Государственное бюджетное профессиональное

образовательное учреждение Московской области

"Колледж "Коломна"

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

НА ТЕМУ:

**«****Разработка автоматизированной информационной системы, реализующей концепцию перераспределения продуктов питания, на основе технологии asp.net MVC»**

ВКР выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Д.А. Минаев/

(подпись) И.О. Фамилия

Коломна

2024

**З А Д А Н И Е**

**на выпускную квалификационную работу студента**

|  |
| --- |
| *Минаева Дениса Алексеевича* |

(фамилия, имя, отчество)

|  |  |
| --- | --- |
| 1. Тема работы: | «Разработка автоматизированной информационной |
| системы, реализующей концепцию перераспределения продуктов питания, | |
| на основе технологии asp.net MVC» | |

утверждена приказом от «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ № \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| 2. Срок сдачи студентом законченной работы | |  |
| 3. Исходные данные к работе | 1) Операционная система Windows 10 | |
| 2) Язык разработки: объектно-ориентированный язык C# | | |
| 3) Интегрированная среда разработки Microsoft Visual Studio 2022 | | |
| 4) Система управления базами данных PostgreSql | | |
| 5) Платформа для веб-разработки asp.net | | |

4. Содержание пояснительной записки (перечень подлежащих разработке вопросов)

|  |
| --- |
| 1. Реферат |
| 2. Введение |
| 3. Обзорно-аналитическая часть |
| 4. Основная часть |
| 5. Экология и безопасность жизнедеятельности |
| 6. Экономическая часть |
| 7. Заключение |
| 8. Список использованных источников |
| 9. Приложения |

|  |  |
| --- | --- |
| 5. Перечень графического материала | ER-диаграмма, схема данных, |
| UML-диаграмма, блок-схема, интерфейс | |

СОДЕРЖАНИЕ

[РЕФЕРАТ 5](#_Toc107953040)

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 6](#_Toc107953041)

[ВВЕДЕНИЕ 7](#_Toc107953042)

[1. ОБЗОРНО-АНАЛИТИЧЕСКАЯ ЧАСТЬ 9](#_Toc107953043)

[1.1. Ценность продуктов и проблема пищевых отходов 9](#_Toc107953044)

[1.2. Модели фудшеринга и её реализации 10](#_Toc107953045)

[2. ОСНОВНАЯ ЧАСТЬ 15](#_Toc107953046)

[2.1. Проектирование 15](#_Toc107953047)

[2.1.1. Автоматизированная информационная система 15](#_Toc107953048)

[2.1.2. Системный анализ 16](#_Toc107953049)

[2.1.3. Инфологическое проектирование 17](#_Toc107953050)

[2.1.4. Выбор СУБД 18](#_Toc107953051)

[2.1.5. Даталогическое проектирование 19](#_Toc107953052)

[2.1.6. Физическое проектирование 20](#_Toc107953053)

[2.2. Разработка клиент-серверной части 24](#_Toc107953054)

[2.2.1. Фреймворк Asp.net 24](#_Toc107953055)

[2.2.2. Шаблон MVC 25](#_Toc107953056)

[2.2.3. Шаблон DI 26](#_Toc107953057)

[2.2.4. Структура программных модулей 27](#_Toc107953058)

[2.2.5. Аутентификация и авторизация 29](#_Toc107953059)

[2.2.6. Профиль пользователя 34](#_Toc107953060)

[2.2.7. Продукты и каталог 36](#_Toc107953061)

[2.2.8. Реализация обмена сообщениями 40](#_Toc107953062)

[2.2.9. Тестирование ПС 43](#_Toc107953063)

[3. ЭКОЛОГИЯ И БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ 47](#_Toc107953064)

[3.1. Введение 47](#_Toc107953065)

[3.2. Безопасность при работе с ЭВМ 47](#_Toc107953066)

[3.3. Рабочий распорядок и перерывы 48](#_Toc107953067)

[3.4. Расчёт освещённости помещений 49](#_Toc107953068)

[3.5. Общие положения о пожарной безопасности 51](#_Toc107953069)

[3.6. Правила эвакуации при пожаре 52](#_Toc107953070)

[4. ЭКОНОМИЧЕСКИЙ РАЗДЕЛ 54](#_Toc107953071)

[4.1. Введение 54](#_Toc107953072)

[4.2. Расчет трудоемкости разработки ПО 55](#_Toc107953073)

[4.3. Составление сметы затрат на разработку 61](#_Toc107953074)

[4.3.1. Расчёт затрат на оборудование и ПО 61](#_Toc107953075)

[4.3.2. Расчёт материальных затрат на расходные материалы 62](#_Toc107953076)

[4.3.3. Расчёт затрат на электроэнергию 63](#_Toc107953077)

[4.3.4. Расчёт амортизационных отчислений 63](#_Toc107953078)

[4.3.5. Основная заработная плата исполнителей 65](#_Toc107953079)

[4.3.6. Дополнительная заработная плата и фонд оплаты труда 65](#_Toc107953080)

[4.3.7. Отчисления на социальные нужды 65](#_Toc107953081)

[4.3.8. Суммарные затраты на разработку 66](#_Toc107953082)

[4.4. Оценка эффективности 67](#_Toc107953083)

[4.4.1. Социальная эффективность 68](#_Toc107953084)

[4.4.2. Экологическая эффективность 69](#_Toc107953085)

[ЗАКЛЮЧЕНИЕ 71](#_Toc107953086)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 73](#_Toc107953087)

[ПРИЛОЖЕНИЕ А 74](#_Toc107953088)

[ПРИЛОЖЕНИЕ Б 80](#_Toc107953089)

# ****РЕФЕРАТ****

Расчетно-пояснительная записка к выпускной квалификационной работе на тему «Разработка автоматизированной информационной системы, реализующей концепцию перераспределения продуктов питания, на основе технологии asp.net mvc» состоит из пояснительной записки на 101 страницу, 4 разделов и 2 приложений.

1. В разделе «Обзорно-аналитическая часть» описываются основные понятия решаемой задачи, производится анализ аналогов и рассматривается актуальность выбранной темы
2. В разделе «Основная часть» описан процесс решения поставленной задачи и инструментальные средства для её реализации
3. В разделе «часть безопасности жизнедеятельности» производится анализ негативных факторов, воздействующих на пользователя при эксплуатации ЭВМ. Производится рекомендуемый распорядок дня на рабочем месте, а также описывается действия при возникновения экстренной ситуации на предприятии.
4. В разделе «Экономика» рассчитывается трудоёмкость разработки программного обеспечения, стоимость затрат на создание программного продукта, его социальная и экологическая эффективность, определен срок окупаемости и составлена смета затрат

В ходе данной работы была разработана автоматизированная информационная система для перераспределения продуктов питания. Результат работы выполнен в виде веб-приложения и предоставляет пользователям обмениваться продуктами питания и коммуницировать посредством чата. Приложение адаптивно для всех современных браузеров и даёт возможность использоваться с устройств с различным расширением экрана начиная с персонального компьютера, заканчивая смартфонами и планшетами.

Ключевые слова: Автоматизированная информационная система, Веб-приложение, продукты, фудшеринг, адаптивность, ASP.NET, PostgreSql, MVC, DI

# ****ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ****

В настоящем ВКР применяют следующие сокращения и обозначения:

БД – база банных

БЖД – безопасность жизнедеятельности

ЗП – заработная плата

НКО – некоммерческая организация

ПК – персональный компьютер

ПО – программное обеспечение

ПС – программная система

СУБД – система управления базами данных

ФОТ – фонд оплаты труда

ЭВМ – электронно-вычислительная машина

MVC - Model View Controller(Модель-представление-контроллер)

DI - Dependency injection(Внедрение зависимости)

# ****ВВЕДЕНИЕ****

В современном мире человек тесно связан с продуктами питания. Он приобретает, поглощает и производит продукты ежедневно, но зачастую не все товары удаётся употребить или же они перестают быть необходимыми и в впоследствии выбрасываются, попадая на свалку. Исследование, проведённое в 2021 году Организацией Объединённых наций с сфере окружающей среды, показало, что около трети всех пищевых продуктов выбрасываются и от 8 до 10 процентов парниковых газов связаны именно с этим. Поэтому во всём мире существует большая потребность в более грамотном распределении, совместном использовании и предотвращении попадания в утиль хороших продуктов.

Движение, занимающееся проблемой распределения еды, имеет название фудшеринг, что дословно означает процесс совместного использования и деления еды. Фудшеринг зародился как способ предотвращения образования пищевых отходов и является проявлением экономики совместного потребления. Также он выполняет важную социальную функцию – помогает обеспечить продовольствием всех желающих в том числе малообеспеченные слои населения, предоставляя им бесплатный и свободный доступ к необходимым им продуктам питания.

Объектом исследования является создание веб-приложения по распределению продуктов питания в целях рационального использования природных ресурсов и сокращения образования пищевых отходов.

Предметом исследования являются способы перераспределения продуктов питания и программные средства для создания веб-приложений.

Актуальность работы заключается в разработке автоматизированной информационной системы, реализующей концепцию перераспределения продуктов питания. Данная система предоставляет из себя интернет-приложение, которое позволяет безвозмездно делиться продуктами и уменьшать продовольственные потери и отходы.

Для разработки будет использоваться фреймворк ASP.NET а в качестве высокоуровневого языка программирования C# 10. Процесс разработки программного обеспечения будет происходить в интегрированной среде разработки (IDE) Microsoft Visual Studio 2022.

Основными требованиями при разработке было создание веб-приложения, которое должно содержать в себе весь необходимый функционал для удобной коммуникации пользователей и интуитивно понятный интерфейс. Общение между участниками должно производится с помощью чата и содержать в себе функционал отправки, получения сообщений, а также сохранение истории. Также должна быть разработана удобная система размещения продуктов. У пользователя должна быть возможность размещать, скрывать, удалять и добавлять в избранное продукты. Для обмена и выбора продуктов необходимо спроектировать каталог продуктов содержащий в себе функционал поиска и перехода на страницу с полным описанием товара и данных его владельца.

Практическая значимость работы заключается в разработке веб-приложения для предотвращения отходобразования и оказанию продовольственной помощи нуждающимся.

# ****ОБЗОРНО-АНАЛИТИЧЕСКАЯ ЧАСТЬ****

* 1. **Ценность продуктов и проблема пищевых отходов**

Продукты питания являются незаменимой и всеобъемлющей частью жизни любого человека. Ежедневно для оптимальной жизнедеятельности человека он должен питаться не менее трёх раз за день удовлетворяя свои естественные потребности. Именно поэтому пища является важнейшим источником энергии, обогащает человека полезными элементами и витаминами.

Каждый пищевой продукт содержит набор различных компонентов имеет в составе такие вещества как: белки, жиры, углеводы, витамины и минералы.

Однако каждое продукт имеет свой срок годности - рекомендуемое время хранения, при котором товар считается качественным и пригодным для употребления или использования. Срок годности применяется к большинству продуктов питания, а также к некоторым промышленным продуктам эксплуатационные качества, которых могут со временем уменьшаться или полностью утрачиваться.

В большинстве случаев, товары, на упаковке которых указан срок годности не рекомендуется употреблять после истечения данного срока. Так, пищевые продукты приходят в непригодность из-за проявления активности бактерий, физических процессов внутри продукта, неправильных условий хранения, а также деформирования товара. Наиболее подвержены порче продукты животного происхождения, такие как молоко, рыба, мяса и яйца. Их стоит утилизировать сразу при истечении срока хранения и при проявлении признаков порчи, таких как деформация, изменения цвета и запаха.

Следующая категория подверженная порче это хлебобулочные изделия, овощи, фрукты и другие продукты растительного происхождения. Они имеют более длительный срок хранения и менее подвержены порче. К каждой категории продуктов следует подходить внимательно, следовать условиям хранения и приготовления, так ко множеству продуктов применятся определённая температура хранения и приготовления и нарушение этих правил может привести к возникновению опасных для употребления микроорганизмов.

Важным принципом фудшеринга является максимально возможное сохранение продуктов и предоставление их нуждающимся в надлежащей и свежей форме, поэтому в большинстве сервисов каждый пользователь желающий стать волонтёром должен пройти тестирование, чтобы стать надёжным, компетентным и хорошо разбирающимся в процессах связанных с сортировкой продуктов членом общества.

## Модели фудшеринга и её реализации

* + 1. **Модель распределения C2C**

**Consumer-to-consumer** (от англ. Consumer-to-consumer, буквально — «Потребитель для Потребителя») – модель, который подразумевает под собой отношениям между потребителем и конечным пользователем, когда покупатель и продавец не являются предпринимателями в юридическом смысле этого слова.

В рамках данной модели люди размещают рекламу на специальной онлайн-платформе или в социальных сетях с фотографиями и описаниями продуктов, которые они хотят раздать. Заинтересованные лица связываются с рекламодателем и договариваются о времени и месте доставки товара. Обычно перевод бесплатный. В некоторых случаях через платформу продаются не только купленные продукты, но и блюда, приготовленные дома.

* + 1. **Модель распределения B2C**

**Business-to-consumer** (B2C) используется пищевыми компаниями, розничными магазинами и предприятиями общественного питания. Компании и физические лица отдают не задействованные или неликвидные продукты питания, чтобы уменьшить экологический след своего бизнеса, а также внести свой вклад в благотворительный процесс.

Некоммерческий фудшеринг B2C основан на работе волонтеров, которые по предварительной договоренности выезжают в организацию для сбора еды, затем раздают ее своим подопечным, социально незащищенным группам через различные площадки фудшеринга а также благотворительные некоммерческие организации .

* 1. **Анализ текущего состояние фудшеринг сервисов**

В России существуют несколько крупных фудшеринг сервисов, которые помогают людям получить качественные продукты питания и предотвратить их попадание на свалку. Каждый сервис реализует различные модели и принципы для реализации поставленной цели, но каждый из них ставит под собой цель предоставлять свои услуги на некоммерческой и благотворительной основе. Благодаря им, начиная с 2012 года удалось сохранить более 38 миллионов килограмм продуктов.

**Фонд продовольствия Русь**

Первый зарегистрированный в России фудшеринг сервис - Фонд продовольствия «Русь». Он начал свою деятельность как благотворительный фонд в 2012 г и зарекомендовал себя как организацию, которая помогает предоставлять бесплатные качественные продукты от производителей, магазинов и волонтёров социально нуждающимся гражданам через сотрудничающие с ними общественные организации. Благодаря им была оказано помощь более 3 миллионов малоимущим семьям с детьми и нуждающимся пенсионерам. Каждый продукт проходит отбор, сортировку и контроль качества с помощью волонтёров фонда.

По данным отчёта фонда за время своего существования он смог добиться тесного сотрудничества с такими компаниями как Danone, Mars International, Cargill, Procter Gamble ,а также с такими крупной розничной сетью как Дикси. Только с начала 2018 года объем задействованной продукции составил более 4 тысяч тонн и обшей стоимостью более 700 миллионов рублей. Структурная схема работы Фона Русь показана на рисунке 1



Рисунок 1– Структурная схема работы фонда Русь

**Фудшеринг.Москва**

Фудшеринг.Москва представляется из себя автоматизированную платформу для волонтёров посредством которой осуществляется приём продуктов у компаний-доноров для передачи нуждающимся. В настоящий момент координация взаимодействия с компаниями-донорами реализована через социальные сети и мессенджеры.

Волонтеры забирают у организаций-доноров продукты с истекшим сроком годности. Заинтересованный человек должен пройти тест на знание правил волонтерского движения, а затем зарегистрироваться в сообществе. Затем он попадает в базу участников и получает уведомления о предстоящих выездах на организацию. Волонтер сам распределяет еду, которую он получил. Он может взять себе что-то из продуктов, которые ему оставили, а затем раздавать нуждающимся. Кроме того, волонтерами в этом проекте могут стать люди, состоящие во многих других благотворительных организациях, которые хотят раздавать продукты своим подопечным.

Таким образом процесс распределения продуктов на платформе Фудшеринг.Москва реализуется следующим образом как показано на рисунке 2:

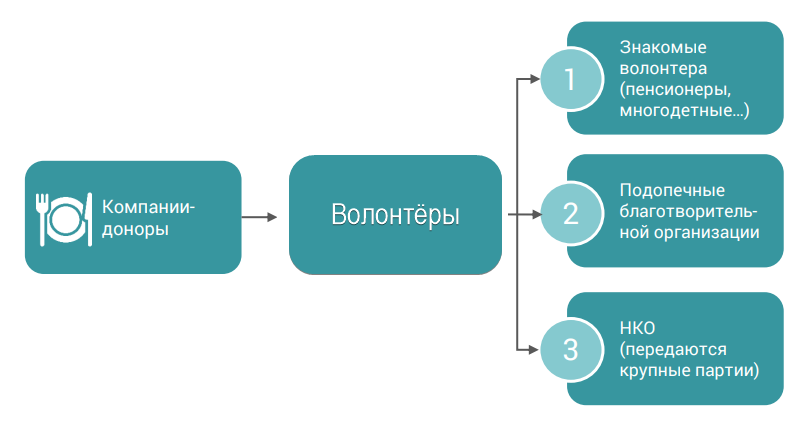


Рисунок 2 – Структурная схема работы фонда Фудшеринг.Москва

**Фудшеринг «Отдам еду даром»**

Данная компания реализую свою благотворительную деятельной в социальной сети ВКонтакте, в которой имеет группу, на которой частные лица или организации размещают объявления с указанием того, какие товары готовы раздать бесплатно. Пользователь, заинтересованный в продуктах, указывает свою заинтересованность под публикацией с товаром и в дальнейшем напрямую договаривается о времени и месте встрече с автором объявления.

Благодаря свой деятельности фудшеринг «Отдам еду даром» снабжает продуктами более 50000 тысяч людей ежемесячно сотрудничая с множеством компаний. Так, по данным из отчёта проекта они спасают около 5 тонн свежей продукции в месяц. Структурная схема работы продемонстрирована на рисунке 3.



Рисунок 3 – Структурная схема работы фонда «Отдам еду даром»

* 1. **Перспективы развития**

Согласно экспертным оценкам, для реализации потенциала фудшеринга в России необходимо три ключевых компонента:

* комфортное государственное регулирование, связанное с определением статуса продукции для фудшеринга и ее налогообложением;
* развитие технологических платформ, обеспечивающих быстрый рост количества участников и мобильность в распределении еды;
* нацеленность бизнеса, работающего с продуктами питания, на максимально эффективное их использование.

1. **ОСНОВНАЯ ЧАСТЬ**
   1. **Проектирование**
      1. **Автоматизированная информационная система**

Под автоматизированной информационной системой(АИС) принято понимать комплекс средств и методов основной задачей которых является хранение, обработка и отображения информации с целью достичь поставленной цели пользователем. При проектировании АИС принято использовать ЭВМ для упрощения вычислений и обработки большого количества информации, а также прикладное программное обеспечение основная задача которой упростить процесс разработки ПО.

На первых этапах разработки АИС следует уделить внимание проектированию эффективного хранилища данных с помощью которого можно будет хранить и оперировать необходимой для работы системы задачи. В качестве такого хранилища чаще всего выступает специализированная база данных(БД), которая помимо основных задач при работе с информаций позволяет поддерживать принцип целостности.

Проектирование БД состоит из нескольких взаимосвязанных шагов основная цель которых заключается в переходе от неформального и словесного описания информационной структуры предметной области к формализированному.

В большинстве случаев при разработке БД принимают следующую последовательность шагов:

1. Системный анализ – подразумевает под собой словесное описание информационных объектов предметной области, взаимосвязь отдельных компонентов системы, а также модели применимые в задаче.
2. Инфологическое проектирование заключается в описании сущностей и связей между ними. Данный этап не оперируется на конкретную БД.
3. Выбор Системы Управления Базами Данных (СУБД). Описание мотивации выбора конкретного типа СУБД
4. Даталогическое проектирование БД – описание данных на языке выбранной СУБД.
5. Физическое проектирование БД - разработка наиболее эффективного размещения данных в БД с учётом выбранной СУБД
   * 1. **Системный анализ**

Системный анализ подразумевает под собой словесное описание и обоснование предметной области результатом которой являются логически связанная последовательность сущностей, взаимодействующих между собой и решающих потребности пользователей.

Литературной базой для проведения анализа служит информационные справочники, специализированные книги и методические материалы, а также тексты научных исследований.

Объектом исследования является проблема нерационального перераспределения продуктов питания.

Предметом исследования являются люди, находящиеся в дефиците материальных средств и продуктов.

Произведём описание необходимых объектов и произведём специфику их применения:

* **Пользователи**. Задачей разработки является создания веб-приложения, в рамках которого люди смогут взаимодействовать с собой.
* **Продукты**. Основной целью коммуникации пользователей является обмен продуктами
* **Сообщения**. Одним из наиболее удобных средств общения считается чат, позволяющий обмениваться данными с информационной сети.

### Инфологическое проектирование

На данном этапе основной задачей является - определить предметную область и сформировать взгляд на нее со стороны пользователей.

В основе модели лежит объяснение структуры и процессов информационных потребностей пользователей в ключевых терминах, понятных пользователю и не зависящих от реализации БД.

В качестве способа для проектирования на данном шаге приянто брать метод "сущность–связь". Определим основные термины, которые будут использованы для решения данной задачи:

**Сущность** —это предмет или явление, которые в системе будут обрабатываться и взаимодействовать. Она имеет ряд таких свойств как наименование и тип. В зависимости функций различают сильные сущности, которые уникальны и существуют сами по себе независимо от других и слабые сущности завиящие от существования сильных.

**Атрибут** – обособленное свойство сущности. Он имеет такие параметры как:

* имя, предназначенное для идентификации,
* тип данных для наиболее полного описания в системе хранимой в нем информации
* ограничение целостности.

**Связь** – это обособленная и осмысленное соединение между взаимодействующими сущностями. Для связи указывается название, тип (факультативная или обязательная), кардинальность (1:1, 1:n или m:n) и степень (унарная, бинарная, тернарная или n-арная).

В следствие анализа предметной области была разработана следующая ER-модель, представленная на рисунке 4.

