МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки, специальности)

Разработка программного обеспечения
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка программного обеспечения для распределения и сбыта тепловой энергии теплоснабжающей организации»

Обучающийся М.Б. Заколюкин _____
(Инициалы Фамилия) (личная подпись)

Руководитель Н.Н. Казаченок
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

Тема выпускной квалификационной работы: «Разработка программного обеспечения для распределения и сбыта тепловой энергии теплоснабжающей организации»

Объектом исследования - процессы являются процессы распределения и сбыта тепловой энергии теплоснабжающей организации

Цель работы – разработка программного обеспечения для распределения и сбыта тепловой энергии теплоснабжающей организации с помощью программ Microsoft Visual Studio распределения и сбыта тепловой энергии теплоснабжающей организации АО «ОЭК».

Предметом исследования является автоматизация процессов распределения и сбыта тепловой энергии теплоснабжающей АО «ОЭК».

Бакалаврская работа состоит из введения, трех глав, заключения и списка литературы.

Во введении бакалаврской работы обосновывается выбор темы, потребности в автоматизации процессов распределения и сбыта тепловой энергии. Формулируются цели и задачи.

В первой главе исследуется предметная область программного обеспечения, инструментов для автоматизации бизнес-процессов в теплоснабжающих организациях.

Вторая глава рассматривает архитектуру программы, раскрывается описание классов и методов работы.

Третья глава рассматривает реализацию приложения на операционной системе Windows с целью оценки его функциональности, производительности и совместимости, оптимизации базы данных, тестированием.

В заключении представлены выводы по проделанной работе.

Оглавление

Введение

С развитием общества росла потребность в широком обмене информацией. Работа человека с информацией при использовании компьютерных технологий предъявляет новые требования к обществу, и, в связи с этим развивается информатизация.

Создание программного продукта в сфере сбыта тепловой энергии позволяет в значимой мере экономить ресурсы компании благодаря автоматизации вычислительных процессов. Сбор и хранение аналитических данных поможет спрогнозировать рост и падение цен, что позволяет найти оптимальный подход.

Так что создание и существование такого рода программы многим упрощает работу. Тем более в нашем современном мире, мире технологий, это крайне необходимо.

Актуальность темы бакалаврской работы заключается в том, что с ростом технологического прогресса появилась потребность автоматизации различных вычислений в сфере бизнеса, что позволяет снизить количество затрачиваемого времени, что позволяет инвестировать сэкономленное время в другие модули компании.

Цель работы – разработка программного обеспечения для распределения и сбыта тепловой энергии теплоснабжающей организации с помощью программ Microsoft Visual Studio 2022 на языке программирования C#.

Объектом исследования являются процессы распределения и сбыта тепловой энергии теплоснабжающей организации

Предметом исследования является автоматизация процессов распределения и сбыта тепловой энергии теплоснабжающей организации

Для достижения поставленной цели необходимо выполнить следующие задачи:

- собрать необходимые данные из открытых источников по разработке ПО;
- определиться с методологией (архитектурой) будущего проекта;
- разработать прототип ПО опираясь на методологию;
- разработать базу данных;
- протестировать проект;
- рассмотреть виды интегрированных сред разработок;

Бакалаврская работа состоит из введения, трех глав, заключения и списка литературы.

Во введении бакалаврской работы обосновывается выбор темы, потребности в автоматизации процессов распределения и сбыта тепловой энергии. Формулируются цели и задачи.

В первой главе исследуется предметная область программного обеспечения, инструментов для автоматизации бизнес-процессов в теплоснабжающих организациях.

Вторая глава рассматривает архитектуру программы, раскрывается описание классов и методов в работе программы

Третья глава рассматривает реализацию программы и ее интерфейса с тестированием.

В заключении представлены выводы по проделанной работе.

Выпускная квалификационная работа состоит из 54 страниц и содержит 26 рисунков, 4 таблицы, 20 источников.

Глава 1 Теоретические аспекты разработки ПО

1.1 Анализ предметной области

Программное обеспечение (ПО) – это программа или совокупность программ для продуктивного управления компьютером. В контексте данной работы основной задачей разработки ПО является автоматизация бизнес-процессов. Это закладывает основы для эффективных затрат рабочего времени сотрудников, расходов на выполнение работ и построения логического отлаженного алгоритма работы всех направлений и подразделений предприятий.

Информационная система распределения и сбыта тепловой энергии позволит автоматизировать и оптимизировать процессы управления теплоснабжением, такие как планирование, производство и распределение, что снижает затраты и повышает эффективность компании.

Теплоснабжающая организация – организация, осуществляющая продажу потребителям и (или) теплоснабжающим организациям произведенных или приобретенных тепловой энергии (мощности), теплоносителя и владеющая на праве собственности или ином законном основании источниками тепловой энергии и (или) тепловыми сетями в системе теплоснабжения, посредством которой осуществляется теплоснабжение потребителей тепловой энергии (данное положение применяется к регулированию сходных отношений с участием индивидуальных предпринимателей)

Российская система теплоснабжения является самой большой в мире. На ее долю приходится более 40% мирового централизованного производства тепловой энергии.

По состоянию на 1 января 2024 года, в Единой энергосистеме России эксплуатировались тепловые электростанции общей установленной

мощностью 163 712 МВт, что составляет 65,98 % от общей мощности электростанций ЕЭС России [14] (таблица 1).

Таблица 1 – Сведения установленной мощности ТЭС

Энергосистема	Всего,МВт	ТЭС	
		МВт	%
ЕЭС России	248 164,88	163 711,96	65,98
ОЭС Центра	50 439,23	34 840,88	69,07
ОЭС Ср. Волги	28 013,08	16 625,18	59,35
ОЭС Урала	53 317,61	49 413,28	92,68
ОЭС Северо-Запада	25 139,55	15 821,87	62,94
ОЭС Юга	27 666,71	13 816,67	49,94
ОЭС Сибири	52 376,81	26 599,69	50,83
ОЭС Востока	11 211,89	6 594,39	58,82

Важным аспектом функционирования информационной системы распределения и сбыта тепловой энергии является интеграция с современными технологиями. Автоматизация процессов не только оптимизирует управление ресурсами, но и способствует своевременному реагированию на изменения в потребительском спросе, что особенно актуально в условиях нестабильного рынка и изменяющихся климатических условий [3].

Внедрение информационных технологий в теплоснабжающих организациях открывает новые горизонты для развития умных сетей. Такие системы позволяют более эффективно управлять потоками энергии, минимизировать потери при транспортировке и распределении, а также улучшать качество обслуживания потребителей.

На основе полученных сведений можно сделать вывод, что разработка программного обеспечения в данной сфере поможет предотвратить

значительные траты времени и повысить качество услуг. Интеграция автоматизированных решений позволит не только оптимизировать внутренние процессы, но и улучшить взаимодействие с клиентами. Это будет способствовать повышению удовлетворенности потребителей и укреплению их доверия к теплоснабжающим организациям.

1.2 Основные требования к приложению

Программа выполняет функции по управлению заявками и складскими данными, предоставляя пользователю интерфейс для взаимодействия с информацией.

Функциональность построена на базе командного интерфейса, который позволяет инициировать различные действия по созданию новых заявок и управление складскими операциями.

Синхронизация данных между пользовательским интерфейсом и базой данных. Программа запрашивает, обновляет и удаляет данные в базе данных, что позволяет всегда показывать актуальную информацию на экране.

Локальное хранение данных дает возможность работать с приложением автономно, без необходимости подключения к внешним серверам, что делает программу автономной для использования.

Автоматически обновляет интерфейс, когда изменяются данные в базе. Каждое добавлении, изменении или удалении записей информация мгновенно отражается на экране, и пользователь всегда видит актуальные данные.

Управление ресурсами, предоставляя функции для добавления новых данных, редактирования информации о ресурсах и удаления ненужных записей. Расширяет возможности приложения и складскими операциями.

Создания позволяет пользователю добавлять новые записи о заявках и операциях. С помощью этой операции можно легко заносить в систему новую информацию, детали о новом клиенте, контактные данные, описание

ресурса или сведения о его поставщике. Ввод данных происходит через диалоговые окна.

Обновления делает доступной всю сохраненную информацию, представляя данные в удобном для просмотра формате. В программе пользователи могут видеть списки текущих заявок и складских позиций, с возможностью быстрого поиска. Это позволяет оперативно получать актуальную информацию и контролировать состояние заявок или ресурсы на складе.

Изменение обеспечивает возможность внесения изменений в уже существующие записи. Пользователи могут легко редактировать любую из заявок или складских записей, обновляя, например, статус заявки, контактные данные или детали ресурсов. Обновление данных сразу отражается в пользовательском интерфейсе, что поддерживает актуальность отображаемой информации.

Удаления позволяет пользователям очищать базу данных от устаревших или ненужных записей. Функция убирает заявки или складских позиции, помогает освобождать место для новых записей.

1.3 Выбор IDE

При проектировании программного обеспечения был сделан выбор в пользу программных продуктов, которые уже зарекомендовали себя при выполнении аналогичных задач.

Учитывались факторы совместимости и интеграции с существующими системами. Предпочтение следует отдавать продуктам с активной поддержкой и обновлениями, что гарантирует своевременное исправление ошибок и внедрение новых функций [2].

Разработка программного обеспечения велась на базе модульной структуры с применением объектно-ориентированного метода, что позволяет

легко расширять или модифицировать функционал (Таблица 2 - Рейтинг IDE на 2024 год).

Таблица 2 - Рейтинг IDE на 2024 год

Область применения	IDE	Рейтинг %
Разработка ПО	Visual Studio	28.31 %
Веб-разработка	Visual Studio Code	13.7 %
Корпоративные системы	Eclipse	13.7 %
Научные приложения	pyCharm	10.72 %
Мобильная разработка	Android Studio	9.78 %
Java разработка	IntelliJ	7.5 %
Учебные проекты	NetBeans	3.97 %
Разработка под Apple	Xcode	2.99 %
Статистический анализ	RStudio	2.88 %
Редактирование	Sublime Text	2.46 %

Рассмотрим наиболее популярные IDE

Microsoft Visual Studio интегрированная среда разработки (IDE), предлагающая различные. Эта среда позволяет разработчикам создавать широкий спектр проектов, включая мобильные приложения, веб-приложения и даже видеоигры. Поддерживает множество инструментов для тестирования, что значительно упрощает процесс отладки и проверки кода. Кроме того, она предлагает поддержку различных языков программирования, таких как C#, C++, Python и другие, что делает её универсальным инструментом для разработчиков.

Гибкость Microsoft Visual Studio делает её идеальным выбором как для студентов, которые только начинают свой путь в программировании, так и для опытных профессионалов, работающих над сложными проектами. С помощью данной IDE разработчики могут эффективно управлять проектами, следить за версиями кода и сотрудничать с другими членами команды, что

способствует повышению продуктивности и качеству разрабатываемого программного обеспечения.

Visual Studio является основным инструментом для разработки WPF-приложений на C# и .NET. Эта интегрированная среда разработки предлагает редактор кода, визуальный дизайнер интерфейсов и мощные средства для отладки и тестирования. Visual Studio имеется визуальный дизайнер, позволяющий перетаскивать элементы управления и настраивать их без необходимости ручного написания XAML, что значительно ускоряет процесс прототипирования [11].

Пошаговая отладка позволяет тщательно отслеживать выполнение кода, находить логические ошибки и анализировать состояние переменных. Визуальные инструменты, такие как Live Visual Tree и Live Property Explorer, дают возможность исследовать и изменять визуальные компоненты интерфейса в реальном времени, что особенно полезно для отладки сложных пользовательских интерфейсов в WPF.

Xcode – это функциональная среда разработки, предназначенная для создания приложений под продукты Apple. Данная IDE подходит как для индивидуальных разработчиков, так и для корпоративных команд, обеспечивая все необходимые инструменты для эффективного программирования. Публикации созданного приложения в App Store необходимо приобрести лицензию разработчика, что позволяет получить доступ к различным ресурсам и инструментам от Apple. Xcode предлагает удобный интерфейс, инструменты для отладки, а также возможности для тестирования и оптимизации приложений [16].

Eclipse – это бесплатная и открытая среда разработки, предоставляет пользователям инструменты для отладки, а также интеграцию с системами управления версиями, такими как Git и CVS. В стандартной комплектации Eclipse включает поддержку языка Java и инструмент для создания плагинов. Появлением множества плагинов и расширений его функциональность значительно возросла. Эта возможность адаптации и добавления новых

модулей стала одной из причин, по которой Eclipse завоевал популярность среди разработчиков [19].

IntelliJ IDEA – это еще одна интегрированная среда разработки, созданная компанией JetBrains. Пользователи могут выбрать между бесплатной версией Community Edition и платной версией, разработчики позиционируют эту среду как «самую умную и удобную IDE для Java», обеспечивающую поддержку всех современных технологий и фреймворков.

Программирование в нашей среде осуществляется на языке C#, который представляет собой строго типизированный объектно-ориентированный язык.

Язык компилируется в промежуточный язык Intermediate Language, который затем выполняется на платформе .NET. Одним из преимуществ этого подхода является то, что скомпилированный код не зависит от конкретной операционной системы или аппаратного обеспечения, что обеспечивает его работоспособность на любом устройстве с установленной средой выполнения .NET.

Важной чертой платформы .NET является продвинутая система безопасности. Она позволяет детально контролировать выполнение приложений, ограничивая доступ к определенным ресурсам и предотвращая несанкционированные операции. Если программа пытается выполнить действия, выходящие за рамки ее разрешений выполнение программы немедленно прерывается, что повышает уровень безопасности и защищает данные пользователя. [10].

1.3.1 Инструменты дизайна и проектирования

Создание удобных интерфейсов являются важнейшей задачей в разработке программного обеспечения. Специальные инструменты позволяют разработчикам визуализировать свои идеи, тестировать их на реальных пользователях и адаптировать в соответствии с полученными отзывами.

Главное преимущество заключается в облачной архитектуре, что позволяет нескольким членам команды работать над проектом одновременно и в реальном времени. Это особенно удобно для распределенных команд, поскольку все изменения синхронизируются мгновенно.

Преимущества:

- облачная платформа позволяет нескольким пользователям работать одновременно над проектом в реальном времени;
- работает в браузере и поддерживает разные операционные системы.

Недостатки:

- зависимость от интернета, возможно подключиться только при наличии интернет-соединения;
- ограниченная доступность без интернета.

Sketch создан исключительно для пользователей macOS. Он предлагает интуитивно понятный интерфейс и обширный набор функций, ориентированных на десктоп дизайне. Высокая скорость работы и возможности создания позволяют легко разрабатывать повторяющиеся элементы интерфейса: кнопки, иконки или меню.

Преимущества:

- интуитивный интерфейс для работы с графическим интерфейсом;
- быстрая работа приложения.

Недостатки:

- работает исключительно на MacOS;
- ограниченные возможности для совместной работы в реальном времени.

1.3.2 Определение СУБД

СУБД выступает в роли посредника между пользователем и базой данных, обеспечивая упорядоченный доступ к информации. Предоставляет конечным пользователям и прикладным программам единое интегрированное представление данных, что упрощает процессы поиска,

обновления и управления информацией в базе данных. Включает инструменты для обеспечения безопасности, резервного копирования и восстановления, что делает её важным компонентом в современных информационных системах [13].

Изображен рейтинг популярности СУБД на 2024 год по версии DB-Engine (тТаблица 3 - Рейтинг СУБД по версии DB-Engines) [7].

