МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

ЗВІТ

З ПЕРЕДАТЕСТАЦІЙНОЇ ПРАКТИКИ СПЕЦІАЛІСТІВ

Місце проходження практики «НВП ''ХАРТРОН-АРКОС''»

у період з "06" лютого 2017 р. по "01" квітня 2017 р.

Тема індивідуального завдання:

Система для планування та управління бюджетом підприємства

|  |  |
| --- | --- |
| Студ. ІПЗс-16-2 Філіпов О.О. | Керівник практики Проф. Дудар З.В.  Робота захищена з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Дата «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2017 |

Комісія:

Проф. Дудар З.В.   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Доц. Кузнєцов Ю.О.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ст. викл. Трощило О.С.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Харків, 2017

РЕФЕРАТ/ABSTRACT

Пояснювальна записка: 25 стор., 6 рис., 13 дж.

Метою роботи є проектування системи корпораційного планування та управління бюджетом підприємства.

У результаті був проведений аналіз предметної області та спроектована система фінансового планування.

ФІНАНСИ, ФІНАНСОВА СИСТЕМА, БЮДЖЕТУВАННЯ, UML, EF, MVC, C#, КОНТРОЛЕР, СЕРВЕР, REST, MS SQL.

Explanatory note 25, p. 6 Fig. 14 joules.

The aim is to design a system collegiate planning and budget management company.

As a result of an analysis of the subject area and designed system of financial planning.

FINANCE, FINANCIAL SYSTEMS, BUDGETING, UML, EF, MVC, C #, CONTROLLER, SERVER, REST, MS SQL.

ЗМІСТ

Вступ........................................................................................................................................4

1 Інформація про підприємство............................................................................................5

* 1. Науково-технічна та виробнича діяльність НВП Хартрон-Аркос лтд...........5
  2. Структура пао «Хартрон» та НВП Хартрон-Аркос лтд..................................6

2 Аналіз проблемної області та постановка задачі..............................................................8

2.1 Теоретичні основи................................................................................................8

2.2 Аналіз аналогів......................................................................................................8

2.3 Постановка задачі...............................................................................................13

3 Перелік вимог до програмної системи............................................................................14

3.1 Призначення розробки.......................................................................................14

3.2 Вимоги до програмного продукту.....................................................................14

3.3 Вимоги до клієнта...............................................................................................15

4 Опис прийнятих проектних рішень.................................................................................16

4.1 Концептуальне моделювання предметної області...........................................16

4.2 Архітектура..........................................................................................................18

4.3 Проектування бази даних...................................................................................19

4.4 Вибір технологій розробки................................................................................21

Висновки................................................................................................................................24

Перелік посилань .................................................................................................................25

ВВЕДЕНИЕ

Планирование бюджета является очень важным фактором в любом предприятии, будь то большая корпорация или маленькое предприятие. Естественно в большинстве предприятий в штате сотрудников присутствуют бухгалтеры, занимающиеся бухгалтерским учетом и работающие со сложными системами для его ведения. Но далеко не всегда бухгалтеры принимают решения о распределении бюджета в компании. Часто этим занимается отдельно лицо, этим лицом может быть менеджер или даже владелец компании. Для такого человека нет необходимости разбираться в огромном количестве кредитов и дебетов. Ему было бы гораздо проще видеть всю финансовую картину целиком, возможно, в графическом виде, и манипулировать финансами используя удобный и интуитивно понятный интерфейс небольшого приложения. Также возможны случаи, когда решения о распределении финансов может принимать два или более человек, поэтому большим плюсом была бы возможность управления финансами предприятия одновременно нескольким лицам, синхронизируя свои действия.

Решением данной проблемы является разработка системы, которая позволит управлять финансами, не требуя специальных навыков. Главное в системе – ее понятность и простота использования. Система позволит пользователю добавлять единоразовую и периодическую прибыль, также определить периодические затраты (такие как выплата зарплат, затраты на обслуживание офиса и т.д.). Кроме периодических затрат присутствуют единоразовые затраты (к примеру единоразовая покупка нового рабочего иныентаря и прочие). Все эти данные пользователь определяет самостоятельно. Система позволит просматривать план бюджета в виде различных диаграмм, таблиц и списков, что упрощает понимание картины в целом, а простой интерфейс позволит безо всяких трудностей вносить изменения. Для удобства использования все финансовые изменения можно объединять в группы. Также система будет отслеживать состояние всех счетов, учитывая запланирыванные операции и отображать состояние счетов за определенный отрезок времени в удобном графическом виде, что позволит пользователю анализировать свои затраты по и в результате при необходимости сократить их.[1].

Система будет взаимодействовать с пользователем через веб-сайт, что делает ее кроссплатформенной и предоставляет возможность пользоваться ее с любого устройства через сеть интернет.

Система подлежит к использованию в малых предприятиях, небольших частных фирмах или в филиалах больших предприятий, но также не исключается использование системы для планирования домашнего бюджета.

2. АНАЛИЗ ПРОБЛЕМНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ

2.1 Теоретические основы

Данная система является частным случаем планировщика бюджета со своими фичами.

Планировщик бюджета – приложение, позволяющее вести учет финансовых средств, и управлять их распределением.

Основным назначением данной системы является управление бюджетом небольших предприятий. Система позволяет избавиться от постоянной путаницы с массой счетов и значительно упрощает расчёт финансов предприятия даже для ничего не смыслящего в финансах человека.

Большие и сложные системы бухгалтерского учета здесь не рассматриваются так как они не являются конкурирующими продуктами. Предприятиям с многочисленными финансовыми оборотами просто не обойтись без систем бухгалтерского учета. Но данная система предназначена не для бухгалтеров, а для управляющих компаниями, для которых в системах бухгалтерского учета будет присутствовать много лишней информации, что затруднит распределение финансов для них. Поэтому данная система может отлично работать параллельно с программными средствами для бухгалтерского учета.

2.2 Анализ аналогов

В данный момент уже существует масса подобных сервисов и приложений. Среди них очень много систем и приложений, предназначенных для управления домашним бюджетом. Либо наоборот функционал и общая концепция системы сильно приближается к концепции систем бухгалтерского учета. Ниже описаны несколько конкурирующих продуктов:

Money Care - финансовый планировщик для IOS и Andriod от компании StudioMoob. Приложение для дотошных счетоводов и просто больших любителей денег, готовых, как Скрудж Макдак, пересчитывать свои доходы каждую свободную минуту. Если вы не боитесь пожертвовать своим временем и удобствами в угоду точности учёта расходов, то это приложение определённо займёт должное место в вашем телефоне. При всей своей неказистости и внешней недружелюбности, Money Care обладает богатым функционалом, позволяющим учитывать массу разных мелочей: смотреть графики своих затрат, делить расходы на несколько человек и так далее[2].

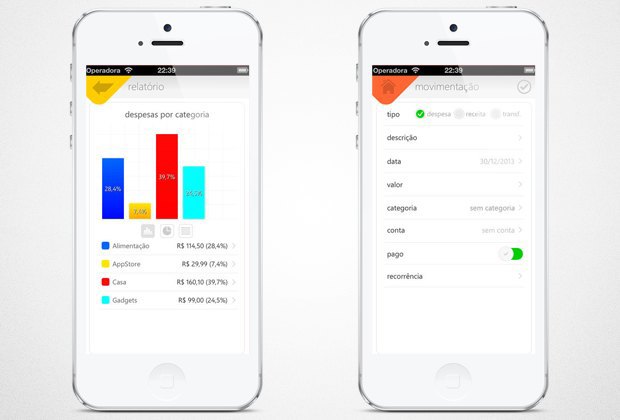


Рисунок 2.1 – Интерфейс приложения Money Care

К преимуществам данного приложения можно отнести возможность резервного копирования данных на почту, Dropbox или Google Drive а также экспорт данных в формате Exel.

Недостатками являются относительно сложный и неудобный интерфейс к примеру, для добавления поступлений и расходов: вбей название транзакции — подтверди, добавь сумму — подтверди, не забудь присвоить категорию — подтверди, а потом ещё раз подтверди всю операцию. Приложение создано только для двух мобильных платформ. Приложения является условно-бесплатным, бесплатно можно сделать до 90 записей, а для большего количества необходимо приобрести полную версию. Также отсутствует возможность планирования бюджета несколькими пользователями одновременно, с постоянной синхронизацией данных между ними.

Oracle Financial Analyzer OFA является самым многофункциональным инструментом моделирования бюджетов. Эта система позволяет применять формулы любой степени сложности, и проводить многомерный анализ данных в любом разрезе (например, анализ доходности компании за II квартал 2002 г. по каждому продукту с разбивкой по дилерам региона)[3].

