Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

**Отчёт по преддипломной практике**

Выполнил студент гр. 753503 Попов В. А.

Руководитель практики от предприятия: начальник отдела Калиновский А. А.

Руководитель практики от университета: доцент кафедры информатики Удовин И. А.

Минск 2021

СОДЕРЖАНИЕ

[1. Обзор предметной области. Постановка задачи. 3](#_Toc69687947)

[1.1. Менеджер финансов 3](#_Toc69687948)

[1.2. RSS-Feed 3](#_Toc69687949)

[1.3. Облачное хранение данных 5](#_Toc69687950)

[1.3.1. Файловое хранилище 9](#_Toc69687951)

[1.3.2. Блочное хранилище 10](#_Toc69687952)

[1.3.3. Объектное хранилище 10](#_Toc69687953)

[1.3.4. База данных 11](#_Toc69687954)

[1.4. Обзор существующих аналогов 12](#_Toc69687955)

[1.4.1. Monefy 12](#_Toc69687956)

[1.4.2. Coinkeeper 12](#_Toc69687957)

[1.4.3. Toshl 13](#_Toc69687958)

[1.4.4. Money Manager 14](#_Toc69687959)

[1.5. Постановка задачи 14](#_Toc69687960)

[1.6. Перспективы развития программного средства 15](#_Toc69687961)

[2. Используемые технологии 16](#_Toc69687962)

[2.1. Язык разметки HTML 16](#_Toc69687963)

[2.2. Каскадные таблицы стилей CSS 17](#_Toc69687964)

[2.3. Язык программирования JavaScript 19](#_Toc69687965)

[2.4. Программная платформа Node.js 21](#_Toc69687966)

[2.5. Облачная база данных Firebase 22](#_Toc69687967)

[2.6. Git / GitHub 23](#_Toc69687968)

[3. Технико-экономическое обоснование эффективности разработки и использования персонального менеджера 27](#_Toc69687969)

[3.1. Описание функций, назначения и потенциальных пользователей программного обеспечения 27](#_Toc69687970)

[3.2. Расчёт затрат на разработку ПО 28](#_Toc69687971)

[3.3. Оценка эффекта от продажи программного обеспечения 30](#_Toc69687972)

[3.4. Расчёт показателей эффективности инвестиций в разработку программного обеспечения 32](#_Toc69687973)

[3.5. Выводы по технико-экономическому обоснованию 33](#_Toc69687974)

1. **Обзор предметной области. Постановка задачи.**
   1. **Менеджер финансов**

Учет личных финансов – первый шаг на пути к богатству. Так считают бизнесмены и консультанты по финансовой грамотности. Но собирать чеки и вносить суммы в гроссбух, переносить в памяти цифры из магазина в компьютер долго и скучно. Записывать все в тетрадку у кассы – неловко. Поэтому проблему решают мобильные приложения и веб-сервисы.

Такое приложение должно предоставлять возможность:

* Разделять бюджет на доходы и расходы
* Разделять доходы/расходы по категориям
* Разделять доходы/расходы по временному периоду
* Устанавливать лимит на расход личного бюджета и уведомлять пользователя, когда он к нему приближается
* Предоставлять итоговую статистику по пользовательским транзакциям в виде графиков
* Иметь интуитивно понятный интерфейс
  1. **RSS-Feed**

RSS ленты сегодня одна из лучших возможностей держать руку на пульсе событий. Например, подборка последних новостей поможет быть в курсе дел вашей ниши или хобби. Не нужно «лопатить» десятки ресурсов, чтобы найти актуальную информацию, достаточно подписаться на RSS каналы и получить сортированные по категориям новости.

Данный формат отлично подходит для инфопорталов, где нет высокой вовлеченности аудитории в материал. То есть новости не комментируют, не лайкают, не делятся в соцсетях. RSS поток – это ознакомление с информацией. Что с ней делать дальше решает сам пользователь: идти на блог и как-то реагировать на статью, просто принимать данные к сведению или пропускать «мимо ушей». В 90% случаев формат RSS успешно используется порталами, блогами (в качестве анонсирования новых материалов), библиотечными ресурсами, досками объявлений.

Для начала разберемся, что такое RSS технически. Изнутри этот файл включает в себя три блочные структуры:

* Метаданные – сюда входят заголовки, подзаголовки, изображения, описания, основной текст.
* Гиперссылки на целевой ресурс для ознакомления с полным вариантом новости.
* Категории статей и метаданных, объединенных по тематике публикаций.

Принцип работы: как только на целевом ресурсе/источнике появляется новая информация, скрипт создает RSS файл, который считывают онлайновые или программные агрегаторы и оповещают пользователей об изменениях.

RSS начал с намерения распространять заголовки новостей. Потенциал для RSS значительно больше и может использоваться в любой точке мира.

Рассмотрите возможность использования RSS для следующего:

* Новые дома — риэлторы могут предоставлять обновленные каналы новых списков домов на рынке.
* Вакансии — Размещение фирм и газет могут предоставить классифицированную подачу вакансий.
* Предметы аукциона — Продавцы аукциона могут предоставлять каналы, содержащие предметы, которые были недавно добавлены на eBay или другие сайты аукциона.
* Распространение прессы — Список новых выпусков.
* Школы — Школы могут передавать домашние задания и быстро объявлять об отмене.
* Новости и объявления — заголовки, уведомления и любой список объявлений.
* Развлечения — списки последних телевизионных программ или фильмов в местных театрах.

RSS растет в популярности. Причина довольно проста. RSS — это бесплатный и простой способ продвижения сайта и его контента без необходимости рекламы или создания сложных партнерств по обмену контентом.

Говорить о плюсах и минусах использования необходимо в разрезе возможностей для пользователя и владельца сайта.

Для пользователя преимущества следующие:

* RSS ленты собирают всю информацию с подключенных источников в единый поток, который удобно быстро просматривать без лишних телодвижений.
* RSS полностью бесплатная опция. Выгода сайтов в новых заинтересованных читателях. Если анонс новости интересен и актуален, то продолжение захотят прочитать на источнике.
* Быстрота уведомлений. Сигнал о выходе новой публикации или новости сразу же передается пользователю. Это позволяет быть в курсе событий и вовремя реагировать на изменения.
* Для мобильного интернета – экономия трафика. Выгоднее подключить RSS, чем открыть десяток страниц в браузере.

Для сайта преимущества следующие:

* Рост трафика. Что такое RSS лента для сайта – это клики по ссылкам в новостях, переходы на ресурс и повышение вовлеченности пользователей. Соответственно продвижение в поисковой выдаче.
* Популярность ресурса. Связка: полезный контент + RSS повышает узнаваемость сайта среди целевой аудитории.
* Теплая ЦА. Прочитав интересный анонс новости, человек в 80% случаев перейдет на сайт, чтобы продолжить ознакомление.

Недостатки для ресурсов и пользователей:

* Переизбыток информации – бич нашего времени. Часто пользователи подписываются на множество лент, которые как-то соответствует тематике их интересов. В итоге информация «бьет ключом», а обработать её и, тем более, как-то отреагировать не хватает времени.
* Поисковики раньше индексируют RSS, чем сам первоисточник. Результат такой «прыти» краулеров плачевен: сайту присваивается статус малополезного ресурса со всеми вытекающими последствиями (понижение в выдаче, фильтры). Решают это проблему так: сокращают новости или статьи до формата анонса и мотивируют пользователей кликнуть по ссылке для перехода на источник-оригинал.
* Кража контента. XML язык считается самым простым и удобным для парсинга. Трастовые ресурсы могут спокойно находить уникальный контент и выдавать его за свой. Даже наличие ссылки не спасает от воровства, а если её и вовсе нет – то доказать статус первоисточника проблематично.

