|  |  |
| --- | --- |
|  | **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 6](#_Toc193537987)

[ГЛАВА 1. ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКАЯ ХАРАКТЕРИСТИКА ОРГАНИЗАЦИИ 8](#_Toc193537988)

[1.1. Общие сведения об организации 8](#_Toc193537989)

[1.2. Организационная структура 11](#_Toc193537990)

[1.3. Технологический процесс 13](#_Toc193537991)

[ГЛАВА 2. ВЫБОР ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И МЕРОПРИЯТИЯ ПО РЕАЛИЗАЦИИ ВЕБ-ПРИЛОЖЕНИЯ. 16](#_Toc193537992)

[2.1 Анализ существующих решений 16](#_Toc193537993)

[2.2 Выбор среды разработки 24](#_Toc193537994)

[2.3 Выбор системы управления базами данных 31](#_Toc193537995)

[2.4 Мероприятия по реализации веб-приложения и введения в эксплуатацию 36](#_Toc193537996)

[ГЛАВА 3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ 39](#_Toc193537997)

[3.1 Проектирование веб-приложения 39](#_Toc193537998)

[3.2. Разработка веб-приложения 46](#_Toc193537999)

[3.3 Интерфейс веб-приложения 62](#_Toc193538000)

# ВВЕДЕНИЕ

В технологическом мире трудно представить жизнь без коммуникации в интернете, так как данная возможность ускоряет много процессов в жизни, а в частности в бизнесе. Одной из активно развивающихся тенденций у бизнеса, работа над сокращением дистанции со своим клиентским сектором, так как быстрая и удобная покупка привлекает платёжеспособного клиента, соответственно увеличивает прибыль компаний. Этот вопрос имеет огромное значение, особенно для малого бизнеса, поскольку ресурсы ограничены, а способность быстро реагировать на изменения рынка крайне важна.

В связи с этим в программировании активно развивается направление веб-программирования. Рынок борется за клиента, улучшая свои веб-приложения, работая над удобством пользовательского интерфейса. Веб-сайты, как правило, содержат основную информацию о бизнесе – кратная информация о компании, услуги и товары, контактные данные, акции и условии акций. Данный способ лэндинга для бизнеса позволяет показать клиенту свой цифровой магазин, одновременно выполняя передачу, хранение и обработку ценной информации. Эта альтернатива привлекает клиента, так как необходимость в трате времени на поездку, простоя в очереди в физическом магазине отпадает.

В рамках данной выпускной квалификационной работы было разработано веб-приложение для автоматизации процесса оформления заказов предназначенное для компании. В момент работы подробно разберутся различные стороны вопроса, начиная от исследования потребностей организации и выбора лучший решений, заканчивая осуществлением и тестированием готового продукта.

Цель данной выпускной квалификационной работы — разработать веб-приложение для компании которое позволит автоматизировать бизнес-процессы, 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.

Сформулированная таким образом цель требует четкой постановки задач:  
1. Изучение бизнес-процессов компании и определение основных задач, которые нужно автоматизировать.

2. Анализ рынка программного обеспечения, предназначенного для автоматизации заказов, и отбор наиболее подходящих вариантов.

3. Создание программного обеспечения для интернет магазина, которое позволит автоматизировать процедуру получения и обработки заказов.

4. Подготовка сотрудников к работе с обновлённой системой.

В процессе работы будут использованы современные решения разработки программного обеспечения, такие как объектно-ориентированное проектирование и использование систем управления базами данных.

# ГЛАВА 1. ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКАЯ ХАРАКТЕРИСТИКА ОРГАНИЗАЦИИ

## 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 проводят 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.** Инженерное Бюро так же предлагает техническое обслуживание вентиляционных систем, обеспечивая при этом бесперебойную работу и продлевая срок службы оборудования. В случае возникновения в установленное время контрактом-договором 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.** Инженерное Бюро 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. Клиенты 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 500-та проектов. Примеры выполненных работ на объектах:

- Муниципальное автономное учреждение Культуры «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 включают:**

*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 улучшать производственные процессы.

