|  |  |
| --- | --- |
|  | **Автономная некоммерческая образовательная организация**  **высшего образования Центросоюза Российской Федерации**  **«Сибирский университет потребительской кооперации»** |

Кафедра информатики

Работа допускается к защите

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.К. Черняков

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

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**(дипломная работа)**

**на тему: «ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ АВТОМАТИЗАЦИИ ПРОДАЖ КЛИМАТИЧЕСКОГО ОБОРУДОВАНИЯ (на материалах ООО «Инженерное Бюро», г. Тюмени)»**

обучающегося очной формы обучения

факультета экономики и управления

ТРОПИНА Даниила Александровича

Шифр ПИБ-О-01-21-009

Специальность 09.03.03

Прикладная информатика

Руководитель: преподаватель (практик) кафедры информатики

БЛОК Иван Николаевич

Новосибирск

2025

Дата защиты

Оценка после защиты \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Протокол заседания ГЭК № \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Секретарь ГЭК \_\_\_\_\_\_\_\_\_\_\_\_

Текст выпускной квалификационной работы

на \_\_\_\_\_\_\_\_ страницах

Приложение на \_\_\_\_\_\_ листах

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись обучающегося) (дата)*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись руководителя) (дата)*

**ТУТ БУДЕТ ЗАДАНИЕ 1 СТР**

**ТУТ БУДЕТ ЗАДАНИЕ 2 СТР**

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 12](#_Toc196212338)

[ГЛАВА 1. ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКАЯ ХАРАКТЕРИСТИКА ОРГАНИЗАЦИИ 15](#_Toc196212339)

[1.1. Общие сведения об организации 15](#_Toc196212340)

[1.2. Организационная структура 18](#_Toc196212341)

[1.3. Технологический процесс 19](#_Toc196212342)

[1.4. Мероприятия по реализации веб-приложения и введения в эксплуатацию 20](#_Toc196212343)

[ГЛАВА 2. ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЧЕСКОГО СТЕКА РАЗРАБОТКИ. 25](#_Toc196212344)

[2.1 Анализ существующих решений 25](#_Toc196212345)

[2.2 Выбор среды разработки 32](#_Toc196212346)

[2.3 Выбор системы управления базами данных 36](#_Toc196212347)

[ГЛАВА 3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ 41](#_Toc196212348)

[3.1 Проектирование веб-приложения 41](#_Toc196212349)

[3.2. Разработка веб-приложения 47](#_Toc196212350)

[3.3 Интерфейс веб-приложения 63](#_Toc196212351)

[3.4 Выбор хостинга и развертывание веб-приложения 67](#_Toc196212352)

[ЗАКЛЮЧЕНИЕ 72](#_Toc196212353)

# ВВЕДЕНИЕ

В современном технологичном мире интернет-коммуникация стала неотъемлемой частью бизнеса, значительно ускоряя ключевые процессы. Компании активно работают над сокращением дистанции между собой и клиентами, поскольку удобство и скорость совершения покупки напрямую влияют на лояльность платёжеспособной аудитории и, как следствие, на прибыль. Особенно это важно для малого бизнеса, где ресурсы ограничены, а гибкость и быстрая адаптация к изменениям рынка критически значимы.

Для повышения конкурентоспособности бизнесы внедряют веб-приложения с продуманной UX-логикой, где каждый элемент интерфейса соответствует ожиданиям целевой аудитории. Современный сайт может выполнять функции цифрового магазина: предоставлять актуальную информацию о компании, товарах и акциях, а также автоматизирует обработку данных. Такой подход устраняет необходимость очного взаимодействия, экономя время клиентов и сокращая операционные издержки бизнеса.

Для ООО «Инженерное Бюро» ручная обработка заказов стала критической проблемой, приводящей к снижению производительности, росту ошибок и ухудшению качества клиентского сервиса.

Прогнозируется, что разрабатываемое решение по автоматизации, подтвержденное аналитикой, проведенной в компании, позволит достичь следующих результатов:

- сокращение времени обработки заказов в 3 раза: сейчас на один заказ уходит 2 часа 45 минут, из которых 30 минут тратится на ручной ввод данных, 20 минут - на проверку, и 1 час 55 минут - на согласование через email. Автоматизация сократит время до 55 минут за счет интеграции с базами данных, автоматической проверки и уведомлений для согласования. По опыту компании «Технологии Сервиса», подобные системы сокращают время обработки с 3 часов до 1 часа;

- ошибки снизятся до 0,2% против текущих 5%: сейчас основные ошибки связаны с адресами (60%), ценами (25%) и дублированием заказов (15%). Автоматизация исключит их за счет валидации данных и уникальных идентификаторов для каждого заказа;

- клиенты получат прозрачность: личный кабинет с уведомлениями на каждом этапе повысит удовлетворенность на 25–30% . Например, клиенты смогут видеть статус заказа в реальном времени: «Принят», «Оплачен», «Отправлен»;

- трудозатраты сотрудников сократятся на 40%, освободив их от рутинных задач, таких как заполнение таблиц и переписка. Сейчас на это уходит 60% рабочего времени. Освободившееся время можно направить на увеличение продаж. Например, компания «Люкс Климат» после внедрения аналогичной системы повысила выручку на 15% в год.

В рамках данной выпускной квалификационной работы будет выполнена разработка веб-приложения для ООО «Инженерное Бюро» для автоматизации процесса оформления заказов. В ходе работы подробно будут разобраны различные стороны вопроса, начиная от исследования потребностей организации и выбора решений для автоматизации, заканчивая реализацией и тестированием готового продукта.

Цель данной выпускной квалификационной работы — спроектировать и разработать веб-приложение для компании, которое позволит автоматизировать бизнес-процессы электронной коммерции, улучшить коммуникацию с клиентами и оптимизировать работу сотрудников. В качестве базы для разработки была выбрана организация ООО «Инженерное Бюро», занимающееся вентиляцией и кондиционированием в помещениях и зданиях.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить бизнес-процессы компании и определить те из них, которые подлежат автоматизации.

2. Проанализировать существующие решения на рынке программного обеспечения.

3. Спроектировать и разработать программное обеспечение, которое позволит автоматизировать процедуру обработки заказов, согласно требованиям заказчика.

4. Подготовить сотрудников к работе с обновлённой системой.

# ГЛАВА 1. ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКАЯ ХАРАКТЕРИСТИКА ОРГАНИЗАЦИИ

## 1.1. Общие сведения об организации

«Инженерное Бюро» является обществом с ограниченной ответственностью. Организация основана 2012 году, компания специализируется на продаже, монтаже и поставки вентиляции и кондиционирования. Основной канал заработка организации, это участие в тендерных процедурах и заключению прямых договоров.

«Инженерное Бюро» предлагает комплексные услуги по монтажу и обслуживанию вентиляционного оборудования, а так же установку современных климатических систем для поддержания комфортного микроклимата в зданиях и помещениях различного назначения. Штат специалистов организации имеет богатый трудовой опыт работающих с различными видами оборудования, это позволяет гарантировать заказчику высокое качество предоставляемых услуг.

В перечень предоставляемых услуг входит:

*Обследование объекта и проектирование системы.* Инженеры проводят детальный анализ, определяя фактические показатели надежности, качества и эффективности работы систем. Специалисты так же используют инструментальное обследование на предмет определения физического износа систем. После проведения регламентных работ, на основе полученных данных формируют дефектные ведомости и технические отчеты с определением конструктивных решений и требуемых объемов работ, необходимых для дальнейшей разработки инженерами-проектировщиками проектно-сметной документации на выполнением необходимых работ.

*Монтаж и ввод в эксплуатацию.*Специалисты осуществляют установку всех необходимых компонентов системы в соответствии с проектом. После завершения монтажа специалисты проводят тестирование и настройку систем, при необходимости привлекая узконаправленных специалистов работающих с конкретным оборудованием предоставляющие сервисные услуги на гарантии. Зафиксировав корректную работу перед заказчиком, заканчивают работу на объекте.

*Техническое обслуживание и ремонт.*Компания так же предлагает техническое обслуживание вентиляционных систем, обеспечивая при этом бесперебойную работу и продлевая срок службы оборудования. В случае возникновения в установленное время контрактом неисправностей специалисты качественно проведут ремонт, используя необходимые запчасти и комплектующие. Так как компания долгое время работает в данном направлении, проблем с поставкой запчастей и комплектующих у организации нет.

*Модернизация и оптимизация.*Компания так же предлагает модернизацию или оптимизацию агрегатов - если система устарела или перестала соответствовать текущим требованиям. Данная услуга позволит повысить эффективность работы оборудования и улучшение качества воздуха в помещениях. После модернизации и оптимизации работы клиенты оставили много положительных отзывов об «Инженерном Бюро». Заказчики отметили, что компания помогла им сэкономить на электроэнергии и зарплатах сотрудников.

«Инженерное Бюро» предлагает широкий спектр вентиляционных решений, адаптивных под различные нужды заказчиков. В ассортименте компании представлены:

- приточные и вытяжные вентиляционные системы для обеспечения свежего воздуха и удаления загрязнённого;

- системы кондиционирования воздуха для поддержания комфортной температуры и уровня влажности;

- холодильные агрегаты, применяемые для охлаждения и нагревания жидких теплоносителей;

- вентиляционные решётки и диффузоры различных форм и размеров для эстетичного оформления выходов вентиляционных каналов;

- воздушные фильтры и очистители для улучшения качества воздуха и защиты от аллергенов и загрязнений;

- шумоглушители и звукоизоляционные материалы для снижения уровня шума от вентиляционного оборудования.

«Инженерное Бюро» имеет богатый опыт в своей области с успешной реализацией более пятисот проектов. Опыт внедрения решений на предприятиях:

- МАУК «Киноконцертный Театр «Космос».

- Областное государственное автономное учреждение «Многофункциональный Центр Предоставления Государственных и Муниципальных Услуг Челябинской Области».

- МКУ «Комитет по Информатизации Города Тюмени».

- Арбитражный Суд Свердловской обл.

- ГБУЗ Тюменской обл. «Областная Клиническая Больница № 1».

**Основные конкуренты компании:**

*ООО «Технологии Сервиса»*: большая компания с идентичным спектром услуг, хорошей производственной базой и большим охватом рынка по всей Западной и Южной России. Основное преимущество – слаженная и быстрая работа команды профессионалов и индивидуальный подход к клиенту.

*ООО «Люкс Климат»*: компания, использующая инновационные технологии и индивидуальный подход к каждому заказчику, что позволяет иметь постоянную базу клиентов.

Изучение конкурентов позволяет сделать вывод: чтобы оставаться конкурентоспособными, необходимо и дальше разрабатывать новаторские идеи, не отставая от трендов и оптимизировать производственные процессы.

Компания «Инженерное Бюро» гарантирует профессиональный монтаж и обслуживание климатических систем. Предлагая эффективные решения для создания комфортного микроклимата с гарантией надежной работы оборудования.

## 1.2. Организационная структура

«Инженерное Бюро» организовано по принципу линейной структуры управления, характеризующееся четким распределением полномочий и четкой иерархией. Организационная модель управления «Инженерное Бюро» включает в себя следующие управленческие элементы:

*- Правление:* руководит компанией в соответствии с законодательством Российской Федерации и несёт всю полноту ответственности за принимаемые решения, устанавливает векторное направление развитие бизнеса.

*- Административный отдел:* занимается сопровождением руководителя на бизнес-встречах и контролем исполнения поручений, взаимодействием с отделами для обеспечения 90% выполнения месячных планов, а также ведением журнала всей корреспонденции и внедрением инструментов, облегчающих управление организацией.

*- Тендерный отдел:* осуществляет анализ рынка, отбор рентабельных тендеров и подготовку полного пакета документации для участия, включая контроль регистраций и своевременного продления аккредитации на площадках госзакупок и коммерческих тендерных площадках, а также формирование необходимого комплекта документов (декларации, платёжные поручения, независимые гарантии, письма о деловой репутации и сопутствующие материалы).

*- Производственный отдел:* под руководством главного инженера осуществляет монтажные работы, взаимодействуя с административным отделом по вопросам закупок и обеспечения необходимыми материалами.

*- Отдел по работе с клиентами:* является новым структурным подразделением компании, создание которого предусмотрено в рамках выпускной квалификационной работы. Отдел по работе с клиентами будет заниматься выполнением заказов клиентов, взаимодействуя с производственным отделом для реализации технических заданий и осуществляя координацию проектов.

