Замечания руководителя

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цифровые технологии в современном мире занимают ведущую роль. На их основе реализуется огромное количество задач, в том числе и в экономическом секторе.

Производство окон – бизнес, для которого характерна высокая конкуренция. На рынке присутствует множество компаний, как небольших, так и крупных, которые борются за своего клиента. Но и в такой конкурентной среде можно занять своё «место под солнцем» и сделать бизнес прибыльным. Необходимо изучить рынок, выработать правильную маркетинговую стратегию, сделать необходимые расчёты по затратам и объёмам производства, тщательно составить бизнес-план по производству пластиковых окон.

В современном мире повсеместно используются окна как элемент зданий различного назначения. Правильно подобранные окна обеспечивают не только комфорт в помещении, но и экономическую эффективность.

Одним из возможных способов повышения качества окон является автоматизация их производства. Для этого необходима информационная система, которая позволит управлять производственными процессами: от закупки материалов до отгрузки готовой продукции. Цель данной курсовой работы – разработать информационную систему для производства окон.

Цифровые ресурсы позволили значительно сократить время на сбор и обработку информации, а также её хранения Новый принцип организации управлением на основе применения цифровых программных продуктов значительно упростил решение управленческих задач в экономике за счёт сведения к минимуму таких нежелательных аспектов, как человеческий фактор и трудность обработки большого количества поступающей информации.

1 Анализ предметной области

1.1 Особенности предметной области

Особенности предметной области «Производство окон» заключаются в уникальности и многогранности её деятельности. Два ключевых преимущества данного бизнеса – это его рентабельность и востребованность. Рентабельность обычно составляет 15–20%, однако при правильно выбранной маркетинговой стратегии она может доходить и до 40%. Весь объем вложений окупается в течение года, максимум двух. Высока и востребованность продукта, особенно характерная в летний и осенний сезоны.

Основные потребители услуг – это частные клиенты, муниципальные структуры, а также организации, осуществляющие строительство или капитальный ремонт. Доля владельцев домов и квартир составляет примерно 50% от всех заказов, строителей – 30%, муниципальных – до 20%. По типам оконных конструкций подавляющая часть приходится именно на пластиковые окна (более 75%), 15% рынка занимают алюминиевые конструкции и 9% – деревянные окна.

В производственном процессе задействованы сотни участников: от директоров и менеджеров по проектам до рабочих и строителей, поэтому грамотное управление всеми участниками проекта играет важную роль для достижения успеха

Среди ключевых особенностей предметной области можно выделить следующие элементы:

— Производство окон – это сложный процесс, требующий комбинации множества технических и технологических процессов. Каждый этап производства окон характеризуется своими особенностями и требованиями.

— Проектирование оконных конструкций. Конструкторская документация должна соответствовать всем нормам и требованиям, предъявляемым к оконным системам. Окна должны быть надежными, устойчивыми к различным атмосферным явлениям (ветрам, дождю, морозу и т.д.), долговечными и безопасными для здоровья людей.

— Изготовление самого оконного блока. Основным материалом для производства оконных конструкций является пластик. Он обладает рядом преимуществ перед другими материалами, такими как дерево, алюминий и другие. Пластиковые окна отличаются высокой тепло- и звукоизоляцией, а также не требуют сложного ухода.

— Производство оконных блоков включает в себя такие технологические процессы, как: Резка профилей;Соединение профилей; Установка стекол; Установка фурнитуры; Сборка оконного блока. Останавливаясь подробнее на каждом процессе, можно выделить ряд особенностей и отличительных особенностей.

1. Резка профилей – это один из первых процессов, который предшествует сборке оконного блока. Для резки профилей используется специальное оборудование – пила по металлу или режущий станок. Этот процесс требует высокой точности и соблюдения технологии изготовления, иначе оконный блок может оказаться ненадежным.

2. Соединение профилей – это дополнительный этап, завершающий предобработку профилей перед установкой стекол. Соединение профилей с помощью специального оборудования осуществляется по технологии сварки или клеевого соединения.

3. Установка стекол – это один из самых важных этапов производства окон. Качество установки стекол напрямую влияет на тепло- и звукоизоляцию оконного блока, а также на его стойкость и долговечность. Стекла устанавливаются в специальные камеры, чтобы избежать микротрещин и обеспечить максимальную герметичность конструкции.

4. Установка фурнитуры – это процесс, включающий в себя монтаж замков, ручек, петель и других элементов, обеспечивающих комфортную эксплуатацию окон. Фурнитура должна быть надежной, безопасной и прочной.

5. Сборка оконного блока – это заключительный процесс, когда все элементы оконной конструкции собираются в единое целое. Выполненная работа

должна соответствовать всем стандартам и требованиям, не допуская неточностей и дефектов.

Состав и количество сотрудников в оконном производстве зависит от его масштабов и производственных возможностей. Однако, можно выделить базовую категорию работников, без которых не обходится ни одно оконное производство:

— Руководители и управляющие: директор производства, руководитель отдела продаж, главный инженер;

— Производственный персонал: мастера по производству окон, рабочие по производству окон (столяры, плотники, стекольщики, монтажники), технологи;

— Вспомогательный персонал: бухгалтер, контролёр качества, охранники, погрузчики и т. д.

В среднем, для работы небольшого оконного производства (с производительностью до 50 окон/день) требуется около 10-15 сотрудников. Для средних и крупных производств с производительностью от 50 до 200 окон/день требуется уже 20-50 человек. А для крупных производств, где производительность около 200 и более окон/день, можно говорить уже о сотнях сотрудников. Кроме того, стоит учитывать, что в производстве окон требуется высокая квалификация сотрудников, а также оборудование, позволяющее выпускать окна с требуемыми характеристиками. Поэтому, для успешного функционирования оконного производства необходимо обеспечить команду квалифицированными специалистами и инновационным оборудованием.

При разработке информационной подсистемы «Производство окон» необходимо провести анализ аналогов. Существует множество компаний, занимающихся производством пластиковых окон. Одним из первых на рынке этих изделий была фирма VEKA — немецкий производитель, который привнес многие инновации в производство оконных конструкций. Отличительной особенностью окон VEKA является высокая прочность профиля и отличная термоизоляция благодаря трехкамерной системе.

Drutex — польская компания, производящая окна с 1985 года. Ее продукция отличается высоким качеством, широким ассортиментом вариантов отделки, а

также уникальной технологией производства, которая позволяет создавать окна больших размеров с минимальным количеством стыков.

KBE — еще одна немецкая компания на рынке пластиковых окон. Профиль KBE производится из экологически чистых материалов и отличается долговечностью и высокой шумоизоляцией.

REHAU — компания, которая изначально занималась производством пластиковых труб для систем отопления и водоснабжения. Однако, в последствии она расширила свой ассортимент и начала производство оконных профилей, отличающихся высокой теплоизоляцией и прочностью.

Schuco — немецкий производитель, известный своей инновационной технологией производства пластиковых окон, которая позволяет создавать энергоэффективные окна с минимальными толщинами профилей и стекол. Это позволяет увеличить площадь остекления и улучшить естественное освещение помещения.

Каждый производитель пластиковых окон имеет свои особенности, но в целом, все они обеспечивают высокий уровень термоизоляции и звукоизоляции при небольшой толщине профиля, за счет использования современных материалов и инновационных технологий производства. Также стоит отметить, что у всех производителей есть широкий выбор оттенков профилей и вариантов отделки, что дает возможность подобрать окна, подходящие под конкретный дизайн помещения. Исходя из анализа аналогов, можно сделать вывод, что на данный момент на рынке имеется несколько систем для производства окон и дверей, но некоторые из них имеют ограниченный функционал. При разработке информационной подсистемы «Производство окон» необходимо учитывать все требования и потребности пользователей, а также создать удобный и доступный интерфейс для работы с программой.

1.2 Описание бизнес-процессов в рамках предметной области

Бизнес процесс – это последовательность связанных действий, направленных на достижение определенной цели в рамках деятельности организации. Он должен быть определен, измерен, анализирован и управляем в целях повышения эффективности и конкурентоспособности бизнеса.

Производство окон включает в себя множество этапов и требует специализированного оборудования и квалифицированных кадров. Однако, при правильной организации производства и контроле качества изделий, производство окон может стать высокоэффективным и прибыльным бизнесом.

При выполнении курсового проекта необходимо рассмотреть примеры бизнес процессов в различных отраслях, описать этапы и компоненты, а также рассмотреть основные подходы к управлению бизнес процессами. Важно учитывать организационные особенности конкретной компании и ее индивидуальные потребности. Бизнес-процессы делят на несколько основных групп: поисковые процессы – это процессы, предполагают исследования, генерирование идеи, поиск моделей и методов развития организации; проектный процесс – уникален и может не повторится, например, разработка веб-сайта или внедрение новых стандартов; информационный процесс – процесс, предоставляет информацию, проявляется в разных формах от обмена информации для заключения сделки до обмена идеями по рабочим процессам; производственный процесс – постоянно повторяющийся однотипный процесс в различные временные периоды для создания продукта или услуги.

Бизнес-процессы в компании могут варьироваться в зависимости от ее специализации и масштаба, но можно выделить следующие основные ключевые процессы:

1. Получение заказа. Первым этапом производства окон является получение заказа от клиента. Заказ обрабатывается менеджером, который уточняет требования заказчика и принимает решение о возможности выполнения заказа. Клиент делает заказ через телефон, интернет-магазин или приходит непосредственно в офис. Приём заказа включает в себя:

— Оформление заявки на заказ: в заявке указываются необходимые параметры окна, количество и другие детали.

— Проверка наличия необходимых материалов: после оформления заявки происходит проверка наличия материалов необходимых для производства заказа.

— Расчёт стоимости заказа: после проверки наличия материалов производится расчёт стоимости заказа и сообщение клиенту.

2. Разработка конструкции. После получения заказа менеджер передаёт его проектировщику, который разрабатывает конструкцию окна в соответствии с требованиями заказчика. Для разработки конструкции используется специальное программное обеспечение, которое позволяет создавать трёхмерные модели оконных конструкций.

3. Подготовка производства. После разработки конструкции менеджер передаёт заказ на производство. В данном этапе производится подбор необходимых материалов, инструментов и оборудования, осуществляется настройка программного обеспечения на оборудовании и подготавливается производственное помещение.

4. Изготовление оконных конструкций. На этапе изготовления оконных конструкций происходит ковка рамы, изготовление стеклопакетов и монтажные работы. Специалисты работают на специально оборудованных рабочих местах, выполняют сборку окон и проверяют их соответствие требованиям заказчика.

— Создание проекта заказа: на основе заявки от клиента создаётся проект со всеми необходимыми параметрами.

— Заказ материалов: после создания проекта заказа происходит закупка необходимых материалов для производства.

— Изготовление изделия: на этом этапе изготавливаются все необходимые элементы окна, и происходит их сборка.

5. Контроль качества. После изготовления оконных конструкций происходит их осмотр и контроль качества. Специалисты проверяют размеры

окон, качество монтажных работ, работоспособность фурнитуры и соответствие конструкции требованиям заказчика.

6. Доставка. После контроля качества окна готовы к доставке на объект. Доставка оконных конструкций осуществляется на специализированном транспорте, которое обеспечивает безопасность и сохранность изделий в процессе транспортировки.

7. Монтаж. После доставки окна устанавливаются на месте и производится их монтаж. Для этого используется специализированное оборудование, обеспечивающее точное выравнивание и надёжное крепление оконных конструкций.

8. Сервисное обслуживание. После установки и монтажа окон специалисты оказывают сервисное обслуживание, включающее регулировку фурнитуры, замену уплотнителей и другие работы по поддержанию работы оконных конструкций в исправном состоянии.

Каждый из этих процессов имеет свои подпроцессы и детализацию, которые зависят от конкретных задач и целей компании.

1.3 Проблемы, возникающие в данной предметной области и перспективы их решения с использованием программных средств.

Одной из основных проблем в производстве окон является необходимость проводить многократный ввод одних и тех же данных в различные информационные системы. Например, данные о заявках на производство окон, данные о заявках на установку окон и данные клиентов хранятся в разных базах данных, что затрудняет взаимодействие систем.

Также одной из проблем является сложность управления производственным процессом. В производственном процессе участвуют различные специалисты (конструкторы, разработчики, рабочие), которые должны сопрягать свои действия на различных этапах производства. В такой сложной системе сложно управлять каждым этапом производства.

Для решения проблем, связанных с производством окон, можно использовать программные средства, которые позволят объединить все данные в единую информационную систему и упростят управление на производстве.

Одной из возможных решений является создание информационной подсистемы для производства окон. Такая система позволит вести учёт производства и управлять всеми этапами производства в единой информационной системе. Создание подсистемы производства окон включает в себя следующие основные шаги:

- сбор требований к системе;
- определение основных функциональных возможностей системы;
- проектирование ИС;
- проектирование интерфейса пользователя;
- разработка и тестирование системы.

1.4 Цели и задачи курсового проектирования

Цель данной курсовой работы заключается в разработке и проектировании информационной подсистемы для повышения эффективности и автоматизации производственного процесса в производстве окон. Основной задачей является создание ПО, позволяющего автоматизировать процесс управления и контроля качества производства окон. Для достижения поставленных цели и задач необходимо решить следующие задачи:

1. Анализ существующих методик производства окон, изучение современных технологий и разработка схемы производственного процесса.

2. Определение требований к информационной подсистеме: функциональные и нефункциональные требования.

3. Проектирование концепции информационной подсистемы на основе анализа требований, разработка её структуры и алгоритмов работы.

4. Разработка пользовательского интерфейса для удобного и эффективного взаимодействия с информационной подсистемой.

5. Разработка программного обеспечения для взаимодействия с базой данных и выполнения операций по управлению и контролю качества производства окон.

6. Тестирование и отладка информационной подсистемы.

7. Внедрение и эксплуатация разработанной информационной подсистемы.