Обратившись в компанию «Инженерное Бюро», смело можно доверить работу и быть уверены в том, что система будет установлена и обслуживаться профессионалами. «Инженерное Бюро» предлагает оптимальные решения для создания комфортного микроклимата в помещении и обеспечит долгосрочную и надёжную работу 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 несет всю полноту ответственности за принимаемые решения, устанавливает векторное направление развитие бизнеса.

- Административный отдел: занимается сопровождением руководителя на бизнес-встречах и контроль исполнения поручений. Взаимодействие с отделами и контроль выполнения задач на 90% от месячных планов. Введение журнала всей корреспонденции, внедрение инструментов облегчающие управление организацией для Директора.

- Тендерный отдел: анализ рынка, поиск рентабельных тендеров и подготовка документации для участия в них. Контроль регистраций и продления на сайте государственных закупок, коммерческих площадках. Сбор и формирование всех необходимых документов (декларации, платёжные поручения, независимая гарантия, письмо о деловой репутации и т. д.).

- Производственный отдел: под руководством главного инженера выполняет монтажные работы. Поддерживает связь с административным отделом, по вопросам закупок и обеспечению необходимыми материалами.

- Отдел по работе с клиентами: новый отдел в компании, выпускная квалификационная работа непосредственно и решает вопрос обеспечения новой площадки для отдела. Отдел будет заниматься выполнением заказов клиентов. Взаимодействуя с производственным отделом для выполнения технических заданий и координацией проектов.

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



Рис. 1 - Организационная структура ООО "Инженерное Бюро"

## 1.3. Технологический процесс

В настоящее время в организации автоматизации внутренних процессов нет, по факту при получении заказа от клиента, заявка фиксируется в ручном формате в CRM системах – **Yougile / Контур Закупки**, со временем информация о заказе теряется, что приводит к ряду проблем, решаемых с помощью веб-приложения. Заказы принимаются по средствам личных встреч, телефону или электронной почте. После получения заказа административный отдел обрабатывает заказ, фиксируя заказы вручную в CRM системе, контактируя с директором и производственным отделом для уточнения деталей оборудования / услуги.

Данный подход требует огромных временных затрат на обработку заказов, а так же замедляет скорость принять заказа у клиента. Следует отметить, что форма принятия заказов в таком виде, не исключает ошибок при записи данных о клиенте либо резерва конкретного оборудования за клиентом. Так же взаимодействие между отделами сильно централизованно в административном отделе, из-за отсутствия единой системы управления заказами.

Следует выделить основные проблемы процесса настоящего управления заказами:

**- отсутствие автоматизации внутренних процессов.** Ручной ввод данных о заказе в CRM-системы (Yougile, Контур Закупки) увеличивает вероятность ошибок и потери информации. Это приводит к снижению эффективности работы. Административный отдел обрабатывает заказы вручную, контактируя с директором и производственным отделом для уточнения деталей оборудования или услуги. Это может привести к задержкам в обработке заказов и увеличению времени на их выполнение.

- **сложности в коммуникации между отделами.** Для уточнения деталей заказа административному отделу необходимо связываться с другими подразделениями компании, может приводить к недопониманию и ошибкам. Улучшение коммуникации между отделами улучшит процесс принятия решений и повысит продуктивность работы.

Перечисленные проблемы решаются внедрением веб-приложения в организацию, при условии обучения сотрудников отдела по работе с клиентами. Внедрение автоматизации в рабочие процессы позволит компании ускорить их и принимать больше заказов. Это, в свою очередь, приведёт к росту прибыли компании. Веб-приложение позволит клиентам самостоятельно делать заказы онлайн через адаптивный интерфейс для пользователя, что значительно сократит время на обработку заказов, так как данные будут автоматически попадать в систему. Кроме того, веб-приложение позволит в режиме реального времени проверить, есть ли необходимое оборудование в наличии. На сайте клиенты смогут увидеть актуальную информацию. Привлекать целевую аудиторию с помощью акций станет проще: каждый клиент будет заходить на сайт под своей учётной записью.

