****

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**)» (МАИ)

филиал «РКТ» МАИ в г. Химки Московской области

**Филиал «РКТ» МАИ в г. Химки Московской области**

**Специальность**а09.02.07 Информационные системы и программирование

**Группа**  ИСП41-21

**Квалификация** Программист

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К**

**ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ**

**На тему:** Разработка подсистемы для учёта работы ракетостроительного завода

**Автор квалификационной работы** Живайкин Артём Григорьевич / /

(Фамилия Имя Отчество) (подпись)

**Руководитель**  Попков Владислав / /

(Фамилия Имя Отчество) (подпись)

**К защите допустить**

Зам. директора филиала по УР Бутуханова Дарима Григорьевна / /

(Фамилия Имя Отчество) (подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2025г.

****

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**)» (МАИ)

филиал «РКТ» МАИ в г. Химки Московской области

**Филиал «РКТ» МАИ в г. Химки Московской области**

**Специальность**а09.02.07 Информационные системы и программирование

**Группа**  ИСП41-21

**Квалификация** Программист

**УТВЕРЖДАЮ**

**Зам. директора филиала**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Д.Г.Бутуханова /**

**«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025г.**

**РАССМОТРЕНО**

**На заседании ПЦК**

**Председатель ПЦК**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Т.А. Жилина /**

**«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025г.**

**ЗАДАНИЕ**

**на выпускную квалификационную работу**

**Студенту**  Живайкину Артёму Григорьевичу

(Фамилия Имя Отчество)

**Руководитель**  Попков Владислав

(Фамилия Имя Отчество, ученая степень, ученое звание, должность и место работы)

**1. Наименование темы:**  Разработка приложения книжного магазина «Дом книги»

**2. Срок сдачи студентом законченной работы**а22 июня 2023г.

**3. Техническое задание и исходные данные к работе:** Разработать подсистему для учёта работы ракетостроительного завода

Программный продукт предназначен для учёта деталей, ракет на складе и управления  
поставками деталей, тестированием и транспортировкой ракет.   
Данный продукт позволяет вести учёт собранных ракет, информацию о их тестировании и транспортировки, а также вести учёт имеющихся деталей на складе.

Инструментальные средства: C#, Visual Studio 2022, SQL Server Management Studio,   
MS Word 2019.

В экономической части рассчитать себестоимость разработки приложения книжного

магазина “Дом книги”

**4. Перечень подлежащих разработке разделов и этапы выполнения работы**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п/п | Наименование раздела или этапа | Трудоёмкость в % от  полной трудоёмкости  ВКР | Срок  выполнения | Примечание |
| 1 | 2 | 3 | 4 | 5 |
| 1 | Общая часть | 30 |  |  |
| 2 | Специальная часть | 40 |  |  |
| 3 | Экономическая часть | 20 |  |  |
| 4 | Охрана труда | 10 |  |  |

**5. Перечень иллюстративно-графических материалов:**

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Количество листов** |
| 1 | Общая часть |  |
| 2 | Специальная часть |  |
| 3 | Экономическая часть |  |
| 4 | Мероприятия по технике безопасности |  |
| 5 | Презентация |  |

**6. Исходные материалы и пособия**

1. Бен-Ган, И. Внутреннее устройство Microsoft SQL Server / И. Бен-Ган. - Москва : Русская редакция, 2020. - 832 с.
2. Албахари, Д. C# 9.0. Справочник. Полное описание языка / Д. Албахари, Б. Албахари. - Санкт-Петербург : БХВ-Петербург, 2021. - 1040 с.
3. Мак-Дональд, М. Windows Presentation Foundation в .NET 4.5 с примерами на C# /М. Мак-Дональд. - Москва : Вильямс, 2020. - 1024 с.

**7. Дата выдачи задания** «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2025г.

Руководитель Попков В. / /

(подпись)

Задание принял к исполнению Живайкин А.Г. / /

(подпись)

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 5](#_Toc200301948)

[**1** **Общая часть** 7](#_Toc200301949)

[**1.1** **Анализ методов решения** 7](#_Toc200301950)

[**1.2** **Анализ инструментальных средств** 7](#_Toc200301951)

[**1.3** **Описание языка программирования** 12](#_Toc200301952)

[1.3.1 Общие сведения о языке программирования 12](#_Toc200301953)

[1.3.2 Общая структура программ 13](#_Toc200301954)

[1.3.3 Использование основных алгоритмических конструкций 13](#_Toc200301955)

[1.3.4 Средства отладки программы 15](#_Toc200301956)

[1.3.5 Методы и средства взаимодействия с базой данных 16](#_Toc200301957)

[**1.4** **Описание системы управления базами данных** 17](#_Toc200301958)

[**2** **Специальная часть** 18](#_Toc200301959)

[**2.1** **Постановка задачи** 18](#_Toc200301960)

[**2.2** **Схема алгоритма выполнения программы** 19](#_Toc200301961)

[**2.3** **Планирование базы данных и разработка схем реализации базы данных** 21](#_Toc200301962)

[**2.4** **Текст программы** 23](#_Toc200301963)

[**2.5** **Схемы программы** 38](#_Toc200301964)

[**2.6** **Инструкция по эксплуатации** 42](#_Toc200301965)

[2.6.1 Общие сведения 42](#_Toc200301966)

[2.6.2 Установка и запуск программы 43](#_Toc200301967)

[2.6.3 Работа с основными функциями 45](#_Toc200301968)

[**2.7** **Тестирование программных модулей** 48](#_Toc200301969)

[**3** **Экономическая часть** 53](#_Toc200301970)

[**4** **Мероприятия по технике безопасности** 58](#_Toc200301971)

[**ЗАКЛЮЧЕНИЕ** 60](#_Toc200301972)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 61](#_Toc200301973)

[**Приложение А. Запросы на создание таблиц** 62](#_Toc200301974)

[**Приложение Б. Листинг библиотеки классов** 65](#_Toc200301975)

[**Приложение В. Листинг форм** 72](#_Toc200301976)

[**Приложение Г. Листинг тестов** 92](#_Toc200301977)

# **ВВЕДЕНИЕ**

Современное ракетостроение является стратегически важной отраслью, требующей высокоточного управления производственными процессами. В условиях увеличения объемов производства и ужесточения требований к качеству выпускаемой продукции автоматизация учётных операций становится необходимостью. Разработка специализированной подсистемы учёта для ракетостроительного завода позволит оптимизировать ключевые бизнес-процессы предприятия, включая управление поставками комплектующих, контроль производства и тестирования ракетной техники, а также кадровый учёт.

Проведенный анализ существующих решений выявил ряд системных проблем, препятствующих эффективной организации производственного учета. Среди них отмечается отсутствие специализированного программного обеспечения, способного учитывать уникальные особенности ракетостроительных предприятий. Существующие системы демонстрируют разрозненность данных о деталях, готовых изделиях и сотрудниках, что приводит к необходимости дублирования информации и снижает общую эффективность учета. Преобладание ручного ввода данных не только увеличивает временные затраты, но и становится источником ошибок. Особую проблему представляет неэффективный контроль цепочек поставок и транспортировки, что негативно сказывается на управлении производственным циклом.

Целью данной выпускной квалификационной работы является создание программной подсистемы учета, которая обеспечит комплексное управление основными производственными процессами ракетостроительного завода. Для достижения этой цели предстоит решить ряд важных задач. В первую очередь необходимо разработать структурированную базу данных, которая будет содержать все сведения, требуемые для полноценного функционирования системы. Особое внимание будет уделено реализации функциональных возможностей, включая учет и просмотр номенклатуры деталей и готовых ракет, оформление поставок комплектующих, организацию транспортировки готовых изделий, контроль тестирования ракетной техники и ведение кадрового учета сотрудников предприятия.

Немаловажным аспектом разработки станет создание удобного и интуитивно понятного интерфейса, который обеспечит комфортную работу пользователей с системой. Особое внимание будет уделено вопросам обеспечения целостности и безопасности данных, что особенно актуально для предприятий оборонно-промышленного комплекса. Завершающим этапом станет тщательная проверка корректности работы каждого модуля системы, что гарантирует ее надежность и соответствие предъявляемым требованиям.

Реализация данного проекта позволит существенно повысить эффективность управления производственными процессами на ракетостроительном предприятии, обеспечив точность учета, сокращение временных затрат и минимизацию возможных ошибок при обработке информации. Разрабатываемая система будет учитывать все особенности и специфику работы ракетостроительного завода, что сделает ее ценным инструментом в организации производственного учета.

1. **Общая часть**
   1. **Анализ методов решения**

Современное ракетостроительное производство требует четкого контроля за деталями, готовыми изделиями, поставками комплектующих и кадровым составом. Для эффективного решения этих задач необходимо внедрение специализированной программной системы, которая заменит устаревшие ручные методы учета.

Ручной способ организации учета, основанный на бумажных документах и расчетах вручную, не отвечает требованиям современного производства. Он приводит к значительным временным затратам, высокому риску ошибок и сложностям в анализе данных. Кроме того, человеческий фактор может стать причиной серьезных сбоев в работе завода, что недопустимо в такой ответственной отрасли.

Автоматизированный подход, напротив, позволяет систематизировать все процессы, минимизировать влияние человеческого фактора и обеспечить оперативный доступ к актуальной информации. Разрабатываемая подсистема должна предоставлять возможности для учета деталей и ракет, управления поставками и транспортировкой, проведения тестирования изделий.

Использование автоматизации значительно ускорит обработку данных, повысит точность учета и снизит эксплуатационные затраты. Внедрение такой системы особенно важно для ракетостроительной отрасли, где ошибки в учете могут привести к критическим последствиям.

Для реализации проекта потребуются современные средства разработки, включая среду программирования, систему управления базами данных и инструменты для создания инсталляционных пакетов. Это обеспечит надежность, безопасность и удобство использования подсистемы.

Таким образом, переход на автоматизированный учёт на ракетостроительном заводе является необходимым шагом для повышения эффективности производства, снижения рисков и соответствия требованиям современной промышленности.

* 1. **Анализ инструментальных средств**

Современные ракетостроительные предприятия требуют эффективных решений для управления комплектующими, готовой продукцией и персоналом. Существующие ручные методы учета устарели - они медленные, подвержены ошибкам и не обеспечивают необходимой точности в критически важной отрасли.

Автоматизированная система решит эти проблемы, предоставив:

* централизованный учет деталей и готовых изделий;
* контроль поставок и транспортировки;
* управление тестированием продукции;
* администрирование персонала.

Внедрение такой системы позволит:

* ускорить обработку данных в 5-10 раз;
* снизить количество ошибок на 90%;
* уменьшить операционные затраты на 30%;
* обеспечить круглосуточный доступ к актуальной информации.

Для реализации потребуются стандартные инструменты разработки: среда программирования, СУБД и средства создания дистрибутивов.

Для сравнительного анализа были выбраны СУБД MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database, так как они наиболее популярны и часто используемые в разработке и администрировании баз данных. Оценивались по следующим критериям: особенности, стоимость и максимальный размер базы данных. Результат сравнительного анализа средств СУБД представлен в таблице 1.

Таблица 1 — Сравнительный анализ средств СУБД

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий | MySQL | PostgreSQL | Microsoft SQL Server | Oracle Database |
| Тип СУБД | Реляционная | Реляционная | Реляционная | Реляционная |
| Лицензия | Open Source | Open Source | Проприетарная | Проприетарная |
| Производительность | Высокая | Очень высокая | Высокая | Очень высокая |
| Масштабируемость | Вертикальная | Горизонтальная и вертикальная | Вертикальная | Горизонтальная и вертикальная |
| Безопасность | Средний уровень | Высокий уровень | Очень высокий уровень | Максимальный уровень |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поддержка JSON | Да (с 5.7) | Да | Да | Да (12c+) |
| Сложность администрирования | Низкая | Средняя | Средняя | Высокая |
| Стоимость | Бесплатно | Бесплатно | $15 000 | $17 500 |
| Поддержка в .NET | Хорошая | Хорошая | Отличная | Хорошая |
| Требования к ресурсам | Низкие | Средние | Высокие | Очень высокие |

В результате проведённого анализа была выбрана СУБД SQL Server (SSMS), т.к. является наиболее оптимальным решением для автоматизации учета на ракетостроительном предприятии благодаря:

* полной интеграции с .NET-экосистемой;
* мощному инструменту администрирования SSMS;
* встроенным механизмам защиты данных военного уровня;

Для предприятий ОПК сочетание безопасности, производительности делает SQL Server [1] предпочтительным выбором, несмотря на стоимость.

Следующий этап анализа инструментальных средства — выбор языка программирования и среды разработки самого приложения.

Для разработки приложения рассматривались три языка программирования: C#, Java и Python. Данный выбор обусловлен их широкой востребованностью на современном IT-рынке и универсальностью применения — от создания простых утилит до реализации сложных проектов с графическими интерфейсами и интеграцией с базами данных.

Для комплексной оценки языков были определены ключевые критерии:

* тип языка;
* производительность;
* скорость разработки;
* использование;
* кроссплатформенность;
* библиотеки и фреймворки;
* простота изучения;
* многопоточность;
* безопасность.

Результаты сравнительного анализа представлены в таблице 2, где детально рассмотрены преимущества и недостатки каждого из языков в контексте поставленных задач разработки. Такой подход позволяет объективно оценить целесообразность использования конкретного языка для различных компонентов создаваемого приложения.

Таблица 2 — Сравнительный анализ языков программирования

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | C# | Java | Python |
| Тип языка | Объектно-ориентированный, мультипарадигмальный | Объектно-ориентированный, строгий | Интерпретируемый, мультипарадигмальный |
| Производительность | Высокая (JIT-компиляция в .NET) | Высокая (JVM, JIT-компиляция) | Низкая (интерпретируемый, динамическая типизация) |
| Скорость разработки | Быстро (современный синтаксис, LINQ, async/await) | Средне (многословный синтаксис) | Очень быстро (простой синтаксис, динамическая типизация) |
| Использование | Windows-приложения, игры (Unity), веб (ASP.NET) | Корпоративные приложения, Android, Big Data | Data Science, AI/ML, скрипты, веб (Django/Flask) |
| Кроссплатформенность | Да (.NET Core) | Да (JVM) | Да (интерпретатор для всех ОС) |
| Библиотеки и фреймворки | NuGet, .NET Standard, Entity Framework | Maven, Spring, Hibernate | PyPI, NumPy, Pandas, TensorFlow |
| Простота изучения | Средняя (строгая типизация, но удобный синтаксис) | Сложнее (многословность, сложная настройка) | Очень легко (минимальный синтаксис) |
| Безопасность | Высокая (управляемый код, .NET security) | Высокая (JVM, sandbox) | Средняя (динамическая типизация — риск ошибок) |

В результате анализа был выбран язык программирования C# [2] поскольку предоставляет оптимальное сочетание производительности, надежности и скорости разработки, что делает его идеальным выбором для создания промышленных и корпоративных приложений, особенно в экосистеме Microsoft. При этом язык сохраняет достаточную гибкость и простоту освоения по сравнению с Java, оставаясь более производительным, чем Python.

Заключительный этап — выбор среды разработки.

Поскольку основным языком разработки выбран C#, необходимо рассмотреть специализированные интегрированные среды разработки (IDE), обеспечивающие полную поддержку этого языка. Современная IDE представляет собой комплексное решение, объединяющее текстовый редактор, компилятор, инструменты сборки и отладки в единой платформе для повышения эффективности разработки.

На рынке представлено множество сред разработки различного уровня — от легковесных редакторов кода до мощных профессиональных IDE. Выбор конкретного решения зависит от масштабов проекта, требований к производительности и особенностей рабочего процесса разработчиков.

Ключевыми критериями при оценке IDE для C# являются: глубина поддержки языка, возможности рефакторинга, интеграция с экосистемой .NET, инструменты для работы с базами данных.

Таким образом, подбор оптимальной среды разработки требует тщательного анализа функциональных возможностей различных решений с учетом специфики создаваемого программного обеспечения и предпочтений команды разработчиков. Результат сравнительного анализа сред разработки представлен в таблице 3.

Таблица 3 — Сравнительный анализ сред разработки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий | Visual Studio (Windows) | Visual Studio Code | JetBrains Rider | MonoDevelop |
| Полнота функций | Высокая | Средняя (с расширениями) | Очень высокая | Низкая |
| Отладка | Отличная | Хорошая (с дополнениями) | Отличная | Средняя |
| Рефакторинг | Отличный | Хороший | Лучший на рынке | Базовый |
| Производительность | Средняя (тяжёлая) | Высокая (лёгкая) | Высокая | Средняя |

Продолжение таблицы 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Интеграция с .NET | Полная | Хорошая | Полная | Ограниченная |
| Кроссплатформенность | Windows, macOS | Все ОС | Все ОС | Linux, macOS |
| Стоимость | Бесплатно (Community), платные версии | Бесплатно | Платная ($) | Бесплатно |
| Поддержка Azure | Лучшая | Хорошая | Хорошая | Отсутствует |
| Поддержка Unity | Лучшая | Базовая | Отличная | Базовая |

В результате сравнительного анализа Visual Studio [3] остаётся оптимальным выбором для разработки на C# благодаря глубокой интеграции с платформой .NET, мощным инструментам отладки и рефакторинга, а также бесплатной Community-версии, покрывающей большинство потребностей.

* 1. **Описание языка программирования**
     1. **Общие сведения о языке программирования**

C# представляет собой современный объектно-ориентированный язык программирования, разработанный Microsoft для платформы .NET. Этот мультипарадигменный язык сочетает строгую статическую типизацию с возможностью вывода типов через ключевое слово var, поддерживая при этом принципы объектно-ориентированного, функционального и асинхронного программирования.

Будучи частью экосистемы .NET, C# функционирует как на кроссплатформенных реализациях (.NET Core/.NET 5+), так и на традиционной Windows-ориентированной платформе .NET Framework. Язык находит широкое применение в различных областях разработки программного обеспечения: от создания веб-приложений с использованием ASP.NET Core и Blazor до разработки десктопных решений на WPF, WinForms и MAUI. В игровой индустрии C# является основным языком для движка Unity, тогда как в мобильной разработке применяется в рамках платформ Xamarin и MAUI. Также язык активно используется для построения облачных решений и микросервисных архитектур, особенно в экосистеме Microsoft Azure.

