МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого»

Факультет автоматизированных и информационных систем
Кафедра «Информационные технологии»

ОТЧЕТ
по технологической практике

База практики: ЗАО «ИТРАНЗИШЭН»

Составил: студент гр. ИТИ-31 _____ Ивашко В. Н.
(подпись, дата)

Руководитель практики
от предприятия: главный инженер _____ Лебедев П. В.
(должность) (подпись, дата)

Руководитель практики
от университета: старший преподаватель _____ Стефановский И. Л.
(должность) (подпись, дата)

Дата защиты: _____
Оценка: _____
Подписи членов комиссии: _____
_____

Гомель 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Технологическая практика является неотъемлемым и важным этапом в подготовке квалифицированных *IT*-специалистов. Она направлена на закрепление и углубление теоретических знаний, полученных в процессе обучения, а также на приобретение практических навыков в области проектирования, разработки и внедрения современных программных продуктов для решения реальных производственных задач.

В рамках данной технологической практики была поставлена задача по разработке многофункционального корпоративного веб-сайта, совмещающего в себе функции электронного каталога продукции и информационно-представительского ресурса. Цель проекта — не просто демонстрация ассортимента товаров, а создание полноценного цифрового представительства компании в сети Интернет. Разрабатываемая система призвана служить единой точкой входа для потребителей и партнёров, объединяя как подробный каталог продукции, так и ключевые разделы: «О нас», «История», «Где купить» и «Контакты», предоставляя пользователям исчерпывающую информацию о деятельности предприятия.

Базой для прохождения технологической практики послужило Коммунальное производственное унитарное предприятие «Мозырские молочные продукты» – одно из ведущих предприятий молочной промышленности региона, специализирующееся на производстве широкого ассортимента качественной и натуральной продукции. Практика проходила в *IT*-отделе предприятия.

В ходе технологической практики были решены следующие ключевые задачи:

– изучение вопросов охраны труда и техники безопасности на рабочем месте и на территории производственного предприятия;

– ознакомление с организационной структурой предприятия, его основной деятельностью и ассортиментом выпускаемой продукции;

– освоение современных подходов к разработке веб-приложений на основе реальных бизнес-требований;

– приобретение практических навыков проектирования баз данных для хранения информации о номенклатуре товаров;

– получение опыта в реализации клиент-серверной архитектуры и создании интуитивно понятного пользовательского интерфейса.

Практическое задание представляет собой разработку корпоративного сайта-каталога, который обеспечивает удобную навигацию по категориям товаров и предоставляет детальные карточки для каждой товарной позиции. Помимо каталога, приложение включает в себя важные информационные блоки: раздел «О нас», знакомящий посетителей с миссией и ценностями компании, «История», отражающая богатый опыт предприятия, а также практические разделы «Где купить» с информацией о точках продаж и «Контакты» для обратной связи. Такая комплексная структура превращает веб-приложение из простого каталога в эффективный инструмент для коммуникации с клиентами, повышения их лояльности и укрепления бренда на цифровом рынке.

1 СВЕДЕНИЯ ОБ ОРГАНИЗАЦИИ

1.1 История организации

Коммунальное производственное унитарное предприятие «Мозырские молочные продукты» является одним из значимых и старейших предприятий пищевой промышленности Полесского региона, чья история неразрывно связана с историей города Мозыря и развитием всей молочной отрасли Беларуси. Путь, пройденный заводом, отражает ключевые этапы становления и модернизации экономики страны: от послевоенного восстановления до внедрения современных международных стандартов качества.

Начало истории предприятия было положено в тяжёлые послевоенные годы. В 1944 году было принято решение о создании Мозырского городского молочного завода. Это было продиктовано острой необходимостью обеспечения жителей города и района базовыми продуктами питания. Завод был размещён в небольшом приспособленном помещении, а его производственные мощности на тот момент были весьма скромными. Ассортимент продукции был узким и включал жизненно необходимые товары: пастеризованное молоко, кефир, сметану, творог и сливочное масло. В условиях разрушенной инфраструктуры и дефицита ресурсов запуск и стабильная работа предприятия стали настоящим трудовым подвигом его первых сотрудников.

Постепенно, по мере восстановления народного хозяйства, завод развивался и наращивал объёмы производства. Значительным шагом вперёд стало строительство в 1976 году нового главного производственного корпуса, а также административно-бытового корпуса. Это позволило не только увеличить производственные площади, но и установить новое, более производительное оборудование. Мощность предприятия выросла в десять раз – до пятидесяти тонн переработки молока в смену. Расширился и ассортимент выпускаемой продукции, начался выпуск мороженого, которое быстро завоевало популярность у местного населения. В этот период завод окончательно закрепил за собой статус ключевого поставщика молочной продукции в регионе.

Новый этап в жизни предприятия начался в постсоветский период, когда экономика страны переходила на рыночные рельсы. В 1995 году, в соответствии с процессами приватизации, завод был преобразован в Открытое акционерное общество «Мозырские молочные продукты». Эта трансформация потребовала от руководства и коллектива адаптации к новым условиям хозяйствования, поиска новых рынков сбыта и повышения конкурентоспособности продукции.

Осознавая, что будущее предприятия зависит от качества и технологий, в 2004 году руководство начало программу масштабной технической модернизации. Этот период можно по праву считать вторым рождением завода. Было закуплено и установлено современное оборудование от ведущих мировых производителей. В частности, была введена в эксплуатацию пастеризационно-охладительная установка марки *APV*, которая позволила значительно улучшить микро-

биологические показатели и продлить сроки хранения продукции. Была запущена новая линия по производству творога на оборудовании польских фирм *OBRAM* и *TEWES-BIS*, что позволило автоматизировать процесс и добиться стабильного качества продукта. Также была установлена автоматизированная линия фасовки сметаны от компании *Finnpak* и проведена модернизация маслоцеха.

Параллельно с техническим перевооружением предприятие активно работало над расширением своей сырьевой зоны, наладив поставки качественного натурального молока из Ельского, Наровлянского и Лельчицкого районов. Это позволило обеспечить стабильную загрузку возросших мощностей и гарантировать использование только натурального сырья.

Важнейшей вехой в современной истории компании стала сертификация системы менеджмента качества и безопасности на соответствие международным стандартам в 2017 году. Получение сертификатов *ISO* и внедрение системы *HACCP* (*Hazard Analysis and Critical Control Points*) подтвердило, что производственные процессы на предприятии соответствуют строжайшим мировым требованиям. Это не только укрепило доверие потребителей на внутреннем рынке, но и открыло новые возможности для экспорта продукции.

Сегодня КПУП «Мозырские молочные продукты» – это современное, динамично развивающееся предприятие, которое успешно сочетает дорогие традиции с передовыми технологиями для предоставления натурального качества. Фундаментом его деятельности являются незыблемые ценности, которые формировались на протяжении всей его богатой истории.

В основе философии компании лежит миссия по обеспечению потребителей вкусной, полезной и безопасной молочной продукцией. Для её реализации предприятие руководствуется следующими ключевыми ценностями:

1. натуральные ингредиенты. Компания принципиально использует только свежее, натуральное молоко от сертифицированных местных фермерских хозяйств. Это обеспечивает экологическую чистоту и превосходные вкусовые качества конечной продукции, формируя основу доверия со стороны покупателей;

2. строгий контроль качества. На заводе внедрена многоступенчатая система контроля, охватывающая всю производственную цепочку — от приёмки сырого молока до упаковки и хранения готовой продукции. Собственная аккредитованная лаборатория осуществляет постоянный мониторинг, что гарантирует полную безопасность и соответствие всем стандартам;

3. бережное отношение к традициям. Предприятие чтит проверенные временем рецепты, которые полюбились многим поколениям потребителей. При этом классические подходы гармонично сочетаются с инновациями и использованием современного высокотехнологичного оборудования, что позволяет совершенствовать продукцию, сохраняя её аутентичный, «тот самый» вкус.

Таким образом, история предприятия «Мозырские молочные продукты» – это наследие вкуса и качества, которое создавалось десятилетиями упорного труда и постоянного стремления к совершенству.

1.2 Работа и обучение в КПУП «Мозырские молочные продукты»

Коммунальное производственное унитарное предприятие «Мозырские молочные продукты» уделяет значительное внимание подготовке и развитию кадров, рассматривая профессионализм сотрудников как один из ключевых факторов высокого качества продукции и стабильности производства. Основной упор в обучении делается на развитие компетенций специалистов в области технологии переработки молока, контроля качества и эксплуатации современного производственного оборудования. Вместе с тем, предприятие обеспечивает поддержку и для специалистов технических и административных подразделений, включая *IT*-отдел.

Для сотрудников, работающих в сферах, не связанных напрямую с производством, в том числе в *IT*-отделе, основной формой профессионального развития является обучение на рабочем месте. Предприятие придерживается подхода, при котором практический опыт, полученный при решении реальных задач, является наиболее эффективным способом повышения квалификации. За каждым молодым специалистом или практикантом закрепляется опытный наставник, который осуществляет общее руководство, консультирует по сложным вопросам и помогает адаптироваться к внутренним процессам и стандартам работы.

Предприятие активно сотрудничает с ведущими профильными колледжами и высшими учебными заведениями региона с целью привлечения молодых и перспективных кадров. В рамках такого партнёрства студентам предоставляется возможность прохождения производственных и технологических практик. Это позволяет будущим специалистам не только применить теоретические знания в условиях реального промышленного объекта, но и получить ценное представление о специфике работы в пищевой отрасли, её требованиях и стандартах.

