Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

|  |
| --- |
| *К защите допустить*: |
| Заведующая кафедрой ПОИТ |
| \_\_\_\_\_\_\_\_\_\_\_\_Н.В. Лапицкая |

Пояснительная записка

к дипломному проекту

на тему

**Веб-приложение «Ежедневник и финансовый трекер»**

БГУИР ДП 1-40 01 01 01 156 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | Е.С. Сасим |
| Руководитель |  | Н.В. Лапицкая |
| Консультанты: |  |  |
| *по экономической части* |  | О.А. Матяс |
| *от кафедры ПОИТ* |  | Н.В. Лапицкая |
| Нормоконтролер |  | Г.В. Данилова |
| Рецензент |  |  |

Минск 2024

# ВВЕДЕНИЕ

В современном мире, где темп жизни постоянно ускоряется, а количество задач и обязанностей экспоненциально увеличивается, все более актуальными становятся инструменты, помогающие эффективно управлять своим временем и личными финансами.

В качестве комплексного решения проблемы управления личным ресурсами может выступить веб-приложение «Ежедневник и финансовый трекер».  
 Приложение включает в себя модуль «Ежедневник», который может использоваться для планирования и управления задачами, создания важных заметок, организации задач и заметок в блоки по релевантным для пользователя признакам; и модуль «Финансовый трекер» – крайне удобный инструмент, созданный для управления личными финансами, позволяющий не только отслеживать доходы и расходы на коротком временном промежутке, но и получать полную удобочитаемую статистику, анализируя которую любой человек сможет улучшить своё финансовое положение.

Программное средство включает в себя интуитивно понятный интерфейс, широкий набор функций, высокую производительность, надёжную систему безопасности, а также поддержку нескольких языков.

Таким образом целью дипломного проектирования является разработка веб-приложения «Ежедневник и финансовый трекер», в удобной для пользователя форме и с поддержкой русского языка. Цель проектирования достигается решением следующих задач:

* анализ прототипов, предметной области и формирование требований к проектируемому веб-приложению;
* моделирование веб-приложения и разработка функциональных требований;
* проектирование веб-приложения;
* разработка веб-приложения;
* тестирование веб-приложения и анализ полученных результатов;
* составление руководства по использованию веб-приложения;
* технико-экономическое обоснование разработки веб-приложения.

# АНАЛИЗ ПРОТОТИПОВ, ПРЕДМЕТНОЙ ОБЛАСТИ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ВЕБ-ПРИЛОЖЕНИЮ

## Анализ предметной области

### Общее описание веб-приложения “Ежедневник и финансовый трекер”

Веб-приложение «Ежедневник и финансовый трекер» (далее – «приложение») инкапсулирует в себя два модуля: «Ежедневник» и «финансовый трекер» соответственно.  
 «Ежедневник» – это инструмент, позволяющий пользователю выполнять:

* создание задач, записей, событий;
* задание приоритета задач;
* ведение списка дел;
* отслеживание прогресса достижения целей;
* мониторинг своего ментального состояния.

“Финансовый трекер” – это инструмент, позволяющий реализовывать:

* запись доходов и расходов;
* категоризацию транзакций;
* составление бюджета;
* анализ расходов;
* отслеживание финансовых целей;
* прогнозирование и вычисление потенциальной прибыли.

Инкапсуляция двух самых важных вспомогательных инструментов в жизни любого человека в рамках одного приложения делает его уникальным на рынке.

## Анализ существующих аналогов

Как было сказано выше прямых аналогов в силу специфики приложения нет. Но можно рассматривать аналоги модулей, из которых состоит данное приложение.

### Программное средство Todoist

В качестве первого программного средства для рассмотрения и анализа было выбрано Todoist, так как данное приложение является самым популярным на рынке. Интерфейс данного ПС изображен на рисунке 1.1.

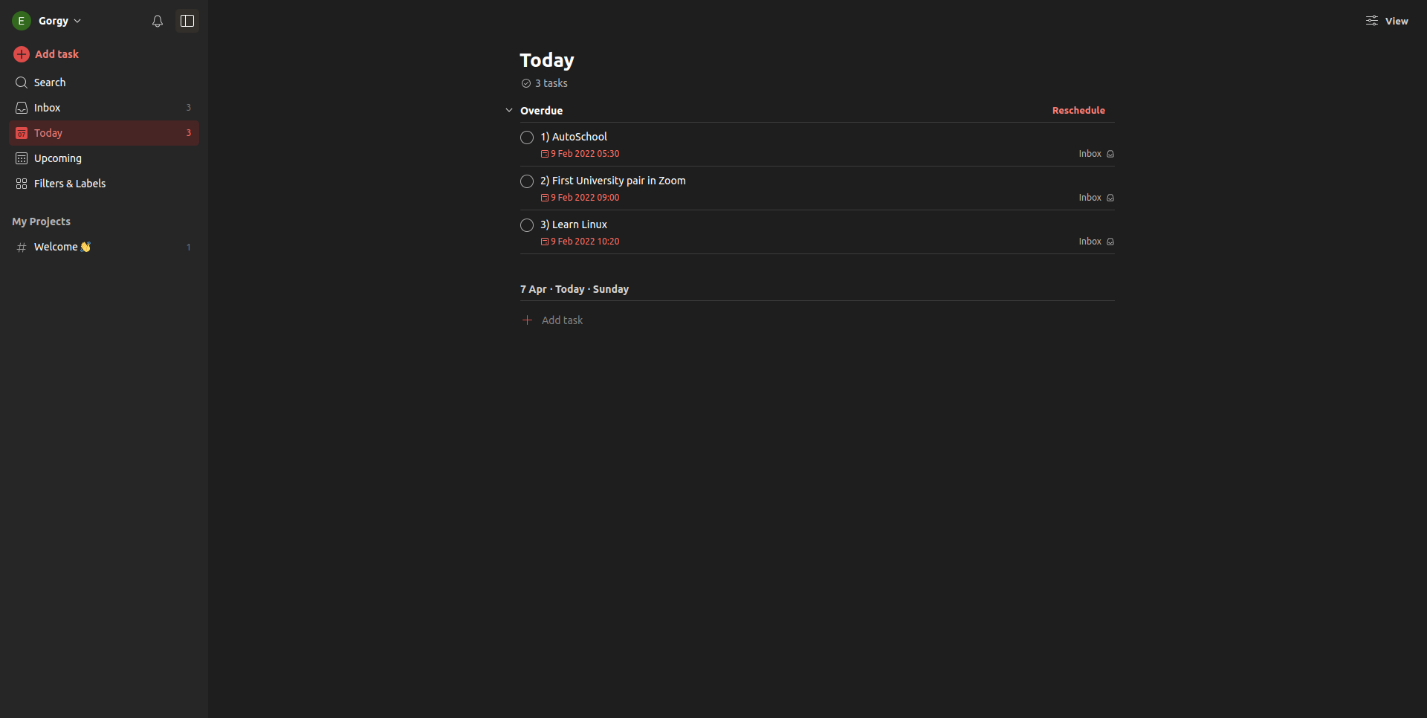


Рисунок 1.1 – Главная страница Todoist.com

Todoist – таск-менеджер для управления задачами проекта. Todoist позволяет командам или отдельному сотруднику структурировать рабочий день или спланировать будущие действия в список задач. Созданные списки задач можно отслеживать и помечать в браузере, в десктопном клиенте и на мобильных устройствах.

Основной элемент в Todoist – задачи. Задачи содержат в себе название, описание, дедлайн и теги. Созданную задачу можно добавить к плану на день или отправить во вкладку «Входящие», где будут храниться нераспределённые задачи. Для планирования будущих действий в Todoist есть вкладка «Предстоящее». В ней пользователь может запланировать задачи на определённые дни. Рутинные, повторяющиеся задачи можно дублировать и распределить их по дням недели.

Командная работа в Todoist представляет из себя работу в проекте. Члены команды могут совместно обсуждать задачи, делегировать новые задачи, устанавливать приоритетность их выполнения, руководители проекты имеют возможность устанавливать дневной план сотруднику. Для просмотра совершённых действия над проектом члены команды могут просматривать активность проекта в целом или отдельного сотрудника, какие задачи он выполнил, а какие у него в работе. Команды могут просматривать проект в виде досок, что позволяет увидеть в проект в разрезе и спланировать будущие действия.

К недостаткам данного ПС можно отнести отсутствие полного функционала в бесплатной версии приложения, сброс сортировки задач по атрибутам при добавлении новой задачи, а также невозможность поменять заменить названия дней недели в дедлайне задачи на дату.

### Программное средство TickTick

TickTick – сервис для создания и организации списков задач. TickTick помогает пользователям фиксировать идеи, создавать планы на день или на долгий срок, чтобы эффективно распределять личное время. Сервис подойдёт как обычным пользователям — студентам, творческим, организованным людям, так и для совместной работы в командах. TickTick доступен в браузере, в виде мобильного приложения и десктопной версии.

TickTick имеет облегчённый интерфейс, где для создания задачи достаточно открыть или создать список, и в строке «Добавить задачу» написать её название или описание. После этого задача попадёт в список, слева от неё появится чек-бокс, который можно активировать поле решения задачи. Пользователи дополнительно к задаче написать описание, выставить приоритет, перенести её на другой день или оставить дополнительный комментарий. Для поиска задач можно воспользоваться поиском по тегам или фильтрами.

Совместная работа в TickTick выглядит следующим образом: пользователи делятся или создают совместные списки задач, создают задачи и назначают их исполнителей. Команды могут просматривать активность выполнения задач в списках и в самих задачах, оставлять комментарии к задачам и устанавливать дедлайны. По завершению задач можно просматривать статистику и время их выполнения. Внешний вид приложения представлен на рисунке 1.2.

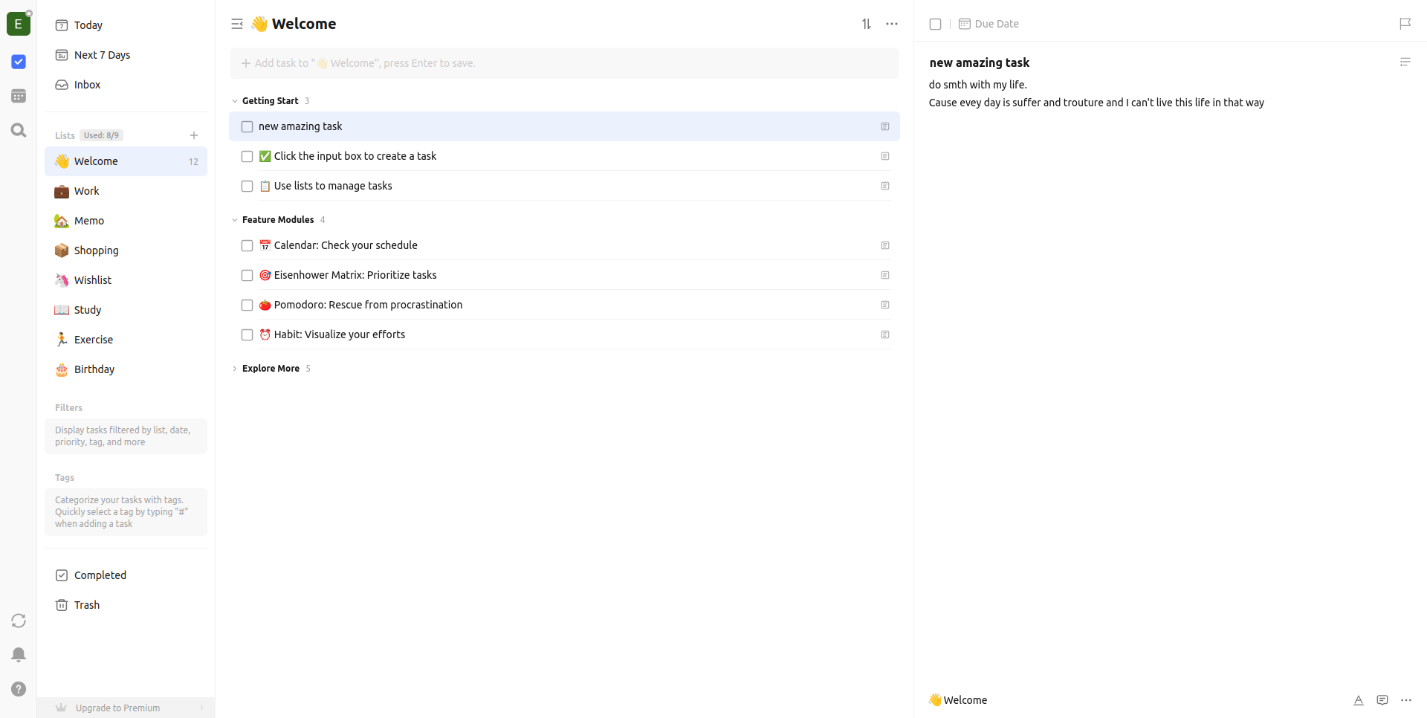


Рисунок 1.2 – Главная страница ticktick.com

К недостаткам данного ПС можно отнести отсутствие режима “в отпуске”, мизерное количество статистики, а также периодическую необходимость обновления страницу браузера для актуализации данных.