Таблица 3 - Рейтинг СУБД по версии DB-Engines

СУБД	Модель СУБД	Рейтинг
Oracle	Реляционная	1240.37
MySQL	Реляционная	1039.46
Microsoft SQL Server	Реляционная	807.65
PostgreSQL	Реляционная	638.91
MongoDB	Документооборотная	429.83
Redis	Ключ-значение	156.77
Snowflake	Реляционная	136.53
Elasticsearch	Поисковая система	130.82
IBM Db2	Реляционная	124.40
SQLite	Реляционная	109.95

SQL (Structured Query Language), предоставляет гибкость и простоту в использовании благодаря своим мощным инструментам для работы с данными. Пользователи могут создавать сложные запросы для поиска и фильтрации информации, объединять данные из нескольких таблиц, а также управлять структурой, добавлять новые таблицы, изменять существующие и устанавливать связи между ними. SQL инструмент для разработчиков и аналитиков, позволяя эффективно манипулировать и получать нужные результаты с минимальными усилиями.

SQL разработан с использованием технологии клиент-сервер. В этой архитектуре запросы пользователя обрабатываются на специализированных серверах. Серверы, в свою очередь, выполняют необходимые операции и возвращают только результаты запросов на клиентские компьютеры. Это позволяет минимизировать нагрузку на клиентские устройства, улучшая общую производительность системы. [5].

Приложения функционирует через интерактивное программное обеспечение, работает автономно и предназначена для непосредственного взаимодействия. Пользователь вводит команды, и результат отображается на экране сразу после их выполнения. Работы в реальном времени обеспечивает быстрый отклик и позволяет пользователю оперативно анализировать данные, делая процесс взаимодействия более интуитивным и визуально наглядным [6].

Встроенный режим предполагает использование команды в код программы, написанной на другом языке программирования. SQL выполняет роль инструмента для управления данными, встроенного прямо в логику программы. Такой подход облегчает работу в рамках более крупных приложений, обеспечивая эффективное взаимодействие между кодом и данными. [18]

Язык SQL состоит из следующих компонентов:

- DML (Data Manipulation Language, язык изменения данных) этот компонент содержит команды, которые позволяют пользователям управлять данными;
- DCL (Data Control Language, язык управления данными) этот язык отвечает за управление доступом пользователей к данным. Команды DCL, используются для определения прав доступа и управления разрешениями на взаимодействие с определенными объектами базы данных;
- DDL (Data Definition Language, язык описания данных) это язык, который используется для описания структуры данных. Он состоит

из набора команд, которые создают, изменяют и удаляют объекты базы данных, такие как таблицы, индексы и представления.

Информация рассматривается как широкое понятие, конкретные значения или характеристики, относящиеся к определённым объектам или явлениям. Такое различие играет ключевую роль при разработке базы данных, поскольку корректное понимание этих терминов обеспечивает успешное создание и управление структурой данных [12] (Рисунок 1 - Схема отношений БД).

Рисунок 1 - Схема отношений БД

Универсальность и простота в использовании делают его незаменимым для разработчиков и аналитиков, обеспечивая эффективное взаимодействие с базами данных. Благодаря различным компонентам SQL, таким как DML, DCL и DDL, пользователи могут не только манипулировать данными, но и управлять доступом и структурой базы данных.

Для реализации системы управления базой данных (СУБД) был выбран SQLite, поскольку он представляет собой легковесное и эффективное решение, не требующее отдельного серверного окружения для работы. Идеально подходит для встроенных приложений и локального хранения данных.

1.3.3 Выбор языка

На выбор языка программирования и реализации разработанного алгоритма влияют много факторов, такие как функционал, способность при заданных условиях удовлетворять потребности, понятный интерфейс. Для определения языка программирования необходимо провести анализ основных возможностей языка. В таблице представлен рейтинг языков программирования по версии TIOBE в 2024 году [9]. TIOBE – это индекс, который оценивает популярность языков программирования, основываясь на количестве поисковых запросов, содержащих название языка (таблица 4).

Таблица 4 - Индекс языков программирования 2024 года

Язык программирования	Рейтинг	Относительное изменение, %
Python	15.39%	+2.93%
C++	10.03%	-1.33%
C	9.23%	-3.14%
Java	8.40%	-2.88%
C#	6.65%	0.06%
JavaScript	3.32%	+0.51%
Go	1.93%	+0.93%
SQL	1.75%	+0.28%
Visual Basic	1.66%	-1.67%
Fortran	1.53%	+0.53%

При выборе оптимального языка программирования для решения поставленной задачи необходимо рассмотреть все особенности, достоинства и недостатки.

Язык программирования Python – это высокоуровневый и интерпретируемый язык программирования, который был создан Гвидо Ван Россумом в 1989 году и выпущен в 1991 году. За последние несколько лет

Python стал использоваться в разработке современного программного обеспечения, управления инфраструктурой, анализа данных и машинного обучения. Его используют в создании веб-приложений и управлении системами. Синтаксис Python разработан таким образом, чтобы он был читабельным и простым. В результате разработчики тратят больше времени на размышления о проблеме, которую они пытаются решить, и меньше времени на размышления о сложности языка или расшифровку кода, составленного другими [20].

C++ – это один из наиболее распространенных языков программирования, широко используемый в индустрии разработки программного обеспечения. Данный язык был разработан в 1979 году Бьерном Страуструпом, который позже получил за него множество наград. C++ в свое время стал улучшением языка C, который был создан в 70-х годах XX века и до сих пор активно используется в программировании. C++ выразительнее языка C и предлагает удобный способ создания типизированных структур данных, а также классов и динамических объектов.

Существуют множество доказательств того, что C++ является языком высокого уровня.

Во-первых, C++ предоставляет множество возможностей для абстракции, которые позволяют программистам работать на более высоком уровне абстракции, чем в случае с C. C++ предлагает более выраженные функции с такими возможностями, как наследование и многоуровневые иерархии.

Во-вторых, C++ является объектно-ориентированным языком программирования, что позволяет создавать объекты, которые могут иметь свойства и методы, а также могут быть такими же разнообразными, как и объекты реального мира. Объекты также могут быть унаследованы от других, и таким образом, создание иерархий классов может быть легко произведено.

В-третьих, C++ обладает широким выбором библиотек и фреймворков, которые помогают ускорить процесс разработки. Эти библиотеки обладают множеством возможностей, которые предоставляются от утилит обработки изображений до поддержки создания приложений в графическом интерфейсе пользователя (GUI). Несмотря на то, что C++ имеет множество преимуществ, этот язык также имеет свои недостатки. Например, C++ язык сложный для изучения и является достаточно медленным при исполнении кода по сравнению с другими языками программирования. Тем не менее, C++ – это достаточно популярный язык для написания высококачественного программного обеспечения [8].

Java используется во многих сферах. На ней пишут: приложения для Android – Java практически единственный язык для них; десктопные приложения; промышленные программы; банковские программы; научные программы; программы для работы с Big Data; веб-приложения, веб-сервера, сервера приложений; встроенные системы – от маленьких чипов до специальных компьютеров; корпоративный софт.

C# – это объектно-ориентированный язык программирования, произносится как C-Sharp. Этот язык был разработан в 2001 году, для конкуренции с Java Microsoft под руководством Андерса Хейлберга и его команды. Всего в языке C# используется 86 ключевых слов [17].

C# обеспечивает поддержку компонентно-ориентированного программирования. Структура современного программного обеспечения все в большей мере основывается на независимых модулях, содержащих свое описание. Суть этих компонентов в том, что они являются частью модели программирования, основанной на свойствах, методах и событиях, имеют атрибуты, обеспечивающие описательную информацию о компонентах, и они самодокументированы. C# обеспечивает языковые конструкции, непосредственно поддерживающие эти концепции, что делает его очень естественным языком для создания и применения компонентов программного обеспечения.

Go язык программирования нацелен на разработку больших программ, который можно использовать для разработки очень больших приложений и их компиляции за секунды на единственном компьютере. Язык Go создавался для более легкого и простого написания надёжных высококачественных программ. Go – практичный язык, в основу которого положены эффективность программ и удобство программиста [1].

Visual Basic является языком программирования третьего поколения, который отчасти унаследовал синтаксис языка Бейсик, также это современный язык программирования, сочетающий процедуры и элементы объектно-ориентированных и компонентно-ориентированных языков программирования. Данный язык встроен во многие приложения, а именно в пакетах приложений Microsoft Office [4].

Язык программирование Fortran (FormulaTranslation), созданный в 50-х годах прошлого века, вошёл в историю, как первый полностью реализованный язык высокого уровня, совершил маленький прорыв в IT-мире, но был не слишком удобен и функционален. Пережил минимум 10 обновлений. Он по-прежнему остаётся актуальным. Столь завидное долголетие языку во многом обеспечили простота и накопление значительного объёма данных. Входит в 30 самых популярных языков программирования. Компиляторы для Фортана за его долгую историю разрабатывались такими компаниями, как IBM, Microsoft, Compaq, HP, Oracle, благодаря чему сегодня язык совместим с Windows, Mac OS и Linux. Более того, удобное приложение с компилятором теперь можно взять с собой, благодаря приложению CCTools для Android.

Сегодня Fortran используется в различных областях наук, в том числе для решения математических и физических задач, для разного рода научных и инженерных расчетов, полностью или частично применяемый для прогноза погоды, океанографии, молекулярной динамики, сейсмологического анализа. Fortran жестко-стандартизированный язык, который без проблем переносится на различные платформы [15].

Для реализации проекта был выбран язык программирования C#. К его преимуществам можно отнести:

– простой, надежный, масштабируемый;

– структурированный;

– поддерживает совместимость языков;

– большая стандартная библиотека;

– легко интегрировать с БД.

Выводы по первому разделу

В процессе написания раздела проведен анализ предметной области для эффективного распределения и сбыта тепловой энергии.

Изложены основные требования к приложению. Рассмотрены и выбраны аналоги программного обеспечения с целью проектирования и разработки приложения.

Глава 2 Проектирование программного обеспечения

2.1 Методология проектирования программного обеспечения и описание основных классов.

Проектирование программного обеспечения процесс, который требует тщательного планирования и применения различных методологических подходов. Одним из наиболее эффективных способов визуализации системы является использование диаграмм UML. Эти диаграммы помогают лучше понять структуру, поведение и взаимодействие компонентов программного обеспечения.

В рамках проектирования программного обеспечения использованы несколько диаграмм, каждая из которых выполняет свою уникальную роль в визуализации и анализе системы.

В контексте проектирования программного обеспечения важно отметить, что применение этих диаграмм не ограничивается только этапом проектирования. Они могут использоваться на всех этапах разработки программного обеспечения. Использование диаграмм упрощает понимание системы и взаимодействий между ее компонентами, что важно в условиях работы.

2.1.1 Диаграмма прецедентов

Диаграмма прецедентов показывает взаимодействие пользователей с системой и основные функции, которые они могут выполнять. Каждый прецедент представляет собой задачу, которую пользователь выполняет, взаимодействуя с графическим интерфейсом приложения. При выборе конкретного действия приложение обращается к базе данных для сохранения данных или получения актуальной информации (

Рисунок 2 – Диаграмма прецедентов)

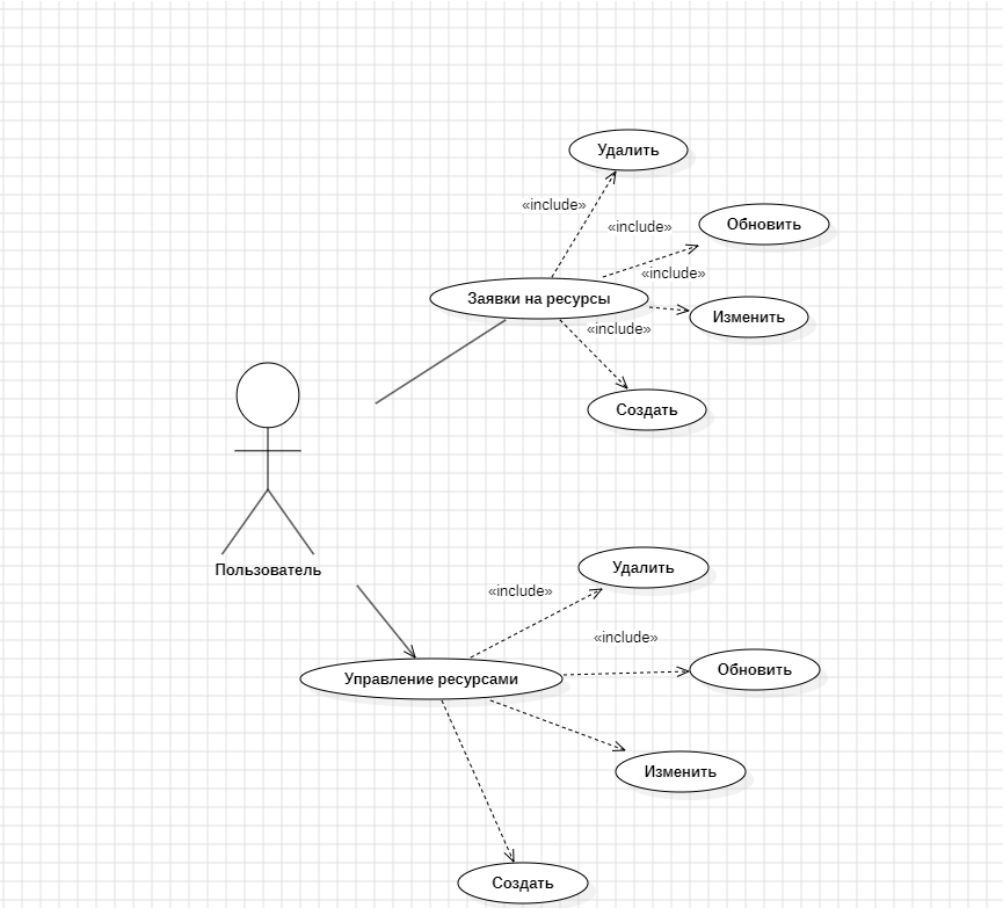


Рисунок 2 – Диаграмма прецедентов

Диаграмма будет фокусироваться на взаимодействии пользователя с различными функциями интерфейса Заявками и управлениями ресурсами. Операции CRUD выполняют функцию взаимодействия с интерфейсом.

Основным актором будет пользователь, который взаимодействует с интерфейсом. Он взаимодействует с приложением через интерфейс, инициируя команды создание заявки или управления ресурсами.

2.1.2 Диаграмма деятельности

Диаграмма деятельности представляет собой важный инструмент для моделирования рабочих процессов и последовательности действий в системе. Она позволяет отобразить логику процессов, которые происходят в рамках системы. улучшению Эта диаграмма способствует понимания процессов и выявлению потенциальных узких мест или избыточных этапов

Одной из ключевых особенностей диаграммы деятельности является ее способность визуализировать параллельные и альтернативные потоки выполнения действий. Это особенно полезно для сложных систем, где несколько процессов могут выполняться одновременно или могут зависеть друг от друга. Благодаря этому можно определить, как оптимизировать работу отдельных процессов, минимизируя время и ресурсы.

Основные потоки действий начинаются с взаимодействия пользователя, который инициирует команду. Эти команды запускают процессы в приложении, и диаграмма деятельности помогает визуализировать весь процесс от начала до завершения (рисунок 3).