2 Моделирование и разработка подсистемы ИС «Производство окон»

2.1 Разработка модели IDEF0

Диаграммы IDEF0 предназначены для описания бизнеспроцессов на предприятии, они позволяют понять, какие объекты или информация служат сырьем для процессов, какие результаты производят работы, что является управляющими факторами и какие ресурсы для этого необходимы. Нотация IDEF0 позволяет выявить формальные недостатки бизнеспроцессов, что существенно облегчает анализ деятельности предприятия.

Для составления контекстной диаграммы системы оконного производства необходимо выделить механизмы, выполняющие работу, средства управления, влияющие на выполнение работы, информацию, которая подаётся на вход системы и используются или преобразуются работой для получения результата (выхода), материал, получающийся в результате функционирования системы(выход).

Механизмы, выполняющие работы – персонал - менеджер по продажам, поставщик, кладовщик, заказчик, сборщик, менеджер по рекламе, доставщик. Средства управления – ГОСТы, регламентирующие производство и продажу окон, Закон РФ о защите прав потребителей, правила внутреннего распорядка, постановления правительства РФ о продаже окон; а также свод нормативных документов в области строительства, принятый органами исполнительной власти, в котором содержатся обязательные требования. На вход подаётся информация и материал – заявка от заказчика. Результат работы системы представляет собой документацию – чеки и другие документы, и готовую продукцию - окно.

На основании выделенных компонентов диаграммы была составлена контекстная диаграмма, представленная на рисунке 1.

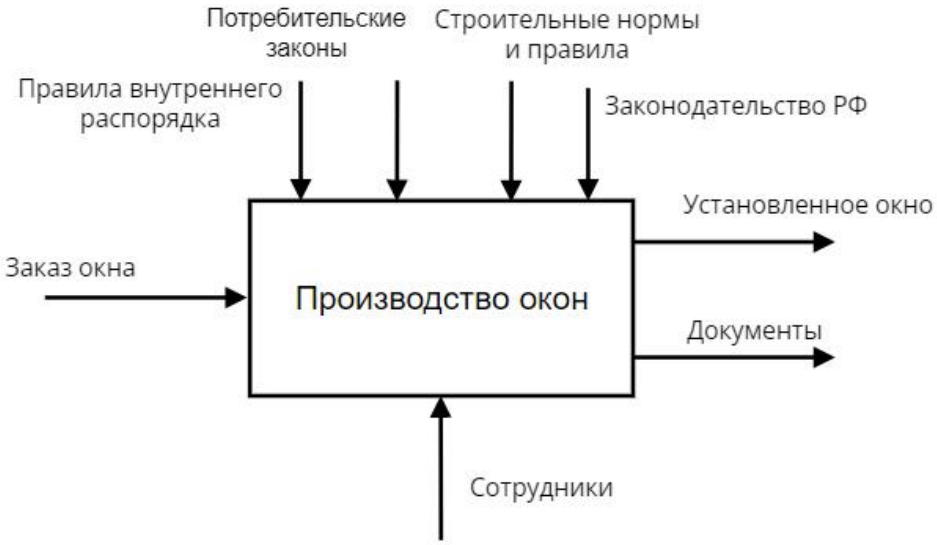

Рисунок 1 – Контекстная диаграмма

Для составления диаграммы модели IDEF0 системы оконного производства необходимо выделить процессы, которые выполняются в системе и их взаимосвязь между собой, механизмы, выполняющие работы, средства управления, влияющие на выполнение работ, информацию, которая подаётся на вход системы и используются или преобразуются работой для получения результата (выхода), материал, получающийся в результате функционирования системы(выход).

Диаграмма первого уровня – декомпозированная диаграмма, на которой крупно показаны основные процессы предприятия, обеспечивающие ее профильную деятельность. Так как декомпозиция – это разложение сложного объекта, на составные части и элементы, то для осуществления декомпозиции необходимо выделить основные элементы рассматриваемой области.

Процессы, выполняющиеся в системе:

1. Выбор модели товара. Данный процесс характеризуется взаимодействием заказчика с каталогом магазина, в котором заказчик выбирает необходимый вид окна. В данном процессе участвует менеджер по продажам, который консультирует заказчика по возникающим вопросам;

2. Производство экземпляров модели. На этом этапе происходит создание необходимого количества товара, выбранной модели, а также в случае

отсутствии ресурсов и сырья для производства, производится обращение к поставщику для получения необходимых материалов.

3. Оформление заказа. На данном этапе происходит оплата заказа, составляется документация, договор о покупке. В оформлении заказа участвуют менеджер по продажам, заказчик и, при условии отсутствия товара на складе, поставщик, которому делается заказ на поставку, и тот отправляет товар в магазин;

4. Подключение услуги доставки товара. На данном этапе после совершения оплаты и наличии окна на складе магазина менеджер по продажам оформляет доставку товара – курьером-доставщиком, либо самовывозом от заказчика.

5. Передача товара клиенту - данный процесс выполняется после завершения всех предыдущих процессов, итогом которого является оформленная документация и наличие купленного товара у заказчика. Здесь участие принимают доставщик, передающий товар заказчику, сам заказчик и установка окна, услуги которого может выбрать заказчик.

На основании выделенных компонентов диаграммы была составлена диаграмма декомпозиции модели IDEF0, представленная на рисунке 2.

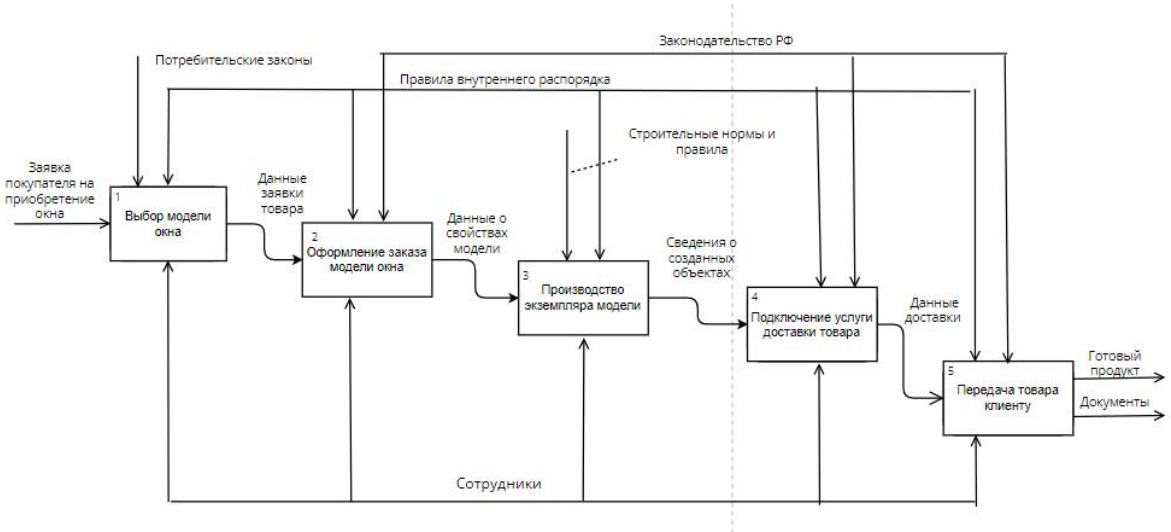

Рисунок 2 – Диаграмма декомпозиции IDEF0

На рисунке 2 можно увидеть, что диаграмма содержит все вышеуказанные выделенные компоненты.

Функционирование системы предполагает, что после получения заявки от заказчика, задействуется персонал предприятия – менеджер по продажам передаёт данные о заказе поставщику, который отправляет продукцию на склад предприятия. Кладовщик передаёт продукцию доставщику, доставщик пересылает окно заказчику. В частных случаях, когда требуется участие установщика, или наоборот, доставщик не требуется, участвующий персонал может изменяться – дополняться или уменьшаться.

2.2 Диаграмма вариантов использования

Диаграмма вариантов использования (англ. use-case diagram) – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей; исходное концептуальное представление или концептуальная модель системы в процессе ее проектирования и разработки. Создание диаграммы вариантов использования имеет следующие цели:

1.Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;

2.Сформулировать общие требования к функциональному поведению проектируемой системы;

3.Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;

4.Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Для составления диаграммы вариантов использования необходимо определить варианты использования системы и дать описание каждому из них:

1. Выбор модели окна. Данный процесс характеризуется взаимодействием клиента с каталогом магазина, в котором клиент выбирает необходимую модель и устанавливает параметры окна.

2. Оформление заказа. Следующий процесс, выполняющийся после выбора модели товара клиентом. На данном этапе происходит оплата заказа, составляется документация, договор о покупке.

3. Подключение услуги доставки товара. На данном этапе после совершения оплаты и наличии модели окна на складе предприятия, укомплектуются мастера установки окон; оформляется доставка товара.

4. Передача товара клиенту. Данный процесс выполняется после завершения всех предыдущих процессов, итогом которого является оформленная документация и наличие купленной модели у клиента.

5. Проверка наличия товара на складе. Данный процесс характеризуется сверкой менеджера по продажам имеющихся на складе магазина товаров вместе с кладовщиком, и в случае отсутствия выбранной модели, оставление заявки на поставку поставщику.

6. Производство экземпляров модели. На данном этапе производится определённое количество моделей окон с необходимыми заявленными параметрами.

7. Поставка ресурсов для сборки. Заказ у поставщика необходимых ресурсов для производства новых экземпляров окон.

На основании выделенных компонентов диаграммы была составлена диаграмма вариантов использования, представленная на рисунке 6.

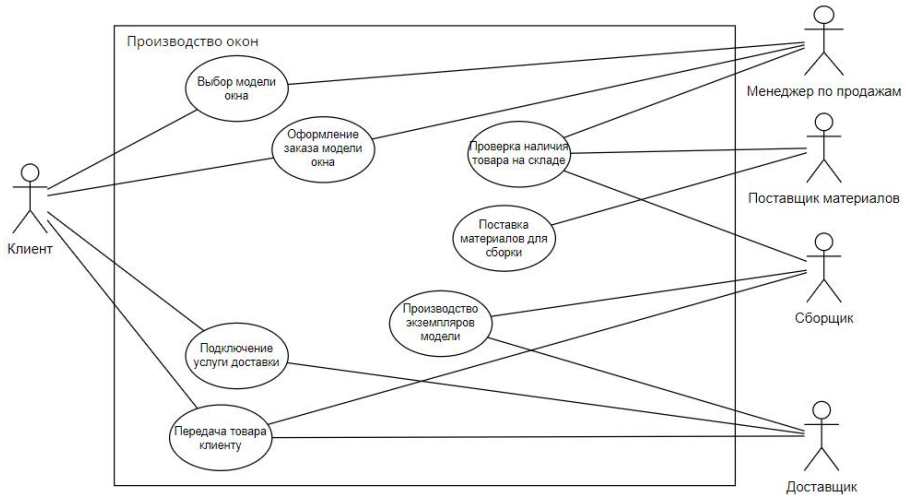

Рисунок 3 – Диаграмма вариантов использования

2.3 Диаграмма классов

Диаграмма классов является инструментом для отображения статической структуры системы, используя терминологию классов объектно-ориентированного программирования. Она позволяет моделировать объекты, из которых состоит система, демонстрировать отношения между объектами и описывать роли, которые эти объекты играют и услуги, которые они предоставляют.

Диаграммы классов широко применяются в объектно-ориентированном моделировании, так как они являются единственными диаграммами UML, которые могут быть отображены непосредственно на объектно-ориентированные языки. В верхней части диаграммы указывается имя класса, посередине располагаются атрибуты класса, а в нижней части содержатся методы класса. Класс может быть связан с одним или несколькими другими классами с помощью различных типов отношений, таких как ассоциация, наследование, реализация, зависимость, агрегация и композиция.

Для выполнения лабораторной работы продолжим работать с производственной системой «Производство окон».

На рисунке 1 представлена диаграмма классов для предприятия «Производство окон». На диаграмме изображены классы, которые содержат: имена, атрибуты и методы:

1. Класс «Клиент». Поля класса: список заказов - тип «Заказ[]», способы оплаты - тип «Оплата[]», email адрес - тип «String», номер телефона - «String». Методы класса: оформить заказ, авторизация, регистрация.

2. Класс «Менеджер по продажам». Поля класса: список клиентов - тип «Клиент[]», производитель - тип «Сборщик». Методы класса: проверить наличие заказа, разработать новую модель, настроить цены.

3. Класс «Сборщик». Поля класса: список заказов - тип «Заказ[]», список материалов - тип «Материал []». Методы класса: собрать модель, проверить наличие материалов, передача заказа клиенту, оформление документации.

4. Класс «Доставщик». Поля класса: наименование компании - тип «String», место доставки - тип «String», транспорт - тип «String». Методы класса: передача заказа клиенту, оформление документации.

5. Класс «Поставщик материалов». Поля класса: наименование компании - тип «String». Методы класса: заказать материалы, доставить материалы.

6. Класс «Оплата». Поля класса: банк провайдер - тип «String», срок действия - тип «Date», CVV-код - тип «String», номер счёта - тип «String». Методы класса: оплатить, пополнить, оформление документации.

7. Класс «Заказ». Поля класса: модель - тип «String», размер - тип «Int», количество пакетов - тип «Int», услуга доставки - тип «Boolean», место выдачи - тип «String», суммарная цена - тип «Decimal». Методы класса: заполнение заявки.

8. Класс «Сотрудник предприятия». Поля класса: ФИО сотрудника- тип «String», код подразделения - тип «Int», заработная плата - тип «Decimal», должность - тип «String». Методы класса: получить сведения.

9. Класс «Отдел предприятия». Поля класса: название отдела - тип «String», местоположение - тип «String», код отдела - тип «Int», телефон - тип «String», электронная почта - тип «String». Методы класса: выплатить зарплату, нанять сотрудника.

10. Класс «Физическое лицо». Поля класса: фамилия - тип «String», имя - тип «String», адрес проживания - тип «String», ИНН - тип «String». Методы класса: получение сведений.

