

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Программные средства информационной
поддержки планирования и организации работ
в электроэнергетике**

Студент

(подпись, дата)

Верясов Денис Евгеньевич
(ФИО полностью)

Руководитель

(подпись, дата)

Самуйлов С.В.
(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись)

Козлов А.Ю.
(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.
(фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов «26»
декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Верясов Денис Евгеньевич гр. 22ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства информационной поддержки планирования и организации работ в электроэнергетике

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы к.т.н., доцент кафедры "МО и ПЭВМ" Самуйлов С.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Назначение: разработка программных средств информационной поддержки планирования и организации работ в электроэнергетике

2) Среда выполнения: Windows 10.

Языковые средства разработки: C#, HTML, CSS, JavaScript, SQL, UML.

Инструментальные средства: Visual Studio 2022, SQL Server Management Studio 20, Visual Paradigm 17.3.

Технологии разработки: ООП, многослойная архитектура.

3) Основные функции приложения:

а) авторизация с разграничением доступа;

б) управление данными в БД (создание, редактирование, удаление);

в) просмотр информации из БД;

г) управление аккаунтами (создание, редактирование, удаление);

д) выполнить расчёт (первичный, детальный);

е) просмотр истории расчётов;

ж) управление расчётом (сохранение, удаление).

5. Перечень подлежащих разработке вопросов:

- 1) Анализ предметной области и постановка задачи;
- 2) Проектирование;
- 3) Реализация;
- 4) Контроль качества программного обеспечения;
- 5) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 117 листов, 47 рисунков, 35 таблиц, 20 использованных источников и 7 приложений.

ПОДДЕРЖКА И ПЛАНИРОВАНИЕ РАБОТ В ЭЛЕКТРОЭНЕРГЕТИКЕ, WEB-ПРИЛОЖЕНИЕ, UML, SQL, C#, ASP.NET CORE, JAVASCRIPT, БАЗА ДАННЫХ.

Объект разработки: программные средства информационной поддержки планирования и организации работ в электроэнергетике.

Цель разработки: спроектировать, затем реализовать программные средства информационной поддержки планирования и организации работ в электроэнергетике.

Результаты разработки: в рамках выпускной квалификационной работы были реализованы клиентская и серверная часть приложения, предназначенного для информационной поддержки планирования и организации работ в электроэнергетике. Все предъявленные требования к созданию приложения выполнены.

Средства разработки: была разработана клиентская часть приложения на языке программирования JavaScript. В среде разработки Microsoft Visual Studio на языке программирования C# была реализована серверная часть приложения. В качестве системы управления базами данных была выбрана Microsoft SQL Server.

					ПГУ 09.03.04-8BP222.04 ПЗ						
Изм.	Лист	№ докум.	Подпись	Дата							
Разраб.		Верясов Д.Е.			Программные средства информационной поддержки планирования и организации работ в электроэнергетике	Лит.		Лист		Листов	
Провер.		Самуйлов С.В.						4		117	
Реценз.						Группа 22ВП2					
Н. Контр.		Такташкин Д.В.									
Утверд.		Козлов А.Ю.									

Содержание

Введение.....	7
1 Постановка задачи и анализ требований	8
1.1 Постановка задачи на разработку приложения	8
1.2 Анализ программ-аналогов.....	8
1.3 Сравнение систем для расчёта стоимости строительных, электромонтажных и ремонтных работ.....	11
1.4 Построение исходной концептуальной модели.....	12
1.5 Анализ функциональных требований.....	20
1.6 Анализ нефункциональных требований.....	22
2 Проектирование приложения для организации и поддержки работ в электроэнергетике.....	24
2.1 Архитектура приложения.....	24
2.2 Проектирование структур данных	26
2.2.1 Концептуальная модель базы данных	26
2.2.2 Логическая модель базы данных.....	27
2.3 Проектирование программных средств.....	27
2.4 Проектирование пользовательского интерфейса	30
2.5 Выбор технологий и инструментальных средств разработки приложения.....	32
2.5.1 Выбор технологий разработки приложения	32
2.5.2 Выбор инструментальных средств разработки приложения	33
3 Разработка программного обеспечения.....	35
3.1. Разработка физической модели базы данных	35
3.2. Реализация серверной части информационной системы.....	36

	6
3.3. Реализация клиентской части информационной системы.....	38
3.4. Реализация интерфейса приложения	40
3.4.1 Реализация интерфейса клиента.....	40
3.4.2 Реализация интерфейса администратора.....	46
4 Отладка и тестирование.	50
4.1 Обоснование выбора режима тестирования.....	50
4.2 Тестирование приложения для планирования и организации работ в электроэнергетике.....	51
4.2.1 Функциональное тестирование.	51
5 Оценка качества программного кода.	56
5.1 Тестирование SLOC.....	56
5.2 Метрика Холстеда.....	58
5.3 Метрика Чепина	60
5.4 Автоматическая проверка.	61
6 Планирование разработки и оценка бюджета.....	62
Заключение	65
Список использованных источников	67
Приложение А (справочное) Голоссарий.....	69
Приложение Б (обязательное) Листинг серверной части приложения	71
Приложение В (обязательное) Листинг клиентской части приложения	78
Приложение Г (рекомендуемое) Листинг модульных тестов	92
Приложение Д (рекомендуемое) Результаты тестирования.....	99
Приложение Е (рекомендуемое) Графическая часть	113
Приложение Ж (рекомендуемое) Акт о внедрении.....	116

Введение

В современном мире, где всё зависит от электричества, люди уже не представляют свою жизнь без него, современные компьютеры, холодильники, все средства аудио-видео информации зависят от него. Современные заводы, магазины нуждаются в правильном и стабильном подключении в электросеть, поэтому его распространение очень необходимо сейчас и поддержка планирования и организации работ в электроэнергетике очень актуально в наше время.

В ходе разработки приложения были проведены:

- анализ предметной области;
- анализ требований к системе;
- проектирование логических моделей баз данных;
- разработка диаграммы вариантов использования;
- проектирование базы данных приложения;
- разработка приложения;
- тестирование приложения;
- планирование разработки.

При создании приложения использовались современные подходы к проектированию и разработке программного обеспечения.

1 Постановка задачи и анализ требований

1.1 Постановка задачи на разработку приложения

Основной задачей выпускной квалификационной работы является разработка приложения для информационной поддержки планирования и организации работ в электроэнергетике.

В рамках выпускной квалификационной работы необходимо разработать приложение, предоставляющее возможность рассчитать примерную стоимость работ по минимально известным данным. Помимо первичного расчёта, программа должна иметь возможность рассчитать стоимость работ по подробным данным для более точного расчёта стоимости.

Для хранения информации о расчётах, клиентах, работниках, бригадах, типах работ, типах проводов, о сечениях проводов, самих проводах и о местности, на которой проводятся работы необходимо использовать реляционные СУБД. Это позволит сохранять в структурированной форме всю информацию о мероприятиях, их состояниях и покупках пользователей.

Для удобства использования приложения его интерфейс должен быть разбит на несколько основных компонентов: – главная страница; – список расчётов. На главной странице необходимо отобразить виды расчётов работ. На странице списков расчётов должны быть видны списки первичных и подробных расчётов.

Администратор может просматривать все основные страницы приложения и пользоваться всеми его функциями.

1.2 Анализ программ-аналогов

Существует несколько программ, которые могут выполнять аналогичные действия для расчёта стоимости строительных, электромонтажных и ремонтных работ. Есть как сложные корпоративные системы, так и более простые и специализированные приложения.

Система «R-Keeper Repairs» – это облачный сервис для управления обслуживанием и ремонтами. Система универсальна, у неё гибкий функционал в использовании как для расчётов стоимости и планирования работ, так и для создания и учёта заявок (рисунок 1).

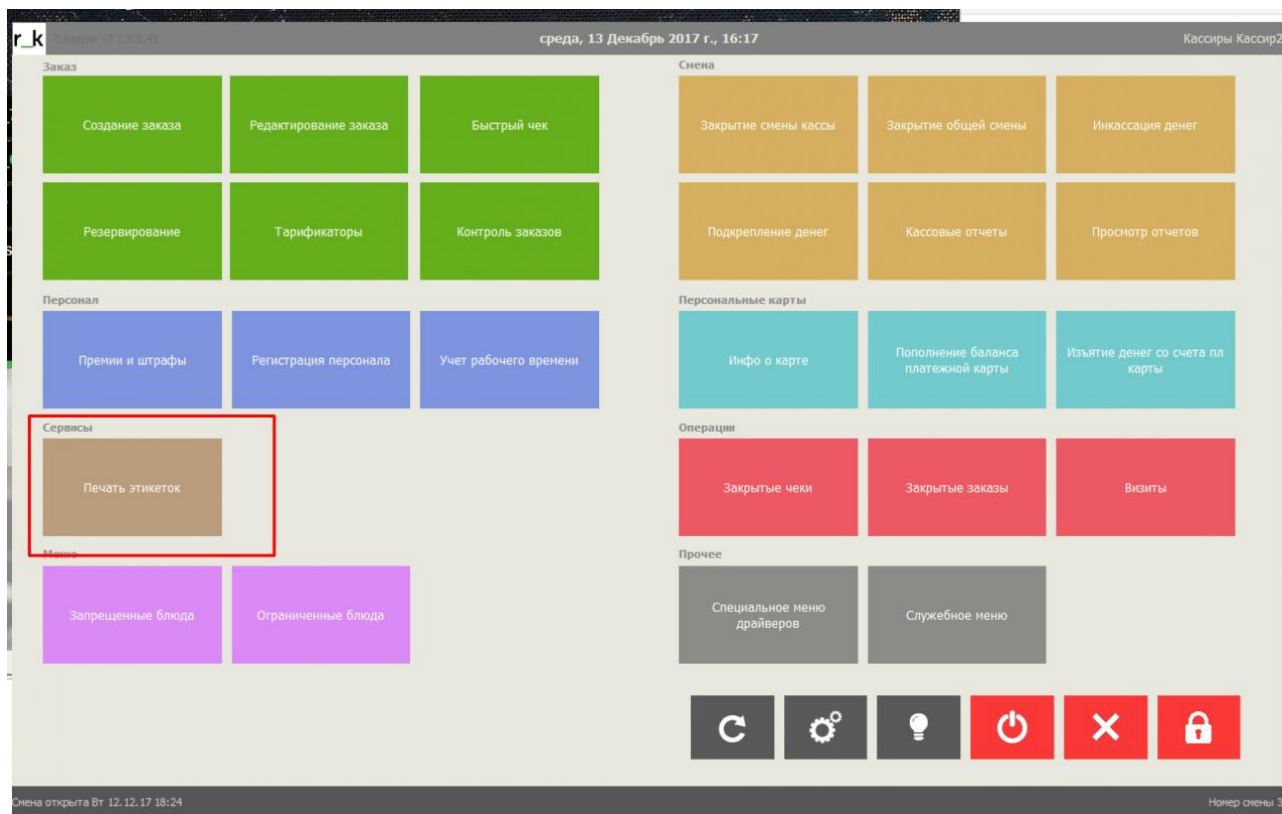


Рисунок 1 – Интерфейс системы «R-Keeper Repairs»

Преимущества данной системы в её обширном функционале для управления задачами, бригадами и заказчиками, такими как: удобное планирование работ и отслеживание их статусов, возможность формирования отчетов и аналитики, интеграция со смежными сервисами.

Недостатки данной системы в её высокой стоимости внедрения и ежемесячной подписки для малого бизнеса, избыточной сложности и универсальности для узкоспециализированной задачи расчета стоимости электромонтажных работ, отсутствии встроенных специализированных калькуляторов для электротехнической отрасли, что требует дополнительной настройки.

Система «1С:ERP Управление строительной организацией» это отечественная программа для управления строительным предприятием. Она включает в себя сметное планирование и расчет себестоимости работ (рисунок 2).

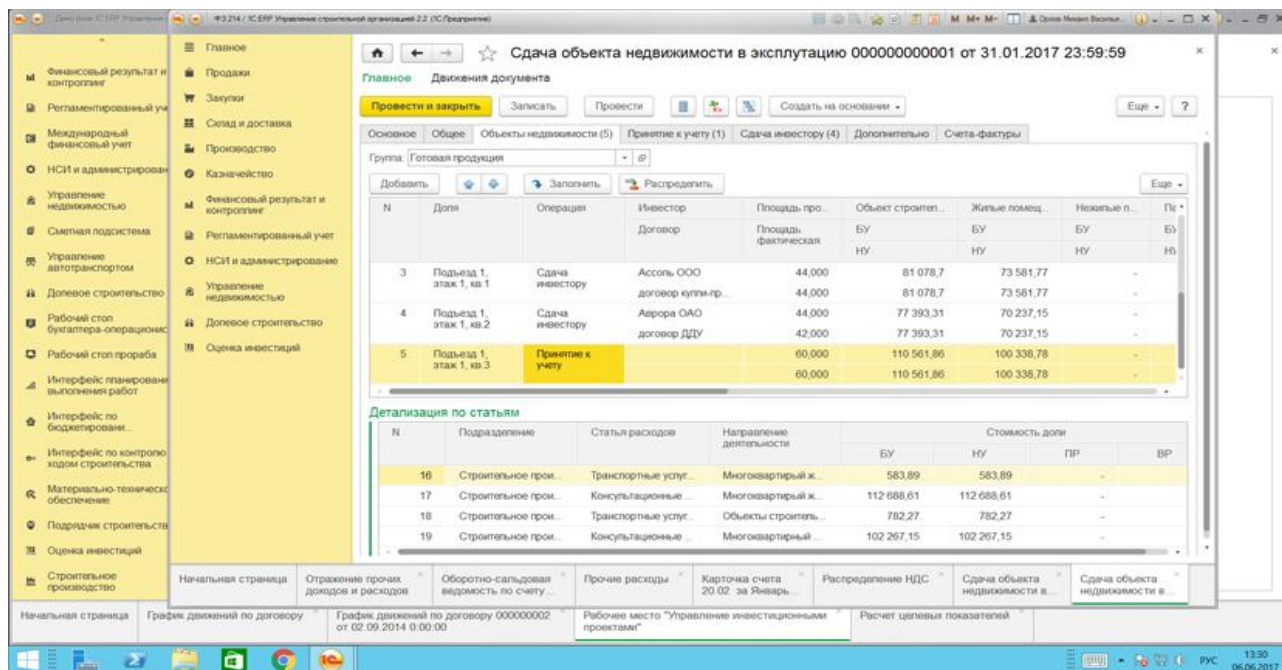


Рисунок 2 – Интерфейс системы «1С:ERP Управление строительной организацией»

Данная система может выполнить следующие задачи: составление смет и калькуляция стоимости работ на основе нормативных баз, управление ресурсами: материалами, персоналом, техникой, планирование проектов и контроль их исполнения, углубленный финансовый учет и анализ затрат, формирование регламентированной отчетности.

Её недостатки: очень высокая сложность внедрения и настройки под конкретные нужды, значительная стоимость лицензий и сопровождения, она требует глубокого обучения персонала, она не является специализированным решением для электромонтажных работ, многие функции могут остаться невостребованными.

Система «Renga» – это программное обеспечение для информационного моделирования в строительстве, разработанное компанией «Аскон». Данная

система предоставляет инструменты для проектирования и расчета различных аспектов строительных проектов (рисунок 3).

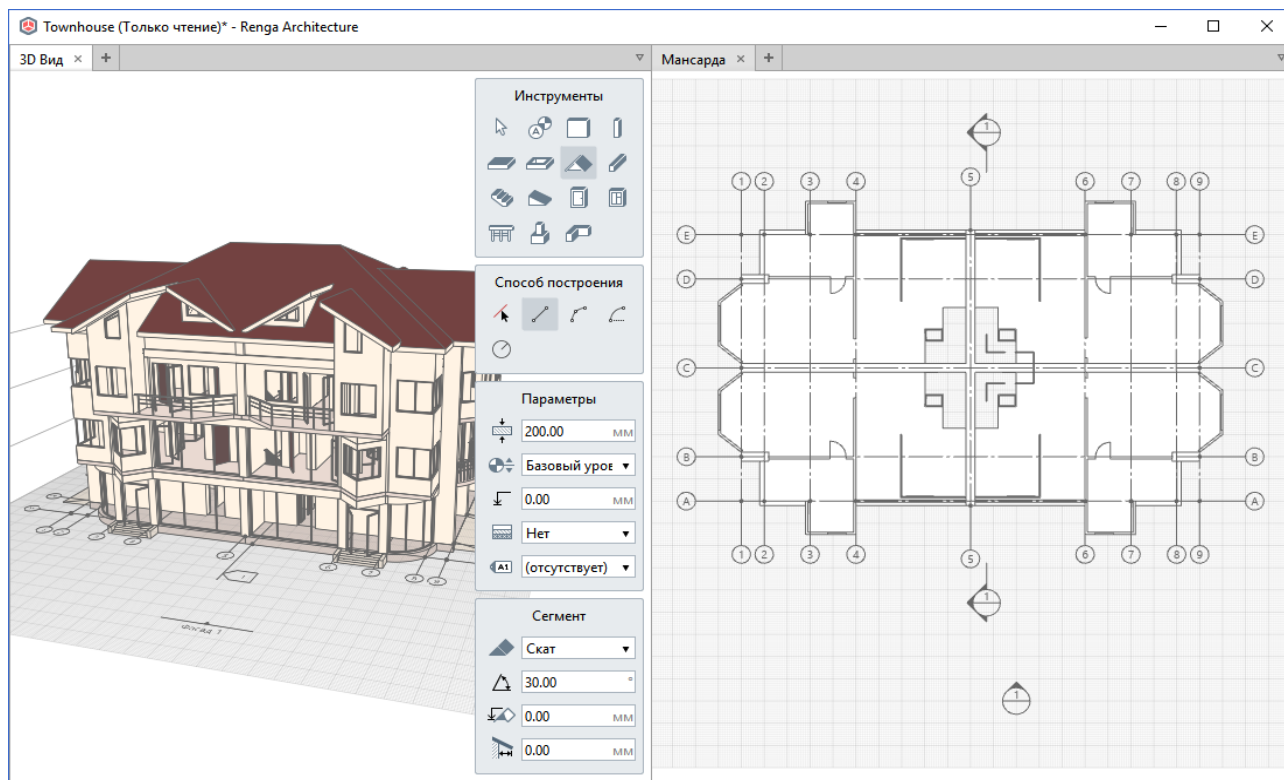


Рисунок 3 – Интерфейс системы «Renga»

К преимуществам системы относится современный интерфейс, поддержка технологии BIM, а также возможность интеграции с другими системами.

Недостатками является ограниченный функционал для расчета стоимости работ без дополнительных модулей, высокая стоимость лицензий и требовательность к ресурсам компьютера, что ограничивает ее применение в небольших организациях.

1.3 Сравнение систем для расчёта стоимости строительных, электромонтажных и ремонтных работ

Для проведения сравнительного анализа информационных систем, расчёта стоимости строительных, электромонтажных и ремонтных работ был определён ряд критериев, на основе которых оценивались существующие решения. Система оценивалась по следующей шкале:

– 0 баллов: функция отсутствует или выполнена неудовлетворительно;

– 1 балл: функция реализована частично или со значительными ограничениями;

– 2 балла: функция реализована полноценно и качественно.

Результаты оценки представлены в таблице 1.

Таблица 1 – Сравнение систем аналогичного назначения

Критерий	R-Keeper Repairs	1C:ERP Управление строительной организацией	Renga
Наличие основных функций для расчёта электромонтажных работ	1	2	1
Простота и удобство интерфейса для неопытных пользователей	2	1	1
Встроенная работа с документами	1	2	1
Интуитивность интерфейса	2	1	0
Итого	6	6	3

Таким образом, существующие аналоги либо слишком сложны и дороги, либо узкоспециализированы и не покрывают всех потребностей в расчете стоимости электромонтажных работ. Принято решение разработать специализированное десктопное приложение, которое совместит в себе простоту первичного расчета, точность подробного расчёта, используя реляционную СУБД для хранения структурированных данных.

1.4 Построение исходной концептуальной модели

Для построения концептуальной модели нужно выделить все типы сущностей предметной области, описать их атрибуты и установить связи между ними. Затем создать графическое представление модели, отображающие сущности, их свойства и отношения.

В результате построения концептуальной модели были выделены сущности предметной области, представленные в таблице 2.

Таблица 2 – Типы сущностей предметной области

Наименование	Описание	Категория	Кол-во экземпляров	Режим обновления
Расчёт	Список рассчитанной стоимости заказа	Стержневая	1000	Ежедневно
Типы местности	Список типов местности, на которой производятся работы	Характеристическая	10	-
Типы проводов	Список проводов, используемых на работах	Характеристическая	10	-
Типы сечения	Список сечений проводов, используемых в работах	Характеристическая	10	-
Типы опор	Список с информацией об опорах, используемых на работах	Характеристическая	10	-
Бригады	Список бригад	Характеристическая	10	Ежедневно
Рабочие	Список сотрудников	Характеристическая	100	Ежедневно
Клиенты	Список с информацией о клиентах.	Характеристическая	1000	Ежедневно
Типы работ	Список с информацией о работах, производимых на объекте	Характеристическая	10	-

Отношения между сущностями базы данных и их пояснения, показаны в таблице 3.

Таблица 3 – Типы связей между типами сущностей предметной области

Тип сущности 1	Тип сущности 2	Описание	Мощность	Обязательность	Ограничения
Типы опор	Расчёт	При расчёте учитываются типы опор	1: M	Обязательная (типы опор учитываются при расчёте стоимости)	-
Типы местности	Расчёт	При расчёте учитываются типы местности	1: M	Необязательная (не все типы местности, обязательно учитываются при расчёте стоимости)	-

Продолжение таблицы 3

Тип сущности 1	Тип сущности 2	Описание	Мощность	Обязательность	Ограничения
Типы проводов	Расчёт	При расчёте учитываются типы проводов	1: M	Необязательная (не все типы проводов учитываются при расчёте стоимости)	-
Типы сечения	Расчёт	При расчёте учитываются типы сечения провода	1: M	Обязательная (типы сечения учитываются при расчёте стоимости)	-
Бригады	Рабочие	Бригады состоят из групп рабочих	1: M	Обязательная (в бригаде должны быть рабочие)	-
Бригады	Расчёт	Бригады учитываются при расчётах	1: M	Необязательная (не все бригады, обязательно находятся на работах)	-
Типы работ	Расчёт	При проведении расчёта должен учитываться тип проводимых работ	1: M	Обязательная (типы работ учитываются при расчёте стоимости)	-
Клиенты	Расчёт	Клиенты закрепляются в расчётах по проведению работ	1: M	Необязательная (не все клиенты, обязательно закрепляются в расчётах)	-

В каждой сущности выделены атрибуты. Описание сущности «Расчёт» и ее атрибутов представлены в таблице 4.

