МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных

технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Web-приложение «Виртуальный холодильник»

Выполнил студент Пахолко Алена Степановна

(Ф.И.О.)

Руководитель проекта преп.-стаж. Левин М.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты преп.-стаж. Левин М.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер преп.-стаж. Левин М.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2019

**Содержание**

[Введение 3](#_Toc8987624)

[1 Постановка задачи 4](#_Toc8987625)

[1.1 Описание архитектуры 4](#_Toc8987626)

[2 Разработка архитектуры проекта 5](#_Toc8987627)

[2.1 Структура курсового проекта 5](#_Toc8987628)

[3 Разработка функциональной модели и модели данных ПС 8](#_Toc8987629)

[3.1 Структура модели данных 8](#_Toc8987630)

[3.2 Функциональная модель 9](#_Toc8987631)

[4 Руководство пользователя 10](#_Toc8987632)

[5 Тестирование 17](#_Toc8987633)

[Заключение 18](#_Toc8987634)

[Список литературы 19](#_Toc8987635)

[Приложение A 20](#_Toc8987636)

[Приложение Б 20](#_Toc8987637)

## **Введение**

В настоящее время существует большое количество программ, которые позволяют анализировать список продуктов, имеющихся прямо сейчас в вашем холодильнике.

В приложении решена задача автоматизации и организации отслеживания срока годности имеющихся продуктов, корзины покупок для похода в магазин, решения возможных трудностей, возникающих при совместном использовании холодильника в реальной жизни и другие.

Рядом с основной панелью, т. н. холодильником, где отображаются имеющиеся продукты и можно оценить, что срок годности подошёл к концу, расположено панель с сообщениями, которые могут оставлять все пользователи, которые имеют доступ к этому холодильнику. Так, можно оставлять сообщение о том, что тот или иной продукт нельзя использовать, так как он нужен для какого-либо рецепта как ингредиент.

Но разработка качественного тестового инструментария – длительный, трудоемкий и дорогостоящий процесс.

Для разработки клиентской части был выбран обширный фреймворк – Angular, предназначенный для создания клиентских приложений с пользовательским интерфейсом. Одной из ключевых особенностей Angular является то, что он использует в качестве основного языка программирования TypeScript.

Для разработки серверной части была выбрана платформа ASP.NET Core – универсальная кроссплатформенная платформа, которая может быть использована для создания приложений для устройств, облака и интернета вещей.

В соответствии с заданием курсового проекта следует разработать кроссплатформенное веб-приложение, построенное на REST-архитектуре, на тему «Виртуальный холодильник».

В ходе разработки было создано само приложение, были проведены интеграционные тесты, была составлена документация к API приложению при помощи Swagger, написана пояснительная записка, а также выполнена публикация и организованно покрытие API Unit-тестами.

Записка состоит из введения, 5 глав, заключения и приложений.

В первой главе рассматривается постановка задачи и общее описание архитектуры.

Во второй главе представлена разработка архитектуры приложения и структура курсового проекта.

В третьей главе описана разработка функциональной модели и модели данных программного средства.

Четвёртая глава состоит из руководства пользователя.

В пятой главе описано тестирование приложения.

В заключении описывается результат работы над курсовым проектом, выполненные цели.

В приложении содержится листинг кода некоторых классов и моделей.

# **Постановка задачи**

При реализации курсового проекта использовалась платформа ASP.NET Core для разработки back-end.

Фреймворк Angular и bootstrap — свободный набор инструментов для создания сайтов и веб-приложений, который включает в себя HTML и CSS, были выбраны реализации front-end.

Microsoft SQL Server взят в качестве базы данных.

Курсовой проект построен на многоуровневой архитектуре. Это клиент-серверная архитектура, в которой разделяются функции представления, обработки и хранения данных.

В курсовом проекте была разработана четырёхуровневая архитектура, содержащая следующие уровни:

1. Уровень «Модель» – описание таблиц в базе данных;
2. Уровень «Доступа к данным» – описание связей между моделями в базе данных и отображение данных в объекты приложения;
3. Уровень «Бизнес-логики» – содержит набор компонентов, которые отвечают за обработку полученных от уровня представлений данных, реализует всю необходимую логику приложения, все вычисления, взаимодействует с базой данных и передаёт уровню представления результат обработки;
4. Уровень «Представления» – включает компоненты пользовательского интерфейса, механизм получения ввода от пользователя. Содержит стили, страницы html, модели представлений, контроллеры.

Данная архитектура основана на следующих принципах:

* Проектирование чётко устанавливает разграничение функций между уровнями;
* Нижние уровни независимы от верхних уровней;
* Верхние уровни вызывают функции нижних уровней, но при этом взаимодействуют только соседние уровни иерархии.

Каждый запрос клиента проходит как минимум через 3 уровня – «Аутентификация», «Авторизация» и «Контроллер», после чего клиент получает ответ.

Чтобы использовать функционал приложения в полной мере, пользователь должен пройти регистрацию и войти в личный кабинет.

# **Разработка архитектуры проекта**

Физическое разделение уровней улучшает масштабируемость и устойчивость. В более высоком слое могут использоваться службы из более низкого слоя, но не наоборот.

В решении курсового проекта была использована n-layer архитектура (рисунок 2.1).

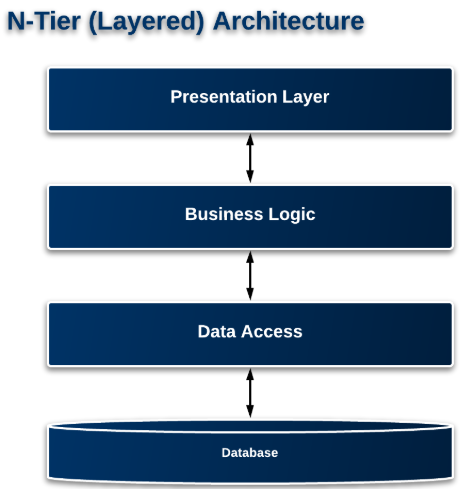


Рисунок 2.1 – Общая схема n-layer архитектуры

Схема данного курсового проекта представлена ниже (рисунок 2.2).

Контекст

Модели

Сервисы

Контроллеры

Авторизация

Аутентификация

Рисунок 2.2 – Структура проекта

В папке FK.Models находятся модели для описания сущностей базы данных (рисунок 2.3).