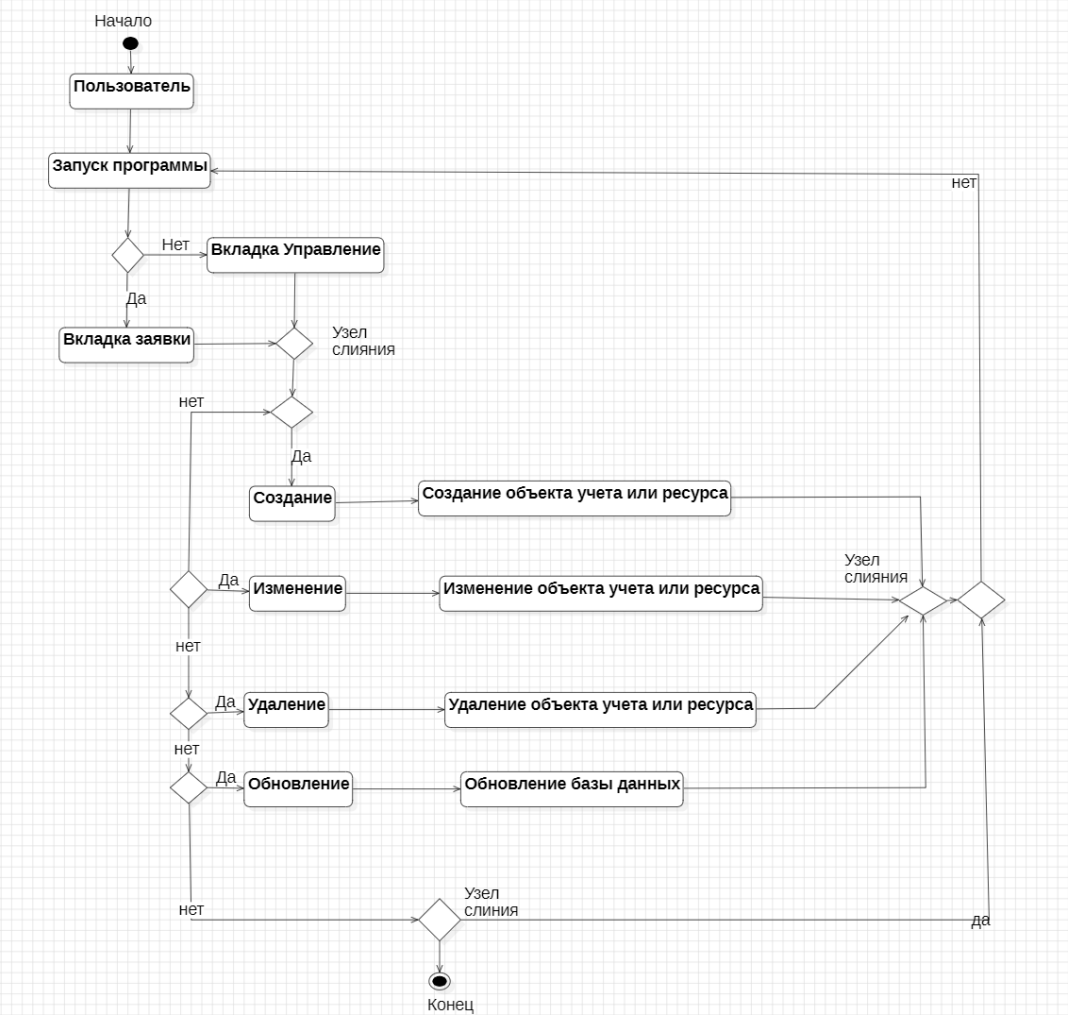


Рисунок 3 – Диаграмма деятельности

Диаграмма включает в себя основные этапы пользователя, запуска программы, CRUD операций и основные действия управление, заявки. При запуске программы пользователь должен выбрать из узла, по которому пути дальше пойдет. Узел слияния объединяет два альтернативные пути вместе, чтобы исполнить операции CRUD. Каждое действие связанно с условным оператором для выбора типа операций, которые попадают в конечный узле для завершения или запуска программы.

2.1.3 Диаграмма развёртывания

Диаграмма развёртывания описывает физическое размещение компонентов программного обеспечения на аппаратных узлах системы. Она помогает визуализировать пользовательский интерфейс, базу данных. Эта диаграмма полезна для определения инфраструктуры развертывания системы, чтобы участники проекта могли понять, где и как выполняются различные части программного обеспечения (рисунок 4).

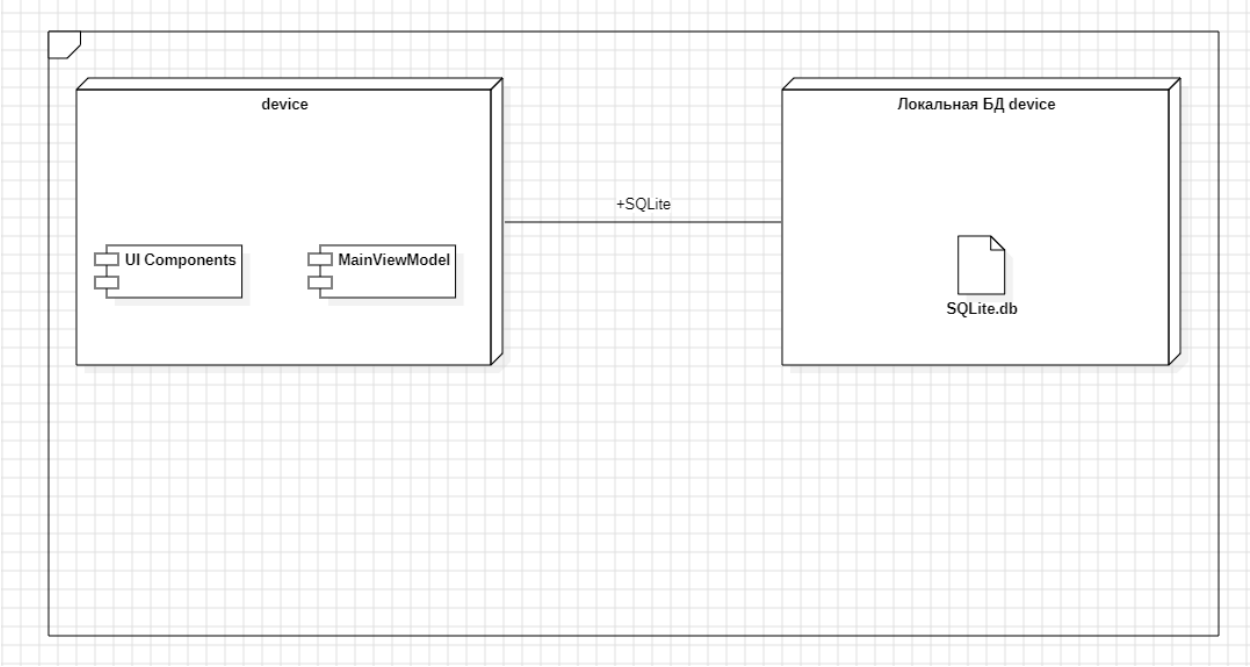


Рисунок 4 – Диаграмма развёртывания

Диаграмма включает клиентский узел, на котором выполняется само приложение, и компонент базы данных. В этой конфигурации приложение выступает в роли клиента, который предоставляет пользовательский интерфейс и доступ СУБД. Клиентский узел работает с логикой, которая обрабатывает пользовательские команды и взаимодействует с SQLite.

2.1.4 Диаграмма классов

При разработке программного обеспечения можно выделить несколько ключевых классов, которые помогут организовать структуру проекта и обеспечить его функциональность. Вот примерный набор классов и их содержимое:

Класс MainViewModel является главным Data Context приложением, отвечает за связь с базой данных и первичной инициализацией данных.

MainViewModel играет ключевую роль в архитектуре для приложений, разработанных на платформе .NET. Основная функция этого класса служит мостом между моделью данных и представлением, обеспечивая управление состоянием пользовательского интерфейса и обработку логики взаимодействия с пользователем.

MainViewModel содержит в себе модель взаимодействия с данными в информационных системах CRUD, представляющая собой набор базовых операций для управления данными в хранилищах. Эти операции охватывают полный цикл работы с сущностями данных, применяются в рамках систем управления базами данных СУБД.

Операции CRUD:

- Create добавление новой записи в хранилище данных, предполагает формирование новой сущности на основе предоставленных данных;
- Read направленная на извлечение данных, создана для получения данных для дальнейшего использования или отображения;
- Update предназначенная для изменения данных, модифицирует значений атрибутов. Операция обновления требует

предварительного чтения записи для её идентификации и корректного изменения;

– Delete полностью удаляет из системы сущность атрибута. Удаление может быть как физическим, так и логическим.

Атрибуты класса Create_Or_Update_ViewModel:

– ID: Идентификатор заявки;
– Address_object, представляющее адрес объекта;
– Applicant, содержащее имя заявителя;
– Number_Phone, представляющее контактный телефон заявителя;
– Description, для краткого описания заявки;
– Priority, указывающее приоритет заявки;
– Date_Time_Reg, представляющее дату и время регистрации
– State_Status, описывающее состояние заявки;
– Status, представляющее текущий статус заявки;
– IsAttached_Fil, указывающее наличие прикрепленных файлов;
– Project содержащее название проекта;

Атрибуты класса Create_Or_Update_WViewModel:

– ID, Уникальный идентификатор для записи;
– Type_Operation, указывает тип выполняемой операции;
– Date_Time_Reg, хранит дату и время регистрации операции;
– Name_Resource, имя поставляемого ресурса;
– Count, представляет количество ресурсов;
– Price, цена ресурса, которая также может быть пустой;
– Code_Provider, связанный с поставщиком ресурса;
– Name_Provider, имя поставщика;
– User_Name, имя пользователя, связанного с операцией.

Диаграмма UML представляет собой графическое представление. Этот тип диаграмм широко используется для создания и анализа структуры реляционных баз данных. Основные элементы UL-диаграмм — это

сущности, их атрибуты и связи, которые отображаются с помощью стандартных графических символов. UML используются не только для проектирования баз данных, но и для понимания сложных систем и их взаимосвязей. Они помогают наглядно показать, какие данные хранятся и как они связаны между собой, что облегчает процесс моделирования и анализа. Благодаря этому диаграммы становятся эффективным инструментом как для архитекторов систем, так и для разработчиков программного обеспечения, предоставляя четкое визуальное представление взаимодействия данных в системе (рисунок 5).

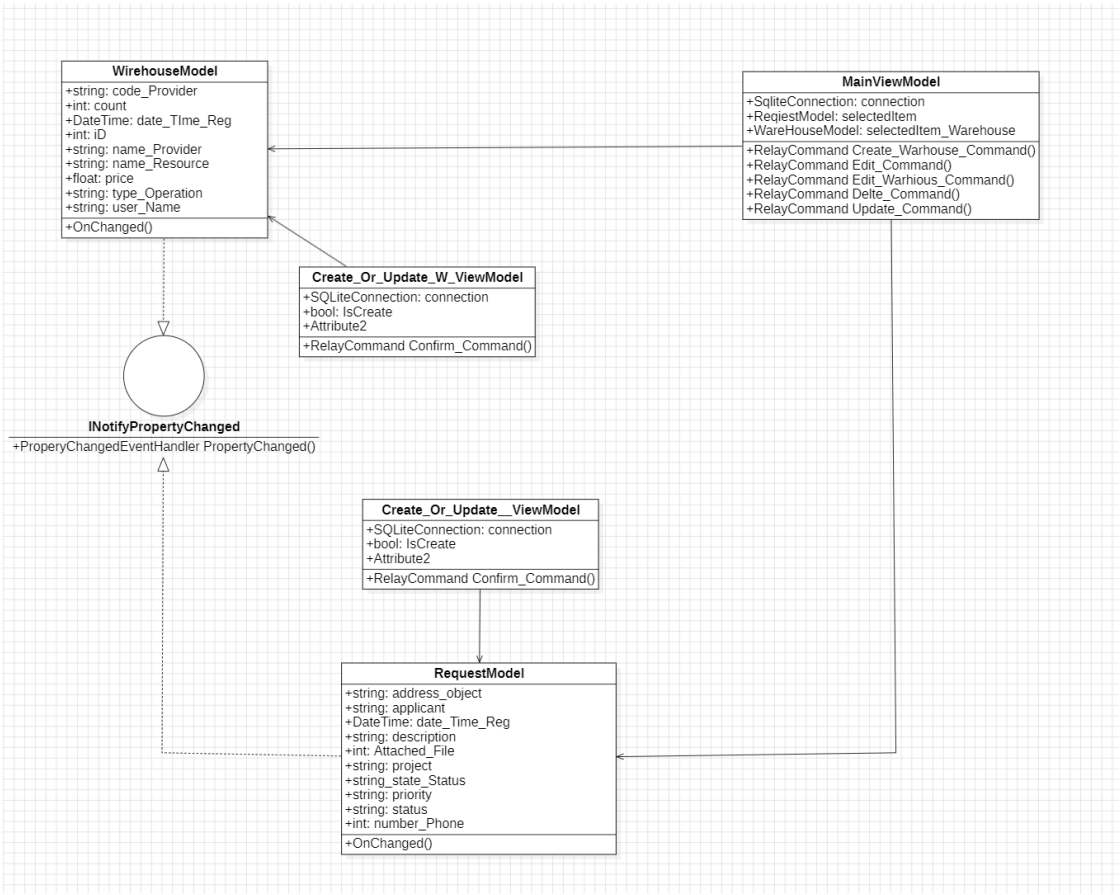


Рисунок 5 - UML-диаграмма классов

Для отображения структуры системы была создана диаграмма на языке моделирования UML. Она описывает структуру систему классов, их атрибутов и методов.

Диаграмма классов ключевой элемент объектно-ориентированного анализа и проектирования, который отражает структуру системы, описывая её основные компоненты: атрибуты, методы и отношения. Классы в диаграмме представляют сущности системы, указывают на свойства и поведение, которые могут отражаться в программе.

Она демонстрирует связи между классами. Ассоциации показывают взаимодействие друг с другом, наследование указывает на иерархию и отношения, агрегация и композиция отображают целостные и частичные зависимости.

В результате создания диаграммы классов мы получили структурированное представление системы, которое позволяет визуализировать основные принципы программы.

2.2 Архитектура проекта

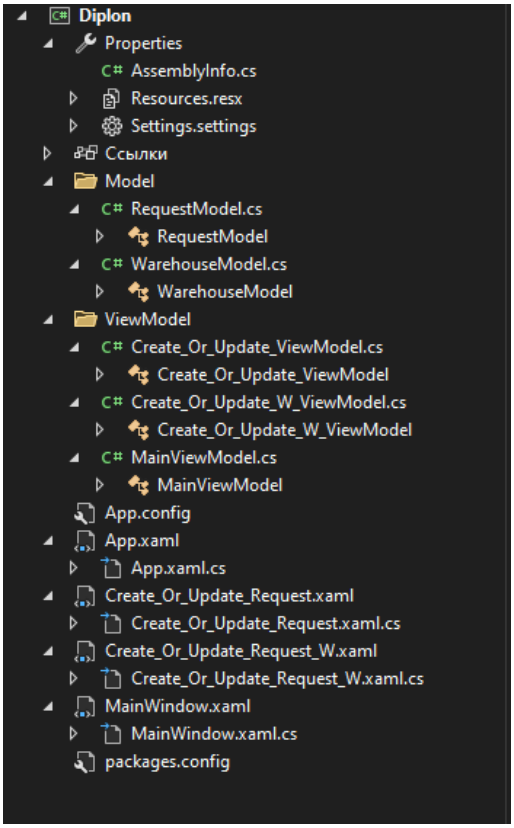


Рисунок 6 - Архитектура приложения

Архитектура MVVM является идеальной для приложений с графическим интерфейсом, где требуется разделение между логикой данных и их отображением. Особенно важно для обеспечения гибкости и масштабируемости приложения, а также для упрощения его поддержки и модульного тестирования.

Папка ViewModel содержит три класса:

– Create_Or_Update_WViewModel;
– Create_Or_Update_WViewModel;
– MainViewModel.cs.

Папка Model содержит классы, которые описывают логику и данные приложения. Могут включать методы для обработки данных. Важное свойство моделей они не знают ничего о представлении ViewModel.

Папка ViewModel содержит классы, которые связывают Model и View. ViewModel отвечает за подготовку данных для представления и за обработку пользовательских команд. Она также использует механизмы двустороннего связывания данных между View и ViewModel, чтобы поддерживать синхронизацию UI и данных.

2.3 Библиотеки

В C# библиотеки представляют собой наборы предопределенных классов и методов, которые позволяют разработчикам значительно ускорить процесс создания программного обеспечения, избегая написания однотипного кода.

Dapper библиотека для работы с базами данных в .NET, которая улучшает взаимодействие с SQL через прямое отображение запросов на объекты C#. Разработчику позволяет писать чистые запросы, сохраняя при этом удобство отображения данных, классов и структур. Он расширяет стандартные классы вводя возможность выполнения запросов напрямую через подключение к базе данных.

