, как итог, скорость работы снизилась бы. Для достижения наилучшего быстродействия эмуляция была оставлена только там, где этого нельзя было избежать, в нашем случае – в Android-приложениях. Как бы то ни было, компания Google пришла к решению использовать Java как основной язык разработки приложений и среды их выполнения. Это было критически важное архитектурное решение, которое поставило Android в стороне от остальных мобильных ОС на основе Linux, представленных в настоящее время. Возьмем для примера MeeGo. Она использует C++ и фреймворк Qt, несмотря на то, что Qt кроссплатформенный, необходимость делать разные сборки для разных платформ не исчезает.

Выбрав Java, нужно было решить, какую виртуальную машину (JVM) использовать. Ввиду ограниченности ресурсов использование стандартной JVM

было затруднено. Единственным возможным выбором было использование Java ME JVM, разработанной для мобильных устройств. Однако счастье Google было бы неполным без разработки собственной виртуальной машины, и появилась Dalvik VM.

Dalvik VM отличается от других виртуальных Java-машин следующим:

1. Она использует специальный формат DEX для хранения двоичных кодов, в противовес форматам JAR и Pack200, которые являются стандартом для других виртуальных Java-машин. Компания Google заявила, что бинарники DEX меньше, чем JAR;

2. Dalvik VM оптимизирована для выполнения нескольких процессов одновременно;

3. Dalvik VM использует архитектуру, основанную на регистрах против стековой архитектуры в других JVM, что приводит к увеличению скорости выполнения и уменьшению размеров бинарников;

4. Она использует собственный набор инструкций (а не стандартный байткод JVM);

5. Возможен запуск (если необходимо) нескольких независимых Android-приложений в одном процессе;

6. Выполнение приложения может охватывать несколько процессов Dalvik VM «естественным образом». Для поддержи этого добавлено:

a. Специальный механизм сериализации объектов, основанный на классах Parcel и Parcelable. Функционально преследуются те же цели, что и Java Serializable, но в результате данные имеют меньший объем и потенциально более терпимы к версионным изменениям классов;

b. Особый способ для выполнения вызовов внутри процессов (inter process calls, IPC), основный на Android Interface Definition Language (AIDL).

7. До Android 2.2 Dalvik VM не поддерживала JIT-компиляцию, что было серьѐзным ударом по производительности. Начиная с версии 2.2, скорость выполнения часто используемых приложений заметно возросла.

Приложения в Android состоят из слабосвязанных компонентов, которые собираются воедино с помощью программного манифеста. Манифест – файл, описывающий все компоненты приложения и способы их взаимодействия, а также метаданные, в том числе требования к платформе и аппаратной конфигурации.

Компоненты, перечисленные ниже – кирпичики, с помощью которых вы можете строить свои приложения:

1. Активности. Уровень представления. Каждый экран приложения будет наследником класса Activity. Активности используют Представления для формирования графического пользовательского интерфейса, отображающего

информацию и взаимодействующего с пользователем. С точки зрения разработки под настольные платформы Активность – эквивалент Формы (Form);

2. Сервисы. Невидимые двигатели вашего приложения. Сервисные компоненты работают в фоновом режиме, запуская уведомления, обновляя Источники данных и видимые Активности. Используются для регулярных операций, которые должны продолжаться даже тогда, когда Активности вашего приложения не на переднем плане;

3. Источники данных. Хранилища информации. Данные компоненты нужны для управления базами данных в пределах одного приложения и предоставления к ним доступа извне. Источники данных задействуются при обмене информацией между разными программами. Это значит, что вы можете настраивать собственные объекты ContentProvider, открывая к ним доступ из других приложений, а также использовать чужие источники, чтобы работать с данными, которые открыли для вас внешние программы. Устройства под управлением Android содержат несколько стандартных Источников, которые предоставляют доступ к полезным базам данных, включая хранилища мультимедийных файлов и контактной информации;

4. Намерения. Система передачи сообщений между приложениями. Используя Намерения, вы можете транслировать сообщения на системном уровне или для конкретных Активностей или Сервисов. Тем самым диктуется необходимость выполнения заданных действий. После этого Android сам определит компоненты, которые должны обработать поступивший запрос;

5. Широковещательные приемники. Компоненты, принимающие транслируемые Намерения. Если вы создадите и зарегистрируете объект BroadcastReceiver, ваше приложение сможет отслеживать трансляцию Намерений, которые соответствуют заданным критериям. Широковещательные приемники автоматически запустят программу, чтобы она могла ответить на принятое Намерение. Благодаря этому данный механизм идеально подходит для создания приложений, использующих событийную модель;

6. Виджеты. Визуальные программные компоненты, которые можно добавлять на домашний экран. Этот особый вид Широковещательных приемников позволяет создавать динамические, интерактивные компоненты, которые пользователи могут встраивать в свои домашние экраны;

7.Уведомления. Система пользовательских уведомлений. Позволяет сигнализировать о чем-либо, не обращая на себя внимание или не прерывая работу текущей Активности. Механизм уведомлений лучше всего подходит для Сервисов и Широковещательных приемников, когда необходимо привлечь внимание пользователя. Например, принимая текстовое сообщение или

входящий звонок, устройство оповещает вас, мигая светодиодами, воспроизводя звуки, отображая значки или показывая сообщения. Вы можете инициировать все эти события из собственного приложения, используя уведомления.

Убрав зависимость между программными компонентами, вы можете делиться и обмениваться такими самостоятельными составляющими, как Источники данных, Сервисы и даже Активности, с другими приложениями (собственными и сторонними).

2.2.3 Уровень доступа к данным

Структурированные данные в Android хранятся с использованием двух механизмов:

1. Базы данных SQLite. Если нужно хранить управляемую, структурированную информацию, можете использовать библиотеку SQLite для работы с реляционными базами данных. Любое приложение может создавать свои собственные базы данных, над которыми оно будет иметь полный контроль;

2. ContentProvider. Источники данных предоставляют общий, строго определенный интерфейс для обмена данными.

2.3. SQLite

SQLite – это система управления реляционными базами данных, которая хорошо себя зарекомендовала. Ее основные характеристики:

- свободная;
- соответствует стандартам;
- легковесная;
- одноуровневая.

Данная система – это компактная библиотека, написанная на языке C, она составляет часть стека программного обеспечения, который поставляется вместе с Android.

Поскольку SQLite реализована в виде библиотеки, а не как отдельный исполняемый процесс, каждая база данных считается частью приложения, которое ее создало. Благодаря этому минимизируется число внешних зависимостей, уменьшаются задержки, упрощаются синхронизация и блокирование при выполнении транзакций.

SQLite зарекомендовала себя в качестве чрезвычайно надежной системы баз данных, которая используется во многих бытовых электронных

устройствах, включая некоторые MP3-проигрыватели, iPhone и iPod Touch.

SQLite – мощная и легковесная, отличается от многих обычных движков баз данных отсутствием типизации каждого столбца. Это значит, что значения в столбце не обязаны иметь один и тот же тип. Вместо этого каждое значение типизируется отдельно для каждой строки. В связи с этим нет необходимости проверять типы при занесении или извлечении данных в контексте каждого столбца в строке.

С помощью SQLite можно создавать для своего приложения независимые реляционные базы данных и использовать их для хранения и управления сложными, структурированными данными приложения.

По умолчанию все базы данных приватные, доступ к ним могут получить только те приложения, которые их создали. Стоит отметить, что наиболее удачные стандартные подходы по работе с базами данных применимы и к Android. В частности, когда вы создаете базы данных для устройств с ограниченными ресурсами (например, мобильных телефонов), важно нормализовать данные, чтобы уменьшить их избыточность.

2.4 Java

Java – объектно-ориентированный язык программирования. Программы на Java могут быть транслированы в байт-код, выполняемый на виртуальной java-машине (JVM) – программе, обрабатывающей байт-код и передающей инструкции оборудованию, как интерпретатор, но с тем отличием, что байт-код, в отличие от текста, обрабатывается значительно быстрее.

Java – это интерпретируемый и компилированный язык программирования. Исходный текст (файлы с расширением a Java) откомпилирован со справкой компилятора Java (javac), который преобразовывает исходный текст в байт-код (файлы с расширением a.class). Цель проектировщиков Java состояла в том, чтобы разработать язык, посредством которого программист мог записать код, который мог бы выполняться всегда, в любое время.

Три ключевых элемента объединились в технологии языка Java:

– Java предоставляет для широкого использования свои апплеты (applets) – небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы Web. Апплеты Java могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы HTML;

– Java сочетает простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты;

– Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода. Ключевая черта этих классов заключается в том, что они обеспечивают создание независимых от используемой платформы абстракций для широкого спектра системных интерфейсов. [3]

Основной причиной выбора данного языка стала его мобильность – независимость от платформы означает лёгкость переноса программы с одного компьютера на другой компьютер без каких-либо трудностей. Также Java – платформа, независима на обоих уровнях, то есть на первичном и на вторичном.