### Программное средство Google Календарь

Google Календарь – онлайн-сервис для планирования встреч и событий, организации рабочего времени. Google Календарь помогает командам организовать рабочие процессы и спланировать будущие мероприятия. Пользоваться календарём можно в браузере и с мобильных устройств на iOS и Android. Google Календарь имеет синхронизацию с другими сервисами корпорации.

В «Календарь» изначально добавлены общие события и праздники, и события из связанных аккаунтов. Для добавления нового события на календарь достаточно нажать на дату и в открывшемся окне заполнить поля. События могут быть трёх видов — мероприятие, задача или напоминание. «Мероприятие» имеет поля: название, дата и время, гости и местоположение, к ним можно прикрепить файлы. К задаче можно написать описание и выставить сроки выполнения. У напоминания есть только название и время выполнения, также можно установить повтор. Добавленные события будут автоматически синхронизироваться на разных устройствах.

Компании и отдельные команды для совместной работы могут создавать групповые календари, где будут отображаться общие события и задачи. С помощью интеграции Google Календаря и других сервисов компании можно оптимизировать рабочие процессы. Например, можно создавать события из писем [Gmail](https://startpack.ru/application/google-gmail" \o "Google Почта" \t "_blank) или из «Календаря» переходить в назначенную конференцию в [Google Meet](https://startpack.ru/application/google-meet" \o "Google Meet" \t "_blank). Также Google Календарь имеет интеграцию с большим количеством сторонних сервисов, которые могут увеличить его функциональность. Интерфейс Google Календарь представлен на рисунке 1.3.

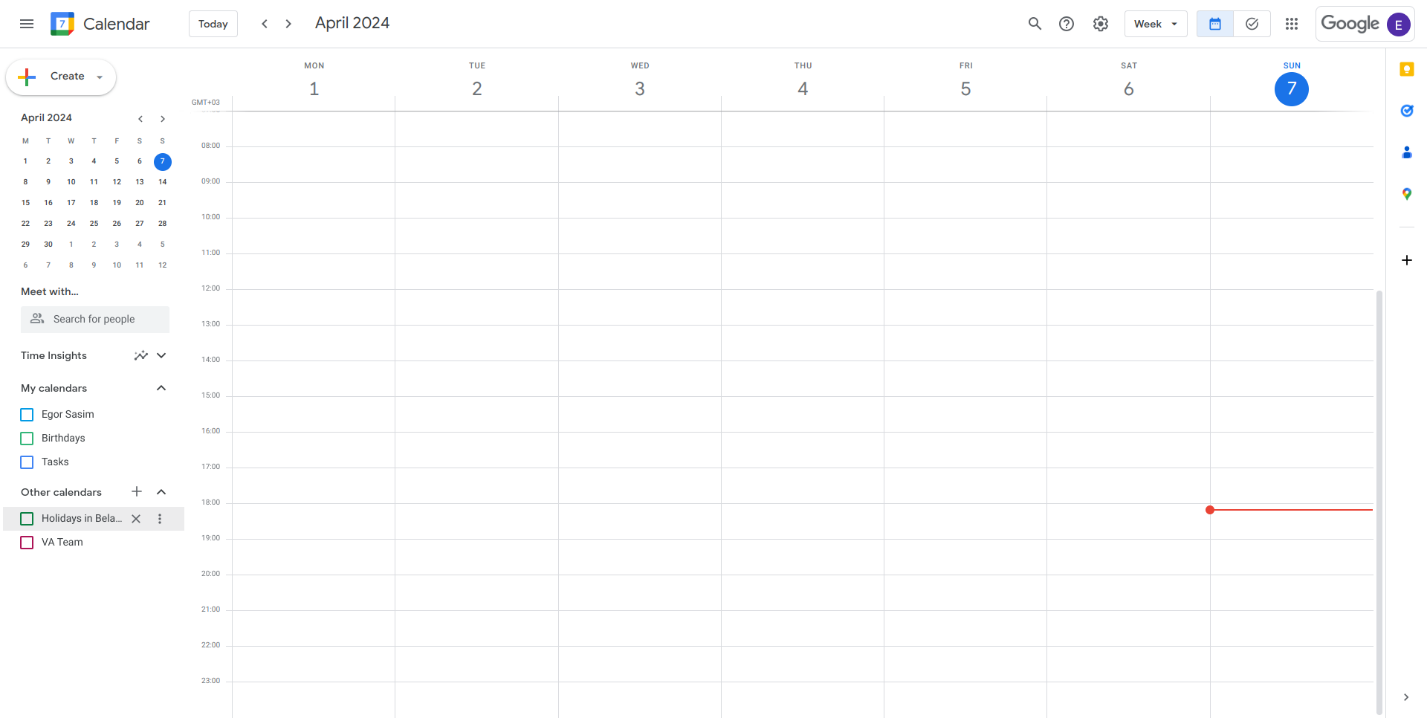


Рисунок 1.3 – Главная страница calendar.google.com

К недостаткам данного ПС можно отнести устаревшее оформление веб версии, а также возможность редактирования втроенного календаря только из google-аккаунта.

### Программное средство YNAB

YNAB – это сервис программного обеспечения, предназначенный для людей, которым необходимо привести свои финансы в порядок и поддерживать их в таком состоянии. Это онлайн-соглашение, которое включает в себя относительно простую серию этапов настройки плюс функции, разработанные для повышения ваших навыков бюджетирования.

Появились новые основные функции, которые позволяют напрямую подключаться к банковским счетам и счетам кредитных карт, а также устанавливать цели управления капиталом с помощью параметра «Цели». После завершения настройки все, что вам действительно нужно сделать, это окунуться в ситуацию и постепенно увеличивать общую картину своих ежемесячных доходов и расходов. Это хороший выбор, если вы хотите сэкономить или погасить долги. Интерфейс YNAB представлен на рисунке 1.4.

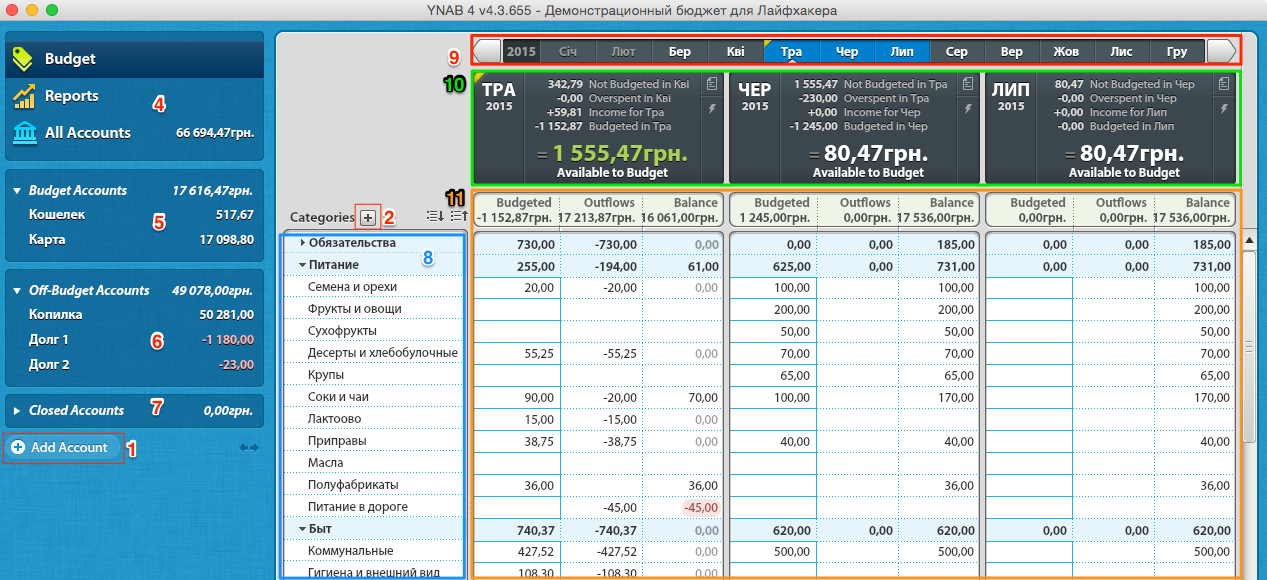


Рисунок 1.4 – Главная страница YNAB

К недостаткам данного ПС можно отнести отсутствие парсинга СМС сообщений, а также дорогостоящую подписку.

### Программное средство CoinKeeper

CoinKeeper – это приложение для управления личными финансами, доступное на Android, IOS и в виде веб-приложения

ПС предоставляет возможность отслеживания доходов и расходов, автоматической категоризации транзакций, создания фильтров и получения уведомлений о превышении лимитов.

Так же приложение предоставляет широкий функционал по аналитике транзакций в виде различных диаграмм и графиков.

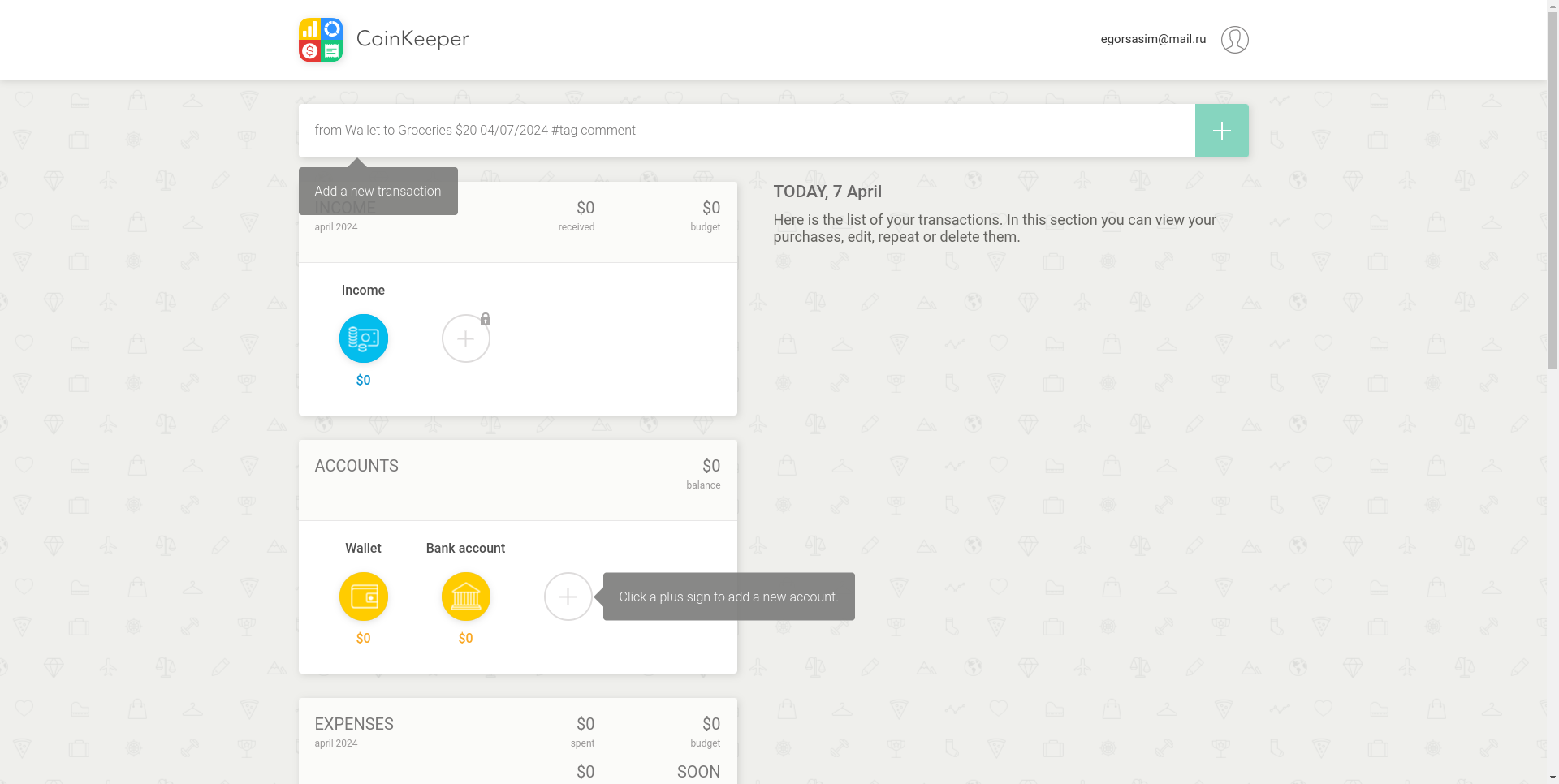
ПС имеет встроенную поддержку синхронизации между устройствами, гибкую настройку интерфейса, импорт и экспорт данных, совместный доступ к бюджету и поддержку тем оформления, что выделяет его на фоне всех остальных ПС. Интерфейс приложения CoinKeeper представлен на рисунке 1.5.  