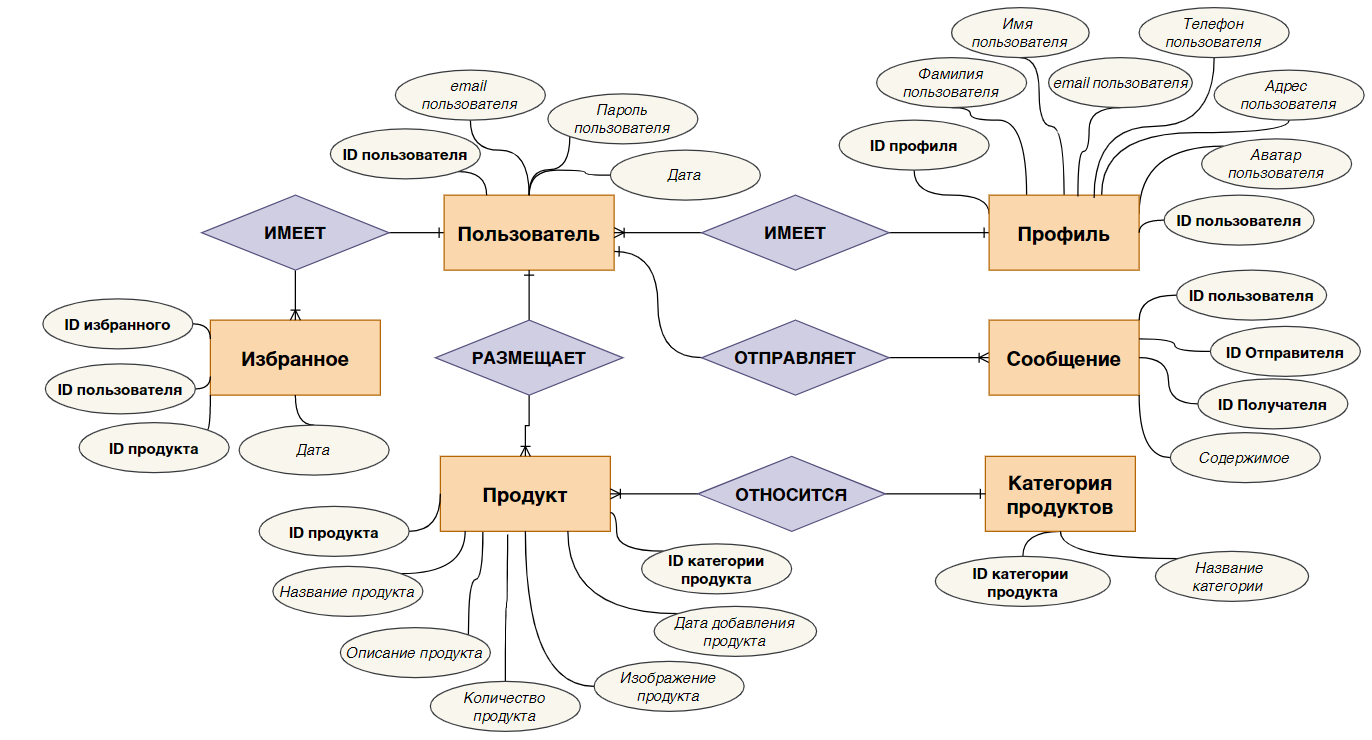


Рисунок 4 – ER-диаграмма

* + 1. **Выбор СУБД**

Важным этапом проектирования БД является этап выбора СУБД. На нем исходя из учёта поставленных задач происходит выбор системы управления хранилищем данных с учётом таких критериев как тип модели данных, используемых в разработке, характеристики производительности и скорость оперирования информацией, функциональные возможности и преимущества, комфортность использования и надёжность при работе, а также рыночная стоимость программного обеспечения для управления базой данных



Рисунок 5– Логотип компании PostgreSQL

В следствие анализа ряда вариантов программных продуктов был выбрана СУБД PostgreSQL, которая является бесплатным программным обеспечением и нацелена на лёгкое и доступное расширение базы данных. Основными причинами выбора данной СУБД является ряд следующих преимуществ:

* Полностью бесплатна и имеет открытый исходный код
* Поддерживает все популярные функции
* Возможность асинхронного выполнения запросов
* Имеет надёжные механизмы транзакций и репликации
* Легко расширяема на веб сервере

Логотип компании представлен на рисунке 5

* + 1. **Даталогическое проектирование**

На данном этапе необходимо разработать концептуальнулью модель данных, которая будет отражать логические взаимосвязи данных в БД. Проектирование осуществялется на основе реалиционной модели данных и заключается в нормализации отношений и приведению модели к третьей нормальной форме путём последовательных транзакций. Данные операция позволяют обеспечить наиболее праивльное выполнениние требований ссылочной целостности и сформировать БД не содержащую в себе аномалии и дублирования данных.

Проведём нормализацию отношений в проектируемой модели на основе данных полученных на этапе инфологического проектирования и обеспечим выполненение поставленных требований целостности согласно специфике выбранной задачи. Результат нормализции представлен на рисунке 6.

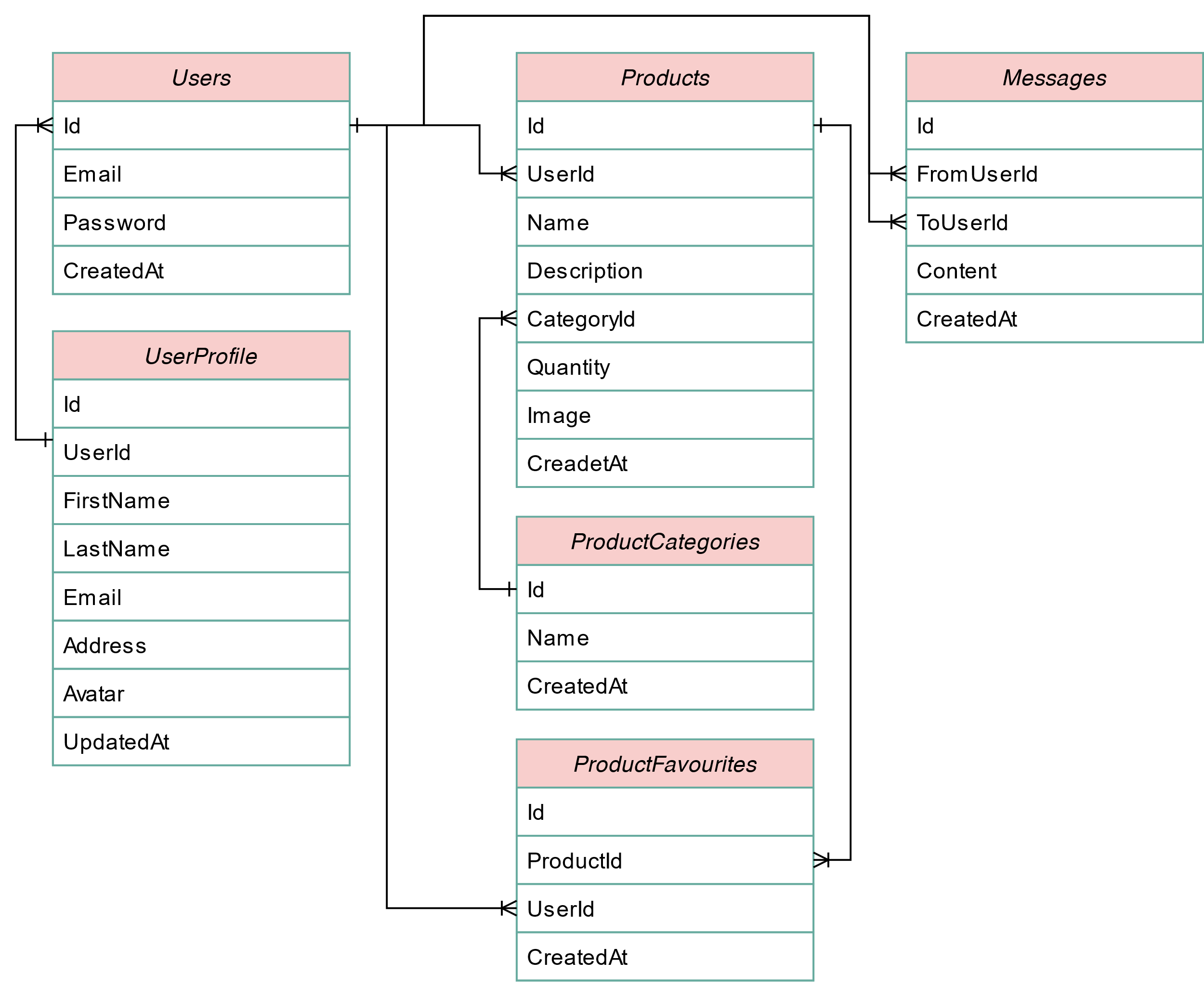


Рисунок 6 – Схема БД

* + 1. **Физическое проектирование**

Физическое проектирование является последним шагом в разработке БД. На данном этапе разработчик принимает решения о способах реализации БД с учётом специфики задачи, а также возможности выбранной СУБД. Физическое состоит из процесса создание таблиц и связей между ними в зависимости от возможностей и особенностей СУБД.

В ходе данного этапа были разработаны следующие таблицы в СУБД PostgreSql:

* Таблица Users (Таблица 1) необходима для хранения данных о пользователях.
* Таблица UsersProfile (Таблица 2) необходима для хранения информации о профилях пользователей, а также для управления и упрощения добавления, редактирования и удаления пользователей.
* Таблица Products (Таблица 3) необходима для хранения данных о продуктах, а также для управления и упрощения добавления, редактирования и удаления продуктов для конкретного пользователя.
* Таблица ProductsCategories (Таблица 4) необходима для хранения данных о категориях продуктов, а также для управления и упрощения внедрения новых категорий.
* Таблица ProductsFavourites (Таблица 5) необходима для хранения данных о избранных товаров пользователя, а также для управления и упрощения добавления и удаления избранных товаров.
* Таблица Messages (Таблица 6) необходима для хранения отправленных пользователями сообщений и информацию о них.

Таблица 1 – Таблица пользователей Users

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Значение NULL |
| **Id** | uuid | NOT NULL |
| Email | varchar | NOT NULL |
| Password | varchar | NOT NULL |
| CreatedAt | timestamptz | NOT NULL |

Таблица 2 – Таблица профилей пользователей UsersProfile

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Значение NULL |
| **Id** | uuid | NOT NULL |
| UserId | uuid | NOT NULL |
| FirstName | varchar | NULL |
| LastName | varchar | NULL |
| Email | varchar | NOT NULL |
| Adress | varchar | NULL |
| Phone | varchar | NULL |
| Avatar | bytea | NULL |

Таблица 3 – Таблица продуктов Products

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Значение NULL |
| **Id** | uuid | NOT NULL |
| UserId | bytea | NOT NULL |
| Name | varchar | NOT NULL |
| Description | varchar | NOT NULL |
| CategoryId | int4 | NOT NULL |
| Quantity | varchar | NOT NULL |
| Image | bytea | NOT NULL |
| Created\_At | timestamptz | NOT NULL |

Таблица 4 – Таблциа категорий продуктов ProductsCategories

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Значение NULL |
| **Id** | uuid | NOT NULL |
| Name | varchar | NOT NULL |
| CreatedAt | timestamptz | NOT NULL |

Таблица 5 – Таблица избранных продуктов ProductsFavourites

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Значение NULL |
| **Id** | uuid | NOT NULL |
| ProductId | uuid | NOT NULL |
| UserId | uuid | NOT NULL |
| CreatedAt | timestamptz | NOT NULL |

Таблица 6 – Таблица сообщений Messages

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Значение NULL |
| **Id** | uuid | NOT NULL |
| FromUserId | uuid | NOT NULL |
| ToUserId | uuid | NOT NULL |
| Content | varchar | NOT NULL |
| CreatedAt | timestamptz | NOT NULL |

Исходя их этого полученная схема БД для поставленной задачи имеет следующий вид как показано на рисунке 7.

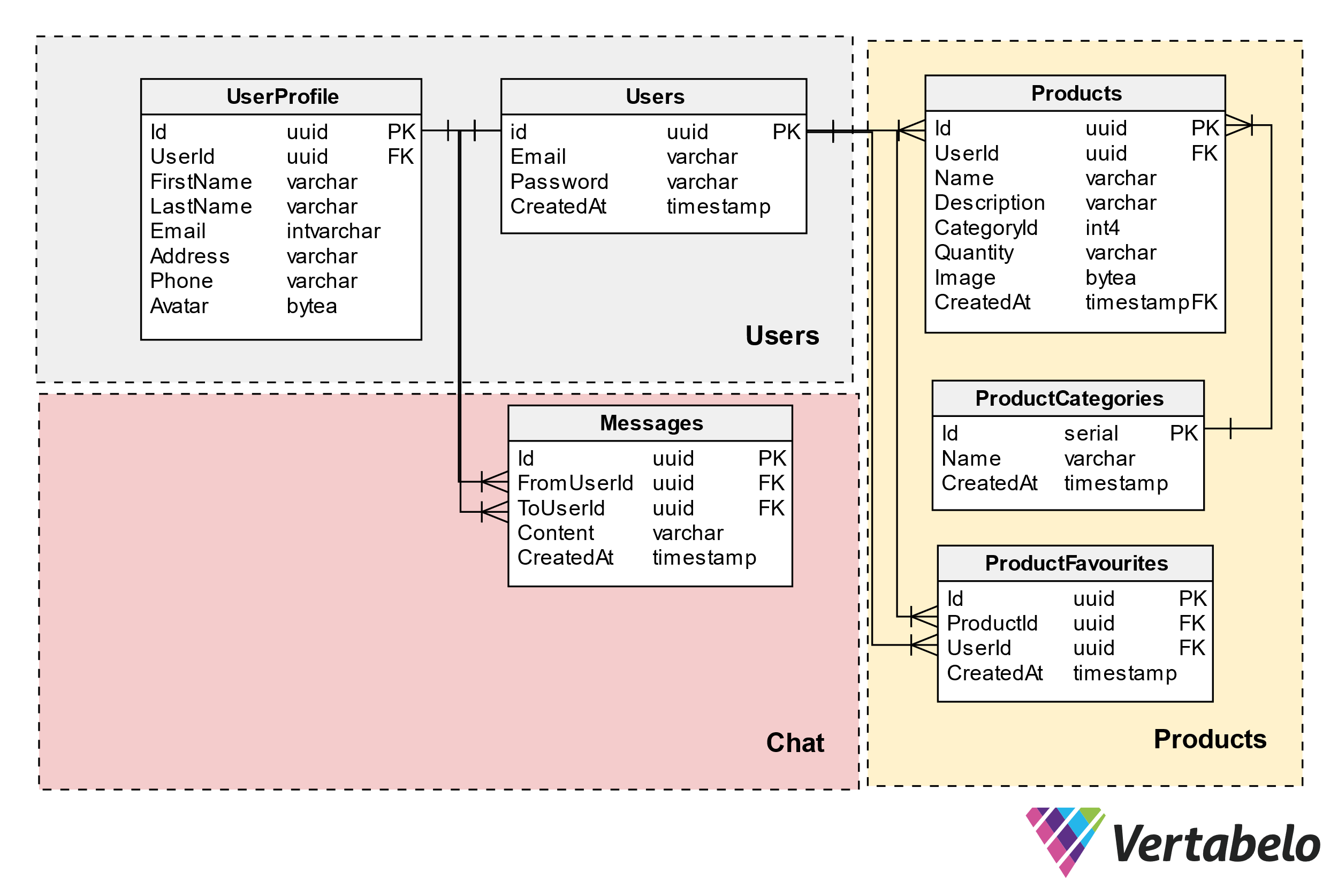


Рисунок 7– схема БД для выбранной субд

* 1. **Разработка клиент-серверной части**

### Фреймворк Asp.net



Рисунок 8 – Логотип платформы ASP.NET

ASP.NET - представляется из себя технологию для упрощения создания веб-приложений на основе платформы .NET. Логотип платформы изображен на рисунке 8. Данное технологическое решение поддерживается компанией Microsoft и позволяет создавать автоматизированные информационные системы различного уровня сложности и масштабируемости. Написание кода происходит на высокоуровневом языке C#, который использует объектно-ориентированный подход. Одним из преимуществ ASP.NET является огромный выбор инструментов и библиотек для решения поставленных задач.

Ключевыми причинами выбора стали следующие особенности платформы:

* Структура обработки и написания веб-запросов на языке C#
* Синтаксис шаблонов веб-страниц RazorPages, который позволяется создавать динамические представления комбинирую язык программирования HTML и C#.
* Готовые библиотеки и инструменты для работы с такими шаблонами как Model View Controller (MVC) и Dependency injection (DI)
* Удобные средства для подключения и реализации аутентификации
* Библиотека ASP.NET также содержит в себе дополнения для подсветки синтаксиса, автозамены и расширение функционала автодополнения IntelliSense

### Шаблон MVC

Для структуризации и разделения проекта на взаимосвязанные части был применён архитектурный шаблон Model-View-Controller(MVC, «Модель-Представление-Контроллер»). Данный шаблон используется в проектах для разделения и компонования логики и позволяет разбить структуру проекта на три взаимосвязанные части, такие как

**Модели** – получают данные от контроллера, выполняют необходимые действия и передают их на представление

**Представление** – получает данные от модели и выводит их для пользователя. Представление в структуре проекта подразумевает под собой страницы на языке HTML.

**Контроллер** – обрабатывает данные и действия пользователя, осуществляет проверку и валидацию и передаёт их модели.

Такая структура разграничения обязанностей помогает масштабировать приложение с точки зрения сложности, потому что при такой организации структуры проще реализовывать новый функционал, отлаживать и тестировать приложения. Без применения данного шаблона проект может содержать сильно связанную структуры с зависимостями из большого количества областей, что в свою очередь усложняет масштабирование и развёртывания проекта.

Графическая структура mvc показана на рисунке 9.

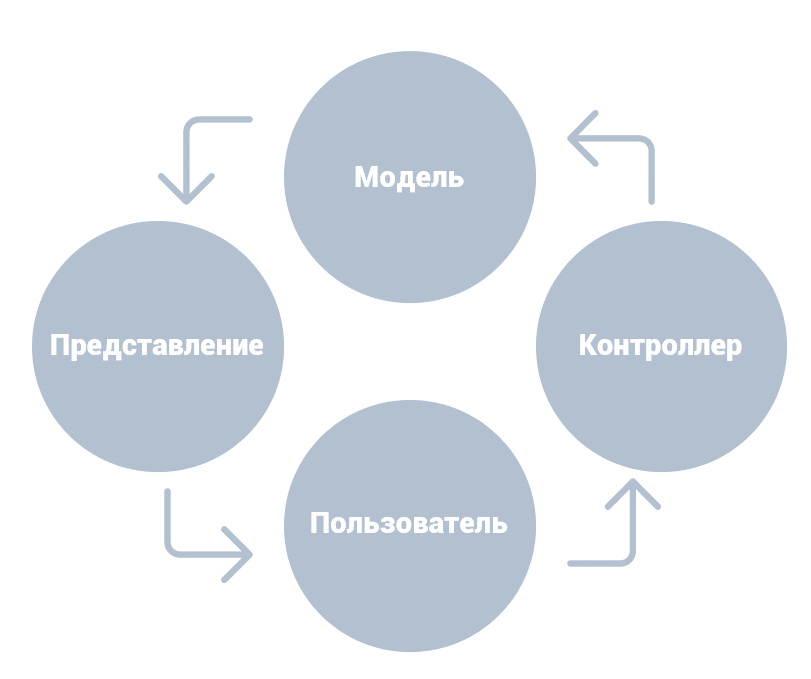


Рисунок 9 – Схема, демонстрирующая работу MVC

### Шаблон DI

Dependency injection (DI) или внедрение зависимости — это механизм, позволяющий сделать взаимодействующие объекты в структуре программы слабо связанными. Так объекты образуют связь между собой посредством интерфейсов, что позволяет сделать систему более адаптивной, гибкой и расширяемой Использование зависимости направлено на разделение проблем создания объектов и их использования, что приводит к слабосвязанным программам.

Для внедрения зависимостей в проектировании обычно используют внедрение сервисов и внутренне разделение на бизнес-логику. Основными элементами такого подхода в коде являются:

* **Конверторы** — содержат в себе методы для проецирования моделей различных типов преобразование одной структуры данных в другую
* **Интерфейсы** — содержат в себе объявление интерфейсов репозитория и сервисов для последующего их объявления в контроллере без реализации
* **Репозитории** — это классы или методы, которые содержат логику, необходимую для доступа к источникам данных.
* **Сервисы** — это классы или методы, которые содержат логику, необходимую для реализации основного функционала программы.

Организация структуры сервисов показана на рисунке 10.



Рисунок 10 – Организация структуры сервисов

* + 1. **Структура программных модулей**

Для разработки веб приложения была использована IDE Microsoft Visual Studio. Она полностью бесплатна и имеет наиболее широкий функционал работы с asp.net.

В интегрированной среде программирования проект с названием FoodSharing. Структура проекта в обозревателе решений имеет следующий вид (Рисунок 11),

где:

* **Connected Services** – сервисы необходимые для серверной публикации проекта на домен
* **Properties** – настройки и веб службы для сборки и компилирования
* **Серверные библиотеки**, такие как bootstrap, jquery, иконка сайта и различные изображения.
* **Зависимости** – все используемые пакеты, фреймворки и библиотеки, необходимые для функционирования проекта.
* **Controllers** – контроллеры, обеспечивающие связь между пользователем и системой
* **Models** – модели предоставляет данные и методы работы с ними: запросы в базу данных, проверки на корректность
* **Services** – бизнес логика веб-приложения, запросы в базу данных, методы и классы реализуемые в контроллере для работы с данными
* **Tools** – вспомогательные методы, которые многократно применяются в приложении, например, подключение к БД, создание куки, чтение файлов.
* **Views** – представления/формы написанные на языке html с использованием css, js и технологии razor.
* **.gitignore, .gitattributes** – вспомогательные файлы для публикации репозитория на GitHub
* **appsettings.json** – файл, содержащий в себе все настройки проекта, пароли и данные, необходимые для подключения к БД.
* **Program.cs** – корневой файл сборки проекта.

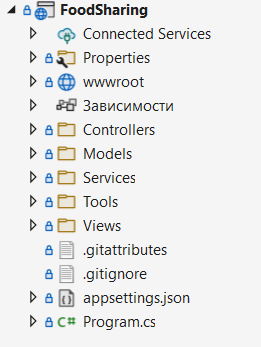


Рисунок 11 – Структура проекта.

### Аутентификация и авторизация

В первую очередь при разработке веб-приложения была осуществлена разрабокта аутентификации и авторизации пользователя на Куки (HTTP Cookie).

**Куки** - фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. Веб-клиент всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP-запроса. Файлы куки преимущественно применяются для сохранения информации пользователя к которой он в последующем будет множество раз обращаться, такие как имя, адрес, пароль, номера плаёжной карты.

Файлы куки аутентификации используются веб-сервисами для проверки того, что пользователь вошёл в систему и с какой именно учётной записи он это совершил. Без файлов куки пользователям пришлось бы аутентифицировать себя, входя в систему на каждой странице, содержащей конфиденциальную информацию, к которой они хотят получить доступ. Шифрование файлов куки осуществляет браузер и фреймворк Asp.net тем самым обечпечивая пользователю защищённость данных.

Для регистрации пользователя применяется метод RegisterUser в контроллере AuthController. В данный контроллер посредством Post запроса, вызываемого с помощью кнопки передаётся модель RegistrationViewModel, которая содержит пароль и почту. Инструкции метода осуществляют валидацию и проверку на корректность введёных данных и в случае их правильности создают Куки в которых в зашифрованном формате хранится почта пользователя и его уникальный хеш ключ(Claim). Также на данном этапе происходит добавление пользователя в бд.



Рисунок 12 – метод RegisterUser

Для осуществления аутентификации и последуещего предоставления доступа к личным данным пользователя используется метод Login. Основная задача его заключается в проверки наличия учётной записи и валидации введённых данных пользователями .

Метод Login продемонстрирован на рисунке 13

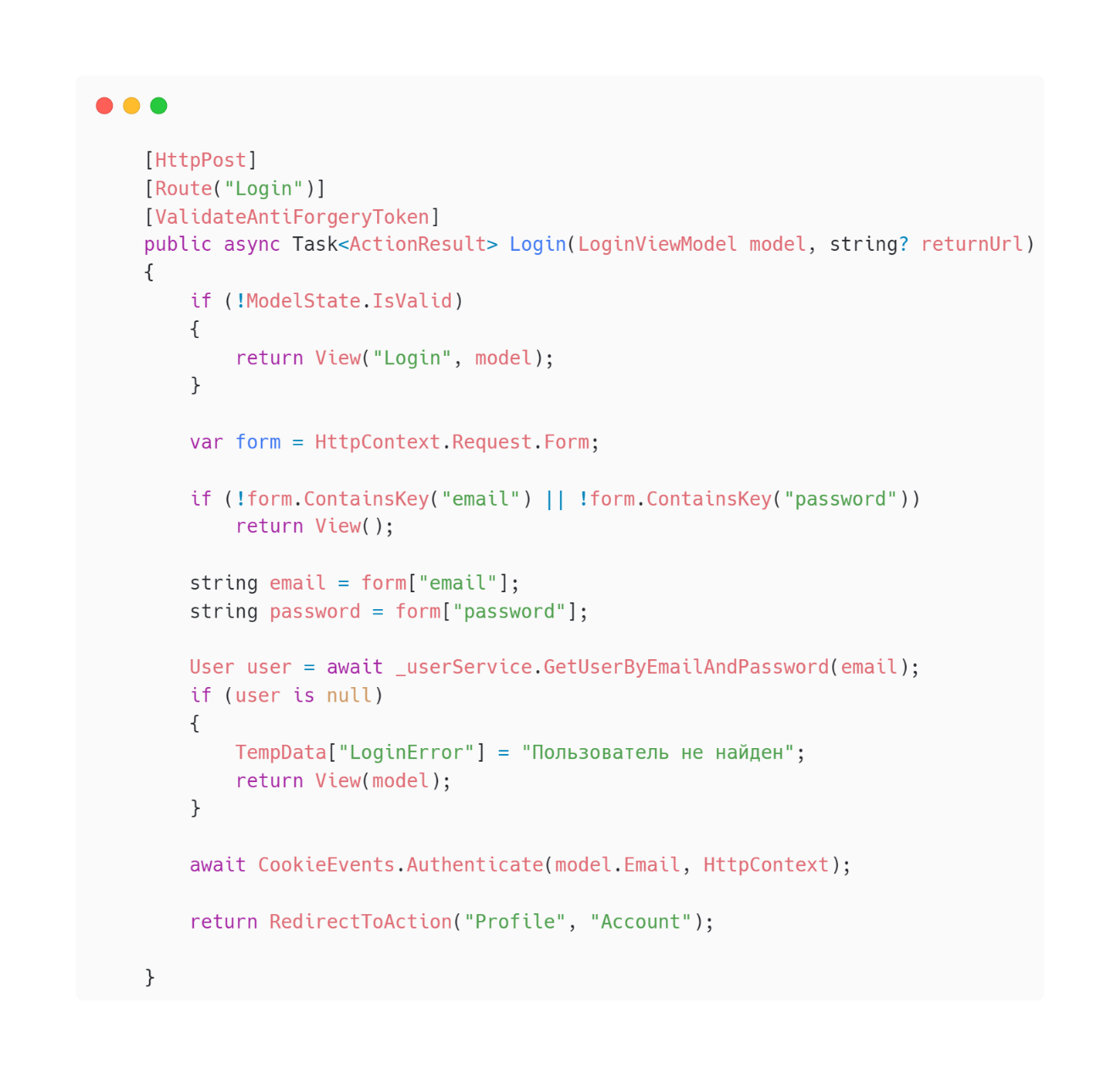


Рисунок 13 – метод Login

Метод реализующий создание куки, проверку идентификации и создание идентификационного клейма показан на рисунке 14.