Основными функциями веб-приложения будут (Рис. 2):

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

- оформление заказа: веб-приложение с адаптивным интерфейсом автоматически зафиксирует данные о клиенте и заказе, поэтому не нужно будет вносить информацию вручную, что поможет избежать ошибок.

- статус заказа: у клиента будет возможность отследить заказ в реальном времени. Данное решение повышает клиент ориентированность компании и прозрачность процесса.

**Иллюстрация процесса:**



Рис. 2 - Динамические аспекты поведения системы

Использование веб-приложения для управления заказами позволит компании «Инженерное Бюро» существенно оптимизировать рабочие процессы, ускорить выполнение заказов и уменьшить количество ошибок. Клиенты будут довольны более высоким качеством обслуживания. Благодаря прозрачности и скорости обработки заказов компания сможет успешнее конкурировать на рынке.

# ГЛАВА 2. ВЫБОР ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И МЕРОПРИЯТИЯ ПО РЕАЛИЗАЦИИ ВЕБ-ПРИЛОЖЕНИЯ.

## 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. 3).



Рис. 3 - 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 мощная система 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.4).



Рис. 4 - 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. 5).



Рис. 5 - CRM системы

Делая вывод в связи с особыми требованиями бизнеса, предпочтительнее разработать собственное веб-приложение. Это позволит создать систему, которая будет органически подходить компании, а также обеспечит высокий уровень безопасности и гибкость при внедрении новых функций.

На текущий момент существует огромное количество 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 современный 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. Отладка приложения сложная по восприятию, чем традиционные JavaScript-приложения.

**Angular** — это мощный 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.

Альтернативным решением для разработки веб-приложений может стать ASP.NET — современная технология, которая позволяет создавать интерактивные веб-приложения с использованием языка программирования C# и платформы .NET. В частности, ASP.NET MVC является привлекательным выбором благодаря следующим преимуществам:

**1. Четкое разделение ответственности**

ASP.NET MVC использует архитектуру Model-View-Controller (MVC), которая разделяет приложение на три компонента:

- *Модель (Model)*: отвечает за данные и бизнес-логику.

*- Представление (View):* отвечает за отображение данных пользователю.

*- Контроллер (Controller):* управляет взаимодействием между моделью и представлением.

Это разделение упрощает разработку, тестирование и поддержку кода, особенно в крупных проектах.

**2. Серверный рендеринг**

ASP.NET MVC генерирует HTML на стороне сервера, что делает его идеальным для:

- *SEO-оптимизации:* поисковые системы легко индексируют страницы, так как HTML генерируется на сервере.

- *Быстрой начальной загрузки*: пользователь сразу получает готовую HTML-страницу, что улучшает восприятие скорости.

**3. Интеграция с экосистемой .NET**

ASP.NET MVC полностью интегрирован с экосистемой .NET, что позволяет:

- Использовать Entity Framework для работы с базой данных.

- Применять Dependency Injection для управления зависимостями.

- Интегрироваться с другими технологиями Microsoft, такими как Azure, SQL Server и Visual Studio.

**4. Гибкость и расширяемость**

ASP.NET MVC предоставляет множество возможностей для настройки и расширения:

- Поддержка RESTful API для создания веб-сервисов.

- Возможность использования Razor для создания динамических HTML-страниц.

- Поддержка фильтров, маршрутизации и валидации моделей.

**5. Поддержка современных стандартов**

ASP.NET MVC поддерживает современные стандарты и технологии, такие как:

- HTTP/2 и WebSockets.

- gRPC для высокопроизводительных API.

- SignalR для реализации функциональности реального времени.