Рисунок 1.4 – Главная страница CoinKeeper.

## Требования к проектируемому веб-приложению

### Назначение разработки

Назначением дипломного проектирования является разработка веб-приложения “Ежедневник и финансовый трекер”. На основании произведенного анализа существующих аналогов, выявленных недостатков и преимуществ приложений, сделан вывод, что для решения поставленной цели необходимо выполнить следующие задачи:

* проектирование:

1. архитектуры веб-приложения;
2. базы данных веб-приложения;

* разработка:

1. серверной части веб-приложения;
2. пользовательского интерфейса веб-приложения;

* тестирование разработанного веб-приложения.

Веб-приложения должно выполнять следующие основные функции:

* регистрация и аутентификация;
* настройка профиля пользователя;
* выбор языка;
* выбор темы оформления;
* создание, редактирование, удаление, фильтрация:
  1. задач;
  2. записей;
  3. пространств;
  4. доходов;
  5. видов доходов;
  6. видов расходов;
  7. расходов;
  8. видов расходов;
* репрезентация данных в виде графиков и диаграмм;
* вычисление сложного процента с пополнением.

### Требования к входным и выходным данным

В качестве входных данных выступает информация, вводимая пользователем при работе с веб-приложением, а также различные опции, предоставляемые пользовательским интерфейсом. В качестве выходных данных выступает информация в графическом виде, получаемая в процессе взаимодействия с пользовательским интерфейсом.

**1.3.3** Требования к составу и параметрам технических и программных средств

Серверная часть веб-приложения должна функционировать на любом персональном компьютере, который соответствует следующим характеристикам:

* процессор Intel Core c тактовой частотой 2 ГГЦ и более;
* оперативная память размерам более 2 Гбайт;
* физическая память формата SSD более 4 Гбайт.

На основании выдвинутых к программному средству функциональных требований и обзора существующих аналогов было принято решение о проектировании веб-приложения доступного с любого вида устройства через веб-браузер.

Клиентская часть веб-приложения должна функционировать на любом персональном компьютере, соответствующем следующим характеристикам:

* процессор Intel Core c тактовой частотой 2 ГГЦ и более;
* оперативная память размерам более 1 Гбайт.

наличие одного из браузеров: браузера «Google Chrome» минимальной версии 74, Mozilla Firefox минимальной версии 68, Opera минимальной версии 58, Apple Safari минимальной версии 10.

# МОДЕЛИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

## Функциональная модель веб-приложения

Функциональная модель разрабатываемого веб-приложения представлена в виде диаграммы вариантов использования. Диаграмма вариантов использования позволяет визуализировать, как система взаимодействует с внешними актерами (как правило, это пользователи, другие системы или внешние компоненты), какие функциональные возможности системы доступны для актеров и какие результаты они могут ожидать. Данная диаграмма помогает лучше понять взаимодействие системы с ее окружением и является важным инструментом при анализе требований и проектировании системы.

На основе требований, выдвинутых в подразделе 1.3, проектируемое веб-приложение предполагает поддержку системы ролей.

В веб-приложении определены следующие роли:

* гость;
* пользователь.

Возможности ролей представлены на рисунке 2.1 в виде диаграммы вариантов использования в нотации UML 2.1.

Основными элементами диаграммы являются актеры и прецеденты. Актеры представляют собой роли внешних объектов, которые взаимодействуют с различными частями программного средства. Актерами могут быть люди, другие системы, подсистемы или классы, представляющие абстрактные сущности. Прецеденты – это описание множества последовательных событий, выполняемых программным средством, и отображение взаимодействия между актерами и системой.

Для роли «Гость» (пользователь, не зарегистрированный в программном средстве или не прошедший авторизацию и аутентификацию) будут реализованы следующие функции:

* регистрация профиля;
* аутентификация, приводящая к смене роли «Гость» на роль «Пользователь».

Для роли «Пользователь» будут реализованы следующие функции:

* смена:
  1. пароля;
  2. логина;
  3. цветовой палитры;
  4. языка;
* управление ежедневником:
  1. управление задачами:
     1. создание;
     2. удаление;
     3. редактирование;
     4. просмотр в виде:
        + списка;
        + графика;
        + календаря;
     5. поиск по заданным критериям;
  2. управление записями:
     1. создание;
     2. удаление;
     3. редактирование;
     4. просмотр в виде списка;
     5. поиск по заданным критериям;
  3. управление пространствами:
     1. создание;
     2. удаление;
     3. редактирование;
     4. просмотр в виде списка;
     5. поиск по заданным критериям;
* управление финансовым трекером:
  1. управление доходами:
     1. создание;
     2. редактирование;
     3. удаление;
     4. просмотр в виде:
        + списка;
        + графика;
     5. поиск по заданным критериям;
  2. управление расходами:
     1. создание;
     2. редактирование;
     3. удаление;
     4. просмотр в виде:
        + списка;
        + графика;
  3. вычисление сложного процента с пополнением;
  4. просмотр текущего баланса в виде:
     1. графика;
     2. текста.

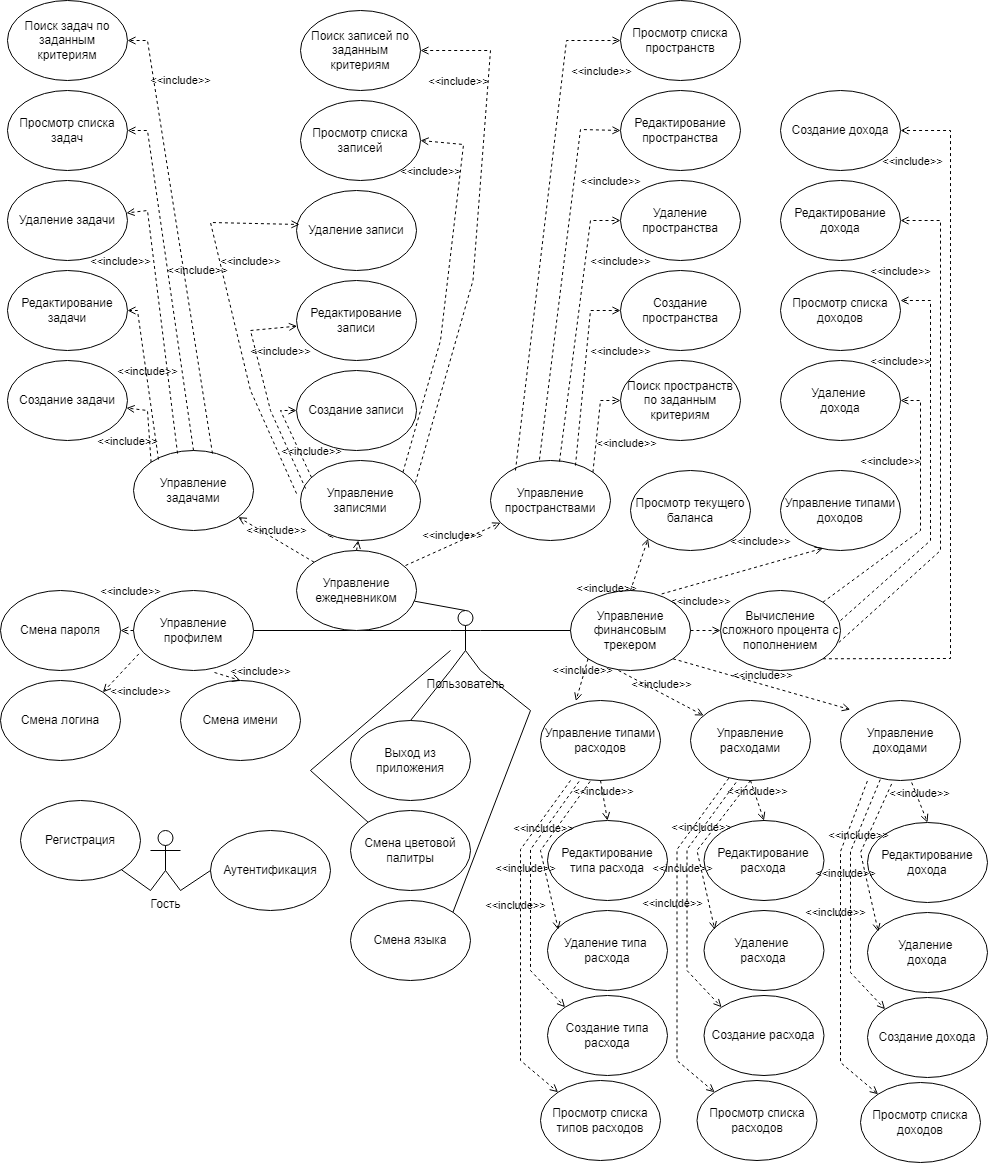


Рисунок 2.1 – Диаграмма вариантов использования

## Разработка спецификации функциональных требований

Для того чтобы более подробно описать функции разрабатываемого веб-приложения, были рассмотрены основные требования, касающиеся внутренней организации системы и взаимодействия системы с пользователем, из раздела 1.3.

### Функция регистрации

Регистрация должна быть реализована с учетом следующих требований:

* инициирование процесса регистрации пользователем c ролью «Гость»;
* осуществление процесса регистрации с помощью трех обязательных параметров: имя, логин, пароль;
* проверка валидности введенных данных c помощью встроенных инструментов разработки;
* отображение пользователю сообщения о невозможности регистрации в случае некорректных введенных данных;
* возможность смены пароля после регистрации;
* возможность редактирования логина после регистрации.

### Функция аутентификации

Аутентификация должна быть реализована с учетом следующих требований:

* инициирование процесса аутентификации клиентом c ролью «Гость»;
* необходимость ввода клиентом логина, указанного при регистрации, и текущего пароля для прохождения данного процесса;
* отображение сообщения о невозможности аутентификации в случае ввода пользователем некорректных данных.