Таблица 4 – Атрибуты для сущности «Расчёт»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id_Заказа	Номер заказа	Целый	4	0001-9999	Нет	PK	1234
Тип расчёта	Тип производимого расчёта	Символьный	10	«А-я»	Нет	-	Вторичный
Дата	Дата проведения расчёта	Дата	8	-	Нет	-	15.02.2026
Длина провода	Длина провода, используемая в расчётах	Целый	10	00001-99999	Нет	-	1000

Продолжение таблицы 4

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Цена	Цена работы	Дробный	18	0000000001-999999999	Нет	-	367000,00
Описание	Описание расчёта	Символьный	100	«А-я»	Да	-	Проведение работ на участке
Id_Типа провода	Номер типа провода	Целый	4	0001-9999	Да	FK	4321
Id_Типа местности	Номер типа местности	Целый	4	0001-9999	Да	FK	1234
Id_Типа работ	Номер типа работ	Целый	4	0001-9999	Нет	FK	3234
Id_Типа опор	Номер типа опор	Целый	4	0001-9999	Да	FK	2234
Id_Типа сечения	Номер типа сечения	Целый	4	0001-9999	Нет	FK	1123
Id_Клиента	Номер клиента	Целый	4	0001-9999	Да	FK	1123
Id_Бригады	Номер Бригады	Целый	4	0001-9999	Да	FK	1123

Описание сущности «Типы опор» и её атрибутов, и свойств представлены в таблице 5.

Таблица 5 – Атрибуты для сущности «Типы опор»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id_Типа опор	Номер типа опор	Целый	4	0001-9999	Нет	PK	2123
Название	Название типа опор	Символьный	60	«А-я»	Нет	-	Железобетонные
Описание опор	Детальное описание опор	Символьный	200	«А-я»	Нет	-	Опора из железобетонна
Стоимость работы	Стоимость закупки и установки опоры	Дробный	10	0000000001-999999999	Нет	-	565230

Описание сущности «Типы местности» и её атрибутов, и свойств представлены в таблице 6.

Таблица 6 – Атрибуты для сущности «Типы местности»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id_Типы местности	Номер типа местности	Целый	4	0001-9999	Нет	РК	2432
Название	Название типа местности	Символьный	60	«А-я»	Нет	-	Равнина
Описание местности	Детальное описание типа местности	Символьный	200	«А-я»	Нет	-	Сухой, свободный ландшафт
Стоимость работ на местности	Стоимость прокладки кабеля на данной местности	Дробный	10	0000000001-999999999	Нет	-	875230,00

Описание сущности «Типы проводов» и её атрибутов, и свойств представлены в таблице 7.

Таблица 7 – Атрибуты для сущности «Типы проводов»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id_Типы проводов	Номер типа провода	Целый	4	0001-9999	Нет	РК	2435
Название	Название типа провода	Символьный	60	«А-я»	Нет	-	Изолированный
Описание провода	Детальное описание типа провода	Символьный	200	«А-я»	Нет	-	Провод с изоляцией
Стоимость провода	Стоимость закупки и установки провода за метр	Дробный	10	0000000001-999999999	Нет	-	325230,00

Описание сущности «Типы сечения» и её атрибутов, и свойств представлены в таблице 8.

Таблица 8 – Атрибуты для сущности «Типы сечения»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id_Типы сечения	Номер типа сечения	Целый	4	0001-9999	Нет	РК	2324
Название	Сечение провода	Символьный	60	«А-я»	Нет	-	13 мм
Описание сечения	Детальное описание типа сечения	Символьный	200	«А-я»	Нет	-	Сечение 13 мм
Стоимость провода	Стоимость закупки и установки провода на метр, с таким-то сечением	Дробный	10	0000000001-9999999999	Нет	-	215230,00

Описание сущности «Бригады» и её атрибутов, и свойств представлены в таблице 9.

Таблица 9 – Атрибуты для сущности «Бригады»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id_Бригады	Номер бригады	Целый	4	0001-9999	Нет	РК	2345
Название	Название бригады	Символьный	60	«А-я»	Нет	-	Бригада «№9»
Специализация	Деятельность, на которой специализируется данная бригада	Символьный	60	«А-я»	Нет	-	Прокладка кабеля
Кол-во работников	Кол-во работников в бригаде	Целый	4	0001-9999	Нет	-	3342

Описание сущности «Рабочие» и её атрибутов, и свойств представлены в таблице 10.

Таблица 10 – Атрибуты для сущности «Рабочие»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id_Рабочего	Номер рабочего	Целый	4	0001-9999	Нет	РК	2234
ФИО	ФИО рабочего	Символьный	60	«А-я»	Нет	-	Иванов Иван Иванович

Продолжение таблицы 10

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Должность	Специализация рабочего	Символьный	60	«А-я»	Нет	-	Прокладчик кабеля
Id_Бригады	Номер бригады	Целый	4	0001-9999	Нет	FK	2436

Описание сущности «Клиенты» и её атрибутов, и свойств представлены в таблице 11.

Таблица 11 – Атрибуты для сущности «Клиенты»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id_Клиента	Номер клиента	Целый	4	0001-9999	Нет	PK	2346
ФИО	ФИО клиента	Символьный	60	«А-я»	Нет	-	Петров Петр Петрович
Номер телефона	Номер телефона клиента	Целый	11	000000000001-999999999999	Да	-	89273649428
Электронная почта	Электронная почта клиента	Символьный	60	«А-я»	Да	-	veryasov.deniska@mail.ru
Дата создания	Дата, когда клиент был добавлен в базу данных	Дата	8	-	Нет	-	15.02.2026
Скидка	Скидка, представляемая клиенту во время расчётов	Дробный	5	0-100	Да	-	13,74
Пароль	Пароль клиента, для входа в учётную запись	Символьный	30	«А-я»	Да	-	Client_36
Роль	Роль пользователя, определяющая его права в управлении приложением	Символьный	10	«А-я»	Нет	-	User

Описание сущности «Типы работ» и её атрибутов, и свойств представлены в таблице 12.

Таблица 12 – Атрибуты для сущности «Типы работ»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id_Типы работ	Номер типа работы	Целый	4	0001-9999	Нет	PK	2423
Название	Название типа работ	Символьный	600	«А-я»	Нет	-	Трансформатор
Описание работ	Детальное описание типа работ	Символьный	200	«А-я»	Нет	-	Работы с трансформатором
Стоимость типа работ	Стоимость прокладки кабеля по этому типу работ	дробный	10	00000000 01- 99999999 9	Нет	-	215230,00

По завершении этапа построения концептуальной модели была разработана диаграмма, представленная на рисунке 4, которая служит основой для перехода к логическому и физическому проектированию.

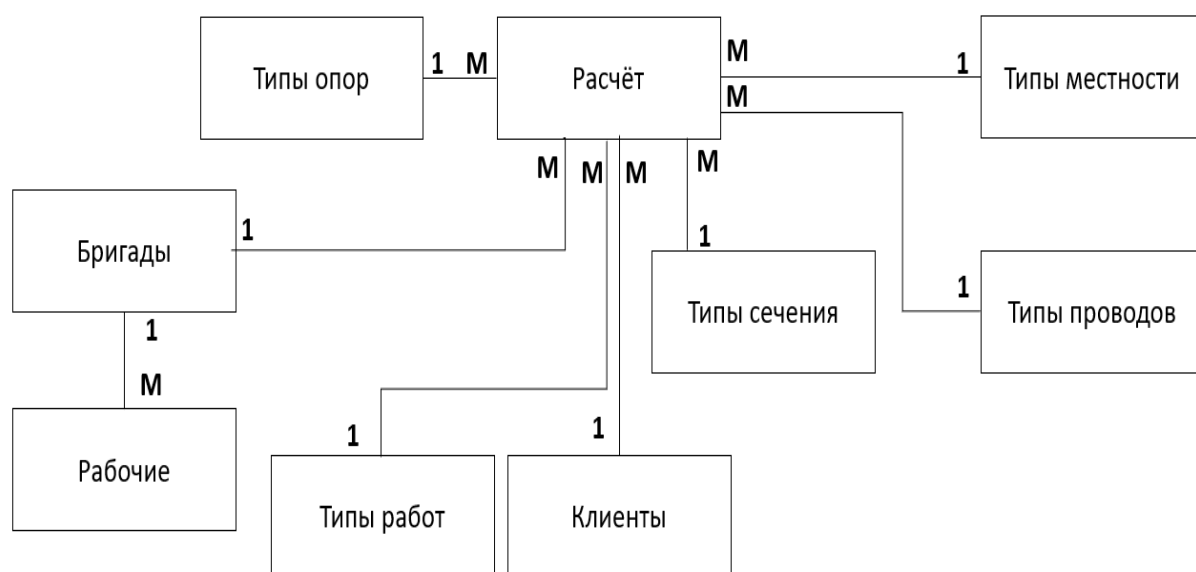


Рисунок 4 – Диаграмма «сущность-связь» предметной области

Прямоугольники на диаграмме отображают основные сущности, которые, линии обозначают их отношения. Центральное место в модели предметной области занимает сущность «Расчёт», обеспечивая удобство в процессе расчёта электромонтажных работ.

1.5 Анализ функциональных требований

В данном приложении следует организовать следующие возможности:

- сделать первичный расчёт стоимости работ;
- сделать подробный расчёт стоимости работ;
- посмотреть историю расчётов;

Диаграмма вариантов использования (use-case diagram), предназначена для описания, функционала разрабатываемой системы пользователю [1]. Диаграмма вариантов использования, соответствующая данным возможностям представлена на рисунке 5.

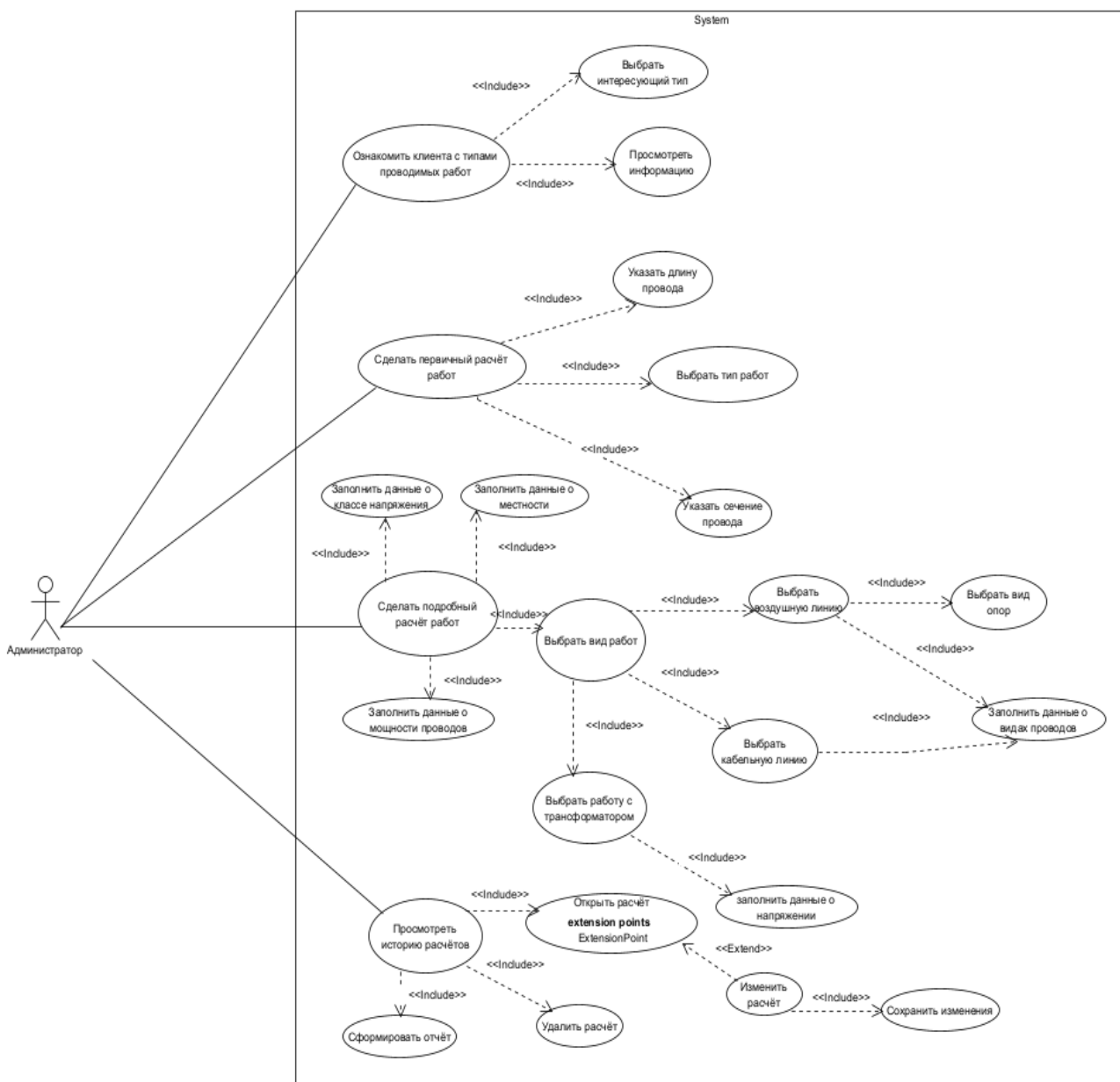


Рисунок 5 – Диаграмма вариантов использования приложения

Далее представлены некоторые спецификации, детализирующие варианты использования разрабатываемого приложения.

Спецификация варианта использования «Сделать первичный расчёт» приведена на рисунке 6.

Наименование: сделать первичный расчёт
ID: 1
Краткое описание: администратор делает первичный расчёт стоимости работ
Действующие лица: администратор
Предусловие: администратор вводит данные для первичного расчёта
Основной поток: <ol style="list-style-type: none"> 1. Администратор вводит основные данные для расчёта работ. 2. Администратор запускает первичный расчёт.
Постусловие: администратор заполняет данные для первичного расчёта стоимости работ.

Рисунок 6 – Спецификация варианта использования «Сделать первичный расчёт»

Спецификация варианта использования «Сделать подробный расчёт» приведена на рисунке 7.

Наименование: сделать подробный расчёт
ID: 2
Краткое описание: администратор делает подробный расчёт стоимости работ
Действующие лица: администратор
Предусловие: администратор вводит данные для подробного расчёта
Основной поток: <ol style="list-style-type: none"> 1. Администратор вводит подробные данные для расчёта работ. 2. Администратор запускает подробный расчёт
Постусловие: администратор заполняет данные для подробного расчёта стоимости работ.

Рисунок 7 – Спецификация варианта использования «Сделать подробный расчёт»

Спецификация варианта использования «Просмотреть историю расчётов» приведена на рисунке 8.

Наименование: просмотреть историю расчётов
ID: 3
Краткое описание: администратор открывает список истории расчётов
Действующие лица: администратор
Предусловие: администратор провёл расчёты
Основной поток: <ol style="list-style-type: none"> 1. Администратор нажимает на вкладку истории расчётов. 2. Администратор переходит на вкладку истории расчётов.
Постусловие: администратор видит списки проведённых расчётов.

Рисунок 8 – Спецификация варианта использования «Просмотреть историю расчётов»

В ходе анализа функциональных требований разрабатываемого приложения были выделены основные сценарии взаимодействия с системой. Для наиболее важных сценариев были приведены спецификации которые раскрывают их подробнее.

1.6 Анализ нефункциональных требований

Нефункциональные требования определяют качество и характеристики работы системы, не связанные напрямую с конкретными функциями. Для программных средств информационной поддержки планирования и организации работ в электроэнергетике можно выделить следующие нефункциональные требования: производительность, безопасность, удобство использования, модульность и расширяемость [2].

Производительность: система должна обеспечить быструю реакцию на действия администратора, включая загрузку страниц, произведения расчётов, отображение списка расчётов и управления данными.

Безопасность: система должна обеспечить надежное хранение личных данных о клиентах и работниках.

Удобство использования: приложение должно обладать простым и интуитивно понятным интерфейсом, чтобы новый администратор без подготовки мог легко им пользоваться. Навигация по системе должна быть интуитивно понятной, чтобы администраторы быстро находили нужную информацию и функции.

Модульность и расширяемость: архитектура системы должна позволять легко добавлять новые функции и модули без изменения уже реализованных функций. Логика работы с базой данных должна быть инкапсулирована, чтобы изменения структуры БД не влияли на остальную часть системы.

2 Проектирование приложения для организации и поддержки работ в электроэнергетике.

2.1 Архитектура приложения

Для данного приложения будет использоваться многослойная монолитная архитектура [3]. Слои архитектуры:

- UI (пользовательский интерфейс);
- бизнес-логика;
- взаимодействие с БД;
- БД.

В каждом слое будут выполняться задачи для своей области. Они будут независимы, что позволяет изменять один слой без влияния на работу остальных. Физическое представление архитектуры представлено на рисунке 9.

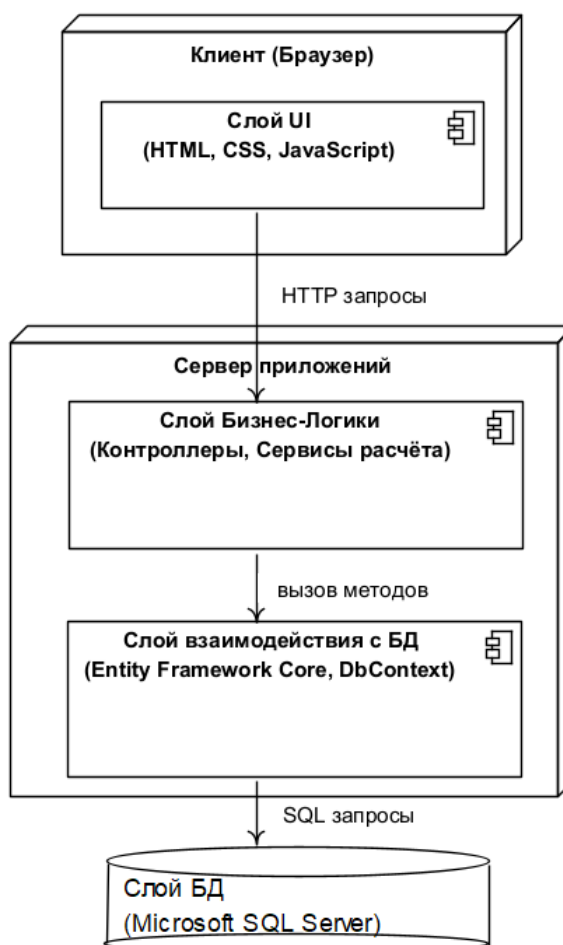


Рисунок 9 – Физическое представление архитектуры

Слой UI (пользовательский интерфейс) обеспечивает взаимодействие с пользователем. Он обрабатывает клиентские запросы, формирует HTML-страницы, выполняет первичную валидацию вводимых данных, а также передаются запросы в слой бизнес-логики. Слой содержит контроллеры для аутентификации и регистрации пользователей (AccountController), для выполнения расчётов (CalculationController) и для административного управления справочными данными (AdminController).

Слой бизнес-логики обеспечивает выполнение основных алгоритмов предметной области. К ним относятся: расчёт стоимости работ по первичной и детальной формулам, применение скидок клиентов, учёт коэффициентов сложности. Функционал этого слоя инкапсулирован в сервисе CalculationService, который реализует интерфейс ICalculationService. Здесь также могут выполняться вспомогательные операции, такие как формирование отчётов, но в текущей реализации они вынесены в контроллеры.

Внутри слоя взаимодействия с БД происходит инкапсуляция логики доступа к данным и управление сущностями. Классы этого слоя инициализируют операции с базой данных: чтение, создание, обновление и удаление. Основным компонентом является ApplicationDbContext, производный от DbContext Entity Framework Core. Он предоставляет наборы сущностей (DbSet<T>), а также настраивает связи и начальные данные. Доступ к данным осуществляется как из сервисов, так и напрямую из контроллеров (для административных операций), что обеспечивает гибкость при сохранении целостности.

Слой БД хранит все данные о расчётах, клиентах, бригадах, рабочих, типах работ, местности, проводах, сечениях и опорах [4, 5]. Это внешняя реляционная база данных (Microsoft SQL Server), которая физически размещена на сервере. Структура базы данных создаётся автоматически с помощью миграций Entity Framework Core, что обеспечивает согласованность схемы с моделями приложения [6].

2.2 Проектирование структур данных

2.2.1 Концептуальная модель базы данных

Концептуальная модель – это упрощённая схема предметной области, показывающая, какие объекты существуют, каковы их атрибуты и как они связаны между собой [7]. Описание сущностей предметной области представлено в таблице 13.

Таблица 13 – Сущности БД

Название сущности	Идентификатор сущности	Назначение сущности
Расчёт	Calculations	Содержит сведения о расчётах
Типы опор	SupportTypes	Содержит сведения о типах опор
Типы местности	TerrainTypes	Содержит сведения о типах местности
Типы проводов	WireTypes	Содержит сведения о типах проводов
Типы работ	WorkTypes	Содержит сведения о типах работах
Типы сечения	SectionTypes	Содержит сведения о типах сечения
Бригады	Teams	Содержит сведения о бригадах
Рабочие	Workers	Содержит сведения о рабочих
Клиенты	Clients	Содержит сведения о клиентах

Графическое представление концептуальной модели базы данных представлено на рисунке 10.

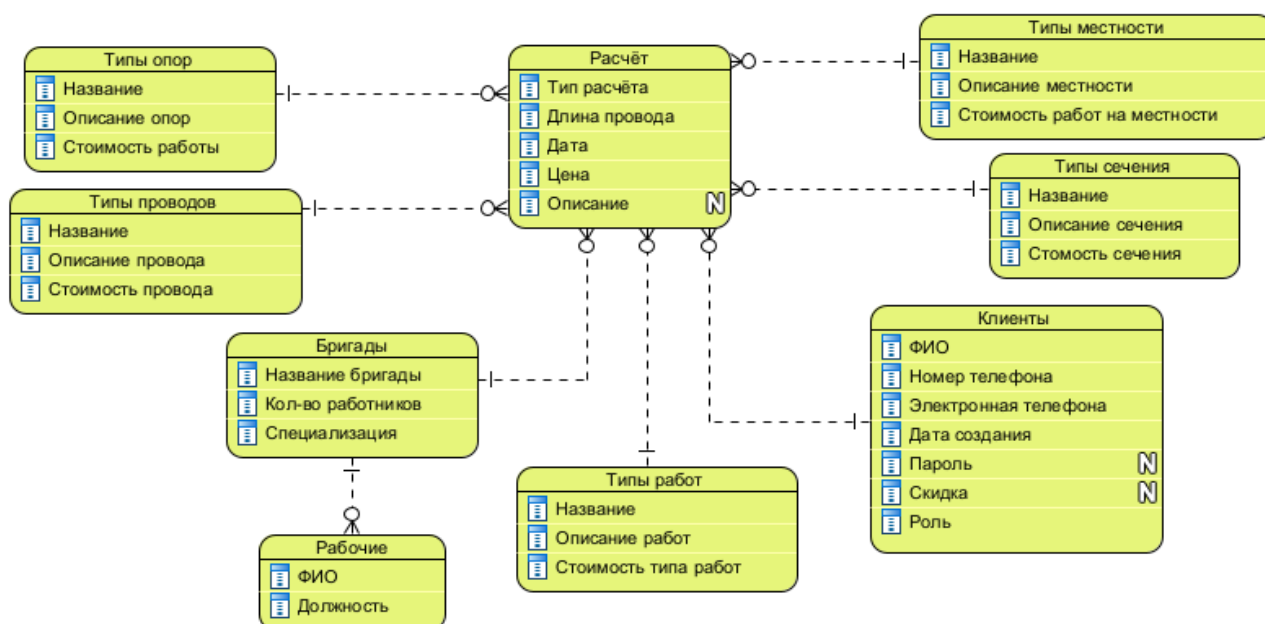


Рисунок 10 – Концептуальная модель базы данных

Таким образом была разработана концептуальная модель разрабатываемой базы данных приложения.