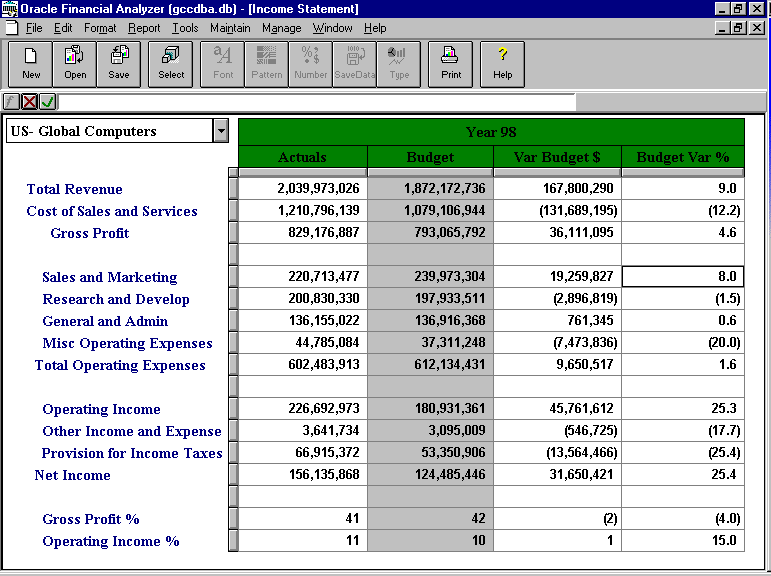


Рисунок 2.2 – Интерфейс приложения Oracle Financial Analyzer

В системе OFA могут работать сотрудники, находящиеся в различных городах, что очень важно для компаний с множеством филиалов. Правда, для этого необходимо установить в каждом филиале свой сервер, через который будут передаваться данные.

В системе есть гибкий контроль доступа к данным. Например, можно настроить программу таким образом, что руководитель будет видеть только агрегированные показатели по подразделениям, а подчиненный — только свои показатели и показатели в целом по компании. Можно также разделять финансовую модель на подмодели по подразделениям.

Однако у программы есть и недостатки. Интерфейс ее не очень нагляден, поэтому работа в OFA требует сложной и дорогостоящей настройки. Кроме того, система не полностью переведена на русский язык: практически весь раздел помощи доступен только в английском варианте. OFA имеет слишком обширный функционал, и использование ее обычным менеджером или владельцем малого предприятия будет запутанным, а также потребуется немало времени чтобы обучиться работать в системе.

В OFA не развиты средства документирования модели, то есть разработанную модель сложно описать с помощью встроенных инструментов, например, получить условную схему модели в виде рисунка или таблицы. Это серьезная проблема, которая встает особенно остро при смене персонала: если в компании нет аналитика, который занимается описанием всех моделей, объяснить новому сотруднику подробности вашей бюджетной инфраструктуры будет трудно.

Наконец, в OFA полностью отсутствуют встроенные средства, позволяющие вести документооборот. Учитывая, что система рассчитана на большое количество пользователей, из-за этого сразу появляются определенные трудности: например, начав работать с бюджетом, вы не знаете, внес ли туда свои данные тот или иной отдел. Поэтому для организации коллективной работы необходимо использовать дополнительное программное обеспечение.

Красный директор

Программный продукт предназначен для использования в повседневной деятельности менеджерами всех уровней и финансовыми директорами.

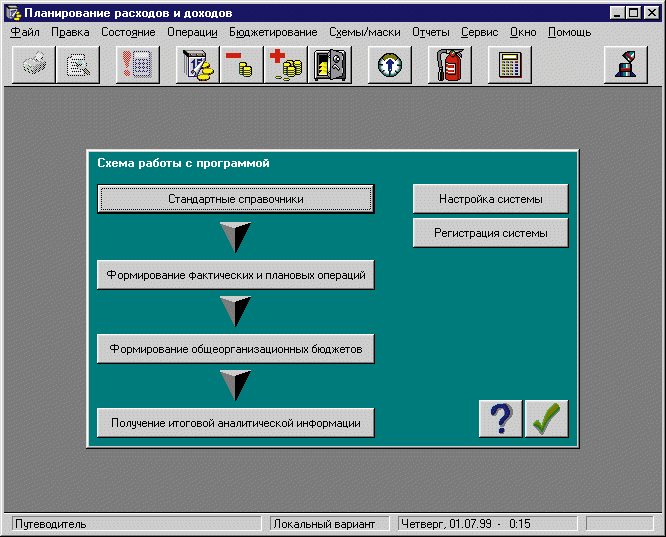


Рисунок 2.3 – Интерфейс приложения «Красный директор»

«Красный директор» — решение для небольших, возможно, для средних предприятий. Для ситуации, когда финансовый директор компании единолично строит бюджет нескольких подразделений, а схемы платежей просты и ведутся вручную, «Красный директор» вполне подойдет как дешевая альтернатива больших комплексов.

Система сделана как коробочный продукт, то есть не требует сложного внедрения, но и особой гибкостью не обладает — это готовое решение, а не инструмент.

Поскольку в «Красном директоре» не было необходимости создавать функционал для групповой работы, согласования бюджетов и многого другого, интерфейс достаточно прост и работу можно начинать сразу после установки. Этому существенно помогает и вынесение всех операций в одно окно с четко прорисованной рекомендованной последовательностью работы.

Бюджетирование, реализованное в программе «Красный директор», позволяет создавать бюджеты недельной (месячной, квартальной, годовой) периодичности для каждого отдельного подразделения организации, направления работы и т. п.

После ввода текущих осуществленных платежей/поступлений по трем видам источников денежных средств (расчетным счетам, кассам и неучтенным кассам) производится автоматический учет произведенных операций с точки зрения исполнения бюджета. Программа автоматически показывает отклонение доходных и расходных статей бюджета от плановых показателей в абсолютном выражении и в процентах.

Данные хранятся в простой файловой базе данных, оперировать с ними довольно сложно, интеграции с какими-либо системами управленческого учета нет, что, впрочем, вполне естественно для системы, ориентированной на небольшие компании.

У разрабатываемого проекта есть несколько преимуществ перед любым из вышеперечисленных для решения поставленных целей. В отличии от Money Care клиентским интерфейсом данной системы является веб-сайт, что делает ее кроссплатформенной. Также в Money Care отсутствует возможность планирования одного бюджета нескольким лицам, а доступный функционал нацелен больше на планирование домашнего бюджета, так как в нем можно только пользоваться определенными «группами расходов», а разрабатываемый проект в этом плане явлется очень гибким и универсальным. Проект позволяет пользователю настроить систему «под себя» создавая любое количество счетов и объединя их в группы. Oracle Financial Analyzer в свою очередь реализует весь необходимый функционал, но минусом в данном случае можно считать его «перенагруженность» другими функциями, которые просто не нужны обыкновенному руководителю небольшой компании или менеджеру. Другими словами использование Oracle Financial Analyzer позволит решить все задачи пользователя, но он будет задействовать только малый процент всего функционала, что не рентабельно. Не говоря об устаревшем и не удобном интерфейсе. Красный директор, в отличии от остальных конкурентов содержит только тот функционал, который в данном случае необходим. Но по сравнению с разрабатываемой системой большим его недостатком является отсутствие возможности многопользовательского планирования бюджета, а также возможности пользователя составлять шаблоны финансовых операций, просматривать состояние счетов, к примеру, за прошлый месяц в графическом виде и экспортирывать таблицы и диаграммы в PDF и EXCEL форматы.

Также отмечу что разрабатываемая система, в отличии от всех вышеперечисленных является полностью бесплатной и платформонезависимой.

2.3 Постановка задачи

Целью работы является проектирование системы корпоративного планирования и управления бюджетом. Данная система должна превосходить аналоги по функциональности и в простоте использования. Используя ее пользователь получает возможность создавать счета, добавлять единоразовые и периодические доходы/расходы, и управлять распределением финансовых ресурсов исходя из данных в системе, а также просматривать статистику состояния счетов за определенный отрезок времени в графическом виде.

Таким образом для достижения поставленной цели система должна поддерживать следующие основные функции:

* регистрацию новых пользователей;
* регистрацию\редактирование\удаление новых финансовых проектов;
* добавление\удаление счетов;
* добавление единоразовых операций со счетами (прибыль, убыток, перевод между счетами);
* добавление периодических операций со счетами (прибыль, убыток, перевод между счетами);
* соединение счетов в группы;
* добавление возможности нескольким пользователям работать с одним финансовым проектом;
* отображение текущего состояния счетов в виде списка и набора диаграмм;
* подсчет состояния счетов к определенной дате учитывая созданные пользователем финансовые операции и его отображение в виде списка счетов и диаграмм.

В ходе выполнения работы необходимо выполнить следующие задания:

* провести анализ предметной области;
* провести концептуальное и uml-моделирование предметной области;
* определить архитектуру системы;
* определить используемые программные средства и архитектурные решения для разработки;
* разработать систему;
* протестирывать систему.

