Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

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

# Пояснительная записка

к дипломному проекту

на тему

**Программное средство**

**учёта коммунальных услуг на предприятии**

БГУИР ДП 1-40 01 01 01 016 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | И.А. Симакович |
| Руководитель |  | Е.Г. Мелких |
| Консультанты: |  |  |
| *от кафедры ПОИТ* |  |  |
| *по экономической части* |  | Е.Е. Марченкова |
|  |  |  |
| Нормоконтролер |  | Ю.Ю. Колотыгина К.Э. Рудак |
|  |  |  |
| Рецензент |  | И.П. Иванов |

Минск 2024

**Р Е Ф Е Р А Т**

ПРОГРАММНОЕ СРЕДСТВО УЧЁТА КОММУНАЛЬНЫХ УСЛУГ НА ПРЕДПРИЯТИИ: дипломный проект / И. А. Симакович. – Минск : БГУИР, 2024, – п.з. – 72 с., чертежей (плакатов) – 6 л. формата А1.

Актуальность дипломного проекта обусловлена необходимостью оптимизации процедур учёта потребления коммунальных услуг на предприятиях. В современных условиях экономики и управления, крупным и средним предприятиям особенно актуальны вопросы точности и оперативности учёта коммунальных расходов. Ручной сбор данных и их последующая обработка не только занимает значительное время инженерно-технических работников, но и сопряжён с риском ошибок, что влияет на финансовые показатели эффективности деятельности предприятия.

Объектом проектирования является программное средство для автоматизации системы учёта коммунальных услуг на предприятии. Данное программное средство позволяет автоматизировать процесс сбора, обработки и хранения информации о потреблении воды, электричества, тепла и других коммунальных ресурсов.

Цель работы состоит в разработке программного средства, предназначенного для снижения нагрузки на инженеров, путём автоматизации процесса сбора показаний счетчиков коммунальных услуг на различных объектах предприятия. Система должна обеспечить возможность удаленного сбора данных, их анализ и формирование отчётности по потребленным ресурсам.

В ходе работы были изучены и проанализированы существующие решения в области учёта коммунальных услуг, произведён обзор требований к подобного рода программным системам, определены ключевые функциональные требования к разрабатываемому программному средству. Результаты анализа легли в основу разработки архитектуры системы и её основных компонентов.

Программное средство реализовано на современном стеке технологий, что обеспечивает его масштабируемость, надежность и доступность для пользователей. Разработанная система позволяет снизить время на сбор и обработку данных о потреблении коммунальных услуг, минимизирует риски возникновения ошибок, предоставляет удобные инструменты для анализа и планирования расходов на коммунальные услуги.

Таким образом, реализация данного дипломного проекта способствует повышению экономической эффективности предприятия за счет оптимизации процессов учёта коммунального потребления, снижения нагрузки на инженерно-технический персонал и обеспечения точности планирования расходов на коммунальные ресурсы. Разработанное программное средство является актуальным и востребованным решением для решения задач современного управления и экономики предприятий различных отраслей.

Министерство образования Республики Беларусь

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Учреждение образования БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| Факультет | КС и С | | | Кафедра | | | ПОИТ | | | | | | | | | | |
| Специальность | 1-40 01 01 | | | Специализация | | | | | | 01 | | | | | | | |
| УТВЕРЖДАЮ | | | | | | | | | | | | | | | | | |
|  | | | | | | | | |  | | | | | Н.В.Лапицкая | | | |
| « | | | | | | | | |  | | » | |  | | | 20 | г. |
| ЗАДАНИЕ | | | | | | | | | | | | | | | | | |
| по дипломному проекту студента | | | | | | | | | | | | | | | | | |
| Симаковича Ивана Андреевича | | | | | | | | | | | | | | | | | |
| (фамилия, имя, отчество) | | | | | | | | | | | | | | | | | |
| 1. Тема проекта: | | Программное средство учёта коммунальных | | | | | | | | | | | | | | | |
| **услуг на предприятии** | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| утверждена приказом по университету от | | | | « | 03 | » | | апреля | | | | 2024 г. | | | № | 700-c | |
| 2 . Срок сдачи студентом законченной работы | | | | | 01 июня 2024 года | | | | | | | | | | | | |
| 3. Исходные данные к проекту | | | Тип операционной системы – ОС Windows 11; | | | | | | | | | | | | | | |
| Язык программирования – C#; | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| Назначение разработки: автоматизация процессов ввода и учета коммунальных услуг | | | | | | | | | | | | | | | | | |
| на предприятии | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| 4. Содержание пояснительной записки (перечень подлежащих разработке вопросов) | | | | | | | | | | | | | | | | | |
| Введение | | | | | | | | | | | | | | | | | |
| 1 Анализ прототипов и формирование | | | | | | | | | | | | | | | | | |
| 2 Разработка функциональных требований | | | | | | | | | | | | | | | | | |
| 3 Проектирование программного средства | | | | | | | | | | | | | | | | | |
| 4 Создание программного средства | | | | | | | | | | | | | | | | | |
| 5 Тестирование и проверка работоспособности | | | | | | | | | | | | | | | | | |
| 6 Руководство по установке и использованию | | | | | | | | | | | | | | | | | |
| 7 Технико-экономическое обоснование | | | | | | | | | | | | | | | | | |
| Заключение | | | | | | | | | | | | | | | | | |
| Список использованных источников | | | | | | | | | | | | | | | | | |
| Приложение А Текст программы | | | | | | | | | | | | | | | | | |
| Приложение Б Текст программы | | | | | | | | | | | | | | | | | |
| Приложение В Текст программы | | | | | | | | | | | | | | | | | |
| Приложение Г Текст программы | | | | | | | | | | | | | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 5. Перечень графического материала (с точным указанием наименования) и обозначения | | | |
| вида и типа материала) | |  | |
| Результат проверки работоспособности. Плакат - формат А1, лист 1. | | | |
| Схема работы модуля прогнозирования. Плакат - формат А1, лист 1. | | | |
| Схема база данных. Плакат -формат А1, лист 1. | | | |
| Диаграмма последовательности распечатки показаний. Схема алгоритма - формат | | | |
| А1, лист 1. | | | |
| Диаграмма классов. Схема программы - формат А1, лист 1. | | | |
| Схема базы данных. Схема данных - формат А1, лист 1. | | | |
| 6. Содержание задания по технико–экономическому обоснованию | | | |
| Расчет экономической эффективности от внедрения программного средства | | | |
| Задание выдал |  | | / Е.Г. Мелких / |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
| Задание выдал: |  | |  |

# КАЛЕНДАРНЫЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапов дипломного проекта (работы) | Объём этапа в % | Срок выполнения этапа | Примечание |
| Анализ прототипов и формирование | 15-20 | 01.02–17.02 |  |
| Разработка функциональных требований, | 15-20 | 18.02–04.05 |  |
| Проектирование программного средства, | 20 | 05.02-27.03 |  |
| Создание программного средства | 20-25 | 28.03–26.04 |  |
| Тестирование и проверка работоспособности | 10 | 27.04-04.05 |  |
| Оформление пояснительной записки |  |  |  |
| и графического материала | 15-20 | 05.04–31.05 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Дата выдачи задания | 01 февраля 2024 г. | | Руководитель | |  | | /Е.Г. Мелких / |
| Задание принял к исполнению | |  | | / И.А. Симакович/ | |

**СОДЕРЖАНИЕ**

Введение.……..........................................................................................................7

1 Анализ прототипов и формирование требований..............................................8

1.1 Исследование области разработки................................................................8

1.2 Назначение разработки...................................................................................8

1.3 Анализ аналогов и конкурентоспособности.................................................9

1.3.1 Анализ аналогов………………………....................................................9

1.3.2 Анализ конкурентоспособности…..…..................................................11

2 Анализ и разработка функциональных требований.........................................13

2.1 Постановка задачи…………........................................................................13

2.2 Входные данные............…............................................................................13

2.3 Выходные данные………….........................................................................14

2.4 Требование к параметрам технических и программных средств..............14

2.5 Обоснование выборов языка и среды разработки………………..............15

3 Проектирование программного средства.........................................................24

3.1 Описание компонентов программного средства........................................24

3.2 Описание архитектурных подходов............................................................24

3.3 Проектирование модели базы данных........................................................26

3.4 Разработка и описание основных алгоритмов…........................................31

4 Создание программного средства......................................................................35

4.1 Создание классов и компонентов................................................................35

5 Тестирование и проверка работоспособности..................................................39

5.1 Тестирование программного средства........................................................39

5.1.1 Регистрация и вход в систему................................................................39

5.1.2 Просмотр показаний счетчиков.............................................................40

5.1.3 Просмотр прогнозируемых показаний..................................................41

6 Руководство по установке и использованию....................................................42

6.1 Установка программного средства..............................................................42

7 Технико-экономическое обоснование……......................................................43

7.1 Описание функций и потенциальных пользователе..................................43

7.2 Расчет затрат на разработку.........................................................................43

7.3 Оценка результата от реализации веб-сервиса...........................................49

7.4 Выводы по технико-экономическому обоснованию.................................51

Заключение.............................................................................................................52

Список использованных источников...................................................................53

Приложение А Текст программного модуля агрегации данных показаний счетчиков.......................................................................................................................54

Приложение Б Текст программного модуля прогнозирования будущих данных..........................................................................................................................59

Приложение В Текст программного модуля шаблона репозиторий.................62

Приложение Г Текст программного модуля контекста базы данных................64

**ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ**

В настоящей пояснительной записке применяются следующие определения и сокращения.

CLR - общая языковая исполняющая среда исполнения (Common Language Runtime).

CTS - общая система типов (Common Type System).

CLI - Интерфейс командной строки (Command Line Interface).

CLS - Общая языковая спецификация, это подмножество общей системы типов Microsoft .NET (Common Language Specification, CLS).

API - программный интерфейс приложения.

MVC - это архитектурный паттерн (шаблон проектирования), используемый для разработки пользовательских интерфейсов. Модель-представление-контроллер(Model-View-Controller).

SOLID - это акроним, который обобщает пять основных принципов объектно-ориентированного программирования и дизайна. Эти принципы направлены на улучшение читаемости, поддерживаемости и расширяемости кода.

**ВВЕДЕНИЕ**

В условиях растущих требований к управлению предприятиями и законодательной поддержки рационального потребления ресурсов, становится приоритетом создание системы для точного и экономичного контроля за использованием коммунальных услуг. Этот подход, обусловленный как экономической выгодой, так и экологическими соображениями, акцентирует внимание на важности интеграции информационных технологий в процессы эффективного управления ресурсами.

Данный проект фокусируется на разработке программного средства, предназначенного для автоматизации учета коммунальных услуг на предприятии, что позволит не только минимизировать человеческий фактор и связанные с ним ошибки в процессе сбора данных, но и обеспечить оперативное получение информации для принятия управленческих решений.

Значительное внимание в работе уделено изучению существующих моделей учета коммунальных услуг, анализу их преимуществ и недостатков, а также выявлению критериев, которым должно соответствовать создаваемое программное решение.

В результате проекта предполагается получение надежной, масштабируемой и удобной в использовании системы, способной адаптироваться к особенностям любого предприятия и обеспечивать бесперебойную работу в условиях реального производственного процесса. Разработанное программное средство будет способствовать повышению экономической эффективности, оптимизации расходования ресурсов, как результат - укреплению экологической ответственности предприятия.

Ожидается, что имплементация представленного программного продукта станет значимым шагом к построению прозрачных и эффективных процессов управления ресурсами на предприятии, снижению эксплуатационных затрат и формированию устойчивой экономической модели бизнеса.

**1 АНАЛИЗ ПРОТОТИПОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ**

**1.1 Исследование области разработки**

Разработка потребует детального изучения области учета потребления коммунальных услуг и энергоресурсов. Необходимо изучить существующие аналоги и инструменты учета, а также потребности предприятий и индивидуальных пользователей в этой области. Важной частью исследования будет анализ требований к функциональности, производительности, надежности и безопасности программного средства.

Также важным аспектом является изучение используемых технологий (MVC .NET 6, SQL Server, ML.NET), их возможностей, преимуществ и ограничений. Это поможет учитывать все нюансы при проектировании и реализации программного средства.

**1.2 Назначение разработки**

Программное средство призвано обеспечить решение принципиально важной проблемы эффективного и точного учета энергоресурсов, обусловленной нехваткой универсальных инструментов на современном рынке.

Главным назначением нашего приложения является предоставление программного средства, которое может быть использовано в малом и среднем бизнесе для учета потребления коммунальных услуг. Основная цель разработки - повышение энергоэффективности и оптимизация расходов на коммунальные услуги и энергоресурсы.

В частности, "Программное средство учета коммунальных услуг на предприятии" способно обеспечить следующую функциональность:

1) регистрацию и хранение текущих показателей потребления коммунальных услуг, включая электричество, воду, газ и теплo;

2) изменение, редактирование и обновление данных о потреблении в удобной и интуитивно понятной форме;

3) построение прогнозов потребления коммунальных услуг на основе прошлых и актуальных данных;

4) возможность взаимодействия с программным средством в любое время и из любого места через веб-интерфейс;

5) предоставление детальной информации об энергопотреблении для более эффективного планирования и контроля;

Среди долгосрочных задач проекта - расширение функционала продукта за счет разработки и внедрения новых модулей и функций. Одним из таких потенциальных решений может быть модуль расчета экологического следа, который позволит пользователям оценить вклад их потребления энергоресурсов в загрязнение окружающей среды.

Таким образом, назначение нашей разработки заключается в том, чтобы предложить гибкое, точное и эффективное решение для организации учета потребления коммунальных услуг, удовлетворяющее потребности как отдельных индивидов, так и организаций различного масштаба.

**1.3** **Анализ аналогов и конкурентоспособности**

**1.3.1** Анализ аналогов

Первый конкурент в списке - ГИС ЖКХ. Это государственная информационная система жилищно-коммунального хозяйства (ГИС ЖКХ) — это современная и удобная платформа, призванная сделать сферу жилищно-коммунального хозяйства для граждан более понятной и прозрачной, обеспечить простой доступ к широкому перечню данных.

Это ресурс, где собираются данные о состоянии ЖКХ со всей страны и всех участников рынка: через ГИС ЖКХ россияне могут взаимодействовать с управляющими и ресурсо-снабжающими организациями, ТСЖ, органами власти различных уровней.

С помощью ГИС ЖКХ возможно:

1) обеспечить устойчивость работы;

2) посмотреть начисления за текущий и предыдущие периоды, а также внести плату за предоставленные жилищно-коммунальные услуги (только для российского региона);

3) вести и проверить показания приборов учета;

4) контролировать работы по дому, проводимые управляющими организациями, а также их стоимость (только для российского региона);

5) проверить наличие лицензии у управляющей организации (только для российского региона);

Главная проблема этого ресурса в том, что она направлена на оплату услуг и решение спорных вопросов с жилищно-эксплуатационным управлением. Кроме этого ресурс более ориентирован на российский рынок.

Второй конкурент "Битрикс24". Представляет собой комплексный инструмент управления предприятием, который включает в себя множество модулей для различных задач - от управления проектами и CRM до коммуникаций и колаборации.

Однако, стоит отметить, что "Битрикс24" изначально не предназначен для специфических задач учета коммунальных услуг, и его функционал в этой области может быть ограничен. Хотя его модули могут быть настроены на выполнение определенных процессов коллективного учета, контроля рабочих процессов и отчетности.

Например, модуль CRM может быть использован для управления контрагентами и договорами, а дополнительные полей может быть создано для хранения специфической информации, такой как показания счетчика.

Кроме того, с помощью модуля задач и проектов можно создавать регулярные задачи по сбору данных или подготовке отчетов.

Сервис также предлагает возможности интеграции с другими системами, включая сайты и ГИС, что может быть полезно для автоматического обмена данными и обеспечения своевременности и точности информации.

Как и у всех достаточно больших систем данный вариант обладает недостатками такими как: сложность в пользовании, не интуитивный интерфейс, множество функций, которые пользователю не нужны, так как в данном ресурсе хоть и можно хранить информацию об услугах ориентированность ресурса все-таки направлена на глобальное управление фирмой и бизнесом.

Третий конкурент "Директум" - это создан для автоматизации рабочих процессов и управления документами. Включает в себя широкий ряд функциональных возможностей, от учета документов до автоматизации рабочих процессов и сотрудничества.

Хотя "Директум" сам по себе не является полноценным решением, специально разработанным для учета коммунальных услуг, его гибкая структура и модульность могут быть полезны в данной сфере. Например, можно модифицировать или разработать специальные документы и рабочие процессы для учета показаний счетчиков, расчета и контроля оплаты коммунальных услуг.

Вот несколько способов, как "Директум" может быть использован для уче-та коммунальных услуг:

1) счёт-фактуры, договоры, акты и другие документы, связанные с учетом коммунальных услуг, могут быть созданы, сохранены и модифицированы. "Директum" также может автоматизировать процесс создания и одобрения этих документов;

2) позволяет оптимизировать и автоматизировать рабочие процессы, связанные с учетом коммунальных услуг, такие как сбор и обработка показаний счетчиков, расчет и оплата услуг;

3) может быть интегрирован с другими системами или приложениями, используемыми предприятием для учета коммунальных услуг;

Однако, стоит отметить, что для настройки "Директум" под потребности учета коммунальных услуг, может потребоваться поддержка IT-специалистов или консультантов.

Необходимо отметить, что точные детали зависят от специфики расчета коммунальных услуг на предприятии и требований к автоматизации, так что окончательная реализация может отличаться.

Как видно из первичного анализа "Директum" имеет ряд ориентировочно направленных плюсов для ведения учета и деятельности в коммунальной сфере, однако нужно заметить что не смотря на полезные части приложения, такие как автоматический сбор данных что бесспорно очень удобно не все предприятия а так же далеко не все точки учета имеют такие специальные счетчики, которым естественно нужно подключение к сети интернет либо подключение к мобильной связи через сим карту, что в свою очередь еще больше увеличивает финансовые траты на ведение такой деятельности. Однако стоит признать что дипломном программном средстве рассматривается введение такой модульной части которая должна поставляться отдельно в случае необходимости за отдельную плату.

Главным конкурентом возможно можно считать компанию МИРТЕК, «МИРТЕК» представляет собой мощного российского актёра в области разработки умных приборов учёта и программного обеспечения, особенно занимаясь автоматизацией учёта энергоресурсов.

Их продукты и услуги включают следующие ключевые элементы:

1) изготавливаемые компанией приборы учета энергии могут быть интегрированы в существующие интеллектуальные системы учета электроэнергии;

2) интеллектуальные системы учета позволяют измерять объёмы потреблённой электроэнергии дистанционно, контролировать расход энергии, осуществлять автоматический сбор, обработку и хранение данных, анализировать потребление энергии, рассчитать и контролировать утечки, мониторить состояние приборов учета. Таким образом, обеспечивая эффективность и точность в учете энергии;

3) с помощью собственных систем передачи данных, позволяет обеспечить непрерывную связь между приборами учета;

4) предоставить спектр услуг от проектирования до сдачи готового объекта в эксплуатацию. Это предлагает клиентам возможность "единого окна" и ответственность за все этапы проекта. Это также обеспечивает более высокий уровень координации и управления проектом;

В общем, продукты и услуги «МИРТЕК» представляют собой современное и инновационное решение для учёта энергоресурсов. Компания предлагает интегрированное решение, которое позволяет клиентам более эффективно и точно управлять потреблением энергии, что может помочь сократить затраты, повысить эффективность и уменьшить воздействие на окружающую среду.

Не смотря на то что "МИРТЕК" действительно предлагает комплексный пакет услуг и продукции для учёта энергоресурсов, но в отличие от нашего программного средства преимущественно ориентируется на промышленные предприятия на российском рынке.

**1.3.2** Анализ конкурентоспособности.

Наше программное средство может предложить ряд значительных преимуществ, которые делают его привлекательным выбором для бизнеса и организаций вне зависимости от их размера и основной сферы деятельности.

Мы стараемся сделать наше программное средство как можно универсальнее. Наше приложение спроектировано для обслуживания как бизнеса (включая розничные и коммерческие организации), так и государственных учреждений с распределенной структурой объектов, независимо от их специфики и отношения к промышленности. Это делает его универсальным инструментом для управления учетом любых ресурсов.

Наше приложение оснащено специализированным модулем прогнозирования, который позволяет пользователям прогнозировать будущие показания и тем самым позволяет делать более точное планирование и оптимизацию потребления ресурсов.