Рисунок 14 – Класс CookieEvents и методы Authenticate и SignOut

При обращении к БД используется метод Add, который в свою очередь требует в качестве параметров запрос и переменные вносимые в БД. Обращение к AddUser происходит в методе RegisterUser и заключается в процессе вносения данные в БД (Рисунок 15)

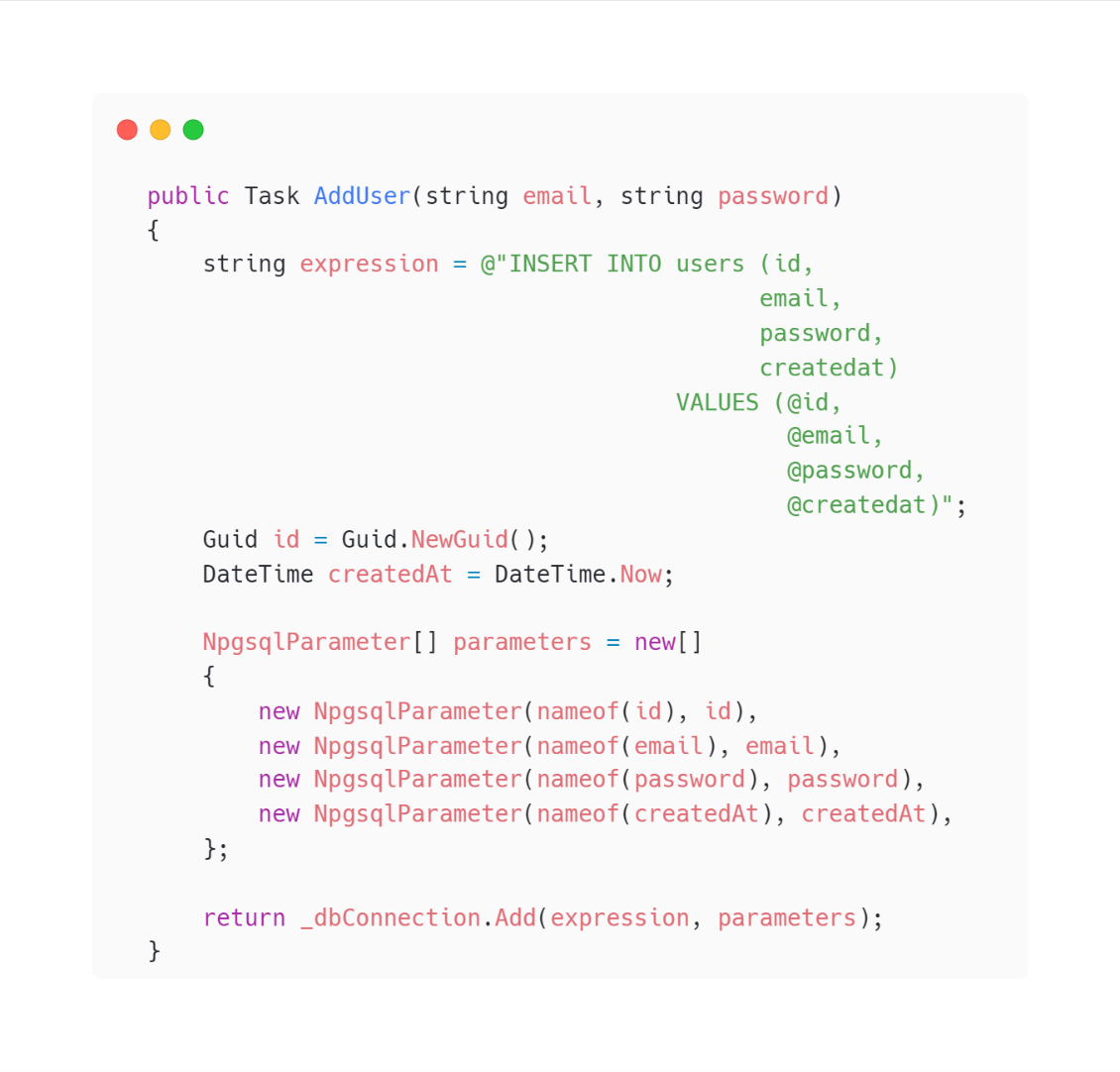


Рисунок 15 –Метод AddUser

Визуальное отображение страницы Авторизации продемонстрированно на рисунке 16. Пользователю необходимо ввести данные своей учётной записи, а также при желании он может запомнить данные в браузере более быстрого входа. Страница регистрации выглядит как показано на рисунке 17.

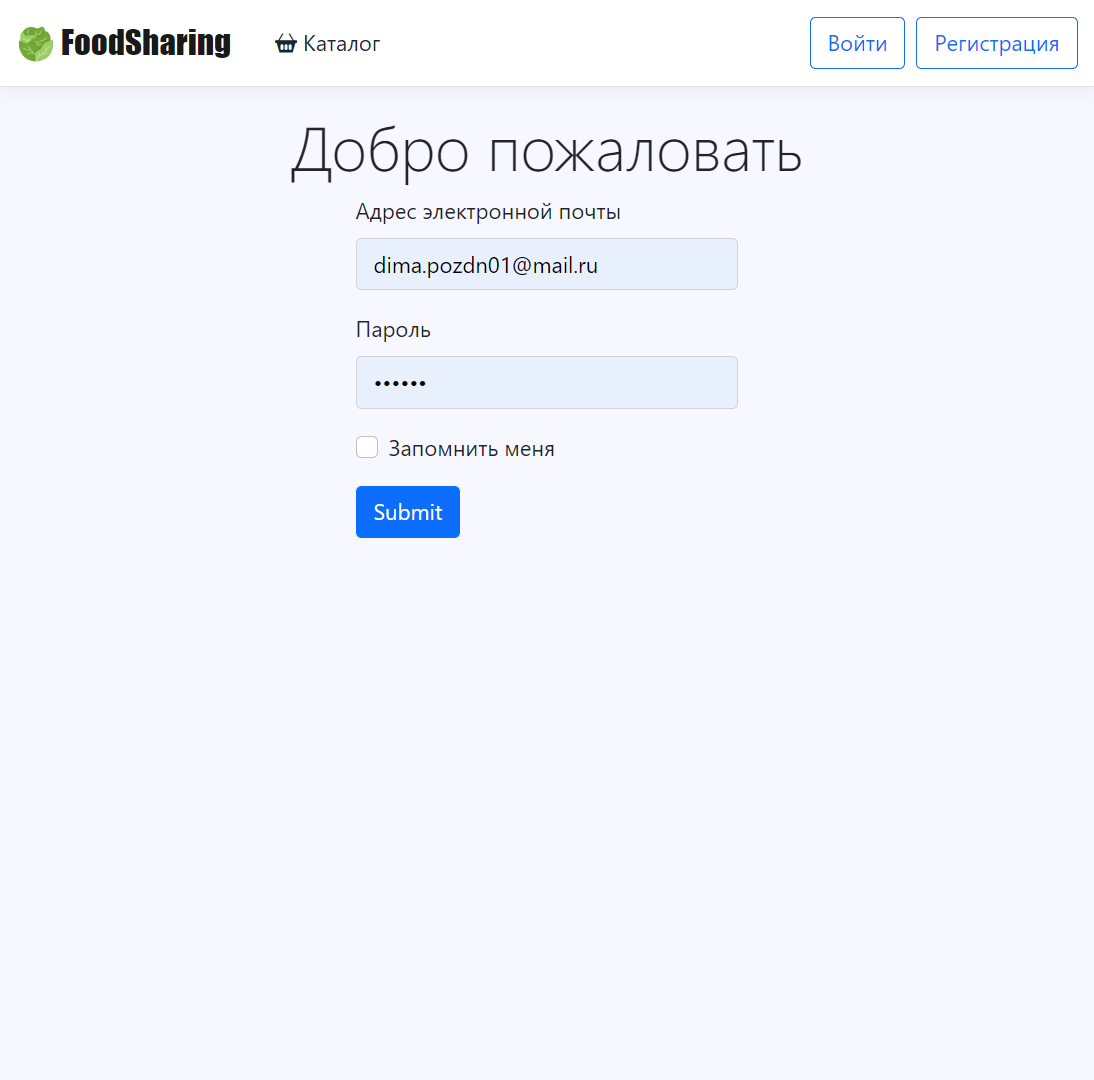


Рисунок 16 – Страница «Авторизация»

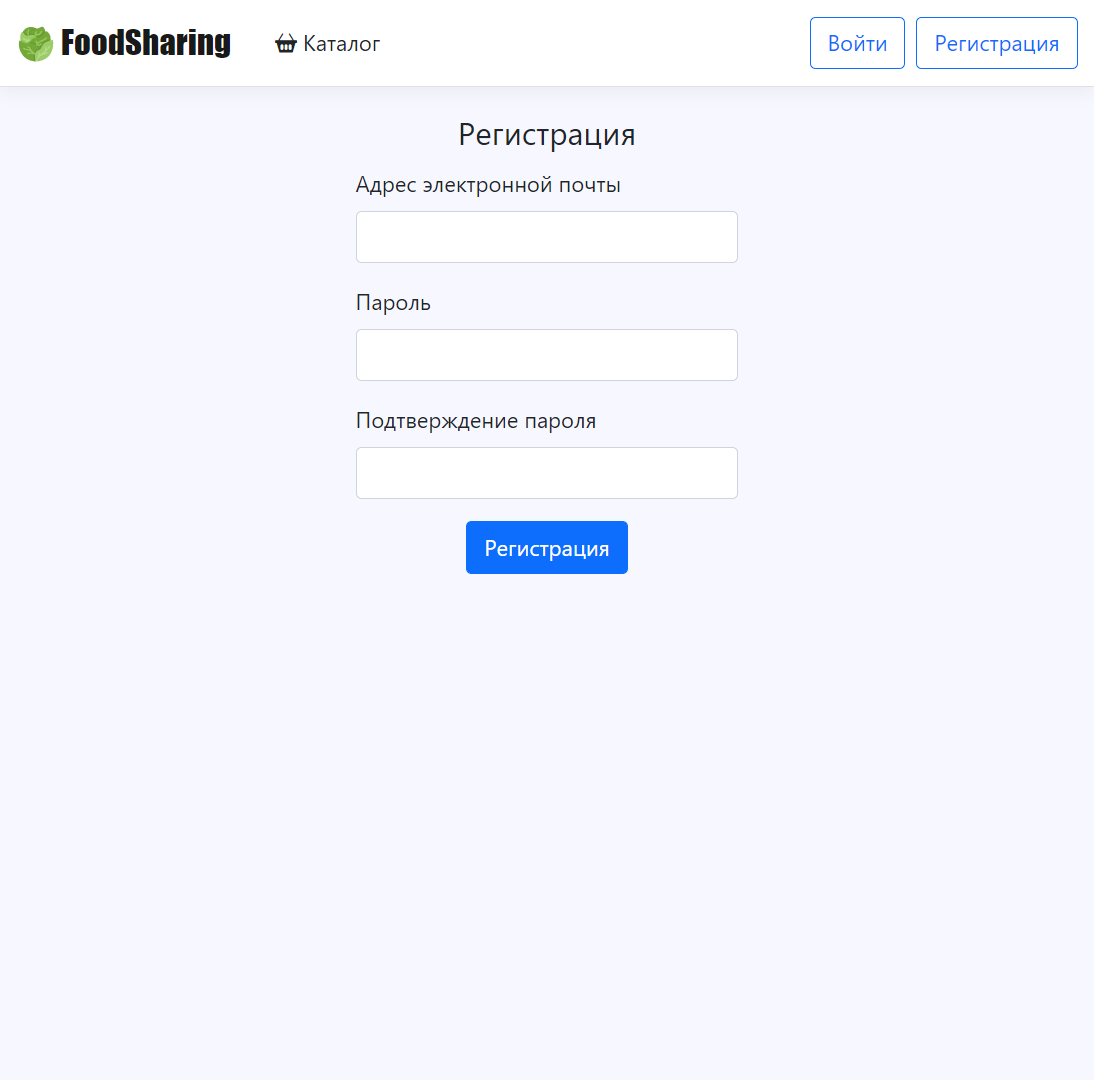


Рисунок 17 – Страница «Регистрация»

### Профиль пользователя

При успешной авторизации пользователь попадает на строницу профиля. На ней ему предоставляется возможность добавить более оширную информацию о себе, чтобы в последующем дать возможность связаться для получения продуктов. Основная программная часть реализации данного функционала предоставлена в AccountController, а именно в методах EditProfile осуществляющий редактирование и изменение данных пользовтеля (Рисунок 18)



Рисунок 18 –Метод EditProfile

Запрос осуществляющей комбинированный запрос INSERT и UPDATE осуществляющий функцию добавления профиля показан на (Рисунке 19). Метод состоит из написанного на языке SQL запроса и обращения к БД PostgreSql посредством библиотеки NpgSql. Основная задача библиотки обеспечить взаимодействие и оперирование с БД посредством кода на языке C#.



Рисунок 19 –Метод AddUserProfile

На рисунке 20 продемонстрированн внешний вид страницы пользователя на которой содержится такие данные как ффамилия, имя, почта и контактынй телефон а также имеющиеся в наличии продукты.

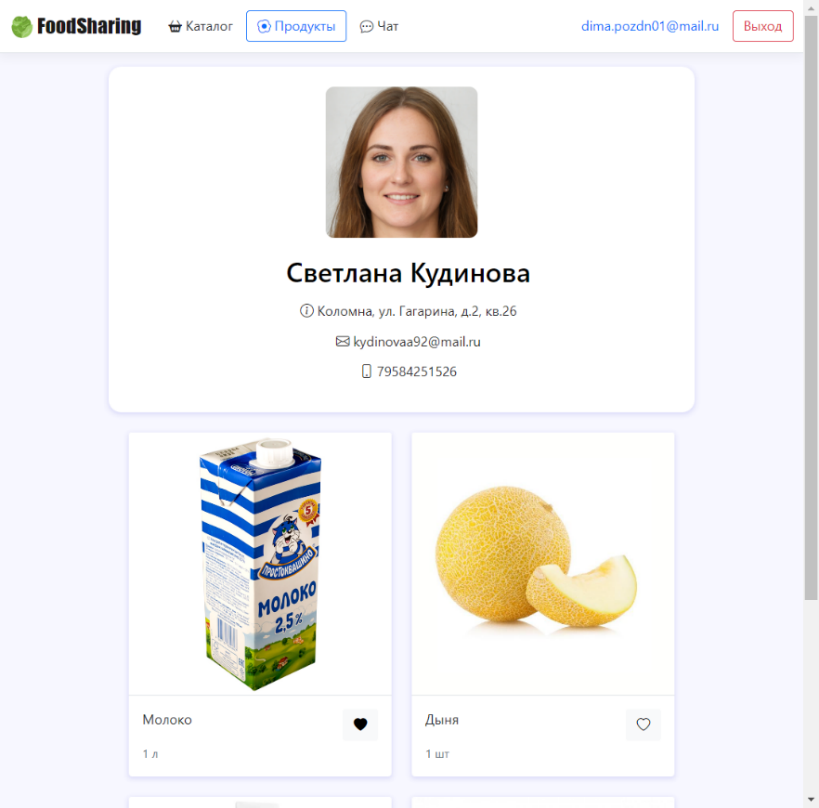


Рисунок 20 – Страница «профиль»

### Продукты и каталог

Для любого пользователя, заполнившего профиль, открывается возможность добавлять продукты для обмена а также смотреть продукты в общем каталоге.

На странице «Мои продукты» показаны выставленные самим пользователем товары и осуществлена возможность редактирования, скрытия и удаления (рисунок 22). Метод GetProduct реализует выполнение данной задачи и показан на рисунке 21.



Рисунок 21 - Метод GetProduct

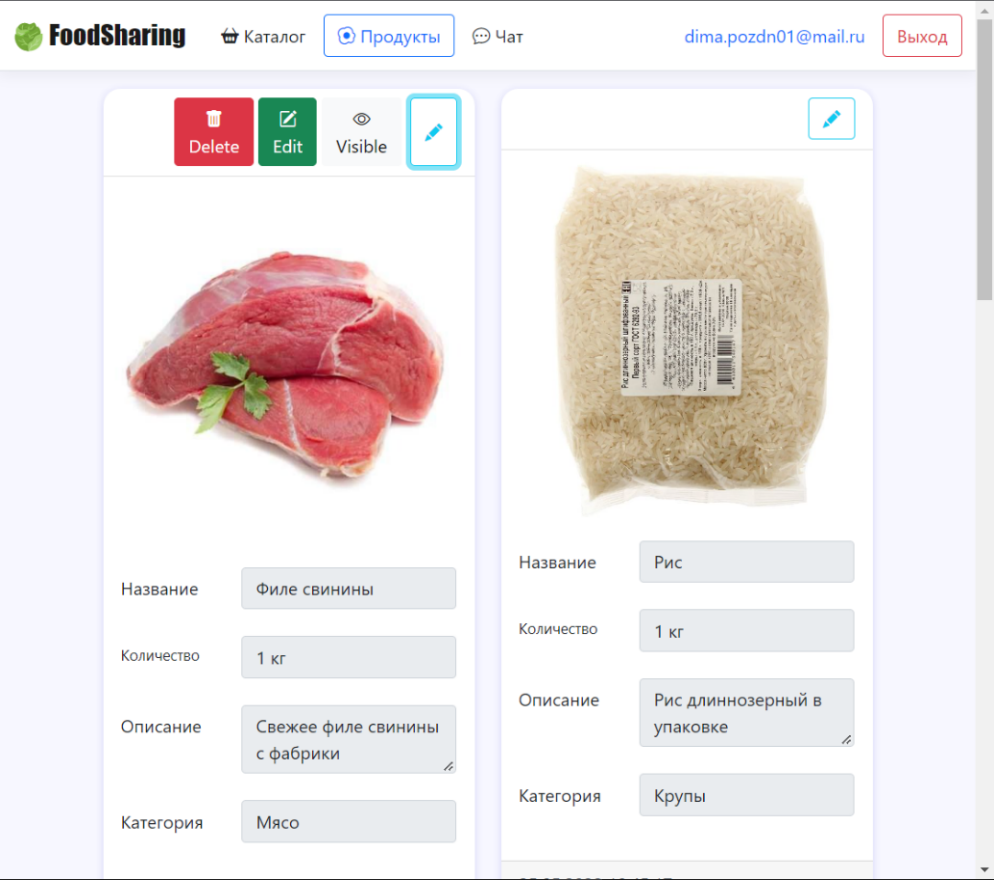


Рисунок 22 – Страница «Мои продукты»

Для добавление нового товара разработан метод NewProduct (рисунок 23). Его работа заключается добавлении нового продукта в список имеющихся у пользователя. На этом этапе осуществялется занесение модели продукта в БД. Страница добавление товаров показана на рисунке 24.



Рисунок 23 - метод NewProduct

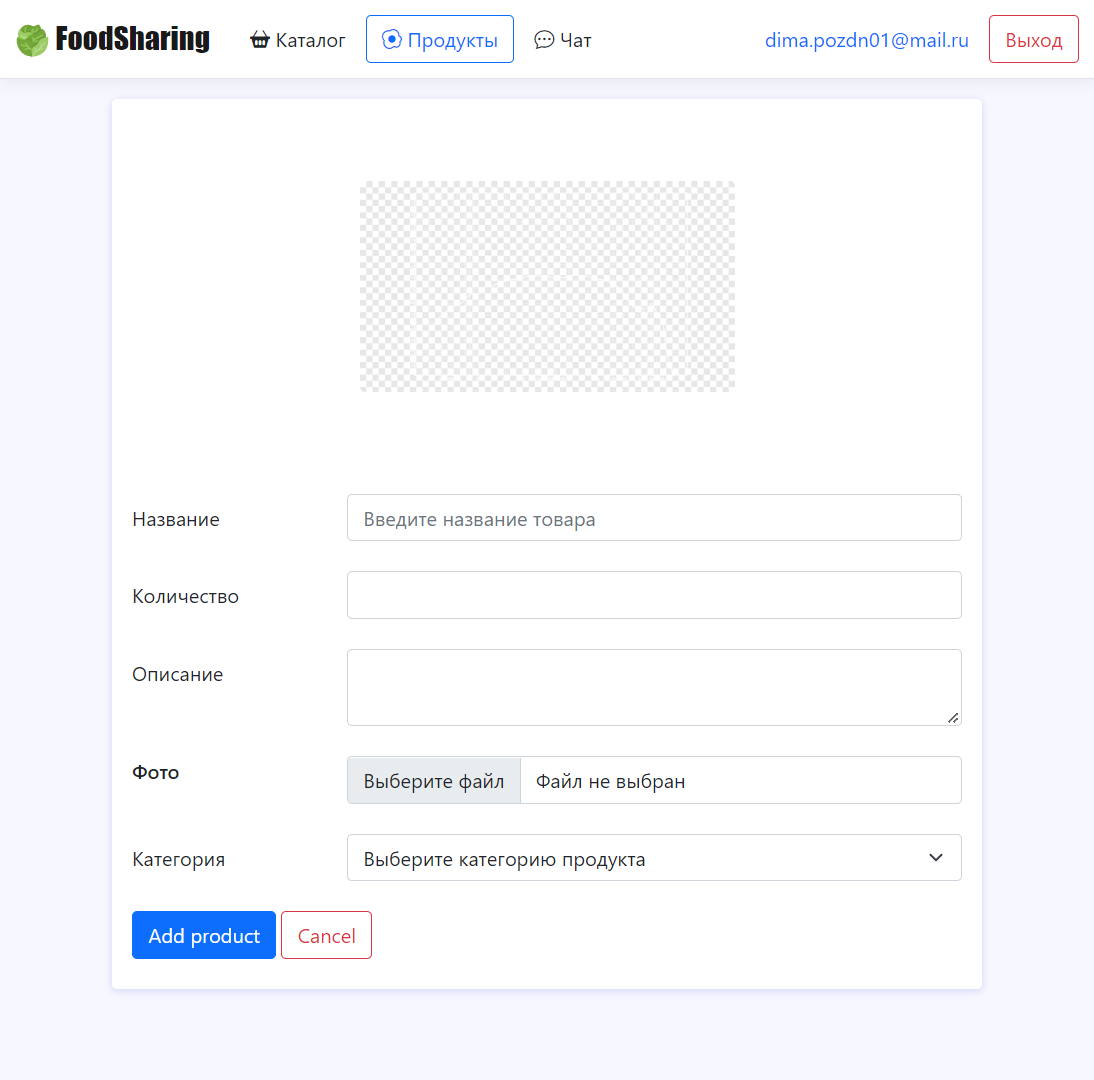


Рисунок 24 – Страница «Добавить товар»

Метод EditProduct предназначен для редактирования имеющегося товара с учётом пожеланий пользователя. Функционал подпрограммы отвечает за изменение имещейся в бд модели продукта и показан на рисунке 25.