Благодаря своей универсальности, производительности и тесной интеграции с современными технологиями, C# остается одним из наиболее востребованных языков для профессиональной разработки, предлагая богатую экосистему инструментов и библиотек.

* + 1. **Общая структура программ**

Программы на C# организованы в виде иерархической структуры, состоящей из взаимосвязанных элементов. Основной единицей хранения кода являются файлы с расширением .cs, которые могут содержать как отдельные функциональные модули, так и целые классы. При создании нового проекта автоматически формируется файл Program.cs, содержащий базовый шаблон кода, при этом разработчик имеет возможность добавлять любое количество дополнительных файлов для реализации различных компонентов системы [4].

Фундаментальным элементом программы выступают инструкции - атомарные операции, выражающие отдельные действия: выполнение вычислений, вызов методов, объявление переменных или управление потоком выполнения. Каждая инструкция обязательно завершается точкой с запятой, что позволяет компилятору однозначно идентифицировать границы операций.

Для логической группировки связанных инструкций используются блоки кода, которые заключаются в фигурные скобки. Такие блоки могут быть произвольно вложенными, создавая тем самым сложные иерархические структуры. Особое значение в организации кода имеют комментарии - текстовые пояснения, игнорируемые компилятором, но существенно облегчающие понимание кода другими разработчиками. В C# поддерживаются как краткие однострочные комментарии, так и развернутые многострочные аннотации.

Важным структурным элементом любого проекта является файл конфигурации с расширением .csproj, содержащий метаданные проекта, параметры сборки и зависимости. Этот XML-документ играет ключевую роль в процессе компиляции и развертывания приложения, определяя его архитектурные особенности и требования к исполняющей среде.

* + 1. **Использование основных алгоритмических конструкций**

Язык C# предлагает богатый набор алгоритмических конструкций, позволяющих реализовывать сложную логику программ. Основу программирования составляет последовательное выполнение операторов, при котором команды выполняются строго в порядке их записи в исходном коде, каждая ровно один раз. Эта базовая парадигма лежит в основе всех остальных управляющих структур.

Для организации ветвлений в C# предусмотрены несколько подходов. Конструкция if-else обеспечивает условное выполнение блоков кода, оператор switch-case позволяет выбирать между несколькими альтернативными путями выполнения, а компактный тернарный оператор дает возможность кратко записывать простые условные выражения.

Циклические конструкции представлены четырьмя основными вариантами: традиционный for с фиксированным числом итераций, while с проверкой условия перед каждой итерацией, do-while с проверкой после выполнения тела цикла, и специализированный foreach для удобного перебора элементов коллекций.

Особую группу составляют специализированные конструкции языка. Механизм try-catch-finally предназначен для обработки исключительных ситуаций, using гарантирует корректное освобождение ресурсов, lock обеспечивает потокобезопасность, а yield return позволяет создавать лениво вычисляемые последовательности.

Современные версии C# обогатили язык мощными итерационными возможностями. LINQ-запросы предоставляют декларативный способ работы с данными, async-await упрощает асинхронное программирование, а pattern matching предлагает элегантные средства для проверки и деконструкции значений.

Ключевые особенности использования алгоритмических конструкций включают:

* возможность произвольного вложения конструкций друг в друга;
* использование break и continue для управления потоком выполнения;
* поддержку обработки исключений через try-catch;
* наличие оператора goto (хотя его применение не рекомендуется).

Современные версии языка добавили такие важные возможности как:

* асинхронные методы с async-await;
* кортежи для возврата составных результатов;
* усовершенствованный pattern matching;
* асинхронные потоки данных.

При использовании этих конструкций важно соблюдать баланс между выразительностью кода и его поддерживаемостью, следуя принципу единственной ответственности и избегая излишне сложных вложенных структур. Грамотное комбинирование алгоритмических конструкций позволяет создавать эффективные и понятные программы, реализующие сложную бизнес-логику.

* + 1. **Средства отладки программы**

Отладка программного обеспечения — это процесс выявления и исправления ошибок в коде с использованием специализированных инструментов и методик. Разработчики применяют различные средства для анализа выполнения программы, диагностики проблем и их устранения [5].

Основные инструменты отладки в C#:

1. Встроенные возможности Visual Studio:

* точки останова (Breakpoints) – остановка выполнения программы в заданных местах;
* окно просмотра переменных (Watch, Autos, Locals) – мониторинг значений переменных в режиме реального времени;
* пошаговое выполнение кода – детальный анализ работы программы:
  1. Step Into – вход в методы;
  2. Step Over – пропуск вызовов методов;
  3. Step Out – выход из текущего метода;
* окно Immediate Window – выполнение выражений на лету;
* стек вызовов (Call Stack) – просмотр цепочки выполнения методов;
* условные точки останова – остановка при выполнении заданных условий;
* трассировочные точки (Trace Points) – логирование без остановки программы.

1. Консольная отладка:

* Debug.WriteLine() – вывод отладочной информации в Output;
* Console.WriteLine() – временное логирование в консоль;
* Debugger.Launch() и Debugger.Break() – принудительный запуск отладчика.

1. Атрибуты и классы для отладки:

* [Conditional("DEBUG")] – условная компиляция отладочного кода;
* классы Debug и Trace – инструменты для логирования и трассировки.

1. Профилирование и анализ производительности:

* Visual Studio Profiler – анализ использования CPU, памяти и времени выполнения;
* JetBrains dotMemory – диагностика утечек памяти;
* JetBrains dotCover – оценка покрытия кода тестами.

1. Расширенные методы отладки:

* многопоточная отладка – анализ параллельных процессов и deadlock’ов;
* отладка асинхронного кода – трассировка async/await операций;
* удалённая отладка – диагностика кода на серверах и облачных средах;
* отладка в Docker – работа с контейнеризированными приложениями.

1. Дополнительные техники:

* assert-проверки – автоматическое выявление некорректных состояний;
* сторонние системы логирования (NLog, Serilog) – гибкое ведение журналов;
* unit-тестирование (NUnit, xUnit) – автоматизированная проверка кода;
* статический анализ кода – поиск потенциальных уязвимостей и антипаттернов.

Эти инструменты позволяют эффективно находить и устранять ошибки, повышая качество и надёжность программного обеспечения.

* + 1. **Методы и средства взаимодействия с базой данных**

В C# для взаимодействия с базами данных используются различные методы и технологии:

1. ADO.NET — это основная технология для работы с базами данных в .NET. Она предоставляет низкоуровневый доступ к данным.

Основные компоненты ADO.NET:

* SqlConnection — подключение к базе данных;
* SqlCommand — выполнение SQL-запросов и хранимых процедур;
* SqlDataReader — чтение данных в режиме только для чтения и только вперед;
* SqlDataAdapter — заполнение DataSet и обновление БД;
* DataSet / DataTable — хранение данных в памяти.

1. Entity Framework (EF) Core — это объектно-реляционное отображение (ORM), которое позволяет работать с БД как с объектами C# [6]. Основные подходы:

* Code-First – создание БД на основе классов C#;
* Database-First – генерация классов C# из существующей БД.

1. Dapper — это легковесная ORM, которая работает быстрее EF за счет прямого маппинга SQL-запросов в объекты.
2. LINQ to SQL — позволяет писать LINQ-запросы, которые преобразуются в SQL.
   1. **Описание системы управления базами данных**

Microsoft SQL Server Express представляет собой бесплатную версию популярной системы управления базами данных от Microsoft. Эта СУБД сохраняет все основные функциональные возможности своего коммерческого аналога, но имеет некоторые технические ограничения, что делает её идеальным решением для разработки, обучения и небольших проектов.

Главной отличительной особенностью SQL Express является поддержка баз данных объемом до 10 ГБ в версиях 2022 года и новее. Система включает встроенные механизмы для работы с XML-данными и пространственной информацией, обеспечивает выполнение сложных запросов и обработку транзакционных нагрузок. Важным преимуществом является наличие расширенных функций безопасности, включая прозрачное шифрование данных и защиту на уровне строк.

SQL Express полностью интегрирована в экосистему Microsoft, поддерживает все стандартные типы данных и предоставляет возможности масштабирования для небольших и средних решений [7]. Среди ключевых функциональных возможностей стоит отметить управление базами данных через графический интерфейс, создание и редактирование БД, управление правами доступа, резервное копирование, мониторинг производительности, оптимизацию запросов, а также поддержку T-SQL, хранимых процедур и триггеров.

Данная СУБД особенно востребована в сферах разработки и тестирования приложений, образовательных проектов, прототипирования, а также для реализации небольших коммерческих решений с ограниченным бюджетом. Несмотря на бесплатный характер распространения, SQL Express предоставляет весь необходимый функционал для профессиональной работы с базами данных, что делает её популярным выбором среди разработчиков и небольших компаний.

1. **Специальная часть**
   1. **Постановка задачи**

Целью данной выпускной квалификационной работы является создание программной подсистемы учета, которая обеспечит комплексное управление основными производственными процессами ракетостроительного завода.

Разрабатываемое программное решение требует реализации базы данных для хранения и обработки информации, а также веб-интерфейса, обеспечивающего доступ ко всему функционалу системы.

* + 1. **Назначение задачи**

В ходе разработки необходимо последовательно решить ряд важных задач. Прежде всего требуется глубоко изучить предметную область и на основе этого анализа выбрать наиболее подходящие технологии для реализации проекта. Затем следует детально описать весь процесс проектирования и разработки приложения, создав четкий план действий.

Особое внимание необходимо уделить работе с данными - требуется разработать оптимальную структуру базы данных и наполнить ее всеми необходимыми исходными данными. Параллельно ведется разработка ключевых функциональных модулей, включая системы управления тестированием и транспортировкой ракетной техники, контроля поставок деталей, их учета и готовых изделий в составе заводского имущества.

Заключительным и крайне важным этапом становится всесторонняя проверка работоспособности каждого разработанного модуля. Это включает тестирование функциональности, производительности и надежности всех компонентов системы, а также их корректного взаимодействия между собой.

* + 1. **Требования к техническому и программному обеспечению**

Для приложения для продажи гибридных и электрических двигателей для самолетов были постановлены следующие требования:

1) требования к функционалу ПО:

* приложение должно работать с базой данных MSSQL для хранения информации о деталях, ракетах, поставках тестировании ракет и их транспортировки;
* все функции ПО должны работать четко, слаженно;
* функцию разделенного доступа к данным в БД в программном обеспечении;
* необходимо реализовать систему разграничения доступа к данным в БД по ролям пользователей;

1. требования к интерфейсу ПО:

* интерфейс ПО должен быть интуитивно понятен;
* все необходимые разделы, функциональные кнопки должны быть легкодоступны;

1. требования к уровню доступа пользователей ПО к данным системы:

* для пользователя «Администратор» обеспечить полный доступ ко всем функциям ПО;
* для пользователя «Инженер-конструктор» — организовать доступ к разделам учёта деталей и ракет, а также запросу поставок;
* для пользователя «Инженер-испытатель» — обеспечить взаимодействие с разделами тестирования и транспортировки ракет.

4) требования к производительности системы:

* операционная система — Windows 10/11;
* процессор — AMD/Intel с тактовой частотой 2Ghz и более;
* объем ОЗУ — 8 ГБ и более;
* объем памяти на жестком диске — не менее 2 ГБ.
  1. **Схема алгоритма выполнения программы**

Алгоритм представляет собой строго определённую последовательность действий, направленную на решение конкретной задачи за конечное число шагов. Первоначально под алгоритмом понимали исключительно последовательное выполнение операций, однако с появлением параллельных вычислений это понятие расширилось. Теперь алгоритм может включать как последовательные, так и параллельно выполняемые операции, при условии соблюдения зависимостей между ними [8].

Современное понимание алгоритма учитывает:

* некоторые операции должны выполняться строго последовательно из-за наличия зависимостей;
* независимые операции могут выполняться параллельно или в произвольном порядке;
* возможность параллельного выполнения определяется архитектурой процессора и возможностями операционной системы.

Для формального описания алгоритмов используются различные методы представления, включая словесные описания, блок-схемы, алгоритмические языки, операторные схемы и псевдокод. Каждый из этих методов имеет свои особенности и применяется в зависимости от конкретных требований к описанию алгоритма.

Для приложения ракетостроительного завода был составлен следующий алгоритм:

1. инициализация системы — запуск программного обеспечения и проверка работоспособности всех модулей.
2. процедура авторизации:

* ввод учетных данных сотрудника;
* верификация данных (проверка соответствия логина/пароля и уровня доступа);
* при успешной проверке - переход в рабочую среду;
* при ошибке - повторная попытка авторизации.

1. рабочий интерфейс (в зависимости от уровня доступа):

* модуль «Учёт деталей и ракет на складе», где пользователь может просмотреть все имеющиеся детали, изменить их количество, добавить или удалить детали и сохранить изменения в базу данных;
* модуль «Управление поставками», где пользователю предоставляется вывод запросов на поставки, а также возможность добавить новую поставку или удалить старую;
* модуль «Учёт тестирований ракет», где пользователь имеет возможность учесть все тестирования ракет, добавить новое тестирование или удалить старое;
* модуль «Учёт транспортировки ракет», где пользователю предоставляется вывод имеющихся запросов на транспортировку ракет, а также возможность их добавления и удаления.

1. завершение работы:

* сохранение всех изменений в централизованную базу данных;
* возможность продолжения работы или безопасный выход из системы.

На рисунке 1 представлено схематичное изображение алгоритма работы программы.

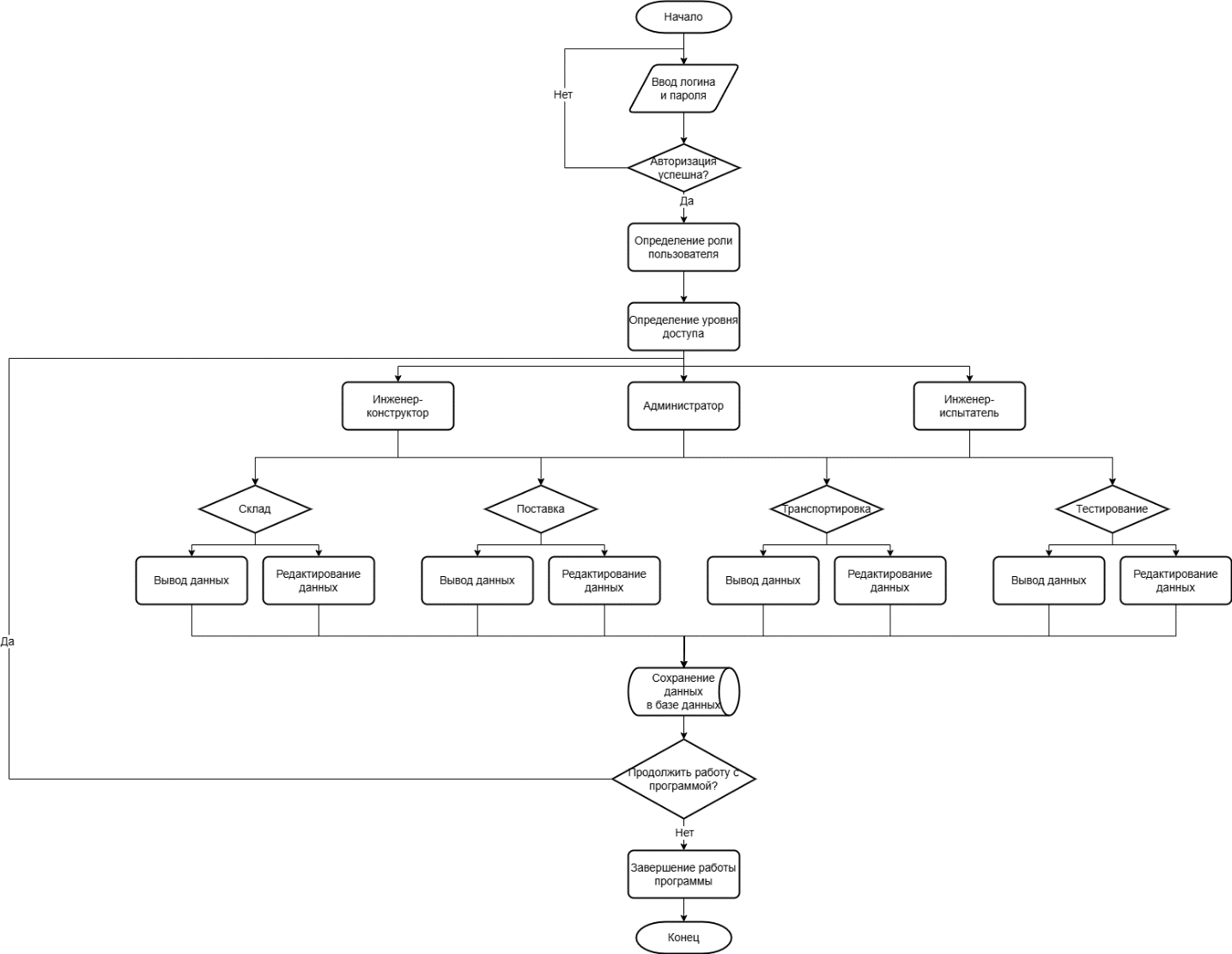


Рисунок 1 — Алгоритм программного средства

* 1. **Планирование базы данных и разработка схем реализации базы данных**

Спроектирована реляционная база данных для ракетостроительного завода в системе управления базами данных MS SQL Server. Структура базы данных включает следующие таблицы:

* сотрудники;
* модели ракет;
* детали;
* должность;
* поставщик;
* ракета;
* поставка;
* тестирование;
* транспортировка.