Помимо продвинутых функций, наше приложение предлагает значительное преимущество в виде более конкурентоспособного ценообразования. Обеспечивая высококачественные услуги по более доступной цене, мы делаем наше приложение доступным для более широкого круга клиентов.

Мы стремимся создать пользовательский интерфейс, который был бы простым и понятным для всех пользователей, независимо от их технического уровня. Мы уверены, что это облегчит процесс внедрения и обучения для наших клиентов.

Продукт предполагается сопровождать непрерывной поддержкой и исправлением найденных дефектов, чтобы обеспечить бесперебойную работу нашего приложения и ее эффективное использование для достижения целей учета энергоресурсов.

Наше приложение, таким образом, представляет собой универсальное, инновационное и конкурентоспособное решение для учёта энергоресурсов, которое эффективно отвечает на индивидуальные потребности любой организации или бизнеса.

Проведя анализ рынка конкурентов видно, что в той или иной степени данный функционал можно найти на рынке, однако в своем большинстве они ориентированы на ведение документооборота, а другая часть на непосредственное взаимодействие для оплаты услуг которую в текущее время можно выполнить многими путями при появлении ЕРИП в Беларуси и других подобных систем оплаты в других странах. Так же часть что предоставляет непосредственный функционал для хранения и оперирования данными соответственно дороже что представлено в технико-экономическом обосновании. Кроме всего этого практически во всех этих программах сотрудникам, непосредственно задействованным в работе с данными средствами, придется проходить обучение, о чем сразу предупреждается и на первых страницах сайта, а возможно и с наймом специальных тренеров в первое время работы. Ведь насколько показывает практика большой функционал порождает много кнопок и сложный интерфейс или доступ к нужным ресурсам что путает пользователей приложения. Ведь основное население не имеют такого опыта в использовании специфических программ у представленных конкурентов.

**2 АНАЛИЗ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

**2.1** **Постановка задачи**

Целью проекта является разработка "Программного средства учета коммунальных услуг на предприятии". Данное программное средство позволит пользователям отслеживать, управлять и оптимизировать расходы коммунальных услуг, а также предугадывать будущее потребление энергоресурсов на основе исторических данных.

Основные задачи включают:

1) анализ и исследование области учета коммунальных услуг и потребления энергоресурсов;

2) разработка функциональных и нефункциональных требований к программному средству;

3) проектирование и разработка программного средства с использованием выбранных технологий (MVC .NET 6, SQL Server, ML.NET);

4) тестирование и отладка созданного программного средства;

5) реализация аналитического модуля для прогнозирования потребления коммунальных услуг;

**2.2** **Входные данные**

Входные данные для программного средства учета коммунальных услуг на предприятии включают информацию о потреблении коммунальных услуг и энергоресурсов, пользователях и их настройках. Ниже мы постараемся рассмотреть их более детально.

Данные о потреблении коммунальных услуг: Это ключевые входные данные для нашего программного средства. Они могут включать, но не ограничиваются следующим:

1) показания счетчиков для различных видов коммунальных услуг (например, электричество, вода, газ, тепло и т.д.);

2) периодичность считывания показаний;

3) исторические данные о потреблении для анализа тенденций и прогнозирования;

4) специфическая информация о коммунальных услугах, такая как тарифы, стоимость и т.д.;

Данные пользователя. Эти данные помогают настроить программное средство в соответствии с потребностями и предпочтениями пользователей.

1) данные для авторизации (логин, пароль);

2) настройки персонального профиля пользователя (например, настройки уведомлений, предпочтения в отношении отображения данных и т.д.);

3) данные о типе пользователя (например, отдельный гражданин, предприятие, государственное учреждение и т.д.) для персонализации функционала и интерфейса;

Все входные данные должны быть предоставлены в цифровой форме и должны соответствовать определенным требованиям к формату и структуре для обеспечения корректной работы программного средства.

**2.3** **Выходные данные**

В результате использования "Программного средства учета коммунальных услуг на предприятии" генерируются выводы и отчеты, предназначенные для организации учета потребления коммунальных услуг и оптимизации использования энергоресурсов. Вот некоторые из ключевых выходных данных, которые предоставляет наше программное средство:

1) программное средство предоставляет данные о текущем потреблении энергоресурсов по различным видам услуг, включая электричество, воду, газ, тепло и прочее. Эти данные обрабатываются и представляются в понятном и наглядном формате;

2) детальная история потребления коммунальных услуг помогает отслеживать тенденции и паттерны потребления, а также оценивать эффективность принятых мер по сохранению энергоресурсов и их оптимизации;

3) на основе текущих и исторических данных программное средство может вычислять прогнозы будущего потребления энергоресурсов. Эти прогнозы могут использоваться для планирования бюджета, оптимизации потребления и минимизации затрат;

4) программное средство предоставляет возможность создания отчетов, собирающих важную статистику по потреблению коммунальных услуг. Отчеты, которые могут генерироваться периодически или по запросу, выводят данные в удобном формате для последующего анализа и использования в принятии решений;

5) в зависимости от настроек пользователя, программное средство может автоматически отправлять уведомления и напоминание о важных событиях, связанных с потреблением коммунальных услуг;

Все выходные данные предоставляются в удобных для пользователя форматах и можно легко экспортировать для дальнейшего использования или анализа.

**2.4** **Требование к параметрам технических и программных средств**

Для успешной работы "Программного средства учета коммунальных услуг на предприятии" требуются определенные технические и программные ресурсы.

Технические средства:

1) компьютер или любое мобильное устройство с доступом к интернету для доступа к веб-версии приложения;

2) надежный сервер с достаточным объемом памяти и оптимальной производительностью для обеспечения хранения и обработки всех пользовательских данных;

Программные средства:

1) операционная система должна быть совместима с различными операционными системами (Windows, MacOS, Linux для ПК, iOS, Android для мобильных устройств) для обеспечения доступа к приложению для как можно большего числа пользователей;

2) для доступа к веб-версии приложения потребуется наличие одного из современных браузеров (Google Chrome, Firefox, Safari, Edge и т.д.) ;

3) наличие системы управления базой данных для организации хранения, обработки и обмена пользовательской информацией;

4) качественное интернет-соединение для постоянного доступа к приложению и обновлению данных в режиме реального времени;

5) программное обеспечение для обеспечения безопасности и конфиденциальности пользовательских данных;

Вспомогательное программное обеспечение:

1) почтовый клиент для отправки уведомлений пользователям по электронной почте;

2) система обратной связи для улучшения качества и эффективности приложения путем получения отзывов и предложений от пользователей;

С учетом этих требований, наше Приложение для учета коммунальных услуг на предприятии сможет работать устойчиво и безотказно, обеспечивая пользователям комфортное и эффективное пользование всеми его функциями.

**2.5** **Обоснование выбора языка и среды разработки**

Ниже перечислены некоторые факторы, повлиявшие на выбор технологий:

1) разрабатываемое программное средство должно работать на операционной системе Windows 10 и более новых версиях системы;

2) среди различных платформ разработки имеющийся программист лучше всего знаком с разработкой на платформе Microsoft .NET;

3) дальнейшей поддержкой проекта, возможно, будут заниматься разработчики, не принимавшие участие в выпуске первой версии;

4) имеющийся разработчик имеет опыт работы с объекто-ориентированными и с функциональными языками программирования;

Основываясь на опыте работы имеющихся программистов разрабатывать ПС целесообразно на платформе Microsoft .NET. Теоретическая часть о .Net и много другое взято из книги Рихтера [9, CLR via C#, Рихтер Джеффри] Приняв во внимание необходимость обеспечения доступности дальнейшей поддержки программного средства, возможно, другой командой программистов, целесообразно не использовать малоизвестные и сложные языки программирования. С учетом этого фактора выбор языков программирования сужается до четырех официально поддерживаемых Microsoft и имеющих изначальную поддержку в Visual Studio 2022: Visual C++/CLI, C#, Visual Basic .NET и F#. Необходимость использования низкоуровневых возможностей Visual C++/CLI в разрабатываемом программном средстве отсутствует, следовательно, данный язык можно исключить из списка кандидатов. Visual Basic .NET уступает по удобству использования двум другим кандидатам из нашего списка. Оставшиеся два языка программирования C# и F# являются первостепенным, элегантными, мультипарадигменными языками программирования для платформы Microsoft .NET. Так как С# наиболее распространен для решение задач подобной нашей, а именно веб клиент-серверных приложениях, то выбор остановился на нем. Таким образом, с учетом вышеперечисленных факторов, целесообразно остановить выбор на следующих технологиях:

1) операционная система Windows 10;

2) платформа разработки Microsoft .NET;

3) язык программирования C#;

4) сервер базы данных SQL Server;

Для реализации поставленной задачи нет необходимости в использовании каких-либо прикладных библиотек с дополнительной лицензией, достаточно использовать стандартные библиотеки указанной платформы разработкиы. Поддержка платформой Microsoft .NET различных языков программирования позволяет использовать язык, который наиболее просто и «красиво» позволяет решить возникающую задачу. Разрабатываемое программное обеспечение в некоторой степени использует данное преимущество платформы. Язык C# больше подходит для создания высокоуровнего программного средства (иерархия классов и интерфейсов, организация пространств имен и публичного программного интерфейса). Далее проводится характеристика используемых технологий и языка программирования более подробно.

Программная платформа Microsoft .NET

Программная платформа Microsoft .NET является одной из реализаций стандарта ECMA-335 и является современным инструментом с здания клиентских и серверных приложений для операционной системы Windows и не только. Первая общедоступная версия .NET Framework вышла в феврале 2002 года. С тех пор платформа активно эволюционировала и на данный момент было выпущено уже достаточно много версии данного продукта. На данный момент номер последней версии .NET 9. Платформа Microsoft .NET была призвана решить некоторые наболевшие проблемы, скопившиеся на момент её выхода, в средствах разработки приложений под Windows. Ниже перечислены некоторые из них:

1) сложность создания надежных приложений;

2) сложность развертывания и управления версиями приложений и библиотек;

3) сложность создания переносимого ПС;

4) проблемы с безопасным исполнением непроверенного кода;

5) великое множество различных технологий и языков программирования, которые не совместимы между собой;

6) проблемы с безопасным исполнением непроверенного кода;

7) отсутствие единой целевой платформы для создателей компиляторов;

Многие из этих проблем были решены. Далее более подробно рассматривается внутреннее устройство Microsoft .NET.

Основными составляющими компонентами Microsoft .NET являются общая языковая исполняющая среда CLR и стандартная библиотека классов. CLR представляет из себя виртуальную машину и набор сервисов обслуживающих исполнение программ, написанных для Microsoft .NET. Ниже приводится перечень задач, возлагаемых на CLR

1) загрузка и исполнение управляемого кода;

2) управление памятью при размещении объектов;

3) изоляция памяти приложений;

4) проверка безопасности кода;

5) преобразование промежуточного языка в машинный код;

6) доступ к расширенной информации о типах - метаданным;

7) обработка исключений, включая межъязыковые исключения;

8) взаимодействие между управляемым и неуправляемым кодом (в том числе и COM-объектами);

9) поддержка сервисов для разработки (профилирование, отладка и

т. д.).

Программы, написанные для Microsoft .NET представляют из себя набор типов, взаимодействующих между собой. Microsoft .NET имеет общую систему типов (Common Type System, CTS). Данная спецификация описывает определения и поведение типов, создаваемых для Microsoft .NET. В частности, в данной спецификации описаны возможные члены типов, механизмы сокрытия реализации, правила наследования, типы-значения и ссылочные типы, особенности параметрического полиморфизма и другие возможности, предоставляемые CLI. Общая языковая спецификация (Common Language Specification, CLS) — подмножество общей системы типов. Это набор конструкций и ограничений, которые являются руководством для создателей библиотек и компиляторов в среде .NET Framework. Библиотеки, построенные в соответствии с CLS, могут быть использованы из любого языка программирования, поддерживающего CLS. Языки, соответствующие CLS (к их числу относятся языки C#, Visual Basic .NET, Visual C++/CLI), могут интегрироваться друг с другом. CLS — это основа межъязыкового взаимодействия в рамках платформы Microsoft .NET .

Некоторые из возможностей, предоставляемых Microsoft .NET: верификация кода, расширенная информация о типах во время исполнения, сборка мусора, безопасность типов, — невозможны без наличия подробных метаданных о типах из которых состоит исполняемая программа. Подробные метаданные о типах генерируются компиляторами и сохраняются в результирующих сборках. Сборка — это логическая группировка одного или нескольких управляемых модулей или файлов ресурсов, является минимальной единицей с точки зрения повторного использования, безопасности и управлениями версиям.

Одной из особенностей Microsoft .NET, обеспечивающей переносимость программ без необходимости повторной компиляции, является представление исполняемого кода приложений на общем промежуточном языке (Common Intermediate Language, CIL). Промежуточный язык является бестиповым, стековым, объекто-ориентированным ассемблером. Данный язык очень удобен в качестве целевого языка для создателей компиляторов и средств автоматической проверки кода для платформы Microsoft .NET, также язык довольно удобен для чтения людьми. Наличие промежуточного языка и необходимость создания производительных программ подразумевают наличие преобразования промежуточного кода в машинный код во время исполнения программы. Одним из компонентов общей языковой исполняющей среды, выполняющим данное преобразование, является компилятор времени исполнения транслирующий промежуточный язык в машинные инструкции, специфические для архитектуры компьютера на котором исполняется программа.

Ручное управление памятью всегда являлось очень кропотливой и подверженной ошибкам работой. Ошибки в управлении памятью являются одними из наиболее сложных в устранении типами программных ошибок, также эти ошибки обычно приводят к непредсказуемому поведению программы, поэтому в Microsoft .NET управление памятью происходит автоматически. Автоматическое управление памятью является механизмом поддержания иллюзии бесконечности памяти. Когда объект данных перестает быть нужным, занятая под него память автоматически освобождается и используется для построения новых объектов данных. Имеются различные методы реализации такого автоматического распределения памяти. В Microsoft .NET для автоматического управления памятью используется механизм сборки. Существуют различные алгоритмы сборки мусора со своими достоинствами и недостатками. В Microsoft .NET используется алгоритм пометок в сочетании с различными оптимизациями, такими как, например, разбиение всех объектов по поколениям и использование различных куч для больших и малых объектов.

Ниже перечислены, без приведения подробностей, некоторые важные функции исполняемые общей языковой исполняющей средой:

1) обеспечение многопоточного исполнения программы;

2) поддержание модели памяти, принятой в CLR;

3) поддержка двоичной сериализации;

4) управление вводом и выводом;

5) структурная обработка исключений;

6) возможность размещения исполняющей среды внутри других процессов.

Как уже упоминалось выше, большую ценностью для Microsoft .NET представляет библиотека стандартных классов - соответствующая CLS-спецификации объектно-ориентированная библиотека классов, интерфейсов и системы типов (типов-значений), которые включаются в состав платформы Microsoft .NET. Эта библиотека обеспечивает доступ к функциональным возможностям системы и предназначена служить основой при разработке .NET-приложений, компонент, элементов управления.

Язык программирования C#

C# - объектно-ориентированный, типобезопасный язык программирования общего назначения. Язык создавался с целью повысить продуктивность программистов. Для достижения этой цели в языке гармонично сочетаются простота, выразительность и производительность промежуточного кода, получаемого после компиляции. Главным архитектором и идеологом языка с первой версии является Андрес Хейлсберг (создатель Turbo Pascal и архитектор Delphi). Язык C# является платформенно нейтральным, но создавался для хорошей работы с Microsoft .NET Этот язык сочетает простой синтаксис, похожий на синтаксис языков C++ и Java, и полную поддержку всех современных объектно-ориентированных концепций и подходов. В качестве ориентира при разработке языка было выбрано безопасное программирование, нацеленное на создание надежного и простого в сопровождении кода.

Язык имеет богатую поддержку парадигмы объекто-ориентированного программирования, включающую поддержку инкапсуляции, наследования и полиморфизма. Отличительными чертами C# с точки зрения объектно-ориентированной парадигмы являются:

Унифицированная система типов. В C# сущность, содержащая данные и методы их обработки, называется типом. В C# все типы, являются ли они пользовательскими типами, или примитивами, такими как число, производны от одного базового класса.

Классы и интерфейсы. В классической объекто-ориентированной парадигме существуют только классы. В C# дополнительно существуют и другие типы, например, интерфейсы. Интерфейс - это сущность напоминающая классы, но содержащая только определения членов. Конкретная реализация указанных членов интерфейса происходит в типах, реализующих данный интерфейс. В частности интерфейсы могут быть использованы при необходимости проведения множественного наследования (в отличие от языков C++ и Eiffel, C# не поддерживает множественное наследование классов).

Свойства, методы и события. В чистой объекто-ориентированной парадигме все функции являются методами. В C# методы являются лишь одной из возможных разновидностей членов типа, в C# типы также могут содержать свойства, события и другие члены. Свойство — это такая разновидность функций, которая инкапсулирует часть состояния объекта. Событие — это разновидность функций, которые реагируют на изменение состояния объекта

В большинстве случаев C# обеспечивает безопасность типов в том смысле, что компилятор контролирует чтобы взаимодействие с экземпляром типа происходило согласно контракту, который он определяют. Например, компилятор C# не скомпилирует код который обращается со строками, как если бы они были целыми числами. Говоря более точно, C# поддерживает статическую типизацию, в том смысле что большинство ошибок типов обнаруживаются на стадии компиляции. За соблюдение более строгих правил безопасности типов следит исполняющая среда. Статическая типизация позволяет избавиться от широкого круга ошибок, возникающих из-за ошибок типов. Она делает написание и изменение программ более предсказуемыми и надежными, кроме того, статическая типизация позволяет существовать таким средствам как автоматическое дополнение кода и его предсказуемый статический анализ. Еще одним аспектом типизации в C# является её строгость. Строгая типизация означает, что правила типизации в языке очень «сильные». Например, язык не позволяет совершать вызов метода, принимающего целые числа, передавая в него вещественное число. Такие требования спасают от некоторых ошибок.

C# полагается на автоматическое управление памятью со стороны исполняющей среды, предоставляя совсем немного средств для управления жизненным циклом объектов. Несмотря на это, в языке все же присутствует поддержка работы с указателями. Данная возможность предусмотрена для случаев, когда критически важна производительность приложения или необходимо обеспечить взаимодействие с неуправляемым кодом.

С момента создания язык C# и платформа .Net потерпела ряд существенных улучшений и теперь стало возможно писать программы на языке C# не только для операционных систем Microsoft, но и ряда других ОС. Существуют инструменты создания приложений на C# для серверных и мобильных платформ, например: iOS, Android, Linux и других.

Создатели языка C# не являются противниками привнесения в язык новых идей и возможностей, в отличии от создателей одного из конкурирующих языков. Каждая новая версия компилятора языка привносит различные полезные возможности, которые отчаются требованиям индустрии. Далее приводится краткий обзор развития языка.

Первая версия C# была похожа по своим возможностям на Java 1.4, несколько их расширяя: так, в C# имелись свойства (выглядящие в коде как поля объекта, но на деле вызывающие при обращении к ним методы класса), индексаторы (подобные свойствам, но принимающие параметр как индекс массива), события, делегаты, циклы foreach, структуры, передаваемые по значению, автоматическое преобразование встроенных типов в объекты при необходимости, атрибуты, встроенные средства взаимодействия с неуправляемым кодом (DLL, COM) и прочее.

Версия Microsoft .NET 2.0 привнесла много новых возможностей в сравнении с предыдущей версией, что отразилось и на языках под эту платформу. Проект спецификации C# 2.0 впервые был опубликован Microsoft в октябре 2003 года; в 2004 году выходили бета-версии (проект с кодовым названием Whidbey), C# 2.0 окончательно вышел 7 ноября 2005 года вместе с Visual Studio 2005 и Microsoft .NET 2.0. Ниже перечислены новые возможности в версии 2.0:

1) частичные типы (разделение реализации класса более чем на один файл);

2) обобщённые, или параметризованные типы. В отличие от шаблонов C++, они поддерживают некоторые дополнительные возможности и работают на уровне виртуальной машины. Вместе с тем, параметрами обобщённого типа не могут быть выражения, они не могут быть полностью или частично специализированы, не поддерживают шаблонных параметров по умолчанию, от шаблонного параметра нельзя наследоваться;

3) новая форма итератора, позволяющая создавать сопрограммы с помощью ключевого слова yield, подобно Python и Ruby;

4) Анонимные методы, обеспечивающие функциональность замыкания;

5) Оператор ??: return obj1 ?? obj2; означает (в нотации C# 1.0) return obj1!=null ? obj1 : obj2;

5) Обнуляемые (nullable) типы-значения (обозначаемые вопросительным знаком, например, int? i = null;), представляющие собой те же самые типы-значения, способные принимать также значение null. Такие типы позволяют улучшить взаимодействие с базами данных через язык SQL;

6) Поддержка 64-разрядных вычислений позволяет увеличить адресное пространство и использовать 64-разрядные примитивные типы данных;

Третья версия языка имела одно большое нововведение - Language Integrated Query (LINQ), для реализации которого в языке дополнительно появилось множество дополнительных возможностей. Ниже приведены некоторые из них:

1) ключевые слова select, from, where, позволяющие делать запросы из SQL, XML, коллекций и т. п.