Для реализации программного продукта выла выбрана среда разработки Microsoft Visual Studio. Сервер, являющийся вычислительным ядром системы будет выполнен в виде ASP .NET MVC WEB API приложения написанный на языке C#. Веб-сайт, позволяющий взаимодействовать пользовател с системой разработан с использованием фреймворка Angular, с использованием таких языков программирования как TypeScript и HTML. При выборе базы данных было решено выбрать Microsoft SQL Server. Также в системе необходимо наличие Windows службы для выполнения дополнительных операций с базой данных, также написанный на языке C#.

Все компоненты должны бать слабо связанны, а сервер разделен на несколько, также слабо связанных компонентов для упрощения их замены при необходимости и расширении функционала. Для этого необходимо использовать такие приемы программирования как внедрение зависимостей с использованием IoC-контейнера и использовать шаблон проектирования Repository.

3 ПЕРЕЧЕНЬ ТРЕБОВАНИЙ К ПРОГРАММНОЙ СИСТЕМЕ

Под требованиями программной системы понимают совокупность свойств, которые должна иметь реализуемая система[4]. Ниже описаны требования к данной программной системе.

3.1 Предназначение разработки

Система планирования корпоративного бюджета предназначена для управления финансовыми средствами в небольших предприятиях. Основная цель системы максимально упростить процесс управления финансами, избавляя пользователя в потребности приобретении специальных навыков использования крупных систем бухгалтерского учета или дополнительном финансовом образовании. Идея заключается в том, что после проведения некоторых конфигураций пользователь будет видеть всю финансовую картину его предприятия перед собой в виде диаграмм и списков и сможет спланировать распределение финансовых ресурсов предприятия в несколько кликов, а также помоч проанализировать состояние финансовых счетов за определенный отрезок времени. Как говорилось ранее потенциальными пользователями системы являются люли, занимающиеся распределением и учетом финансов не вдаваясь в подробности, такие как менеджеры или владельцы небольших предприятий.

3.2 Требования к программному продукту

По первичным требованиям система должна представлять из себя сервер, веб-сайт и Windows службу. Для разработки сервера необходимо использовать ASP.NET MVC WEB API. Программный код должен быть написан на языке C#. Для упрощения процесса сопровождения продукта при разработке сервера использовать подходы разработки Dependency Injection исполльзуя IoC-контейнер StructureMap и шаблона проектирования Repository.

Серверная часть является вычислительным ядром системы и должна выполнять все ее основные функции. К основным функциям относится:

* регистрация пользователей системы;
* авторизация пользователей в системе;
* управление конфигурациями пользовательских аккаунтов;
* выдавача разрешений другим пользователям управлять распределением финансов;
* добавление\редактирование\удаление финансовых пректов;
* Добавление\удаление счетов;
* просмотр распределения средств между счетами и группами счетов в виде диаграмм/списков;
* добавление единоразовых операций со счетами (прибыль, убыток, перевод между счетами);
* добавление периодических операций со счетами (прибыль, убыток, перевод между счетами);
* соединение счетов в группы;
* добавление возможности нескольким пользователям работать с одним финансовым проектом;
* отображение текущего состояния счетов в виде списка и набора диаграмм;
* просмотр истории состояния счетов за определенный отрезок времени;
* экспорт отчетов (диаграмм и списков счетов) в EXCEL и PDF;
* подсчет состояния счетов к определенной дате учитывая созданные пользователем финансовые операции и его отображение в виде списка счетов и диаграмм.

Также необходимо реализовать веб-сайт, который является пользовательским интерфейсом всей системы. Именно с помощью веб-сайта пользователи системы будут получать доступ к ее функционалу Веб-сайт должен быть реализован с использованием фреймворка Angular, на таких языках программирования как TypeScript, HTML. Верстка веб-страниц должна быть проведена с использованием Bootstrap framework.

Windows сервис необходимо написать на языке C#, при разработке возможно использование библиотек работы с базой данных сервера системы

В качестве системы управления базами данных необходимо использовать Microsoft SQL Server. Для связи с базой данных использовать Entity Framework.

3.3 Требования к клиенту

Данная система, как любая другая нуждается в обслуживании, поэтому ее пользователи должны соблюдать некоторые требования. К этим требованиям относятся:

* для использования системы необходимо быть зарегистрированным в ней;
* для использования системы необходимо подключение к сети интернет;
* в начале работы необходимо провести конфигурацию системы (создать счета, добавить опрации).

4 ОПИСАНИЕ ПРИНЯТЫХ ПРОЕКТНЫХ РЕШЕНИЙ

4.1 Концептуальное моделирование предметной области

Концептуальная модель ⎯ описание структуры проекта, выполненное с использованием таблиц, диаграмм, и других средств, понятных всем людям, работающим над проектом. Основным компонентом концептуальной модели является описание объектов предметной области и связей между ними.

В данной системе финансовым проектом является созданная пользователем сущность, которая описывает одно предприятие, и в рамках которой можно создавать счета и проводить прочие финансовые операции.

Всех пользователей данной системы можно разбить на две категории:

1. пользователи-владельцы – те пользователи, которые создали финансовый проект на своем аккаунте;
2. остальные пользователи – пользователи которым дали право редактировать финансовый проект, созданный другим пользователем.

У пользователей обеих категорий выделяются одинаковые требования к функционалу системы:

Потребности основного количества пользователей:

* зарегистрироваться в системе;
* авторизоваться в системе;
* создать финансовый проект;
* дать пользователю права на редактирование финансового проекта;
* забрать у пользователя права на редактирование финансового проекта;
* удалить финансовый проект;
* добавить счет;
* просмотреть распределение средств на счетах и группах счетов в виде диаграмм /списков;
* добавить единоразовые операции со счетами (прибыль, убыток, перевод между счетами);
* добавить периодические операции со счетами (прибыль, убыток, перевод между счетами);
* добавить группу;
* добавить счет в группу;
* просмотреть историю финансовых операций по счету;
* рассчитать состояние счетов к определенной дате с учетом заданных фиансовых операций;
* экспортировать отчет в виде диаграммы/таблицы в PDF или EXCEL;
* посмотреть историю состояния счетов за определенный отрезок времени.

Потребности пользователя-владельца включают в себя потребности обычного пользователя, а также расширяются возможностью выдавать разрешение другим пользователям управлять распределением финансов вашего предприятия.

Исходя из наведенных выше данных составим диаграмму прецедентов, изображенную на рисунке 4.1.

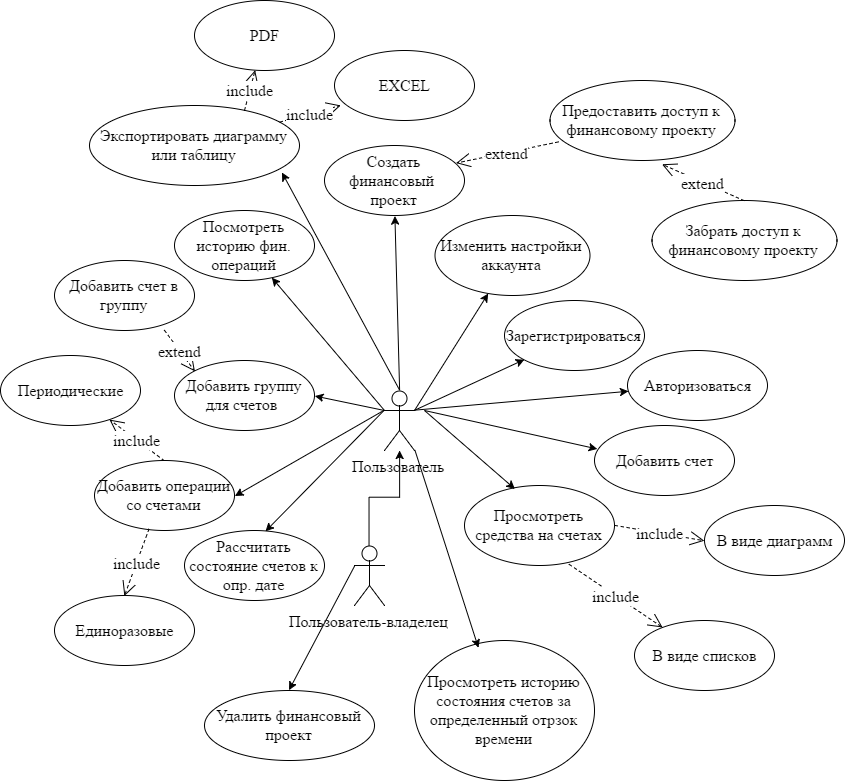


Рисунок 4.1 – Диаграмма прецедентов

Из диаграммы прецедентов видим, что пользователь-владелец имеет тот же функционал что и обычный пользователь (на диаграмме это отображено линией наследования), но обычный пользователь, в свою очередь не может дать разрешение другому пользователю управлять финансами.