### Функция управления профилем

Управление профилем должно быть реализовано с учётом следующих требований:

* изменение логина;
* изменение пароля;
* изменение электронной почты;

### Функция управления задачами

Управление задачами должно быть реализовано с учётом следующих требований:

* управление задачами должно быть доступно только клиентам с ролью «Пользователь»;
* пользователь может взаимодействовать только с созданными им задачами;
* пользователь должен иметь возможность создавать, редактировать и удалять задачи;
* пользователь должен иметь возможность просматривать список всех задач, а также инструмент для поиска задач по заданным критериям;

### Функция управления записями

Управление задачами должно быть реализовано с учётом следующих требований:

* управление записями должно быть доступно только клиентам с ролью «Пользователь»;
* пользователь может взаимодействовать только с созданными им записями;
* пользователь должен иметь возможность создавать, редактировать и удалять записями;
* пользователь должен иметь возможность просматривать список всех записей, а также инструмент для поиска записей по заданным критериям;

### Функция управления пространствами

Управление задачами должно быть реализовано с учётом следующих требований:

* управление пространствами должно быть доступно только клиентам с ролью «Пользователь»;
* пользователь может взаимодействовать только с созданными им пространствами;
* пользователь должен иметь возможность создавать, редактировать и удалять пространствами;
* пользователь должен иметь возможность просматривать список всех пространствами, а также инструмент для поиска пространств по заданным критериям;

### Функция управления доходами

Управление доходами должно быть реализовано с учётом следующих требований:

* управление доходами должно быть доступно только клиентам с ролью «Пользователь»;
* пользователь может взаимодействовать только с созданными им доходами;
* пользователь должен иметь возможность создавать, редактировать и удалять доходы;
* пользователь должен иметь возможность просматривать список всех доходов, а также инструмент для поиска транзакций по заданным критериям;

### Функция управления расходами

Управление расходами должно быть реализовано с учётом следующих требований:

* управление расходами должно быть доступно только клиентам с ролью «Пользователь»;
* пользователь может взаимодействовать только с созданными им расходами;
* пользователь должен иметь возможность создавать, редактировать и удалять расходы;
* пользователь должен иметь возможность просматривать список всех расходов, а также инструмент для поиска транзакций по заданным критериям;

### Функция управления видами доходов

Управление видами доходов должно быть реализовано с учётом следующих требований:

* управление видами доходов должно быть доступно только клиентам с ролью «Пользователь»;
* пользователь может взаимодействовать только с созданными им видами доходов;
* пользователь должен иметь возможность создавать, редактировать и удалять виды доходов;
* пользователь должен иметь возможность просматривать список всех видов доходов;

### Функция управления видами расходов

Управление видами расходов должно быть реализовано с учётом следующих требований:

* управление видами расходов должно быть доступно только клиентам с ролью «Пользователь»;
* пользователь может взаимодействовать только с созданными им видами расходов;
* пользователь должен иметь возможность создавать, редактировать и удалять виды расходов;
* пользователь должен иметь возможность просматривать список всех видов расходов;

### Функция просмотра текущего баланса

Управление просмотра текущего баланса должно быть реализовано с учётом следующих требований:

* просмотр текущего баланса должен быть доступен только клиентам с ролью «Пользователь»;
* пользователь может просматривать только свой баланс;

## Инфологическая модель базы данных веб-приложения

Инфологическая модель базы данных – это модель, которая описывает логическую организацию данных в базе данных без учета специфических деталей реализации, таких как физическая структура хранения данных или язык запросов базы данных. Она представляет абстрактное представление данных и их отношений, основываясь на концептуальной модели данных и требованиях к базе данных.

Инфологическая модель базы данных обычно включает следующие элементы:

* сущности, описывающие основные объекты, которые будут храниться в базе данных, и их атрибуты;
* связи, описывающие отношения между сущностями и определяющие, как они взаимодействуют друг с другом;

Инфологическая модель базы данных является важным этапом проектирования базы данных, и она служит основой для создания физической модели базы данных, которая включает в себя сами детали реализации. Инфологическая модель не зависит от конкретной СУБД, а только описывает структуру данных и их связи.

Основными сущностями веб-приложения являются: «Пользователь», «Задача», «Запись», «Пространство», «Трата», «Доход», «Тип траты», «Тип доходы». Дополнительные сущности возникают в результате взаимодействия между собой основных сущностей, если связь между ними «many-to-many».

Сущность «Пользователь» содержит данные для авторизации, а также персональную информацию пользователя веб-приложения. Атрибутами сущности являются:

* идентификатор пользователя (id);
* имя пользователя (login);
* зашифрованный пароль (password);
* электронная почта (email);

Сущность «Задача» содержит данные о созданной задаче. Атрибутами сущности являются:

* идентификатор задачи (id);
* заголовок (title);
* дата создания (creation date);
* описание (description);
* приоритет (priority);
* дата начала (start date);
* дата окончания (end date);
* статус (status);
* цвет(color);
* идентификатор пользователя – внешний ключ на сущность «Пользователь».

Сущность «Запись» содержит данные о созданной записи. Атрибутами сущности являются:

* идентификатор записи (id);
* заголовок (title);
* описание (description);
* идентификатор пользователя – внешний ключ на сущность «Пользователь»

Сущность «Пространство» содержит данные о созданном пространстве. Атрибутами сущности являются:

* идентификатор пространства (id);
* имя (title);
* идентификатор пользователя – внешний ключ на сущность «Пользователь»

Сущность «Трата» содержит данные о созданной трате. Атрибутами сущности являются:

* идентификатор траты (id);
* количество потраченных денежных средств (amount);
* тип (type);
* дата (date);
* комментарий (comment);
* идентификатор пользователя – внешний ключ на сущность «Пользователь»

Сущность «Доход» содержит данные о созданной транзакции, приносящей доход. Атрибутами сущности являются:

* идентификатор дохода (id);
* количество заработанных денежных средств (amount);
* тип (type);
* дата (date);
* комментарий (comment);
* идентификатор пользователя – внешний ключ на сущность «Пользователь»

Сущность «Тип траты» содержит данные о созданном типе траты. Атрибутами сущности являются:

* идентификатор типа траты (id);
* имя типа (name);
* идентификатор пользователя – внешний ключ на сущность «Пользователь»

Сущность «Тип дохода» содержит данные о созданном типе траты. Атрибутами сущности являются:

* идентификатор типа дохода (id);
* имя типа (name);
* идентификатор пользователя – внешний ключ на сущность «Пользователь»

Результатом проделанного моделирования стала инфологическая модель базы данных, представленная на рисунке 2.2.

В полученной модели каждая сущность представляется прямоугольником, а связи между сущностями – линиями. Атрибуты каждой сущности отображаются внутри прямоугольника, а первичный ключ является первым полем в каждой таблице и отделяется от остальных полей линией.

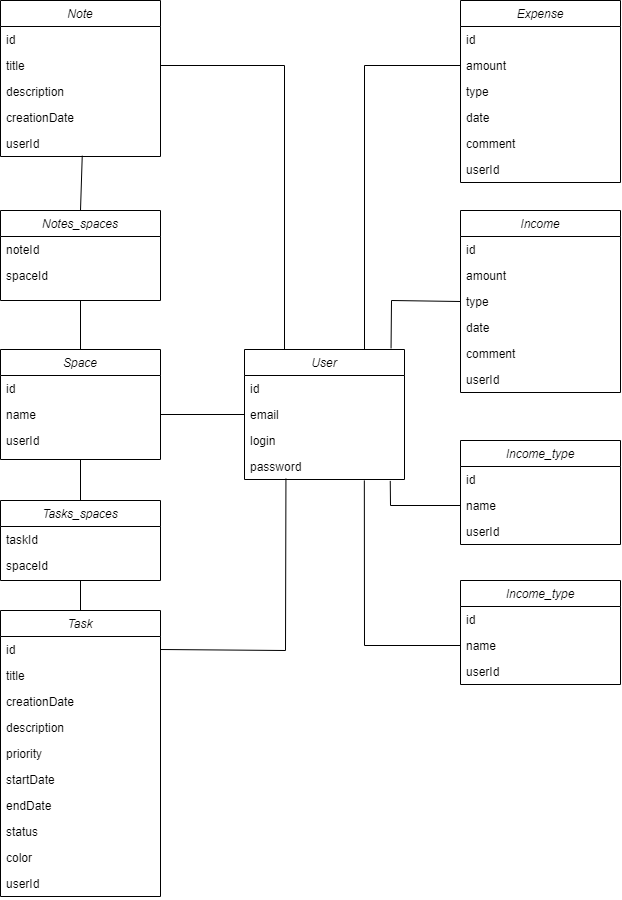


Рисунок 2.2 – Инфологическая модель базы данных

## Выбор языка и инструментов разработки

Выбор языка и инструментов разработки был сделан на основе проведённого анализа предметной области веб-приложения. Выбранные языки программирования и инструменты разработки включают:

* инструменты разработки:
  1. Visual Studio Code;
  2. DBeaver;
  3. Docker Desktop;
* серверная часть программного средства:
  1. язык программирования TypeScript;
  2. фреймворк NestJS;
* клиентская часть приложения:
  1. язык программирования TypeScript;
  2. фреймворк Angular;
* инструменты управления данными
  1. СУБД PosgreSQL;

### Инструменты разработки

Разработка клиентской и серверной части веб-приложения проводилась в редакторе кода Visual Studio Code(VS Code). Visual Studio Code – это легковесный редактор кода, разработанный компанией Microsoft. Он доступен для Windows, macOS и Linux, а также существует веб-версия, не требующая установки. Редактор быстро завоёвывает популярность среди разработчиков кода благодаря своей простоте использования, расширяемости и богатому функциональному набору.

Управление базой данных проводилось при помощи DBeaver и Docker Desktop. DBeaver – это универсальный инструмент управления базами данных с открытым исходным кодом, доступный для Windows, macOS и Linux. Он поддерживает широкий спектр баз данных, включая SQL, NoSQL и облачные базы данных. Помимо этого, DBeaver позволяет выполнять SQL-запросы, организовывать управление пользователями и правами доступа, сравнивать объекты базы данных, экспортировать и импортировать данные в различных форматах и производить визуальное редактирование данных.

Docker Desktop – это программное обеспечение, которое позволяет запускать Docker-контейнеры на локальном компьютере. Docker-контейнеры – это лёгкие, изолированные среды, которые содержат в себе необходимые для работы приложения, включая код, зависимости и операционную систему.

DBeaver и Docker Desktop можно использовать вместе для работы с PostgreSQL. DBeaver используется для подключения к PostgreSQL-контейнеру, запущенному при помощи Docker Desktop. Это позволяет разработчикам и администраторам баз данных работать с PostgreSQL-базами данных в контейнерах так же, как с локальными базами данных, что значительно упрощает и ускоряет разработку.

### Серверная часть веб-приложения

Для написания серверной части веб-приложения был выбран язык программирования TypeScript. **TypeScript** – это язык программирования, надмножество JavaScript, которое добавляет статическую типизацию. TypeScript разработан компанией Microsoft и является одним из самых быстрорастущих языков программирования в мире. TypeScript позволяет разработчикам писать более надежный, читаемый и поддерживаемый код JavaScript.

В качестве основного средства написания серверной части веб-приложения был выбран фреймворк NestJS. **NestJS** – это прогрессивный фреймворк Node.js, построенный на TypeScript. Он основан на принципах объектно-ориентированного программирования (ООП), функционального программирования (ФП) и реактивного функционального программирования (РФП). NestJS предоставляет набор инструментов и функций, упрощающих разработку высоко масштабируемых, надежных и легко обслуживаемых серверных приложений Node.js.

### Клиентская часть веб-приложения

Для разработки клиентской части веб-приложения так же, как и для серверной был использован высокоуровневый строго типизируемый язык программирования TypeScript, но уже с фреймворком Angular. Angular – это фреймворк с открытым исходным кодом для создания современных веб-приложений на языке TypeScript. Разработанный и поддерживаемый Google, Angular является одним из самых популярных фреймворков JavaScript, используемых сегодня. Он используется для создания различных веб-приложений, включая одностраничные приложения (SPA), прогрессивные веб-приложения (PWA) и корпоративные приложения.

# ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ

## Разработка архитектуры веб-приложения

В качестве архитектуры была выбрана трехуровневая клиент-серверная архитектура. Данная модель разработки приложений разделяет приложение на три уровня: клиентский уровень, бизнес-логику и уровень хранения данных.

На клиентском уровне находится интерфейс приложения, который взаимодействует с пользователем. Для взаимодействия с бизнес-логикой приложения используются запросы и ответы, которые позволяют получать данные для пользователей и выполнять определенные действия. В веб-приложении данным слоем можно считать веб-браузер.

Бизнес-логика находится на сервере и отвечает за обработку запросов и хранение данных приложения. Она также отвечает за логику приложения, например, за обработку запросов, валидацию данных, проверку прав доступа. На уровне данного слоя происходят все логические операции: математические расчеты, операции с данными, обращения к другим сервисам или хранилищам данных.