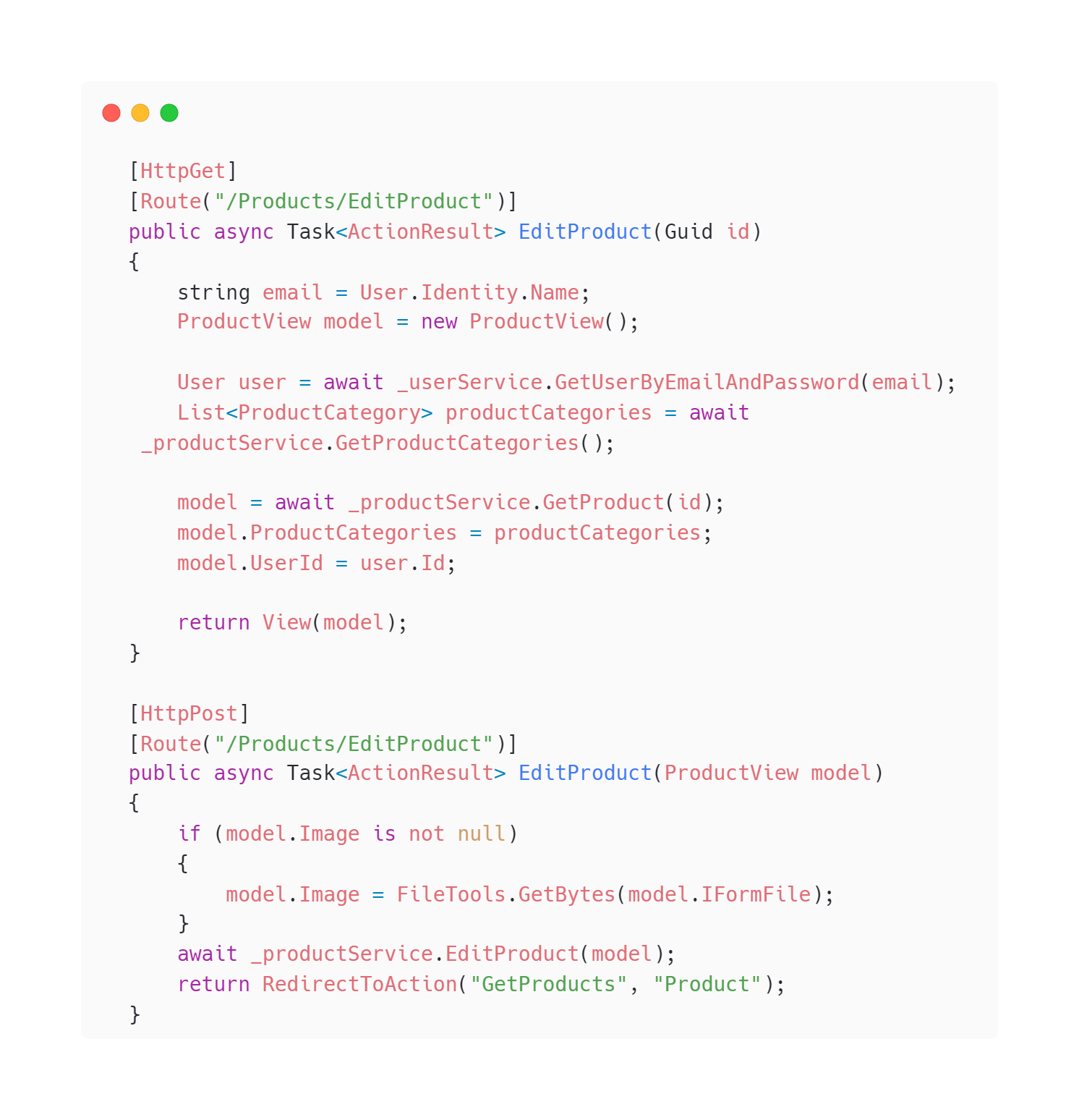


Рисунок 25 - Post и Get методы EditProduct



Рисунок 26 – Страница «Изменить товар»

Метод Catalog реализуют отображение каталога (рисунок 27) и поиск товара по категории и названию. Для реализации поиска на странице добавлена специальная панель с возможностью ввода наименования продукта и категории из списка имеющихся



Рисунок 27 - метод Catalog

На странице каталога (рисунок 28) пользователь просматривает все размещённые и свободные для обмена товары

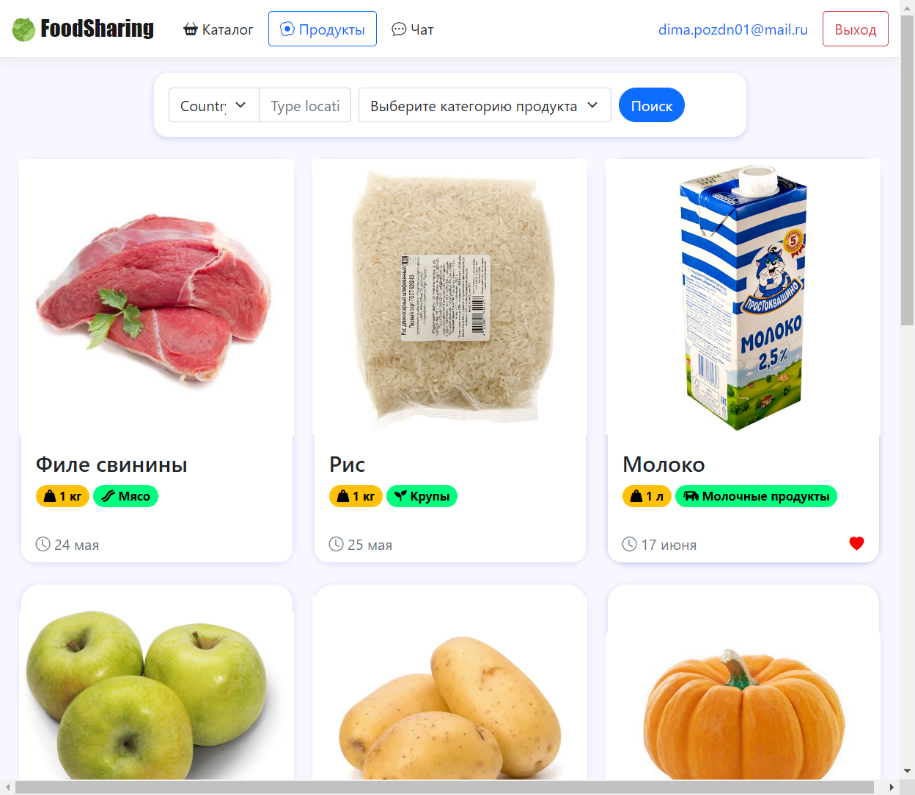


Рисунок 28 – страница «Каталог продуктов»

* + 1. **Реализация обмена сообщениями**



Рисунок 29 – логотип библиотеки SignalR

Для разработки функицонала для обмена сообщенями используется библиотека SignalR, которая предназначена для создания приложений, работающих в редиме реального времени.

SignalR использует двустороннюю связь для обмена сообщениями между клиентом и серверной системой, поэтому он может передавать необходимые данные всем клиентам в режиме реального времени.

В качестве средства связи для передачи сообщений между клиентом и сервером SignallR используется ряд механизмов:

* WebSocket – протоко связи для обомена сообщения между клиентом и сервером в режиме реального времени поверх TCP-соединения
* Server-Side Events - Данная технология позволяет отправлять сообщения с серверов к веб-браузеру в форме DOM-событий.
* Long Polling – технология, устанавливающая постоянное HTTP-соединение для отправки push уведомлений

Инфраструктура SignalR выбирает наиболее удобный механизм взаимодействия с клиентом и сервером. В частности, самым лучшим является WebSockets, поскольку клиент и сервер могут использовать этот механизм, то взаимодействие происходит через WebSockets. Если же WebSockets не поддерживается, используется Server-Side Events. Если же SSE не поддерживается и используется Long Polling, то применяется Long Polling.

Для разработки функционала чата был создан отдельный класс ChatHub. Его основная задача заключается в реализации методов для асинхронной отправки сообщений в режиме реального времени. Для отправки сообщений был реализован метод SendMessage (рисунок 30)

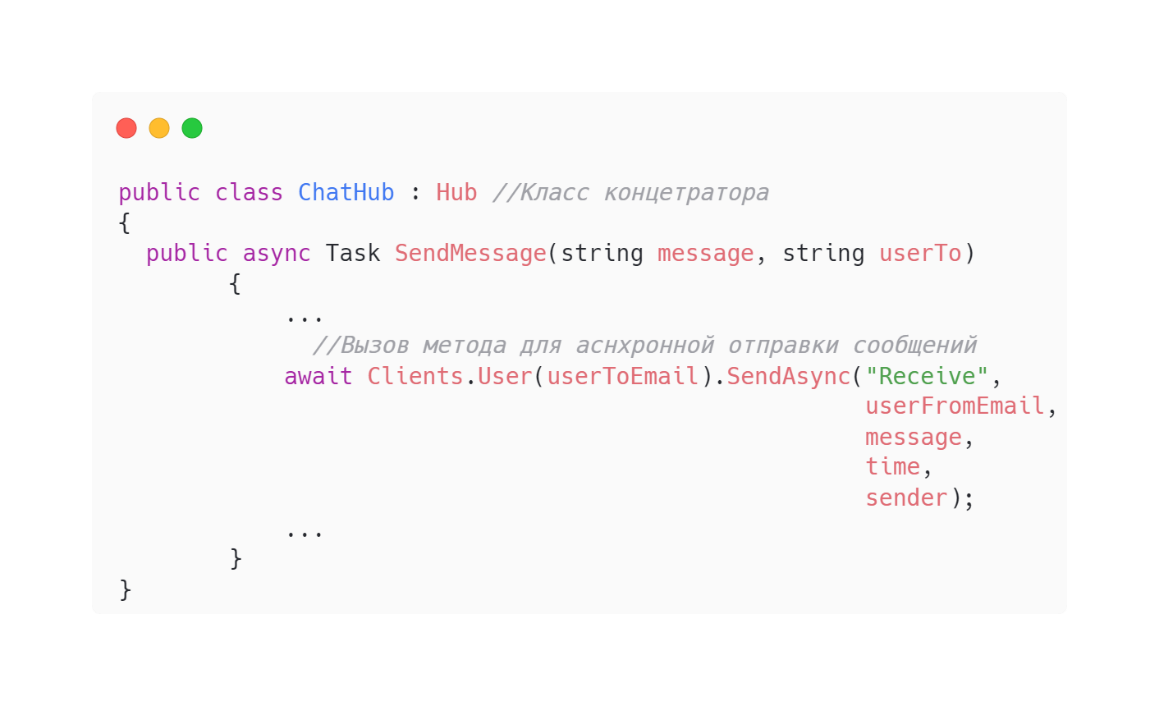


Рисунок 30 – метод для отправки сообщений SendMessage.

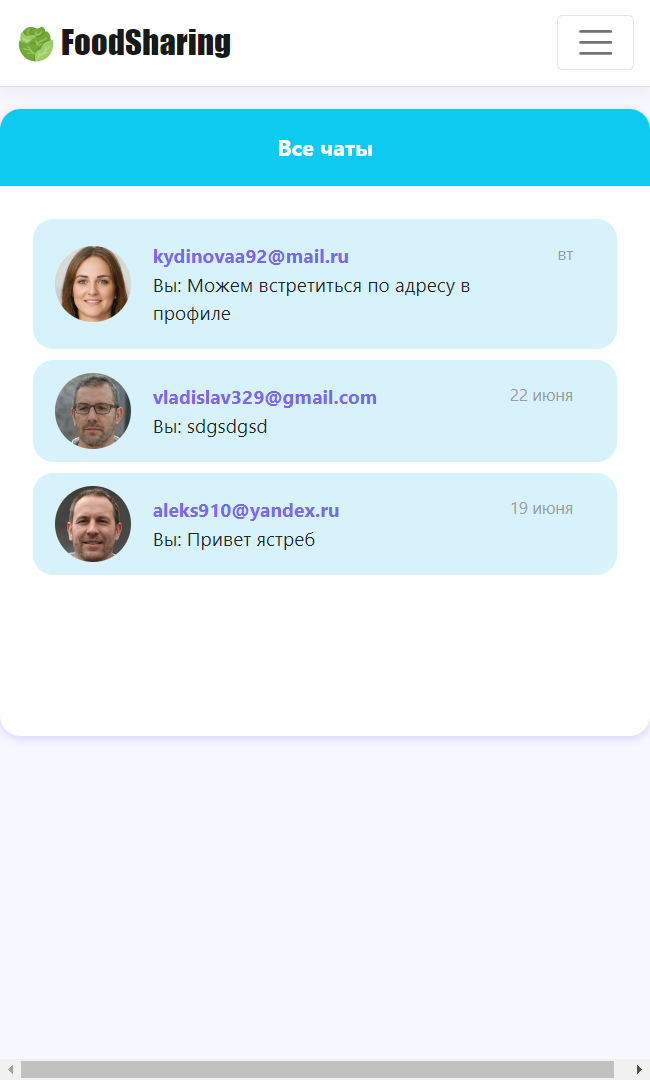
Процесс установления соединения и подключения к библиотеке SignalR выглядит следующим образом как показано рисунке 31.



Рисунок 31 – установка соединения и подключения к SignalR

Важной задачей в разработке чата было сделать максимально понятный и удобный пользовательский опыт обмена сообщениями. Для этого был сделан упор на приятный для пользователя интерфейс и обеспечение адаптивности, чтобы процесс коммуникации был независим от устройства.

На рисунке 32 изображена страница с имеющимися собеседниками, открытая со смартофона



Риисунок 32– страница «Все чаты»

На рисунке 33 продемонстрирован интерфейс чата в котором пользователи могут обмениваться сообщениями.

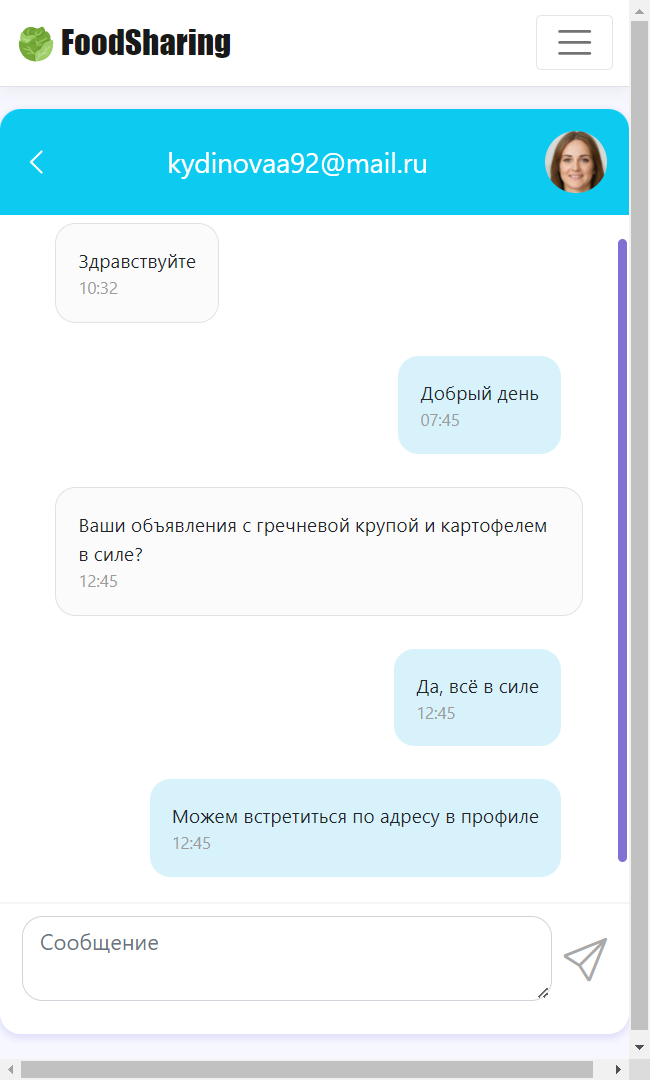


Рисунок 33 – Страница «Чат»

### Тестирование ПС

В данном разделе необходимо тестирование программного обеспечения так как любой программный продукт необходимо проверить и произвести отладку переде внедрением и началом эксплуатации. Для проверки кореектного функционирования важно проверить правильность работы и отображения необходимой информации, а также проследить, чтобы приложение работало на устрйоствах, удовлетворяющих минимальным системным требованиям.

Первым делом протестируем первоночальные окна веб приложения а именно страницу авторизации и регистрации. Для это опишем следующие варинанты действий пользователя.

1. Пользователь вводит некорреткный адрес почты

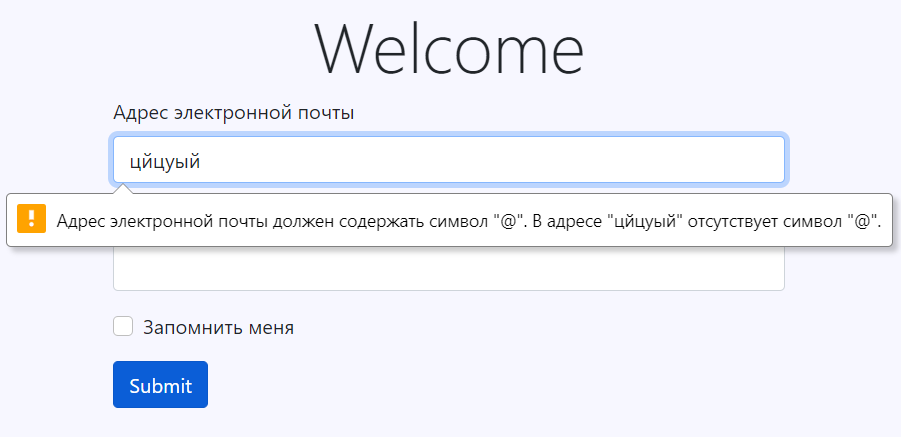


Рисунок 34– Уведомление о некорректности введённых данных

1. Пользователь нажимает на кнопку, оставив поля пустыми

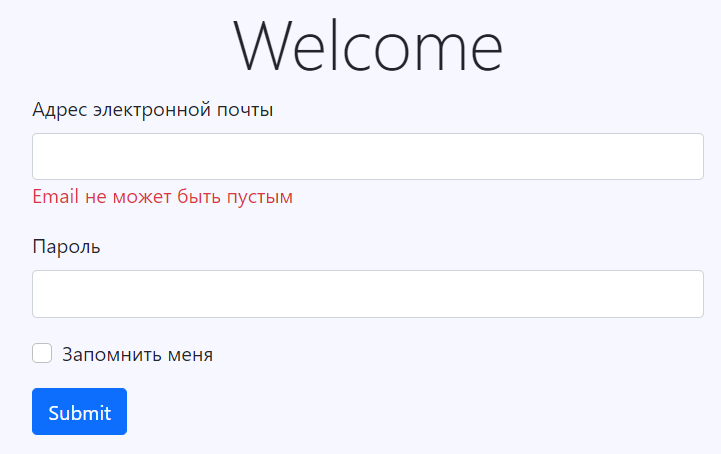


Рисунок 35– уведомление о пустом поле почты

1. Пользователь верно вводит почту, но ошибается в написании пароля

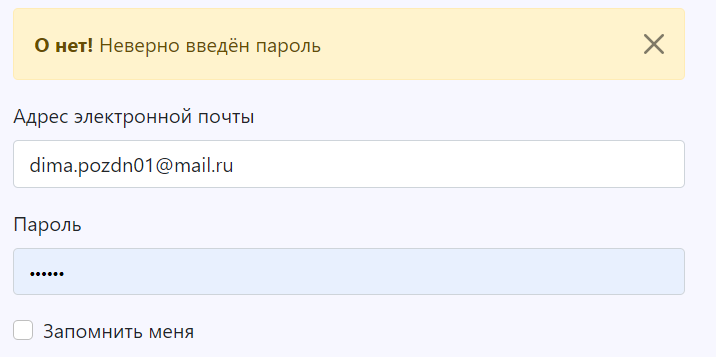


Рисунок 36 – Уведомление о неправильно введёном пароле

1. Пользователь корректно вводит данные, но не имеет учетной записи

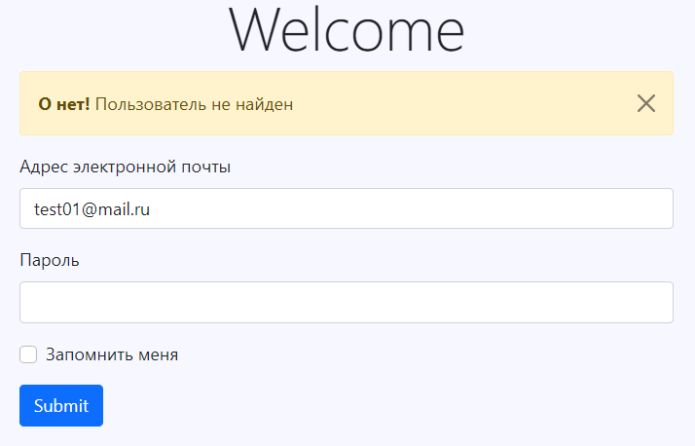


Рисунок 37– Уведомлении об отсутсвии учётной записи

1. Пользователь некорректно вводит данные при регистрации

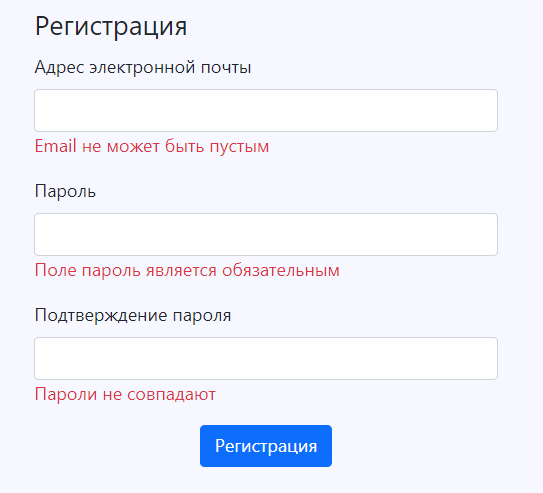


Рисунок 38– Общее количество уведомлений при регистрации

Исходя из полученных результатов сделаем вывод что страницы авторизации и регистрации работают корректно. Валидация данных позволяет обеспечить правильную обратку информации и ошибок. Далее при успешной авторизации попробуем добавить продукт оставив поля пустыми. Результа выполнения показан на рисунке 39

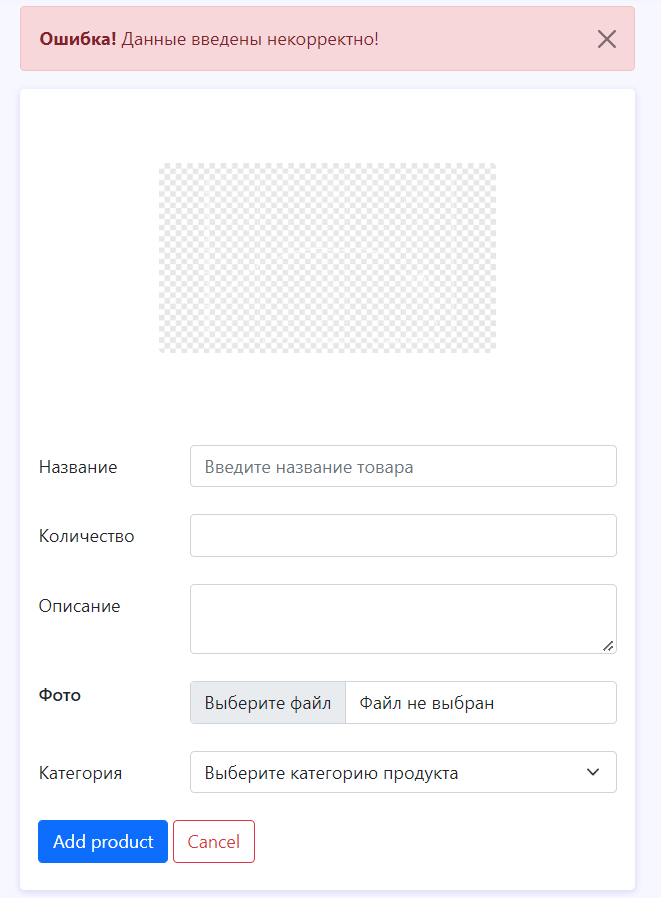


Рисунок 39– Уведомление о некорректности данных при добавлении товара

В процессе тестирования основных разделов программы было установлено, что программа работает корректно и не дает пользователям вводить некорректные данные, которые смогли бы нарушить процесс работы веб-приложения

1. **ЭКОЛОГИЯ И БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ**
   1. **Введение**

В данной выпускной квалификационной работе разрабатывалось автоматизированная информационная система для перераспределения продуктов питания.