Также хочется отметить что функция предоставления добавления группы в счет расширяет функцию добавления группы для счетов это значит, что пользователь не может добавить счет в еще не созданную группу.

4.2 Архитектура

Для реализации проекта было решено использовать трехуровневую архитектуру. Это решение было принято так как трёхуровневая архитектура обеспечивает, как правило, большую масштабируемость (за счёт горизонтальной масштабируемости сервера приложений и мультиплексирования соединений), боольшую конфигурируемость (за счёт изолированности уровней друг от друга). На рисунке 4.2 изображена структура проекта.

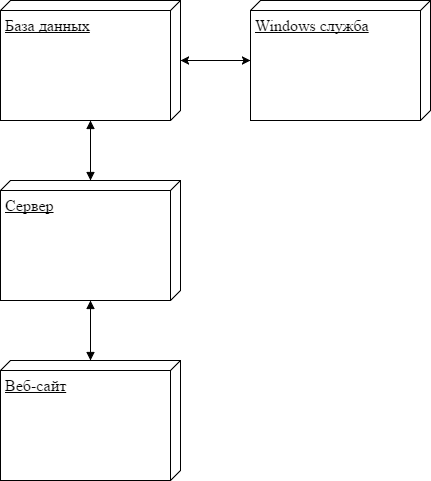


Рисунок 4.2 – Структура архитектуры проекта

Трёхуровневая архитектура (трёхзвенная архитектура, англ. three-tier) — архитектурная модель программного комплекса, предполагающая наличие в нём трёх компонентов: клиента, сервера приложений (к которому подключено клиентское приложение) и сервера баз данных (с которым работает сервер приложений)[6].

Как видно на рисунке 4.2 в данной системе клиентом выступает веб-сайт, далее идет сервер приложений, который управляет работой всей системы, последним элементом является сервер базы данных. Также с базой данных работает Windows служба, которая ежедневно обновляет состояние счетов и сограняет его для ведения статистики.

Использование трехуровневой архитектуры отделяет часть UI от логической части, в также выводит сервер базы данных на новый уровень, что делает систему безопаснее и менее тесно связанной.

Как дополнительные архитектурные решения, в целях упрощения сопровождения и дальнейшей разработки продукта решено использовать следующие шаблоны проектирования:

Внедрение зависимости (англ. Dependency injection, DI) — процесс предоставления внешней зависимости программному компоненту. Является специфичной формой «инверсии управления» (англ. Inversion of control, IoC), когда она применяется к управлению зависимостями. В полном соответствии с принципом единой обязанности объект отдаёт заботу о построении требуемых ему зависимостей внешнему, специально предназначенному для этого общему механизму;

Репозиторий (англ. Repository) — посредничает между уровнями области определения и распределения данных (domain and data mapping layers), используя интерфейс, схожий с коллекциями для доступа к объектам области определения.

Серверную часть как свмую большую часть проекта решено разделить на компоненты (переносимые библиотеки). Структура компонентов серверной части а также отношения между ними изобрадены на рисунке 4.3.

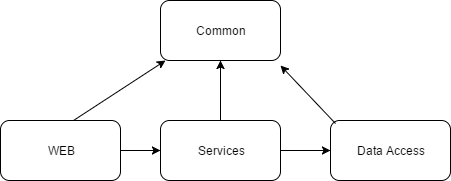


Рисунок 4.3 – Структура комопнентов сервера

Как видно на рисунке выше серверная часть состоит из четырех основных компонентов. Первым является проект, содержащий WEB API контроллеры, которые отвечают за связь сервера с клиентами. Следующим идет библиотека с сервисами. В этой библиотеке производятся все основные вычислительные операции системы. Библиотека Data Access отвечает за связь сервера с базой данных. Последней библиотекой является Common. Как видно на рисунке – все остальные библиотеки ссылаются на нее так как она содержит общие элементы для всей системы, такие как серверныеошибки, которые могут выбрасываться из любого из серверных компонентов. Благодаря такой структуре сервера появляется возможность полностью заменить любой из его компонентов не внося изменения в другие.в данном случае это осуществлено с помощью инструмента StructureMap, который позволяет наладить слабую связанность между компоненнтами сервера, пользуясь только интерфейсами.

4.3 Проектирование базы данных

Для базы данных было решено использовать MS SQL Server так как все остальные элементы системы также будут реализованы средствами, разработки от Microsoft, что предоставит меньше трудностей при разработке. Также одной из причин является наличие опыта работы с данной системой.

Процесс проектирования базы данных представляет из себя выделение основных свойств сущностей системы и последующая их нормализация.

Нормализация предназначена для приведения структуры БД к виду, обеспечивающему минимальную логическую избыточность, и не имеет целью уменьшение или увеличение производительности работы или же уменьшение или увеличение физического объёма базы данных. Конечной целью нормализации является уменьшение потенциальной противоречивости хранимой в базе данных информации. Общее назначение процесса нормализации заключается в следующем:

* исключение некоторых типов избыточности;
* устранение некоторых аномалий обновления;
* разработка проекта базы данных, который является достаточно «качественным» представлением реального мира, интуитивно понятен и может служить хорошей основой для последующего расширения;
* упрощение процедуры применения необходимых ограничений целостности.

Определим свойства, необходимые для основных сущностей системы:

User – сущность, описывающая пользователя системы. Для него необходимы следующие свойства:

* имя пользователя;
* логин для авторизации;
* пароль;
* список финансовых проектов, открытых для редактирования пользователем.

FinanceProject – сущность финансового проекта, в рамках которого будут создаваться счета, проводиться финансовые опреации и рассчеты. Эта сущность является прообразом предприятия. Ниже приведены свойства финансового проекта:

* имя проекта;
* пользователь-владелец проекта;
* список пользователей, которые могут редактировать этот проект;
* список групп счетов.

Группа счетов (BillGroup) – сущность для связи счетов в группы. Группирование упростит поиск счетов пользователем, а также даст возможность строить отчеты по группам счетов. Группа счетов должна содержать следующие свойства:

* имя группы;
* список счетов.

Bill – финансовый счет. Это сущность, про которой будут вестись все рассчеты. Пользователь системы создает счета для чего он сам захочет. Ниже описанны свойства финансового счета:

* имя счета;
* текущий баланс.

Одна из основных сцщностей системы – финансовая операция (Operation). Финансовые операции проедставляют собой миграцию средств между счетами. Важно отметить что оперции нас счетами могут быть как единоразовыми, так и перидическими (с определенным интервало выполнения). Таким образом сущность для операций должна содержить следующие свойства:

* счет с которого переводятся средства (поле меожет принимать значение null так как предпологается возможность поступления средств извне);
* счет, на который переводятся средства (поле меожет принимать значение null так как предпологается возможность выхода средств из финансового проекта);
* дата проведения операции;
* количество дней между повторением оперции (если это число равно нулю, то операция считается единоразовой);
* пользователь, который создал операцию;
* сумма перевода;
* описание для операции.

Последней сущностью является история состояния счета для ведеиня статистики. Для нее необходимыми свойствами являются:

* счет для которого записывается история;
* дата сохранения состояния счета;
* состояние его баланса.

Проанализировав сущности системы и их аттрибуты составим первоначальный вариант таблиц базы данных, а также приведем их к 1НФ. Переменная отношения находится в первой нормальной форме (1НФ) тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов.

User

* Name;
* Login;
* Password;
* FinanceProject.

FinanceProject

* Name;
* Owner;
* User;
* BillGroup.

BillGroup

* Name;
* Bill.

Bill

* Name;
* Balance.

Operation

* BillFrom;
* BillTo;
* Date;
* Interval;
* User;
* Amount;
* Description.

History

* Bill;
* Date;
* Balance.

База данных находится в 1НФ, так как все ее кортежи содержат только одно значение для каждого аттрибута, но в текущем состоянии база данных не может корректно функционировать, поэтому необходимо продолжить процесс нормализации над ней. Переведем таблицы в 2НФ, добавив первичные ключи и наладив связь между сущностями таблицы. Переменная отношения находится во второй нормальной форме тогда и только тогда, когда она находится в первой нормальной форме и каждый неключевой атрибут неприводимо (функционально полно) зависит от её потенциального ключа.

Между таблицами User и FinanceProject присутствует связь многие-ко-многим. Поэтому нужно добавить третью, связующую таблицу. Также сразу определим внешние ключи для таблиц, тем самым наладив отношения между ними.

Permit

* Id;
* UserId;
* FinanceProjectId.

User

* Id;
* Name;
* Login;
* Password.