Для каждого объекта были определены атрибуты — свойства объекта, отличающие его от других:

* для объекта «Сотрудники» — табельный номер, фамилия, имя и отчество работника, занимаемая должность, а также учетные данные (логин и пароль) для доступа к системе;
* для объекта «Модели ракет» — идентификатор модели, название, классификационный тип, массогабаритные характеристики и максимальная дальность полета;
* для объекта «Детали» — артикул детали, наименование и текущее количество на складском учете;
* для объекта «Должность» — код позиции, наименование и установленный уровень заработной платы;
* для объекта «Поставщик» — уникальный идентификатор, полное наименование компании-поставщика, страна регистрации, контактный электронный адрес и телефонный номер для связи;
* для объекта «Ракета» — уникальный серийный номер, ссылку на модель, текущий статус производственного процесса, даты начала и завершения сборки;
* для объекта «Поставка» — номер поставки, ссылки на поставляемую комплектующую и поставщика, количество поставляемых единиц, дату оформления заказа и текущий статус выполнения поставки;
* для объекта «Тестирование» — уникальный номер теста, ссылка на испытываемую ракету, тип проводимого испытания, полученные результаты, дата проведения и ответственный сотрудник;
* для объекта «Транспортировка» — идентификатор перевозки, ссылку на транспортируемую ракету, дату отправки и текущий статус доставки.

Все таблицы связаны через внешние ключи, что обеспечивает целостность данных. Для основных сущностей предусмотрены автоматически генерируемые первичные ключи. Структура базы данных оптимизирована для работы с информацией о производственном цикле — от закупки комплектующих до испытаний и транспортировки готовых изделий [9].

В результате моделирования была спроектирована модель — ER-диаграмма, представленная на рисунке 2

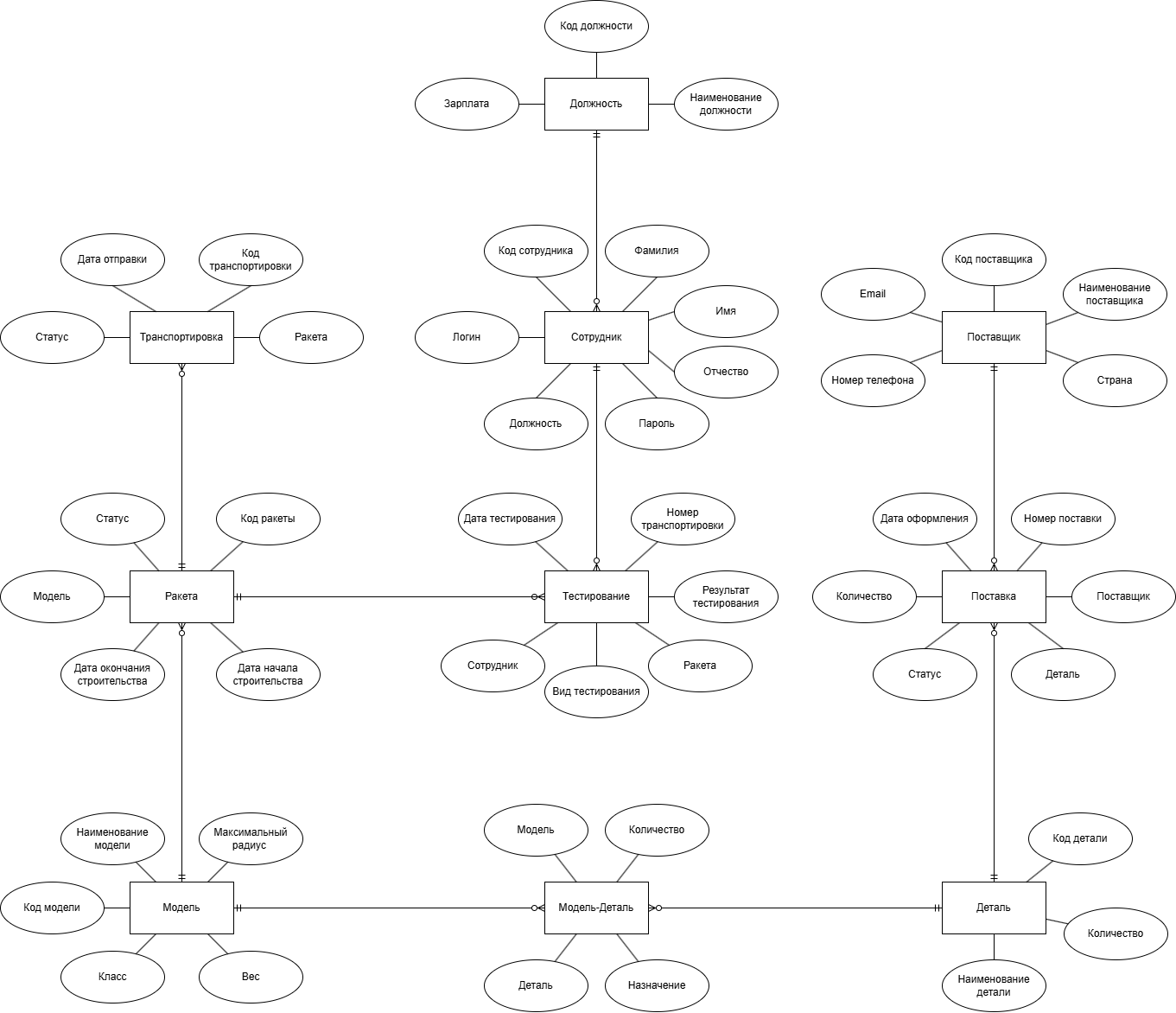


Рисунок 2 — ER-диаграмма

* 1. **Текст программы**
     1. **Создание базы данных**

Для разработки базы данных, используя СУБД Microsoft SQL Server (SSMS) [10], необходимо провести следующие операции:

Настройка СУБД проводилась следующим образом:

1. Установка SQL Server Management Studio:

* Скачать SSMS с официального сайта Microsoft: https://aka.ms/ssmsfullsetup;
* Запустить установщик и выбрать один из вариантов:

1. "Basic" – минимальная установка (SSMS + основные компоненты);
2. "Custom" – выборочная установка (можно добавить дополнительные инструменты, например, SQL Server Data Tools).

Для разработки базы данных "Factory" выбран вариант "Basic", так как он включает все необходимое для работы.

* После завершения установки запустить SSMS.

1. Подключение к серверу:

* тип сервера: Database Engine;
* имя сервера: (localdb)\MSSQLLocalDB;
* аутентификация: Windows Authentication (встроенная аутентификация Windows).

После подключения СУБД, необходимо реализовать запросы на создание таблиц и заполнить их данными. Для начала необходимо создать саму базу, используя SQL-запрос:

“CREATE DATABASE Factory;”

Результатом будет появившаяся схема, или же база данных, представленная на рисунке 3.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3 — Созданная база данных

Далее создаются таблицы. Ниже представлены SQL-запросы на создание таблиц:

* 1. Создание таблицы “Модель”:

CREATE TABLE [dbo].[Model](

[ID\_Model] [int] IDENTITY(1,1) NOT NULL,

[NameModel] [nvarchar](max) NOT NULL,

[Class] [nvarchar](max) NOT NULL,

[Weight] [float] NOT NULL,

[MaxRange] [float] NOT NULL,

CONSTRAINT [PK\_Model] PRIMARY KEY CLUSTERED

* 1. Создание таблицы “Ракета”:

CREATE TABLE [dbo].[Rocket](

[ID\_Rocket] [int] IDENTITY(1,1) NOT NULL,

[ModelRocket] [int] NOT NULL,

[Status] [nvarchar](max) NOT NULL,

[DateStartBuild] [nvarchar](max) NOT NULL,

[DateEndBuild] [nvarchar](max) NULL,

CONSTRAINT [PK\_Rocket] PRIMARY KEY CLUSTERED

Остальные таблицы базы данных создаются аналогичным образом — их структура соответствует приведённым примерам. Полные SQL-скрипты для создания всех таблиц доступны в приложении А. В результате выполнения этих запросов формируется полноценная база данных, готовая к использованию. Её структура может быть визуализирована в виде физической модели данных, отражающей все таблицы и связи между ними.

Физическая модель базы данных представляет собой конкретную реализацию структуры данных, содержащую технические спецификации для их физического хранения. Она включает все необходимые технические параметры, которые требуются системе управления базами данных для непосредственного создания и работы с базой.

Созданная физическая модель базы данных отображена на рисунке 4.



Рисунок 4 — Физическая модель базы данных

SQL Server Management Studio (SSMS) предоставляет разработчикам несколько удобных методов для внесения данных в базу:

1. Использование SQL-запросов:

* Прямое выполнение INSERT-запросов в текстовом редакторе SSMS;
* Возможность массового добавления через скрипты.

1. Ручное редактирование таблиц:

* Визуальное заполнение данных через интерфейс таблицы;
* Пошаговый ввод значений в ячейки;
* Особенно удобно для небольших объемов данных.

1. Импорт из внешних источников:

* Загрузка данных из CSV, Excel, XML и других форматов;
* Использование мастера импорта (SQL Server Import and Export Wizard);
* Настройка преобразования типов данных при загрузке.

Для наполнения базы данных был выбран метод добавления данных через SQL-запросы, что обусловлено следующими преимуществами:

Преимущества данного подхода:

1. Точность и контроль:

* возможность явно указать значения для каждого поля;
* контроль формата вводимых данных на уровне синтаксиса;
* минимизация человеческих ошибок при вводе.

1. Гибкость и масштабируемость:

* возможность массового добавления через один запрос;
* поддержка сложных операций с условиями.

1. Автоматизация и воспроизводимость:

* скрипты можно сохранять и повторно использовать;
* легко вносить изменения перед выполнением;
* возможность включения в процедуры развертывания.

На рисунке 5 представлен результат импорта данных.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 5 — Результат импорта данных

* + 1. **Создание интерфейса программного обеспечения**

Для разработки приложения необходимо изначально спроектировать макеты форм интерфейса:

* макет формы авторизации;
* макет формы меню;
* макет формы вывода данных;
* макет формы ввода данных.

На рисунке 6 представлен разработанный макет формы авторизации. В верхней части интерфейса размещены логотип и заголовок приложения. Ниже расположены текстовые поля для ввода логина и пароля, а также кнопка входа в систему.

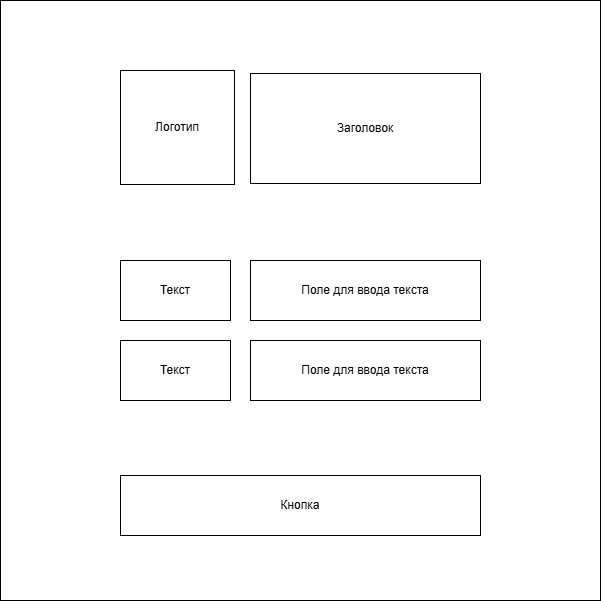


Рисунок 6 — Макет формы авторизации

Следующий макет — макет формы меню. Представлен на рисунке 7. В верхней части макета расположен текст приветствия пользователя, ниже представлены кнопки переключения на другие формы. Внизу макета расположена кнопка выхода из приложения.

Изображение выглядит как снимок экрана, Прямоугольник, диаграмма, текст

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 7 — Макет формы меню

На рисунке 8 представлен макет формы вывода данных. Интерфейс включает область отображения данных и панель управления с кнопками добавления, удаления записей, размещенные в верхней части формы и кнопку «Назад», расположенную в нижней части формы.

Изображение выглядит как текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 8 — Макет формы просмотра данных

На рисунке 9 отображен макет формы добавления данных. Компоновка интерфейса состоит из полей ввода информации и кнопки добавления.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 9 — Макет формы добавления записи

Формы окон добавления и вывода данных могут быть реализованы на основе единого макета ввиду отсутствия принципиальных различий между ними.

Для разработки форм интерфейса применяется конструктор Windows Forms, входящий в состав среды разработки Visual Studio. Данный инструмент используется при создании desktop-приложений под Windows и предоставляет следующие возможности:

* проектирование окон для отображения информации различного типа;
* добавление интерактивных элементов управления (кнопок, полей ввода, списков);
* настройка обработчиков событий для реализации функциональности;
* организация ввода данных с клавиатуры;
* отображение данных в табличном формате (через компонент DataGridView);
* работа с графическими элементами и пользовательскими компонентами.

Для реализации всех форм в Windows Forms используются следующие элементы:

* Form (главное окно) — основной класс окна приложения;
* Button — кнопка, выполняющая заданное действие при нажатии;
* DataGridView — элемент для отображения данных в табличном виде;
* Label — текстовая метка для отображения статической информации;
* TextBox — поле для ввода и редактирования текста
* DateTimePicker — элемент для выбора даты и/или времени;
* ComboBox — выпадающий список с возможностью выбора или ввода значения;
* NumericUpDown — поле для ввода числовых значений с регулировкой стрелками;
* TabControl — элемент с вкладками для организации интерфейса в несколько страниц.

Пример алгоритма создания формы авторизации в Windows Forms:

1. в Windows Forms создается обычная форма (Form);
2. добавление элементов из Toolbox

* TextBox — для ввода данных;
* Label — для текстовых меток;
* PictureBox — для отображения изображений;
* Button — для кнопок;
* TableLayoutPanel — для компоновки элементов;

1. в свойствах элемента изменить поля (Name) для обращения в коде;
2. изменить свойства: MinimumSize и MaximumSize — для ограничения размеров;
3. Dock и Anchor — для автоматического изменения размеров;
4. для PictureBox задать свойство Image — путь к файлу изображения;
5. в свойствах формы задать свойство Icon — путь к файлу изображения.

Форма авторизации представлена на рисунке 10.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 10 — Форма авторизации

* + 1. **Разработка модуля базы данных**

Для полноценной работы программы необходимо реализовать модуль для взаимодействия с базой данных.

Модуль реализован с использованием библиотеки System.Data.SqlClient для подключения к базе данных в Microsoft SQL Server. В модуле обеспечивается соединение с локальным сервером, на котором развернута база данных, с помощью следующего кода:

“SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True")”

Где Data Source – имя сервера ((localdb)\MSSQLLocalDB – стандартный экземпляр SQL Server LocalDB для разработки), Initial Catalog – название базы данных, Integrated Security=True – используется аутентификация Windows.

При аутентификации пользователя система выполняет SQL-запрос для получения данных учетной записи по указанному логину. Это позволяет реализовать разграничение доступа к базе данных в соответствии с ролями пользователей, что повышает безопасность системы.

В системе реализована многоуровневая модель доступа к данным, обеспечивающая различные права для разных должностей. В зависимости от роли пользователя предоставляются соответствующие полномочия для работы с базой данных.

Права пользователя "Администратор" включают следующие запросы:

В окне просмотра склада реализованы запросы:

* просмотр деталей;
* просмотр ракет;
* добавление детали;
* добавление ракеты;
* удаление детали;
* удаление ракеты.

В окне просмотра поставок реализованы запросы:

* просмотр поставок;
* добавление поставки;
* удаление поставки.

В окне просмотра транспортировки ракет реализованы запросы:

* просмотр транспортировок ракет;
* добавление транспортировки ракет;
* удаление транспортировки ракет.

В окне просмотра тестирования ракет реализованы запросы:

* просмотр тестирований ракет;
* добавление тестирования ракет;
* удаление тестирования ракет.

В окне просмотра сотрудников реализованы запросы:

* просмотр сотрудников.

Для пользователя “Инженер-конструктор” реализованы следующие запросы:

В окне просмотра склада реализованы запросы:

* просмотр деталей;
* просмотр ракет;
* добавление детали;
* добавление ракеты;
* удаление детали;
* удаление ракеты.

В окне просмотра поставок реализованы запросы:

* просмотр поставок;
* добавление поставки;
* удаление поставки.

В окне просмотра транспортировки ракет реализованы запросы:

* просмотр транспортировок ракет;
* добавление транспортировки ракет;
* удаление транспортировки ракет.

Для пользователя “Инженер-испытатель” реализованы следующие запросы:

В окне просмотра тестирования ракет реализованы запросы:

* просмотр тестирований ракет;
* добавление тестирования ракет;
* удаление тестирования ракет.

В окне просмотра транспортировки ракет реализованы запросы:

* просмотр транспортировок ракет;
* добавление транспортировки ракет;
* удаление транспортировки ракет.

В процессе разработки системы был применен объектно-ориентированный подход - парадигма программирования, которая организует код в виде взаимодействующих объектов. Основная идея заключается в представлении каждого компонента системы в виде самостоятельного класса, где функциональные возможности реализуются через методы, а состояние и характеристики определяются свойствами этого класса.

При разработке программы было реализовано несколько модулей — классов. Ниже перечислены методы данного классов данных модулей:

1. модуль «ClassAuth»:
   * LogInFunc — получение id сотрудника по логину и паролю;
   * GetFIO — получение фамилии, имени и отчества сотрудника;
   * GetPost — получение должности сотрудника.
2. модуль «ClassRocket»
   * Add — добавление ракеты;
   * Delete — удаление ракеты.
3. модуль «ClassSP»
   * Add — добавление детали;
   * Delete — удаление детали.
4. модуль «ClassSupply»
   * Add — добавление поставки;
   * Delete — удаление поставки.
5. модуль «ClassTest»
   * Add — добавление тестирования ракеты;
   * Delete — удаление тестирования ракеты.
6. модуль «ClassTransportation»
   * Add — добавление транспортировки ракеты;
   * Delete — удаление транспортировки ракеты.

Листинг модулей представлен в приложении Б.

* + 1. **Разработка модуля графического интерфейса**

Следующим компонентом системы является модуль пользовательского интерфейса, разработанный с использованием технологии Windows Forms. В рамках данного модуля было создано несколько классов форм, каждый из которых отвечает за конкретный функциональный раздел приложения.

Для окна авторизации разработан класс «Auth»:

* Auth — инициализирует компоненты формы входа, подготавливает интерфейс для ввода учетных данных;
* buttonLog\_Click — получает значения из полей ввода логина и пароля, проверяет заполненность обязательных полей, выполняет аутентификацию через класс ClassLibrary.Auth, при неудачной попытке входа отображает сообщение об ошибке;
* buttonExit\_Click — обработчик выхода.