11. Класс «Материал». Поля класса: количество - тип «Int», наименование - тип «String», теплопроводность - тип «Double», прочность - тип «Double». Методы класса: получить характеристики.

Класс «Клиент» и «Заказ» (а также «Оплата») связаны композицией, так как клиент напрямую использует сведения о заказе и не может существовать без этого

класса. Класс «Клиент» наследуется от класса «Физическое лицо», которое определяет базовые операции и поля для всех пользователей системы.

Менеджер по продажам связан с классом «Клиент» агрегацией, так как Менеджер использует в себе класс Клиентов для определения заказов. Остальные классы связаны между собой зависимостью.

На основании выделенных компонентов диаграммы была составлена диаграмма классов, представленная на рисунке 7.

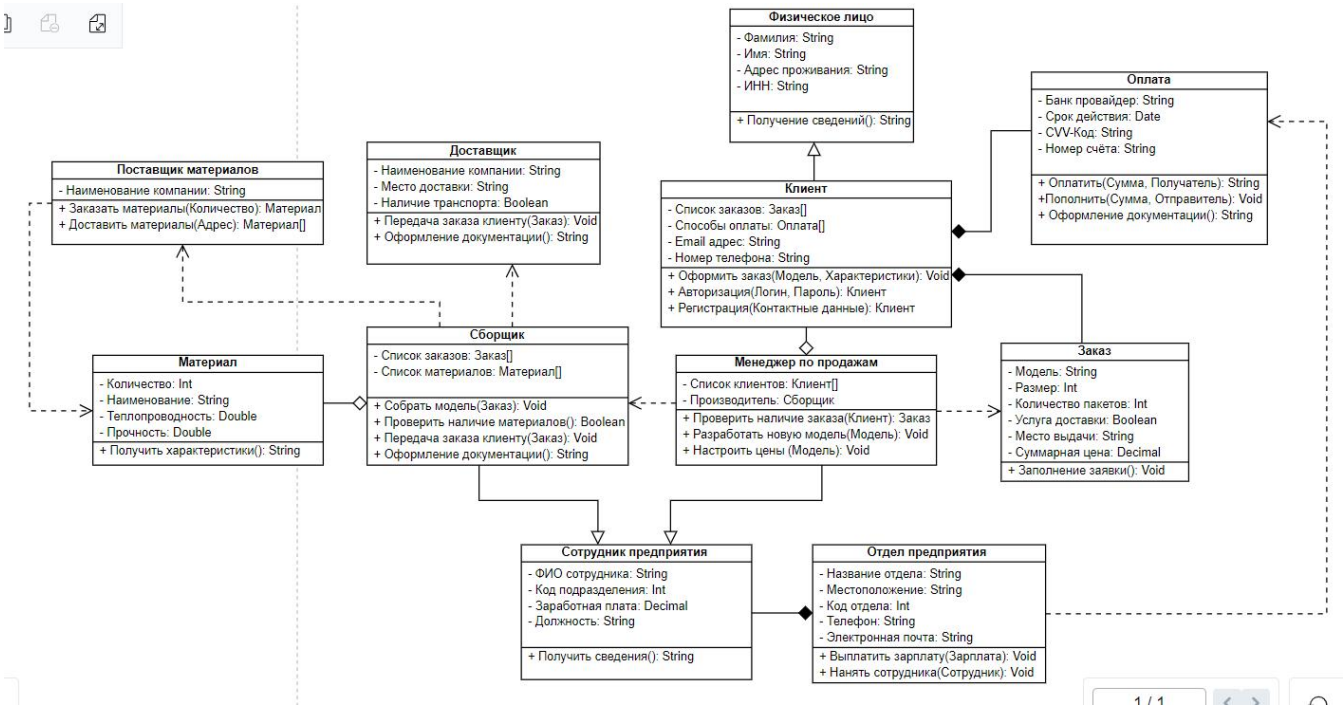

Рисунок 4 – UML диаграмма классов

2.4 Разработка диаграммы последовательностей

Диаграммы последовательностей – это графические модели, которые используются разработчиками для описания взаимодействий между объектами в рамках конкретного сценария использования. Они позволяют проиллюстрировать порядок взаимодействия объектов для выполнения определенных функций, а также описать сообщения, которые передаются между ними.

Диаграммы последовательностей имеют временную шкалу, которая начинается сверху и постепенно опускается, отображая последовательность

взаимодействий. Каждый объект на диаграмме имеет свою колонку, а сообщения между объектами отображаются стрелками.

Для составления диаграммы последовательности необходимо выделить компоненты, которые будут отображены на диаграмме:

1. Клиент;
2. Менеджер;
3. Заказ;
4. Сборщик;
5. Оплата;
6. Поставщик материалов;
7. Материалы;
8. Доставщик;

Каждый объект имеет свою временную линию, изображаемую пунктиром под объектом. Сценарий действий включает в себя:

1. Клиент совершает процесс авторизации.
2. Клиент выбирает модель из каталога и передаёт эти сведения менеджеру.
3. Далее у клиента имеется возможность отслеживать состояние заказа.
4. Менеджер обрабатывает список заказов от клиента и составляет заявку на производство модели.
5. Сборщик обрабатывает заявку от менеджера.
6. Сборщик проверяет наличие необходимых ресурсов для создания модели по клиентской заявке.
7. В случае, если образовался недостаток ресурсов, то сборщик оставляет заявку на материалы у поставщика.
8. Поставщик материалов производит доставку материалов.
9. Сборщик оформляет доставку для клиента.
10. Клиент получает заказ через курьерскую организацию.
11. Клиент оплачивает заказ и в качестве результата оформляется документация.

На основании выделенных компонентов диаграммы была составлена диаграмма последовательности, представленная на рисунке 8.

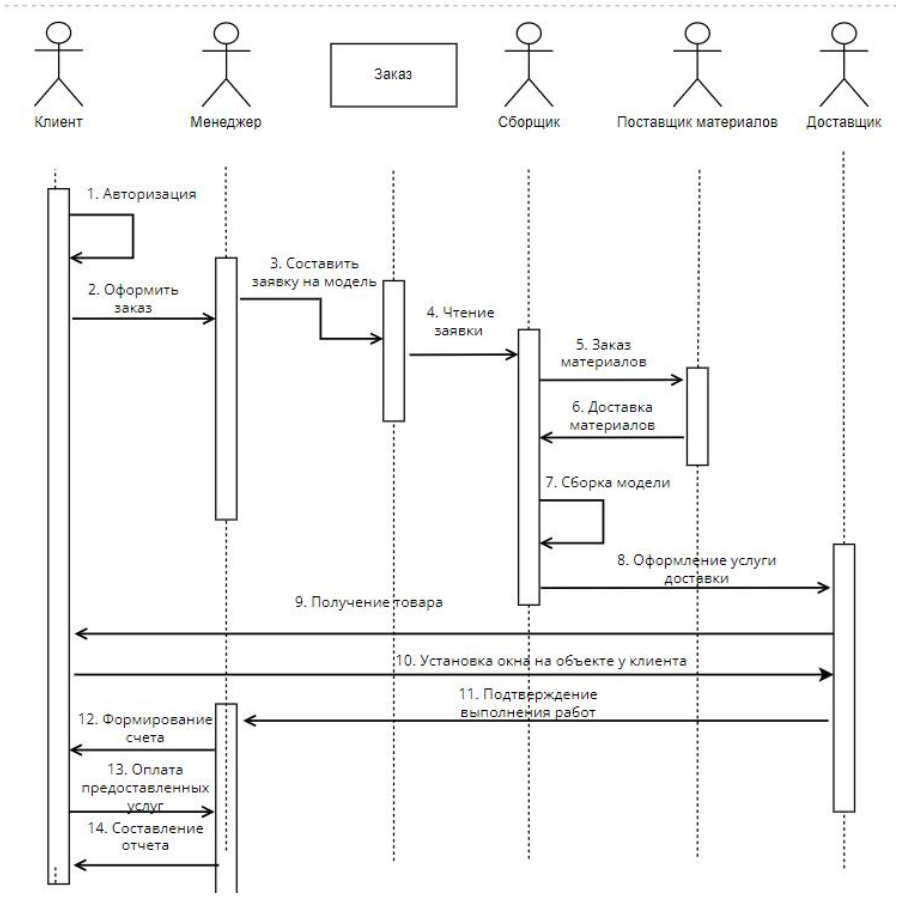

Рисунок 5 – Диаграмма последовательности

2.5 Разработка диаграммы деятельности

Диаграмма деятельности - это одна из диаграмм UML (Unified Modeling Language), которая позволяет описать последовательность действий и процессов, которые происходят в системе. Она используется для моделирования бизнес-процессов и описания алгоритмов, которые используются в системе [9].

Для начала необходимо перечислить действия, которые отображаются на диаграмме деятельности: выбор модели окна, оформление заказа, производство модели, обновление статуса, получение ресурсов со складов, проверка доступных заказов, доставка, оплата заказа, а также установка окна.

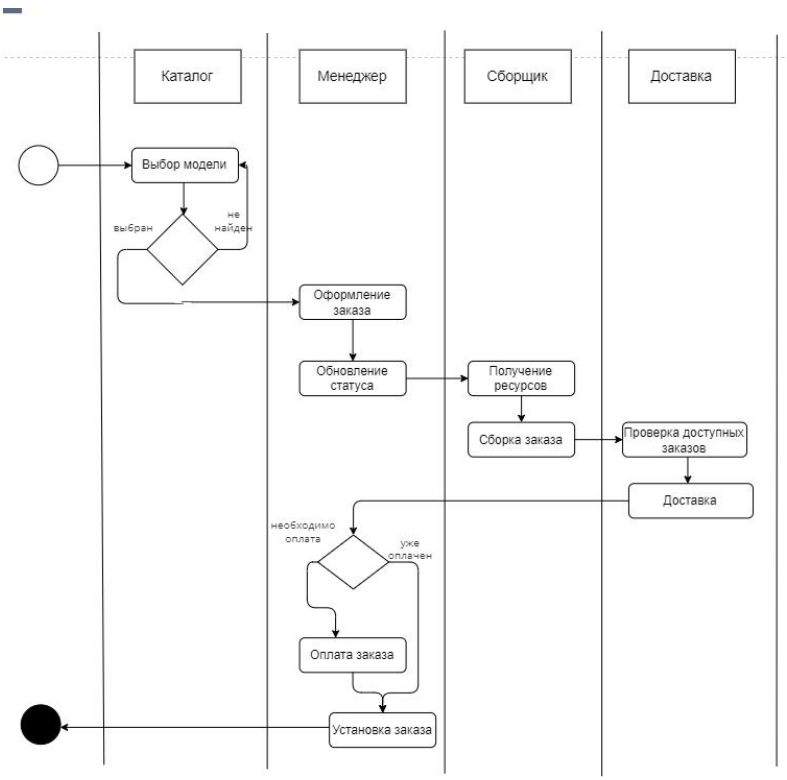

Рисунок 6 – Диаграмма деятельности

На данной диаграмме действия деятельности отражены на четырёх ключевых состояниях заказа в подсистеме: каталог сервиса, страница менеджера и страница доставщика, а также профиль сборщика окон.

2.6 Проектирование базы данных

В процессе работы программного обеспечения происходит неоднократное обращение к базе данных для получения, обновления и внесения новых данных. Для понимания взаимодействия данных, спроектируем физическую модели данных. Шаги проектирования физической модели БД:

1. Определение схемы БД - определение таблиц, связей между ними и атрибутов каждой таблицы.

2. Определение типов данных - выбор наиболее подходящего типа данных для каждого атрибута.

3. Определение ключевых полей - выбор поля или полей, которые будут использоваться для уникальной идентификации записей в таблице.

4. Определение ограничений - определение ограничений на поля таблицы (например, ограничение на ввод пустого значения).

В процессе моделирования применяемой базы данных был разработан и представлен следующий список основных сущностей:

— «Authorization» - отвечает за хранение сведений авторизации пользователей разрабатываемой системы;

— «Accounts» - представляет собой сущность, которая содержит контактные данные о каждом пользователе приложения;

— «Clients» - представляет собой описание клиентов;

— «Payments» - представляет собой сведения средств оплаты;

— «ResourceProviders» - отвечает за хранение данных о поставщиках;

— «Resources» - отвечает за хранение данных о ресурсах, хранящихся на складе производства;

— «Employees» - отвечает за хранение данных о сотрудниках предприятия;

— «Orders» - описывает сведения об оформляемых клиентских заказах.

Рисунок 7 – Физическая модель базы данных

3. Разработка приложения информационной системы

3.1 Используемые средства разработки информационной системы

Одним из важнейших этапов реализации системы является выбор средства разработки. От этого зависят как возможности системы, её быстродействие, так и поддерживаемые платформы. Средства разработки программного обеспечения - совокупность приёмов, методов, методик, а также набор инструментальных программ, используемых разработчиком для создания программного кода.

.NET 7 - это платформа программирования, разрабатываемая компанией Microsoft. Она включает в себя языки программирования, библиотеки, средства разработки, которые позволяют создавать приложения для ОС Windows. Она обеспечивает возможность создавать масштабируемые и быстрые приложения, которые выполняются на любой поддерживаемой платформе. Ниже приведены причины использования платформы .NET:

1. Большое сообщество разработчиков: .NET является одной из самых крупных платформ в мире, и это означает, что в случае возникновения проблемы или нужды в совете, всегда можно найти помощь в различных сообществах.

2. Безопасность: Высокий уровень безопасности является одним из знаковых преимуществ платформы .NET. Платформа NET использует систему кодовой подписи, что обеспечивает безопасность кода.

3. Обширные библиотеки классов: .NET имеет обширную библиотеку классов, включающую множество готовых решений для решения актуальных вопросов в разработке.

4. Высокая производительность: .NET обеспечивает высокую производительность благодаря тому, что является компилирующейся платформой.