FinanceProject

* Id;
* Name;
* OwnerId.

BillGroup

* Id;
* Name;
* FinanceProjectId;

Bill

* Id;
* Name;
* Balance;
* BillGroupId.

Operation

* Id;
* BillFromId;
* BillToId;
* Date;
* Interval;
* UserId;
* Amount;
* Description.

History

* Id;
* BillId;
* Date;
* Balance.

Теперь все таблицы имеют первичный ключ, а также связанны между собой внешними ключами.

Для правильного функционирования базы данных достаточным условием является нахождение ее таблиц в 3НФ.

Переменная отношения находится в третьей нормальной форме (3НФ) тогда и только тогда, когда она находится во второй нормальной форме, и отсутствуют транзитивные функциональные зависимости неключевых атрибутов от ключевых.

Проанализирывав описанные выше таблицы можно сделать вывод что в них отсутствуют транзитивные функциональные зависимости неключевых атрибутов от ключевых, а соответственно база данных отвечает минимальным требованиям и готова к использованию.

Исходя из приведеных выше таблиц построим схему базы данных данного проекта, которая наглядно будет отразать все сущности, всязи между ними а также свойства этих сущностей.

Схема базы данных приведена ниже на рисунке 4.4.

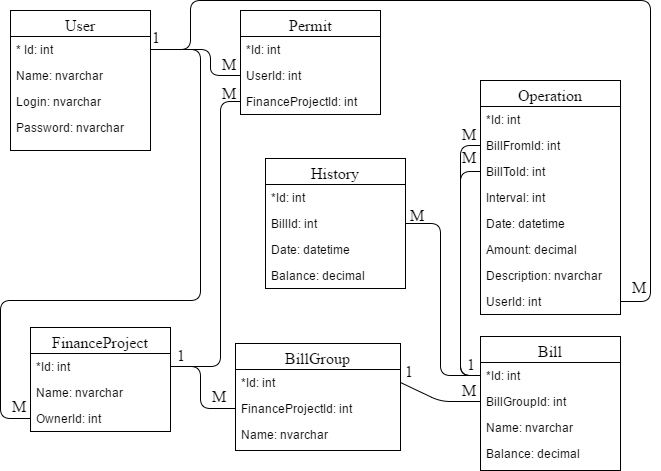


Рисунок 4.4 – Схема базы данных

На приведенной выше схеме можно увидеть все сущностисистемы, которые будут хранить ся в базе данных, а также связи между ними. Первичные ключи отмечены «\*». В системе таблицы User и FinanceProject имеют связь многие-ко-многим, это реализованно с помошью вспомогательной таблицы Permit. Остальные связи между таблицами в базе данных один-ко-многим. А именно финансовый проект может иметь много групп счетов, группа счетов может иметь много счетов, а счет в свою очередб может иметь много операций.

4.4 Выбор инструментов и технологий разработки

Для имплементации системы решено использовать высокоуровневые языки программирования такие как C#, TypeScript. Верстка веб-страниц будет выполняться на языке HTML с использоваинем CSS.

Под инструментами разработки подразумеваются программные средства, ускоряющие и упрощающие процесс разработки программного обеспечния. При разработке данного продукта предполагается использование таких инструментов как:

* Visual Studio 2015 — интегрированный набор средств обеспечения производительности разработчика, облачных служб и расширений, которые позволяют создавать уникальные приложения и игры для Интернета, Магазина Windows, настольных компьютеров, для Android и iOS.
* SQL Server Management Studio (SSMS) — это интегрированная среда для доступа, настройки, администрирования и разработки всех компонентов SQL Server, а также управления ими. Среда SSMS сочетает в себе обширный набор графических инструментов с рядом отличных редакторов скриптов, обеспечивая разработчикам и администраторам любой квалификации доступ к SQL Server.
* ReSharper (R#) — дополнение (плагин), разработанное компанией JetBrains для повышения продуктивности работы в Microsoft Visual Studio.Проводит статический анализ кода (поиск ошибок в коде до компиляции) в масштабе всего решения, предусматривает дополнительные средства автозаполнения, навигации, поиска, подсветки синтаксиса, форматирования, оптимизации и генерации кода, предоставляет 40 автоматизированных рефакторингов, упрощает юнит-тестирование в средах MSTest и NUnit и др. Поддерживает языки программирования C# и VB.NET, а также предоставляет средства для работы с ASP.NET, ASP.NET MVC, XML, XAML, сценариями сборки NAnt и MSBuild.

Далее описаны различные фреймворки и программные библиотеки, которые решено использовать при зазработке данного программного продукта:

– REST Services (сокр. от англ. Representational State Transfer — «передача репрезентативного состояния») — метод взаимодействия компонентов распределённого приложения в сети Интернет, при котором вызов удаленной процедуры представляет собой обычный HTTP-запрос (обычно GET или POST), а необходимые данные передаются в качестве параметров запроса. Этот способ является альтернативой более сложным методам, таким как SOAP, CORBA и RPC. В широком смысле REST поддерживает концепцию построения распределённого приложения, при которой компоненты взаимодействуют наподобие взаимодействия клиентов и серверов во Всемирной паутине. Этот метод в данной программной системе используется для общения между сервером и клиентскими приложениями (веб-сайтом и эмулятором запирающего устройства) [7].

– Model-view-controller (MVC, «модель-представление-контроллер», «модель-вид-контроллер») — схема использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные. Данная схема проектирования часто используется для построения архитектурного каркаса, когда переходят от теории к реализации в конкретной предметной области. Основная часть программной системы, сервер – написана с использованием шаблона MVC.

– Microsoft SQL Server — система управления реляционными базами данных (РСУБД), 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.

– ADO.NET Entity Framework (EF) — объектно-ориентированная технология доступа к данным, является object-relational mapping (ORM) решением для .NET Framework от Microsoft. Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL. Для облегчения построения web-решений используется как ADO.NET Data Services (Astoria), так и связка из Windows Communication Foundation и Windows Presentation Foundation. Entity SQL представляет собой язык, подобный языку SQL, который позволяет выполнять запросы к концептуальным моделям в Entity Framework. LINQ to Entitis – это альтернативный интерфейс LINQ API, используемый для обращения к базе данных. Для доступа к данным используется только этот фреймворк.

– Json.NET — фреймворк от NewtonSoft, который уменьшает затраты производительности и сил на серриализацию и десерриализацию объектов. Используется так как при общении между сервером и клиентами данные передаются в формате Json.

– Bootstrap 3 — фреймворк для разработки адаптивных и мобильных web-проектов. С помощю него создан веб-интерфейс системы.

* Angular представляет фреймворк от компании Google для создания клиентских приложений. Прежде всего он нацелен на разработку SPA-решений (Single Page Application), то есть одностраничных приложений. В этом плане Angular является наследником другого фреймворка AngularJS. В то же время Angular это не новая версия AngularJS, а принципиальной новый фреймворк.

– Highcharts — библиотека для создания чартов написанная на JavaScript, позволяет легко добавлять интерактивные, анимированные графики на сайт или в веб-приложение. На данный момент чарты поддерживают большое количество диаграмм линейных, круговых, колоночных рассеивающих и многих других типов.

Инверсия управления — важный принцип объектно-ориентированного программирования, используемый для уменьшения зацепления в компьютерных программах. Также архитектурное решение интеграции, упрощающее расширение возможностей системы, при котором контроль над потоком управления программы остаётся за каркасом. Одной из реализаций IoC в применении к управлению зависимостями является внедрение зависимостей. Внедрение зависимости используется во многих фреймворках, которые называются IoC-контейнерами.

* StructureMap является самым старым, постоянно используемым контейнером IoC / DI для .Net, относящимся к его первой публичной версии и использованию продукции вплоть до июня 2004 г. на .Net 1.1. Нынешний выпуск 4. \* представляет собой более 12 лет опыта, накопленных в StructureMap и более широком сообществе .Net, а также уничтожает множество концептуальных решений, которые уже не имеют смысла сегодня.

5 ОПИСАНИЕ ПРОГРММНОГО ПРОДУКТА

Даннаясистема является универсальным среством для планирывания и анализа как бюджета компании так и личных расходов. Основным приимуществом системы является простота и гибковсть в испоьзовании. В программе нет предопределенных категорий и заумной финансовой терминологии. Состояние средств на счетах, а также их движение отображено в удобном виде, и позволяет планировать бюджет не ямея дополнительных знаний в предметной области. Гибкость систебмы обксловлена тем что пользователь может сконфигурирывать свой финансовый проект полностью так как он захочет. Пользователь сам создаст нужные ему счета, объеденит и в группы и сразу сможет просматривать отчеты по состоянию как счетов, так и групп счетов.

5.1 Описание структуры проекта

Как и предпологалось на стадии разработки вся система состоит из сервера, веб-сайта и Windows-службы.