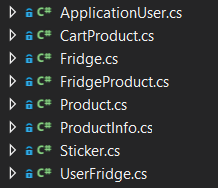


Рисунок 2.3 – Папка FK.Models

В папке FK.DAL находятся контекст базы данных, инициализатор, а также классы для реализации паттернов Repository и UnitOfWork (рисунок 2.4).

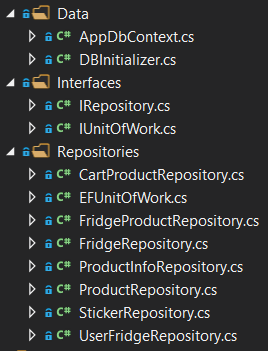


Рисунок 2.4 – Папка FK.DAL

В папке FK.BLL находятся модели для взаимодействия слоев, интерфейсы сервисов и их реализации (рисунок 2.5).

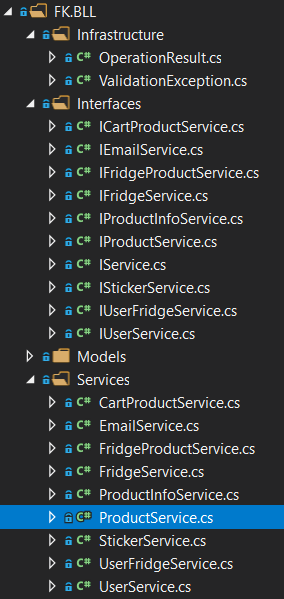


Рисунок 2.5 – Папка FK.BLL

В проекте CoreWebApi находятся API-контроллеры, а также классы для настройки приложения (рисунок 2.6).

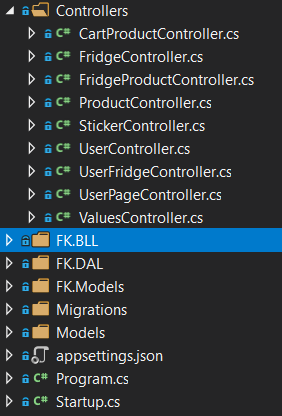


Рисунок 2.6 – Проект CoreWebApi

# **Разработка функциональной модели и модели данных ПС**

Для создания, хранения и управления данными, была использованная реляционная СУБД Microsoft SQL Server.

## **Структура модели данных**

Ниже (рисунок 3.1) представлена модель базы данных (таблицы, отмеченные на них первичных и вторичных ключей) курсового проекта. База данных была подвергнута процедуре нормализации данных.

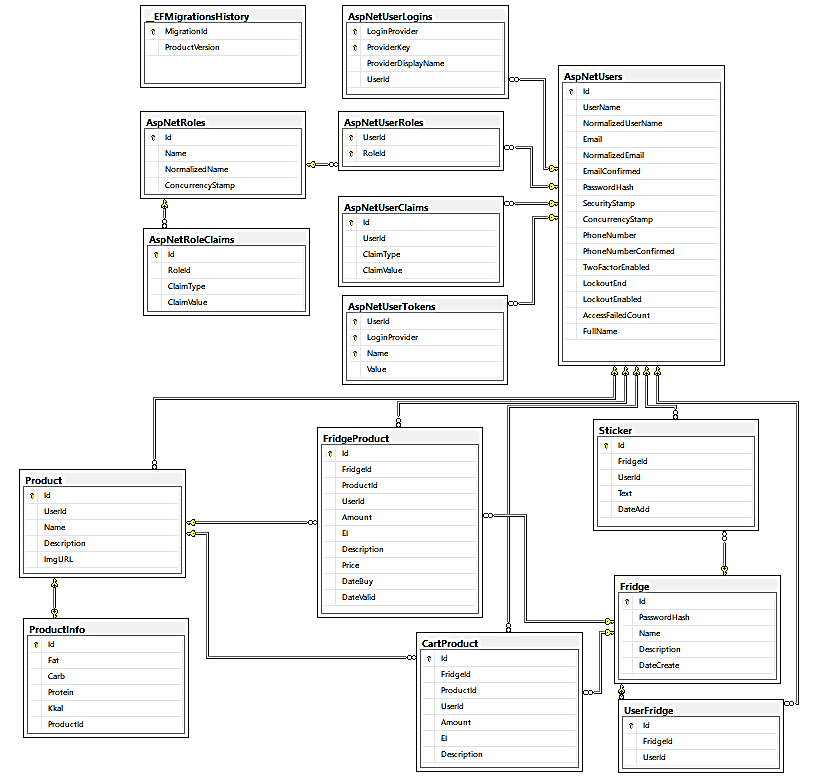


Рисунок 3.1 – Общая схема базы данных

Таблица [AspNetUsers] для работы с пользователями приложения. Является связующей для технологии Identity и других специфичных таблиц.

Таблица [Fridge] для работы с холодильниками пользователей. Так, даже не имея владельца, к холодильнику можно получить доступ по его имени и паролю.

Таблица [UserFridge] для контроля связей между пользователями и холодильниками. Так, при вводе правильной пары имя-пароль для доступа к холодильнику, создается соответствующая запись.

Таблицы [Product] и [ProductInfo] для работы с продуктами приложения. Так, основной является [Product], а дополнительные уточняющие параметры заносятся в таблицу [ProductInfo].

Таблица [FridgeProduct] для работы с продуктами, которые добавлены в тот или иной холодильник. Содержит основную полезную информацию для разделения продуктов в одном холодильнике между разными пользователями.

Таблица [CartProduct] для работы с пользовательской корзиной покупок. Так, все пользователи, у которых есть доступ к холодильнику, могут видеть и редактировать список общей корзины покупок для похода в магазин.

Таблица [Sticker] для работы с записками на холодильниках, созданные тем или иным пользователем на некотором холодильнике. Так, все пользователи, у которых есть доступ к холодильнику, могут видеть и редактировать список записок на холодильнике.

## **Функциональная модель**

В данном приложении реализованы два вида пользователей: администратор приложения и пользователь.

Администратор имеет возможность просматривать отчет о всех пользователях, их холодильниках, а также продуктах. Так же реализована возможность выполнения действий типа CRUD над объектами без ограничений.

Пользователь имеет право просматривать отчет о доступных ему холодильниках, а также продуктах, созданных администратор и самим пользователем. Так же реализована возможность выполнения действий типа CRUD над теми объектами, на которые у них есть право доступа.

Для WebAPI механизма авторизации используются JWT-токены. Это веб-стандарт, который определяет способ передачи данных о пользователе в формате JSON в зашифрованном виде.