Программа технологической практики для студентов *IT*-специальностей строится вокруг решения конкретной прикладной задачи, актуальной для предприятия. В данном случае такой задачей стала разработка корпоративного сайта-каталога. Процесс практики включал в себя несколько этапов:

– погружение в предметную область: изучение ассортимента продукции, её характеристик и особенностей для грамотного проектирования структуры каталога;

– постановка задачи: совместно с руководителем практики от предприятия были определены цели, функциональные требования и ожидаемые результаты проекта;

– самостоятельная работа: основной объём времени был посвящён непосредственно разработке – проектированию архитектуры, созданию базы данных, написанию кода и вёрстке пользовательского интерфейса;

– регулярные консультации: ежедневные встречи с наставником для обсуждения текущего прогресса, решения возникших проблем и получения обратной связи.

Такой подход позволяет не просто выполнить изолированное учебное задание, а пройти через все ключевые стадии реального проекта. Работа над индивидуальным заданием по созданию веб-приложения была направлена на развитие целого ряда профессиональных компетенций: от навыков проектирования баз данных и веб-архитектуры до умения понимать и транслировать бизнес-требования заказчика в готовый программный продукт.

1.3 Охрана труда и техника безопасности на рабочем месте

Система охраны труда объединяет законодательные нормы, организационные решения, технические мероприятия, гигиенические и медико-профилактические действия, направленные на обеспечение безопасности и поддержание здоровья трудящихся.

По мере развития технологий и научно-технического прогресса трудовые условия для специалистов интеллектуального труда становятся все более сложными и предъявляют повышенные требования к физическим, эмоциональным и умственным ресурсам. В связи с этим охрана здоровья и обеспечение безопасности работников выступает как приоритетная задача современного общества. Для её решения требуется масштабное применение передовых методов организации трудовой деятельности, сокращение доли ручного и неквалифицированного труда, формирование трудовой среды, исключающей профессиональные заболевания и несчастные случаи на производстве.

В Республике Беларусь вопросам безопасности труда придается безусловное приоритетное значение. Право работника на безопасный труд – одна из важнейших гарантий в области труда, установленных Конституцией Республики Беларусь.

Отличительная черта политики в области охраны труда в Республике Беларусь – это активное влияние государства на процессы в сфере охраны труда через доступные ему механизмы и процедуры. Это, прежде всего, разработка и принятие законодательства, устанавливающее обязательные требования и гарантии, определяет компетенцию в области охраны труда всех заинтересованных сторон [1].

Создание безопасной рабочей среды напрямую связано с грамотной организацией рабочих мест, что выступает ключевым фактором охраны труда. Рабочее место представляет собой определенную зону пространства, где сотрудник проводит основную часть своего трудового времени, выполняя профессиональные обязанности. Правильно спроектированное рабочее место, отвечающее требованиям по площади, конфигурации и габаритам, создает для программиста комфортные условия работы и способствует росту производительности труда при снижении физических и психологических нагрузок. Грамотная организация рабочего места может повысить эффективность труда программиста на 10-25 процентов.

В соответствии с ГОСТ 12.2.032-78, рабочее место и взаимосвязь всех его компонентов должны отвечать антропометрическим, физиологическим и психологическим критериям. При создании рабочего места программиста необходимо учитывать такие факторы:

- рациональное размещение технического оборудования;
- обеспечение свободного пространства для перемещений;
- соблюдение допустимых норм шума;
- организация качественного естественного и искусственного освещения для выполнения рабочих задач.

Организация рабочих мест для деятельности в сидячем положении регламентируется ГОСТ 12.2.032-78 «Система стандартов безопасности труда. Рабочее место при выполнении работ сидя. Общие эргономические требования».

Нормативы организации труда при использовании персональных компьютеров и оргтехники устанавливаются СанПиН 9-131 РБ 2000 «Гигиенические требования к видео дисплейным терминалам, электронно-вычислительным машинам и организации работы».

Специалисты-программисты проводят значительную долю рабочего времени в неподвижном положении тела в позе сидя, что делает особенно актуальным создание эргономически правильного и гигиенически безопасного рабочего места. Рациональное размещение предметов труда, инструментов и документации в рабочей зоне способствует снижению нагрузки на органы восприятия и опорно-двигательный аппарат, а также увеличению эффективности труда.

Качественное освещение в рабочем помещении также выступает важным элементом создания эргономически оптимального рабочего места. Естественное освещение обладает большей эффективностью и благоприятно воздействует на здоровье, поэтому обеспечение достаточного объема естественного света в помещении должно быть приоритетом. Искусственное освещение играет существенную роль в ситуациях недостатка естественного света.

Площадь рабочего места с видеодисплейным терминалом (ВДТ) и персональным компьютером (ПЭВМ) для взрослых пользователей должна составлять минимум 6 м², что создает комфортные условия для размещения программиста и снижает вероятность развития нарушений опорно-двигательной системы. Необходимо также принимать во внимание объем помещения, который должен быть не менее 20 м³ для обеспечения адекватной вентиляции и профилактики заболеваний, вызванных недостаточным воздухообменом. Для создания безопасных условий труда в помещениях с ВДТ и ПЭВМ требуется организация равномерного и достаточного искусственного освещения.

Для создания равномерного освещения в помещении применяется система общего равномерного освещения. Для этого необходимо правильно подобрать тип осветительных приборов и их размещение для достижения однородности освещения по всей площади помещения.

Также может применяться комбинированная система освещения, при которой к общему освещению добавляются локальные источники света,

например, настольные лампы. Это способствует улучшению качества освещения на рабочем месте и снижению нагрузки на зрительный аппарат.

Для оптимального результата рекомендуется использовать осветительные приборы с возможностью регулировки яркости, что позволяет настраивать освещение в соответствии с потребностями конкретного рабочего места.

Освещенность на рабочей поверхности стола должна находиться в диапазоне от 300 до 500 лк, что обеспечивает комфортную работу с документами. Местное освещение не должно создавать отражений на поверхности экрана и увеличивать освещенность экрана свыше 300 лк.

При организации освещения помещений с ВДТ и ПЭВМ выбор источников света имеет особое значение. Люминесцентные лампы представляют собой подходящий вариант, поскольку они создают равномерное и мягкое освещение, которое уменьшает утомляемость глаз при продолжительной работе за компьютером.

Рабочие столы рекомендуется располагать так, чтобы ВДТ были повернуты боковой стороной к оконным проемам, что обеспечивает падение естественного света преимущественно слева, позволяя максимально эффективно использовать естественное освещение, поскольку свет будет попадать на рабочую поверхность под оптимальным углом. Установка регулируемых жалюзи на окнах дает возможность контролировать количество естественного света, что особенно важно в зависимости от времени суток и погодных условий.

Конструктивные особенности рабочего стола играют значительную роль в обеспечении удобства и эффективности работы. Он должен обладать достаточной площадью и оптимальным размещением для всех необходимых компонентов, включая клавиатуру, мышь, ВДТ и прочее оборудование. Рабочий стол должен соответствовать современным эргономическим стандартам для минимизации риска развития заболеваний, связанных с работой за компьютером, таких как синдром запястного канала или болевые ощущения в области спины и шеи.

Кресло для программиста должно обеспечивать комфорт и эргономичность для поддержания правильного положения тела в течение длительного периода работы. Подъемно-поворотное кресло позволяет регулировать угол наклона сиденья и спинки, а также корректировать высоту сиденья и расстояние между спинкой и сиденьем. Это дает возможность подобрать оптимальные параметры для каждого пользователя с учетом его физических особенностей и телосложения. Регулировка каждого параметра должна быть независимой и иметь надежную фиксацию во избежание непроизвольного смещения кресла во время работы.

Экран видеомонитора имеет важное значение для создания комфортного и безопасного рабочего места. Оптимальное расстояние между глазами пользователя и экраном составляет 600-800 мм с учетом размеров символов и знаков на экране. Расстояние не должно быть менее 500 мм во избежание перенапряжения глаз и проблем со зрением. Важно также учитывать угол наклона экрана и его высоту для предотвращения отражений и бликов на экране.

Высота рабочей поверхности стола имеет важное значение для обеспечения правильной позы тела программиста. Оптимальная высота стола должна регулироваться в диапазоне 680-800 мм для подбора оптимального уровня для каждого пользователя. При отсутствии такой возможности высота стола должна составлять 725 мм. Важно также учитывать ширину и глубину стола для обеспечения достаточного пространства для работы и размещения необходимых материалов.

Для создания комфортного и безопасного рабочего места рабочий стол должен иметь достаточное пространство для ног: высотой не менее 600 мм, шириной не менее 500 мм и глубиной не менее 450 мм на уровне колен и не менее 650 мм на уровне вытянутых ног. Это способствует снижению нагрузки на ноги и позвоночник пользователя в процессе работы.

Клавиатуру следует размещать на рабочей поверхности стола на расстоянии не менее 300 мм от края. Это способствует уменьшению нагрузки на запястья и предотвращению возможных травм. Дополнительно можно использовать специальную регулируемую по высоте рабочую поверхность для клавиатуры, отделенную от основной столешницы. Это позволяет настроить высоту клавиатуры под конкретного пользователя и снизить риск развития травматических заболеваний.

Микроклимат рабочей зоны имеет существенное значение для здоровья и самочувствия работников. СанПиН 9-131 РБ 2000 устанавливает требования к влажности, температуре, скорости движения воздуха и другим параметрам микроклимата для обеспечения комфортной и безопасной рабочей среды.

Техника безопасности представляет собой комплекс организационных мероприятий и технических решений, направленных на предотвращение воздействия на работающих опасных производственных факторов. Это включает меры по защите от пожара, аварийных ситуаций, электрических и химических опасностей и другие, а также обучение работников правилам безопасности и применению необходимых средств защиты.