Технология RSS выгодна и для владельцев сайтов, и для интернет-пользователей. Благодаря RSS-подписке читатели могут собрать в одном сервисе все свои любимые сайты и быть в курсе публикации новых статей, не заходя на каждый из них самостоятельно. Более того, богатый функционал позволяет пользователям настраивать чтение новостей так, как они пожелают.

В свою очередь, вебмастера благодаря RSS получают постоянную целевую аудиторию, которой интересен контент, публикуемый на сайте. Все это может сказывать положительно на популярности ресурса.

* 1. **Облачное хранение данных**

Отличительной чертой нашего времени является постоянный рост объема деловой информации. Дизайнеры, маркетологи, копирайтеры, представители IT-профессий, а также компании, работающие с огромными массивами данных, постоянно нуждаются в надежном месте, в котором можно было бы хранить ценные файлы. Если раньше их держали на дискетах, флешках и компакт-дисках, то сейчас лучше всего отправлять их в облачные хранилища.

В общем и целом, облачное хранилище - это специально выделенное место на серверах, куда любой пользователь может закачать различные документы: текстовые файлы, любимые аудиозаписи и видеоролики, картинки, гиф-картинки, переписку из мессенджеров и многое другое. При этом серверы могут находиться где угодно: в Европе, Азии или Северной Америке.

Механизм облачного хранилища очень прост: нужно установить клиентское приложение и зарегистрироваться в нем. После чего можете спокойно сбрасывать в «облако» любую информацию, обмениваться ей с коллегами, обновлять ее, просматривать и так далее. Доступ к нему можно получить с любого устройства и из любого места, в котором есть Интернет.

У модели две стороны: клиент и поставщик услуги (провайдер). Клиент арендует место на серверах провайдера, сохраняет на них документы, приложения, статический контент сайтов, получая доступ к ним удаленно. Поставщик организует хранение, обслуживание, безопасность и доступ к данным. Эта модель имеет несколько преимуществ:

1. Экономия. Для собственной СХД понадобятся: помещение, стойки, серверы, охлаждение, оборудование для инфраструктуры. Потребуется организовать резервное копирование с покупкой ПО и дополнительных накопителей. А еще все это должны обслуживать администраторы, например, проводить периодические профилактические работы с отключениями и обновлениями. В случае с облачным хранилищем большинство операционных расходов сокращаются, а капитальные отсутствуют.
2. Надежность. Облачные хранилища обслуживаются инженерами со специализированным опытом в эксплуатации систем такого типа. Администраторы провайдеров регулярно обновляют железо, улучшают ПО, работают над безопасностью. При этом данные хранятся «с запасом»: для хранения 1 Гб данных клиента поставщик резервирует 2 Гб. Серверы часто распределены по нескольких городам или странам, что добавляет отказоустойчивости при форс-мажорах.
3. Безопасность.
   1. Обычно ЦОД — это режимное, круглосуточно охраняемое здание с видеонаблюдением, системами контроля и учета доступа. Внутри стоят системы охлаждения, пожаротушения и резервного питания, которые задублированы для надежности.
   2. Второй — установка и настройка прав доступа учетных записей, мониторингом, шифрованием во время загрузки, чтении и хранении данных. Не считая дополнительных услуг поставщиков по защите.
4. Доступность. Данными можно управлять через графические интерфейсы, консоль или API.
5. Масштабирование**.**Объем быстро увеличивается за счет подключения дополнительных серверов и СХД. В физическом варианте это также быстро, но только если заранее позаботится о масштабировании. А еще будет дорого и «vendor lock-in». Это значит, что вы будете привязаны к поставщику (вендору) и его платформе, технологиям, ПО. Когда захотите сменить поставщика, придется строить все заново, потому что все железо и ПО завязано на вендоре.
6. Управление расходами. Платить нужно ровно столько, сколько потреблять ресурсов. В объектных хранилищах есть классы — стандартное, «холодное», «ледяное». Классы помогают управлять стоимостью хранения. Например, когда к данным нужно часто обращаться, можно платить дороже за хранение, но дешевле за трафик (обращения). Для архивов наоборот — можно платить за работу с файлами дороже, но за хранение дешевле, потому что к ним редко обращаются.
7. Бизнес-процессы упрощаются, когда облачное хранилище доступно для сотрудника, например, из дома на выходных. А еще не забываем о восстановлении данных, когда бизнес-процессы не прерываются форс-мажорами из-за потери документов или репозитория.

Облачное хранилище во многих случаях может стать хорошей альтернативой традиционным решениям по хранению в корпоративной системе (on-premise). Однако, во многих случаях резервирование файлов в облаке имеет некоторые минусы:

1. **Зависимость от Интернет-соединения.**Если оно нарушается, файлы в облаке становятся недоступными. Важным фактором остается доступная полоса пропускания: даже при самом быстродействующем хранилище доступ к данным будет медленным из-за низкой скорости соединения. Особенно это касается мобильных сетей.
2. **Зависимость от провайдера.** Если у провайдера происходят какие-то проблемы, или он волюнтаристски меняет условия контракта, заказчик может поменять провайдера, но это процесс не одномоментный.
3. **Безопасность.** Пересылка данных за файерволл корпоративной сети – это всегда риск. Не все провайдеры предоставляют услугу шифрования хранимых данных. Несмотря на то, что хорошие провайдеры всегда стараются обеспечить высший уровень безопасности своих систем, инфраструктура провайдера – желанная цель для атак хакеров.
4. **Защита данных.** Как данные будут защищены в инфраструктуре провайдера – основной вопрос, который необходимо выяснить при заключении контракта на облачное хранение данных. Это, однако, «палка о двух концах», похожая на вечный спор о том, где лучше хранить деньги – в банке или дома в сейфе. И там, и там их могут украсть. Однако не подлежит сомнению, что, в целом, банк может обеспечить более высокую степень защиты средств своих вкладчиков. Однако, в отличие от денежной аналогии, компрометация информации, хранящейся у провайдера – это невосполнимый ущерб для клиента облачной услуги: при похищении денег клиента банк, в принципе, способен компенсировать этот ущерб.

Хранение в облаке используется:

* Для хранения массивных данных, например, видеозаписей с камер видеонаблюдения.
* В качестве репозиториев контента, например, публичных баз данных, школ дистанционного образования или мультимедиа ресурсов.
* Для хранения массивов данных Big Data, «Интернета вещей» и машинного обучения.
* Крупные СМИ интегрируют облака в цепочки поставки контента, например, для архивации или хранения для последующей аналитики.
* Для хранения данных игровых платформ, вроде Google Stadia.
* Видеохостинги или фотостоки используют хранилища для потоковой раздачи контента.
* В качестве хостинга интернет-магазинов, порталов, блогов и других статических сайтов.
* Для микросервисов: облачные хранилища поддерживают контейнеризацию, изоляцию процессов и совместный доступ.

Но чаще встречается пять сценариев.

1. Резервное копирование и восстановление. Большинство файловых систем облаков совместимы с базами данных, поэтому хранилища часто используют для резервирования, например, при обновлениях. Резервирование в облаке проще настроить, при этом надежность хранения данных лучше, потому что провайдер услуги распределяет копии по ЦОДам.
2. Разработка ПО и тестирование. Часто разработка требует дублирования сред, которые потом нужно удалять, и совместной работы. Использование облачных ресурсов для этого — стандартная практика среди разработчиков ПО. Также, облака интегрируются с разными приложениями без дополнительных «костылей».
3. Совместный доступ. Например, для команд разработки и тестирования из разных офисов или городов. Если данные хранятся на сервере внутри сети предприятия, часто нужен VPN. Но можно обойтись без этого и перенести часть общих файлов, к которым обычно и нужен доступ, в облачное хранилище.
4. Миграция данных в облако облегчает обслуживание своей инфраструктуры, но это серьезная задача, требующего многолетнего опыта у системного администратора. Однако есть сервисы, облегчающие этот процесс.
5. Big Data и IoT. Например, для Big Data массив данных в 100 Терабайт не так уж много, но держать на локальных серверах такой объем дорого, поэтому для этого часто используют облака. Хранить в «облаке» массивы удобно: в облачных сервисах обычно высокая пропускная способность, низкие задержки, и возможность настроить запросы не извлекая данные.