Организационная структура организации «Инженерное Бюро» представлена на Рисунке 1:

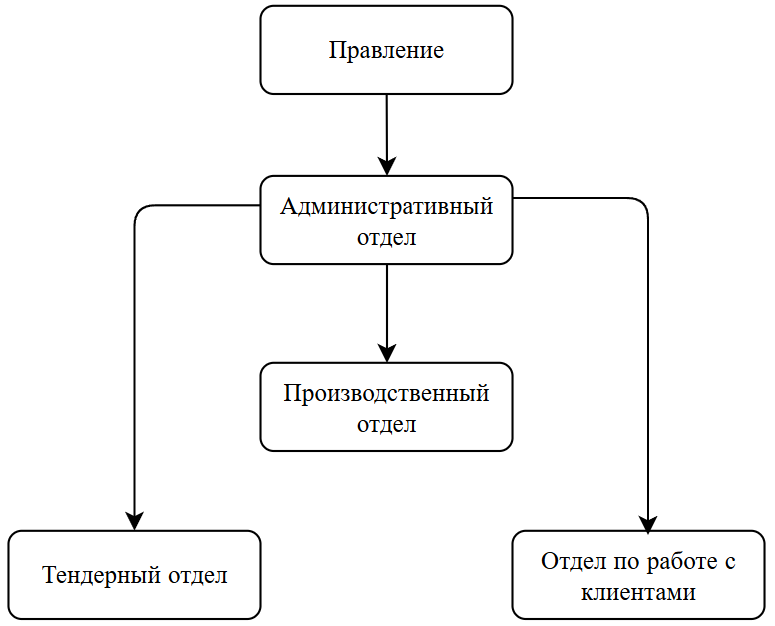


Рис. 1 - Организационная структура ООО «Инженерное Бюро»

## 1.3. Технологический процесс

В компании полностью отсутствует автоматизация внутренних процессов. В текущей работе заказы от клиентов поступают через личные встречи, телефонные звонки или электронную почту. Административный отдел вручную регистрирует эти заявки в CRM-системах *(Yougile/Контур Закупки),* что приводит к систематической потере информации о заказах. Такой подход создает несколько серьезных проблем: требует значительных временных затрат на обработку, увеличивает вероятность ошибок при записи данных клиентов или резервировании оборудования, а также вызывает задержки из-за необходимости постоянного согласования с директором и производственным отделом.

Основные недостатки существующей системы заключаются в следующем. Во-первых, ручной ввод данных в CRM не только снижает общую эффективность работы, но и существенно повышает риск ошибок. Во-вторых, чрезмерно централизованная система коммуникации, где все взаимодействия между отделами проходят через административный отдел, регулярно приводит к задержкам и недопониманию, что в целом негативно влияет на эффективность бизнес-процессов, качество обслуживания клиентов и общую производительность компании.

## 1.4. Мероприятия по реализации веб-приложения и введения в эксплуатацию

На основе представленного анализа текущей ситуации в компании был сделан вывод о необходимости разработки собственного веб-приложения для автоматизации этих процессов. Новый программный продукт позволит клиентам самостоятельно оформлять заказы онлайн. Система будет отображать актуальное наличие оборудования, значительно ускоряя процесс обработки заказов. Дополнительные преимущества включают возможность персонализированного продвижения акций через личные кабинеты клиентов, повышение общей прозрачности бизнес-процессов и рост клиентоориентированности компании. В результате внедрения организация сможет обрабатывать больше заказов, что положительно повлияет на увеличение прибыли.

Веб-приложение будет включать следующие функции:

*-* регистрация пользователя на сайте (в зависимости от объема заказов в месяц будет предоставляться индивидуальная скидка);

- оформление заказа: система регистрирует данные клиента и параметры заказа, сокращая необходимость ручного ввода информации и связанные с этим ошибки;

- просмотр статуса заказа: у клиента будет возможность отследить заказ в реальном времени.

Процесс функционирования системы показан на Рисунке 2.

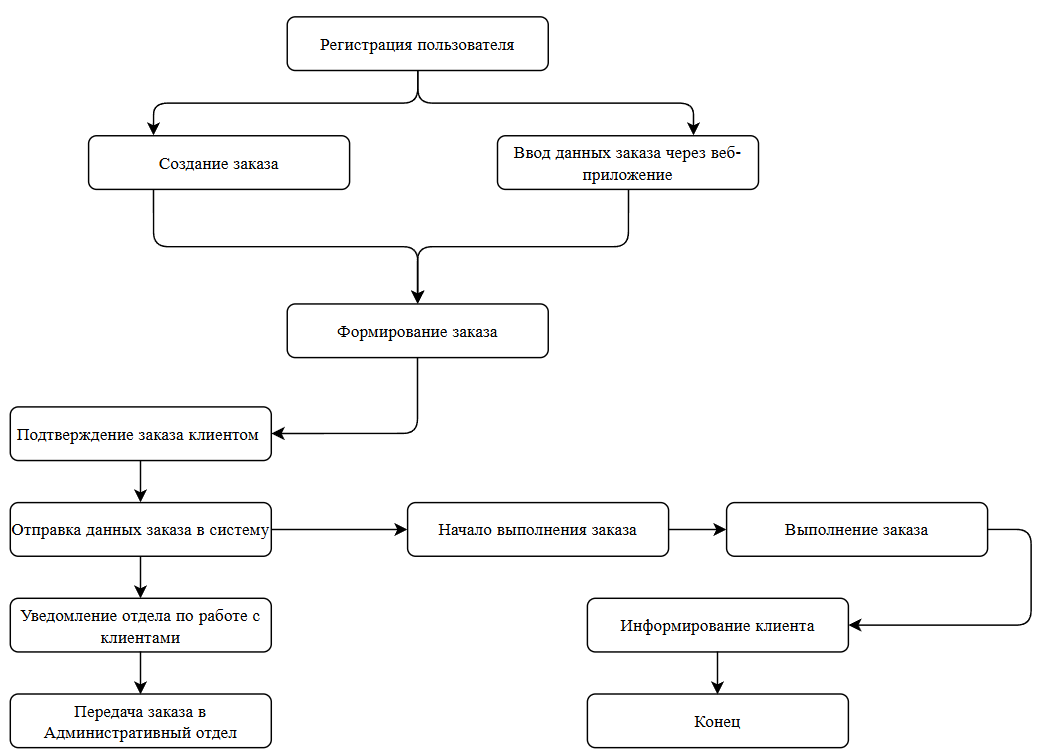


Рис. 2 - Динамические аспекты поведения системы

Выбор клиент-серверной архитектуры для реализации проектируемого приложения может быть обоснован тем, что данная модель отвечает всем необходимым требованиям, к примеру: централизованное управления, улучшенная безопасность, а так же возможность масштабирования (Рис. 3). Реализация распределенной системы на основе SQL-сервера обеспечит оптимальное распределение вычислительных ресурсов между клиентским приложением и центральным хранилищем данных. Централизованное хранение информации в SQL-сервере позволит поддерживать актуальность данных и обеспечить их согласованность во всех точках системы. SQL-сервер будет выполнять функции координатора взаимодействия клиентского приложения, обеспечивая эффективную обработку сложных запросов с использованием хранимых процедур и триггеров. Такой подход оптимизирует сетевое взаимодействие за счет минимизации объема передаваемых данных и исключения конфликтных ситуаций при одновременном доступе к данным.



Рис. 3 - Клиент-серверная архитектура

Клиентская часть представляет собой интерфейс, доступный через браузер, что позволяет пользователям выполнять все нужные действия независимо от их местоположения. Интерфейс включает главную страницу с каталогом продукции, а также удобную навигационную панель для быстрого перехода к детальной информации по каждому товару.

После достижения базовой готовности проект переходит на этап внедрения, который является ключевым для команды разработчиков. На этом этапе проверяется соответствие созданного продукта первоначальным требованиям заказчика. После успешного завершения внедрения система передается в эксплуатацию, где могут возникнуть новые задачи, связанные с развитием бизнес-процессов или изменениями в области технологий. На этом этапе продолжается работа над оптимизацией приложения, включая повышение производительности, исправление недочетов и добавление требуемых функций.

Процесс ввода системы в эксплуатацию включает несколько важных направлений:

*- Полное тестирование:* проводится всесторонняя проверка всех функциональных и нефункциональных аспектов системы для гарантии ее корректной работы в реальных условиях.

*- Обучение сотрудников:* организуются специальные тренинги и семинары для пользователей, которые будут работать с новым решением. Это помогает им быстро освоить все возможности системы и использовать ее максимально эффективно.

*- Подготовка документации:* разрабатываются подробные технические материалы, такие как руководства пользователя, инструкции администратора и справочные документы. Эти ресурсы становятся важным инструментом для последующего сопровождения системы.

*- Техническая поддержка:* в период активной эксплуатации организуется постоянное сопровождение системы, которое включает оперативное решение проблем, регулярное обновление программного обеспечения и консультирование пользователей.

Для передачи программного обеспечения составляется специальный план, включающий описание всей системы, список необходимых документов и рекомендации по подготовке пользователей. Также указываются возможные направления развития, которые могут быть актуальны в будущем.

Кроме рассмотренных планов, документация, согласно ГОСТ Р 51904-2002, должна включать ряд описаний.

Опытная эксплуатация проводится согласно заранее подготовленному плану, который устанавливает условия использования системы, продолжительность тестового периода и порядок устранения выявленных недочетов. Во время этого этапа ведется подробный журнал, куда заносятся все значимые события, включая отказы, корректировки и замечания пользователей. По результатам опытной эксплуатации принимается решение о готовности системы к финальным испытаниям. Эти испытания выполняются согласно утвержденному плану, который определяет объекты проверки, критерии успешного прохождения, методику проведения и ответственных лиц. По завершении испытаний оформляется официальный акт о принятии системы в эксплуатацию.

Приемочные испытания проводят в соответствии с программой, в который указывают:

1) Перечень объектов, выделенных в системе для испытаний и перечень требований, которым должны соответствовать объекты (со ссылкой на пункты ТЗ);

2) Критерии приемки приложения и его частей;

3) Условия и сроки проведения испытаний;

4) Средства для проведения испытаний;

5) Фамилии лиц, ответственных за проведение испытаний;

6) Методику испытаний и обработки их результатов;

7) Перечень оформляемой документации.

# ГЛАВА 2. ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЧЕСКОГО СТЕКА РАЗРАБОТКИ.

## 2.1 Анализ существующих решений

Сфера ритейла требует современных решений для эффективного взаимодействия с клиентами и управления проектами. Существует ряд специализированных программных продуктов, которые предлагают широкий функционал для этой отрасли, включая каталоги товаров и услуг, возможность онлайн-оформления заказов, интеграцию с внутренними системами управления, а также инструменты для продвижения услуг и аналитики.

В ходе исследования рынка были изучены следующие программные решения:

*1. Drupal* — это система управления контентом (CMS) с открытым исходным кодом, написанная на PHP. Она используется для создания сайтов различной сложности: от простых блогов до крупных корпоративных порталов, интернет-магазинов, социальных сетей и даже приложений. Drupal известен своей гибкостью и масштабируемостью.

При помощи Drupal можно создавать уникальные структуры сайтов, при помощи системы категорий и типов контента, уникальные поля дают возможность гибко адаптироваться к нестандартным требованиям заказчика, позволяя реализовывать уникальные идеи и задачи. Drupal в своём арсенале имеет огромное хранилище модулей (более 40 000), благодаря этому расширяется функциональность системы. При помощи модулей у пользователя появляется возможность интеграции с внешними сервисами, такими как CRM системы, платежные системы, аналитические инструменты и другие.

Drupal принято считать одной из безопасных CMS (Content Management System) — система управления контентом, платформа для создания и управления сайтами.

Разработчики регулярно выпускают обновления и работают над уязвимостью. В системе встроенные механизмы защиты атак (SQL- инъекции), благодаря такому авторитету на рынке государственные учреждения пользуются Drupal . Пропускная способность, отлично справляется с большим трафиком контента, что делает Drupal востребованным среди новостных порталов и социальных сетей. Drupal поддерживает создание многоязычных сайтов «из коробки», что делает его удобным для международных проектов. Регулярно проводятся конференции (DrupalCon), где обсуждаются новые тенденции и разработки (Рис. 4).



Рис. 4 - Интерфейс Drupal

*2. TYPO3* — это профессиональная система управления контентом (CMS) с открытым исходным кодом, которая широко используется для создания сложных, многофункциональных и масштабируемых веб-сайтов. Она особенно популярна среди крупных организаций, корпораций и разработчиков, которым требуется высокая гибкость, безопасность и возможность управления большими объемами контента. Следует подробнее рассмотреть особенности, преимущества, недостатки и сферы применения.

TYPO3 позволяет создавать сайты любой сложности – от крупных корпоративных порталов до мультисайтовых платформ (преимущество для крупных организаций с множеством проектов). Система поддерживает множество языков и позволяет создавать многоязычные сайты. TYPO3 поддерживает систему расширений, которая позволяет добавлять функции: интеграции с CRM и ERP-системы, электронной коммерции расширение (TYPO3 Commerce), сложные опросы и формы. Разработчикам предоставляется возможность создавать расширения самостоятельно, это приветствуется, так как появляется возможность создать уникальные проекты. TYPO3 известен своей высокой безопасностью. Регулярные обновления и строгая архитектура делают его подходящим для проектов, где безопасность данных критически важна (например, банки) (Рис.5).



Рис. 5 - Интерфейс TYPO3

*3. CRM-системы:* программные решения, предназначенные для автоматизации бизнес-процессов внутри компании. Ключевые функции таких систем включают стандартизацию и оптимизацию деятельности менеджеров, настройку схемы взаимодействия с клиентами, а также сбор и анализ данных для повышения эффективности бизнеса. В стандартный набор возможностей CRM-систем входят: настройка интерфейса под нужды компании, управление и обработка клиентской базы, конфигурация воронок продаж, инструменты для автоматизации задач и интеграция с другими платформами (Рис. 6).



