МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

Web-приложение «Планировщик бюджета»

Выполнил студент Мацуев Илья Михайлович

(Ф.И.О.)

Руководитель проекта асс. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2020

# Реферат

Пояснительная записка курсового проекта состоит из 41 страница, 29 рисунков, 3 приложений, 6 источников литературы.

Основная цель курсового проекта: разработка web-приложения «Планировщик бюджета».

В первом разделе рассматриваются основные технологии, которые использовались в разработке данного приложения, постановка задач, а также обзор аналогов.

Во втором разделе описана архитектура курсового проекта.

В третьем разделе предоставлена информация о разработке приложения, объектах базы данных.

Четвертый раздел содержит описание процесса упаковки курсового проекта в docker, руководство пользователя для разработанного клиентского приложения.

В пятом разделе представлены результаты тестирования приложения.

В заключении описывается результат курсового проектирования и задачи, которые были решены в ходе разработки приложения.

Содержание

[Реферат 2](#_Toc59048184)

[Введение 5](#_Toc59048185)

[1 Постановка задачи 6](#_Toc59048186)

[1.1 Алгоритмы решения 6](#_Toc59048187)

[1.2 Обзор прототипов 7](#_Toc59048188)

[1.2.1 Slack 7](#_Toc59048192)

[1.2.2 Flock 8](#_Toc59048193)

[1.2.3 Workplace 9](#_Toc59048194)

[2 Проектирование программного средства 11](#_Toc59048195)

[2.1. Проектирование архитектуры проекта 11](#_Toc59048196)

[2.2 Разработка диаграммы вариантов использования 14](#_Toc59048197)

[2.3 Структура моделей и классов 16](#_Toc59048198)

[2.4 Структура курсового проекта 18](#_Toc59048199)

[3 Разработка программного средства 21](#_Toc59048200)

[3.1 Описание функциональных возможностей приложения 21](#_Toc59048201)

[3.2 Разработка базы данных 21](#_Toc59048202)

[3.3 Описание алгоритма аутентификации пользователя 26](#_Toc59048203)

[3.4 Процесс отправки сообщений 26](#_Toc59048204)

[4 Руководство программиста 28](#_Toc59048205)

[4.1 Руководство пользователя 29](#_Toc59048206)

[4.1.1 Регистрация пользователя 29](#_Toc59048207)

[4.1.2 Вход пользователя в систему 30](#_Toc59048208)

[4.1.3 Отправка сообщений 31](#_Toc59048209)

[5 Тестирование 33](#_Toc59048210)

[Заключение 36](#_Toc59048211)

[Список литературы 37](#_Toc59048212)

[ПРИЛОЖЕНИЕ А 38](#_Toc59048213)

[ПРИЛОЖЕНИЕ Б 39](#_Toc59048214)

[ПРИЛОЖЕНИЕ В 41](#_Toc59048215)

[ПРИЛОЖЕНИЕ Г 42](#_Toc59048216)

# Введение

В данном курсовом проекте разработано WEB-приложение «Планировщик бюджета». Логически оно разделено на две части: серверную, написанную на ASP.NET Core [1], и клиентскую, которая написана с помощью JavaScript-библиотеки React с использованием Apollo Client [2], физически же WEB-приложение представляет собой одно приложение.

Чтобы стать богатым человеком, нужно уделять внимание не только тому, сколько вы зарабатываете, но и тому, сколько тратите. Человек постоянно старается извлечь максимальную выгоду из любой сделки и любого случая, поэтому всегда пользовались спросом приложения и утилиты, позволяющие улучшить уже существующие процессы менеджмента. Один из таких процессов – это подход к мониторингу состояния человека или компании. Реализовать такой подход в настоящее время достаточно просто — необходимо начать отслеживать свои расходы и доходы с помощью одного из специальных приложений для мобильных телефонов или персональных компьютеров. В сети существует огромное количество подобного рода приложений, однако очень многие из них не подходят для повседневного использования или попросту неудобны. Этот проект призван перенять лучшие качества и нивелировать худшие качества аналогов с целью создания приложения, которое было бы лучше других как со стороны пользователя, так и со стороны бизнеса.

Подводя итоги: целью курсового проекта является разработка WEB-приложения «Планировщик бюджета», которое поможет вам осуществить вашу мечту. Я сделал выбор своей курсовой темы в связи с актуальностью данной задачи в современном мире. Данное приложение содержит тот функционал, который поможет сохранить вам ваши миллионы для лучшей жизни.

# Постановка задачи

Перед началом разработки необходимо определить цели, задачи, все варианты использования программного средства. Для этого необходимо построить диаграмму использования, которая приведена во второй главе, раздел 2.2.

Постановка задачи ­­– это процесс формулировки назначения программного обеспечения и основных требований к нему. Описываются функциональные требования, определяющие функции, которые должно выполнять программное обеспечение, и эксплуатационные требования, определяющие характеристики его функционирования.

Этап постановки задачи заканчивается разработкой технического задания с принятием основных проектных решений.

Главная задача данного приложение – начать контролировать свои расходы и доходы. Проблема нашего поколения заключается в том, что люди не понимают, куда и почему так быстро траться деньги. И чтобы решить данную проблему, первое, что необходимо сделать, – это взять её под контроль, в этом вам и поможет разработанное мною веб-приложение «Планировщик бюджета». Данное веб-приложение должно выполнять основные функции по созданию учётной записи пользователя, его личного счёта, возможность добавлять и редактировать расходы и доходы, а также смотреть отчетность. В данном курсовом проекте требовалось реализовать следующие задачи:

* Cохранение рабочей информации в централизованной базе данных;
* Регистрация одиночных и повторяющихся денежных транзакций;
* Возможность получения оповещений об отчетах бюджета за месяц;
* Анализ бюджета и транзакций на основе расходов и доходов за промежуток времени;
* Настройка параметров работы приложения для пользователя.

# Алгоритмы решения

После окончательной постановки задач необходимо приступить к их решению.

Когда требуется решить задачу, первое, что должно быть сделано, ­­– исследовать ее. Необходимо установить, что дано, что нужно сделать. Надо быть уверенным, что задача решаема, иначе последующие шаги могут состоять из напрасных усилий.

После исследования задачи рассматриваются шаги, которые требуются для решения, и порядок, в котором они должны быть выполнены. Шаги, которые необходимы для решения задачи и их последовательность, ­­– это и есть алгоритм.

Когда мы будем удовлетворены тем, как выполняется задание, мы можем снова вернуться к предыдущим шагам для их улучшения. И так этот цикл может повторяться несколько раз, пока мы не будем удовлетворены полностью. В компьютерной терминологии такой цикл называют циклом разработки программного обеспечения.

# Обзор прототипов



# Money Lover

В качестве одного из прототипов было выбрано приложение «MoneyLover», представленное на рисунке 1.1. Существуют версии этого приложения для всех платформ, в том числе и веб-версия.

Приложение предназначено для учёта и анализа собственных расходов и доходов. Приложение имеет довольно красивый и понятный любому пользователю интерфейс и функционал.

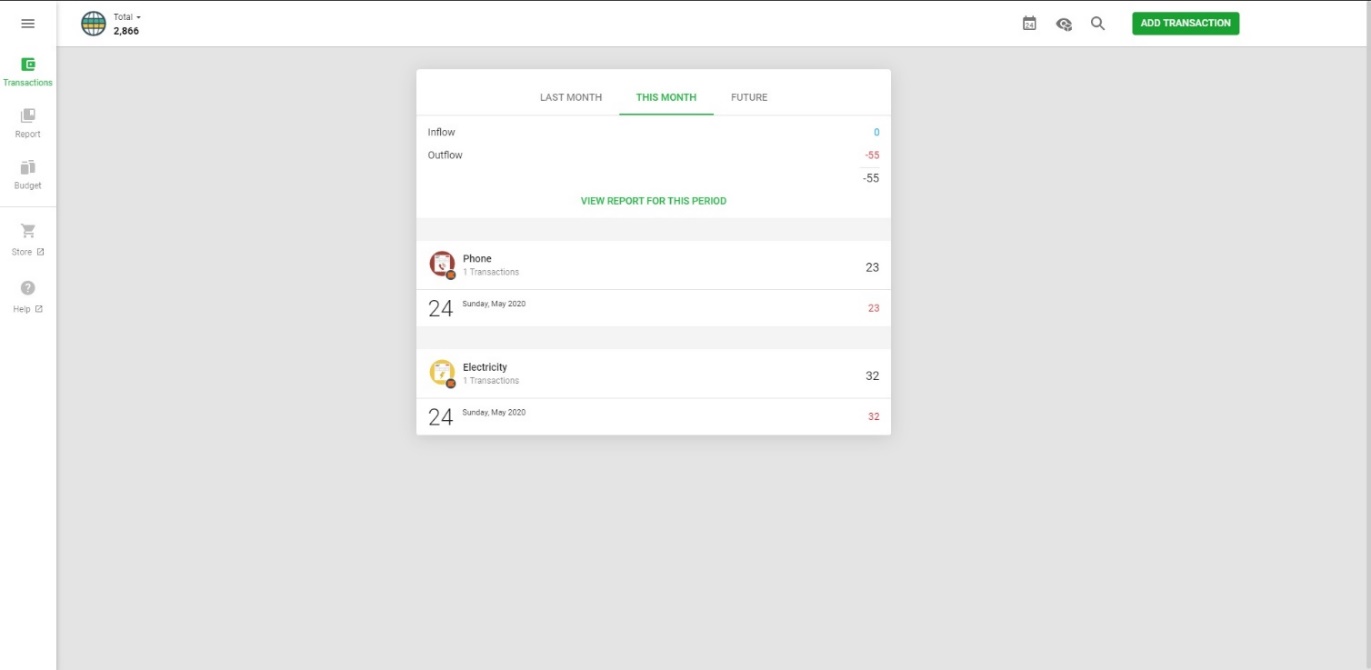
****

Рисунок 1.1 – Веб-приложение «MoneyLover»

# Spendee

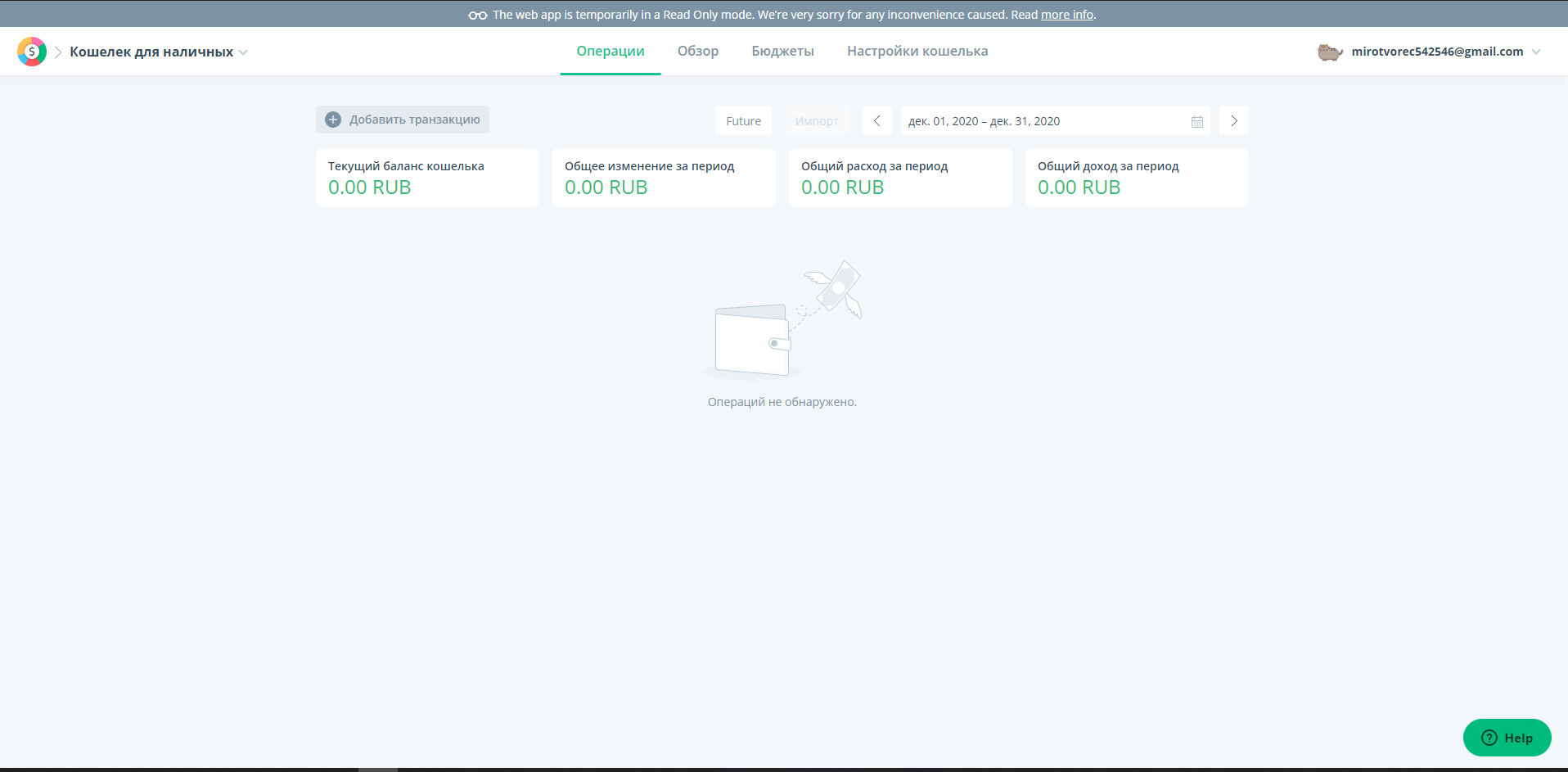
Еще одним отличным аналогом является приложение «Spendee», представленное на рисунке 1.2 с версиями под все мобильные устройства, в том числе и веб-версия. Отличительной особеностью является возможность привязки банковских счетов для автоматического заполнения транзакций по счетам. 

Рисунок 1.2 – Веб-приложение «Spendee»

Приложение имеет возможность создавать кошельки в разной валюте, добавлять транзакции вручную, а так же просматривать статистику за определенный промежуток времени.

# Проектирование программного средства

# 2.1. Проектирование архитектуры проекта

Хорошая архитектура ­­– это прежде всего выгодная архитектура, делающая процесс разработки и сопровождения программы более простым и эффективным. Программу с хорошей архитектурой легче расширять и изменять, а также тестировать, отлаживать и понимать.