Dapper совместима с базой данных SQL Server. Это делает её многофункциональной и адаптируемой для использования в различных проектах. Основной принцип работы библиотеки основан на расширениях подключения к базе данных. Используя метод можно выполнить запрос и автоматически преобразовать полученные данные в коллекцию объектов, что значительно облегчает обработку результатов запросов.

Dapper обеспечивает эффективное взаимодействие с базами данных SQL, минимизируя шаблонный код и упрощая маппинг данных на объекты.

System Data SQLite позволяет работать с базой данных SQLite. Представляет собой встраиваемую, легкую реляционную базу данных, сохраняющую все данные в одном файле. Предоставляет разработчикам платформы возможность использовать базу для локальных баз данных в настольных приложениях, обеспечивая доступ к её функциональности через интерфейсы.

Требует отдельного сервера для работы, что делает её отличным выбором для небольших проектов, мобильных приложений или приложений с локальным хранением данных. Она поддерживает большинство стандартных SQL-запросов, транзакции с ACID-совместимостью, и обеспечивает простую интеграцию с различными языками программирования, включая C#.

System Data SQLite решение для создания локальных баз данных, что делает её отличным выбором для настольных.

GalaSoft MvvmLight Command создана для упрощения работы с паттерном MVVM в приложениях. Этот компонент предлагает удобные инструменты для управления командами, которые играют ключевую роль в установлении связи между пользовательским интерфейсом. Использование команд вместо обработчиков событий позволяет инкапсулировать связанные с элементами управления, что делает код более чистым и структурированным.

GalaSoft MvvmLight значительно облегчает использование паттерна MVVM в приложениях, позволяя инкапсулировать логику в командах и отделять её от пользовательского интерфейса.

Библиотеки играют ключевую роль в упрощении разработки приложений, предоставляя разработчикам мощные и удобные инструменты для работы с базами данных и реализацией паттернов проектирования. Способствуют более быстрому и удобному процессу разработки.

Выводы по второму разделу

В представленном разделе описаны проектирование и архитектура программы. Разработаны базовые диаграммы на языке UML, которые будут помогать в реализации программного обеспечения. Использован паттерна MVVM в WPF, который способствует четкому разделению логики приложения и интерфейса.

В проектировании были выбраны библиотеки языка C#, чтобы значительно укорить процессы разработки приложения.

Глава 3 Реализация и тестирование программного обеспечения

3.1 Реализация программного обеспечения

Класс MainViewModel реализует архитектуру MVVM и управляет данными, связанными с запросами. Он интегрирует библиотеку Dapper для работы с базой данных, что обеспечивает эффективное выполнение SQL-запросов.

Атрибуты:

– Request представляет собой коллекцию объектов типа, которые загружаются из базы данных. Оно используется для отображения списка запросов в интерфейсе;
– Warehouse хранит информацию о объектов, представляющих данные.

Конструктор MainViewModel, происходит инициализация соединения с базой данных и загрузка данных из таблиц Request и Warehouse (рисунок 7)

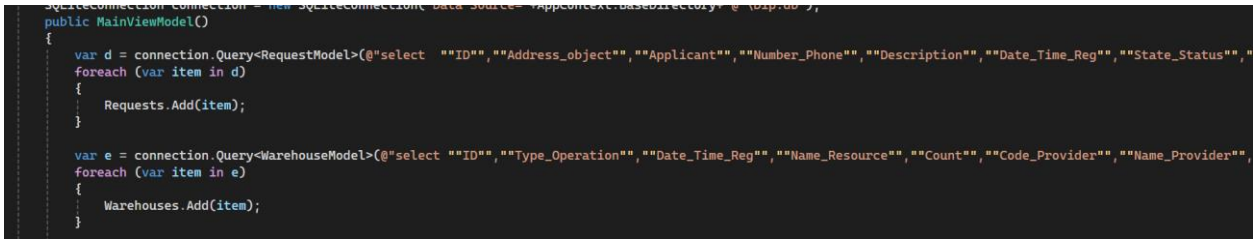


Рисунок 7 - Конструктор MainViewModel

Create_Command метод создает новый запрос, открывая диалоговое окно для ввода данных. После завершения операции он обновляет коллекции, загружая актуальные данные из базы (рисунок 8).

Рисунок 8 - Конструктор MainViewModel

Edit_Commad метод для редактирования выбранного запроса. Он открывает диалоговое окно с текущими данными выбранного элемента (рисунок 9).

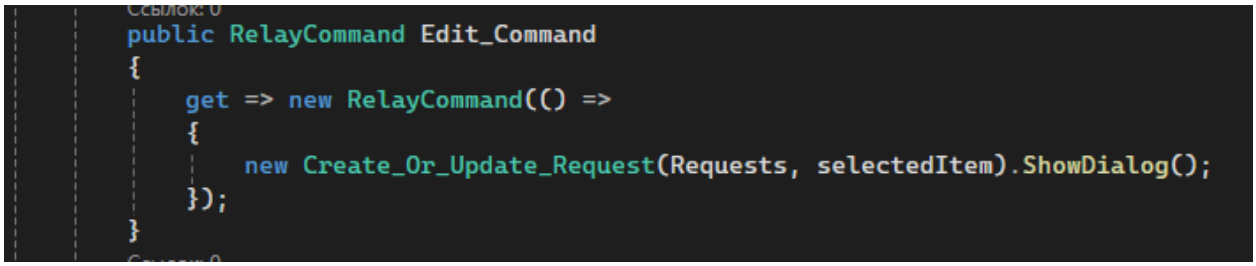


Рисунок 9 - Класса MainViewModel метод Edit_Command

Delete_Command метод удаляет выбранный запрос из базы данных и из коллекции Requests (рисунок 10).

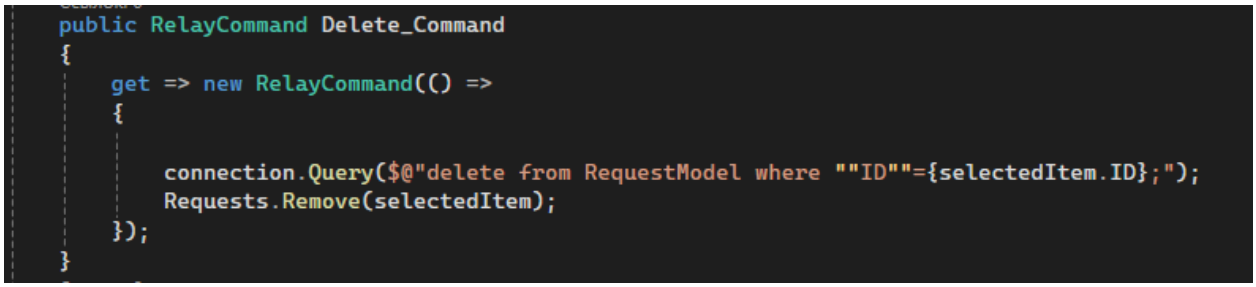


Рисунок 10 - Класса MainViewModel метод Delete_Command

Update_Command метод для обновления данных в коллекциях. Он очищает текущие коллекции (рисунок 11).

Рисунок 11 - Класса MainViewModel метод Update_Command

Класс RequestModel объект описывающее свойство и события объектов. Представляет собой модель данных для заявки в системе и реализует интерфейс INotifyPropertyChanged, что позволяет уведомлять пользовательский интерфейс об изменениях в свойствах объекта. Каждый сеттер вызывает метод OnChanged.

Метод OnChanged вызывается каждый раз, когда свойство устанавливается, что вызывает событие PropertyChanged. Уведомляет любые связанные элементы пользовательского интерфейса о том, что данные изменились (рисунок 12).

Рисунок 12 - Свойство и метод в классе RequestModel

Каждое свойство в классе WarehouseModel set и get. Вызывается OnChanged(), чтобы уведомить всех слушателей об изменении. Эта схема обеспечивает возможность реакции пользовательского интерфейса на изменения состояния модели (рисунок 13).

```
public class WarehouseModel : INotifyPropertyChanged
{
    private int iD;
    private string type_Operation;
    private DateTime? date_Time_Reg;
    private string name_Resource;
    private int count;
    private float? price;
    private string code_Provider;
    private string name_Provider;
    private string user_Name;

    public event PropertyChangedEventHandler PropertyChanged;
    Ссылок: 9
    protected void OnChanged([CallerMemberName] string PropertyName = "")
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(PropertyName));
    }
    /// <summary>
    /// Номер
    /// </summary>
    Ссылок: 6
    public int ID { get => iD; set { iD = value; OnChanged(); } }
    /// <summary>
    /// Вид операции
    /// </summary>
    Ссылок: 6
    public string Type_Operation { get => type_Operation; set { type_Operation = value; OnChanged(); } }
    /// <summary>
    /// Время регистрации
    /// </summary>
    Ссылок: 7
    public DateTime? Date_Time_Reg { get => date_Time_Reg; set { date_Time_Reg = value; OnChanged(); } }
    /// <summary>
    /// Наименование ресурса
    /// </summary>
    Ссылок: 6
    public string Name_Resource { get => name_Resource; set { name_Resource = value; OnChanged(); } }
```

Рисунок 13 - Свойство и метод в классе WarehouseModel

Класс Create_Or_Update_ViewModel модель создания и редактирования записей запроса (рисунок 14).

Отвечает за логику создания и обновления записей в базе данных SQLite в приложении с архитектурой MVVM. Этот класс используется для управления данными и взаимодействует с пользовательским интерфейсом через команды. В конструкторе класса определяется, выполняется ли создание новой записи или редактирование существующей. Если объект передан, данные заполняются значениями из существующей записи, что указывает на режим редактирования.

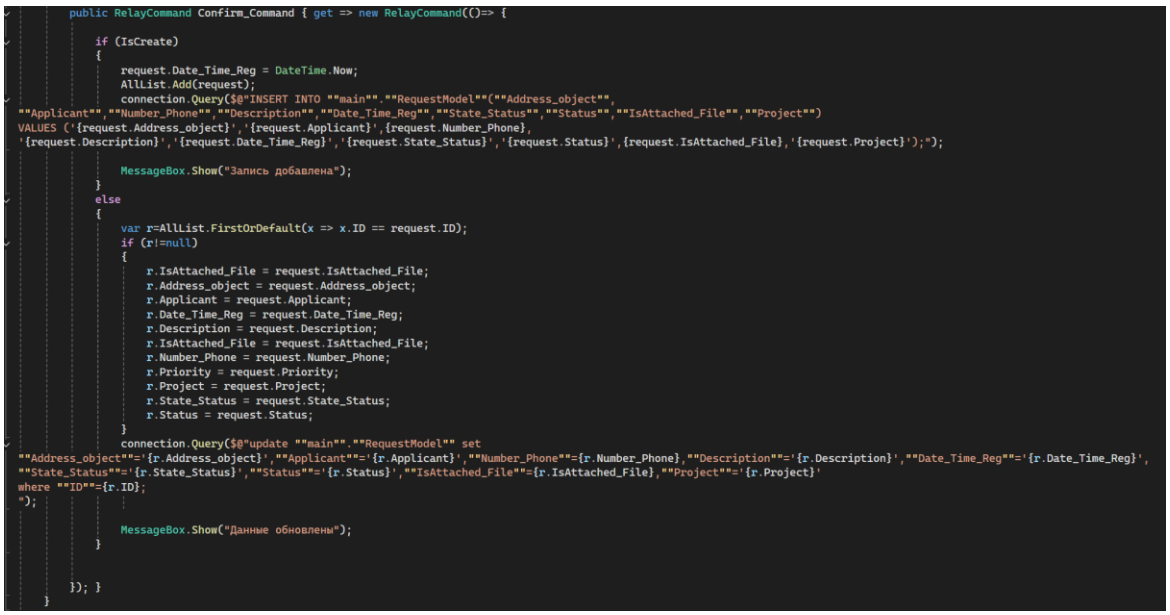


Рисунок 14 - Обработка действий Create_Or_Update_ViewModel

Класс Create_Or_Update_WViewModel модель создания и редактирования записей ресурсов (рисунок 15).

Рисунок 15 – Обработка действий Create_Or_Update_WViewMoidel

Confirm_Command отвечает за обработку действий пользователя при создании или обновлении объекта RequestModel. Если происходит создание, она добавляет новый запрос в список и базу данных, устанавливая текущее

время регистрации, обновляет его свойства и сохраняет изменения в базе данных.

Основной смысл MainViewModel заключается в управлении состоянием интерфейса и взаимодействием с базой данных через CRUD-операции. Класс включает в себя коллекции объектов Request и Warehouse, представляющих заявки и ресурсы, которые загружаются из базы данных для отображения и управления в интерфейсе.

Классы RequestModel и WarehouseModel описывают свойства и события объектов. Они поддерживают механизм уведомления интерфейса об изменениях через метод OnChanged, который автоматически вызывается при изменении любого свойства модели. Это обеспечивает динамическое обновление пользовательского интерфейса при изменении состояния данных.

3.1.1 Интерфейс и руководство программы

Реализация программы представляет собой базу данных, которая осуществляет помощь и навигацию по клиенткой базе для более эффективного распределения и сбыта тепловой энергии.

Стартовый экран включает в себя набор полей введения текста, кнопки и сетку данных с возможностью фильтрации и множественной сортировки (рисунок 16).

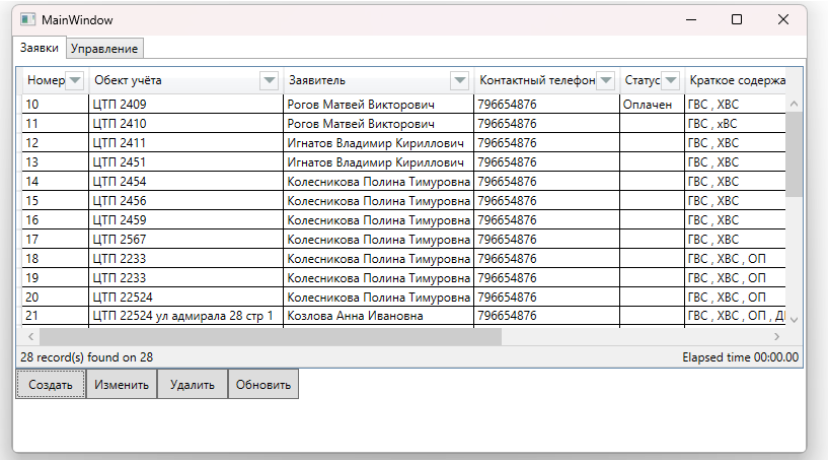


Рисунок 16 - Стартовое окно

Для проведения операций CRUD (create, read, update and delete) необходимо нажать на одну из следующих кнопок (создать, изменить), после нажатия вызывается одна из встроенных команд посредством соединения View и View Model через паттерн MVVM, которая является основополагающей на платформе WPF (Windows Presentation Foundation). Далее раскрывается модульное окно для введения корректировок и создания записей, использован паттерн PropertyGrid (рисунок 17).