Рис. 6 - CRM системы

Проведенный анализ показал, что готовые платформы (Drupal, TYPO3, стандартные CRM) не обеспечивают полного соответствия специфике работы ООО «Инженерное Бюро». В частности, Drupal и TYPO3, несмотря на свою гибкость, требуют значительных доработок для поддержки ключевых бизнес-процессов компании - например, интеграции производственного калькулятора для расчета стоимости систем вентиляции и кондиционирования, который планируется внедрить на этапе развития системы. Стандартные CRM системы, не поддерживают специализированные функции для инженерного оборудования: учет технических параметров (производительность, энергопотребление), привязку спецификаций и автоматизацию взаимодействия с производством, что является критическим ограничением для реализации будущей аналитики, т.к. готовые CRM системы не предоставляют достаточного уровня доступа к БД.

Разработка собственного решения обеспечит:

1. Полнофункциональную реализацию (включая калькулятор).

2. Специализированный учет оборудования (включая построение настраиваемых отчетов).

3. Оптимизацию взаимодействия между отделами.

4. Поэтапное развитие системы.

Данный подход сократит затраты лицензирование, сохраняя гибкость и полный контроль над развитием системы.

На текущий момент существует множество технологий для создания веб-приложений, при выборе из которых важно учитывать различные аспекты, которые могут повлиять на производительность, масштабируемость и удобство использования в проекте. Можно выделить такие технологии, как Blazor, Angular, ASP.Net MVC.

*Blazor* - это Фреймворк от Microsoft для создания интерактивных веб-приложений с использованием C# и .NET. Blazor позволяет разработчикам писать клиентскую и серверную логику на одном языке программирования - C#, что устраняет необходимость использования JavaScript для фронтенда. Blazor поддерживает два режима работы: Blazor WebAssembly (клиентская сторона) и Blazor Server (серверная сторона). Основные ключевые особенности являются: за счет компонентной архитектуры у разработчика есть возможность создавать и повторно использовать компоненты, минимальное использование JavaScript позволяется писать клиентскую логику на С#. Так же у разработчиков есть возможность писать frontend и backend на одном языке программирования C#. К недостаткам можно отнести: приложение может быть «тяжелым» по размеру из-за необходимости загрузки .NET runtime, так же ограниченная поддержка браузеров, поддерживается не всеми старыми браузерами.

*Angular* - это клиентский фреймворк для создания одностраничных приложений SPA *(Single Page Applications)* на JavaScript и TypeScript. Разработанный компанией Google, Angular активно используется для создания динамических, высокопроизводительных веб-приложений. Он предоставляет богатую экосистему инструментов и функций, которые упрощают разработку сложных приложений. Основные ключевые особенности являются: двусторонняя привязка данных упрощает синхронизацию данных между моделью и представлением, за счет компонентной архитектуры у разработчика есть возможность создавать и повторно использовать компоненты. Angular легко интегрируется с Progressive Web Apps. К недостаткам можно отнести: Angular имеет сложную архитектуру и требует больше времени для освоения, для полноценной работы Angular требуется JavaScript, что является большим недостатком для .NET разработчиков.

ASP.NET представляет собой современную платформу для создания интерактивных веб-приложений на языке C#. Ее ключевое преимущество - использование архитектуры MVC (Model-View-Controller), которая обеспечивает четкое разделение компонентов приложения.

- Model: управляет данными и бизнес-логикой;

- View: отвечает за представление данных;

- Controller: обрабатывает взаимодействие между Model и View

Такая структура значительно упрощает разработку, тестирование и поддержку кода, особенно в крупных проектах.

Важным преимуществом ASP.NET MVC является серверный рендеринг, который обеспечивает SEO-оптимизацию и быструю начальную загрузку страниц. Платформа полностью интегрирована с экосистемой .NET, включая Entity Framework для работы с базами данных, Dependency Injection для управления зависимостями, а также совместимость с такими технологиями Microsoft как Azure, SQL Server и Visual Studio.

ASP.NET MVC отличается высокой гибкостью и расширяемостью, поддерживая создание RESTful API, использование Razor для динамических HTML-страниц, а также предоставляя инструменты для маршрутизации, валидации и фильтрации. Платформа соответствует современным стандартам, включая поддержку HTTP/2, WebSockets, gRPC для высокопроизводительных API и SignalR для работы в реальном времени.

Сравнительный анализ технологий отображен в таблице 2.1.

*Таблица 2.1* **- Сравнение технологий для создания веб-приложения**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Критерий | ASP.NET MVC | Blazor | Angular |
| 1 | Тип лицензии | Открытая (MIT License) | Открытая (MIT License) | Открытая (MIT License) |
| 2 | Язык программирования | C# | C# | TypeScript/JavaScript |
| 3 | Появление на рынке | С 2009 года (ASP.NET MVC) | С 2018 года (Blazor) | С 2010 года (AngularJS), с 2016 года (Angular 2+) |
| 4 | Работа приложений | Серверный рендеринг (HTML генерируется на сервере) | Клиентский (WebAssembly) или серверный (Blazor Server) | Клиентский (SPA, выполняется в браузере) |
| 5 | Маршрутизация | Серверная маршрутизация (через  RouteConfig  или атрибуты  [Route]) | Клиентская маршрутизация через компонент  <Router> | Клиентская маршрутизация через модуль  RouterModule |
| 6 | Привязка данных | Односторонняя привязка через Razor-шаблоны | Двусторонняя привязка данных через  @bind | Двусторонняя привязка данных через  [(ngModel)] |

На основе проведенного анализа было принято решение об использовании ASP.NET MVC, которое подходит для разработки корпоративных веб-приложений с повышенными требованиями к безопасности и управляемости. Технология обеспечивает полноценный серверный рендеринг, гарантирующий SEO-совместимость и быструю загрузку страниц без дополнительных настроек.

Ключевые преимущества:

1. Полный контроль над HTTP-запросами — отсутствует необходимость в дополнительных клиент-серверных взаимодействиях, характерных для SPA-решений.

2. Встроенные механизмы безопасности через ASP.NET Identity, включая аутентификацию, управление ролями и поддержку OAuth — функциональность, которую в Angular и Blazor требуется реализовывать самостоятельно.

3. Строгая типизация C# на всех уровнях приложения исключает ошибки, характерные для динамической типизации JavaScript.

4. Готовые корпоративные функции: кэширование, валидация данных, локализация и фильтрация запросов работают без необходимости подключения дополнительных библиотек.

## 2.2 Выбор среды разработки

Разработка программного обеспечения начинается с выбора инструментов. Современные IDE (Integrated Development Environment – интегрированная среда разработки) представляют собой программные комплексы, объединяющие все необходимые инструменты для эффективной работы - от написания кода до его отладки и развертывания.   
При выборе IDE следует обратить внимание:

1. Цена - на рынке большое количество бесплатных решений с открытым исходным кодом, но доступных функций при таких условиях ограниченное количество.

2. Поддержка необходимой операционной системы и языков программирования – в долгосрочной перспективе необходимо рассматривать кроссплатформенные решения, т. к. при развитии ИТ инфраструктуры в компании технические специалисты могут работать на разных операционных системах и создавать проекты на разных языках программирования.

3. Возможность совместной работы – для работы с общим репозиторием в Git, этот момент так же важен для упрощения работы техническим специалистам и развитии данного направления в организации.

Далее представлен обзор некоторых популярных IDE, доступных в настоящее время для разработки программного обеспечения.

*JetBrains Rider* - это кроссплатформенная интегрированная среда разработки (IDE), созданная компанией JetBrains, которая специализируется на разработке приложений на .NET, .NET Core, ASP.NET, а также Unity-проектов (Рис. 7).

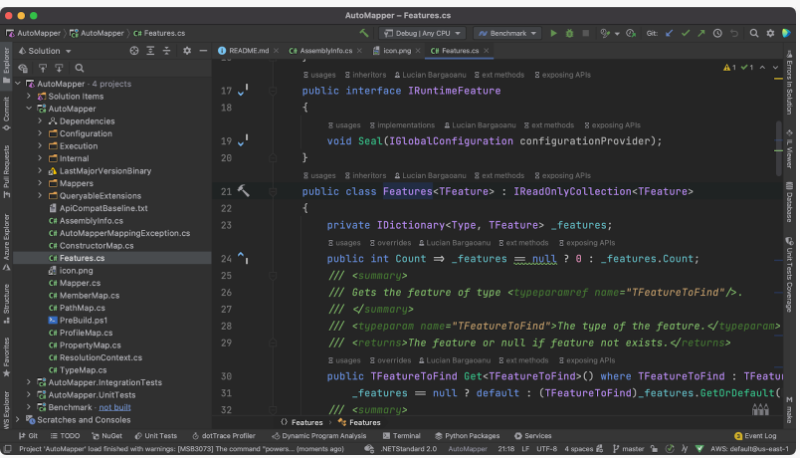


Рис. 7 - Интерфейс Jetbrains Rider

Rider сочетает в себе технологии IntelliJ IDEA и ReSharper, что делает его универсальным и востребованным инструментом для разработчиков. К основным особенностям можно отнести: производительность и скорость, запуск занимает секунды, а интерфейс остается отзывчивым даже при работе с крупными решениями, архитектура Rider разделяет UI и backend-процессы, что минимизирует ошибки в системе. Интеллектуальный анализ кода, поддерживает стили кода по стандартам Microsoft и гибкие настройки под команду. Расширенные возможности рефакторинга, более 60 вариантов рефакторинга, включая безопасное переименование, инлайн-методы и генерацию кода (например, автоматическое создание свойств из полей). К недостаткам можно отнести: отсутствие поддержки устаревших технологий Microsoft (например, WCF), а так же время, требуемое на освоение.

*Microsoft Visual Studio* — это интегрированная среда разработки (IDE) от Microsoft, которая подходит для работы с фреймворком ASP.NET MVC. Она предоставляет обширный набор инструментов для разработки веб-приложений на платформе .NET и поддерживает язык программирования C# и другие технологии, используемые в экосистеме .NET. К преимуществам можно отнести то, что версия «Community Edition» предлагает достаточно широкий функционал для бесплатного использования, хотя крупным организациям может потребоваться приобретение лицензии. Так же среди ключевых преимуществ стоит выделить: встроенные инструменты отладки и тестирования, которые значительно упрощают процесс разработки и помогают быстро находить и исправлять ошибки, прямую интеграцию с экосистемой Microsoft , включая GitHub , SQL Server и другие сервисы, что делает её идеальным выбором для проектов, связанных с облачными решениями или корпоративными системами. (Рис. 8).



Рис. 8 - Интерфейс Microsoft Visual Studio

*Visual Studio Code* — кроссплатформенный редактор кода, который при помощи плагинов можно превратить в подобие IDE. Кроссплатформенность позволяет использовать VS Code на разных операционных системах, таких как MacOS, Linux и Windows. Данный редактор кода является open-source решением и большинство плагинов доступны бесплатно, хотя существуют и платные расширения, которые предлагают дополнительный функционал. Также для того, чтобы приступить к работе не потребуется много ресурсов, так как он является достаточно легковесным. Несмотря на поддержку .NET-платформы, Visual Studio Code неудобен для сложных проектов. К минусам можно отнести сомнительную надежность, так как многие надстройки имеют низкое качество сборки и не всегда выполняют даже основные функции (Рис.9).

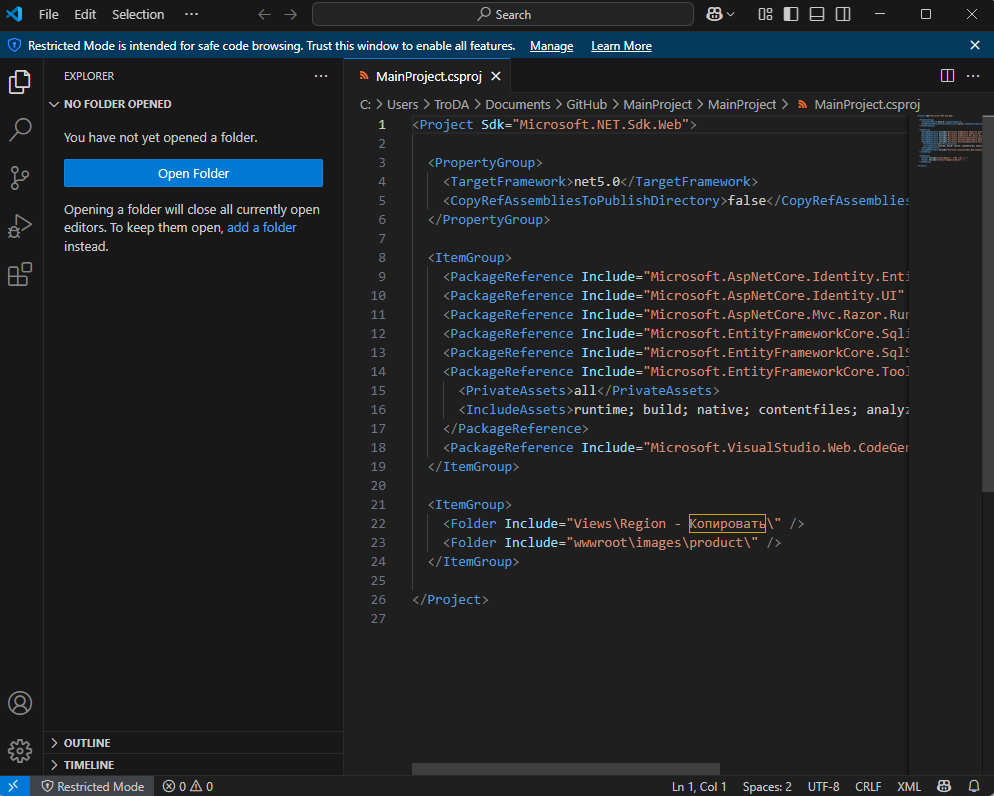


Рис. 9 - Интерфейс Visual Studio Code

На основе проведенного обзора было принято решение использовать Microsoft Visual Studio. Прежде всего, Visual Studio предлагает глубокую интеграцию с экосистемой .NET, что обеспечивает доступ ко всем необходимым инструментам для создания веб-приложений. Благодаря встроенной поддержке ASP.NET MVC, среда разработки позволяет эффективно реализовывать паттерн Model-View-Controller, гарантируя четкое разделение логики приложения, представлений и моделей данных. Это не только упрощает процесс разработки, но и способствует созданию масштабируемых и легко поддерживаемых решений.