2) инициализацию объекта вместе с его свойствами: Customer c = new Customer(); c.Name = "James"; c.Age=30; можно записать как Customer c = new Customer { Name = "James", Age = 30 };

3) лямбда-выражения: listOfFoo.Where(delegate(Foo x) { return x.size > 10; }); теперь можно записать как listOfFoo.Where(x => x.size > 10);

4) деревья выражений - лямбда-выражения теперь могут быть представлены в виде структуры данных, доступной для обхода во время выполнения, тем самым позволяя транслировать строго типизированные C#- выражения в другие домены (например, выражения SQL).

5) вывод типов локальной переменной: var x = "hello"; вместо string

x = "hello";

6) безымянные типы: var x = new { Name = "James"};

7) методы-расширения — добавление метода в существующий класс с помощью ключевого слова this при первом параметре статической функции.

8) автоматические свойства: компилятор сгенерирует закрытое поле и соответствующие аксессор и мутатор для кода вида public string Name { get; private set; } C# 3.0 совместим с C# 2.0 по генерируемому MSIL-коду, улучшения в языке — чисто синтаксические и реализуются на этапе компиляции.

Visual Basic .NET 10.0 и C# 4.0 были выпущены в апреле 2010 года, одновременно с выпуском Visual Studio 2010. Новые возможности в версии 4.0:

1) возможность использования позднего связывания;

2) именованные и опциональные параметры;

3) новые возможности COM interop;

4) ковариантность и контрвариантность интерфейсов и делегатов;

5) контракты в коде (Code Contracts);

В C# 5.0 было немного нововведений, но они носят большую практическую ценность. В новой версии появилась упрощенная поддержка выполнения асинхронных функций с помощью двух новых слов — async и await. Ключевым словом async помечаются методы и лямбда-выражения, которые внутри содержат ожидание выполнения асинхронных операций с помощью оператора await, который отвечает за преобразования кода метода во время компиляции.

Это основные, но не все нововведения в язык и платформу. И язык и платформа активно развиваются и в данное время усилиями компании так и сообщество ведь весь код системы выложен в открытый доступ и в развитии может поучаствовать любой желающий.

База данных на основе SQL Server.

При разработке программного средства учета коммунальных услуг на предприятии стоит уделить особое внимание выбору технологии базы данных, поскольку это является основой для хранения, обработки и обеспечения доступа к данным. В данном случае выбор пал на SQL Server от компании Microsoft, и вот почему:

1) надежность и производительность: SQL Server известен своей надежностью и высокой производительностью. Он надежно сохраняет данные, обеспечивая быстрый доступ и обработку запросов, что критически важно для нашего программного средства учета коммунальных услуг.

2) SQL Server позволяет легко масштабировать решение в соответствии с растущими требованиями и размерами данных. Это означает, что по мере роста количества пользователей и объема данных, программное средство будет способна поддерживать высокую производительность и обеспечивать необходимый сервис.

3) возможность интеграции с другими продуктами Microsoft. SQL Server хорошо интегрируется с другими продуктами Microsoft, такими как Visual Studio и Azure, что облегчает процесс разработки и развертывания.

4) SQL Server поддерживает транзакционную модель работы с данными, что обеспечивает целостность и согласованность данных в много пользовательской среде.

5) безопасность SQL Server включает в себя множество продвинутых функций, таких как шифрование на уровне столбца, проверка подлинности SQL и т.д., которые помогут защитить данные пользователей.

6) сервер способен эффективно управлять большими объемами данных, что может быть полезно, учитывая потенциальный рост числа пользователей и объемов собираемых данных в нашем программном средстве.

Таким образом, учитывая надежность, производительность, масштабируемость, безопасность и другие преимущества SQL Server, это становится превосходным выбором для нашего программного средства учета коммунальных услуг.

Модуль прогнозирования ML.NET:

ML.NET является библиотекой платформы .NET, предлагающей мощные функции для интеграции машинного обучения в приложения. Она разработана Microsoft и включает в себя множество классов, интерфейсов и методов для работы с моделями машинного обучения. Вот некоторые причины, почему ML.NET идеально подходит для нашего «Программного средства учета коммунальных услуг на предприятии»:

1) легко вписывается в уже существующую структуру приложений на платформе .NET, что упрощает процесс разработки и упрощает интеграцию машинного обучения в приложение.

2) содержит множество алгоритмов прогнозирования для различных типов данных и сценариев, что позволит нашему приложению предсказывать будущее потребление коммунальных услуг на поразительно точном уровне.

3) предоставляет полный набор инструментов для обработки и предварительной обработки данных, что важно для подготовки данных о потреблении коммунальных услуг для обучения модели.

4) API ML.NET был разработан с акцентом на простоту использования, но также позволяет разработчикам использовать более сложные сценарии машинного обучения и тонко настраивать модели и процессы обучения.

5) предоставляет функцию AutoML, которая автоматически отбирает наиболее подходящую модель машинного обучения для задачи, что может упростить и ускорить процесс создания модели.

Исходя из этих причин, ML.NET с высокой степенью уверенности можем признать наиболее подходящей технологией для использования в контексте «Программного средства учета коммунальных услуг на предприятии».

**3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**3.1** **Описание компонентов программного средства**

Для реализации нашего программного средства на основе MVC .NET 6 с встроенной аутентификацией и использованием Razor Pages для UI, программное средство можно разбить на следующие компоненты: модель - предствление - контроллер

1. бизнес логика;

2. представления;

3. контроллер;

4. сервис аутентификации;

5. база данных;

6. сервис прогнозирования;

Бизнес логика описывает основные объекты программного средства (например, Пользователь, Потребление коммунальных услуг), их свойства и взаимодействия. Модель также включает в себя классы для работы с базой данных (такие как Entity Framework контекст), которые служат для сохранения и извлечения данных.

Представления служат для отображения информации пользователю. Они могут быть построены с использованием Razor Pages, которые обеспечивают удобный и гибкий синтаксис для создания динамических HTML страниц.

Контроллеры в MVC .NET 6 отвечают за обработку запросов от пользователя, обработку пользовательских данных и взаимодействие с моделью для выполнения нужной бизнес-логики. Они также отвечают за отправку данных в представления для отображения пользователям.

Встроенная в .NET 6 система аутентификации позволяет управлять пользователями, их регистрацией, аутентификацией и авторизацией. Это важный компонент, который обеспечивает безопасность и персонализацию пользовательского опыта.

База данных служит для хранения всей информации, включая данные об аутентификации пользователей и информацию о потреблении коммунальных услуг.

Данный компонент, использующий ML.NET, отвечает за аналитическую составляющую нашего приложения. Он используется для анализа данных и создания прогнозов по потреблению коммунальных услуг.

Эти компоненты взаимодействуют между собой для обеспечения всех функций и требований "Программного средства учета коммунальных услуг на предприятии".

**3.2** **Описание архитектурных подходов**

Статическая страница на HTML не умеет реагировать на действия пользователя. Для двухстороннего взаимодействия нужны динамические веб-страницы. MVC — ключ к пониманию разработки динамических веб-приложений, поэтому разработчику нужно знать эту модель.

MVC расшифровывается как «модель-представление-контроллер» (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

Компоненты MVC:

1) модель - этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете приложение-напоминалку, код компонента «модель» будет определять список задач и отдельные задачи.

2) представление - этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.

3) контроллер - этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения.

Архитектура MVC как никакая другая лучше подходит к нашему приложению с реализацией веб клиент серверного подхода. Она уже интегрирована в .Net непосредственно на уровне фреймоворка. Общая схема работы изображена на схеме 3.2.1.

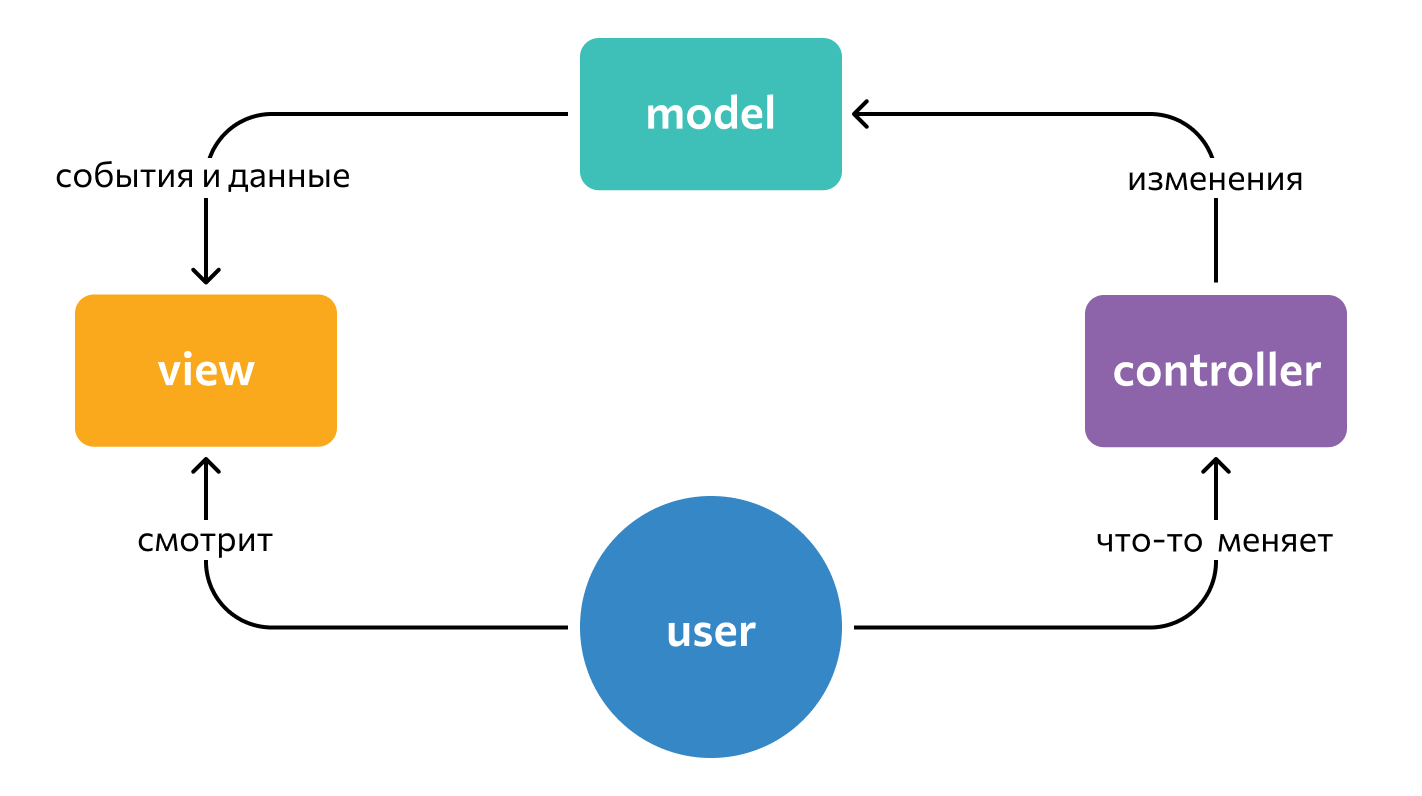


Рисунок 3.2.1 Принцип работы MVC шаблона

**3.3 Проектирование модели базы данных**

Разрабатываемое решение для программного средства учета коммунальных услуг на предприятии включает в себя базу данных, в которой хранится информация о пользователе, об объектах учета, устройствах учета и данных потребления.

Таблицы являются ключевыми объектами реляционной базы данных, храня в себе всю информацию, которая необходима для работы приложения. Информация в таблице организована в столбцы и строки.

Для реализации спроектированной базы данных была выбрана СУБД SQL Server. Данное средство управления базами данных от Microsoft обеспечивает высокую производительность и надежность, а также широкие возможности по взаимодействию с приложениями, разработанными на платформе .NET.

Ниже представлены созданные для реализации проекта таблицы и их составляющие:

Таблица: AspNetRoleClaims. Хранит утверждения, связанные с ролями пользователей.

Поля:

1) Id (INT): Уникальный идентификатор утверждения роли;

2) RoleId (NVARCHAR(450)): Идентификатор роли, к которой относится утверждение;

3) ClaimType (NVARCHAR(MAX)): Тип утверждения роли;

4) ClaimValue (NVARCHAR(MAX)): Значение утверждения роли;

Таблица: AspNetRoles. Описание таблицы. Управление ролями пользователей.

Поля:

1) Id (NVARCHAR(450)): Уникальный идентификатор роли;

2) Name (NVARCHAR(256)): Название роли;

3) NormalizedName (NVARCHAR(256)): Нормализованное название роли;

4) ConcurrencyStamp (NVARCHAR(MAX)): Метка для контроля параллелизма;

Таблица: AspNetUserClaims. Описание таблицы. Хранение утверждений, связанных напрямую с пользователями.

Поля:

1) Id (INT): Уникальный идентификатор утверждения пользователя;

2) UserId (NVARCHAR(450)): Идентификатор пользователя, к которому относится утверждение;

3) ClaimType (NVARCHAR(MAX)): Тип утверждения пользователя;

4) ClaimValue (NVARCHAR(MAX)): Значение утверждения пользователя;

Таблица: AspNetUserLogins. Описание таблицы. Управление внешними учетными записями пользователей.

Поля:

1) LoginProvider (NVARCHAR(128)): Идентификатор провайдера входа;

2) ProviderKey (NVARCHAR(128)): Ключ провайдера;

3) ProviderDisplayName (NVARCHAR(MAX)): Отображаемое имя провайдера;

4) UserId (NVARCHAR(450)): Идентификатор пользователя;

Таблица: AspNetUserRoles. Описание таблицы. Связь между пользователями и их ролями.

Поля:

1) UserId (NVARCHAR(450)): Идентификатор пользователя;

2) RoleId (NVARCHAR(450)): Идентификатор роли;

Таблица: AspNetUsers. Описание таблицы. Управление учетными записями пользователей.

Поля:

1) Id (NVARCHAR(450)): Уникальный идентификатор пользователя;

2) UserName (NVARCHAR(256)): Имя пользователя;

3) NormalizedUserName (NVARCHAR(256)): Нормализованное имя пользователя;

4) Email (NVARCHAR(256)): Электронная почта пользователя;

5) NormalizedEmail (NVARCHAR(256)): Нормализованная электронная почта пользователя;

6) EmailConfirmed (BIT): Флаг подтверждения электронной почты;

7) PasswordHash (NVARCHAR(MAX)): Хэш пароля;

8) SecurityStamp (NVARCHAR(MAX)): Стамп безопасности;

9) ConcurrencyStamp (NVARCHAR(MAX)): Метка для контроля параллелизма;

10) PhoneNumber (NVARCHAR(MAX)): Номер телефона пользователя;

11) PhoneNumberConfirmed (BIT): Флаг подтверждения номера телефона;

12) TwoFactorEnabled (BIT): Флаг включения двухфакторной аутентификации;

13) LockoutEnd (DATETIMEOFFSET(7)): Время окончания блокировки;

14) LockoutEnabled (BIT): Флаг разрешения блокировки;

15) AccessFailedCount (INT): Количество неудачных попыток доступа;

Таблица: AspNetUserTokens. Описание таблицы. Хранение токенов для внешних поставщиков аутентификации.

Поля:

1) UserId (NVARCHAR(450)): Идентификатор пользователя;

2) LoginProvider (NVARCHAR(128)): Идентификатор провайдера входа;

3) Name (NVARCHAR(128)): Название токена;

4) Value (NVARCHAR(MAX)): Значение токена;

Таблица: ClientProfile. Описание таблицы. Хранит дополнительные сведения о клиентах.

Поля:

1) Id (INT): Уникальный идентификатор профиля клиента;

2) Address (NVARCHAR(200)): Адрес клиента;

3) UserId (NVARCHAR(450)): Идентификатор пользователя, ассоциированного с профилем;

Таблица: Town. Описание таблицы. Содержит информацию о городах.

Поля:

1) Id (INT): Уникальный идентификатор города;

2) Name (NVARCHAR(200)): Название города;

Таблица: PlaceSection. Храненит информации о территориях на которых распологается объект.

Поля:

1) Id (INT): Уникальный идентификатор секции места;

2) PlaceName (NVARCHAR(200)): Название помещения или раздела;

3) AdresSection (NVARCHAR(200)): Адрес секции;

4) AreaSection (FLOAT): Площадь секции;

5) Kadastr (NVARCHAR(200)): Кадастровый номер;

6) DataResh (DATETIME): Дата решения;

7) TypeArenda (NVARCHAR(200)): Тип аренды;

8) Certificate (NVARCHAR(200)): Номер сертификата;

9) DateArenda (DATETIME): Дата начала аренды;

Таблица: Organization. Описание таблицы. Содержит информацию об организациях.

Поля:

1) Id (INT): Уникальный идентификатор организации;

2) Name (NVARCHAR(200)): Название организации;

3) Telefon (INT): Телефон организации;

4) Email (NVARCHAR(200)): Электронная почта организации;

Таблица: Place. Описание таблицы. Управление местами, такими как здания, земельные участки и др.

Поля:

1) Id (INT): Уникальный идентификатор места;

2) Name (NVARCHAR(200)): Название места;

3) ModelPlace (NVARCHAR(200)): Модель или тип места;

4) Arenda (DATETIME): Срок аренды;

5) TownId (INT): Идентификатор города, ссылка на таблицу Town;

6) Address (NVARCHAR(200)): Адрес места;

7) Area (FLOAT): Площадь места;

8) PlaceSectionId (INT): Идентификатор секции места, ссылка на таблицу PlaceSection;

Таблица: Contract. Описание таблицы. Хранение контрактов с организациями.

Поля:

1) Id (INT): Уникальный идентификатор контракта;

2) OrganizationId (INT): Идентификатор организации, ссылка на таблицу Organization;

3) PlaceId (INT): Идентификатор места, ссылка на таблицу Place;

4) Dogovor (INT): Номер договора;

5) Limit (INT): Лимит использования или другое ограничение, определенное контрактом;

Таблица: Equipment. Описание таблицы. Сведения об оборудовании.

Поля:

1) Id (INT): Уникальный идентификатор оборудования;

2) ModelEq (NVARCHAR(200)): Модель оборудования;

3) Description (NVARCHAR(MAX)): Описание оборудования;

4) PowerEq (INT): Мощность оборудования;

5) PlaceId (INT): Идентификатор места установки, ссылка на таблицу Place;

Таблица: MeasureType. Описание таблицы. Категории или типы измерений, производимых оборудованием.

Поля:

1) Id (INT): Уникальный идентификатор типа измерения;

2) Name (NVARCHAR(200)): Название типа измерения;

Таблица: Schetchik. Описание таблицы. Информация о счетчиках и измерительных приборах.

Поля:

1) Id (INT): Уникальный идентификатор счетчика;

2) NomerSchetchika (NVARCHAR(200)): Номер счетчика;

3) ModelSchetchika (NVARCHAR(200)): Модель счетчика;

4) TexUchet (BIT): Флаг технического учета;

5) TwoTarif (BIT): Флаг наличия двойного тарификации;

6) Poverka (DATETIME): Дата последней поверки;

7) Poteri (INT): Значение потерь в процессе измерения;

8) PlaceId (INT): Идентификатор места установки, ссылка на таблицу Place;

9) MeasureTypeId (INT): Идентификатор типа измерения, ссылка на таблицу MeasureType;

Таблица: Indication. Описание таблицы. Записи показаний измерительных устройств.

Поля:

1) Id (INT): Уникальный идентификатор показания;

2) Month (DATETIME): Месяц и год показания;

3) Value (DECIMAL(8,2)): Значение показания;

4) Tarif1 (FLOAT): Тариф за используемый ресурс;

5) Archive (BIT): Флаг архивации записи;

6) SchetchikId (INT): Идентификатор счетчика, ссылка на таблицу Schetchik;

Таблица: UserPlace. Описание таблицы. Связь пользователей с местами учета.