В целом, платформа .NET является мощным инструментом для разработки высокопроизводительных, масштабируемых и безопасных приложений.

Для разработки информационной системы можно использовать следующие средства:

1. Visual Studio - интегрированная среда разработки (IDE), которая позволяет создавать приложения на платформе .NET. Разработчики могут использовать различные языки программирования, такие как C#, Visual Basic, F# и др. Одним из важнейших этапов реализации системы является выбор средства разработки. От этого зависят как возможности системы, её быстродействие, так и поддерживаемые платформы. Средства разработки программного обеспечения - совокупность приёмов, методов, методик, а также набор инструментальных программ, используемых разработчиком для создания программного кода.

2. ASP.NET - фреймворк для создания веб-приложений на платформе .NET. Он предоставляет множество инструментов и функциональности, которые облегчают процесс создания веб-приложений.

3. Entity Framework - ORM (Object-Relational Mapping) фреймворк, который упрощает работу с базами данных на платформе .NET. Он позволяет разработчикам работать с данными, используя объектно-ориентированный подход.

4. PostgreSQL - реляционная СУБД, которая является открытым и бесплатным программным обеспечением. Она позволяет хранить, обрабатывать и управлять данными в информационной системе на dotnet asp net entity framework.

Выбор этих средств разработки обусловлен следующими позможностями. Использование Visual Studio позволяет ускорить процесс разработки, так как она предоставляет широкий спектр инструментов для создания, отладки и развертывания приложений. Применение в процессе разработки платформу ASP.NET позволяет создавать высокопроизводительные веб-приложения с меньшим количеством кода. Entity Framework упрощает работу с базами данных и позволяет разработчикам сфокусироваться на разработке приложения, не тратя много времени на написание SQL-запросов. PostgreSQL является открытым программным обеспечением, что позволяет сократить расходы на покупку лицензий и использовать его для разработки коммерческих приложений. Кроме того, PostgreSQL обладает хорошей производительностью и надежностью, что очень важно для информационных систем.

3.2 Разработка программного обеспечения

Программное обеспечение будет разработано с целью предоставления клиентам ИС возможности в формате онлайн осуществлять заказ и оплату услуги производства оконных моделей, а также будет обеспечивать доставщиков, сборщиков и менеджеров соответствующими необходимыми инструментами для управления отдельными частями систем, информацией о состояниях оформленных заказов.

Информационная подсистема будет иметь следующие примерные технические требования для сервера баз данных и ПК пользователя.

Технические характеристики серверов:

- процессор: Intel® Xeon® E5450 2,60 ГГц;
- объем оперативной памяти: 32 Гб;
- жесткие диски: Общий объем памяти 1ТБ;
- сетевой адаптер:250 Мбит/с;
- ОС: Windows 2016 Server.

Технические характеристики ПК пользователя:

- тактовая частота процессора – 800 МГц;
- объем оперативной памяти – 512 Мб;
- количество свободного места на диске – 350 Мб;
- версия DirectX1 или выше.

Веб-приложение будет реализовано в программном интерфейсе ASP.NET, что может позволит пользователям системы использовать различные устройства и операционные системы с установленным на них браузерами. Разделение на классы необходимо для более удобной модульной разработки программного обеспечения Далее опишем основной функционал классов и модулей разрабатываемой системы, представленный в таблице 1.

Таблица 1 - Описание классов и модулей программы

Название	**Описание**
ProfileController	Используется всеми UI компонентами системы для управлением автоматической переадресацией, чтобы использовать авторизацию и аутентификацию на основе Cookie.
AuthorizationFilter	Применяется обработчиками запросов контроллеров авторизации, для использования валидации передаваемых с запросом данными, чтобы в дальнейшем была возможность отображения сообщений пользователю.
ProfileFilter	Применяет обработчиками запросов контроллеров настройки пользовательских контактных данных, для использования функции валидации значений запроса.
Models	Пространство имён, которое содержит определение модели (множество сущностей - классов) для взаимодействия с БД через технологию ORM
ProfileService	Применяется для упрощения взаимодействия с таблицами БД, хранящие значения аккаунтов пользователей системы.
BuilderPages	Группа страниц, которая отвечает за пользователя системы «сборщик»
ClientPages	Группа страниц, которая отвечает за пользователя системы «клиент»
DeliveryPages	Группа страниц, которая отвечает за пользователя системы «доставщик»
ProviderPages	Группа страниц, которая отвечает за пользователя системы «поставщик»
ManagerPages	Группа страниц, которая отвечает за пользователя системы «менеджер»

Таблица 2 - Описание методов программы

Название	**Описание**
ProfileService.UpdateAccount	Применяется для изменения данных таблицы БД, ответственной за хранение контактных данных пользователя.
ProfileService.UpdateEmployee	Применяется для изменения данных таблицы БД, ответственной за хранение данных сотрудников производства.
DatabaseContext.OnModelerCreate	Настраивает подключение к БД, используя для этого конфигурацию приложения.
HandlerError.OnExecuted	Производит проверку данных запросы перед процессом обработки запроса
Profile.Login	Производит авторизацию пользователя, и настраивает конфигурационные куки
Profile.Logout	Описывает процесс выхода из учётной записи пользователя, а также удаляет права на посещение страниц.
Client.OnGetAsync	Производит проверку доступа и рендеринг представления для страницы пользователя приложения
Provider.OnGetAsync	Производит проверку доступа и рендеринг представления для страницы поставщика
Manager.OnGetAsync	Производит проверку доступа и рендеринг представления для страницы менеджера приложения, и загрузку данных о сотрудниках производства.
Builder.OnGetAsync	Производит проверку доступа и рендеринг представления для страницы сборщика предприятия, а также загружает склад.

3.3 Пример работы программного обеспечения

Программное обеспечение встречает пользователя окном авторизации, где предлагается войти в аккаунт или создать новую учётную запись, которая будет соответствовать одному из пользователю системы (рисунок 8).

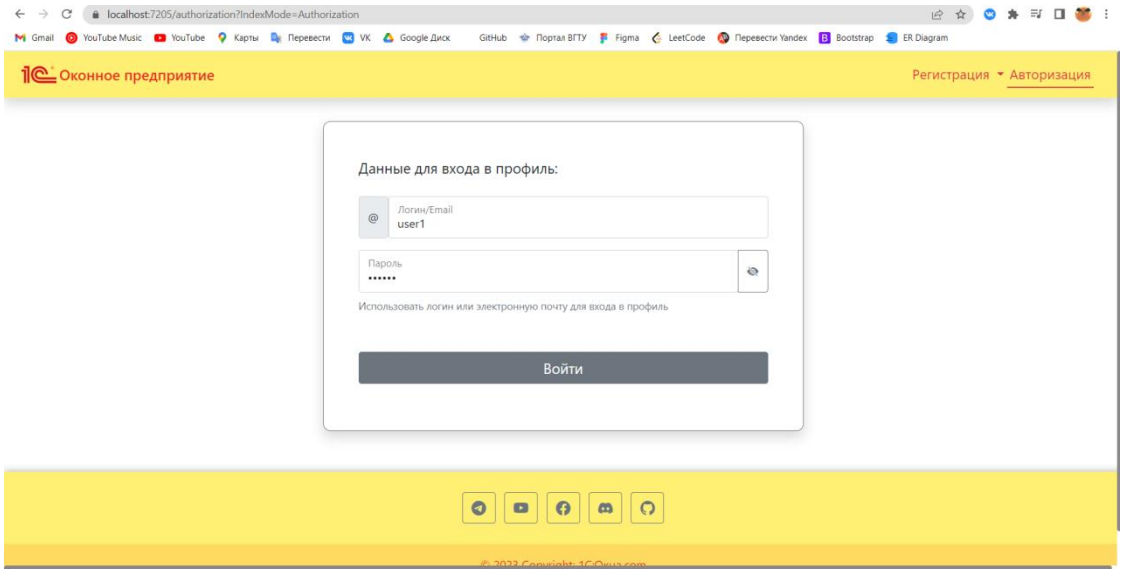

Рисунок 8 – Окно авторизации

А также можно провести регистрацию для 5 типов пользователей системы, что показано на рисунке 9.

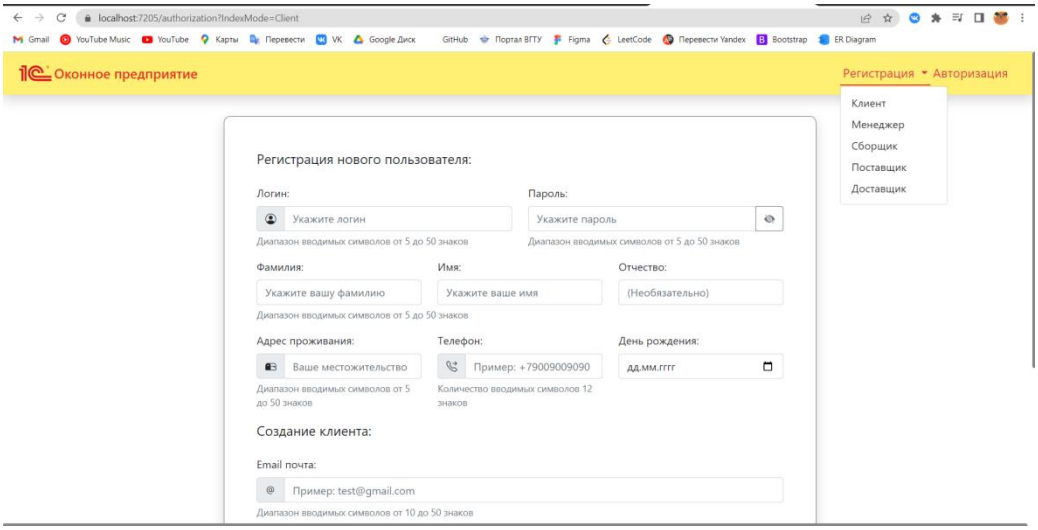

Рисунок 9 – Окно регистрации

Сначала проведём заказ окон как клиент. Для этого нужно войти с помощью логина и пароля, после чего, если в системе будет найден пользователь с данными контактными данными, то будет выполнена переадресация (рисунок 10).

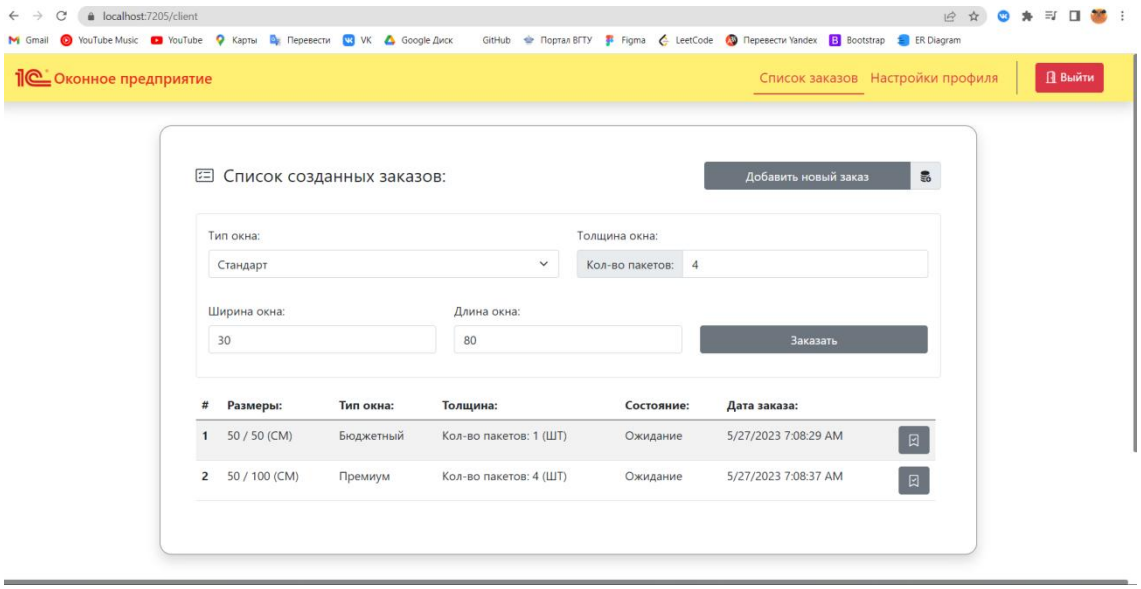

Рисунок 9 – Оформление заказа окна

Каждому пользователю системы, доступна вкладка «Настройка профиля», где открывается возможность для настройки данных профиля, что продемонстрировано на рисунке 10.

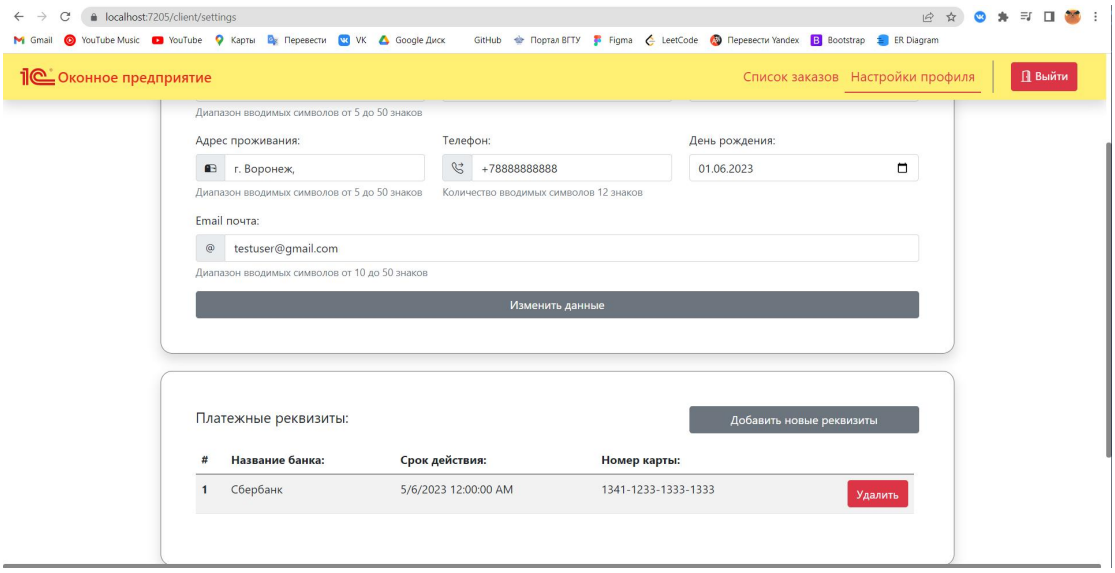

Рисунок 10 – Настройка профиля

После чего рассмотрим следующего пользователя системы - «Менеджер», который обладает возможностью управления штатом сотрудников, т.е.подтверждать заявки на работу других пользователей, а также контролировать свой штат сборщиков, которым он может назначать работу, что показано на рисунке 11.