Кроме того, Visual Studio предоставляет инструменты отладки, автодополнения кода и тестирования, которые значительно ускоряют разработку и помогают минимизировать количество ошибок. В сравнении с Visual Studio Code и JetBrains Rider, Microsoft Visual Studio предлагает более комплексное решение, особенно для сложных проектов, где требуется максимальная производительность и надежность.

Все эти преимущества делают Microsoft Visual Studio оптимальным выбором для реализации задач выпускной квалификационной работы, где важно максимально использовать потенциал ASP.NET MVC для создания качественного и функционального приложения.

## 2.3 Выбор системы управления базами данных

Эффективность взаимодействия пользователя с веб-приложением во многом определяется подходов к управлению данными. Если приложение не способно быстро обрабатывать и доставлять информацию, то даже самые совершенные элементы, такие как чистый код или привлекательный интерфейс, не будут иметь значения. Так же следует не забывать про защиту данных от несанкционированного доступа, критично важно предотвращать утечку данных злоумышленникам. Для достижения этих целей необходимо тщательней подойти к выбору системы управления базами данных.

*База данных –* это хранилище, где собирается и структурируется информация используемая приложением.

*Система управления базой данных (СУБД) –* это программное обеспечение (ПО), позволяющие эффективно управлять собранной, хранимой информацией в базе данных.

На рынке представлено большое количество решений СУБД, что делает выбор подходящей системы сложной задачей.

Базы данных можно разделить на два основных типа: *SQL и NoSQL.* Правильный выбор между ними играет ключевую роль в разработке веб-приложения. Оба типа обладают высокой производительностью, однако между ними существуют важные различия, которые необходимо учитывать.

Реляционная база данных представляет собой набор таблиц, связанных между собой определенными отношениями. Для работы с такой базой данных и выполнения запросов используется язык структурированных запросов (SQL). Данные в реляционных базах организованы в виде строк (кортежей) и столбцов (атрибутов). Каждая строка в таблице использует одинаковый набор атрибутов, что делает такие базы идеальными для хранения структурированной информации.

Примером реляционной СУБД является MS SQL Server. Для управления этой системой используется интегрированная среда SQL Server Management Studio (SSMS), которая предоставляет удобные инструменты для администрирования баз данных, создания запросов и оптимизации работы с данными. SSMS позволяет разработчикам эффективно управлять инфраструктурой SQL, обеспечивая высокую производительность и безопасность данных. Она имеет широкий спектр преимуществ в сравнении с альтернативными СУБД. Главными преимуществами является, MS SQL Server может быть установлен не только на Windows, но и на Linux, это расширяет возможности его использования, за счет легкой интеграция с облачными сервисами, предоставляется возможность выполнять резервное копирование и использовать гибридные сценарии, а так же выполнять миграцию данных в облако. Пользователь получает готовую к использованию систему «из коробки», что позволяет быстро начать работу без необходимости установки дополнительных инструментов или сложной настройки. MS SQL Server поддерживает разные форматы данных, такие как: JSON для работы с полуструктурированными данными. Геопространственные данные для работы с координатами и геопозициями. Графовые данные для обработки сложных связей между объектами. Возможность создания пользовательских типов данных, что позволяет адаптировать базу данных под конкретные задачи и устанавливать необходимые ограничения.

Максимальный объем оперативной памяти, который может использовать Express Edition, ограничен 1 ГБ, а количество процессорных ядер — 4. Эти ограничения делают её подходящей для небольших нагрузок, но при этом она сохраняет высокую производительность благодаря использованию того же движка, что и в платных версиях.

MS SQL Server соответствует требованиям ACID (атомарность, согласованность, изолированность, устойчивость), что гарантирует надежность и целостность данных даже в условиях высокой нагрузки. Это особенно важно для приложений, где критична точность и сохранность информации.

Хотя реляционная базы данных имеют множество преимуществ, они так же имеют и недостатки в виде ограничений. За счет ограниченной гибкости, затрудняется работа с полуструктурированными и неструктурированными данными, соответственно задачи с большими объемами данных и сложными аналитическими запросами становятся проблемой, следует также отметить, что при усложнении структуры данных взаимодействие между различными программными решениями может становиться проблемой.

Альтернативой традиционным реляционным базам выступают нереляционные базы данных, известные как NoSQL. Они предоставляют разработчикам больше свободы и масштабируемости, так как позволяют хранить и обрабатывать разнородные данные, такие как изображения, аудиофайлы и другие неформализованные типы информации. Важное преимущество таких систем — возможность динамически изменять структуру данных без необходимости модификации существующих записей. Более того, нереляционные базы данных часто организованы в распределенной архитектуре, что повышает их отказоустойчивость: даже при выходе из строя одного из узлов система продолжает функционировать. Однако стоит отметить, что эти технологии пока не достигли уровня зрелости реляционных решений и могут жертвовать соблюдением принципов ACID ради гибкости и производительности.

Для реализации и отладки веб-приложения по обработке заказов была выбрана реляционная база данных MS SQL Server Expess, имеющая лимит на размер базы данных – 10 гб на одну базу. Данные ограничения удовлетворяют требованиям небольшого проекта, тестирования продукта и обучения. При успешном вводе в эксплуатацию приложения и увеличении объема хранимых данных, есть возможность перейти на платные версии SQL Server (Standard или Enterprise) для увеличения места и мощности.

Работа с данными в реляционных базах осуществляется через SQL-запросы, хранимые процедуры и прочие инструменты СУБД. Однако в современных приложениях данные чаще всего представлены в виде объектов, что создает необходимость в преобразовании между реляционной моделью базы данных и объектной моделью программы. Для этого используется технология *ORM (Object-Relational Mapping)*, которая позволяет создавать «виртуальную объектную базу данных». На платформе C# существуют два популярных фреймворка для работы с базами данных: Entity Framework и Dapper . Проведем их сравнительный анализ:

*1. Поддержка баз данных***:** Entity Framework поддерживает множество СУБД, включая SQL Server, PostgreSQL, MySQL и SQLite. С другой стороны Dapper , будучи микро-ORM, работает практически со всеми базами данных, поддерживающими ADO.NET, но требует дополнительных библиотек для конкретных СУБД.

*2. Зрелость***:** Dapper был создан как легковесная ORM и остается относительно простым инструментом. Он не стремится к максимальной абстракции, что делает его идеальным выбором для проектов, где важна производительность. Entity Framework имеет долгую историю развития, начиная с версии .NET Framework, и постоянно совершенствуется. Современная версия Entity Framework Core использует опыт предыдущих релизов и предлагает современную архитектуру.

*3. Подходы к преобразованию данных:* Entity Framework предоставляет Code-First и Database-First подходы для работы с данными, а также Fluent API для конфигурации и генерирования схемы базы данных. Dapper не предлагает готовых решений для миграции или автоматической генерации схемы. Разработчикам нужно явно писать SQL-запросы и самостоятельно настраивать маппинг данных.

*4. Запросы:* Dapper требует написания явных SQL-запросов, но предоставляет удобные методы для их выполнения и маппинга результатов в объекты. Entity Framework поддерживает LINQ-запросы, что позволяет писать строго типизированные запросы к данным. Он также поддерживает выполнение чистых SQL-запросов и автоматическую конвертацию LINQ в SQL.

*5. Поддержка сообщества и экосистема:* Entity Framework является более популярной ORM-системой среди разработчиков C#. Dapper , хотя и менее распространен, активно используется в проектах, где важна производительность. Он имеет меньше учебных материалов, но пользуется спросом в определенных нишах.

На основе проведённого анализа было принято решение использовать Entity Framework Core в качестве ORM-технологии для обеспечения взаимодействия с базами данных в разрабатываемом веб-приложении. Данный выбор обусловлен современностью фреймворка, его гибкостью, развитой экосистемой и высоким уровнем удобства использования, что позволяет значительно упростить процесс разработки и сократить время на реализацию рутинных операций с данными.

# ГЛАВА 3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ

## 3.1 Проектирование веб-приложения

Для создания веб-приложения требуется разработать концептуальную модель данных в виде ER-диаграммы, которая поможет определить основные сущности и установить возможные связи между ними.

Концептуальная модель базы данных, построенная с использованием нотации Мартина, представлена на Рисунке 10.

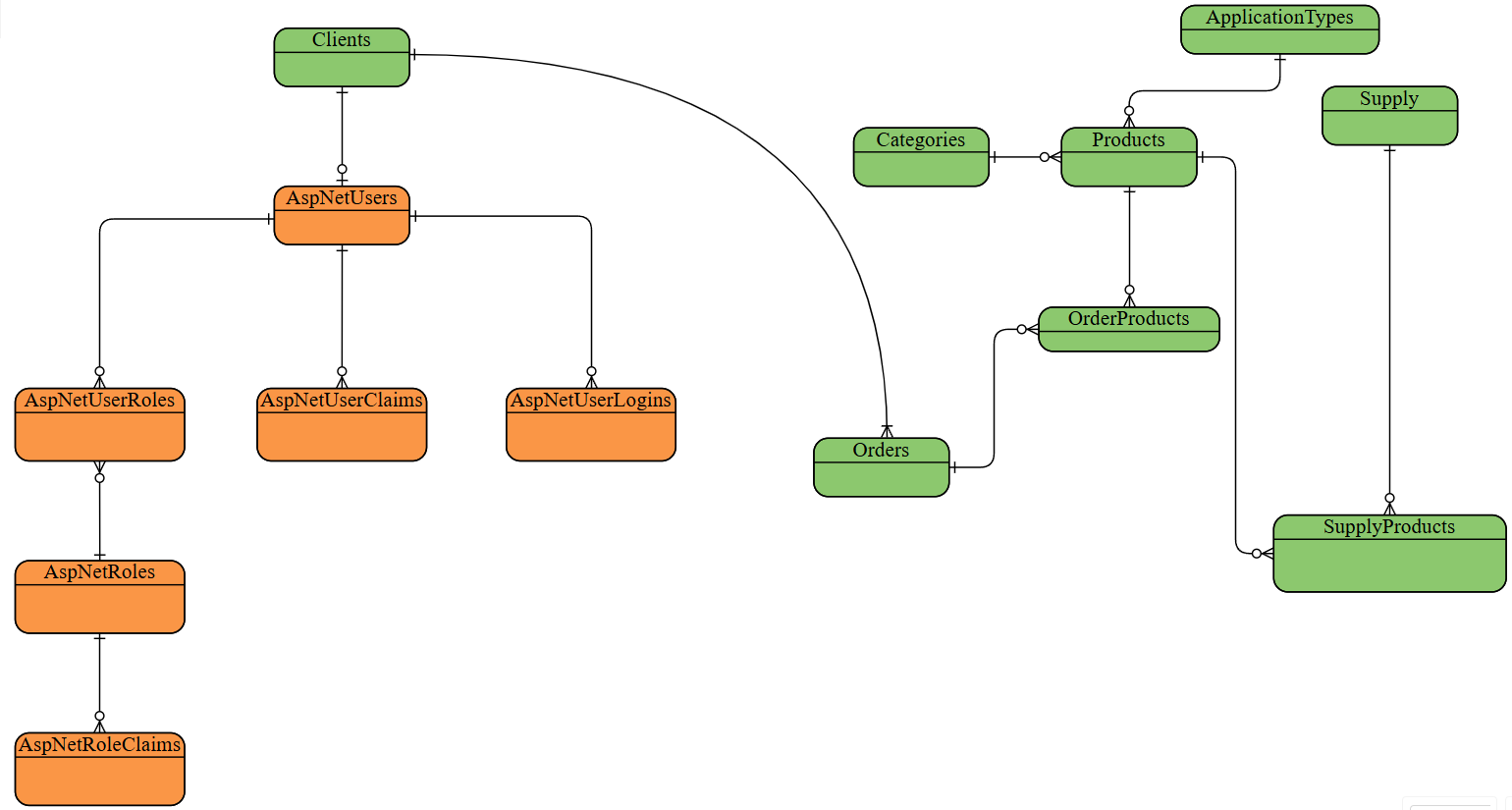


Рис. 10- Концептуальная диаграмма базы данных

Таблица 3.1 **- Пояснение к концептуальной диаграмме базы данных**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Тип компонента | Компонент | Описание |
| 1 | 2 | 3 | 4 |
| 1 | Авторизация ASP.NET Identity | AspNetUsers | Пользователи |
| AspNetUserRoles | Связи пользователей и ролей |
| AspNetRoles | Роли |
| AspNetUserClaims | Дополнительные утверждения пользователей |
| AspNetUserLogins | Варианты входа пользователей |
| AspNetUserLogins | Утверждения ролей |
| *«Окончание табл. 3.1»* | | | |
| 1 | 2 | 3 | 4 |
| 2 | Основные сущности | Products | Продукты |
| Clients | Клиенты |
| Supply | Поставка |
| ApplicationTypes | Типы продуктов |
| Categories | Категории |
| Orders | Заказы |

На основе концептуальной диаграммы необходимо разработать логическую модель базы данных. Эта модель будет отображать, какие данные хранятся в базе, включая объекты предметной области, их атрибуты и связи между ними. Структура логической модели представлена на Рисунке 11.