2.2.2 Логическая модель базы данных

Логическая модель данных представляет собой более детализированную и углублённую версию концептуальной модели. В ней уточняются атрибуты, сущности и отношения между ними. Графическое представление логической модели базы данных представлено на рисунке 11.

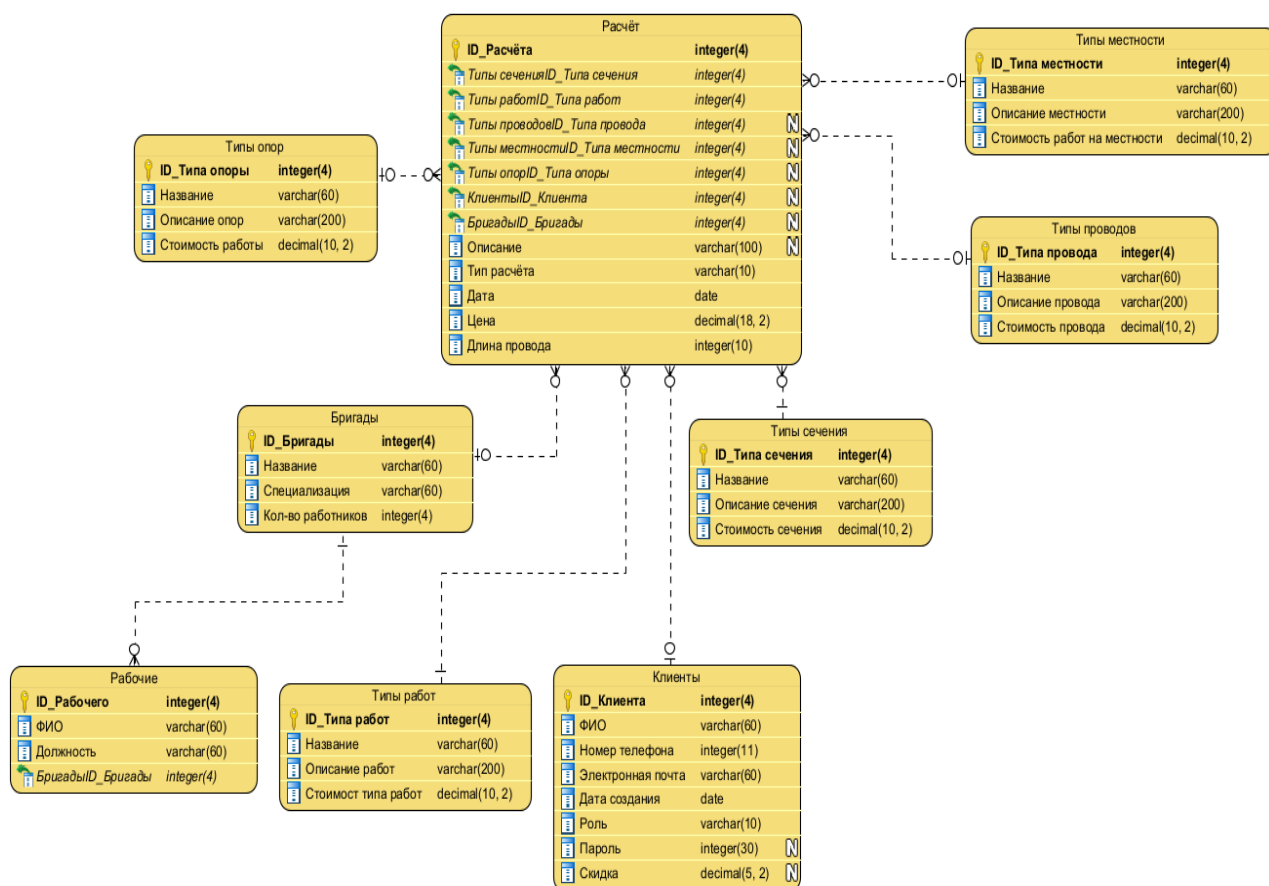


Рисунок 11 – Логическая модель базы данных

Таким образом, разработанные модели полностью отражают структуру предметной области и обеспечат основу для перехода к физической реализации.

2.3 Проектирование программных средств

На основании разработанной архитектуры была построена диаграмма классов. Диаграмма классов представлена на рисунке Е.1 в приложении Е.

Серверная часть системы реализуется с использованием фреймворка ASP.NET Core, который обеспечит модульную, кроссплатформенную и производительную среду для разработки веб-приложений [8]. Серверное приложение будет следовать многослойной архитектуре, разделяя логику на несколько уровней:

- слой контроллеров (Controllers): обрабатывает HTTP-запросы, вызывает соответствующие методы бизнес-логики и возвращает результаты на клиентскую часть;

- слой бизнес-логики (Services): содержит основную логику информационной системы, включая обработку данных и выполнение бизнес-процессов;

- слой интерфейсов (Interfaces): содержит интерфейсы для сервисов, что позволит обеспечить абстракцию, гибкость и облегчить тестирование и расширяемость системы;

- слой доступа к данным (Data Access): взаимодействует с базой данных через Entity Framework Core (DbContext), предоставляя методы для чтения, записи, обновления и удаления данных;

- слой моделей (Models): определяет структуры данных, используемых в системе.

Краткая спецификация контроллеров представлена в таблице 14.

Таблица 14 – Контроллеры

Название класса	Назначение
AccountController	Обрабатывает запросы на регистрацию, аутентификацию и выход пользователей из системы
CalculationController	Управляет процессом расчёта стоимости работ: первичный и детальный расчёт, история расчётов, отображение результатов
AdminController	Обеспечивает административное управление справочными данными: типы работ, типы местности, типы проводов, типы сечения, типы опор, клиенты, бригады, рабочие

Каждый контроллер взаимодействует с сервисом, который изолирует бизнес-логику. Сервисы обрабатывают данные, выполняют проверки и обеспечивают выполнение бизнес-процессов приложения. Спецификация сервиса приведена в таблице 15.

Таблица 15 – Сервис

Название класса	Назначение
CalculationService	Реализует бизнес-логику расчёта стоимости работ: первичный расчёт (на основе типа работ и сечения), детальный расчёт (с учётом всех параметров, скидок клиентов и коэффициентов сложности), сохранение результатов в истории

Сервис имеет свой интерфейс, который определяет методы для взаимодействия с другими компонентами системы. Описание интерфейса сервиса представлено в таблице 16.

Таблица 16 – Интерфейс сервиса

Название интерфейса	Назначение
ICalculationService	Определяет методы для выполнения первичного и детального расчёта, получения истории расчётов и получения расчёта по идентификатору

Слой доступа к данным реализован с использованием Entity Framework Core. В качестве основного компонента выступает класс ApplicationDbContext, который предоставляет наборы сущностей (DbSet<T>) для выполнения операций чтения, записи, обновления и удаления. Перечень основных наборов данных приведён в таблице 17.

Таблица 17 – Наборы данных

Название набора	Назначение
Calculations	Хранит результаты всех выполненных расчётов
Clients	Содержит данные о клиентах (физических лицах) и администраторах
Teams	Управляет информацией о бригадах исполнителей
Workers	Хранит данные о работниках, входящих в бригады
WorkTypes	Содержит справочную информацию о типах работ (воздушная линия, кабельная линия и т. д.)
TerrainTypes	Определяет типы местности и соответствующие коэффициенты стоимости
WireTypes	Содержит информацию о типах проводов (изолированный, неизолированный) и их стоимости за метр
SectionTypes	Хранит данные о сечениях проводов и дополнительной стоимости, связанной с ними
SupportTypes	Содержит данные о типах опор (деревянные, железобетонные, металлические) и стоимости их установки

Для каждого набора данных не требуется отдельного интерфейса репозитория, так как Entity Framework Core предоставляет универсальные

методы работы с сущностями через DbContext. Это упрощает архитектуру и снижает количество кода, сохраняя при этом гибкость и тестируемость.

Диаграммы классов, сопровождающие данное описание, наглядно иллюстрируют структурированную архитектуру системы.

2.4 Проектирование пользовательского интерфейса

Пользовательский интерфейс разрабатываемого приложения представляет собой набор связанных страниц с единой системой навигации реализована логика перехода между страницами с сохранением контекста и состояний через параметры маршрутов.

На главной странице пользователю предлагается выбор типа расчёта (первичный или детальный) и ссылка на историю расчётов. Для неавторизованных пользователей доступны «Вход» и «Регистрация», для авторизованных – приветствие и кнопка выхода. Взаимосвязь разделов страницы пользователя показана на рисунке 12.



Рисунок 12 – Переходы между разделами страницы пользователя

Страница первичного расчёта содержит форму с обязательными полями: тип работ, сечение провода, длина провода, а также необязательное поле «Примечания». После отправки формы происходит расчёт и переход на страницу результата.

Страница детального расчёта включает расширенный набор параметров: тип работ, тип провода, тип местности, сечение провода, тип опор, длина провода, выбор клиента, выбор бригады и примечания. Расчёт учитывает скидки клиентов и сложность работ.

Страница результата отображает итоговую стоимость, параметры расчёта и кнопки для печати, возврата к истории или на главную.

Страница истории расчётов предоставляет таблицу всех сохранённых расчётов с фильтрацией по типу расчёта и дате, а также возможностью экспорта и удаления записей.

Для администратора система предоставляет отдельную панель управления. После входа на главной странице становится доступной ссылка на админ-панель. Администратор может управлять справочными данными: типами работ, типами местности, типами проводов, типами сечения, типами опор, клиентами, бригадами и рабочими. Каждая сущность имеет собственную страницу с таблицей и модальными окнами для добавления и редактирования записей. Взаимосвязь разделов страницы администратора показана на рисунке 13.

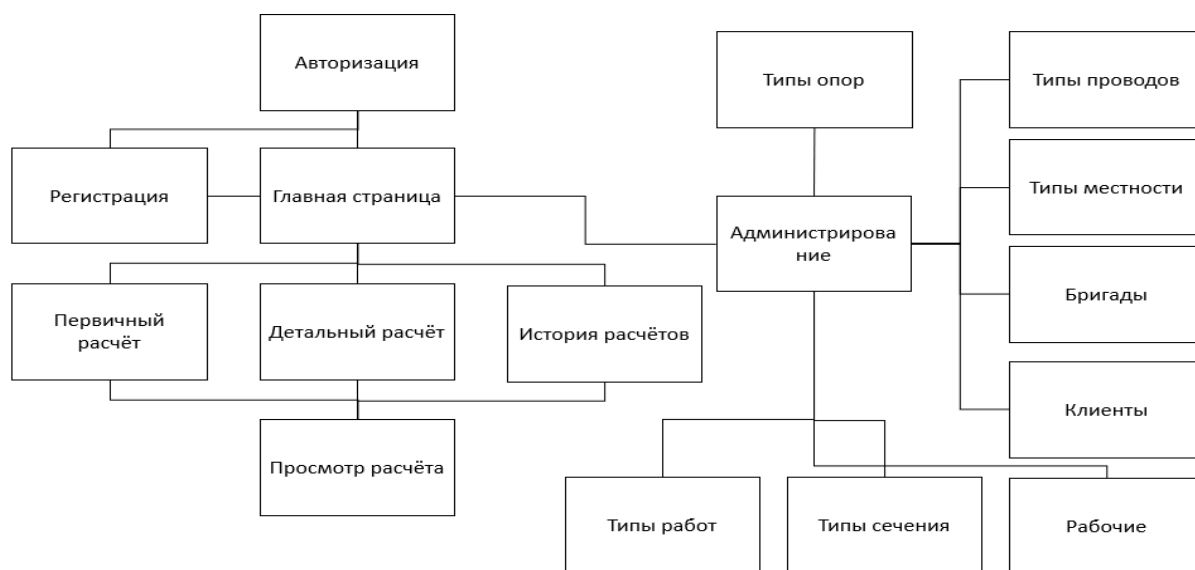


Рисунок 13 – Переходы между разделами страницы администратора

Для добавления или редактирования записи в любом справочнике используется модальное окно с формой, содержащей соответствующие поля (название, описание, стоимость и т. д.). После сохранения данные обновляются на текущей странице без перезагрузки всей страницы благодаря асинхронным запросам. Это обеспечивает удобство работы администратора и быстрый отклик интерфейса.

2.5 Выбор технологий и инструментальных средств разработки приложения

2.5.1 Выбор технологий разработки приложения

В проекте применена объектно-ориентированная парадигма (ООП), позволяющая разбить код на логические модули, что упрощает разработку. Кроме того, инкапсуляция закрывает внутреннюю реализацию и обеспечивает изоляцию компонентов. В итоге система остаётся гибкой для расширения и целостной при любых изменениях.

Организация процесса разработки строится на основе методологии Rational Unified Process (RUP). Данный подход ориентирован на коллективную работу, предполагает наличие единой базы знаний, согласованных взглядов на разработку и унифицированного языка моделирования. RUP является итеративной методологией, где весь цикл разбит на четыре фазы: старт, детализация, построение и передача. Каждая фаза включает несколько итераций. Это даёт структурированность, гибкость при правках и позволяет на каждом этапе получать работающий продукт.

В качестве платформы для серверной части выбран фреймворк ASP.NET Core. Он поддерживает MVC, внедрение зависимостей и встроенную аутентификацию. Использование ASP.NET Core позволяет строить масштабируемые и безопасные веб-приложения с чётким разделением слоёв.

Для реализации пользовательского интерфейса применяются технологии HTML, CSS и JavaScript. Клиентская часть строится на основе Razor-представлений с добавлением динамических элементов через нативный

JavaScript, – таких как модальные окна, предварительный расчёт и фильтрация. Такой подход обеспечивает быстрый отклик интерфейса и удобство работы пользователя.

В качестве технологии доступа к данным выбран Entity Framework Core – объектно-ориентированное решение, входящее в состав платформы .NET. EF Core позволяет работать с данными через модель предметной области, абстрагируясь от конкретной СУБД. Разработчик оперирует сущностями, а EF Core автоматически формирует SQL-запросы и выполняет отображение на реляционную базу данных. Это ускоряет разработку, упрощает сопровождение кода и обеспечивает независимость от диалекта SQL конкретной СУБД.

2.5.2 Выбор инструментальных средств разработки приложения

Для веб-приложения с серверной частью и базой данных использовалась среда Visual Studio 2026. Данная среда предоставляет широкие возможности для написания кода на C#, отладки, профилирования, управления пакетами NuGet, а также встроенную поддержку Entity Framework Core и системы контроля версий Git.

Для управления базой данных выбрана система управления базами данных Microsoft SQL Server. SQL Server обладает высокой производительностью, масштабируемостью и надёжностью, что критически важно для хранения и обработки данных о расчётах, клиентах, бригадах и справочниках. Поддержка Transact-SQL и интеграция с Entity Framework Core позволяют эффективно разрабатывать и сопровождать схему данных.

В качестве вспомогательного средства для моделирования базы данных и документирования архитектуры использовался инструмент Visual Paradigm. С его помощью была построена диаграмма физического представления архитектуры, отражающая структуру системы. Visual Paradigm позволяет создавать наглядные графические модели, что очень удобно использовать в проектировании приложения.

В качестве языка программирования выбран C# – объектно-ориентированный язык платформы .NET, который позволяет реализовать все принципы ООП, обладает богатой стандартной библиотекой и тесной интеграцией с ASP.NET Core и Entity Framework Core. Выбор C# обусловлен также его надёжностью, производительностью и широкими возможностями для создания веб-приложений.

3 Разработка программного обеспечения

3.1. Разработка физической модели базы данных

Опираясь на разработанные ранее концептуальную и логическую модели базы данных, была построена физическая модель представления данных [9].

Физическая модель базы данных – это точное отображение базы данных, которое определяет её реализацию для конкретной СУБД и организацию файлов. Диаграмма физической модели представлена на рисунке 14.

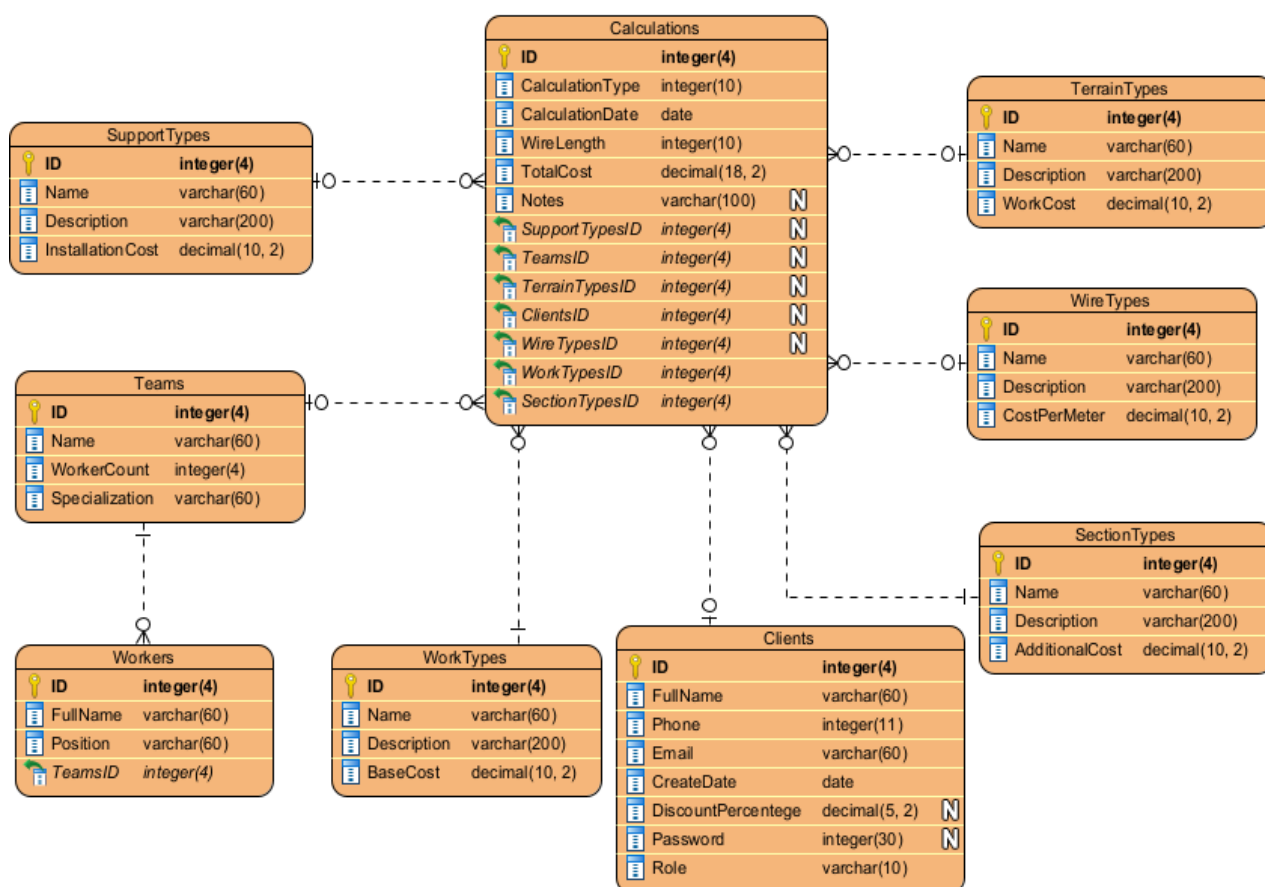


Рисунок 14 – Физическая модель базы данных

Таким образом была реализована база данных для поддержки планирования и организации работ в электроэнергетике, которая обеспечивает хранение, обработку и управление данными, необходимыми для функционирования системы.

3.2. Реализация серверной части информационной системы

Серверная часть системы для поддержки планирования и организации работ в электроэнергетике разрабатывалась в среде разработки Microsoft Visual Studio 2026, на языке программирования C#. Фрагменты кода серверной части приведены в приложении Б.

Серверная часть представляет собой многослойную монолитную архитектуру. Присутствуют классы моделей данных, отображающие сущности из базы данных. Контроллеры обрабатывают запросы от клиентской части и возвращают ответ. Сервисы содержат функционал и обеспечивают работу между контроллерами и базой данных.

Серверная часть взаимодействует с клиентской через REST API, реализованный в каждом модуле. Методы API поддерживают стандартные HTTP-операции (GET, POST, PUT, DELETE) и обеспечивают полный цикл работы с сущностями системы.

Реализованные методы для модуля работы с аккаунтами описаны в таблице 18.

Таблица 18 – Методы API модуля «Аккаунты»

HTTP метод	URL запроса	Описание
GET	/Account/Register	Отображение формы регистрации
POST	/Account/Register	Регистрация нового клиента
GET	/Account/Login	Отображение формы входа
POST	/Account/Login	Аутентификация клиента
POST	/Account/Logout	Выход из системы

Реализованные методы для модуля работы со справочной информации: типы работ, проводов, местности, сечения, опор, а также управления клиентами, бригадами и рабочими, доступные для пользователя с ролью администратора описаны в таблицах 19-26.