Поля:

1) UserId (NVARCHAR(450)): Идентификатор пользователя, ссылка на таблицу AspNetUsers;

2) PlaceId (INT): Идентификатор места, ссылка на таблицу Place;

Эти таблицы формируют основу схемы базы данных, охватывая пользователей, роли, места, оборудование, измерения и другие сферы деятельности.

Нормализация базы данных выполняется для устранения избыточности данных, предотвращения аномалий доступа и обновления данных, а также для упрощения структуры базы данных. Процесс нормализации включает применение серии нормальных форм, каждая из которых представляет определенные требования к структуре данных. Рассмотрим процесс нормализации нашей базы данных на основе предыдущих описаний таблиц:

1) 1НФ (Первая нормальная форма). Все таблицы нашей базы данных соответствуют 1НФ, что означает, что каждое поле таблицы содержит атомарные значения, и каждая запись уникальна. В таблицах нет повторяющихся групп полей, что гарантирует атомарность данных.

2) 2НФ (Вторая нормальная форма). Таблицы удовлетворяют требованиям 2НФ, т.е. они находятся в 1НФ, и все неключевые поля полностью функционально зависят от первичного ключа. Это устраняет частичную функциональную зависимость, гарантируя, что каждое неключевое поле зависит от всего первичного ключа, а не его части.

3) 3НФ (Третья нормальная форма). Наша база данных соответствует требованиям 3НФ. Все таблицы находятся в 2НФ, и между неключевыми полями нет функциональных зависимостей. Это устраняет транзитивную зависимость, что предотвращает потенциальные аномалии обновления, вставки и удаления данных.

На основе новой структуры таблиц, отношения "один ко многим" были определены следующим образом:

AspNetUsers и AspNetRoles. Каждый пользователь может быть связан только с одной ролью, в то время как одна роль может быть присвоена множеству пользователей. Это классический пример отношения "многие ко многим", которое было реализовано через промежуточную таблицу AspNetUserRoles для нормализации и избежания избыточности данных.

AspNetUsers и Place. Один пользователь может отвечать за множество мест (например, зданий или земельных участков), но каждое место связано только с одним пользователем.

Place и Equipment. Одно место может содержать множество единиц оборудования, но каждая единица оборудования принадлежит только одному месту.

Equipment и Measurement. Она устройство может производить множество измерений, в то время как каждое измерение принадлежит только одному оборудованию.

Эти отношения обеспечивают не только нормализованную структуру данных, но и упрощают обработку и запросы данных, минимизируя избыточность и повышая эффективность хранения информации в базе данных. Общая схема базы данных:

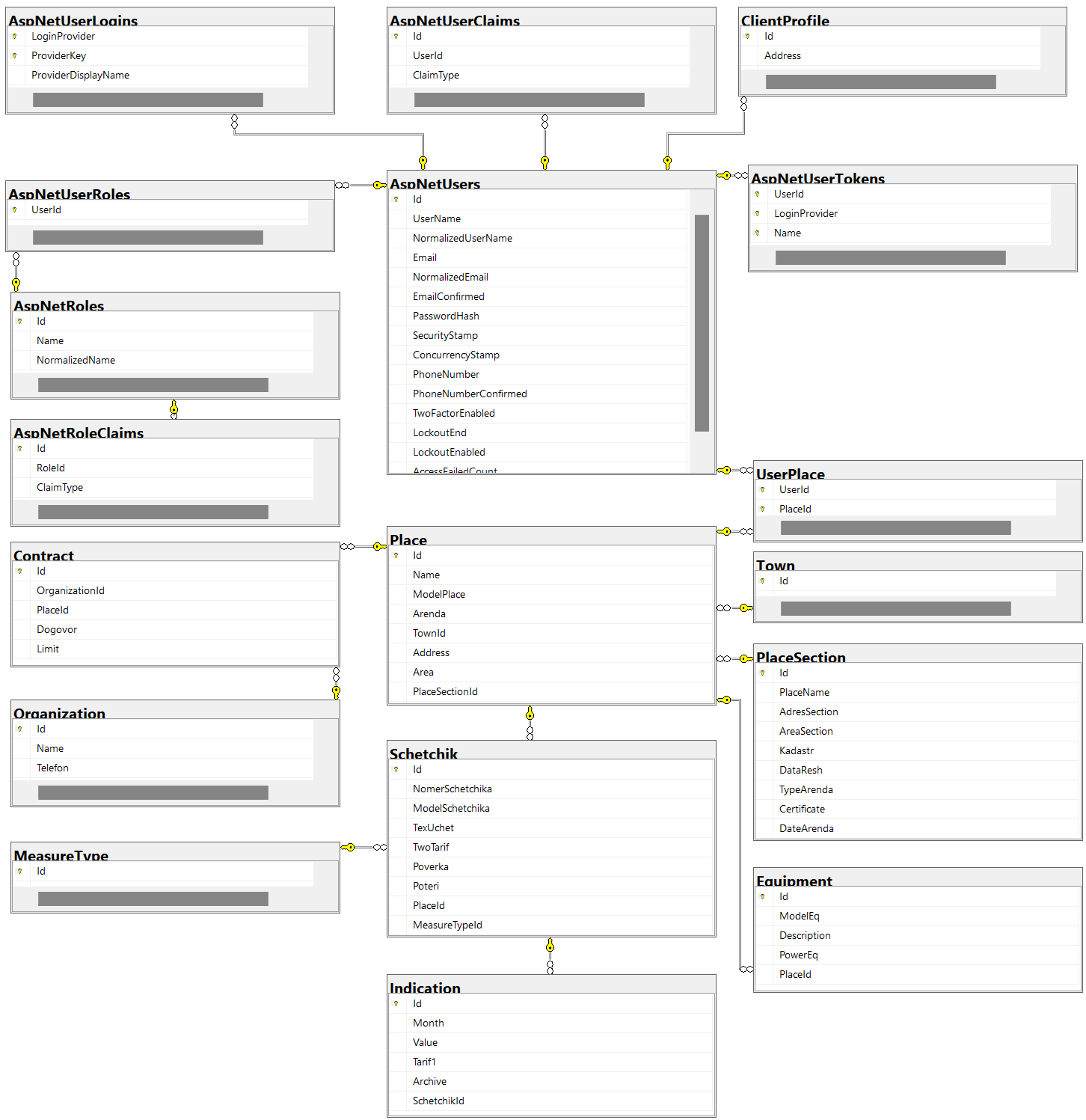


Рисунок 3.3.1 Схема базы данных

**3.4 Разработка и описание основных алгоритмов**

Pабочий цикл приложения MVC .NET 6 представляет собой последовательность шагов, выполняемых при выполнении каждого запроса от клиента до сервера и обратно. Вот основные шаги в этом процессе:

1) Запрос: Все начинается, когда браузер пользователя делает запрос к серверу, например, когда пользователь нажимает гиперссылку или отправляет форму на веб-странице;

2) Маршрутизация: MVC-приложение получает запрос и определяет маршрут запроса. Маршрутизация обрабатывается в middleware-компонентах ASP.NET и служит для преобразования URL-адреса запроса в определенный метод действия в контроллере;

3) Инициализация контроллера: После определения подходящего метода действия, MVC-среда создает новый экземпляр соответствующего контроллера и запускает метод действия;

4) Выполнение действия контроллера: Метод действия обрабатывает запрос. Это может включать в себя извлечение данных из базы данных, выполняя вычисления и т.д. В конце этого шага, метод действия вернет представление;

5) Поиск и визуализация представления: Представление является HTML-страницей, которая отправляется обратно в браузер. MVC-среда находит подходящее представление и компилирует его в HTML;

6) Формирование ответа: HTML генерируется и возвращается обратно в браузер в форме HTTP-ответа;

7) Отправка ответа: ASP.NET отправляет HTTP-ответ обратно в браузер пользователя;

8) Закрытие и уничтожение контекста запроса. Запрос завершен, и все ресурсы, связанные с ним, освобождаются;

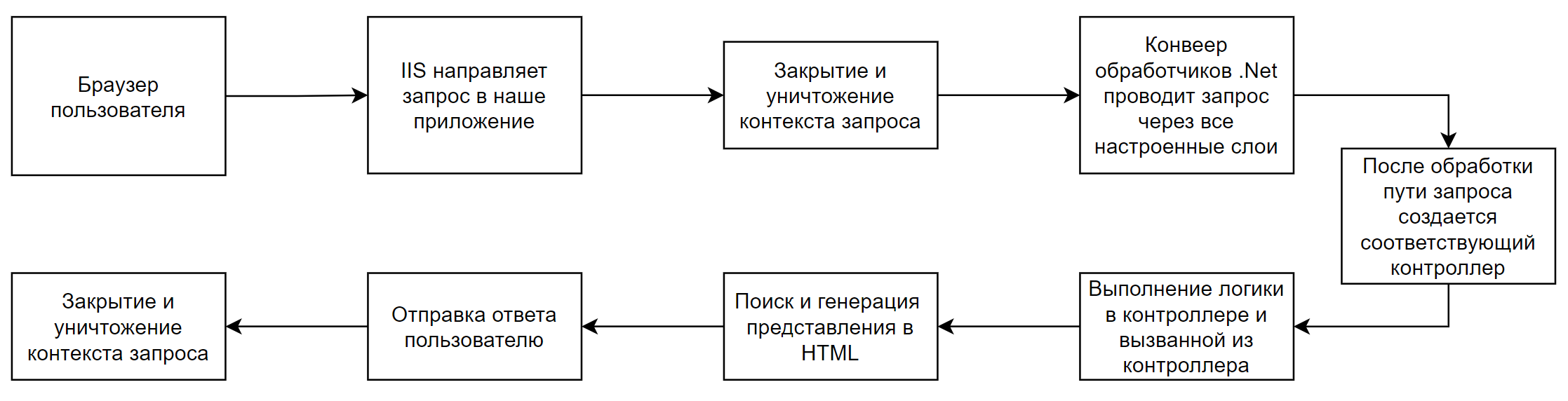


Рисунок 3.4.1 Рабочий цикл работы MVC

Алгоритм печати информации по электроэнергии за определенный период;

1) Пользователь открывает браузер и вводит URL домашней страницы приложения;

2) Сервер обрабатывает запрос и возвращает HTML, CSS и JavaScript домашней страницы;

3) Браузер рендерит страницу и отображает ее пользователю;

4) Пользователь кликает на ссылку или кнопку, ведущую на страницу с показаниями электроэнергии;

5) Браузер делает новый запрос на сервер для загрузки страницы показаний электроэнергии;

6) Сервер возвращает HTML, CSS и JavaScript страницы показания электроэнергии;

7) Браузер рендерит страницу и отображает ее пользователю;

8) Пользователь выбирает даты "с" и "по" для просмотра показаний и нажимает кнопку "Загрузить";

9) Браузер делает запрос на сервер с выбранными датами в качестве параметров;

10) Сервер обрабатывает запрос, извлекает соответствующие данные из базы данных и возвращает их на браузер;

11) Браузер обновляет страницу, показывая показания электроэнергии за выбранный период;

12) Пользователь нажимает кнопку "Печать";

13) Браузер открывает диалоговое окно печати, и пользователь указывает нужные настройки и нажимает "Печать";

14) При возникновении ошибки на любом из этих этапов, процесс прерывается, и модальное окно с сообщением об ошибке отображается на экране;

Диаграмма последовательности алгоритма печати данных о электроэнергии представлена на чертеже 1.

Алгоритм регистрации пользователя:

1) Пользователь переходит на страницу регистрации через интерфейс приложения (например, нажав на кнопку "Регистрация");

2) Пользователь вводит необходимые для регистрации данные в соответствующие поля формы (например, имя пользователя, адрес электронной почты, пароль);

3) Пользователь подтверждает введенные данные, нажав на кнопку "Регистрация" или "Отправить";

4) После нажатия кнопки в браузере пользователя формируется HTTP-запрос на сервер с введенными пользователем данными;

5) Сервер получает запрос и начинает его обработку;

6) Первым шагом сервер проверяет корректность введенных данных. Это может включать в себя проверку на наличие всех необходимых полей, длину и сложность пароля, уникальность адреса электронной почты и т.д;

7) Если данные не валидны, сервер формирует ответ с ошибкой и отправлят его обратно в браузер пользователя. Пользователю отображается сообщение об ошибке;

8) Если все данные валидны, сервер хэширует пароль пользователя для обеспечения безопасности. Хэш пароля, а не исходный пароль, будет сохранен в базе данных;

9) Сервер генерирует SQL-запрос на вставку нового пользователя в таблицу "Users" базы данных. В запрос включаются введенные пользователем данные и хэш пароля;

10) SQL-запрос выполняется, и данные нового пользователя сохраняются в базе данных;

11) После успешного сохранения данных сервер формирует успешный ответ и отправляет его в браузер пользователя;

12) Пользователю отображается сообщение об успешной регистрации. Пользователь теперь может войти с использованием своих учетных данных;

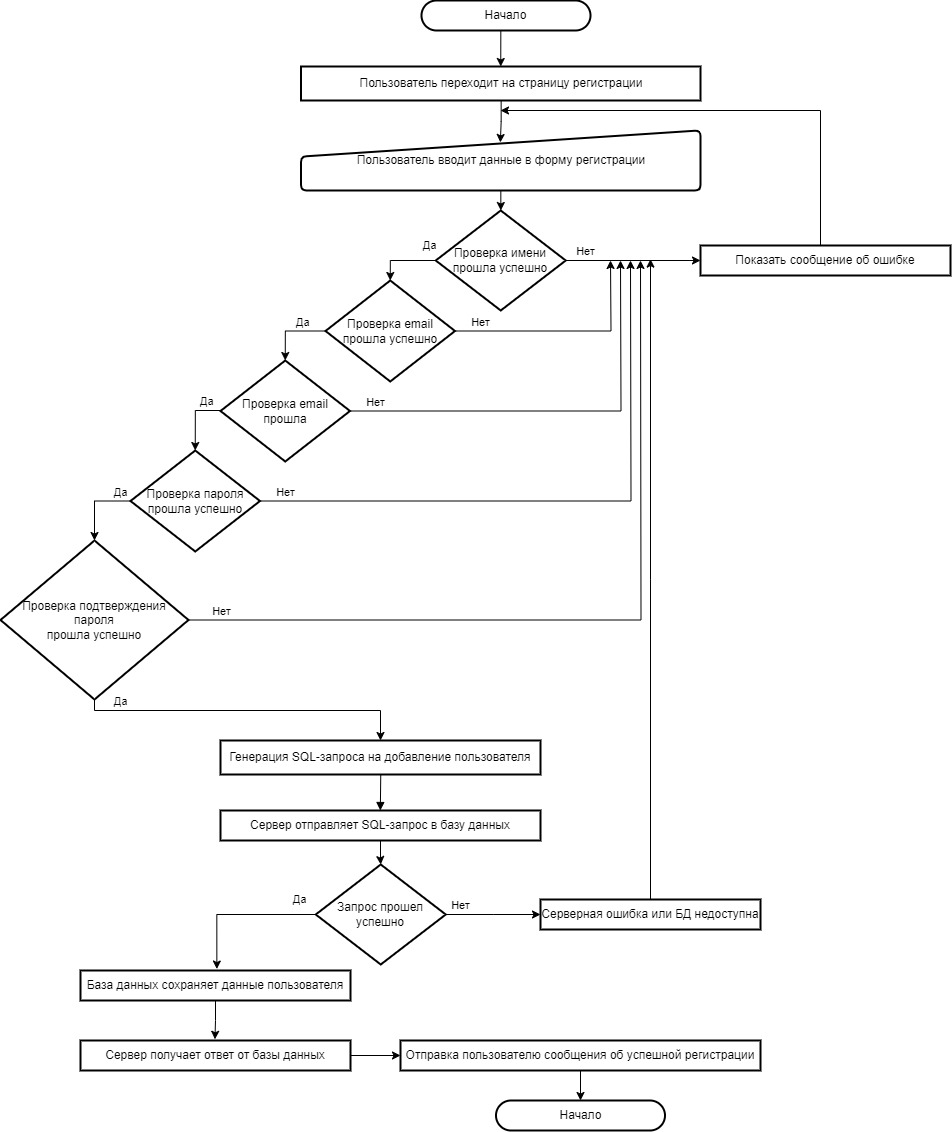


Рисунок 3.4.2 Диаграмма регистрации пользователя

**4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**4.1** **Создание классов и компонентов**

В этом разделе для проекта на базе .NET 6 MVC с использованием Entity Framework для соединения с SQL Server рассматривается процесс реализации ключевых аспектов приложения. Учитывая контекст особое внимание уделено разработке основных компонентов системы, включая создание классов, контроллеров и представлений.

Программное средство включает в себя серию классов, максимально отражающих структуру базы данных для упрощения взаимодействия с данными. К примеру, классы, такие как AspNetUser и Contract, напрямую отображают таблицы в базе данных, что позволяет эффективно управлять пользовательскими данными, контрактами и прочими аспектами операций через Entity Framework. Эта методика облегчает маппинг данных и упрощает выполнение CRUD операций.

Вот краткое описание основных классов:

1) AspNetRole.cs представляет роли пользователей в системе. Определяет типы ролей и их свойства для управления доступом пользователей к различным частям приложения.

2) AspNetUser.cs описывает пользователей системы, включая их учетные данные, информацию для контактов и статус. Используется для аутентификации и авторизации пользователей.

3) ClientProfile.cs содержит детализированную информацию о клиентах, такую как личные данные, контактная информация и история операций. Используется для управления данными клиентов.

4) Contract.cs отражает информацию о контрактах или соглашениях между организацией и клиентами, включая сроки и условия.

5) Equipment.cs представляет собой данные об оборудовании или устройствах, используемых в системе, включая спецификации и статус.

6) ErrorViewModel.cs класс, используемый для передачи информации об ошибках в представления, помогает управлять отображением ошибок пользователю.

7) Indication.cs содержит показания или измерения, собранные с оборудования или датчиков, для дальнейшей обработки и анализа.

8) KioskSection.cs описывает разделы или категории в пользовательском интерфейсе, например, для киосков самообслуживания или секций на веб-сайте.

9) MeasureType.cs определяет типы измерений или данных, которые могут быть собраны системой, например, температура, давление, и т.д.

10) Organization.cs содержит информацию об организациях, с которыми взаимодействует система, включая название, контактные данные и адрес.

11) Place.cs класс для работы с местоположениями, такими как адреса, места установки оборудования и т.д.

12) Schetchik.cs служит для учета счетчиков или измерительных устройств в системе.

13) Town.cs создан для управление списком городов, в которых оперирует система или доступны услуги.

14) UserKiosk.cs нужен для управление доступом пользователей к киоскам или специализированным разделам системы.Эти классы, наряду с вспомогательными классами, такими как ViewModel для передачи данных в представления и моделями для передачи данных между сервисами, контроллерами и другими частями программного средства, формируют основу программного продукта. ViewModel классы обеспечивают необходимую абстракцию для эффективной передачи и отображения данных, в то время как модели служат для обмена данными внутри приложения, упрощая взаимодействие между его компонентами и слоями.

После создания основных классов, важным этапом в разработке программного средства стало проектирование и реализация контроллеров, которые служат связующим звеном между пользовательским интерфейсом и логикой приложения. Для того что б понять как именно работает программное средство нужно понять как работают основные контроллеры.

HomeController контроллер управляет основными страницами приложения, включая домашнюю страницу, контактную информацию и общие сведения. Он обрабатывает запросы, связанные с навигацией по сайту, и предоставляет пользователю базовую информацию о системе.

IdentityController.cs отвечает за регистрацию новых пользователей, аутентификацию и управление профилями пользователей. IdentityController использует возможности ASP.NET Core Identity для обеспечения безопасности и управления ролями пользователей в приложении.

IndicationsController.cs специализированный контроллер, через который осуществляется ввод, просмотр и анализ показаний или измерений, полученных от оборудования. Он позволяет пользователю работать с индивидуальными показаниями и историческими данными за различные периоды.

MeasureTypesController.cs управляет типами измерений, которые может регистрировать система, например, температура, давление, и т.д. Этот контроллер позволяет добавлять, редактировать и удалять типы измерений, обеспечивая необходимую гибкость в работе с разнообразными данными оборудования.

PredictionController.cs реализует функциональность, связанную с анализом данных и созданием прогнозов на основе аккумулированных показаний. Этот контроллер использует алгоритмы машинного обучения и статистическую обработку данных для предсказания будущих тенденций.

TownsController.cs предоставляет возможности по управлению списком городов, в которых доступны услуги приложения. Включает функции для добавления новых городов, редактирования и удаления существующих записей.