Поскольку данные бывают разные, то и хранить их лучше в подходящих для этого местах. По типу организации облачные хранилища делятся на:

* файловые,
* блочные,
* объектные,
* базы данных.
  + 1. **Файловое хранилище**

В основе файловой системы лежит иерархическая структура: корневая запись, от которой отходят данные о файлах и их атрибутах. Все они, в свою очередь, организованы в удобную структуру каталогов – зная имя того или иного документа, доступ к нему можно получить, щелкнув мышью по его имени. С ними можно осуществлять любые операции – открывать, изменять, переименовывать, удалять, копировать, перемещать в другую папку.

Файловое хранилище может быть двух видов: физическим и виртуальным. В первом случае данные сохраняются на жестком диске, во втором – на виртуальном. Последний имеет намного больший объем чем жесткий, а еще туда можно настроить удаленный доступ. В качестве примера можно привести Dropbox, «Облако Mail.Ru», «Google Диск», «Яндекс. Диск» и другие аналогичные им сервисы.

**Преимущества:**

* Простая и понятная структура.
* В таком хранилище легко ориентироваться, искать нужные документы.

**Недостатки:**

* Ограниченность в объеме, по мере заполнения которого падает скорость доступа, а вместе с ней и производительность.

**Для чего подходит:**для работы с небольшими объемами разных данных.

* + 1. **Блочное хранилище**

В блочном хранилище структура размещения та же, но все попадающие туда файлы делятся системой на блоки, каждому из которых присваивается свой идентификатор. С его помощью система собирает файлы в случае надобности.

**Преимущества:**

* Каждая пользовательская среда находится отдельно, за счет чего можно рассортировывать данные и обеспечить отдельный доступ к ним.
* БХ обеспечивает повышенную производительность: благодаря хост-адаптеру шины, который разгружает процессор и освобождает его ресурсы для выполнения других задач.

**Недостатки:**

* Оно дороже, и им трудно управлять, поскольку работа с блоками создает дополнительную нагрузку на базу данных.
* Оно, как и файловое, ограничено в объеме.

**Для чего подходит:**для работы с корпоративными базами данных

* + 1. **Объектное хранилище**

Это самый популярный тип хранилища. Вместо файловой системы в нем есть плоское пространство, состоящее из множества объектов, каждый из которых состоит из идентификатора и метаданных. Идентификатор – это присвоенный адрес, в роли которого выступает 128-битное число. Зная его можно без труда найти нужный файл. Метаданные (информация о файле) – его имя, размер, координаты и другая информация.

Объектные хранилища бывают частными или публичными. В первом случае оно создается в частном облаке, во втором – облако берут в аренду у провайдера публичных облаков.

**Достоинства:**

* Возможность работы с колоссальным объемом информации. Общий объем данных, хранящихся в Haystack Facebook, оценивается в 357 петабайт.
* Возможность хранения резервных копий данных, особенно тех, от которых зависит жизнедеятельность системы (например, файлы для аварийного восстановления).
* Возможность проверки корректности файлов и обеспечения быстрого доступа к ним.

**Недостатки:**

* Сложно называть объекты.
* Во многих объектных хранилищах отсутствует интерфейс для загрузки и управления файлами.

**Для чего подходит:** для хранения больших данных, текстовых документов, изображений, медиафайлов, переписок и многого другого.

* + 1. **База данных**

База данных – это совокупность определенной информации, хранящаяся в строго установленном порядке на физических или виртуальных носителях. Она управляется специальной программой под названием СУБД (Система Управления Базами Данных). СУБД позволяет обрабатывать любые тексты, графику, медиа; с ними можно делать все что угодно: хранить, анализировать, тестировать продукты и обновления, запускать новые проекты.

Она очень хорошо подходят для постоянных типовых операций. Например, туда записывается информация о заказах, поступающих в интернет-магазин, на основе которой приложение автоматически выписывает счет на оплату. Примером такой базы может стать нереляционная высокопроизводительная СУБД Redis, она хранит данные в оперативной памяти.

Базы данных могут находиться либо на сервере, либо в облаке. Облачные СУБД сегодня являются самыми популярными в своей области. Согласно исследованиям Market Realist, их используют 35% респондентов, экспериментируют с ними 14%, планируют внедрение – 12%.

**Преимущества:**

* Облачные базы данных имеют практически неограниченный объем хранения.
* Есть функция резервного копирования.
* Они обладают высоким внешним и внутренним уровнем безопасности, который обеспечивается техническими средствами и экспертами.
* Поддержка многозадачного и многопользовательского режимов.

**Недостатки:**

* Сложность управления, что требует затрат на соответствующий персонал и ПО.
* В случае нахождения их на физическом носителе имеют ограниченный объем, так что может потребоваться увеличение дискового пространства.
* Высокая стоимость разработки и эксплуатации.

**Для чего подходят:**для управления однородными массивами данных.

* 1. **Обзор существующих аналогов**

Рассмотрим существующие аналоги программных средств для мониторинга и анализа микроклиматических условий и выявим их преимущества и недостатки, чтобы в последствие учесть их при составлении технических требований к разрабатываемому продукту. Источники информации об аналогах – магазины мобильных приложений Play Market и App Store, а также веб-сайты из поисковых систем Google и Yandex.

* + 1. Monefy

Monefy – крайне простое и удобное приложение для тех, кто хочет тщательно следить за расходами. Траты распределяются по категориям, а для каждой категории можно присвоить собственную иконку. Расходы отображаются в виде диаграммы. В приложении есть встроенный калькулятор и нет рекламы.



Рисунок 1.4.1 - интерфейс Monefy

* + 1. Coinkeeper

У приложения необычный интерфейс, напоминающий монетницу. Контролировать расходы и доходы можно простым перетаскиванием монет из кошелька в расходные статьи. Облачная синхронизация поможет семейным парам вести общий бюджет на разных устройствах.

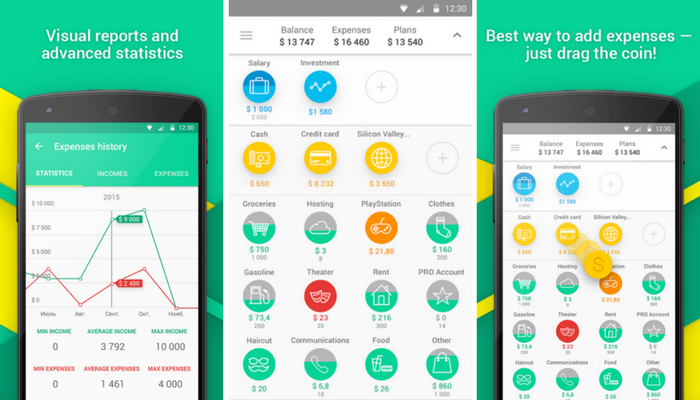


Рисунок 1.4.2 - интерфейс Coinkeeper

* + 1. Toshl

Самое важное в Toshl это простота использования. Вам не нужно каждый раз вбивать полное название траты, достаточно воспользоваться тегами. Например, все напитки пометить тегом «drink», а траты на мобильный — «mobile». Такая система значительно упрощает ввод данных и существенно сокращает время, нужное для этого.

Во вкладке «Бюджет» вы добавляете все ваши доходы. Здесь же отображается прогресс-бар с остатками средств. Так же можно выбрать на что выделен тот или иной «бюджет».

При желании можно экспортировать всю статистику в отдельный файл формата .cvs, .xls или .pdf. Иногда программа присылает забавные уведомления вроде «Я заметил, что вы много денег тратите в выходные, обратите на это внимание» (к сожалению, скриншота нет, так как они не всегда появляются). Выглядит это забавно и ненавязчиво. При этом продуктивность таких советов довольно высока.