Java – это язык со строгим контролем типов, что означает, что мы должны объявлять тип для каждой переменной. И эти типы данных в Java одинаковы для всех платформ. Java имеет свои собственные библиотеки фундаментальных классов, которые облегчают запись кода для программиста, который может быть перемещен с одной машины на другую, без потребности перезаписи кода. Независимость от платформы на исходном уровне означает, что мы можем переместить наш исходный текст из одной системы в другую, компилируя код, и работая в системе. [4]

Язык Java активно используется для создания мобильных приложений под операционную систему Android. При этом программы компилируются в нестандартный байт-код, для использования их виртуальной машиной Dalvik (начиная с Android 5.0 Lollipop виртуальная машина заменена на ART). Для такой компиляции используется дополнительный инструмент, а именно Software Development Kit, разработанный компанией Google. [5]

Разработку приложений можно вести в среде Android Studio, NetBeans, в среде Eclipse, используя при этом плагин Android Development Tools (ADT) или в IntelliJ IDEA. Версия JDK при этом должна быть 5.0 или выше.

В моем случае использовалась среда разработки Android Studio так как она предоставляет исчерпывающий инструментария для разработки программного обеспечения с повышенными требованиями к дизайну.

2.5. JSOUP library

В процессе разработки приложения одной из самых сложных задач для меня стала разработка парсера. Учитывая большое количество сайтов, необходимо было оптимизировать скорость работы приложения, количество кода и расход интернет-трафика (а учитывая что за раз парсер должен собирать больше сотни статей – это сложная задача). Значительно упростить задачу мне помогло использование библиотеки JSOUP.

JSOUP – библиотека Java для работы с HTML. Данная библиотека обеспечивает удобный API для извлечения и обработки данных, используя DOM, CSS и методы jQuery.

2.6. Fragments in Android

Фрагмент (класс Fragment) представляет поведение или часть пользовательского интерфейса в операции (класс Activity). Разработчик может объединить несколько фрагментов в одну операцию для построения многопанельного пользовательского интерфейса и повторного использования фрагмента в нескольких операциях. Фрагмент можно рассматривать как модульную часть операции. Такая часть имеет свой жизненный цикл и самостоятельно обрабатывает события ввода. Кроме того, ее можно добавить или удалить непосредственно во время выполнения операции. Это нечто вроде вложенной операции, которую можно многократно использовать в различных операциях.

Фрагмент всегда должен быть встроен в операцию, и на его жизненный цикл напрямую влияет жизненный цикл операции. Например, когда операция приостановлена, в том же состоянии находятся и все фрагменты внутри нее, а когда операция уничтожается, уничтожаются и все фрагменты. Однако пока операция выполняется (это соответствует состоянию *возобновлена* жизненного цикла), можно манипулировать каждым фрагментом независимо, например добавлять или удалять их. Когда разработчик выполняет такие транзакции с фрагментами, он может также добавить их в стек переходов назад, которым управляет операция. Каждый элемент стека переходов назад в операции является записью выполненной транзакции с фрагментом. Стек переходов назад позволяет пользователю обратить транзакцию с фрагментом (выполнить навигацию в обратном направлении), нажимая кнопку *Назад*.

Когда фрагмент добавлен как часть макета операции, он находится в объекте ViewGroupвнутри иерархии представлений операции и определяет собственный макет представлений. Разработчик может вставить фрагмент в макет операции двумя способами. Для этого следует объявить фрагмент в файле макета операции как элемент <fragment> или добавить его в существующий объектViewGroup в коде приложения. Впрочем, фрагмент не обязан быть частью макета операции. Можно использовать фрагмент без интерфейса в качестве невидимого рабочего потока операции.

2.7 XML в Android

Android представляет собой платформу с открытым кодом для разработки приложений для мобильных устройств. С ее помощью можно получить доступ ко всем компонентам устройства, на котором выполняется эта ОС, начиная от низкоуровневого программирования графики и заканчивая использованием встроенной камеры. XML является распространенным форматом для обмена информацией в Интернете, поэтому велика вероятность, что он понадобится для доступа к данным в Web. Кроме того, XML может потребоваться для передачи данных, например, Web-сервису.

Спецификация XML описывает язык и ряд вопросов, касающихся кодировки и обработки документов. Материал этой секции представляет собой сокращённое изложение описания языка в Спецификации XML, адаптированное для настоящей статьи.

2.8 Android-studio

Android-studio – официальная среда разработки (IDE) для разработки Android-приложений. Основана на программном обеспечении IntelliJ IDEA от компании JetBrains. Android-studio предлагает следующие возможности:

- гибкая система сборки;
- расширенный редактор макетов WYSIWYG;
- способность работать с UI компонентами при помощи Drag-and-Drop;
- функция предпросмотра макета на нескольких конфигурациях экрана;
- рефакторинг кода;
- шаблоны основных макетов и компонентов Android;
- статический анализатор кода.

3 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

3.1 Разработка архитектуры системы

Создание программного продукта требует предварительного проектирования, построения структуры и планирования сроков. Только после утверждения плана разработки и выбора технологий и структуры программного продукта начинается его реализация.

Архитектура системы проектировалась в соответствии с моделью приложений для Java EE. Приложение разделено на модули в соответствии с функциональностью. Интеграция с внешними системами производится с использованием web-сервисов.

Проектируемая система состоит из нескольких крупных компонентов. Рассмотрим эти компоненты:

1. Элементы пользовательского интерфейса платформы Android (уровень представления);
2. Бизнес-логика приложений;
3. Компоненты для доступа к данным.

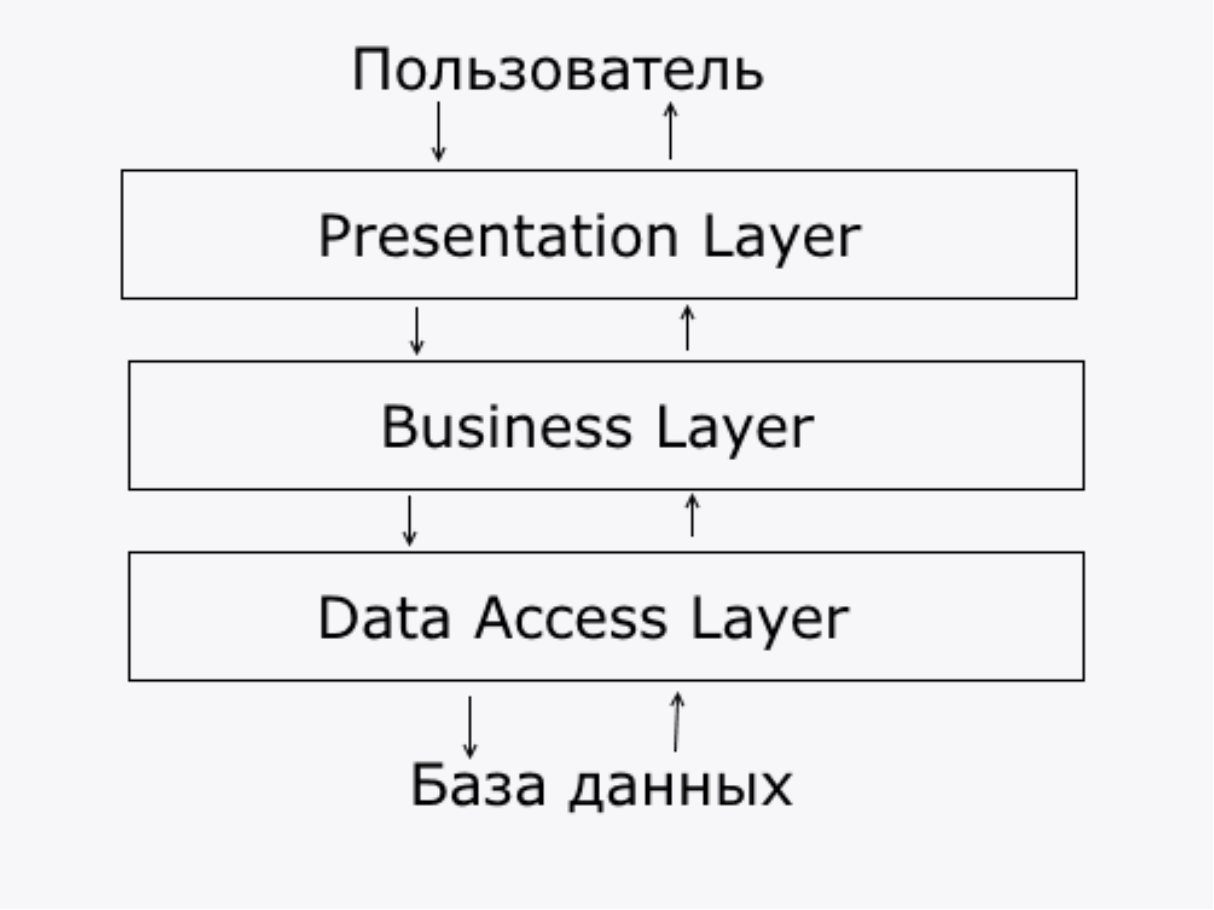

Рисунок 3.1 – Обзор архитектуры

Клиент – это интерфейсный (обычно графический) компонент, который представляет первый уровень, собственно приложение для конечного пользователя. Первый уровень не должен иметь прямых связей с базой данных (по требованиям безопасности), быть нагруженным основной бизнес-логикой (по требованиям масштабируемости) и хранить состояние приложения (по требованиям надежности). На первый уровень может быть вынесена и обычно выносится простейшая бизнес-логика: интерфейс авторизации, алгоритмы шифрования, проверка вводимых значений на допустимость и соответствие формату, несложные операции (сортировка, группировка, подсчет значений) с данными, уже загруженными на терминал.