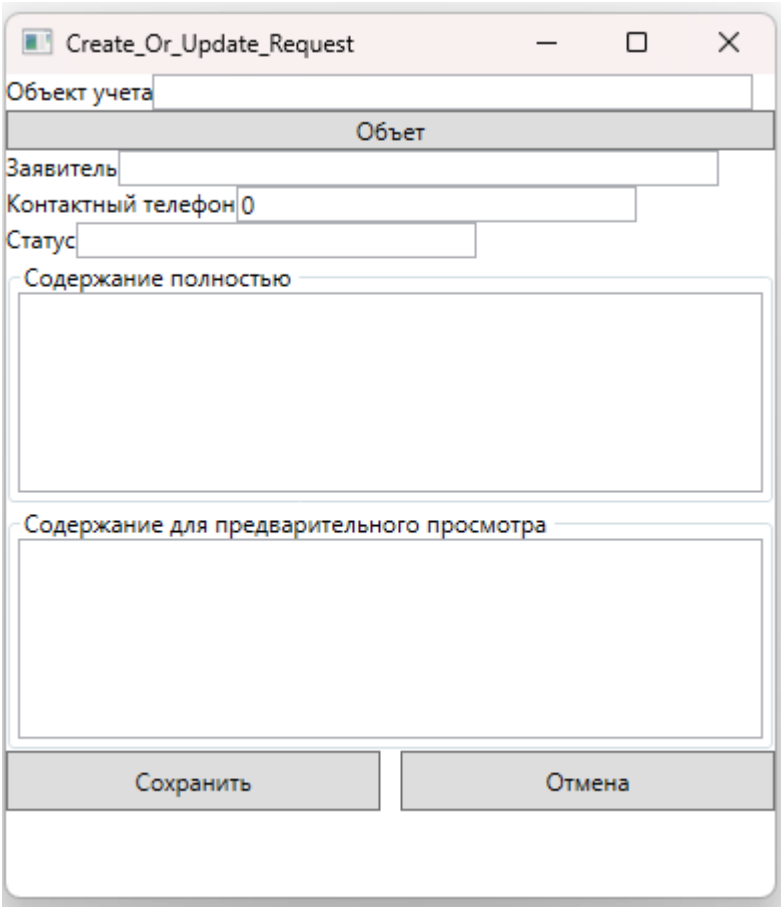


Рисунок 17 - Модульное окно

Код представлен в Приложении А.

Второстепенной вкладкой для распределения и сбыта тепловой энергии выступает вкладка «управление». Здесь хранится номер, вид операций, время

регистрации, наименование ресурса, количество, цена, код поставщика, наименования поставщика и пользователь вводившие данные (рисунок 18).

Основными составляющими данного окна является компоновщик UI элементов для правильного масштабирования в условиях различных разрешений экрана.

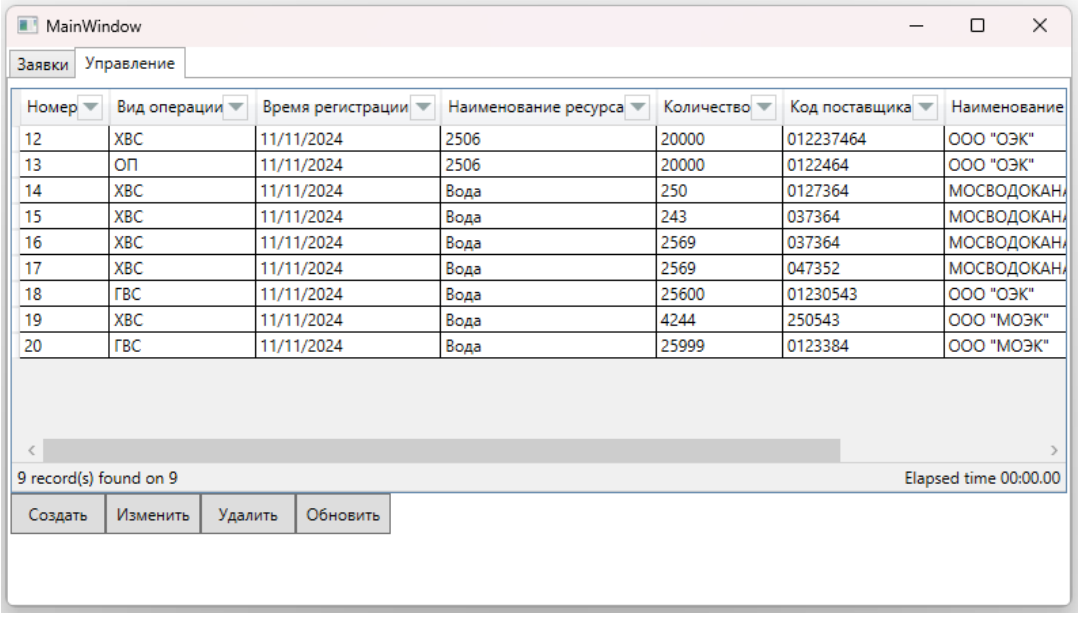


Рисунок 18 - Вкладка «управление»

Вся схема была написана одним из множества SQL подобных диалектов. Транзакционное создание таблиц и последующих скриптов позволяют откатывать (rollback) пакета транзакций. С помощью систем управления базами данных (СУБД) SQLite Browse и встроенной средой разработки Visual Studio 22 (рисунок 19).

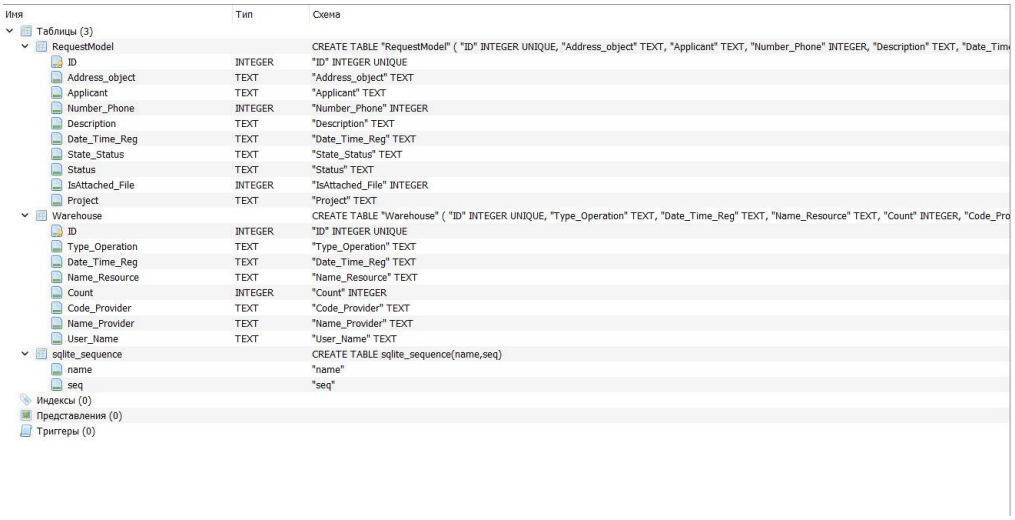


Рисунок 19 - Схема базы данных SQlite

Создана структура базы данных с использованием SQL диалекта, обеспечивающая целостность и надежность данных. Система транзакций и возможность отката (rollback) гарантируют сохранность данных, а использование инструментов SQLite Browser и Visual Studio 2022 упрощает управление и тестирование схемы базы данных.

3.2 Нагрузочное тестирование программы

Для проведения тестирования были использованы настольный персональный компьютер, операционная система Windows 11. Характеристики персонального компьютера:

- процессор: AMD Ryzen 5 5600, 6 ядерный и 12 потоковый процессор, обеспечивают отличную производительность в многозадачных сценариях для тестирования приложения.
- оперативная память (RAM): 16 ГБ (2x8 ГБ) DDR4, оптимально для большинства рабочих приложений, высокая частота способствует лучшей производительности, особенно с процессорами AMD,

двухканальная конфигурация увеличивает скорость доступа к данным.

– накопитель SSD: M.2, 512 ГБ, повышает скорость загрузки операционной системы и приложений, ускоряет работу с файлами копирование, перемещение, чтение, где нужно часто работать с большими файлами.

Один из ключевых этапов загрузочное тестирование, при котором в систему загружается большой объем записей. Это тестирование помогает определить, как приложение справляется с задачами отображения, обновления и удаления значительных объемов данных, а также выявить возможные задержки или замедления в интерфейсе.

Код использует коллекции для хранения данных о запросах, так как ObservableCollect автоматически уведомляет об изменениях, что упрощает обновление пользовательского интерфейса и минимизирует ручное управление состоянием. Using для соединения с SQLiteConnection, подключение создается каждый раз, когда это необходимо.

Код создает миллион записей для двух коллекций данных Requests и Warehouses. Добавляет каждый новый элемент в соответствующий список с использованием цикла, который повторяется миллион раз (рисунок 20).

```
for (int i = 0; i < 1000000; i++)
{
    Requests.Add(new RequestModel { ID = i, Applicant = "Заколюкин",
        Date_Time_Reg = DateTime.Now,
        Number_Phone = 764634373,
        Address_object = "Абрамово"  });
    Warehouses.Add(new WarehouseModel { ID = i,
        Code_Provider = "аЗаколюкин",
        Date_Time_Reg = DateTime.Now,
        Name_Provider = "Абрамово" });
}
```

Рисунок 20 – код для нагрузки системы

Итерация цикла добавляет в коллекцию Requests объект RequestModel, содержащий уникальный идентификатор, значение поля, текущую дату и время регистрации, контактный номер телефона, а также адрес Эти данные представляют стандартную информацию для регистрации

Информация отображаются в заявках и управлении. Отображение такого большого объема данных позволяет проверить производительность и отклик интерфейса при высокой нагрузке, а также продемонстрировать, как система справляется с обработкой и представлением информации в большом масштабе (рисунок 21).

999986	Абрамово	Заколюкин	764634373
999987	Абрамово	Заколюкин	764634373
999988	Абрамово	Заколюкин	764634373
999989	Абрамово	Заколюкин	764634373
999990	Абрамово	Заколюкин	764634373
999991	Абрамово	Заколюкин	764634373
999992	Абрамово	Заколюкин	764634373
999993	Абрамово	Заколюкин	764634373
999994	Абрамово	Заколюкин	764634373
999995	Абрамово	Заколюкин	764634373
999996	Абрамово	Заколюкин	764634373
999997	Абрамово	Заколюкин	764634373
999998	Абрамово	Заколюкин	764634373
999999	Абрамово	Заколюкин	764634373

Рисунок 21 – Загрузка данных

Данные автоматически интегрируются в интерфейс программы, что создает условия для проведения различных операций без дополнительных действий со стороны пользователя, позволяя им сразу же взаимодействовать с большими объемами информации.

Загрузка такого объема информации позволяет оценить производительность и масштабируемость программы при интенсивной загрузке данных (рисунок 22).

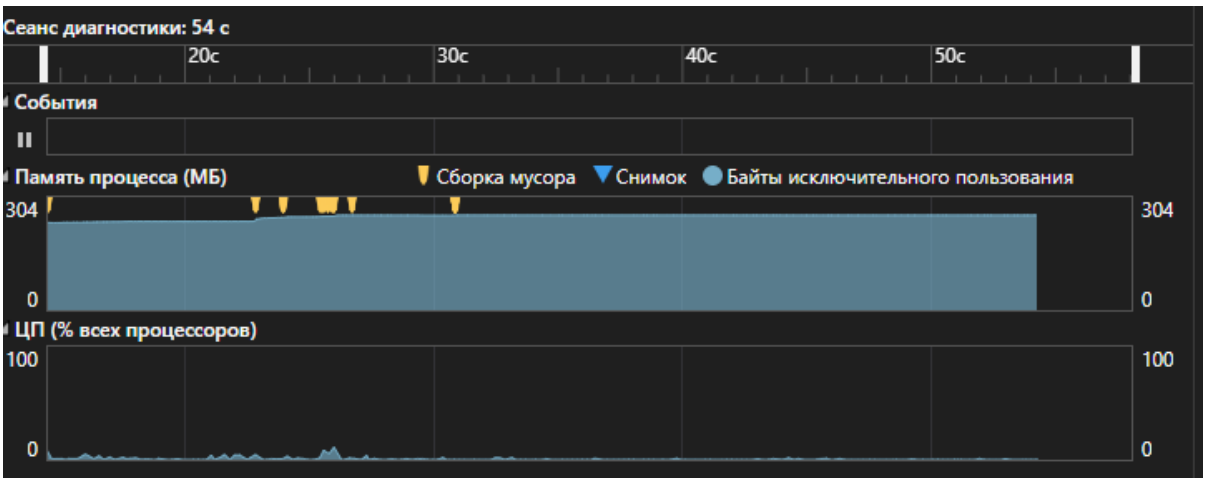


Рисунок 22 - Средства диагностики

Программа демонстрирует эффективное потребление ресурсов, используя всего 304 МБ оперативной памяти и два процента процессорного времени в обычном рабочем режиме. Эти показатели свидетельствуют о низкой нагрузке на систему, что особенно важно для приложений, взаимодействующих с базами данных, так как позволяет улучшить отзывчивость и производительность при выполнении различных операций. СУБД работают с большими объемами данных и сложными запросами, могут потреблять значительно больше ресурсов. MySQL или PostgreSQL в зависимости от конфигурации и загруженности используют от 150 до 500 МБ оперативной памяти и более десяти процентов процессорного времени на стандартных операциях.

Проведена проверка интерфейса и производительность программного обеспечения. Нагрузка на систему позволило выявить возможные трудности и улучшить взаимодействие с приложением. Также были проведен анализ производительности, чтобы убедиться, что приложение эффективно использует системные ресурсы, особенно в условиях многозадачности.

3.3 Юнит тестирование программы

Юнит-тестирование процесс тестирования программного обеспечения, при котором отдельные компоненты или модули программы проверяются на корректность работы. Представляют собой небольшие, изолированные фрагменты кода, методы или классы. Проводится на ранних этапах разработки, чтобы отдельные части программы выполняют свои задачи правильно.

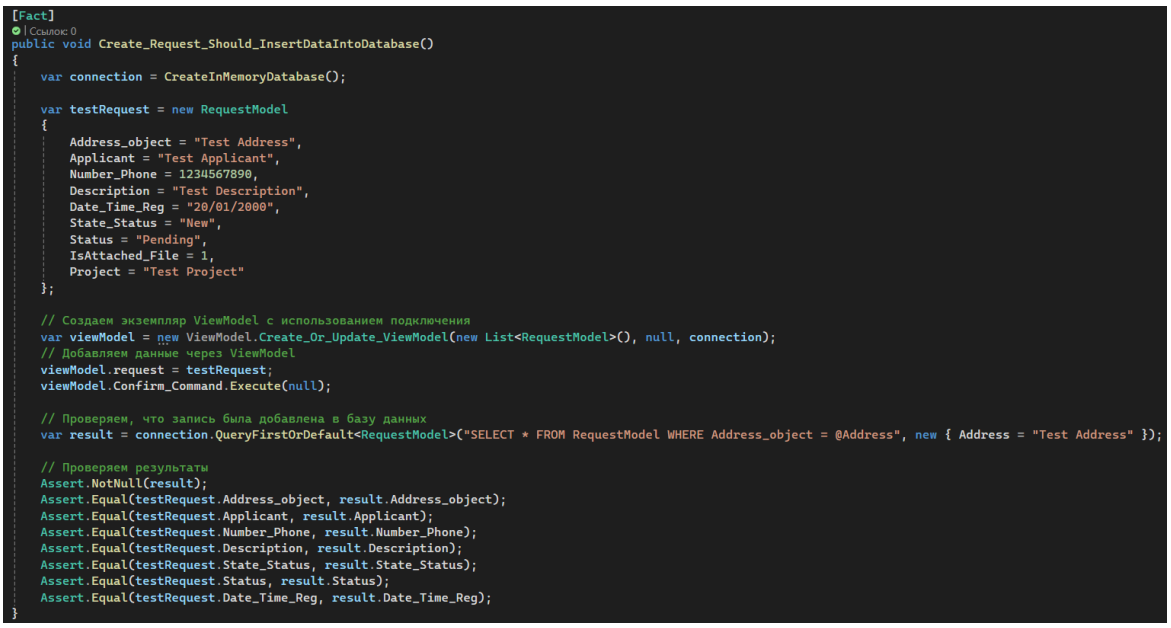
Целью тестирования является выявление ошибок в самом базовом уровне системы, прежде чем они смогут повлиять на более крупные модули или всю систему в целом. Успешное тестирование должно выдавать положительный для корректных участков кода и обнаруживать ошибки. Правильное выполнение тестирование дает уверенность, что на уровне отдельных модулей программа работает стабильно, что облегчает последующую интеграцию и поддержание проекта.