Ключевые контроллеры взаимодействуют с моделями данных и представлениями для обработки пользовательских запросов и представления результатов. Разработка этих контроллеров требовала тщательного планирования и тестирования для обеспечения целостности и безопасности приложения. Реализация контроллеров сопровождалась созданием соответствующих маршрутов в приложении для обработки запросов и осуществления навигации.

Для реализации бизнес-логики и обработки данных были разработаны специализированные сервисы, такие как AIService и MeasurementService. Эти сервисы представляют из себя отдельные компоненты, которые можно использовать для выполнения конкретных задач, не перегружая контроллеры. Использование сервисов также способствует соблюдению принципов SOLID и упрощает тестирование приложения.

Представления формируют пользовательский интерфейс приложения. Для каждой функциональной возможности созданы соответствующие представления, позволяющие пользователям выполнять нужные действия, например Index.cshtml для вывода основной информации и Edit.cshtml для изменения данных. Благодаря модульной архитектуре ASP.NET MVC, представления могут использовать общие элементы, такие как \_Layout.cshtml, что обеспечивает единообразный вид всего приложения.

Дополнительное описание для упомянутых представлений ASP.NET MVC поможет подчеркнуть их роль и важность в создании дружелюбного и эффективного пользовательского интерфейса веб-приложения:

Основные представления:

1) Index.cshtml является стартовой страницей приложения, обычно отображает обзорную информацию или вводные данные, доступные пользователю сразу после входа в приложение.

2) PredictOne.cshtml специализированное представление, используемое для отображения результатов предиктивного анализа или прогнозирования, предоставляя пользователю глубокий анализ на основе собранных данных.

3) \_MeasureMenu.cshtml обычно представляет собой частичное представление, встраиваемое в другие страницы, и предлагает пользователю навигационное меню или опции, связанные с измерениями и аналитикой.

4) Error.cshtml предназначено для отображения информации об ошибках, предоставляя пользователям четкое понимание проблемы и возможные пути её решения.

Макеты и стили:

1) \_Layout.cshtml основной шаблон для всех страниц веб-приложения, определяющий общий макет, включая заголовки, подвалы и навигацию, чтобы обеспечить консистентность пользовательского интерфейса.

2) \_LoginPartial.cshtml частичное представление для отображения интерфейса входа или информации о текущем пользователе, обычно встроено в верхнюю часть макета.

3) \_SecondLayout.cshtml может быть использовано для определения альтернативного макета страниц для специальных случаев или разделов приложения, требующих отличающегося стиля или структуры.

4) \_ValidationScriptsPartial.cshtml: Частичное представление, содержащее скрипты валидации для форм, помогает обеспечить корректный ввод данных пользователем и улучшает взаимодействие с формами.

CRUD представления для управления данными:

Create.cshtml, Delete.cshtml, Details.cshtml, Edit.cshtml: Эти представления обрабатывают стандартные операции CRUD (создание, чтение, удаление, обновление) для каждой сущности в приложении. Предоставляют пользователю формы для создания новых записей (Create.cshtml), отображают подробную информацию об объекте (Details.cshtml), позволяют редактировать существующие данные (Edit.cshtml) или удалять их (Delete.cshtml).

Вместе эти элементы формируют основу пользовательского интерфейса веб-приложения, поддерживая как основной функционал, так и необходимые операции управления и взаимодействия с пользователем. Создание интуитивно понятных и функциональных представлений способствует повышению удобства работы с приложением и обеспечивает положительный пользовательский опыт.

Это все обеспечило не только модульность и удобство использования приложения, но и высокий уровень абстракции, способствующий повышению безопасности и надежности программного средства.

В разработке программного средства для обеспечения аутентификации и авторизации пользователей принято решение не создавать отдельный модуль безопасности с нуля. Вместо этого, используется встроенный механизм безопасности Identity для платформы .NET, настроенный на работу с индивидуальными учетными записями. Компонент предоставляет полноценную поддержку для работы с системой аутентификации и авторизации, включая управление пользователями, ролями, защиту паролей, подтверждение по электронной почте и многое другое. Это решение позволяет ускорить разработку за счет использования готовых инструментов и подходов, а также повысить безопасность приложения за счет применения передовых практик и алгоритмов шифрования. средства.

В ходе работы над проектом был произведен не только выбор необходимых технологий и языка программирования, но и детальная разработка архитектуры приложения, что включало создание базовых и специализированных классов, контроллеров для обработки запросов пользователей и сервисов для выполнения бизнес-логики. Особое внимание было уделено разработке удобного и функционального пользовательского интерфейса через систему представлений. Общий результат работы - функционально полноценное программное средство, готовое к использованию в соответствии с заданными требованиями.

**5 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ**

**5.1** **Тестирование программного средства**

В этом разделе представлены результаты работы ключевых функциональных возможностей программного средства, подтверждающие его работоспособность и соответствие заявленным требованиям. В процессе тестирования были получены скриншоты, иллюстрирующие ключевые этапы взаимодействия пользователя с системой.

**5.1.1** Регистрация и вход в систему

Регистрация и вход в систему являются начальными точками взаимодействия пользователя с веб-приложением. Процесс регистрации поддерживает создание новых учетных записей с последующей возможностью аутентификации и доступа к персонализированным функциям системы.

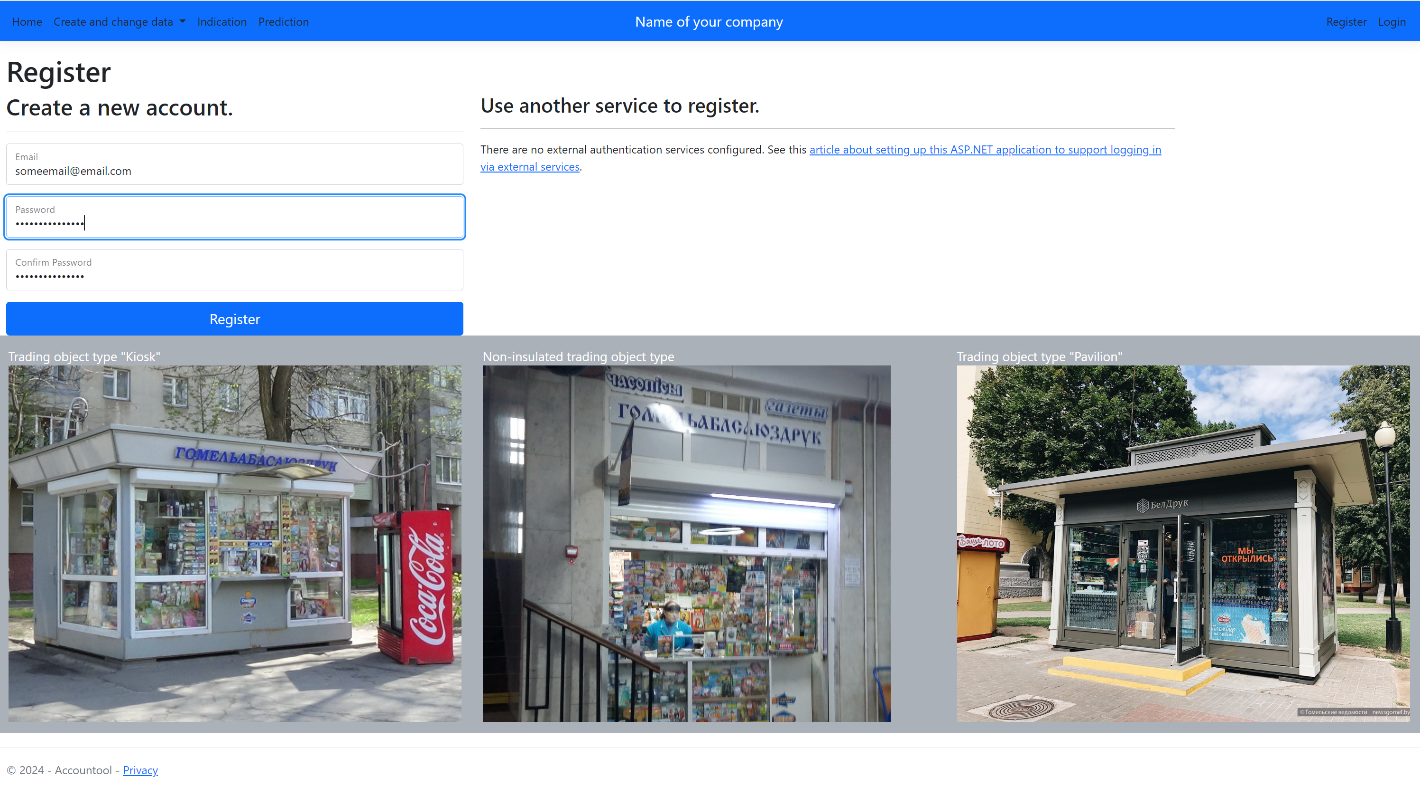


Рисунок 5.1.1.1 Регистрация пользователя

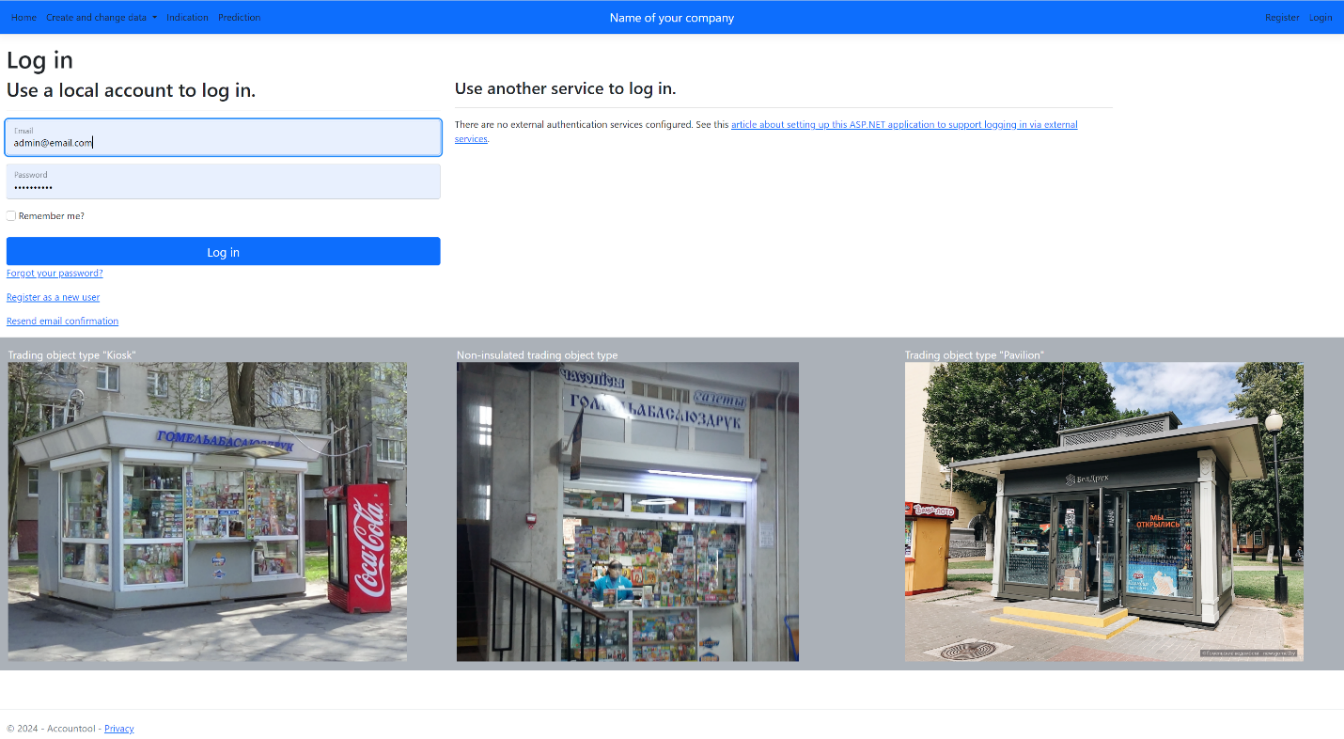


Рисунок 5.1.1.2 Вход в систему

**5.1.2** Просмотр показаний счетчиков

Для просмотра показаний счетчиков по заданным параметрам, пользователь должен перейти на страницу "Indications" через навигационное меню. Выбор типа показаний "Electricity" позволяет фильтровать данные в соответствии с интересующим типом энергии. Фильтрация данных осуществляется на основе местоположения, временного диапазона и города. Оставление полей фильтрации пустыми не ограничивает выборку, предоставляя пользователю максимально полный набор данных.

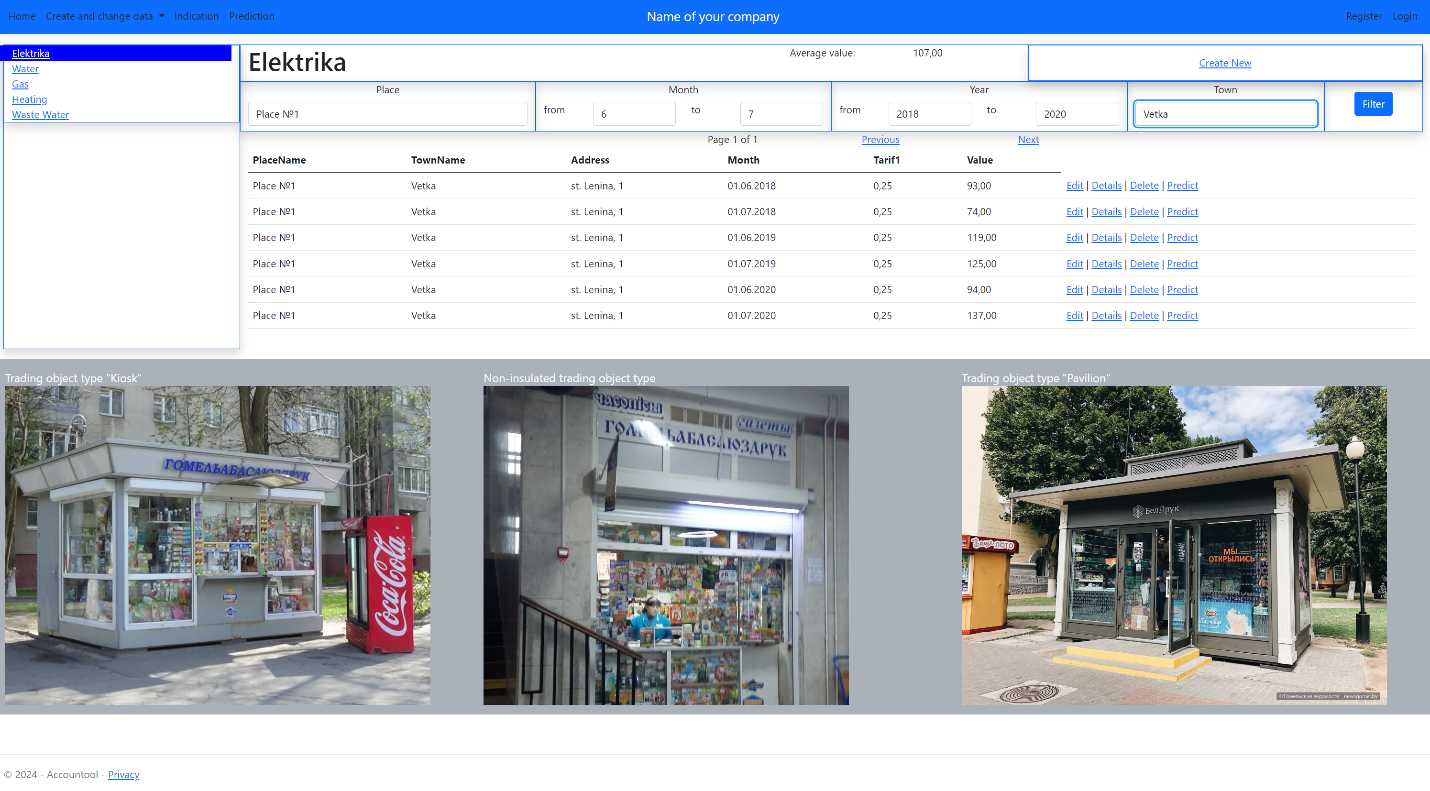


Рисунок 5.1.1.3 Просмотр показаний счетчиков

**5.1.3** Просмотр прогнозируемых показаний

Функционал прогнозирования позволяет оценить будущие показания по определенному месту. Для этого необходимо выбрать соответствующий раздел "Prediction" в меню и указать интересующее место и месяц. Такой анализ может быть полезен для планирования расходов и более эффективного использования ресурсов.



Рисунок 5.1.1.4 Прогнозируемые показания

Тестирование программного средства показало его высокую работоспособность и соответствие функциональным требованиям, заданным на этапе проектирования. Полученные результаты подтверждают эффективность реализованных алгоритмов обработки данных, удобство пользовательского интерфейса, а также быстродействие системы при работе с большими объемами информации.

**6 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ**

**6.1** **Установка программного средства**

Этот раздел предоставляет инструкции по установке и началу работы с .NET 6 MVC приложением для Windows, использующим сервер базы данных SQL Server. Подробно рассмотрены шаги установки самого приложения, настройки сервера баз данных, а также выполнения скриптов для создания базы данных и её наполнения начальными данными.

Подготовка к установке

Перед началом установки убедитесь, что на вашем компьютере установлены следующие компоненты:

1) Windows 10 или выше;

2) .NET 6 SDK и Runtime;

3) SQL Server (Express Edition подойдет для разработки и тестирования).

Установка приложения:

1) скачайте архив проекта или клонируйте репозиторий с помощью Git;

2) откройте командную строку или терминал в папке проекта и выполните команду `dotnet restore` для восстановления всех необходимых NuGet пакетов;

3) выполните команду `dotnet build`, чтобы скомпилировать проект и проверить наличие ошибок;

Настройка сервера баз данных SQL Server

1) установите SQL Server, если он еще не установлен. Можно использовать Express Edition для разработки и тестирования;

2) запустите SQL Server Management Studio (SSMS) и войдите в систему. Выполните скрипт `CREATE Accountool.sql` для создания структуры базы данных;

3) после создания базы данных и структуры выполните скрипт `FillIndication.sql` для инициализации базы начальными данными;

Запуск приложения

После настройки базы данных вернитесь в командную строку или терминал вашего проекта и запустите приложение командой `dotnet run`.

Первый запуск и использование

1) перейдите в браузере по указанному в консоли URL (обычно это `http://localhost:5000` или `https://localhost:5001`);

2) ипользуйте интерфейс приложения для регистрации нового пользователя и входа;

3) перейдите на страницу `Indications` в навигационном меню для просмотра показаний счетчиков и тестирования части работы с БД;

4) через меню в шапке сайта перейдите в раздел `Prediction`, чтобы просмотреть прогнозируемые показания для выбранного места и месяца для тестирования работоспособности модуля расчета будущих показаний;

Это руководство охватывает базовую установку, настройку и использование .NET 6 MVC приложения, включая интеграцию с SQL Server. В случае возникновения нестандартных ситуаций или ошибок, обратитесь к документации по соответствующим инструментам и технологиям или на форумы поддержки.

**7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ**

**7.1** **Описание функций и потенциальных пользователей**

Целью дипломного проекта является разработка программного средства для учета коммунальных услуг на предприятии. Коммунальные услуги - важный аспект для любого предприятия, и эффективное управление ими может привести к существенной экономии ресурсов. В условиях постоянно меняющихся тарифов и не всегда прозрачного подсчета ресурсов, проведение качественного и своевременного учета коммунальных услуг становится сложной задачей. Именно поэтому актуальна разработка специализированного программного средства, которое может автоматизировать большинство процессов и осуществлять прогнозирование на основе накопленных данных.

К основным функциям разрабатываемого программного средства можно отнести:

1) Ввод и отслеживание данных о потреблении коммунальных услуг;

2) Возможность хранения данных о потреблении в базе данных SQL Server;

3) Анализ потребления коммунальных услуг с помощью модуля ML.NET;

4) Прогнозирование показаний коммунальных услуг на следующий месяц;

5) Оповещение о предстоящем увеличении тарифов;

6) Работа с личным кабинетом с возможностью просмотра статистики потребления;

7) Задачи, которые решаются во время разработки:

8) Создание интуитивно понятного интерфейса для ввода и просмотра данных о потребляемых коммунальных услугах;

9) Разработка бекэнда для обработки запросов, взаимодействующего с базой данных и модулем прогнозирования;

10) Интеграция с ML.NET для создания прогнозов по потреблению коммунальных услуг;

11) Разработка схемы БД и алгоритмов взаимодействия с ней;

12) Реализация системы оповещения о предстоящем изменении тарифов;

13) Реализация системы авторизации и защиты данных пользователей;

Указанное программное обеспечение будет обеспечивать не только упрощение процедуры учета коммунальных услуг, но и будет служить инструментом для анализа и оптимизации их потребления на предприятиях.