Программное обеспечение разрабатывалось в виде веб-приложения и своей функциональностью не несет никакой опасности для человека и окружающей среды. Программный продукт разрабатывается для использования на электронно-вычислительной машине (ЭВМ), при взаимодействии и работе, с которой могут возникать различные вредоносные и опасные факторы. Поэтому были приняты меры по охране здоровья и безопасности, для выявления рисков и их предотвращения и избегания.

Эти меры включены в законодательство, поэтому работодатели и сотрудники должны соблюдать их, чтобы обеспечить ответственное использование компьютеров. Законодательство охватывает ряд факторов, связанных с использованием компьютера, которые составляют основу оценки риска рабочей станции.

Исходя из этого, наука БЖД обязана рассматривать влияние этих факторов на человека. Только при таком системном подходе можно в комплексе реализовать конечную цель БЖД.

## Безопасность при работе с ЭВМ

Вот некоторые из рекомендаций HSE по снижению риска для сотрудников в результате использования DSE:

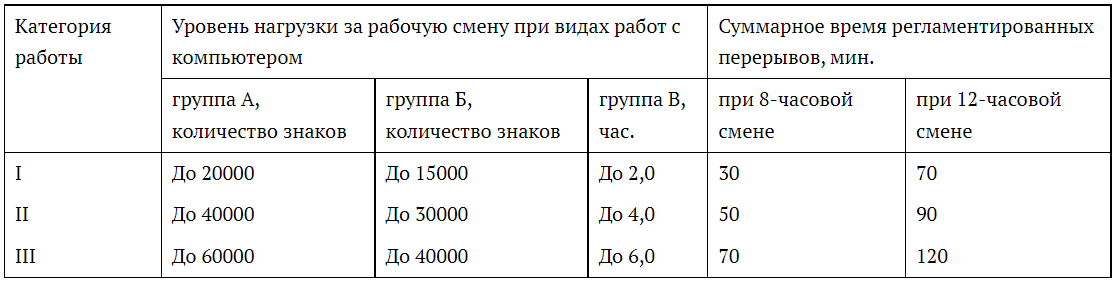
При работе с персональным компьютером следует придерживаться таким рекомендациям как:

* Необходимо избегать бликов или ярких отражений на экране. Также следует убедиться в отсутствии яркого света из оконных проёмов
* Необходимо использовать шторы или жалюзи для того, чтобы заблокировать назойливый свет.
* Под столом должно быть место, чтобы сотрудники могли передвигать ноги и чувствовать себя свободно
* Перед клавиатурой должно быть оставлено место для рук и запястий, когда они не печатают.
* Пользователь должен сидеть прямо и близко к столу, чтобы рука, использующая мышь, не была вытянута.
* Символы на экране должны быть четкими и в фокусе, не должны мерцать или двигаться: в противном случае может потребоваться регулировка или ремонт монитора.
* Яркость и контрастность должны быть отрегулированы в соответствии с окружающей средой.
* Текст должен быть достаточно крупным, чтобы его можно было читать в обычном удобном рабочем положении, и следует избегать цветовых схем, которые бросаются в глаза.

## Рабочий распорядок и перерывы

Согласно «Типовой инструкции по охране труда при работе на персональном компьютере ТОИ Р-45-084-01» каждый сотрудник для продуктивного взаимодействия с ЭВМ должен совершать перерывы для отдыха в соответствии с тяжестью работы, которую он выполняет (таблица 6).

Так при высокой нагрузке за компьютером и в соответствии с нормой восьмичасового графика работы работнику необходимо предоставлять перерывы общей продолжительностью 15 минут каждый час. Также для полноценного восстановления и питания следую обеспечить обеденное время длительностью 45 – 60 минут в середине рабочей смены.

Таблица 6– Суммарное время регламентированных перерывов при работе с ЭВМ. 

* 1. **Расчёт освещённости помещений**

В предприятии ООО «ВиртумЛаб» офис имеет следующие размеры

Таблица 7 – Переменные для вычисления освещённости

|  |  |
| --- | --- |
| Параметр | Величина, м |
| Ширина | 5 |
| Длина | 4 |
| Площадь | 12 |
| Расстояние между светильником и горизонтальной поверхностью | 4 |
| Число светильников в помещении | 5 |

Офис представляет из себя просторное помещение с панорамными окнами со встроенными жалюзи. Освещение ведётся за счёт люминесцентных ламп, которые располагаются преимущественно на потолке белого цвета.

Согласно справочной книге для проектирования электрического освещения [9] определим следующие данные

Таблица 8 – справочные данные для вычисления

|  |  |
| --- | --- |
| Коэффициент | Величина |
| Минимально допустимая освещенность | 300 Лк |

Продолжение таблицы 8

|  |  |
| --- | --- |
| Коэффициент неравномерности освещения для люминесцентных ламп | 1,5 |
| Коэффициент запыления светильников | 0,9 |

Для расчета индекса используется формула 1:

|  |  |
| --- | --- |
|  | (1) |

где

S — площадь помещения в м2

A и B — длина и ширина помещения

h — расстояние между светильником и горизонтальной поверхностью

Отсюда коэффициент помещения i будет равен:

При заданных условиях задачи и согласно справочным данным [9]:

— Коэффициент отражения потолка – 80%;

— Коэффициент отражения стен – 60%;

— Коэффициент отражения пола – 25%;

Произведём расчёт освещения помещения по формуле 2

|  |  |
| --- | --- |
|  | (2) |

где

Е - заданная минимальная освещенность

r - коэффициент запаса;

S - освещаемая площадь;

z – характеристика неравномерного освещения;

N – первоначальное количество светильников;

h - коэффициент использования.

Отсюда получим

* 1. **Общие положения о пожарной безопасности**

Работодателю необходимо регулярно проводить анализирование рисков возникновения экстренных ситуаций, связанных с пожаром. Оценка риска должна включать в себя такие категории процессов как взрывоопасные, пожароопасные.

В следствии проверки должны быть установлены наиболее значимые риски возникновения очага пожара такие как конструктивные недостатки здания и помещений, дефекты и износ электрооборудования, несоблюдение правил пожарной безопасности сотрудниками. При выявлении таких мер работодатель должен позаботиться о их скорейшем устранении во избежание рисков получения травм и гибели людей в результате пожара.

В целях предотвращения пожара обязательной мерой является проведение профилактических и организационных мероприятий, направленных на такие аспекты как:

* информирование сотрудников о базовых требованиях пожарной безопасности,
* об индивидуальных свойствах отдельных устройств и правилах его эксплуатации
* о способах и правилах использования противопожарных средств
* Проведения инструктажа о последовательности действии по время пожара, а также способах эвакуации
* Проведение уроков об оказании первой медицинской помощи

Для предотвращения пожаров на предприятии ООО «ВиртумЛаб» проводятся противопожарные инструктажи, на которых сотрудников знакомят с правилами пожарной безопасности, а также обучают обращению с огнетушителями.

* 1. **Правила эвакуации при пожаре**

Эвакуация при пожаре – это комплекс мер направленных на мобильное и наиболее быстрое перемещение людей с территории очага пожара в зоны, где отсутствует риск здоровью жизни человека.

На любом предприятии обязательна разработка плана эвакуации с подробным описанием маршрутов перемещения к выходу из здания, а также с подробным описанием действий персонала при возникновении аварии и пожара. В обязательном порядке план должен содержать такие обозначения как:

* Направление движения к эвакуационному выходу
* Расположение запасного выхода в случае отсутствия возможности проложить маршрут к основному
* Расположение аптечки первой медицинской помощи
* Расположение кнопки экстренного оповещения
* Расположение ручного пожарного извещателя
* Положения огнетушителей и пожарных кранов
* Расположение электрощита

Административное здание панельного типа имеет железобетонные стены и фасад, оборудовано автоматизированной системой сигнализации и пожаротушения. Вблизи каждой лестничной клетки расположен огнетушители и средства экстренного оповещения. Каждый коридор имеет ширину 3 метра и обеспечивает мобильность большого количества людей в экстренной ситуации и последующей эвакуации.

План эвакуации с офиса на базе которого разрабатывался проект расположен вблизи выхода из здания. Маршрут эвакуации выглядит следующим образом:

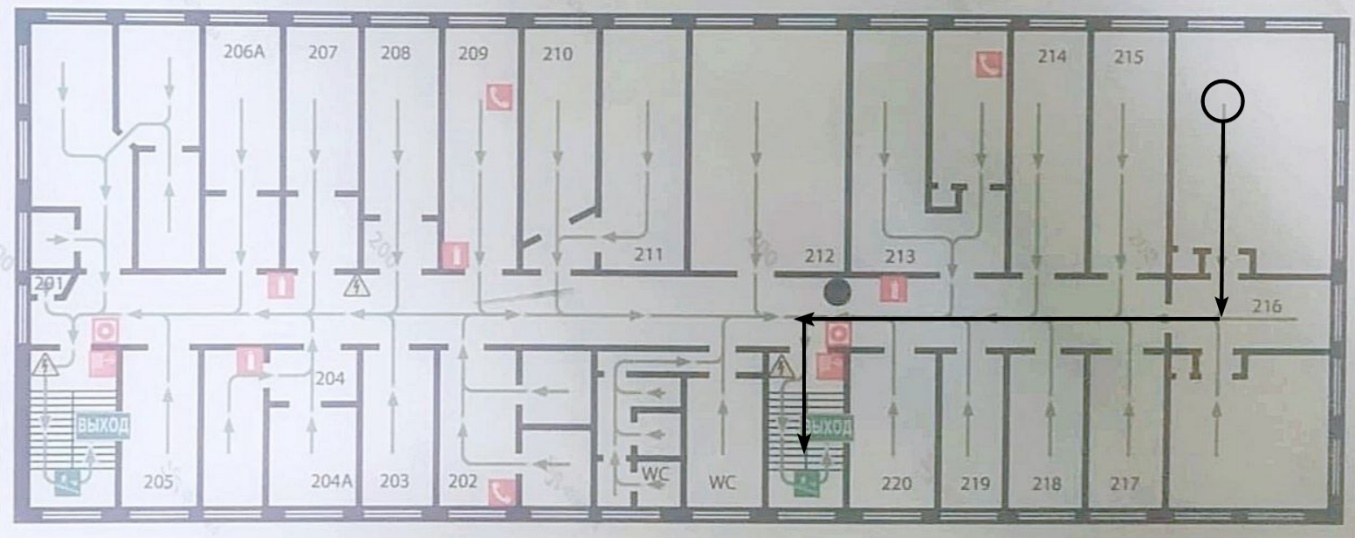


Рисунок 40 – Схема эвакуации сотрудников предприятия

1. **ЭКОНОМИЧЕСКИЙ РАЗДЕЛ**
   1. **Введение**

Стратегия экономического развития – это комплекс мер, направленных на повышение эффективности производства, рациональное распределение материальных средств для разработки, а также стремление к минимизации человеческих ресурсов и времени. Исходя их этого главным приоритетом работы конструкторов является умение выполнять задачи при наименьших затратах средств и времени, учитывать наилучшие мировые тенденции в разработке и выборе программного обеспечения. Поэтому экономическая составляющая предоставляет возможность понять целесообразность проекта, его актуальность и экономическую выгоду для разработки.

Проектируемый продукт выпускной квалификационной работы представляет собой автоматизированную информационную систему, предназначенную для перераспределения продуктов питания. Веб-приложение представляет собой многостраничный сайт, который запускается в браузере.

Целью проекта является предоставление возможности пользователям различных социальных уровней обмениваться продовольственными товарами и сократить расходы на питание. Разрабатываемое веб-приложение помимо своей основной цели позволяет сократить объём пищевых отходов и ежегодный рост свалок, а также привить пользователям рациональное использование природных и трудовых ресурсов.

* 1. **Расчет трудоемкости разработки ПО**

Особое внимание при проектировании нового программного продукта уделяется минимизации материальных и трудовых ресурсов.

Основным методом для реализации этого служит разбиение всего проектируемого процесса на несколько последовательно взаимосвязанных этапов, каждый из которых отвечает за отдельные производственные процессы и служит обособленной частью проектирования. Каждый этап в свою очередь может включать отдельные шаги выполнение которых будет отражать переход к следующему этапу разработки. После завершения одного этапа начинается новые, которые в свою очередь обладают новой спецификой и влекут за собой увеличение затрат на разработку продукта.

Для создания различных проектов выбирается индивидуальный набор этапов, отражающий особенности и требования к поставленной цели разработки. Так для разработки веб-приложения весь процесс проектирования можно разделить на стадии, установленные ГОСТом 19.102-77 «Стадии разработки», который разделяется на следующие этапы:

1. Анализ требований

Производится оценка особенностей производственного процесса и ставятся цели и задачи проекта, которым он должен соответствовать и выполнять. Анализ позволяет определить ориентированные сроки разработки и стоимость.

1. Проектирование

Выбирается платформа на который будет реализован проект. Определяется система языков на которых будет написан основной функционал, а также осуществляется оценка системных требований для обеспечения необходимой работоспособности решения.

1. Реализация ПО

Происходит написание программной составляющей разрабатываемого обеспечения, структуризация кода. По окончанию данного этапа имеется рабочий продукт, включающий в себя все требования поставленные в предшествующих этапов

1. Тестирование и отладка

Устраняются ошибки и недочёты, полученные в результате разработки. Происходит анализ и переработка кода.

1. Внедрение и поддержка

На данном этапе производится настройка системы. Обеспечивается автономное функционирование системы для последующего использования и эксплуатации. Разработка документации для использования приложения.

Трудоемкость разработки ПО определяется по следующей формуле 3:

|  |  |
| --- | --- |
|  | (3) |

где

Т – общие затраты труда,

То – затраты труда на описание задачи,

Ти – затраты на исследование предметной области,

Тп – затраты на программирование,

Тотл – затраты на отладку программы,

Тд – затраты на подготовку документации.

Общее потраченное время на каждом этапе рассчитывается в человеко-часах для единой унификации вычислений и последующем более простом вычислении стоимости и заработной платы.

Переменная берется по фактическому времени, которое было потрачено на описание задачи, а время на последующих этапах вычисляется путем умножения на условное число команд .

Так , условное число команд рассчитывается по следующей формуле 4

|  |  |
| --- | --- |
|  | (4) |

где

q – коэффициент, который строго нормирован и обозначающий число условных команд для реализации задачи в проекте. Значение q берётся из таблицы 9

c - коэффициент, определяющийся с учётом сложности и новизны используемого языка программирования и используемых в проекте программных средств и технологий. Значение с берётся из таблицы 10

Таблица 9 - Значения коэффициентов условного количества команд

|  |  |
| --- | --- |
| Тип задачи | Пределы изменений коэффициента |
| Задачи учёта | от 1400 до 1500 |
| Задачи оперативного управления | от 1400 до 1500 |
| Задачи оперативного управления | от 3000 до 3500 |
| Многовариантные задачи | от 4500 до 5000 |
| Комплексные задачи | от 5000 до 5500 |

Таблица 10 - Значения коээфициентов, учитывающих сложность и новизну ПО

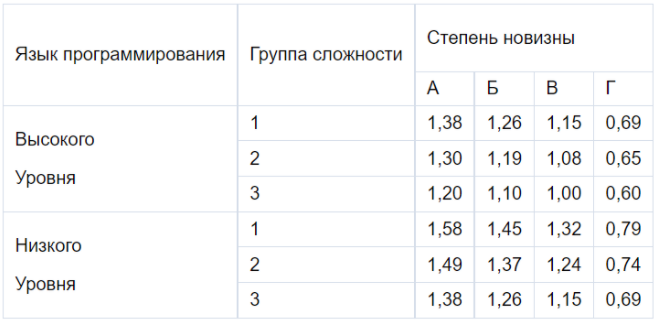


Таблица 11 - Значения коээфициентов уровня новизны

|  |  |
| --- | --- |
| Группа | Уровень новизны |
| А | Разработка принципиально новых задач |
| Б | Разработка оригинальных программ |
| В | Разработка оригинальных программ |
| Г | Разработка оригинальных программ |

Таблица 12 - Значения коээфициентов степени сложности

|  |  |
| --- | --- |
| № | Степень сложности |
| 1 | Алгоритмы оптимизации и моделирования систем |
| 2 | Задачи учета, отчетности и статистики |
| 3 | Задачи учета, отчетности и статистики |

Таблица 13 - Значения коээфициентов, учитывающих стаж программиста

|  |  |
| --- | --- |
| Стаж программиста | Значение коэффициента |
| до 2-х лет | 0,8 |
| от 2 до 3 лет | 1,0 |
| от 3 до 5 лет | 1,1 — 1,2 |
| от 5 до 10 лет | 1,2 — 1,3 |
| от 5 до 10 лет | 1,2 — 1,3 |

Затраты труда на каждом этапе рассчитываются по формуле 5

|  |  |
| --- | --- |
|  | (5) |

где

В – коэффициент увеличения затрат труда в следствие недостаточного описания задачи (средний диапазон 1,2 – 1,5). В данной формуле данный параметр применяется на этапе исследования предметной области из-за недостаточной информационной базы и необходимости изучения большого количества информации.

Q - условное число команд в зависимости от решаемой задачи и её сложности

K - коэффициент, учитывающий квалификацию программиста. Определяется по таблице 13

S – количество операторов, приходящихся на 1 чел.час. Оператор – предписание о выполнении конкретных действий. В зависимости от решаемой задачи варьируется в диапазоне от 4 до 70.Данный параметр уменьшается на каждом этапе проектирования в следствие повышения сложности решаемых задач.

**1 этап – составление тз**

Время на подготовку описания задачи берётся по фактическому времени и не подвергается расчётам. Затраты на разработку и согласование *ТЗ* составляют 24 часа.

**2 этап – исследование предметной области**

Затраты времени на анализ предметной области, изучение и подбор специализированной литературы, а также выбор инструментария разработки рассчитывается по будут равны:

**3 этап – реализация по**

Реализации программного обеспечения включает разработку функционала и выполнение основных поставленных задач по проектированию.

Исходя из выбранных коэффициентов затраты на данном этапе составят:

**4 этап – Тестирование и отладка**

Тестирование и отладка включают в себя процесс устранение ошибок, отладку и тестирование функционала и составляют

**5 этап - затраты на подготовку документации**

На данном этапе происходит финальные шаги проектирования программного обеспечения, а именно написание рабочей и проектной документации, оформление руководства пользователя и графической части приложения. Также на данном шаге происходит написание оформление пояснительной записки. Время затраченной на данный этап составит:

Рассчитав значения всех затрат на каждом этапе, можно вычислить общее время разработки по формуле 1. Оно составит:

* 1. **Составление сметы затрат на разработку**

Важным этапом проектирования являтся планирование себестоимости разработки. Он представляет процесс по расчёту и анализу затрат на производство а также обоснование и выявление мест по их уменьшению или увелечению для увеличения максимальной прибыли и эффективность разработки.

Согласно экономическому содержанию проекта затраты объединяются в следующие группы:

* Затраты на оборудование и программное обеспечение
* Материальные затраты на расходные материалы
* Затраты на электроэнергию
* Затраты на амортизационные отчисления
* Затраты на оплату труда
* Отчисления на социальные нужды

### Расчёт затрат на оборудование и ПО

Данный раздел содержит расчёт стоимости затрат на техничсекое оборудование инструментальные средства для разработки и операционную систему.

Таблица 14 – общая стоимость оборудования

|  |  |  |  |
| --- | --- | --- | --- |
| **Наименование оборудования** | **Закупочная цена (руб.)** | **Количество (шт.)** | **Стоимость (руб.)** |
| Персональный компьютер | 45000 | 1 | 45000 |
| Монитор | 11000 | 2 | 22000 |
| Мышь | 1000 | 1 | 1000 |
| Клавиатура | 1500 | 1 | 1500 |
| **Итого** | | | **69500** |

Для дальнейшей разработки проекта следует рассчитать стоимость программного обеспечения и средств для разработки. Данные по расчету представлены в таблице 16.

Таблица 16 – стоимость программного обеспечения

|  |  |  |  |
| --- | --- | --- | --- |
| **Наименование программного обеспечения** | **Закупочная цена (руб.)** | **Количество (шт.)** | **Стоимость (руб.)** |
| Windows 10 Professional | 13290 | 1 | 13290 |
| Visual Studio Professional | 27000 | 1 | 27000 |
| Visual Studio Code | в свободном доступе | 1 | 0 |
| Postman Basic | в свободном доступе | 1 | 0 |
| Dbeaver | в свободном доступе | 1 | 0 |
| PostgreSql | в свободном доступе | 1 | 0 |
| **Итого** | | | **40290** |

* + 1. **Расчёт материальных затрат на расходные материалы**

В качестве материальных затрат могут быть использованы материалы, используемые при проектировании, но не составляющие амортизационные отчисления. Расходы на данный вид затрат формируются из суммы цены на используемый материал. Перечень материалов приведен в таблице 16

Таблица 16 – материальные расходы

|  |  |  |  |
| --- | --- | --- | --- |
| **Вид материала** | **Количество** | **Цена за единицу, руб** | **Стоимость, руб** |
| Учебник по С# (ASP.NET) | 1 | 1086 | 1086 |
| Учебник по веб-программированию на языках HTML,JS,CSS | 1 | 1400 | 1400 |
| Флешка SanDisk Ultra Fit USB 3.1 32 ГБ | 1 | 489 | 489 |
| Бумага SvetoCopy A4 Classic 80 г/м², 500 л. | 1 | 436 | 436 |
| Набор шариковых ручек BRAUBERG, М-500 0.7 мм, 10шт | 1 | 139 | 139 |
| Картридж для принтера Canon PG-445XL | 1 | 2500 | 2500 |
| Диск CD-RW SmartTrack 700Mb 4-12x(10 шт) | 1 | 650 | 650 |

Продолжение таблицы16

|  |  |  |  |
| --- | --- | --- | --- |
| Папка для листов А4 | 1 | 130 | 130 |
| Файл-вкладыш (А4, 100 шт) | 1 | 360 | 360 |
| **Итого:** | | | **7190** |

### Расчёт затрат на электроэнергию

Расчёты затрат на электроэнергию для предприятия рассчитываются по формуле 6

(6)

где

𝑊 – мощность, потребляемая оборудованием, кВт/час;

𝑡 – время работы оборудования, дней;

𝑇 – тариф электроэнергии, руб.

Расчет затрат электроэнергии представлен в таблице 17.

Таблица 17 – Расчет затрат электроэнергии за 8-часовой рабочий день.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Оборудование** | **Время эксплуатации (час)** | **Количество** | **Мощность (кВт/час)** | **Тариф на электроэнергию (руб./кВт/ч)** | **Затраты на электроэнергию** |
| Персональный компьютер | 8 | 1 | 1,7 | 5,87 | 79,8 |
| **Итого** | | | | | 79,8 |

Общее количество затрат электроэнергии за время выполнения проекта составит:

* + 1. **Расчёт амортизационных отчислений**

На раздел амортизационных отчислений относят затраты, включающие выделение средств главных денежных фондов для возмещения последующего износа, обновления или ремонта оборудования. Определим амортизационные отчисления для ЭВМ, а также программных средств, используемых при проектировании. Расчёт амортизации на время выполнения проекта находится по формуле 7

|  |  |
| --- | --- |
|  | (7) |

где:

***Qф*** – стоимость программных и инструментальных средств, руб.

***На***- норма амортизации

Для произведения вычислением примем во внимание следующее:

* Ежедневное использование ЭВМ в одну смену в течение 8 часов;
* Средний эксплуатационный срок службы ЭВМ 5 лет;
* 250 – среднее количество рабочих дней в году.
* Персональный компьютер включает в себя такие устройства ввода-вывода как системный блок, монитор, клавиатура, мышь.