В случае запроса страницы, к которой нет права доступа, пользователю будет выводиться страница-заглушка.

В случае попытки обращения анонимного пользователя к API приложения, его перенаправит на страницу логина с параметром callback, в который будет занесён предыдущий запрос. После успешной аутентификации пользователь будет перенаправлен на первоначальный запрос.

Все запросы к контроллерам приложения выполнены в стиле REST.

# **Руководство пользователя**

Для получения полного доступа к функционалу сайта, пользователю необходимо зарегистрироваться либо войти в аккаунт (рисунки 4.1 – 4.2).

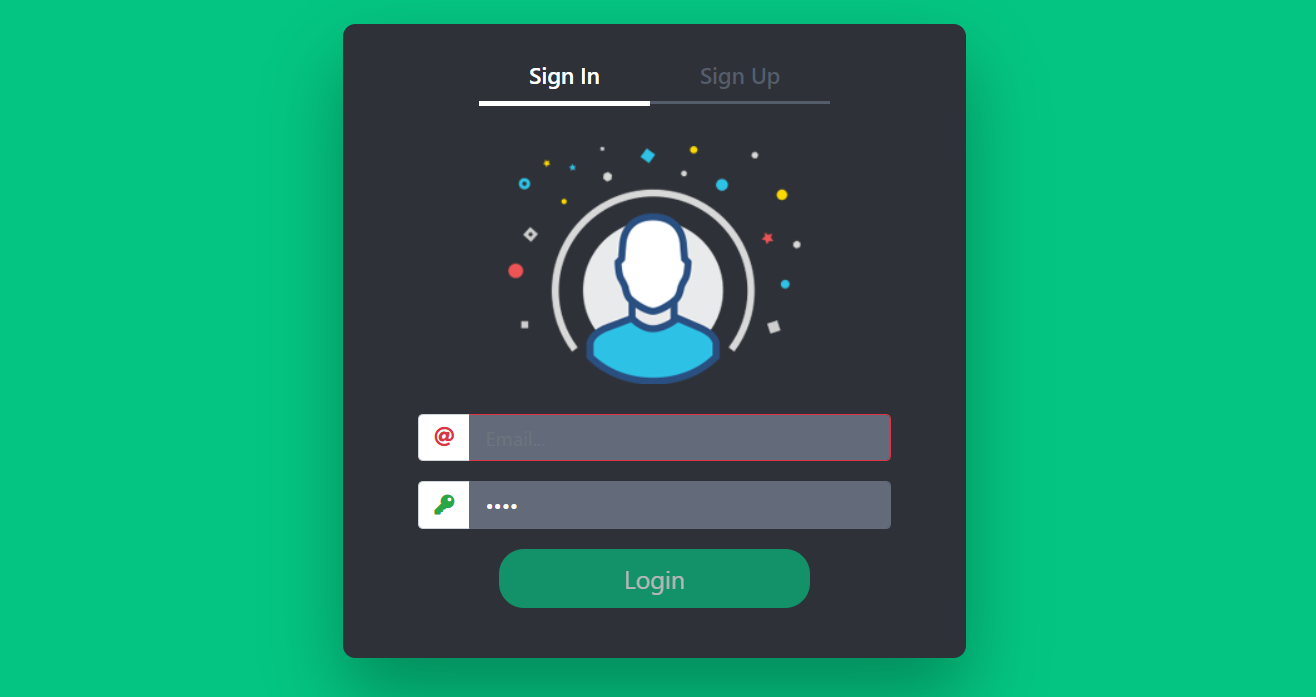


Рисунок 4.1 ­­– Страница входа в аккаунт

Выше представлена базовая форма аутентификации, на которой пользователю необходимо ввести логин и пароль. В случае ввода некорректной информации при регистрации также будут выданы подсказки, указывающие на поля, которые надо перезаполнить. Кнопка регистрации недоступна, пока не будут введены все данные.

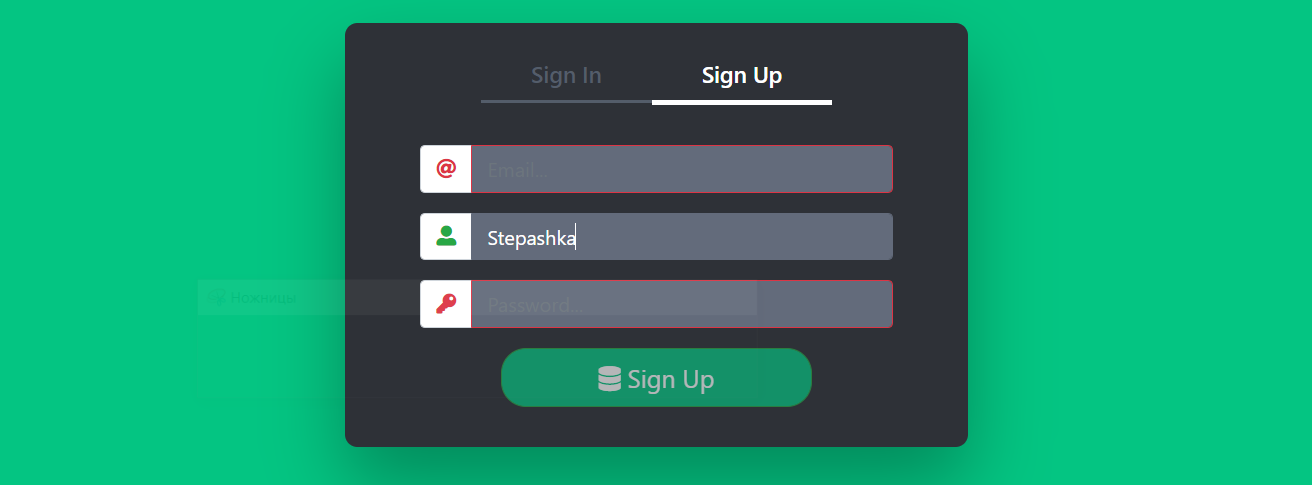


Рисунок 4.2 – Страница регистрации

При успешной аутентификации и авторизации пользователя перенаправляет на основную стартовую страницу приложения, работа с которой будет проводиться в течении всего пользования (рисунок 4.3).