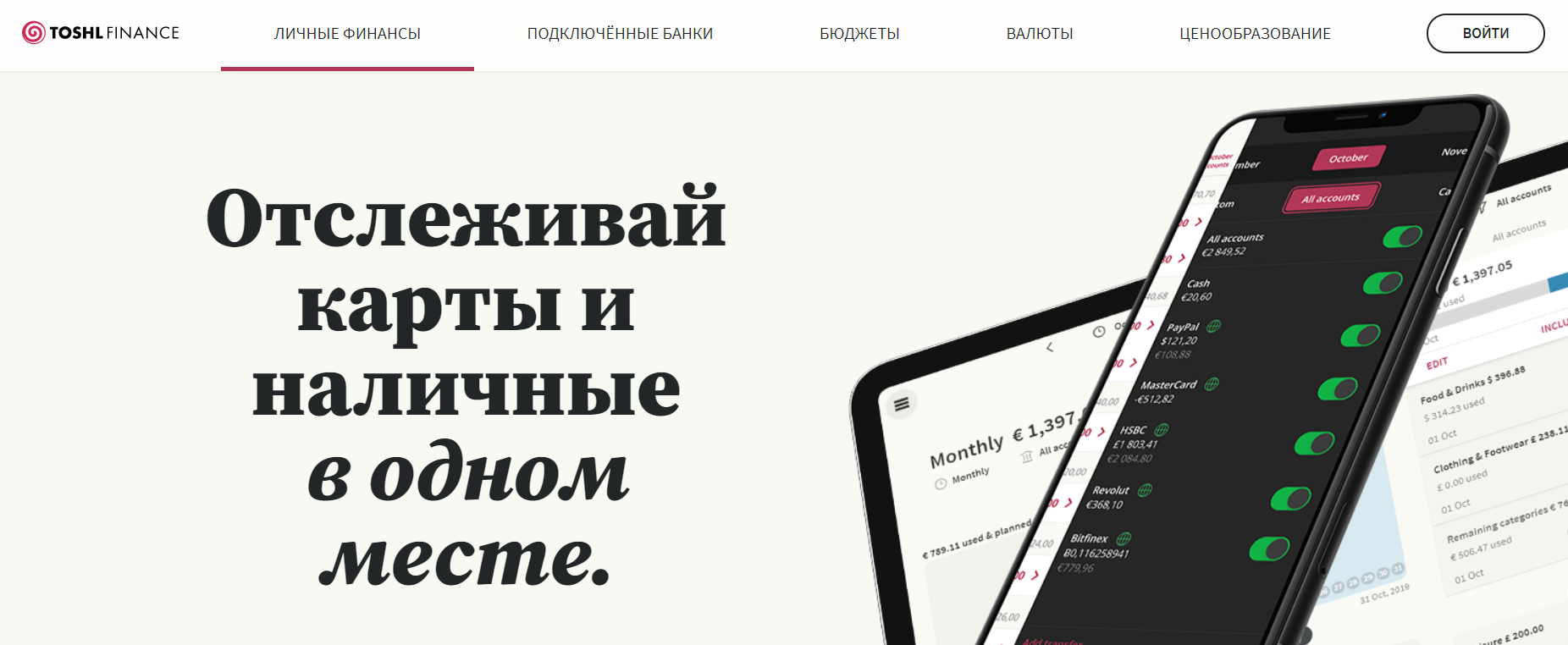


Рисунок 1.4.3 – интерфейс Toshl

* + 1. Money Manager

Money Manager - одно из наиболее функциональных приложений, с помощью которого можно не только держать под контролем свои траты, но и получать статистику за любой период времени. Здесь вы можете управлять кредитными и дебетовыми картами, получать статистику и отслеживать на графиках состояние активов. Кроме того, приложение имеет следующие особенности:

* система двойной записи;
* формирование бюджета по выбранным категориям;
* доступ с ПК;
* защита доступа паролем;
* перевод средств между активами;
* отслеживание расходов и доходов по активам;
* встроенный калькулятор;
* поиск по категориям.

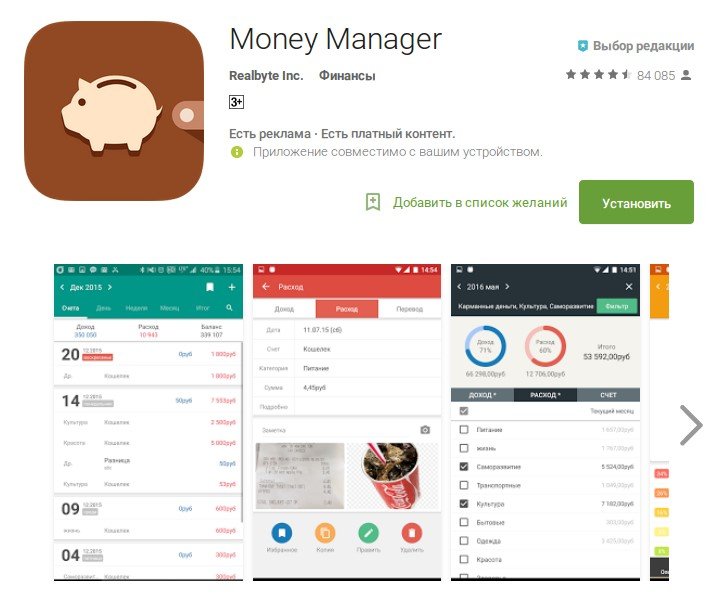


Рисунок 1.4.4 - интерфейс Money Manager

* 1. **Постановка задачи**

В рамках данного проекта поставлена задача разработать веб-сервис для анализа статистики личных доходов/расходов, добавления заметок и напоминаний, а также отображения статей новостных веб-сайтов в виде RSS-ленты.

Потенциальной целевой аудиторией программного продукта можно считать частных лиц, заинтересованных в удобном отслеживании своего личного бюджета, заметок и напоминаний, просмотра ленты новостей, а также юридических лиц, заинтересованных в качественном ведении бухгалтерского учёта.

При разработке программного обеспечения были поставлены следующие задачи:

* Разработать систему, реализующую деятельность персонального менеджера
* Спроектировать хранилище данных, способное оперировать достаточно большими объёмами данных
* Создать дружелюбный и интуитивно понятный пользовательский интерфейс
  1. **Перспективы развития программного средства**

Разрабатываемый продукт представляет собой веб-сервис, использующий реальные данные пользователя и имеющий реальное практическое применение. Однако, в перспективе, веб-сервис будет дополнен мобильным приложением под мобильные операционные Android и iOS. Таким образом продукт станет масштабируемым для разных платформ. В дополнение к этому, будет добавлена функция парсинга смс, приходящих от банков, для последующего автоматического разделения по категориям доходов и расходов.

1. **Используемые технологии**
   1. **Язык разметки HTML**

HTML (Hypertext Markup Language) - это код, который используется для структурирования и отображения веб-страницы и её контента. Например, контент может быть структурирован внутри множества параграфов, маркированных списков или с использованием изображений и таблиц данных.

HTML был изобретён [Тимом Бернерсом-Ли](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D1%80%D0%BD%D0%B5%D1%80%D1%81-%D0%9B%D0%B8,_%D0%A2%D0%B8%D0%BC), физиком из исследовательского института ЦЕРН в Швейцарии. Он придумал идею интернет-гипертекстовой системы.

Hypertext означает текст, содержащий ссылки на другие тексты, которые зрители могут получить немедленно. Он опубликовал первую версию HTML в 1991 году, состоящую из 18 тегов HTML. С тех пор каждая новая версия языка HTML появилась с разметкой новых тегов и атрибутов (модификаторов тегов).

Согласно Справочнику [HTML Element Reference](https://developer.mozilla.org/ru/docs/Web/HTML/Element) от Mozilla Developer Network, в настоящее время существует 140 тегов HTML, хотя некоторые из них уже устарели (не поддерживаются современными браузерами).