То есть, на самом деле можно сформулировать список вполне разумных и универсальных критериев:

1. Масштабируемость (Scalability) ­­­– возможность расширять систему и увеличивать ее производительность, за счет добавления новых модулей;
2. Ремонтопригодность (Maintainability) ­­– изменение одного модуля не требует изменения других модулей;
3. Заменимость модулей (Swappability) ­­– модуль легко заменить на другой;
4. Возможность тестирования (Unit Testing) ­­– модуль можно отсоединить от всех остальных и протестировать / починить;
5. Переиспользование (Reusability) ­­– модуль может быть переиспользован в других программах и другом окружении;
6. Сопровождаемость (Maintenance) ­­– разбитую на модули программу легче понимать и сопровождать.

Архитектура ­­– это организация системы, воплощенная в ее компонентах, их отношениях между собой и с окружением.

В первую очередь следует, конечно же, стремиться к тому, чтобы модули были предельно автономны. Поэтому проводить ее нужно таким образом, чтобы модули изначально слабо зависели друг от друга.

Для разработки данного проекта была выбрана архитектура клиент-сервер, которая позволяет разделять функционал и вычислительную нагрузку между клиентскими приложениями и серверными приложениями.

Практические реализации такой архитектуры называются клиент-серверными технологиями. Каждая технология определяет собственные или использует имеющиеся правила взаимодействия между клиентом и сервером, которые называются протоколом обмена (протоколом взаимодействия).

Для разработки данного проекта была выбрана трёхзвенная архитектура, которая реализуется на основе модели сервера приложений, где сетевое приложение разделено на две и более частей, каждая из которых может выполняться на отдельном компьютере.

Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате. В этом случае двухзвенная клиент-серверная архитектура становится трехзвенной (three-tier, 3-tier).

Представление данных ­­– на стороне клиента. Прикладной компонент ­­– на выделенном сервере приложений (как вариант, выполняющем функции промежуточного ПО). Управление ресурсами ­­– на сервере БД, который и представляет запрашиваемые данные.

Как правило, третьим звеном в трехзвенной архитектуре становится сервер приложений, таким образом компоненты распределяются как показано на рисунке 2.1.

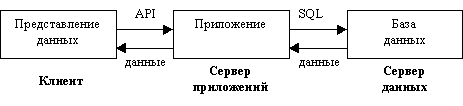


Рисунок 2.1 – Трехзвенная клиент-серверная архитектура

Представление данных ­­– на стороне клиента. Прикладной компонент ­­– на выделенном сервере приложений (как вариант, выполняющем функции промежуточного ПО). Управление ресурсами ­­– на сервере БД, который и представляет запрашиваемые данные.

В решении курсового проекта для сервера была использована MVC архитектура, которая представлена на рисунке 2.2.

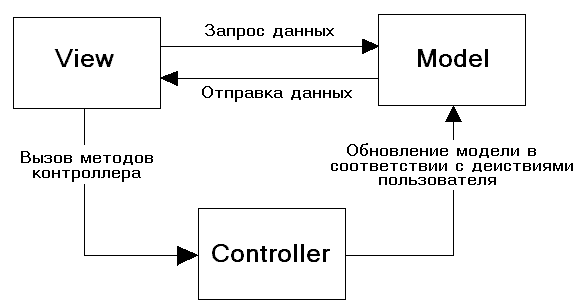


Рисунок 2.2 – Общая схема MVC архитектуры

Концепция паттерна (шаблона) MVC (model - view - controller) предполагает разделение приложения на три компонента:

Контроллер (controller) представляет класс, обеспечивающий связь между пользователем и системой, представлением и хранилищем данных. Он получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления. Представление (view) — это собственно визуальная часть или пользовательский интерфейс приложения. Как правило, html-страница, которую пользователь видит, зайдя на сайт. Модель (model) представляет класс, описывающий логику используемых данных.

Благодаря этому реализуется концепция разделение ответственности, в связи с чем легче построить работу над отдельными компонентами. Кроме того, вследствие этого приложение обладает лучшей тестируемостью.

Приложение-клиент представляет собой веб-приложение, написанное на языках JavaScipt, HTML, CSS придерживаясь технологии React [2].

React — популярная технология для Frontend-разработки, как и Angular — фреймворк для JavaScript, но это только View (Представление) слой, а значит, у нас в распоряжении только V от MVC — Model-View-Controller [3] (Модель — Представление — Контроллер) архитектуры. React часто упоминается среди других фреймворков, но все, что он дает — это View). Архитектура React приложения показана на следующем рисунке 2.3.

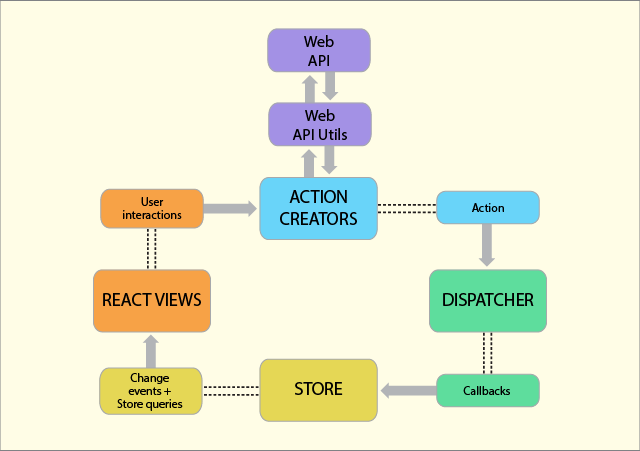


Рисунок 2.3 ­– Архитектура приложения с React архитектурой

В стандартной архитектуре Flux следующие компоненты:

1. Actions — помощники, которые передают данные в Dispatcher.
2. Dispatcher — получает эти действия и передает полезную нагрузку зарегистрированным callback-ом.
3. Stores — действуют как контейнеры для состояния приложения и логики. Реальная работа приложения происходит в Stores. Stores, зарегистрированные для прослушивания действий Dispatcher, будут соответственно и обновлять View.
4. Controller Views — React компоненты захватывают состояние из Stores, а затем передают дочерним компонентам.

Контроллеры в MVC и Flux различаются. Здесь контроллеры являются Controller-View и находятся на самой вершине иерархии. View — это React компоненты.

Так же в моем курсовом проекте я использовал Apollo Client. Это пакет, позволяющий облегчить работу по получению, обработке и отправке GraphQL запросов на GraphQL сервер. Основной особенность Apollo Client является несколько типов кэширования информации, что позволяет уменьшить количество запросов на сервер.

Весь функционал, как правило, находится в Store. В Store выполняется вся работа и сообщается Dispatcher, какие события/действия он прослушивает.

Когда происходит событие, Dispatcher отправляет “полезную нагрузку” в Store, который зарегестрирован для прослушивания конкретно этого события. Теперь в Store необходимо обновить View, что в свою очередь вызывает действие. Дествие, точно также как и имя, тип события и много другое известны заранее.

View распространяет Action через центральный Dispatcher, и это будет отправлено в различные Stores. Эти Stores будут отображать бизнес-логику приложения и другие данные. Store обновляет все View.

Наиболее хорошо это работает совместно со стилем программирования React, тогда Store отправляет обновления без необходимости подробно описывать, как изменять представления между состояниями.

Это доказывает, что шаблон проектирования Flux следует за однонаправленным потоком данных. Action, Dispatcher, Store и View — независимые узлы с конкретными входными и выходными данными. Данные проходят через Dispatcher, центральный хаб, который в свою очередь управляет всеми данными.

# Разработка диаграммы вариантов использования

UML – это унифицированный графический язык моделирования для описания, визуализации, проектирования и документирования объектно-ориентированных систем. UML призван поддерживать процесс моделирования ПС на основе объектно-ориентированного подхода, организовывать взаимосвязь концептуальных и программных понятий, отражать проблемы масштабирования сложных систем. Модели на UML используются на всех этапах жизненного цикла ПС, начиная с бизнес-анализа и заканчивая сопровождением системы.

Диаграмма вариантов использования веб-ресурса – диаграмма, отражающая отношения между актерами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Диаграммы вариантов использования применяются при бизнес-анализе для моделирования видов работ, выполняемых организацией, и для моделирования функциональных требований к ПС при ее проектировании и разработке. Построение модели требований при необходимости дополняется их текстовым описанием.

Диаграмму вариантов использования есть смысл строить во время изучения технического задания, она состоит из графической диаграммы, описывающей действующие лица и прецеденты, а также спецификации, представляющего собой текстовое описание конкретных последовательностей действий (потока событий), которые выполняет пользователь при работе с системой.

Спецификация затем станет основой для тестирования и документации, а на следующих этапах проектирования она дополняется и оформляется в виде диаграммы (в рамках ICONIX используется диаграмма последовательности, но в UML для этого имеются также диаграммы деятельности).

На диаграмме использования изображаются:

1. Актёры ­­– группы лиц или систем, взаимодействующих с нашей системой.
2. Варианты использования (прецеденты) ­­– сервисы, которые наша система предоставляет актёрам.
3. Комментарии.
4. Отношения между элементами диаграммы.

Сплошные линии на диаграмме представляют собой отношения ассоциации, отражающие возможность использования актёром прецедента.

Варианты использования могут быть связаны друг с другом тремя видами связей: обобщением (generalization), расширением (extend relationship) и включением (include relationship). Действующие лица также могут быть связаны друг с другом с помощью связей обобщения (generalization).

После того, как определен набор вариантов использования, можно приступать к составлению сценариев. Сценарии должны описываться с точки зрения пользователя, при этом важно описывать взаимодействие пользователя с элементами интерфейса.

В результате анализа требований, сформулированных на этапе планирования, была разработана диаграмма вариантов использования. С ее использованием будет проходить дальнейшая разработка функциональных возможностей веб-приложения.

Диаграмма вариантов использования представлена на рисунке 2.4.

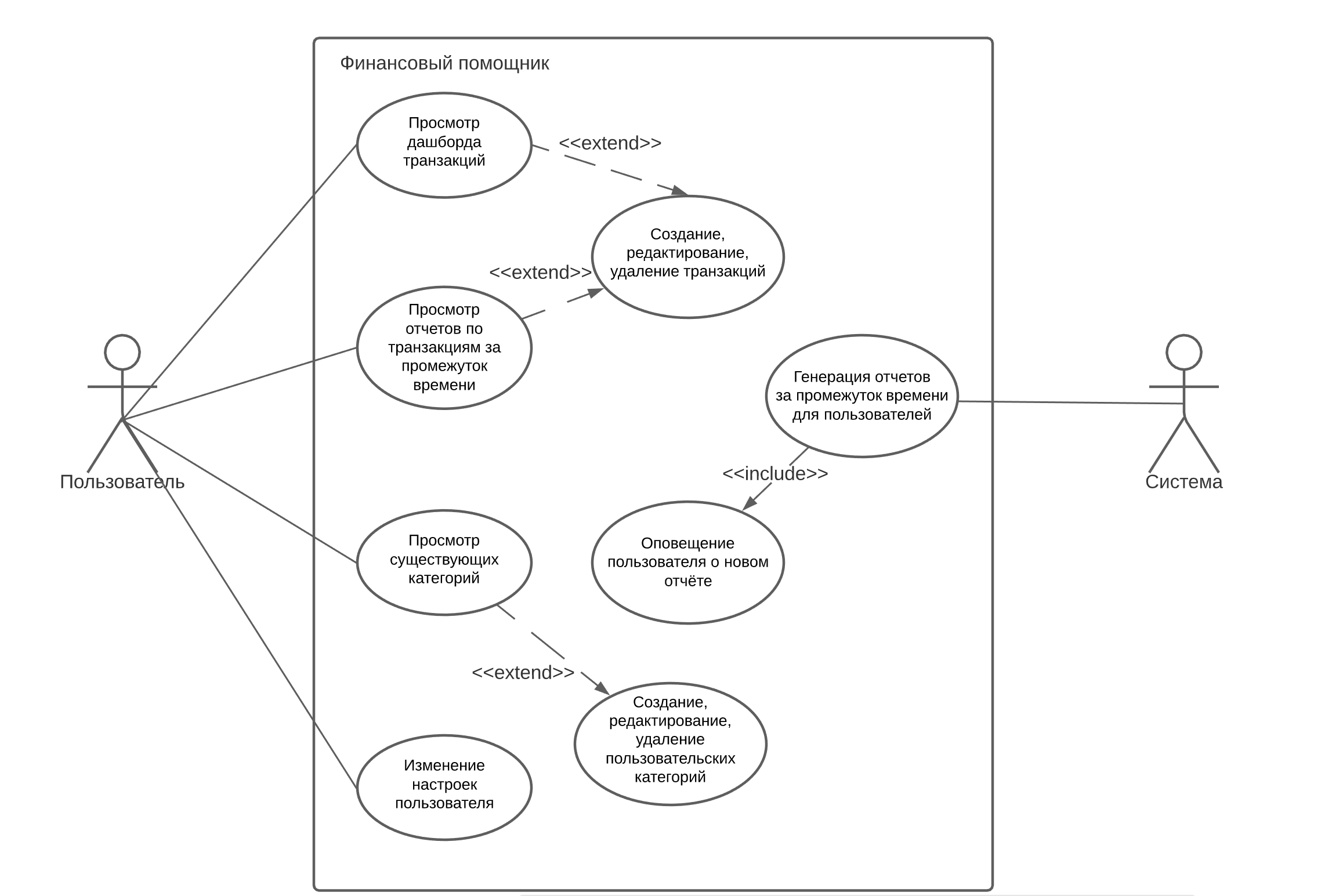


Рисунок 2.4 – Диаграмма вариантов использования

Из диаграммы вариантов использования видно действующих лиц, их взаимодействие с системой и ожидаемую функциональность системы. С ее использованием будет проходить дальнейшая разработка веб-приложения.

# Структура моделей и классов

Как известно, база данных – это хранилище структурированной информации, пассивное по своей сути. Бизнес-логика приложения реализуется где-то вне базы, в виде «набора действий для достижения требуемого результата». В случае внесения изменений в хранимый набор данных результатом должно стать новое состояние базы.

Если предметная область автоматизации представляет собой систему взаимодействующих значений, то ее можно описать ER-моделью.

Диаграмма классов является ключевым элементом редактора UML-диаграмм, поскольку зачастую приложения генерируются именно с диаграммы классов. Диаграмма классов – это набор статических, декларативных элементов модели. Диаграммы классов могут применяться и при прямом проектировании, то есть в процессе разработки новой системы, и при обратном проектировании – описании существующих и используемых систем. Информация с диаграммы классов напрямую отображается в исходный код приложения. Таким образом, диаграмма классов – конечный результат проектирования и отправная точка процесса разработки.

В соответствии с моделью базы данных и поставленными задачами была разработана модель классов, представленная на рисунке 2.5.