Для обеспечения безопасности при работе с компьютером и другими устройствами программист должен руководствоваться «Инструкцией по охране труда и технике безопасности для программиста при работе с ПЭВМ и ВДТ». Данная инструкция содержит рекомендации и правила, способствующие защите работника от возможных опасностей, связанных с работой на компьютере. Это может включать правила использования электропитания, экранных фильтров, антивирусного программного обеспечения, а также правила обработки и хранения данных. Соблюдение данной инструкции является важным аспектом техники безопасности, который способствует снижению риска возникновения производственных несчастных случаев и повышению уровня безопасности на рабочем месте программиста.

Для обеспечения безопасности во время работы с компьютером необходимо соблюдать следующие требования:

– избегать применения металлических предметов для извлечения заблокированных клавиш клавиатуры, поскольку это может привести к повреждению клавиатуры и возникновению короткого замыкания;

– поддерживать чистоту и порядок на рабочем месте, не загромождать его для уменьшения риска травм и повреждений;

– быть внимательным и не отвлекаться во время работы, а также не позволять другим людям отвлекать вас, и не допускать лиц, не имеющих отношения к работе, на рабочее место;

– не прилагать чрезмерных усилий к кнопкам переключения во избежание повреждения клавиатуры и других устройств;

– следовать инструкции по эксплуатации для всех ВТ-средств для уменьшения риска повреждения оборудования и защиты себя от возможных опасностей;

– ограничивать продолжительность непрерывной работы на компьютере до 2 часов для уменьшения риска развития синдрома компьютерного зрения и других заболеваний, связанных с работой за компьютером.

В случае возникновения аварийных ситуаций необходимо соблюдать следующие требования безопасности:

– оборудование должно быть немедленно обесточено для предотвращения дальнейшего возникновения опасности;

– должна быть незамедлительно вызвана аварийная служба, которая примет меры для предотвращения угрозы безопасности и ликвидации аварийной ситуации;

– при возникновении аварийной ситуации необходимо принимать действия по ликвидации ее последствий для предотвращения возможных негативных последствий;

– в случае несчастного случая на рабочем месте свидетель и потерпевший (при возможности) должны принять меры по оказанию первой помощи и предотвращению травмирования других лиц. Они также должны незамедлительно сообщить о происшествии непосредственному руководителю потерпевшего или другому должностному лицу для принятия необходимых мер по организации помощи и предотвращению повторного возникновения подобных ситуаций в будущем.

2 АРХИТЕКТУРА ПРОГРАММНОГО ПРОДУКТА

2.1 Паттерны проектирования для реализации веб-приложения

Для разработки веб-приложения «Электронный каталог продукции» был выбран современный и гибкий набор инструментов, обеспечивающий высокую производительность и масштабируемость. В качестве основной среды разработки использовался редактор кода *Visual Studio Code*, который предоставляет широкие возможности для кастомизации и работы с различными языками и фреймворками благодаря своей развитой системе расширений. Основой для серверной и клиентской частей послужила платформа *Node.js*, позволяющая использовать язык *JavaScript* (*TypeScript*) для всего цикла разработки.

Для организации кодовой базы проекта была выбрана архитектура монорепозитория (*monorepo*). Управление этим монорепозиторием осуществлялось с помощью инструментария *Nx*, который предоставляет мощные средства для построения и масштабирования сложных приложений. Такой подход был выбран для обеспечения тесной интеграции и переиспользования кода между клиентской частью (*frontend*), реализованной на фреймворке *Next.js*, и серверной частью (*backend*), построенной на базе фреймворка *NestJS*. Использование *Nx* позволило создать общие библиотеки, например, для интерфейсов передачи данных (*DTO*) и правил валидации, что гарантирует консистентность и снижает дублирование логики во всём приложении.

Для управления базой данных *PostgreSQL* и сопутствующими сервисами, такими как *pgAdmin*, была применена технология контейнеризации *Docker*. Использование *Docker Compose* позволило декларативно описать и запустить многоконтейнерное приложение, что гарантирует идентичность и воспроизводимость окружения на любой машине, упрощает настройку и исключает конфликты зависимостей. В качестве *ORM* (*Object-Relational Mapping*) для взаимодействия с базой данных была выбрана *Prisma*, которая обеспечивает строгую типизацию, автоматическую генерацию клиента для запросов и удобный язык для описания схемы данных.

В процессе создания любого программного продукта, от простого сайта до сложной корпоративной системы, ключевую роль играет выбор архитектурных подходов. Правильно подобранный паттерн проектирования способен кардинально повлиять на успешность проекта, превратив потенциально хрупкое и дорогое в поддержке решение в надёжное, гибкое и долговечное. Использование устоявшихся шаблонов позволяет не только повысить качество кода, но и сделать его более структурированным и понятным для других разработчиков, что особенно важно при командной работе или дальнейшей передаче проекта.

Для построения качественной и поддерживаемой архитектуры в данном проекте было решено использовать паттерн проектирования «Репозиторий» (*Repository*). Этот шаблон применяется для создания слоя абстракции между бизнес-логикой приложения и механизмом доступа к данным.

«Репозиторий» является структурным паттерном проектирования, основная задача которого – полная изоляция деталей хранения и доступа к данным от остальной части приложения. Он имитирует коллекцию объектов в памяти, предоставляя приложению удобный интерфейс для работы с сущностями (например, продуктами, категориями), скрывая при этом всю сложность выполнения *SQL*-запросов, работы с *ORM* или любым другим источником данных. В рамках проекта это означает создание специальных сервисов-репозиториев, которые инкапсулируют в себе всю логику взаимодействия с клиентом *Prisma*.

Ключевым преимуществом паттерна «Репозиторий» является достигаемый уровень абстракции и гибкости. Изоляция бизнес-логики от конкретной реализации источника данных делает код значительно более тестируемым и поддерживаемым. Например, при написании модульных тестов можно легко подменить реальный репозиторий его имитацией (*mock*), что позволяет тестировать бизнес-логику в полной изоляции от базы данных.

Тем не менее, следует упомянуть и о недостатках данного паттерна. Основным минусом является введение дополнительного слоя абстракции, что может показаться избыточным для очень простых приложений с минимальным количеством операций чтения-записи. В некоторых случаях это может привести к написанию дополнительного шаблонного кода (*boilerplate*) и незначительному увеличению времени разработки на начальных этапах.

Несмотря на возможные недостатки, применение паттерна «Репозиторий» в проекте «Электронный каталог продукции» полностью оправдано. Необходимость его использования обусловлена потенциальной сложностью бизнес-логики, связанной с фильтрацией, поиском и сортировкой товаров по различным критериям. Централизация всех запросов к базе данных в одном месте упрощает управление этими сложными операциями. На рисунке 2.1 представлена обобщённая схема реализации паттерна «Репозиторий» в контексте приложения.

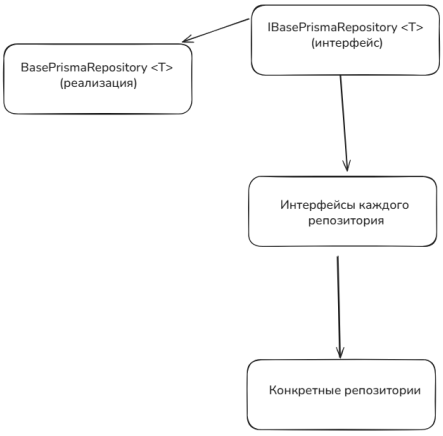

Рисунок 2.1 – Схема реализации паттерна «Репозиторий»

2.2 Проектирование архитектуры приложения и базы данных

Крайне важно избегать самонадеянных решений и с самого начала со всей серьезностью отнестись к качеству архитектуры продукта. Правильно заложенный фундамент не только упрощает процесс разработки, но и определяет, насколько легко будет поддерживать, масштабировать и развивать приложение в будущем.

Процесс проектирования веб-приложения «Электронный каталог продукции» был разделен на следующие логические этапы:

– анализ требований: определение основных целей и ожидаемой функциональности. На этом этапе были сформулированы ключевые требования: отображение каталога продукции с подробной информацией, публикация актуальных вакансий предприятия и предоставление списка точек продаж;

– декомпозиция: разбиение всей системы на основные компоненты. Были выделены: пользовательский интерфейс (*frontend*), серверная часть (*backend*) и хранилище данных (база данных). Также были идентифицированы их взаимосвязи и функциональные обязанности;

– определение интерфейсов: описание контрактов взаимодействия между компонентами. В частности, был спроектирован *API* (*Application Programming Interface*) для обмена данными между клиентской и серверной частями;

– разработка архитектурных диаграмм: создание визуальных схем, которые наглядно показывают структуру приложения и базы данных;

– проверка архитектуры: анализ спроектированной архитектуры на предмет узких мест, потенциальных проблем с производительностью и масштабируемостью.

При проектировании был использован метод декомпозиции, который заключается в разделении сложной системы на более простые, слабо связанные компоненты. Каждый компонент имеет чётко определённый интерфейс для взаимодействия с другими. Такой подход помогает эффективно управлять сложностью системы, позволяя вести разработку и поддержку отдельных частей приложения более независимо.