Самым большим обновлением языка стало внедрение **HTML5** в 2014 году. Было добавлено несколько новых семантических тегов к разметке, которые показывают смысл их собственного контента, например **<article>, <header>** и**<footer>.**

HTML-документы — это файлы, которые заканчиваются расширением **.html**или**.htm**. Вы можете просматривать его с помощью любого веб-браузера (например, Google Chrome, Safari или Mozilla Firefox). Браузер читает HTML-файл и отображает его содержимое, чтобы пользователи интернета могли его просматривать.

Каждая HTML-страница состоит из набора **тегов** (также называемых **элементами**), которые вы можете назвать строительными блоками веб-страниц. Они создают иерархию, которая структурирует контент по разделам, параграфам, заголовкам и другим блокам контента.

Большинство элементов HTML имеют открытие и закрытие, в которых используется синтаксис <tag></tag>*.*

Рассмотрим элемент абзаца более подробно.



Рисунок 2.1.1 – элемент абзаца

Главными частями нашего элемента являются:

1. Открывающий тег (Opening tag): Состоит из имени элемента (в данном случае, "p"), заключённого в открывающие и закрывающие угловые скобки.  Открывающий тег указывает, где элемент начинается или начинает действовать, в данном случае — где начинается абзац.
2. Закрывающий тег (Closing tag): Это то же самое, что и открывающий тег, за исключением того, что он включает в себя косую черту перед именем элемента. Закрывающий элемент указывает, где элемент заканчивается, в данном случае — где заканчивается абзац. Отсутствие закрывающего тега является одной из наиболее распространённых ошибок начинающих и может приводить к странным результатам.
3. Контент (Content): Это контент элемента, который в данном случае является просто текстом.
4. Элемент(Element): Открывающий тег, закрывающий тег и контент вместе составляют элемент.

Элементы также могут иметь атрибуты, которые выглядят так:



Рисунок 2.1.2 – атрибут элемента

Атрибуты содержат дополнительную информацию об элементе, которую вы не хотите показывать в фактическом контенте. В данном случае, class это *имя* *атрибута,* а editor-note это *значение атрибута*. Класс позволяет дать элементу идентификационное имя, которое может позже использоваться, чтобы обращаться к элементу с информацией о стиле и прочих вещах.

Атрибут всегда должен иметь:

1. Пробел между ним и именем элемента (или предыдущим атрибутом, если элемент уже имеет один или несколько атрибутов).
2. Имя атрибута, за которым следует знак равенства.
3. Значение атрибута, заключённое с двух сторон в кавычки.
   1. **Каскадные таблицы стилей CSS**

CSS – это язык, с помощью которого описывается внешний вид документа HTML, XML, XHTML. Название означает «каскадная таблица стилей», или Cascading Style Sheets. CSS-стили незаменимы при оформлении страниц сайтов: в одном файле содержатся сведения об отображении всех элементов документа.

По сути, таблица стилей – это файл, где описывается, как будет выглядеть каждый из элементов на странице. В HTML-документе, таким образом, остается только структура странички: сами блоки, их содержимое и расположение. Создать страницу и оформить ее можно и без использования таблиц, прописывая визуальные свойства каждого элемента в его описании.

Но, если страниц сотни и тысячи, применять такой метод неудобно: при изменении оформления приходится менять множество документов, вдобавок это загромождает верстку. Поэтому использование CSS считается золотым стандартом оформления сайтов: так получилось благодаря гибкости и многообразию возможностей каскадных таблиц.

Преимущества CSS:

* Это существенно упрощает верстку и снижает временные затраты. Один созданный файл стилей можно распространить на множество страниц, так что внешний вид элементов достаточно описать один раз.
* Если что-то нужно изменить, достаточно внести правки в один файл. Это касается и изменений в оформлении, и найденных ошибок.
* Применение CSS серьезно облегчает структуру документа, что хорошо и для пользователей, и для поисковых программ.
* Вариативность оформления становится шире. CSS поддерживает намного больше возможностей, чем имеется при использовании чистого HTML, вдобавок к одной странице можно применить несколько стилей в зависимости от обстоятельств (размер монитора пользователя, устройство, с которого выполнен вход, – ПК или мобильное).
* Страницы начинают загружаться быстрее: браузер кеширует таблицу стилей при первом посещении сайта, при последующих подгружаются только данные, что намного быстрее.

Начало развития было положено в 1990-х, когда консорциум W3C решил, что технология, позволяющая разделять содержание и представление документов, необходима. Стандарт CSS1 появился в 1996 году и позволял изменять с помощью таблиц параметры шрифтов, цвета элементов, свойства блоков и текстов, такие как отступы и выравнивание. Длина и ширина блоков задавались там же. С развитием интернета появились новые уровни:

* CSS2. Стандарт расширил технические возможности, дал возможность работать с аудио и страничными носителями (например, при печати документов), включил в себя поддержку блочной структуры и генерируемого содержимого;
* CSS3. Еще более масштабное расширение, находится в разработке до сих пор, поддерживает сглаживание, градиенты, тени и анимацию, для этого не приходится использовать JavaScript;
* CSS4. Находится в разработке, новые модули пока доступны как черновики. Дополняют предыдущие версии новыми расширениями и значениями.

Файл CSS сводится к набору правил, описанных по определенному синтаксису. Правило состоит из селекторной части и блока объявлений: ими описываются всевозможные элементы страницы. Формат примерно таков: селектор { параметр: значение }.

Селекторы указывают, к каким элементам будут применяться те или иные параметры стиля. Пишутся в начале строки, по сути, являются названиями тегов, для которых справедливо правило. Среди особенности стоит выделить:

* Можно использовать любой тег, написанный латиницей.
* Если вариантов стиля для одного типа элементов несколько, используются так называемые классы. У одного тега их может быть несколько (применяются все стили, что описаны в таблице). Запись в этом случае выглядит так: тег.Класс { параметр: значение }.
* Есть возможность видоизменить только один конкретный элемент. Это делается с помощью идентификаторов – уникальных имен, которые можно присвоить элементам. Идентификатор будет использоваться как селектор.
  1. **Язык программирования JavaScript**

[JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript)  — это полноценный [динамический язык программирования](https://developer.mozilla.org/en-US/docs/Glossary/Dynamic_programming_language), который применяется к [HTML](https://developer.mozilla.org/en-US/docs/Glossary/HTML) документу, и может обеспечить динамическую интерактивность на веб-сайтах сайт (например: игры, отклик при нажатии кнопок или при вводе данных в формы, динамические стили, анимация). Его разработал Brendan Eich, сооснователь проекта Mozilla, Mozilla Foundation и Mozilla Corporation.

JavaScript сам по себе довольно компактный, но очень гибкий. Разработчиками написано большое количество инструментов поверх основного языка JavaScript, которые разблокируют огромное количество дополнительных функций с очень небольшим усилием. К ним относятся:

* Программные интерфейсы приложения ([API](https://developer.mozilla.org/en-US/docs/Glossary/API)), встроенные в браузеры, обеспечивающие различные функциональные возможности, такие как динамическое создание HTML и установку CSS стилей, захват и манипуляция видеопотоком, работа с веб-камерой пользователя или генерация 3D графики и аудио сэмплов.
* Сторонние API позволяют разработчикам внедрять функциональность в свои сайты от других разработчиков, таких как Twitter или Facebook.
* Также вы можете применить к вашему HTML сторонние фреймворки и библиотеки, что позволит вам ускорить создание сайтов и приложений.

Базовой особенностью этого языка отмечается то, что на него повлияли другие (Python, Java и др.) языки программирования с целью придания максимального комфорта JavaScript и лёгкости в понимании его теми пользователями, которые не имеют соответствующего образования и глубинных знаний – не программистами. JavaScript – официально зарегистрированная торговая марка компании Oracle.