На третьем уровне находится уровень хранения данных, который отвечает за сохранение данных, используемых приложением. Этот уровень представлен сервером базы данных.

Трехуровневая модель упрощает разработку приложения, так как каждый уровень может быть разработан и поддерживаться независимо от других уровней. Она повышает масштабируемость приложения, каждый уровень может быть масштабирован независимо от других уровней. Кроме того, этот подход упрощает тестирование приложения и обеспечивает более высокую безопасность, так как бизнес-логика и уровень хранения данных могут быть отделены от клиентского уровня, что снижает вероятность возникновения уязвимостей в безопасности приложения.

### Диаграмма развёртывания

Для представления общей конфигурации и топологии веб-приложения в UML была создана диаграмма развертывания.

Диаграмма развертывания – это одна из диаграмм UML, которая описывает аппаратное и программное обеспечение системы, а также их взаимосвязи. Диаграмма развертывания позволяет проектировать и отображать физическую структуру системы, ее компоненты, взаимодействие и расположение на оборудовании.

Диаграмма развертывания включает в себя узлы – физические устройства или виртуальные машины, на которых размещаются компоненты системы, и связи между ними. Также на диаграмме могут быть изображены компоненты, их интерфейсы и порты, которые позволяют им взаимодействовать с другими компонентами и узлами.

Графически узел изображается в форме трехмерного куба, внутри которого указывается его имя и, возможно, дополнительная информация в виде помеченного значения.

На диаграмме развертывания помимо изображений узлов также отображаются отношения между ними. Они могут быть представлены в виде физических соединений между узлами или зависимостей между узлами и компонентами, которые также могут быть изображены на диаграмме. Соединения представляют собой вид ассоциации и обозначаются линиями без стрелок. Если линия присутствует на диаграмме, это указывает на необходимость создания физического канала для обмена информацией между соответствующими узлами.

Диаграмма развертывания используется для моделирования физической архитектуры системы, ее развертывания и конфигурации. Она позволяет увидеть, какие компоненты системы расположены на каких узлах, как они взаимодействуют и какие сетевые протоколы и протоколы взаимодействия используются. Разработанная диаграмма развертывания изображена на рисунке 3.1.

На диаграмме развертывания можно выделить два основных узла:

* сервер базы данных с СУБД PostgreSQL;
* сервер веб-приложения.

Сервер базы данных с СУБД PostgreSQL является центральным компонентом программной системы – в базе данных хранится вся информация веб-приложения. База данных развернута на отдельном от приложения сервере, так как предполагается, что она будет использоваться как единый источник информации во всей информационной системе.

Веб-сервис DFTClient, развернутый на отдельном сервере, будет служить единой точкой доступа к базе данных, и будет, по сути своей, представлять CRUD-интерфейс для работы с ней. Сервис будет предоставлять программный интерфейс, которым может быть использован во всем веб-приложении. На этом же сервере будет развернуто веб-приложение клиентского уровня, предоставляющее графический интерфейс для взаимодействия с веб-сервисом бизнес-логики.

Веб-сервис DFTClient представляет собой программный интерфейс для работы с базой данных и обслуживания запросов, реализованный с использование архитектурного подхода REST, использующим HTTP-запросы для получения, извлечения, размещения и удаления данных из ресурса.

Данный подход к созданию программы весьма гибкий и подходит для масштабирования. Одна из причин подобной гибкости заключается в том, что REST не хранит состояние. Это значит, что он не хранит информацию о пользовательских сессиях на сервере, а лишь предоставляет доступ к ресурсу. Каждый запрос должен содержать всю необходимую информацию, чтобы сервер мог выполнить его, без необходимости сохранять какое-либо состояние.

REST использует протокол HTTP для передачи данных между клиентом и сервером, и основывается на уникальных идентификаторах ресурсов (URI) для доступа к данным. Необходимая для обработки информация передается в качестве параметров HTTP-запроса.

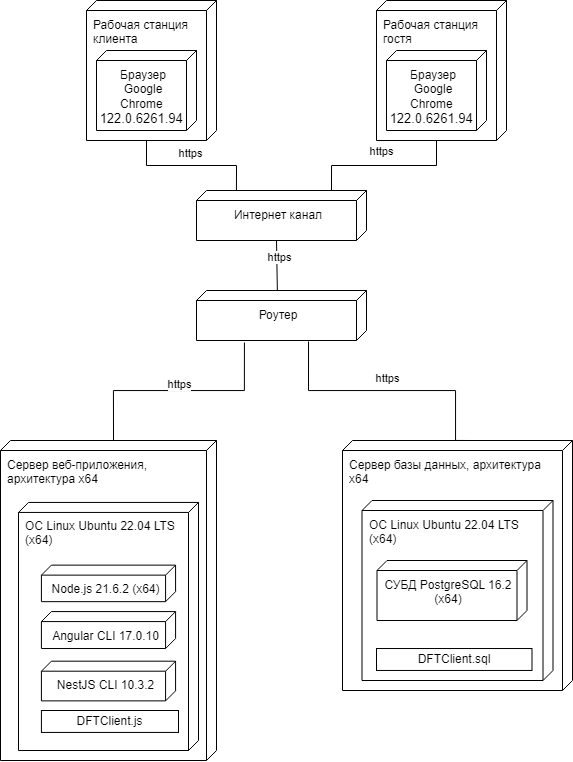


Рисунок 3.1 – Диаграмма развёртывания

## Проектирование даталогической модели базы данных

В качестве СУБД для веб-приложения была выбрана СУБД PostgreSQL. PostgreSQL – это мощная и надежная СУБД, которая предоставляет множество функций и возможностей для разработки широкого спектра приложений. Ее преимущества и гибкость делают ее привлекательным выбором для разработки данного веб-приложения [15].

Среди преимуществ PostgreSQL стоит назвать следующие:

* высокая надежность и стабильность, что делает PostgreSQL привлекательным выбором для критически важных приложений, где высокая доступность и надежность являются критически важными факторами;
* мощные функции и возможности, такие как поддержка хранимых процедур, триггеров, транзакций и многопоточности, которые делают его гибким и масштабируемым выбором для различных типов приложений;
* высокий уровень безопасности и контроля доступа к данным;
* соответствие стандартам ANSI SQL и поддержка множества дополнительных функций и расширений;
* активное сообщество разработчиков и множество доступных инструментов и библиотек.

Проектирование даталогической модели базы данных было произведено на основе разработанной в подразделе 2.3 инфологической, а также с учетом выбора PostgreSQL в качестве СУБД. Описание сущностей даталогической модели приведено в таблице 3.1.

Таблица 3.1 – Описание сущностей даталогической модели

|  |  |  |  |
| --- | --- | --- | --- |
| Название таблицы | Связь между таблицами в направлении «дочерняя-родительская» | Атрибут и его ключи | Тип данных |
| User | Нет | id – PK, NN, AI | integer |
| email – NN | varchar(50) |
| login – NN | varchar(50) |
| password – NN | varchar(50) |
| Task | связь «многие-к-одному» с таблицей «User»,  связь «многие-ко-многие» с таблицей «Space» | id – PK, NN, AI | integer |
| title – NN | varchar(128) |
| creationDate | timestamp |
| description | varchar(2048) |
| priority | varhar(20) |
| startDate | timestamp |
| endDate | timestamp |
| status | varchar(20) |
| userId – FK | integer |
| Space | связь «многие-к-одному» с таблицей «User», связь «многие-ко-многим» с таблицей «Task»,  связь «многие-ко-многим» с таблицей «Note», | id – PK, NN, AI | integer |
| name – NN | varchar(128) |
| userId – FK | integer |
| Note | связь «многие-к-одному» с таблицей «User»,  связь «многие-ко-многие» с таблицей «Space» | id –PK, NN, AI | integer |
| title – NN | varchar(128) |
| description | varchar(2048) |
| user – FK | integer |
| Expense | связь «многие-к-одному» с таблицей «User» | id – PK, NN, AI | integer |
| amount – NN | integer |
| type | varchar(20) |
| date – NN | timestamp |
| comment | varchar(2048) |
| userId – FK | integer |
| Income | связь «многие-к-одному» с таблицей «User» | id – PK, NN, AI | integer |
| amount – NN | integer |
| type | varchar(20) |
| date – NN | timestamp |
| comment | varchar(2048) |
| userId – FK | integer |
| Income\_type | связь «многие-к-одному» с таблицей «User» | id – PK, NN, AI | integer |
| name – NN | varchar(20) |
| userId – FK | integer |
| Expense\_type | связь «многие-к-одному» с таблицей «User» | id – PK, NN, AI | integer |
| name – NN | varchar(20) |
| userId – FK | integer |
| Notes\_spaces | связь «многие-к-одному» с таблицами «Note» и «Space», связующая таблица | noteId – FK | integer |
| spaceId – FK | integer |
| Notes\_spaces | связь «многие-к-одному» с таблицами «Task» и «Space», связующая таблица | taskId – FK | integer |
| spaceId – FK | integer |

Даталогическая модель обычно представляется в виде диаграммы сущностей-связей, на которой сущности изображаются в виде прямоугольников, а связи – в виде линий между сущностями, связывающими их атрибуты. Каждая сущность имеет свой уникальный идентификатор, который позволяет однозначно определить запись в таблице, а связи могут быть одним из трех типов: один-к-одному, один-ко-многим или многие-ко-многим.

Создание даталогической модели помогает уточнить требования к системе и определить наиболее эффективную структуру данных для хранения и управления информацией в БД. Полученная даталогическая модель представлена на рисунке 3.2.

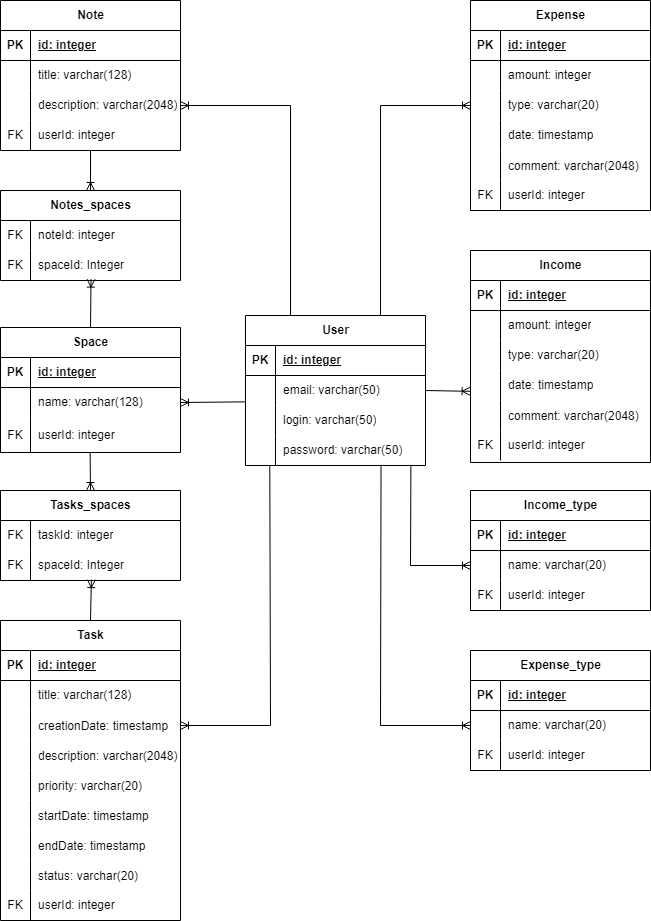


Рисунок 3.2 – Даталогическая модель базы данных

## Проектирование и разработка алгоритмов веб-приложения

### Обобщённый алгоритм веб-приложения

Обобщенный алгоритм работы веб-приложения представляет из себя типичный алгоритм работы веб-сервиса: после инициализации он ожидает запросы от клиентов и обрабатывает их, если клиент авторизован (имеет доступ). Если клиент не авторизован, то сервер перенаправляет его на страницу авторизации, где пользователь должен ввести свои логин и пароль. Затем генерируется токен доступа и пользователь перенаправляется на первоначальную страницу, передавая токен как параметр запроса. После успешной авторизации, пользователь может выполнять запросы к ресурсам или операциям, к которым у него есть доступ.