Таблица 19 – Методы API модуля «Типы местности»

HTTP метод	URL запроса	Описание
GET	/Admin/GetTerrainTypes	Получение всех типов местности
POST	/Admin/SaveTerrainType	Добавление или обновление типа местности

Продолжение таблицы 19

HTTP метод	URL запроса	Описание
DELETE	/Admin/DeleteTerrainType?id={id}	Удаление типа местности

Таблица 20 – Методы API модуля «Типы сечения»

HTTP метод	URL запроса	Описание
GET	/Admin/GetSectionTypes	Получение всех типов сечения
POST	/Admin/SaveSectionType	Добавление или обновление типа сечения
DELETE	/Admin/DeleteSectionType?id={id}	Удаление типа сечения

Таблица 21 – Методы API модуля «Типы опор»

HTTP метод	URL запроса	Описание
GET	/Admin/GetSupportTypes	Получение всех типов опор
POST	/Admin/SaveSupportType	Добавление или обновление типа опоры
DELETE	/Admin/DeleteSupportType?id={id}	Удаление типа опоры

Таблица 22 – Методы API модуля «Типы работ»

HTTP метод	URL запроса	Описание
GET	/Admin/GetWorkTypes	Получение всех типов работ
POST	/Admin/SaveWorkType	Добавление или обновление типа работ
DELETE	/Admin/DeleteWorkType?id={id}	Удаление типа работ

Таблица 23 – Методы API модуля «Клиенты»

HTTP метод	URL запроса	Описание
GET	/Admin/GetClients	Получение списка клиентов
POST	/Admin/SaveClient	Добавление или обновление клиента (ФИО, телефон, email, скидка)
DELETE	/Admin/DeleteClient?id={id}	Удаление клиента (с проверкой использования)

Таблица 24 – Методы API модуля «Бригады»

HTTP метод	URL запроса	Описание
GET	/Admin/GetTeams	Получение всех бригад
POST	/Admin/SaveTeam	Добавление или обновление бригады (название, количество рабочих, специализация)
DELETE	/Admin/DeleteTeam?id={id}	Удаление бригады (с проверкой участия в расчётах)
GET	/Admin/TeamWorkers?id={id}	Просмотр рабочих, закреплённых за бригадой

Таблица 25 – Методы API модуля «Рабочие»

HTTP метод	URL запроса	Описание
GET	/Admin/GetWorkers	Получение всех рабочих
POST	/Admin/SaveWorker	Добавление или обновление рабочего (ФИО, должность, бригада)
DELETE	/Admin/DeleteWorker?id={id}	Удаление рабочего

Реализованные методы для модуля работы с первичным и детальным расчётами, а также просмотр их истории и результатов описаны в таблице 26.

Таблица 26 – Методы API модуля «Расчёты»

HTTP метод	URL запроса	Описание
GET	/Calculation	Главная страница выбора типа расчёта
GET	/Calculation/Primary	Отображение формы первичного расчёта
POST	/Calculation/Primary	Выполнение первичного расчёта (только тип работ, сечение, длина)
GET	/Calculation/Detailed	Отображение формы детального расчёта
POST	/Calculation/Detailed	Выполнение детального расчёта (учитываются все параметры: тип провода, местность, опоры, клиент, бригада)
GET	/Calculation/History	Просмотр истории всех расчётов
GET	/Calculation/Result?id={id}	Детальный просмотр результата расчёта
DELETE	/Calculation/DeleteCalculation/{id}	Удаление расчёта из истории

Данная система обладает высокой степенью модульности несмотря на то, что она реализована в виде монолита. Это позволит ей в дальнейшем перейти к микросервисной архитектуре при необходимости, без значительных архитектурных изменения.

3.3. Реализация клиентской части информационной системы.

Клиентская часть системы отвечает за отображение пользовательского интерфейса, обработку действий пользователя и взаимодействие с серверной частью через REST API. Фрагменты кода клиентской части приведены в приложении В.

Диаграмма компонентов представлена на рисунке Е.2 в приложении Е. Описание компонентов клиентской части содержится в таблице 27.

Таблица 27 – Описание компонентов клиентской части

Компонент	Назначение
Calculation/Index.cshtml	Главная страница с выбором типа расчёта (первичный/детальный), информацией о системе, кнопками входа/регистрации и помощи.
Account/Login.cshtml	Страница входа в систему (email, пароль).
Account/Register.cshtml	Страница регистрации нового клиента (ФИО, телефон, email, пароль) с валидацией на стороне клиента.
Calculation/Primary.cshtml	Форма первичного расчёта (тип работ, сечение провода, длина провода, примечания). Включает динамическое отслеживание заполнения полей, предварительный расчёт, ограничение длины примечаний.
Calculation/Detailed.cshtml	Форма детального расчёта (все параметры: тип работ, провода, местности, сечения, опоры; выбор клиента и бригады; длина провода, примечания).
Calculation/History.cshtml	История выполненных расчётов с таблицей, фильтрацией по типу расчёта и дате, экспортом в CSV, статистикой.
Calculation/Result.cshtml	Страница детального просмотра результата расчёта с параметрами, итоговой стоимостью, возможностью печати.
Admin/Index.cshtml	Панель администратора: виджеты статистики (количество расчётов, клиентов, бригад, рабочих, типов работ, местности, проводов, сечений, опор). Быстрые ссылки на управление сущностями.
Admin/Clients.cshtml	Управление клиентами (CRUD) с модальным окном, AJAX-загрузкой, уведомлениями.
Admin/Teams.cshtml	Управление бригадами (название, количество рабочих, специализация). Редактирование и удаление.
Admin/Workers.cshtml	Управление рабочими (ФИО, должность, привязка к бригаде) с выпадающим списком бригад.
Admin/TeamWorkers.cshtml	Просмотр рабочих конкретной бригады.
Admin/WorkTypes.cshtml	Управление типами работ (название, описание, базовая стоимость).
Admin/TerrainTypes.cshtml	Управление типами местности (название, описание, стоимость работ).
Admin/WireTypes.cshtml	Управление типами проводов (название, описание, стоимость за метр).
Admin/SectionTypes.cshtml	Управление типами сечения (название, описание, дополнительная стоимость).
Admin/SupportTypes.cshtml	Управление типами опор (название, описание, стоимость установки)

Клиентская часть реализована с использованием C#HTML в среде Microsoft Visual Studio 2026, что позволяет комбинировать серверный код C# и клиентские технологии. Все страницы адаптированы для современных браузеров и поддерживают мобильное отображение.

3.4. Реализация интерфейса приложения

3.4.1 Реализация интерфейса клиента

В соответствии с заданием, для пользователя было спроектировано web-приложение, в котором будут доступны следующие страницы:

- страница авторизации;
- страница регистрации;
- главная страница;
- страница первичного расчёта;
- страница детального расчёта;
- страница истории расчёта;
- страница результата расчёта.

После запуска приложения пользователь попадает на страницу авторизации, представленной на рисунке 15.

The image shows a login form with the title "Вход в систему". Below the title are two input fields: "Email" and "Пароль". Below these fields is a blue button labeled "Войти". To the right of the "Войти" button is a blue link labeled "Зарегистрироваться". Three numbered arrows point to specific elements: arrow "1" points to both the "Email" and "Пароль" input fields; arrow "2" points to the "Войти" button; arrow "3" points to the "Зарегистрироваться" link.

Рисунок 15 – Вход в систему

Основные элементы страницы авторизации:

- 1 – поля для ввода данных;

2 – кнопка «Войти»;

3 – ссылка на страницу регистрации.

При взаимодействии с кнопкой «Войти», система проверяет наличие и корректность введённых данных. При корректном вводе данных пользователь попадает на главную страницу (рисунок 16), иначе система предлагает проверить данные на корректность или зарегистрироваться.

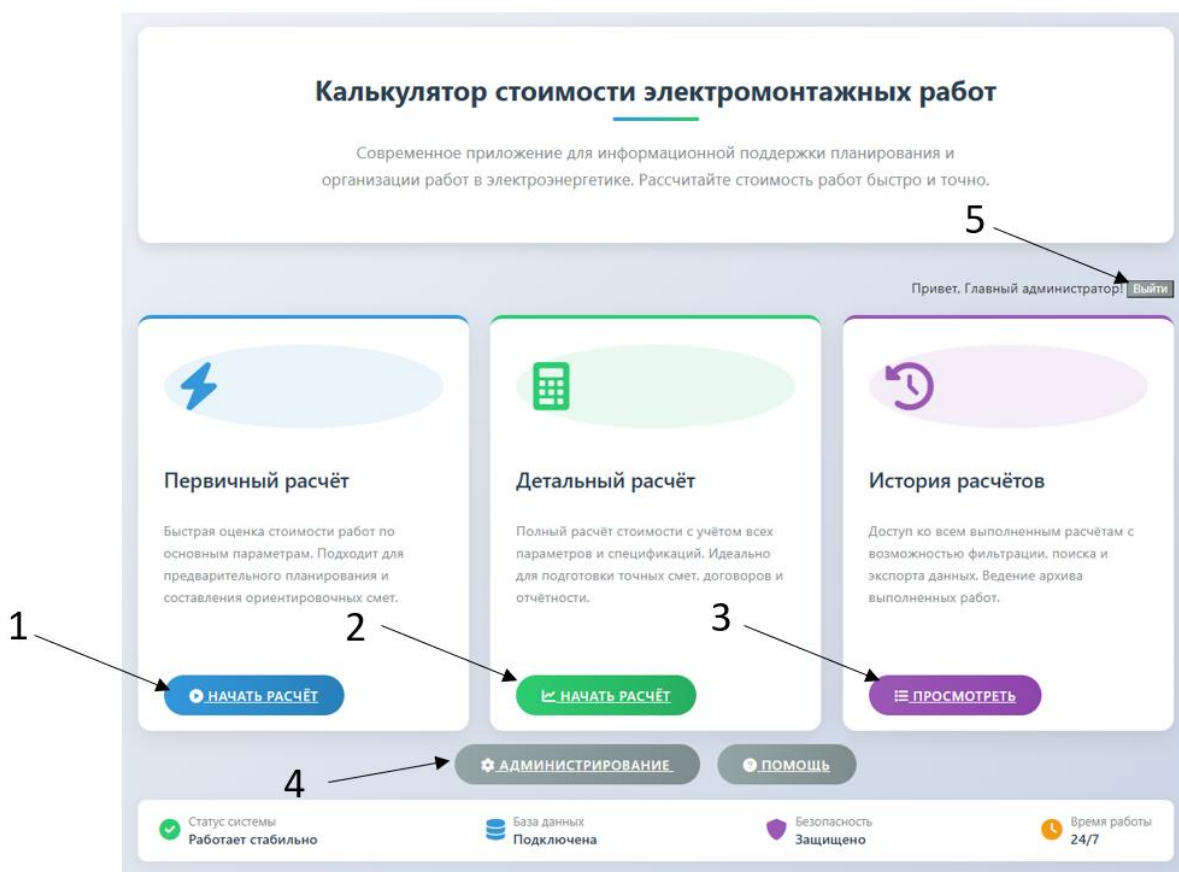


Рисунок 16 – Главная страница

Основные элементы главной страницы:

1 – вкладка «Первичный расчёт»;

2 – вкладка «Детальный расчёт»;

3 – вкладка «Истории расчётов»;

4 – кнопка «Администрирование»;

5 – кнопка «Выйти».

После входа на главную страницу пользователь может начать свой собственный расчёт, на основании имеющихся у него данных (первичный или детальный), а также посмотреть историю уже произведённых расчётов, или выйти из своего аккаунта. При входе пользователя под ролью «Администратор» на главной странице появляется кнопка «Администрирование».

При перемещении на страницу первичного расчёта (рисунок 17), пользователь может выбрать данные для расчёта стоимости нужных ему работ.

⚡ Первичный расчёт стоимости работ

Быстрый расчёт стоимости электромонтажных работ по основным параметрам. Заполните форму ниже для получения ориентировочной стоимости.

1 из 3 заполнено

Прогресс заполнения формы: 33%

Основные параметры расчёта

1

Тип работ * -- Выберите тип работ --

Сечение провода * -- Выберите сечение провода --

Большее сечение = выше стоимость материалов

Длина провода (метры) * 100 м

☐ Дополнительная информация

2

Примечания

Дополнительная информация о расчёте (необязательно)...

Максимум 500 символов 0/500

4

3

Рассчитать стоимость Назад

Рисунок 17 – Первичный расчёт

Основные элементы страницы первичного расчёта:

1 – поля для выбора данных при расчёте;

2 – поле для ввода примечания;

3 – кнопка «Назад»;

4 – кнопка «Рассчитать стоимость».

Аналогичная ситуация происходит при выборе пользователем вкладки «Детальный расчёт» (рисунок 18), за исключением того, что данный расчёт более точный и подробный.

Детальный расчёт стоимости работ

Тип работ *	Тип опор
-- Выберите тип работ --	-- Не выбрано --
Тип провода *	Длина провода (м) *
-- Выберите тип провода --	100
Тип местности *	Клиент
-- Выберите тип местности --	-- Не выбрано --
Сечение провода *	Бригада
-- Выберите сечение провода --	-- Не выбрано --

Примечания

Дополнительная информация...

Рассчитать стоимость
Назад

О детальном расчёте

Детальный расчёт учитывает все параметры для максимально точного определения стоимости работ. Заполните все необходимые поля для получения точной сметы.

Рисунок 18 – Детальный расчёт

Основные элементы страницы детального расчёта:

1 – поля для выбора данных при расчёте;

2 – поле для ввода примечания;

3 – кнопка «Назад»;

4 – кнопка «Рассчитать стоимость».

На странице истории расчётов (рисунок 19) пользователь может взаимодействовать с уже произведёнными расчётами, фильтруя или удаляя их, а также просматривая статистику расчётов, с помощью кнопки «Экспорт в CSV» он может скачать таблицу всех расчётов.

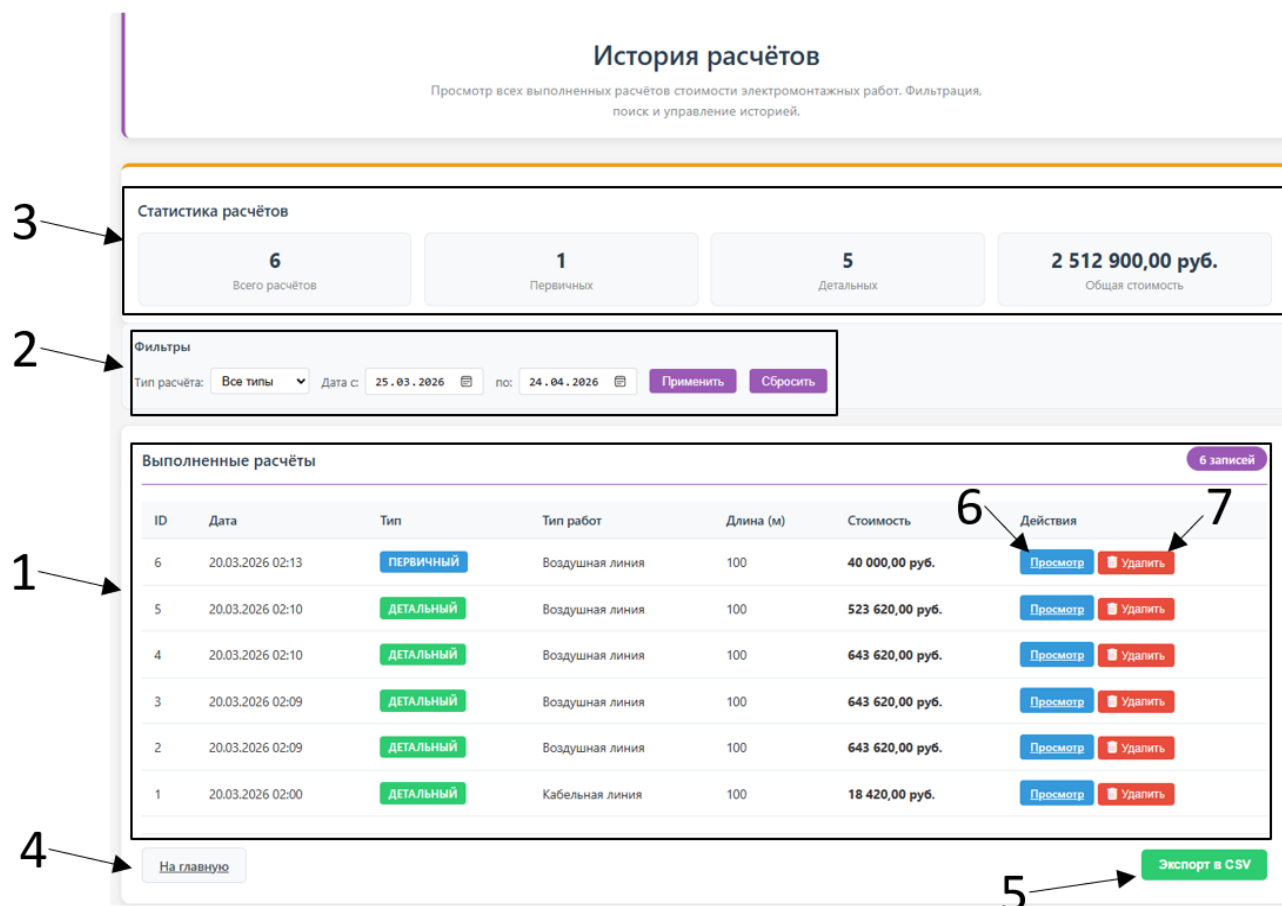


Рисунок 19 – История расчётов

Основные элементы страницы истории расчётов:

- 1 – таблица с произведёнными расчётами;
- 2 – фильтрация расчётов;
- 3 – статистика расчётов;
- 4 – кнопка «На главную»;
- 5 – кнопка «Экспорт в CSV».
- 6 – кнопка «Просмотр»;
- 7 – кнопка «Удалить».

При нажатии кнопки «Просмотр» пользователь переходит на страницу результата расчёта (рисунок 20).

The screenshot shows a web interface for the 'Result of calculation' page. At the top, the title 'Результат расчёта' is displayed, followed by a subtitle 'Детализация выполненного расчёта стоимости электромонтажных работ.' Below this, a section titled 'Расчёт стоимости работ' contains a large box showing the 'Итоговая стоимость работ' as '523 620,00 руб.' with a timestamp 'Детальный расчёт • 20.03.2026 02:10'. A table below lists the 'Параметры расчёта' in two columns. At the bottom, there are three buttons: 'На главную', 'К истории', and 'Печать'. Numbered annotations point to specific elements: '1' points to the table of parameters, '2' points to the 'Печать' button, '3' points to the 'На главную' button, and '4' points to the 'К истории' button.

Параметры расчёта	
Тип расчёта	Дата расчёта
Детальный	20.03.2026 02:10
Тип работ	Тип провода
Воздушная линия	Изолированный
Длина провода	Тип местности
100 м	Город
Сечение провода	Тип опор
16 мм ²	Деревянная
Клиент	Бригада
Иванов Иван Иванович	Бригада №1

Buttons: На главную, К истории, Печать

Рисунок 20 – Результат расчёта

Основные элементы страницы результата расчёта:

1 – таблица с параметрами расчёта;

- 2 – кнопка «Печать»;
- 3 – кнопка «На главную»;
- 4 – кнопка «К истории».

В просмотре результата расчёта, можно увидеть его подробную информацию. При нажатии кнопки «Печать», пользователь может моментально распечатать результат или сохранить его, экспортировав в нужный ему формат.

3.4.2 Реализация интерфейса администратора

При авторизации с ролью администратора, пользователю доступны все страницы клиента, а также у него появляется доступ к страницам, предназначенным для полного управления всеми справочными данными приложения доступным только администратору. Эти страницы нужны для полного управления всеми справочными данными приложения:

- панель администратора;
- страница управления клиентами;
- страница управления бригадами;
- страница управления рабочими;
- страница управления типами работ;
- страница управления типами местности;
- страница управления типами проводов;
- страница управления типами сечения;
- страница управления типами опор.

Открыв панель администратора (рисунок 21), администратор может просмотреть информацию о текущем состоянии базы данных и о количестве записей в ключевых таблицах, а также перейти во вкладки управления данными для их просмотра и редактирования. На данной странице у него есть полный контроль над всей справочной информацией приложения.

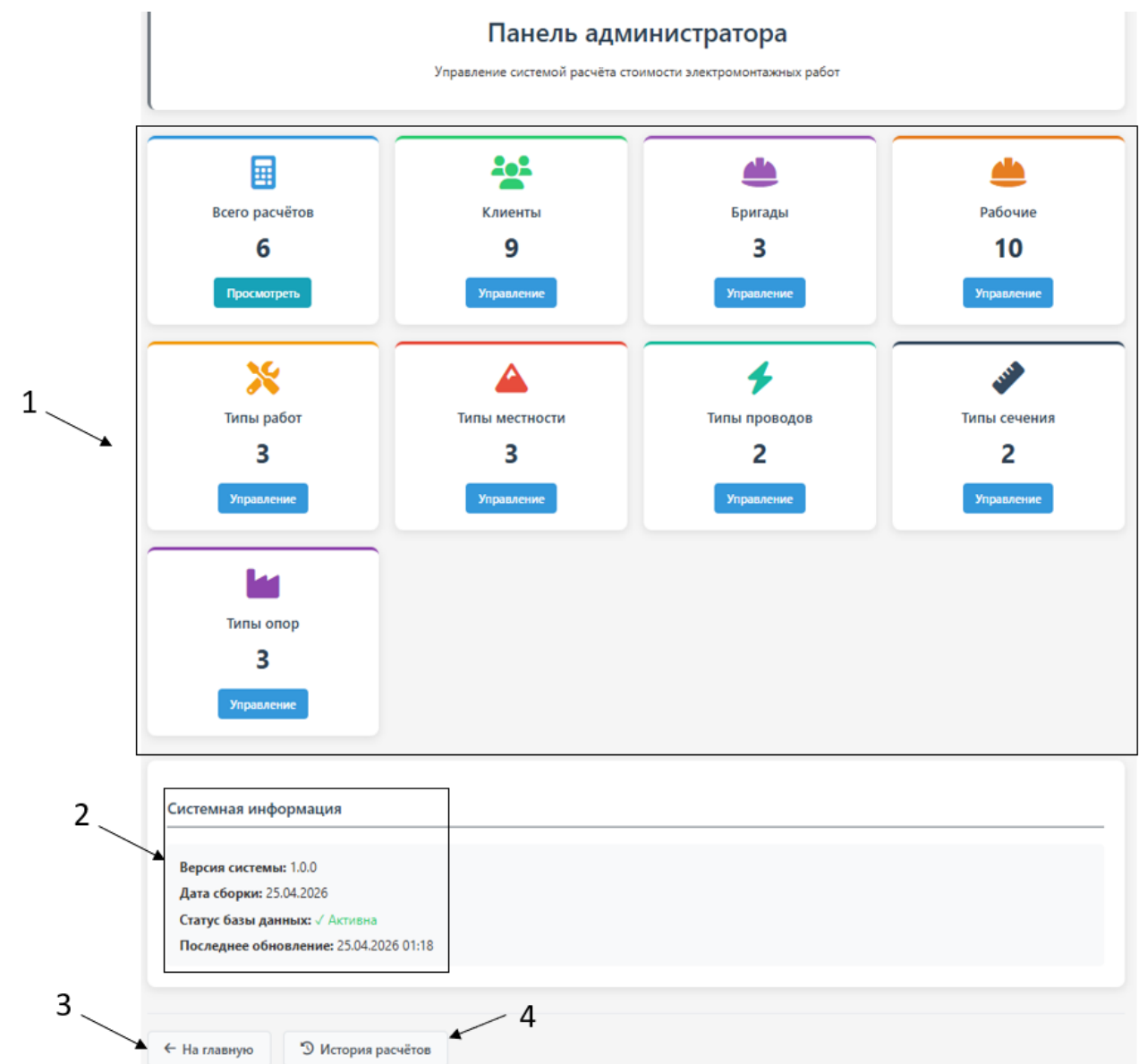


Рисунок 21 – Панель администратора

Основные элементы панели администратора:

- 1 – вкладки управления данными;
- 2 – панель системной информации;
- 3 – кнопка «На главную»;
- 4 – кнопка «История расчётов».