Создаваемое программное средство может найти свое применение во множестве организаций, в особенности в тех, что имеют сеть точек продаж (ларьки, магазины и т.д.), расположенных в различных местах.

Программное средство разрабатывается для определенного заказчика, который в последствии будет обновлять, сопровождать и реализовывать приложение по собственной лицензии конечным пользователям. Заказчиком выступает ОАО “ЭПАМ Системз”, а программное средство разрабатывается ИП “WarmCode”.

Ожидаемые результаты использования разработанного программного продукта на предприятии включают повышение эффективности учета коммунальных услуг и улучшение качества принимаемых решений благодаря более аккуратным прогнозам. Так, с использованием функционала авторизации пользователя, есть возможность локализовать сбор и ввод данных непосредственно на местах установки счетчиков, что позволяет упростить и ускорить процесс сбора данных, при этом уменьшить нагрузку на инженеров, совершающих выезды для чтения показаний.

Применение реляционной СУБД обеспечивает безопасное, надежное и прозрачное хранение данных. Кроме того, архитектура реляционной СУБД способствует упорядочиванию данных и обеспечению их целостности.

Благодаря модулю ML.NET, предприятие получает возможность анализировать и прогнозировать расходы на коммунальные услуги. Это может помочь оптимизировать расходы и позволить контролировать и планировать бюджет на основе более точных и обоснованных данных.

В целом, внедрение такого программного средства даст возможность предприятию систематизировать и автоматизировать процесс учета коммунальных услуг, что должно привести к снижению затрат на сбор и обработку информации, улучшению контроля за расходами и повышению общей эффективности работы предприятия в этой области.

**7.2** **Расчет затрат на разработку**

На основании сметы затрат и анализа рынка подобных решений, определяется плановая цена продукта. Для составления объема затрат на создание программного средства, необходима предварительная оценка его трудоемкости и объёма. Расчет объёма частей программного продукта (эстимация) был получен при планировании архитектуры и разделении приложения на компонентные части. Оценка взята в часах на основании уже выполненных проектов с такими же компонентами для специалиста 1-го разряда. Подразумеваются фулстек-специалисты, которые имеют навыки работы как с базой данных, так и с пользовательским интерфейсом.

Таблица 7.2.1 - Компоненты системы и оценка создания компонентов

| Название компонента | Затраты времени на выполнение работы(час.) |
| --- | --- |
| Серверная часть бизнес логики | 87 |
| Компонент взаимодействия с базой данных | 40 |
| Компонент авторизации | 16 |
| Компонент пользовательского интерфейса | 54 |
| Компонент прогнозирования | 91 |
| Итого | 288 |

, (7.2.1)

На основе общей трудоемкости и требуемых сроков реализации проекта вычисляется плановое количество исполнителей. Численность исполнителей проекта рассчитывается по формуле:

, (7.2.2)

Где:

1) - общая трудоемкость разработки проекта, чел./дн.;

2) - эффективный фонд времени работы одного работника в течение года;

3) - срок разработки проекта, лет;

Эффективный фонд времени работы одного разработчика вычисляется по формуле:

, (7.2.3)

Где:

1) - количество дней в году;

2) - количество праздничных дней в году, не совпадающих с выходными днями [3];

3) - количество выходных дней в году [4];

4) - количество дней отпуска [5];

Согласно данным, приведенным в производственном календаре для пятидневной рабочей недели в 2024 году для Беларуси, фонд рабочего времени составит:

, (7.2.4)

Учитывая срок разработки проекта =1мес.=0,083 года, общую трудоемкость и фонд эффективного времени одного работника, вычисленные ранее, можем рассчитать численность исполнителей проекта

, (7.2.5)

Вычисленные оценки показывают, что для выполнения запланированного проекта в указанные сроки необходимо два программиста.

Заработная плата взята на основе средней зарплаты специалистов данного уровня по стране. Источником сведений является [9] и составляет 871 рубля.

Таблица 7.2.2 - Расчет заработной платы команды разработчиков

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Участник  команды | Выполняемые работы | Месячная  заработная  плата, р. | Часовая  заработная  плата, р. | Трудоемкость работ, часов | Основная  заработная  плата, тыс. р. |
| Разработчик 1 | Разработка функционала, тестирование,  исправление ошибок | 871 | 5.44 | 144 | 783 |
| Разработчик 2 | Разработка функционала, тестирование,  исправление ошибок | 871 | 5.44 | 144 | 783 |
| Премиальные 25% | 391.5 |  |  |  |  |
| Итого затраты на основную заработную плату разработчиков | 1957.5 |  |  |  |  |

Дополнительная заработная плата включает выплаты, предусмотренные законодательством в области труда, и определяется по нормативу в процентах от основной заработной платы:

, (7.2.6)

Где:

1) - норматив заработной платы в %;

Приняв норматив дополнительной заработной платы =20% и подставив известные данные в формулу (7.2.6) получим

, (7.2.7)

Согласно действующему законодательству отчисления в фонд социальной защиты населения составляют 35% [7] от фонда основной и дополнительной заработной платы исполнителей. Общие отчисления на социальную защиту рассчитываются по формуле

, (7.2.8)

Подставив вычисленные ранее значения в формулу (7.2.8) получаем

, (7.2.9)

По статье «материалы» проходят расходы на бумагу, краску для принтеров и другие материалы, используемые при разработке, оформлении деятельности и сопровождения договора.

Поскольку всю разработку выполняют фрилансеры со своим оборудованием и рабочими местами, нет необходимости в оплате аренды офиса или коммунальных услуг. Разработка осуществляется с использованием только бесплатных библиотек и фреймворков, а в качестве среды разработки применяется бесплатная версия Visual Studio Code.

Норма расходов определяется либо в расчете на 100 строк исходного кода, либо в процентах от основной зарплаты исполнителей 3%-5%. Затраты на материалы вычисляются по формуле:

, (7.2.10)

Где:

1) - норматив прочих затрат (3%);

=59, (7.2.11)

Таблица 7.2.3 - Затраты на прочие расходы

|  |  |  |
| --- | --- | --- |
| Статья прочих  затрат | Описание  затрат | Сумма, руб. |
| Канцелярия | Канцелярия  (бумага,краска  для принтеров,  и др.) | 59 |
| Итого прочие  расходы | 59 | |

Полная сумма затрат на разработку программного обеспечения находится путем суммирования всех рассчитанных статей затрат.

Таблица 7.2.4 - Затраты на разработку программного обеспечения

|  |  |
| --- | --- |
| Статья затрат | Сумма, руб. |
| Основная заработная плата команды разработчиков | 1957.5 |
| Дополнительная заработная плата команды разработчиков | C:\Users\BeRocker\AppData\Local\Microsoft\Windows\Clipboard\HistoryData\{59920C2A-62A2-4886-99CF-0F93DA55BEE3}\{A0C3F5FF-C19F-4B12-8F46-06973022278D}\ResourceMap\{CDB4F1D9-0567-4AAF-A546-D57AD68B1F6C} |
| Отчисления на социальные нужды | 821.9 |
| Прочие затраты | 59 |
| Общая сумма затрат на разработку | 3229.9 |

**7.3** **Оценка результата от реализации веб-сервиса**

При создании программного средства для прямого заказчика экономический эффект должен быть рассчитан для компании-разработчика. Программное средство разрабатывается по заказу компании “Эпам Системз”. Заказчик стремиться автоматизировать все бизнес-процессы внутри предприятия-от финансового учета до управления персоналом. Компания не просто приобретает программное средство, она инвестирует в полностью разработанное решение с кодовой базой и лицензионными правами. Поскольку заказчик станет полноправным владельцем программы, он может продавать программное средство своим клиентам, открывая новые перспективы для своего бизнеса. "Эпам Системз" безусловно способна самостоятельно реализовать данный проект, однако время работы наших разработчиков стоит дешевле, по сравнению с временем специалистов "Эпам Системз", занятых выполнением международных заказов. Таким образом, сотрудничество с нами является более выгодной и экономически эффективной стратегией. Проведем анализ других непосредственных конкурентов:

1) ИООО «ЭПАМ Системз» [10] Цена: 6188 рублей;

2) «МИРТЕК» [11] Цена 5638 рубля;

Проведя анализ конкурентов выяснилось, что некоторые конкуренты предлагают подобные решения. Но наше программное средство будет отличаться высокой степенью персонализации и простотой в использовании. Так же стало ясно что большинство аналогов имеют более высокую цену чем наше программное средство. Это позволяет установить конкурентные цены в сочетании с качеством и удобством в использовании.

Экономический эффект для нашей компании состоит в получении прибыли от создания программного продукта для клиента. Мы можем установить уровень доходности, который соответствует нашим бизнес-целям, и на его основе определить цену нашего программного средства. Если наша цена окажется ниже среднерыночной стоимости похожих программных продуктов, то при прочих равных условиях клиент, скорее всего, отдаст предпочтение нашему предложению. Так как наша организация только выходит на данный рынок и заинтересована в привлечении клиентов, то целесообразно применить описанный подход.

, (7.3.1)

Где:

1) - запланированный норматив рентабельности в 27%;

Подставив данные в формулу (7.3.1) получаем цену ПО без налогов:

, (7.3.2)

Налог на добавленную стоимость рассчитывается по формуле:

, (7.3.3)

Где:

1) Ндс - норматив НДС в %;

Норматив НДС составляет Ндс = 20%, подставляя известные значения в формулу (7.3.3) получаем:

, (7.3.4)

Расчет прогнозируемой отпускной цены осуществляется по формуле:

, (7.3.5)

Подставляя известные данные в формулу (7.3.5) получаем прогнозируемую отпускную цену:

, (7.3.6)

Чистую прибыль от реализации проекта можно рассчитать по формуле.

, (7.3.7)

Где:

1) - норматив налога на прибыль в 25%;

, (7.3.8)

**7.4 Выводы по технико-экономическому обоснованию**

Подобное программное обеспечение на рынке услуг Беларуси можно приобрести от 5638 руб. Настройка серверов и сопровождение программного средства не входит в стоимость. А как следствие, организация-разработчик имеет конкурентное преимущество, что делает этот проект экономически целесообразным.

В данной модели работы заказчик, согласившийся с условиями контракта и стоимостью программного продукта, делает предоплату, которая обеспечивает финансирование начального этапа разработки. Это делает процесс экономически безопасным для разработчика, поскольку расходы на разработку и тестирование продукта покрываются заранее, избавляя от необходимости привлекать дополнительные инвестиции для реализации проекта и делая технико-экономическое обоснование проекта более предсказуемым.

Программное обеспечение разрабатывалось для одного заказчика. В связи с этим, экономический эффект для разработчика будет выражаться в чистой прибыли от его реализации. Таким образом, было произведено технико-экономическое обоснование разрабатываемого проекта, составлена смета затрат, рассчитана прогнозируемая прибыль и показана экономическая целесообразность разработки.

**ЗАКЛЮЧЕНИЕ**

В данном дипломном проекте была разработана и исследована система автоматизации учёта коммунальных услуг на предприятии, что позволило значительно упростить и оптимизировать процесс учёта и контроля за расходованием коммунальных ресурсов. Проанализированы существующие методы и подходы к автоматизации учета коммунальных услуг, на основе которых было создано программное средство, включающее в себя функции ввода, обработки и хранения данных о потреблении ресурсов, а также модуль формирования отчётов для анализа эффективности использования ресурсов предприятием. Результаты работы над проектом показали, что автоматизация процессов учета коммунальных услуг позволяет существенно повысить точность данных, сократить время на обработку информации и улучшить управление ресурсами, что в итоге способствует повышению экономической эффективности предприятия. Созданное программное средство может быть адаптировано под специфику различных организаций, что подтверждает его универсальность и эффективность в решении задач автоматизации учета коммунальных услуг.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Ресурс министерства труда - <https://mintrud.gov.by/system/extensions/> spaw/uploads/files/Post-mintruda-6.pdf

[2] Единая тарифная сетка (ЕТС), установленная Поста- новлением Совета министров Республики Беларусь № 138 от 28.02.2019, Ссылка на текст правового акта в актуальном состоянии в информационно-поисковой системе «Эталон-online»: https://etalonline.by/document/?regnum=C21900138)

[3] Министерство труда: праздничные и выходные дни - https://www.mintrud.gov.by/ru/prazdnichnie-i-vihodnie-dni-ru

[4] Министерство труда: количество выходных дней - https://www.mintrud.gov.by/uploads/files/Proizvodstvennyj-kalendar-2024.pdf

[5] Министерство труда: Виды трудовых отпусков. Дополнительные отпуска - https://www.mintrud.gov.by/ru/vidu-trud-otpuskov-ru

[6] Национальный правовой Интернет-портал Республики Беларусь - https://pravo.by/novosti/novosti-pravo-by/2023/november/75891

[7] Фонд социальной защиты населения - https://ssf.gov.by/ru/ind-predprin-ru/view/informatsija-dlja-individualnyx-predprinimatelej-advokatov-notariusov-o-porjadke-i-srokax-uplaty-10742

[8] Официальная статистика Белстат - <https://www.belstat.gov.by/>

ofitsialnaya-statistika/realny-sector-ekonomiki/stoimost-rabochey-sily/operativnye-dannye/o-nachislennoy-sredney-zarabotnoy-plate-rabotnikov/o-nachislennoy-sredney-zarabotnoy-plate-rabotnikov-respubliki-belarus-v-dekabre-2023-g

[9] CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. | Рихтер Джеффри - <https://ozon.by/product/clr-via-c-programmirovanie-na-platforme-microsoft-net-framework-4-5-na-yazyke-c-4-e-izd-211432904>

[10] Отдел продаж Эпам Системз - https://www.epam.com/about/who-we-are/contact

[11] Официальный сайт МИРТЕК - https://mirtekgroup.com/kontakty

**ПРИЛОЖЕНИЕ А**

**Текст программного модуля агрегации данных показаний счетчиков**

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Accountool.Models.Entities;

using Accountool.Models.DataAccess;

using Accountool.Models.ViewModel;

using System.Data;

using Accountool.Models.Models;

using Humanizer;

using System;

namespace Accountool.Models.Services

{

/// <summary>

/// Определяет необходимые методы для операций Службы измерений,

/// связанных с показаниями, измерениями и связанными с ними сущностями.

/// </summary>

public interface IMeasurementService

{

/// <summary>

/// Извлекает показание по его уникальному идентификатору.

/// </summary>

/// <param name="id">Идентификатор показания.</param>

/// <returns>Задача, представляющая асинхронную операцию. Результат задачи содержит показание.</returns>

Task<Indication> GetById(int id);

/// <summary>

/// Добавляет новое показание.

/// </summary>

/// <param name="entity">Показание для добавления.</param>

/// <returns>Задача, представляющая асинхронную операцию.</returns>

Task Add(Indication entity);

/// <summary>

/// Обновляет существующее показание.

/// </summary>

/// <param name="entity">Показание для обновления.</param>

/// <returns>Задача, представляющая асинхронную операцию.</returns>

Task Update(Indication entity);

/// <summary>

/// Удаляет показание по его уникальному идентификатору.

/// </summary>

/// <param name="id">Идентификатор показания.</param>

/// <returns>Задача, представляющая асинхронную операцию.</returns>

Task Remove(int id);

/// <summary>

/// Извлекает все показания для указанного счетчика.

/// </summary>

/// <param name="SchetchikId">Идентификатор счетчика.</param>

/// <returns>Задача, представляющая асинхронную операцию. Результат задачи содержит показания.</returns>

Task<IEnumerable<Indication>> GetForSchetchik(int SchetchikId);

/// <summary>

/// Получает отфильтрованный список полных моделей показаний на основе заданных критериев.

/// </summary>

/// <param name="measureTypeid">Идентификатор типа измерения для фильтрации.</param>

/// <param name="townId">Необязательный идентификатор города для фильтрации.</param>

/// <param name="placeId">Необязательный идентификатор места для фильтрации.</param>

/// <param name="monthFrom">Необязательный начальный месяц для фильтрации.</param>

/// <param name="monthTo">Необязательный конечный месяц для фильтрации.</param>

/// <param name="yearFrom">Необязательный начальный год для фильтрации.</param>

/// <param name="yearTo">Необязательный конечный год для фильтрации.</param>

/// <returns>Задача, представляющая асинхронную операцию. Результат задачи содержит отфильтрованные полные модели показаний.</returns>

Task<IQueryable<FullIndicationModel>> GetFilteredMeasures(int measureTypeid, int? townId = null, int? placeId = null, int? monthFrom = null, int? monthTo = null, int? yearFrom = null, int? yearTo = null);

/// <summary>

/// Получает последнее измерение по счетчику и типу измерения.

/// </summary>

/// <param name="measureTypeid">Идентификатор типа измерения.</param>

/// <param name="counterId">Идентификатор счетчика.</param>

/// <returns>Задача, представляющая асинхронную операцию. Результат задачи содержит последнюю полную модель показаний.</returns>

Task<FullIndicationModel> GetLastMeasureByCounter(int measureTypeid, int counterId);

/// <summary>

/// Получает все типы измерений.

/// </summary>

/// <returns>Задача, представляющая асинхронную операцию. Результат задачи содержит типы измерений.</returns>

Task<IEnumerable<MeasureType>> GetAllMeasureTypes();

/// <summary>

/// Получает все показания.

/// </summary>

/// <returns>Задача, представляющая асинхронную операцию. Результат задачи содержит показания.</returns>

Task<IEnumerable<Indication>> GetAllIndications();

/// <summary>

/// Получает минимальный и максимальный год показаний для данного типа измерения.

/// </summary>

/// <param name="measureTypeid">Идентификатор типа измерения.</param>

/// <returns>Задача, представляющая асинхронную операцию. Результат задачи содержит модель первого и последнего года.</returns>

Task<FirstLastYearModel> GetMinMaxYear(int measureTypeid);

/// <summary>

/// Получает места, отфильтрованные по типу измерения.

/// </summary>

/// <param name="measureTypeid">Идентификатор типа измерения.</param>

/// <returns>Задача, представляющая асинхронную операцию. Результат задачи содержит места.</returns>

Task<IEnumerable<IdNameModel>> GetPlaces(int measureTypeid);

/// <summary>

/// Получает города, отфильтрованные по типу измерения.

/// </summary>

/// <param name="measureTypeid">Идентификатор типа измерения.</param>

/// <returns>Задача, представляющая асинхронную операцию. Результат задачи содержит города.</returns>

Task<IEnumerable<IdNameModel>> GetTowns(int measureTypeid);

}

/// <summary>

/// Реализует интерфейс IMeasurementService для управления

/// операциями, связанными с измерениями и показаниями.

/// </summary>

public class MeasurementService : IMeasurementService

{

private readonly IRepository<Indication> \_indications;

private readonly IRepository<Schetchik> \_schetchiks;

private readonly IRepository<Place> \_places;

private readonly IRepository<Town> \_town;

private readonly IRepository<MeasureType> \_measureTypes;

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="MeasurementService"/>.