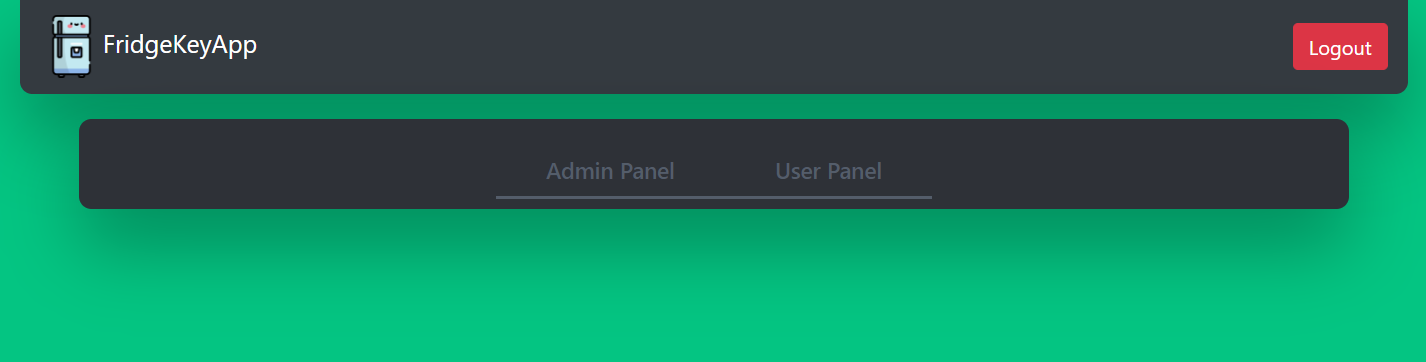


Рисунок 4.3 – Стартовая страница приложения

Но для разных пользователей отображается разное меню. Пример меню обычного пользователя (рисунок 4.4).

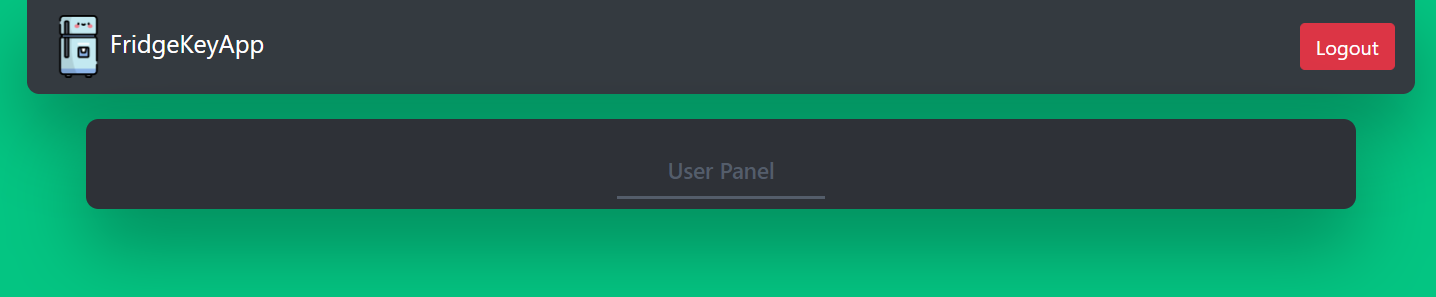


Рисунок 4.4 – Вкладка меню пользователя

Вкладка «FridgeKey» навигационной панели предназначена для возврата на главную страницу, которая была показана выше.

При помощи вкладки Logout пользователь может выйти из приложения (сопровождается удалением объекта JWT-токена из локального хранилища.

Далее предлагается выбор активной панели (администратор или пользователь), где можно работать с панелями «My Products», «My Page», «My Fridges» (рисунок 4.5).

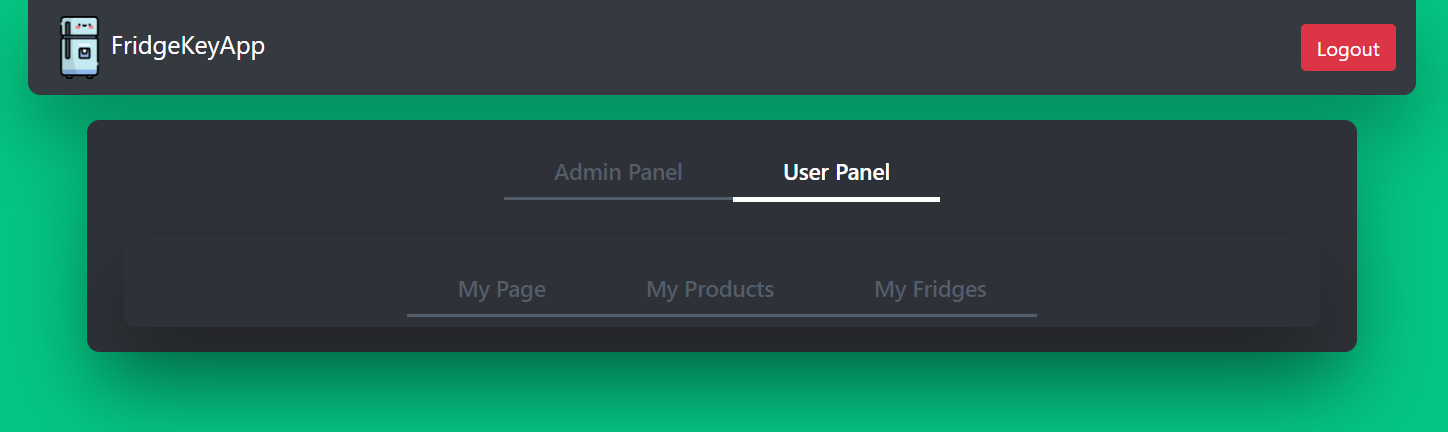


Рисунок 4.5 – Выбор активной панели

Во вкладке «My Page» отображается личная информация пользователя, как введенная самим пользователем, так и сгенерированная приложением (рисунок 4.6).