**6. Кроссплатформенность**

С появлением .NET Core, ASP.NET MVC стал кроссплатформенным. Это означает, что приложения можно разрабатывать и запускать на различных операционных системах таких как:

- Windows

- Linux

- macOS

**7. Простота для .NET-разработчиков**

Если команда разработчиков уже работает на C# и .NET, то ASP.NET MVC будет естественным выбором. Это снижает затраты на обучение и ускоряет разработку.

**8. Поддержка legacy-систем**

ASP.NET MVC отлично подходит для поддержки и модернизации legacy-приложений, созданных на основе классического ASP.NET.

Сравнительный анализ технологий отображен в таблице 2.1.

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | ASP.NET MVC | Blazor | Angular |
| **Тип лицензии** | Открытая (MIT License) | Открытая (MIT License) | Открытая (MIT License) |
| **Язык программирования** | C# | C# | TypeScript/JavaScript |
| **Время на рынке** | С 2009 года (ASP.NET MVC) | С 2018 года (Blazor) | С 2010 года (AngularJS), с 2016 года (Angular 2+) |
| **Работа приложений** | Серверный рендеринг (HTML генерируется на сервере) | Клиентский (WebAssembly) или серверный (Blazor Server) | Клиентский (SPA, выполняется в браузере) |
| **Встроенные функции** | - Маршрутизация - Привязка моделей - Валидация данных - Razor-шаблоны - Поддержка REST API - Интеграция с Entity Framework | - Компонентная архитектура - Поддержка WebAssembly - Встроенная маршрутизация - Dependency Injection - Поддержка SignalR (Blazor Server) | - Компонентная архитектура - Двусторонняя привязка данных - Встроенная маршрутизация - Dependency Injection - Поддержка PWA - Интеграция с RxJS |

Таким образом, ASP.NET MVC представляет собой оптимальный выбор для создания веб-приложений благодаря своей гибкости, масштабируемости и удобству в тестировании и поддержке кода. Этот фреймворк позволит создать качественное и надёжное приложение, соответствующие современным требованиям и стандартам разработки.

## 2.2 Выбор среды разработки

Перед тем как приступить к созданию приложения, следует выбрать среду разработки. В настоящий момент существует множество решений. Интегрированная среда разработки (Integrated Development Environment) – система программных средств, используемых для разработки программного обеспечения.

Следует не забывать, что есть возможность написать код и в текстовом редакторе, но для создания благоприятного процесса и скорости обработки информации, необходимо обратить внимание на IDE.

Функционал IDE будет полезен и удобен для создания веб-приложения основные элементы в единой платформе:

**компилятор** – превращение кода в исполняемый файл;

**интерпретатор** – способствует запуску скриптов, необходимость компиляции отпадает;

**отладчик** – осуществляет быстрый поиск ошибок в коде;

**инструменты автоматизации** – автоматизируется сборка проекта и ускоряет процесс разработки.

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.

Ниже приведен обзор некоторых из существующих на данный момент программ для написания кода.

**IntelliJ IDEA** – мощная интегрированная среда разработки (IDE) для написания программного обеспечения, данная среда разработана крупным игроком на рынке JetBrains.



Рис. 6 - Пользовательский интерфейс IntelliJ IDEA

Благодаря своей функциональности и удобству пользования активно используется в мире Java-разработки. К основным особенностям можно отнести: изначально среда была заточена под Java, со временем она начала поддерживать множество других языков программирования, таких как Kotlin, Scala, Groovy, Python, JavaScript, TypeScript, HTML/CSS, SQL и многие другие. IntelliJ IDEA имеет отличную поддержку множества популярных фреймворков и библиотек, таких как Spring, Hibernate, Android SDK, React, Angular и Vue.js. Это позволяет разработчикам использовать эти технологии без необходимости дополнительной настройки. К минусам можно отнести: IntelliJ IDEA требует значительных вычислительных ресурсов (CPU, RAM), особенно если проекты большие и используются много плагинов, а так же Ultimate Edition является платной, что может быть недоступно для некоторых разработчиков или маленьких компаний (Рис. 6).