Во вкладке управлении клиентами (рисунок 22), администратор может изменять данные клиента: удаляя, добавляя или редактируя их.

Управление клиентами

[+ Добавить клиента](#)

[illegible]

Рисунок 22 – Вкладка управления клиентами

Основные элементы страницы управления клиентами:

- 1 – таблица с информацией о клиентах;
- 2 – кнопка «Назад в админ-панель»;
- 3 – кнопка «Редактировать»;
- 4 – кнопка «Удалить»;
- 5 – кнопка «Добавить клиента».

В окне редактирования (рисунок 23) администратор может не только просматривать, но и при необходимости изменять, а также добавлять выбранные данные. Любые внесённые им изменения в и новые записи будут сохраняться в базе данных.

The image shows a web application window titled "Редактировать клиента" (Edit Client) with a close button (X) in the top right corner. Below the title bar, there are four input fields, each with a label and an asterisk indicating a required field:

- ФИО ***: The input field contains the text "Иванов Иван Иванович".
- Телефон ***: The input field contains the text "+7 (123) 456-78-90".
- Email**: The input field contains the text "ivanov@example.com".
- Скидка, %**: The input field contains the text "Например: 10.5".

At the bottom of the window, there are two buttons:

- A light gray button labeled "Отмена" (Cancel).
- A green button labeled "Обновить" (Update).

Numbered arrows point to specific elements:

- Arrow **1** points to the four input fields.
- Arrow **2** points to the "Отмена" button.
- Arrow **3** points to the "Обновить" button.

Рисунок 23 – Окно редактирования

Основные элементы окна редактирования:

- 1 – поля с данными клиента;
- 2 – кнопка «Отмена»;
- 3 – кнопка «Обновить».

Остальные страницы управления справочными данными имеют похожий функционал. Изменение данных через страницы управления, упрощает работу со справочными данными, а также помогает отслеживать ошибки и предупреждения при работе с ними.

4 Отладка и тестирование.

4.1 Обоснование выбора режима тестирования.

Задача тестирования – проанализировать приложение с целью обнаружения ошибок и повышения качества [10, 11, 12]. Классификация тестирования позволяет организовать знания и способствует более быстрому планированию и разработке тест-кейсов. Существует множество видов и направлений тестирования. Наибольшей известностью пользуются следующие виды:

- функциональное тестирование;
- модульное тестирование;
- интеграционное тестирование;
- системное тестирование.

Результаты функционального тестирования показывают, соответствует ли функционал программного обеспечения заявленным требованиям. Такое тестирование включает в себя проверку интерфейсов, клиент-серверных приложений и общей работы программного обеспечения.

Модульное тестирование заключается в проверке работоспособности отдельных модулей или частей программного обеспечения. Такое тестирование помогает быстро исправить ошибки при добавлении нового функционала в приложение, при этом не тестируется программное обеспечение полностью.

Интеграционное тестирование же наоборот направлено на проверку работы нескольких частей программного обеспечения, их взаимодействия.

Системным называется тестирование полного программного продукта. Проверяется соответствие требованиям всего программного обеспечения. Такое тестирование проводится уже после завершения интеграционного.

Было принято решение о проведении функционального и модульного тестирования разработанного приложения, что позволит оценить работу приложения со стороны пользователя, а также оценить соответствие требованиям отдельных компонентов.

4.2 Тестирование приложения для планирования и организации работ в электроэнергетике.

4.2.1 Функциональное тестирование.

Было проведено функциональное тестирование методом черного ящика на примере выбора длины провода для расчёта. Анализируем несколько полей, поведение системы на основе технического задания. Выделяем причины и следствия событий. Записываем их в таблицу 28.

Таблица 28 – Причины и следствия

Причины	
П1	Длина провода корректна (1-10000)
П2	Выбран тип работ
П3	Выбран тип провода
П4	Выбран тип местности
П5	Выбрано сечение провода
Следствия	
С1	Сообщение: «Некорректная длина провода»
С2	Сообщение: «Не выбран тип работ»
С3	Сообщение: «Не выбран тип провода»
С4	Сообщение: «Не выбран тип местности»
С5	Сообщение: «Не выбрано сечение провода»
С6	«Расчёт успешно выполнен»

Используя таблицу 28, разрабатываем функциональную диаграмму рисунок 24.

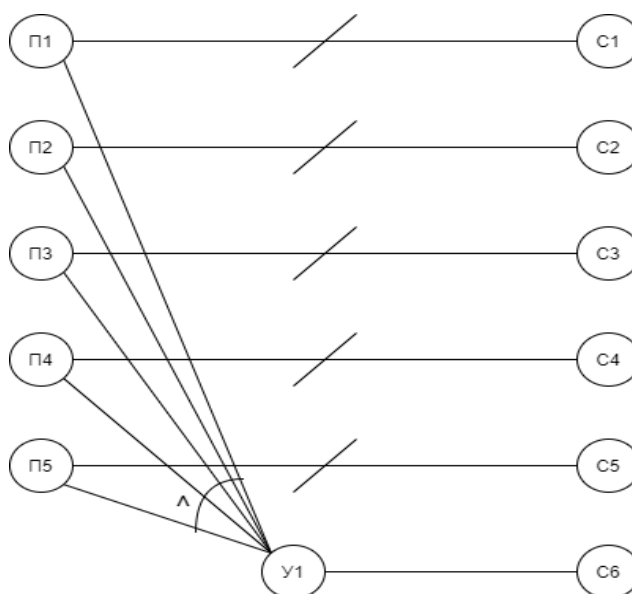


Рисунок 24 – Функциональная диаграмма

В соответствии с причинами и следствиями, была разработана таблица 29, таблица выделенных классов эквивалентности.

Таблица 29 – Классы эквивалентности

Длина провода	Тип работы	Тип провода	Тип местности	Сечение провода
1-10000	Кабельная линия	Изолированный	Леса	16
<1	Воздушная линия	Неизолированный	Луга	25
10000-∞	Трансформатор	-	Болота	-

Опираясь на выделенные классы эквивалентности, мы разработали таблицу 30, таблицу попарного тестирования.

Таблица 30 – Попарное тестирование

№	Длина провода	Тип работ	Тип провода	Тип местности	Сечение провода
1	1-10000	Трансформатор	изолированный	Болота	16
2	1-10000	Кабельная линия	неизолированный	Леса	25
3	<1	Воздушная линия	изолированный	Леса	16
4	<1	Трансформатор	неизолированный	Леса	25
5	<1	Кабельная линия	изолированный	Луга	16
6	<1	Кабельная линия	неизолированный	Болота	25
7	10000-∞	Трансформатор	изолированный	Болота	16
8	10000-∞	Кабельная линия	изолированный	Леса	16
9	10000-∞	Воздушная линия	неизолированный	Луга	25
10	1-10000	Кабельная линия	изолированный	Луга	16
11	1-10000	Кабельная линия	неизолированный	Болота	25
12	1-10000	Воздушная линия	изолированный	Леса	16
13	1-10000	Трансформатор	неизолированный	Леса	25

С помощью выделенных вариантов попарного тестирования, мы создаём таблицу тест-кейсов, таблица 31.

Таблица 31 – Тест – кейсы

№	Входные данные	Ожидаемый результат	Фактический результат
1	Длина: 1-10000; тип работ: трансформатор; тип провода: изолированный; тип местности: болота; сечение: 16	Успешный расчёт	Успешный расчёт (рисунок Д.1)
2	Длина: 1-10000; тип работ: кабельная линия; тип провода: неизолированный; тип местности: леса; сечение: 25	Успешный расчёт	Успешный расчёт (рисунок Д.2)
3	Длина: <1; тип работ: воздушная линия; тип провода: изолированный; тип местности: леса; сечение: 16	Ошибка	Ошибка: слишком маленькое значение (рисунок Д.3)

Продолжение таблицы 31

№	Входные данные	Ожидаемый результат	Фактический результат
4	Длина: <1; тип работ: трансформатор; тип провода: неизолированный; тип местности: леса; сечение: 25	Ошибка	Ошибка: слишком маленькое значение (рисунок Д.4)
5	Длина: <1; тип работ: кабельная линия; тип провода: изолированный; тип местности: луга; сечение: 16	Ошибка	Ошибка: слишком маленькое значение (рисунок Д.5)
6	Длина: <1; тип работ: кабельная линия; тип провода: неизолированный; тип местности: болота; сечение: 25	Ошибка	Ошибка: слишком маленькое значение (рисунок Д.6)
7	Длина: 10000-∞; тип работ: трансформатор; тип провода: изолированный; тип местности: болота; сечение: 16	Ошибка	Ошибка: слишком большое значение (рисунок Д.7)
8	Длина: 10000-∞; тип работ: кабельная линия; тип провода: изолированный; тип местности: леса; сечение: 16	Ошибка	Ошибка: слишком большое значение (рисунок Д.8)
9	Длина: 10000-∞; тип работ: воздушная линия; тип провода: неизолированный; тип местности: луга; сечение: 25	Ошибка	Ошибка: слишком большое значение (рисунок Д.9)
10	Длина: 1-10000; тип работ: кабельная линия; тип провода: изолированный; тип местности: луга; сечение: 16	Успешный расчёт	Успешный расчёт (рисунок Д.10)
11	Длина: 1-10000; тип работ: кабельная линия; тип провода: неизолированный; тип местности: болота; сечение: 25	Успешный расчёт	Успешный расчёт (рисунок Д.11)
12	Длина: 1-10000; тип работ: воздушная линия; тип провода: изолированный; тип местности: леса; сечение: 16	Успешный расчёт	Успешный расчёт (рисунок Д.12)
13	Длина: 1-10000; тип работ: трансформатор; тип провода: неизолированный; тип местности: леса; сечение: 25	Успешный расчёт	Успешный расчёт (рисунок Д.13)

При прохождении тест-кейсов, мы выявили что программа работает корректно. Результаты тестирования представлены в приложении Д.

Для проведения модульного тестирования был создан новый проект, в котором использовалась библиотека xUnit в среде Microsoft Visual Studio [13]. Код Unit-тестов приведён в приложении Г. В соответствии с разработанными тестами была разработана таблица 32.

Таблица 32 – План модульного тестирования

Тест	Описание	Ожидаемый результат	Актуальный результат
CalculateDetailedAsync_ValidInput_ReturnsCorrectTotalCost	Проверка корректности расчёта при заполнении всех полей (тип работ, тип провода, тип местности, сечение, тип опор, бригада, длина провода)	Пройден	Пройден
CalculateDetailedAsync_WithClientDiscount_AppliesDiscount	Проверка применения скидки клиента (10%)	Пройден	Пройден
CalculateDetailedAsync_WithoutOptionalFields_UsesDefaults	Проверка обработки отсутствия необязательных полей (тип опор, бригада, клиент)	Пройден	Пройден
CalculateDetailedAsync_TeamWithZeroWorkers_AddsNoTeamCost	Проверка, что бригада с нулевым количеством рабочих не добавляет стоимость	Пройден	Пройден
CalculateDetailedAsync_WithMissingWorkType_UsesDefault	Проверка обработки несуществующего типа работ (WorkTypeId = 999)	Пройден	Пройден
CalculateDetailedAsync_WithMissingWireType_UsesDefault	Проверка обработки несуществующего типа провода (WireTypeId = 999)	Пройден	Пройден
CalculateDetailedAsync_WithMissingTerrainType_UsesDefault	Проверка обработки несуществующего типа местности (TerrainTypeId = 999)	Пройден	Пройден
CalculateDetailedAsync_WithMissingSectionType_UsesDefault	Проверка обработки несуществующего типа сечения (SectionTypeId = 999)	Пройден	Пройден

На рисунке 25 приведены результаты выполнения модульного тестирования.

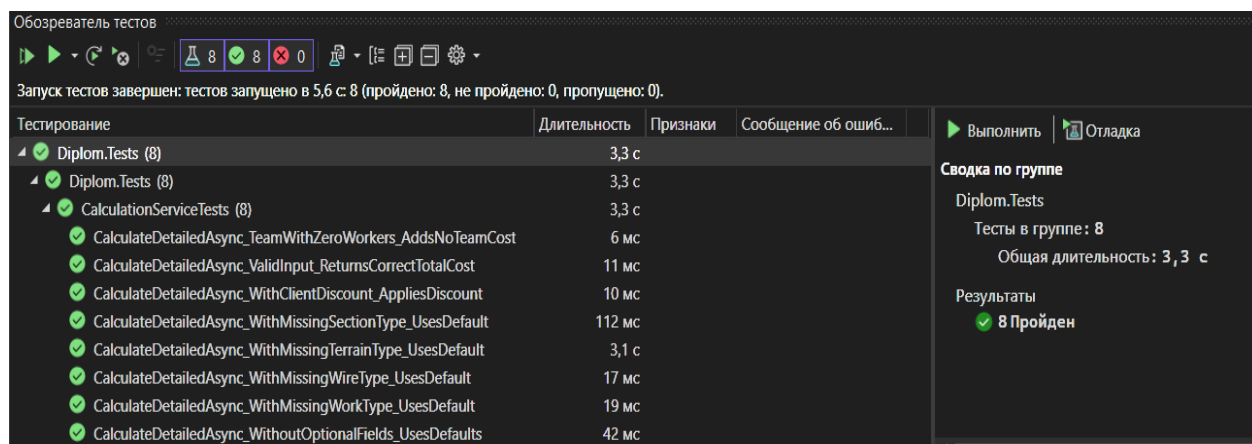


Рисунок 25 – Результаты выполнения модульных тестов

В ходе выполнения модульного тестирования ошибок в коде серверной части не было обнаружено. Полученные результаты подтверждают, что методы работают корректно.

5 Оценка качества программного кода.

5.1 Тестирование SLOC

Для оценки качества программного кода метода `CalculateDetailedAsync` были применены различные метрики: SLOC (анализ строк кода), метрика Холстеда, метрика Чепина, а также проведена автоматическая проверка с помощью встроенных инструментов Visual Studio. Данные метрики позволяют количественно оценить такие характеристики кода, как его объем, сложность, читаемость и поддерживаемость.

Анализ SLOC (Source Lines of Code) позволяет оценить физический и логический объем программного кода.

Исходный код метода `CalculateDetailedAsync` указан в листинге 1.

Листинг 1 – Код метода `CalculateDetailedAsync`

```
public async Task<Calculation> CalculateDetailedAsync(Calculation
calculation)
{
    calculation.CalculationType = "detailed";

    // Получаем все необходимые данные для детального расчёта
    var workType = await
_context.WorkTypes.FindAsync(calculation.WorkTypeId);
    var wireType = await
_context.WireTypes.FindAsync(calculation.WireTypeId);
    var terrainType = await
_context.TerrainTypes.FindAsync(calculation.TerrainTypeId);
    var sectionType = await
_context.SectionTypes.FindAsync(calculation.SectionTypeId);
    var supportType = await
_context.SupportTypes.FindAsync(calculation.SupportTypeId);

    // ДЕТАЛЬНЫЙ РАСЧЁТ: все параметры
    // Добавляем получение бригады
    Team team = null;
    if (calculation.TeamId.HasValue)
    {
        team = await
_context.Teams.FindAsync(calculation.TeamId.Value);
    }
    decimal baseCost = workType?.BaseCost ?? 0;
    decimal materialCost = (wireType?.CostPerMeter ?? 0) *
calculation.WireLength;
    decimal terrainCost = terrainType?.WorkCost ?? 0;
    decimal sectionCost = sectionType?.AdditionalCost ?? 0;
```

```

decimal supportCost = supportType?.InstallationCost ?? 0;
// Добавляем стоимость рабочих
decimal teamCost = (team?.WorkerCount ?? 0) * 100000m;
// Коэффициент сложности
decimal complexityFactor = 1.2m;

calculation.TotalCost = (baseCost + materialCost + terrainCost
+
                                sectionCost + supportCost+ teamCost) *
complexityFactor;

// Применяем скидку клиента, если она есть
if (calculation.ClientId.HasValue)
{
    var client = await
_context.Clients.FindAsync(calculation.ClientId.Value);
    if (client?.DiscountPercentage > 0)
    {
        calculation.TotalCost *= (100 -
client.DiscountPercentage.Value) / 100;
    }
}

_context.Calculations.Add(calculation);
await _context.SaveChangesAsync();

return calculation;
}

```

Результаты анализа метода:

- LOC (Lines of Code) – Общее количество физических строк в файле: 47;
- BLANK (Blank Lines) – Количество пустых строк: 5;
- CLOC (Comment Lines of Code) – Строки, содержащие комментарии: 8;
- SLOC (Source Lines of Code) – Значимых строк кода: 37;
- LLOC (Logical Lines of Code) – Логические строки кода: 32.

Несмотря на то, что метод занимает 47 физические строки, количество логических инструкций (32) почти на 40% меньше. Наличие комментариев и пустых строк (около 27% от объёма) улучшает читаемость и поддерживаемость кода.

5.2 Метрика Холстеда.

Метрики Холстеда были посчитаны для функции CalculateDetailedAsync [14]. Эта функция реализует детализированный расчёт работ. Код метода представлен в листинге приложения В.

Для расчета метрик Холстеда нужно подсчитать 4 характеристики программного кода: словарь, длину, объем и сложность программы. Словарь – это количество уникальных операторов и операндов. Длина программы – это общее количество операторов и операндов. Расчеты операторов приведены в таблице 33.

Таблица 33 – Расчёты операторов

Оператор	Количество повторений
async (тип данных)	1
Task (тип данных)	1
Calculation (тип данных)	1
await	5
var	10
=	3
?? (объединяющий оператор)	3
* (умножение)	2
+ (сложение)	4
- (вычитание)	1
/ (деление)	1
> (сравнение)	1
if	3
? (условный оператор)	4
FindAsync (вызов метода)	7
Add (вызов метода)	1
SaveChangesAsync (вызов метода)	1
() (круглые скобки)	14
{ } (фигурные скобки)	2
; (точка с запятой)	16
, (запятая)	7
Уникальных операторов (n1)	21
Всего операторов (N1)	85

Расчеты операндов приведены в таблице 34. Операнды – это данные, над которыми совершаются действия.

Таблица 34 – Расчёты Операндов

Операнд	Количество повторений
calculation (параметр)	7
_context	4
workType	2
wireType	2
terrainType	2
sectionType	2
supportType	2
Team	2
baseCost	1
materialCost	1
terrainCost	1
sectionCost	1
supportCost	1
teamCost	1
complexityFactor	1
Client	1
Константа «detailed»	1
Константа 0	4
Константа 100000m	1
Константа 1.2m	1
Константа 100	1
Уникальных операндов (n2)	21
Всего операндов (N2)	48

Теперь, когда у нас есть все исходные данные, мы подставляем их в классические формулы Холстеда:

- словарь программы (n): $n = n1 + n2 = 21 + 21 = 42$;
- длина программы (N): $N = N1 + N2 = 85 + 48 = 133$;
- объем программы (V): $V = N * \log_2(n) = 133 * 5,4 = 718,2$;
- сложность программы (D): $D = \left(\frac{n1}{2}\right) * \left(\frac{N2}{n2}\right) = 10,5 * 2,3 = 24,15$;
- усилие на понимание (E): $E = D * V = 24,15 * 718,2 = 17344,53$.

Сложность функции (D = 24.15) находится в допустимых пределах. В коде сбалансировано количество операторов и переменных (n1 = 21, n2 = 21). Объем программы (V = 718,2) умеренный – метод решает чётко поставленную задачу расчёта и не перегружен лишними конструкциями.

5.3 Метрика Чепина

Рассчитаем метрику Чепина для функции `CalculateDetailedAsync` приложение В [15, 16].

Для расчета этой метрики множество переменных разбивается на четыре группы по функциональности: вводимые, модифицируемые, управляющие, транзитные. Расчеты переменных приведены в таблице 35.

Таблица 35 – Группы Чепина

Группа	Вес	Переменные из кода	Количество
Р (вводимые) – данные только для чтения, не участвуют в управлении потоком	1	context, workType, wireType, terrainType, sectionType, supportType, team, baseCost, materialCost, terrainCost, sectionCost, supportCost, teamCost, complexityFactor	14
М (модифицируемые) – изменяются, но не управляют ветвлением	2	calculation (меняется), client (не изменяется, но в расчёте используется)	2
С (управляющие) – участвуют в условиях (if, ?:)	3	calculation.TeamId.HasValue, calculation.ClientId.HasValue, client?.DiscountPercentage > 0 – логические выражения, но непосредственными переменными-управляющими можно считать team (через HasValue) и client (через проверку скидки)	2
Т (транзитные) – вспомогательные, не несут значимой нагрузки	0.5	отсутствуют	0

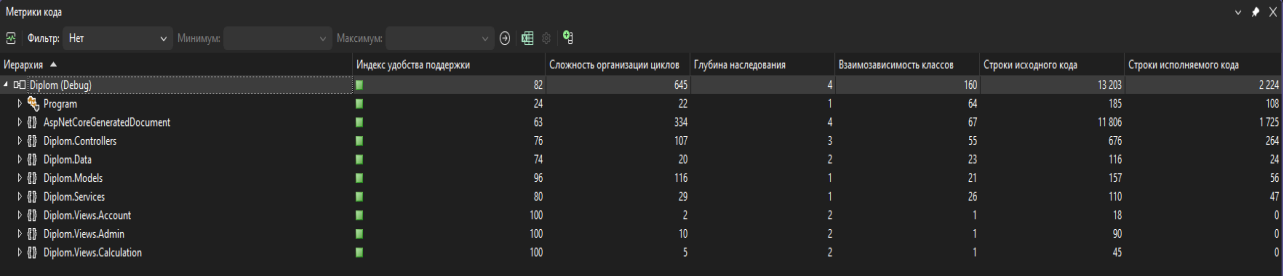
Расчёт по формуле метрики Чепина выглядит так:

- $Q = a1 * P + a2 * M + a3 * C + a4 * T$;
- $Q = (1 * 14) + (2 * 2) + (3 * 2) + (0,5 * 0)$;
- $Q = 14 + 4 + 6 + 0$;
- $Q = 24$.

Информационная сложность метода `CalculateDetailedAsync` составляет 24. Основной вклад (14 баллов) дают входные данные (группа Р), что характерно для методов, выполняющих сложные вычисления на основе множества параметров. Управляющие переменные (группа С) представлены двумя условиями, что оправдано логикой метода (проверка наличия бригады и скидки клиента). Транзитные переменные отсутствуют, код написан оптимально.

5.4 Автоматическая проверка.

Автоматическая проверка проводилась с помощью встроенных инструментов Visual Studio рисунок 26.