Сами операции получения списков, добавления, удаления или редактирования информации в большинстве случаев представляют собой обычный CRUD (Create-Read-Update-Delete) интерфейс для работы с сущностями в базе данных.

### Алгоритм расчёта данных для сложного процента с пополнением

Веб-приложение предоставляет удобный инструмент для расчета сложного процента, позволяя пользователям прогнозировать будущий рост инвестиций. Пользователю требуется заполнить поля:

* стартовый капитал;
* срок инвестирования;
* единица времени срока инвестирования;
* процентная ставка;
* период реинвестирования;
* размер дополнительного пополнения;
* период дополнительного пополнения.

В качестве единицы времени срока инвестирования может быть выбран

месяц или год. Период реинвестирования и дополнительного пополнения могут принимать значения:

* месяц;
* квартал;
* полгода;
* год;

При желании пользователь может выбрать расчёт без реинвестирования.

После заполнения происходит проверка всех полей на валидность данных, и далее начинается вычисление данных, состоящее из пяти основных блоков вычисления:

* ежемесячных данных;
* годовых данных;
* суммы всех дополнительных вложений;
* дохода;
* итоговой суммы;

Алгоритм вычисления данных для сложного процента с пополнением представлен на рисунке 3.3.

### Алгоритм выявления психологического состояния пользователя по цветам, ассоциированных с задачами

В приложении у пользователя есть возможность при создании задачи выбрать цвет, с которым эта задача ассоциируется. Пользователь может выбрать один из 16581375 цветов. После этого данный алгоритм будет находить максимально приближенный цвет из десяти основных из «психологии цветов»:

* Чёрный;
* Белый;
* Красный;
* Голубой;
* Зелёный;
* Жёлтый;
* Фиолетовый;
* Коричневый;
* Оранжевый;
* Розовый.

Для оптимизации в базе данных выбранные цвета хранятся в HEX формате (в виде строки). Для ассоциации выбранного цвета с одним из десяти, требуется знать в каких пропорциях в этом цвете присутствует:

* красный;
* зелёный;
* голубой.

Таким образом алгоритм включает в себя конвертацию из формата

HEX в формат RGB. Для оптимизации этот процесс реализован через побитовые сдвиги. Реализация алгоритма представлена на рисунке 3.4.

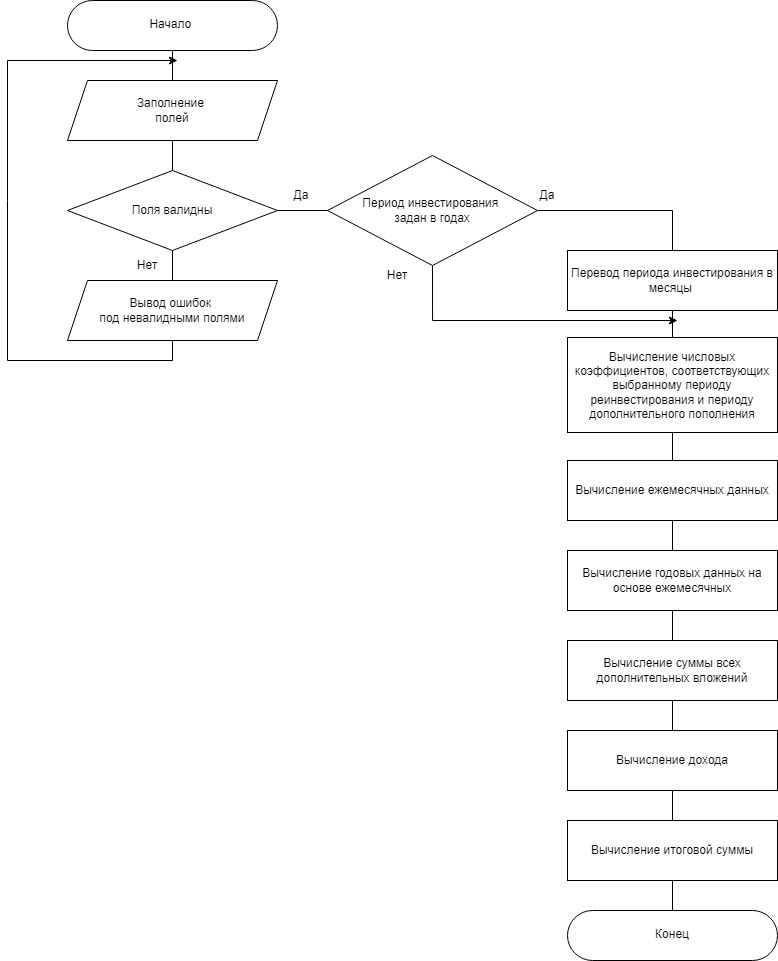


Рисунок 3.3 – Схема алгоритма расчёта сложного процента с пополнением

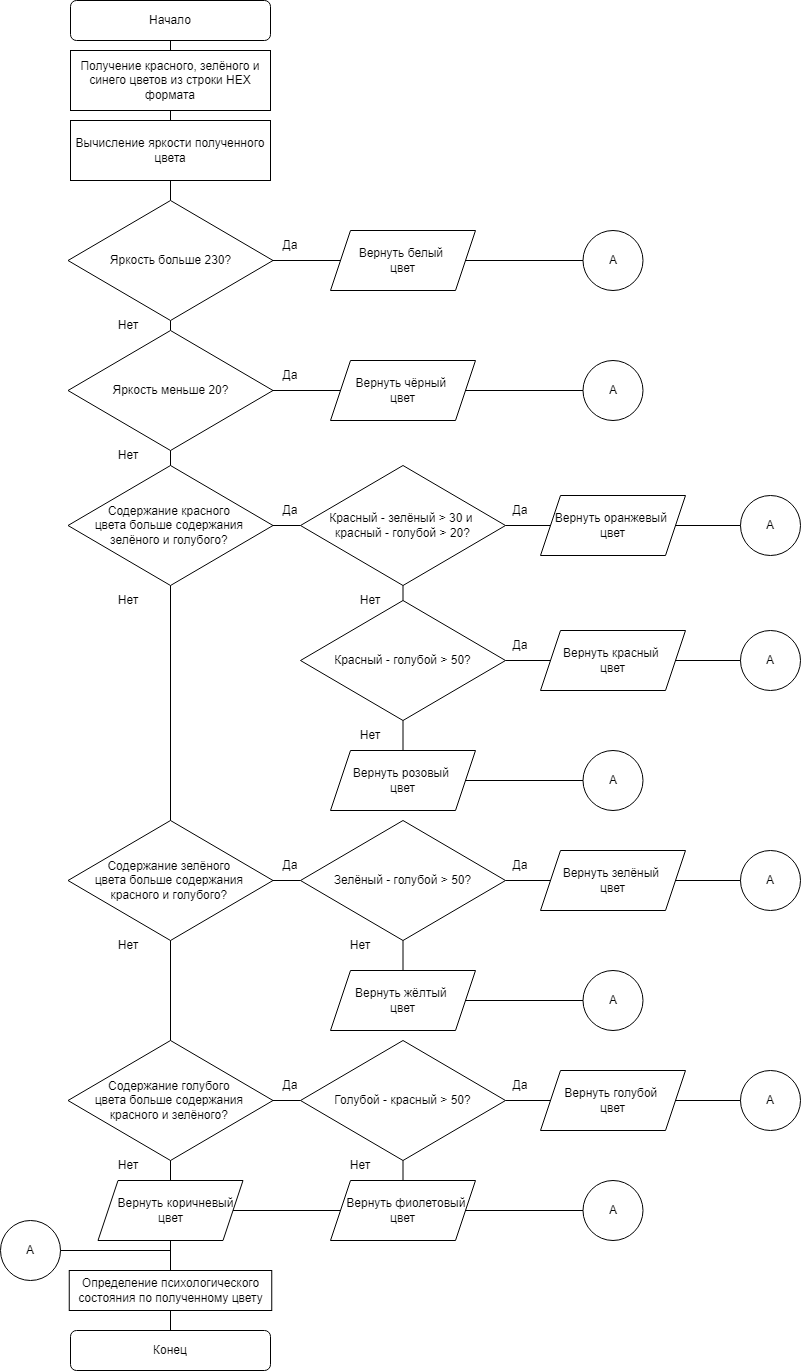


Рисунок 3.4 – Вычисление состояния пользователя по цвету задачи

# Разработка ВЕБ-ПРИЛОЖЕНИЯ

## Разработка API и моделей передачи данных между клиентом и сервером

При разработке API и моделей передачи данных между клиентом сервером важно учитывать не только функциональные требования, но и безопасность, производительность, а также масштабируемость приложения. В данном случае, использование фреймворков Angular на стороне клиента, NestJS на стороне сервера и postgreSQL в качестве базы данных позволяет создать надёжное и эффективное приложение. Так же выбранные инструменты обеспечивают легкую, гибкую и масштабируемую архитектуру, которая позволяет клиентам и серверам обмениваться информацией в формате, который понятен обеим сторонам.

Angular и NestJS предоставляют мощный стек инструментов для разработки современных веб-приложений. Angular является фронтенд фреймворком, который позволяет создавать динамические и отзывчивые интерфейсы пользователя, в то время как NestJS является серверным фреймворком, основанным на Node.js, который обеспечивает возможности для разработки серверных приложений с использованием TypeScript.

Одной из ключевых возможностей связки Angular и NestJS является механизм обмена данными между фронтендом и бэкендом. Взаимодействие между Angular и NestJS может осуществляться с помощью передачи данных в формате JSON. Angular может выполнять HTTP-запросы к API, предоставляемому NestJS, и обмениваться данными в формате JSON.

Для удобства работы с данными в формате JSON в связке Angular и NestJS можно использовать библиотеку HttpClientModule, предоставляемую Angular, для выполнения HTTP-запросов, и библиотеку NestJS для обработки запросов и отправки данных в формате JSON обратно на клиентский интерфейс.

Таким образом, связка Angular и NestJS позволяет создавать современные веб-приложения, в которых эффективно осуществляется обмен данными между клиентской и серверной частями приложения в формате JSON.

Для упрощения процесса обмена данными между слоями веб-приложения было принято решение использовать паттерн DTO [17]. DTO-класс представляет собой простой класс, который содержит только поля данных и не содержит логики или поведения. Вместо того чтобы передавать сущности напрямую из базы данных в представление, используя DTO, осуществляется передача только необходимых полей. Таким образом, DTO служат в качестве промежуточного звена между бизнес-моделью веб-приложения и REST-контроллерами, которые принимают и возвращают данные в формате JSON.

Для начала представим описание конечных точек API веб-приложения:

* /auth/signIn – аутентификация пользователей;
* /auth/signUp – регистрация пользователей;
* управление:
  1. /expense – затратами пользователя;
  2. /expenseType – типами затрат пользователя;
  3. /income – доходами пользователя;
  4. /incomeType – типами доходов пользователя;
  5. /note – записями пользователя;
  6. /space – пространствами пользователя;
  7. /task – задачами пользователя;
  8. /user – профилем пользователя;

## Разработка ORM моделей веб-приложения

Для создания ORM моделей и управления их взаимодействием с базой данных в приложении, использующем Angular на клиентской стороне и NestJS на серверной, применяется TypeORM. Этот инструмент позволяет описывать сущности базы данных в виде классов TypeScript, которые затем могут быть использованы в приложении NestJS. TypeORM автоматически генерирует SQL-запросы, что позволяет разработчику избежать необходимости вручную обрабатывать данные и конвертировать объекты, обеспечивая при этом переносимость приложения на различные SQL базы данных, включая PostgreSQL.