Сервер – оснавная вычислительная часть системы. Для удобства его сопровождения сервер был разделен на четыре проекта:

* Finance.Web;
* Finance.Services;
* Finance.DataAccess;
* Finance.Common.

Первая часть Finance.Web – проект, задачей которого является общение сервера с клиентами. В нем находятся WEB API контроллеры, на которые клиенты посылают HTTP запросы.

Ниже приведена структура данного проекта:

* app;
* App\_Start;
* Content;
* Controllers;
* DependencyResolution;
* Filters;
* Helpers;
* Interfaces;
* Providers;
* Scripts;
* Views;
* favicon.ico;
* Global.asax;
* package.json;
* packages.config;
* Startup.cs;
* systemjs.config.js;
* tsconfig.json;
* Web.config.

app – каталог в котором содержится веб-сайт, созданный с помощью Angular.

Он содержит в себе каталоги моделей, сервисов и компонентов описанные ниже.

\_directives – кталог содержащий компоненты «многоразового» использования, а именно alert.component и popup.component. Оба компонента являются кастомными контроллами. alert.component служит для вывода пользователю сообщений об ошибках, и предупреждениях. popup.component представляет из себя всплывающее окно, содержание которого изменяется в зависимости от контекста. Он используется для создания финансовых операций и как всплывающее окно для подтвержения удаления.

В данном проекте каждый Ayngular-компонент чаще всего представляет из себя каталог, содержащий три файла, к примеру alert.component содержит:

alert.component.html – html-страница для данного компонента.

alert.component.ts – файл, написанный на языке TypeScript, описывающий логику работы компонента. В нем есть единственный метод-событие OnInit, который отрабатывает при згрузке компонента и отрисовывает переданное сообщение.

index.ts – сециальный файл для экспорта класов. Классы экспортируются для того чтобы другие классы моли их использовать, а файл index создан для упрощения работы с помледующим импортом.

popup.component содержит такю же структуру файлов и представляет из себя всплывающее окно, контент которого задается отдельно.

Следующий каталог auth.guard, в нет нет HTML страницы, и занимается отслеживанием авторизованности пользователя.

Далее в каталоге app находится подкаталог \_models. В нет содержатся модели, использующиеся в системе. Если быть точным – это вьюмодели (viewmodels). Они созданы таким образом, чтобы их было легко отображать в пользовательском интерфейсе, и также передвавть меджу сервером и клиентом. В проекте веб-сайта используются вследующие модели:

* user – модель пользователя системы;
* financeProject – модель финансового проекта;
* billGroup – модель для представления группы счетов;
* bill – модель для представления финансового счета;
* billiStoryItem – модель, представляющая из себя один элеиент исмории состояния счета.

Следующий каталог \_services. В нем находятся файлы с сервисами для системы. Сервисы также не имеют представления на пользовательском интерфейсе, а служат как вспомогатеьные компоненты для работы web-сайта и аго связи с сервером.

Ниже приведен список всех сервисов:

* auth.service сервис для аутентификаци пользователя, содержищий методы login и logout;
* bill.service – сервис для работы со счетами, содержит методы для получения информации о счетахи группах, а также их добавления и удаления;
* error.service – сервис для перехвата ошибок, возвращенных из сервера, и последующего их отображения;
* export.service – сервис управляющий экспортом отчетов в форматы PDF и EXCEL;
* date.service – сервис для работы с датами, содержащий методы математических операций над датами;
* finance-project.service – сервис, предоставляющий методы для добавления и удаления финансовых проектов из системы, а также получения с сервера информации о них и редактирования их свойств;
* global.service – сервис, предоставляющий доступ к глобальным переменным на клиентской част приложения, используется для сохранения состояния приложения;
* http.service – сервис для работы с http, в нем есть методы GET и POST для обмена сообщениями с сервером;
* operation.service – сервис для получения информации о финансовых операциях, их добавления и удаления;
* plan.service – сервис для построения прогноза состояния счетов к определенному моменту времени. Содержит методы для получения информации о состоянии счетов к определенному моменту как всех, так и по отдельности;
* register.service – сервис для регистрации новых пользователей;
* report.service – сервис для построения отчетов. Получает истории состояния счетов зв выбранный период времени;
* user.service. – сервис для получния информации о пользователях в системе.

Следующие подкаталоги это компоненты веб-сайта. Как было описанно ранее все эти компоненты состоят из каталога, содержащего три файла: HTML страницу, TypeScript файл и файл index.ts для экспорта классов. Повторюсь что каждый компонент содержит эти три файла, поэтому каждый из них по отдельности описываться не будет, а будет описываться компонент в целом.

Ниже приведен список и описанны все компоненты веб-сайта:

* home – компонент домашней страницй сайта, отображает текущее состояние счетов в виде списка и нескольких графиков;
* login – компонент для авторизации пользователя;
* operation – компонент для добавления финансовых операций, этот компонент встраивается в popup.component описанный выше. В нем содердатся все поля для создания финансовой операции;
* plan-budget – компонент для планирования бюджета, в нем пользователь сможет простмотреть состояние счетов к опрделенному периоду и все финансовые операции, привязанные к выбранному счету;
* project-crud – этот компонент представляет из себя страницу для редактирования финансового проекта, добавления или удаления счетов, также он позволяет давать или отбирать доступ к редактированию данного проекта у других пользователей;
* register – компонент, представляющий собой страницу регистрации новых пользователей системы;
* report -компонент, в котором отображен отчет о состоянии счетов за выбранный отрезок времени. Отображается отчет в виде нескольких диаграмм.

Также хочется отметить что во всех компонентах, где присутствуют списки счетов или диаграмы присутствует функция экспорта в EXCEL или PDF.

Далее в папке app следуют несколько файлов, а именно:

* app.component.html – HTML представление гдавного компонента системы. В нем нахолится навигационное меню сайта и контейнер для остальных компонентов;
* app.component.ts – TypeScript файл для основного компонента приложения. В нем содержится логика работы навигационного меню сайта;
* app.module.ts – один из конфигурационных файлов Angular приложения, в нем указуются все компоненты сервисы и прочие части сстымы для того чтобы при старте прилодения правильно подтянулись все зависимости;
* app.routing.ts – еще один конфигурационны файл приложения. В нем описана вся навигация в приложении. Хоть приложение и является одностраничным, содержание веб-сайта изменяется в зависимости от того что написанно в адресной строке браузера.;
* main.ts – точка входа приложения веб-сайта.

Ниже приведен пример TupeScript файла компонента приложния.

//Секция импортов классов  
import { Component, OnInit } from '@angular/core';

import { Router, ActivatedRoute } from '@angular/router';

import { GlobalService as Global, UserService, AlertService, FinanceProjectService} from "./\_services/index";

import {FinanceProjectViewModel} from './\_models/index';

//Декоратор компонента  
@Component({

moduleId: module.id,

selector: 'app',

templateUrl: 'app.component.html'

})

//Тело класса

export class AppComponent implements OnInit{

fProjects: Array<FinanceProjectViewModel> = new Array<FinanceProjectViewModel>();

constructor(private userService: UserService,

private route: ActivatedRoute,

private router: Router,

private alertService: AlertService,

private projectService: FinanceProjectService) {

Global.updateIsLoggedIn();

}

ngOnInit() {

this.getProjects();

}

getProjects() {

this.projectService.getAll().subscribe(

(data: Array<FinanceProjectViewModel>) => {

this.fProjects = data;

this.selectProject(this.fProjects[0].id);

},

error => this.alertService.error(error));

}

}

Как видно в листинге программного кода, компонент стостоит из трех основных частей. Первая – секция импортов, в которой к компоненту добавляются другие классы. Далее идет декоратор компонента, в него передается ид модуля для роутинга, а также указывается ссылка на HTML часть компонента и HTML тэг, в который будет помещаться этот компонент.

Следующим каталогом в проекте Finance.Web является App\_Start. Этот каталог содержит файлы:

AppUserManager – класс для проведения авторизации пользователя используя Claims;

BundleConfig – конфигурационный файл в котором настраиваются Bundles.Это средства для подключения скриптов и стилей к проекту;

FIlterConfig – конфигурационный класс для фильтрации запросов;

RouteConfig - класс, в котором задается шаблон маршрутов. Также в нем указывается стандартный маршрут, то есть маршрут, который откроется если в адресной строке будет только адрес веб-сайта. Стандартно шаблон маршрутов выглядит как «имя контроллера/имя метода действия/переданное значение» но MVC позволяет создать любой маршрут;

Startup.Auth – конфигурационный файл для Owin аутентификации;

StructuremapWebApi – файл, в котором создается объект IoC-контейнера при старте приложения;

WebApiConfig. класс, аналогичный RouteConfig.cs. Он несет в себе тот же функционал только для Web API.