Содержит набор компонентов, которые отвечают за обработку полученных от уровня представлений данных, реализует всю необходимую логику приложения, все вычисления, взаимодействует с базой данных и передает уровню представления результат обработки.

Уровень доступа к данным - хранит модели, описывающие используемые сущности, также здесь размещаются специфичные классы для работы с разными технологиями доступа к данным. Здесь также хранятся репозитории, через которые уровень бизнес-логики взаимодействует с базой данных.

Уровень доступа к данным не зависит от других уровней, уровень бизнес-логики зависит от уровня доступа к данным, а уровень представления – от уровня бизнес-логики.

Компоненты, как правило, должны быть слабосвязанными, поэтому неотъемлемым звеном многоуровневых приложений является внедрение зависимостей.

3.2 Проектирование слоя бизнес-логики

Слой бизнес-логики определяет совокупность правил, принципов и зависимостей поведения объектов предметной области. Вследствие этого одна из наиболее важных задач при разработки данного уровня – покрыть набор функциональных требований с учетом нефункциональных. Таким образом, архитектура создается непосредственно на основе требований, модели предметной области и моделей, полученных на этапе анализа.

Модель предметной области может содержать больше информации, чем необходимо для реализации поставленных требований. Поэтому БД, сделанная на основе модели предметной области, может иметь избыточную структуру, в то время как архитектура BLL, покрывая конкретный набор требований, предоставляет точные сведения о том, какие именно данные нужны для ее функционирования. То есть архитектура BLL задает минимальные границы базы данных.

Так же, реализация бизнес-функций требует наличия необходимого интерфейса для использования конечным пользователем данной функции. То есть, уровень бизнес-логики оказывает существенное влияния на структуру уровня представления.

По этим причинам, логичнее всего начать проектирование системы со слоя бизнес-логики.

Для реализация набора операций, связанных с данными, которыми может оперировать пользователь, уровень бизнес-логики должен содержать следующий набор операций:

– обработка финансовой информации;
– создание целей;
– обновление событий;
– анализ данных пользователя;
– работа с календарными событиями;
– отрисовка статистик по дням;
– визуализация данных.

Такие операции как обработка информации, обновление событий и работа с календарными событиями, отрисовка статистик, визуализация данных, не содержат сложной бизнес-логики. Они предполагают либо сохранение изменений в БД, либо получение данных из БД и передачу на уровень представления, который определяет как эти данные будут показаны пользователю.

Именно поэтому, с целью оптимизации и минимизации кода данные операции будут выполняться верхним слоем DAL, который предоставляет достаточно широкие возможности для работы с источником данных и при этом не зависит от конкретного поставщика БД. Реализация таких операций в слое бизнес-логики приведет к дублированию логики верхнего слоя, так как в ней будут присутствовать только частные операции, которые потребовались уровню представления в конкретный момент, и малейшее изменение в потребностях уровня представления приведет к необходимости дополнения или модификации реализации BLL.

3.3 Проектирование базы данных

Проектирование слоя доступа к данным включает проектирование источника данных и проектирование логики работы с ним.

Так как БД проектируется на основе предметной области и набора требований с учетом минимальных потребностей бизнес-логики, важно разработать схему БД, которая, с одной стороны, обладает полнотой и

завершенностью структуры с точки зрения предметной области, а с другой стороны является, по мере возможности, простой и удобной для дальнейшего расширения. Другой крайне важный момент – соблюдение практических рекомендаций по разработке структуры БД, таких как требования к первичным ключам, минимизация избыточности, создание индексов и др.

Схема реализации базы данных показана на рисунке 3.2.

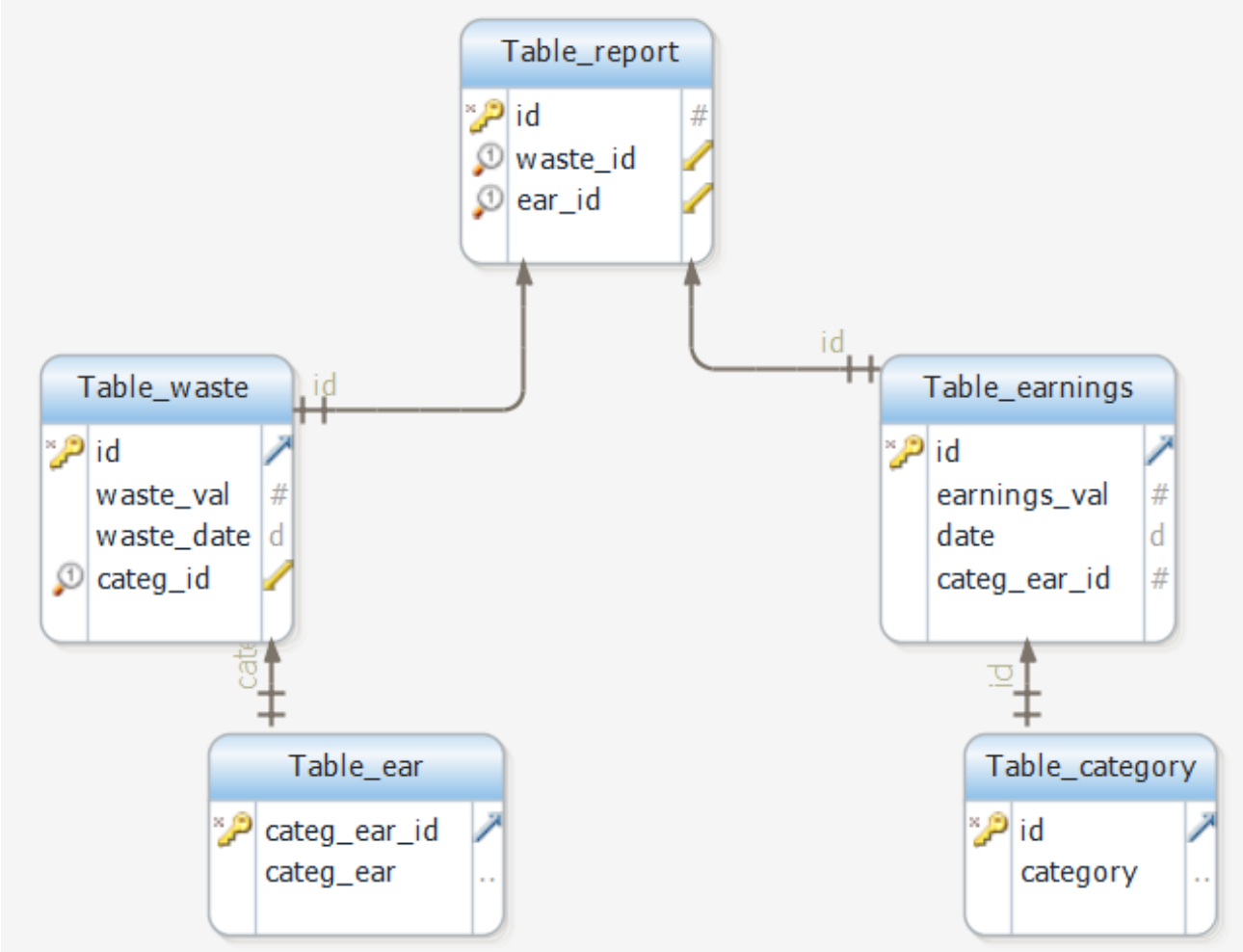

Рисунок 3.2 – схема базы данных

Учитывая медленную скорость работы баз данных SQLite, объемы хранимых данных необходимо свести к минимуму.

Для работы со значениями хранимыми в базах приложение собирает все данные из указанной таблицы и на уровне бизнес-логики выбирает все необходимые.