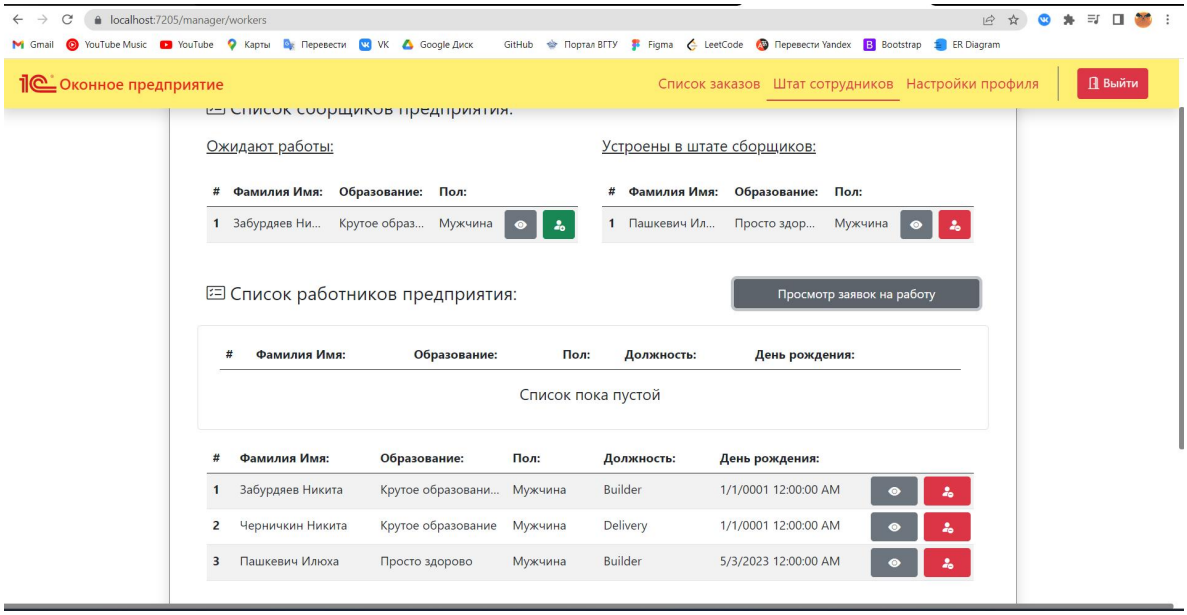

Рисунок 11 – Вкладка управления штатом сотрудников

Также менеджер обладает полномочиями управления заявками клиентов на создание окон, используя для этого вкладку «Список заказов» (рисунок 12)

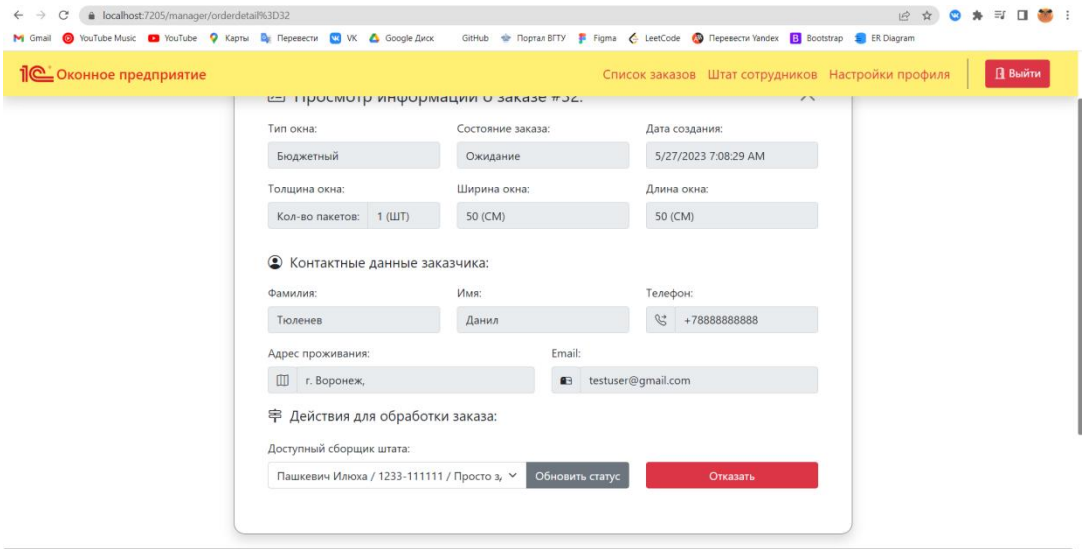

Рисунок 12 – Управление заявками клиентов

При создании профиля сотрудника системы, пользователь не может использовать функции, до того момента, пока его заявку не обработает другой пользователь системы, с ролью «Менеджер» (рисунок 13).

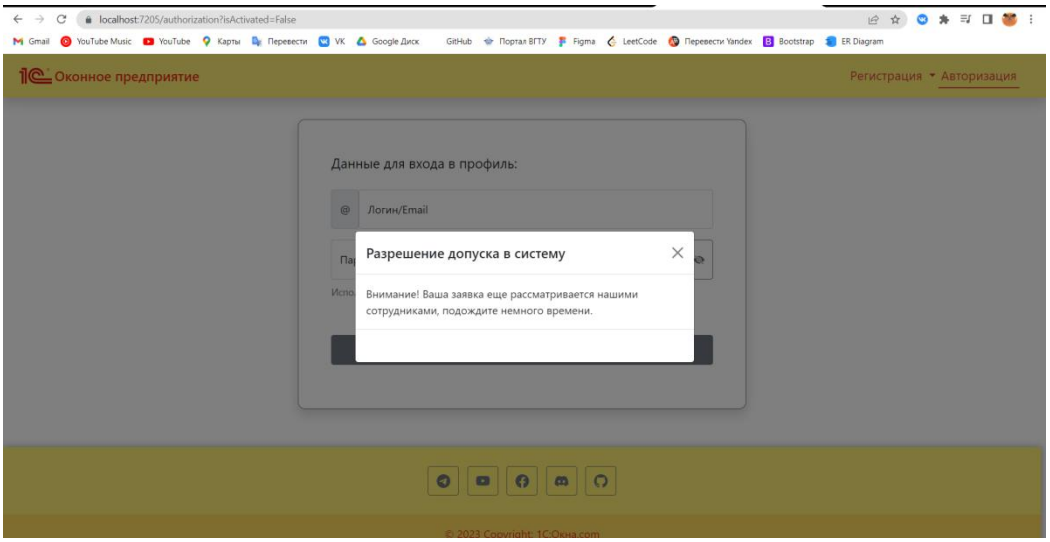

Рисунок 13 –Ожидание предоставления доступа

После того, как заявка пользователя через менеджера была передана сборщику, последний может обновить её статус до «Готово к отправке», т.е. произведёт сборку необходимой модели, при условии, если будет найдено необходимое количество ресурсов (рисунок 14).

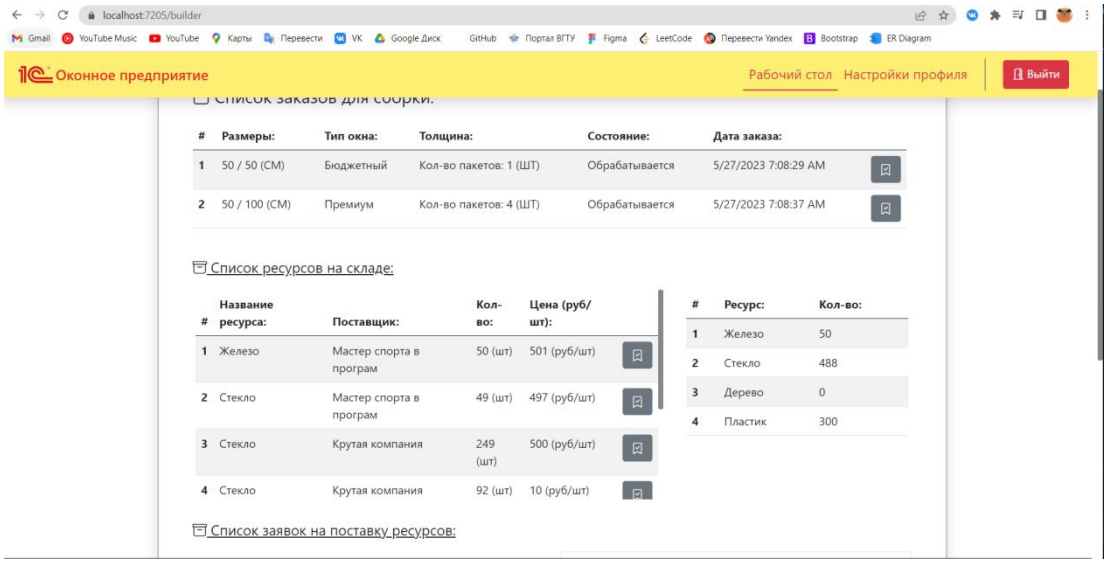

Рисунок 14 – Рабочий стол пользователя «Сборщик»

Если ресурсов оказалось недостаточно, то сборщик формирует заявку на добавление ресурсов, которую может обработать пользователь с роль «Поставщик», что показано на рисунке 15.

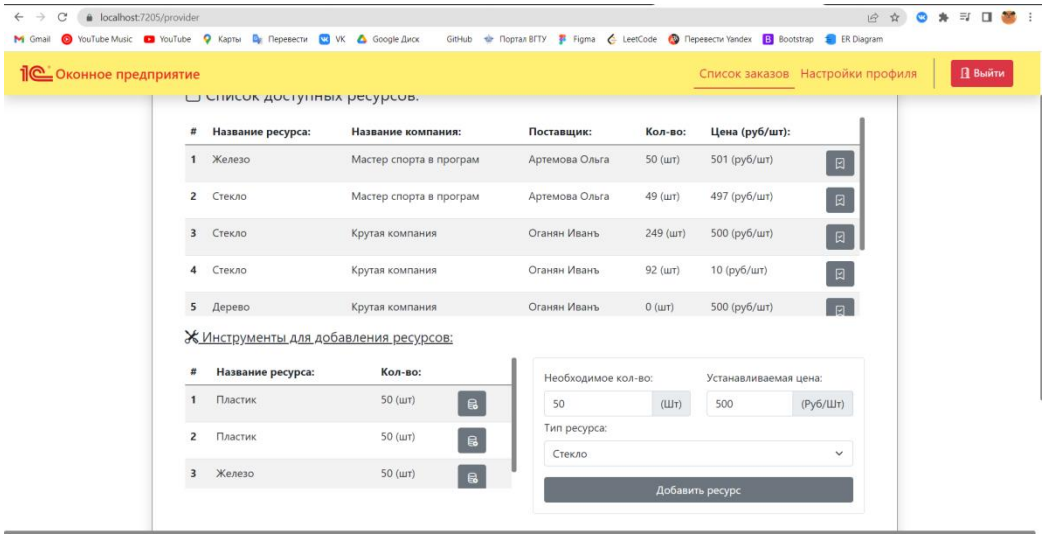

Рисунок 15 – Обработка заявок на ресурсы

После того, как заказ клиента прошёл все стадии обработки, клиент может получить заказ, выбрав предварительно при оформлении способ оплаты. Как результат клиент получить чек и готовый товар, что показано на рисунках 16 и 17.