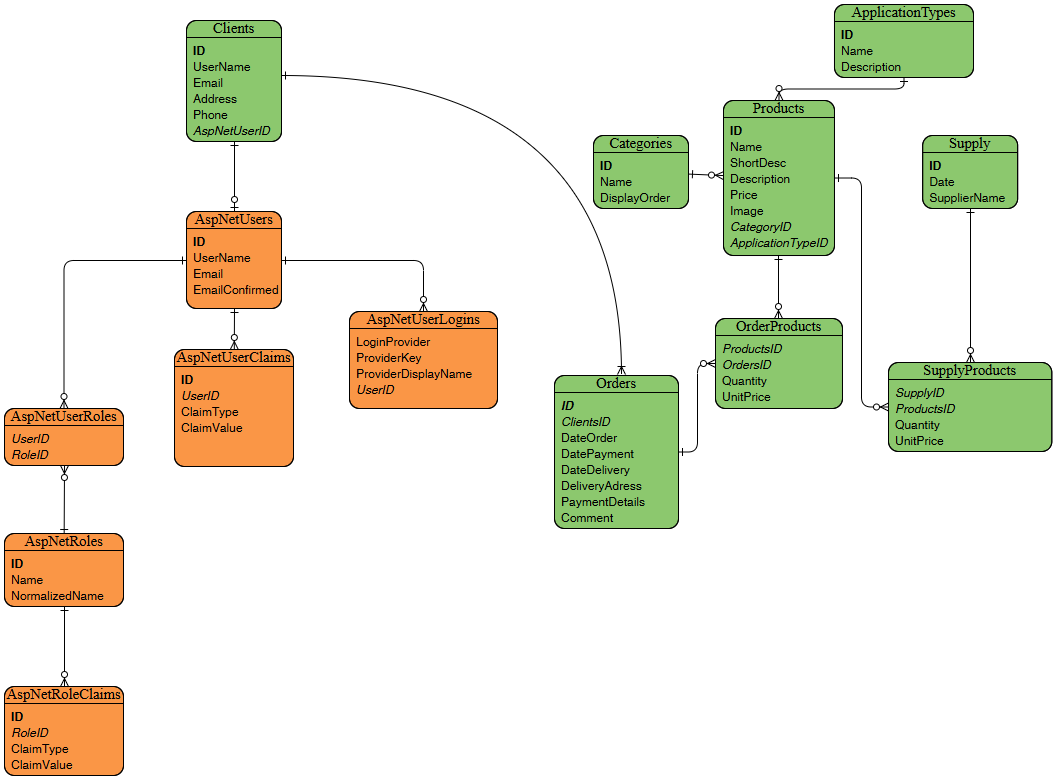


Рис. 11- Логическая модель базы данных

Все связи между сущностями реализованы через внешние ключи, что обеспечивает целостность данных и позволяет эффективно управлять отношениями между таблицами.

Опишем сущности логической модели: рассмотрим связи, предназначение и ограничения полей (3.2-3.15).

Таблица 3.2 - **Описание полей таблицы Clients**

|  |  |  |  |
| --- | --- | --- | --- |
| **Clients** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
| Первичный | Id | Да | Id Клиента |
|  | UserName | Да | Имя пользователя |
|  | Email | Да | Адрес электронной почты |
|  | Address | Да | Адрес клиента |
|  | Phone | Да | Номер телефона |
| Внешний | AspNetUsersID | Да | Ссылка на пользователя в системе аутентификации |

Таблица 3.3 - ***Описание полей таблицы AspNetUsers***

|  |  |  |  |
| --- | --- | --- | --- |
| **AspNetUsers** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
| Первичный | Id | Да | Id Пользователя |
|  | UserName | Да | Имя пользователя в системе |
|  | Email | Да | Адрес электронной почты пользователя |
|  | EmailConfirmed | Да | Статус подтверждения email |

Таблица 3.4 - **Описание полей таблицы AspNetUserRoles**

|  |  |  |  |
| --- | --- | --- | --- |
| **AspNetUserRoles** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
| Внешний | UserId | Да | Id Пользователя |
| Внешний | RoleId | Да | Id Роли |

Таблица 3.5 - **Описание полей таблицы AspNetRoles**

|  |  |  |  |
| --- | --- | --- | --- |
| **AspNetRoles** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
| Первичный | Id | Да | Id Роли |
|  | Name | Да | Название роли |
|  | NormalizedName | Да | Нормализированное название роли |

Таблица 3.6 - **Описание полей таблицы AspNetRoleClaims**

|  |  |  |  |
| --- | --- | --- | --- |
| **AspNetRoleClaims** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
| Первичный | Id | Да | Id Требования |
| Внешний | RoleId | Да | Id Роли |
|  | ClaimType | Да | Тип требования |
|  | ClaimValue | Да | Размер требования |

Таблица 3.7 - **Описание полей таблицы AspNetUserClaims**

|  |  |  |  |
| --- | --- | --- | --- |
| **AspNetUserClaims** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
| Первичный | Id | Да | Id Требования |
| Внешний | UserId | Да | Id Пользователя |
|  | ClaimType | Да | Тип пользователя |
|  | ClaimValue | Да | Значение пользователя |

Таблица 3.8 - **Описание полей таблицы AspNetUserLogins**

|  |  |  |  |
| --- | --- | --- | --- |
| **AspNetUserLogins** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
|  | LoginProvider | Да | Провайдер аутентификации |
|  | ProviderKey | Да | Ключ провайдера |
|  | ProviderDisplayName | Да | Отображаемое имя провайдера |
| Внешний | UserID | Да | Id Пользователя |

Таблица 3.9 - **Описание полей таблицы Products**

|  |  |  |  |
| --- | --- | --- | --- |
| **Products** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
| Первичный | Id | Да | Id Продукта |
|  | Name | Да | Название Продукта |
|  | ShortDesc | Нет | Краткое описание |
|  | Description | Нет | Полное описание |
|  | Price | Да | Цена Продукта |
|  | Image | Нет | Изображение продукта |
| Внешний | CategoryID | Да | Id Категории |
| Внешний | ApplicationTypeID | Да | Id Типа Продукта |

Таблица 3.10 - **Описание полей таблицы Orders**

|  |  |  |  |
| --- | --- | --- | --- |
| **Orders** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
| Первичный | Id | Да | Id Продукта |
| Внешний | ClientsId | Да | Id Клиента |
|  | DateOrder | Да | Дата создания заказа |
|  | DatePayment | Да | Дата оплаты |
|  | DateDelivery | Да | Дата доставки |
|  | DeliveryAdress | Да | Адрес доставки |
|  | PaymentDetails | Да | Реквизиты платежа |
|  | Comment | Да | Комментарий к заказу |

Таблица 3.11 - **Описание полей таблицы OrderProducts**

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderProducts** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
| Внешний | ProductsID | Да | Id Продукта |
| Внешний | OrdersID | Да | Id Заказа |
|  | Quantity | Да | Количество |
|  | UnitPrice | Да | Цена за единицу Продукта |

Таблица 3.12 - **Описание полей таблицы Categories**

|  |  |  |  |
| --- | --- | --- | --- |
| **Categories** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
| Первичный | Id | Да | Id Категории |
|  | Name | Да | Имя Категории |
|  | DisplayOrder | Нет | Порядок отображения |
|  |  |  |  |

Таблица 3.13 - **Описание полей таблицы ApplicationTypes**

|  |  |  |  |
| --- | --- | --- | --- |
| **ApplicationTypes** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
| Первичный | Id | Да | Id Типа |
|  | Name | Да | Имя Типа |
|  | Description | Нет | Описание Типа |

Таблица 3.14 - **Описание полей таблицы Supply**

|  |  |  |  |
| --- | --- | --- | --- |
| **Supply** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
| Первичный | Id | Да | Id Поставки |
|  | Date | Да | Дата Поставки |
|  | Description | Да | Название Поставщика |

Таблица 3.15 - **Описание полей таблицы SupplyProducts**

|  |  |  |  |
| --- | --- | --- | --- |
| **SupplyProducts** | | | |
| **Ключ** | **Поле** | **Обязательное** | **Примечение** |
| Внешний | SupplyID | Да | Id Поставки |
| Внешний | ProductsID | Да | Id Продукта |
|  | Quantity | Да | Количество |
|  | UnitPrice | Да | Цена за единицу Продукта |

После проектирования приложения разрабатываем интерфейс с навигацией по товарам и заказам на основе проверенных решений:

- главная страница: служит центральным узлом для навигации по различным разделам приложения;

- разделы и категории: компрессоры и кондиционирование, показывают заказчику, наличия оборудования;

- список продуктов: включают информацию о дизайне, функциональности и области применения, а также подробное описание технических характеристик и функций, таких как режимы работы и технология;

- корзина заказа: ориентирован на предоставление пользователю возможности просмотра и управления выбранными товарами перед завершением покупки. Он включает детальную информацию о продуктах, их цены и возможность продолжить процесс оформления заказа;

- оформление заказа: включает в себя проверку контактной информации пользователя и предоставляет возможность вернуться к корзине для внесения изменений или отправить запрос для завершения заказа.

## 3.2. Разработка веб-приложения

Перед тем как приступить к разработке веб-приложения необходимо подготовить рабочее пространство разработчика, данный этап подготовки включает в себя настройку Microsoft Visual Studio, Asp.Net Core Identity , а так же конфигурацию Git управления версиями исходного кода.

Git является важным составляющим инструментом в процессе разработки веб-приложения и в целом при создании проекта. Одним из ключевых аспектов разработки является сохранение истории изменений, так как в будущем может потребоваться доработка или модификация кода и этим вероятно, будет заниматься другой специалист, Git обеспечивает надежное и организованное хранение всех изменений в проекте.

Первым этапом следует установить программу Git на рабочую станцию разработчика и произвести настройку основных параметров: имя пользователя и адрес электронной почты, это требуется для идентификации автора в истории проекта.

При создании Git-репозитория в проекте формируется необходимая структура для отслеживания изменений в исходном коде. В ходе разработки используется файл .gitignore, где указываются файлы и папки, которые не следует включать в версионирование, например, временные или системные файлы. Это позволяет сохранять репозиторий организованным и исключать ненужные данные из контроля версий. Создание коммитов в Git, позволяет зафиксировать внесенные изменения в код, формируя таким образом последовательную историю разработки. Это облегчает контроль над прогрессом проекта и дает возможность при необходимости вернуться к ранним версиям кода.

Для совместной работы над проектом в Git позволяет настроить удаленные репозитории на таких платформах как: GitLab, Bitbucket, Gogs, Gitea, Apache Allura, RhodeCode или отечественные легитимные решения GitVerse, GitFlic. Разработчики могут синхронизировать свои локальные изменения с центральным репозиторием, обмениваться кодом с коллегами и объединять изменения, внесенные другими участниками команды, что обеспечивает эффективное сотрудничество при совместной разработке.

Microsoft Visual Studio облегчает процесс разработки благодаря встроенной поддержке Git, позволяя разработчикам использовать инструменты контроля версий прямо внутри среды разработки без необходимости перехода к внешним приложениям. За счет встроенной поддержи Git, есть возможность выполнять создание, переключение, слияние веток, коммитинг изменений и разрешение конфликтов не покидая интерфейс IDE. Данные возможности, повышают продуктивность разработчикам, упрощая управление версиями и способствуя более эффективному и бесконфликтному процессу работы над проектом.

Настройка Git создаёт необходимую основу для систематического и эффективного управления исходным кодом веб-приложения на каждом этапе его разработки. Это не только способствует улучшению качества кода, но и облегчает процесс интеграции изменений, а также обеспечивает чёткую и прозрачную организацию работы команды разработчиков.

После завершения настройки Git структура проекта была спроектирована в соответствии с общепринятыми стандартами разработки на основе MVC и платформы .NET Core. Проект был логически разделён на следующие компоненты:

- Model (Модель). Отвечает за управление данными приложения, включая структуры, схемы и интерфейсы, а также за взаимодействие с базой данных и её полями.

- View (Представление). Включает в себя все элементы пользовательского интерфейса, с которыми взаимодействует конечный пользователь.

- Controller (Контроллер). Выполняет функции обработки бизнес-логики, выступая связующим звеном между моделью и представлением, а также управляет входящими запросами.

Такое разделение позволяет значительно упростить процесс разработки, тестирования и дальнейшего обслуживания приложения, делая его более организованным и структурированным (Рис. 12).