TODO: Создать свою диаграмму классов и вставить

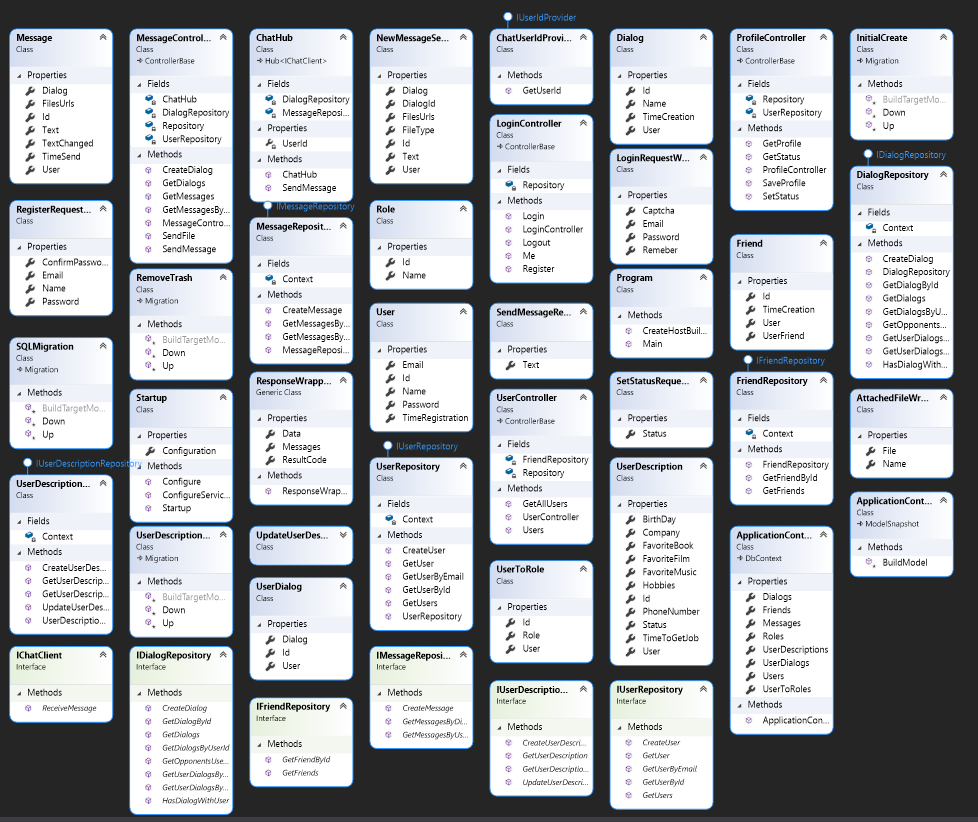


Рисунок 2.5 – Разработанная модель классов

Такая модель не только образует логическую структуру базы данных, но обладает всеми свойствами программы, каковой, по сути, и является – в вычислительной среде, образованной методами упомянутых абстрактных сущностей.

# Структура курсового проекта

Разработанный веб-ресурс построен по архитектуре клиент-сервер, которые взаимодействуют по протоколу HTTP.

Такая архитектура была выбрана с целью максимального отделения представления от бизнес-логики, то есть в том, что любое пользовательское приложение вначале делится на два модуля ­­– один из которых отвечает за реализацию собственно самой бизнес логики (Модель), а второй ­­– за взаимодействие с sпользователем (Пользовательский Интерфейс или Представление).

Серверная часть была написана с помощью фреймворка ASP .NET Core MVC.

Платформа ASP.NET Core MVC представляет собой фреймворк для создания сайтов и веб-приложений с помощью реализации паттерна MVC.

Благодаря этому реализуется концепция разделение ответственности, в связи с чем легче построить работу над отдельными компонентами. Кроме того, вследствие этого приложение обладает лучшей тестируемостью.

При разработке серверной части веб-ресурса были созданы проекты на основе MVC архитектуры. Структура проекта представлена на рисунке 2.6.

TODO: свою структуру

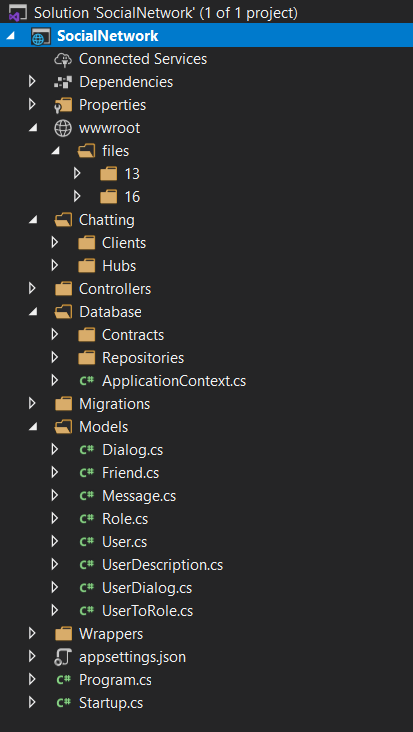


Рисунок 2.6 – Структура серверной части веб-ресурса

Для написания клиентской части веб-ресурса использовался фреймворк React [2]. Файл со всеми используемыми модулями находится в приложении Б.

React ­­– это инструмент для создания пользовательских интерфейсов. Его главная задача ­­– обеспечение вывода на экран того, что можно видеть на веб-страницах. React значительно облегчает создание интерфейсов благодаря разбиению каждой страницы на небольшие фрагменты. Мы называем эти фрагменты компонентами.

Разработанная с помощью вышеописанных технологий клиентская часть имеет структуру, представленную на рисунке 2.7.

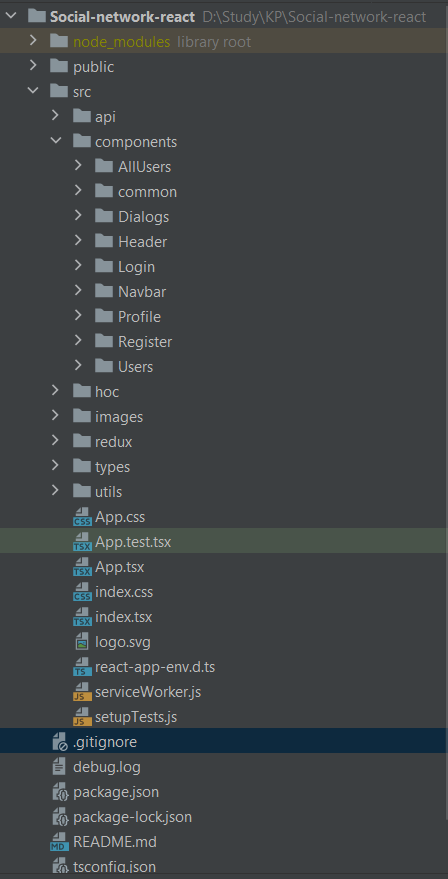


Рисунок 2.7 – Структура клиентской части веб-ресурса

Компонентно-ориентированный подход, возможность с легкостью изменять имеющиеся компоненты и переиспользовать код превращают React разработку в непрерывный процесс улучшения. Таким образом, ничто не мешает использовать их снова и снова в проектах разного типа. Возможность с легкостью заново использовать уже имеющийся код повышает скорость разработки, упрощает процесс тестирования, и, как результат, понижает затраты.

# Разработка программного средства

На основе сформированных требований разрабатывается база данных для того, чтобы определить состав информации, которая должна находиться в базе данных для решения всего комплекса задач предметной области, а также выявить логические взаимосвязи данных, размещаемых в БД. Состав и взаимосвязи данных должны быть отображены моделью данных предметной области. На основе этой модели необходимо определить соответствующую ей логическую структуру базы данных для выбранной СУБД. Процесс разработки необходимо осуществлять в соответствии с концепцией логической организации данных выбранной СУБД.

# Описание функциональных возможностей приложения

Разрабатываемый веб-ресурс представляет собой веб-ресурс, позволяющий удобно контролировать жизненный цикл IT проекта. В результате была создана база данных, которая удовлетворяет поставленным задачам.

Данная реализация достаточно проста и универсальна, с ее помощью в дальнейшем можно изменять и дополнять структуру базы данных.

Функционально программное средство должно выполнять следующие задачи:

TODO: Написать свои функции

1. Сохранение рабочей информации в централизованной базе данных
2. Просмотр и изменение профиля пользователя
3. Просмотр и создание диалогов
4. Просмотр и создание сообщений
5. Пересылка файлами между пользователями
6. Просмотр всех пользователей
7. Добавление и удаление друзей
8. Авторизация и регистрация

# Разработка базы данных

Для разработки и управления базой данных курсового проекта использовалась объектно-реляционная система управления базами данных «MS SQL Server».

Выбор данного системного продукта произошел по нескольким очевидным причинам: легкая инсталляция, бесплатна для разработки, раннее полученные знания по использованию данной базы данных.

База данных — представленная в объективной форме совокупность самостоятельных материалов (статей, расчётов, нормативных актов и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины. Реляционная база данных — база данных, основанная на реляционной модели данных.

Microsoft SQL Server [4] – система управления реляционными базами данных, разработанная корпорацией Microsoft. Основной используемый язык запросов – Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

Для базы данных было разработано 7 таблиц.

Логическая схема базы данных со всеми таблицами и связями представлена в приложении А.

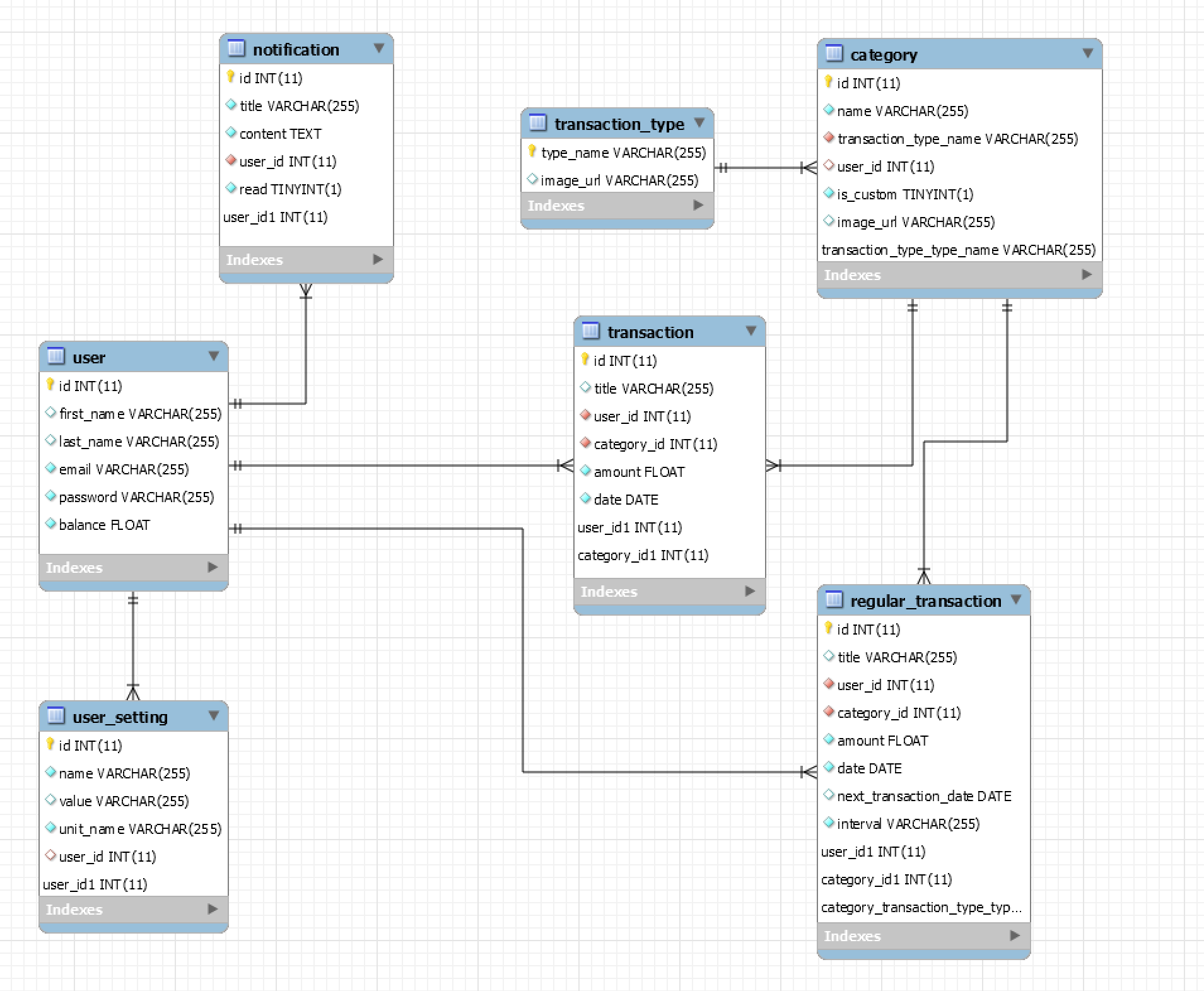


Рисунок 3.1 – Структура базы данных

На рисунке 3.2 представлена структура таблицы User, которая содержит записи зарегистрированных пользователей.

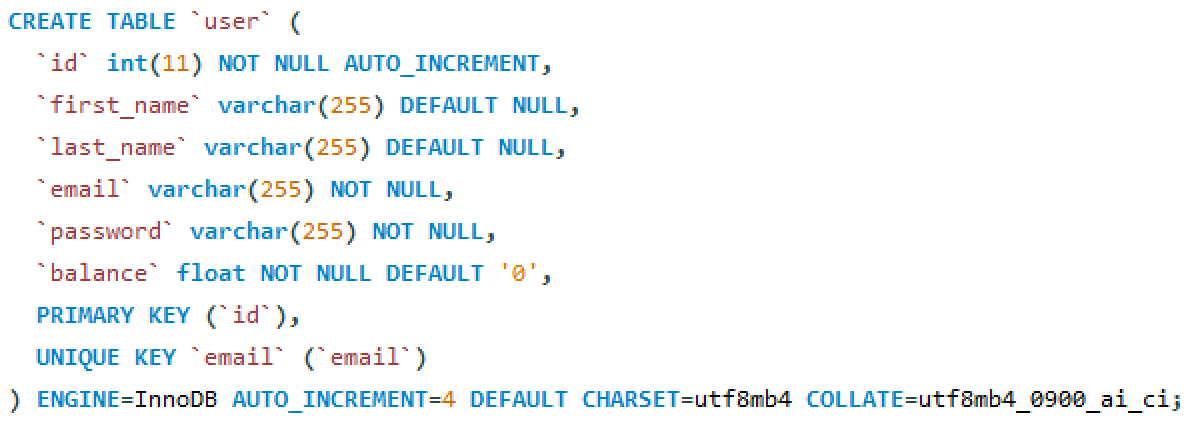


Рисунок 3.2 – Таблица User

Таблица включает поля:

* id – первичный ключ, идентификатор пользователя;
* First Name – имя пользователя;
* Last Name – фамилия пользователя;
* Email – почта, уникальное имя пользователя;
* Password – пароль пользователя;
* Balance – баланс пользователя.