Основные компоненты архитектуры

Разработка архитектуры приложения потребовала внимания к деталям и учёту следующих основных компонентов и их взаимодействия:

– клиентская часть (*frontend*): отвечает за пользовательский интерфейс. Реализована на фреймворке *Next.js* и включает в себя *React*-компоненты, страницы (каталог, вакансии, магазины) и модули для взаимодействия с серверным *API*;

– серверная часть (*backend*): представляет собой *API*, реализованный на фреймворке *NestJS*. Он обрабатывает запросы от клиента, реализует бизнес-логику и взаимодействует с базой данных;

– слой бизнес-логики: инкапсулирован в сервисах *NestJS* и определяет основные операции с данными: получение списка продуктов, поиск вакансий, фильтрация магазинов по городу и другие аспекты работы приложения;

– слой доступа к данным: включает репозитории для работы с базой данных. Этот слой, реализованный с помощью *Prisma ORM*, абстрагирует детали хранения данных от бизнес-логики, что теоретически позволяет легко заменить технологию хранения без изменения остальной части системы.

Связь между клиентской и серверной частями осуществляется через *RESTful API*, который обеспечивает передачу данных в формате *JSON*. Взаимодействие основано на принципах *REST*, что обеспечивает слабую связность и независимость развития этих частей приложения.

Многоуровневая архитектура

В архитектуре приложения «Электронный каталог продукции» особое внимание уделено разделению ответственности между компонентами. Для серверной части применяется многоуровневая архитектура с чётким разделением на слои:

– слой контроллеров (*Controllers*) отвечает за обработку *HTTP*-запросов, валидацию входящих данных и взаимодействие с клиентом;

– слой сервисов (*Services*) инкапсулирует основную бизнес-логику приложения;

– слой доступа к данным (*Data Access Layer*) реализует паттерн «Репозиторий», обеспечивая абстракцию над технологией хранения данных (*Prisma*).

В клиентской части используется компонентный подход *Next.js* (*React*), который позволяет создавать переиспользуемые *UI*-компоненты (например, карточка товара, элемент списка вакансий) и организовывать их в иерархические структуры страниц.

Проектирование базы данных

База данных приложения построена на *PostgreSQL* – мощной и надёжной объектно-реляционной СУБД с открытым исходным кодом. Выбор *PostgreSQL* обусловлен его высокой производительностью, стабильностью и расширенными возможностями для работы со сложными запросами.

Структура базы данных включает следующие основные таблицы, описанные с помощью схемы *Prisma*:

Product – хранит информацию о продукции предприятия:

– *id* – уникальный идентификатор товара (UUID);

– *name* – наименование продукта;

– *description* – подробное описание продукта;

– *imageUrl* – ссылка на изображение продукта;

– *proteins* – содержание белков на 100 г;

– *fats* – содержание жиров на 100 г;

– *carbs* – содержание углеводов на 100 г;

– *calories* – калорийность на 100 г;

– *createdAt* – дата и время создания записи;

– *updatedAt* – дата и время последнего обновления записи.

Vacancy – содержит информацию об открытых вакансиях на предприятии:

– *id* – уникальный идентификатор вакансии (UUID).

– *title* – название должности.

– *department* – отдел или подразделение.

– *description* – подробное описание вакансии, требования и условия.

– *location* – местоположение (по умолчанию «Главный офис завода»).

– *postedAt* – дата и время публикации вакансии.

Store – хранит информацию о точках продаж, где можно приобрести продукцию:

– *id* – уникальный идентификатор магазина (UUID);

– *name* – название магазина или торговой сети;

– *address* – точный адрес;

– *city* – город;

– *postalCode* – почтовый индекс;

– *latitude* – географическая широта для отображения на карте;

– *longitude* – географическая долгота для отображения на карте.

В представленной схеме данных на текущем этапе разработки сущности являются независимыми и не имеют прямых реляционных связей друг с другом. Такая структура была выбрана для простоты и соответствует текущим требованиям приложения, где продукты, вакансии и магазины являются самостоятельными информационными блоками.

Для оптимизации производительности запросов предполагается создание индексов по наиболее часто используемым полям, например:

– по полю *name* в таблице *Product* для ускорения поиска по названию;

– по полю *city* в таблице *Store* для быстрой фильтрации магазинов по городу.

Схема базы данных спроектирована с учетом особенностей работы *Prisma ORM*, используемого в приложении для доступа к данным. Такая архитектура обеспечивает гибкость, масштабируемость и удобство разработки, что полностью соответствует современным требованиям к веб-приложениям.

2.3 Структура классов веб-приложения «Электронный каталог продукции»

При написании кода серверной части особое внимание уделялось его читаемости, структурированности и соответствию лучшим практикам разработки. Использовались ясные и описательные имена для переменных, классов и методов. Этот подход, часто называемый «чистым кодом», критически важен, поскольку он упрощает дальнейшую поддержку, отладку и расширение функционала приложения, делая его понятным не только для автора, но и для других разработчиков.

Процесс организации и построения классов в приложении следовал логической последовательности, основанной на принципах многоуровневой архитектуры:

– абстракция и интерфейсы: определение основных архитектурных слоёв, таких как контроллеры, сервисы и репозитории;

– базовые классы: создание абстрактного базового класса для репозиториев (*BasePrismaRepository*), который содержит общие методы для взаимодействия с базой данных, что соответствует принципу *DRY (Don't Repeat Yourself)*;

– контракты данных (*DTO*): определение классов *DTO (Data Transfer Objects)* для строгого описания структуры данных, передаваемых между клиентом и сервером;

– реализация репозиториев: создание конкретных классов репозиториев для каждой сущности, отвечающих за инкапсуляцию логики прямого доступа к данным через *Prisma*;

– реализация сервисов: создание классов сервисов, содержащих основную бизнес-логику приложения, таких как валидация, обработка ошибок и координация операций;

– реализация контроллеров: создание классов контроллеров, которые обрабатывают входящие *HTTP*-запросы, взаимодействуют с сервисами и возвращают ответы клиенту.

Такой подход к организации классов позволяет создать гибкую, модульную и легко расширяемую архитектуру для веб-приложения.

При запуске приложения инициализируется экземпляр *NestJS*, который настраивается в файле *main.ts*. В этой точке входа происходит создание веб-сервера, регистрация глобальных компонентов, таких как каналы валидации (*Validation Pipes*), и запуск прослушивания сетевого порта.

Основой архитектуры *NestJS* является модульная система. Корневым элементом приложения выступает AppModule, который, в свою очередь, импортирует все остальные функциональные модули. Как видно из структуры проекта, функционал сгруппирован по сущностям (например, products, stores, vacancies). Каждый такой блок оформлен как отдельный модуль (например, StoresModule), который инкапсулирует в себе все связанные компоненты:

– *controllers*: массив контроллеров, которые будут обрабатывать запросы для этого модуля;

– *providers*: массив сервисов и репозиториев, которые будут доступны внутри этого модуля через механизм внедрения зависимостей;

– *imports / exports*: списки для импорта других модулей или экспорта собственных провайдеров.

Такая организация обеспечивает сильную инкапсуляцию и низкую связанность между различными функциональными блоками приложения.

Ключевым паттерном, используемым в *NestJS*, является внедрение зависимостей (*Dependency Injection, DI*). Вместо того чтобы классы сами создавали экземпляры своих зависимостей, фреймворк берёт эту задачу на себя. Классы, помеченные декоратором *Injectable*, регистрируются в контейнере *DI*. Когда другому классу (например, контроллеру) требуется сервис, он просто объявляет его в своём конструкторе, и *NestJS* автоматически предоставляет нужный экземпляр.

Слой сервисов (например, *StoresService*) является «мозгом» приложения. Он инкапсулирует бизнес-логику и координирует работу с репозиториями. Через механизм *DI* сервис получает экземпляр репозитория (*StoreRepository*). Методы

сервиса (*create, findAll, findOne* и др.) содержат логику приложения, например, проверку на существование записи перед её обновлением и обработку специфических ошибок базы данных, таких как ошибка *P*2025 от *Prisma*, которая преобразуется в понятный для клиента ответ *NotFoundException*.

Для обеспечения стандартизированного интерфейса доступа к данным используется паттерн «Репозиторий». Был создан абстрактный класс *BasePrismaRepository*, который определяет общие операции для всех репозиториев: *findUnique, findMany, create, update, delete*.

Конкретная реализация, например, *VacancyRepository*, наследует от этого базового класса. В конструкторе происходит инициализация сервиса *PrismaService*, который предоставляет прямое подключение к БД, и определяется модель *Prisma* (*this.prisma.vacancy*), с которой будет работать данный репозиторий.

Контроллеры (например, *StoresController*) обрабатывают *HTTP*-запросы и служат точкой входа в *API*. Они должны быть «тонкими», то есть их основная задача – принять запрос, извлечь из него необходимые данные и передать их на обработку в сервисный слой. Для привязки методов класса к URL-адресам и *HTTP*-методам используются декораторы *NestJS* (Controller, Get, Post и т.д.).

Для обеспечения безопасности и валидации входящих данных используются объекты передачи данных (*DTO*). Это классы (например, *CreateStoreDto*), которые описывают структуру данных, ожидаемых в теле запроса. С помощью дополнительных библиотек, таких как class-validator, эти классы можно снабдить декораторами (*IsString, IsNotEmpty*), а глобальный *ValidationPipe* будет автоматически проверять все входящие запросы на соответствие этим правилам, отклоняя некорректные данные ещё до того, как они попадут в бизнес-логику.

Такая чётко определённая структура, где каждый класс имеет единственную зону ответственности, обеспечивает высокую поддерживаемость и тестируемость кода, что является залогом успешного развития программного продукта.

3 РЕАЛИЗАЦИЯ ВЕБ-ПРИЛОЖЕНИЯ «ЭЛЕКТРОННЫЙ КАТЛОГ ПРОДУКЦИИ»

3.1 Верификация веб-приложения

При переходе на страницу пользователя встречает современный и эстетически выверенный лендинг. Его композиция продумана таким образом, чтобы последовательно знакомить посетителя с ключевой информацией. При её проектировании ставилась задача создать не просто информативную, а визуально привлекательную и интуитивно понятную среду, которая с первых секунд формирует у посетителя положительное впечатление о компании и её продукции.