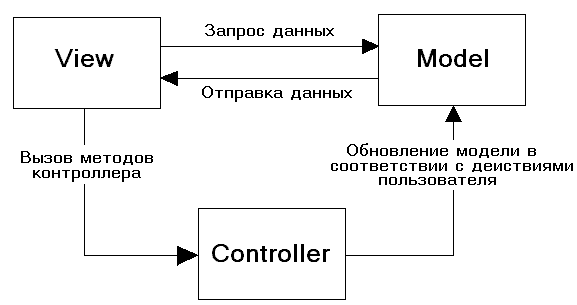


Рис. 12- Визуальное представление MVC

Так же следует отметить, что веб-приложение было спроектировано по в соответствии с принципами «*Чистой архитектуры* *(Clean Architecture)»*, которая представляет собой современный метод проектирования программных решений, направленный на четкое распределение обязанностей между различными частями приложения (Рис. 13).

Ключевым принципом «*Чистой архитектуры»* является правило зависимостей, согласно которому все связи в коде должны быть направлены внутрь, к центральной части системы. Внутренние компоненты не должны зависеть от внешних, и никакие элементы, определенные во внешних слоях, не должны использоваться внутри ядра.

Чистая архитектура включает четыре основных уровня, организованных по принципу «луковицы». Центральный слой — сущности (Entities) — содержит бизнес-логику уровня предприятия, которая остается стабильной даже при изменениях внешних условий. Слой сценариев использования (Use Cases) отвечает за бизнес-логику приложения и управляет потоком данных между сущностями. Слой интерфейсов и адаптеров преобразует данные между форматами, удобными для внутренних слоев и внешних систем, включая пользовательские интерфейсы, такие как MVC. Внешний слой включает фреймворки и драйверы, такие как базы данных и веб-инструменты.

Преимущества *Чистой архитектуры* очевидны. Во-первых, она обеспечивает независимость от конкретных фреймворков и библиотек, что позволяет легко их заменять без серьезных изменений в коде. Во-вторых, она отделяет пользовательский интерфейс от бизнес-логики, что упрощает модификацию UI. В-третьих, она делает систему независимой от способов хранения данных, что облегчает переход между различными СУБД. В-четвертых, она повышает тестируемость кода, что критически важно для долгосрочной поддержки проекта.

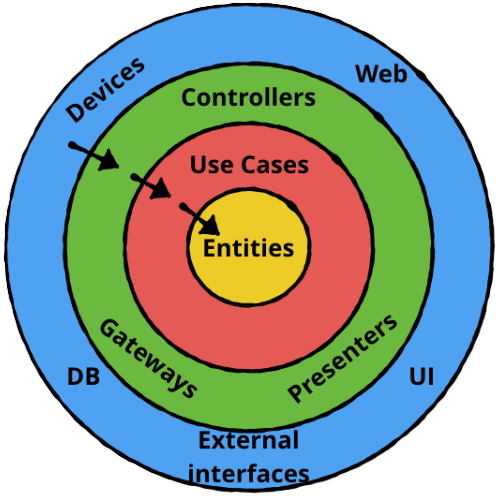


Рис. 13- Представление архитектуры проекта

При внедрении *Чистой архитектуры* важно соблюдать принцип инверсии зависимостей, который предполагает создание интерфейсов во внутренних слоях и их реализацию во внешних. Это позволяет изменять внешние компоненты, не затрагивая ядро системы. Также важно, чтобы данные, передаваемые между слоями, были простыми и не зависели от конкретных технологий. Для хранения данных в реляционной базе SQL был использован подход «Entity Framework Core Code-First». Данная методология позволяет описывать структуру данных с помощью классов на языке C#, а сам фреймворк создает и поддерживает соответствующую схему базы данных на основе этих классов.

Основные шаги для реализации подхода Code-First включают:

1. Определение моделей данных:

В проекте разрабатываются классы, которые отражают сущности предметной области. Эти классы включают свойства, которые в дальнейшем будут связаны с таблицами и столбцами в базе данных. Пример класса Category ниже:

public class Category

{

[Key]

public int Id { get; set; }

[Required]

public string Name { get; set; }

[DisplayName("DisplayOrder")]

[Required]

[Range (1, int.MaxValue,ErrorMessage ="отображения значения категории должен быть больше 0")]

public int DisplayOrder { get; set; }

}

2. Создание контекста данных:

Контекст базы данных является центральным компонентом, который управляет объектами сущностей и выполняет операции с базой данных. Контекст наследуется от класса DbContext и включает свойства типа DbSet<T> (T – класс, который конвертируется в таблицу в базе данных), представляющие коллекции сущностей. Пример контекста данных ниже:

public class ApplicationDbContext : IdentityDbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

public DbSet<Category> Category { get; set; }

public DbSet<ApplicationType> ApplicationType { get; set; }

public DbSet<Product> Product { get; set; }

public DbSet<ApplicationUser> ApplicationUser { get; set; }

public DbSet<OrderVM> Orders { get; set; }

}

*Наследование от IdentityDbContext:*

- включает встроенную поддержку аутентификации и авторизации

- автоматически создает таблицы для пользователей и ролей

- обеспечивает базовую структуру для безопасности

*Конструктор контекста:*

- принимает DbContextOptions для настройки подключения

- вызывает базовый конструктор через base(options)

- позволяет конфигурировать подключение к базе данных

*DbSet свойства:*

- каждый DbSet представляет таблицу в базе данных

- типизированы по соответствующим классам-моделям

- предоставляют доступ к CRUD-операциям через LINQ

3. Настройка строки подключения:

В файле конфигурации проекта (appsettings.json) указывается строка подключения к базе данных SQLEXPRESS в формате:

{

"ConnectionStrings": {

"DefaultConnection": "Server=Адрес сервера \\SQLEXPRESS;Database=IBCompany;Trusted\_Connection=True; MultipleActiveResultSets=True"

}

}

4. Миграции:

EF Core включает инструмент миграций, предназначенный для актуализации структуры базы данных при внесении изменений в модель. С помощью команды Add-Migration создаются скрипты миграций, основанные на модификациях модели, а команда Update-Database применяет эти изменения к самой базе данных (Рис. 14).



Рис. 14- Миграции в проекте

Применение методологии Code-First в EF Core предоставляет разработчикам гибкость и полный контроль над структурой данных и конфигурацией базы. Это позволяет работать с информацией через объекты, абстрагируясь от низкоуровневых аспектов взаимодействия с реляционной базой. Такой подход упрощает поддержку и масштабирование проекта, поскольку любые корректировки модели данных можно легко перенести на схему базы с использованием механизма миграций.

Дальнейшим шагом была разработка логики приложения, описанная в слое *Infrastructure*, который обеспечивает взаимодействие с базой данных, внешними сервисами и другими инфраструктурными компонентами. Этот слой включает в себя репозитории для работы с данными и сервисы для обработки бизнес-логики, обеспечивая выполнение всех операций, связанных с управлением данными и их хранением.

*Репозитории* в слое *Infrastructure* реализуют интерфейсы, определенные в доменном слое, и отвечают за взаимодействие с базой данных. Они инкапсулируют всю логику доступа к данным и предоставляют методы для выполнения *CRUD-операций* (создание, чтение, обновление, удаление) над сущностями. Использование репозиториев позволяет абстрагировать логику работы с базой данных от остальной части приложения, улучшая модульность и тестируемость кода.

Пример общего класса *репозитория*:

public class RepositoryAsync<T> : IRepositoryAsync<T> where T : class

{

internal readonly ApplicationDbContext \_db;

DbSet<T> \_dbSet;

public RepositoryAsync(ApplicationDbContext dbContext)

{

\_db = dbContext;

this.\_dbSet = \_db.Set<T>();

}

public virtual async Task<List<T>> GetAllAsync()

{

return await \_db.Set<T>().ToListAsync();

}

public async Task AddAsync(T entity)

{

await \_dbSet.AddAsync(entity);

await Save();

}

public async Task Save()

{

await \_db.SaveChangesAsync();

}

}

Пример реализации репозитория для сущности *Product*:

public class ProductRepository : RepositoryAsync<Product>, IProductRepository

{

public ProductRepository(ApplicationDbContext dbContext) : base(dbContext) { }

public async Task<Product> GetProductByIdAsync(int id)

{

return await \_db.Products

.Include(p => p.Category)

.Include(p => p.ApplicationType)

.FirstOrDefaultAsync(p => p.Id == id);

}

public async Task<List<Product>> GetProductsByCategoryAsync(int categoryId)

{

return await \_db.Products

.Where(p => p.CategoryId == categoryId)

.ToListAsync();

}

}

*Сервисы* в слое *Infrastructure* отвечают за выполнение бизнес-логики, связанной с обработкой данных. Они используют репозитории для доступа к данным и могут включать дополнительные операции, такие как валидация данных, управление транзакциями и взаимодействие с внешними API. Сервисы обеспечивают координацию работы между различными компонентами приложения и служат точками входа для выполнения бизнес-операций.

Пример сервиса для работы с *продуктами*:

public class ProductService : IProductService

{

private readonly IProductRepository \_productRepository;

public ProductService(IProductRepository productRepository)

{

\_productRepository = productRepository;

}

public async Task<List<Product>> GetProductsByCategoryAsync(int categoryId)

{

return await \_productRepository.GetProductsByCategoryAsync(categoryId);

}

public async Task AddProductAsync(Product product)

{

await \_productRepository.AddAsync(product);

}

}

**Разработка пользовательского интерфейса в *ASP.NET Core MVC*.**

В ASP.NET Core MVC пользовательский интерфейс строится на основе Razor-представлений, которые представляют собой HTML-страницы с встроенной логикой на C#. Каждое представление связано с контроллером, который обрабатывает запросы пользователя и возвращает соответствующие данные.

Пример контроллера для работы с продуктами *(ProductController):*

[Authorize(Roles = WC.AdminRole)]

public class ProductController : Controller

{

private readonly ApplicationDbContext \_db;

private readonly IWebHostEnvironment \_webHostEnvironment;

public ProductController(ApplicationDbContext db, IWebHostEnvironment webHostEnvironment)

{

\_db = db;

\_webHostEnvironment = webHostEnvironment;

}

public IActionResult Index()

{

IEnumerable<Product> objlist = \_db.Product.Include(u => u.Category).Include(u => u.ApplicationType);

return View(objlist);

}

public IActionResult Upsert(int? id)

{

ProductVM productVM = new ProductVM()

{

Product = new Product(),

CategorySelectList = \_db.Category.Select(i => new SelectListItem

{

Text = i.Name,

Value = i.Id.ToString()

}),

ApplicationTypeSelectList = \_db.ApplicationType.Select(i => new SelectListItem

{

Text = i.Name,

Value = i.Id.ToString()

})

};

if (id == null)

{

return View(productVM);

}

else

{

productVM.Product = \_db.Product.Find(id);

if (productVM.Product == null)

{

return NotFound();

}

return View(productVM);

}

}

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Upsert(ProductVM productVM)

{

if (ModelState.IsValid)

{

var files = HttpContext.Request.Form.Files;

string webRootPath = \_webHostEnvironment.WebRootPath;

if (productVM.Product.Id == 0)

{

string upload = webRootPath + WC.ImagePath;

string fileName = Guid.NewGuid().ToString();

string extension = Path.GetExtension(files[0].FileName);

using (var fileStream = new FileStream(Path.Combine(upload, fileName + extension), FileMode.Create))

{

files[0].CopyTo(fileStream);

}

productVM.Product.Image = fileName + extension;

\_db.Product.Add(productVM.Product);

}

else

{

var objFromDb = \_db.Product.AsNoTracking().FirstOrDefault(u => u.Id == productVM.Product.Id);

if (files.Count > 0)

{

string upload = webRootPath + WC.ImagePath;

string fileName = Guid.NewGuid().ToString();

string extension = Path.GetExtension(files[0].FileName);

var oldFile = Path.Combine(upload, objFromDb.Image);

if (System.IO.File.Exists(oldFile))

{

System.IO.File.Delete(oldFile);

}

using (var fileStream = new FileStream(Path.Combine(upload, fileName + extension), FileMode.Create))

{

files[0].CopyTo(fileStream);

}

productVM.Product.Image = fileName + extension;

}

else

{

productVM.Product.Image = objFromDb.Image;

}

\_db.Product.Update(productVM.Product);

}

\_db.SaveChanges();

return RedirectToAction("Index");

}

productVM.CategorySelectList = \_db.Category.Select(i => new SelectListItem

{

Text = i.Name,

Value = i.Id.ToString()

});

productVM.ApplicationTypeSelectList = \_db.ApplicationType.Select(i => new SelectListItem

{

Text = i.Name,

Value = i.Id.ToString()

});

return View(productVM);

}

public IActionResult Delete(int? id)

{

if (id == null || id == 0)

{

return NotFound();

}

Product product = \_db.Product.Include(u => u.Category).Include(u => u.ApplicationType).FirstOrDefault(u => u.Id == id);

if (product == null)

{

return NotFound();

}

return View(product);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public IActionResult DeletePost(int? id)

{

var obj = \_db.Product.Find(id);

if (obj == null)

{

return NotFound();

}

string upload = \_webHostEnvironment.WebRootPath + WC.ImagePath;

var oldFile = Path.Combine(upload, obj.Image);

if (System.IO.File.Exists(oldFile))

{

System.IO.File.Delete(oldFile);

}

\_db.Product.Remove(obj);

\_db.SaveChanges();

return RedirectToAction("Index");

}

}

Пример представления для отображения списка продуктов*(Index.cshtml):*

@model IEnumerable<Product>

<h2>Products</h2>

<table class="table">

<thead>

<tr>

<th>Name</th>

<th>Price</th>

<th>Category</th>

<th>Application Type</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

@foreach (var product in Model)

{

<tr>

<td>@product.Name</td>

<td>@product.Price</td>

<td>@product.Category.Name</td>

<td>@product.ApplicationType.Name</td>

<td>

<a asp-action="Upsert" asp-route-id="@product.Id">Edit</a> |

<a asp-action="Delete" asp-route-id="@product.Id">Delete</a>

</td>

</tr>

}

</tbody>

</table>

<a asp-action="Upsert" class="btn btn-primary">Create New</a>

*Загрузка файлов (изображений)*

В проекте реализована возможность загрузки изображений для продуктов. Это делается с помощью встроенных возможностей ASP.NET Core для работы с файлами.