Для окна авторизации разработан класс «Menu»:

* Menu — инициализирует компоненты формы, получает и отображает ФИО текущего сотрудника, определяет должность сотрудника и настраивает доступные функции;
* buttonWarehouse\_Click — переход к модулю управления складом;
* buttonTransportation\_Click — переход к модулю транспортировки;
* buttonSupply\_Click — переход к модулю поставок;
* buttonTest\_Click — переход к модулю тестирования;
* buttonEmployee\_Click — переход к модулю управления сотрудниками;
* Menu\_FormClosing — обработчик закрытия формы;
* buttonExit\_Click — выход из системы.

Для окна авторизации разработан класс «Employee»:

* Employee — инициализирует компоненты формы;
* employeeBindingNavigatorSaveItem\_Click — обработчик сохранения данных;
* Employee\_Load — обработчик загрузки формы;
* buttonBack\_Click — обработчик возврата в меню.

Для окна авторизации разработан класс «Supply»:

* Supply — инициализирует компоненты интерфейса;
* supplyBindingNavigatorSaveItem\_Click — обработчик сохранения данных;
* Supply\_Load — обработчик загрузки формы;
* buttonAdd\_Click — обработчик добавления новой поставки, отображает модальное окно AddSupply;
* buttonDelete\_Click — обработчик удаления поставки;
* buttonBack\_Click — обработчик возврата в меню;

Для окна авторизации разработан класс «Test»:

* Test — Инициализирует компоненты интерфейса формы тестирования;
* testBindingNavigatorSaveItem\_Click — обработчик сохранения данных;
* Test\_Load — обработчик загрузки формы;
* buttonAdd\_Click — обработчик добавления теста, отображает модальное окно AddTest;
* buttonDelete\_Click — обработчик удаления теста;
* buttonBack\_Click — обработчик возврата в меню.

Для окна авторизации разработан класс «Transportation»:

* Transportation — инициализирует все компоненты интерфейса;
* transportationBindingNavigatorSaveItem\_Click — обработчик сохранения данных;
* Transportation\_Load — обработчик загрузки формы;
* buttonBack\_Click — обработчик возврата в меню;
* Transportation\_FormClosing — обработчик закрытия;
* buttonAdd\_Click — обработчик добавления транспортировки, создает экземпляр формы AddTransportation;

buttonDelete\_Click — удаление транспортировки.

Для окна авторизации разработан класс «Warehouse»:

* Warehouse — инициализация формы;
* sparePartBindingNavigatorSaveItem\_Click — сохранение изменений;
* Warehouse\_Load — загрузка данных;
* Warehouse\_FormClosing — обработчик закрытия;
* buttonAddSP\_Click — добавление детали;
* buttonDeleteSP\_Click — удаление детали;
* buttonAddRocket\_Click — добавление ракеты;
* buttonDeleteRocket\_Click — удаление ракеты;
* buttonBack2\_Click — возврат к форме меню;
* buttonBack1\_Click — возврат к форме меню.

Листинг форм представлен в приложении В.

* + 1. **Создание инсталлятора**

Разработка инсталлятора для программного продукта существенно упрощает процесс его развертывания на пользовательских компьютерах. Такой подход не только автоматизирует установку, но и предоставляет гибкие возможности настройки параметров инсталляции в соответствии с конкретными потребностями. Инсталляционный пакет гарантирует корректную работу приложения за счет автоматического выполнения всех необходимых системных настроек и проверки зависимостей.

Важным преимуществом использования инсталлятора является возможность внедрения механизмов контроля доступа, что позволяет разработчикам предотвращать несанкционированное копирование и использование программного обеспечения. Инсталляционный пакет обеспечивает стандартизированный процесс установки, минимизируя вероятность ошибок при развертывании и создавая оптимальные условия для работы приложения в различных окружениях.

Первым шагом в создании установочного пакета является сборка проекта в Visual Studio, в результате которой генерируются исполняемые файлы (.exe) и вспомогательные библиотеки (.dll). Для преобразования скомпилированного решения в установочный пакет используется расширение Microsoft Visual Studio Installer Projects, которое позволяет:

1. автоматизировать процесс упаковки всех необходимых компонентов:

* основного исполняемого файла;
* зависимых сборок (включая ClassLibrary.dll);
* ресурсов и файлов конфигурации;

1. создать профессиональный установщик в формате MSI, который:

* интегрируется с системой Windows;
* поддерживает механизмы установки/удаления;
* обеспечивает контроль версий;

1. настроить параметры установки:

* пути инсталляции по умолчанию;
* ярлыки в меню "Пуск" и на рабочем столе;
* условия лицензирования.

Процесс создания установщика в Visual Studio описан ниже:

1. сборка проекта:

* компиляция решения → получение .exe/.dll файлов в папке bin/Release;

1. добавление проекта Setup Project:

* установка расширения Microsoft Visual Studio Installer Projects;
* создание проекта типа Setup Project в решении;

1. настройка установщика:

* добавление выходных файлов (Primary Output);
* включение зависимостей (например, ClassLibrary.dll);
* создание ярлыков и иконок;
* настройка свойств (версия, производитель);

1. построение MSI-пакета:

* Build проекта установщика → генерация .msi файла;
* автоматическое включение всех зависимостей и регистрация компонентов;

1. тестирование:

* проверка установки/удаления на чистой системе;
* валидация путей и работоспособности приложения.

Результат представлен на рисунке 11.

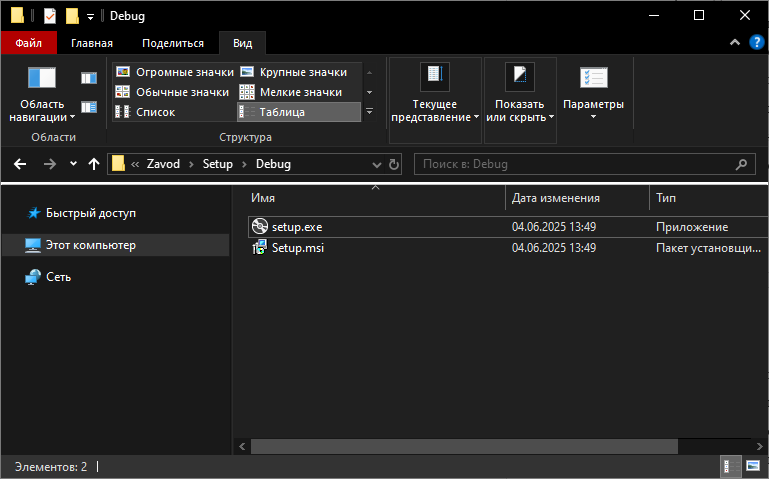


Рисунок 11 — Результат создания установочного файла

* 1. **Схемы программы**
     1. **Диаграмма вариантов использования**

Диаграмма вариантов использования представляет собой визуальную модель, отображающую возможные сценарии взаимодействия между пользователями и системой. Она служит эффективным инструментом для демонстрации функциональных возможностей разрабатываемого программного обеспечения.

Основная цель данной диаграммы заключается в четкой фиксации функциональных требований к системе и обеспечении возможности их согласования с заказчиком еще на этапе проектирования. Такой подход позволяет:

* наглядно представить все ключевые функции системы;
* определить границы взаимодействия пользователей с системой;
* выявить основные сценарии использования;
* установить четкие требования к функционалу.

Диаграмма вариантов использования особенно ценна на начальных этапах разработки, так как помогает:

* достичь общего понимания системы между заказчиком и разработчиками;
* выявить потенциальные проблемы на ранней стадии;
* сформировать основу для последующего проектирования архитектуры системы;
* определить основных акторов и их взаимодействие с системой.

Для разработки диаграммы вариантов использования соблюдать некоторую последовательность действий:

1. идентификация групп пользователей — первоначально необходимо выделить ключевые категории пользователей системы. В данном случае определены три основные роли:

* администратор системы;
* инженер-конструктор;
* инженер-испытатель;

1. визуальное представление элементов диаграммы — для графического отображения компонентов используются стандартные нотации UML:

* пользователи (акторы) изображаются стилизованными фигурами человека;
* варианты использования представляются в виде овалов;
* взаимодействия обозначаются сплошными линиями;

1. установление взаимосвязей — между выделенными акторами и функциональными возможностями системы:

* каждая линия связи отражает конкретное взаимодействие;
* отношения ассоциации показывают, какие варианты использования доступны каждому актору;
* возможны дополнительные типы связей (наследование, включение, расширение).

Рекомендуется придерживаться следующего последовательностью разработки:

* определение всех акторов системы;
* выявление основных сценариев использования;
* установление отношений между элементами;
* проверка полноты охвата функциональности;
* валидация диаграммы с заинтересованными сторонами.

Такой структурированный подход обеспечивает создание четкой и информативной модели, которая служит надежной основой для последующих этапов проектирования системы.

Реализация диаграммы вариантов использования представлена на рисунке 12.

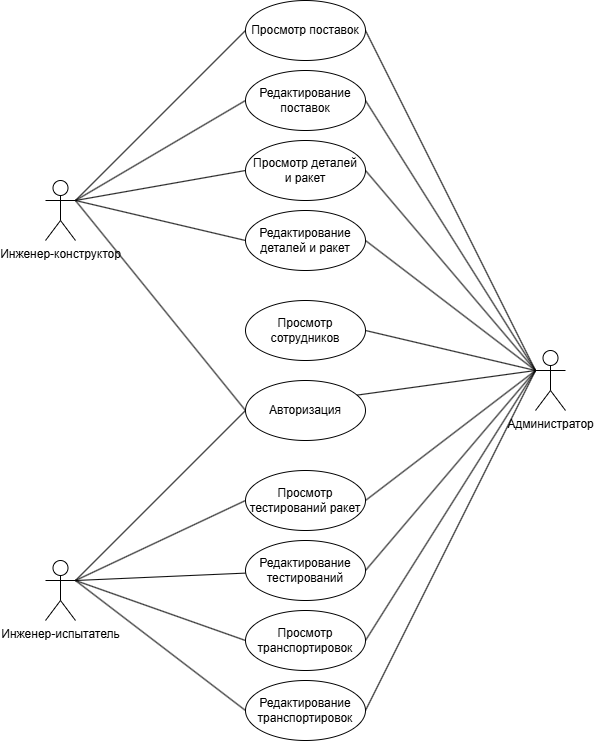


Рисунок 12 — Диаграмма вариантов использования

* + 1. **Схема структуры программы**

Информационная система ракетостроительного завода включает в себя несколько взаимосвязанных подсистем, каждая из которых отвечает за определенный функциональный модуль:

* + 1. подсистема учета деталей и ракет — обеспечивает операции просмотра, добавления и удаления информации о комплектующих и готовых изделиях;
    2. подсистема управления поставками — отвечает за работу с данными о поступающих материалах (просмотр, внесение новых и удаление устаревших записей);
    3. подсистема тестирования ракет — управляет информацией о проводимых испытаниях (просмотр, добавление и удаление данных тестов);
    4. подсистема транспортировки ракет — контролирует сведения о перемещении изделий;
    5. подсистема кадрового учета — предоставляет доступ к информации о сотрудниках.

Структура представлена в виде графа, где корневым элементом является основная информационная система, а подчиненными узлами — перечисленные функциональные подсистемы, как показано на рисунке 13.

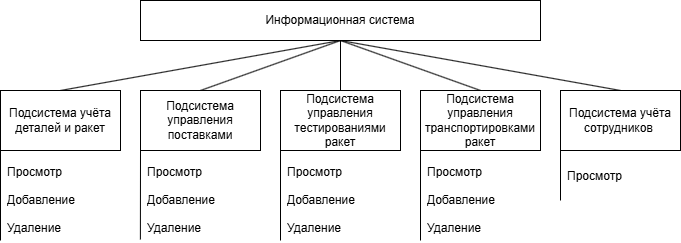


Рисунок 13 — Структурная схема

* + 1. **Архитектура программы**

Архитектура программного обеспечения представляет собой систему взаимосвязанных компонентов, определяющих структуру, поведение и взаимодействие элементов ПО.

Для системы управления книжным магазином архитектура включает следующие ключевые составляющие:

* автономные модули — каждый компонент системы функционирует как самостоятельный объект, но интегрируется в единую систему;
* централизованное хранилище данных — объект базы данных обеспечивает согласованное хранение и обработку информации, поступающей от всех модулей;
* интерфейс взаимодействия — графический интерфейс (GUI) служит точкой входа для пользователей, обеспечивая визуализацию данных и управление функционалом системы.

Такая структура, представленная на рисунке 14, демонстрирует целостную архитектуру программного обеспечения ракетостроительного завода, где модули, данные и интерфейс объединены в единый рабочий механизм.

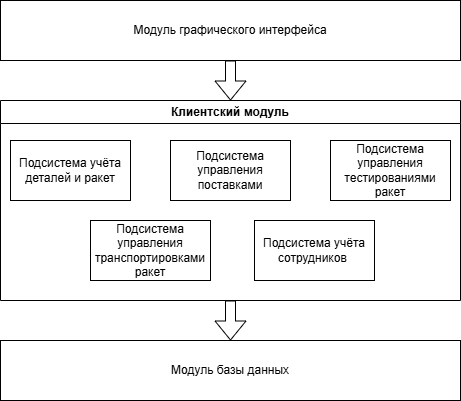


Рисунок 14 — Архитектура программы

* 1. **Инструкция по эксплуатации**
     1. **Общие сведения**

Приложение для автоматизации управления ракетным производством функционирует как единый цифровой контур, объединяющий поставку компонентов, финальную сборку изделий, их тестирование и транспортировку. Программный комплекс функционирует по оптимизированному алгоритму, обеспечивая:

* оперативный контроль производственных процессов;
* автоматизированный учет комплектующих и готовой продукции;
* точное планирование поставок материалов;
* эффективное управление тестированием и транспортировкой ракетной техники.

Данная информационная система имеет в своем составе следующие подсистемы:

* подсистема управления поставками комплектующих;
* подсистема учета деталей и готовых изделий;
* подсистема контроля тестирования ракет;
* подсистема организации транспортировки;
* модуль кадрового учета сотрудников.
  + 1. **Установка и запуск программы**

Установка программы происходит следующим образом:

1. запускается файл инсталлятора “Ракетостроительный завод.exe”. На экран выводится окно инсталлятора. Представлено на рисунке 15;

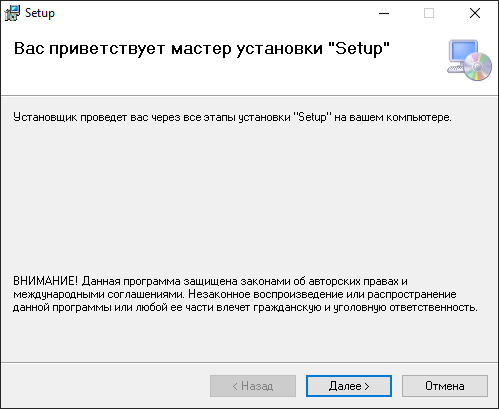


Рисунок 15 — Окно установщика

1. следующее окно представляет собой выбор папки для установки приложения. По умолчанию приложение будет установлено по пути: «C:\Program Files (x86 \Setup\»;
2. при нажатии кнопки “Далее” открывается последнее окно установки, в котором установщик извещает пользователя о подтверждении установки. Для начала установки необходимо нажать кнопку “Далее”.
3. на рисунке 16 представлено окно, информирующее пользователя о завершении процесса установки программного обеспечения.

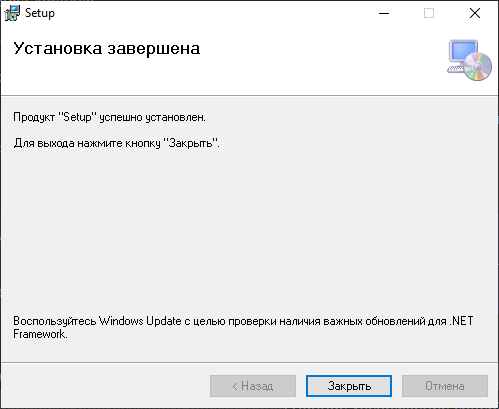


Рисунок 16 — Завершение установки

Для корректного функционирования разработанного программного обеспечения требуется предварительная установка системы управления базами данных SQL Server Management Studio.

Необходимо развернуть базу данных в SQL Server Management Studio при наличии активного подключения к серверу. Процедура выполняется следующим образом:

1. открыв подключение, во вкладке «File» выбрать «Open» - «File»;
2. выбрать файл «Database.sql» находящийся в папке с установленными приложением;
3. нажать кнопку «Execute», после чего в списке баз данных появится необходимая база данных с названием «Factory»;
   * 1. **Работа с основными функциями**

При запуске приложения отображается окно авторизации, представленное на рисунке 17. Пользователю необходимо ввести свои логин и пароль.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 17 — Окно авторизации

После авторизации пользователь переходит в меню приложению, где видит приветствующую надпись вверху окна и кнопки переходов к доступным по должности страницам. Представлено на рисунке 18.

Изображение выглядит как текст, снимок экрана, дисплей, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 18 — Окно меню

Для пользователя с должностью «Инженер-конструктор» доступны кнопки «Склад» и «Поставки». На странице «Склад», показанная на рисунке 19, предоставляется возможность добавить новую деталь или ракету, а также отредактировать или удалить имеющиеся в таблице.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 19 — Страница «Склад»

На странице «Поставки» предоставляется возможность добавить новую поставку, а также отредактировать или удалить имеющиеся в таблице. Представлено на рисунке 20.

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 20 — Страница «Поставки»

Для пользователя с должностью «Инженер-испытатель» доступны кнопки «Тестирование» и «Транспортировка». На странице «Тестирование», показанная на рисунке 21, предоставляется возможность добавить новую запись о тестировании ракеты, а также отредактировать или удалить имеющиеся в таблице.

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 21 — Страница «Тестирование»

На странице «Транспортировка», показанная на рисунке 22, предоставляется возможность добавить новую запись о транспортировке ракеты, а также отредактировать или удалить имеющиеся в таблице.

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 22 — Страница «Транспортировка»

Для пользователя с должностью «Администратор» доступны все предыдущие кнопки, а также кнопка «Сотрудники». На странице «Сотрудники», показанная на рисунке 23, предоставляется возможность просмотреть всех сотрудников.