На рисунке 3.3 представлена структура таблицы User\_Setting, которая содержит информацию о настройках пользователей.

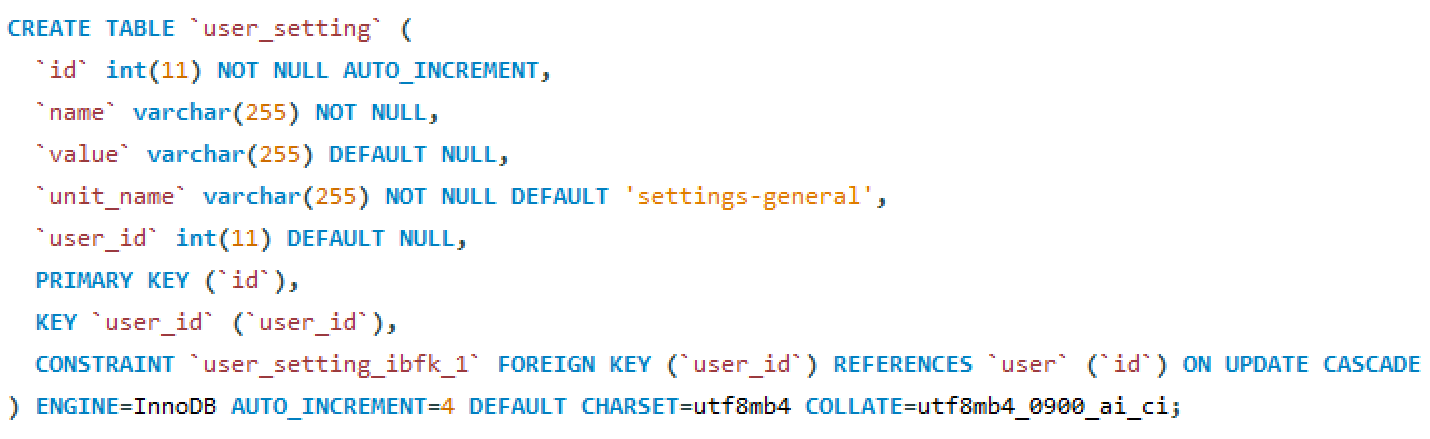


Рисунок 3.3 – Таблица User\_Setting

Таблица включает поля:

* Id – первичный ключ, идентификатор настройки;
* Name – название настройки;
* Value – текущее значение настройки;
* Unit Name – название категории настройки;
* User Id – идентификатор пользователя с этой настройкой;

На рисунке 3.4 представлена структура таблицы Notification, которая содержит оповещения пользователей.

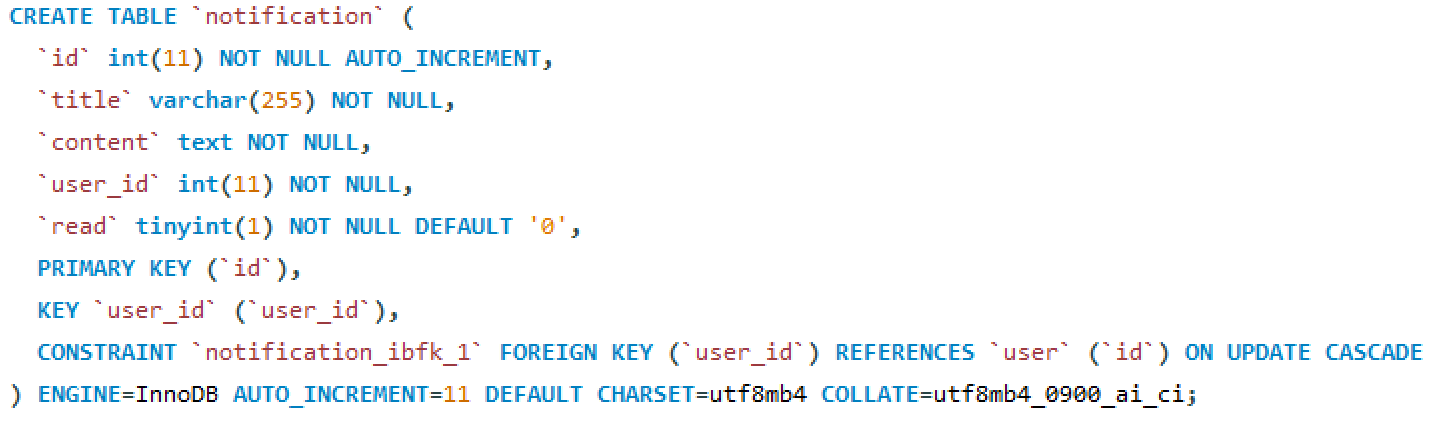


Рисунок 3.4 – Таблица Notification

Таблица включает поля:

* Id – первичный ключ, идентификатор оповещений;
* Title – заголовок оповещения;
* Content – тело оповещения;
* Read – показывает было ли сообщение прочитано пользователем;
* User Id – идентификатор пользователя.

На рисунке 3.5 представлена структура таблицы Transaction\_Type, которая содержит информацию о возможных типах транзакций.

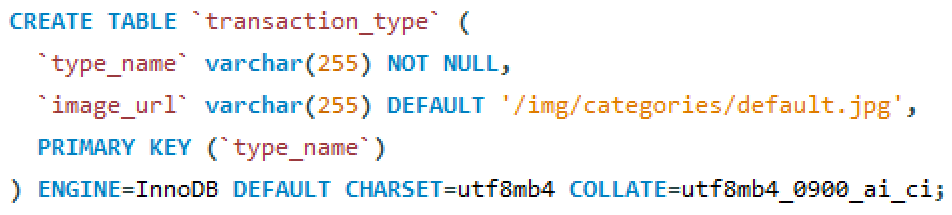


Рисунок 3.5 – Таблица Transaction\_Type

Таблица включает поля:

* Type Name – название типа транзакции, первичный ключ;
* Image Url – ссылка на иконку для типа транзакции.

На рисунке 3.6 представлена структура таблицы Category, которая содержит информацию о возможных категориях транзакций.

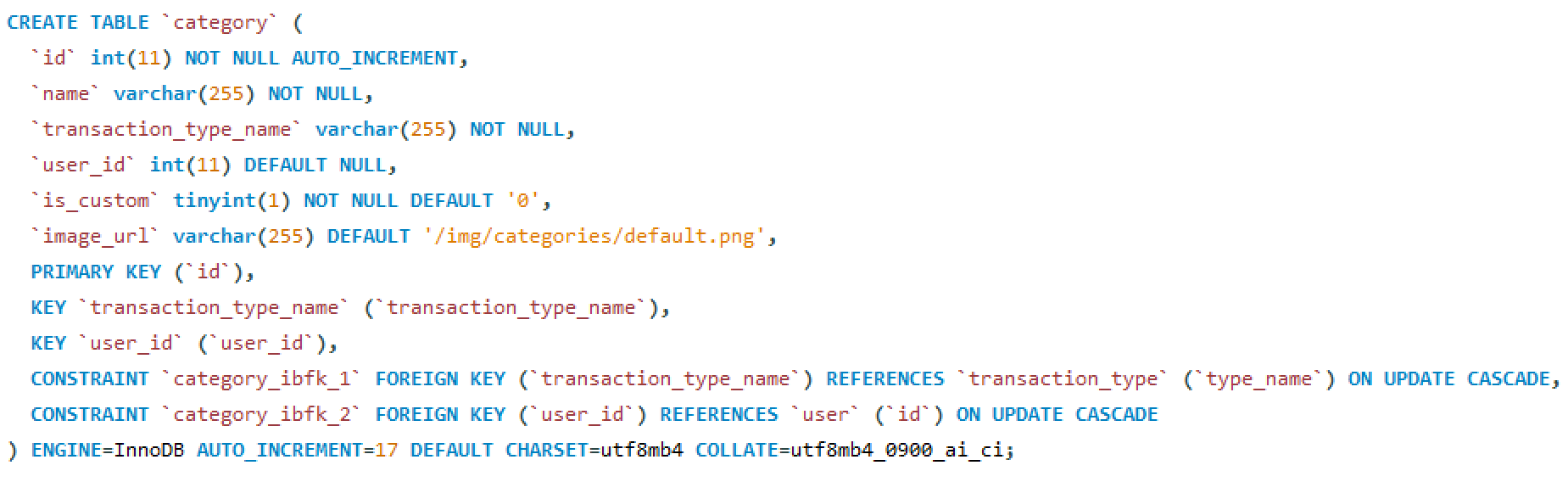


Рисунок 3.6 – Таблица Category

Таблица включает поля:

* Id – идентификатор категории, первичный ключ;
* Name – имя категории;
* Transaction Type Name – тип транзакции, к которому относится категория;
* User Id – идентификатор пользователя к которому относится категория;
* Is Custom – определяет, является ли категория пользовательской или стандартной;
* Image Url – ссылка на иконку для категории.

На рисунке 3.7 представлена структура таблицы Transaction, которая содержит транзакции (платежи или доходы) пользователей.

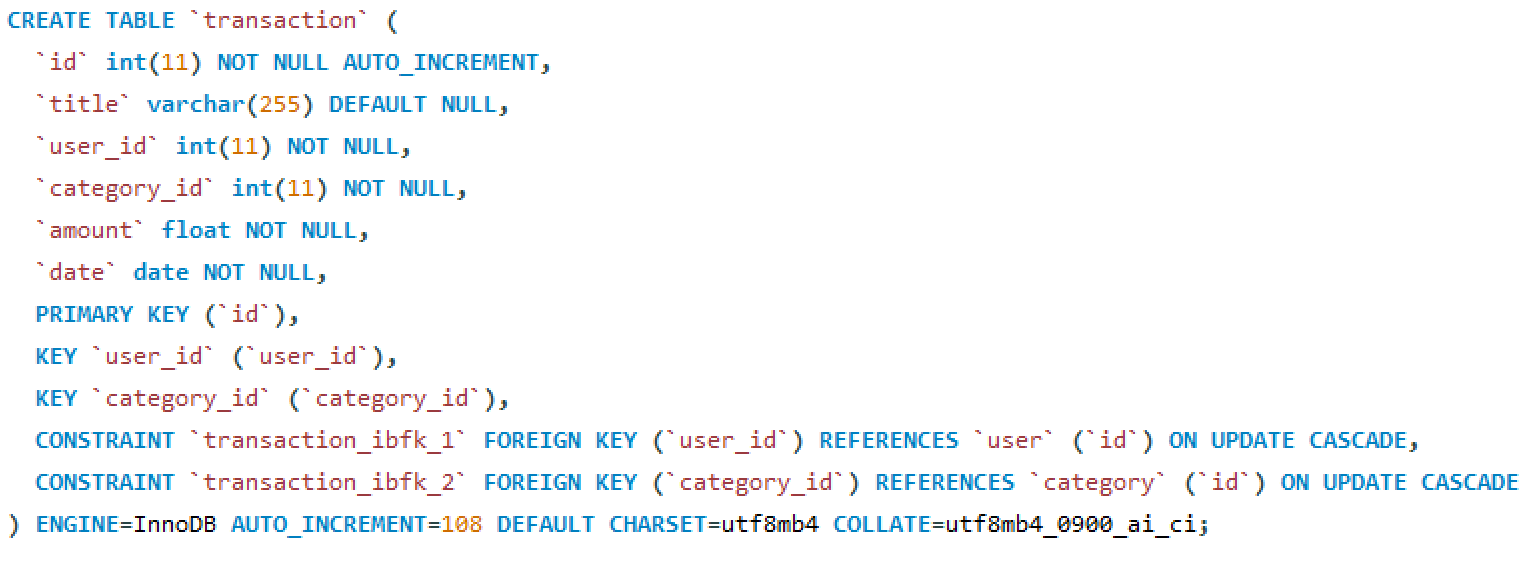


Рисунок 3.7 – Таблица Transaction

Таблица включает поля:

* Id – идентификатор транзакции, первичный ключ;
* Title – имя категории;
* User Id – идентификатор пользователя;
* Category Id – идентификатор категории, к которой относится транзакция;
* Amount – сумма денег за транзакцию;
* Date – дата, когда была произведена транзакция.

На рисунке 3.8 представлена структура таблицы Regular\_Transaction, которая содержит регулярные (повторяющиеся) транзакции (платежи или доходы) пользователей.



Рисунок 3.8 – Таблица Regular\_Transaction

Таблица включает поля:

– Id – идентификатор транзакции, первичный ключ;

– Title – имя категории;

– User Id – идентификатор пользователя;

– Category Id – идентификатор категории, к которой относится транзакция;

– Amount – сумма денег за транзакцию;

– Next Transaction Date – следующая дата, когда будет произведена регулярная транзакция;

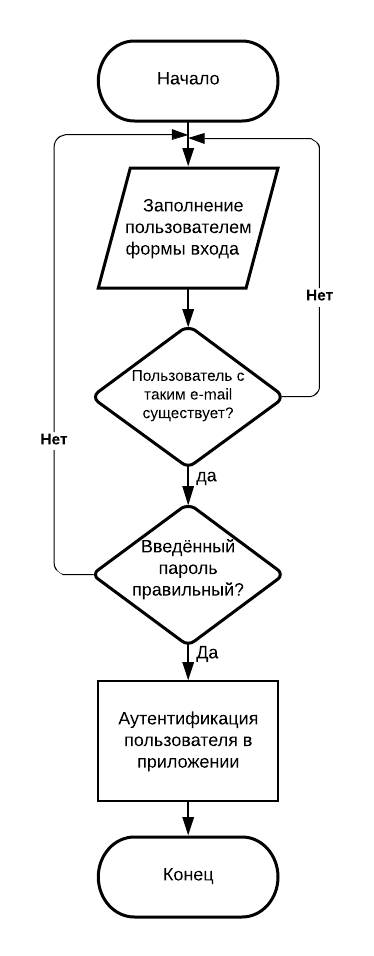
– Interval – интервал повторения транзакций;

– Date – дата, когда была произведена транзакция.

# 3.3 Описание алгоритма аутентификации пользователя

Аутентификация пользователя происходит на странице Login, листинг кода приведен в приложение В. Также предусмотрена валидация данных.

Общая схема процесса аутентификации пользователя представлена на блок-схеме 1.1.



Блок-схема 1.1 ­­– Схема аутентификации

Пароли хранятся в хэшированном виде с помощью алгоритма MD5.

Регистрация пользователя происходит в разделе Registration, листинг которого находится в приложении Г. При регистрации предусмотрена валидация данных. Логин и пароль пользователя уникальны. Каждый пользователь сначала должен зарегистрироваться, после чего он сможет авторизоваться.