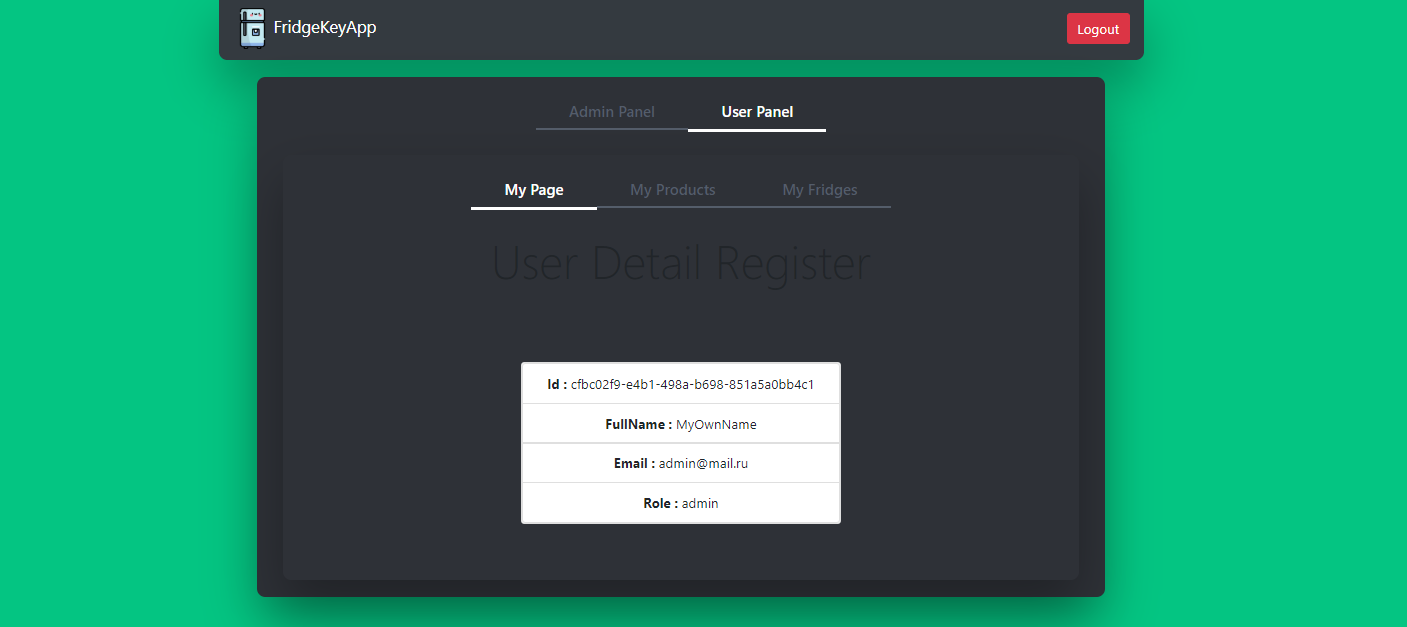


Рисунок 4.6 – Информация о пользователе

Во вкладке «Product» содержится две панели – создание/редактирование продукта и список доступных продуктов (рисунок 4.7). Все поля имеют определенный набор правил валидации, при нарушении которых пользователь будет уведомлён об этом.

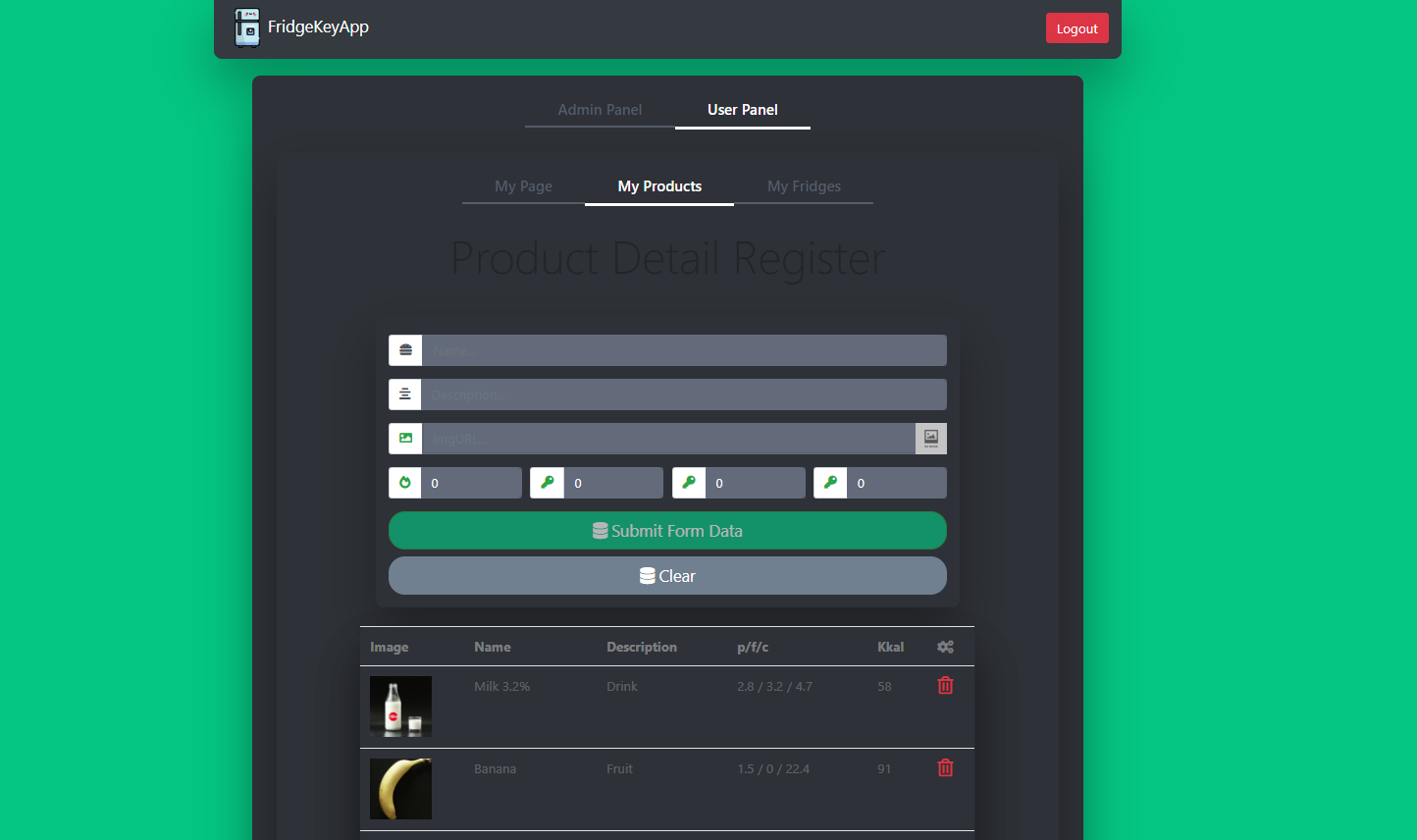


Рисунок 4.7 – Доступные продукты

Во вкладке «Fridge» содержится две панели – добавление/создание/редактирование холодильника и список доступных холодильников (рисунок 4.8). Все поля так же обязательны для заполнения, а также имеют определенный набор правил валидации, при нарушении которых пользователь будет уведомлён об этом. Следует отметить, что при открытии холодильника (по нажатию соответствующей кнопки), идентификатор самого холодильника сохраняет в локальном хранилище, что позволяет активировать панель «Open Fridge».