С помощью него доступны к исполнению следующие функции:

* возможность изменять страницы браузеров;
* добавление или удаление тегов;
* изменение стилей страницы;
* информация о действиях пользователя на странице;
* запрос доступа к случайной части исходного кода страницы;
* внесение изменений в этот код;
* выполнение действия с cookie-файлами.

Область применения этого языка удивительно обширна и ничем не ограничена: среди программ, которые используют JS, присутствуют и тестовые редакторы, и приложения (как для компьютеров, так и мобильные и даже серверные), и прикладное ПО.

Преимущества JavaScript:

* Ни один современный браузер не обходится без поддержки JavaScript.
* С использованием написанных на JavaScript плагинов и скриптов справится даже не специалист.
* Полезные функциональные настройки.
* Взаимодействие с приложением может осуществляется даже через текстовые редакторы – Microsoft Office и Open Office.
* Перспектива использования языка в процессе обучения программированию и информатике.

Недостатки JavaScript:

* Пониженный уровень безопасности ввиду повсеместного и свободного доступа к исходным кодам популярных скриптов.
* Множество мелких раздражающих ошибок на каждом этапе работы. Большая часть из них легко исправляется, но их наличие позволяет считать этот язык менее профессиональным, сравнительно с другими.
* Повсеместное распространение. Своеобразным недостатком можно считать тот факт, что часть активно используемых программ (особенно приложений) перестанут существовать при отсутствии языка, поскольку целиком базируются на нем.
  1. **Программная платформа Node.js**

Node.js — среда выполнения кода JavaScript вне браузера. Эта платформа позволяет писать серверный код для динамических веб-страниц и веб-приложений, а также для программ командной строки. С помощью Node.js реализуется парадигма «JavaScript для всего». Она предполагает использование одного языка программирования для разработки веб-приложений вместо применения разных языков для работы над фронтендом и бэкендом.

Node.js — не отдельный язык программирования, а платформа для использования JavaScript на стороне сервера. Если говорить о языке, то как для фронденда, так и для бэкенда используется один и тот же JavaScript. Разница только в наборе API, которые используют фронтендеры и бэкендеры.

Браузерный JavaScript использует Web API, которые обеспечивают доступ к DOM и пользовательскому интерфейсу страниц и веб-приложений. Серверный JavaScript использует API, обеспечивающие доступ к файловой системе приложений, http-запросам, потокам.

То есть Node.js — это технология для использования JS на бэкенде. С особенностями и перспективами развития языка JavaScript можно ознакомиться в [соответствующей статье](https://ru.hexlet.io/blog/posts/stoit-li-uchit-javascript-perspektivy-situatsiya-na-rynke-truda-mneniya-ekspertov), а здесь речь идёт об одной из технологий данного языка.

Платформа Node.js была представлена в 2009 году. Её создал инженер Райан Дал, а спонсором разработки выступила компания Joyent. Компания известна поддержкой опенсорсных проектов, включая Node.js, Illumos, SmartOS.

Райан Дал использовал для создания Node.js движок V8. Платформа реализована с низкоуровневой неблокирующей моделью ввода/вывода, которая построена на событийно-ориентированной модели.

Как отмечалось выше, Node.js применяется для [бэкенд-разработки на JavaScript](https://ru.hexlet.io/professions/backend). Если в браузере JavaScript тотально доминирует, и конкурентов этого языка на фронтенде не видно даже на горизонте, то в бэкенд-разработке ситуация другая. Здесь JS конкурирует с PHP, Python и другими языками.

Выбор серверного JavaScript для бэкенда обеспечивает проекту ряд преимуществ:

* рост эффективности разработки благодаря использованию одного языка для фронт- и бэкенда и возможности переиспользования кода;
* возможность использовать [npm](https://www.npmjs.com/) — самый большой пакетный менеджер;
* более простой по сравнению с другими стеками поиск исполнителей, так как JavaScript входит в число самых популярных языков программирования.

Node.js хорошо подходит для разработки RTA — веб-приложений, реагирующих на действия пользователя в режиме реального времени. Например, это может быть онлайн-редактор типа Google Docs, который позволяет работать над одним документом нескольким пользователям одновременно.

**Node.js легко обрабатывает большое количество запросов одновременно и обеспечивает быстродействие приложения. Поэтому серверный JavaScript часто используют для создания SPA — одностраничных веб-приложений, в которых рендеринг выполняется на стороне клиента. Node.js на бэкенде используют Netflix, Uber, eBay, Groupon, Yahoo и другие известные организации, и проекты.**

* 1. **Облачная база данных Firebase**

Firebase – это платформа разработки мобильных приложений с огромным функционалом. Начиналась она как стартап, а сегодня ее используют при разработке лучших кроссплатформенных приложений. Главное достоинство платформы в том, что она позволяет разработчику не отвлекаться на создание бэкенда, то есть скрытой от пользователя программной части проекта, например, серверного кода. И это упрощает и ускоряет создание мобильных приложений, дает возможность полностью сосредоточиться именно на UX/UI, то есть, на пользовательском интерфейсе и опыте.

Firebase – это одно из BaaS-решений (Backend as a Service), которое дает разработчику массу возможностей.

Это и сервер, и база данных, и хостинг, и аутентификация в одной платформе. Так, Firebase Realtime Database предоставляет разработчикам API, который синхронизирует данные приложения между клиентами и хранит их в облачном хранилище.

Приложение подключается к базе данных через WebSocket, который отвечает за синхронизацию данных в течение всего сеанса.

Также Firebase выступает в качестве хранилища файлов. Firebase Storage обеспечивает надежную загрузку и выгрузку файлов для приложения. Облачное хранение файлов видео, аудио или любого другого типа поддерживается Google Cloud Storage. Содержимое облачного хранилища надежно защищено собственной системой безопасности.

Создавать систему аутентификации каждый раз с нуля довольно затратно, причем затраты эти чаще всего не оправданы. Справится с большинством вызовов позволяет система аутентификации Firebase Auth, в которой возможна аутентификация пользователя приложения по паролю и электронной почте. Поддерживает Firebase Auth также открытый протокол авторизации OAuth 2.0, используемый Google, Twitter, Facebook. Система аутентификации Firebase интегрируется непосредственно в базу данных.

Статические файлы приложения размещаются на хостинге Firebase. Поддерживается хостинг файлов JavaScript, HTML, CSS и других. Через Cloud Functions реализована динамическая поддержка Node.js. Передача файлов осуществляется через сеть доставки контента с использованием защищенных протоколов SSL и HTTPS.

* 1. **Git / GitHub**

Git — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано.

Среди проектов, использующих Git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, ряд дистрибутивов Linux.

Программа является свободной и выпущена под лицензией GNU GPL версии 2. По умолчанию используется TCP порт 9418.

Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone, Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удаленный доступ к репозиториям Git обеспечивается git-демоном, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа.

Ядро Git представляет собой набор утилит командной строки с параметрами. Все настройки хранятся в текстовых файлах конфигурации. Такая реализация делает Git легко портируемым на любую платформу и дает возможность легко интегрировать Git в другие системы (в частности, создавать графические git-клиенты с любым желаемым интерфейсом).

Репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранятся операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы.

Структура хранилища файлов не отражает реальную структуру хранящегося в репозитории файлового дерева, она ориентирована на повышение скорости выполнения операций с репозиторием. Когда ядро обрабатывает команду изменения (неважно, при локальных изменениях или при получении патча от другого узла), оно создает в хранилище новые файлы, соответствующие новым состояниям изменённых файлов. Существенно, что никакие операции не изменяют содержимого уже существующих в хранилище файлов.

По умолчанию репозиторий хранится в подкаталоге с названием «.git» в корневом каталоге рабочей копии дерева файлов, хранящегося в репозитории. Любое файловое дерево в системе можно превратить в репозиторий git, отдав команду создания репозитория из корневого каталога этого дерева (или указав корневой каталог в параметрах программы). Репозиторий может быть импортирован с другого узла, доступного по сети. При импорте нового репозитория автоматически создается рабочая копия, соответствующая последнему зафиксированному состоянию импортируемого репозитория (то есть не копируются изменения в рабочей копии исходного узла, для которых на том узле не была выполнена команда commit).