# 3.4 Процесс отправки сообщений

TODO: Здесь нужно выбрать какую нить функцию основную и описать ее, например транзакции твои, привести листинг кода и кратенько описать код

В 3.4 и 3.5 можешь добавить код и описать именно его, т.е. теоритически уже описаны

Пример :

Для отправки сообщений между сотрудниками использовалась такая технология, как SignalR Core [5]. Отправка сообщений происходит по протоколу WebSocket.

SignalR автоматически выбирает лучший транспортный метод, который находится в пределах возможностей сервера и клиента.

Пример основного класса отвечающий за отправку сообщений приведем в листинг 3.1.

public class ChatHub : Hub<IChatClient>  
{  
private readonly IDialogRepository DialogRepository;  
private readonly IMessageRepository MessageRepository;  
private string UserId => Context.GetHttpContext().Request.Cookies["userId"];  
public ChatHub(IDialogRepository dialogRepository, IMessageRepository messageRepository)  
{  
DialogRepository = dialogRepository;  
MessageRepository = messageRepository;  
}  
public async Task SendMessage(NewMessageSentWrapper message)  
{  
var userDialogs = await DialogRepository.GetUserDialogsByDialogId(message.DialogId);  
if (userDialogs != null && !string.IsNullOrEmpty(UserId))  
{  
var addresserUserDialog = userDialogs.First(d => [d.User.Id](https://vk.com/away.php?to=http%3A%2F%2Fd.User.Id&cc_key=) == Convert.ToInt32(UserId));  
await MessageRepository.CreateMessage(new Message  
{  
Text = message.Text,  
TimeSend = [DateTime.Now](https://vk.com/away.php?to=http%3A%2F%2FDateTime.Now&cc_key=),  
User = addresserUserDialog.User,  
Dialog = addresserUserDialog.Dialog  
});  
await Clients.Users([userDialogs.Select](https://vk.com/away.php?to=http%3A%2F%2FuserDialogs.Select&cc_key=)(d => d.User.Id.ToString()).ToArray()).ReceiveMessage(message);}  
 }  
}

Листинг 3.1 – Класс ChatHub

В данном классе функция SendMessage является основной. Здесь мы получаем айди диалога, который выбрал пользователь, затем проверяем его на наличие, если userDialogs не пустой и не null, мы получаем userDialog отправителя, далее мы создаем новое сообщение, которое ввел пользователь и записываем его в базу данных. После чего мы отправляем сообщение через протокол WebSocket всем клиентам, которые относятся к диалогу.

# 3.4 Идентификация, авторизация и аутентификация

TODO: Добавить листинг и описание

Контроллер предназначен для обработки запросов связанных с идентификацией, авторизацией и аутентификацией пользователей, их регистрацией и входом в систему.

Для обеспечения безопасности пользователей был выбран путь реализации протокола OAuth2.0. Его идея заключается в том, что каждому пользователю будет соответствовать 2 токена: один из них это токен доступа (access token), другой – токен обновления (refresh token). Токен доступа предназначен для того, чтобы пользователь мог идентифицировать то, что к серверу обращается конкретно он. Этот токен хранится на стороне пользователя в local storage. Токен обновления служит для того, чтобы обновлять токен авторизации, когда время его жизни подходит к концу. Токен обновления живёт намного больше, нежели токен доступа.

# 3.5 Запросы и мутации моделей

TODO: Добавить листинг и описание

Эти контроллеры предназначены для работы с моделями данных, их получением, обновлением и удалением. Как альтернатива REST архитектуры использовалась архитектура с применением GraphQL. Она позволяет быстро построить веб-интерфейс и спецификацию к нему для более легкой поддержки и расширения в будущем.

В проекте реализованы методы для получения и мутации данных таких моделей, как:

* Пользователи;
* Настройки пользователей;
* Категории;
* Транзакции;
* Регулярные транзакции.

Чтобы посмотреть полную GraphQL схему можно перейти в приложении Г, там все подробно продемонстрировано.

# Руководство программиста

В данной главе будут представлены шаги, описывающие как развернуть приложение. Разворачивать приложение будем при помощи Docker.

Docker – платформа для разработки, доставки и эксплуатации приложений.

Основное назначение – упростить развертывание приложения.

Нам понадобится Docker последней версии. В Docker Hub заранее были загружены images, поэтому понадобится лишь загрузить их при помощи команд представленных в листинге 5.1:

docker pull millesya200/schedule-server:latest

docker pull millesya200/schedule-client:latest

Листинг 4.1 – Команды Docker

Далее нужно обратиться к файлу docker-compose.yml (представлен в приложении К) в нём определены 3 сервиса два из которых представляют выгруженные клиент и сервер сервисы, а третий представляет базу данных. Запуск производится с помощью команды представленной на рисунке 5.1

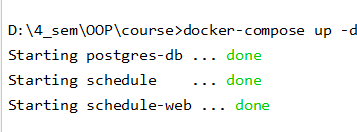


Рисунок 5.1 – Результат выполнения сборки и запуска при помощи compose

После этого сервизы будут запущены и можно будет работать с приложением предварительно заполнив базу данных пользовательскими данными.

Docker [6] — это открытая платформа для разработки, доставки и эксплуатации приложений. Docker разработан для более быстрого выкладывания ваших приложений. С помощью docker вы можете отделить ваше приложение от вашей инфраструктуры и обращаться с инфраструктурой как управляемым приложением. Docker помогает выкладывать ваш код быстрее, быстрее тестировать, быстрее выкладывать приложения и уменьшить время между написанием кода и запуска кода. Docker делает это с помощью легковесной платформы контейнерной виртуализации, используя процессы и утилиты, которые помогают управлять и выкладывать ваши приложения.

В своем ядре docker позволяет запускать практически любое приложение, безопасно изолированное в контейнере. Безопасная изоляция позволяет вам запускать на одном хосте много контейнеров одновременно. Легковесная природа контейнера, который запускается без дополнительной нагрузки гипервизора, позволяет вам добиваться больше от вашего железа.

Платформа и средства контейнерной виртуализации могут быть полезны в следующих случаях:

* 1. упаковывание вашего приложения (и так же используемых компонент) в docker контейнеры;
  2. раздача и доставка этих контейнеров вашим командам для разработки и тестирования;
  3. выкладывания этих контейнеров на ваши продакшены, как в дата центры так и в облака.

Для работы с докером его нужно было установить на свой компьютер. Чтобы установить докер я следовала инструкциям по установке на официальном сайте.

В проекте создала и описала процесс сборки контейнера докеров в файле DockerFile.

Чтобы упаковать свое приложение в докер, использовала команды в командной строке, которые приведены в листинге 1.2.

Листинг 1.2. – Упаковка приложения в Docker контейнер

//Для сборки проекта

docker build . -t docker-homework-courses

docker build . -t docker-homework-courses —no-cache

docker build -t docker-homework-courses .

//Просмотр всех имэджей

docker images

//Запуск контейнера

docker run -p 8080:8080 -it —rm docker-homework-courses

//Просмотр запущенных контейненров

docker ps

//Остановить контейнер

docker stop <CONTAINER\_ID>

# 4.1 Руководство пользователя

Данная глава содержит описание некоторых функций приложения для более легкого восприятия конечного пользователя.

TODO: 4.1.1 должна быть регистрация пользователей, 4.1.2 вход в систему, 4.1.3 уже главная страница, 4.1.4 добавление ед. Транзакции и т.д

# 4.1.1 Главная страница

На главной странице продемонстрирован персональный календарь. В окошках на каждый день отображается сумма доходов за этот день (зеленый цвет), расходов за день (красный), запланированный доход (синий), запланированный расход (желтый). Все это продемонстрировано на рисунке 4.1.

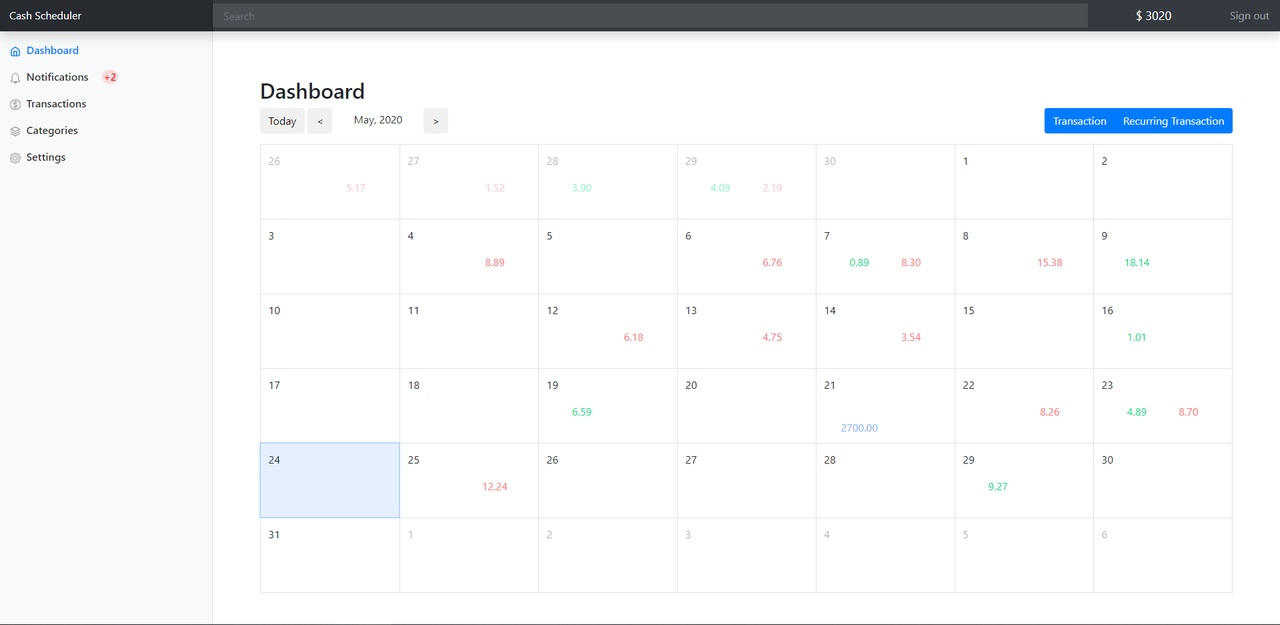


Рисунок 4.1 – Главная страница приложения

# 4.1.2 Добавление единичной транзакции

Для того чтобы добавить единичную транзакцию пользователю необходимо на главной страницу нажать на кнопку «Transaction». После нажатия на эту кнопку пользователь увидит форму, которая показана на рисунке 4.2.

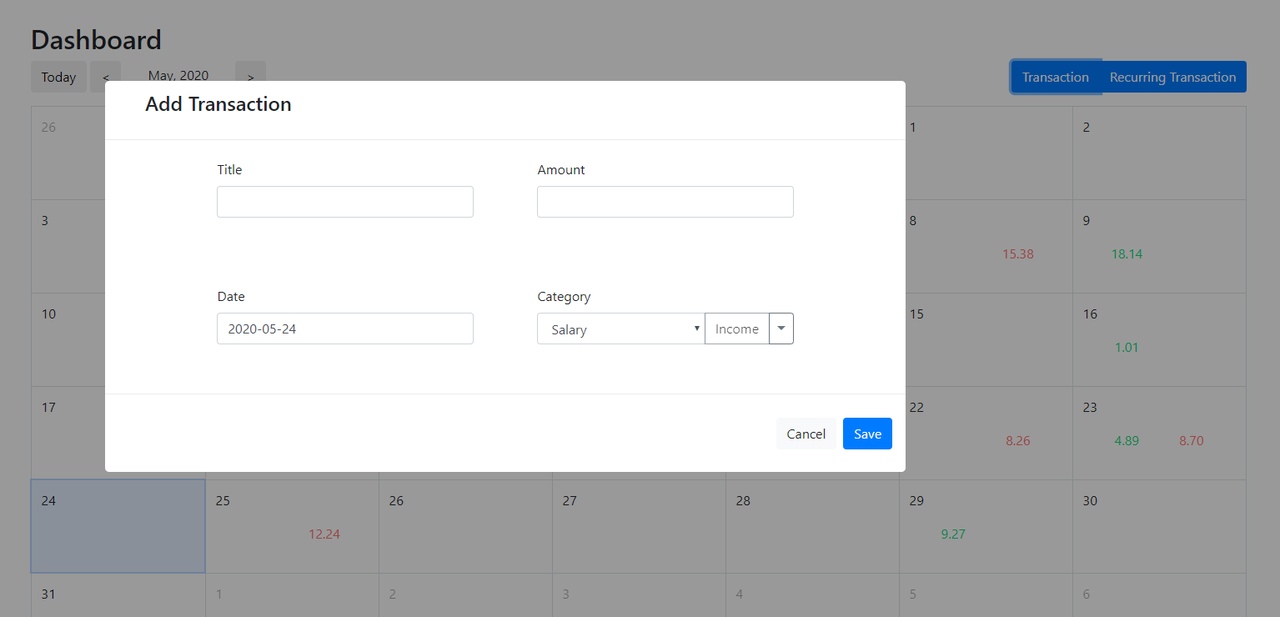


Рисунок 4.2 – Форма для добавления транзакции

Здесь можно выбрать вид транзакции расход это, либо доход, в зависимости от этого будет выбор категории. Все возможные категории продемонстрированы на рисунке 4.3.

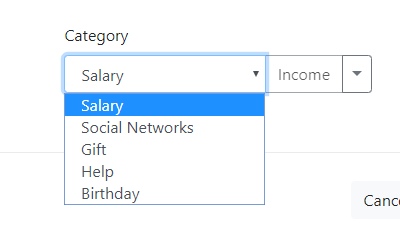


Рисунок 4.3 – Категории

Так же в форме существует возможность добавления повторяющейся транзакции (рисунок 4.4), которая будет автоматически повторяться через указанный пользователем интервал (рисунок 4.5).