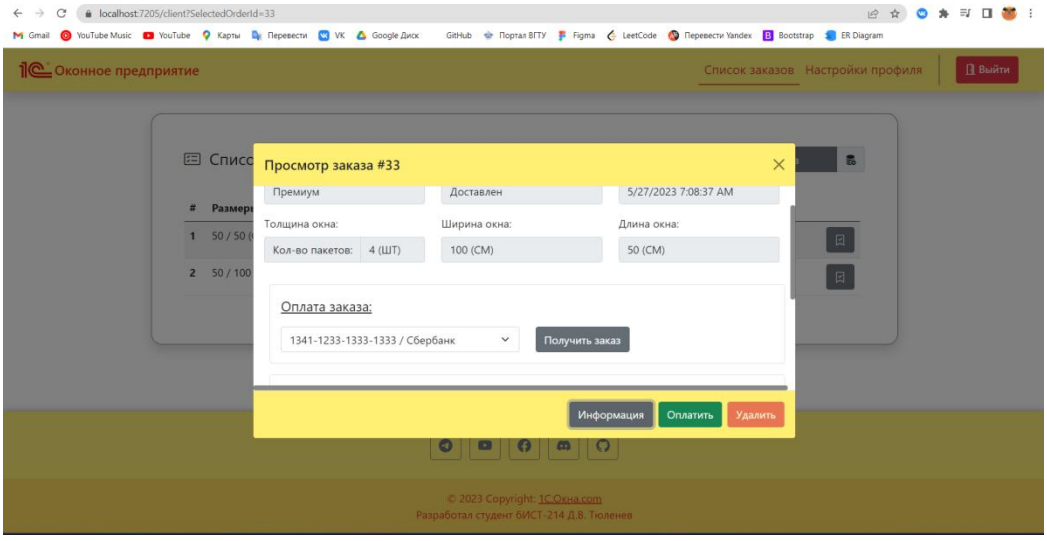

Рисунок 16 – Оплата и получение заказа

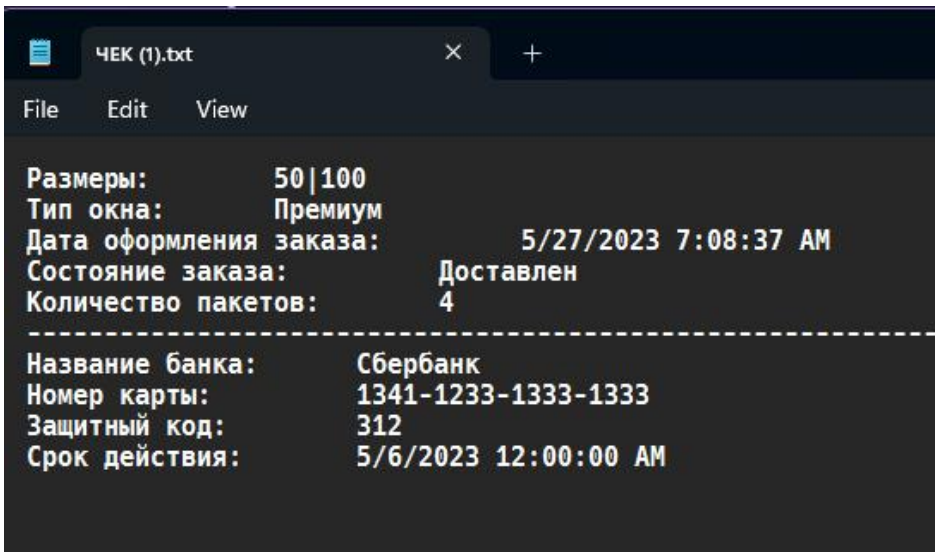

Рисунок 17 – Товарный чек

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта по разработке информационной подсистемы «Производство окон» с помощью унифицированного языка моделирования UML были соблюдены все поставленные цели и задачи, разработано приложение для автоматизации покупки окон в магазине, реализовано отображение всех данных у клиента, менеджера и других сотрудников производства.

В процессе выполнения курсового проекта были пройдены такие этапы как:

- анализ предметной области и аналогов;
- моделирование информационной подсистемы с помощью унифицированного языка моделирования UML;
- разработано программное обеспечение.

Во время анализа предметной области были определены основные тонкости и аспекты работы бизнес-процессов в данной сфере. При моделировании информационной системы мы поняли как нам более удобно, конкретно и определено подобраться к разработке программного обеспечения. Были спроектированы следующие диаграммы:

- диаграммы вида IDEF0;
- диаграмма вариантов использования;
- диаграмма классов;
- диаграмма деятельности;
- диаграмма последовательности;
- физическая модель базы данных.

Благодаря предшествующим анализу и моделированию, разработка программного обеспечения проходила легко, по заранее спрогнозированному сценарию. Это позволило выполнить все поставленные цели и задачи курсового проектирования.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Д. Н. Куликов. Производство металлопластиковых конструкций. Основы технологии и оборудование. — М.: Грамота, 2012. — 448 с.

2. А. И. Ложкин. Технология производства пластиковых оконных конструкций. — М.: Издательство «Инфра-М», 2015. — 304 с.

3. Е. А. Беликова, Н. В. Мироненко. Производство конструкций из алюминия и ПВХ. — М.: Грамота, 2011. — 304 с.

4. С. А. Алексеев. Технология производства пластиковых окон и дверей. — М.: Издательство «Экзамен», 2013. — 256 с.

5. М. В. Данилов, О. Б. Конькова. Алюминиевые конструкции. Конструирование, рассчет, технология производства. — М.: БиблиоГлобус, 2012. — 416 с.

6. Буч Г., Рамбо Д., Якобсон И. Введение в UML от создателей языка. 2-е изд.: Пер. с англ. Мухин Н. – М.: ДМК Пресс, 2015. – 496 с

7. Самохвалов Э.Н. Введение в проектирование и разработку приложений на языке программирования C# / Э.Н. Самохвалов, Г.И. Ревунков, Ю.Е. Гапанюк. учебное пособие. - М.:МГТУ 2018. - 244с.

8. Богданов М. С., Шалыто А. А. Сравнение “Sequence Diagram” и диаграмм “State Chart”.

9. Зеленко О. В., Валеева Л. Р., Климанов С. Г. Обзор современных Web-технологий //Вестник Казанского технологического университета. – 2015. – Т. 18. – №. 2. – С. 354-356.

10. Бабушкина Н. Е., Рачев А. А. ОРГАНИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ ENTITY FRAMEWORK С БАЗОЙ ДАННЫХ //Юность и знания-гарантия успеха-2020. – 2020. – С. 303-305.

11. Моргунов Е. П., Рогова Е. В., Лузанова П. В. PostgreSQL. Основы языка SQL //учеб. пособие/ЕП Моргунов. – 2018.