На рисунке 3.1 представлен пример главной страницы.

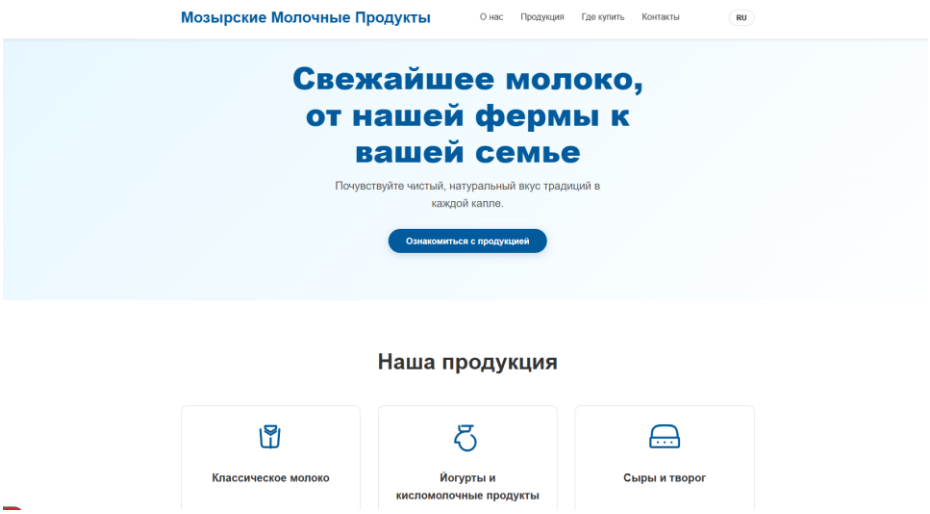

Рисунок 3.1 – Главная страница приложения

С целью повышения доступности и удобства использования веб-приложения для различных категорий пользователей, была реализована функция многоязычной поддержки интерфейса. В верхней панели навигации расположен интуитивно понятный переключатель, позволяющий пользователю в любой момент изменить язык отображения контента. На выбор предложено три языка: русский, как основной для целевого рынка, белорусский, для подчёркивания национальной идентичности бренда, и английский, ориентированный на иностранных партнёров и посетителей.

При смене языка все текстовые элементы интерфейса, включая навигационные ссылки, заголовки и описания, динамически обновляются без перезагрузки страницы, что обеспечивает плавный и комфортный пользовательский опыт. Пример главной страницы, переключенной на английский язык, представлен на рисунке 3.2.

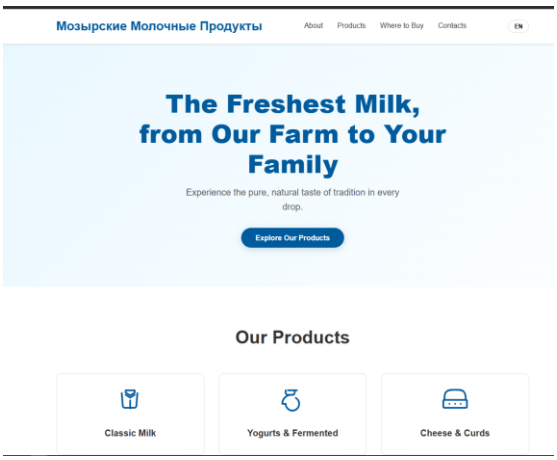

Рисунок 3.2 – Главная страницы на английском языке

Помимо основного каталога, в структуру веб-приложения был включён раздел «О нас». Этот раздел играет важную имиджевую роль, поскольку позволяет установить более тесную эмоциональную связь с потребителем. Перейдя на эту страницу, пользователь может не просто ознакомиться с сухими фактами, но и погрузиться в историю предприятия, узнать о его наследии и ключевых этапах развития.

Информация на странице подаётся в сжатой, но содержательной форме, рассказывая о пути, который компания прошла с момента своего основания. Текст подкреплён архивными фотографиями или инфографикой, что делает повествование более живым и интересным. Цель этого раздела — показать, что за брендом «Мозырские молочные продукты» стоят десятилетия опыта, традиции качества и люди, преданные своему делу. Таким образом, страница «О нас» способствует повышению лояльности и доверия к бренду, превращая его из безликого производителя в компанию с богатой историей.

Пример страницы представлен на рисунке 3.3.

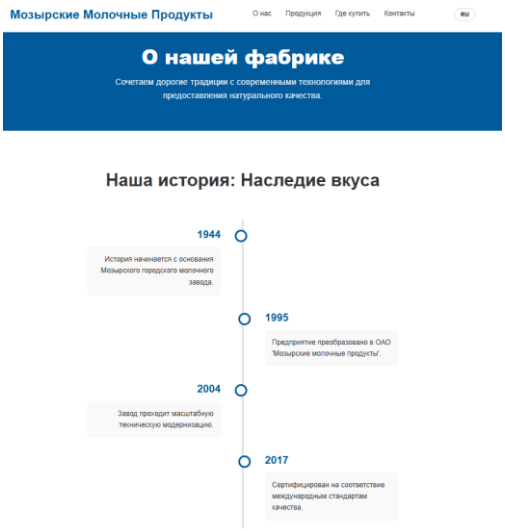

Рисунок 3.3 – Страница «О нас»

Для максимального удобства потребителей, желающих найти продукцию предприятия в розничной продаже, был разработан специальный раздел «Где купить». Эта страница представляет собой не просто статический список адресов, а интерактивный инструмент, направленный на решение конкретной практической задачи пользователя.

Центральным элементом страницы является функциональная строка поиска, которая позволяет мгновенно отфильтровать торговые точки по городу или названию. Это особенно актуально для покупателей из разных регионов, так как избавляет их от необходимости вручную просматривать длинный перечень магазинов. Результаты поиска отображаются в виде наглядного и структурированного списка, где для каждой точки продаж указано её название и точный адрес. Такая реализация значительно улучшает пользовательский опыт, делая процесс поиска продукции быстрым и эффективным.

Пример работы страницы «Где купить» с активной функцией поиска представлен на рисунке 3.4.

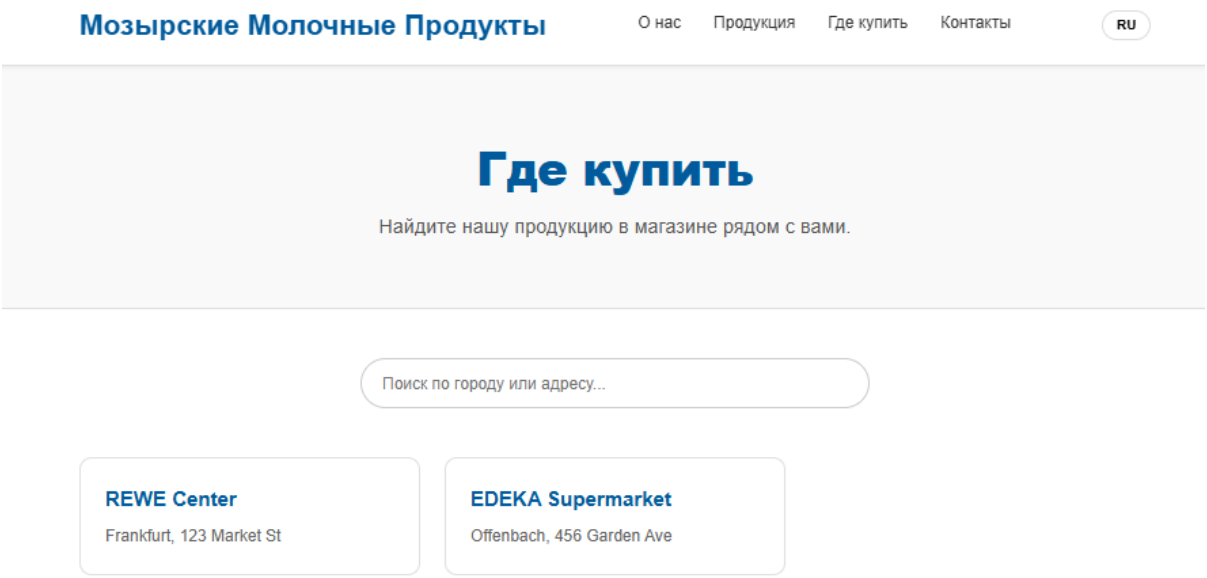

Рисунок 3.4 – Страница «Где купить»

Раздел «Контакты» является неотъемлемой частью любого корпоративного сайта, поскольку он служит прямым каналом для обратной связи с клиентами, партнёрами и соискателями. Чтобы сделать эту страницу максимально информативной и полезной, она была спроектирована с учётом всех потенциальных запросов от посетителей.

На странице размещена вся необходимая контактная информация: юридический адрес предприятия, телефоны ключевых отделов (приёмная, отдел сбыта, отдел кадров) и официальный адрес электронной почты. Для визуализации местоположения главного офиса на страницу интегрирована интерактивная **карта**. Этот элемент позволяет пользователям не только увидеть, где находится пред-

приятие, но и построить удобный маршрут проезда из любой точки. Карта является динамической, её можно масштабировать и перемещать, что обеспечивает высокий уровень удобства при взаимодействии.

Таким образом, страница «Контакты» полностью выполняет свою функцию, предоставляя исчерпывающую информацию для связи в наглядном и доступном формате. Внешний вид страницы представлен на рисунке 3.5.