Следующий каталог Content. Этот каталог содержит ресурсы проекта, в данном случае в нем хранятся .css стили элементов интерфейса. Присутствуют как стили Bootstrap, так и собственные.

Следующий каталог – Controllers. Один из основных каталогов MVC проекта. Он содержит в себе контроллеры. Контроллер обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель для реализации необходимой реакции. В данном проекте несколько контроллеров:

* AccountController – контроллер, содержащий методы регистрации и выхода из сети;
* BillController – контроллер, выполняющий операции добавления и удаления счетов и групп счетов, а также предоставления информации об их состоянии;
* ExportControlle – контроллер, отвечающий за экспорт диаграмм и списков в PDF и EXCEL;
* HomeController – контроллер, который отрабытывает на старте приложения, он загружает стартовую страницу вместе с Angular приложением;
* OperationController – контроллер для передачи клиентам мнформации об операциях, а также для их добавления и удаления;
* PlanController – контроллер, возвращающий предположительное состояние счетов и групп счетов к указанной дате, учитывая заданные финансовые операции;
* ProjectController – конроллер, позволяющий получать информацию об проекте, а также редактировать и удалять его;
* ReportController – контроллер возвращающий статистику состояния счетов за определенный отрезок времени;
* UserController – контроллер, отвечающий за выдачу инвормацции о пользователях системы.

Каталог DependencyResolution содержит файлы для внедрения зависимостей, а именно каонфигурационные файлы(DefaultRegestry), файл IoC-контейнера IoC, и несколько файлов для корректной работы StructureMap (StructureMapDependencyScope, StructureMapScopeModule, StructureMapDependencuResolver).

Далее находится каталог Filters. В нем находится два фильтра ошибок. Первый CustomExceptionFilter отлавливает ошибки, которые выбрасывает сам сервер, а OopsExceptionFilter отлавливает остальные ошибки при работе сервера и изменяет статус код ответа сервера и передает сообщение об ошибке.

Следующий каталог Helpers, содержащий единственный класс HelperService, который сдужит для получения данных пользователя который авторизовался в системе.

Сдедующим каталогом яаляется каталог Intefaces. В нем находятся интерфейсы сля классов в системе.

Каталог Providers тоже содержит единственный класс ApplicationAuthProvider, необходивый для связи аутентификационного приспособления Owin и базы данных пользователей.

Scripts – каталог, содержащий JavaScript скрипты. В нем содержатся скрипты из различных библиотек, используемых проектом.

Последний каталог в этом проекте это Views. В нем есть два подкатаога. Вобщем здесь хранится четыре веб страницы, написанные на Razor. Это Index, \_Layout, Error и \_ViewStart. На всех этих страницах Подключается Angular прилажение. То есть они загружаются и запускают веб-сайт на Angular.

Следующими файлами в проекта Finance.Web являются:

* favicon.ico – иконка сайта для браузера;
* Global.asax – класс MVCApplication, являющийся точкой входа всего приложения;
* package.json – конфигурационный файл, описывающий библиотеки, необходимые для работы Angular приложения;
* packages.config – конфигурационный файл, содердащий список необходимых библтотек для функционирования Finance.Web приложения;
* Startup.cs – класс, экземпляр которого создается при старте приложения;
* systemjs.config.js – конфигурационный файл для корректной работы Angular приложения, также его наличие позволяет отлаживатьTypeScript код;
* tsconfig.json – конфигурационный файл для TypeScript интерпретатора;
* Web.config – конфигурационный файл всего проекта, в нем прописываются подключения к базам данных, требуемые версии .Net Framework и прочего.

Следующим компонентов сервера является библиотрека Finance.Services. В данной библиотеке выполняются все вычислительные действия и оня является всязующим звеном проекта Finance.Web с проектом Finance.DataAccess.

Структура этой библиотеки описана ниже:

* Interfaces;
* ViewModels;
* BillService;
* ExportService;
* FinanceProjectService;
* OperationService;
* PlanService;
* ReportService;
* UserService.

Interfaces – каталог, содержащий набор интерфейсов для сервисов системы. Эти интерфейсы нужны для внедрения зависимостей StructureMap. Таким образом для каждого сервича есть итерфейс, полностью описывающий его. Присутствуют интерфейсы:

* IBillService;
* IExportService;
* IFinanceProjectService;
* IOperationService;
* IPlanService;
* IReportService;
* IUserService.

Следующий каталог ViewModels. В нем находятся вьюмодели. Чаще всего они аналогичны моделям и представлены в таком виде, который проще всего будет отображать в пользовательском интервейсе. В каталоге находятся следующие вьюмодели:

* BillViewModel – модель счета;
* BillGroupViewModel – модель группы счетов;
* FinanceProjectViewModel – модель финансового проекта;
* OperationViewModel – модель операции;
* ReportItemViewModel – модель для истории;
* UserViewModel – модель пользователя системы.

Далее в проекте идут сервисы:

* BillService – сервис, позволяющий получать информацию о счетах, а также создавать и удалять их;
* ExportService – сервис для экспорта диаграмм и списков счетов в EXCEL и PDF;
* FinanceProjectService – сервис для добавления финансовых проектов, а также их удаления и исзменения их свойств (права пользователю на редактирывание, изменение счетов);
* OperationService – сервис для добавления, удаления операций. Также сервис возвращает информацию об операции;
* PlanService – самый большой срвис. Он содержит функции рассчета состояния счетов к определенному моменту времени, используя сохраненные в базе данных финансовые операции;
* ReportService – сервис, возвращающий информацию о состоянии счетов за определенный отрезок времени;
* UserService – сервис для добавдения пользователей в систему. Также он участвует в процессе авторизации, предоставляя информацию о пользователях системы.

Следующим компонентом сервера является библиотека Finance.DataAccess. Эта библиотека отвечает за связь сервера с базой данных, и занимается передачей данных и БД в слой сервисов.

Структура библиотеки Finance.DataAccess такова:

* Entities;
* Interfaces;
* Repositories;
* Context.cs.

Entities – каталог сущностей системы. Все сушности соответствуют сущностям в базе данных. В данном каталоге находятся следующте сущности:

* Bill – модель счета;
* BillGroup – модель группы счетов;
* FinanceProject – модель финансового проекта;
* Operation – модель операции;
* ReportItem – модель для истории;
* User – модель пользователя.

Далее идет каталог с интерфейсами. Как и в случе с сервисами интерфейсы описывают репозитории.

Repositories – каталог с репозиториями для системы. Репозиторий – это класс, содержаший определенные для сущности методы, которые совершают операции с базой данных. В данной библиотеке несколько репозиториев:

* Repository – базовый дженерик репозиторий, содержащий методы добавления, удаления и поиска по Id для всех сущностей системы. Все остальные репозитории унаследованы от данного.
* BillRepository – репозиторий, содержащий методы для работы с базой данных с сущностью счетов;
* FinanceProjectRepository – репозиторий, содержащий методы для работы с базой данных с сущностью финансовых проектов системы;
* OperationRepository – репозиторий, содержащий методы для работы с базой данных с сущностью операций со счетами;
* ReportRepository – репозиторий, содержащий методы для работы с базой данных с сущностью отчетов из истории состояния счетов;
* UserRepository – репозиторий, содержащий методы для работы с базой данных с сущностью пользователя системы.

Последним файлом в данной библиотеке является Context. Это класс контекста базы данных для EntityFramework.

Последняя библиотека – Finance.Common. Это особенная библиотека так как на нее ссылаются все остальные проескты. Она содержит общие элементы для всей системы, такие как кастомная ошибка сервера FinanceException, которая описана в одноименном классе, находящемся в этой библиотеке. Так как ошибки могут появляться в любой из частей проекта – появилась необходимость в Common проекте.

Последним важным компонентом всей системы является Windows-служба. Данная служба работает напрямую с базой данных. Ежедневно она выполняет финансовые операции, которым пришел срок выпониться тем самым обновляя состояния счетов в вистеме. Также по окончанию всех рассчетов производится сохранение состояния счета для последующего построения отчета о состоянии счетов сервером. Данный сервис пользуется библиотекой Finance.DataAccess сервера для доступа к базе данных, и имеет один класс Update с методами обновления состояния счетов, и класс Log для сохранения состояния счета.

5.2 Описание пользовательского интерфейса и логики работы системы

Так как целю работы было создать универсальный плвнировщик бюджета, который может подстроиться под любые запросы пользователя я хочу показать пример именно такого использование системы в этом описании.

Для начала работы в системе необходимо зарегистрироваться. При посещении пользоватедем веб-сайта перед ним открывается страница авторизации, изображенная на рисунке 5.1.