**-** IWebHostEnvironment: предоставляет доступ к корневой папке веб-приложения (wwwroot), где хранятся статические файлы, такие как изображения.

*-* HttpContext.Request.Form.Files**:** позволяет получить файлы, отправленные через форму.

Пример загрузки *изображения*:

if (productVM.Product.Id == 0)

{

string upload = webRootPath + WC.ImagePath;

string fileName = Guid.NewGuid().ToString();

string extension = Path.GetExtension(files[0].FileName);

using (var fileStream = new FileStream(Path.Combine(upload, fileName + extension), FileMode.Create))

{

files[0].CopyTo(fileStream);

}

productVM.Product.Image = fileName + extension;

}

*Аутентификация и авторизация*

Так как в проекте используется встроенная система аутентификации и авторизации ASP.NET Core. Это позволяет ограничивать доступ к определенным действиям контроллера только для авторизованных пользователей с определенными ролями.

Пример использования:

[Authorize(Roles = WC.AdminRole)]

public class ProductController : Controller

{

// Методы контроллера

}

## 3.3 Интерфейс веб-приложения

Интерфейс веб-приложения представляет собой многостраничную структуру, где каждая страница выполняет определённые функции, обеспечивая удобство навигации и взаимодействия с системой (Рис. 15).

Главная страница служит точкой входа в приложение и предоставляет пользователям обзор доступных товаров. Дизайн выполнен в минималистичном стиле с акцентом на удобство использования. Основные элементы главной страницы включают:

- навигационная панель: расположена в верхней части экрана и закреплена при прокрутке, содержит ссылки на ключевые разделы, такие как, «Главная страница», «Корзина», «Авторизация пользователя»;

- галерея оборудования: товары представлены в виде карточек с изображениями, краткими описаниями и ценой, каждая карточка кликабельна при нажатии открывается страница с детализированной информацией о товаре, реализована адаптивная сетка, обеспечивающая корректное отображение на разных устройствах.

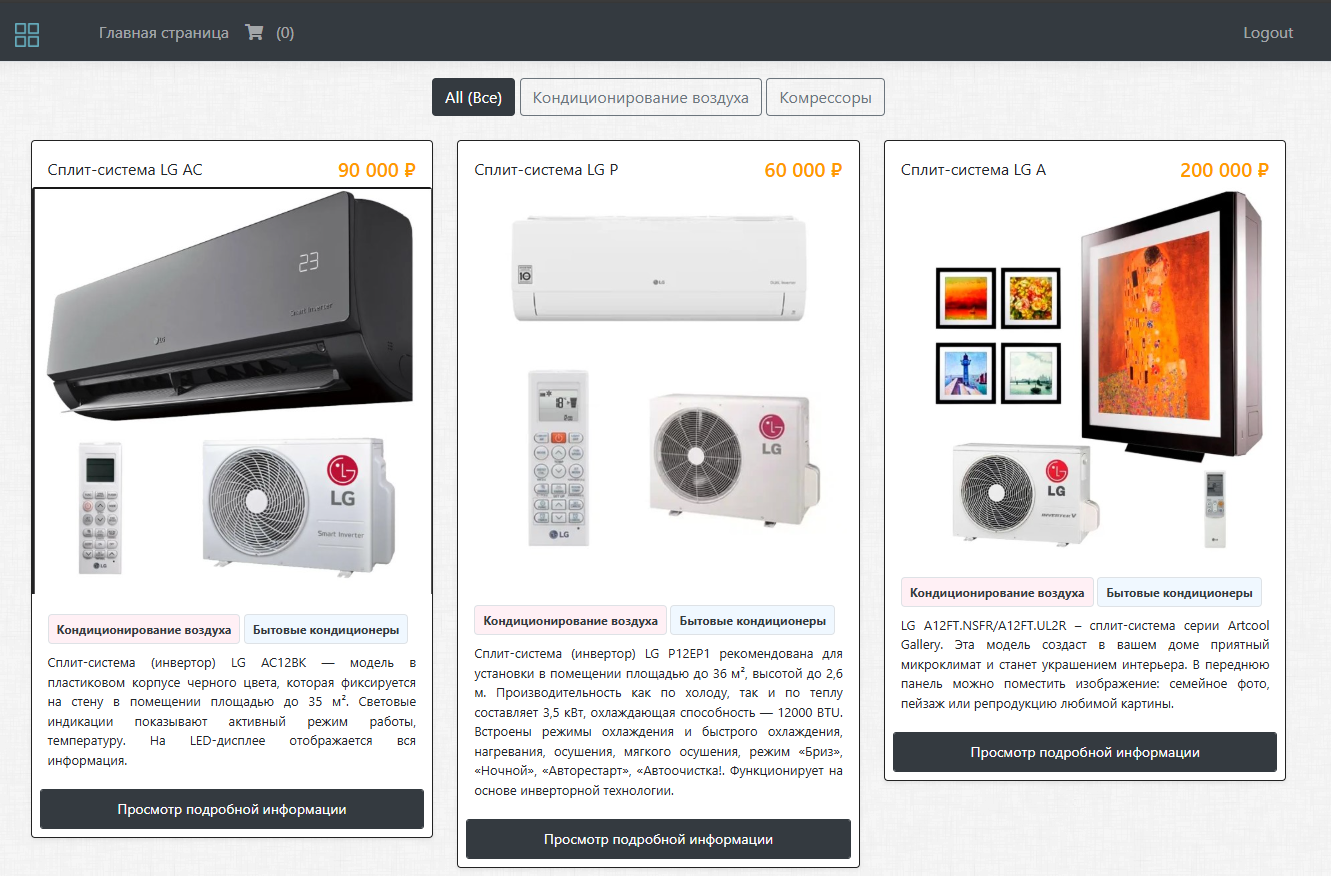


Рис. 15- Главная страница

Для удобства поиска оборудование разделено на категории, такие как,   
«Все оборудование в наличии» показывает полный перечень доступных позиций, «Кондиционирование воздуха» климатическая техника и сопутствующие товары, «Компрессорное оборудование» промышленные и бытовые компрессоры. Переключение между категориями происходит без перегрузки страницы (AJAX-фильтрация) (Рис. 16).

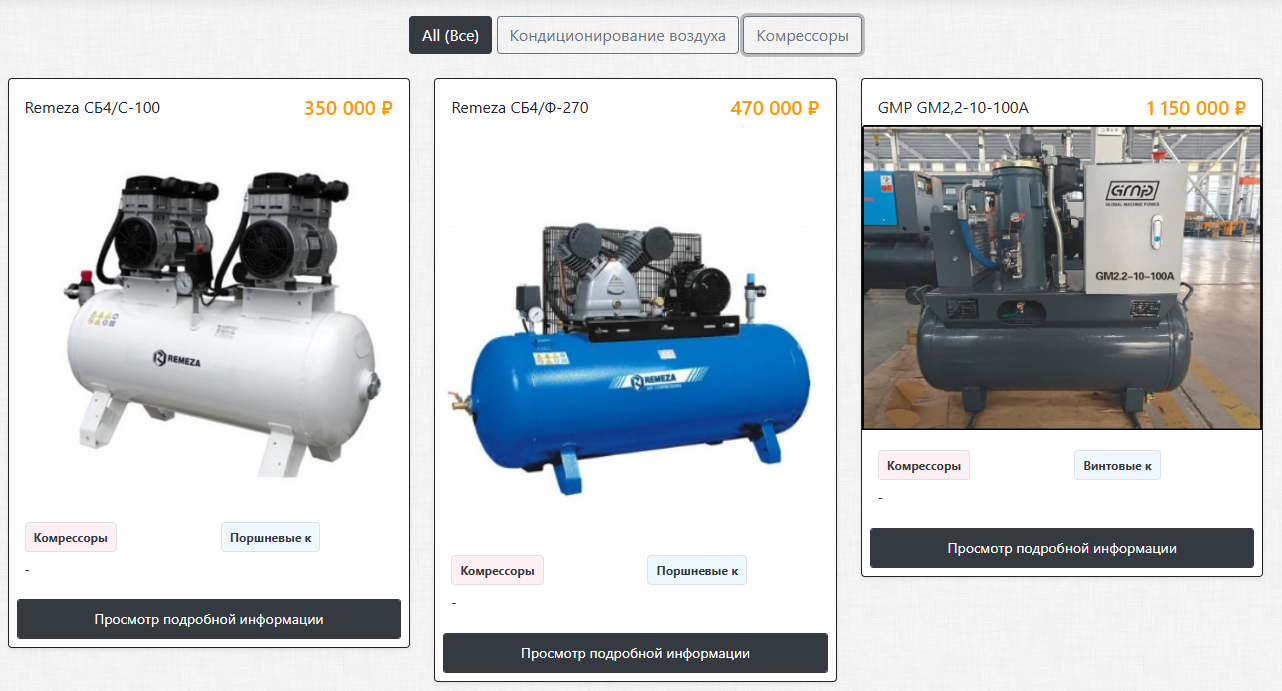


Рис. 16 - Категория «Компрессорное оборудование»

При нажатии «Просмотр подробной информации» из галереи открывается страница с информацией о выбранной позиции. Пользователь видит фотогалерею (основное изображение + превью для увеличения), технические характеристики в виде текста, кнопки «Добавить в корзину» и «Вернуться к списку» (Рис. 17).



Рис. 17- Страница товара

При переходе на вкладку «Корзина» пользователи должны пройти авторизацию. Страница включает в себя форму авторизации с полями для ввода логина и пароля и кнопку «Войти» (Рис. 18).



Рис. 18 - Страница Авторизации

Страница «Корзина» отображает список выбранных товаров с возможностью редактирования. Кнопка «Продолжить» позволяет перейти к форме отправки запроса менеджеру (Рис. 19).

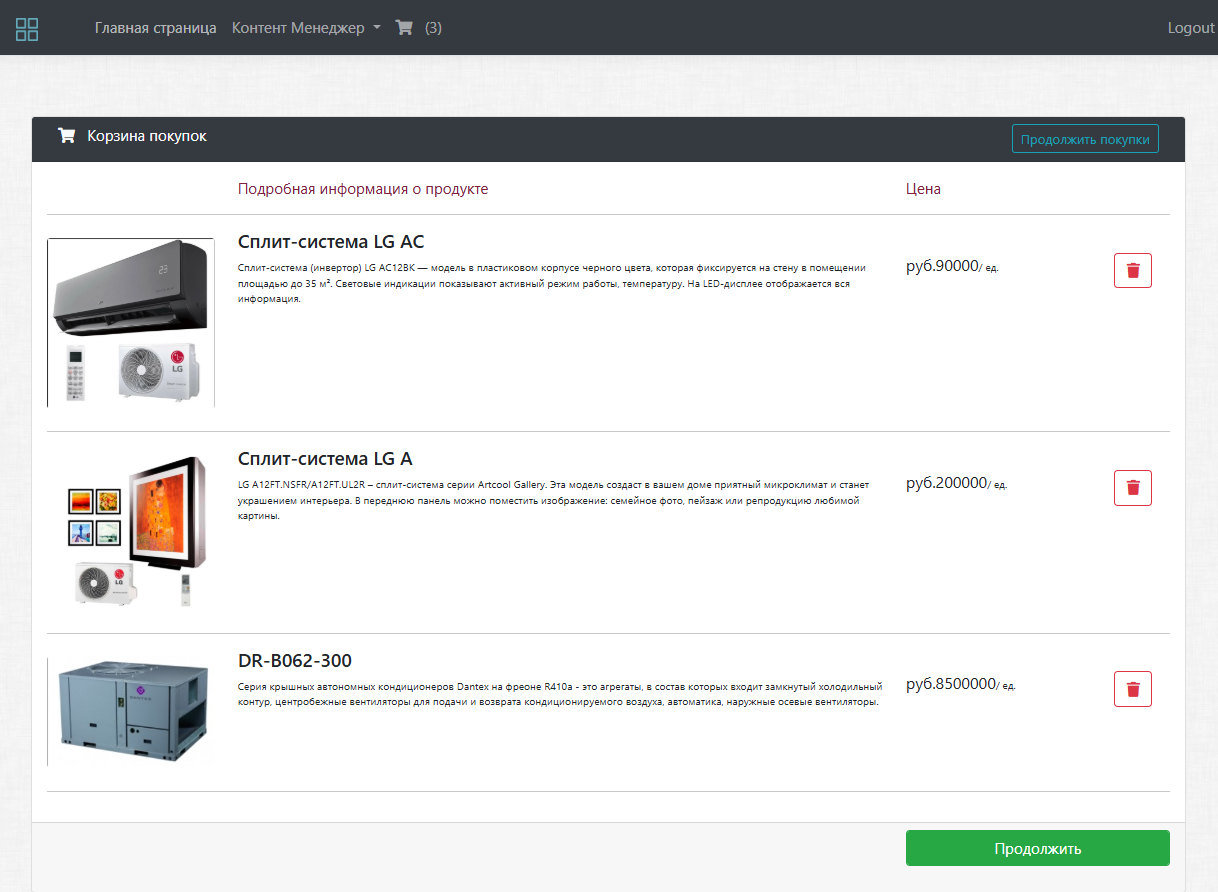


Рис. 19 - Страница Корзина

При оформлении заказа на странице корзины можно увидеть выбранные услуги, и оформить заказ заполнив данные клиента (Рис. 20). Основные элементы включают:

* форма для ввода данных: поля для имени, номера телефона, email.
* кнопка «оформить заказ» для оформления заказа.