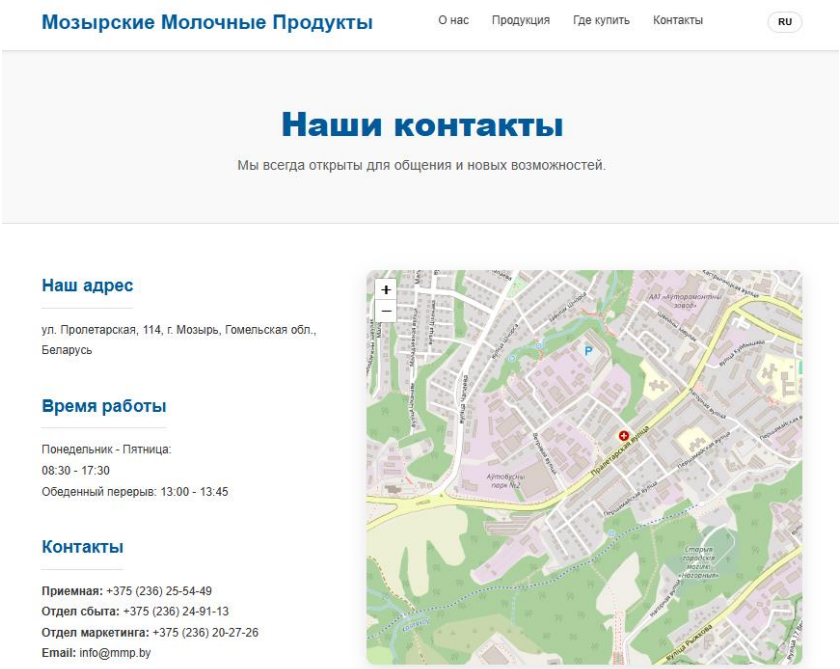

Рисунок 3.5 – Страница «Контакты»

Раздел «Продукция» является функциональным ядром всего веб-приложения. Он спроектирован как удобный и визуально приятный каталог, главная цель которого – предоставить пользователям полный и наглядный обзор всего ассортимента товаров, выпускаемых предприятием.

При переходе в этот раздел пользователь попадает на страницу со списком всей продукции. Для удобства навигации и восприятия товары представлены в виде структурированной сетки, где каждый элемент – это карточка отдельного продукта. Такая карточка содержит самую необходимую информацию для первичного ознакомления: качественное изображение товара, его точное наименование и, возможно, ключевую характеристику, например, жирность для молока или вес для творога. Весь элемент карточки является интерактивным и служит ссылкой для перехода к более подробной информации, интуитивно приглашая пользователя к дальнейшему изучению.

Общий вид страницы каталога, демонстрирующий сетку продукции, представлен на рисунке 3.6.

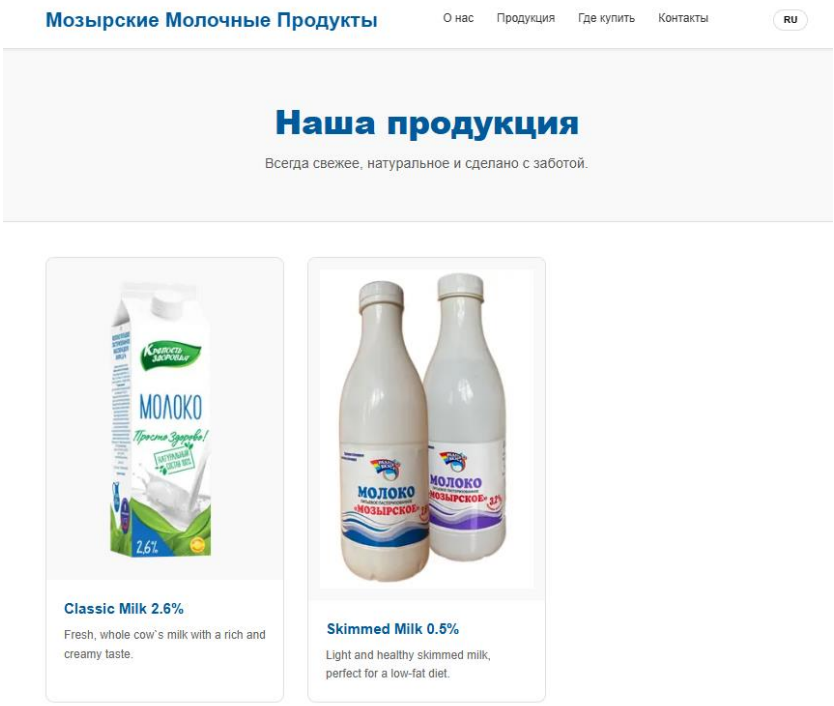

Рисунок 3.6 – Страница «Продукция»

При нажатии на любую из карточек в общем каталоге пользователь перенаправляется на страницу, полностью посвящённую выбранному продукту. Эта страница предоставляет исчерпывающую информацию, необходимую потребителю для принятия решения о покупке.

Композиция страницы, как правило, разделена на две части. С одной стороны располагается крупное, детализированное изображение продукта, позволяющее рассмотреть упаковку и внешний вид. С другой – подробный текстовый блок, который включает в себя:

– полное наименование продукта;

– развёрнутое описание: здесь раскрываются вкусовые качества, особенности консистенции и уникальные преимущества товара;

– пищевая ценность: представлена в виде структурированной таблицы или списка, где указано точное содержание белков, жиров, углеводов и калорийность на 100 грамм продукта;

– дополнительные сведения: важная для потребителя информация, такая как условия хранения, срок годности, состав и масса нетто.

Эта страница является финальной точкой в знакомстве с продуктом, предоставляя всю полноту данных в доступном и хорошо организованном виде. Пример страницы с детальной информацией о конкретном продукте представлен на рисунке 3.7.

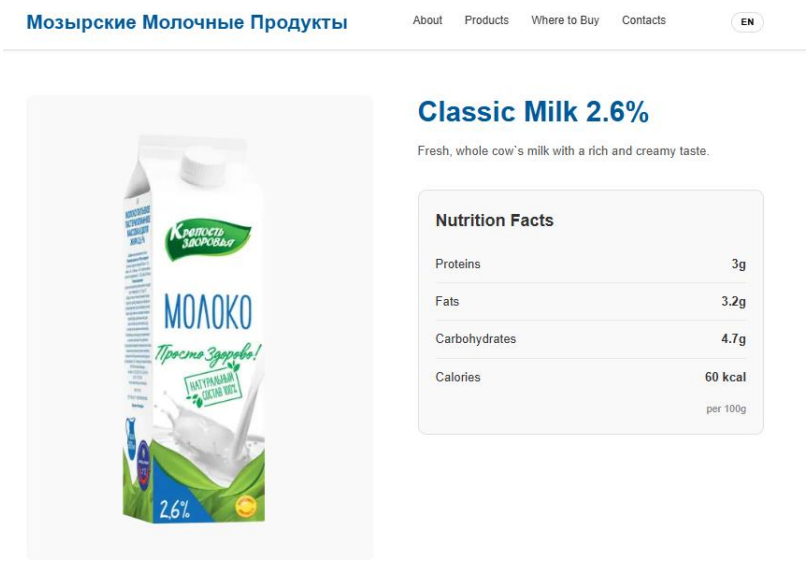

Рисунок 3.7 – Страница с детальной информацией о конкретном продукте

3.2 Соответствие с исходным заданием

Разработанное в ходе технологической практики веб-приложение «Электронный каталог продукции» в полной мере соответствует целям и требованиям, указанным в исходном задании. Проект был успешно доведён от этапа проектирования до реализации готового программного продукта, решающего поставленные бизнес-задачи.

Основные требования, которые были выполнены:

– создание современного и функционального корпоративного сайта-каталога для предприятия КПУП «Мозырские молочные продукты»;

– предоставление пользователям исчерпывающей информации об ассортименте продукции, истории компании, актуальных вакансиях и точках продаж;

– реализация многоязычного пользовательского интерфейса для охвата различных аудиторий;

– обеспечение интуитивно понятной навигации и адаптивного дизайна для корректного отображения на различных устройствах, включая настольные компьютеры, планшеты и смартфоны.

В приложении были детально проработаны все ключевые информационные разделы:

– каталог продукции, реализованный в виде двухуровневой структуры: общая страница со списком всех товаров и отдельные страницы для каждого продукта с его детальным описанием, изображением и пищевой ценностью;

– информационные страницы «О нас», «Контакты» и «Где купить», предоставляющие пользователю всю необходимую информацию о компании и способах взаимодействия с ней;

– интерактивные элементы, такие как поиск по точкам продаж и интерактивная карта с местоположением офиса, значительно улучшающие пользовательский опыт.

В ходе работы был реализован следующий основной функционал:

– просмотр общего списка продукции в виде наглядной сетки;

– переход на страницу с детальным описанием, характеристиками и изображением каждого продукта;

– ознакомление с историей, миссией и ценностями компании.

– поиск магазинов по названию города или адресу;

– просмотр контактной информации предприятия;

– динамическое переключение языка интерфейса между русским, белорусским и английским;

При разработке приложения использовался современный и востребованный технологический стек:

– *Nx* для управления архитектурой монорепозитория;

– *Next.js* с *TypeScript* для разработки клиентской части (*frontend*);

– *NestJS* с *TypeScript* для реализации серверной части (*backend*);

– *Prisma* в качестве *ORM* для типобезопасного взаимодействия с базой данных;

– *PostgreSQL* в качестве основной системы управления базами данных;

– *Docker* и *Docker Compose* для обеспечения консистентности и изоляции рабочего окружения.