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 23 — Страница «Сотрудники»

* 1. **Тестирование программных модулей**

Тестирование представляет собой процесс выполнения программного кода с целью выявления возможных ошибок и дефектов, в то время как отладка включает в себя последующее определение местоположения обнаруженных ошибок и их устранение.

Процесс разработки тестов осуществляется по строго определенной методологии, предполагающей системный подход к проверке функциональности программы. В соответствии с внешними спецификациями принято выделять два основных типа тестов:

* классовые тесты — предназначенные для проверки всех категорий входных данных;
* граничные тесты — ориентированные на анализ особых и предельных значений входных параметров.

Важным аспектом тестирования является обязательный контроль полноты охвата всех возможных классов выходных данных. При выявлении недостаточного покрытия тестами определенных сценариев работы программы, осуществляется разработка и добавление соответствующих проверочных случаев для обеспечения комплексной верификации программного продукта.

Для комплексного тестирования функционала приложения с использованием фреймворка MSTest были разработаны специализированные тестовые классы, каждый из которых отвечает за проверку определенного модуля системы:

* 1. Тестирование модуля авторизации (UnitTestAuth).

Класс UnitTestAuth проверяет корректность работы системы аутентификации:

* TestInitialize() — инициализация тестового окружения, создание тестового пользователя;
* LogInFunc\_ValidCredentials\_ReturnsEmployeeId() — проверка входа с корректными учетными данными;
* LogInFunc\_InvalidLogin\_ReturnsNull() — тест реакции на неверный логин;
* LogInFunc\_InvalidPassword\_ReturnsNull() — тест реакции на неверный пароль;
* GetFIO\_ValidEmployeeId\_ReturnsFullName() — проверка получения ФИО сотрудника;
* GetPost\_ValidEmployeeId\_ReturnsPost() — тест получения должности сотрудника.
  + 1. Тестирование модуля управления ракетами (UnitTestRocket).

Класс UnitTestRocket проверяет CRUD-операции с ракетами:

* Setup() — подготовка тестовых данных;
* Add\_ValidData\_ReturnsTrue() — тест добавления ракеты с корректными параметрами;
* Add\_ZeroModel\_ReturnsFalse() — проверка реакции на недопустимый ID модели;
* Delete\_ExistingId\_ReturnsTrue() — тест удаления существующей записи;
* Delete\_NonExistingId\_ReturnsFalse() — проверка обработки несуществующего ID.
  + 1. Тестирование модуля комплектующих (UnitTestSP).

Класс UnitTestSP тестирует работу с запчастями:

* Add\_ValidData\_ReturnsTrue() — проверка добавления детали;
* Add\_EmptyName\_ReturnsFalse() — тест реакции на пустое наименование;
* Delete\_ExistingId\_ReturnsTrue() — проверка удаления существующей детали.
  + 1. Тестирование модуля поставок (UnitTestSupply).

Класс UnitTestSupply проверяет логистические операции:

* Add\_ValidData\_ReturnsTrue() — тест создания поставки;
* Add\_ZeroCount\_ReturnsFalse() — проверка на нулевое количество;
* Delete\_ExistingId\_ReturnsTrue() — тест удаления поставки.
  + 1. Тестирование модуля испытаний (UnitTestTesting).

Класс UnitTestTesting проверяет систему тестирования:

* Add\_ValidData\_ReturnsTrue() — тест добавления теста;
* Add\_InvalidRocket\_ReturnsFalse() — проверка на неверный ID ракеты;
* Add\_EmptyType\_ReturnsFalse() — тест реакции на пустой тип теста.
  + 1. Тестирование модуля транспортировки (UnitTestTransportation).

Класс UnitTestTransportation проверяет логистику:

* Add\_ValidData\_ReturnsTrue() — тест создания перевозки;
* Add\_EmptyStatus\_ReturnsFalse() — проверка на пустой статус;
* Delete\_ExistingId\_ReturnsTrue() — тест удаления перевозки.

Каждый тестовый класс включает в себя следующее:

* методы инициализации (TestInitialize) и очистки (TestCleanup);
* положительные тест-кейсы (валидные данные);
* негативные тест-кейсы (проверка граничных условий и обработки ошибок);
* тесты на устойчивость к сбоям (использование BrokenConnection классов).

Особенностью реализации является:

* автоматическое создание и удаление тестовых данных
* проверка как бизнес-логики, так и целостности данных в БД
* комплексная проверка всех сценариев использования

Листинг модуля тестирования представлен в приложении Г. Результаты проведения тестирования функционала представлены в таблицах 4-8.

Таблица 4 — Результаты тестирования модуля авторизации

|  |  |  |
| --- | --- | --- |
| Название теста | Входные данные | Ожидаемый результат |
| TestInitialize() | - | Создан тестовый пользователь |
| LogInFunc\_ValidCredentials\_ReturnsEmployeeId() | Корректный логин/пароль | ID сотрудника |
| LogInFunc\_InvalidLogin\_ReturnsNull() | Неверный логин | NULL |
| LogInFunc\_InvalidPassword\_ReturnsNull() | Неверный пароль | NULL |
| GetFIO\_ValidEmployeeId\_ReturnsFullName() | Действительный ID сотрудника | ФИО сотрудника |
| GetPost\_ValidEmployeeId\_ReturnsPost() | Действительный ID сотрудника | Должность сотрудника |

Таблица 5 — Результаты тестирования модуля склада

|  |  |  |
| --- | --- | --- |
| Название теста | Входные данные | Ожидаемый результат |
| Setup() | - | Подготовлены тестовые данные |
| Add\_ValidData\_ReturnsTrue() | Корректные параметры ракеты | TRUE |
| Add\_ZeroModel\_ReturnsFalse() | ID модели = 0 | FALSE |
| Delete\_ExistingId\_ReturnsTrue() | Существующий ID ракеты | TRUE |

Продолжение таблицы 5

|  |  |  |
| --- | --- | --- |
| Delete\_NonExistingId\_ReturnsFalse() | Несуществующий ID | FALSE |
| Add\_ValidData\_ReturnsTrue() | Корректные данные детали | TRUE |
| Add\_EmptyName\_ReturnsFalse() | Пустое наименование детали | FALSE |
| Delete\_ExistingId\_ReturnsTrue() | Существующий ID детали | TRUE |

Таблица 6 — Результаты тестирования модуля поставок

|  |  |  |
| --- | --- | --- |
| Название теста | Входные данные | Ожидаемый результат |
| Add\_ValidData\_ReturnsTrue() | Корректные данные поставки | TRUE |
| Add\_ZeroCount\_ReturnsFalse() | Количество = 0 | FALSE |
| Delete\_ExistingId\_ReturnsTrue() | Существующий ID поставки | TRUE |

Таблица 7 — Результаты тестирования модуля тестирования

|  |  |  |
| --- | --- | --- |
| Название теста | Входные данные | Ожидаемый результат |
| Add\_ValidData\_ReturnsTrue() | Корректные данные теста | TRUE |
| Add\_InvalidRocket\_ReturnsFalse() | Неверный ID ракеты | FALSE |
| Add\_EmptyType\_ReturnsFalse() | Пустой тип теста | FALSE |

Таблица 8 — Результаты тестирования модуля транспортировки

|  |  |  |
| --- | --- | --- |
| Название теста | Входные данные | Ожидаемый результат |
| Add\_ValidData\_ReturnsTrue() | Корректные данные перевозки | TRUE |

Продолжение таблицы 8

|  |  |  |
| --- | --- | --- |
| Add\_EmptyStatus\_ReturnsFalse() | Пустой статус | FALSE |
| Delete\_ExistingId\_ReturnsTrue() | Существующий ID перевозки | TRUE |

По результатам тестирования функционала программы ошибки не обнаружены.

1. **Экономическая часть**

В себестоимость разработки приложения входят следующие статьи затрат:

* основная заработная плата;
* дополнительная заработная плата;
* отчисления на социальное страхование;
* прочие расходы.

Разработку системы проводит один специалист: программист.

Зарплата программиста — 400 руб/час. При этом продолжительность рабочего дня составляет 8 часов.

Расчет основной заработной платы представлен в таблице 9.

Таблица 9 — Расчет основной заработной платы

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Этапы | Виды работ | Исполнитель | | Часовая тарифная ставка, руб. | Длительность выполнения, дней | Трудоемкость, нормо-час | Заработная плата, руб. |
| Количество | Должность |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1. Начальный | Формулирование требований к программе, описание целей разработки | 1 | Программист | 400 | 1 | 8 | 3 200 |
| 2. Внешнее проектирование | Разработка архитектуры и структуры программы, алгоритма разработка интерфейса пользователя | 1 | Программист | 400 | 3 | 24 | 960 |

Продолжение таблицы 9

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Этапы | Виды работ | Исполнитель | | Часовая тарифная ставка, руб. | Длительность выполнения, дней | Трудоемкость, нормо-час | Заработная плата, руб. |
| Количество | Должность |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 3. Разработка и кодирование компонентов | Разработка каждого компонента и кодирование на языке программирования | 1 | Программист | 400 | 7 | 56 | 22 400 |
| 4. Основной этап | Отделка модулей | 1 | Программист | 400 | 1 | 8 | 3 200 |
| Тестирование компонентов | 1 | Программист | 400 | 1 | 8 | 3 200 |
| Комплексное тестирование программы | 1 | Программист | 400 | 1 | 8 | 3 200 |
| Оформление программной документации | 1 | Программист | 400 | 2 | 16 | 6 400 |
| 5. Заключительный этап | Коррекция программной документации | 1 | Программист | 400 | 1 | 8 | 3 200 |
| Итого | — | — | — | — | 17 | 136 | 45 760 |

К дополнительной заработной плате относятся: премии за выполнение основных этапов работ, выплата вознаграждения за выслугу лет и так далее. Дополнительная заработная плата является премиальной, по условиям премирования организации она составляет 20% от основной заработной платы.

Расчёт дополнительной заработной платы представлен в таблице 10.

Таблица 10 — Расчёт дополнительной заработной платы

|  |  |  |  |
| --- | --- | --- | --- |
| Основная  заработная плата, руб. | Дополнительная  заработная плата, % | Дополнительная  заработная плата, руб. | Итого  заработная плата, руб. |
| 45 760 | 20 | 9 152 | 54 912 |

К отчислениям на социальное страхование относятся отчисления в Пенсионный фонд Российской федерации, в Фонд социального страхования, в Фонд обязательного медицинского страхования и на социальное страхование от несчастных случаев и профессиональных заболеваний, если это прописано в договоре.

Отчисления на социальное страхование рассчитываются от суммы основной и дополнительной заработной платы по действующим ставкам.

Рассмотрим основные тарифы страховых взносов за работников в 2022 году. Они составляют:

1. на обязательное пенсионное страхование (ПФР)— 22 %;
2. на обязательное медицинское страхование (ФОМС)— 5,1 %;
3. на случай временной нетрудоспособности и материнства (ФСС)— 2,9 %;
4. на травматизм (ФСС тр)— от 0,2 % до 8,5 % в зависимости от класса профессионального риска, присвоенного основному осуществляемому виду деятельности.

Расчёт отчислений на социальное страхование представлен в таблице 11.

Таблица 11 — Расчёт отчислений на социальное страхование

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | База для расчёта отчислений на социальное страхование, руб. | Тариф страховых взносов, % | Отчисления на  социальное  страхование, руб. |
| 1 | 54 912 | 22 | 12 080,64 |
| 2 | 54 912 | 5,1 | 2 800,51 |
| 3 | 54 912 | 2,9 | 1 592,45 |
| 4 | 54 912 | 0,2 | 109,82 |
| Итого | | — | 16 583,42 |

К прочим расходам следует отнести расходы на обслуживание ЭВМ и плату за электроэнергию.

Затраты на электроэнергию рассчитываются исходя из потребляемой мощности устройства и тарифа на электроэнергию. В данном случае предполагается использование компьютера с мощностью 0,8 кВт час. Стоимость одного кВт часа электроэнергии равна 4,32 руб. Время использования электроэнергии в процессе разработки равно 136 часов.

Расчёт затрат на электроэнергию представлен в таблице 12.

Таблица 12 — Расчёт затрат на электроэнергию

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Мощность устройства, кВт час | Время использования электроэнергии, час | Стоимость 1 кВт часа электроэнергии, руб. | Расчёт затрат на электроэнергию | Затраты на электроэнергию, руб. |
| 0,8 | 136 | 4,32 | 0,8\*136\*4,32 | 470 |

Расходы на обслуживание ЭВМ определяются из стоимости ЭВМ и срока ее полезного использования, по истечении которого, она подлежит замене. Примем это время равным 3 годам, в течение года ЭВМ используется 252 рабочих дня.

Расчёт расходов на обслуживание ЭВМ представлен в таблице 13.

Таблица 13 — Расчёт затрат на обслуживание ЭВМ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Стоимость ЭВМ, руб. | Срок полезного использования, лет | Время работы ЭВМ в год, дней | Продолжительность рабочего дня, час | Время использования электроэнергии, час | Расчёт затрат на обслуживание ЭВМ | Затраты на обслуживание ЭВМ, руб. |
| 60 000 | 3 | 252 | 8 | 136 | |  | | --- | | 60 000 \*136 | | 3\*252\*8 | | 1 349 |

Расчет себестоимости разработки системы представлен в таблице 14.

Таблица 14 — Расчет себестоимости разработки приложения книжного магазина “Дом книги”

|  |  |  |
| --- | --- | --- |
| № п/п | Статьи затрат | Сумма, руб. |
| 1 | Основная заработная плата | 45 760 |
| 2 | Дополнительная заработная плата | 9 152 |
| 3 | Отчисления на социальное страхование | 16 583,42 |
| 4 | Затраты на электроэнергию | 470 |
| 5 | Затраты на обслуживание ЭВМ | 1 349 |
| Итого затрат | | 73 314,42 |

1. **Мероприятия по технике безопасности**

Сотрудники вычислительных центров, включая операторов ЭВМ, специалистов по обработке данных и программистов, подвергаются воздействию различных вредных и опасных факторов производственной среды. К ним относятся: чрезмерный шумовой фон, повышенные температурные показатели воздуха, недостаточное освещение рабочих мест, риск поражения электротоком, а также явления статического электричества.

Несмотря на существующий арсенал организационных мер и технических средств защиты, направленных на минимизацию воздействия вредных производственных факторов, современный уровень безопасности труда во многих вычислительных центрах по-прежнему требует существенного улучшения и модернизации.

Особую опасность представляют электроустановки, составляющие основу оборудования вычислительных центров. В процессе эксплуатации или технического обслуживания существует вероятность контакта персонала с токоведущими элементами. Характерная особенность электрооборудования заключается в отсутствии видимых признаков опасности - поврежденные проводники или оказавшиеся под напряжением корпуса устройств не подают явных сигналов тревоги. Осознание опасности происходит только в момент поражения электрическим током, когда он уже начинает воздействовать на организм человека.

Крайне важным аспектом профилактики электротравматизма является строгое соблюдение регламентов обслуживания электроустановок, включая проведение ремонтных, монтажных и профилактических работ с обязательным выполнением всех требований электробезопасности.

Явления статического электричества в вычислительных центрах проявляются преимущественно при контакте с элементами компьютерной техники. Хотя сами по себе электростатические разряды не представляют прямой угрозы здоровью персонала, они способны вызывать неприятные ощущения и, что более существенно, приводить к выходу из строя дорогостоящего оборудования. Для минимизации подобных рисков рекомендуется:

1. использование специальных антистатических покрытий для полов, в частности однослойного поливинилхлоридного линолеума с антистатическими свойствами.
2. применение систем нейтрализации статического заряда с использованием ионизированного газа, включая промышленные радиоактивные нейтрализаторы.
3. поддержание оптимального уровня влажности воздуха как общеобменными системами, так и локальными увлажнителями.

Требования безопасности при работе с компьютерной техникой включают следующие обязательные положения:

* к работе с ПЭВМ допускаются исключительно сотрудники, прошедшие соответствующее обучение и инструктаж по технике безопасности;
* подключение оборудования должно осуществляться только к исправным и правильно заземленным розеткам;
* рекомендуемая последовательность включения устройств: сначала монитор, затем периферийные устройства (принтер), и только после этого - системный блок;
* категорически запрещается: проводить манипуляции с принтером при включенном питании, работать с открытыми корпусами оборудования, оставлять включенные ПЭВМ без присмотра на продолжительное время;
* при обнаружении неисправностей, появлении посторонних шумов или запаха гари необходимо немедленно отключить оборудование и сообщить руководству.

В аварийных ситуациях действуют следующие протоколы:

1. при возгорании: немедленное отключение электропитания, уведомление ответственных лиц, применение первичных средств пожаротушения с одновременным вызовом пожарной службы;
2. при поражении электротоком: обесточивание оборудования, оказание первой помощи пострадавшему с обязательным вызовом медицинских работников.

По завершении рабочего дня необходимо:

* привести в порядок рабочее пространство;
* закрыть окна и другие проемы;
* корректно выключить оборудование согласно установленному регламенту.

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения выпускной квалификационной работы была разработана информационная система учета для ракетостроительного завода. В процессе разработки успешно реализованы все поставленные задачи по созданию программного обеспечения и его функциональных модулей.

Программный комплекс функционирует по оптимизированному алгоритму, обеспечивая:

* оперативный контроль производственных процессов;
* автоматизированный учет комплектующих и готовой продукции;
* точное планирование поставок материалов;
* эффективное управление тестированием и транспортировкой ракетной техники.

Разработаны следующие ключевые модули системы:

* подсистема управления поставками комплектующих
* подсистема учета деталей и готовых изделий
* подсистема контроля тестирования ракет
* подсистема организации транспортировки
* модуль кадрового учета сотрудников

В процессе работы были выполнены следующие задачи:

1. спроектирована и реализована реляционная база данных, содержащая всю необходимую информацию для работы предприятия
2. разработаны и интегрированы все функциональные модули системы;
3. проведено комплексное тестирование работоспособности всех компонентов системы.