Процент амортизации для группы основных средств примем 25% в год.

Таблица 18 – расчёт амортизационный отчислений

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Виды инструментальных средств | Стоимость, руб. | Норма амортизации, % | Амортизационные отчисления, руб.. | |
| за 1 год,***Агод*** | на время проекта |
| Вычислительная техника | | | | | |
| 1. | Персональный компьютер | 69500 | 25 | 17375 | 5699 |
| Только для создания сайта или портала | | | | |  |
| 1. | Windows 10 Professional | 13290 | 25 | 3323 | 1090 |
| 2. | Visual Studio Professional | 27000 | 25 | 6750 | 2214 |
| **Итого:** | | | | 27448 | 9003 |

* + 1. **Основная заработная плата исполнителей**

В данном разделе происходит расчёт заработной платы всех сотрудников, принимающих участие создании проекта.

В разработке принимал участие один сотрудник. При наличии высшего образования и отсутствия опыта ему назначается должность «Инженер-программист согласно "Квалификационному справочнику должностей руководителей, специалистов и других служащих" (утв. Постановлением Минтруда России от 21.08.1998 N 37) (ред. от 27.03.2018).

В таблице 19 приведены расчетные затраты на оплату труда разработчиков программного продукта.

Таблица 19- Затраты на заработную плату

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель | Тарифная ставка, руб./мес. | Тарифная ставка, руб./день | Трудоемкость, чел. дней | Зарплата, руб |
| Инженер-программист | 30000 | 1193 | 82 | 97826 |
| **Итого:** | | | | 97826 |

* + 1. **Дополнительная заработная плата и фонд оплаты труда**

В период разработки программного обеспечения дополнительные выплаты рассчитываются относительно основной заработной платы и составляют десять процентов от ее величины:

|  |  |
| --- | --- |
|  | (8) |

Фонд оплаты труда (ФОТ) – это сумма основной и дополнительной заработной платы:

|  |  |
| --- | --- |
|  | (9) |

* + 1. **Отчисления на социальные нужды**

Отчисления на социальные нужды составляют 30% от суммы основной и дополнительной заработной платы:

|  |  |
| --- | --- |
|  | (10) |

* + 1. **Суммарные затраты на разработку**

На основе полученных данных составляется калькуляция плановой себестоимости разработки, представленная в таблице 20.

Из приведенной таблицы видно, что полная себестоимость программного продукта *Сп* = 272416 руб.

Таблица 20 − Суммарные затраты на разработку программного обеспечения

|  |  |  |  |
| --- | --- | --- | --- |
| Номер №/п | Наименование статьи | Расходы, руб. | В % к итогу |
| 1 | Оборудование и программное обеспечение | 109790 | 40,3 |
| 2 | Материальные затраты | 7190 | 2,6 |
| 3 | Затраты на электроэнергию | 6543 | 2,4 |
| 4 | Амортизационные отчисления | 9003 | 3,3 |
| 5 | Основная заработная плата | 97826 | 35,9 |
| 6 | Дополнительная заработная плата | 9782 | 3,6 |
| 7 | Отчисления на социальные нужды | 32282 | 11,9 |
| **Итого:** | | **272416** | **100** |

На рисунке 41 представлен состав затрат в процентном соотношении к общей стоимости разработки программного продукта.



Рисунок 41 - Состав затрат на разработку системы.

* 1. **Оценка эффективности**

Общие показатели экономической эффективности являются отношением полученных результатов разработки на сумму затрат на реализации программного обеспечения в стоимостном выражении.

В наше проекте разрабатывались информационная система задачей которой было упростить и улучшить процесс перераспределения продуктов питания. Реализация данной задачи не влечёт извлечение прибыли и ставит задачу некоммерческой помощи всем желающим.

Основные задачи деятельности некоммерческих организаций – это создание и увеличение общественных благ, а также достижение социальных и культурных целей в рамках удовлетворений потребностей граждан.

Эффективность данных организаций определяется как степень удовлетворённости работы проекта и в основном отражает в себе такие виды эффективности как: социальную, экологическую, социально-экономическую. При проектировании приложения ставились следующие требования эффективности:

* Сокращение потребностей благополучателей в расходах бюджета и уменьшение роста бедности
* Снижение роста полигонных сохранений и выброса вредных веществ
* Рациональное использование трудовых и природных ресурсов
  + 1. **Социальная эффективность**

Вычисления будем проводить, условно взяв численность населения среднего города России равным 100 тысяч жителей. По данным Росстата (по итогам 2020 года) общее количество населения находящихся за чертой бедности составило 15%. Люди, находящиеся в таком положении вынуждены тратить на продукты питания более половины своего месячного дохода. Также 6% россиян, зарабатывающих немного более прожиточного минимума вынуждены экономить на еде. Расчёты проведем ориентируюсь на то, что превалирующая часть пользователей имеет проблемы с приобретением продуктов питания. Количество населения в городе составит (таблица 21)

Таблица 21 – количество населения с достатком ниже среднего в среднем городе

|  |  |  |  |
| --- | --- | --- | --- |
| Номер №/п | Уровень дохода | В % от общей численности населения | Количество тыс. человек |
| 1 | Ниже прожиточного минимума | 15 | 15 |
| 2 | Ниже среднего уровня дохода | 6 | 6 |
| **Итого:** | | | **21** |

Условно взяв за расчёт то, что на старте приложением будут пользоваться 5% от общего количество получим ориентировочное и оно составит 1050 человек.

Количество затрат на продукты составит (таблица 22):

Таблица 22 – Затраты на продукты населения с достатком ниже среднего

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер №/п | Уровень дохода | Средний уровень дохода, руб. | Затраты на продукты в мес. % | Затраты на продукты в мес., руб |
| 1 | Ниже прожиточного минимума | 12000 | 45 | 5400 |
| 2 | Ниже среднего уровня дохода | 20000 | 32 | 6400 |

Приложение для перераспределения продуктов питания при средней базе пользователей в размере 2100 человек может позволить получать около 50% ежедневного рациона питания бесплатно.

Таблица 23 - Общая ежемесячная экономия расходов бюджета

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа доходов №/п | Количество человек | Затраты на продукты в мес., руб | Сэкономленные затраты в мес. на человека, руб | Сэкономленные затраты в мес., тыс,руб |
| 1 | 750 | 5400 | 2700 | 2025 |
| 2 | 300 | 6400 | 3200 | 960 |
| **Итого:** | | | | 2985 |

Исходя из произведённых расчётов сокращение затрат в месяц с помощью сервиса для населения с достатком ниже среднего составит 2,985 млн рублей.

* + 1. **Экологическая эффективность**

Для расчёта экологической эффективности необходимо узнать размер потребительской корзины на душу населения. **Потребительская корзина** — это набор продуктов питания необходимых человеку для здоровья и удовлетворения минимальных потребностей. Согласно федеральному закону «О потребительской корзине по Российской Федерации» от 03.12.2012 N 227-ФЗ человеку в месяц необходимо 84 кг продуктов.

Расчёт спасенной от выбрасывания еды представлен в таблице 24. Вычисления ведутся с учётом того, что пользователь будет получать через веб-приложение половину продовольственной корзины.

Таблица 24 – размер спасённой продукции от утилизации и выброса

|  |  |  |  |
| --- | --- | --- | --- |
| Группа доходов №/п | Количество человек | Размер половины продовольственной корзины в мес., кг | Размер спасённых от выброса продуктов в мес., тонн |
| 1 | 750 | 42 | 31,5 |
| 2 | 300 | 12,6 |
| **Итого:** | | | 44,1 |

Общее количество сохранённой от утилизации продукции составит 44,1 тонн

# ЗАКЛЮЧЕНИЕ

При выполнении выпускной квалификационной работы на тему: «Разработка автоматизированной информационной системы, реализующей концепцию перераспределения продуктов питания, на основе технологии ASP.NET MVC» получены следующие результаты:

1. Проведена постановка задачи проектирования и рассмотрена актуальность выбранного направления разработки. Выбрана модель распределения продуктов питания C2C, а также способ взаимодействия пользователей посредством чата.
2. Произведён анализ сервисов для перераспределения продуктов питания, а также рассмотрены их структурные модели работы и перспективы развития.
3. Определены требования к программной среде и разрабатываемому функционалу основными из которых являются:
   * Адаптивность и простота использования;
   * Производительность и минимальные технические требования;
   * Масштабируемость и возможность лёгкого модульного расширения.
4. Спроектирована база данных для обеспечения эффективного хранилища информации. Была выбрана современная ситема управления базами данных PostgreSQL.
5. Выбран фреймворк для разработки веб приложения Asp.net а в качестве высокоуровневого языка программирования C# 10 и Microsoft Visual Studio Professional 2022 в качестве интегрированной среды программирования.
6. Разработано программное обеспечение для перераспределения продуктов питания. Данный программный продукт позволяет решать следующие задачи:
   * Возможность авторизироваться и иметь профиль с личными данными и контактной информацией;
   * Возможность просмотра каталога продуктов для обмена на безвозмездной основе;
   * Возможность взаимодействовать и общаться с другими пользователями посредством чата;
   * Возможность выставлять продукты, а также производить их редактирование, скрытие и удаление;
   * Возможность добавления продуктов в избранное.
7. Рассмотрены общие вопросы безопасности жизнедеятельности при работе с ЭВМ. Проведён анализ негативных факторов, воздействующих на сетчатку глаза, а также рассчитаны показатели освещённости помещения. Рассмотрены профилактические нормы для предотвращения пожаров и рассмотрен план эвакуации при возникновении экстренной ситуации.
8. Произведены расчёты трудоемкости разработки программного обеспечения.
9. Разрабатываемое программное обеспечение позволяет существенно снизить затраты на приобретение продуктов питания и добиться уменьшения возрастания пищевых отходов и свалок. В ходе работы была определена стоимость затрат на программное обеспечение, и оно составило 272416 рублей.
10. Произведён анализ социальной и экологической эффективности, в результате которого удалось выяснить, что использование ПО позволяет достичь сокращение затрат у населения с достатком ниже среднего в размере 2,985 млн рублей в месяц, а также сохранить от утилизации 44,1 тонн продукции.
11. На основании проведенного исследования подтверждена целесообразность разработки данного программного обеспечения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальная техническая документация Microsoft [электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/>  (дата обращения: 15.06.22).
2. Официальная документация PostgreSql [электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/  (дата обращения: 15.06.22).
3. Осипов Н.А., Разработка приложений ASP.NET с применением ASP.NET [Текст] / СПб: Университет ИТМО, 2016. – 80 с.
4. Чамберс Джеймс, Пэкетт Дэвид, Тиммс Саймон ASP.NET Core. Разработка приложений. [Текст] / СПб.: Питер, 2018. — 464 с..
5. Исследование ТИАР-Центра «Фудшеринг в России. Эффективные способы спасения продовольствия» [электронный ресурс]. – Режим доступа: <https://tiarcenter.com/wp-content/uploads/2019/10/Foodsharing-in-Russia_October-1.pdf> (дата обращения: 15.06.22).
6. Исследование ТИАР-Центра «ESG-подход к решению проблемы нерационального использования продовольствия в России» [электронный ресурс]. – Режим доступа: https://tiarcenter.com/wp-content/uploads/2022/03/Foodsharing-Report-2022.pdf (дата обращения: 15.06.22).
7. Н.В. Крепша, Безопасность жизнедеятельности: учебное пособие для студентов. . [Текст] /Национальный исследовательский Томский политехнический университет 2014. – 198 с.
8. Рождественская Н.В., Богуславская С.Б., Боброва О.С.: Оценка эффективности проектов некоммерческих организаций, социального предпринимательства и гражданских инициатив. . [Текст] /   СПб, Издательство Политехнического университета, 2016. — 168 с
9. Справочная книга для проектирования электрического освещения. Под ред. Г. М. Кнорринга. Л.. [Текст] / «Энергия», 1976. 384 с

# ПРИЛОЖЕНИЕ А

**Руководство пользователя**

Основной целью руководства пользователя является описание функционала и помощь пользователям в последующем использовании ПО.

Веб-приложение создано с целью решить проблемы перераспределения питания. Разрабатываемое средство предоставляет пользователям обмениваться продуктами питания и коммуницировать посредством чата. Приложение адаптивно для всех современных браузеров и даёт возможность использоваться с устройств с различным расширением экрана начиная.

Для того чтобы быть участником системы необходимо пройти этап регистрации. Пользователь должен нажать на кнопку «Регистрация» и ввести почту и уникальный пароль для входа в систему. Процесс показан на рисунке А.1.

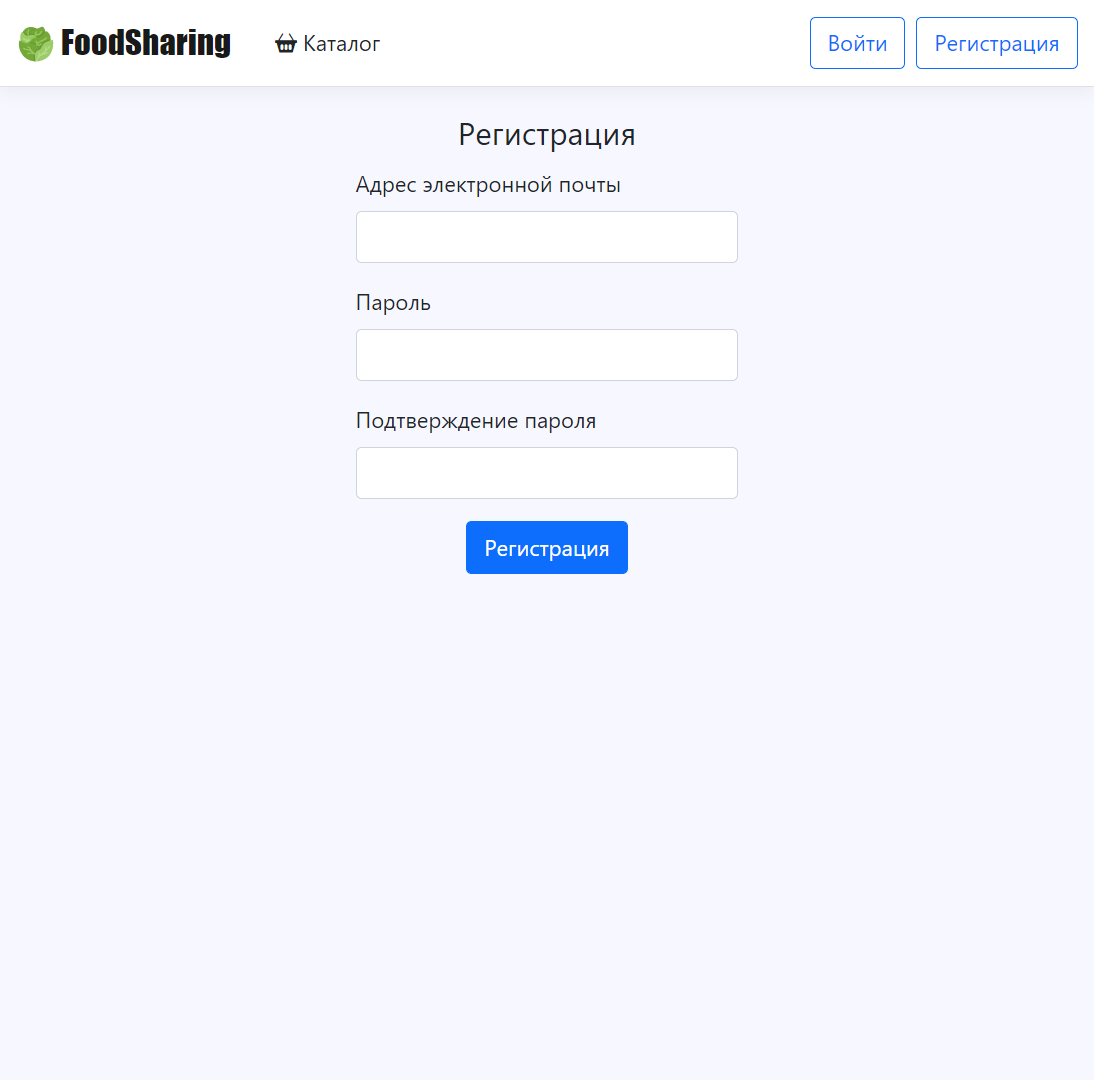


Рисунок А.1 – Страница Регистрации

После создания учетной записи пользователю необходимо заполнить страницу профиля введя в ячейки для ввода Фамилию, Имя, Телефон, а также адрес проживания. При желании пользователь может загрузить фотографию профиля. Страница профиля выглядит как показано на рисунке А.2.

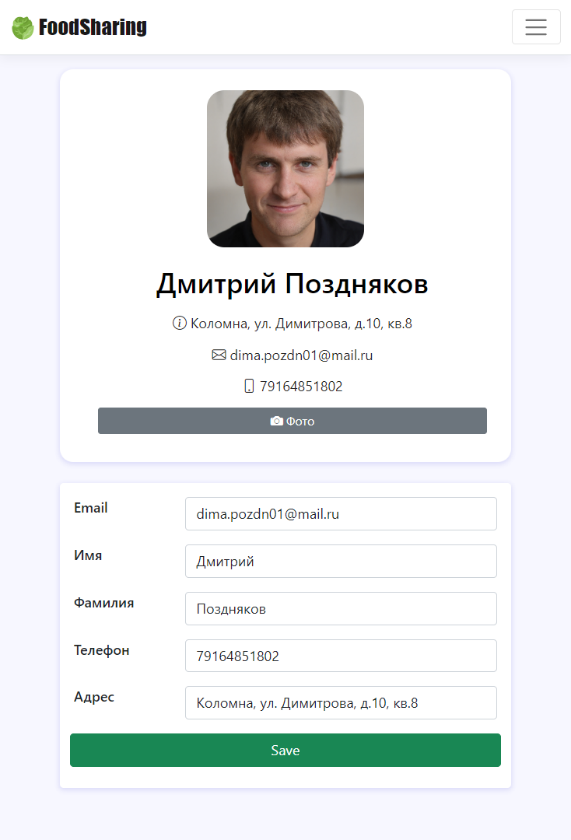


Рисунок А.2 – Страница профиля

После заполнения контактной информации пользователь может просматривать каталог товаров, а также осуществлять поиск по необходимой категории. Страница каталога товаров изображена на рисунке А.3.

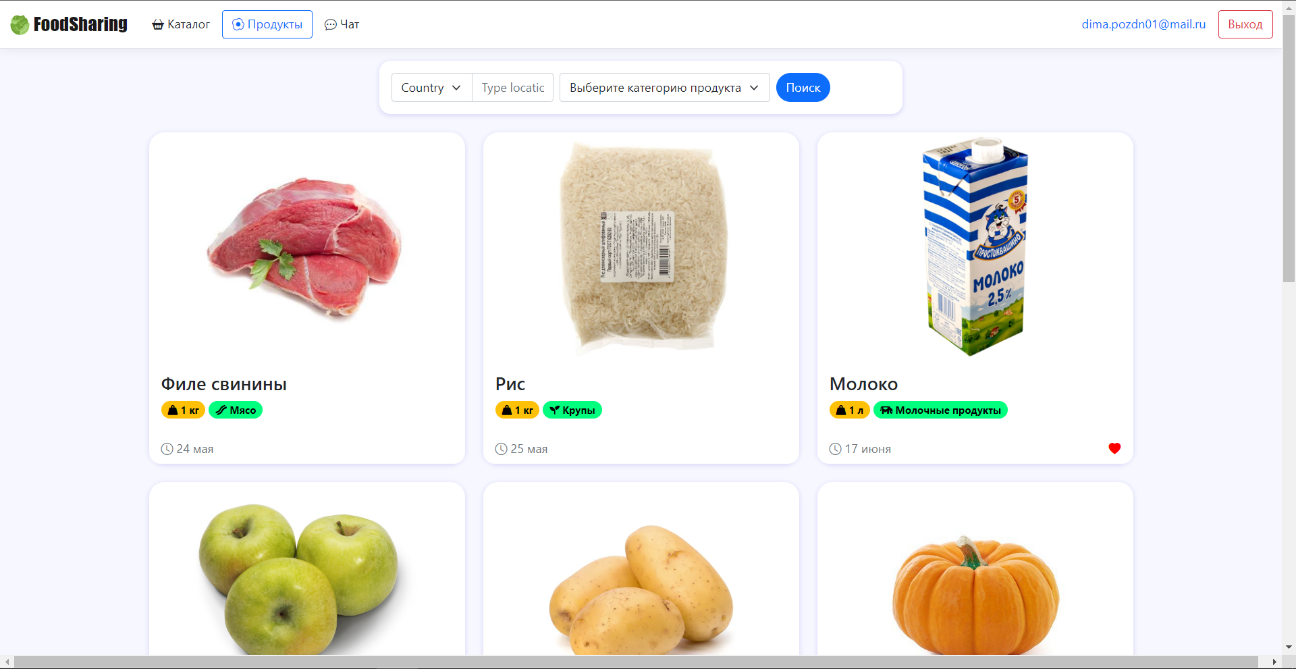


Рисунок А.3 – Страница каталога продуктов

При заинтересованности в продукте пользователь может нажать на карточку с продуктом и ему откроется подробная информация о нём, а также контактная информация держателя товара. Страница данных о товаре показана на рисунке А.4.



Рисунок А.4 – Страница информации о товаре и го держателе

Со страницы данных о товаре реализован процесс просмотра контактного телефона по кнопке «Показать телефон» а также коммуникация с держателем товара посредством перехода в чат с помощью кнопки «Написать сообщение». Процесс перехода на страницу чата продемонстрирован на рисунке А.5.

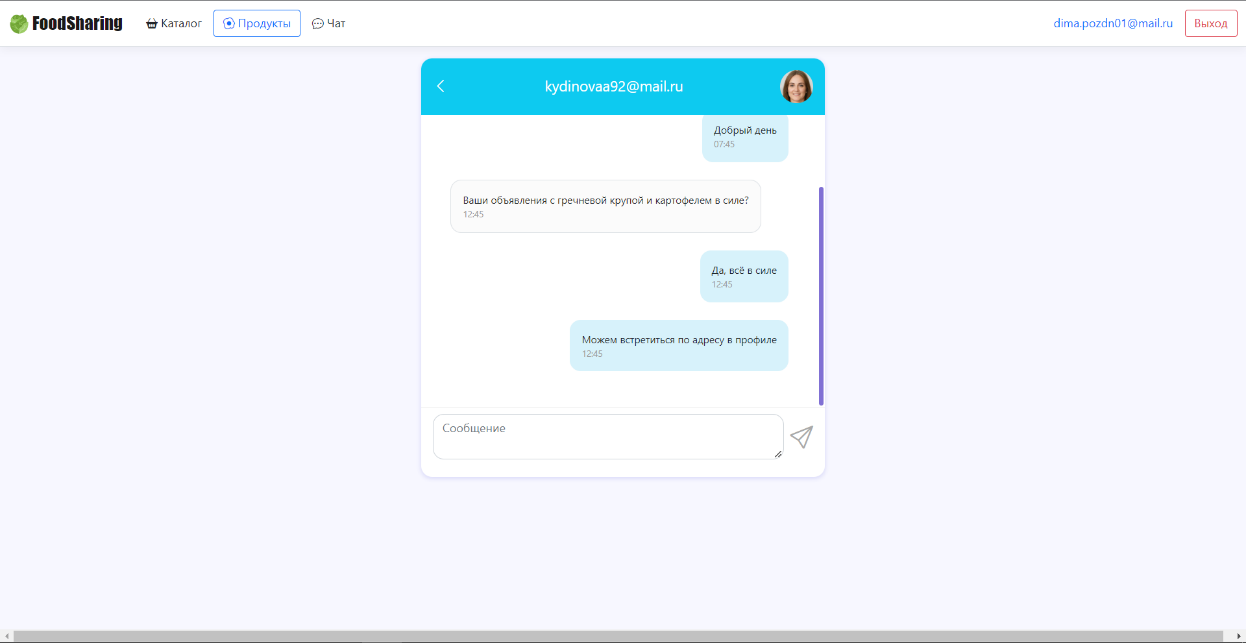


Рисунок А.5 – Страница чата

Со страницы чата пользователь может перейти в меню выбора собеседника посредством нажатия стрелки назад на верхнем меню чата, где отображены все имеющиеся контакты, с которыми была осуществлена переписка. Страница списка собеседников изображена на рисунке А.6.