**Microsoft Visual Studio** — это интегрированная среда разработки (IDE) от Microsoft, которая оптимально 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 то, что бесплатная версия «Community Edition» для рядового пользователя будет иметь вполне себе обширный функционал, который можно расширить, подключив плагины (Рис. 7).



Рис. 7 - Интерфейс Microsoft Visual Studio

**Visual Studio Code** — кроссплатформенный редактор кода, который при помощи плагинов можно превратить в подобие IDE. Кроссплатформенным его делает то, что он работает на MacOS, Ubuntu и Windows. Данный редактор кода является open-source редактором и плата за плагины не требуется. 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.8).



Рис. 8 - Интерфейс Visual Studio Code

**WebStorm** представляет собой продвинутую интегрированную среду для разработки (IDE), созданную компанией JetBrains, специально ориентированную на создание и поддержку веб-приложений и проектов с использованием JavaScript. Этот инструмент предлагает разработчикам обширный набор функций для разработки современных веб-решений, включая HTML, CSS, JavaScript, TypeScript, а также популярные фреймворки, такие как React, Angular и Vue.js. WebStorm является частью экосистемы продуктов JetBrains и использует ту же платформу, что и другие известные продукты компании, такие как IntelliJ IDEA, PyCharm и PhpStorm. К недостаткам можно отнести: запутанные настройки, начинающему разработчику трубно разобраться с ними, если нет опыта работы с продуктами JetBrains, а так же WebStorm является коммерческим продуктом, и его использование требует покупки лицензии, бесплатная пробная версия выдается на 30 дней (Рис. 9).



Рис. 9 - Интерфейс WebStorm

**MonoDevelop** – бесплатная свободная среда разработки от Xamarin, аналогичная IDE для создания приложений на множестве языков, а так же поддержка основных операционных систем. Возможность кастомизации позволяет пользователям настраивать расположение функций и окон в соответствии с их личными предпочтениями. К сожалению, несмотря на то, что у MonoDevelop есть собственный отладчик и инструменты для работы с кодом, в части поддержки разных платформ и проектов – это пока сырая IDE (Рис.10).



Рис. 10 - Интерфейс MonoDevelop

Таким образом, Microsoft Visual Studio представляет собой оптимальный выбор среды разработки благодаря своей глубокой интеграции с ASP.NET MVC . Эта интеграция обеспечивает плавное и эффективное создание веб-приложений, позволяя разработчикам максимально использовать все преимущества фреймворка ASP.NET MVC , такие как четкое разделение логики, представлений и моделей данных, что способствует созданию масштабируемых и поддерживаемых решений.

Кроме того, Visual Studio предоставляет мощные инструменты для кастомизации и настройки рабочего пространства, что делает процесс разработки более гибким и удобным. Среда разработки также предлагает продвинутые функции отладки, автодополнения кода (IntelliSense) и автоматической проверки ошибок, что значительно ускоряет процесс написания и тестирования кода.

Особого внимания заслуживает экосистема Visual Studio , которая обеспечивает бесшовную интеграцию с другими продуктами Microsoft, такими как Azure для облачного развертывания, GitHub для управления версиями кода и Docker для контейнеризации приложений. Это позволит сосредоточиться на создании качественного программного обеспечения, не отвлекаясь на сложности настройки инфраструктуры.

Сравнение с альтернативными средствами разработки, описанными выше, показывает, что, несмотря на их сильные стороны, они не обеспечивают такого же уровня интеграции с технологиями Microsoft и удобства работы в рамках экосистемы ASP.NET MVC , как Microsoft Visual Studio .

## 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 на данных момент несколько сотни 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.