Нижний уровень git является так называемой контентно-адресуемой файловой системой. Инструмент командной строки git содержит ряд команд по непосредственной манипуляции этим репозиторием на низком уровне. Эти команды не нужны при нормальной работе с git как с системой контроля версий, но нужны для реализации сложных операций (ремонт повреждённого репозитория и так далее), а также дают возможность создать на базе репозитория git свое приложение.

Для каждого объекта в репозитории вычисляется SHA-1-хеш, и именно он становится именем файла, содержащего данный объект в каталоге .git/objects. Для оптимизации работы с файловыми системами, не использующими деревья для каталогов, первый байт хеша становится именем подкаталога, а остальные — именем файла в нём, что снижает количество файлов в одном каталоге (ограничивающий фактор производительности на таких устаревших файловых системах).

В классическом обычном сценарии в репозитории git есть три типа объектов — файл, дерево и «коммит». Файл есть какая-то версия какого-то пользовательского файла, дерево — совокупность файлов из разных подкаталогов, «коммит» — дерево и некая дополнительная информация (например, родительские коммиты, а также комментарий).

Репозиторий Git бывает локальный и удаленный. Локальный репозиторий — это подкаталог .git, создаётся (в пустом виде) командой git init и (в непустом виде с немедленным копированием содержимого родительского удалённого репозитория и простановкой ссылки на родителя) командой git clone.

Практически все обычные операции с системой контроля версий, такие, как коммит и слияние, производятся только с локальным репозиторием. Удалённый репозиторий можно только синхронизировать с локальным как «вверх» (push), так и «вниз» (pull).

Наличие полностью всего репозитория проекта локально у каждого разработчика даёт Git ряд преимуществ перед SVN. Так, например, все операции, кроме push и pull, можно осуществлять без наличия интернет-соединения.

Очень мощной возможностью git являются ветви, реализованные куда более полно, чем в SVN: по сути, ветвь git есть не более чем именованная ссылка, указывающая на некий коммит в репозитории (используется подкаталог refs). Коммит без создания новой ветви всего лишь передвигает эту ссылку на себя, а коммит с созданием ветви — оставляет старую ссылку на месте, но создает новую на новый коммит, и объявляет её текущей. Заменить локальные девелоперские файлы на набор файлов из иной ветви, тем самым перейдя к работе с ней — так же тривиально.

Команда push передает все новые данные (те, которых еще нет в удалённом репозитории) из локального репозитория в репозиторий удаленный. Для исполнения этой команды необходимо, чтобы удалённый репозиторий не имел новых коммитов в себя от других клиентов, иначе push завершается ошибкой, и придётся делать pull и слияние.

Команда pull — обратна команде push. В случае, если одна и та же ветвь имеет независимую историю в локальной и в удаленной копии, pull немедленно переходит к слиянию.

Слияние в пределах разных файлов осуществляется автоматически (всё это поведение настраивается), а в пределах одного файла — стандартным двухпанельным сравнением файлов. После слияния нужно объявить конфликты как разрешенные.

Результатом всего этого является новое состояние в локальных файлах у того разработчика, что осуществил слияние. Ему нужно немедленно сделать коммит, при этом в данном объекте коммита в репозитории окажется информация о том, что коммит есть результат слияния двух ветвей и имеет два родительских коммита.

Также Git имеет временный локальный индекс файлов. Это — промежуточное хранилище между собственно файлами и очередным коммитом (коммит делается только из этого индекса). С помощью этого индекса осуществляется добавление новых файлов (git add добавляет их в индекс, они попадут в следующий коммит), а также коммит не всех измененных файлов (коммит делается только тем файлам, которым был сделан git add). После git add можно редактировать файл далее, получатся три копии одного и того же файла — последняя, в индексе (та, что была на момент git add), и в последнем коммите.

Имя ветви по умолчанию: master. Имя удалённого репозитория по умолчанию, создаваемое git clone во время типичной операции «взять имеющийся проект с сервера себе на машину»: origin.

Таким образом, в локальном репозитории всегда есть ветвь master, которая есть последний локальный коммит, и ветвь origin/master, которая есть последнее состояние удаленного репозитория на момент завершения и исполнения последней команды pull или push.

Команда fetch (частичный pull) — берёт с удалённого сервера все изменения в origin/master, и переписывает их в локальный репозиторий, продвигая метку origin/master.

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

* Прямо на сайте можно просмотреть файлы проектов с подсветкой синтаксиса для большинства языков программирования.
* Можно создавать приватные репозитории, которые будут видны только вам и выбранным вами людям. Раньше возможность создавать приватные репозитории была платной.
* Есть возможность прямого добавления новых файлов в свой репозиторий через веб-интерфейс сервиса.
* Код проектов можно не только скопировать через Git, но и скачать в виде обычных архивов с сайта.
* Кроме Git, сервис поддерживает получение и редактирование кода через SVN и Mercurial.

# **Технико-экономическое обоснование эффективности разработки и использования персонального менеджера**

## Описание функций, назначения и потенциальных пользователей программного обеспечения

Разрабатываемый в дипломном проекте программный модуль предназначен для отслеживания доходов и расходов по категориям и периодам отдельного пользователя, для создания заметок, а также для просмотра ленты новостей с указанного сайта.

Разрабатываемый продукт представляет собой веб-сайт, позволяющий:

* Отображать визуализацию общих доходов и расходов в периоде, а также доходов и расходов по категориям
* Отображать, создавать, редактировать и удалять заметки.
* Хранить данные в облачной базе данных.
* Отображать ленту новостей с указанного сайта.
* Работать адаптивно в веб-браузере как десктопных, так и мобильных устройств.

Среди потенциальной аудитории программного продукта можно выделить частных лиц, заинтересованных в удобном отслеживании своего бюджета, личных заметок и просмотра ленты новостей. Модуль «Финансовый менеджер» позволит пользователю анализировать свои траты по заданным категориям, а также научит ограничивать и оптимизировать свой бюджет. Модуль «Заметки» позволит лёгким и доступным образом создавать себе напоминания о каких-либо грядущих событиях. Модуль «Лента новостей» запрашивает у пользователя URL-адрес сайта с новостями, откуда будут считываться последние на данный момент статьи и в удобном формате преподноситься пользователю приложения.

Программное средство разрабатывается для свободной реализации на рынке информационных технологий и использования широким кругом потребителей. Разработка и финансирование разработки системы осуществляется за счёт собственных средств. Как личные неимущественные, так и имущественные (исключительные) права принадлежат разработчику программного обеспечения.

Расчёты приведены по состоянию на апрель 2021 г.

Источник расчётных формул – методическое пособие.

Экономическое обоснование разработки и реализации ПО будет осуществляться в соответствии с п. 3.3. методического пособия.

## Расчёт затрат на разработку ПО

Для разработки данного программного средства необходимо привлечь на часть срока проведения работ менеджера проекта, тестировщика и дизайнера, на весь срок – программиста. Длительность разработки составит три месяца месяца или же 504 часов рабочего времени (согласно данным Минтруда о количестве рабочих часов в месяце).

Расчет затрат на основную заработную плату команды разработчиков осуществляется по формуле:

где *n* – количество исполнителей, занятых разработкой программного продукта;

– часовая тарифная ставка *i*-го исполнителей, *руб*.;

– трудоёмкость работ, выполняемых исполнителем *i*-й категории, определяется исходя из сложности разработки программного обеспечения и объема выполняемых им функций, *ч*.