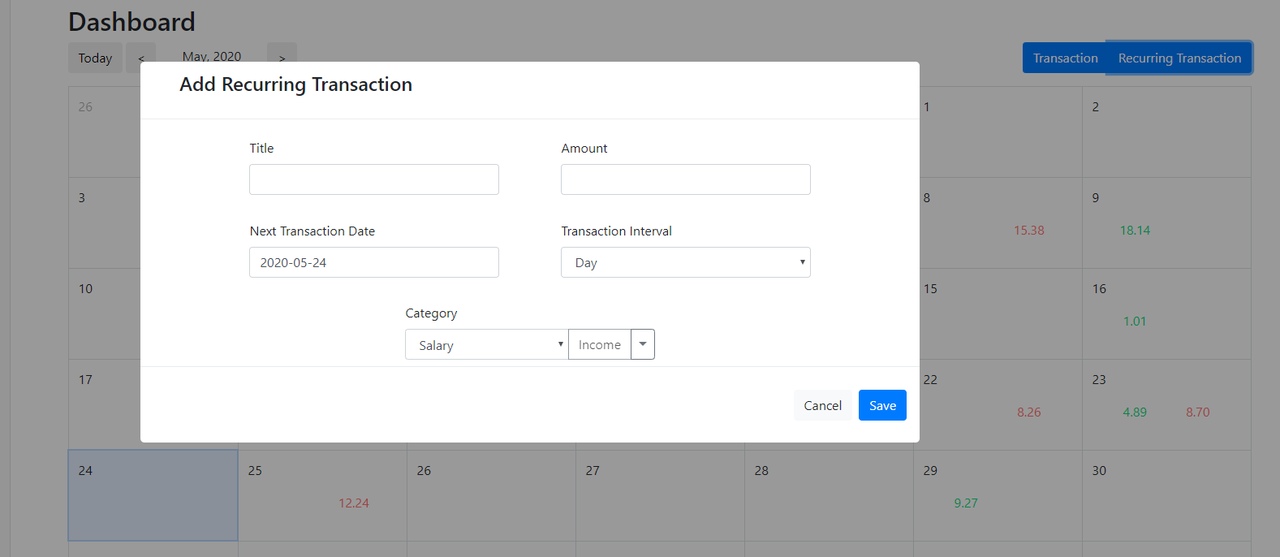


Рисунок 4.4 – Добавление повторяющейся транзакции

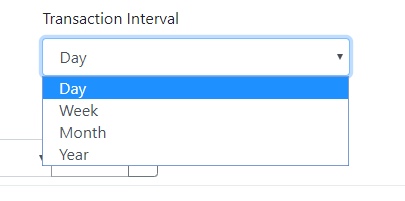


Рисунок 4.5 – Интервал

TODO: Добавить пару предложений, нельзя чтобы раздел заканчивался на просто картинке

# 4.1.3 Страница оповещений

В данном разделе продемонстрированы оповещения. Чтобы перейти на эту страницу просто нужно нажать в меню на раздел «Notifications». На эту страницу приходят оповещения, связанные с тем, как пользователь распоряжается своим балансом. Пользователь сам отправлять ничего не может, он только получает оповещения (рисунок 4.6).

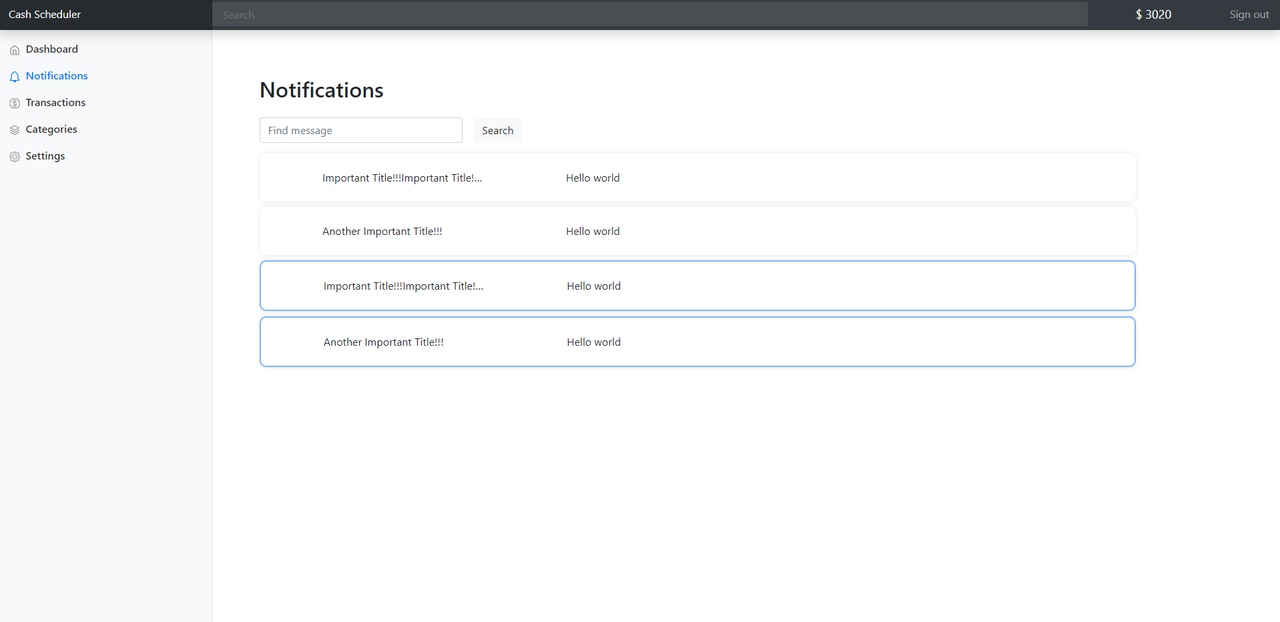


Рисунок 4.6 – Страница оповещений

На рисунке 4.7 показано как выглядит открытое сообщение, если сообщение еще является непрочитанным, то оно будет.

Предусмотрен и поиск по оповещениям, который так же продемонстрирован на рисунке 4.7. Если оповещений будет очень много, то будет произведено разделение страницы (пагинация).

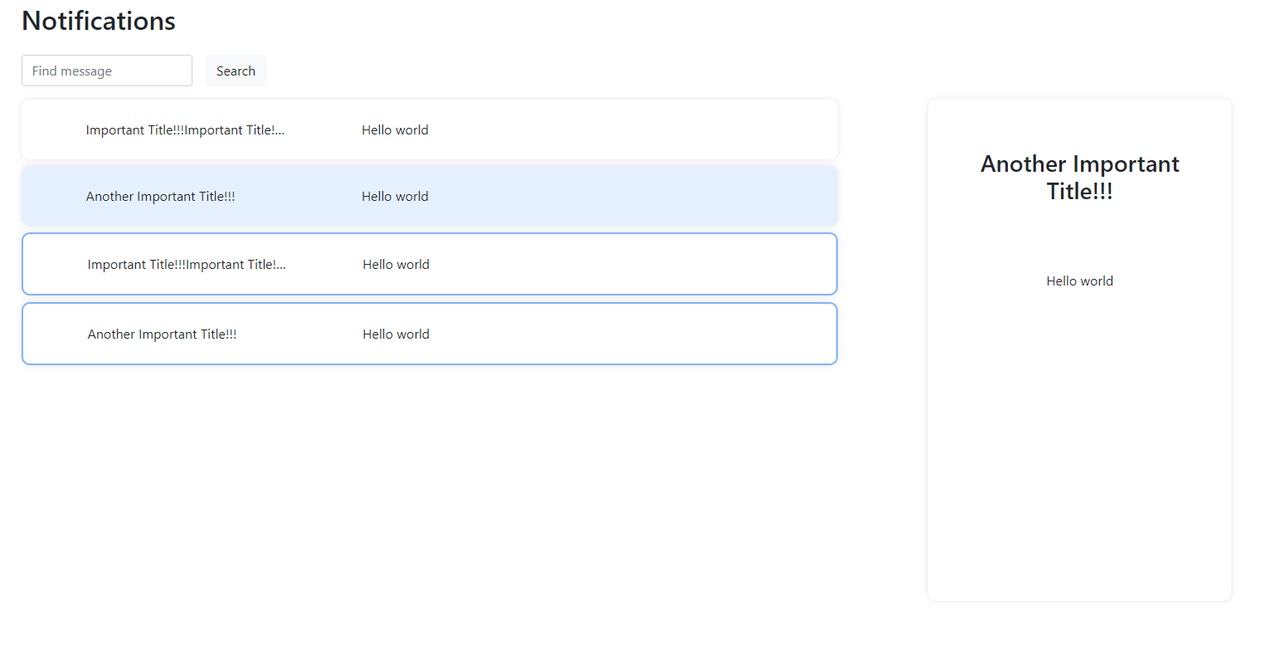


Рисунок 4.7 – Открытое оповещение

TODO: Добавить пару предложений, нельзя чтобы раздел заканчивался на просто картинке

# 4.1.4 Страница транзакций

Чтобы перейти на данную страницу просто нужно нажать в меню на раздел «Transactions». На этой странице пользователь может просматривать отчеты трат, доходов и их соотношения по месяцам (рисунок 4.8).

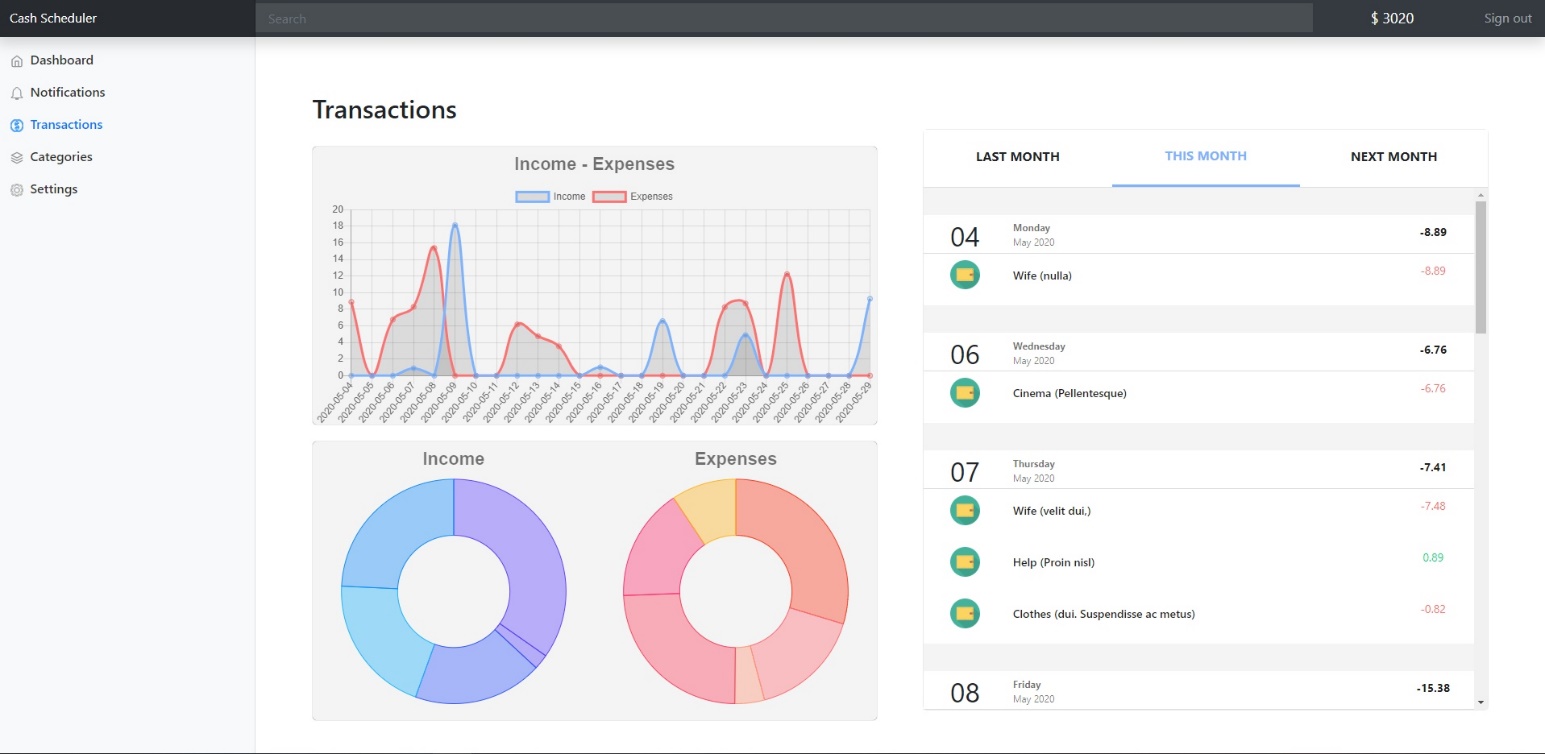


Рисунок 4.8 – Страница транзакций

Можно посмотреть, как много было потрачено или заработано на определенную категорию за выбранный месяц. Это можно увидеть на графике внизу страницы, показано на рисунке 4.9.

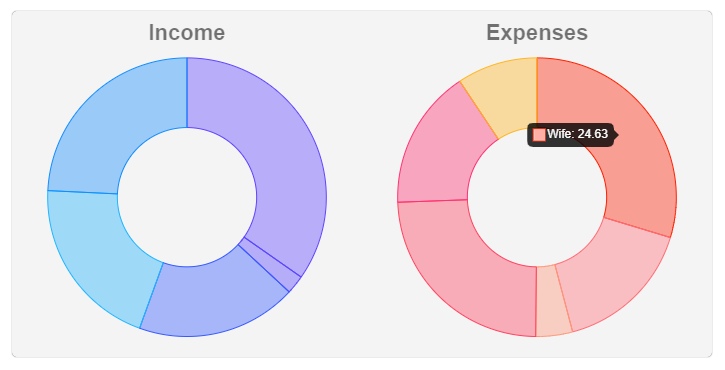


Рисунок 4.9 – Соотношение расходов и доходов по категориям

Следующий же график показывает траты и доходы за каждый день выбранного месяца. Он находиться вверху страницы транзакций, пример его продемонстрирован на рисунке 4.10.

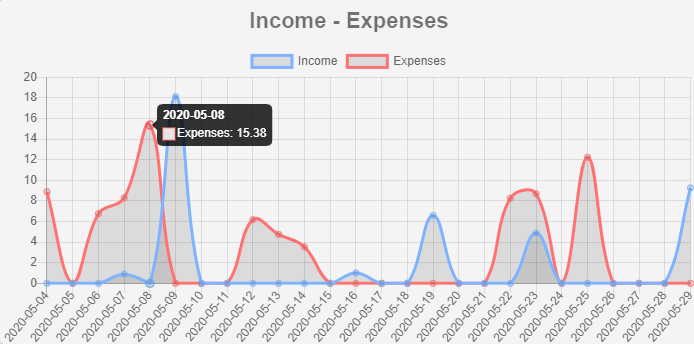


Рисунок 4.10 – Количество расходов и доходов по дням

На странице транзакций справа можно увидеть список транзакций за выбранный месяц сгруппированный по дням месяца. Реализована возможность для пользователя выбрать какую-либо транзакцию из списка и отредактировать либо ее название, либо количество денег, больше изменить ничего нельзя, но еще предусмотрено удаление (рисунок 4.11).