В разделе проведено три юнит-теста, каждый из которых направлен на проверку основных функции программы. Тесты охватывают добавление записи в базу данных и взаимодействия с UI приложения добавление записей, обновление существующей записи и изменение. Эти тесты используются для проверки базовых CRUD-операций, которые составляют основу работы с таблицей RequestModel.

Всего в коде было проведено три юнит-теста:

- create_Request_Should_InsertDataIntoDatabase;
- update_Request_Should_UpdateDataInDatabase;
- delete_Request_Should_RemoveDataFromDatabas.

Тест Create_Request_Should_InsertDataIntoDatabase проверяет, что новая запись корректно добавляется в таблицу базы данных с помощью ViewModel. Он создает объект RequestModel, добавляет его в базу данных, а затем извлекает его с помощью SQL-запроса, что данные совпадают с ожидаемыми (рисунок 23).



```
[Fact]
| Ссылок: 0
public void Create_Request_Should_InsertDataIntoDatabase()
{
    var connection = CreateInMemoryDatabase();

    var testRequest = new RequestModel
    {
        Address_object = "Test Address",
        Applicant = "Test Applicant",
        Number_Phone = 1234567890,
        Description = "Test Description",
        Date_Time_Reg = "20/01/2000",
        State_Status = "New",
        Status = "Pending",
        IsAttached_File = 1,
        Project = "Test Project"
    };

    // Создаем экземпляр ViewModel с использованием подключения
    var viewModel = new ViewModel.Create_Or_Update_ViewModel(new List<RequestModel>(), null, connection);
    // Добавляем данные через ViewModel
    viewModel.request = testRequest;
    viewModel.Confirm_Command.Execute(null);

    // Проверяем, что запись была добавлена в базу данных
    var result = connection.QueryFirstOrDefault<RequestModel>("SELECT * FROM RequestModel WHERE Address_object = @Address", new { Address = "Test Address" });

    // Проверяем результаты
    Assert.NotNull(result);
    Assert.Equal(testRequest.Address_object, result.Address_object);
    Assert.Equal(testRequest.Applicant, result.Applicant);
    Assert.Equal(testRequest.Number_Phone, result.Number_Phone);
    Assert.Equal(testRequest.Description, result.Description);
    Assert.Equal(testRequest.State_Status, result.State_Status);
    Assert.Equal(testRequest.Status, result.Status);
    Assert.Equal(testRequest.Date_Time_Reg, result.Date_Time_Reg);
}
```

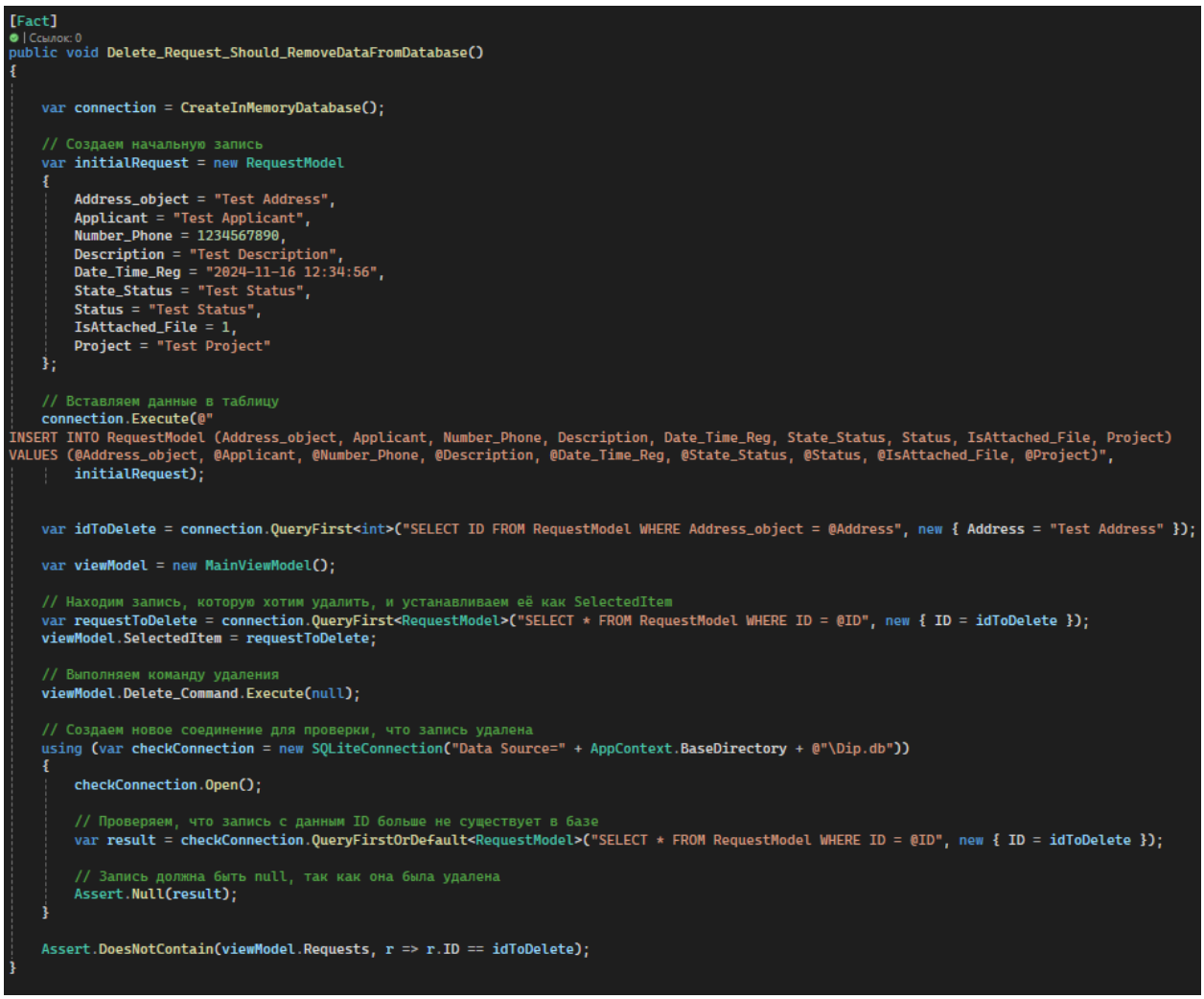
Рисунок 23 - Тест Create_Request_Should_InsertDataIntoDatabase

Тест Update_Request_Should_UpdateDataInDatabase проверяет существование запись, которая может быть изменена. Он вставляет и изменяет данные через SQL-запрос и проверяет, что данные обновились (рисунок 24).

```
[Fact]
● | Ссылок: 0
public void Update_Request_Should_UpdateDataInDatabase()
{
    var connection = CreateInMemoryDatabase();

    // Добавляем начальную запись
    var initialRequest = new RequestModel
    {
        Address_object = "Old Address",
        Applicant = "Old Applicant",
        Number_Phone = 987654210,
        Description = "Old Description",
        Date_Time_Reg = "2024-11-16 12:34:56",
        State_Status = "Old Status",
        Status = "Old Status",
        IsAttached_File = 1,
        Project = "Old Project"
    };

    connection.Execute(@"
INSERT INTO RequestModel (Address_object, Applicant, Number_Phone, Description, Date_Time_Reg, State_Status, Status, IsAttached_File, Project)
VALUES (@Address_object, @Applicant, @Number_Phone, @Description, @Date_Time_Reg, @State_Status, @Status, @IsAttached_File, @Project)",
        initialRequest);

    // Обновляем запись через команду
    var updatedRequest = new RequestModel
    {
        ID = 1,
        Address_object = "Updated Address",
        Applicant = "Updated Applicant",
        Number_Phone = 1234567890,
        Description = "Updated Description",
        Date_Time_Reg = "2024-11-17 12:34:56",
        State_Status = "Updated Status",
        Status = "Updated Status",
        IsAttached_File = 1,
        Project = "Updated Project"
    };

    // Обновляем запись в базе данных
    connection.Execute(@"
UPDATE RequestModel
SET Address_object = @Address_object,
    Applicant = @Applicant,
    Number_Phone = @Number_Phone,
    Description = @Description,
    Date_Time_Reg = @Date_Time_Reg,
    State_Status = @State_Status,
    Status = @Status,
    IsAttached_File = @IsAttached_File,
    Project = @Project
WHERE ID = @ID", updatedRequest);


    var result = connection.QueryFirstOrDefault<RequestModel>(
        "SELECT * FROM RequestModel WHERE ID = @ID",
        new { ID = 1 }
    );


    Assert.NotNull(result);
    Assert.Equal(updatedRequest.Address_object, result.Address_object);
    Assert.Equal(updatedRequest.Applicant, result.Applicant);
    Assert.Equal(updatedRequest.Number_Phone, result.Number_Phone);
    Assert.Equal(updatedRequest.Description, result.Description);
    Assert.Equal(updatedRequest.Date_Time_Reg, result.Date_Time_Reg); // Проверяем, что дата обновилась
}
```

Рисунок 24 - Update_Request_Should_UpdateDataInDatabase

Тест Delete_Request_Should_RemoveDataFromDatabase удаление записи из базы данных, так и из коллекции ViewModel. Для этого создается запись, которая затем удаляется через ViewModel. Тест проверяет, что удаленной записи больше нет в базе, а также проверяет коллекцию ViewModel (рисунок 25).



```
[Fact]
● | Ссылок: 0
public void Delete_Request_Should_RemoveDataFromDatabase()
{

    var connection = CreateInMemoryDatabase();

    // Создаем начальную запись
    var initialRequest = new RequestModel
    {
        Address_object = "Test Address",
        Applicant = "Test Applicant",
        Number_Phone = 1234567890,
        Description = "Test Description",
        Date_Time_Reg = "2024-11-16 12:34:56",
        State_Status = "Test Status",
        Status = "Test Status",
        IsAttached_File = 1,
        Project = "Test Project"
    };

    // Вставляем данные в таблицу
    connection.Execute(@"
INSERT INTO RequestModel (Address_object, Applicant, Number_Phone, Description, Date_Time_Reg, State_Status, Status, IsAttached_File, Project)
VALUES (@Address_object, @Applicant, @Number_Phone, @Description, @Date_Time_Reg, @State_Status, @Status, @IsAttached_File, @Project)",
        initialRequest);


    var idToDelete = connection.QueryFirst<int>("SELECT ID FROM RequestModel WHERE Address_object = @Address", new { Address = "Test Address" });

    var viewModel = new MainViewModel();

    // Находим запись, которую хотим удалить, и устанавливаем её как SelectedItem
    var requestToDelete = connection.QueryFirst<RequestModel>("SELECT * FROM RequestModel WHERE ID = @ID", new { ID = idToDelete });
    viewModel.SelectedItem = requestToDelete;

    // Выполняем команду удаления
    viewModel.Delete_Command.Execute(null);

    // Создаем новое соединение для проверки, что запись удалена
    using (var checkConnection = new SQLiteConnection("Data Source=" + AppContext.BaseDirectory + @"\Dip.db"))
    {
        checkConnection.Open();

        // Проверяем, что запись с данным ID больше не существует в базе
        var result = checkConnection.QueryFirstOrDefault<RequestModel>("SELECT * FROM RequestModel WHERE ID = @ID", new { ID = idToDelete });

        // Запись должна быть null, так как она была удалена
        Assert.Null(result);
    }

    Assert.DoesNotContain(viewModel.Requests, r => r.ID == idToDelete);
}
```

Рисунок 25 - Тест Delete_Request_Should_RemoveDataFromDatabase

Класс RequestModel представляет модель данных и соответствует таблице в базе данных. Он содержит свойства, которые используются для вставки, обновления, и проверки данных в таблице. Create_Or_Update_ViewModel используется для работы с операциями добавления и обновления записей в базе. Он обрабатывает команды ViewModel, такие как Confirm_Command, которые вызываются в тестах.

MainViewModel управляет коллекцией запросов и содержит команды для удаления записей из базы данных. Все три класса взаимодействуют с базой данных через SQLite, что позволяет тестам быть изолированными и независимыми от внешней среды (рисунок 26)

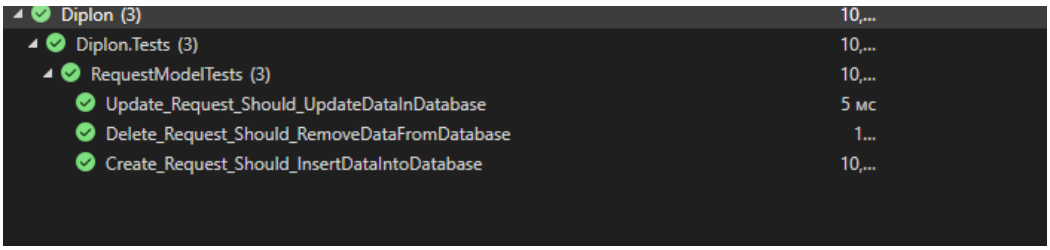


Рисунок 26 – Результат тестирования

Все тесты показывают успешное выполнение. Операций на создание подтверждает, что данные сохраняются в таблице и соответствуют ожидаемым значениям. Обновление показывает, что запись корректно изменяется, и данные обновляются в таблице. Тест на удаление подтверждает, что запись удаляется как из базы и из коллекции ViewMode.

Выводы по третьему разделу

В представленном разделе были описаны реализация программы и ее интерфейса с руководством пользователя. Стартовый экран программы содержит поля для ввода текста, кнопки и данных. Для работы с данными предусмотрены операции CRUD, что обеспечивает гибкость в управлении записями. Использование паттерна MVVM (Model-View-ViewModel) в WPF способствует четкому разделению логики приложения.

Программа была протестирована на ресурсы оперативной памяти и процессора с загрузкой СУБД. Выполнен анализ производительности производительность программного обеспечения. Созданы три юнит теста.

Заключение

В рамках выпускной квалификационной работы было реализовано программное обеспечение, которая осуществляет помощь и навигацию по клиенткой базе для более эффективного распределения и сбыта тепловой энергии.

В ходе работы была выбрана IDE Visual studio, язык C#, для персональных компьютеров под операционной системой Windows.

В ходе выполнения работы были реализованы следующие задачи:

- выполнен сравнительный анализ методологий разработки программного обеспечения;
- проанализирована предметная область;
- разработан прототип программного обеспечения, система распределения и сбыта тепловой энергии;
- разработана база данных SQLite;
- рассмотрены виды интегрированных сред разработки;
- протестировано разработанное приложение.

Разработанное приложение способствует повышению эффективности управления тепловыми ресурсами, обеспечивая более точный контроль за потреблением и распределением энергии. Это позволит теплоснабжающим организациям минимизировать потери, снизить эксплуатационные расходы и повысить качество предоставляемых услуг.

Система может быть адаптирована для масштабирования, что делает её гибким инструментом для использования в различных условиях.

Созданная система удовлетворяет ключевым требованиям, и может быть успешно внедрена в работу теплоснабжающей компании.