TypeORM использует декораторы TypeScript для определения связей между объектами, а также для указания различных параметров и ограничений сущностей. Ниже приведен пример конфигурации класса User, который соответствует ранее спроектированной даталогической модели базы данных:

@Entity('user')

export class UserEntity {

@PrimaryGeneratedColumn()

id: number;

@Column()

email: string;

@Column({ unique: true })

login: string;

@Column()

password: string;

@OneToMany(() => TaskEntity, (task) => task.user)

tasks: TaskEntity[];

@OneToMany(() => NoteEntity, (note) => note.user)

notes: NoteEntity[];

@OneToMany(() => SpaceEntity, (space) => space.user)

spaces: SpaceEntity[];

@OneToMany(() => IncomeEntity, (income) => income.user)

incomes: IncomeEntity[];

@OneToMany(() => IncomeTypeEntity, (income\_type) => income\_type.user)

income\_types: IncomeTypeEntity[];

@OneToMany(() => ExpenseEntity, (expense) => expense.user)

expenses: ExpenseEntity[];

@OneToMany(() => ExpenseTypeEntity, (epxense\_type) => epxense\_type.user)

expense\_types: ExpenseTypeEntity[];

}

В этом примере UserEntity является основной сущностью, которая будет отображаться в таблицу user в базе данных. Декоратор @Entity('user') указывает, что класс является сущностью TypeORM и должен быть связан с таблицей user. @PrimaryGeneratedColumn() определяет первичный ключ, который будет автоматически генерироваться. @Column используется для определения столбцов таблицы, а @OneToMany указывает на связи один ко многим с другими сущностями, такими как TaskEntity, NoteEntity и так далее, которые представляют различные аспекты данных пользователя. Это позволяет легко управлять и получать доступ к связанным данным пользователя в приложении.

Используя ранее спроектированную даталогическую модель базы данных, были разработаны следующие классы объектно-реляционных моделей:

* user;
* task;
* note;
* space;
* income;
* income\_type;
* expense\_type.

TypeORM значительно упрощает взаимодействие с базой данных, предоставляя для этого специальный объект, называемый репозиторием. Репозиторий принимает в себя интерфейс сущности и возвращает объект уже настроенный для комфортной работы с ней.

Вместо того чтобы использовать методы репозитория непосредственно, в приложении они используются внутри классов-сервисов, ответственных за реализацию бизнес-логики и операции над данными. Они используют репозитории для доступа к данным и предоставляют высокоуровневые методы, которые инкапсулируют сложную логику и дополнительную валидацию. Ниже приведен пример кода сервиса для работы с сущностями User.

@Injectable()

export class UserApiService {

constructor(

@InjectRepository(UserEntity)

private usersRepository: Repository<UserEntity>,

private passwordService: PasswordService,

@Inject(forwardRef(() => IncomeTypeApiService))

private incomeTypeApiService: IncomeTypeApiService,

private expenseTypeApiService: ExpenseTypeApiService,

) {}

public async isSameUserExists(login: string): Promise<boolean> {

return !!(

await this.usersRepository.find({

where: { login },

})

).length;

}

public async getUserByLogin(login: string): Promise<UserEntity> {

return await this.usersRepository.findOne({ where: { login } });

}

public async getUserById(id: number): Promise<UserEntity> {

return await this.usersRepository.findOne({ where: { id } });

}

public async addUser(user: Omit<User, 'id'>): Promise<UserEntity> {

const userEntity = await this.usersRepository.save(user);

await this.setUserDefaultExpenseTypes(userEntity);

await this.setUserDefaultIncomeTypes(userEntity);

return;

}

public async updateUser(

id: User['id'],

updateParams: UserEdit,

): Promise<UserEntity> {

const user = await this.getUserById(id);

if (!updateParams.newPassword || !updateParams.oldPassword) {

const userEntity: UserEntity = {

...user,

email: updateParams.email,

};

return this.usersRepository.save(userEntity);

}

if (

!(await this.passwordService.validatePassword(

updateParams.oldPassword,

user.password,

))

) {

throw new UnauthorizedException([HttpErrorCode.InvalidOldPassword]);

}

const hashedPassword = await this.passwordService.hashPassword(

updateParams.newPassword,

);

const userEntity: UserEntity = {

...user,

password: hashedPassword,

email: updateParams.email,

};

return this.usersRepository.save(userEntity);

}

private async setUserDefaultIncomeTypes(

user: UserEntity,

): Promise<UserEntity> {

const defaultIncomeTypes: IncomeTypeEntity[] = await Promise.all(

INCOME\_DEFAULT\_TYPE\_NAMES.map((name) =>

this.incomeTypeApiService.createIncomeType(user.id, { id: null, name }),

),

);

const userEntity: UserEntity = {

...user,

income\_types: defaultIncomeTypes,

};

return await this.usersRepository.save(userEntity);

}

private async setUserDefaultExpenseTypes(

user: UserEntity,

): Promise<UserEntity> {

const defaultExpenseTypes: IncomeTypeEntity[] = await Promise.all(

EXPENSE\_DEFAULT\_TYPE\_NAMES.map((name) =>

this.expenseTypeApiService.createExpenseType(user.id, {

id: null,

name,

}),

),

);

const userEntity: UserEntity = {

...user,

income\_types: defaultExpenseTypes,

};

return await this.usersRepository.save(userEntity);

}

}

## Разработка валидации данных

В веб-приложениях, использующих Angular, валидация данных на клиентской стороне является ключевым элементом для обеспечения корректности пользовательского ввода перед отправкой данных на сервер. Angular предоставляет мощные инструменты для реализации валидации через реактивные формы, которые позволяют управлять состоянием формы и валидацией в реактивном стиле.

Реактивные формы в Angular используют набор встроенных валидаторов, таких как Validators.required, Validators.email, Validators.minLength, и другие. Однако, одной из сильных сторон Angular является возможность создания **кастомных валидаторов**, которые позволяют реализовать специфические правила валидации, соответствующие бизнес-логике приложения.

Ниже приведен пример создания кастомного валидатора для проверки старого и нового паролей в реактивной форме:

export function oldNewPasswordValidator(): ValidatorFn {

return (control: AbstractControl): ValidationErrors | null => {

const form = control as FormGroup;

const oldPassword = form.controls['oldPassword'].value;

const newPassword = form.controls['newPassword'].value;

if (oldPassword && newPassword && oldPassword === newPassword) {

form.controls['newPassword'].setErrors({ oldNewPasswordMatch: true });

return { oldNewPasswordMatch: true };

}

form.controls['newPassword'].setErrors(null);

return null;

};

}

В этом примере oldNewPasswordValidator является функцией, которая возвращает валидатор. Этот валидатор проверяет, что новый пароль не совпадает со старым. Если условие не выполняется, валидатор устанавливает ошибку oldNewPasswordMatch для поля нового пароля. Это позволяет отобразить соответствующее сообщение об ошибке в пользовательском интерфейсе и предотвратить отправку формы с некорректными данными.

Использование кастомных валидаторов в Angular значительно повышает гибкость и мощь валидации, позволяя разработчикам точно настраивать поведение форм в соответствии с требованиями проекта. Это обеспечивает более высокий уровень контроля над пользовательским вводом и повышает качество вводимых данных.

Cтоит отметить, что реактивные формы предоставляют разработчикам возможность реагировать на изменения в форме в реальном времени. Это достигается благодаря использованию потоков данных (streams) и операторов библиотеки RxJS, которые позволяют создавать сложные валидационные проверки, реагирующие на изменения в нескольких полях или даже разных формах.

К примеру, можно реализовать валидатор, который будет проверять уникальность введенного пользователем email, обращаясь к серверу и проверяя, не зарегистрирован ли уже такой email. Такой подход позволяет предотвратить отправку формы с email, который уже используется другим пользователем, и тем самым улучшить пользовательский опыт.

export function uniqueEmailValidator(userService: UserService): AsyncValidatorFn {

return (control: AbstractControl): Observable<ValidationErrors | null> => {

return userService.isEmailTaken(control.value).pipe(

map(isTaken => (isTaken ? { emailTaken: true } : null)),

catchError(() => of(null))

);

};

}

В этом примере uniqueEmailValidator является асинхронным валидатором, который использует сервис UserService для проверки уникальности email. Если email уже занят, валидатор добавляет ошибку emailTaken к полю формы.

Важно также упомянуть, что Angular предоставляет возможность группировать валидаторы с помощью функции Validators.compose, что позволяет создавать более сложные валидационные правила, комбинируя несколько валидаторов в один.

Реактивные формы и механизмы валидации в Angular обеспечивают разработчикам мощный инструментарий для создания интерактивных и надежных веб-приложений, где корректность данных играет важную роль. Благодаря гибкости и расширяемости системы валидации, можно легко адаптировать формы под любые требования, обеспечивая при этом высокий уровень удобства для пользователей.

## Разработка механизма динамической смены языков

Так как приложение разработано для широкого круга пользователей, в него встроена поддерживает 14 языков:

* Русский
* Английский
* Испанский
* Арабский
* Французский
* Португальский
* Японский
* Турецкий
* Немецкий
* Корейский
* Итальянский
* Польский
* Хинди
* Персидский

Многоязычность в веб-приложениях играет ключевую роль в обеспечении доступности и удобства для пользователей по всему миру. В современной разработке веб-приложений, особенно тех, которые ориентированы на международный рынок, возможность легко переключаться между языками является не просто желательной, но и часто необходимой функцией.

Angular, как один из ведущих фреймворков для разработки веб-приложений, предлагает разработчикам эффективные инструменты для реализации многоязычности. Библиотека ngx-translate дополняет Angular, предоставляя гибкий и мощный механизм для управления переводами. Это позволяет разработчикам поддерживать одну кодовую базу, избегая при этом необходимости создавать отдельные версии приложения для каждого языка.

С помощью ngx-translate, разработчики могут определить переводы в JSON-файлах, которые затем могут быть легко загружены и использованы в приложении. Это упрощает процесс добавления новых языков и обновления существующих переводов. Кроме того, ngx-translate поддерживает динамическую загрузку переводов, что означает, что переводы могут быть загружены "на лету", без необходимости перезагружать приложение.

Динамическая смена языков с ngx-translate не только улучшает пользовательский опыт, но и способствует оптимизации производительности приложения, так как переводы загружаются по мере необходимости, а не все сразу. Это особенно важно для приложений с большим количеством текста и поддержкой множества языков.

Далее представлен пример использования ngx-translate в Angular приложении.

Сначала необходимо установить ngx-translate и плагин для загрузки файлов перевода, например @ngx-translate/http-loader:

npm install @ngx-translate/core @ngx-translate/http-loader –save

Дале необходимо произвести настройку модуля:

export function HttpLoaderFactory(http: HttpClient) {

return new TranslateHttpLoader(http);

}

@NgModule({

imports: [

HttpClientModule,

TranslateModule.forRoot({

loader: {

provide: TranslateLoader,

useFactory: HttpLoaderFactory,

deps: [HttpClient]

}

})

],

})

export class AppModule { }

После этого для каждого поддерживаемого языка создаются JSON-файлы с переводами, например en.json, ru.json и т.д., где ключи соответствуют текстам, которые нужно перевести.

В результате получаем возможность использовать переводы в компонентах. Это достигается за счёт использования интерполяции и translate pipe в html файлах:

<p>{{ 'HELLO' | translate }}</p>

В целом, интеграция ngx-translate в Angular приложения является эффективным решением для достижения многоязычности, обеспечивая при этом легкость в поддержке и расширении языковых опций. Это делает Angular в сочетании с ngx-translate предпочтительным выбором для разработки многоязычных веб-приложений, ориентированных на глобальную аудиторию.

## Разработка механизма динамической смены цветовых тем веб приложения

Разработка механизма динамической смены цветовых тем в веб-приложении на Angular с использованием `lithium-ngx-material-theming` представляет собой интересную задачу, которая может значительно улучшить пользовательский опыт. Это позволяет пользователям выбирать цветовую схему интерфейса в соответствии с их предпочтениями или условиями окружающей среды, например, темную тему для ночного времени или светлую тему для дневного.

Angular является мощным инструментом для создания динамичных и адаптивных веб-приложений, и `lithium-ngx-material-theming` расширяет его возможности, предоставляя гибкий механизм для управления темами Material Design. Вот как можно реализовать динамическую смену тем с помощью этой библиотеки:

Для начала необходимо установить библиотеку и настроить ее в вашем Angular проекте. Это включает в себя создание нескольких темовых файлов SCSS, которые определяют цветовые палитры для различных элементов интерфейса.

Далее необходимо создайте сервис, который будет отвечать за хранение текущей выбранной темы и предоставление методов для ее изменения. Сервис может использовать `LocalStorage` или `Cookies` для сохранения выбора пользователя.