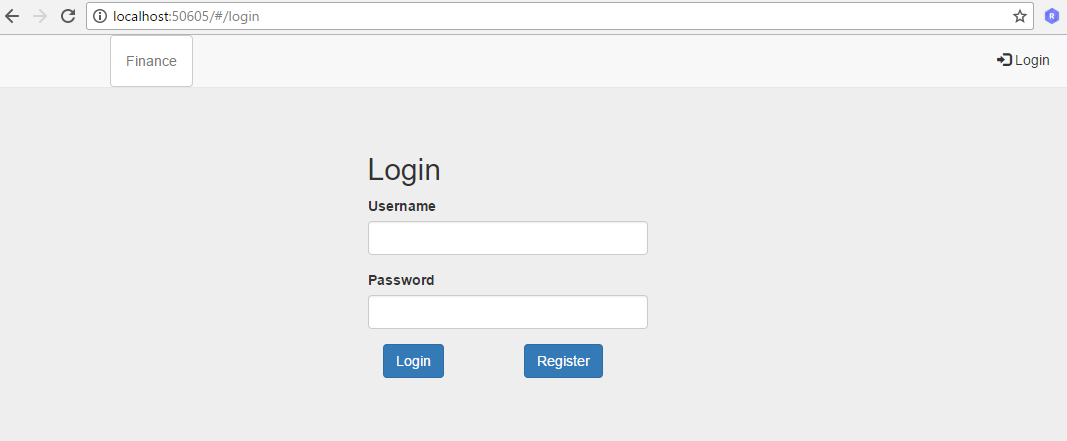


Рисунок 5.1 – Страница авторизации

На странице можно увидеть навигационное меню вверху окна, содержащее дле кнопки. Кнопка с надписью Finance ведет на домашнюю страницу, но неавторизованного пользователя она перенесет на страницу авторизации. Также присутствует кнопка с надписью Login, которая в любом случае приведет пользователя на страницу авторизации в системе.

В центре окна присутствуют два поля для ввода логина и пароля пользователя. На всез полях ввода присутствует валидация поэтому при вводе некорректных данных, или если пользователь оставит поле пустым – окно примет вид, показанный на рисунке 5.2.

Если данные пройдут валидацию, то на сервер пойдет запрос автризации и при положительном исходе пользователь авторизуется и перенесется на другую страницу. Иначе появится сообщение о том что данные были введены не верно.

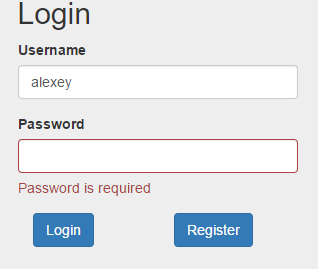


Рисунок 5.2 – Ракция сайта на ввод некорректных данных

Если у пользователя еще нет учетной записи в системе, то ему прийдется пройти на страницу регистрации, нажав на кнопку с надписью Regiter. Скриншот втраницы регистрации изображен на рисунке 5.3. На данной странице также присутствует валидация, к примеру пользователь не сможет ввести пароль длиной менее шести символов, либо он не сможет зарегистрироваться если при повторном вводе пароля он допустит ошибку.

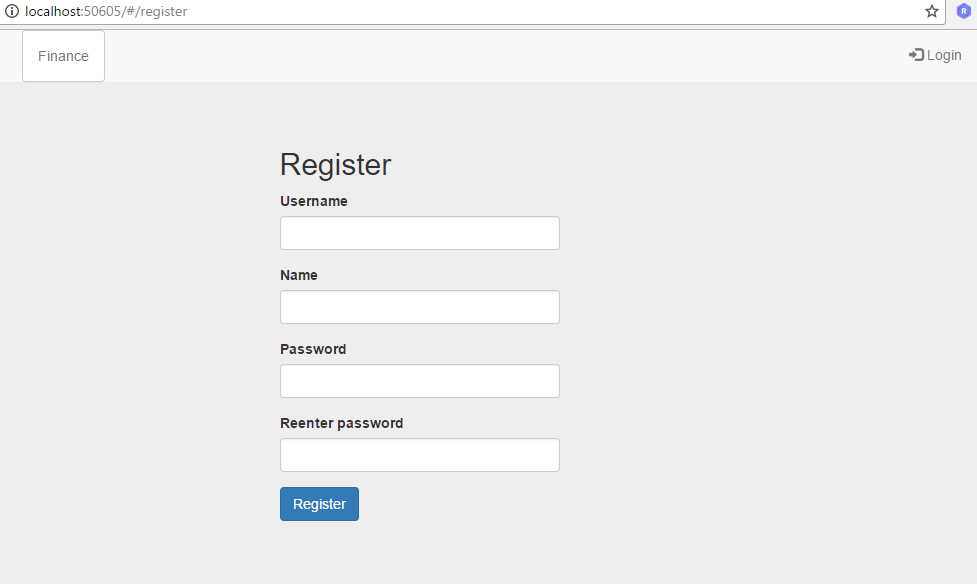


Рисунок 5.3 – Страница регистрации

Навигационное меню на этой странице находится в таком же состоянии как и ранее, но в центре окна появилось больше полей ввода. Поля выделены для ввода логина пользователя, его имени, личного пароля, а также повторение пароля во избежяния ошибки при первом вводе. Если все данные пройдут валидацию, они передадутся на сервер и для этого поьзователя создастся учетная запись. Затем пользователь автоматически авторизуется и перенесется на одну из двух страниц (если пользователь просто войдет в систему, используя всои логин и пароль произойдет то же самое). Если у пользователя есть доступный для реадактирывания финансовый проек – то он перенесется на домашнюю страницу, иначе он увидит перед собой страницу создания нового финансового проекта, показанную на рисунке 5.4.

ВЫВОДЫ

В ходе прохождения практики была спроектирована программная система, которая должна стать удобной и простой системой планирования и расчёта финансов на небольших предприятиях.

В начале работы был проведен анализ предметной области, а также анализ систем-аналогов, которые уже присутствуют на рынке, и успешно реализуют свою продукцию.

Также были определены требования к программному продукту и к потенциальным пользователям.

На этапе проектирования было выполнено концептуальное моделирование предметной области и определена архитектура проекта. Также была спроектирована и разработана модель базы данных и определены технологии, используемые при разработке программного продукта.

Следующим этапом – будет этап разработки, на котором будет реализована данная система в виде сервера и веб-сайта, используя средства и технологии, определенные на этапе проектирования системы.

ПЕРЕЧЕНЬ ССЫЛОК

1. Вільна енциклопедія Википедия [Електронний ресурс] / Wikipedia, the free encyclopedia - Forefront TMG 2010: - Режим доступу: https://en.wikipedia.org/. - Загл. с екрана.
2. Money Care - Bills monitor [Електронний ресурс] Money Care - Bills monitor - Режим доступу: https://itunes.apple.com/ru/app/money-care-bills-monitor/id420669776?mt=8. - Загл. с екрана.
3. Oracle Financial Analyzer [Електронний ресурс] / Oracle Financial Analyzer: - Режим доступу: http://www.oracle.com/technetwork/documentation/ofa-097121.html. - Загл. с екрана.
4. Буч, Г. Язык UML. Руководство пользователя [Текст] / Г. [Буч](http://padabum.com/search.php?author=%D0%91%D1%83%D1%87%20%D0%93.)  – М.: ДМК Пресс, 2-е издание, 2006. – 248 с.
5. Грекул, В. И. Проектирование информационных систем[Текст] / Н. Л. Коровкина – Изд-во: ИНТУИТ, 2005. – 240с.
6. Гарсия-Молина, Г. Системы баз данных. Полный курс. [Текст]: Пер. с англ./ Г.Гарсия-Молина, Дж. Ульман, Дж. Уидом. – Изд-во: Вильямс, 2003. – 1088 с.
7. Кренке, Г. Теорія й практика побудови баз даних [Текст] / Г. Кренке. - Спб.: Питер, 2001. - 858с.
8. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C# [Текст]/ Дж. Рихтер. – Изд-во: Питер, 2013. – 896 с.
9. Троелсен Э. C# и платформа .NET. Библиотека программиста [Текст]/ Э. Троелсен. – Изд-во: Питер, 2005. – 796 с.
10. Фримен, А. ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов [Текст]/ Адам Фримен, 5-е издание. – Изд-во: Вильямс, 2014. – 736 с.
11. Мартин, Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста [Текст]/ Р. Мартин. – Изд-во: Вильямс, 2015. – 464 с.
12. [Metanit.сom](http://metanit.com/index.php): Сайт о программировании, про создание сайтов и IT-технологии [Електронний ресурс] / metanit.com, 2012-2014. Режим доступу: http://metanit.com/index.php. - Загл. с екрана.
13. Котляров, В.П. Основы тестирования программного обеспечения [Текст]/ В. Котляров. – Изд-во: Бином, 2009. – 288 с.