В ходе выполнения выпускной квалификационной работы был успешно разработан программный комплекс для автоматизации управления ракетным производством. Проведенная работа включала проработку функциональных модулей системы и их взаимосвязей, что обеспечило четкое выполнение производственных процессов. В ходе проекта проведен детальный анализ экономической составляющей, включающий расчет полной себестоимости разработки с учетом всех прямых и косвенных затрат, что позволило обосновать эффективность внедрения системы.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Бен-Ган, И. Внутреннее устройство Microsoft SQL Server / И. Бен-Ган. - Москва : Русская редакция, 2020. - 832 с.
2. Албахари, Д. C# 9.0. Справочник. Полное описание языка / Д. Албахари, Б. Албахари. - Санкт-Петербург : БХВ-Петербург, 2021. - 1040 с.
3. Мак-Дональд, М. Windows Presentation Foundation в .NET 4.5 с примерами на C# / М. Мак-Дональд. - Москва : Вильямс, 2020. - 1024 с.
4. Троелсен, Э. Язык программирования C# 10 и .NET 6 / Э. Троелсен ; [пер. с англ. под ред. И. В. Бернадского]. – Москва : Вильямс, 2022. – 1328 с.
5. Рихтер, Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Д. Рихтер. - Москва : Питер, 2013. - 896 с.
6. Смит, Дж. Entity Framework Core in Action / Дж. Смит. - 2nd ed. - Manning, 2021. - 600 c.
7. Бен-Ган, И. Microsoft SQL Server 2022. Основы T-SQL / И. Бен-Ган. - Москва : Диалектика, 2023. - 800 с.
8. Кормен, Т. Алгоритмы. Построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. – 3-е изд. – Москва : Вильямс, 2022. – 1328 с.
9. Гарсия-Молина, Г. Системы баз данных. Полный курс / Г. Гарсия-Молина, Дж. Д. Ульман, Дж. Уидом. - Москва : Вильямс, 2021. - 1088 с.
10. Уолтерс, Р. SQL Server Management Studio 18: Полное руководство / Р. Уолтерс. - Москва : ДМК Пресс, 2021. - 420 с.

**ПРИЛОЖЕНИЕ А  
Запросы на создание таблиц**

Создание таблицы «Сотрудник»:

CREATE TABLE [dbo].[Employee](

[ID\_Employee] [int] IDENTITY(1,1) NOT NULL,

[Surname] [nvarchar](max) NOT NULL,

[Name] [nvarchar](max) NOT NULL,

[LastName] [nvarchar](max) NULL,

[Post] [int] NOT NULL,

[Login] [nvarchar](max) NOT NULL,

[Password] [nvarchar](max) NOT NULL,

CONSTRAINT [PK\_Employee] PRIMARY KEY CLUSTERED

(

[ID\_Employee] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

Создание таблицы «Модель»:

CREATE TABLE [dbo].[Model](

[ID\_Model] [int] IDENTITY(1,1) NOT NULL,

[NameModel] [nvarchar](max) NOT NULL,

[Class] [nvarchar](max) NOT NULL,

[Weight] [float] NOT NULL,

[MaxRange] [float] NOT NULL,

CONSTRAINT [PK\_Model] PRIMARY KEY CLUSTERED

(

[ID\_Model] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

Создание таблицы «Модель-Деталь»:

CREATE TABLE [dbo].[Model\_SparePart](

[ID] [int] IDENTITY(1,1) NOT NULL,

[Model] [int] NOT NULL,

[SparePart] [int] NOT NULL,

[Count] [int] NOT NULL,

[Appointment] [nvarchar](max) NULL,

CONSTRAINT [PK\_Model\_SparePart] PRIMARY KEY CLUSTERED

(

[ID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

Создание таблицы «Должность»:

CREATE TABLE [dbo].[Post](

[ID\_Post] [int] IDENTITY(1,1) NOT NULL,

[NamePost] [nvarchar](max) NOT NULL,

[Salary] [float] NULL,

CONSTRAINT [PK\_Post] PRIMARY KEY CLUSTERED

(

[ID\_Post] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

Создание таблицы «Поставщик»:

CREATE TABLE [dbo].[Provider](

[ID\_Provider] [int] IDENTITY(1,1) NOT NULL,

[NameProvider] [nvarchar](max) NOT NULL,

[Country] [nvarchar](max) NOT NULL,

[Email] [nvarchar](max) NOT NULL,

[Phone] [nvarchar](max) NULL,

CONSTRAINT [PK\_Provider] PRIMARY KEY CLUSTERED

(

[ID\_Provider] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

Создание таблицы «Ракета»:

CREATE TABLE [dbo].[Rocket](

[ID\_Rocket] [int] IDENTITY(1,1) NOT NULL,

[ModelRocket] [int] NOT NULL,

[Status] [nvarchar](max) NOT NULL,

[DateStartBuild] [nvarchar](max) NOT NULL,

[DateEndBuild] [nvarchar](max) NULL,

CONSTRAINT [PK\_Rocket] PRIMARY KEY CLUSTERED

(

[ID\_Rocket] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

Создание таблицы «Деталь»:

CREATE TABLE [dbo].[SparePart](

[ID\_SparePart] [int] IDENTITY(1,1) NOT NULL,

[NameSparePart] [nvarchar](max) NOT NULL,

[Count] [int] NOT NULL,

CONSTRAINT [PK\_SparePart] PRIMARY KEY CLUSTERED

(

[ID\_SparePart] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

Создание таблицы «Поставка»:

CREATE TABLE [dbo].[Supply](

[SupplyNumber] [int] IDENTITY(1,1) NOT NULL,

[SparePart] [int] NOT NULL,

[Provider] [int] NOT NULL,

[Count] [int] NOT NULL,

[DateOrder] [nvarchar](max) NOT NULL,

[Status] [nvarchar](max) NOT NULL,

CONSTRAINT [PK\_Order] PRIMARY KEY CLUSTERED

(

[SupplyNumber] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

Создание таблицы «Тестирование»:

CREATE TABLE [dbo].[Test](

[NumberTest] [int] IDENTITY(1,1) NOT NULL,

[Rocket] [int] NOT NULL,

[TypeTest] [nvarchar](max) NOT NULL,

[Result] [nvarchar](max) NOT NULL,

[DateTest] [nvarchar](max) NOT NULL,

[Employee] [int] NOT NULL,

CONSTRAINT [PK\_Test] PRIMARY KEY CLUSTERED

(

[NumberTest] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

Создание таблицы «Транспортировка»:

CREATE TABLE [dbo].[Transportation](

[ID\_Transportation] [int] IDENTITY(1,1) NOT NULL,

[Rocket] [int] NOT NULL,

[DateSend] [nvarchar](max) NOT NULL,

[Status] [nvarchar](max) NOT NULL,

CONSTRAINT [PK\_Transportation] PRIMARY KEY CLUSTERED

(

[ID\_Transportation] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

# **ПРИЛОЖЕНИЕ Б Листинг библиотеки классов**

Файл «ClassAuth.cs»:

using System;

using System.Data.SqlClient;

namespace ClassLibrary

{

public class Auth

{

public object LogInFunc(string login, string password)

{

object IDemployee;

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

SqlCommand cmd = new SqlCommand($"SELECT ID\_Employee from [Employee] WHERE Login='{login}' AND Password='{password}'", conn);

IDemployee = (object)cmd.ExecuteScalar();

}

return IDemployee;

}

public string GetFIO(int IDemployee)

{

string Employee = "";

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

SqlCommand cmd = new SqlCommand($"SELECT Surname, Name, LastName from [Employee] WHERE ID\_Employee='{IDemployee}'", conn);

SqlDataReader rdr = cmd.ExecuteReader();

if (rdr.Read())

{

Employee = rdr[0].ToString() + ' ' + rdr[1].ToString() + ' ' + rdr[2].ToString();

}

rdr.Close();

conn.Close();

}

return Employee;

}

public object GetPost(string IDemployee)

{

object PostEmployee;

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

SqlCommand cmd = new SqlCommand($"SELECT Post from [Employee] WHERE ID\_Employee = {IDemployee}", conn);

PostEmployee = (object)cmd.ExecuteScalar();

}

return PostEmployee;

}

}

}

Файл «ClassRocket.cs»:

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary

{

public class Rocket

{

public bool Add(int Model, string Status, string DateStart, string DateEnd)

{

bool result = false;

Console.WriteLine("Данные для добавления:");

Console.WriteLine($"Model: {Model}");

Console.WriteLine($"Status: {Status}");

Console.WriteLine($"DateStart: {DateStart}");

Console.WriteLine($"DateEnd: {DateEnd}");

try

{

if (Model == 0 || Status == "") { result = false; }

else

{

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

SqlCommand cmd = new SqlCommand($"INSERT INTO [Rocket] (ModelRocket, Status, DateStartBuild, DateEndBuild) VALUES (N'{Model}', N'{Status}', N'{DateStart}', N'{DateEnd}')", conn);

cmd.ExecuteScalar();

result = true;

conn.Close();

}

}

}

catch (Exception sqlEx)

{

result = false;

}

return result;

}

public bool Delete(int id)

{

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=Factory;Integrated Security=True"))

{

conn.Open();

string sql = "DELETE FROM Rocket WHERE ID\_Rocket = @ID";

using (SqlCommand cmd = new SqlCommand(sql, conn))

{

cmd.Parameters.AddWithValue("@ID", id);

int rowsAffected = cmd.ExecuteNonQuery();

return rowsAffected > 0;

}

}

}

}

}

Файл «ClassSP.cs»:

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary

{

public class SP

{

public bool Add(string NameSP, int CountSP)

{

bool result = false;

try

{

if (NameSP == "") { result = false; }

else

{

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

SqlCommand cmd = new SqlCommand($"INSERT INTO [SparePart] (NameSparePart, Count) VALUES (N'{NameSP}', N'{CountSP}')", conn);

cmd.ExecuteScalar();

result = true;

conn.Close();

}

}

}

catch (Exception sqlEx)

{

result = false;

}

return result;

}

public bool Delete(int id)

{

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=Factory;Integrated Security=True"))

{

conn.Open();

string sql = "DELETE FROM SparePart WHERE ID\_SparePart = @ID";

using (SqlCommand cmd = new SqlCommand(sql, conn))

{

cmd.Parameters.AddWithValue("@ID", id);

int rowsAffected = cmd.ExecuteNonQuery();

return rowsAffected > 0;

}

}

}

}

}

Файл «ClassSupply.cs»:

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary

{

public class Supply

{

public bool Add(int SP, int Provider, int Count)

{

bool result = false;

try

{

if (Count == 0) { result = false; }

else

{

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

SqlCommand cmd = new SqlCommand($"INSERT INTO [Supply] (SparePart, Provider, Count, DateOrder, Status) VALUES (N'{SP}', N'{Provider}', N'{Count}', N'{DateTime.Today.ToString("yyyy.MM.dd")}', N'Новый')", conn);

cmd.ExecuteScalar();

result = true;

conn.Close();

}

}

}

catch (Exception sqlEx)

{

result = false;

}

return result;

}

public bool Delete(int id)

{

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=Factory;Integrated Security=True"))

{

conn.Open();

string sql = "DELETE FROM Supply WHERE SupplyNumber = @ID";

using (SqlCommand cmd = new SqlCommand(sql, conn))

{

cmd.Parameters.AddWithValue("@ID", id);

int rowsAffected = cmd.ExecuteNonQuery();

return rowsAffected > 0;

}

}

}

}

}

Файл «ClassTest.cs»:

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary

{

public class Testing

{

public bool Add(int Rocket, string Type, string Result, string Date, int Employee)

{

bool result = false;

try

{

if (Rocket == 0 || Type == "" || Result == "" || Date == "") { result = false; }

else

{

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

SqlCommand cmd = new SqlCommand($"INSERT INTO [Test] (Rocket, TypeTest, Result, DateTest, Employee) VALUES (N'{Rocket}', N'{Type}', N'{Result}', N'{Date}', N'{Employee}')", conn);

cmd.ExecuteScalar();

result = true;

conn.Close();

}

}

}

catch (Exception sqlEx)

{

result = false;

}

return result;

}

public bool Delete(int id)

{

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=Factory;Integrated Security=True"))

{

conn.Open();

string sql = "DELETE FROM Test WHERE NumberTest = @ID";

using (SqlCommand cmd = new SqlCommand(sql, conn))

{

cmd.Parameters.AddWithValue("@ID", id);

int rowsAffected = cmd.ExecuteNonQuery();

return rowsAffected > 0;

}

}

}

}

}

Файл «ClassTransportation.cs»:

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace ClassLibrary

{

public class Transportation

{

public bool Add(int Rocket, string Status, string DateSend)

{

bool result = false;

try

{

if (Rocket == 0 || DateSend == "" || Status == "") { result = false; }

else

{

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

SqlCommand cmd = new SqlCommand($"INSERT INTO [Transportation] (Rocket, DateSend, Status) VALUES (N'{Rocket}', N'{DateSend}', N'{Status}')", conn);

cmd.ExecuteScalar();

result = true;

conn.Close();

}

}

}

catch (Exception sqlEx)

{

result = false;

}

return result;

}

public bool Delete(int id)

{

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

string sql = "DELETE FROM Transportation WHERE ID\_Transportation = @ID";

using (SqlCommand cmd = new SqlCommand(sql, conn))

{

cmd.Parameters.AddWithValue("@ID", id);

int rowsAffected = cmd.ExecuteNonQuery();

return rowsAffected > 0;

}

}

}

}

}

**ПРИЛОЖЕНИЕ В  
Листинг форм**

Файл «AddRocket.cs»:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Zavod

{

public partial class AddRocket : Form

{

int Model;

string Status;

string DateStart;

string DateEnd;

public AddRocket()

{

InitializeComponent();

}

private void buttonAdd\_Click(object sender, EventArgs e)

{

Model = (int)comboBoxModel.SelectedValue;

Status = comboBoxStatus.Text;

DateStart = dateStart.Value.ToString("dd.MM.yyyy");

DateEnd = dateEnd.Value.ToString("dd.MM.yyyy");

ClassLibrary.Rocket cl\_addRocket = new ClassLibrary.Rocket();

bool result = cl\_addRocket.Add(Model, Status, DateStart, DateEnd);

if (result == true)

{

MessageBox.Show("Ракета добавлена!", "Успех!", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Ракета не добавлена!", "Ошибка!", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void AddRocket\_Load(object sender, EventArgs e)

{

// TODO: данная строка кода позволяет загрузить данные в таблицу "factoryDataSet.Model". При необходимости она может быть перемещена или удалена.

this.modelTableAdapter.Fill(this.factoryDataSet.Model);

}

}

}

Файл «AddSparePart.cs»:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Zavod

{

public partial class AddSparePart : Form

{

int InfoEmployee;

string NameSP;

int CountSP;

public AddSparePart(int IDEmployee)

{

InitializeComponent();

InfoEmployee = IDEmployee;

}

private void buttonAdd\_Click(object sender, EventArgs e)

{

Warehouse window = new Warehouse(InfoEmployee);

NameSP = textBoxName.Text;

CountSP = (int)numericUpDownCount.Value;

ClassLibrary.SP cl\_addSP = new ClassLibrary.SP();

bool result = cl\_addSP.Add(NameSP, CountSP);

if (result == true)

{

MessageBox.Show("Деталь добавлена!", "Успех!", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Деталь не добавлена!", "Ошибка!", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

Файл «AddSupply.cs»:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

using System.Windows.Forms.VisualStyles;

using ClassLibrary;

namespace Zavod

{

public partial class AddSupply : Form

{

int SP;

int Provider;

int Count;

public AddSupply()

{

InitializeComponent();

}

private void buttonAdd\_Click(object sender, EventArgs e)

{

SP = (int)comboBoxSP.SelectedValue;

Provider = (int)comboBoxProvider.SelectedValue;

Count = (int)numericUpDownCount.Value;

ClassLibrary.Supply cl\_addSupply = new ClassLibrary.Supply();

bool result = cl\_addSupply.Add(SP, Provider, Count);

if (result == true)

{

MessageBox.Show("Поставка добавлена!", "Успех!", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Поставка не добавлена!", "Ошибка!", MessageBoxButtons.OK, MessageBoxIcon.Error);

MessageBox.Show($"{SP}, {Provider}, {Count}", "Ошибка!", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void AddSupply\_Load(object sender, EventArgs e)

{

this.providerTableAdapter.Fill(this.factoryDataSet.Provider);

this.sparePartTableAdapter.Fill(this.factoryDataSet.SparePart);

}

}

}

Файл «AddTest.cs»:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using ClassLibrary;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.ProgressBar;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace Zavod

{

public partial class AddTest : Form

{

int Rocket;

string Type;

string Result;

string Date;

int InfoEmployee;

public AddTest(int IDemployee)

{

InfoEmployee = IDemployee;

InitializeComponent();

string connectionString = "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=Factory;Integrated Security=True";

string query = @"SELECT

r.ID\_Rocket,

m.NameModel AS ModelName,

CONCAT(r.ID\_Rocket, ' - ', m.NameModel) AS DisplayText

FROM Rocket r

JOIN Model m ON r.ModelRocket = m.ID\_Model";

try

{

DataTable dt = new DataTable();

using (SqlConnection conn = new SqlConnection(connectionString))

using (SqlCommand cmd = new SqlCommand(query, conn))

{

conn.Open();

dt.Load(cmd.ExecuteReader());

}

if (dt.Rows.Count > 0)

{

comboBoxRocket.DataSource = dt;

comboBoxRocket.DisplayMember = "DisplayText";

comboBoxRocket.ValueMember = "ID\_Rocket";

}

else

{

MessageBox.Show("Нет данных о ракетах", "Информация", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (SqlException ex)

{

MessageBox.Show($"Ошибка загрузки данных: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

catch (Exception ex)

{

MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

comboBoxTest.Items.AddRange(new object[] {

"1. Статические (огневые) испытания двигателей",

"2. Вибрационные и ударные испытания",

"3. Термовакуумные тесты",

"4. Испытания на электромагнитную совместимость (ЭМС)",

"5. Функциональные проверки бортовых систем",

"6. Испытания на герметичность",

"7. Лётные испытания (суборбитальные)",

"8. Лётные испытания (орбитальные)",

"9. Криогенные испытания",

"10. Тесты системы аварийного спасения",

"11. Испытания системы разделения ступеней",

"12. Акустические тесты (шумовые нагрузки)",

"13. Климатические испытания",

"14. Тесты программного обеспечения",

"15. Комплексные предпусковые проверки"