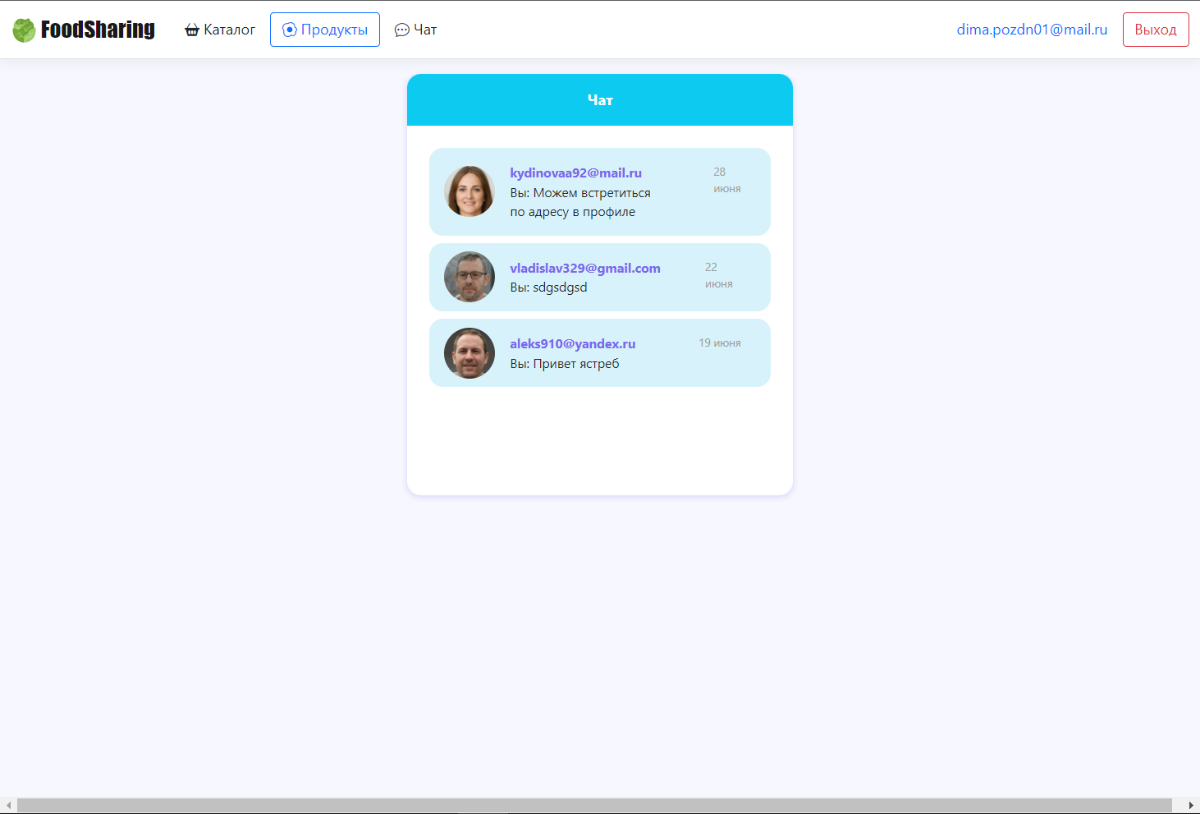


Рисунок А.6 – Страница выбора собеседника

Также со страницы чата пользователю предоставлена возможность перейти на страницу профиля человека, с которым ведётся переписка. На странице других пользователей будет отображена контактная информация, а также имеющиеся в наличии товары для получения. На рисунке А.7 изображена страница профиля собеседника.



Рисунок А.7 – страница профиля держателя продукта

При необходимости пользователь может добавлять свои продуты. Для этого он должен посредством нажатия кнопки «Добавить продукт» перейти на страницу добавления. На ней необходимо заполнить такие данные о продукте как:

* Название
* Описание
* Количество
* Фото
* Категория

Процесс перехода изображен на рисунке А.8.

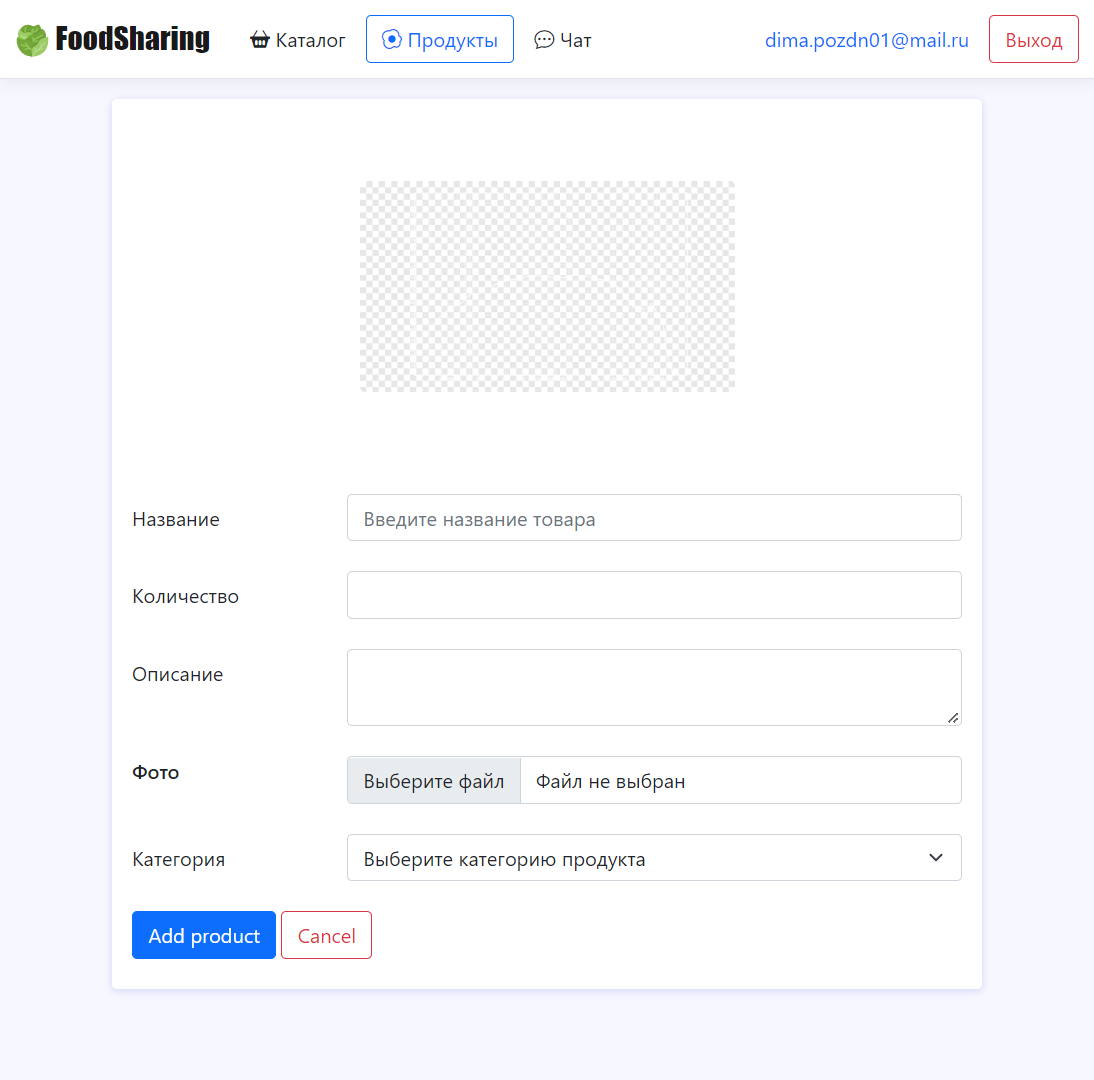


Рисунок А.8 – Страница добавления продукта

Когда пользователь будет иметь в своём наличии выставленные товары, то ему откроется возможность такого взаимодействия с ними как редактирование, удаление и скрытие. Страница «Мои продукты» изображена на рисунке А.9

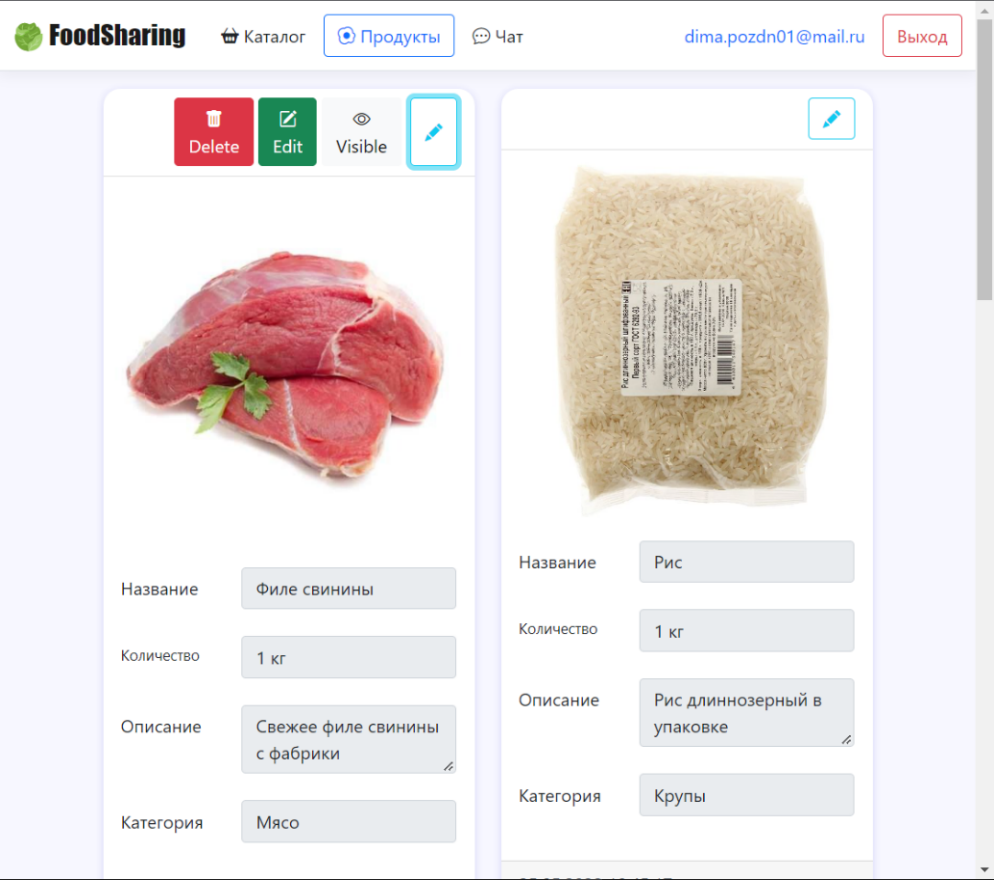


Рисунок А.9 – Страница «Мои продуты»

# ПРИЛОЖЕНИЕ Б

**Листинг проекта**

**Класс** **AccountController**

[Authorize]

public class AccountController : Controller

{

private readonly IConfiguration \_config;

private IUserService \_userService;

private IProductService \_productService;

public AccountController(IConfiguration config, IUserService userService, IProductService productService)

{

\_config = config;

\_userService = userService;

\_productService = productService;

}

[Route("Profile")]

public async Task<ActionResult> Profile()

{

UserProfileView userProfile = await \_userService.GetUserProfile(User.Identity.Name);

return View("Profile", userProfile);

}

[HttpGet]

public async Task<ActionResult> ProfileInfo(Guid userid)

{

string email = User.Identity.Name;

Guid currentUserId = await \_userService.GetUserIdByEmail(email);

ProfileInfoView profileInfoView = await \_userService.GetUserProfileInfo(userid, currentUserId);

return View(profileInfoView);

}

[HttpGet]

public async Task<ActionResult> ChangeProductFavourites(Guid productid)

{

string email = User.Identity.Name;

Guid userId = await \_userService.GetUserIdByEmail(email);

Guid userProfileId = await \_userService.GetUserIdByProductId(productid);

\_productService.ChangeProductFavourite(userId, productid);

return RedirectToAction("ProfileInfo", "Account", new { userid = userProfileId });

}

[HttpPost]

public async Task<ActionResult> EditProfile(UserProfileView model)

{

if (!ModelState.IsValid) return View();

await \_userService.SaveUserProfile(model, User.Identity.Name);

TempData["SaveChanges"] = "Изменения были применены";

return RedirectToAction("Profile", "Account");

}

[HttpPost]

public async Task<ActionResult> SavePhoto(UserProfileView model)

{

string email = User.Identity.Name;

UserProfileView userProfile = await \_userService.GetUserProfile(email);

model.Id = userProfile.Id;

if (model.Image is null)

{

model.Avatar = userProfile.Avatar;

TempData["UploadPhotoError"] = "Фото не было загружено";

return View("Profile", model);

}

await \_userService.SaveUserProfile(model, email);

TempData["SaveChanges"] = "Изменения были применены";

return RedirectToAction("Profile", "Profile");

}

}

**Класс AuthController**

[AllowAnonymous]

public class AuthController : Controller

{

private readonly IConfiguration \_config;

private IUserService \_userService;

public AuthController(IConfiguration config, IUserService userService)

{

\_config = config;

\_userService = userService;

}

[HttpGet]

[Route("Login")]

public IActionResult Login()

{

return View();

}

[HttpGet]

[Route("Registration")]

public IActionResult Registration()

{

return View();

}

[HttpPost]

[Route("Login")]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Login(LoginViewModel model)

{

if (!ModelState.IsValid) return View("Login", model);

User user = await \_userService.GetUserByEmail(model.Email);

if (user is null)

{

TempData["LoginError"] = "Пользователь не найден";

return View(model);

}

if (user.Password != model.Password)

{

TempData["LoginError"] = "Неверно введён пароль";

return View(model);

} else

{

await CookieEvents.Authenticate(model.Email, HttpContext);

}

return RedirectToAction("Profile", "Account");

}

[Authorize]

public async Task<IActionResult> LogOut()

{

await CookieEvents.SignOut(HttpContext);

return RedirectToAction("Login", "Auth");

}

[HttpPost]

[Route("RegisterUser")]

[ValidateAntiForgeryToken]

public async Task<ActionResult> RegisterUser(RegistrationView model)

{

if (!ModelState.IsValid) return View("Registration", model);

User user = await \_userService.GetUserByEmail(model.Email);

if (user is null)

{

await \_userService.RegisterUser(model);

await CookieEvents.Authenticate(model.Email, HttpContext);

return RedirectToAction("Profile", "Account");

}

else

{

TempData["RegisterError"] = "Пользователь с данной почтой уже существует";

return View("Registration", model);

}

}

}

**Класс ChatController**

[Authorize]

public class ChatController : Controller

{

private readonly IConfiguration \_config;

private IChatService \_chatService;

private IUserService \_userService;

public ChatController(IConfiguration config, IChatService chatService, IUserService userService)

{

\_config = config;

\_chatService = chatService;

\_userService = userService;

}

public IActionResult Index()

{

return View();

}

[HttpGet]

public async Task<IActionResult> ChatMessage(Guid userId)

{

if (userId == Guid.Empty) return View();

MessagesHistoryView messegesHistory = await \_chatService.GetMessagesHistory(userId, User.Identity.Name);

return View(messegesHistory);

}

public async Task<IActionResult> ChatUsers()

{

Guid UserId = await \_userService.GetUserIdByEmail(User.Identity.Name);

AllDialogsView messages = await \_chatService.GetTalkers(UserId);

messages.User = User.Identity.Name;

return View(messages);

}

}

**Класс HomeController**

[Authorize]

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

public IActionResult Index()

{

return View();

}

public IActionResult Products()

{

return View();

}

public IActionResult NewProduct()

{

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

**Класс ProductController**

public class ProductController : Controller

{

private readonly IConfiguration \_config;

private readonly IProductService \_productService;

private readonly IUserService \_userService;

public ProductController(IConfiguration config, IProductService productService, IUserService userService)

{

\_config = config;

\_productService = productService;

\_userService = userService;

}

public IActionResult Products()

{

return RedirectToAction("GetProducts", "Product");

}

public async Task<ActionResult> GetProducts(ProductView model)

{

User user = await \_userService.GetUserByEmail(User.Identity.Name);

List<ProductView> productViews = await \_productService.GetProductsViews(user.Id);

return View("Products", productViews);

}

[HttpGet]

public async Task<ActionResult> DeleteProduct(Guid id)

{

await \_productService.DeleteProduct(id);

return RedirectToAction("GetProducts", "Product");

}

[HttpGet]

[Route("/Products/NewProduct")]

public async Task<ActionResult> NewProduct()

{

ProductView model = new ProductView();

List<ProductCategory> productCategories = await \_productService.GetProductCategories();

model.ProductCategories = productCategories;

return View(model);

}

[HttpGet]

[Route("/Products/Favourites")]

public async Task<ActionResult> Favourites()

{

string email = User.Identity.Name;

Guid userId = await \_userService.GetUserIdByEmail(email);

List<ProductView> productViews = await \_productService.GetProductsFavouritesViews(userId);

return View(productViews);

}

[HttpPost]

[Route("/Products/NewProduct")]

public async Task<ActionResult> NewProduct(ProductView model)

{

if (model.IFormFile is null)

{

TempData["UploadPhotoError"] = "Фото не было загружено";

return View("NewProduct", model);

}

User user = await \_userService.GetUserByEmail(User.Identity.Name);

model.UserId = user.Id;

await \_productService.SaveProduct(model);

TempData["AddProductSeccess"] = "Товар был добавлен";

return RedirectToAction("NewProduct", "Product");

}

[HttpGet]

[Route("/Products/EditProduct")]

public async Task<ActionResult> EditProduct(Guid id)

{

User user = await \_userService.GetUserByEmail(User.Identity.Name);

List<ProductCategory> productCategories = await \_productService.GetProductCategories();

ProductView model = await \_productService.GetProduct(id);

model.ProductCategories = productCategories;

model.UserId = user.Id;

return View(model);

}

[HttpPost]

[Route("/Products/EditProduct")]

public async Task<ActionResult> EditProduct(ProductView model)

{

if (model.IFormFile is not null)

{

model.Image = FileTools.GetBytes(model.IFormFile);

}

await \_productService.EditProduct(model);

return RedirectToAction("GetProducts", "Product");

}

[HttpGet]

public IActionResult Сatalog()

{

return RedirectToAction("GetCatalog", "Product");

}

[Route("/Products/Сatalog")]

public async Task<ActionResult> GetCatalog(CatalogListView model)

{

CatalogListView catalogListView = new CatalogListView();

if (User.Identity.Name is not null)

{

Guid currentUserId = await \_userService.GetUserIdByEmail(User.Identity.Name);

catalogListView.CatalogViews = await \_productService.GetCatalogViews(model.CategoryId, currentUserId);

} else catalogListView.CatalogViews = await \_productService.GetCatalogViews(model.CategoryId);

catalogListView.UserEmail = User.Identity.Name;

catalogListView.ProductCategories = await \_productService.GetProductCategories();

return View("Сatalog", catalogListView);

}

[HttpGet]

public async Task<ActionResult> ChangeProductFavourites(Guid productid)

{

string email = User.Identity.Name;

Guid userId = await \_userService.GetUserIdByEmail(email);

Guid userProfileId = await \_userService.GetUserIdByProductId(productid);

\_productService.ChangeProductFavourite(userId, productid);

return RedirectToAction("Сatalog", "Product");

}

[HttpGet]

public async Task<ActionResult> ProductInfo(Guid id)

{

ProductView productView = await \_productService.GetProduct(id);

UserProfileView userProfileViewModel = await \_userService.GetUserProfile(productView.UserId);

string email = User.Identity.Name;

Guid userId = await \_userService.GetUserIdByEmail(email);

Guid UserIdforProduct = await \_userService.GetUserIdByProductId(id);

string EmailForProduct = await \_userService.GetUserEmailById(UserIdforProduct);

Boolean favourites = await \_productService.GetProductFavourites(userId, id);

productView.IsFavourite = favourites;

productView.Email = EmailForProduct;

ProductInfoView model = new ProductInfoView(userProfileViewModel, productView, email);

return View(model);

}

}

**Класс ChatHub**

[Authorize]

public class ChatHub : Hub

{

private readonly IConfiguration \_config;

private IUserService \_userService;

private IChatService \_chatService;

public ChatHub(IConfiguration config, IUserService userService, IChatService chatService)

{

\_config = config;

\_userService = userService;

\_chatService = chatService;

}

public async Task SendMessage(string message, string userTo)

{

Guid userToId = new Guid(userTo);

string userFromEmail = Context.User.Identity.Name; // отправителя

string userToEmail = await \_userService.GetUserEmailById(userToId); // почта получателя

Message messageModel = new Message();

messageModel.Id = Guid.NewGuid();

messageModel.FromUserId = await \_userService.GetUserIdByEmail(userFromEmail);

messageModel.ToUserId = userToId;

messageModel.Content = message;

messageModel.CreatedAt = DateTime.Now;

string time = messageModel.CreatedAt.ToString("HH:mm");

await \_chatService.Send(messageModel);

string avatar = Convert.ToBase64String(await \_userService.GetAvatar(messageModel.FromUserId));

string sender = userToEmail;

await Clients.User(userToEmail).SendAsync("Receive", userFromEmail, message, time, sender);

sender = userFromEmail;

await Clients.User(userFromEmail).SendAsync("Receive", userFromEmail, message, time, sender);

}

public override async Task OnConnectedAsync()

{

await Clients.All.SendAsync("Notify", $"{Context.User.Identity.Name}");

await base.OnConnectedAsync();

}

public override async Task OnDisconnectedAsync(Exception exception)

{

await Clients.All.SendAsync("Notify", $"{Context.User.Identity.Name}");

await base.OnDisconnectedAsync(exception);

}

}

**Класс CookieEvents**

public static class CookieEvents

{

public static async Task Authenticate(string username, HttpContext httpContext)

{

var claims = new List<Claim>

{

new Claim(ClaimTypes.Name, username)

};

ClaimsIdentity claimsIdentity = new ClaimsIdentity(claims, "Cookies");

await httpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(claimsIdentity));

}

public static async Task SignOut(HttpContext httpContext)

{

await httpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

}

}

**Класс FileTools**

public static class FileTools

{

public static byte[] GetBytes(IFormFile file)

{

byte[] imageData = null;

using (var binaryReader = new BinaryReader(file.OpenReadStream()))

{

imageData = binaryReader.ReadBytes((int)file.Length);

}

return imageData;

}

}

**Класс ProductService**

public class ProductService : IProductService

{

private readonly IProductRepository \_productRepository;

private readonly IUserRepository \_userRepository;

public ProductService(IProductRepository productRepository, IUserRepository userRepository)

{

\_productRepository = productRepository;

\_userRepository = userRepository;

}

public Task SaveProduct(ProductView model)

{

model.Image = FileTools.GetBytes(model.IFormFile);

return \_productRepository.AddProduct(model);

}

public Task DeleteProduct(Guid id)

{

return \_productRepository.DeleteProduct(id);

}

public async Task<List<ProductView>> GetProductsFavouritesViews(Guid userid)

{

List<Guid> favouritesId = await \_productRepository.GetUserProductFavouritesIds(userid);

List<Product> products = await \_productRepository.GetSelectionProducts(favouritesId.ToArray());

if (products is null) return new List<ProductView>();

int[] categoryIds = products.Select(x => x.CategoryId).ToArray();

List<ProductCategory> productCategories = await \_productRepository.GetProductCategories(categoryIds);

Guid[] userIds = products.Select(x => x.UserId).Distinct().ToArray();

Guid[] productIds = products.Select(x => x.Id).ToArray();

List<User> users = await \_userRepository.GetUsers(userIds);

return products.Select(x =>

{

ProductCategory? productCategory = productCategories.FirstOrDefault(c => c.Id == x.CategoryId);