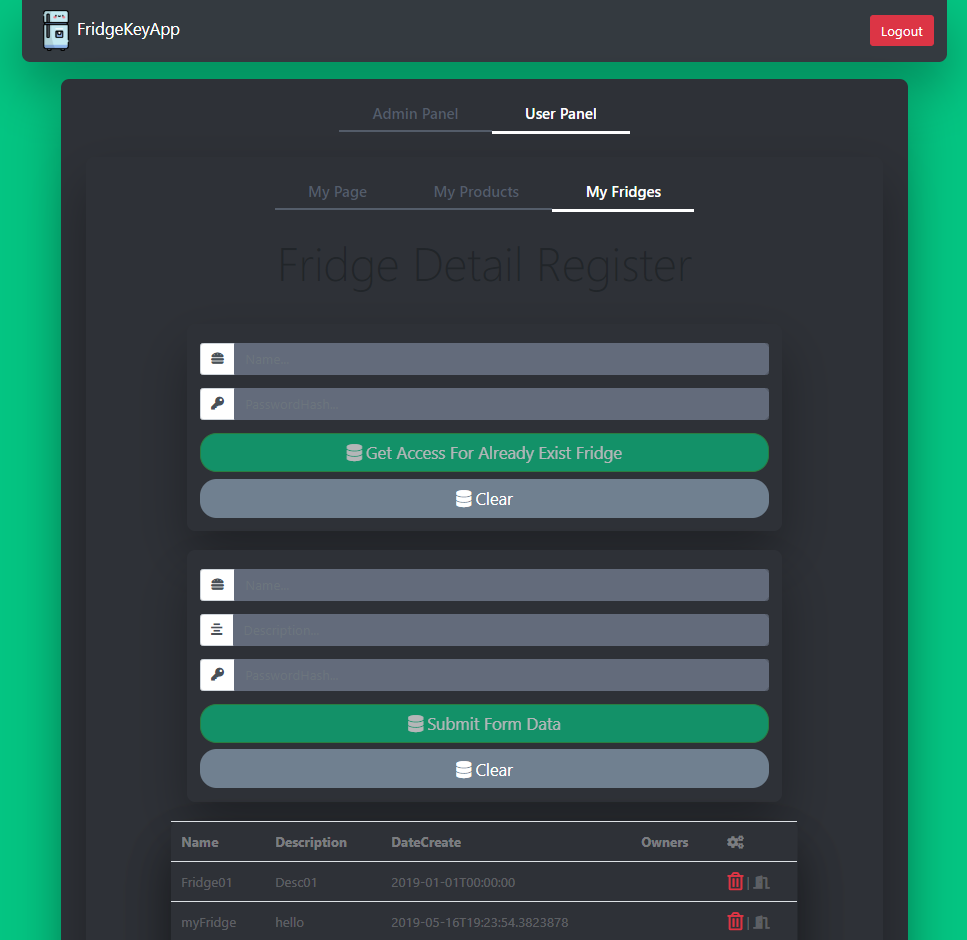


Рисунок 4.8 – Доступные холодильники

Перейдем к панели администратора. Во вкладке «User» отображается личная информация всех пользователей приложения, включая такие поля, как идентификатор пользователя, его почтовый ящик, имя в приложении и роль самого пользователя (рисунок 4.9).

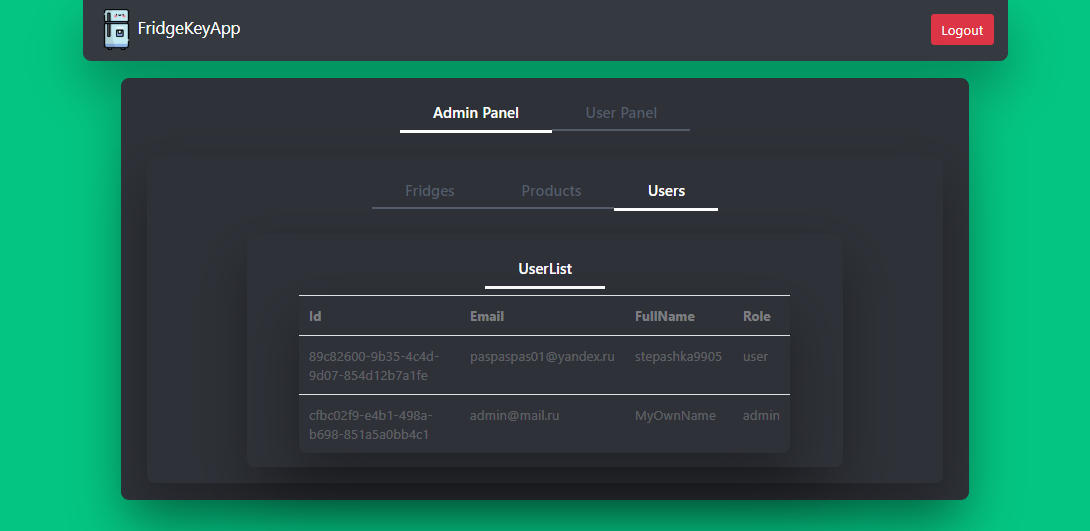


Рисунок 4.9 – Все пользователи

Во вкладке «Product» отображается список всех существующих продуктов приложения с возможностью удаления каждого (рисунок 4.10).