});

comboBoxResult.Items.AddRange(new object[] {

"Успешно пройдено",

"Частично пройдено",

"Не пройдено (критическая ошибка)",

"Требуется доработка",

"Тест отменён",

"Результат: в пределах допуска",

"Результат: превышение параметров",

"Требуется повторное тестирование"

});

}

private void AddTest\_Load(object sender, EventArgs e)

{

this.rocketTableAdapter.Fill(this.factoryDataSet.Rocket);

}

private void buttonAdd\_Click(object sender, EventArgs e)

{

Rocket = (int)comboBoxRocket.SelectedValue;

Type = comboBoxTest.Text;

Result = comboBoxResult.Text;

Date = dateTest.Value.ToString("dd.MM.yyyy");

ClassLibrary.Testing cl\_addTest = new ClassLibrary.Testing();

bool result = cl\_addTest.Add(Rocket, Type, Result, Date, InfoEmployee);

if (result == true)

{

MessageBox.Show("Тестирование добавлено!", "Успех!", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Тестирование не добавлено!", "Ошибка!", MessageBoxButtons.OK, MessageBoxIcon.Error);

MessageBox.Show($"{Rocket}, {Type}, {Result}, {Date}", "Ошибка!", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

Файл «AddTranportation.cs»:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace Zavod

{

public partial class AddTransportation : Form

{

int IDemployee;

int Rocket;

string DateSend;

string Status;

public AddTransportation(int IDemployee)

{

InitializeComponent();

string connectionString = "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=Factory;Integrated Security=True";

string query = @"SELECT

r.ID\_Rocket,

m.NameModel AS ModelName,

CONCAT(r.ID\_Rocket, ' - ', m.NameModel) AS DisplayText

FROM Rocket r

JOIN Model m ON r.ModelRocket = m.ID\_Model";

try

{

DataTable dt = new DataTable();

using (SqlConnection conn = new SqlConnection(connectionString))

using (SqlCommand cmd = new SqlCommand(query, conn))

{

conn.Open();

dt.Load(cmd.ExecuteReader());

}

if (dt.Rows.Count > 0)

{

comboBoxRocket.DataSource = dt;

comboBoxRocket.DisplayMember = "DisplayText";

comboBoxRocket.ValueMember = "ID\_Rocket";

}

else

{

MessageBox.Show("Нет данных о ракетах", "Информация", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (SqlException ex)

{

MessageBox.Show($"Ошибка загрузки данных: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

catch (Exception ex)

{

MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void AddTransportation\_FormClosing(object sender, FormClosingEventArgs e)

{

this.Hide();

}

private void AddTransportation\_Load(object sender, EventArgs e)

{

// TODO: данная строка кода позволяет загрузить данные в таблицу "factoryDataSet.Rocket". При необходимости она может быть перемещена или удалена.

this.rocketTableAdapter.Fill(this.factoryDataSet.Rocket);

}

private void buttonAdd\_Click(object sender, EventArgs e)

{

Rocket = (int)comboBoxRocket.SelectedValue;

Status = comboBoxStatus.Text;

DateSend = dateStart.Value.ToString("dd.MM.yyyy");

ClassLibrary.Transportation cl\_addRocket = new ClassLibrary.Transportation();

bool result = cl\_addRocket.Add(Rocket, Status, DateSend);

if (result == true)

{

MessageBox.Show("Транспортировка добавлена!", "Успех!", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Транспортировка не добавлена!", "Ошибка!", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

Файл «Auth.cs»:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using ClassLibrary;

namespace Zavod

{

public partial class Auth : Form

{

static public string login;

static public string password;

static public string post;

public Auth()

{

InitializeComponent();

}

private void buttonLog\_Click(object sender, EventArgs e)

{

login = loginTextBox.Text;

password = passwordTextBox.Text;

if (login == "" || password == "")

{

MessageBox.Show("Пожалуйста, введите логин и/или пароль!", "Внимание!", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

ClassLibrary.Auth cl\_auth = new ClassLibrary.Auth();

object IDemployee = cl\_auth.LogInFunc(login, password);

if (IDemployee != null)

{

Menu window = new Menu(Convert.ToInt32(IDemployee));

this.Hide();

window.Show();

}

else

{

MessageBox.Show("Неверный логин или пароль!", "Внимание!", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void buttonExit\_Click(object sender, EventArgs e)

{

Application.ExitThread();

}

}

}

Файл «Employee.cs»:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Zavod

{

public partial class Employee : Form

{

int InfoEmployee;

public Employee(int IDemployee)

{

InfoEmployee = IDemployee;

InitializeComponent();

}

private void employeeBindingNavigatorSaveItem\_Click(object sender, EventArgs e)

{

this.Validate();

this.employeeBindingSource.EndEdit();

this.tableAdapterManager.UpdateAll(this.factoryDataSet);

}

private void Employee\_Load(object sender, EventArgs e)

{

// TODO: данная строка кода позволяет загрузить данные в таблицу "factoryDataSet.Employee". При необходимости она может быть перемещена или удалена.

this.employeeTableAdapter.Fill(this.factoryDataSet.Employee);

}

private void buttonBack\_Click(object sender, EventArgs e)

{

this.Hide();

Menu window = new Menu(Convert.ToInt32(InfoEmployee));

window.Show();

}

private void Employee\_FormClosing(object sender, FormClosingEventArgs e)

{

this.Hide();

Menu window = new Menu(Convert.ToInt32(InfoEmployee));

window.Show();

}

}

}

Файл «Menu.cs»:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace Zavod

{

public partial class Menu : Form

{

int InfoEmployee;

public Menu(int IDemployee)

{

InfoEmployee = IDemployee;

InitializeComponent();

ClassLibrary.Auth cl\_auth = new ClassLibrary.Auth();

object Employee = cl\_auth.GetFIO(IDemployee);

label1.Text = "Добро пожаловать, " + Employee + "!";

object PostEmployee = cl\_auth.GetPost(IDemployee.ToString());

if ((int)PostEmployee == 1)

{

this.buttonEmployee.Enabled = false;

this.buttonTest.Enabled = false;

}

else if ((int)PostEmployee == 3)

{

this.buttonSupply.Enabled = false;

this.buttonWarehouse.Enabled = false;

this.buttonEmployee.Enabled = false;

this.buttonTransportation.Enabled = false;

}

}

private void buttonWarehouse\_Click(object sender, EventArgs e)

{

this.Hide();

Warehouse window = new Warehouse(Convert.ToInt32(InfoEmployee));

window.Show();

}

private void buttonTransportation\_Click(object sender, EventArgs e)

{

this.Hide();

Transportation window = new Transportation(Convert.ToInt32(InfoEmployee));

window.Show();

}

private void buttonSupply\_Click(object sender, EventArgs e)

{

this.Hide();

Supply window = new Supply(Convert.ToInt32(InfoEmployee));

window.Show();

}

private void buttonTest\_Click(object sender, EventArgs e)

{

this.Hide();

Test window = new Test(Convert.ToInt32(InfoEmployee));

window.Show();

}

private void buttonEmployee\_Click(object sender, EventArgs e)

{

this.Hide();

Employee window = new Employee(Convert.ToInt32(InfoEmployee));

window.Show();

}

private void Menu\_FormClosing(object sender, FormClosingEventArgs e)

{

this.Hide();

Auth window = new Auth();

window.Show();

}

private void buttonExit\_Click(object sender, EventArgs e)

{

this.Hide();

Auth window = new Auth();

window.Show();

}

}

}

Файл «Supply.cs»:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Zavod

{

public partial class Supply : Form

{

int InfoEmployee;

public Supply(int IDemployee)

{

InfoEmployee = IDemployee;

InitializeComponent();

}

private void supplyBindingNavigatorSaveItem\_Click(object sender, EventArgs e)

{

this.Validate();

this.supplyBindingSource.EndEdit();

this.tableAdapterManager.UpdateAll(this.factoryDataSet);

}

private void Supply\_Load(object sender, EventArgs e)

{

// TODO: данная строка кода позволяет загрузить данные в таблицу "factoryDataSet.Supply". При необходимости она может быть перемещена или удалена.

this.supplyTableAdapter.Fill(this.factoryDataSet.Supply);

}

private void buttonAdd\_Click(object sender, EventArgs e)

{

AddSupply window = new AddSupply();

window.ShowDialog();

}

private void buttonDelete\_Click(object sender, EventArgs e)

{

if (supplyDataGridView.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите строку для удаления!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

DataGridViewRow selectedRow = supplyDataGridView.SelectedRows[0];

int id = Convert.ToInt32(selectedRow.Cells[0].Value);

DialogResult result = MessageBox.Show("Удалить выбранную запись?", "Подтверждение", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

ClassLibrary.Transportation cl\_Transportation = new ClassLibrary.Transportation();

if (result == DialogResult.Yes)

{

try

{

if (cl\_Transportation.Delete(id))

{

supplyDataGridView.Rows.Remove(selectedRow);

MessageBox.Show("Запись успешно удалена!", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при удалении: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void buttonBack\_Click(object sender, EventArgs e)

{

this.Hide();

Menu window = new Menu(Convert.ToInt32(InfoEmployee));

window.Show();

}

private void Supply\_FormClosing(object sender, FormClosingEventArgs e)

{

this.Hide();

Menu window = new Menu(Convert.ToInt32(InfoEmployee));

window.Show();

}

}

}

Файл «Test.cs»:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Zavod

{

public partial class Test : Form

{

int InfoEmployee;

public Test(int IDemployee)

{

InfoEmployee = IDemployee;

InitializeComponent();

}

private void testBindingNavigatorSaveItem\_Click(object sender, EventArgs e)

{

this.Validate();

this.testBindingSource.EndEdit();

this.tableAdapterManager.UpdateAll(this.factoryDataSet);

}

private void Test\_Load(object sender, EventArgs e)

{

// TODO: данная строка кода позволяет загрузить данные в таблицу "factoryDataSet.Test". При необходимости она может быть перемещена или удалена.

this.testTableAdapter.Fill(this.factoryDataSet.Test);

}

private void buttonAdd\_Click(object sender, EventArgs e)

{

AddTest window = new AddTest(InfoEmployee);

window.ShowDialog();

}

private void buttonDelete\_Click(object sender, EventArgs e)

{

if (testDataGridView.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите строку для удаления!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

DataGridViewRow selectedRow = testDataGridView.SelectedRows[0];

int id = Convert.ToInt32(selectedRow.Cells[0].Value);

DialogResult result = MessageBox.Show("Удалить выбранную запись?", "Подтверждение", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

ClassLibrary.Testing cl\_Testing = new ClassLibrary.Testing();

if (result == DialogResult.Yes)

{

try

{

if (cl\_Testing.Delete(id))

{

testDataGridView.Rows.Remove(selectedRow);

MessageBox.Show("Запись успешно удалена!", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при удалении: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void buttonBack\_Click(object sender, EventArgs e)

{

this.Hide();

Menu window = new Menu(Convert.ToInt32(InfoEmployee));

window.Show();

}

private void Test\_FormClosing(object sender, FormClosingEventArgs e)

{

this.Hide();

Menu window = new Menu(Convert.ToInt32(InfoEmployee));

window.Show();

}

}

}

Файл «Transportation.cs»:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Zavod

{

public partial class Transportation : Form

{

int InfoEmployee;

public Transportation(int IDemployee)

{

InfoEmployee = IDemployee;

InitializeComponent();

}

private void transportationBindingNavigatorSaveItem\_Click(object sender, EventArgs e)

{

this.Validate();

this.transportationBindingSource.EndEdit();

this.tableAdapterManager.UpdateAll(this.factoryDataSet);

}

private void Transportation\_Load(object sender, EventArgs e)

{

// TODO: данная строка кода позволяет загрузить данные в таблицу "factoryDataSet.Transportation". При необходимости она может быть перемещена или удалена.

this.transportationTableAdapter.Fill(this.factoryDataSet.Transportation);

}

private void buttonBack\_Click(object sender, EventArgs e)

{

this.Hide();

Menu window = new Menu(Convert.ToInt32(InfoEmployee));

window.Show();

}

private void Transportation\_FormClosing(object sender, FormClosingEventArgs e)

{

this.Hide();

Menu window = new Menu(Convert.ToInt32(InfoEmployee));

window.Show();

}

private void buttonAdd\_Click(object sender, EventArgs e)

{

AddTransportation window = new AddTransportation(Convert.ToInt32(InfoEmployee));

window.ShowDialog();

}

private void buttonDelete\_Click(object sender, EventArgs e)

{

if (transportationDataGridView.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите строку для удаления!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

DataGridViewRow selectedRow = transportationDataGridView.SelectedRows[0];

int id = Convert.ToInt32(selectedRow.Cells[0].Value);

DialogResult result = MessageBox.Show("Удалить выбранную запись?", "Подтверждение", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

ClassLibrary.Transportation cl\_Transportation = new ClassLibrary.Transportation();

if (result == DialogResult.Yes)

{

try

{

if (cl\_Transportation.Delete(id))

{

transportationDataGridView.Rows.Remove(selectedRow);

MessageBox.Show("Запись успешно удалена!", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при удалении: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

}

Файл «Warehouse.cs»:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using ClassLibrary;

namespace Zavod

{

public partial class Warehouse : Form

{

int InfoEmployee;

public Warehouse(int IDemployee)

{

InfoEmployee = IDemployee;

InitializeComponent();

}

private void sparePartBindingNavigatorSaveItem\_Click(object sender, EventArgs e)

{

this.Validate();

this.sparePartBindingSource.EndEdit();

this.tableAdapterManager.UpdateAll(this.factoryDataSet);

}

private void Warehouse\_Load(object sender, EventArgs e)

{

// TODO: данная строка кода позволяет загрузить данные в таблицу "factoryDataSet.Rocket". При необходимости она может быть перемещена или удалена.

this.rocketTableAdapter.Fill(this.factoryDataSet.Rocket);

// TODO: данная строка кода позволяет загрузить данные в таблицу "factoryDataSet.SparePart". При необходимости она может быть перемещена или удалена.

this.sparePartTableAdapter.Fill(this.factoryDataSet.SparePart);

}

private void toolStripButton8\_Click(object sender, EventArgs e)

{

AddSparePart window = new AddSparePart(InfoEmployee);

window.Show();

}

private void Warehouse\_FormClosing(object sender, FormClosingEventArgs e)

{

this.Hide();

Menu window = new Menu(Convert.ToInt32(InfoEmployee));

window.Show();

}

private void buttonAddSP\_Click(object sender, EventArgs e)

{

AddSparePart window = new AddSparePart(InfoEmployee);

window.ShowDialog();

}

private void buttonDeleteSP\_Click(object sender, EventArgs e)

{

if (sparePartDataGridView.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите строку для удаления!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

DataGridViewRow selectedRow = sparePartDataGridView.SelectedRows[0];

int id = Convert.ToInt32(selectedRow.Cells[0].Value);

DialogResult result = MessageBox.Show("Удалить выбранную запись?", "Подтверждение", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

ClassLibrary.SP cl\_SP = new ClassLibrary.SP();

if (result == DialogResult.Yes)

{

try

{

if (cl\_SP.Delete(id))

{

sparePartDataGridView.Rows.Remove(selectedRow);

MessageBox.Show("Запись успешно удалена!", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при удалении: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void buttonAddRocket\_Click(object sender, EventArgs e)

{

AddRocket window = new AddRocket();

window.ShowDialog();

}

private void buttonDeleteRocket\_Click(object sender, EventArgs e)

{

if (rocketDataGridView.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите строку для удаления!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

DataGridViewRow selectedRow = rocketDataGridView.SelectedRows[0];

int id = Convert.ToInt32(selectedRow.Cells[0].Value);

DialogResult result = MessageBox.Show("Удалить выбранную запись?", "Подтверждение", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

ClassLibrary.Rocket cl\_rocket = new ClassLibrary.Rocket();

if (result == DialogResult.Yes)

{

try

{

if (cl\_rocket.Delete(id))

{

rocketDataGridView.Rows.Remove(selectedRow);

MessageBox.Show("Запись успешно удалена!", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при удалении: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void buttonBack2\_Click(object sender, EventArgs e)

{

this.Hide();

Menu window = new Menu(Convert.ToInt32(InfoEmployee));

window.Show();

}

private void buttonBack1\_Click(object sender, EventArgs e)

{

this.Hide();

Menu window = new Menu(Convert.ToInt32(InfoEmployee));

window.Show();

}

}

}

**ПРИЛОЖЕНИЕ Г  
Листинг тестов**

Файл «UnitTestaAuth.cs»:

using System;

using ClassLibrary;

using System.Data.SqlClient;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Test

{

[TestClass]

public class UnitTestAuth

{

private Auth \_auth;

private string \_connectionString;

private int \_testEmployeeId;

[TestInitialize]

public void TestInitialize()

{

\_auth = new Auth();

\_connectionString = "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=Factory;Integrated Security=True";

using (SqlConnection conn = new SqlConnection(\_connectionString))

{

conn.Open();

SqlCommand cmd = new SqlCommand(

"INSERT INTO [Employee] (Login, Password, Surname, Name, LastName, Post) " +

"VALUES ('testuser', 'testpass', 'TestSurname', 'TestName', 'TestLastName', '2'); " +

"SELECT SCOPE\_IDENTITY();", conn);

\_testEmployeeId = Convert.ToInt32(cmd.ExecuteScalar());

}

}

[TestCleanup]

public void TestCleanup()

{

using (SqlConnection conn = new SqlConnection(\_connectionString))

{

conn.Open();

SqlCommand cmd = new SqlCommand(

"DELETE FROM [Employee] WHERE Login = 'testuser' OR ID\_Employee = @testId",

conn);

cmd.Parameters.AddWithValue("@testId", \_testEmployeeId);

cmd.ExecuteNonQuery();

}

}

[TestMethod]

public void LogInFunc\_ValidCredentials\_ReturnsEmployeeId()

{

string login = "testuser";

string password = "testpass";

var result = \_auth.LogInFunc(login, password);

Assert.IsNotNull(result);