Список используемой литературы

1. Алан А.А., Донован Б., Кернинган У. Язык программирования Go [Электронный ресурс] - URL: https://library-it.com/wp-content/uploads/2021/02/jazyk_programmirovanija_go_2016.pdf (Дата обращения 05.07.2024)
2. Востокин С.В. Применение метода парного взаимодействия объектов для построения сред разработки распределенных приложений // Вестн. Сам. гос. техн. ун-та. Сер.: Физ.-мат. науки. 2005. №38. URL: https://cyberleninka.ru/article/n/primenenie-metoda-parnogo-vzaimodeystviya-obektov-dlya-postroeniya-sred-razrabotki-raspredelennyh-prilozheniy (дата обращения: 06.10.2024).
3. Е Ю. Артюшевская ТРУДНОСТИ РЕАЛИЗАЦИИ ЭНЕРГОСБЕРЕЖЕНИЯ И АВТОМАТИЗАЦИИ В СИСТЕМАХ ТЕПЛОСНАБЖЕНИЯ // Вестник Амурского государственного университета. Серия: Гуманитарные науки. 2020. №89. URL: https://cyberleninka.ru/article/n/trudnosti-realizatsii-energosberezheniya-i-avtomatizatsii-v-sistemah-teplosnabzheniya-1 (дата обращения: 18.10.2024).
4. Ельсуков Д.А. Python - язык программирования // Экономика и социум. 2021. №11-1 (90). URL: https://cyberleninka.ru/article/n/python-yazyk-programmirovaniya (дата обращения: 06.08.2024).
5. Жалолов О.И., Хаятов Х.У. Понятие SQL и реляционной базы данных // Universum: технические науки. 2020. №6-1 (75). URL: https://cyberleninka.ru/article/n/ponyatie-sql-i-relyatsionnoy-bazy-dannyh (дата обращения: 06.10.2024).
6. Ильясов А., Чарыева Г. База данных и требования к базе данных // CETERIS PARIBUS. 2022. №12. URL: https://cyberleninka.ru/article/n/baza-dannyh-i-trebovaniya-k-baze-dannyh (дата обращения: 06.10.2024).
7. Индекс DB-engines [Электронный ресурс] URL: https://db-engines.com/en/ranking (Дата обращения 07.07.2024)

8. Индекс PYPL [Электронный ресурс] - URL: https://pypl.github.io/IDE.html (Дата обращения 07.07.2024)

9. Индекс TIOBE. [Электронный ресурс] - URL: https://www.tiobe.com/tiobe-index/ (Дата обращения 05.07.2024)

10. Ишмурадова И.И., Еремина И.И., Лысанов Д.М., Ахметьянова Э.А. Программная реализация мониторинга ит-инфраструктуры предприятия с использованием Microsoft Visual Studio // НК. 2021. №4. URL: https://cyberleninka.ru/article/n/programmnaya-realizatsiya-monitoringa-it-infrastruktury-predpriyatiya-s-ispolzovaniem-microsoft-visual-studio (дата обращения: 06.10.2024).

11. Колмакова Е.Н. Test-driven development - Разрабока через тестирование: преимущества и недостатки // Экономика и социум. 2016. №3 (22). URL: https://cyberleninka.ru/article/n/test-driven-development-razraboka-cherez-testirovanie-preimuschestva-i-nedostatki (дата обращения: 06.10.2024).

12. Костриков В.В. Применение каскадной модели и гибких методологий при разработке программных продуктов // Известия ТулГУ. Технические науки. 2022. №9. URL: https://cyberleninka.ru/article/n/primenenie-kaskadnoy-modeli-i-gibkih-metodologiy-pri-razrabotke-programmnyh-produktov (дата обращения: 06.10.2024).

13. Мамедли Р.Э. Системы управления базами данных [Электронный ресурс] - URL: https://nvsu.ru/ru/Intellekt/2316/Mamedli_R.EH._Sistemy_upravleniya_bazami_dannykh.pdf?ysclid=m1xtbcase3745859123 (дата обращения 29.07.2024)

14. Отчет о функционировании ЕЭС России в 2023 году. [Электронный ресурс] - URL: https://www.so-ups.ru/fileadmin/files/company/reports/disclosure/2024/ups_rep2023.pdf (дата обращения 29.06.2024).

15. Павловская Т.А. C++ программирование на языке выского уровня URL: https://cyberleninka.ru/article/n/python-yazyk-programmirovaniya (дата обращения: 06.10.2024).

16. Рябов В. И. Применение среды Xcode для разработки приложений для iOS // Вестник МГУП. 2013. №2. URL: https://cyberleninka.ru/article/n/primenenie-sredy-xcode-dlya-razrabotki-prilozheniy-dlya-ios (дата обращения: 08.10.2024).

17. Сравнительная характеристика технологий .Net и .JAVA [Электронный источник] - URL: https://studfile.net/preview/3675377/ (дата обращения 19.08.2024)

18. Старушенкова Екатерина Евгеньевна ОСНОВНЫЕ ЭТАПЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ // E-Scio. 2021. №11 (62). URL: https://cyberleninka.ru/article/n/osnovnye-etapy-proektirovaniya-baz-dannyh (дата обращения: 18.11.2024).

19. Таратынов М. С., Егунова А. И., Аббакумов А. А. Разработка приложения для управления информационной системой в среде разработки eclipse // Инновационная наука. 2017. №6. URL: https://cyberleninka.ru/article/n/razrabotka-prilozheniya-dlya-upravleniya-informatsionnoy-sistemoy-v-srede-razrabotki-eclipse (дата обращения: 06.10.2024).

20. Таршхоева Ж.Т. Язык программирования Python. Библиотеки Python / Ж. Т. Таршхоева. – Текст : непосредственный // Молодой ученый. – 2021. – № 5 (347). – С. 20-21. – URL: https://moluch.ru/archive/347/78102/ (дата обращения 05.07.2024).