3.4 Проектирование слоя пользовательского интерфейса

Пользователи современных Android-приложений требовательны к дизайну и юзабилити программных продуктов. При разработке пользовательского интерфейса основной упор делался как на удобство пользования и красивый интерфейс, так и на информативное предоставление данных.

Пользовательский интерфейс включает в себя следующие страницы:

1. Главная страница – страница, отображающая общую сумму расходов и доходов, остаток до цели и рекомендуемые статьи;

2. Финансы – страница, содержащая информацию по всем расходам и кнопки для перехода на страницы «добавить расходы», «добавить доходы», «добавить ограничения». Данные страницы позволяет добавлять соответствующие значения в базу данных;

3. Статистика – отображение статистики по доходам и расходам. Страница реализована в двух вариантах – «отчет по дням» и «отчет по месяцам»;

4. Цели – страница с отображением основной информацией по цели. Содержит кнопку для добавления новой цели. В целях мотивации предоставляется возможность добавить только одну цель;

5. Статьи – страница содержащая подразделы по разным тематикам. Тематики, в свою очередь, делятся на страницу отдельных статей.

4 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

4.1 Разработка пользовательского интерфейса

Один из важных моментов в реализации приложения, от которого во многом зависит успех программного продукта - дизайн. Дизайн мобильных приложений – разновидность дизайна, в задачи которого входит проектирование пользовательских интерфейсов.

При подготовке дизайна моего приложения, были поставлены цели следовать основным тенденциям в области дизайна (минимализм, flat-дизайн, предоставление исчерпывающей информации на одной странице) но, при этом, сделать упор на удобство пользователя, простоту интерфейса и интуитивно понятную навигацию.

Учитывая особенности целевой аудитории (что было рассмотрено при анализе данных полученных методом социологического опроса) было решено выбрать нейтральные цвета. В дизайн-концепции продемонстрировано цветовое оформление мобильного приложения «финансовый помощник студента», расположение блоков, данных и других элементов.

Так же, с целью упрощения использования приложения были использованы элементы инфографики – интуитивно понятные изображения.

Дизайн-концепция приложения отображена на рисунках 4.1 и 4.2.

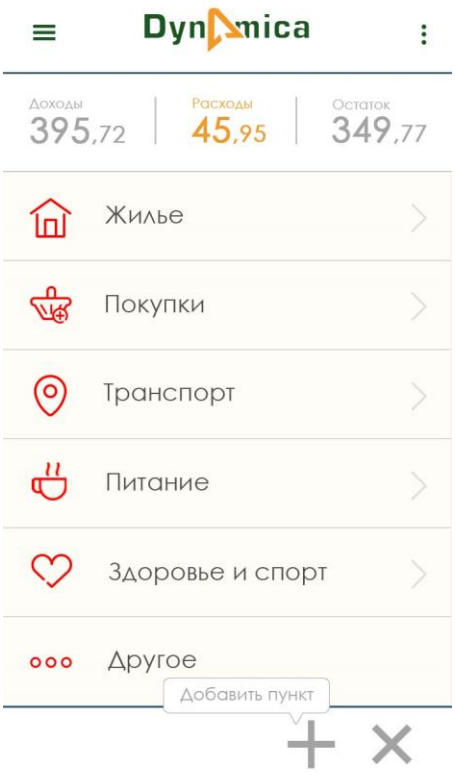

Рисунок 4.1 – дизайн-концепция страницы «финансы»

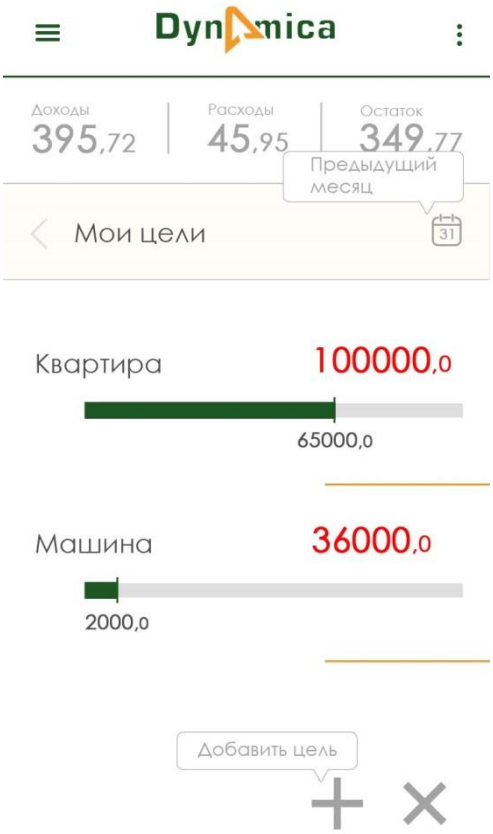

Рисунок 4.2 – дизайн-концепция страницы «цели»

При разработке программных средств для операционной системы Android есть несколько вариантов реализации уровня пользовательского интерфейса – представление страниц через activity, через HTML или через fragment [8].

В случае моего приложения была выбрана реализация через activity.

Все страницы реализованы на XML. Ниже, в качестве примера, приведен код главной страницы:

```
….<?xml version="1.0" encoding="utf-8"?>
….<RelativeLayout xmlns:android=http://schemas.android.com/apk/res/android
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   app:layout_behavior="@string/appbar_scrolling_view_behavior"
   tools:context="by.bsuir.dp10.MainActivity"
   tools:showIn="@layout/activity_main"
   android:background="#272727"
   >
   <FrameLayout
      android:layout_width="match_parent"
```

```
    android:layout_height="200dp"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:background="@drawable/bgim"
    android:id="@+id/frameLayout">
    <FrameLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_gravity="left|top"
        android:background="#9c2b2b2b">
    </FrameLayout>
</FrameLayout>
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="100dp"
    android:layout_below="@+id/frameLayout"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:background="#ffffff"
    android:id="@+id/frameLayout3">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text=""
        android:id="@+id/textView9"
        android:layout_gravity="right|center_vertical"
        android:textSize="30dp" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Всего заработано"
        android:id="@+id/textView16"
        android:layout_gravity="right|top" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Всего потрачено"
```

```
        android:id="@+id/textView15"
        android:layout_gravity="left|top" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:id="@+id/textView8"
        android:layout_gravity="left|center_vertical"
        android:textSize="30dp" />
</FrameLayout>
<FrameLayout
    android:layout_width="wrap_content"
    android:layout_height="150dp"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:background="#ffb300"
    android:id="@+id/frameLayout2">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Рекомендуется к прочтению:"
        android:id="@+id/textView18"
        android:layout_gravity="left|top" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Поиск работы"
        android:id="@+id/textView19"
        android:layout_gravity="center"
        android:textSize="26dp"
        android:textColor="#ffffff" />
</FrameLayout>
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="До цели осталось: "
        android:id="@+id/textView17"
        android:layout_above="@+id/frameLayout2"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/frameLayout3"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:gravity="center"
        android:textColor="#ffffff"
        android:textSize="26dp" />
</RelativeLayout>
```

4.2 Реализация меню и перелинковки страниц

Меню в приложении реализовано следующим образом:

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
```

- метод для обработки меню

```
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        //noinspection SimplifiableIfStatement
        if (id == R.id.finance) {
            Intent FinAct = new Intent(this, financeAc.class);
            startActivity(FinAct);
            return true;
        } else  if (id == R.id.information){
            Intent InfAct = new Intent(this, informationAc.class);
            startActivity(InfAct);
        } else  if (id == R.id.target){
            Intent TarAct = new Intent(this, targetAc.class);
            startActivity(TarAct);
```

```
        }else if (id == R.id.all){
            Intent all = new Intent(this, MainActivity.class);
            startActivity(all);
        }
        return super.onOptionsItemSelected(item);
    }
}
```

В Android studio для реализации переходов между страниц необходимо обрабатывать отдельное нажатие кнопки и, через метод onClick экземпляр класса Intent загружается новое activity [7]:

```
    @Override
public void onClick(View v) {
    switch (v.getId()){
        case R.id.addTargetOnActBtn:
        Intent intent = new Intent(this, targetAc.class);
            intent.putExtra("targetName", tarName.getText().toString());
            intent.putExtra("targetValue", tarVal.getText().toString());
        startActivity(intent);
        break;
        default:
        break;
    }
}
```

4.3 Парсер JSOUP

Одна из основных особенностей приложение – динамическое обновление данных. Обновление происходит при помощи парсера JSOUP [9]. Пример реализации парсера для сайта “shkolazhisni.ru”

```
    public class NewThread extends AsyncTask<String, Void, String>{
        @Override
        protected String doInBackground(String... params) {
            org.jsoup.nodes.Document doc;
            try {
                doc =
Jsoup.connect("http://shkolazhizni.ru/tag/%D1%84%D0%B8%D0%BD%D0%B0%
D0%BD%D1%81%D1%8B/").get();
                content = doc.select(".post_preview_desc");
```