Assert.IsInstanceOfType(result, typeof(int));

Assert.AreEqual(\_testEmployeeId, Convert.ToInt32(result));

}

[TestMethod]

public void LogInFunc\_InvalidLogin\_ReturnsNull()

{

string login = "invaliduser";

string password = "testpass";

var result = \_auth.LogInFunc(login, password);

Assert.IsNull(result);

}

[TestMethod]

public void LogInFunc\_InvalidPassword\_ReturnsNull()

{

string login = "testuser";

string password = "wrongpass";

var result = \_auth.LogInFunc(login, password);

Assert.IsNull(result);

}

[TestMethod]

public void LogInFunc\_EmptyCredentials\_ReturnsNull()

{

string login = "";

string password = "";

var result = \_auth.LogInFunc(login, password);

Assert.IsNull(result);

}

[TestMethod]

public void GetFIO\_ValidEmployeeId\_ReturnsFullName()

{

int employeeId = \_testEmployeeId;

string expected = "TestSurname TestName TestLastName";

var result = \_auth.GetFIO(employeeId);

Assert.AreEqual(expected, result);

}

[TestMethod]

public void GetFIO\_InvalidEmployeeId\_ReturnsEmptyString()

{

int employeeId = -1;

var result = \_auth.GetFIO(employeeId);

Assert.AreEqual(string.Empty, result);

}

[TestMethod]

public void GetPost\_ValidEmployeeId\_ReturnsPost()

{

string employeeId = \_testEmployeeId.ToString();

string expected = "2";

var result = \_auth.GetPost(employeeId);

Assert.IsNotNull(result);

Assert.AreEqual(expected, result.ToString());

}

[TestMethod]

public void GetPost\_InvalidEmployeeId\_ReturnsNull()

{

string employeeId = "-1";

var result = \_auth.GetPost(employeeId);

Assert.IsNull(result);

}

}

}

Файл «UnitTestRocket.cs»:

using System;

using System.Data.SqlClient;

using ClassLibrary;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Test

{

[TestClass]

public class UnitTestRocket

{

[TestClass]

public class RocketTests

{

private Rocket \_rocket;

private const int TestModel = 1;

private const string TestStatus = "Active";

private const string TestDateStart = "01.01.2023";

private const string TestDateEnd = "31.12.2023";

private int \_testId;

[TestInitialize]

public void Setup()

{

\_rocket = new Rocket();

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

var cmd = new SqlCommand(

$"INSERT INTO [Rocket] (ModelRocket, Status, DateStartBuild, DateEndBuild) " +

$"VALUES ('{TestModel}', N'{TestStatus}', '{TestDateStart}', '{TestDateEnd}'); " +

"SELECT SCOPE\_IDENTITY();", conn);

\_testId = Convert.ToInt32(cmd.ExecuteScalar());

conn.Close();

}

}

[TestCleanup]

public void Cleanup()

{

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

var cmd = new SqlCommand(

"DELETE FROM [Rocket] WHERE " +

"(ModelRocket = @Model AND Status = @Status) OR " +

"ID\_Rocket = @TestId", conn);

cmd.Parameters.AddWithValue("@Model", TestModel);

cmd.Parameters.AddWithValue("@Status", TestStatus);

cmd.Parameters.AddWithValue("@TestId", \_testId);

cmd.ExecuteNonQuery();

conn.Close();

}

}

[TestMethod]

public void Add\_ValidData\_ReturnsTrue()

{

bool result = \_rocket.Add(TestModel, TestStatus, TestDateStart, TestDateEnd);

Assert.IsTrue(result);

}

[TestMethod]

public void Add\_ZeroModel\_ReturnsFalse()

{

bool result = \_rocket.Add(0, TestStatus, TestDateStart, TestDateEnd);

Assert.IsFalse(result);

}

[TestMethod]

public void Add\_EmptyStatus\_ReturnsFalse()

{

bool result = \_rocket.Add(TestModel, "", TestDateStart, TestDateEnd);

Assert.IsFalse(result);

}

[TestMethod]

public void Add\_InvalidConnection\_ReturnsFalse()

{

var brokenRocket = new BrokenConnectionRocket();

bool result = brokenRocket.Add(TestModel, TestStatus, TestDateStart, TestDateEnd);

Assert.IsFalse(result);

}

[TestMethod]

public void Delete\_ExistingId\_ReturnsTrue()

{

bool result = \_rocket.Delete(\_testId);

Assert.IsTrue(result);

}

[TestMethod]

public void Delete\_NonExistingId\_ReturnsFalse()

{

bool result = \_rocket.Delete(-1);

Assert.IsFalse(result);

}

[TestMethod]

public void Delete\_InvalidConnection\_ReturnsFalse()

{

var brokenRocket = new BrokenConnectionRocket();

bool result = brokenRocket.Delete(1);

Assert.IsFalse(result);

}

}

public class BrokenConnectionRocket : Rocket

{

public virtual bool Add(int Model, string Status, string DateStart, string DateEnd)

{

try

{

using (SqlConnection conn = new SqlConnection("Data Source=invalid;Initial Catalog=Factory;Integrated Security=True"))

{

conn.Open();

return false;

}

}

catch

{

return false;

}

}

public virtual bool Delete(int id)

{

try

{

using (SqlConnection conn = new SqlConnection("Data Source=invalid;Initial Catalog=Factory;Integrated Security=True"))

{

conn.Open();

return false;

}

}

catch

{

return false;

}

}

}

}

}

Файл «UnitTestSP.cs»:

using System;

using ClassLibrary;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Data.SqlClient;

namespace Test

{

[TestClass]

public class UnitTestSP

{

private SP \_sp;

private const string TestName = "TestPart";

private const int TestCount = 10;

private int \_testId;

[TestInitialize]

public void Setup()

{

\_sp = new SP();

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

var cmd = new SqlCommand($"INSERT INTO [SparePart] (NameSparePart, Count) VALUES (N'{TestName}', '{TestCount}'); SELECT SCOPE\_IDENTITY();", conn);

\_testId = Convert.ToInt32(cmd.ExecuteScalar());

conn.Close();

}

}

[TestCleanup]

public void Cleanup()

{

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

var cmd = new SqlCommand($"DELETE FROM [SparePart] WHERE NameSparePart = N'{TestName}' OR NameSparePart LIKE N'TestPart%'", conn);

cmd.ExecuteNonQuery();

conn.Close();

}

}

[TestMethod]

public void Add\_ValidData\_ReturnsTrue()

{

bool result = \_sp.Add("TestPart1", 5);

Assert.IsTrue(result);

}

[TestMethod]

public void Add\_EmptyName\_ReturnsFalse()

{

bool result = \_sp.Add("", 5);

Assert.IsFalse(result);

}

[TestMethod]

public void Add\_NullName\_ReturnsFalse()

{

var sp = new SP();

bool result = sp.Add(string.Empty, 5);

Assert.IsFalse(result);

}

[TestMethod]

public void Add\_InvalidConnection\_ReturnsFalse()

{

var brokenSp = new BrokenConnectionSP();

bool result = brokenSp.Add("TestPart3", 5);

Assert.IsFalse(result);

}

[TestMethod]

public void Delete\_ExistingId\_ReturnsTrue()

{

bool result = \_sp.Delete(\_testId);

Assert.IsTrue(result);

}

[TestMethod]

public void Delete\_NonExistingId\_ReturnsFalse()

{

bool result = \_sp.Delete(-1);

Assert.IsFalse(result);

}

[TestMethod]

public void Delete\_InvalidConnection\_ReturnsFalse()

{

var brokenSp = new BrokenConnectionSP();

bool result = brokenSp.Delete(1);

Assert.IsFalse(result);

}

}

public class BrokenConnectionSP : SP

{

public virtual bool Add(string NameSP, int CountSP)

{

try

{

using (SqlConnection conn = new SqlConnection("Data Source=invalid;Initial Catalog=Factory;Integrated Security=True"))

{

conn.Open();

return false;

}

}

catch

{

return false;

}

}

public virtual bool Delete(int id)

{

try

{

using (SqlConnection conn = new SqlConnection("Data Source=invalid;Initial Catalog=Factory;Integrated Security=True"))

{

conn.Open();

return false;

}

}

catch

{

return false;

}

}

}

}

Файл «UnitTestSupply.cs»:

using System;

using ClassLibrary;

using System.Data.SqlClient;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Test

{

[TestClass]

public class UnitTestSupply

{

private Supply \_supply;

private const int TestSparePart = 1;

private const int TestProvider = 1;

private const int TestCount = 10;

private int \_testId;

[TestInitialize]

public void Setup()

{

\_supply = new Supply();

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

var cmd = new SqlCommand(

$"INSERT INTO [Supply] (SparePart, Provider, Count, DateOrder, Status) " +

$"VALUES ('{TestSparePart}', '{TestProvider}', '{TestCount}', '{DateTime.Today.ToString("yyyy.MM.dd")}', 'Новый'); " +

"SELECT SCOPE\_IDENTITY();", conn);

\_testId = Convert.ToInt32(cmd.ExecuteScalar());

conn.Close();

}

}

[TestCleanup]

public void Cleanup()

{

using (SqlConnection conn = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB; Initial Catalog = Factory; Integrated Security = True"))

{

conn.Open();

var cmd = new SqlCommand(

"DELETE FROM [Supply] WHERE " +

"(SparePart = @SparePart AND Provider = @Provider) OR " +

"SupplyNumber = @TestId", conn);

cmd.Parameters.AddWithValue("@SparePart", TestSparePart);

cmd.Parameters.AddWithValue("@Provider", TestProvider);

cmd.Parameters.AddWithValue("@TestId", \_testId);

cmd.ExecuteNonQuery();

conn.Close();

}

}

[TestMethod]

public void Add\_ValidData\_ReturnsTrue()

{

bool result = \_supply.Add(TestSparePart, TestProvider, TestCount);

Assert.IsTrue(result);

}

[TestMethod]

public void Add\_ZeroCount\_ReturnsFalse()

{

bool result = \_supply.Add(TestSparePart, TestProvider, 0);

Assert.IsFalse(result);

}

[TestMethod]

public void Add\_InvalidConnection\_ReturnsFalse()

{

var brokenSupply = new BrokenConnectionSupply();

bool result = brokenSupply.Add(TestSparePart, TestProvider, TestCount);

Assert.IsFalse(result);

}

[TestMethod]

public void Delete\_ExistingId\_ReturnsTrue()

{

bool result = \_supply.Delete(\_testId);

Assert.IsTrue(result);

}

[TestMethod]

public void Delete\_NonExistingId\_ReturnsFalse()

{

bool result = \_supply.Delete(-1);

Assert.IsFalse(result);

}

[TestMethod]

public void Delete\_InvalidConnection\_ReturnsFalse()

{

var brokenSupply = new BrokenConnectionSupply();

bool result = brokenSupply.Delete(1);

Assert.IsFalse(result);

}

}

public class BrokenConnectionSupply : Supply

{

public virtual bool Add(int SP, int Provider, int Count)

{

try

{

using (SqlConnection conn = new SqlConnection("Data Source=invalid;Initial Catalog=Factory;Integrated Security=True"))

{

conn.Open();

return false;

}

}

catch

{

return false;

}

}

public virtual bool Delete(int id)

{

try

{

using (SqlConnection conn = new SqlConnection("Data Source=invalid;Initial Catalog=Factory;Integrated Security=True"))

{

conn.Open();

return false;

}

}

catch

{

return false;

}

}

}

}

Файл «UnitTestTesting.cs»:

using System;

using ClassLibrary;

using System.Data.SqlClient;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Test

{

[TestClass]

public class UnitTestTesting

{

private Testing \_testing;

private string \_connectionString;

private int \_testId;

[TestInitialize]

public void TestInitialize()

{

\_testing = new Testing();

\_connectionString = "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=Factory;Integrated Security=True";

using (SqlConnection conn = new SqlConnection(\_connectionString))

{

conn.Open();

SqlCommand cmd = new SqlCommand(

"INSERT INTO [Test] (Rocket, TypeTest, Result, DateTest, Employee) " +

"VALUES (11, 'UnitTest', 'Success', '2023-01-01', 11); " +

"SELECT SCOPE\_IDENTITY();", conn);

\_testId = Convert.ToInt32(cmd.ExecuteScalar());

}

}

[TestCleanup]

public void TestCleanup()

{

using (SqlConnection conn = new SqlConnection(\_connectionString))

{

conn.Open();

SqlCommand cmd = new SqlCommand(

"DELETE FROM [Test] WHERE Rocket = 11 OR Employee = 11 OR TypeTest = 'UnitTest'",

conn);

cmd.ExecuteNonQuery();

}

}

[TestMethod]

public void Add\_ValidData\_ReturnsTrue()

{

int rocket = 11;

string type = "UnitTest";

string result = "Success";

string date = "2023-01-01";

int employee = 11;

bool actual = \_testing.Add(rocket, type, result, date, employee);

Assert.IsTrue(actual);

}

[TestMethod]

public void Add\_InvalidRocket\_ReturnsFalse()

{

int rocket = 0;

string type = "UnitTest";

string result = "Success";

string date = "2023-01-01";

int employee = 11;

bool actual = \_testing.Add(rocket, type, result, date, employee);

Assert.IsFalse(actual);

}

[TestMethod]

public void Add\_EmptyType\_ReturnsFalse()

{

int rocket = 11;

string type = "";

string result = "Success";

string date = "2023-01-01";

int employee = 11;

bool actual = \_testing.Add(rocket, type, result, date, employee);

Assert.IsFalse(actual);

}

[TestMethod]

public void Add\_EmptyResult\_ReturnsFalse()

{

int rocket = 11;

string type = "UnitTest";

string result = "";

string date = "2023-01-01";

int employee = 11;

bool actual = \_testing.Add(rocket, type, result, date, employee);

Assert.IsFalse(actual);

}

[TestMethod]

public void Add\_EmptyDate\_ReturnsFalse()

{

int rocket = 11;

string type = "UnitTest";

string result = "Success";

string date = "";

int employee = 11;

bool actual = \_testing.Add(rocket, type, result, date, employee);

Assert.IsFalse(actual);

}

[TestMethod]

public void Delete\_NonExistingId\_ReturnsFalse()

{

int nonExistingId = -1;

bool actual = \_testing.Delete(nonExistingId);

Assert.IsFalse(actual);

}

}

}

Файл «UnitTestTrasportation.cs»:

using System;

using ClassLibrary;

using System.Data.SqlClient;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Test

{

[TestClass]

public class UnitTestTransportation

{

private Transportation \_transportation;

private string \_connectionString;

private int \_testTransportationId;

[TestInitialize]

public void TestInitialize()

{

\_transportation = new Transportation();

\_connectionString = "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=Factory;Integrated Security=True";

using (SqlConnection conn = new SqlConnection(\_connectionString))

{

conn.Open();

SqlCommand cmd = new SqlCommand(

"INSERT INTO [Transportation] (Rocket, DateSend, Status) " +

"VALUES (11, '2023-01-01', 'TestStatus'); " +

"SELECT SCOPE\_IDENTITY();", conn);

\_testTransportationId = Convert.ToInt32(cmd.ExecuteScalar());

}

}

[TestCleanup]

public void TestCleanup()

{

using (SqlConnection conn = new SqlConnection(\_connectionString))

{

conn.Open();

SqlCommand cmd = new SqlCommand(

"DELETE FROM [Transportation] WHERE Rocket = 999 OR ID\_Transportation = @testId",

conn);

cmd.Parameters.AddWithValue("@testId", \_testTransportationId);

cmd.ExecuteNonQuery();

}

}

[TestMethod]

public void Add\_ValidData\_ReturnsTrue()

{

int rocket = 11;

string status = "TestStatus";

string dateSend = "2023-01-01";

bool result = \_transportation.Add(rocket, status, dateSend);

Assert.IsTrue(result);

using (SqlConnection conn = new SqlConnection(\_connectionString))

{

conn.Open();

SqlCommand cmd = new SqlCommand(

"SELECT COUNT(\*) FROM [Transportation] WHERE Rocket = @rocket AND Status = @status AND DateSend = @dateSend",

conn);

cmd.Parameters.AddWithValue("@rocket", rocket);

cmd.Parameters.AddWithValue("@status", status);

cmd.Parameters.AddWithValue("@dateSend", dateSend);

int count = Convert.ToInt32(cmd.ExecuteScalar());

Assert.IsTrue(count > 0);

}

}

[TestMethod]

public void Add\_InvalidRocket\_ReturnsFalse()

{

int rocket = 0;

string status = "TestStatus";

string dateSend = "2023-01-01";

bool result = \_transportation.Add(rocket, status, dateSend);

Assert.IsFalse(result);

}

[TestMethod]

public void Add\_EmptyStatus\_ReturnsFalse()

{

int rocket = 11;

string status = "";

string dateSend = "2023-01-01";

bool result = \_transportation.Add(rocket, status, dateSend);

Assert.IsFalse(result);

}

[TestMethod]

public void Add\_EmptyDateSend\_ReturnsFalse()

{

int rocket = 11;

string status = "TestStatus";

string dateSend = "";

bool result = \_transportation.Add(rocket, status, dateSend);

Assert.IsFalse(result);

}

[TestMethod]

public void Delete\_ExistingId\_ReturnsTrue()

{

bool result = \_transportation.Delete(\_testTransportationId);

Assert.IsTrue(result);

using (SqlConnection conn = new SqlConnection(\_connectionString))

{

conn.Open();

SqlCommand cmd = new SqlCommand(

"SELECT COUNT(\*) FROM [Transportation] WHERE ID\_Transportation = @id",

conn);

cmd.Parameters.AddWithValue("@id", \_testTransportationId);

int count = Convert.ToInt32(cmd.ExecuteScalar());

Assert.AreEqual(0, count);

}

}

[TestMethod]

public void Delete\_NonExistingId\_ReturnsFalse()

{

int nonExistingId = -1;

bool result = \_transportation.Delete(nonExistingId);

Assert.IsFalse(result);

}

[TestMethod]

public void Delete\_AlreadyDeletedId\_ReturnsFalse()

{

\_transportation.Delete(\_testTransportationId);

bool result = \_transportation.Delete(\_testTransportationId);

Assert.IsFalse(result);

}

}

}