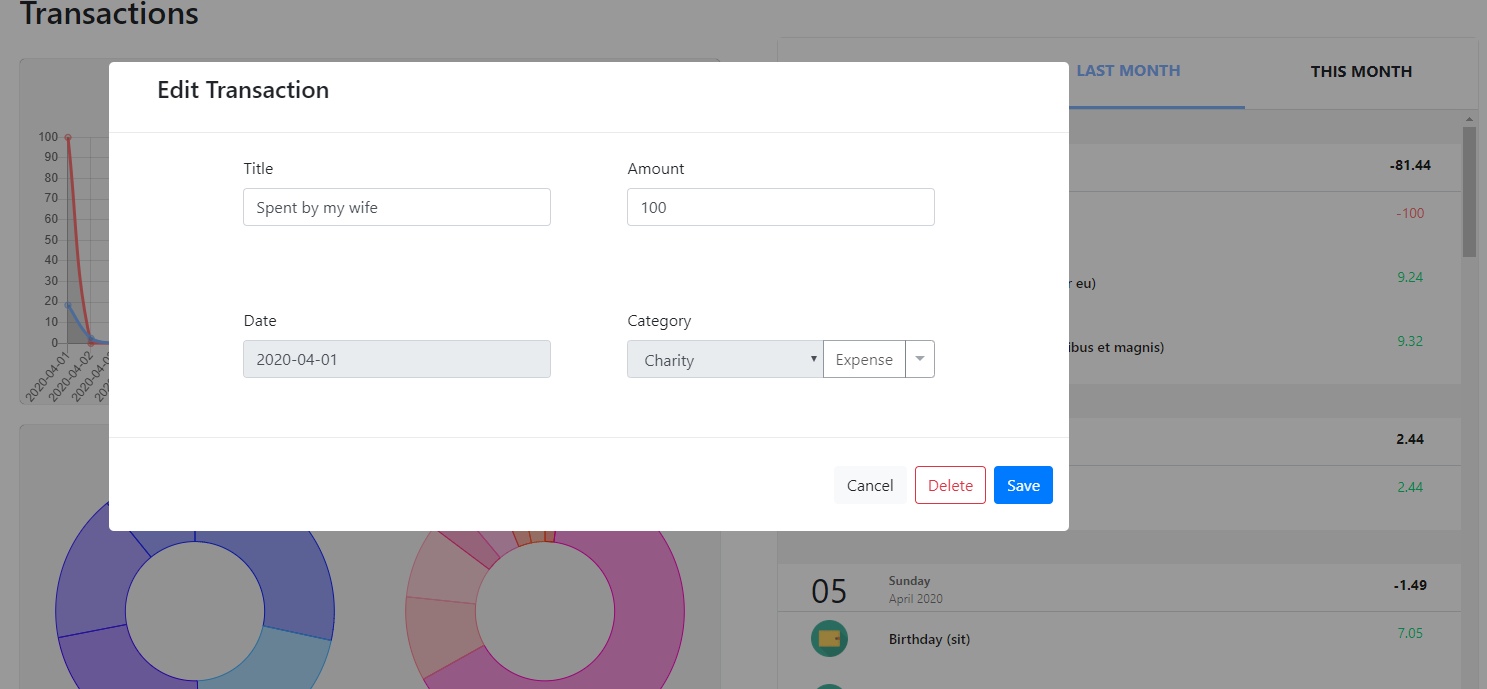


Рисунок 4.11 – Редактирование транзакций

TODO: Добавить пару предложений, нельзя чтобы раздел заканчивался на просто картинке

# 4.1.5 Страница категорий

Чтобы перейти на данную страницу просто нужно нажать в меню на раздел «Categories». Здесь можно увидеть, как стандартные, так и пользовательские категории. К стандартным относятся: salary, social networks, gift, help. Более сложными по-своему функционалу являются пользовательские категории, которые показаны на рисунке 4.12.

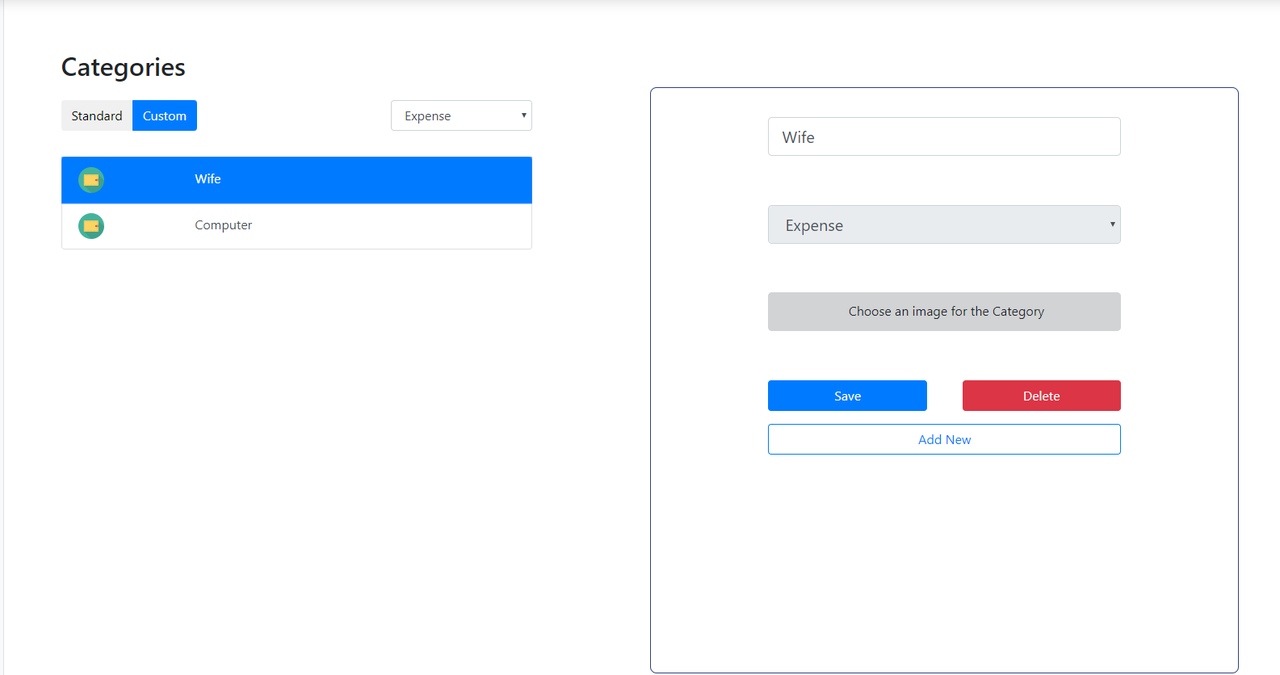


Рисунок 4.12 – Пользовательские категории

Пользовательские категории можно редактировать, добавлять свои новые и удалять, тем самым стандартные категории изменить невозможно. Пример добавления новой пользовательской категории продемонстрирован на рисунке 4.13.

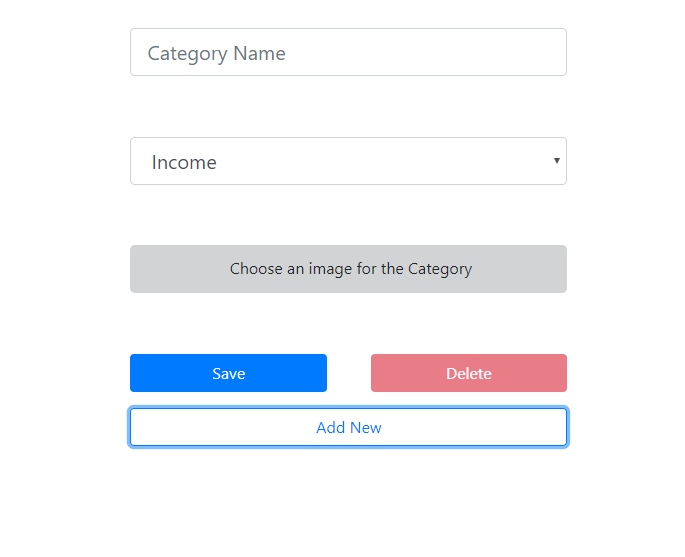


Рисунок 4.13 – Добавление пользовательской категории

TODO: Добавить пару предложений, нельзя чтобы раздел заканчивался на просто картинке

# 4.1.6 Страница настроек

Чтобы перейти на данную страницу просто нужно нажать в меню на раздел «Settings». На этой странице реализовано три раздела: general, notifications, for developers. В первом разделе предусмотрено включение и выключение отображения баланса, если включить эту функцию, то баланс пользователя будет отображаться сверху слева (рисунок 4.14).

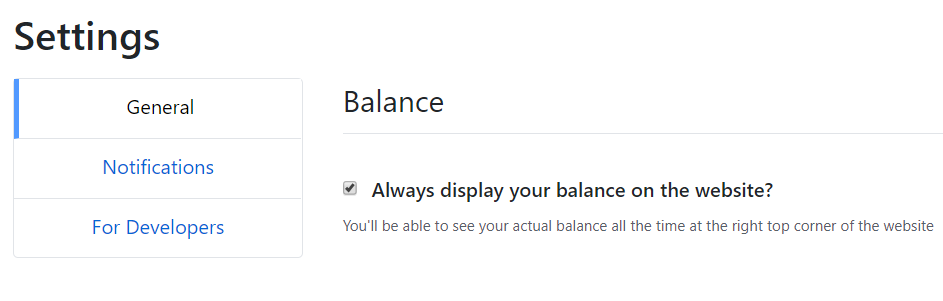


Рисунок 4.14 – Добавление пользовательской

Все настройки хранятся в базе данных, поэтому если обновить страницу, изменения сохранятся.

Во втором разделе настроек есть такие настройки как включение уведомлений, возможность получать копию уведомлений на почту, включение и выключение звука в браузере при оповещении (рисунок 4.15).

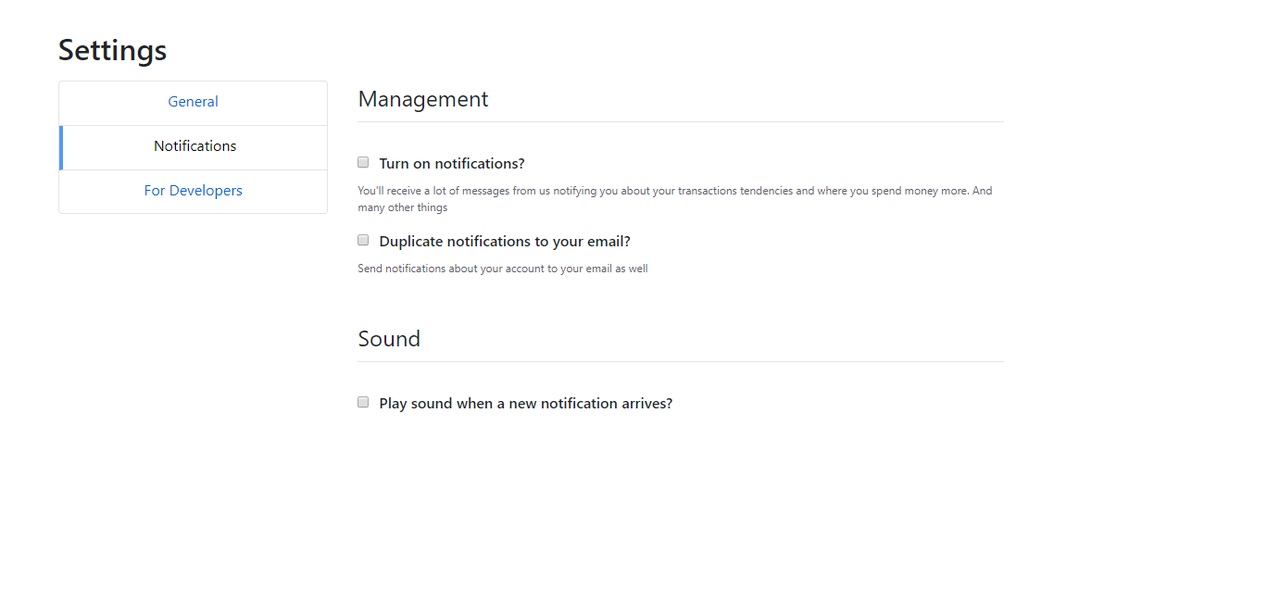


Рисунок 4.15 – Раздел настроек «Notifications»

Третий раздел настроек разработан специально для разработчиков. В нем имеется кнопка, которая копирует в буфер обмена токен доступа к вашему аккаунту, чтобы вы могли работать со своим IP через другие приложения. Пример этого показан на рисунке 4.16.

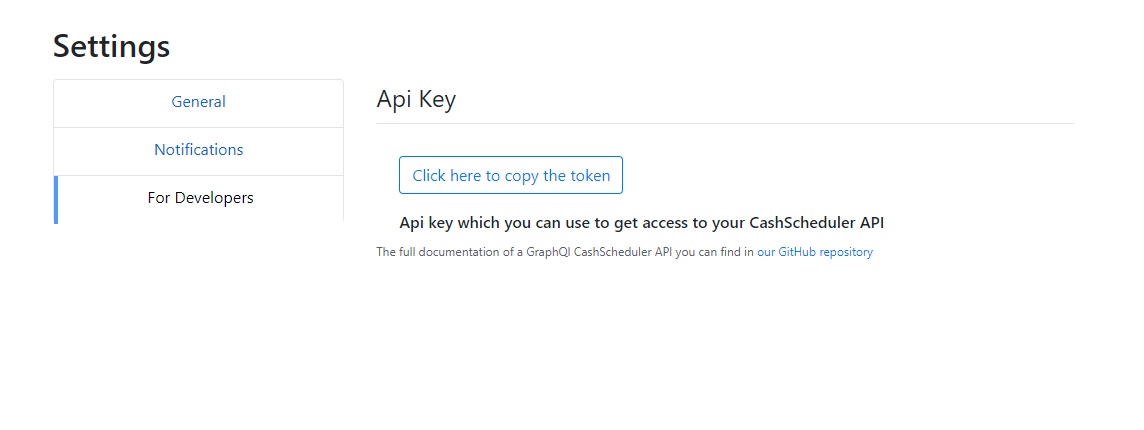


Рисунок 4.16 – Раздел настроек «For developers»

TODO: Добавить пару предложений, нельзя чтобы раздел заканчивался на просто картинке

**На странице должно быть 80% заполненности...Картинки адекватных размеров**

# Тестирование

До запуска приложения в производство, когда оно станет доступно пользователям, важно убедиться, что данное приложение функционирует, как и должно, что в нем нет ошибок, которые могли бы полностью заблокировать основные сценарии взаимодействия с приложением.

Для обеспечения корректности работы программы, обрабатываются различные ошибки, возникающие в процессе работы. Данное программное средство использует подключение к базе данных, следовательно, неправильно введенные данные или же их отсутствие может повлечь за собой неработоспособность приложения.