```
            titleList.clear();
            for (org.jsoup.nodes.Element contents: content){
                titleList.add(contents.text());
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    return null;
    }
    @Override
    protected void onPostExecute(String result){
        infoWorkParseLW.setAdapter(adapter);
    }
  }
} [10]
```

Парсер собирает все данные с сайта в переменную doc. После сбора данных приложение отбирает данные из элементов HTML принадлежащих классу post_preview_desc. Таким образом, после выбора необходимых данных из класса post_preview_desc они выводятся на страницу статей по тематике «поиск работы/ карьера».

4.4 Алгоритм обучения

Одна из основных функций приложения – обучение пользователя основам финансовой грамотности. Обучение должно быть ненавязчивым и не вызывать негатива у пользователя. Перейти к информационным страницам можно из меню или из раздела главной страницы «рекомендуется к прочтению».

Раздел «рекомендуется к прочтению» заполняется на основании анализа пользовательских данных. В зависимости от объемов трат и доходов, от целей и лимитов выводится категория статей, которая может быть интересна пользователю. В настоящий момент предусмотрено 7 тематик статей:

- планирование бюджета;
- работа;
- бизнес;
- инвестиции;
- страхование;
- кредит;
- депозиты.

Реализация алгоритма показана на рисунке 4.3.

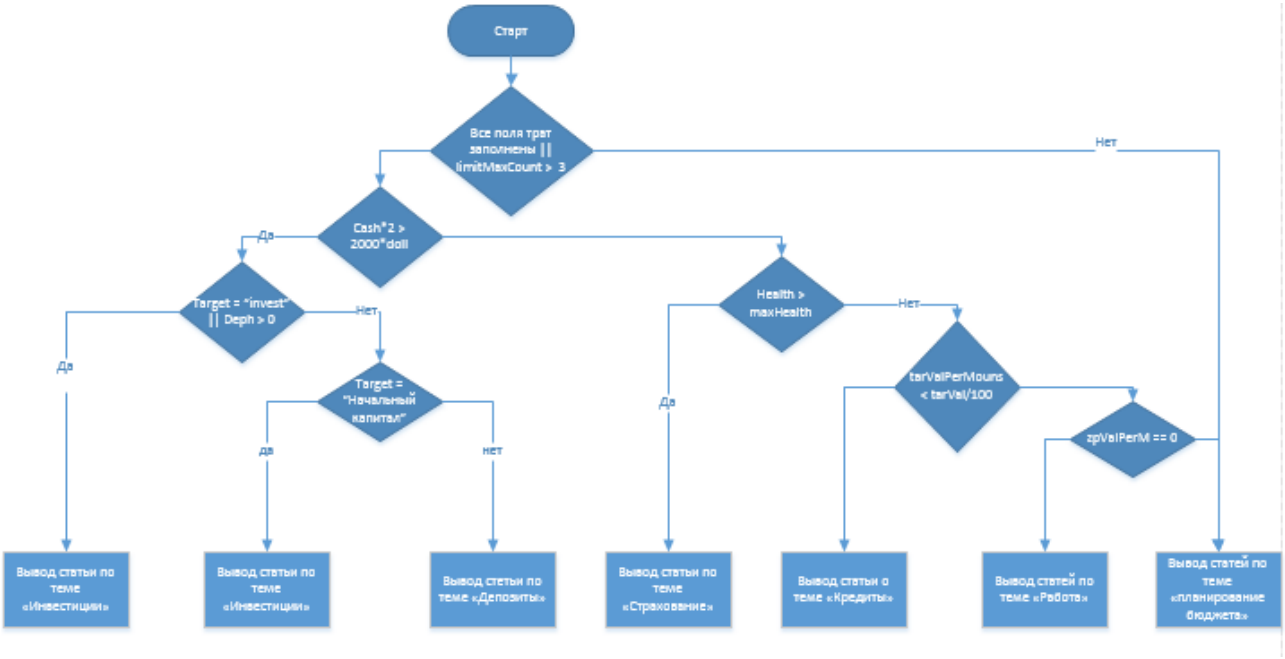

Рисунок 4.3 – алгоритм выбора предлагаемых статей

4.5 Реализация доступа к базе данных

Реализация взаимодействия приложения с базами данных (уровень данных) реализован в классе databaseHelper и представлена следующим кодом:

```
public DatabaseHelper(Context context) {
   super(context, DATABASE_NAME, null, 1);
   SQLiteDatabase db = this.getWritableDatabase();
}
@Override
public void onCreate(SQLiteDatabase db) {
   db.execSQL("create table " + TABLE_NAME + " ( " + ID + " INTEGER
PRIMARY KEY AUTOINCREMENT, " +
         WASTE + " INTEGER , " + CATEGORY + " TEXT, " + DATE + " TEXT );");
}
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
   db.execSQL("DROP TABLE  IF EXISTS" + TABLE_NAME);
   onCreate(db);
}
public boolean insertData(int Waste, String category, String date){
```