The screenshot shows the 'Metrics' window in Visual Studio Code. It displays a table of code quality metrics for a project named 'Diplom (Debug)'. The table has seven columns: 'Иерархия' (Hierarchy), 'Индекс удобства поддержки' (Index of maintainability), 'Сложность организации циклов' (Cycle organization complexity), 'Глубина наследования' (Depth of inheritance), 'Взаимозависимость классов' (Class interdependency), 'Строки исходного кода' (Lines of source code), and 'Строки исполняемого кода' (Lines of executable code). The table lists various components of the project, including 'Program', 'AspNetCoreGeneratedDocument', 'Diplom.Controllers', 'Diplom.Data', 'Diplom.Models', 'Diplom.Services', 'Diplom.Views.Account', 'Diplom.Views.Admin', and 'Diplom.Views.Calculation'. Each component has a corresponding value for each metric, with green bars indicating the relative quality score.

Иерархия	Индекс удобства поддержки	Сложность организации циклов	Глубина наследования	Взаимозависимость классов	Строки исходного кода	Строки исполняемого кода
Diplom (Debug)	82	645	4	160	13 203	2 224
Program	24	22	1	64	185	108
AspNetCoreGeneratedDocument	63	334	4	67	11 806	1 725
Diplom.Controllers	76	107	3	55	676	264
Diplom.Data	74	20	2	23	116	24
Diplom.Models	96	116	1	21	157	56
Diplom.Services	80	29	1	26	110	47
Diplom.Views.Account	100	2	2	1	18	0
Diplom.Views.Admin	100	10	2	1	90	0
Diplom.Views.Calculation	100	5	2	1	45	0

Рисунок 26 – Результат работы расчётов

С помощью встроенных инструментов Visual Studio был исследован проект (Diplom). Анализатор выделил 6 критериев: индекс удобства поддержки (82), сложность организации циклов (645), глубина наследования (4), взаимосвязанность классов (160), строки исходного кода (13203), строки исполняемого кода (2224). По высокому показателю индекса удобства поддержки (>75), мы можем сделать вывод, что код написан оптимально [17, 18, 19].

6 Планирование разработки и оценка бюджета

Для удобной визуализации и управления задачами разработки приложения для поддержки планирования и организации работ в электроэнергетике было выбрано приложение Microsoft Project. С помощью этого приложения была построена диаграмма гранта, представленная на рисунке 27 [20].

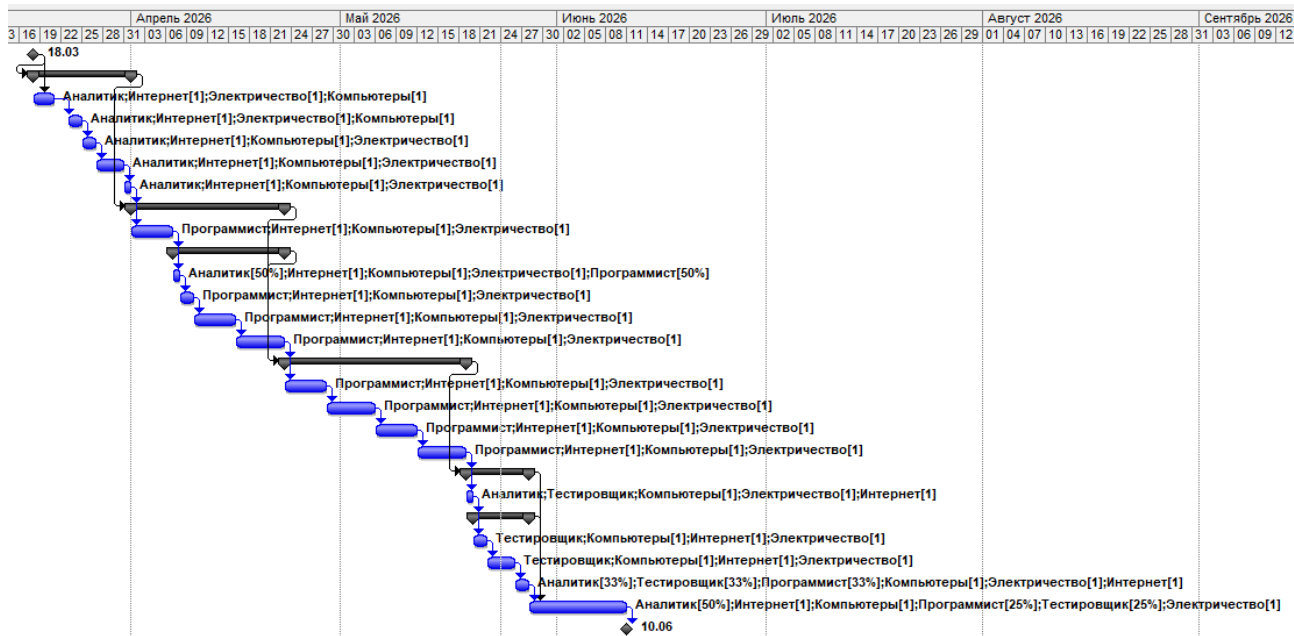


Рисунок 27 – Диаграмма Ганта

Дата начала разработки проекта 18 марта 2026 года, дата окончания – 10 июня того же года. Общая продолжительность разработки составила 84 дня.

Для оценки затрат на разработку приложения, были назначены ресурсы и указана стоимость их использования, представленная на рисунке 28.

	Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных	Затраты на использ.
1	Программист	Трудовой	500,00 Р/ч	725,00 Р/ч	0,00 Р
2	Тестировщик	Трудовой	400,00 Р/ч	650,00 Р/ч	0,00 Р
3	Аналитик	Трудовой	400,00 Р/ч	630,00 Р/ч	0,00 Р
4	Компьютеры	Материальный	0,00 Р		0,00 Р
5	Интернет	Материальный	7,00 Р		0,00 Р
6	Электричество	Материальный	10,00 Р		0,00 Р

Рисунок 28 – Данные о ресурсах

Было выделено 27 задач, сроки и описания которых представлены на рисунке 29.

	Название задачи	Длительность	Начало	Окончание
1	Начало проекта	0 дней	Ср 18.03.26	Ср 18.03.26
2	Постановка задачи и анализ требований	10 дней	Ср 18.03.26	Вт 31.03.26
3	Постановка задачи	3 дней	Ср 18.03.26	Пт 20.03.26
4	Анализ систем аналогичного назначения	2 дней	Пн 23.03.26	Вт 24.03.26
5	Формулировка требований	1 день	Ср 25.03.26	Чт 26.03.26
6	Выбор и обоснование технологий	2 дней	Пт 27.03.26	Пн 30.03.26
7	Планирование разработки и оценка бюджета	1 день	Вт 31.03.26	Вт 31.03.26
8	Проектирование приложения для поддержки планирования и организации работ в элетроэнеогетике	16 дней	Ср 01.04.26	Ср 22.04.26
9	Архитектура приложения	4 дней	Ср 01.04.26	Пн 06.04.26
10	Проектирование структур данных	12 дней	Вт 07.04.26	Ср 22.04.26
11	Концептуальная модель базы данных	1 день	Вт 07.04.26	Вт 07.04.26
12	Логическая модель базы данных	2 дней	Ср 08.04.26	Чт 09.04.26
13	Проектирование пользовательского интерфейса	4 дней	Пт 10.04.26	Ср 15.04.26
14	Проектирование классов приложения для поддержки планирования и организации работ в элетроэнеогет	5 дней	Чт 16.04.26	Ср 22.04.26
15	Реализация приложения для поддержки планирования и организации работ в элетроэнеогетике	18 дней	Чт 23.04.26	Пн 18.05.26
16	Реализация базы данных	4 дней	Чт 23.04.26	Вт 28.04.26
17	Реализация серверной части приложения	5 дней	Ср 29.04.26	Вт 05.05.26
18	Реализация клиентской части приложения	4 дней	Ср 06.05.26	Пн 11.05.26
19	Реализация интерфейса приложения	5 дней	Вт 12.05.26	Пн 18.05.26
20	Контроль качества приложения	7 дней	Вт 19.05.26	Ср 27.05.26
21	Обоснование режима тестирования и метрик кода	1 день	Вт 19.05.26	Вт 19.05.26
22	Тестирование приложения для поддержки планирования и организации работ в элетроэнеогетике	6 дней	Ср 20.05.26	Ср 27.05.26
23	Функциональное тестирование	2 дней	Ср 20.05.26	Чт 21.05.26
24	Автоматическое тестирование	2 дней	Пт 22.05.26	Пн 25.05.26
25	Оценка качества программного кода	2 дней	Вт 26.05.26	Ср 27.05.26
26	Оформление пояснительной записки	10 дней	Чт 28.05.26	Ср 10.06.26
27	Конец проекта	0 дней	Ср 10.06.26	Ср 10.06.26

Рисунок 29 – Планирование разработки

Денежные затраты по данным срокам и задачам, представлены на рисунке 30.

	Название задачи	Оставшиеся затраты
1	Начало проекта	0,00 Р
2	Постановка задачи и анализ требований	28 885,00 Р
3	Постановка задачи	9 617,00 Р
4	Анализ систем аналогичного назначения	6 417,00 Р
5	Формулировка требований	3 217,00 Р
6	Выбор и обоснование технологий	6 417,00 Р
7	Планирование разработки и оценка бюджета	3 217,00 Р
8	Проектирование приложения для поддержки планирования и организации работ в элетроэнеогетике	63 685,00 Р
9	Архитектура приложения	16 017,00 Р
10	Проектирование структур данных	47 668,00 Р
11	Концептуальная модель базы данных	3 617,00 Р
12	Логическая модель базы данных	8 017,00 Р
13	Проектирование пользовательского интерфейса	16 017,00 Р
14	Проектирование классов приложения для поддержки планирования и организации работ в элетроэнеогет	20 017,00 Р
15	Реализация приложения для поддержки планирования и организации работ в элетроэнеогетике	72 068,00 Р
16	Реализация базы данных	16 017,00 Р
17	Реализация серверной части приложения	20 017,00 Р
18	Реализация клиентской части приложения	16 017,00 Р
19	Реализация интерфейса приложения	20 017,00 Р
20	Контроль качества приложения	26 132,00 Р
21	Обоснование режима тестирования и метрик кода	6 417,00 Р
22	Тестирование приложения для поддержки планирования и организации работ в элетроэнеогетике	19 715,00 Р
23	Функциональное тестирование	6 417,00 Р
24	Автоматическое тестирование	6 417,00 Р
25	Оценка качества программного кода	6 881,00 Р
26	Оформление пояснительной записки	28 017,00 Р
27	Конец проекта	0,00 Р

Рисунок 30 – Оценка бюджета

По результатам оценки затрат, видно сколько и на какой этап будет затрачено средств. Сводка по всему проекту предоставлена на рисунке 31.

<u>Даты</u>			
Начало:	Ср 18.03.26	Окончание:	Ср 10.06.26
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
<u>Длительность</u>			
Текущий план:	61 дней	Оставшаяся:	61 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	61 дней	Процент завершени	0%
<u>Трудозатраты</u>			
Текущий план:	474,5 ч	Оставшиеся:	474,5 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	474,5 ч	Процент завершени	0%
<u>Затраты</u>			
Текущий план:	218 787,00 Р	Оставшиеся:	218 787,00 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	218 787,00 Р		
<u>Состояние задачи</u>		<u>Состояние ресурса</u>	
Еще не начатые задачи	27	Трудовые ресурсы:	3
Выполняющиеся задачи	0	Трудовые ресурсы с прев. дос	0
Завершенные задачи:	0	Материальные ресурсы:	3
Итого задач:	27	Итого ресурсов:	6

Рисунок 31 – Сводка по проекту

На сводке показаны количество дней, затраченных на реализацию проекта, дата начала и окончания проекта и затраты на проект. Затраты, составляющие 218 787 рублей и оценка их сроков показывают, что данный проект экономически выгоден.

Заключение

В ходе выпускной квалификационной работы было разработано и внедрено приложение для информационной поддержки планирования и организации работ в электротехнической отрасли, позволяющее производить первичный и детальный расчёт затрат на электромонтажные работы, а также вести историю расчётов и осуществлять административное управление справочными данными.

Этап анализа требований включил изучение предметной области, сравнение существующих аналогов и определение совокупности функциональных и нефункциональных требований. Также были построены концептуальная модель предметной области и диаграмма вариантов использования.

В процессе проектирования приложения были созданы концептуальная, логическая и физическая модели базы данных, выбрана многоуровневая монолитная архитектура приложения, определены контроллеры, сервисы и интерфейсы серверной части, а также разработан пользовательский интерфейс для двух категорий пользователей: клиент и администратор.

Разработка приложения включала развёртывание базы данных на Microsoft SQL Server с использованием Entity Framework Core, создание серверной части на платформе ASP.NET Core в среде Microsoft Visual Studio 2022, а также клиентскую реализацию на основе представлений Razor с применением HTML, CSS и JavaScript.

Этап отладки и тестирования приложения включал функциональное тестирование (метод «чёрного ящика» с построением функциональной схемы и попарным тестированием) и модульное тестирование с использованием библиотеки xUnit. Все тестовые случаи были успешно пройдены, что подтвердило корректную работу приложения и его соответствие функциональным требованиям.

На этапе оценки качества кода рассчитаны метрики SLOC, Холстеда и Чепина для ключевого метода (CalculateDetailedAsync). Значение индекса

сопровождаемости (82) и умеренные показатели сложности ($D=24,15$, $Q=24$) свидетельствуют о высоком качестве и поддерживаемости кода. Автоматическая проверка средствами Visual Studio также не выявила критических замечаний.

Планирование разработки завершилось построением диаграммы Ганта. В результате проведённых расчётов общая продолжительность всех этапов разработки составила 84 дня, а ориентировочная стоимость реализации приложения – 218 787 руб.

Таким образом, основная цель выпускной квалификационной работы – разработка приложения для информационного поддержки планирования и организации работ в электроэнергетике – успешно достигнута, все поставленные задачи выполнены. Разработанное приложение демонстрирует высокий уровень качества, удобный интерфейс и полностью соответствует предъявляемым требованиям.

Список использованных источников

1. Использование диаграммы вариантов использования UML при проектировании программного обеспечения // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/566218/> (дата обращения: 10.05.2026)
2. Что такое нефункциональные требования: примеры, определение, полное руководство // VISURESOLUTIONS [Электронный ресурс]. URL: <https://visuresolutions.com/ru/Блог/нефункциональные-требования/> (дата обращения: 10.05.2026)
3. Слоистая архитектура // Школа системного анализа и проектирования [Электронный ресурс]. URL: <https://systems.education/layered-architecture> (дата обращения: 10.05.2026)
4. Когда использовать базу данных MS SQL Server? // Группа компаний ДБ-сервис [Электронный ресурс]. URL: <https://www.dbserve.ru/blog/kogda-ispolzovat-bazu-dannyh-ms-sql-server> (дата обращения: 10.05.2026)
5. Хомоненко, А. Д. Базы данных. Учебник для вузов / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев. – М. :ДМК Пресс, 2013. – 232 с.
6. Система управления базами данных SQL Server // Программирование на C#, платформа .NET Framework [Электронный ресурс]. URL: https://professorweb.ru/my/sqlserver/2012/level1/1_1.php (дата обращения: 10.05.2026).
7. Концептуальное моделирование (Conceptual Modeling): основы, методы и применение // Я зерокодер [Электронный ресурс]. URL: <https://ya.zerocoder.ru/pgt-konceptualnoe-modelirovanie-conceptual-modeling-osnovy-metody-i-primenenie/> (дата обращения: 10.05.2026)
8. ASP.NET Core // WORKSPACE [Электронный ресурс]. URL: <https://workspace.ru/tools/framework/asp-dot-net-core/> (дата обращения: 10.05.2026)
9. Физическая модель данных // APPMASTER [Электронный ресурс]. URL: <https://appmaster.io/ru/glossary/fizicheskaia-model-dannykh> (дата обращения: 10.05.2026)

10. Котляров, В. П. Основы тестирования программного обеспечения / В. П. Котляров, Т. В. Коликова. – М. :Бином. Лаборатория знаний, 2006. – 285 с.
11. Майерс, Г. Искусство тестирования программ / Г. Майерс. – М. :Финансы и статистика, 2004. – 267 с.
12. Макконнелл, С. Совершенный код / С. Макконнелл. – М. :Русская редакция, 2019. – 896 с.
13. Месарош, Д. xUnit Test Patterns: Рефакторинг тестового кода / Д. Месарош. – СПб. :Питер, 2020. – 896 с.
14. Diplom // GitHub [Электронный ресурс]. URL: <https://github.com/Denis6522/Diplom.git> (дата обращения: 10.05.2026).
15. Чепин, М. Метрики качества программного обеспечения / М. Чепин. – М. :Наука, 1983. – 180 с.
16. Лексический анализ. Метрики Чепина // StudFiles [Электронный ресурс]. URL: <https://studfile.net/preview/6196490/page:8/> (дата обращения: 10.05.2026).
17. Метрики кода программного обеспечения // PVS-studio [Электронный ресурс]. URL: <https://pvs-studio.ru/ru/blog/posts/a0045/> (дата обращения: 10.05.2026).
18. Значения метрик кода // Microsoft [Электронный ресурс]. URL: <https://msdn.microsoft.com/ru-ru/library/bb385914.aspx> (дата обращения: 10.05.2026).
19. Роль качества кода: критические метрики и их влияние // IN-COM [Электронный ресурс]. URL: <https://www.in-com.com/ru/blog/the-role-of-code-quality-critical-metrics-and-their-impact/> (дата обращения: 07.06.2025)
20. Куперштейн, В. И. Microsoft Project 2010 в управлении проектами / В. И. Куперштейн, А. В. Цветкова. – СПб. :БХВ-Петербург, 2011. – 416 с.

Приложение А
(справочное)
Глоссарий

БД (База данных) – организованная структура, направленная на управление большими объемами данных.

СУБД (Система управления базами данных) – комплекс программных инструментов, предназначенный для создания баз данных и работы с ними.

Сущность – реальный или абстрактный объект, информация о котором хранится в базе данных и является доступной. В базе данных сущность представлена в виде таблицы.

Атрибут – характеристика или признак, описывающий объект.

UML (Unified Modeling Language) – графический язык объектного моделирования для визуализации и проектирования программного обеспечения.

RUP (Rational Unified Process) – методология создания программного обеспечения, которая основывается на итеративном подходе, чётком распределении ролей в команде, управлении требованиями и учёте ограничений, влияющих на процесс разработки.

API (Application Programming Interface) – набор правил и инструментов, с помощью которых взаимодействуют и обмениваются данными между собой разные части программной системы.

REST (Representational State Transfer) – архитектурный стиль, который применяется для построения распределённых и легко масштабируемых систем.

Приложение Б
(обязательное)
Листинг серверной части приложения

Листинг Б.1 – Файл CalculationController.cs

```

namespace Diplom.Controllers
{
    public class CalculationController : Controller
    {
        private readonly ICalculationService _calculationService;
        private readonly ApplicationDbContext _context;

        public CalculationController(ICalculationService
calculationService, ApplicationDbContext context)
        {
            _calculationService = calculationService;
            _context = context;
        }

        public IActionResult Index()
        {
            ViewData["CalculationTypes"] = new List<string> {
"primary", "detailed" };
            return View();
        }

        public async Task<IActionResult> Primary()
        {
            ViewBag.WorkTypes = await
_context.WorkTypes.ToListAsync();
            ViewBag.SectionTypes = await
_context.SectionTypes.ToListAsync();

            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Primary(Calculation
calculation)
        {
            if (ModelState.IsValid)
            {
                var result = await
_calculationService.CalculatePrimaryAsync(calculation);
                return RedirectToAction("Result", new { id =
result.Id });
            }

            ViewBag.WorkTypes = await
_context.WorkTypes.ToListAsync();
            ViewBag.SectionTypes = await
_context.SectionTypes.ToListAsync();

            return View(calculation);
        }
    }
}

```

```

        // Страница детального расчёта
        public async Task<IActionResult> Detailed()
        {
            // ВСЕ типы для детального расчёта
            ViewBag.WorkTypes = await
            _context.WorkTypes.ToListAsync();
            ViewBag.WireTypes = await
            _context.WireTypes.ToListAsync();
            ViewBag.TerrainTypes = await
            _context.TerrainTypes.ToListAsync();
            ViewBag.SectionTypes = await
            _context.SectionTypes.ToListAsync();
            ViewBag.SupportTypes = await
            _context.SupportTypes.ToListAsync();
            ViewBag.Clients = await
            _context.Clients.ToListAsync();
            ViewBag.Teams = await _context.Teams.ToListAsync();

            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Detailed(Calculation
calculation)
        {
            if (ModelState.IsValid)
            {
                var result = await
            _calculationService.CalculateDetailedAsync(calculation);
                return RedirectToAction("Result", new { id =
result.Id });
            }

            ViewBag.WorkTypes = await
            _context.WorkTypes.ToListAsync();
            ViewBag.WireTypes = await
            _context.WireTypes.ToListAsync();
            ViewBag.TerrainTypes = await
            _context.TerrainTypes.ToListAsync();
            ViewBag.SectionTypes = await
            _context.SectionTypes.ToListAsync();
            ViewBag.SupportTypes = await
            _context.SupportTypes.ToListAsync();
            ViewBag.Clients = await
            _context.Clients.ToListAsync();
            ViewBag.Teams = await _context.Teams.ToListAsync();

            return View(calculation);
        }

        public async Task<IActionResult> History()

```

```

        {
            var calculations = await
            _calculationService.GetCalculationHistoryAsync();
            return View(calculations);
        }

        public async Task<IActionResult> Result(int id)
        {
            var calculation = await
            _calculationService.GetCalculationByIdAsync(id);
            if (calculation == null)
            {
                return NotFound();
            }

            return View(calculation);
        }

        [HttpDelete("/Calculation/DeleteCalculation/{id}")]
        public async Task<IActionResult> DeleteCalculation(int id)
        {
            try
            {
                var calculation = await
                _context.Calculations.FindAsync(id);
                if (calculation == null)
                    return Json(new { success = false, message =
                    "Расчёт не найден." });

                _context.Calculations.Remove(calculation);
                await _context.SaveChangesAsync();

                return Json(new { success = true, message =
                "Расчёт удалён." });
            }
            catch (Exception ex)
            {
                return Json(new { success = false, message =
                "Ошибка: " + ex.Message });
            }
        }
    }
}

```

Листинг Б.2 – Файл CalculationService.cs

```

namespace Diplom.Services
{
    public interface ICalculationService
    {
        Task<Calculation> CalculatePrimaryAsync(Calculation
        calculation);
        Task<Calculation> CalculateDetailedAsync(Calculation
        calculation);
    }
}