User? user = users.FirstOrDefault(c => c.Id == x.UserId);

Task<String?> time = TimeConverter.GetTime(x.CreatedAt);

return new ProductView(x.Id, x.UserId, user?.Email ?? "", x.Name, x.Description, x.CategoryId,

productCategory?.Name ?? "", productCategory.Icon, x.Quantity, x.Image, true, x.CreatedAt, time.Result);

}).ToList();

}

public async Task<List<ProductView>> GetProductsViews(Guid userid)

{

List<Product> products = await \_productRepository.GetProducts(userid);

if (products is null) return new List<ProductView>();

int[] categoryIds = products.Select(x => x.CategoryId).ToArray();

List<ProductCategory> productCategories = await \_productRepository.GetProductCategories(categoryIds);

Guid[] userIds = products.Select(x => x.UserId).Distinct().ToArray();

Guid[] productIds = products.Select(x => x.Id).ToArray();

List<User> users = await \_userRepository.GetUsers(userIds);

List<Favourite> favourites = await \_productRepository.GetProductFavourites(userIds, productIds);

return products.Select(x =>

{

ProductCategory? productCategory = productCategories.FirstOrDefault(c => c.Id == x.CategoryId);

User? user = users.FirstOrDefault(c => c.Id == x.UserId);

Favourite? favourite = favourites.FirstOrDefault(c => c.UserId == x.UserId && c.ProductId == x.Id);

Task<String?> time = TimeConverter.GetTime(x.CreatedAt);

return new ProductView(x.Id, x.UserId, user?.Email ?? "", x.Name, x.Description, x.CategoryId,

productCategory?.Name ?? "", productCategory.Icon, x.Quantity, x.Image, favourite != null, x.CreatedAt, time.Result);

}).ToList();

}

public async Task<List<ProductView>> GetProductsViews(Guid userid, Guid currentUserId)

{

List<Product> products = await \_productRepository.GetProducts(userid);

if (products is null) return new List<ProductView>();

int[] categoryIds = products.Select(x => x.CategoryId).ToArray();

List<ProductCategory> productCategories = await \_productRepository.GetProductCategories(categoryIds);

Guid[] userIds = products.Select(x => x.UserId).Distinct().ToArray();

Guid[] productIds = products.Select(x => x.Id).ToArray();

List<User> users = await \_userRepository.GetUsers(userIds);

List<Favourite> favourites = await \_productRepository.GetProductFavourites(new[] { currentUserId }, productIds);

return products.Select(x =>

{

ProductCategory? productCategory = productCategories.FirstOrDefault(c => c.Id == x.CategoryId);

User? user = users.FirstOrDefault(c => c.Id == x.UserId);

Favourite? favourite = favourites.FirstOrDefault(c => c.UserId == currentUserId && c.ProductId == x.Id);

Task<String?> time = TimeConverter.GetTime(x.CreatedAt);

return new ProductView(x.Id, x.UserId, user?.Email ?? "", x.Name, x.Description, x.CategoryId,

productCategory?.Name ?? "", productCategory.Icon, x.Quantity, x.Image, favourite != null, x.CreatedAt, time.Result);

}).ToList();

}

public async Task<List<ProductView>> GetCatalogViews(int categoryId, Guid currentUserId = default)

{

List<Product> products = (categoryId == null || categoryId == 0)

? await \_productRepository.GetCatalog()

: await \_productRepository.GetCatalog(categoryId);

if (products.Count == 0) return new List<ProductView>();

int[] categoryIds = products.Select(x => x.CategoryId).ToArray();

Guid[] userIds = products.Select(x => x.UserId).Distinct().ToArray();

Guid[] productIds = products.Select(x => x.Id).ToArray();

List<ProductCategory> productCategories = new List<ProductCategory>();

productCategories = await \_productRepository.GetProductCategories(categoryIds);

List<Favourite> favourites = await \_productRepository.GetProductFavourites(new[] { currentUserId }, productIds);

List<User> Users = await \_userRepository.GetUsers(userIds);

return products.Select(x =>

{

ProductCategory? productCategory = productCategories.FirstOrDefault(c => c.Id == x.CategoryId);

User? user = Users.FirstOrDefault(c => c.Id == x.UserId);

Favourite? favourite = favourites.FirstOrDefault(c => c.UserId == currentUserId && c.ProductId == x.Id);

Task<String?> time = TimeConverter.GetTime(x.CreatedAt);

return new ProductView(x.Id, x.UserId, user?.Email ?? "", x.Name,

x.Description, x.CategoryId, productCategory?.Name ?? "", productCategory.Icon, x.Quantity, x.Image, favourite != null, x.CreatedAt, time.Result);

}).ToList();

}

public async Task<ProductView> GetProduct(Guid productid)

{

Product product = await \_productRepository.GetProduct(productid);

ProductCategory productCategory = await \_productRepository.GetProductCategory(product.CategoryId);

if (product is null) return null;

var configuration = new MapperConfiguration(cfg =>

{

cfg.CreateMap<Product, ProductView>();

});

var mapper = configuration.CreateMapper();

var productsViewModel = mapper.Map<Product, ProductView>(product);

productsViewModel.CategoryName = productCategory.Name;

return productsViewModel;

}

public async Task EditProduct(ProductView model)

{

Product product = await \_productRepository.GetProduct(model.Id);

//model.Image = product.Image;

await \_productRepository.EditProduct(model);

}

public async Task<List<ProductCategory>> GetProductCategories()

{

return await \_productRepository.GetProductCategories();

}

public async Task ChangeProductFavourite(Guid userid, Guid productid)

{

Favourite favourites = await \_productRepository.GetProductFavourites(userid, productid);

if (favourites is null)

{

favourites = new Favourite();

favourites.Id = Guid.NewGuid();

favourites.ProductId = productid;

favourites.UserId = userid;

favourites.CreatedAt = DateTime.Now;

await \_productRepository.AddProductFavourites(favourites);

} else

await \_productRepository.DeleteProductFavourites(favourites.Id);

}

public async Task<Boolean> GetProductFavourites(Guid userid, Guid productid)

{

Favourite favorites = await \_productRepository.GetProductFavourites(userid, productid);

if (favorites is null)

return false;

else return true;

}

}

**Класс ProductRepository**

public class ProductRepository : IProductRepository

{

private DbConnection \_dbConnection;

public ProductRepository(IConfiguration config)

{

\_dbConnection = new DbConnection(config.GetConnectionString("DefaultConnection"));

}

public Task AddProduct(ProductView model)

{

string expression = @"INSERT INTO products (id, userid, name, description, categoryid, quantity, image, createdat)

VALUES (@id, @userid, @name, @description, @categoryid, @quantity, @image, @createdat)";

model.Id = Guid.NewGuid();

model.CreatedAt = DateTime.Now;

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(model.Id), model.Id),

new NpgsqlParameter(nameof(model.UserId), model.UserId),

new NpgsqlParameter(nameof(model.Name), model.Name),

new NpgsqlParameter(nameof(model.Description), model.Description),

new NpgsqlParameter(nameof(model.CategoryId), model.CategoryId),

new NpgsqlParameter(nameof(model.Quantity), model.Quantity),

new NpgsqlParameter(nameof(model.Image), model.Image),

new NpgsqlParameter(nameof(model.CreatedAt), model.CreatedAt),

};

return \_dbConnection.Add(expression, parameters);

}

public Task<List<Product>> GetProducts(Guid userid)

{

string expression = @"SELECT \* FROM products WHERE userid = @userid";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(userid), userid),

};

return \_dbConnection.GetList(expression, ProductConverter.MapToProducts, parameters);

}

public Task<List<Product>> GetSelectionProducts(Guid[] userids)

{

string expression = @"SELECT \* FROM products WHERE id = ANY(@userids)";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(userids), userids),

};

return \_dbConnection.GetList(expression, ProductConverter.MapToProducts, parameters);

}

public Task<List<Product>> GetCatalog(int categoryId)

{

string expression = @"SELECT \* FROM products WHERE categoryId = @categoryId";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(categoryId), categoryId),

};

return \_dbConnection.GetList(expression, ProductConverter.MapToProducts, parameters);

}

public Task<List<Product>> GetCatalog()

{

string expression = @"SELECT \* FROM products";

return \_dbConnection.GetList(expression, ProductConverter.MapToProducts);

}

public Task<Product> GetProduct(Guid id)

{

string expression = @"SELECT \* FROM products WHERE id = @id";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(id), id),

};

return \_dbConnection.Get(expression, ProductConverter.MapToProduct, parameters);

}

public Task<List<ProductCategory>> GetProductCategories()

{

string expression = @"SELECT \* FROM products\_categories";

return \_dbConnection.GetList(expression, ProductConverter.MapToProductCategories);

}

public Task<List<ProductCategory>> GetProductCategories(int[] ids)

{

string expression = @"SELECT \* FROM products\_categories WHERE id = ANY(@ids)";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(ids), ids),

};

return \_dbConnection.GetList(expression, ProductConverter.MapToProductCategories, parameters);

}

public Task<List<Favourite>> GetProductFavourites(Guid[] userids, Guid[] productids )

{

string expression = @"SELECT \* FROM product\_favourites WHERE (userid = ANY(@userids) AND productid = ANY(@productids))";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(userids), userids),

new NpgsqlParameter(nameof(productids), productids),

};

return \_dbConnection.GetList(expression, ProductConverter.MapToFavourites, parameters);

}

public Task<List<Favourite>> GetUserProductFavourites(Guid userid)

{

string expression = @"SELECT \* FROM product\_favourites WHERE userid = @userid";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(userid), userid),

};

return \_dbConnection.GetList(expression, ProductConverter.MapToFavourites, parameters);

}

public Task<List<Guid>> GetUserProductFavouritesIds(Guid userid)

{

string expression = @"SELECT productid FROM product\_favourites WHERE userid = @userid";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(userid), userid),

};

return \_dbConnection.GetList(expression, ProductConverter.MapToGuid, parameters);

}

public Task<ProductCategory> GetProductCategory(int id)

{

string expression = @"SELECT \* FROM products\_categories WHERE id = @id";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(id), id),

};

return \_dbConnection.Get(expression, ProductConverter.MapToProductCategory, parameters);

}

public Task EditProduct(ProductView model)

{

string expression = @"UPDATE products SET

name = @name,

description = @description,

categoryid = @categoryid,

quantity = @quantity,

image = @image WHERE id = @id";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(model.Id), model.Id),

new NpgsqlParameter(nameof(model.Name), model.Name),

new NpgsqlParameter(nameof(model.Description), model.Description),

new NpgsqlParameter(nameof(model.CategoryId), model.CategoryId),

new NpgsqlParameter(nameof(model.Quantity), model.Quantity),

new NpgsqlParameter(nameof(model.Image), model.Image),

};

return \_dbConnection.Add(expression, parameters);

}

public Task DeleteProduct(Guid id)

{

string expression = @"DELETE FROM products WHERE id = @id";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(id), id),

};

return \_dbConnection.Add(expression, parameters);

}

public Task AddProductFavourites(Favourite model)

{

string expression = @"INSERT INTO product\_favourites (id, productid, userid, createdat)

VALUES (@id, @productid, @userid, @createdat)";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(model.Id), model.Id),

new NpgsqlParameter(nameof(model.ProductId), model.ProductId),

new NpgsqlParameter(nameof(model.UserId), model.UserId),

new NpgsqlParameter(nameof(model.CreatedAt), model.CreatedAt),

};

return \_dbConnection.Add(expression, parameters);

}

public Task DeleteProductFavourites(Guid id)

{

string expression = @"DELETE FROM product\_favourites WHERE id = @id";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(id), id),

};

return \_dbConnection.Add(expression, parameters);

}

public Task<Favourite> GetProductFavourites(Guid userid, Guid productid)

{

string expression = @"SELECT \* FROM product\_favourites WHERE userid = @userid AND productid = @productid ";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(userid), userid),

new NpgsqlParameter(nameof(productid), productid),

};

return \_dbConnection.Get(expression, ProductConverter.MapToFavourite, parameters);

}

}

**Класс ChatService**

public class ChatService : IChatService

{

private readonly IChatRepository \_chatRepository;

private IUserService \_userService;

public ChatService(IChatRepository chatRepository, IUserService userService)

{

\_chatRepository = chatRepository;

\_userService = userService;

}

public Task Send(Message model)

{

return \_chatRepository.Send(model);

}

public async Task<List<MessageView>> GetMessages(Guid fromuserid, Guid touserid)

{

List<Message> messages = await \_chatRepository.GetMessages(fromuserid, touserid);

if (messages.Count == 0) return new List<MessageView>();

return messages.Select(x =>

{

Task<string> toUserEmail = \_userService.GetUserEmailById(x.ToUserId);

Task<string> fromUserEmail = \_userService.GetUserEmailById(x.FromUserId);

return new MessageView(x.Id, x.FromUserId, fromUserEmail.Result, x.ToUserId, toUserEmail.Result, x.Content, x.CreatedAt);

}).ToList();

}

public async Task<MessagesHistoryView> GetMessagesHistory(Guid userId, string email)

{

MessagesHistoryView messegesHistory = new MessagesHistoryView();

messegesHistory.FromUserId = await \_userService.GetUserIdByEmail(email);

messegesHistory.ToUserId = userId;

messegesHistory.FromUserAvatar = await \_userService.GetAvatar(messegesHistory.FromUserId);

messegesHistory.ToUserAvatar = await \_userService.GetAvatar(messegesHistory.ToUserId);

messegesHistory.FromUserEmail = email;

messegesHistory.ToUserEmail = await \_userService.GetUserEmailById(userId);

messegesHistory.Messages = (await GetMessages(messegesHistory.FromUserId, messegesHistory.ToUserId)).OrderBy(x => x.CreatedAt).ToList();

return messegesHistory;

}

public async Task<Dialog> GetDialog(Guid fromuserid, Guid touserid)

{

List<Message> messages = await \_chatRepository.GetMessages(fromuserid, touserid);

Message message = messages.OrderBy(x => x.CreatedAt).Last();

string time = await TimeConverter.GetTime(message.CreatedAt);

string toUserEmail = await \_userService.GetUserEmailById(message.ToUserId);

string fromUserEmail = await \_userService.GetUserEmailById(message.FromUserId);

byte[] toUserAvatar = await \_userService.GetAvatar(message.ToUserId);

byte[] fromUserAvatar = await \_userService.GetAvatar(message.FromUserId);

Dialog dialog = new Dialog(message.Id, message.FromUserId, fromUserEmail, fromUserAvatar, message.ToUserId, toUserEmail, toUserAvatar, message.Content, message.CreatedAt, time);

if (dialog is null)

return new Dialog();

else

return dialog;

}

public async Task<AllDialogsView> GetTalkers(Guid userid)

{

List<Guid> userids = await GetTalkersId(userid);

string email = await \_userService.GetUserEmailById(userid);

AllDialogsView allDialogs = new AllDialogsView();

foreach (Guid user in userids)

{

Dialog dialog = await GetDialog(user, userid);

if (allDialogs.Dialog is null)

allDialogs.Dialog = new List<Dialog>() { dialog };

else

allDialogs.Dialog.Add(dialog);

}

allDialogs.Dialog = allDialogs.Dialog.OrderByDescending(x => x.CreatedAt).ToList();

return allDialogs;

}

public async Task<List<Guid>> GetTalkersId(Guid userid)

{

List<Guid> toTalkers = await \_chatRepository.GetToTalkers(userid);

List<Guid> fromTalkers = await \_chatRepository.GetFromTalkers(userid);

fromTalkers.AddRange(toTalkers);

return fromTalkers.Distinct().ToList();

}

}

**Класс ChatRepository**

public class ChatRepository : IChatRepository

{

private DbConnection \_dbConnection;

public ChatRepository(IConfiguration config)

{

\_dbConnection = new DbConnection(config.GetConnectionString("DefaultConnection"));

}

public Task Send (Message model)

{

string expression = @"INSERT INTO messages (id, fromuserid, touserid, content, createdat)

VALUES (@id, @fromuserid, @touserid, @content, @createdat)";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(model.Id), model.Id),

new NpgsqlParameter(nameof(model.FromUserId), model.FromUserId),

new NpgsqlParameter(nameof(model.ToUserId), model.ToUserId),

new NpgsqlParameter(nameof(model.Content), model.Content),

new NpgsqlParameter(nameof(model.CreatedAt), model.CreatedAt),

};

return \_dbConnection.Add(expression, parameters);

}

public Task<List<Message>> GetMessages(Guid fromuserid, Guid touserid)

{

string expression = @"SELECT \* FROM messages WHERE (fromuserid = @fromuserid AND touserid = @touserid) OR (fromuserid = @touserid AND touserid = @fromuserid) ";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(fromuserid), fromuserid),

new NpgsqlParameter(nameof(touserid), touserid),

};

return \_dbConnection.GetList(expression, ChatConverter.MapToMessages, parameters);

}

public Task<List<Message>> GetAllMessages(Guid fromuserid)

{

string expression = @"SELECT \* FROM messages WHERE (fromuserid = @fromuserid OR touserid = @fromuserid) ";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(fromuserid), fromuserid),

};

return \_dbConnection.GetList(expression, ChatConverter.MapToMessages, parameters);

}

public Task<List<Guid>> GetTalkers(Guid userid)

{

string expression = @"SELECT touserid FROM messages WHERE ( fromuserid = @userid OR touserid = @userid ) GROUP BY touserid HAVING touserid <> @userid ";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(userid), userid),

};

return \_dbConnection.GetList(expression, ChatConverter.MapToGuid, parameters);

}

public Task<List<Guid>> GetToTalkers(Guid userid)

{

string expression = @"SELECT touserid FROM messages WHERE ( fromuserid = @userid OR touserid = @userid ) GROUP BY touserid HAVING touserid <> @userid ";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(userid), userid),

};

return \_dbConnection.GetList(expression, ChatConverter.MapToGuid, parameters);

}

public Task<List<Guid>> GetFromTalkers(Guid userid)

{

string expression = @"SELECT fromuserid FROM messages WHERE ( fromuserid = @userid OR touserid = @userid ) GROUP BY fromuserid HAVING fromuserid <> @userid ";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(userid), userid),

};

return \_dbConnection.GetList(expression, ChatConverter.MapFromGuid, parameters);

}

}

**Класс UserService**

public class UserService : IUserService

{

private readonly IUserRepository \_userRepository;

private IProductService \_productService;

public UserService(IUserRepository userRepository, IProductService productService)

{

\_userRepository = userRepository;

\_productService = productService;

}

public Task AddUser(string email, string password)

{

return \_userRepository.AddUser(email, password);

}

public async Task RegisterUser(RegistrationView registrationView)

{

await AddUser(registrationView.Email, registrationView.Password);

Guid userId = await GetUserIdByEmail(registrationView.Email);

UserProfileView userprofile = new UserProfileView();

userprofile.Id = Guid.NewGuid();

userprofile.Email = registrationView.Email;

userprofile.UserId = userId;

await AddUserProfile(userprofile);

}

public Task<User> GetUserByEmail(string email)

{

return \_userRepository.GetUserByEmail(email);

}

public async Task<Guid> GetUserIdByEmail(string email)

{

User user = await \_userRepository.GetUserByEmail(email);

return user.Id;

}

public async Task<string> GetUserEmailById(Guid userid)

{

User user = await \_userRepository.GetUserById(userid);

return user.Email;

}

public async Task<Guid> GetUserIdByProductId(Guid productid)

{

ProductView product = await \_productService.GetProduct(productid);

return product.UserId;

}

public Task<User> GetUserById(Guid userid)

{

return \_userRepository.GetUserById(userid);

}

public Task AddUserProfile(UserProfileView model)

{

return \_userRepository.SaveUserProfile(model);

}

public async Task SaveUserProfile(UserProfileView model, string email)

{

UserProfileView userProfile = await GetUserProfile(email);

model.Id = userProfile.Id;

if (model.Image != null)

{

model.Avatar = FileTools.GetBytes(model.Image);

}

else

{

model.Avatar = userProfile.Avatar;

}

await \_userRepository.SaveUserProfile(model);

}

public async Task<UserProfileView> GetUserProfile(Guid userid)

{

UserProfile userProfile = await \_userRepository.GetUserProfile(userid);

if (userProfile is null) return null;

return UserConverter.MapToUserProfileView(userProfile);

}

public async Task<UserProfileView> GetUserProfile(string email)

{

User user = await GetUserByEmail(email);

UserProfile userProfile = await \_userRepository.GetUserProfile(user.Id);

if (userProfile is null) return null;

return UserConverter.MapToUserProfileView(userProfile);

}

public async Task<byte[]> GetAvatar(Guid userid)

{

UserProfile userProfile = await \_userRepository.GetUserProfile(userid);

if (userProfile is null) return null;

return userProfile.Avatar;

}

public async Task<List<User>> GetUsers(Guid[] ids)

{

return await \_userRepository.GetUsers(ids);

}

public async Task<ProfileInfoView> GetUserProfileInfo(Guid userid, Guid currentUserId)

{

ProfileInfoView profileInfoView = new ProfileInfoView();

UserProfile userProfile = await \_userRepository.GetUserProfile(userid);

List<ProductView> productView = await \_productService.GetProductsViews(userid, currentUserId);

UserProfileView userProfileViewModel = UserConverter.MapToUserProfileView(userProfile);

profileInfoView.ProductViews = productView;

profileInfoView.UserProfileView = userProfileViewModel;

return profileInfoView;

}

}

**Класс UserRepository**

public class UserRepository : IUserRepository

{

private DbConnection \_dbConnection;

public UserRepository(IConfiguration config)

{

\_dbConnection = new DbConnection(config.GetConnectionString("DefaultConnection"));

}

public Task AddUser(string email, string password)

{

string expression = @"INSERT INTO users (id, email, password, createdat)

VALUES (@id, @email, @password, @createdat)";

Guid id = Guid.NewGuid();

DateTime createdAt = DateTime.Now;

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(id), id),

new NpgsqlParameter(nameof(email), email),

new NpgsqlParameter(nameof(password), password),

new NpgsqlParameter(nameof(createdAt), createdAt),

};

return \_dbConnection.Add(expression, parameters);

}

public Task<User> GetUserByEmail(string email)

{

string expression = @"SELECT \* FROM users WHERE email = @email";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(email), email),

};

return \_dbConnection.Get(expression, UserConverter.MapToUser, parameters);

}

public Task<User> GetUserById(Guid userid)

{

string expression = @"SELECT \* FROM users WHERE id = @userid";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(userid), userid),

};

return \_dbConnection.Get(expression, UserConverter.MapToUser, parameters);

}

public Task SaveUserProfile(UserProfileView model)

{

string expression = @"INSERT INTO usersprofile(id, userid, firstname, lastname, email, adress, phone, avatar)

VALUES(@id, @userid, @firstname, @lastname, @email, @adress, @phone, @avatar)

ON CONFLICT (id) DO UPDATE SET

firstname = EXCLUDED.firstname,

lastname = EXCLUDED.lastname,

adress = EXCLUDED.adress,

phone = EXCLUDED.phone,

avatar = EXCLUDED.avatar;";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(model.Id), model.Id),

new NpgsqlParameter(nameof(model.UserId), model.UserId),

new NpgsqlParameter(nameof(model.FirstName), model.FirstName),

new NpgsqlParameter(nameof(model.LastName), model.LastName),

new NpgsqlParameter(nameof(model.Email), model.Email),

new NpgsqlParameter(nameof(model.Adress), model.Adress),

new NpgsqlParameter(nameof(model.Phone), model.Phone),

new NpgsqlParameter(nameof(model.Avatar), model.Avatar),

};

return \_dbConnection.Add(expression, parameters);

}

public Task<UserProfile> GetUserProfile(Guid userid)

{

string expression = @"SELECT \* FROM usersprofile WHERE userid = @userid";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(userid), userid),

};

return \_dbConnection.Get(expression, UserConverter.MapToUserProfile, parameters);

}

public Task<List<User>> GetUsers(Guid[] ids)

{

string expression = @"SELECT \* FROM users WHERE id = ANY(@ids)";

NpgsqlParameter[] parameters = new[]

{

new NpgsqlParameter(nameof(ids), ids),

};

return \_dbConnection.GetList(expression, UserConverter.MapToUsers, parameters);

}

}