```
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put(WASTE, Waste);
    contentValues.put(CATEGORY, category);
    contentValues.put(DATE, date);
    long res = db.insert(TABLE_NAME, null, contentValues);
    if (res == -1)
        return false;
    else
        return true;
}
public Cursor getAllData(){
    SQLiteDatabase db = this.getWritableDatabase();
    Cursor res = db.rawQuery( "select * from " + TABLE_NAME, null);
    return res;
}
```

onCreate – метод для создания базы данных. Данный метод срабатывает в случае, если приложение впервые запущено.

onUpgrade – метод обновления базы данных. Срабатывает в случае изменения структуры базы данных.

insertData – метод добавления данных в таблицы. Вызывается при любом добвалении пользователем данных.

Cursor getAllData – собирает все данные из таблицы. Теоритически – это не самый правильный метод работы с базами данных, но учитывая тот факт, что SQLite достаточно медленная база данных, я решил собирать все данные и выбирать необходимые уже на уровне бизнес-логики [11].

5 ТЕСТИРОВАНИЕ СИСТЕМЫ

Тестирование мобильных приложений отличается от тестирования обычного ПО наличием ряда уникальных требований. Прежде всего, мобильные приложения должны правильно выполняться в любое время и в любом месте. Необходимо, чтобы они корректно функционировали на разных платформах, которые отличаются используемыми операционными системами, размерами экрана, вычислительными ресурсами и продолжительностью непрерывной работы от батарей. Мобильные приложения должны поддерживать множество каналов ввода (клавиатура, голос, жесты и т. д.), мультимедийные технологии и обладать другими особенностями, повышающими удобство их использования. Для удержания низких цен на оборудование необходимо широко использовать средства моделирования и виртуализации. И наконец, поскольку большинство мобильных сервисов поддерживают достаточно широкий спектр беспроводных сетей (2G, 3G, 4G, Wi-Fi, WiMax), мобильные приложения должны нормально функционировать в неоднородной сетевой среде.

С учетом указанных требований при тестировании мобильных приложений необходимо сосредоточиться на следующих целях и мероприятиях:

- тестирование функциональности и поведения — позволяет оценить сервисные функции, интерфейсы API для мобильных веб-приложений, поведение внешних систем, интеллектуальность системы и пользовательские интерфейсы;
- тестирование QoS – позволяет оценить нагрузку на систему, производительность, устойчивость и готовность, масштабируемость и пропускную способность;
- тестирование интероперабельности – дает возможность проверить способность системы работать в условиях различных устройств, платформ, браузеров и беспроводных сетей;
- тестирование удобства использования и интернационализации – позволяет сделать оценку контента пользовательского интерфейса, потоков и сценариев операций пользователей, применения мультимедийных средств и управления при помощи жестов;
- тестирование безопасности и конфиденциальности – осуществляется для проверки процедур аутентификации пользователей, безопасности устройств, безопасности сеансов работы, возможности проникновения в системы и сети, соблюдения безопасности сквозных транзакций и конфиденциальности пользовательской информации;

- тестирование мобильности – позволяет оценить работу функций, использующих информацию о местоположении, профили клиентов, системные и пользовательские данные;

- тестирование совместимости и связности – позволяет проверить совместимость с мобильными браузерами и платформами и оценить возможность диверсификации соединений беспроводных сетей.

Для создания среды тестирования мобильных приложений применяется несколько различных стратегий.

В случае реализации дипломного проекта использовалось тестирование на основе эмуляции. Этот способ тестирования предусматривает использование эмулятора мобильного устройства, имитирующего его поведение в виртуальной машине. Довольно часто такие эмуляторы бывают включены в состав комплекта инструментов для разработчика, прилагаемого к мобильной платформе. В дипломном проекте использовался эмулятор Genymotion.

Один из способов контроля правильности работы приложения – это использование модульного тестирования (Unit тестирования). Модульное тестирование заключается в изолированной проверке каждого отдельного элемента путем запуска тестов в искусственной среде. Оценивая каждый элемент изолированно и подтверждая корректность его работы, установить проблему значительно проще чем, если бы элемент был частью системы.

Модульное тестирование также позволяет проводить рефакторинг, сохраняя корректную работоспособность модуля.

Поскольку некоторые классы могут использовать другие классы, тестирование отдельного класса часто распространяется на классы, связанные с ним. Например, класс бизнес-логики использует DAO класс. Для того чтобы не привязываться к реализации DAO класса для конкретной базы данных, DAO класс используется в виде интерфейса.

Каждый тест вызывает определенный метод, передает ему тестовые параметры, проверяет результат работы метода. В ходе такого вызова из-за наличия операторов условного перехода в самом методе, некоторые операции могут выполняться, а некоторые – нет. Качество теста определяется покрытием всех операторов вызываемого метода, а также покрытием ветвей выполнения программы.

Таблица 5.1 – Результат модульного тестирования приложения

Имя класса	Покрытие строк всех методов, %	Покрытие всех ветвей выполнения, %
DatabaseHelper	95	100
DataParsClass	91	95

Продолжение таблицы 5.1 – Результат модульного тестирования приложения

Имя класса	Покрытие строк всех методов, %	Покрытие всех ветвей выполнения, %
DataAnalyticsCl	89	99
OutInpUserData	100	100
TargetCalc	100	100

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

6.1 Ведение и исходные данные

Целью дипломного проекта явилась разработка мобильного приложения для удобного анализа доходов/расходов и повышения финансовой грамотности студентов. Разработка велась в интересах национального банка Республики Беларусь.

Экономический эффект достигается за счет экономии трудовых, материальных и финансовых ресурсов, а так же полученной прибыли. Для определения экономической эффективности рассчитываются капитальные затраты, зарплаты исполнителей, трудоемкость и чистая прибыль.

Применение программного продукта повысит уровень финансового образование выбранной целевой аудитории за счет использования инструментов сбора и анализа данных пользователя.

Приложение можно отнести к категории ПО организации вычислительного процесса и третьей категории сложности.

6.2 Расчет сметы затрат и цены программного продукта

Исходные данные для расчета сметы затрат и отпускной цены представлены в таблице 6.1.

Таблица 6.1. Исходные данные

Наименование показателей	Буквенные обозначения	Единицы измерения	Количество
Коэффициент новизны	$К_н$	единиц	0,9
Группа сложности	–	единиц	3
Дополнительный коэффициент сложности	$К_{сл}$	единиц	0,08
Поправочный коэффициент, учитывающий использование типовых программ	$К_т$	единиц	0,7
Установленная плановая продолжительность разработки	$Т_р$	лет	0,47
Продолжительность рабочего дня	$Т_ч$	ч	8
Тарифная ставка 1-го разряда	$Т_{м1}$	тыс. руб.	298
Коэффициент премирования	$К_п$	единиц	1,4

Продолжение таблицы 6.1 – Исходные данные

Наименование показателей	Буквенные обозначения	Единицы измерения	Количество
Норматив дополнительной заработной платы	$Н_д$	%	20
Отчисления в фонд социальной защиты населения	$З_{сз}$	%	34
Отчисления в Белгосстрах	$Н_{дс}$	%	0,6
Расходы на научные командировки	$Р_{нк}$	%	30
Прочие прямые расходы	$П_з$	%	20

Для определения объема ПС воспользуемся нормативными данными, приведенными в таблице 6.2:

Таблица 6.2 – Характеристика функций и их объем

Код функции	Наименование (содержание) функции	Объем функций (строк исходного текста)
101	Организация ввода информации	150
102	Контроль, предварительная обработка и ввод информации	450
109	Организация ввода/вывода информации в интерактивном режиме	320
201	Генерация структуры базы данных	4300
203	Формирование баз данных	2180
204	Обработка наборов и записей баз данных	2670
207	Манипулирование базами данных	200
208	Организация поиска и поиск в базе данных	970
506	Обработка ошибочных и сбойных ситуаций	410
703	Расчет показателей	460
707	Графический вывод результата	480
	Итого	12590

$$V_o = \sum_{i=1}^{n} V_i$$

где *Vo* — общий объем ПС;

Vi — объем отдельной функции ПС;

n — общее число функций.

Список функций, реализуемый ПС и их объем представлен в таблице 6.2. На основании этих данных получим:

Vo = 150 + 450 + 320 + 4300 + 2180 + 2670 + 200 + 970 + 410 +460 + 480 = 12590 (условных машинных команд).

6.3 Расчет нормативной трудоемкости

На основании общего объема ПС определим нормативную трудоемкость ($T_н$), которая устанавливается с учетом сложности ПС:

$T_н$ = 278 (человеко-дней).

То же с учетом поправочного коэффициента, учитывающего разработку программного средства с использованием современных ПЭВМ:

$T_н$ = 278 * 0,7 = 195 (человеко-дней).

С учетом дополнительного коэффициента сложности КСЛ рассчитаем общую трудоемкость ПС по формуле:

$$T_o = T_н \cdot K_{сл}$$

где T_o – общая трудоемкость ПС.

Подставив данные в формулу, получим:

T_o = 195 * 1.08 = 172 (человеко-дня)

Учитывая, что стадии «Технический проект» и «Рабочий проект» сведены в одну стадию – «Технический проект», то трудоемкость «Технорабочего проекта» определяется по формуле:

$$T_{трп} = 0{,}85 \cdot T_{тп} + 1 \cdot T_{рп}$$

где $T_{ТРП}$ – трудоемкость стадии «Технорабочий проект»; $T_{ТП}$ – трудоемкость стадии «Технический проект»; $T_{РП}$ – трудоемкость стадии «Рабочий проект».

Она составляет:

$\mathrm{T_{трп}} = 0{,}85 \cdot 53{,}07 + 1 \cdot 50{,}12 = 95{,}23$ человеко-дня.

На основе уточненной трудоемкости разработки ПС и установленного периода разработки рассчитаем общую плановую численность разработчиков:

$$Ч_р = \frac{T_{ут}}{T_{рд} \cdot Ф_{эф}}$$

где $Ч_Р$ – плановая численность разработчиков (чел.);

$Ф_{ЭФ}$ – годовой эффективный фонд времени работы одного работника в течение года (дней в год);

$T_{РД}$ – плановая продолжительность разработки ПС (лет).

Подставив данные в формулу, получим:

$Чр = 139 / (0{,}47 * 233) = 2$ человека.

Эффективный фонд времени работы одного работника ($Ф_{эф}$)

$Ф_{эф} = Д_г - Д_п - Д_в - Д_о,$

где $Д_г$ – количество дней в году;

$Д_п$ – количество праздничных дней в году;

$Д_в$ – количество выходных дней в году;

$Д_о$ – количество дней отпуска.

$Ф_{эф} = 366 - 6 - 103 - 24 = 233$ (дня в год)

При утверждении плановой численности разработчиков продолжительность разработки определяется по формуле:

$$T_p = \sum_{i=1}^{m} \frac{T_i}{Ч_{pi} \cdot Ф_{эф}},$$

где $T_р$ – срок разработки ПС (лет);

T_i – трудоемкость разработчиков ПС на i-й стадии (человеко-дней);

$Ч_{рi}$ – численность разработчиков ПС на i-й стадии (чел.);

$Ф_{эф}$ – годовой эффективный фонд времени работы одного работника в течение года (дней в год);

М – число стадий.

$$T_p = \frac{15{,}16}{2 \cdot 233} + \frac{36{,}01}{2 \cdot 233} + \frac{95{,}23}{2 \cdot 233} + \frac{20{,}85}{2 \cdot 233} \approx 0{,}46 \text{ года.}$$

Таблица 6.3 – Расчет уточненной трудоемкости ПС и численности исполнителей по стадиям

	Стадии						Итого
	ТЗ	ЭП	ТП	РП	ТРП	ВН	
Коэффициенты удельных весов (d_{cmi})	0,08	0,19	0,28	0,34	-	0,11	1
Коэф-т сложности	1,08	1,08	1,08	1,08	-	1,08	

Продолжение таблицы 6.3 – Расчет уточненной трудоемкости ПС и численности исполнителей по стадиям

	Стадии						Итого
Коэффициенты, учитывающие использование типовых программ (K_m)	-	-	-	0,7	-	-	-
Коэффициенты новизны ($K_н$)	0,9	0,9	0,9	0,9	-	0,9	-
Уточнённая трудоёмкость стадий ($T_у$), человеко-дней	15,16	36,01	53,07	50,12	95,23	20,85	167,25
Численность исполнителей ($Ч_р$), человек	5,0	5,0	-	-	5,0	5,0	5,0
Срок разработки, лет	0,04	0,09	-	-	0,28	0,05	0,46

6.4 Расчет основной заработной платы исполнителей

Реализацией проекта занимались два человека. В соответствии с численностью и выполняемыми функциями устанавливается штатное расписание группы специалистов-разработчиков.

Расчет основной заработной платы осуществляется в следующей последовательности.

Определим месячные *($Т_м$)* и часовые *($Т_ч$)* тарифные ставки начальника отдела (тарифный разряд – 16; тарифный коэффициент – 3,72) и инженера-программиста 1-й категории (тарифный разряд – 12; тарифный коэффициент – 2,84).

Месячную тарифную ставку исполнителя *($Т_м$)* определим путем умножения действующей месячной тарифной ставки 1-го разряда *($Т_{м1}$)* на тарифный коэффициент *($Т_к$)*, соответствующий установленному тарифному разряду:

$$T_м = T_{м1} * T_к$$

Часовую тарифную ставку рассчитаем путем деления месячной тарифной ставки на установленный при восьмичасовом рабочем дне, фонд рабочего времени – 168 часов.

$$T_ч = \frac{T_м}{Ф_р}$$

где $T_Ч$ – часовая тарифная ставка (руб.);

$T_М$ – месячная тарифная ставка (руб.).

Определим месячную и часовую тарифные ставки исполнителя – инженера-программиста первой категории:

$T_м = 298 \times 2{,}84 = 846{,}32$ тыс. рублей

$T_ч = 846{,}32/168 = 5{,}04$ тыс. рублей

Месячная и часовая тарифные ставки начальника отдела равны соответственно:

$T_м = 298 \times 3{,}72 = 1108{,}56$ тыс. рублей

$T_ч = 1108{,}56/168 = 6{,}6$ тыс. рублей

Расчет месячных и почасовых тарифных ставок сведен в таблицу 6.4.

Таблица 6.4 - Штатное расписание группы разработчиков

Должность	Количество ставок	Тарифный разряд	Тарифный коэффициент	Месячная тарифная ставка (тыс. руб.)	Часовая тарифная ставка (тыс. руб.)
Начальник отдела (ведущий инженер программист)	1,00	16	3,72	1108,56	6,6
Инженер-программист 1-ой категории	1,00	12	2,84	846,32	5,04

Основная заработная плата исполнителей рассчитывается по формуле

$$З_{оі} = \sum_{i=1}^{n} T_{чі} \cdot T_ч \cdot Ф_{эі} \cdot K$$

где n – количество исполнителей, занятых разработкой конкретного ПС;

$T_{Чi}$ – часовая тарифная ставка *i-го* исполнителя (руб.);

$Ф_{Эi}$ – эффективный фонд рабочего времени *i-го* исполнителя (дней);

$T_ч$ – количество часов работы в день (ч);

$K_п$ – коэффициент премирования.

Следовательно, сумма основной заработной платы составляет:

$$З_п = 6{,}6 \cdot 8 \cdot 110 \cdot 1{,}4 + 5{,}04 \cdot 8 \cdot 110 \cdot 1{,}4 = 13838{,}56 \text{ тыс. руб.}$$

Дополнительная заработная плата ($З_д$) определяется по нормативу в процентах к основной заработной плате по формуле:

$$З_д = \frac{З_д \cdot Н_д}{100}$$

где $Н_д$ – норматив дополнительной заработной платы в целом по научной организации, равный 20%.

Дополнительная заработная плата ($З_д$) составляет:

$$З_д = 13838{,}56 \cdot 20/100 = 2767{,}712 \text{тыс.руб.}$$

Таблица 6.5 - Расчет себестоимости и отпускной цены ПС

Наименование статей	Норматив	Методика расчета	Значение, тыс. руб.
Отчисления в фонд социальной защиты	$Н_{сз}$ = 34%	$З_{сз} = (З_о + З_д) \cdot Н_{зсз}/100$	5632,3
Отчисления в Белгосстрах	$Н_с$ = 0,6%	$Н_з = (З_о + З_д) \cdot Н_{нз}/100$	99,64
Материалы и комплектующие	—	$М = Н_м \cdot V_о/100$ $Н_м = 380$ руб./100 строк	47,84
Спецоборудование	—	$Р_с = Ц_с$	2000
Машинное время	—	$Р_м = Ц_м \cdot V_о/100 \times Н_{мв}$ $Ц_м = 600$ руб $Н_{мв} = 15$ машино–часов	1133,1
Расходы на научные командировки	$Н_{рнк}$ = 30%	$Р_{нк} = З_о \cdot Н_{рнк}/100$	4151,55
Прочие прямые расходы	$Н_{пз}$ = 20%	$П_з = З_о \cdot Н_{пз}/100$	2767,7
Накладные расходы	$Н_{рн}$ = 100%	$Р_н = З_о \cdot Н_{рн}/100$	13838,5

Продолжение таблицы 6.5 – Расчет себестоимости и отпускной цены ПС

Наименование статей	Норматив	Методика расчета	Значение, тыс. руб.
Полная себестоимость	—	$С_п = З_о + З_д + З_{сз} + Н_з + Н_м + Р_с + Р_м + Р_{нк} + П_з + Р_н$	46276,9
Прогнозируемая прибыль	$У_р = 25\%$	$П_{пс} = С_п \cdot У_р / 100$	11569,23
Прогнозируемая цена без налогов	—	$Ц_п = С_п + П_{пс}$	57846,13
Налог на добавленную стоимость	$Н_{дс} = 20\%$	$НДС = Ц_п \cdot Н_{дс} / 100$	11569,22
Прогнозируемая отпускная цена	—	$Ц_о = Ц_п + О_{мб} + О_{рб} + НДС$	71762,9
Освоение ПС	$Н_о = 20\%$	$Р_о = С_п \cdot Н_о / 100$	9255,38
Сопровождение ПС	$Н_с = 20\%$	$Р_с = С_п \cdot Н_с / 100$	9255,38

Заказчик оплачивает организации-разработчику всю сумму расходов по проекту, включая прибыль. После уплаты налогов из прибыли в распоряжении заказчика остается чистая прибыль от проекта. Ввиду того что ПО разрабатывалось для одного объекта, чистую прибыль можно считать в качестве экономического эффекта организации-разработчика от реализованного проекта.

6.5 Оценка экономической эффективности применения ПС у пользователя

Для расчета экономического эффекта применения нового ПС необходимы данные имеющегося внедренного на производстве аналога (базового варианта). Некоторые из показателей не могут быть получены (составляют коммерческую тайну либо защищены авторскими правами разработчиков). Поэтому расчет экономического эффекта будем производить, опираясь на известные данные показателей базового и нового варианта ПС. Показатели обоих вариантов приведены в таблице 6.6.

Таблица 6.6 - Исходные данные для расчета экономического эффекта

Наименование показателей	Обозначения	Единицы измерения	Значение показателя	
			Базовый вариант	Новый вариант
1	2	3	4	5
Капиталовложения, включая стоимость услуг по сопровождению и адаптации ПС	$К_{пр}$	тыс. руб	-	57846,13
Затраты на освоение ПС	$К_{ос}$	тыс. руб	-	9255,32
Затраты на сопровождение ПС	$К_{с}$	тыс. руб	-	9255,32
Затраты на укомплектование ВТ техническими средствами в связи с внедрением нового ПО	$К_{тс}$	тыс. руб.	-	3412
Затраты на пополнение оборотных фондов, связанных с эксплуатацией нового ПС	$К_{об}$	тыс. руб	-	925
Всего затрат:				80693,77
Время простоя сервиса, обусловленное ПО, в день	$П_1$,$П_2$	мин	35	5