```

```

        Task<List<Calculation>> GetCalculationHistoryAsync();
        Task<Calculation> GetCalculationByIdAsync(int id);
    }

    public class CalculationService : ICalculationService
    {
        private readonly ApplicationDbContext _context;

        public CalculationService(ApplicationDbContext context)
        {
            _context = context;
        }

        public async Task<Calculation>
        CalculatePrimaryAsync(Calculation calculation)
        {
            calculation.CalculationType = "primary";

            var workType = await
            _context.WorkTypes.FindAsync(calculation.WorkTypeId);
            var sectionType = await
            _context.SectionTypes.FindAsync(calculation.SectionTypeId);

            decimal baseCost = workType?.BaseCost ?? 0;
            decimal materialCost = (sectionType?.AdditionalCost ??
            0) * calculation.WireLength;

            calculation.TotalCost = baseCost + materialCost;

            _context.Calculations.Add(calculation);
            await _context.SaveChangesAsync();

            return calculation;
        }

        public async Task<Calculation>
        CalculateDetailedAsync(Calculation calculation)
        {
            calculation.CalculationType = "detailed";

            var workType = await
            _context.WorkTypes.FindAsync(calculation.WorkTypeId);
            var wireType = await
            _context.WireTypes.FindAsync(calculation.WireTypeId);
            var terrainType = await
            _context.TerrainTypes.FindAsync(calculation.TerrainTypeId);
            var sectionType = await
            _context.SectionTypes.FindAsync(calculation.SectionTypeId);
            var supportType = await
            _context.SupportTypes.FindAsync(calculation.SupportTypeId);

            Team team = null;
            if (calculation.TeamId.HasValue)

```

```

        {
            team = await
_context.Teams.FindAsync(calculation.TeamId.Value);
        }
        decimal baseCost = workType?.BaseCost ?? 0;
        decimal materialCost = (wireType?.CostPerMeter ?? 0) *
calculation.WireLength;
        decimal terrainCost = terrainType?.WorkCost ?? 0;
        decimal sectionCost = sectionType?.AdditionalCost ??
0;
        decimal supportCost = supportType?.InstallationCost ??
0;
        decimal teamCost = (team?.WorkerCount ?? 0) * 100000m;
        decimal complexityFactor = 1.2m;

        calculation.TotalCost = (baseCost + materialCost +
terrainCost +
                                sectionCost + supportCost+
teamCost) * complexityFactor;
        if (calculation.ClientId.HasValue)
        {
            var client = await
_context.Clients.FindAsync(calculation.ClientId.Value);
            if (client?.DiscountPercentage > 0)
            {
                calculation.TotalCost *= (100 -
client.DiscountPercentage.Value) / 100;
            }
        }

        _context.Calculations.Add(calculation);
        await _context.SaveChangesAsync();

        return calculation;
    }

    public async Task<List<Calculation>>
GetCalculationHistoryAsync()
    {
        return await _context.Calculations
            .Include(c => c.WorkType)
            .Include(c => c.SectionType)
            .OrderByDescending(c => c.CalculationDate)
            .ToListAsync();
    }

    public async Task<Calculation> GetCalculationByIdAsync(int
id)
    {
        return await _context.Calculations
            .Include(c => c.WorkType)
            .Include(c => c.SectionType)
            .Include(c => c.WireType)

```

```
        .Include(c => c.TerrainType)
        .Include(c => c.SupportType)
        .Include(c => c.Client)
        .Include(c => c.Team)
        .FirstOrDefaultAsync(c => c.Id == id);
    }
}
```

Приложение В
(обязательное)
Листинг клиентской части приложения

Листинг В.1 – Файл Detailed.cshtml

```

<div class="container">
    <h1>Детальный расчёт стоимости работ</h1>

    <!-- Сообщения об ошибках -->
    @if (!ViewData.ModelState.IsValid)
    {
        <div class="validation-summary">
            <strong>Исправьте следующие ошибки:</strong>
            <ul>
                @foreach (var error in
ViewData.ModelState.Values.SelectMany(v => v.Errors))
                {
                    <li>@error.ErrorMessage</li>
                }
            </ul>
        </div>
    }

    <form method="post" asp-action="Detailed">
        @Html.AntiForgeryToken()

        <div class="row">
            <div class="col">
                <div class="form-group">
                    <label>Тип работ <span
class="required">*</span></label>
                    <select name="WorkTypeId" required>
                        <option value="">-- Выберите тип работ --
</option>

                        @if (ViewBag.WorkTypes != null)
                        {
                            foreach (var item in
ViewBag.WorkTypes)
                            {
                                <option value="@item.Id" data-
cost="@item.BaseCost">
                                    @item.Name -
                                    @item.BaseCost.ToString("N0") руб.
                                </option>
                            }
                        }
                    </select>
                    <span class="field-validation-
error">@Html.ValidationMessageFor(m => m.WorkTypeId)</span>
                </div>

                <div class="form-group">
                    <label>Тип провода <span
class="required">*</span></label>
                    <select name="WireTypeId" required>

```

```

--</option>
        <option value="">-- Выберите тип провода -
        @if (ViewBag.WireTypes != null)
        {
            foreach (var item in
ViewBag.WireTypes)
            {
                <option value="@item.Id" data-
cost-per-meter="@item.CostPerMeter">
                    @item.Name -
@item.CostPerMeter.ToString("N2") руб./м
                </option>
            }
        }
    </select>
    <span class="field-validation-
error">@Html.ValidationMessageFor(m => m.WireTypeId)</span>
</div>

    <div class="form-group">
        <label>Тип местности <span
class="required">*</span></label>
        <select name="TerrainTypeId" required>
            <option value="">-- Выберите тип местности
--</option>
            @if (ViewBag.TerrainTypes != null)
            {
                foreach (var item in
ViewBag.TerrainTypes)
                {
                    <option value="@item.Id" data-
work-cost="@item.WorkCost">
                        @item.Name -
@item.WorkCost.ToString("N2") руб.
                    </option>
                }
            }
        </select>
        <span class="field-validation-
error">@Html.ValidationMessageFor(m => m.TerrainTypeId)</span>
    </div>

    <div class="form-group">
        <label>Сечение провода <span
class="required">*</span></label>
        <select name="SectionTypeId" required>
            <option value="">-- Выберите сечение
провода --</option>
            @if (ViewBag.SectionTypes != null)
            {
                foreach (var item in
ViewBag.SectionTypes)
                {

```

```

                                <option value="@item.Id" data-
additional-cost="@item.AdditionalCost">
                                @item.Name - @item.Description
- @item.AdditionalCost руб./метр
                                </option>
                                }
                                }
                                </select>
                                <span class="field-validation-
error">@Html.ValidationMessageFor(m => m.SectionTypeId)</span>
                                </div>
                                </div>

                                <div class="col">
                                <div class="form-group">
                                <label>Тип опор</label>
                                <select name="SupportTypeId">
                                <option value="">-- Не выбрано --</option>
                                @if (ViewBag.SupportTypes != null)
                                {
                                foreach (var item in
ViewBag.SupportTypes)
                                {
                                <option value="@item.Id" data-
installation-cost="@item.InstallationCost">
                                @item.Name -
                                @item.InstallationCost.ToString("N2") руб.
                                </option>
                                }
                                }
                                </select>
                                </div>

                                <div class="form-group">
                                <label>Длина провода (м) <span
class="required">*</span></label>
                                <input type="number" name="WireLength" min="1"
max="10000" required
                                value="@ (Model?.WireLength.ToString()
?? "100") ">
                                <span class="field-validation-
error">@Html.ValidationMessageFor(m => m.WireLength)</span>
                                </div>

                                <div class="form-group">
                                <label>Клиент</label>
                                <select name="ClientId">
                                <option value="">-- Не выбрано --</option>
                                @if (ViewBag.Clients != null)
                                {
                                foreach (var item in ViewBag.Clients)
                                {

```

```

                                <option value="@item.Id"
selected="@ (Model?.ClientId == item.Id) ">
                                @item.FullName
@ (item.DiscountPercentage > 0 ? $"(-{item.DiscountPercentage}%) " :
"")
                                </option>
                                }
                                }
                                </select>
                                </div>

                                <div class="form-group">
                                    <label>Бригада</label>
                                    <select name="TeamId">
                                        <option value="">-- Не выбрано --</option>
                                        @if (ViewBag.Teams != null)
                                        {
                                            foreach (var item in ViewBag.Teams)
                                            {
                                                <option value="@item.Id" data-
worker-count="@item.WorkerCount">
                                                    @item.Name ( @item.WorkerCount
чел. )
                                                </option>
                                            }
                                        }
                                    </select>
                                </div>
                                </div>
                                </div>

                                <div class="form-group">
                                    <label>Примечания</label>
                                    <textarea name="Notes" rows="3"
placeholder="Дополнительная
информация...">@Model?.Notes</textarea>
                                </div>

                                <div id="previewResult" class="preview-result">
                                    <div>Предварительная стоимость:</div>
                                    <div class="preview-cost" id="previewCost">0
руб.</div>
                                </div>

                                <div class="button-group">
                                    <button type="submit" class="btn btn-
calculate">Рассчитать стоимость</button>
                                    <a href="/Calculation" class="btn btn-back">Назад</a>
                                </div>
                                </form>
                                </div>

                                <script>

```

```

function calculatePreview() {
    const workType =
document.querySelector('[name="WorkTypeId"]').value;
    const wireType =
document.querySelector('[name="WireTypeId"]').value;
    const terrainType =
document.querySelector('[name="TerrainTypeId"]').value;
    const wireLength =
document.querySelector('[name="WireLength"]').value;
    const sectionType =
document.querySelector('[name="SectionTypeId"]').value;
    const supportType =
document.querySelector('[name="SupportTypeId"]').value;

    if (!workType || !wireType || !terrainType || !wireLength)
{
        alert("Пожалуйста, заполните все обязательные поля
(помеченные *)");
        return;
    }

    if (wireLength < 1 || wireLength > 10000) {
        alert("Длина провода должна быть от 1 до 10000
метров");
        return;
    }

    const workCost = 5000;
    const wireCost = 150;
    const terrainCost = 1000;
    const sectionCost = sectionType ? 500 : 0;
    const supportCost = supportType ? 2000 : 0;

    let totalCost = workCost + (wireCost * wireLength) +
terrainCost + sectionCost + supportCost;

    totalCost = totalCost * 1.1;

    const formattedCost = totalCost.toLocaleString('ru-RU', {
        minimumFractionDigits: 2,
        maximumFractionDigits: 2
    });

    document.getElementById('previewCost').textContent =
formattedCost + ' руб.';
    document.getElementById('previewResult').style.display =
'block';

    document.getElementById('previewResult').scrollIntoView({
behavior: 'smooth' });
}
</script>

```

Листинг В.2 – Файл Primary.cshtml

```

<div class="page-container">
    <!-- ЗАГОЛОВОК СТРАНИЦЫ -->
    <div class="page-header">
        <h1 class="page-title">
            <i class="fas fa-bolt"></i> Первичный расчёт стоимости
работ
        </h1>
        <p class="page-subtitle">
            Быстрый расчёт стоимости электромонтажных работ по
основным параметрам.
            Заполните форму ниже для получения ориентировочной
стоимости.
        </p>
    </div>

    <!-- ФОРМА РАСЧЁТА -->
    <div class="form-container">
        <!-- СТАТУС ФОРМЫ -->
        <div class="form-status">
            <div class="status-text">
                <i class="fas fa-info-circle"></i> Заполните все
обязательные поля для расчёта
            </div>
            <div class="field-counter">
                <span id="filledFields">0</span> из <span
id="totalFields">4</span> заполнено
            </div>
        </div>
        <form id="primaryForm" method="post"
action="@Url.Action("Primary", "Calculation")">
            @Html.AntiForgeryToken()

            <!-- РАЗДЕЛ: ОСНОВНЫЕ ПАРАМЕТРЫ -->
            <div class="form-section">
                <h3 class="form-section-title">
                    <i class="fas fa-sliders-h"></i> Основные
параметры расчёта
                </h3>

                <div class="form-row">
                    <!-- ТИП РАБОТ -->
                    <div class="form-group">
                        <label class="form-label"
for="WorkTypeId">
                            Тип работ <span class="required-
marker">*</span>
                        </label>
                        <select id="WorkTypeId" name="WorkTypeId"
class="form-control-custom" required>
                            <option value="">-- Выберите тип работ
--</option>

```

```

@if (ViewBag.WorkTypes != null)
{
    foreach (var item in
ViewBag.WorkTypes)
    {
        <option value="@item.Id" data-
cost="@item.BaseCost">
            @item.Name -
            @item.BaseCost.ToString("N0") руб.
        </option>
    }
}
</select>
</div>

<!-- СЕЧЕНИЕ ПРОВОДА -->
<div class="form-group">
    <label class="form-label"
for="SectionTypeId">
        Сечение провода <span class="required-
marker">*</span>
        <span class="tooltip">
            <i class="fas fa-question-circle
tooltip-icon"></i>
            <span class="tooltip-text">
                Выберите сечение
электрического провода.
                От сечения зависит стоимость
материалов и допустимая нагрузка.
            </span>
        </span>
    </label>
    <select id="SectionTypeId"
name="SectionTypeId" class="form-control-custom" required>
        <option value="">-- Выберите сечение
провода --</option>
        @if (ViewBag.SectionTypes != null)
        {
            foreach (var item in
ViewBag.SectionTypes)
            {
                <option value="@item.Id" data-
cost="@item.AdditionalCost">
                    @item.Name -
                    @item.Description - @item.AdditionalCost руб./метр
                </option>
            }
        }
    </select>
    <span class="form-hint">Большее сечение =
выше стоимость материалов</span>
</div>

```

```

<div class="form-row">
  <!-- ДЛИНА ПРОВОДА -->
  <div class="form-group">
    <label class="form-label"
for="WireLength">
      Длина провода (метры) <span
class="required-marker">*</span>
    </label>
    <div style="position: relative;">
      <input type="number" id="WireLength"
name="WireLength"
      class="form-control-custom"
      min="1" max="10000" required
      placeholder="Введите длину от 1
до 10000 метров"
      value="100">
      <div style="position: absolute; right:
15px; top: 50%; transform: translateY(-50%); color: #7f8c8d;">
        М
      </div>
    </div>
  </div>
</div>

<!-- РАЗДЕЛ: ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ -->
<div class="form-section">
  <h3 class="form-section-title">
    <i class="fas fa-sticky-note"></i>
Дополнительная информация
  </h3>

  <div class="form-group">
    <label class="form-label" for="Notes">
      Примечания
      <span class="tooltip">
        <i class="fas fa-question-circle
tooltip-icon"></i>
        <span class="tooltip-text">
          Дополнительная информация о
заказе.
          Необязательное поле.
        </span>
      </span>
    </label>
    <textarea id="Notes" name="Notes" class="form-
control-custom"
      rows="4" placeholder="Дополнительная
информация о расчёте (необязательно)..."></textarea>
    <span class="form-hint">Максимум 500
символов</span>
    <div style="text-align: right; margin-top:
5px;">

```

```

        <small id="charCount">0/500</small>
      </div>
    </div>
  </div>

  <!-- БЛОК ПРЕДВАРИТЕЛЬНОГО РАСЧЁТА -->
  <div class="preview-section" id="previewSection">
    <div class="preview-title">
      <i class="fas fa-eye"></i> Предварительный
расчёт
    </div>
    <div class="preview-result" id="previewCost">
      0 руб.
    </div>
    <div style="text-align: center; margin-top: 15px;
color: #7f8c8d; font-size: 0.9rem;">
      <i class="fas fa-info-circle"></i> Это
предварительная оценка. Точный расчёт будет выполнен после
отправки формы.
    </div>
  </div>

  <!-- КНОПКИ УПРАВЛЕНИЯ -->
  <div class="button-group">
    <button type="submit" class="btn-custom btn-
calculate animate-pulse">
      <i class="fas fa-calculator"></i> Рассчитать
стоимость
    </button>
    <a href="@Url.Action("Index", "Calculation")"
class="btn-custom btn-back">
      <i class="fas fa-arrow-left"></i> Назад
    </a>
  </div>
</form>
</div>
</div>

<script>
  const requiredFields = ['WorkTypeId', 'SectionTypeId',
'WireLength'];
  let filledFields = 0;

  function updateFormProgress() {
    filledFields = 0;
    requiredFields.forEach(fieldId => {
      const field = document.getElementById(fieldId);
      if (field && field.value && field.value.trim() !== '')
{
        filledFields++;
      }
    });
  }

```

```

    const progressPercent = Math.round((filledFields /
requiredFields.length) * 100);

    document.getElementById('filledFields').textContent =
filledFields;
    document.getElementById('totalFields').textContent =
requiredFields.length;
    document.getElementById('progressPercent').textContent =
progressPercent + '%';
    document.getElementById('progressFill').style.width =
progressPercent + '%';
  }

  const notesTextarea = document.getElementById('Notes');
  const charCount = document.getElementById('charCount');

  if (notesTextarea && charCount) {
    notesTextarea.addEventListener('input', function() {
      const length = this.value.length;
      charCount.textContent = length + '/500';

      if (length > 500) {
        charCount.style.color = '#e74c3c';
        this.value = this.value.substring(0, 500);
      } else if (length > 450) {
        charCount.style.color = '#f39c12';
      } else {
        charCount.style.color = '#7f8c8d';
      }
    });
  }

  requiredFields.forEach(fieldId => {
    const field = document.getElementById(fieldId);
    if (field) {
      field.addEventListener('change', updateFormProgress);
      field.addEventListener('input', updateFormProgress);
    }
  });

  function calculatePreview() {
    const workType = document.getElementById('WorkTypeId');
    const sectionType =
document.getElementById('SectionTypeId');
    const wireLengthInput =
document.getElementById('WireLength');

    const wireLength = parseInt(wireLengthInput.value) || 0;

    if (!workType || !sectionType || !wireLengthInput) {
      alert('Пожалуйста, заполните все обязательные поля');
      return;
    }
  }

```

```

        if (wireLength < 1 || wireLength > 10000) {
            alert('Длина провода должна быть от 1 до 10000
метров');
            wireLengthInput.focus();
            return;
        }

        if (!workType.value || !sectionType.value) {
            alert('Пожалуйста, выберите все обязательные
параметры');
            return;
        }

        const workTypeCost =
workType.options[workType.selectedIndex]?.dataset.cost || 0;
        const sectionTypeCost =
sectionType.options[sectionType.selectedIndex]?.dataset.cost || 0;

        const totalCost =
            parseInt(workTypeCost) +
            (parseFloat(sectionTypeCost) * wireLength);

        const formattedCost = totalCost.toLocaleString('ru-RU', {
            minimumFractionDigits: 2,
            maximumFractionDigits: 2
        });

        document.getElementById('previewCost').innerHTML = `
            <div style="font-size: 1.2rem; margin-bottom: 5px;
color: #7f8c8d;">Примерная стоимость:</div>
            <div style="font-size: 2.5rem; font-weight: 800;
color: #2ecc71;">${formattedCost} руб.</div>
        `;

        const previewSection =
document.getElementById('previewSection');
        previewSection.style.display = 'block';

        previewSection.scrollIntoView({ behavior: 'smooth', block:
'nearest' });

        const costElement =
document.getElementById('previewCost');
        costElement.style.animation = 'pulse 1s';
        setTimeout(() => {
            costElement.style.animation = '';
        }, 1000);
    }

    document.getElementById('primaryForm').addEventListener('submit',
function(event) {

```

```

let isValid = true;
let errorMessage = '';

requiredFields.forEach(fieldId => {
  const field = document.getElementById(fieldId);
  if (field && (!field.value || field.value.trim() ===
'')) {
    isValid = false;
    field.style.borderColor = '#e74c3c';
    field.style.boxShadow = '0 0 0 3px rgba(231, 76,
60, 0.2)';

    const fieldName = field.labels?.[0]?.textContent
|| 'поле';
    errorMessage += `• ${fieldName} не заполнено\n`;
  } else if (field && fieldId === 'WireLength') {
    const value = parseInt(field.value);
    if (isNaN(value) || value < 1 || value > 10000) {
      isValid = false;
      field.style.borderColor = '#e74c3c';
      field.style.boxShadow = '0 0 0 3px rgba(231,
76, 60, 0.2)';
      errorMessage += `• Длина провода должна быть
от 1 до 10000 метров\n`;
    }
  }
});

if (!isValid) {
  event.preventDefault();
  alert('Пожалуйста, исправьте следующие ошибки:\n\n' +
errorMessage);

  setTimeout(() => {
    requiredFields.forEach(fieldId => {
      const field =
document.getElementById(fieldId);
      if (field) {
        field.style.borderColor = '';
        field.style.boxShadow = '';
      }
    });
  }, 3000);
}

});

document.addEventListener('DOMContentLoaded', function() {
  updateFormProgress();

  const firstField = document.getElementById('WorkTypeId');
  if (firstField) {
    setTimeout(() => {
      firstField.focus();
    });
  }
});

```

```

        }, 500);
    }

    const elements = document.querySelectorAll('.form-
container, .info-container');
    elements.forEach((element, index) => {
        element.style.opacity = '0';
        element.style.transform = 'translateY(20px)';

        setTimeout(() => {
            element.style.transition = 'opacity 0.5s ease,
transform 0.5s ease';
            element.style.opacity = '1';
            element.style.transform = 'translateY(0)';
        }, 100 * index);
    });
});
</script>

```

Приложение Г
(рекомендуемое)
Листинг модульных тестов

```

namespace Diplom.Tests
{
    public class CalculationServiceTests : IDisposable
    {
        private readonly ApplicationDbContext _context;
        private readonly CalculationService _service;
        private readonly ITestOutputHelper _output;

        public CalculationServiceTests(ITestOutputHelper output)
        {
            Console.OutputEncoding = Encoding.UTF8;
            _output = output;
            var options = new
DbContextOptionsBuilder<ApplicationDbContext>()
                .UseInMemoryDatabase(databaseName:
Guid.NewGuid().ToString())
                .Options;
            _context = new ApplicationDbContext(options);
            SeedData();
            _service = new CalculationService(_context);
        }

        private void SeedData()
        {
            _context.WorkTypes.Add(new WorkType
            {
                Id = 1,
                Name = "Воздушная линия",
                Description = "Прокладка воздушной линии",
                BaseCost = 5000
            });

            _context.WireTypes.Add(new WireType
            {
                Id = 1,
                Name = "Изолированный",
                Description = "Изолированный провод",
                CostPerMeter = 150
            });

            _context.TerrainTypes.Add(new TerrainType
            {
                Id = 1,
                Name = "Город",
                Description = "Городская местность",
                WorkCost = 1000
            });

            _context.SectionTypes.Add(new SectionType
            {
                Id = 1,
                Name = "16 мм²",
                Description = "Промышленное оборудование",

```

```

        AdditionalCost = 350
    });

    _context.SupportTypes.Add(new SupportType
    {
        Id = 1,
        Name = "Деревянная",
        Description = "Деревянная опора",
        InstallationCost = 15000
    });

    _context.Teams.Add(new Team
    {
        Id = 1,
        Name = "Бригада №1",
        WorkerCount = 5
    });

    _context.Clients.Add(new Client
    {
        Id = 1,
        FullName = "Иванов И.И.",
        Phone = "+7 (123) 456-78-90",
        Email = "ivanov@example.com",
        DiscountPercentage = 10
    });

    _context.SaveChanges();
}

[Fact]
public async Task
CalculateDetailedAsync_ValidInput_ReturnsCorrectTotalCost()
{
    Console.WriteLine("Тест 1: Проверка корректного
расчёта при заполнении всех полей");
    var calculation = new Calculation
    {
        WorkTypeId = 1,
        WireTypeId = 1,
        TerrainTypeId = 1,
        SectionTypeId = 1,
        SupportTypeId = 1,
        TeamId = 1,
        WireLength = 100
    };

    var result = await
_service.CalculateDetailedAsync(calculation);
    decimal expected = ((5000 + (150 * 100) + 1000 + 350 +
15000 + (5 * 100000)) * 1.2m);
    Assert.Equal(expected, result.TotalCost);
}