Архитектура приложения была построена с применением современных паттернов проектирования, обеспечивающих его надёжность, масштабируемость и поддерживаемость:

– «репозиторий» – для создания слоя абстракции над логикой доступа к данным;

– «внедрение зависимостей» (*Dependency Injection*) – как основной принцип организации кода в *NestJS* для достижения слабой связанности компонентов;

– «объект передачи данных» (*Data Transfer Object, DTO*) – для валидации и строгого определения структуры данных в *API*;

– модульная архитектура, присущая фреймворку *NestJS*, для логической инкапсуляции функционала.

Таким образом, разработанное веб-приложение полностью выполняет поставленные задачи, демонстрирует применение современных технологий и подходов в веб-разработке и является готовым к дальнейшему развитию и внедрению.

ЗАКЛЮЧЕНИЕ

В ходе прохождения технологической практики на базе коммунального производственного унитарного предприятия «Мозырские молочные продукты» был получен ценный практический опыт в области проектирования и разработки современных веб-приложений. Работа в условиях реального производственного предприятия предоставила уникальную возможность применить теоретические знания для решения конкретных бизнес-задач, понять специфику взаимодействия *IT*-отдела с другими подразделениями и внести свой вклад в цифровизацию компании. Это позволило не только развить технические навыки, но и получить представление о том, как программные решения интегрируются в бизнес-процессы предприятий, не относящихся к сфере информационных технологий.

В рамках индивидуального практического задания было успешно спроектировано и разработано комплексное веб-приложение «Электронный каталог продукции». Данное приложение представляет собой многофункциональный корпоративный сайт-каталог, который служит полноценным цифровым представительством предприятия в сети Интернет. Система предоставляет пользователям удобный и структурированный доступ ко всей ключевой информации: подробному каталогу выпускаемой продукции с детальными описаниями и характеристиками, истории и миссии компании, списку актуальных вакансий, а также интерактивному перечню точек продаж. Разработанный продукт решает важную имиджевую и маркетинговую задачу, повышая узнаваемость бренда и улучшая коммуникацию с потребителями и партнёрами.

Для реализации проекта был использован современный и востребованный стек технологий, основанный на экосистеме *TypeScript*. Серверная часть (*backend*) была реализована с использованием фреймворка *NestJS*, который обеспечил чёткую и масштабируемую модульную архитектуру. Клиентская часть (*frontend*) была создана на базе фреймворка *Next.js*, что позволило добиться высокой производительности и отличной поисковой оптимизации (*SEO*) благодаря рендерингу на стороне сервера. Взаимодействие с базой данных *PostgreSQL* осуществлялось через *ORM Prisma*, а для управления средой разработки применялась технология контейнеризации *Docker*. Весь проект был организован в виде монорепозитория под управлением инструментария *Nx*, что способствовало переиспользованию кода и упрощению сборки.

Особое внимание в ходе разработки было уделено качеству пользовательского опыта (*User Experience*). Был реализован интуитивно понятный интерфейс с логичной навигацией и адаптивным дизайном, который корректно отображается на всех типах устройств, от настольных компьютеров до мобильных телефонов. Внедрение многоязычной поддержки (русский, белорусский, английский) значительно расширило потенциальную аудиторию сайта и повысило его доступность. Архитектура приложения была построена с применением современных паттернов проектирования, таких как «Репозиторий» и «Внедрение зависимостей», что обеспечило слабую связанность компонентов и высокую поддерживаемость кодовой базы.

Технологическая практика позволила в полной мере применить и углубить знания, полученные в университете, трансформировав их в конкретные практические навыки. Был получен опыт полного цикла разработки программного продукта: от анализа требований и проектирования архитектуры до реализации, тестирования и развёртывания. Были освоены современные подходы к созданию клиент-серверных приложений, работе с реляционными базами данных, использованию *ORM*-систем и инструментов контейнеризации. Важным результатом стало приобретение экспертизы в создании веб-приложений с использованием популярных *JavaScript*-фреймворков и серверных технологий на платформе *Node.js*.

Исходя из всего вышеизложенного, можно сделать вывод, что поставленные в рамках технологической практики цели и задачи были полностью решены и достигнуты. Разработанная система представляет собой современное, функциональное и надёжное веб-приложение, которое демонстрирует владение передовыми технологиями и готово к дальнейшему развитию. Опыт, полученный в ходе этой работы, является бесценным вкладом в профессиональное становление и послужит прочным фундаментом для дальнейшей карьеры в сфере *IT*.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Охрана труда: Президент Республики Беларусь. – Электрон. данные. – Режим доступа: https://president.gov.by/ru/belarus/social/zashhitanaselenija/ohrana-truda. – Дата доступа: 23.06.2025.

2. Мартин, Р. Чистая архитектура / Р. Мартин. – СПб.: Питер, 2024.

3. Мартин, Р. Чистый код / Р. Мартин. – СПб.: Питер, 2019.

ПРИЛОЖЕНИЕ Б

(обязательное)
Листинг программы

main.ts
```
/**
 * This is not a production server yet!
 * This is only a minimal backend to get started.
 */

import { Logger } from '@nestjs/common';
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app/app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  const globalPrefix = 'api';
  app.setGlobalPrefix(globalPrefix);
  const port = process.env.PORT || 3000;
  app.enableCors();
  await app.listen(port);
  Logger.log(
    `🚀 Application is running on: http://localhost:${port}/${globalPrefix}`
  );
}

bootstrap();
```

vacancy.repository.ts
```
import { Injectable } from '@nestjs/common';
import { Prisma, Vacancy } from '@prisma/client';
import { BasePrismaRepository } from '../base/repos/base.repository';
import { PrismaService } from '../prisma/prisma.service';

@Injectable()
export class VacancyRepository extends BasePrismaRepository<
  Vacancy,
  Prisma.VacancyFindUniqueArgs,
  Prisma.VacancyFindManyArgs,
  Prisma.VacancyCreateArgs,
  Prisma.VacancyUpdateArgs,
  Prisma.VacancyDeleteArgs
> {
  constructor(private readonly prisma: PrismaService) {
    super();
  }

  protected readonly model = this.prisma.vacancy;
}
```

store.repository.ts
```
import { Injectable } from '@nestjs/common';
import { Prisma, Store } from '@prisma/client';
import { PrismaService } from '../prisma/prisma.service';
import { BasePrismaRepository } from '../base/repos/base.repository';

@Injectable()
export class StoreRepository extends BasePrismaRepository<
```

```
  Store,
  Prisma.StoreFindUniqueArgs,
  Prisma.StoreFindManyArgs,
  Prisma.StoreCreateArgs,
  Prisma.StoreUpdateArgs,
  Prisma.StoreDeleteArgs
> {
  constructor(private readonly prisma: PrismaService) {
    super();
  }

  protected readonly model = this.prisma.store;
}
```

product.repository.ts

```
import { Injectable } from '@nestjs/common';
import { BasePrismaRepository } from '../base/repos/base.repository';
import { Product, Prisma } from '@prisma/client';
import { PrismaService } from '../prisma/prisma.service';

@Injectable()
export class ProductRepository extends BasePrismaRepository<
  Product,
  Prisma.ProductFindUniqueArgs,
  Prisma.ProductFindManyArgs,
  Prisma.ProductCreateArgs,
  Prisma.ProductUpdateArgs,
  Prisma.ProductDeleteArgs
> {
  constructor(private readonly prisma: PrismaService)
    super();
  }

  protected readonly model = this.prisma.product;
}
```

prisma.service.ts

```
import { Injectable, OnModuleInit } from '@nestjs/common';
import { PrismaClient } from '@prisma/client';

@Injectable()
export class PrismaService extends PrismaClient implements OnModuleInit {
  async onModuleInit() {
    await this.$connect();
  }
}
```

prisma.module.ts

```
import { Module } from '@nestjs/common';
import { PrismaService } from './prisma.service';

@Module({
  providers: [PrismaService],
  exports: [PrismaService],
})
export class PrismaModule {}
```

base.repository.ts

```
export abstract class BasePrismaRepository<
  TModel,
  TFindUniqueArgs,
```

```
  TFindManyArgs,
  TCreateArgs,
  TUpdateArgs,
  TDeleteArgs
> {
  protected abstract readonly model: {
    findUnique(args: TFindUniqueArgs): Promise<TModel | null>;
    findMany(args?: TFindManyArgs): Promise<TModel[]>;
    create(args: TCreateArgs): Promise<TModel>;
    delete(args: TDeleteArgs): Promise<TModel>;
    update(args: TUpdateArgs): Promise<TModel>;
  };

  findUnique(args: TFindUniqueArgs): Promise<TModel | null> {
    return this.model.findUnique(args);
  }

  findMany(args?: TFindManyArgs): Promise<TModel[]> {
    return this.model.findMany(args);
  }

  create(args: TCreateArgs): Promise<TModel> {
    return this.model.create(args);
  }

  delete(args: TDeleteArgs): Promise<TModel> {
    return this.model.delete(args);
  }

  update(args: TUpdateArgs): Promise<TModel> {
    return this.model.update(args);
  }
}
```

app.module.ts

```
import { Module } from '@nestjs/common';
import { PrismaModule } from '../prisma/prisma.module';
import { ProductsModule } from './products/products.module';
import { VacanciesModule } from './vacancies/vacancies.module';
import { StoresModule } from './stores/stores.module';

@Module({
  imports: [PrismaModule, ProductsModule, VacanciesModule, StoresModule],
  controllers: [],
  providers: [],
})
export class AppModule {}ы
```

vacancies.service.ts

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { Prisma } from '@prisma/client';
import { CreateVacancyDto } from '../../dto/vacancy/create-vacancy.dto';
import { UpdateVacancyDto } from '../../dto/vacancy/update-vacancy.dto';
import { VacancyRepository } from '../../repos/vacancy.repository';