Заработная плата менеджера проекта составляет 2460$ (5208р.), программиста – 1680$ (4368р.), тестировщика – 1070$ (2790р.), дизайнера – 1360$ (3534р.). Путём деления месячной заработной платы на количество рабочих часов в месяце, имеем часовую заработную плату для каждого из разработчиков: менеджер проекта - 31р., программист – 26р., тестировщик – 15р., дизайнер – 19р. Расчёт затрат на основную заработную плату осуществляется в табличной форме (табл. 2.1.).

Таблица 2.1. – Расчет затрат на основную заработную плату разработчиков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Участник команды | Часовая ставка, *руб.* | Трудоемкость работ, *ч.* | Зарплата по тарифу, *руб.* |
| 1 | Менеджер проекта | 31 | 252 | 7 812 |
| 2 | Программист (разработчик веб-приложения и базы данных) | 26 | 504 | 13 104 |
| 3 | Тестировщик | 15 | 120 | 1 800 |
| 4 | Дизайнер | 19 | 120 | 2 280 |
| Итого затраты на основную заработную плату | | | | 24993 |

Затраты на дополнительную заработную плату команды разработчиков и определяется по формуле:

где ‒ норматив дополнительной заработной платы (25 %), включающий премию в размере 5%.

–затраты на основную заработную плату, руб.;

Затраты на дополнительную заработную плату составят:

= 6 249 *руб*.

Отчисления на социальные нужды (в фонд социальной защиты населения и на обязательное страхование) определяются в соответствии с действующими законодательными актами по формуле:

где ‒ ставка отчислений в ФСЗН и Белгосстрах (35 %).

Затраты на социальные нужды составят:

(24996 + 6 249) 0,35 = 10 935,75 *руб*.

Прочие затраты включаются в себестоимость разработки ПО в процентах от затрат на основную заработную плату команды разработчиков по формуле:

где ‒ норматив прочих расходов, (35 %).

Прочие затраты составят:

8 748,6 *руб*.

Полная информация о формировании затрат для разработки программного средства приведена в таблице 2.2.

Таблица 2.2. – Затраты на разработку программного обеспечения

|  |  |
| --- | --- |
| Статья затрат | Сумма, *руб*. |
| Основная заработная плата команды разработчиков, | 24 996 |
| Дополнительная заработная плата команды разработчиков, | 6 249 |
| Отчисления на социальные нужды, | 10 935,75 |
| Прочие затраты, | 8 748,6 |
| Всего | 50 929,35 |

Таким образом, общая сумма затрат на разработку программного средства «Персональный менеджер» составит 50 929,35 *руб*.

## Оценка эффекта от продажи программного обеспечения

Экономический эффект организации-разработчика программного средства представляет собой прибыли (чистую прибыль) от его продажи на рынке потребителям, величина которой зависит от объема продаж, цены реализации и затрат на разработку программного средства.

В течение трех месяцев будут введены все основные функции и программное средство будет полностью готово к запуску. В первый год выхода на рынок ожидается более 10000 пользователей. Был проведен социологический опрос среди потенциальных пользователей приложения (совершеннолетних работающих людей), в результате которого было выявлено, какую сумму они готовы ежемесячно тратить на данный продукт.

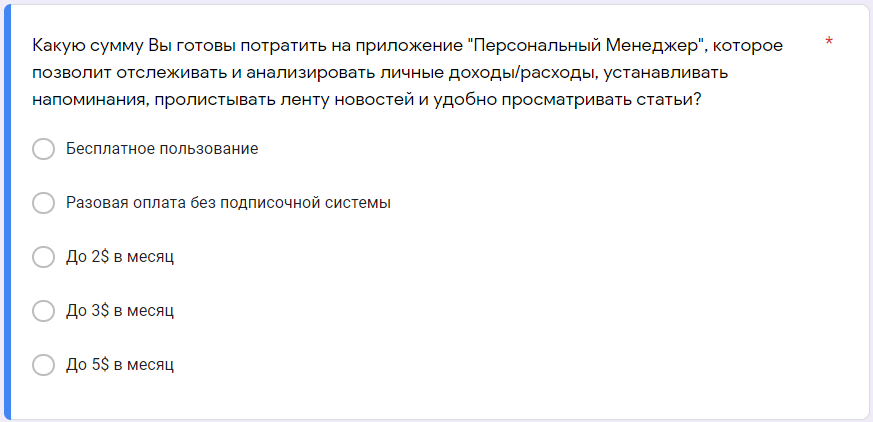


Рисунок 3.5. - Условие социологического опроса

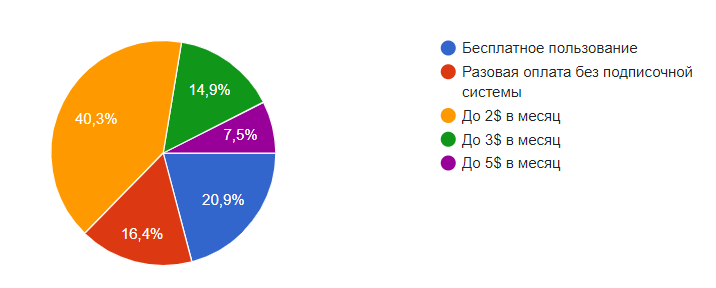


Рисунок 3.6. - Результат социологического опроса

Из социологического опроса (рис. 3.1-3.2) следует, что оптимальная стоимость месячной подписки за пользование приложением - 2$ (5 *руб*.). Пользователю предлагается выгодная годовая подписка, которая включает в себя один бесплатный месяц пользования. Из этого следует, что цена годовой подписки будет равна = 5 \* 11 = 55 *руб.* при количестве годовых подписок равном штук.

Организация-разработчик является налогоплательщиком (налог на прибыль), следовательно, экономический эффект можно рассчитать по формуле:

где П – прибыль за использование программного продукта;

согласно законодательству, равная 18%

Прибыли, полученные разработчиком от реализации программного средства на рынке, можно рассчитать по формуле:

где – цена подписки, *руб*.;

‒ количество подписок, реализуемое за год, шт.;

‒ сумма налога на добавленную стоимость, *руб*.;

Налог на добавленную стоимость определяется по формуле:

где – ставка налога на добавленную стоимость в соответствии с действующим законодательством (20 %).

Налог на добавленную стоимость равен:

*руб*.

В таком случае, прибыль, согласно формуле 3.2, равна:

= 162 961,59 *руб*.

Экономический эффект равен:

## Расчёт показателей эффективности инвестиций в разработку программного обеспечения

Оценка экономической эффективности разработки и реализации программного средства на рынке зависит от результата сравнения инвестиций в его разработку (модернизацию, совершенствование) и полученного годового прироста чистой прибыли.

Так как сумма инвестиций на разработку меньше суммы годового экономического эффекта, то есть инвестиции окупятся менее чем за год, оценка экономической эффективности инвестиций в разработку программного средства осуществляется с помощью расчета простой нормы прибыли (рентабельности инвестиций) по формуле:

где ‒ прирост чистой прибыли, *руб.;*

‒ затраты на разработку программного средства, *руб*.

Таким образом, рентабельность инвестиций будет равна:

Инвестиции на разработку программного средства и его реализация на рынке информационных технологий будут экономически эффективными, если рентабельность инвестиций превысит 100 % (100 % плюс ставка по банковским долгосрочным депозитам). А поскольку ставка по долгосрочным депозитам не превышает 15%, следовательно, программное средство целесообразно разрабатывать и реализовывать по установленной цене, т.к. рентабельность инвестиций превышает 115% более чем в 2 раза.

## Выводы по технико-экономическому обоснованию

На основании данных результатов можно сделать вывод, что проект представляется выгодным как для разработчика, так и для инвестора: реализация программного средства на рынке экономически эффективна.

Спустя год после внедрения данного программного средства заказчик не только покрывает собственные затраты, но и имеет прибыль. В свою очередь исполнитель также получает прибыль в короткие сроки.

Важно отметить, что статьей дохода приложения также может стать рекламная интеграция в виде специальных баннеров на сайте.