Стоимость одного часа простоя	$С_{п}$	тыс. руб	50	50
Среднемесячная зарплата одного программиста	$З_{см}$	тыс. руб	977	977
Коэффициент начислений на зарплату	$К_{нз}$	-	2,1	2,1
Среднемесячное количество рабочих дней	$Д_{р}$	день	21	21
Количество типовых задач, решаемых за год	$З_{т1}$, $З_{т2}$	задача	8050	8050

Продолжение таблицы 6.6 - Исходные данные для расчета экономического эффекта

Наименование показателей	Обозначения	Единицы измерения	Базовый вариант	Новый вариант
Объем выполняемых работ	A_1, A_2	задача	8050	8050
Средняя трудоемкость работ в расчете на задачу	T_{c1}, T_{c2}	человеко-часов на задачу	1,5	0,2
Количество часов работы в день	$T_ч$	час	8	8
Ставка налога на прибыль	$H_п$	%	20	20

6.6 Расчет капитальных затрат

Общие капитальные вложения (K_o) заказчика (потребителя), связанные с приобретением, внедрением и использованием ПС, включают в себя затраты на приобретение, освоение ПС, а также затраты на доукомплектацию техническими средствами в связи с внедрением нового ПС, а также затраты пополнение оборотных средств, в связи с использованием нового ПС.

Таблица 6.7 - Расчет капитальных затрат

Наименование	Методика расчета	Значение (тыс. руб.)
Затраты пользователя на приобретение ПС по отпускной цене разработчика	$K_{\text{пр}}$	71762,9
Затраты на освоение ПС	$K_{\text{ос}}$	9255,38
Затраты на сопровождение ПС	$K_{\text{с}}$	9255,32
Затраты на укомплектование ВТ техническими средствами в связи с внедрением нового ПО	$K_{\text{тс}}$	3412
Затраты на пополнение оборотных средств в связи с использованием нового ПС	$K_{\text{об}}$	925

Продолжение таблицы 6.7 – Расчет капитальных затрат

Наименование	Методика расчета	Значение (тыс. руб.)
Общие капитальные вложения	$K_{\text{о}} = K_{\text{пр}} + K_{\text{ос}} + K_{\text{с}} + K_{\text{тс}} + K_{\text{об}}$	94610,6
Экономия затрат на заработную плату в расчете на 1 задачу	$\text{С}_{\text{зе}} = \text{З}_{\text{см}} \times (\text{Т}_{\text{с1}} - \text{Т}_{\text{с2}}) / (\text{Т}_{\text{ч}} \times \text{Д}_{\text{р}})$	7,5
Экономия на заработную плату при использовании нового ПС	$\text{С}_{\text{з}} = \text{С}_{\text{зе}} \times \text{А}_2$	60375
Экономия с учётом начисления на заработную плату	$\text{С}_{\text{н}} = \text{С}_{\text{з}} \times \text{К}_{\text{нз}}$	126787,5
Экономия за счёт сокращения простоя сервиса	$\text{С}_{\text{с}} = (\text{П}_1 - \text{П}_2) \times \text{С}_{\text{П}} \times \text{Д}_{\text{рг}}/60$	5825
Общая годовая экономия текущих затрат, связанных с использованием нового ПС	$\text{С}_{\text{ос}} = \text{С}_{\text{н}} + \text{С}_{\text{с}}$	132612,5

Внедрение нового ПС позволит пользователю сэкономить на текущих затратах.

6.7 Расчет экономического эффекта

Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль – дополнительная прибыль, остающаяся в его распоряжении ($\Delta\text{П}_\text{ч}$), которая определяется по формуле:

$$\Delta\text{П}_\text{ч} = \text{С}_{\text{ос}} - \text{С}_{\text{ос}} \cdot \frac{\text{Н}_\text{п}}{100},$$

где $\text{Н}_\text{п}$ – ставка налога на прибыль, равная 20%.

Тогда чистая прибыль будет равна:

$$\Delta\text{П}_\text{ч} = 132612{,}5 - 132612{,}5 * 0{,}2 = 106090 \text{ тыс. руб.}$$

В процессе использования нового ПС чистая прибыль в конечном итоге возмещает капитальные затраты. Однако полученные при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени – расчетному году (за расчетный год принят 2016 год) путем умножения результатов и затрат за каждый год на коэффициент приведения (α_t), который рассчитывается по формуле:

$$\alpha_t = (1 + E)^{\,t_p - t},$$

Где Е – норматив дисконтирования разновременных затрат и результатов;

t_p – расчетный год, $t_p = 1$;

t – номер года, результаты и затраты которого приводятся к расчетному (2016 – 1, 2017 – 2, 2018 – 3, 2019 – 4). [6]

Коэффициентам приведения (α_t) по годам будут соответствовать следующие значения:

$\alpha_1 = (1 + 0{,}15)^{1-1} = 1{,}000$ – расчётный год;

$\alpha_1 = (1 + 0{,}15)^{1-2} = 0{,}869$ – 2017 год;

$\alpha_1 = (1 + 0{,}15)^{1-3} = 0{,}756$ – 2018 год;

$\alpha_1 = (1 + 0{,}15)^{1-4} = 0{,}658$ – 2019 год;

Таблица 6.8 – Расчет экономического эффекта от использования нового программного средства

Показатели	Ед. измерения	2016	2017	2018	2019
Результаты:					
Прирост прибыли за счет экономии затрат ($П_ч$)	тыс. руб.	-	106090	106090	106090
Прирост прибыли с учетом фактора времени	тыс. руб.	-	92192	80204	69807
Затраты:					
Приобретение ПС ($К_{пр}$)	тыс. руб.	71762,9	-	-	-
Освоение ПС ($К_{ос}$)	тыс. руб.	9255,38	-	-	-
Затраты на укомплектование ВТ техническими средствами в связи с внедрением нового ПО	тыс. руб.	3412	-	-	-
Сопровождение ПС	тыс. руб.	9255,38	-	-	-

Продолжение таблицы 6.8 – Расчет экономического эффекта от использования нового программного средства

Показатели	Ед. измерения	2016	2017	2018	2019
Пополнение оборотных средств ($К_{об}$)	тыс. руб.	925	-	-	-
Всего затрат	тыс. руб.	94610,66	-	-	-
То же с учетом фактора времени	тыс. руб.	94610,66	-	-	-
Экономический эффект:					
Превышение результата над затратами	тыс. руб.	-94610,66	92192	80204	69807
То же с нарастающим итогом	тыс. руб.	-94610,66	-2418,66	77785,34	147592,34
Коэффициент приведения	единиц	1,000	0,869	0,756	0,658

6.8 Выводы по технико-экономическому обоснованию

На основании проведенных расчетов и полученных результатом, можно сказать, что данный проект, то есть разработка и применение данного программного продукта, является экономически выгодной, так как, несмотря на значительные капиталовложения, уже через два года после внедрения данного ПС, предприятие – заказчик покрывает затраты на его приобретение и начинает получать дополнительную прибыль в размере 92192 тыс.руб. Внедрение данного средства значительно снижает затраты по расчету и начислению заработной платы работникам (при сравнении с ручным трудом).

В свою очередь исполнитель так же получает прибыль в размере 1156922 тыс.руб за реализацию продукта. Хотелось бы сказать, что доходы от ПС могут быть не только как оплата со стороны заказчика, но и как выплаты по партнерским программам и выплаты за рекламу, что делаем мой проект экономически выгодным и целесообразным.

ЗАКЛЮЧЕНИЕ

В настоящем документе описан полный цикл разработки мобильного приложения на основе 3-х уровневой архитектуры – от анализа предметной области до тестирования.

В качестве платформы для разработки использовалась платформа Android. В настоящее время данная платформа является одной из ведущих технологий в области разработки мобильного программного обеспечения во всем мире. Она построена для того, чтобы решать задачи будущего, которые становятся все более актуальными в настоящем, – задачи по созданию гибких, расширяемых и надежных IT-инфраструктур крупных компаний.

Результатом дипломного проектирования явилась разработка мобильного приложения – финансовый помощник студента. Благодаря использованию приложения появляется возможность грамотно планировать бюджет и ставить цели для достижения результатов. Так же, система анализирует данные и рекомендует статьи к прочтению.

Использование разработанного приложения позволяет повысить уровень финансовой грамотности населения. Под финансовой грамотностью понимается не только основы, представленные Национальным Банком Республики Беларусь, но и статьи общего плана – поиск работы, открытие своего дела, информация по страховке и инвестициям.

Так же, в ходе выполнения работы были получены новые знания и структурированы старые знания полученные за время обучения

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Майер Р. Android 2. Программирование приложений для планшетных компьютеров и смартфонов. – ЭКСМО, 2011. – 672 с.

[2] Голощапов А.Л. Google Android: программирование для мобильных устройств. – СПб.: БХВ-Петербург, 2010. – 448 с.

[3] К. Хорстманн, Г. Корнелл Java 2. Бибилиотека профессионала. Том 1. Основы. – Вильямс, 2011. – 816 с.

[4] The Java EE 5 Tutorial / Sun Microsystems Inc. – Santa Clara, CA 95054, 2008. – 1060 с.

[5] Философия Java. Библиотека программиста. – Питер, 2012. – 640

[6] Технико-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Проекты программного обеспечения/ В.А. Палицын. – Минск: БГУИР, 2006. – 76 с.

[7] Android Developers [Электронный ресурс] - Электронные данные. - Режим доступа: http://developer.android.com/index.html

[8] J2EE Design Patterns Repository [Электронный ресурс] - Электронные данные. - Режим доступа: http://www.theserverside.com/patterns

[9] Eclipse IDE [Электронный ресурс] - Электронные данные. - Режим доступа: http://www.eclipse.org/

[10] Официальный блог разработчиков Android [Электронный ресурс] – Электронные данные. - Режим доступа: http://android-developers.blogspot.com/

[11] Блог разработчиков [Электронный ресурс] – Электронные данные. – Режим доступа http://stackoverflow.com/