@Injectable()
export class VacanciesService {
  constructor(private readonly vacancyRepository: VacancyRepository) {}

  create(createVacancyDto: CreateVacancyDto) {
    return this.vacancyRepository.create({ data: createVacancyDto });
  }
```

```
  findAll() {
    return this.vacancyRepository.findMany({ orderBy: { postedAt: 'desc' } });
  }

  async findOne(id: string) {
    const vacancy = await this.vacancyRepository.findUnique({ where: { id } });
    if (!vacancy) {
      throw new NotFoundException(`Vacancy with ID #${id} not found`);
    }
    return vacancy;
  }

  async update(id: string, updateVacancyDto: UpdateVacancyDto) {
    try {
      return await this.vacancyRepository.update({
        where: { id },
        data: updateVacancyDto,
      });
    } catch (error) {
      if (
        error instanceof Prisma.PrismaClientKnownRequestError &&
        error.code === 'P2025'
      ) {
        throw new NotFoundException(`Vacancy with ID #${id} not found`);
      }
      throw error;
    }
  }

  async remove(id: string) {
    await this.findOne(id);
    return this.vacancyRepository.delete({ where: { id } });
  }
}
```

vacancies.module.ts

```
import { Module } from '@nestjs/common';
import { VacanciesService } from './vacancies.service';
import { VacanciesController } from './vacancies.controller';
import { VacancyRepository } from '../../repos/vacancy.repository';
import { PrismaModule } from '../../prisma/prisma.module';

@Module({
  imports: [PrismaModule],
  controllers: [VacanciesController],
  providers: [VacanciesService, VacancyRepository],
})
export class VacanciesModule {}
```

vacancies.controller.ts

```
import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
  ParseUUIDPipe,
} from '@nestjs/common';
import { CreateVacancyDto } from '../../dto/vacancy/create-vacancy.dto';
```

```
import { UpdateVacancyDto } from '../../dto/vacancy/update-vacancy.dto';
import { VacanciesService } from './vacancies.service';

@Controller('vacancies')
export class VacanciesController {
  constructor(private readonly vacanciesService: VacanciesService) {}

  @Post()
  create(@Body() createVacancyDto: CreateVacancyDto) {
    return this.vacanciesService.create(createVacancyDto);
  }

  @Get()
  findAll() {
    return this.vacanciesService.findAll();
  }

  @Get(':id')
  findOne(@Param('id', ParseUUIDPipe) id: string) {
    return this.vacanciesService.findOne(id);
  }

  @Patch(':id')
  update(
    @Param('id', ParseUUIDPipe) id: string,
    @Body() updateVacancyDto: UpdateVacancyDto
  ) {
    return this.vacanciesService.update(id, updateVacancyDto);
  }

  @Delete(':id')
  remove(@Param('id', ParseUUIDPipe) id: string) {
    return this.vacanciesService.remove(id);
  }
}
```

stores.service.ts

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { Prisma } from '@prisma/client';
import { CreateStoreDto } from '../../dto/store/create-store.dto';
import { UpdateStoreDto } from '../../dto/store/update-store.dto';
import { StoreRepository } from '../../repos/store.repository';

@Injectable()
export class StoresService {
  constructor(private readonly storeRepository: StoreRepository) {}

  create(createStoreDto: CreateStoreDto) {
    return this.storeRepository.create({ data: createStoreDto });
  }

  findAll() {
    return this.storeRepository.findMany();
  }

  async findOne(id: string) {
    const store = await this.storeRepository.findUnique({ where: { id } });
    if (!store) {
      throw new NotFoundException(`Store with ID #${id} not found`);
    }
    return store;
  }
```

```
  async update(id: string, updateStoreDto: UpdateStoreDto) {
    try {
      return await this.storeRepository.update({
        where: { id },
        data: updateStoreDto,
      });
    } catch (error) {
      if (
        error instanceof Prisma.PrismaClientKnownRequestError &&
        error.code === 'P2025'
      ) {
        throw new NotFoundException(`Store with ID #${id} not found`);
      }
      throw error;
    }
  }

  async remove(id: string) {
    await this.findOne(id);
    return this.storeRepository.delete({ where: { id } });
  }
}
```

stores.module.ts

```
import { Module } from '@nestjs/common';
import { StoresService } from './stores.service';
import { StoresController } from './stores.controller';
import { StoreRepository } from '../../repos/store.repository';
import { PrismaModule } from '../../prisma/prisma.module';

@Module({
  imports: [PrismaModule],
  controllers: [StoresController],
  providers: [StoresService, StoreRepository],
})
export class StoresModule {}
```

stores.controller.ts

```
import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
  ParseUUIDPipe,
} from '@nestjs/common';
import { CreateStoreDto } from '../../dto/store/create-store.dto';
import { UpdateStoreDto } from '../../dto/store/update-store.dto';
import { StoresService } from './stores.service';

@Controller('stores')
export class StoresController {
  constructor(private readonly storesService: StoresService) {}

  @Post()
  create(@Body() createStoreDto: CreateStoreDto) {
    return this.storesService.create(createStoreDto);
  }
```

```
  @Get()
  findAll() {
    return this.storesService.findAll();
  }

  @Get(':id')
  findOne(@Param('id', ParseUUIDPipe) id: string) {
    return this.storesService.findOne(id);
  }

  @Patch(':id')
  update(
    @Param('id', ParseUUIDPipe) id: string,
    @Body() updateStoreDto: UpdateStoreDto
  ) {
    return this.storesService.update(id, updateStoreDto);
  }

  @Delete(':id')
  remove(@Param('id', ParseUUIDPipe) id: string) {
    return this.storesService.remove(id);
  }
}
```

products.service.ts

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { Prisma } from '@prisma/client';
import { CreateProductDto } from '../../dto/product/create-product.dto';
import { UpdateProductDto } from '../../dto/product/update-product.dto';
import { ProductRepository } from '../../repos/product.repository';

@Injectable()
export class ProductsService {
  constructor(private readonly productRepository: ProductRepository) {}

  async create(createProductDto: CreateProductDto) {
    return await this.productRepository.create({
      data: createProductDto,
    });
  }

  findAll() {
    return this.productRepository.findMany();
  }

  async findOne(id: string) {
    const product = await this.productRepository.findUnique({
      where: { id },
    });

    if (!product) {
      throw new NotFoundException(`Product with ID #${id} not found`);
    }

    return product;
  }

  async update(id: string, updateProductDto: UpdateProductDto) {
    try {
      return await this.productRepository.update({
        where: { id },
        data: updateProductDto,
```

```
      });
    } catch (error) {
      if (error instanceof Prisma.PrismaClientKnownRequestError) {
        if (error.code === 'P2025') {
          throw new NotFoundException(`Product with ID #${id} not found`);
        }
      }
      throw error;
    }
  }

  async remove(id: string) {
    await this.findOne(id);
    await this.productRepository.delete({ where: { id } });
  }
}
```

products.module.ts

```
import { Module } from '@nestjs/common';
import { ProductsService } from './products.service';
import { ProductsController } from './products.controller';
import { PrismaModule } from '../../prisma/prisma.module';
import { ProductRepository } from '../../repos/product.repository';

@Module({
  imports: [PrismaModule],
  controllers: [ProductsController],
  providers: [ProductsService, ProductRepository],
})
export class ProductsModule {}
```

products.controller.ts

```
import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
  ParseUUIDPipe,
} from '@nestjs/common';
import { ProductsService } from './products.service';
import { CreateProductDto } from '../../dto/product/create-product.dto';
import { UpdateProductDto } from '../../dto/product/update-product.dto';

@Controller('products')
export class ProductsController {
  constructor(private readonly productsService: ProductsService) {}

  @Post()
  async create(@Body() createProductDto: CreateProductDto) {
    return await this.productsService.create(createProductDto);
  }

  @Get()
  findAll() {
    return this.productsService.findAll();
  }

  @Get(':id')
  async findOne(@Param('id', ParseUUIDPipe) id: string) {
```

```
    return await this.productsService.findOne(id);
  }

  @Patch(':id')
  async update(
    @Param('id', ParseUUIDPipe) id: string,
    @Body() updateProductDto: UpdateProductDto
  ) {
    return await this.productsService.update(id, updateProductDto);
  }

  @Delete(':id')
  async remove(@Param('id', ParseUUIDPipe) id: string) {
    await this.productsService.remove(id);
  }
}
```

create-product.dto.ts

```
import { IsInt, IsNotEmpty, IsNumber, IsString } from 'class-validator';

export class CreateProductDto {
  @IsNotEmpty()
  @IsString()
  name: string;

  @IsNotEmpty()
  @IsString()
  description: string;

  @IsNotEmpty()
  @IsString()
  imageUrl: string;

  @IsNotEmpty()
  @IsNumber()
  proteins: number;

  @IsNotEmpty()
  @IsNumber()
  carbs: number;

  @IsNotEmpty()
  @IsNumber()
  fats: number;

  @IsNotEmpty()
  @IsNumber()
  @IsInt()
  calories: number;
}
```

update-product.dto.ts

```
import { PartialType } from '@nestjs/mapped-types';
import { CreateProductDto } from './create-product.dto';

export class UpdateProductDto extends PartialType(CreateProductDto) {}
```

create-store.dto.ts

```
import { IsNotEmpty, IsNumber, IsString } from 'class-validator';

export class CreateStoreDto {
  @IsString()
  @IsNotEmpty()
  name: string;

  @IsString()
  @IsNotEmpty()
  address: string;

  @IsString()
  @IsNotEmpty()
  city: string;

  @IsString()
  @IsNotEmpty()
  postalCode: string;

  @IsNumber()
  latitude: number;

  @IsNumber()
  longitude: number;
}
```