Приложение А

Листинг кода

```
<Window x:Class="Diplon.Create_Or_Update_Request"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Diplon"
        mc:Ignorable="d"
        Title="Create_Or_Update_Request" Height="450" Width="400">
    <Grid>
        <StackPanel>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="Объект учета"/>
                <TextBox HorizontalAlignment="Stretch" Width="300" Text="{Binding
request.Address_object, Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
            </StackPanel>
            <Button Content="Объет" HorizontalContentAlignment="Center"/>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="Заявитель"/>
                <TextBox HorizontalAlignment="Stretch" Width="300" Text="{Binding
request.Applicant, Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
            </StackPanel>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="Контактный телефон" />
                <TextBox HorizontalAlignment="Stretch" Width="200" Text="{Binding
request.Number_Phone, Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
            </StackPanel>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="Приоритет"/>
                <TextBox HorizontalAlignment="Stretch" Width="200" Text="{Binding
request.Priority, Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
            </StackPanel>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="Статус"/>
                <TextBox HorizontalAlignment="Stretch" Width="200" Text="{Binding
request.Status, Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
            </StackPanel>
            <GroupBox Header="Содержание полностью">
                <TextBox Height="100" Text="{Binding request.Description,
Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
            </GroupBox>
            <GroupBox Header="Содержание для предварительного просмотра">
                <TextBox Height="100" Text="{Binding request.Description,
Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
            </GroupBox>
            <Grid Height="30">
```

```
                <Grid.ColumnDefinitions>
                    <ColumnDefinition/>
                    <ColumnDefinition Width="10"/>
                    <ColumnDefinition/>
                </Grid.ColumnDefinitions>
                <Button Content="Сохранить" Command="{Binding Confirm_Command}"
Grid.Column="0"/>
                <Button Content="Отмена" Click="Button_Click" Grid.Column="2"/>
            </Grid>
        </StackPanel>
    </Grid>
</Window>

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;

namespace Diplon.Model
{
    /*CREATE TABLE "RequestModel" (
            "ID"    INTEGER UNIQUE,
            "Address_object"    TEXT,
            "Applicant"    TEXT,
            "Number_Phone"    INTEGER,
            "Description"  TEXT,
            "Date_Time_Reg"    TEXT,
            "State_Status" TEXT,
            "Status"    TEXT,
            "IsAttached_File"    INTEGER,
            "Project"    TEXT,
            PRIMARY KEY("ID" AUTOINCREMENT)
);*/
    /// <summary>
    /// Модель заявки
    /// </summary>
    public class RequestModel : INotifyPropertyChanged
    {
        private string address_object;
        private string applicant;
        private long number_Phone;
        private string description;
        private string priority;
        private DateTime? date_Time_Reg;
        private string state_Status;
        private string status;
        private int isAttached_File;
```

```
        private string project;

        public event PropertyChangedEventHandler PropertyChanged;
        protected void OnChanged([CallerMemberName] string PropertyName = "")
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(PropertyName));
        }
        /// <summary>
        /// Номер
        /// </summary>
        public int? ID { get; set; }
        /// <summary>
        /// Обект учёта
        /// </summary>
        public string Address_object { get => address_object; set { address_object = value; OnChanged(); } }
        /// <summary>
        /// Заявитель
        /// </summary>
        public string Applicant { get => applicant; set { applicant = value; OnChanged(); } }
        /// <summary>
        /// Контактный телефон
        /// </summary>
        public long Number_Phone { get => number_Phone; set { number_Phone = value; OnChanged(); } }
        /// <summary>
        /// Краткое содержание
        /// </summary>
        public string Description { get => description; set { description = value; OnChanged(); } }
        /// <summary>
        /// Приоритет
        /// </summary>
        public string Priority { get => priority; set { priority = value; OnChanged(); } }
        /// <summary>
        /// Дата и время регистрации
        /// </summary>
        public DateTime? Date_Time_Reg { get => date_Time_Reg; set { date_Time_Reg = value; OnChanged(); } }
        /// <summary>
        /// Статус состояния
        /// </summary>
        public string State_Status { get => state_Status; set { state_Status = value; OnChanged(); } }
        /// <summary>
        /// Статус
        /// </summary>
        public string Status { get => status; set { status = value; OnChanged(); } }
        /// <summary>
```

```
            /// Наличие присоединенных файлов
            /// </summary>
            public int IsAttached_File { get => isAttached_File; set { isAttached_File = value;
OnChanged(); } }
            /// <summary>
            /// Название Проекта
            /// </summary>
            public string Project { get => project; set { project = value; OnChanged(); } }

        }
    }

    using System;
    using System.Collections.Generic;
    using System.ComponentModel;
    using System.Linq;
    using System.Runtime.CompilerServices;
    using System.Text;
    using System.Threading.Tasks;

    namespace Diplon.Model
    {
        /*CREATE TABLE "Warehouse" (
                "ID"    INTEGER UNIQUE,
                "Type_Operation"    TEXT,
                "Date_Time_Reg"     TEXT,
                "Name_Resource"     TEXT,
                "Count"         INTEGER,
                "Code_Provider"     TEXT,
                "Name_Provider"     TEXT,
                "User_Name" TEXT,
                PRIMARY KEY("ID" AUTOINCREMENT)
    );*/
        /// <summary>
        /// Модель Склада
        /// </summary>
        public class WarehouseModel : INotifyPropertyChanged
        {
            private int iD;
            private string type_Operation;
            private DateTime? date_Time_Reg;
            private string name_Resource;
            private int count;
            private float? price;
            private string code_Provider;
            private string name_Provider;
            private string user_Name;

            public event PropertyChangedEventHandler PropertyChanged;
            protected void OnChanged([CallerMemberName] string PropertyName = "")
```

```
            {
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(PropertyName));
            }
            /// <summary>
            /// Номер
            /// </summary>
            public int ID { get => iD; set { iD = value; OnChanged(); } }
            /// <summary>
            /// Вид операции
            /// </summary>
            public string Type_Operation { get => type_Operation; set { type_Operation = value; OnChanged(); } }
            /// <summary>
            /// Время регистрации
            /// </summary>
            public DateTime? Date_Time_Reg { get => date_Time_Reg; set { date_Time_Reg = value; OnChanged(); } }
            /// <summary>
            /// Наименование ресурса
            /// </summary>
            public string Name_Resource { get => name_Resource; set { name_Resource = value; OnChanged(); } }
            /// <summary>
            /// Количество
            /// </summary>
            public int Count { get => count; set { count = value; OnChanged(); } }
            /// <summary>
            /// Цена
            /// </summary>
            public float? Price { get => price; set { price = value; OnChanged(); } }
            /// <summary>
            /// Код поставщика
            /// </summary>
            public string Code_Provider { get => code_Provider; set { code_Provider = value; OnChanged(); } }
            /// <summary>
            /// Наименование поставщика
            /// </summary>
            public string Name_Provider { get => name_Provider; set { name_Provider = value; OnChanged(); } }
            /// <summary>
            /// Пользователь
            /// </summary>
            public string User_Name { get => user_Name; set { user_Name = value; OnChanged(); } }
        }
    }

    using Dapper;
```

```
using GalaSoft.MvvmLight;
using GalaSoft.MvvmLight.Command;

using System;
using System.Collections.Generic;
using System.Data.SQLite;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;

namespace Diplon.ViewModel
{
   public class Create_Or_Update_ViewModel:ViewModelBase
   {
      private bool IsCreate = true;
      public Model.RequestModel request { get; set; }
      private readonly IList<Model.RequestModel> AllList;
      SQLiteConnection connection = new SQLiteConnection("Data Source=" +
AppContext.BaseDirectory + @"\Dip.db");
      public Create_Or_Update_ViewModel(IList<Model.RequestModel>
AllList,Model.RequestModel request)
      {
         this.AllList = AllList;
         if (request is null)
         {
            IsCreate = true;
            this.request = new Model.RequestModel(); /*
            {
               Address_object = request.Address_object,
               Applicant = request.Applicant,
               Date_Time_Reg = request.Date_Time_Reg,
               Description = request.Description,
               IsAttached_File = request.IsAttached_File,
               Number_Phone = request.Number_Phone,
               Priority = request.Priority,
               Project = request.Project,
               State_Status = request.State_Status,
               Status = request.Status
            };*/
         }
         else
         {
            IsCreate = false;
            this.request = new Model.RequestModel
            {  ID=request.ID,
               Address_object = request.Address_object,
               Applicant = request.Applicant,
               Date_Time_Reg = request.Date_Time_Reg,
               Description = request.Description,
```

```
                    IsAttached_File = request.IsAttached_File,
                    Number_Phone = request.Number_Phone,
                    Priority = request.Priority,
                    Project = request.Project,
                    State_Status = request.State_Status,
                    Status = request.Status
                };
            }
        }

        public RelayCommand Confirm_Command { get => new RelayCommand(()=> {

            if (IsCreate)
            {
                request.Date_Time_Reg = DateTime.Now;
                AllList.Add(request);
                connection.Query($@"INSERT                                                    INTO
""main"".""RequestModel""(""Address_object"",""Applicant"",""Number_Phone"",""Descriptio
n"",""Date_Time_Reg"",""State_Status"",""Status"",""IsAttached_File"",""Project"")
    VALUES
('{request.Address_object}','{request.Applicant}',{request.Number_Phone},'{request.Descriptio
n}','{request.Date_Time_Reg}','{request.State_Status}','{request.Status}',{request.IsAttached_Fi
le},'{request.Project}');");

                MessageBox.Show("Запись добавлена");
            }
            else
            {
                var r=AllList.FirstOrDefault(x => x.ID == request.ID);
                if (r!=null)
                {
                    r.IsAttached_File = request.IsAttached_File;
                    r.Address_object = request.Address_object;
                    r.Applicant = request.Applicant;
                    r.Date_Time_Reg = request.Date_Time_Reg;
                    r.Description = request.Description;
                    r.IsAttached_File = request.IsAttached_File;
                    r.Number_Phone = request.Number_Phone;
                    r.Priority = request.Priority;
                    r.Project = request.Project;
                    r.State_Status = request.State_Status;
                    r.Status = request.Status;
                }
                connection.Query($@"update ""main"".""RequestModel"" set
    ""Address_object""='{r.Address_object}',""Applicant""='{r.Applicant}',""Number_Phon
e""={r.Number_Phone},""Description""='{r.Description}',""Date_Time_Reg""='{r.Date_Time_
Reg}',
    ""State_Status""='{r.State_Status}',""Status""='{r.Status}',""IsAttached_File""={r.IsAttac
hed_File},""Project""='{r.Project}'
    where ""ID""={r.ID};
```

```
");

                MessageBox.Show("Данные обновлены");
            }


        }); }
    }
}

using Dapper;

using GalaSoft.MvvmLight;
using GalaSoft.MvvmLight.Command;

using System;
using System.Collections.Generic;
using System.Data.SQLite;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;

namespace Diplon.ViewModel
{
    public class Create_Or_Update_W_ViewModel : ViewModelBase
    {
        private bool IsCreate = true;
        public Model.WarehouseModel request { get; set; }
        private readonly IList<Model.WarehouseModel> AllList;
        SQLiteConnection connection = new SQLiteConnection("Data Source=" +
AppContext.BaseDirectory + @"\Dip.db");

        public Create_Or_Update_W_ViewModel(IList<Model.WarehouseModel> AllList,
Model.WarehouseModel request)
        {
            this.AllList = AllList;
            if (request is null)
            {
                IsCreate = true;
                this.request = new Model.WarehouseModel(); /*
                {
                    Address_object = request.Address_object,
                    Applicant = request.Applicant,
                    Date_Time_Reg = request.Date_Time_Reg,
                    Description = request.Description,
                    IsAttached_File = request.IsAttached_File,
                    Number_Phone = request.Number_Phone,
                    Priority = request.Priority,
                    Project = request.Project,
```

```
                State_Status = request.State_Status,
                Status = request.Status
            };*/
        }
        else
        {
            IsCreate = false;
            this.request = new Model.WarehouseModel
            {
                ID = request.ID,
                Code_Provider = request.Code_Provider,
                Count = request.Count,
                Date_Time_Reg = request.Date_Time_Reg,
                Name_Provider = request.Name_Provider,
                Name_Resource = request.Name_Resource,
                Price = request.Price,
                Type_Operation = request.Type_Operation,
                User_Name = request.User_Name
            };
        }
    }

    public RelayCommand Confirm_Command
    {
        get => new RelayCommand(() =>
        {

            if (IsCreate)
            {
                request.Date_Time_Reg = DateTime.Now;
                request.User_Name = Environment.UserName;
                AllList.Add(request);

                connection.Query($@"INSERT INTO ""main"".""Warehouse""(""Type_Operation"",""Date_Time_Reg"",""Name_Resource"",""Count"",""Code_Provider"",""Name_Provider"",""User_Name"") VALUES ('{request.Type_Operation}','{request.Date_Time_Reg}','{request.Name_Resource}',{request.Count},'{request.Code_Provider}','{request.Name_Provider}','{request.User_Name}');");
                MessageBox.Show("Запись добавлена");

            }
            else
            {
                var r = AllList.FirstOrDefault(x => x.ID == request.ID);
                if (r != null)
                {
                    r.Code_Provider = request.Code_Provider;
                    r.Count = request.Count;
                    r.Date_Time_Reg = request.Date_Time_Reg;
                    r.Name_Provider = request.Name_Provider;
```

```
                    r.Name_Resource = request.Name_Resource;
                    r.Price = request.Price;
                    r.Type_Operation = request.Type_Operation;
                    r.User_Name = request.User_Name;
                    connection.Query($@"update ""main"".""Warehouse""
        ""Type_Operation""='{r.Type_Operation}',""Date_Time_Reg""='{r.Date_Time_Reg}',""
Name_Resource""='{r.Name_Resource}',""Count""={r.Count},""Code_Provider""='{r.Code_Pr
ovider}',""Name_Provider""='{r.Name_Provider}',""User_Name""='{r.User_Name}'
        set where ""ID""={r.ID};");
                    MessageBox.Show("Данные обновлены");

                }
            }


        });
    }
  }
}

using Dapper;

using Diplon.Model;

using GalaSoft.MvvmLight;
using GalaSoft.MvvmLight.Command;

using System;
using System.Collections.ObjectModel;
using System.Data.SQLite;
using System.Windows.Data;

namespace Diplon.ViewModel
{
  public class MainViewModel : ViewModelBase
  {
    private RequestModel selectedItem = new Model.RequestModel();
    private WarehouseModel selectedItem_Warehouse = new
Model.WarehouseModel();

    public ObservableCollection<Model.RequestModel> Requests { get; set; } = new
ObservableCollection<Model.RequestModel>();
    public ObservableCollection<Model.WarehouseModel> Warehouses { get; set; } =
new ObservableCollection<Model.WarehouseModel>();
    public Model.RequestModel SelectedItem
    {
      get => selectedItem;

      set
      {
```

```
            selectedItem = value; RaisePropertyChanged();
        }
    }
    public Model.WarehouseModel SelectedItem_Warehouse
    {
        get => selectedItem_Warehouse; set
        {
            selectedItem_Warehouse = value; RaisePropertyChanged();
        }
    }
    public DateTime Date_Start { get; set; }
    public DateTime Date_End { get; set; }
    SQLiteConnection connection = new SQLiteConnection("Data Source="+AppContext.BaseDirectory+ @"\Dip.db");
    public MainViewModel()
    {
        var d = connection.Query<RequestModel>(@"select ""ID"",""Address_object"",""Applicant"",""Number_Phone"",""Description"",""Date_Time_Reg"",""State_Status"",""Status"",""IsAttached_File"",""Project"" from RequestModel e;");
        foreach (var item in d)
        {
            Requests.Add(item);
        }

        var e = connection.Query<WarehouseModel>(@"select ""ID"",""Type_Operation"",""Date_Time_Reg"",""Name_Resource"",""Count"",""Code_Provider"",""Name_Provider"",""User_Name"" from Warehouse;");
        foreach (var item in e)
        {
            Warehouses.Add(item);
        }

        //for (int i = 0; i < 10; i++)
        //{
        //    Requests.Add(new RequestModel { ID = i, Applicant = "Заколюкин Максим", Date_Time_Reg = DateTime.Now });
        //    Warehouses.Add(new WarehouseModel { ID = i, Code_Provider = "Заколюкин", Date_Time_Reg = DateTime.Now });
        //}
        //                                                    Requests = (CollectionView)CollectionViewSource.GetDefaultView(_Requests);
    }
    public RelayCommand Create_Command
    {
        get => new RelayCommand(() =>
        {
            new Create_Or_Update_Request(Requests).ShowDialog();
            Warehouses.Clear();
            Requests.Clear();
```

```
                var d = connection.Query<RequestModel>(@"select ""ID"",""Address_object"",""Applicant"",""Number_Phone"",""Description"",""Date_Time_Reg"",""State_Status"",""Status"",""IsAttached_File"",""Project"" from RequestModel e;");
                    foreach (var item in d)
                    {
                        Requests.Add(item);
                    }

                    var e = connection.Query<WarehouseModel>(@"select ""ID"",""Type_Operation"",""Date_Time_Reg"",""Name_Resource"",""Count"",""Code_Provider"",""Name_Provider"",""User_Name"" from Warehouse;");
                    foreach (var item in e)
                    {
                        Warehouses.Add(item);
                    }
                });
            }
            public RelayCommand Edit_Command
            {
                get => new RelayCommand(() =>
                {
                    new Create_Or_Update_Request(Requests, selectedItem).ShowDialog();
                });
            }
            public RelayCommand Delete_Command
            {
                get => new RelayCommand(() =>
                {

                    connection.Query($@"delete from RequestModel where ""ID""={selectedItem.ID};");
                    Requests.Remove(selectedItem);
                });
            }
            public RelayCommand Create_Warehouse_Command
            {
                get => new RelayCommand(() => { new Create_Or_Update_Request_W(Warehouses).ShowDialog(); Warehouses.Clear();
                    Requests.Clear();
                    var d = connection.Query<RequestModel>(@"select ""ID"",""Address_object"",""Applicant"",""Number_Phone"",""Description"",""Date_Time_Reg"",""State_Status"",""Status"",""IsAttached_File"",""Project"" from RequestModel e;");
                    foreach (var item in d)
                    {
                        Requests.Add(item);
                    }

                    var e = connection.Query<WarehouseModel>(@"select ""ID"",""Type_Operation"",""Date_Time_Reg"",""Name_Resource"",""Count"",""Code_Provider"",""Name_Provider"",""User_Name"" from Warehouse;");
```

```
                foreach (var item in e)
                {
                    Warehouses.Add(item);
                }
            });
        }
        public RelayCommand Edit_Warehouse_Command
        {
            get => new RelayCommand(() => { new Create_Or_Update_Request_W(Warehouses, selectedItem_Warehouse).ShowDialog(); });
        }
        public RelayCommand Delete_Warehouse_Command
        {
            get => new RelayCommand(() =>
            {
                connection.Query($@"delete from RequestModel e where ""ID""={selectedItem_Warehouse.ID};");
                Warehouses.Remove(selectedItem_Warehouse);
            });
        }
        public RelayCommand Update_Command
        {
            get => new RelayCommand(() => {
                Warehouses.Clear();
                Requests.Clear();
                var d = connection.Query<RequestModel>(@"select ""ID"",""Address_object"",""Applicant"",""Number_Phone"",""Description"",""Date_Time_Reg"",""State_Status"",""Status"",""IsAttached_File"",""Project"" from RequestModel e;");
                foreach (var item in d)
                {
                    Requests.Add(item);
                }

                var e = connection.Query<WarehouseModel>(@"select ""ID"",""Type_Operation"",""Date_Time_Reg"",""Name_Resource"",""Count"",""Code_Provider"",""Name_Provider"",""User_Name"" from Warehouse;");
                foreach (var item in e)
                {
                    Warehouses.Add(item);
                }
            });
        }
    }
}


using Dapper;
using Diplon.Model;
using System.Collections.Generic;
using System.Data.SQLite;
```

```
using Xunit;
using Diplon.ViewModel;
using System;

namespace Diplon.Tests
{
   public class RequestModelTests
   {
      private SQLiteConnection CreateInMemoryDatabase()
      {
         var connection = new SQLiteConnection("Data Source=:memory:");
         connection.Open();

         // Создаем таблицы для теста
         connection.Execute(@"
            CREATE TABLE RequestModel (
               ID INTEGER PRIMARY KEY AUTOINCREMENT,
               Address_object TEXT,
               Applicant TEXT,
               Number_Phone INTEGER,
               Description TEXT,
               Date_Time_Reg TEXT,
               State_Status TEXT,
               Status TEXT,
               IsAttached_File INTEGER,
               Project TEXT
            );
         ");
         connection.Execute(@"
            CREATE TABLE Warehouse (
               ID INTEGER PRIMARY KEY AUTOINCREMENT,
               Type_Operation TEXT,
               Date_Time_Reg TEXT,
               Name_Resource TEXT,
               Count INTEGER,
               Code_Provider TEXT,
               Name_Provider TEXT,
               User_Name TEXT
            );
         ");
         return connection;
      }

      [Fact]
      public void Create_Request_Should_InsertDataIntoDatabase()
      {
         var connection = CreateInMemoryDatabase();

         var testRequest = new RequestModel
         {
```

```
            Address_object = "Test Address",
            Applicant = "Test Applicant",
            Number_Phone = 1234567890,
            Description = "Test Description",
            Date_Time_Reg = "20/01/2000",
            State_Status = "New",
            Status = "Pending",
            IsAttached_File = 1,
            Project = "Test Project"
        };

        // Создаем экземпляр ViewModel с использованием подключения
        var viewModel = new ViewModel.Create_Or_Update_ViewModel(new
List<RequestModel>(), null, connection);
        // Добавляем данные через ViewModel
        viewModel.request = testRequest;
        viewModel.Confirm_Command.Execute(null);

        // Проверяем, что запись была добавлена в базу данных
        var result = connection.QueryFirstOrDefault<RequestModel>("SELECT *
FROM RequestModel WHERE Address_object = @Address", new { Address = "Test Address"
});

        // Проверяем результаты
        Assert.NotNull(result);
        Assert.Equal(testRequest.Address_object, result.Address_object);
        Assert.Equal(testRequest.Applicant, result.Applicant);
        Assert.Equal(testRequest.Number_Phone, result.Number_Phone);
        Assert.Equal(testRequest.Description, result.Description);
        Assert.Equal(testRequest.State_Status, result.State_Status);
        Assert.Equal(testRequest.Status, result.Status);
        Assert.Equal(testRequest.Date_Time_Reg, result.Date_Time_Reg);
    }



    [Fact]
    public void Update_Request_Should_UpdateDataInDatabase()
    {
        var connection = CreateInMemoryDatabase();

        // Добавляем начальную запись
        var initialRequest = new RequestModel
        {
            Address_object = "Old Address",
            Applicant = "Old Applicant",
            Number_Phone = 987654210,
            Description = "Old Description",
            Date_Time_Reg = "2024-11-16 12:34:56",
            State_Status = "Old Status",
```

```
            Status = "Old Status",
            IsAttached_File = 1,
            Project = "Old Project"
        };

        connection.Execute(@"
    INSERT INTO RequestModel (Address_object, Applicant, Number_Phone, Description, Date_Time_Reg, State_Status, Status, IsAttached_File, Project)
    VALUES (@Address_object, @Applicant, @Number_Phone, @Description, @Date_Time_Reg, @State_Status, @Status, @IsAttached_File, @Project)",
            initialRequest);

        // Обновляем запись через команду
        var updatedRequest = new RequestModel
        {
            ID = 1,
            Address_object = "Updated Address",
            Applicant = "Updated Applicant",
            Number_Phone = 1234567890,
            Description = "Updated Description",
            Date_Time_Reg = "2024-11-17 12:34:56",
            State_Status = "Updated Status",
            Status = "Updated Status",
            IsAttached_File = 1,
            Project = "Updated Project"
        };

        // Обновляем запись в базе данных
        connection.Execute(@"
    UPDATE RequestModel
    SET Address_object = @Address_object,
        Applicant = @Applicant,
        Number_Phone = @Number_Phone,
        Description = @Description,
        Date_Time_Reg = @Date_Time_Reg,
        State_Status = @State_Status,
        Status = @Status,
        IsAttached_File = @IsAttached_File,
        Project = @Project
    WHERE ID = @ID", updatedRequest);


        var result = connection.QueryFirstOrDefault<RequestModel>(
            "SELECT * FROM RequestModel WHERE ID = @ID",
            new { ID = 1 }
        );


        Assert.NotNull(result);
        Assert.Equal(updatedRequest.Address_object, result.Address_object);
```

```
            Assert.Equal(updatedRequest.Applicant, result.Applicant);
            Assert.Equal(updatedRequest.Number_Phone, result.Number_Phone);
            Assert.Equal(updatedRequest.Description, result.Description);
            Assert.Equal(updatedRequest.Date_Time_Reg, result.Date_Time_Reg); // Проверяем, что дата обновилась
        }


        [Fact]
        public void Delete_Request_Should_RemoveDataFromDatabase()
        {

            var connection = CreateInMemoryDatabase();

            // Создаем начальную запись
            var initialRequest = new RequestModel
            {
                Address_object = "Test Address",
                Applicant = "Test Applicant",
                Number_Phone = 1234567890,
                Description = "Test Description",
                Date_Time_Reg = "2024-11-16 12:34:56",
                State_Status = "Test Status",
                Status = "Test Status",
                IsAttached_File = 1,
                Project = "Test Project"
            };

            connection.Execute(@"
        INSERT INTO RequestModel (Address_object, Applicant, Number_Phone, Description, Date_Time_Reg, State_Status, Status, IsAttached_File, Project)
        VALUES (@Address_object, @Applicant, @Number_Phone, @Description, @Date_Time_Reg, @State_Status, @Status, @IsAttached_File, @Project)",
                initialRequest);


            var idToDelete = connection.QueryFirst<int>("SELECT ID FROM RequestModel WHERE Address_object = @Address", new { Address = "Test Address" });

            var viewModel = new MainViewModel();

            // Находим запись
            var requestToDelete = connection.QueryFirst<RequestModel>("SELECT * FROM RequestModel WHERE ID = @ID", new { ID = idToDelete });
            viewModel.SelectedItem = requestToDelete;

            // Выполняем команду удаления
            viewModel.Delete_Command.Execute(null);
```

```
                // Создаем новое соединение для проверки, что запись удалена
                using (var checkConnection = new SQLiteConnection("Data Source=" +
AppContext.BaseDirectory + @"\Dip.db"))
                {
                    checkConnection.Open();

                    // Проверяем, что запись с данным ID больше не существует в базе
                    var result = checkConnection.QueryFirstOrDefault<RequestModel>("SELECT
* FROM RequestModel WHERE ID = @ID", new { ID = idToDelete });

                    // Запись должна быть null, так как она была удалена
                    Assert.Null(result);
                }

                Assert.DoesNotContain(viewModel.Requests, r => r.ID == idToDelete);
            }
        }



    }
```