Приложение А

```
using Microsoft.AspNetCore.Authorization.Infrastructure;
using Microsoft.EntityFrameworkCore;
using System.Diagnostics;
using System.Text.RegularExpressions;
using WebApplication.Models;

namespace WebApplication.Services
{
    public interface IProfileService : IAsyncDisposable
    {
        public abstract Task<ProfileError?> UpdateAccount(Account account, int profileId);
        public abstract Task<ProfileError?> UpdateEmployee(Employee employee, int
profileId);
        public record ProfileError(string ErrorMessage);
    }
    public partial class ProfileService : System.Object, IProfileService
    {
        protected readonly ILogger<ProfileService> Logger = default!;
        protected IDbContextFactory<MispisCourseworkContext> DatabaseFactory { get; set; }
= default!;

        public ProfileService(IDbContextFactory<MispisCourseworkContext> factory,
ILogger<ProfileService> logger)
            : base() { this.Logger = logger; this.DatabaseFactory = factory; }
        public ValueTask DisposeAsync() => ValueTask.CompletedTask;

        public async Task<IProfileService.ProfileError?> UpdateAccount(Account account,
int profileId)
        {
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                var profileRecord = await dbcontext.Accounts.FirstOrDefaultAsync(item =>
item.Accountid == profileId);
                if (profileRecord == null) return new
IProfileService.ProfileError("Аккаунт не найден");

                profileRecord.Surname = account.Surname; profileRecord.Name =
account.Name;
                profileRecord.Patronymic = account.Patronymic;

                profileRecord.Address = account.Address; profileRecord.Phonenumber =
account.Phonenumber;
                profileRecord.Birthday = account.Birthday;

                await dbcontext.SaveChangesAsync();
            }
            return default(IProfileService.ProfileError);
        }
        public async Task<IProfileService.ProfileError?> UpdateEmployee(Employee employee,
int profileId)
        {
            var validatePassport = employee.Passport != null &&
Regex.IsMatch(employee.Passport, @"^[0-9]{4}-[0-9]{6}$");
            if (!validatePassport) return new IProfileService.ProfileError("Неверный
паспорт");

            if (employee.Education == null || employee.Education.Length < 5)
            {
                return new IProfileService.ProfileError("Неверные данные об образовании");
            }
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
```

```
            {
                await dbcontext.Employees.Where(item => item.Accountid ==
profileId).ExecuteUpdateAsync(item =>
                item.SetProperty(p => p.Passport, p => employee.Passport)
                    .SetProperty(p => p.Gender, p => employee.Gender).SetProperty(p =>
p.Education, p => employee.Education));

                await dbcontext.SaveChangesAsync();
            }
            return default(IProfileService.ProfileError);
        }
    }
}
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.FileProviders;
using System.Security.Claims;
using WebApplication.Models;
using WebApplication.Pages;
using WebApplication.Services;

namespace WebApplication
{
    using ASPNETBuilder = Microsoft.AspNetCore.Builder;
    public partial class Program : object
    {
        public static void Main(string[] args)
        {
            var builder = ASPNETBuilder::WebApplication.CreateBuilder(args);

            builder.Services.AddRazorPages(); builder.Services.AddControllers();

builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme).Add
Cookie(options =>
            {
                options.AccessDeniedPath = new PathString("/authorization");
                options.LoginPath = new PathString("/authorization");
            });
            builder.Services.AddAuthorization(options =>
            {
                options.AddPolicy("Client", item => item.RequireClaim(ClaimTypes.Role,
IndexMode.Client.ToString()));
                options.AddPolicy("Provider", item => item.RequireClaim(ClaimTypes.Role,
IndexMode.Provider.ToString()));

                options.AddPolicy("Delivery", item => item.RequireClaim(ClaimTypes.Role,
IndexMode.Delivery.ToString()));
                options.AddPolicy("Manager", item => item.RequireClaim(ClaimTypes.Role,
IndexMode.Manager.ToString()));
                options.AddPolicy("Builder", item => item.RequireClaim(ClaimTypes.Role,
IndexMode.Builder.ToString()));
            });
            builder.Services.AddDbContextFactory<MispisCourseworkContext>(options =>
            {

options.UseNpgsql(builder.Configuration.GetConnectionString("DefaultConnection"));
            });
            builder.Services.AddTransient<IProfileService, ProfileService>();

            var application = builder.Build();
            application.Map("/", (HttpContext context) =>
Results.LocalRedirect("/profile/login"));

            application.UseHttpsRedirection().UseStaticFiles();
            application.UseStaticFiles(new StaticFileOptions()
```

```
            {
                FileProvider = new
PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), @"Icons")),
                RequestPath = new PathString("/icons")
            });
            application.UseRouting().UseAuthentication().UseAuthorization();

            application.MapControllers(); application.MapRazorPages();
            application.Run();
        }
        public Program() : base() { }
    }
}
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using WebApplication.Pages;
using Microsoft.AspNetCore.Authorization;
using Microsoft.EntityFrameworkCore.Internal;
using Microsoft.EntityFrameworkCore;
using WebApplication.Models;

namespace WebApplication.Controller
{
    [ControllerAttribute, RouteAttribute("profile")]
    public partial class ProfileController : Microsoft.AspNetCore.Mvc.Controller
    {
        protected ILogger<ProfileController> Logger { get; set; } = default!;
        public ProfileController(ILogger<ProfileController> logger) : base()
{ this.Logger = logger; }

        [RouteAttribute("login", Name = "login"), HttpGetAttribute, AuthorizeAttribute]
        public async Task<IActionResult> Login([FromServices]
IDbContextFactory<MispisCourseworkContext> factory)
        {
            var profileId =
int.Parse(this.HttpContext.User.FindFirst(ClaimTypes.PrimarySid)!.Value);
            using (var dbcontext = await factory.CreateDbContextAsync())
            {
                var record = await dbcontext.Accounts.FirstOrDefaultAsync(item =>
item.Accountid == profileId);
                if (record != null) return
this.RedirectToPage($"/{record.Profiletype}Pages/{record.Profiletype}Page");
            }
            return await this.Logout();
        }
        [RouteAttribute("logout", Name = "logout"), HttpGetAttribute]
        public async Task<IActionResult> Logout()
        {
            await
this.HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
            return base.LocalRedirect("/");
        }
    }
}
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using System.Text.RegularExpressions;
using WebApplication.Pages;

namespace WebApplication.Filters
{
    [AttributeUsage(AttributeTargets.Class)]
```

```
    public partial class AuthorizationFilterAttribute : Attribute, IPageFilter
    {
        protected IndexMode RedirectMode { get; set; } = default;
        public AuthorizationFilterAttribute() : base() { }

        protected virtual IActionResult? HandlerError(string errorMessage, bool
validateValue)
        {
            if (validateValue == true) return default(IActionResult);
            return new RedirectToPageResult("Index", new { ErrorMessage = errorMessage,
IndexMode = RedirectMode });
        }
        public void OnPageHandlerExecuting(PageHandlerExecutingContext context)
        {
            if (context.HttpContext.Request.Method != "POST") return;
            Predicate<string?> checkTextField = (string? value) => value != null &&
value.Length >= 5;

            this.RedirectMode =
Enum.Parse<IndexMode>(context.HttpContext.Request.Form["IndexMode"].FirstOrDefault()!);
            IActionResult? validationResult = default;
            var validationList = new Dictionary<string, string?>()
            {
                { "Неверный логин",
context.HttpContext.Request.Form["AuthorizationModel.Login"].FirstOrDefault() },
                { "Неверный пароль",
context.HttpContext.Request.Form["AuthorizationModel.Password"].FirstOrDefault() },
            };
            if(this.RedirectMode != IndexMode.Authorization)
            {
                validationList.Add("Неверная фамилия",
context.HttpContext.Request.Form["AccountModel.Surname"].FirstOrDefault());
                validationList.Add("Неверное имя",
context.HttpContext.Request.Form["AccountModel.Name"].FirstOrDefault());
                validationList.Add("Неверное адрес проживания",
context.HttpContext.Request.Form["AccountModel.Address"].FirstOrDefault());
            }
            if (this.RedirectMode == IndexMode.Provider)
            {
                validationList.Add("Неверное имя компании",
context.HttpContext.Request.Form["ProviderModel.Companyname"].FirstOrDefault());
                validationList.Add("Неверный адрес офиса",
context.HttpContext.Request.Form["ProviderModel.Address"].FirstOrDefault());
            }
            else if ((IndexMode.Manager | IndexMode.Builder |
IndexMode.Delivery).HasFlag(this.RedirectMode))
            {
                validationList.Add("Ошибка с образованием",
context.HttpContext.Request.Form["EmployeeModel.Education"].FirstOrDefault());
            }
            foreach (KeyValuePair<string, string?> record in validationList)
            {
                if ((validationResult = this.HandlerError(record.Key,
checkTextField(record.Value))) != null)
                { context.Result = validationResult; return; }
            }
            var phoneValue =
context.HttpContext.Request.Form["AccountModel.Phonenumber"].FirstOrDefault();
            var emailValue =
context.HttpContext.Request.Form["ClientModel.Emailaddress"].FirstOrDefault();
            var passportValue =
context.HttpContext.Request.Form["EmployeeModel.Passport"].FirstOrDefault();

            var validatePhone = phoneValue == null || Regex.IsMatch(phoneValue, @"^\+7[0-
9]{10}$");
```

```
            if ((validationResult = this.HandlerError("Неправильный формат номера
телефона", validatePhone)) != null)
            { context.Result = validationResult; return; }

            var validateEmail = emailValue == null || Regex.IsMatch(emailValue,
@"^\w{6,}@(mail|gmail|yandex){1}.(ru|com){1}$");
            if ((validationResult = this.HandlerError("Неправильный формат email",
validateEmail)) != null)
            { context.Result = validationResult; return; }

            var validatePassport = passportValue == null || Regex.IsMatch(passportValue,
@"^[0-9]{4}-[0-9]{6}$");
            if ((validationResult = this.HandlerError("Ошибка паспорта",
validateEmail)) != null)
            { context.Result = validationResult; return; }
        }
        public void OnPageHandlerExecuted(PageHandlerExecutedContext context) { }
        public void OnPageHandlerSelected(PageHandlerSelectedContext context) { }
    }
}
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.Extensions.Primitives;
using System.ComponentModel.DataAnnotations;
using System.Text.RegularExpressions;

namespace WebApplication.Filters
{
    [AttributeUsageAttribute(AttributeTargets.Class | AttributeTargets.Method)]
    public partial class ProfileFilterAttribute : Attribute, IPageFilter
    {
        protected string FailedRedirect { get; private set; } = default!;
        public string? MethodValidate { get; set; } = default;
        public ProfileFilterAttribute(string failureLink) : base()  { this.FailedRedirect
= failureLink; }

        protected virtual IActionResult? HandlerError(string errorMessage, bool
validateValue)
        {
            if (validateValue == true) return default(IActionResult);
            return new RedirectToPageResult(this.FailedRedirect, new { ErrorMessage =
errorMessage });
        }
        public void OnPageHandlerExecuting(PageHandlerExecutingContext context)
        {
            var handler = context.HttpContext.Request.Query["handler"].FirstOrDefault();
            if (handler != this.MethodValidate || context.HttpContext.Request.Method !=
"POST") return;

            Predicate<string?> checkTextField = (string? value) => value != null &&
value.Length >= 5;

            IActionResult? validationResult = default;
            var validationList = new Dictionary<string, string?>()
            {
                { "Неверная фамилия",
context.HttpContext.Request.Form["Surname"].FirstOrDefault() },
                { "Неверное имя",
context.HttpContext.Request.Form["Name"].FirstOrDefault() },

                { "Неверное адрес проживания",
context.HttpContext.Request.Form["Address"].FirstOrDefault() },
            };
            foreach(var record in validationList)
            {
```

```
                if ((validationResult = this.HandlerError(record.Key,
checkTextField(record.Value))) != null)
                { context.Result = validationResult; return; }
            }
            var phoneValue =
context.HttpContext.Request.Form["Phonenumber"].FirstOrDefault();
            var validatePhone = phoneValue == null || Regex.IsMatch(phoneValue, @"^\+7[0-
9]{10}$");

            if ((validationResult = this.HandlerError("Неправильный формат номера
телефона", validatePhone)) != null)
            { context.Result = validationResult; return; }
        }
        public void OnPageHandlerExecuted(PageHandlerExecutedContext context) { }
        public void OnPageHandlerSelected(PageHandlerSelectedContext context) { }
    }
}
@page "/authorization"
@model IndexModel
@{
    const string underlineClass = "border-bottom border-2 border-danger";
    this.ViewData["Title"] = "Главная страница";

    var gendersList = new SelectList(new string[] { "Мужчина", "Женщина", "Не
указывать" });
    var transportsList = new SelectList(new string[] { "Легковая машина", "Грузовая
машина", "Велосипед", "Мотоцикл", "Самокат" });
}
@section Supports {
<ul class="navbar-nav my-3 my-sm-0 my-lg-0 navbar-nav-scroll"
    style="--bs-scroll-height:84px;font-size:14pt;">
    <li class="nav-item">
        <div class="dropdown nav-link text-danger p-1 @((this.Model.IndexMode !=
IndexMode.Authorization ? underlineClass : ""))">
            <a class="text-danger text-decoration-none dropdown-toggle" role="button"
data-bs-toggle="dropdown">
                Регистрация
            </a>
            <ul class="dropdown-menu">
                <li><a class="dropdown-item" asp-page="Index" asp-route-
IndexMode="@IndexMode.Client">Клиент</a></li>
                <li><a class="dropdown-item" asp-page="Index" asp-route-
IndexMode="@IndexMode.Manager">Менеджер</a></li>
                <li><a class="dropdown-item" asp-page="Index" asp-route-
IndexMode="@IndexMode.Builder">Сборщик</a></li>
                <li><a class="dropdown-item" asp-page="Index" asp-route-
IndexMode="@IndexMode.Provider">Поставщик</a></li>
                <li><a class="dropdown-item" asp-page="Index" asp-route-
IndexMode="@IndexMode.Delivery">Доставщик</a></li>
```

```
            </ul>
        </div>
    </li>
    <li class="nav-item">
        <a     class="nav-link     text-danger     p-1     @((this.Model.IndexMode     ==
IndexMode.Authorization ? underlineClass : ""))"
            asp-page="Index"                                                    asp-route-
IndexMode="@IndexMode.Authorization">Авторизация</a>
    </li>
</ul>
}
@section Scripts {
<script id="password-hideshow" type="text/javascript">
    $(this.document).ready(function() {
        var changeClass = (added, deleted) => {
            $('#password-icon').addClass(added); $('#password-icon').removeClass(deleted);
        };
        var isPasswordHide = false;
        $('#button-addon').click(function(){
            if(isPasswordHide == false) {
                $('#password').attr('type', 'text'); changeClass('bi-eye-fill', 'bi-eye-
slash-fill');
            }
            else  {  $('#password').attr('type',  'password');  changeClass('bi-eye-slash-
fill', 'bi-eye-fill'); }
            isPasswordHide = !isPasswordHide;
        });
    });
    $(window).on('load', () => $('#myModal').modal('show'));
</script>
}

@if(this.Model.IsAccountActivated == false)
{
<div class="modal fade" id="myModal" tabindex="-1">
    <div class="modal-dialog modal-dialog-centered">
        <div class="modal-content">
            <div class="modal-header">
                <h1 class="modal-title fs-5" id="exampleModalLabel">Разрешение допуска в
систему</h1>
                <button  type="button"  class="btn-close"  data-bs-dismiss="modal"  aria-
label="Close"></button>
            </div>
```

```
            <div class="modal-body">
                Внимание! Ваша заявка еще рассматривается нашими сотрудниками, подождите немного времени.
            </div>
            <div class="modal-footer"> </div>
        </div>
    </div>
</div>
}
<div class="container p-0 my-5">
    <div class="row justify-content-center">
    @if(this.Model.HasError == true)
    {
    <div class="mb-4 alert alert-danger alert-dismissible fade show" role="alert" style="border-radius:16px;">
        <i class="bi bi-exclamation-triangle-fill"></i>
        <strong>Уведомление!</strong> @this.Model.ErrorMessage
        <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
    </div>
    }
    @if (this.Model.IndexMode == IndexMode.Authorization)
    {
        <div class="col-12 col-sm-6 shadow border border-secondary p-5" style="border-radius:10px;">
            <form asp-page="Index" asp-page-handler="Login" method="post" asp-antiforgery="true">

                <input type="hidden" asp-for="IndexMode" value="@this.Model.IndexMode" />

            <label class="form-label mb-4 fs-5 fw-semibold"> Данные для входа в профиль: </label>
            <div class="input-group mb-3 has-validation">
                <span class="input-group-text" id="basic-addon">@@</span>
                <div class="form-floating flex-grow-1 mx-0">
                    @this.Html.TextBox("AuthorizationModel.Login", "", new {@class=$"form-control {(this.Model.HasError ? "is-invalid" : "")}",
                        placeholder="Логин/Email", maxlength="50"})
                    <label for="login">Логин/Email</label>
                </div>
            </div>
            <div class="input-group mb-2 has-validation">
                <div class="form-floating flex-grow-1 @(this.Model.HasError ? "is-invalid" : "")">
```

```
                    @this.Html.TextBox("AuthorizationModel.Password",       "",       new
{id="password", @class=$"form-control {(this.Model.HasError ? "is-invalid" : "")}",
                        type="password", placeholder="Пароль", maxlength="50"})
                    <label for="password">Пароль</label>
                </div>
                <button   class="btn   btn-outline-secondary"   type="button"   id="button-
addon">
                    <i id="password-icon" class="bi bi-eye-slash-fill"></i>
                </button>
            </div>
            <div id="passwordHelpBlock" class="form-text mb-4">
                Использовать логин или электронную почту для входа в профиль
            </div>
            <div class="form-text mb-2 text-danger">
                    @if(this.Model.HasError)   {   <text>@(this.Model.ErrorMessage   ??
"Неверные данные")</text> }
            </div>
            <div class="d-flex justify-content-center mx-auto mb-3">
                <input class="btn btn-secondary w-100 fs-5" type="submit" value="Войти"/>
            </div>

            </form>
        </div>
    }
    else {
        <div class="col-12 col-sm-8 shadow border border-secondary p-5" style="border-
radius:10px;">
            <label   class="form-label   mb-4   fs-5   fw-semibold">   Регистрация   нового
пользователя: </label>
            <form   asp-page="Index"   asp-antiforgery="true"   method="post"   asp-page-
handler="Registration">
                <input type="hidden" asp-for="IndexMode" value="@this.Model.IndexMode" />
                <div class="row mb-3">
                    <div class="col-12 col-sm-6">
                        <label class="form-label">Логин:</label>
                        <div class="input-group has-validation">
                            <span    class="input-group-text"><i    class="bi    bi-person-
circle"></i></span>
                            <input       class="form-control"       type="text"       asp-
for="AuthorizationModel.Login" maxlength="50" placeholder="Укажите логин"/>
                        </div>
                        <div  class="form-text">Диапазон  вводимых  символов  от  5  до  50
знаков</div>
```

```
                    </div>
                    <div class="col-12 col-sm-6">
                        <label class="form-label">Пароль:</label>
                        <div class="input-group has-validation">
                            <input id="password" class="form-control" type="password"
asp-for="AuthorizationModel.Password" maxlength="50" placeholder="Укажите пароль"/>
                            <button class="btn btn-outline-secondary" type="button"
id="button-addon">
                                <i id="password-icon" class="bi bi-eye-slash-fill"></i>
                            </button>
                        </div>
                        <div class="form-text">Диапазон вводимых символов от 5 до 50
знаков</div>
                    </div>
                </div>
                <div class="row mb-3">
                    <div class="col-12 col-sm-4">
                        <label class="form-label">Фамилия:</label>
                        <input class="form-control" type="text" asp-
for="AccountModel.Surname" maxlength="50" placeholder="Укажите вашу фамилию"/>
                    </div>
                    <div class="col-12 col-sm-4">
                        <label class="form-label">Имя:</label>
                        <input class="form-control" type="text" asp-
for="AccountModel.Name" maxlength="50" placeholder="Укажите ваше имя"/>
                    </div>
                    <div class="col-12 col-sm-4">
                        <label class="form-label">Отчество:</label>
                        <input class="form-control" type="text" asp-
for="AccountModel.Patronymic" maxlength="50" placeholder="(Необязательно)"/>
                    </div>
                    <div class="form-text">Диапазон вводимых символов от 5 до 50
знаков</div>
                </div>
                <div class="row mb-3">
                    <div class="col-12 col-sm-4">
                        <label class="form-label">Адрес проживания:</label>
                        <div class="input-group has-validation">
                            <span class="input-group-text"><i class="bi bi-
mailbox2"></i></span>
                            <input class="form-control" type="text" asp-
for="AccountModel.Address" maxlength="50" placeholder="Ваше местожительство"/>
                        </div>
```

```
                        <div class="form-text">Диапазон вводимых символов от 5 до 50
знаков</div>
                    </div>
                    <div class="col-12 col-sm-4">
                        <label class="form-label">Телефон:</label>
                        <div class="input-group has-validation">
                            <span class="input-group-text"><i class="bi bi-telephone-
forward"></i></span>
                            <input class="form-control" type="text" asp-
for="AccountModel.Phonenumber" maxlength="12" placeholder="Пример: +79009009090"/>
                        </div>
                        <div class="form-text">Количество вводимых символов 12
знаков</div>
                    </div>
                    <div class="col-12 col-sm-4">
                        <label class="form-label">День рождения:</label>
                        <input class="form-control" type="date" asp-
for="AccountModel.Birthday"/>
                    </div>
                </div>
        @switch(this.Model.IndexMode) {
            case IndexMode.Client:

                <label class="form-label mb-4 fs-5 fw-semibold"> Создание клиента:
</label>
                <div class="row mb-3">
                    <label class="form-label">Email почта:</label>
                    <div class="input-group has-validation">
                        <span class="input-group-text"><i class="bi bi-at"></i></span>
                        <input class="form-control" type="text" asp-
for="ClientModel.Emailaddress" maxlength="50" placeholder="Пример: test@gmail.com"/>
                    </div>
                    <div class="form-text">Диапазон вводимых символов от 10 до 50
знаков</div>
                </div>

            break;
            case IndexMode.Builder:
                <label class="form-label mb-4 fs-5 fw-semibold"> Создание сборщика:
</label>

                await this.RenderEmployeeForm(gendersList);
            break;
```

```
            case IndexMode.Manager:
                <label  class="form-label  mb-4  fs-5  fw-semibold">  Создание  менеджера:
</label>
                await this.RenderEmployeeForm(gendersList);
            break;
            case IndexMode.Provider:
                <label  class="form-label  mb-4  fs-5  fw-semibold">  Создание  поставщика:
</label>
                <div class="row mb-3">
                    <div class="col-12 col-sm-6">
                        <label class="form-label">Название компании:</label>
                        <input        class="form-control"        type="text"        asp-
for="ProviderModel.Companyname" maxlength="50" placeholder="Введите название"/>
                    </div>
                    <div class="col-12 col-sm-6">
                        <label class="form-label">Адрес офиса:</label>
                        <input        class="form-control"        type="text"        asp-
for="ProviderModel.Address" maxlength="50" placeholder="Введите адрес"/>
                    </div>
                </div>
            break;
            case IndexMode.Delivery:
                <label  class="form-label  mb-4  fs-5  fw-semibold">  Создание  доставщика:
</label>
                await this.RenderEmployeeForm(gendersList);
                <div class="row mb-3">
                    <div class="col-12 col-sm-6">
                        <label class="form-label">Водительская лицензия:</label>
                        <input        class="form-control"        type="text"        asp-
for="DeliverymanModel.Driverlicence"    maxlength="50"    placeholder="Введите    данные
лицензии"/>
                    </div>
                    <div class="col-12 col-sm-6">
                        <label class="form-label">Тип транспорта:</label>
                        @this.Html.DropDownListFor(item                                  =>
item.DeliverymanModel.Transporttype, transportsList, new {@class="form-select"})
                    </div>
                </div>
            break;
        }
                <input    type="submit"    class="btn    mt-3    w-100    btn-secondary"
value="Зарегистрироваться"/>
            </form>
```

```
        </div>
    }
    </div>
</div>
@functions {
    public async Task RenderEmployeeForm(SelectList gendersList)
    {
        <div>
        <div class="row mb-3">
            <div class="col-12 col-sm-6">
                <label class="form-label">Паспорт:</label>
                <input  class="form-control"  type="text"  asp-for="EmployeeModel.Passport"
maxlength="11" placeholder="Пример: 0000-000000"/>
            </div>
            <div class="col-12 col-sm-6">
                <label class="form-label">Пол:</label>
                <div class="input-group has-validation">
                    <span   class="input-group-text"><i   class="bi   bi-universal-access-
circle"></i></span>
                    @this.Html.DropDownListFor(item     =>     item.EmployeeModel.Gender,
gendersList, new {@class="form-select"})
                </div>
            </div>
        </div>
        <div class="row mb-3">
            <div class="col-12">
                <label class="form-label">Образование:</label>
                <input class="form-control" type="text" asp-for="EmployeeModel.Education"
maxlength="50" placeholder="Укажите образование"/>
            </div>
            <div class="form-text">Диапазон вводимых символов от 5 до 50 знаков</div>
        </div>
        </div>
    }
}
@page "/delivery"
@model WebApplication.Pages.DeliveryPages.DeliveryPageModel
@{
    this.ViewData["Title"] = "Просмотр заказов";
}
@section Supports { @{ await this.Html.RenderPartialAsync("DeliveryTabsPartial"); } }
<div class="container p-0 my-5">
    <div class="row justify-content-center">
        @if(this.Model.ErrorMessage != null)
        {
        <div class="mb-4 alert alert-danger alert-dismissible fade show" role="alert"
style="border-radius:16px;">
```

```
            <i class="bi bi-exclamation-triangle-fill"></i>
            <strong>Уведомление!</strong> @this.Model.ErrorMessage
            <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
        </div>
        }
        <div class="col-12 col-sm-10 shadow border border-secondary p-5 mb-4"
style="border-radius:16px;">

            <label class="form-label mb-3 fs-4 fw-semibold"><i class="bi bi-
archive"></i> Список доступных заказов: </label>
            <div class="mb-2" style="overflow:scroll;max-height:300px;">
                <table class="table table-striped">
                    <thead>
                        <tr>
                            <th scope="col">#</th>
                            <th scope="col">Размеры:</th>
                            <th scope="col">Тип окна:</th>
                            <th scope="col">Толщина:</th>
                            <th scope="col">Состояние:</th>
                            <th scope="col">Дата заказа:</th>
                            <th width="30" scope="col"></th>
                        </tr>
                    </thead>
                    <tbody>
                        @for (var index = 0; index < this.Model.OrdersList.Count;
index++ )
                        {
                        var orderRecord = this.Model.OrdersList.ElementAt(index);
                        <tr>
                            <th scope="row">@(index + 1)</th>
                            <td>@($"{orderRecord.Width} / {orderRecord.Height}
(CM)")</td>
                            <td>@orderRecord.Windowtype</td>
                            <td>@($"Кол-во пакетов: {orderRecord.Packetcount} (ШТ)")</td>
                            <td>@orderRecord.State</td>
                            <td>@orderRecord.Ordertime</td>
                            <td>
                                <form method="post" asp-page="DeliveryPage">
                                    <input type="hidden" name="selectedId"
value="@orderRecord.Orderid" />
                                    <input type="submit" class="btn btn-success"
value="Доставить"/>
                                </form>
                            </td>
                        </tr>
                        }
                    </tbody>
                </table>
                @if(this.Model.OrdersList.Count <= 0)
                {
                <div class="d-flex flex-row justify-content-center">
                    <label class="form-label fs-5 fw-semibold">Список пока пустой</label>
                </div>
                }
            </div>
        </div>
    </div>
</div>
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Internal;
using System.Security.Claims;
```

```
using WebApplication.Filters;
using WebApplication.Models;

namespace WebApplication.Pages
{
    [FlagsAttribute]
    public enum IndexMode : sbyte
    {
        Authorization = 32, Client = 1, Manager = 2, Builder = 4, Delivery = 8, Provider
= 16,
    }
    [AuthorizationFilterAttribute]
    public partial class IndexModel : PageModel
    {
        protected readonly ILogger<IndexModel> _logger = default!;
        protected IDbContextFactory<MispisCourseworkContext> DatabaseFactory { get; set; }
= default!;

        [BindPropertyAttribute(SupportsGet = true)]
        public IndexMode IndexMode { get; set; } = IndexMode.Authorization;

        [BindPropertyAttribute(SupportsGet = true)]
        public string? ErrorMessage { get; set; } = default;
        public bool HasError { get => this.ErrorMessage != null; }

        [BindPropertyAttribute]
        public Authorization AuthorizationModel { get; set; } = default!;

        [BindPropertyAttribute]
        public Account AccountModel { get; set; } = default!;

        [BindPropertyAttribute]
        public Employee EmployeeModel { get; set; } = default!;

        [BindPropertyAttribute]
        public Client ClientModel { get; set; } = default!;

        [BindPropertyAttribute]
        public Deliveryman DeliverymanModel { get; set; } = default!;

        [BindPropertyAttribute]
        public Resourceprovider ProviderModel { get; set; } = default!;

        [BindPropertyAttribute(SupportsGet = true, Name = "isActivated")]
        public bool? IsAccountActivated { get; set; } = default!;

        public IndexModel(ILogger<IndexModel> logger,
IDbContextFactory<MispisCourseworkContext> factory) : base()
        { this._logger = logger; this.DatabaseFactory = factory; }

        public virtual Task<IActionResult> OnGetAsync()
        {
            this._logger.LogInformation("Authorization GET Method");
            if (this.HttpContext.User.Identity != null &&
this.HttpContext.User.Identity.IsAuthenticated)
            {
                return
Task.FromResult<IActionResult>(base.LocalRedirect("/profile/login"));
            }
            return Task.FromResult<IActionResult>(this.Page());
        }
        public async Task<IActionResult> OnPostRegistrationAsync()
        {
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
```

```
                this.AuthorizationModel.Accesscode = Guid.NewGuid().ToString();
                if(dbcontext.Authorizations.Where(item => item.Login ==
this.AuthorizationModel.Login).Count() > 0)
                {
                    return this.RedirectToPage("Index", new { ErrorMessage = "Логин уже
используется", IndexMode = this.IndexMode });
                }
                await dbcontext.Authorizations.AddAsync(this.AuthorizationModel);
                this.AccountModel.Authorization = this.AuthorizationModel;

                this.AccountModel.Profiletype = this.IndexMode.ToString();
                await dbcontext.Accounts.AddAsync(this.AccountModel);

                await dbcontext.SaveChangesAsync();
                if((IndexMode.Manager | IndexMode.Builder |
IndexMode.Delivery).HasFlag(this.IndexMode))
                {
                    this.EmployeeModel.Accountid = this.AccountModel.Accountid;
                    this.EmployeeModel.Activated = default(bool);
                    await dbcontext.Employees.AddAsync(this.EmployeeModel);
                }
                switch (this.IndexMode)
                {
                    case IndexMode.Builder:
                        await dbcontext.Builders.AddAsync(new Builder() { Employee =
this.EmployeeModel }); break;
                    case IndexMode.Client:
                        this.ClientModel.Accountid = this.AccountModel.Accountid;
                        await dbcontext.Clients.AddAsync(this.ClientModel);
                        break;
                    case IndexMode.Provider:
                        this.ProviderModel.Accountid = this.AccountModel.Accountid;
                        await dbcontext.Resourceproviders.AddAsync(this.ProviderModel);
break;
                    case IndexMode.Manager:
                        await dbcontext.Managers.AddAsync(new Manager() { Employee =
this.EmployeeModel, Isadmin = default });
                        break;
                    case IndexMode.Delivery:
                        this.Delivery manModel.Employee = this.EmployeeModel;
                        await dbcontext.Deliverymen.AddAsync(this.Delivery manModel);
                        break;
                }
                await dbcontext.SaveChangesAsync();
            }
            return await this.OnPostLoginAsync();
        }
        public async Task<IActionResult> OnPostLoginAsync()
        {
            await Console.Out.WriteLineAsync("POST LOGIN");
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                var profileRecord = await dbcontext.Authorizations.Include(item =>
item.Account).FirstOrDefaultAsync(item =>
                    item.Login == this.AuthorizationModel.Login && item.Password ==
this.AuthorizationModel.Password);

                if(profileRecord == null || profileRecord.Account == null)
                { return this.RedirectToPage("Index", new { ErrorMessage = "Профиль не
найден", IndexMode = this.IndexMode }); }

                this.AccountModel = profileRecord.Account;
                var employeeRecord = await dbcontext.Employees.FirstOrDefaultAsync(item
=> item.Accountid == this.AccountModel.Accountid);
```

```
                if (employeeRecord != null && employeeRecord.Activated == false) return
base.RedirectToPage("Index", new { isActivated = false });
            }
            var profilePrinciple = new ClaimsIdentity(new List<Claim>()
            {
                new Claim(ClaimTypes.PrimarySid, this.AccountModel.Accountid.ToString()),
                new Claim(ClaimTypes.Role, this.AccountModel.Profiletype),
            },
            CookieAuthenticationDefaults.AuthenticationScheme);
            await this.HttpContext.SignInAsync(new ClaimsPrincipal(profilePrinciple));

            return
this.RedirectToPage($"/{this.AccountModel.Profiletype}Pages/{this.AccountModel.Profiletyp
e}Page");
        }
    }
}
```