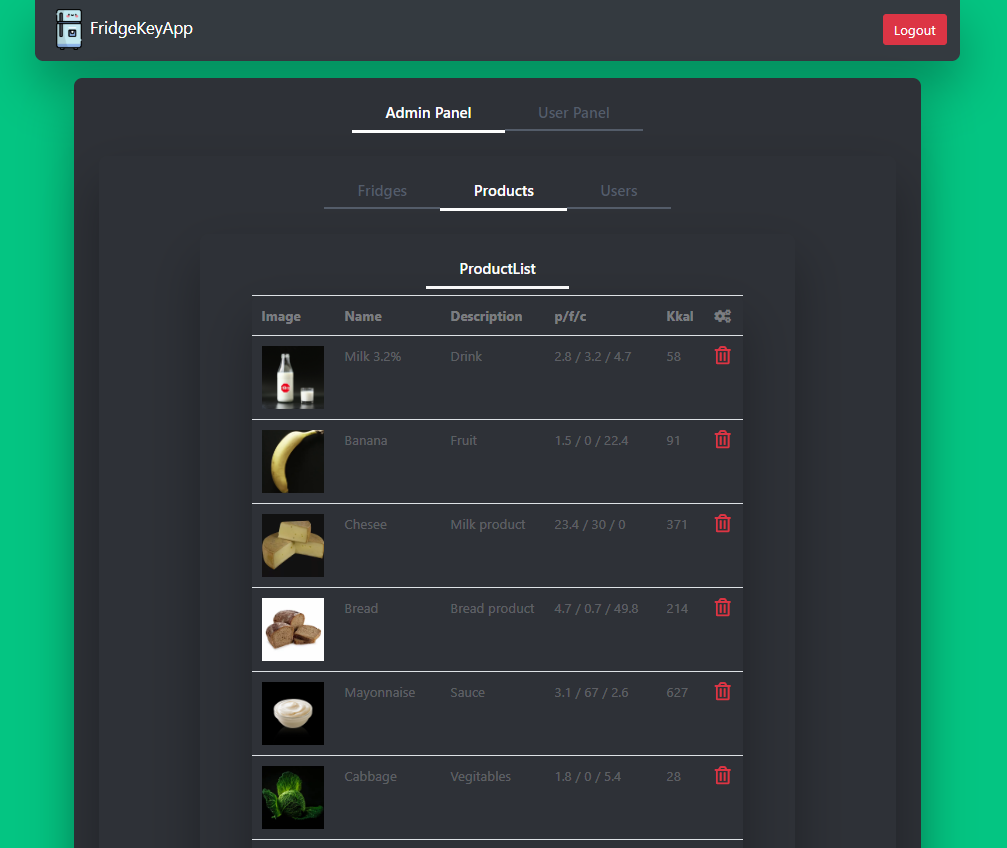


Рисунок 4.10 – Все продукты

Во вкладке «Fridge» отображается список всех существующих холодильников приложения с возможностью удаления (рисунок 4.11).

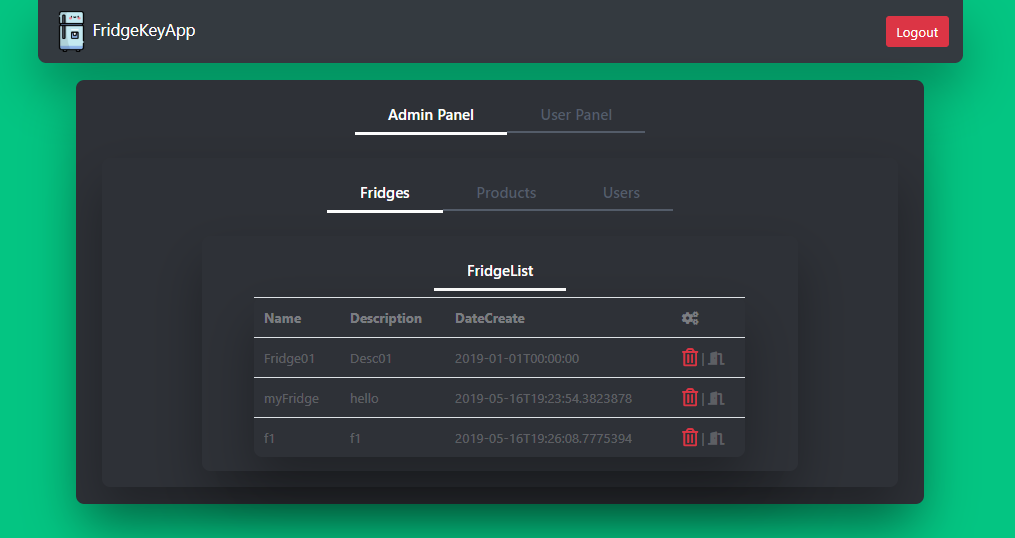


Рисунок 4.11 – Все холодильники

Перейдем к панели открытого холодильника, которая становится активна после нажатия соответствующей кнопки в списке холодильников. Во вкладке «Products» отображается список всех продуктов в этом холодильнике, с возможностью создания/редактирования/удаления каждого из них (рисунок 4.12).

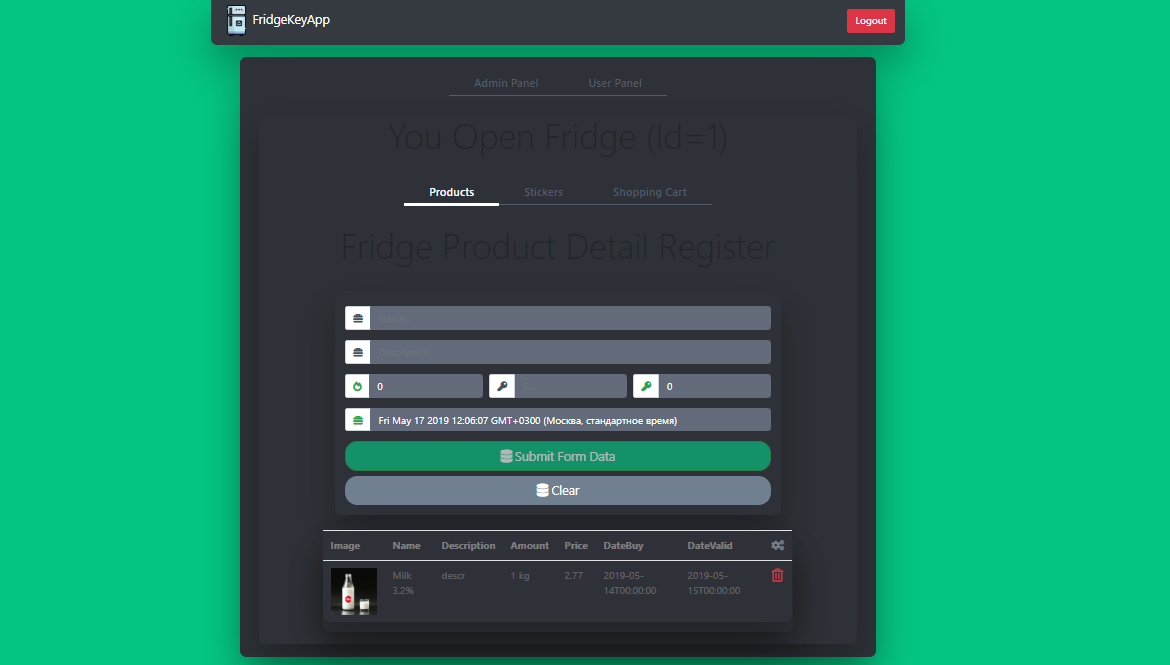


Рисунок 4.12 – Продукты в холодильнике

Во вкладке «Sticker» отображается список всех записок на этом холодильнике с возможностью удаления каждого (рисунок 4.13).