/// </summary>

/// <param name="indications">Репозиторий показаний.</param>

/// <param name="schetchiks">Репозиторий счетчиков.</param>

/// <param name="place">Репозиторий мест.</param>

/// <param name="town">Репозиторий городов.</param>

/// <param name="measureTypes">Репозиторий типов измерений.</param>

public MeasurementService(

IRepository<Indication> indications,

IRepository<Schetchik> schetchiks,

IRepository<Place> place,

IRepository<Town> town,

IRepository<MeasureType> measureTypes)

{

\_indications = indications; // Назначает репозиторий показаний

\_schetchiks = schetchiks; // Назначает репозиторий счетчиков

\_measureTypes = measureTypes; // Назначает репозиторий типов измерений

\_places = place; // Назначает репозиторий мест

\_town = town; // Назначает репозиторий городов

}

/// <inheritdoc />

public async Task<IEnumerable<Indication>> GetAllIndications()

{

return \_indications.GetAll(); // Получает все показания

}

/// <inheritdoc />

public async Task<IEnumerable<MeasureType>> GetAllMeasureTypes()

{

return \_measureTypes.GetAll(); // Получает все типы измерений

}

/// <inheritdoc />

public async Task<IQueryable<FullIndicationModel>> GetFilteredMeasures(

int measureTypeid,

int? townId = null,

int? placeId = null,

int? monthFrom = null,

int? monthTo = null,

int? yearFrom = null,

int? yearTo = null)

{

// Конструирует запрос для фильтрации показаний

var indications = from mt in \_measureTypes.GetAll()

join s in \_schetchiks.GetAll() on mt.Id equals s.MeasureTypeId

join i in \_indications.GetAll() on s.Id equals i.SchetchikId

join k in \_places.GetAll() on s.PlaceId equals k.Id

join t in \_town.GetAll() on k.TownId equals t.Id

where mt.Id == measureTypeid

&& (placeId == null || k.Id == placeId)

&& (townId == null || k.TownId == townId)

select new { i, k, t, mt, s }; // Строит запрос для фильтрации показаний

// Отфильтровывает показания по указанным критериям

if (monthFrom != null)

{

indications = indications.Where(x => x.i.Month.Month >= monthFrom); // Фильтрует показания по начальному месяцу

}

if (monthTo != null)

{

indications = indications.Where(x => x.i.Month.Month <= monthTo); // Фильтрует показания по конечному месяцу

}

if (yearFrom != null)

{

indications = indications.Where(x => x.i.Month.Year >= yearFrom); // Фильтрует показания по начальному году

}

if (yearTo != null)

{

indications = indications.Where(x => x.i.Month.Year <= yearTo); // Фильтрует показания по конечному году

}

// Преобразует результат запроса в FullIndicationModel

return indications.Select(x => new FullIndicationModel()

{

PlaceId = x.k.Id, // Устанавливает идентификатор места

PlaceName = x.k.Name, // Устанавливает название места

Address = x.k.Address, // Устанавливает адрес

TownName = x.t.Name, // Устанавливает название города

Indication = x.i, // Устанавливает показание

MeasureTypeId = x.mt.Id // Устанавливает идентификатор типа измерения

}); // Преобразует результат запроса в FullIndicationModel

}

/// <inheritdoc />

public async Task<FullIndicationModel> GetLastMeasureByCounter(

int measureTypeid,

int counterId)

{

// Конструирует запрос для поиска последнего показания по счетчику

var indications = from mt in \_measureTypes.GetAll()

join s in \_schetchiks.GetAll() on mt.Id equals s.MeasureTypeId

join i in \_indications.GetAll() on s.Id equals i.SchetchikId

join k in \_places.GetAll() on s.PlaceId equals k.Id

join t in \_town.GetAll() on k.TownId equals t.Id

where mt.Id == measureTypeid

&& s.Id == counterId

select new { i, k, t }; // Конструирует запрос для поиска последнего показания по счетчику

// Сортирует найденные показания по дате и выбирает первое (самое старое)

var indication = indications.OrderBy(x => x.i.Month).FirstOrDefault(); // Сортирует найденные показания по месяцу и выбирает первое

// Преобразует детали показания в FullIndicationModel

var result = new FullIndicationModel()

{

PlaceId = indication.k.Id, // Устанавливает идентификатор места

PlaceName = indication.k.Name, // Устанавливает название места

Address = indication.k.Address, // Устанавливает адрес

TownName = indication.t.Name, // Устанавливает название города

Indication = indication.i // Устанавливает показание

}; // Преобразует детали показания в FullIndicationModel

return result; // Возвращает результат

}

/// <inheritdoc />

public async Task<FirstLastYearModel> GetMinMaxYear(

int measureTypeid)

{

// Конструирует запрос для поиска минимального и максимального года показаний для типа измерения

var minMaxDate = from mt in \_measureTypes.GetAll()

join s in \_schetchiks.GetAll() on mt.Id equals s.MeasureTypeId

join i in \_indications.GetAll() on s.Id equals i.SchetchikId

where mt.Id == measureTypeid

select i.Month; // Конструирует запрос для поиска дат

if (minMaxDate != null && minMaxDate.Any())

{

var minMaxDateTime = new FirstLastYearModel

{

FirstDate = minMaxDate.Min(), // Находит самую раннюю дату

LastDate = minMaxDate.Max() // Находит самую позднюю дату

}; // Создает и заполняет модель FirstLastYearModel

return minMaxDateTime; // Возвращает модель

}

return null; // Возвращает null, если даты не найдены

}

/// <inheritdoc />

public async Task<IEnumerable<IdNameModel>> GetPlaces(

int measureTypeid)

{

// Конструирует запрос для получения мест, отфильтрованных по типу измерения

var places = from mt in \_measureTypes.GetAll()

join s in \_schetchiks.GetAll() on mt.Id equals s.MeasureTypeId

join i in \_indications.GetAll() on s.Id equals i.SchetchikId

join k in \_places.GetAll() on s.PlaceId equals k.Id

where mt.Id == measureTypeid

select new IdNameModel { Id = k.Id, Name = k.Name }; // Конструирует запрос для получения мест

return places; // Возвращает места

}

/// <inheritdoc />

public async Task<IEnumerable<IdNameModel>> GetTowns(

int measureTypeid)

{

// Конструирует запрос для получения городов, отфильтрованных по типу измерения

var townsId = from mt in \_measureTypes.GetAll()

join s in \_schetchiks.GetAll() on mt.Id equals s.MeasureTypeId

join i in \_indications.GetAll() on s.Id equals i.SchetchikId

join k in \_places.GetAll() on s.PlaceId equals k.Id

join t in \_town.GetAll() on k.TownId equals t.Id

where mt.Id == measureTypeid

select new IdNameModel { Id = t.Id, Name = t.Name }; // Конструирует запрос для получения городов

return townsId; // Возвращает города

}

/// <inheritdoc />

public async Task<Indication> GetById(int id)

{

return await \_indications.GetByIdAsync(id); // Извлекает показание по его ID

}

/// <inheritdoc />

public async Task Add(Indication entity)

{

await \_indications.AddAsync(entity); // Добавляет новое показание

}

/// <inheritdoc />

public async Task Update(Indication entity)

{

\_indications.Update(entity); // Обновляет существующее показание

}

/// <inheritdoc />

public async Task Remove(int id)

{

var model = await \_indications.GetByIdAsync(id); // Извлекает показание по его ID

\_indications.Remove(model); // Удаляет показание

}

/// <inheritdoc />

public async Task<IEnumerable<Indication>> GetForSchetchik(int SchetchikId)

{

return \_indications.GetAll().Where(i => i.SchetchikId == SchetchikId).AsEnumerable(); // Получает все показания для указанного счетчика

}

}

}

**ПРИЛОЖЕНИЕ Б**

**Текст программного модуля прогнозирования будущих данных**

using System;

using System.Linq;

using System.Threading.Tasks;

using Accountool.Models.DataAccess;

using Accountool.Models.Entities;

using Microsoft.ML;

using Microsoft.ML.Data;

using Microsoft.ML.Transforms;

namespace Accountool.Models.Services

{

/// <summary>

/// Определяет методы для сервиса ИИ, направленные на прогнозирование потребления.

/// </summary>

public interface IAIService

{

/// <summary>

/// Осуществляет одиночное прогнозирование потребления на основе заданных параметров.

/// </summary>

/// <param name="measureTypeId">Идентификатор типа измерения.</param>

/// <param name="month">Месяц, для которого требуется прогноз.</param>

/// <param name="measurementObjectId">Идентификатор объекта измерения.</param>

/// <returns>Одиночное прогнозирование потребления.</returns>

Task<ConsumptionPrediction> SinglePrediction(int measureTypeId = 1, int month = 2, int measurementObjectId = 0);

/// <summary>

/// Генерирует прогнозируемые показания счетчиков на основе заданных параметров.

/// </summary>

/// <param name="measureTypeId">Идентификатор типа измерения.</param>

/// <param name="month">Месяц, для которого требуется прогноз.</param>

/// <param name="measurementObjectId">Идентификатор объекта измерения.</param>

/// <returns>Коллекция прогнозируемых показаний счетчиков.</returns>

Task<IEnumerable<ConsumptionPrediction>> PredictedMeterReadings(int measureTypeId = 1, int month = 2, int measurementObjectId = 0);

}

/// <summary>

/// Реализует интерфейс IAIService, предоставляя методы машинного обучения для прогнозирования потребления.

/// </summary>

public class AIService : IAIService

{

private readonly IRepository<Indication> \_indications;

private readonly IRepository<MeasureType> \_measureTypes;

private readonly IRepository<Schetchik> \_schetchiks;

private readonly IRepository<Place> \_places;

public AIService(

IRepository<Indication> indications,

IRepository<MeasureType> measureTypes,

IRepository<Schetchik> schetchiks,

IRepository<Place> places)

{

\_indications = indications; // Репозиторий показаний

\_measureTypes = measureTypes; // Репозиторий типов измерений

\_schetchiks = schetchiks; // Репозиторий счетчиков

\_places = places; // Репозиторий мест

}

/// <inheritdoc />

public async Task<ConsumptionPrediction> SinglePrediction(int measureTypeId = 1, int month = 2, int measurementObjectId = 0)

{

var mlContext = new MLContext(); // Инициализация контекста ML.NET

// Запрос к базе данных для извлечения данных для обучения

var indications = from mt in \_measureTypes.GetAll()

join s in \_schetchiks.GetAll() on mt.Id equals s.MeasureTypeId

join i in \_indications.GetAll() on s.Id equals i.SchetchikId

join k in \_places.GetAll() on s.PlaceId equals k.Id

where mt.Id == measureTypeId

select new ConsumptionData { Month = i.Month.Month, Label = Convert.ToSingle(i.Value), MeasurementObjectId = s.Id };

var data = mlContext.Data.LoadFromEnumerable(indications.ToList()); // Загрузка данных для обучения

// Предварительная обработка данных

var processedData = mlContext.Transforms.CustomMapping(

(Action<ConsumptionData, ConsumptionData>)Mapping, // Определяет пользовательское преобразование, применяемое к данным. Преобразование задается функцией Mapping, которая переводит входные данные из формата ConsumptionData в выходные данные того же формата, потенциально модифицируя их.

contractName: "CleanupMapping" // Задает уникальное имя (контракт) для этого преобразования, чтобы его можно было идентифицировать в контексте ML.NET pipeline.

).Fit(data) // Применяет определенное пользовательское преобразование к предоставленным данным, обучая трансформер. В этом случае Fit на самом деле не выполняет обучение, а подготавливает данные к трансформации, проверяя соответствие и типы данных.

.Transform(data); // Преобразует данные с использованием обученного трансформера, получая на выходе обработанный набор данных, готовый к дальнейшему использованию в ML pipeline.

// Заполнение отсутствующих значений

processedData = mlContext.Transforms.ReplaceMissingValues(

"Label", // Указывает на столбец, в котором необходимо заменить пропущенные значения.

replacementMode: MissingValueReplacingEstimator.ReplacementMode.Mean // Задает режим замены пропущенных значений на среднее значение по столбцу.

).Fit(processedData) // Применяет операцию к данным, подготавливает и вычисляет необходимые параметры (в данном случае среднее значение).

.Transform(processedData); // Производит замену пропущенных значений на рассчитанное среднее значение в столбце "Label".

// Построение конвейера машинного обучения

var pipeline = mlContext.Transforms.Categorical.OneHotEncoding("Month") // Преобразует категориальный столбец "Month" в числовой формат с использованием метода One-Hot Encoding.

.Append(mlContext.Transforms.Categorical.OneHotEncoding("MeasurementObjectId")) // Преобразует категориальный столбец "MeasurementObjectId" в числовой формат с использованием метода One-Hot Encoding.

.Append(mlContext.Transforms.Concatenate("Features", "Month", "MeasurementObjectId")) // Объединяет преобразованные столбцы "Month" и "MeasurementObjectId" в один столбец "Features".

.Append(mlContext.Transforms.NormalizeMinMax("Features")) // Нормализует значения в столбце "Features" между 0 и 1.

.Append(mlContext.Regression.Trainers.FastTree()) // Добавляет алгоритм регрессии FastTree для тренировки на подготовленных данных.

.AppendCacheCheckpoint(mlContext); // Добавляет точку кеширования для улучшения производительности путем кеширования данных в памяти после преобразований.

var model = pipeline.Fit(processedData); // Обучение модели

// Создание функции для предсказания на основе модели

var predictionFunction = mlContext.Model.CreatePredictionEngine<ConsumptionData, ConsumptionPrediction>(model);

// Прогнозирование для указанного месяца

var singlePrediction = predictionFunction.Predict(new ConsumptionData() { Month = month, MeasurementObjectId = measurementObjectId });

return singlePrediction; // Возвращение результата прогноза

}

/// <inheritdoc />

public async Task<IEnumerable<ConsumptionPrediction>> PredictedMeterReadings(int measureTypeId = 1, int month = 2, int measurementObjectId = 0)

{

var mlContext = new MLContext(); // Инициализация контекста ML.NET

// Запрос к базе данных для извлечения данных для прогнозирования

var indications = from mt in \_measureTypes.GetAll()

join s in \_schetchiks.GetAll() on mt.Id equals s.MeasureTypeId

join i in \_indications.GetAll() on s.Id equals i.SchetchikId

join k in \_places.GetAll() on s.PlaceId equals k.Id

where mt.Id == measureTypeId

select new ConsumptionData { Month = i.Month.Month, Label = Convert.ToSingle(i.Value) };

var data = mlContext.Data.LoadFromEnumerable(indications.ToList()); // Загрузка данных для прогнозирования

// Предварительная обработка данных

// Применяет пользовательское преобразование к данным

var processedData = mlContext.Transforms.CustomMapping(

(Action<ConsumptionData, ConsumptionData>)Mapping, // Указывает делегат действия, который определяет, как данные должны быть изменены

contractName: "CleanupMapping" // Название контракта для преобразования, необходимо для внутренней организации ML.NET

).Fit(data) // "Обучает" преобразование. Для CustomMapping это просто подготовка данных без истинного обучения

.Transform(data); // Преобразует исходные данные согласно пользовательскому преобразованию

// Заполнение отсутствующих значений в данных

processedData = mlContext.Transforms.ReplaceMissingValues(

"Label", // Указывает имя столбца, в котором нужно заменить пропущенные значения

replacementMode: MissingValueReplacingEstimator.ReplacementMode.Mean // Указывает, что пропущенные значения будут заменены на среднее по столбцу

).Fit(processedData) // Подготавливает преобразование, вычисляя необходимые статистики (среднее)

.Transform(processedData); // Применяет преобразование, заменяя пропущенные значения

// Построение конвейера машинного обучения

var pipeline = mlContext.Transforms.Categorical.OneHotEncoding("Month") // Переводит категориальные значения месяца в формат One-Hot encoding

.Append(mlContext.Transforms.Concatenate("Features", "Month", "MeasurementObject")) // Конкатенирует столбцы "Month" и "MeasurementObject" в один столбец "Features"

.Append(mlContext.Transforms.NormalizeMinMax("Features")) // Применяет нормализацию к столбцу "Features", масштабируя данные к диапазону [0, 1]

.Append(mlContext.Regression.Trainers.FastTree()) // Добавляет тренер регрессии FastTree для построения модели

.AppendCacheCheckpoint(mlContext); // Добавляет кеширование данных после преобразований для улучшения производительности

var model = pipeline.Fit(processedData); // Применяет конвейер к обработанным данным, обучая модель

// Создание предсказаний на основе обученной модели

var prediction = model.Transform(processedData); // Применяет модель к данным для генерации предсказаний

// Получение перечисляемого списка предсказаний

var predictedMeterReadings = mlContext.Data.CreateEnumerable<ConsumptionPrediction>(prediction, reuseRowObject: false); // Создает энумератор для перебора предсказаний, не переиспользуя объекты строк

return predictedMeterReadings; // Возвращение коллекции предсказаний

}

/// <summary>

/// Определяет пользовательское преобразование данных для подготовки к обучению модели.

/// </summary>

private static void Mapping(ConsumptionData input, ConsumptionData output)

{

output.Month = input.Month; // Назначает входной месяц в выходной объект данных месяца. Это позволяет перенести значение месяца из исходных данных в модель для дальнейшего использования.

output.MeasurementObjectId = input.MeasurementObjectId; // Назначает входной идентификатор объекта измерения в выходной объект данных. Переносит идентификатор объекта измерения в модель данных для дальнейшего анализа или использования.

// Проверка на допустимость значения

if (input.Label <= 0.0f || input.Label > 1000.0f) // Проверяет, попадает ли значение входной метки (предположительно потребление или показание) в допустимый диапазон. Значения вне диапазона считаются ошибочными или недопустимыми.

{

output.Label = float.NaN; // Присваивание NaN для недопустимых значений. Если входное значение метки недопустимо, выходной метке присваивается значение NaN, что обычно используется для обозначения отсутствия или невозможности значения в числовых данных.

}

else

{

output.Label = input.Label; // Копирование допустимого значения. Если входное значение метки допустимо, оно назначается выходной метке без изменений, сохраняя изначальное значение потребления или показаний для дальнейшей обработки.

}

}

}

/// <summary>

/// Представляет входные данные для модели ML.NET.

/// </summary>

public class ConsumptionData

{

[LoadColumn(0)] public int Month;

[LoadColumn(1)] public float Label;

[LoadColumn(2)] public int MeasurementObjectId;

}

/// <summary>

/// Представляет выходные данные модели ML.NET.

/// </summary>

public class ConsumptionPrediction

{

[ColumnName("Score")] public float PredictedLabel;

}

}

**ПРИЛОЖЕНИЕ В**

**Текст программного модуля реализации шаблона репозиторий**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using System.Threading.Tasks;

using Accountool.Data; // Подключение пространства имен, где, вероятно, находится контекст базы данных

using Microsoft.EntityFrameworkCore; // Подключение Entity Framework Core для работы с базой данных

namespace Accountool.Models.DataAccess

{

// Определяет интерфейс репозитория для осуществления базовых операций CRUD над сущностями TEntity

public interface IRepository<TEntity> where TEntity : class

{

// Определение метода для получения экземпляра TEntity по идентификатору

Task<TEntity> GetByIdAsync(int id);

// Определение метода для получения всех экземпляров TEntity

IQueryable<TEntity> GetAll();

// Определение асинхронного метода добавления нового экземпляра TEntity

Task AddAsync(TEntity entity);

// Определение метода обновления существующего экземпляра TEntity

void Update(TEntity entity);

// Определение метода удаления экземпляра TEntity

void Remove(TEntity entity);

}

// Реализует интерфейс IRepository<TEntity> для выполнения операций над сущностями в контексте базы данных

public class Repository<TEntity> : IRepository<TEntity> where TEntity : class

{

// Защищенное поле контекста, через которое будет осуществляться работа с базой данных

protected readonly DbContext Context;

// Конструктор, инициализирующий поле контекста конкретным экземпляром ApplicationDbContext

public Repository(ApplicationDbContext context)

{

Context = context; // Присваивание контекста базы данных переданному в конструктор параметру

}

// Асинхронный метод поиска и возврата экземпляра TEntity по его идентификатору

public async Task<TEntity> GetByIdAsync(int id)

{

// Асинхронный поиск экземпляра по идентификатору в наборе TEntity контекста

return await Context.Set<TEntity>().FindAsync(id);

}

// Возвращает запрос, представляющий все экземпляры TEntity, хранимые в базе данных

public IQueryable<TEntity> GetAll()

{

// Получение DbSet для TEntity из контекста

return Context.Set<TEntity>();

}

// Асинхронное добавление нового экземпляра TEntity в контекст с последующим сохранением изменений в базе данных

public async Task AddAsync(TEntity entity)

{

// Асинхронное добавление экземпляра entity в DbSet<TEntity>

await Context.Set<TEntity>().AddAsync(entity);

// Асинхронное сохранение изменений в базе данных

await Context.SaveChangesAsync();

}

// Обновляет существующий экземпляр TEntity в контексте и сохраняет изменения в базе данных

public void Update(TEntity entity)

{

// Обновление экземпляра entity в DbSet<TEntity>

Context.Set<TEntity>().Update(entity);

// Сохранение изменений в базе данных

Context.SaveChanges();

}

// Удаление существующего экземпляра TEntity из контекста и сохранение изменений в базе данных

public void Remove(TEntity entity)

{

// Удаление экземпляра entity из DbSet<TEntity>

Context.Set<TEntity>().Remove(entity);

// Сохранение изменений в базе данных

Context.SaveChanges();

}

}

}

**ПРИЛОЖЕНИЕ Г**

**Текст программного модуля контекста базы данных**

using Accountool.Models.Entities;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

namespace Accountool.Data // Определяем пространство имен для контекста данных приложения

{

// Создаем класс контекста данных, наследуясь от IdentityDbContext для интеграции Identity

public partial class ApplicationDbContext : IdentityDbContext

{

// Конструктор без параметров

public ApplicationDbContext()

{

}

// Конструктор с параметрами для настройки контекста через объект опций

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options) // Вызываем конструктор базового класса с параметрами

{

}

// Далее идет определение свойств DbSet для каждого типа сущности.