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;
using WebApplication.Filters;
using WebApplication.Models;
using WebApplication.Services;

namespace WebApplication.Pages.ProviderPages
{
    [AuthorizeAttribute(Policy = "Provider"), ProfileFilterAttribute("ProviderSettings")]
    public partial class ProviderSettingsModel : PageModel
    {
        protected IDbContextFactory<MispisCourseworkContext> DatabaseFactory { get; set; }
= default!;

        protected readonly ILogger<ProviderSettingsModel> Logger = default!;
        public ProviderSettingsModel(ILogger<ProviderSettingsModel> logger,
IDbContextFactory<MispisCourseworkContext> factory)
            : base() { this.Logger = logger; this.DatabaseFactory = factory; }

        [BindPropertyAttribute]
        public Resourceprovider ProviderModel { get; set; } = default!;

        [BindPropertyAttribute]
        public Account Account { get => this.ProviderModel.Account; set =>
this.ProviderModel.Account = value; }

        [BindPropertyAttribute(SupportsGet = true)]
        public string? ErrorMessage { get; set; } = default;

        public virtual async Task<IActionResult> OnGetAsync()
        {
            var profileId =
int.Parse(this.HttpContext.User.FindFirstValue(ClaimTypes.PrimarySid));
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                this.ProviderModel = (await dbcontext.Resourceproviders.Where(item =>
item.Accountid == profileId)
                    .Include(item => item.Account).FirstOrDefaultAsync())!;

                if (this.ProviderModel == null) return base.BadRequest("Данные невозможно
загрузить");
            }
            return this.Page();
        }
```

```
        public virtual async Task<IActionResult> OnPostAsync([FromServices]
IProfileService profileService)
        {
            var profileId =
int.Parse(this.HttpContext.User.FindFirstValue(ClaimTypes.PrimarySid));
            var errorMessage = await
profileService.UpdateAccount(this.ProviderModel.Account, profileId);

            if (errorMessage != null) return base.RedirectToPage("ProviderSettings", new
{ ErrorMessage = errorMessage.ErrorMessage });
            var errorModel = new Dictionary<string, string?>()
            {
                { "Неверная компания", this.ProviderModel.Companyname }, { "Неверный
адрес компании", this.ProviderModel.Address },
            };
            foreach (KeyValuePair<string, string?> error in errorModel)
            {
                if(error.Value == null || error.Value.Length < 5) return
base.RedirectToPage("ProviderSettings", new { ErrorMessage = error.Key });
            }
            using (var dbcontext = await this.DatabaseFactory.CreateDbContextAsync())
            {
                await dbcontext.Resourceproviders.Where(item => item.Accountid ==
profileId)
                    .ExecuteUpdateAsync(item => item.SetProperty(p => p.Companyname, p =>
this.ProviderModel.Companyname)
                    .SetProperty(p => p.Address, p => this.ProviderModel.Address));

                await dbcontext.SaveChangesAsync();
            }
            return await this.OnGetAsync();
        }
    }
}
```