В курсовом проекте организована валидация полей на случай неочевидного или неправильного поведения пользователя.

Валидация приложения – это один из основных гарантов надёжности приложения и всякий программист должен предусмотреть и предотвратить непредвиденное поведение пользователя.

При регистрации обрабатываются вводимые данные пользователя. Так как поле «email» является уникальным, в базе данных не могут храниться пользователи с одинаковыми логинами, если в базе данных уже существует пользователь с таким же логином, то клиента уведомят об ошибке, это продемонстрировано на рисунке 5.1.

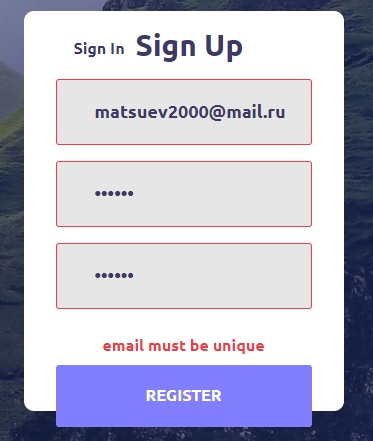


Рисунок 5.1 – Некорректный логин

Также возникают ошибки, если пользователь пытается ввести неверные данные в поле «Confirm password» (рисунок 5.2). Введенный пароль в поле «Password» должен полностью совпадать с введенным паролем в поле «Confirm password».

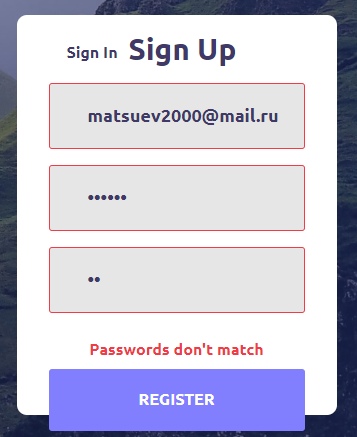


Рисунок 5.2 – Несовпадение паролей

Существует проверка на пустые поля. Если пользователь не ввел данные в поля, система так же уведомит его об этом. Пример не введённых данных в поля «Password» и «Confirm Password» продемонстрирован на рисунке 5.3.

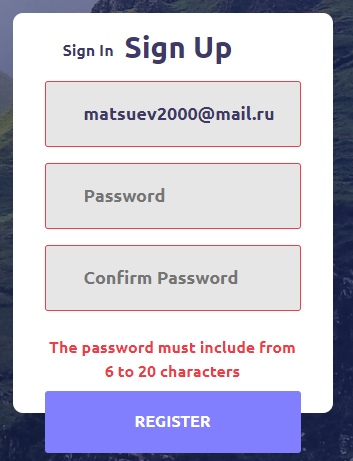


Рисунок 5.3 – Пустые поля

При входе в систему также обрабатываются вводимые данные пользователя. Если пользователь неверно ввел данные в поля для входа, его уведомят об этом, пример уведомления об ошибке предоставлен на рисунке 5.4.

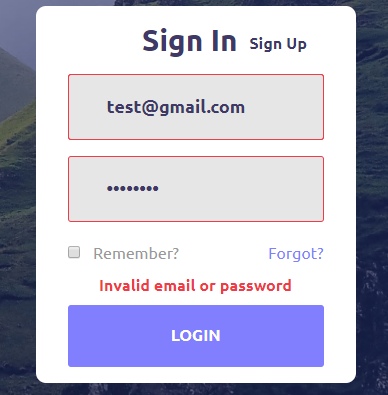
****

Рисунок 5.4 – Неверно введены данные

В данной курсовой работе разработана функция для тех пользователей которые забыли пароль. В этом разделе так же предусмотрена проверка на вводимые данные. Поле для логина является уникальным поэтому создана проверка на его существование. Соответственно если введенного логина нет в базе данных восстановить данные не удастся, пример работы предоставлен на рисунке 5.5.

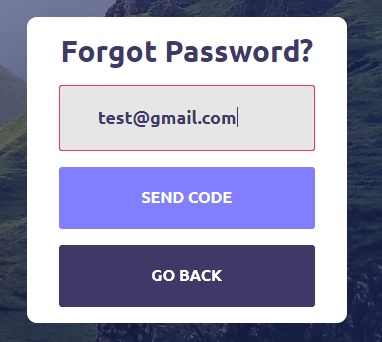


Рисунок 5.5 – Наличие пользователя в базе данных

Следующим этапом проверки является проверка на правильно введенный код подтверждения для восстановления пароля (рисунок 5.6).

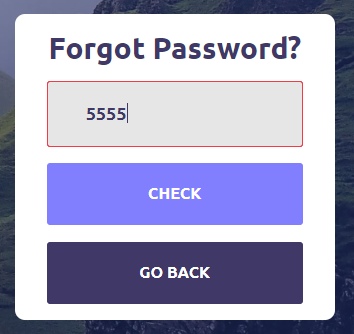


Рисунок 5.6 – Неверный код подтверждения

# Заключение

В результате курсового проектирования было разработано приложение, полностью удовлетворяющее ранее поставленным задачам. Таким образом, была создана база данных и приложение «Планировщик бюджета». Были реализованы сервисы регистрации и входа в учётную запись. Авторизированный пользователь имеет доступ ко всему функционалу приложения, то есть создание, редактирование и удаление расходов и доходов, а так же категорий, и просмотр отчётов по расходам и доходам.

При реализации курсового проекта использовались технологии Node.js, фреймворк Express, ORM библиотека sequelize, WebSoket для реализации back-end, для front-end части Bootstrap, JQuery, JavaScript, GraphQL, CSS. MY SQL в качестве базы данных.

Для удалённого доступа пользователей приложение было опубликовано на платформу Heroku. Так же был создан удаленный репозиторий для удаленного доступа к базе данных на основе сервиса RemoteMySQL. Ссылка перехода: <https://cash-scheduler.herokuapp.com/>.

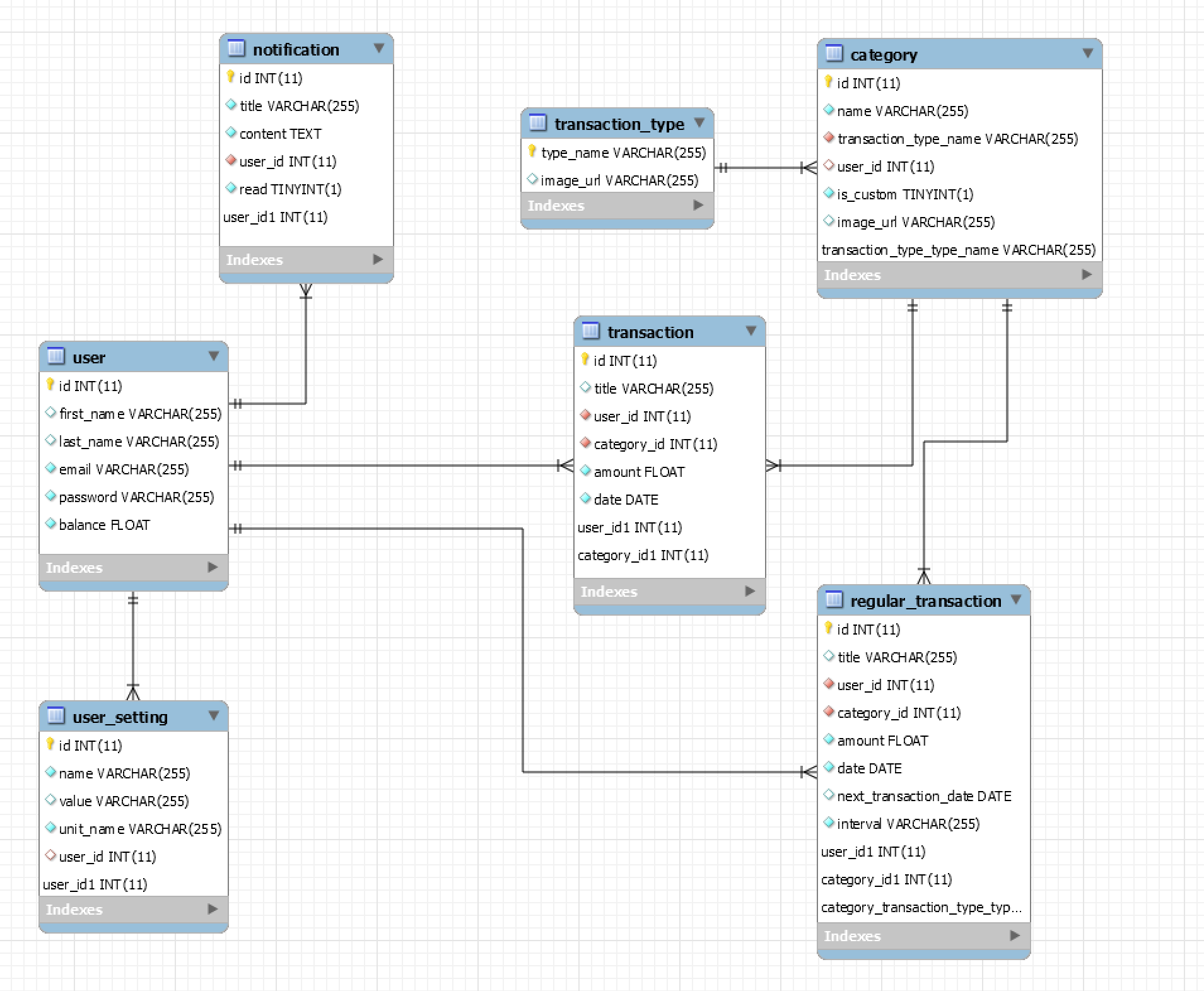
В соответствии с полученным результатом работы программы, можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

# Список литературы

1. Руководство по ASP.NET Core 5 [Электронный ресурс]. — Режим доступа:  https://metanit.com/sharp/aspnet5/ Дата доступа: 13.11.2020.
2. React Java Script library for building user interfaces [[Электронный](file:///C:\Users\pavel.dzemasiuk\Desktop\Электронный)ресурс]. – Режим доступа: [https://reactjs.org/. – Дата](https://reactjs.org/.%20–%20Дата) доступа: 27.11.2020.
3. Design Patterns - MVC Pattern [Электронный ресурс]. — Режим доступа:  <https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm> Дата доступа: 27.11.2020.
4. Введение в MS SQL Server и T-SQL [Электронный ресурс]. — Режим доступа https://metanit.com/sql/sqlserver/1.1.php Дата доступа: 30.11.2020.
5. Get started with ASP.NET Core SignalR [Электронный ресурс]. — Режим доступа: https://docs.microsoft.com/en-us/aspnet/core/tutorials/signalr?view=aspnetcore-5.0&tabs=visual-studio Дата доступа: 03.12.2020.
6. Docker [Электронный ресурс]. — Режим доступа: https://docs.docker.com/ Дата доступа: 05.12.2020.

# ПРИЛОЖЕНИЕ А

**Логическая схема базы данных**



# ПРИЛОЖЕНИЕ Б

**Файл используемых модулей package.json**

{  
 "name": "social-network-react",  
 "version": "0.1.0",  
 "private": *true*,  
 "dependencies": {  
 "@microsoft/signalr": "^5.0.0",  
 "@testing-library/jest-dom": "^4.2.4",  
 "@testing-library/react": "^9.5.0",  
 "@testing-library/user-event": "^7.2.1",  
 "@types/jest": "^26.0.15",  
 "@types/node": "^14.14.6",  
 "@types/react": "^16.9.56",  
 "@types/react-dom": "^16.9.9",  
 "axios": "^0.19.2",  
 "classnames": "^2.2.6",  
 "js-cookie": "^2.2.1",  
 "react": "^16.13.1",  
 "react-dom": "^16.13.1",  
 "react-redux": "^7.2.1",  
 "react-router-dom": "^5.2.0",  
 "react-scripts": "3.4.1",  
 "react-semantic-toasts": "^0.6.5",  
 "redux": "^4.0.5",  
 "redux-form": "^8.3.6",  
 "redux-thunk": "^2.3.0",  
 "semantic-ui-css": "^2.4.1",  
 "semantic-ui-react": "^2.0.1",  
 "typescript": "^4.0.5"  
 },  
 "scripts": {  
 "start": "react-scripts start",  
 "build": "react-scripts build",  
 "test": "react-scripts test",  
 "eject": "react-scripts eject"  
 },  
 "eslintConfig": {  
 "extends": "react-app"  
 },  
 "browserslist": {  
 "production": [  
 ">0.2%",  
 "not dead",  
 "not op\_mini all"  
 ],  
 "development": [  
 "last 1 chrome version",  
 "last 1 firefox version",  
 "last 1 safari version"  
 ]  
 },  
 "devDependencies": {  
 "@types/classnames": "^2.2.11",  
 "@types/react-redux": "^7.1.11",  
 "@types/react-router-dom": "^5.1.6",  
 "@types/react-test-renderer": "^16.9.3",  
 "@types/redux-form": "^8.3.0",  
 "formik": "^2.2.2",  
 "react-test-renderer": "^17.0.1",  
 "reselect": "^4.0.0"  
 }  
}

# ПРИЛОЖЕНИЕ В

**Функция отвечающая за вход в приложение**

[HttpPost]

[Route("login")]

public async Task<ResponseWrapper<User>> Login(LoginRequestWrapper request)

{

ResponseWrapper<User> response;

var user = await Repository.GetUserByEmail(request.Email);

if (user != null && user.Password == Models.User.getHash(request.Password))

{

HttpContext.Response.Cookies.Append("userId", user.Id.ToString());

response = new ResponseWrapper<User>(user);

}

else

{

response = new ResponseWrapper<User>(null, new List<string>

{

"Such user doesn`t exist."

});

}

return response;

}

# ПРИЛОЖЕНИЕ Г

**Функция отвечающая за регистрацию пользователей**

[HttpPost]

[Route("register")]

public async Task<ResponseWrapper<User>> Register(RegisterRequestWrapper request)

{

ResponseWrapper<User> response;

var user = await Repository.GetUserByEmail(request.Email);

if (user == null)

{

if (request.Password != request.ConfirmPassword) {

return response = new ResponseWrapper<User>(null, new List<string> {

"Passwords do not match."

});

}

if (request.Password.Length < 6)

{

return response = new ResponseWrapper<User>(null, new List<string> {

"Your password is incorrect, it must be at least 6 characters long."

});

}

User newUser = new User();

newUser.Name = request.Name;

newUser.Email = request.Email;

newUser.Password = Models.User.getHash(request.Password);

newUser.TimeRegistration = DateTime.Now;

await Repository.CreateUser(newUser);

return new ResponseWrapper<User>(newUser);

}

else

{

response = new ResponseWrapper<User>(null, new List<string>

{

"A user with this email is already in the system."

});

}

return response;}