// Эти свойства позволяют взаимодействовать с соответствующими таблицами в базе данных.

public virtual DbSet<AspNetRole> AspNetRoles { get; set; } // Таблица ролей пользователей

public virtual DbSet<AspNetRoleClaim> AspNetRoleClaims { get; set; } // Таблица претензий ролей

public virtual DbSet<AspNetUser> AspNetUsers { get; set; } // Таблица пользователей

public virtual DbSet<AspNetUserClaim> AspNetUserClaims { get; set; } // Таблица претензий пользователей

public virtual DbSet<AspNetUserLogin> AspNetUserLogins { get; set; } // Таблица логинов пользователей

public virtual DbSet<AspNetUserToken> AspNetUserTokens { get; set; } // Таблица токенов пользователей

public virtual DbSet<ClientProfile> ClientProfiles { get; set; } // Таблица профилей клиентов

public virtual DbSet<Equipment> Equipment { get; set; } // Таблица оборудования

public virtual DbSet<Indication> Indications { get; set; } // Таблица показаний

public virtual DbSet<Place> Places { get; set; } // Таблица мест

public virtual DbSet<PlaceSection> PlaceSections { get; set; } // Таблица секций мест

public virtual DbSet<Contract> Contracts { get; set; } // Таблица контрактов

public virtual DbSet<Organization> Organizations { get; set; } // Таблица организаций

public virtual DbSet<Schetchik> Schetchiks { get; set; } // Таблица счетчиков

public virtual DbSet<Town> Towns { get; set; } // Таблица городов

public virtual DbSet<UserPlace> UserPlaces { get; set; } // Таблица связей пользователей и мест

public virtual DbSet<AspNetUserRole> AspNetUserRoles { get; set; } // Таблица ролей пользователей

public virtual DbSet<MeasureType> MeasureTypes { get; set; } // Таблица типов измерений

// Метод настройки подключения к базе данных

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

=> optionsBuilder.UseSqlServer("ВАША\_СТРОКА\_ПОДКЛЮЧЕНИЯ"); // Используем SQL Server как хранилище данных

// Метод настройки модели с использованием Fluent API

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Для каждой сущности можно настроить отдельные аспекты, такие как связи, индексы, ограничения и т.д.

// Ниже приведено несколько примеров конфигурации сущностей

// Настраиваем свойства в таблице AspNetRole

modelBuilder.Entity<AspNetRole>(entity =>

{

entity.Property(e => e.Name).HasMaxLength(256); // Устанавливаем максимальную длину для имени роли

entity.Property(e => e.NormalizedName).HasMaxLength(256); // Устанавливаем максимальную длину для нормализованного имени роли

});

// Для сущности AspNetUser устанавливается конфигурация атрибутов и связей

modelBuilder.Entity<AspNetUser>(entity =>

{

// Устанавливаются максимальные длины для свойств, связанных с пользователем, для поддержания целостности данных

entity.Property(e => e.Email).HasMaxLength(256);

entity.Property(e => e.NormalizedEmail).HasMaxLength(256);

entity.Property(e => e.NormalizedUserName).HasMaxLength(256);

entity.Property(e => e.UserName).HasMaxLength(256);

// Определение композитного ключа для сущности UserPlace, основанного на двух полях

modelBuilder.Entity<UserPlace>()

.HasKey(uk => new { uk.UserId, uk.PlaceId });

// Установление связи многие-ко-многим между пользователями и местами через сущность UserPlace

modelBuilder.Entity<UserPlace>()

.HasOne(uk => uk.User) // С каждой записью UserPlace связан единственный AspNetUser

.WithMany(u => u.UserPlaces) // У одного AspNetUser может быть множество записей UserPlace

.HasForeignKey(uk => uk.UserId); // Внешний ключ для AspNetUser в UserPlace

modelBuilder.Entity<UserPlace>()

.HasOne(uk => uk.Place) // С каждой записью UserPlace связано единственное место

.WithMany(k => k.UserPlaces) // У одного места может быть множество записей UserPlace

.HasForeignKey(uk => uk.PlaceId); // Внешний ключ для места в UserPlace

// Определение композитного ключа для роли пользователя, основанного на UserID и RoleID

modelBuilder.Entity<AspNetUserRole>()

.HasKey(ur => new { ur.UserId, ur.RoleId });

// Установление связи многие-ко-многим между пользователями и ролями через AspNetUserRole

modelBuilder.Entity<AspNetUserRole>()

.HasOne(ur => ur.User) // С каждой записью AspNetUserRole связан единственный AspNetUser

.WithMany(u => u.UserRoles) // У одного AspNetUser может быть множество записей AspNetUserRole

.HasForeignKey(ur => ur.UserId); // Внешний ключ для AspNetUser в AspNetUserRole

modelBuilder.Entity<AspNetUserRole>()

.HasOne(ur => ur.Role) // С каждой записью AspNetUserRole связана единственная роль

.WithMany(r => r.UserRoles) // У одной роли может быть множество записей AspNetUserRole

.HasForeignKey(ur => ur.RoleId); // Внешний ключ для роли в AspNetUserRole

});

// Настройка сущности пользователя AspNetUser

modelBuilder.Entity<AspNetUser>(entity =>

{

// Установление максимальной длины строки для свойства Email

entity.Property(e => e.Email).HasMaxLength(256);

// Установление максимальной длины строки для свойства NormalizedEmail

entity.Property(e => e.NormalizedEmail).HasMaxLength(256);

// Установление максимальной длины строки для свойства NormalizedUserName

entity.Property(e => e.NormalizedUserName).HasMaxLength(256);

// Установление максимальной длины строки для свойства UserName

entity.Property(e => e.UserName).HasMaxLength(256);

// Определение ключа для связи многие ко многим между пользователями и местами (UserPlaces)

modelBuilder.Entity<UserPlace>()

.HasKey(uk => new { uk.UserId, uk.PlaceId });

// Настройка связи один ко многим между UserPlace и AspNetUser

modelBuilder.Entity<UserPlace>()

.HasOne(uk => uk.User) // Указание навигационного свойства

.WithMany(u => u.UserPlaces) // Указание коллекции на стороне пользователя

.HasForeignKey(uk => uk.UserId); // Установление внешнего ключа

// Настройка связи один ко многим между UserPlace и Place

modelBuilder.Entity<UserPlace>()

.HasOne(uk => uk.Place) // Указание навигационного свойства

.WithMany(k => k.UserPlaces) // Указание коллекции на стороне места

.HasForeignKey(uk => uk.PlaceId); // Установление внешнего ключа

// Определение ключей для сущности AspNetUserRole

modelBuilder.Entity<AspNetUserRole>()

.HasKey(ur => new { ur.UserId, ur.RoleId });

// Настройка связи один ко многим между AspNetUserRole и AspNetUser

modelBuilder.Entity<AspNetUserRole>()

.HasOne(ur => ur.User) // Указание навигационного свойства пользователя

.WithMany(u => u.UserRoles) // Указание коллекции ролей пользователя

.HasForeignKey(ur => ur.UserId); // Установление внешнего ключа пользователя

// Настройка связи один ко многим между AspNetUserRole и AspNetRole

modelBuilder.Entity<AspNetUserRole>()

.HasOne(ur => ur.Role) // Указание навигационного свойства роли

.WithMany(r => r.UserRoles) // Указание коллекции пользователей роли

.HasForeignKey(ur => ur.RoleId); // Установление внешнего ключа роли

});

// Настройка сущности AspNetUserClaim

modelBuilder.Entity<AspNetUserClaim>(entity =>

{

// Установление максимальной длины строки для свойства UserId

entity.Property(e => e.UserId).HasMaxLength(450);

// Настройка связи один ко многим между AspNetUserClaim и AspNetUser

entity.HasOne(d => d.User) // Указание навигационного свойства

.WithMany(p => p.AspNetUserClaims) // Указание коллекции претензий пользователя

.HasForeignKey(d => d.UserId); // Установление внешнего ключа

});

// Настройка сущности AspNetUserLogin

modelBuilder.Entity<AspNetUserLogin>(entity =>

{

// Определение составного ключа из LoginProvider и ProviderKey

entity.HasKey(e => new { e.LoginProvider, e.ProviderKey });

// Установление максимальной длины строк для свойств LoginProvider и ProviderKey

entity.Property(e => e.LoginProvider).HasMaxLength(128);

entity.Property(e => e.ProviderKey).HasMaxLength(128);

entity.Property(e => e.UserId).HasMaxLength(450); // и для UserId

// Настройка связи один ко многим между AspNetUserLogin и AspNetUser

entity.HasOne(d => d.User) // Указание навигационного свойства

.WithMany(p => p.AspNetUserLogins) // Указание коллекции логинов пользователя

.HasForeignKey(d => d.UserId); // Установление внешнего ключа

});

// Настройка сущности AspNetUserToken

modelBuilder.Entity<AspNetUserToken>(entity =>

{

// Определение составного ключа из UserId, LoginProvider и Name

entity.HasKey(e => new { e.UserId, e.LoginProvider, e.Name });

// Установление максимальной длины строк для свойств LoginProvider и Name

entity.Property(e => e.LoginProvider).HasMaxLength(128);

entity.Property(e => e.Name).HasMaxLength(128);

// Настройка связи один ко многим между AspNetUserToken и AspNetUser

entity.HasOne(d => d.User) // Указание навигационного свойства

.WithMany(p => p.AspNetUserTokens) // Указание коллекции токенов пользователя

.HasForeignKey(d => d.UserId); // Установление внешнего ключа

});

// Настройка сущности ClientProfile

modelBuilder.Entity<ClientProfile>(entity =>

{

// Определение первичного ключа

entity.HasKey(e => e.Id).HasName("PK\_\_ClientProfile\_\_3214EC07FDD58F6E");

// Указание таблицы в базе данных

entity.ToTable("ClientProfile");

// Установление максимальной длины строк для свойств

entity.Property(e => e.Address).HasMaxLength(200);

entity.Property(e => e.UserId).HasMaxLength(450);

// Настройка связи один ко многим между ClientProfile и AspNetUser

entity.HasOne(d => d.User) // Указание навигационного свойства

.WithMany(p => p.ClientProfiles) // Указание коллекции профилей клиента

.HasForeignKey(d => d.UserId) // Установление внешнего ключа пользователя

.OnDelete(DeleteBehavior.Cascade) // Указание каскадного удаления

.HasConstraintName("FK\_ClientProfile\_AspNetUsers"); // Наименование внешнего ключа

});

// Настройка сущности Equipment (Оборудование)

modelBuilder.Entity<Equipment>(entity =>

{

// Определяем первичный ключ сущности

entity.HasKey(e => e.Id).HasName("PK\_\_Equipmen\_\_3214EC07700D4081");

// Устанавливаем максимальную длину для модели оборудования

entity.Property(e => e.ModelEq).HasMaxLength(200);

// Настраиваем отношение один ко многим между Place и Equipment

entity.HasOne(d => d.Place).WithMany(p => p.Equipments)

.HasForeignKey(d => d.PlaceId) // Устанавливаем внешний ключ на Place

.OnDelete(DeleteBehavior.ClientSetNull) // При удалении Place, Equipment не удаляется

.HasConstraintName("FK\_Equipment\_Place"); // Называем внешний ключ

});

// Настройка сущности Indication (Показания счетчиков)

modelBuilder.Entity<Indication>(entity =>

{

// Определяем первичный ключ сущности

entity.HasKey(e => e.Id).HasName("PK\_\_Indicati\_\_3214EC074F02BB09");

// Указываем, что сущность соответствует таблице Indication

entity.ToTable("Indication");

// Устанавливаем тип данных и максимальную длину для месяца показания

entity.Property(e => e.Month).HasColumnType("datetime");

// Настраиваем отношение один ко многим между Schetchik и Indication

entity.HasOne(d => d.Schetchik).WithMany(p => p.Indications)

.HasForeignKey(d => d.SchetchikId) // Устанавливаем внешний ключ на Schetchik

.HasConstraintName("FK\_Indication\_Schetchik"); // Называем внешний ключ

});

// Настройка сущности Place (Место)

modelBuilder.Entity<Place>(entity =>

{

// Определяем первичный ключ сущности

entity.HasKey(e => e.Id).HasName("PK\_\_Place\_\_3214EC0772418DFD");

// Указываем, что сущность соответствует таблице Place

entity.ToTable("Place");

// Устанавливаем максимальные длины для различных строковых полей сущности

entity.Property(e => e.Address).HasMaxLength(200);

entity.Property(e => e.Arenda).HasColumnType("datetime"); // Устанавливаем тип данных для даты аренды

entity.Property(e => e.ModelPlace).HasMaxLength(200);

entity.Property(e => e.Name).HasMaxLength(200);

// Настраиваем отношения один ко многим между PlaceSection и Place и между Town и Place

entity.HasOne(d => d.PlaceSection).WithMany(p => p.Places)

.HasForeignKey(d => d.PlaceSectionId) // Внешний ключ на PlaceSection

.HasConstraintName("FK\_Place\_PlaceSection"); // Называем внешний ключ

entity.HasOne(d => d.Town).WithMany(p => p.Places)

.HasForeignKey(d => d.TownId) // Внешний ключ на Town

.HasConstraintName("FK\_Place\_Town"); // Называем внешний ключ

});

OnModelCreatingPartial(modelBuilder); // Вызов частичного метода для возможных дополнительных настроек

}

// Настройка сущности PlaceSection (Секция места)

modelBuilder.Entity<PlaceSection>(entity =>

{

// Определение первичного ключа

entity.HasKey(e => e.Id).HasName("PK\_\_PlaceSec\_\_3214EC071F446306");

// Указание, что сущность отображается на таблицу "PlaceSection"

entity.ToTable("PlaceSection");

// Настройка длины строковых полей

entity.Property(e => e.AdresSection).HasMaxLength(200); // Адрес секции

entity.Property(e => e.Certificate).HasMaxLength(200); // Сертификат

entity.Property(e => e.DataResh).HasColumnType("datetime"); // Дата решения

entity.Property(e => e.DateArenda).HasColumnType("datetime"); // Дата аренды

entity.Property(e => e.Kadastr).HasMaxLength(200); // Кадастровый номер

entity.Property(e => e.PlaceName).HasMaxLength(200); // Название места

entity.Property(e => e.TypeArenda).HasMaxLength(200); // Тип аренды

});

// Настройка сущности Organization (Организация)

modelBuilder.Entity<Organization>(entity =>

{

// Определение первичного ключа

entity.HasKey(e => e.Id).HasName("PK\_\_Organiza\_\_3214EC073C8B27FC");

// Указание, что сущность отображается на таблицу "Organization"

entity.ToTable("Organization");

// Настройка длины строковых полей

entity.Property(e => e.Name).HasMaxLength(200); // Название организации

entity.Property(e => e.Email).HasMaxLength(200); // Электронная почта организации

});

// Настройка сущности Contract (Контракт)

modelBuilder.Entity<Contract>(entity =>

{

// Определение первичного ключа

entity.HasKey(e => e.Id).HasName("PK\_\_Contract\_\_3214EC07EF196410");

// Указание, что сущность отображается на таблицу "Contract"

entity.ToTable("Contract");

// Настройка связей с удалением и внешним ключом

entity.HasOne(d => d.Organization).WithMany(p => p.Contracts) // Связь контракта с организацией

.HasForeignKey(d => d.OrganizationId) // Внешний ключ

.OnDelete(DeleteBehavior.ClientSetNull) // При удалении организации контракт остается

.HasConstraintName("FK\_Contract\_Organization");

entity.HasOne(d => d.Place).WithMany(p => p.Contracts) // Связь контракта с местом

.HasForeignKey(d => d.PlaceId) // Внешний ключ

.OnDelete(DeleteBehavior.ClientSetNull) // При удалении места контракт остается

.HasConstraintName("FK\_Contract\_Place");

});

// Настройка сущности Schetchik (Счетчик)

modelBuilder.Entity<Schetchik>(entity =>

{

// Определение первичного ключа

entity.HasKey(e => e.Id).HasName("PK\_\_Schetchi\_\_3214EC07CC0F6E20");

// Указание, что сущность отображается на таблицу "Schetchik"

entity.ToTable("Schetchik");

// Настройка длины строковых полей и типов данных

entity.Property(e => e.ModelSchetchika).HasMaxLength(200); // Модель счетчика

entity.Property(e => e.NomerSchetchika).HasMaxLength(200); // Номер счетчика

entity.Property(e => e.Poverka).HasColumnType("datetime"); // Дата поверки

// Настройка связей с удалением и внешним ключом

entity.HasOne(d => d.Place).WithMany(p => p.Schetchiks) // Связь счетчика с местом

.HasForeignKey(d => d.PlaceId) // Внешний ключ

.OnDelete(DeleteBehavior.Cascade) // Каскадное удаление счетчиков при удалении места

.HasConstraintName("FK\_Schetchik\_Place");

entity.HasOne(d => d.MeasureType).WithMany(p => p.Schetchiks) // Связь счетчика с типом измерения

.HasForeignKey(d => d.MeasureTypeId) // Внешний ключ

.OnDelete(DeleteBehavior.Cascade) // Каскадное удаление счетчиков при изменении типа измерения

.HasConstraintName("FK\_Schetchik\_MeasureType");

});

// Настройка сущности Town (Город)

modelBuilder.Entity<Town>(entity =>

{

// Определение первичного ключа

entity.HasKey(e => e.Id).HasName("PK\_\_Town\_\_3214EC077070A904");

// Указание, что сущность отображается на таблицу "Town"

entity.ToTable("Town");

// Настройка длины строковых полей

entity.Property(e => e.Name).HasMaxLength(200); // Название города

});

// Настройка сущности MeasureType (Тип измерения)

modelBuilder.Entity<MeasureType>(entity =>

{

// Определение первичного ключа

entity.HasKey(e => e.Id).HasName("PK\_\_MeasureType\_\_3214EC088080A803");

// Указание, что сущность отображается на таблицу "MeasureType"

entity.ToTable("MeasureType");

// Настройка длины строковых полей

entity.Property(e => e.Name).HasMaxLength(200); // Название типа измерения

});

// Частичный метод для дополнительной конфигурации моделей

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

}

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | | | | *Дополнительные сведения* | |
|  | | | | | Текстовые документы | | | |  | |
|  | | | | |  | | | |  | |
| БГУИР ДП 1–40 01 01 01 016 ПЗ | | | | | Пояснительная записка | | | | 72 с. | |
|  | | | | | Отзыв руководителя | | | |  | |
|  | | | | | Рецензия | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | | Графические документы | | | |  | |
|  | | | | |  | | | |  | |
| ГУИР.991051-01 СА | | | | | Диаграмма последовательности | | | | Формат А1 | |
|  | | | | | распечатки показаний | | | |  | |
|  | | | | | Схема алгоритма | | | |  | |
| ГУИР. 991051-01  СП | | | | | Диаграмма классов. | | | | Формат А1 | |
|  | | | | | Схема программы | | | |  | |
| ГУИР. 991051-01  СД | | | | | Схема база данных. | | | | Формат А1 | |
|  | | | | | Схема данных | | | |  | |
| ГУИР. 991051-01 ПЛ | | | | | Диаграмма классов. | | | | Формат А1 | |
|  | | | | | Плакат | | | |  | |
|  | | | | | Схема работы модуля прогнозиро | | | |  | |
| ГУИР. 991051-02  ПЛ | | | | | вания | | | | Формат А1 | |
|  | | | | | Плакат | | | |  | |
| ГУИР. 991051-03  ПЛ | | | | | Результаты проверки работоспо- | | | | Формат А1 | |
|  | | | | | собности. | | | |  | |
|  | | | | | Плакат | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  |  |  |  |  | *БГУИР ДП 1-40 01 01 01 016 Д1* | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* | *Программное средство учета коммунальных услуг на предприятии.*  *Ведомость дипломного  проекта* | *Литера* | | | *Лист* | *Листов* |
| *Разраб.* | | *Симакович И.А.* |  | *28.05* |  | *Т* |  | *72* | *72* |
| *Пров.* | | *Мелких Е.Г.* |  | *28.05* | *Кафедра ПОИТ*  *гр. 991051* | | | | |
| *Т.контр.* | | *Марченкова Е.Е.* |  | *28.05* |
| *Н.контр.* | | *Колотыгина Ю.Ю.* |  | *28.05* |
| *Утв.* | |  |  | *10.06* |