MS SQL Server имеет лимит на размер базы данных – 10 гб на одну базу, что идеально подходит для небольшого проекта, тестирования продукта и обучения. При успешном использовании приложения, есть возможность перейти на платные версии SQL Server (Standard или Enterprise) для увеличения места и мощности.

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 веб-приложения по обработке заказов была выбрана реляционная база данных MS SQL Server, благодаря широкому набору функциональных возможностей и множеству преимуществ, которые она предлагает для данного типа задач.

Работа с данными в реляционных базах обычно 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 . Рассмотрим 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#. По данным GitHub, он упоминается в 13,405 проектах. 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.

## 2.4 Мероприятия по реализации веб-приложения и введения в эксплуатацию

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. 11). 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. 11 - 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 3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ

## 3.1 Проектирование веб-приложения

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

Концептуальная модель базы данных представлена на Рисунке 12:



Рис. 12 - Концептуальная диаграмма базы данных

**Пояснение к диаграмме:**

**Обозначения связей:**

- || означает "один";

- o{ означает "многие";

- направление стрелок показывает направление связи.

**Основные компоненты схемы:**

**Авторизация ASP.NET Identity:**

- AspNetUsers (пользователи);

- AspNetUserRoles (связи пользователей и ролей);

- AspNetRoles (роли);

- AspNetUserClaims (дополнительные утверждения пользователей);

- AspNetUserLogins (варианты входа пользователей);

- AspNetRoleClaims (утверждения ролей).

**Основные компоненты:**

- Products (продукты);

- ApplicationTypes (типы продуктов);

- Categories (категории);

- Orders (заказы).

Все связи соответствуют указанным в запросе отношениям "один ко многим" и "многие к одному".

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 13.



Рис. 13 - Логическая модель базы данных

**Пояснение к логической схеме:**

**Обозначения в диаграмме:**

- PK - первичный ключ (Primary Key);

- FK - внешний ключ (Foreign Key);

- || означает "один";

- o{ означает "многие".

- стрелки показывают направление связи.

**Типы данных:**

- int - целое число;

- string - текстовая строка;

- boolean - булево значение (да/нет);

- datetime - дата и время;

- decimal - десятичная дробь для денежных значений.

**Структура таблиц:**

**Система пользователей ASP.NET Identity:**

- AspNetUsers - основная таблица пользователей;

- AspNetUserRoles - промежуточная таблица для ролей;

- AspNetRoles - таблица ролей;

- AspNetUserClaims - дополнительные утверждения пользователей;

- AspNetUserLogins - варианты входа пользователей;

- AspNetRoleClaims - утверждения ролей.

**Основные компоненты:**

- Products (продукты);

- ApplicationTypes (типы продуктов);

- Categories (категории);

- Orders (заказы).

Все связи соответствуют логике работы системы: один пользователь может иметь множество ролей, утверждений и вариантов входа, а каждый продукт принадлежит к одной категории.

На основе логической модели, созданной ранее, выполняется детальное представление сущностей в таблице 1:

**Таблица 1. Описание таблиц**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сущность | Предназначение | Ключевые поля | Связи | Ограничения полей |
| AspNetUsers | Пользователи системы | Id (PK), UserName, Email | Связь с AspNetUserRoles (один ко многим)  Связь с AspNetUserClaims (один ко многим)  Связь с AspNetUserLogins (один ко многим)  Связь с Orders (один ко многим) | Id: уникальный идентификатор  UserName: обязательное поле  Email: обязательное поле |
| AspNetUserRoles | Связь между пользователями и ролями | UserId (FK), RoleId (FK) | Связь с AspNetUsers (многие к одному)  Связь с AspNetRoles (многие к одному) | UserId: обязательное поле  RoleId: обязательное поле  Уникальная комбинация UserId и RoleId |
| AspNetRoles | Хранение ролей системы | Id (PK)  Name  NormalizedName | Связь с AspNetUserRoles (один ко многим)  Связь с AspNetRoleClaims (один ко многим) | Id: уникальный идентификатор  Name: обязательное поле  NormalizedName: уникальное поле |
| AspNetUserClaims | Хранение дополнительных утверждений пользователей | Id (PK)  UserId (FK)  ClaimType  ClaimValue | Связь с AspNetUsers (многие к одному) | Id: уникальный идентификаторUserId: обязательное поле  ClaimType: обязательное поле  ClaimValue: обязательное поле |
|  |  |  |  |  |
| AspNetUserLogins | Внешние системы аутентификации | LoginProvider (PK)  ProviderKey (PK)  UserId (FK)  ProviderDisplayName | Связь с AspNetUsers (многие к одному) | LoginProvider: обязательное поле  ProviderKey: обязательное поле  UserId: обязательное поле  Уникальная комбинация LoginProvider и ProviderKey |
| AspNetRoleClaims | Хранение дополнительных утверждений ролей | Id (PK)  RoleId (FK)  ClaimType  ClaimValue | Связь с AspNetRoles (многие к одному) | Id: уникальный идентификаторRoleId: обязательное поле  ClaimType: обязательное поле  ClaimValue: обязательное поле |
| Products | Хранение информации о продуктах | Id (PK)  Name  ShortDesc  Description  Price  Image  CategoryId (FK)  ApplicationTypeId (FK) | Связь с Categories (многие к одному)  Связь с ApplicationTypes (многие к одному)  Связь с Orders (один ко многим) | Id: уникальный идентификатор  Name: обязательное поле  Price: положительное число  CategoryId: обязательное поле  ApplicationTypeId: обязательное поле |
| Categories | Хранение категорий продуктов | Id (PK), Name, DisplayOrder | Связь с Products (один ко многим) | Id: уникальный идентификатор  Name: обязательное поле  DisplayOrder: положительное число |
| ApplicationTypes | Хранение типов продуктов | Id (PK),  Name,  Description | Связь с Products (один ко многим) | Id: уникальный идентификатор  Name: обязательное поле |
| Orders | Заказы клиентов | Id (PK),  Name,  Adress,  Status | Связь с AspNetUsers (многие к одному)  Связь с Products (один ко многим) | Id: уникальный идентификатор  Name: обязательное поле  Adress: обязательное поле  Status: обязательное поле |

Обозначения в таблице:

- PK - первичный ключ;

- FK - внешний ключ;

- "обязательное" означает, что поле не может быть NULL;

- "опциональное" означает, что поле может быть NULL.

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

Следующим этапом после проектирования веб-приложения, будет разработка плана графического интерфейса, ключевой целью является навигация. Навигация должна быть проста и интуитивно понятна, с возможностью перехода к деталям продукта и его заказом. На основе анализа существующих решений на рынке было принято решение использовать следующую структуру:

- 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 разработки 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. 14).



Рис. 14 - Визуальное представление MVC

Так же следует отметить, что веб-приложение было спроектировано по стандартам - *Чистая архитектура* *(Clean Architecture)* представляет собой современный метод проектирования программных решений, направленный на четкое распределение обязанностей между различными частями приложения (Рис. 15).

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. 15 - 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 enables разработчикам описывать структуру данных с помощью классов на языке C#, а сам фреймворк автоматически создает и поддерживает соответствующую схему базы данных на основе этих классов.

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 автоматически создает и поддерживает соответствующую схему базы данных на основе созданных 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; }

}

1. **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

1. **Настройка строки подключения:**

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

{

"ConnectionStrings": {

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

}

}

1. **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. 16 - 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 (Рис. 16).

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

}

**Маршрутизация и навигация**

В *ASP.NET Core MVC* маршрутизация позволяет определять, как URL-адреса сопоставляются с действиями контроллеров. Это делается с помощью атрибутов, таких как [HttpGet], [HttpPost], и директив в представлениях, таких как asp-action и asp-route.

Пример маршрутизации:

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

## 3.3 Интерфейс веб-приложения