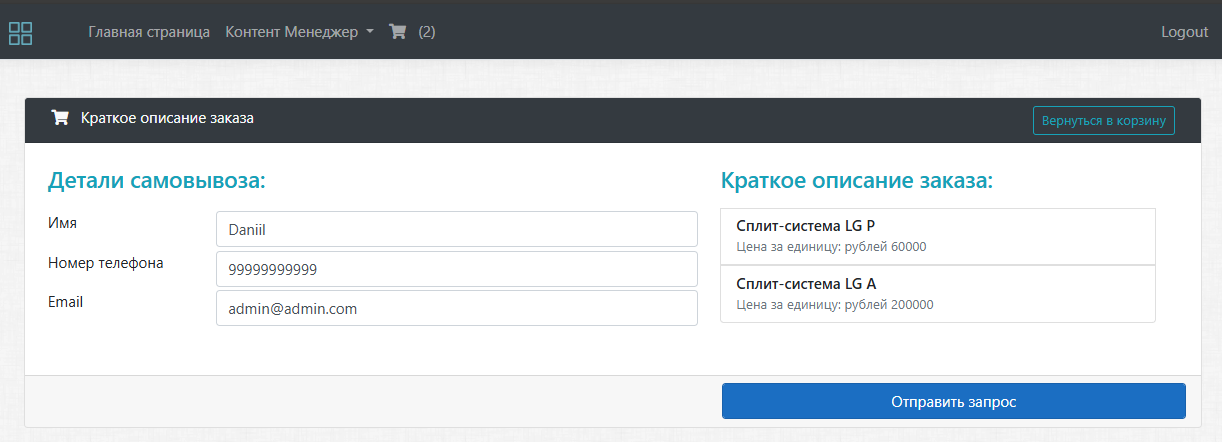


Рис. 20 - Страница Оформление заказа

При авторизации от лица администратора можно просматривать заказы, видеть их статус и переходить на каждый заказ для его управления (Рис. 21).



Рис. 21 - Заказы клиентов

## 

## 3.4 Выбор хостинга и развертывание веб-приложения

После создания веб-приложения, следует выбрать хостинг для проекта, данная работа является одной из ключевых этапов успешного запуска и дальнейшего функционирования веб-приложения. Хостинг предоставляет необходимую инфраструктуру для размещения приложения, обеспечивает его доступность, безопасность и производительность. При выборе хостинга необходимо учитывать множество факторов: технические требования проекта, бюджет, географическое положение пользователей, уровень поддержки, а также особенности законодательства (например, требования к размещению данных на территории России). В данном разделе рассматриваются основные виды хостинга, их преимущества и недостатки, а также обзор популярных российских хостинг-провайдеров. Также приводится сравнительная таблица видов хостинга для наглядного анализа.

Общий хостинг (Shared Hosting) - это наиболее простой и доступный вид хостинга, при котором несколько сайтов размещаются на одном физическом сервере. Этот тип хостинга идеально подходит для небольших проектов, таких как личные блоги, сайты-визитки или небольшие интернет-магазины. Общий хостинг имеет низкую стоимость благодаря совместному использованию ресурсов, что делает его экономически выгодным решением для начинающих пользователей. Управление сайтом упрощено за счет удобных панелей управления, таких как ISPmanager или cPanel, которые позволяют легко управлять сайтом даже без технических знаний. Однако ограниченные ресурсы могут стать проблемой для проектов с высокой нагрузкой, а зависимость от «соседей» по серверу может привести к снижению скорости работы или временным отключениям. Кроме того, возможности настройки сервера минимальны, так как конфигурация определяется провайдером.

Виртуальные выделенные сервера (Virtual Dedicated Server / Virtual Private Server) - это виртуализированное решение, при котором один физический сервер делится на несколько виртуальных машин. Каждая виртуальная машина имеет гарантированные ресурсы и работает независимо от других. Этот тип хостинга подходит для средних проектов, которым требуется больше контроля и ресурсов, чем предоставляет общий хостинг. Основными преимуществами VDS/VPS являются гарантированные ресурсы, гибкость в настройке программного обеспечения и возможность масштабирования. Однако управление сервером требует базовых навыков администрирования, хотя многие провайдеры предлагают услуги по управлению за дополнительную плату. Также стоимость VDS/VPS выше, чем у общего хостинга.

Облачный хостинг (Cloud Hosting) использует технологии распределенных вычислений и виртуализации, где ресурсы выделяются динамически в зависимости от текущей нагрузки, что позволяет легко масштабировать проект. Этот тип хостинга особенно популярен среди современных веб-приложений. Преимуществами облачного хостинга являются отказоустойчивость благодаря размещению данных и приложений на нескольких серверах, гибкость в управлении ресурсами и возможность оплаты только за фактически использованные ресурсы. Недостатками являются зависимость от провайдера, что может усложнить переход на другую инфраструктуру, а также высокая стоимость при длительном использовании, если нагрузка на проект стабильно высока.

Хостинг контейнеров (Container Hosting) позволяет запускать приложения в изолированных средах (контейнерах), что упрощает их развертывание и масштабирование. Этот тип хостинга становится все более популярным среди разработчиков, особенно в микросервисной архитектуре. Преимуществами хостинга контейнеров являются эффективное использование ресурсов благодаря компактности контейнеров, удобство разработки и тестирования за счет создания одинаковых сред, а также быстрое развертывание приложений. Однако работа с контейнерами требует навыков работы с Docker и Kubernetes, а для небольших сайтов контейнеризация может быть избыточной и сложной для внедрения. Сравнительный анализ видов хостинга приведен в таблице 3.1.

Таблица 3.1 **- Сравнительный анализ видов хостинга**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Характеристика | Общий хостинг | VDS /VPS | Выделенный сервер | Облачный хостинг | Хостинг контейнеров |
| 1 | Стоимость | Низкая | Средняя | Высокая | Переменная | Средняя |
| 2 | Управление | Простое | Средняя | Сложное | Среднее | Сложное |
| 3 | Масштабируемость | Ограниченная | Высокая | Высокая | Очень высокая | Очень высокая |
| 4 | Производительность | Низкая | Средняя | Высокая | Высокая | Высокая |
| 5 | Контроль над сервером | Минимальный | Частичный | Полный | Частичный | Частичный |
| 6 | Безопасность | Средняя | Высокая | Очень высокая | Высокая | Высокая |

Проанализировав различные виды хостинга, был сделан выбор в пользу облачного хостинга, так как он наиболее полно соответствует требованиям современного веб-приложения. Облачный хостинг предлагает гибкость и масштабируемость, что позволяет динамически адаптироваться к нагрузке и легко увеличивать вычислительные мощности при росте проекта. Это особенно важно для приложений, которые планируют развиваться и привлекать больше пользователей, так как исключает необходимость частой миграции на более мощные серверы. Высокая отказоустойчивость облачного хостинга обеспечивается за счет распределенной архитектуры, где данные и приложения размещаются на нескольких серверах, что минимизирует риск простоев и повышает надежность работы приложения. Экономическая эффективность модели оплаты по фактическому использованию делает облачный хостинг выгодным решением, особенно для проектов с переменной нагрузкой, так как отсутствует необходимость платить за неиспользуемые ресурсы. Таким образом, облачный хостинг сочетает в себе высокую производительность, безопасность, удобство управления и экономическую целесообразность, что делает его оптимальным выбором для развертывания и поддержки веб-приложения.

На российском рынке облачных хостингов представлено множество провайдеров, каждый из которых предлагает свои уникальные особенности. Таким образом необходимо провести анализ существующих решений.

*Selectel* предлагает высокопроизводительные процессоры Intel Xeon последних поколений, возможность выбора между SSD и NVMe дисками для максимальной скорости обработки данных, а также надежную инфраструктуру с центрами обработки данных в нескольких городах России, что обеспечивает отказоустойчивость. Однако услуги Selectel имеют относительно высокую стоимость по сравнению с конкурентами, а сложный интерфейс может быть непонятен начинающим пользователям.

*Timeweb Cloud* выделяется простотой использования благодаря интуитивно понятной панели управления, быстрыми SSD-накопителями и поддержкой Docker и Kubernetes для контейнеризации приложений. Провайдер регулярно предлагает акции и скидки для новых клиентов. В то же время, выбор процессоров ограничен, а гибкость в настройке конфигураций недостаточна для крупных проектов.

*RUVDS* предоставляет доступные цены при высоком качестве обслуживания, быструю активацию серверов и возможность масштабирования. Удобный интерфейс управления и хорошая техническая поддержка делают этот сервис привлекательным для небольших проектов. Однако возможности для сложных корпоративных решений ограничены, а стабильность работы сети может снижаться при высокой нагрузке.

*Yandex Cloud* использует современные процессоры Intel и AMD, предлагает интеграцию с экосистемой Яндекса, включая Yandex.Disk и Yandex.Metrica, а также поддержку технологий, таких как Kubernetes, Docker и машинное обучение. Гибкая система оплаты и возможность тестирования сервисов бесплатно являются дополнительными преимуществами. Тем не менее, интерфейс может показаться сложным для новичков, а документация для некоторых специализированных сервисов недостаточно подробна.

*Mail.ru Cloud Solutions (MCS)* обеспечивает высокую производительность за счет использования современных процессоров и NVMe-дисков, широкий набор инструментов для автоматизации и управления инфраструктурой, а также соответствие требованиям законодательства РФ. Техническая поддержка и SLA до 99,95% делают этот сервис надежным выбором для бизнеса. Однако высокая стоимость услуг может быть неприемлемой для малого бизнеса, а сложность настройки может отпугнуть неопытных пользователей.

Анализируя все предложенные решения, можно сделать вывод, что выбор облачного хостинга зависит от масштаба проекта, бюджета и требований к производительности. Для небольших проектов с ограниченным бюджетом RUVDS или Timeweb Cloud являются оптимальными вариантами благодаря доступной цене и простоте использования. Для средних и крупных проектов, требующих высокой производительности и надежности, лучше выбрать Selectel или Mail.ru Cloud Solutions, которые предлагают современные технологии и широкие возможности для масштабирования.

В данном случае, учитывая потребности проекта в высокой производительности, отказоустойчивости и гибкости настройки, был сделан выбор в пользу Yandex Cloud. Этот провайдер сочетает в себе современные технологии, надежную инфраструктуру и возможность тестирования сервисов бесплатно, что делает его наиболее подходящим решением для развертывания и поддержки веб-приложения.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была проведена всесторонняя оценка работы ООО «Инженерное Бюро», включающая анализ организационной структуры, технологического процесса и основных бизнес-процессов компании. Были рассмотрены различные программные решения для создания веб-приложения, предназначенного для улучшения взаимодействия с клиентами и автоматизации обработки заказов.

Ключевым моментом работы стала разработка веб-приложения для автоматизации процесса заказа в сфере продаж оборудования в ООО «Инженерное Бюро» г. Тюмени. При разработке веб-приложения главной задачей было предоставление достаточной информации по оборудованию для клиентов.

В ходе работы были использованы такие средства, как: Microsoft Visual Studio, на основе, которой велась основная разработка. ASP.Net MVC для разработки благодаря удобству разработки на языка программирования C#, его интеграции с существующими системами управления и высокому уровню безопасности. Для доступа к данным была выбрана СУБД MS SQL Server, известная своей надежностью, масштабируемостью и открытым исходным кодом. В качестве ORM для работы с СУБД был выбран Entity Framework Core, который предоставляет удобные инструменты для управления данными и взаимодействия с базой данных.

Кроме того, разработка архитектуры приложения была выполнена с использованием практики разделения кода на слои (Domain, Application, Infrastructure, Presentation). Это позволило добиться модульности и упрощения процесса поддержки и масштабируемости проекта. Такой подход обеспечил гибкость и возможность быстрого внесения изменений в систему без нарушения ее общей структуры.

Таким образом, предложенное решение позволит компании «Инженерное Бюро», автоматизировать внутренние процессы и увеличить экономическую эффективность бизнеса. Создание и запуск веб-приложения с использованием технологий ASP.Net MVC и MS SQL Server позволит компании укрепить свои позиции на рынке, повысить конкурентоспособность и создать прочную основу для динамичного роста.