```

```

        Console.WriteLine($"    Результат: {result.TotalCost}
руб., ожидалось: {expected} руб. - успешно");
    }

    [Fact]
    public async Task
CalculateDetailedAsync_WithClientDiscount_AppliesDiscount()
    {
        Console.WriteLine("Тест 2: Проверка применения скидки
клиента");
        var calculation = new Calculation
        {
            WorkTypeId = 1,
            WireTypeId = 1,
            TerrainTypeId = 1,
            SectionTypeId = 1,
            SupportTypeId = 1,
            TeamId = 1,
            WireLength = 100,
            ClientId = 1
        };

        var result = await
_service.CalculateDetailedAsync(calculation);
        decimal beforeDiscount = ((5000 + (150 * 100) + 1000 +
350 + 15000 + (5 * 100000)) * 1.2m);
        decimal expected = beforeDiscount * 0.9m;
        Assert.Equal(expected, result.TotalCost);
        Console.WriteLine($"    Итоговая стоимость:
{result.TotalCost} руб. (скидка 10% применена)");
    }

    [Fact]
    public async Task
CalculateDetailedAsync_WithoutOptionalFields_UsesDefaults()
    {
        Console.WriteLine("Тест 3: Проверка обработки
отсутствия необязательных полей");
        var calculation = new Calculation
        {
            WorkTypeId = 1,
            WireTypeId = 1,
            TerrainTypeId = 1,
            SectionTypeId = 1,
            WireLength = 100
        };

        var result = await
_service.CalculateDetailedAsync(calculation);
        decimal expected = ((5000 + (150 * 100) + 1000 + 350 +
0 + 0) * 1.2m);
        Assert.Equal(expected, result.TotalCost);
    }

```

```

        Console.WriteLine($" Стоимость без учёта опор и
бригады: {result.TotalCost} руб.");
    }

    [Fact]
    public async Task
CalculateDetailedAsync_TeamWithZeroWorkers_AddsNoTeamCost()
    {
        Console.WriteLine("Тест 4: Проверка, что бригада с
нулём рабочих не добавляет стоимость");
        _context.Teams.Add(new Team { Id = 99, Name = "Пустая
бригада", WorkerCount = 0 });
        _context.SaveChanges();

        var calculation = new Calculation
        {
            WorkTypeId = 1,
            WireTypeId = 1,
            TerrainTypeId = 1,
            SectionTypeId = 1,
            SupportTypeId = 1,
            TeamId = 99,
            WireLength = 100
        };

        var result = await
_service.CalculateDetailedAsync(calculation);
        decimal expected = ((5000 + (150 * 100) + 1000 + 350 +
15000 + 0) * 1.2m);
        Assert.Equal(expected, result.TotalCost);
        Console.WriteLine($" Стоимость при бригаде без
рабочих: {result.TotalCost} руб.");
    }

    [Fact]
    public async Task
CalculateDetailedAsync_WithMissingWorkType_UsesDefault()
    {
        Console.WriteLine("Тест 5: Проверка, что при
отсутствии типа работ используется базовая стоимость 0");
        var calculation = new Calculation
        {
            WorkTypeId = 999, // несуществующий ID
            WireTypeId = 1,
            TerrainTypeId = 1,
            SectionTypeId = 1,
            WireLength = 100
        };

        var result = await
_service.CalculateDetailedAsync(calculation);
        decimal expected = ((0 + (150 * 100) + 1000 + 350 + 0
+ 0) * 1.2m);

```

```

        Assert.Equal(expected, result.TotalCost);
        Console.WriteLine($" Стоимость с несуществующим типом
работ: {result.TotalCost} руб.");
    }

    [Fact]
    public async Task
CalculateDetailedAsync_WithMissingWireType_UsesDefault()
    {
        Console.WriteLine("Тест 6: Проверка, что при
отсутствии типа провода стоимость материала 0");
        var calculation = new Calculation
        {
            WorkTypeId = 1,
            WireTypeId = 999,
            TerrainTypeId = 1,
            SectionTypeId = 1,
            WireLength = 100
        };

        var result = await
_service.CalculateDetailedAsync(calculation);
        decimal expected = ((5000 + (0 * 100) + 1000 + 350 + 0
+ 0) * 1.2m);
        Assert.Equal(expected, result.TotalCost);
        Console.WriteLine($" Стоимость с несуществующим типом
провода: {result.TotalCost} руб.");
    }

    [Fact]
    public async Task
CalculateDetailedAsync_WithMissingTerrainType_UsesDefault()
    {
        Console.WriteLine("Тест 7: Проверка, что при
отсутствии типа местности стоимость terrainCost = 0");
        var calculation = new Calculation
        {
            WorkTypeId = 1,
            WireTypeId = 1,
            TerrainTypeId = 999,
            SectionTypeId = 1,
            WireLength = 100
        };

        var result = await
_service.CalculateDetailedAsync(calculation);
        decimal expected = ((5000 + (150 * 100) + 0 + 350 + 0
+ 0) * 1.2m);
        Assert.Equal(expected, result.TotalCost);
        Console.WriteLine($" Стоимость с несуществующим типом
местности: {result.TotalCost} руб.");
    }

```

```

        [Fact]
        public async Task
        CalculateDetailedAsync_WithMissingSectionType_UsesDefault()
        {
            Console.WriteLine("Тест 8: Проверка, что при
отсутствии типа сечения additionalCost = 0");
            var calculation = new Calculation
            {
                WorkTypeId = 1,
                WireTypeId = 1,
                TerrainTypeId = 1,
                SectionTypeId = 999,
                WireLength = 100
            };

            var result = await
            _service.CalculateDetailedAsync(calculation);
            decimal expected = ((5000 + (150 * 100) + 1000 + 0 + 0
+ 0) * 1.2m);
            Assert.Equal(expected, result.TotalCost);
            Console.WriteLine($" Стоимость с несуществующим типом
сечения: {result.TotalCost} руб.");
        }
        public void Dispose()
        {
            _context.Database.EnsureDeleted();
            _context.Dispose();
        }
    }
}

```

Приложение Д
(рекомендуемое)
Результаты тестирования

Итоговая стоимость работ

25 020,00 руб.

Детальный расчёт • 15.05.2026 00:48

Параметры расчёта

Тип расчёта

Детальный

Дата расчёта

15.05.2026 00:48

Тип работ

Трансформатор

Тип провода

Изолированный

Длина провода

100 м

Тип местности

Болота

Сечение провода

16 мм²

Тип опор

Не указано

Клиент

Не указано

Бригада

Не указано

На главную

К истории

Печать

Рисунок Д.1 – Успешный расчёт

Итоговая стоимость работ

18 600,00 руб.

Детальный расчёт • 15.05.2026 00:54

Параметры расчёта

Тип расчёта

Детальный

Дата расчёта

15.05.2026 00:54

Тип работ

Кабельная линия

Тип провода

Неизолированный

Длина провода

100 м

Тип местности

Леса

Сечение провода

25 мм²

Тип опор

Не указано

Клиент

Не указано

Бригада

Не указано

На главную

К истории

Печать

Рисунок Д.2 – Успешный расчёт

Детальный расчёт стоимости работ

Тип работ *	Тип опор
<div>Воздушная линия - 5 000 руб. ▾</div>	<div>-- Не выбрано -- ▾</div>
Тип провода *	Длина провода (м) *
<div>Изолированный - 150,00 руб./м ▾</div>	<div>-100 ▴ ▾</div>
Тип местности *	Клиент
<div>Леса - 1 000,00 руб. ▾</div>	<div>-- Не выбрано -- ▾</div>
Сечение провода *	Бригада
<div>16 мм² - Промышленное оборудование - 350,00 руб./ме ▾</div>	<div>-- Не выбрано -- ▾</div>
Примечания	
<div>Дополнительная информация...</div>	

Рассчитать стоимость

Назад

Рисунок Д.3 – Результат тестирования класса эквивалентности «Длина меньше 1»

Детальный расчёт стоимости работ

Тип работ *	Тип опор
Трансформатор - 3 000 руб. ▼	-- Не выбрано -- ▼
Тип провода *	Длина провода (м) *
Неизолированный - 100,00 руб./м ▼	-100 ▼
Тип местности *	Клиент
Леса - 1 000,00 руб. ▼	-- Не выбрано -- ▼
Сечение провода *	Бригада
25 мм ² - Магистральные линии - 500,00 руб./метр ▼	-- Не выбрано -- ▼
Примечания	
Дополнительная информация...	

Рассчитать стоимость

Назад

Рисунок Д.4 – Результат тестирования класса эквивалентности «Длина меньше 1»

Детальный расчёт стоимости работ

Тип работ *	Тип опор
Кабельная линия - 4 000 руб. ▼	-- Не выбрано -- ▼
Тип провода *	Длина провода (м) *
Изолированный - 150,00 руб./м ▼	-1 ▲▼
Тип местности *	Клиент ! Значение должно быть больше или равно 1.
Луга - 1 500,00 руб. ▼	-- Не выбрано -- ▼
Сечение провода *	Бригада
16 мм ² - Промышленное оборудование - 350,00 руб./ме ▼	-- Не выбрано -- ▼
Примечания	
Дополнительная информация...	

Рассчитать стоимость

Назад

Рисунок Д.5 – Результат тестирования класса эквивалентности «Длина меньше 1»

Детальный расчёт стоимости работ

Тип работ *	Тип опор
Кабельная линия - 4 000 руб. ▼	-- Не выбрано -- ▼
Тип провода *	Длина провода (м) *
Неизолированный - 100,00 руб./м ▼	-675 ▴ ▾
Тип местности *	Клиент
Болота - 2 500,00 руб. ▼	-- Не выбрано -- ▼
Сечение провода *	Бригада
25 мм ² - Магистральные линии - 500,00 руб./метр ▼	-- Не выбрано -- ▼
Примечания	
Дополнительная информация...	

[Рассчитать стоимость](#)[Назад](#)

Рисунок Д.6 – Результат тестирования класса эквивалентности «Длина меньше 1»

Детальный расчёт стоимости работ

Тип работ *	Тип опор
Трансформатор - 3 000 руб. ▼	-- Не выбрано -- ▼
Тип провода *	Длина провода (м) *
Изолированный - 150,00 руб./м ▼	1000000 ▲▼
Тип местности *	Клиент *
Болота - 2 500,00 руб. ▼	-- Не выбрано -- ▼
Сечение провода *	Бригада
16 мм ² - Промышленное оборудование - 350,00 руб./м ▼	-- Не выбрано -- ▼
Примечания	
Дополнительная информация...	

Рассчитать стоимость

Назад

Рисунок Д.7 – Результат тестирования класса эквивалентности «Длина больше 10000»

Детальный расчёт стоимости работ

Тип работ *	Тип опор
Кабельная линия - 4 000 руб. ▼	-- Не выбрано -- ▼
Тип провода *	Длина провода (м) *
Изолированный - 150,00 руб./м ▼	745745645 ▲▼
Тип местности *	Клиент *
Леса - 1 000,00 руб. ▼	-- Не выбрано -- ▼
Сечение провода *	Бригада
16 мм ² - Промышленное оборудование - 350,00 руб./м ▼	-- Не выбрано -- ▼
Примечания	
Дополнительная информация...	

Рассчитать стоимость

Назад

Рисунок Д.8 – Результат тестирования класса эквивалентности «Длина больше 10000»

Детальный расчёт стоимости работ

Тип работ *	Тип опор
Воздушная линия - 5 000 руб. ▼	-- Не выбрано -- ▼
Тип провода *	Длина провода (м) *
Неизолированный - 100,00 руб./м ▼	67765 ▼
Тип местности *	Клиент
Луга - 1 500,00 руб. ▼	-- Не выбрано -- ▼
Сечение провода *	Бригада
25 мм² - Магистральные линии - 500,00 руб./метр ▼	-- Не выбрано -- ▼
Примечания	
Дополнительная информация...	

Рассчитать стоимость

Назад

Рисунок Д.9 – Результат тестирования класса эквивалентности «Длина больше 10000»

Итоговая стоимость работ

104 760,00 руб.

Детальный расчёт • 15.05.2026 01:02

Параметры расчёта

Тип расчёта

Детальный

Дата расчёта

15.05.2026 01:02

Тип работ

Кабельная линия

Тип провода

Изолированный

Длина провода

543 м

Тип местности

Луга

Сечение провода

16 мм²

Тип опор

Не указано

Клиент

Не указано

Бригада

Не указано

На главную

К истории

Печать

Рисунок Д.10 – Успешный расчёт

Итоговая стоимость работ

658 860,00 руб.

Детальный расчёт • 15.05.2026 01:03

Параметры расчёта

Тип расчёта

Детальный

Дата расчёта

15.05.2026 01:03

Тип работ

Кабельная линия

Тип провода

Неизолированный

Длина провода

5437 м

Тип местности

Леса

Сечение провода

16 мм²

Тип опор

Не указано

Клиент

Не указано

Бригада

Не указано

На главную

К истории

Печать

Рисунок Д.11 – Успешный расчёт

Итоговая стоимость работ

185 280,00 руб.

Детальный расчёт • 15.05.2026 01:03

Параметры расчёта

Тип расчёта

Детальный

Дата расчёта

15.05.2026 01:03

Тип работ

Воздушная линия

Тип провода

Изолированный

Длина провода

987 м

Тип местности

Леса

Сечение провода

16 мм²

Тип опор

Не указано

Клиент

Не указано

Бригада

Не указано

На главную

К истории

Печать

Рисунок Д.12 – Успешный расчёт

Итоговая стоимость работ

70 560,00 руб.

Детальный расчёт • 15.05.2026 01:05

Параметры расчёта

Тип расчёта

Детальный

Дата расчёта

15.05.2026 01:05

Тип работ

Трансформатор

Тип провода

Неизолированный

Длина провода

543 м

Тип местности

Леса

Сечение провода

25 мм²

Тип опор

Не указано

Клиент

Не указано

Бригада

Не указано

[На главную](#)[К истории](#)[Печать](#)

Рисунок Д.13 – Успешный расчёт

Приложение Е
(рекомендуемое)
Графическая часть

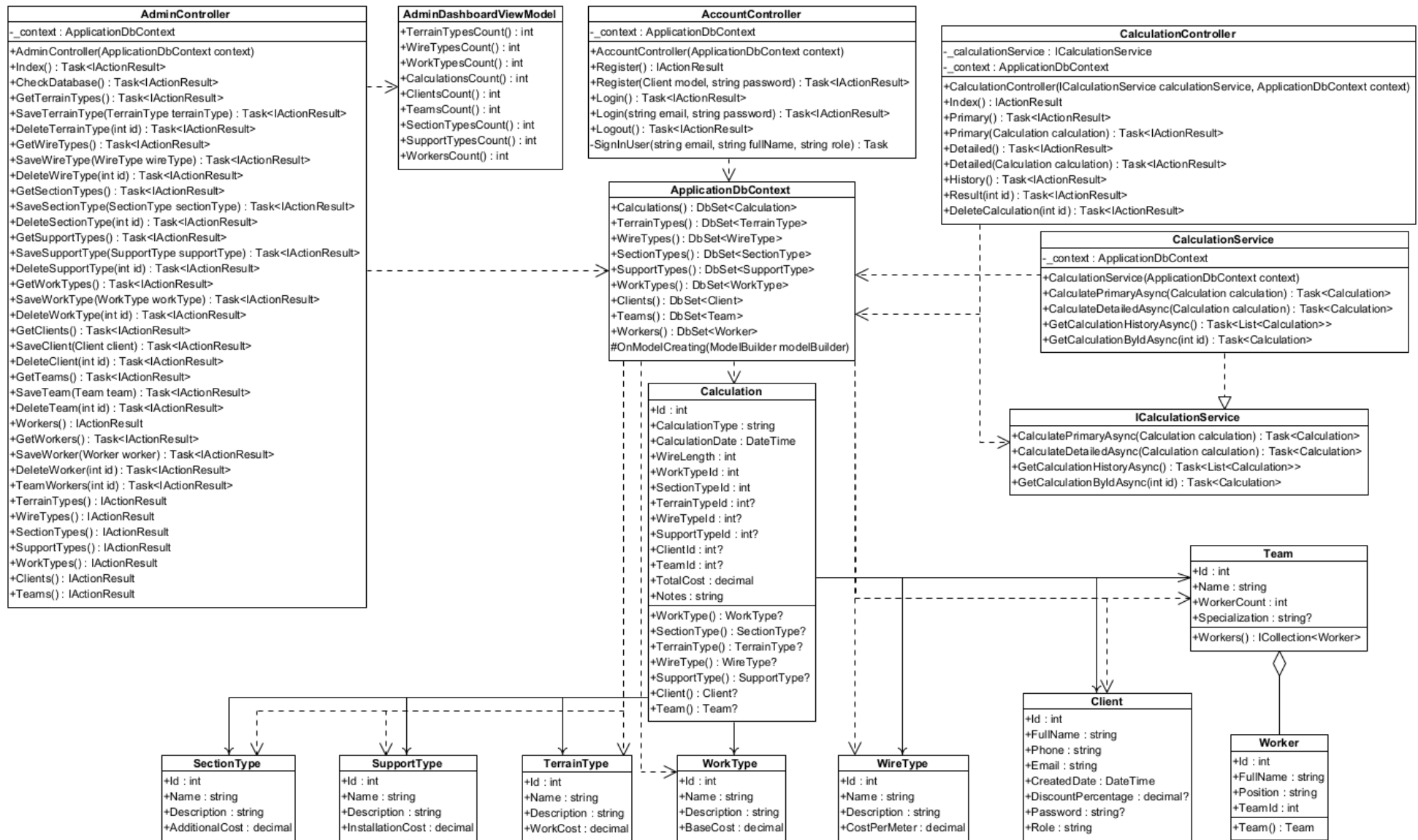


Рисунок Е.1 – Диаграмма классов приложения

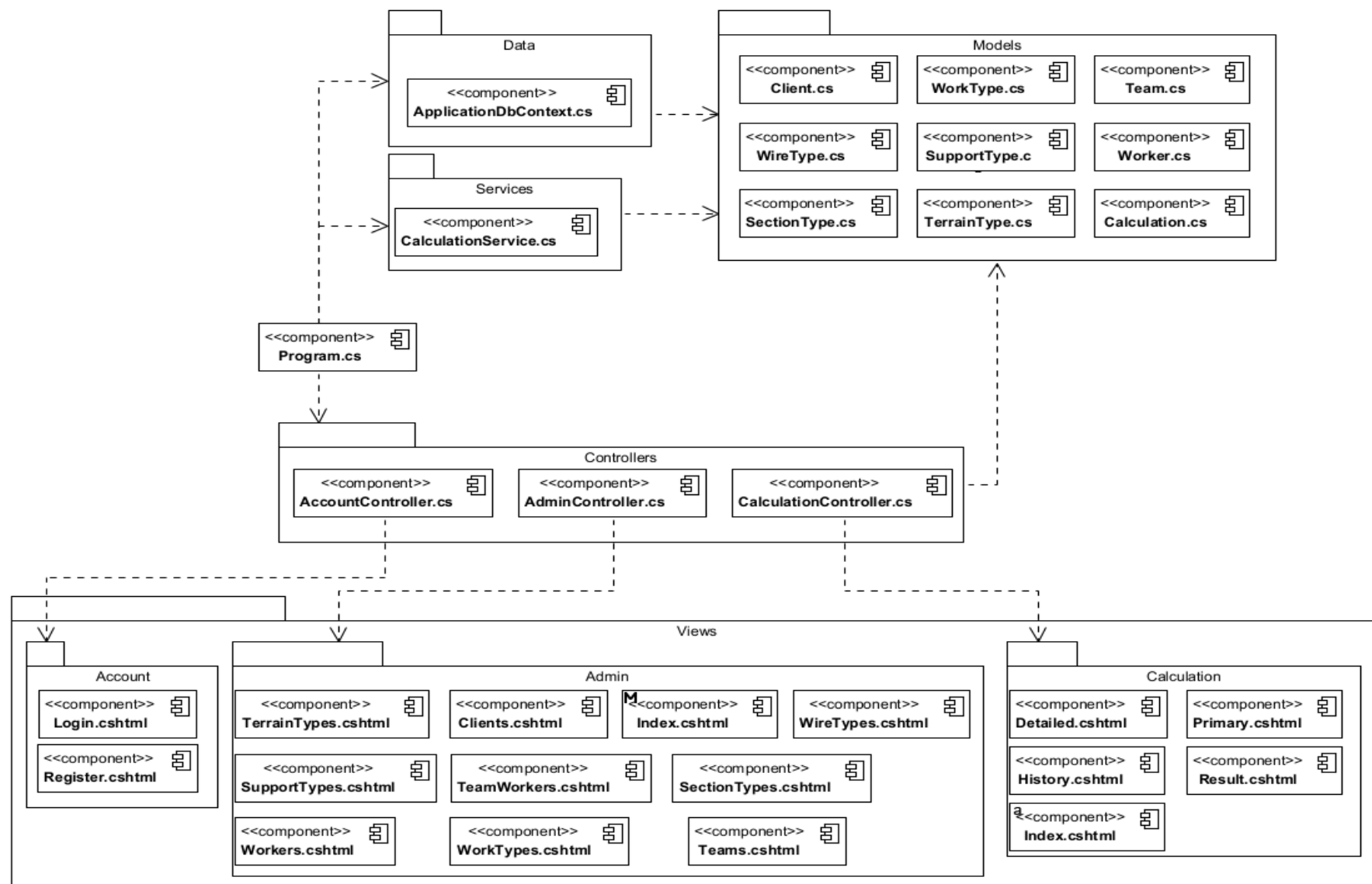


Рисунок Е.2 – Диаграмма компонентов приложения

Приложение Ж
(рекомендуемое)
Акт о внедрении

«Энергетика и Технологии»
Общество с ограниченной ответственностью

Пенза, ул. Ставского, стр. 4, оф. 22
ИНН/КПП 5836901381/583601001
ОГРН 1235800006177
р/с 40702810529170007234
ФИЛИАЛ «НИЖЕГОРОДСКИЙ»
АО «АЛЬФА-БАНК»
к/с 30101810200000000824
БИК 042202824
E-mail: enteh58@ya.ru
Тел.: +79063956275

25.04.2026

Акт о внедрении

Результаты теоретических и практических исследований, приведённые в выпускной квалификационной работе Верясова Дениса Евгеньевича на тему «Программные средства информационной поддержки планирования и организации работ в электроэнергетике», внедрены в практическую деятельность ООО «Энергетика и Технологии».

Генеральный директор
ООО «Энергетика и Технологии»



А.В. Решетников