В компонентах Angular можно использовать директиву, которая связана с сервисом тем, для применения текущей темы к элементам интерфейса. Это может быть реализовано через динамическое добавление класса к корневому элементу приложения.

После этого нужно предоставить пользователю интерфейс для выбора темы. В приложении это реализовано в виде выпадающего меню. При выборе темы сервис должен обновить текущую тему и применить ее ко всем компонентам.

Также можно реализовать функционал, который будет автоматически менять тему в зависимости от системных настроек пользователя, например, использовать темную тему, если в операционной системе включен соответствующий режим.

Ниже приведён пример кода, который демонстрирует создание сервиса для управления темами:

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class ThemeService {

private currentTheme: string;

constructor() {

this.currentTheme = localStorage.getItem('theme') || 'light-theme';

}

setTheme(theme: string) {

localStorage.setItem('theme', theme);

this.currentTheme = theme;

document.body.className = theme;

}

getTheme() {

return this.currentTheme;

}

}

Таким образом используя lithium-ngx-material-theming, разработчики могут создавать веб-приложения, которые не только выглядят привлекательно, но и обладают высокой степенью пользовательской адаптивности. Это улучшает взаимодействие с приложением и делает его более доступным и комфортным для широкого круга пользователей.

# ТЕСТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

Тестирование разрабатываемого веб-приложения является важным этапом, который играет решающую роль в обеспечении высокого качества программного обеспечения. Оно позволяет выявить и исправить ошибки и проблемы в приложении до его внедрения в production среду, минимизируя потенциальные негативные последствия для конечных пользователей.

Основные цели тестирования включают в себя проверку:

* функциональности;
* безопасности;
* производительности;
* совместимости.

По доступности исходного кода процессы проведения тестирования подразделяются на:

− метод «черного ящика»:

− метод «белого ящика»;

− метод «серого ящика».

Метод «черного ящика» основан на проверке внешнего поведения системы без информации о внутренней реализации и структуре кода. В данном методе тестирующий рассматривает систему как «черный ящик», не обращая внимания на внутренние детали и механизмы работы. Вместо этого фокусируется на входных данных и ожидаемых выходных результатах. Тестирующий имеет доступ только к интерфейсу или внешнему поведению системы, через которое взаимодействует с ней.

Суть метода «белого ящика» заключается в анализе внутренней структуры и логики программы, а также тестировании каждой отдельной функции и ветви кода. В данном методе тестирующий имеет доступ к полному исходному коду программы и использует эту информацию для создания тестовых случаев. Основная цель метода «белого ящика» состоит в обеспечении покрытия кода тестами, чтобы каждая ветвь кода, условие и функция были протестированы.

Метод «серого ящика» является комбинацией метода «черного ящика» и метода «белого ящика». В отличие от метода «черного ящика», где тестирующий не имеет информации о внутренней реализации системы, в методе «серого ящика» тестировщик имеет некоторое базовое знание о внутренней структуре и логике системы. В методе «серого ящика» тестировщик анализирует внешнее поведение системы, как в методе «черного ящика», но также использует некоторую информацию о внутренних механизмах и структуре кода. Тестирование проводится на основе понимания системы и ее компонентов, что позволяет обнаружить проблемы, связанные с внутренней логикой, алгоритмами и структурами данных.

После проведения анализа различных методов тестирования было принято решение проводить ручное тестирование программного средства по методу «черного ящика». Ручное тестирование – это процесс проверки программного обеспечения, выполняемый тестировщиком вручную без использования автоматизированных средств или скриптов. В этом методе тестирования тестировщик взаимодействует с приложением, проверяет его функциональность, проводит различные сценарии использования и анализирует результаты.

Сценарии взаимодействия с веб-приложением, которые подверглись тестированию, представлены в таблице 5.1.

Таблица 5.1 – Сценарии проведения тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| Тестируемая функциональность | Описание сценария | Ожидаемый результат | Результат |
| 1. Регистрация нового пользователя | 1. Открыть веб-браузер. 2. Перейти по адресу <https://localhost:4200>. 3. Автоматически произойдёт переход на страницу регистрации. 4. Заполнить необходимые поля корректными данными. 5. Отправить запрос на регистрацию. | 1. Открылся веб-браузер. 2. Открылась страница регистрации. 3. Поля ввода были заполнены корректными данными. 4. Регистрация прошла успешно. 5. Произошёл переход на главную страницу. | Тест пройден успешно |
| 1. Регистрация нового пользователя. Проверка ввода некорректных данных. | 1. Открыть веб-браузер. 2. Перейти по адресу <https://localhost:4200>. 3. Автоматически произойдёт переход на страницу регистрации. 4. Заполнить необходимые поля некорректными данными. 5. Отправить запрос на регистрацию. | 1. Открылся веб-браузер. 2. Открылась страница регистрации. 3. Поля ввода были заполнены некорректными данными. 4. Регистрация не прошла. 5. Невалидно заполненные поля были подсвечены, под ними отобразились текста ошибок | Тест пройден успешно |
| 1. Аутентификация пользователя | 1. Открыть веб-браузер. 2. Перейти по адресу <https://localhost:4200>. 3. Перейти на страницу аутентификации. 4. Заполнить необходимые поля корректными данными. 5. Отправить запрос на аутентификацию. | 1. Открылся веб-браузер. 2. Открылась страница регистрации. 3. Поля ввода были заполнены корректными данными. 4. Регистрация прошла успешно. 5. Произошёл переход на главную страницу. |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Экономическое обоснование разработки и реализации на рынке эффективности веб-приложения “Ежедневник и финансовый трекер”

## Характеристика веб-приложения

## Ежедневник и финансовый трекер: Ваш путь к успеху.

Целью разработки веб-приложения "Ежедневник и финансовый трекер" является помощь пользователям в планировании задач, учете времени и финансов.

Приложение подойдёт широкому кругу людей: студентам, офисным работникам, руководителям предприятий, предпринимателям, любому человеку, ценящему своё время.

Программное стредство обладает богатым спектром возможностей и позоволяет ставить цели и задачи; планировать день, неделю и месяц; эффективно управлять своим временем; вести бюджет; контролировать расходы и при этом отслеживать выполнение всех действий в удобном формате.

Удобный формат работы с веб-приложением достигается за счёт реализации мощного функционала, представленного инструментами такими как: календарь, список дел, заметки, таймер, а также графики.

## Расчет инвестиций в разработку веб-приложения

### Основная заработная плата

Расчет затрат на основную заработную плату команды разработчиков осуществляется исходя из состава и численности команды, размера месячной заработной платы каждого участника команды, а также трудоемкости работ, выполняемых при разработке веб-приложения отдельными исполнителями по формуле

, (7.1)

где Кпр – коэффициент премий и иных стимулирующих выплат;

n – количество исполнителей, занятых разработкой веб-приложения;

Зчi – часовая заработная плата i-го исполнителя, руб.;

ti – трудоемкость работ, выполняемых i-м исполнителем, ч.

Количество рабочих часов в месяце принято равным 160 часам.

В организации, являющейся местом прохождения преддипломной практики, следующее соотношение позиция – месячная зарплата(руб.):

1. Бизнес-аналитик – 2800
2. Системный архитектор – 3300
3. Инженер-программист – 2780
4. Тестировщик – 1800
5. Дизайнер – 2000

Премия составляет 55 процентов от месячной заработной платы.

Результаты расчетов основной заработной платы участников команды разработки программного средства приведены в таблице 1.

Таблица 1 – Расчёт затрат на основную заработную плату участников команды разработки программного средства

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория исполнителя | Месячный оклад, руб. | Часовой оклад, руб. | Трудоёмкость работ, ч. | Итого, руб. |
| Бизнес-аналитик | 2800 | 17.5 | 60 | 1050 |
| Системный архитектор | 3300 | 20.625 | 145 | 2990.625 |
| Инженер-программист | 2780 | 17.375 | 180 | 3127.5 |
| Тестировщик | 1800 | 11.25 | 75 | 843.75 |
| Дизайнер | 2000 | 12.5 | 100 | 1250 |
| Итого | | | | 9261.875 |
| Премия(50%) | | | | 4630.9375 |
| Всего затрат на основную заработную плату разработчиков | | | | 13892.82 |

### Дополнительная заработная плата

Дополнительная заработная плата включает выплаты, предусмотренные законодательством о труде, и рассчитывается по формуле:

(7.2)

где – затраты на основную заработную плату разработчиков, руб.;

– норматив дополнительной заработной платы, (15%).

### Отчисления на социальные нужды

Отчисления на социальные нужды (в фонд социальной защиты населения и на обязательное страхование) определяются в соответствии с действующим законодательством по формуле

, (7.3)

Где – норматив отчислений на социальные нужды (34,6%).

Отчисления на социальные нужды составят:

### 

### Прочие затраты

Прочие затраты включают аренду помещения, амортизацию основных средств, затраты на инфраструктуру, хранение данных и пр.

Прочие затраты рассчитываются по формуле:

где – норматив прочих затрат, (35%).

Таким образом, прочие затраты составят:

### Расходы на реализацию

где Hp – норматив расходов на реализацию(3%)

Таким образом расходы на реализацию составят:

### Общая сумма

Общая сумма затрат на разработку веб-приложения рассчиты-вается по формуле:

Полученные значения затрат на разработку по основным статьям представлены в таблице 2

Таблица 2 – Затраты на разработку программного средства

|  |  |  |
| --- | --- | --- |
| Статья затрат | Формула/таблица для рассчёта | Сумма, руб. |
| Основная заработная плата команды разработки | Таблица 1 | 13892.82 |
| Дополнительная заработная плата команды разработки |  | 2083.93 |
| Отчисления на социальные выплаты |  | 5527.96 |
| Прочие затраты |  | 4862.49 |
| Расходы на реализацию |  | 416.79 |
| Общая сумма затрат на разработку |  | 26784 |

## Расчет экономического эффекта от реализации веб-приложения на рынке

Экономический эффект организации-разработчика программного  
средства представляет собой прирост чистой прибыли от продажи платных  
лицензий.

Цена лицензии была определена на основе цен на схожее программное обеспечение, представленное на рынке. Цены аналогов, таких как “TickTick”, “ToDoist”, “ClickUp” и “Notion” колеблятся в диапазоне от 30 до 70 долларов США. С учётом текущего курса для данного ПО целесообразным будет установить цену годовой лицензии в размере 135 рублей. Число лицензий, которые возможно продать, оценивается в 1000 экземпляров.

Сумма НДС рассчитывается по формуле:

где Ндс – ставка налога на добавленную стоимость согласно действующему законодательству (Ндс = 20%).

Ц – цена копии(лицензии) ПС;

N – количество конпий(лицензий) ПС, реализуемоу за год, шт;

Таким образом, налог составит:

Расчет прироста чистой прибыли осуществляется по формуле

где – ставка налога на прибыль, 20%;

– отпускная цена копии(лицензии) ПС;

N – количество конпий(лицензий) ПС, реализуемоу за год, шт;

НДС – сумма налога на добавленную стоимость;

– рентабельность продаж копий(лицензий)(40%);

## Расчет показателей экономической эффективности разработки и реализации веб-приложения на рынке

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

Т.к. сумма инвестиций на разработку меньше суммы годового экономического эффекта, т.е. инвестиции окупятся менее чем за год, оценка экономической эффективности инвестиций в разработку ПО осуществляется с помощью расчёта рентабельности инвестиций по формуле:

* 1. ,где – затраты на разработку и реализацию.
  2. Расчёт рентабельности составит:

## Вывод по результатам расчета

Показатели:

1. Высокий спрос: Рынок приложений для планирования и управления финансами активно развивается. Анализ конкурентов показал, что существует большая аудитория пользователей, ищущих подобные решения.
2. Уникальная ценность: Приложение "Ежедневник и финансовый трекер" обладает широким функционалом и удобным интерфейсом, что выгодно отличает его от аналогов.
3. Доступная цена: Цена приложения конкурентоспособна и соответствует его ценности.

На основании полученных показателей, можно сделать вывод о высокой потенциальной эффективности разработки и реализации приложения "Ежедневник и финансовый трекер" на массовом рынке.