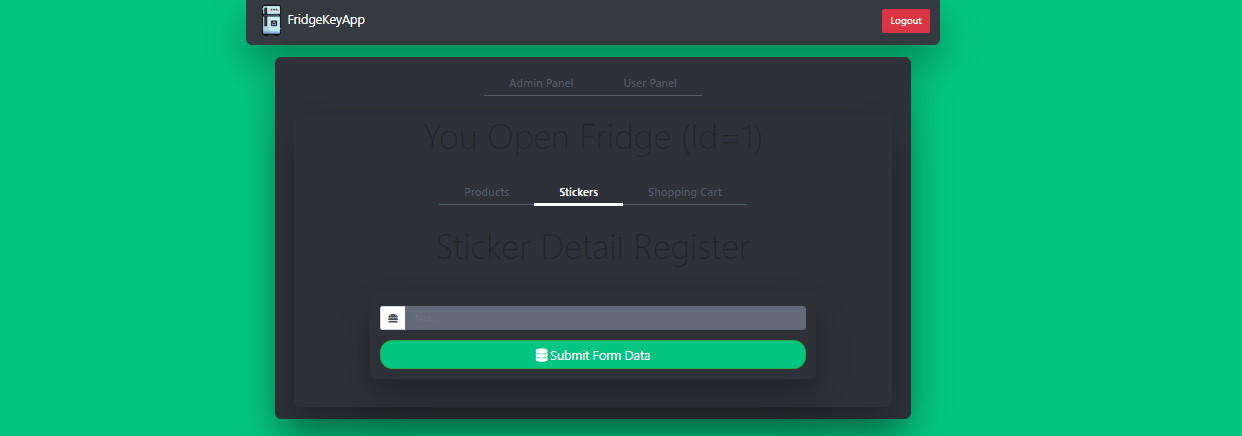


Рисунок 4.13 – Записки на холодильнике

Во вкладке «Shopping Cart» отображается список всех продуктов в корзине покупок для похода в магазин (рисунок 4.14).

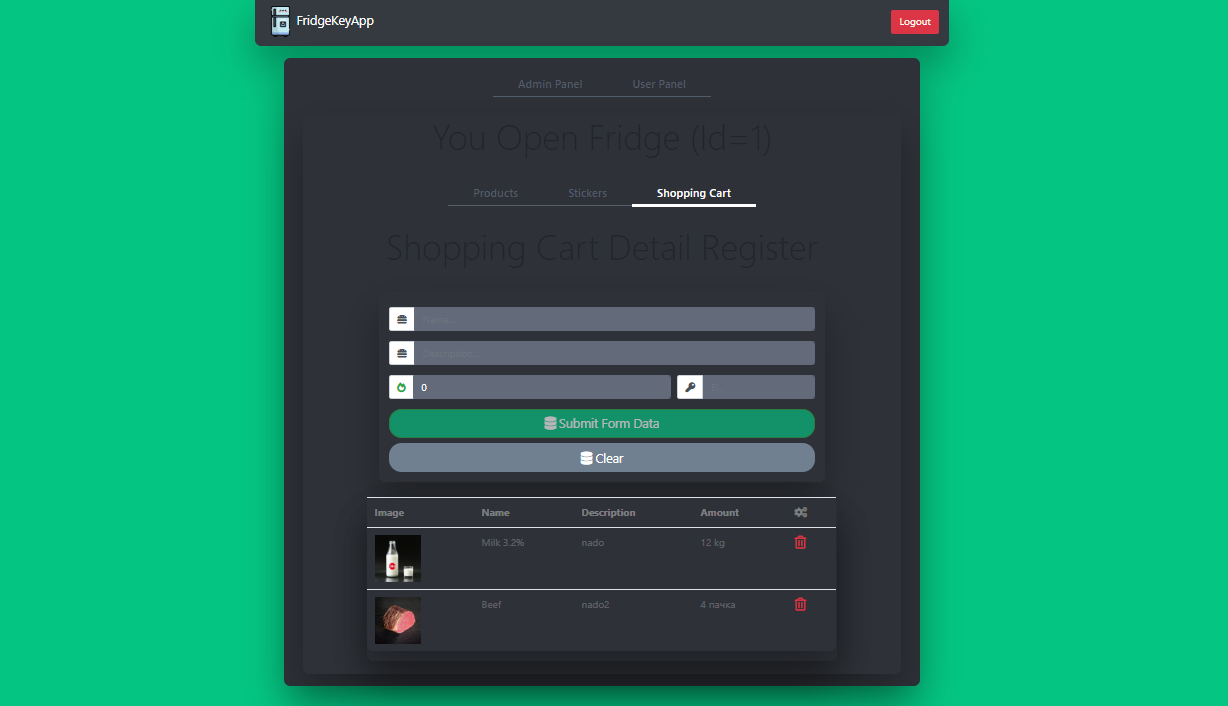


Рисунок 4.14 – Корзина покупок

Так, с помощью переключения активных панелей и выбора открываемого компонента, реализована навигация в приложении.

# **Тестирование**

На основную логику проекта были разработаны и успешно выполнены интеграционные тесты. Существуют модульные и интеграционные тесты. Модульное тестирование является способом проверки поведения отдельных классов в изоляции от остальной части приложения.

Интеграционные тесты представляют собой проверку поведения множества компонентов, работающих совместно, вплоть до целого приложения. Ценность интеграционного тестирования заключается в том, что оно может моделировать взаимодействие пользователя с пользовательским интерфейсом и покрывать весь стек технологий, применяемых в приложении, включая веб-сервер и базу банных. Они дают уверенность, что ваша программа работает, как задумано. Такие тесты можно запускать многократно. Успешное выполнение тестов покажет разработчику, что его изменения не сломали ничего, что ломать не планировалось.

Провалившийся тест позволит обнаружить ошибки, которые ломают его поведение. Исследование ошибки и сравнение ожидаемого результата с полученным даёт возможность понять, где возникла ошибка. В данном курсовом проекте все тесты завершились успехом (рисунок 5.1).

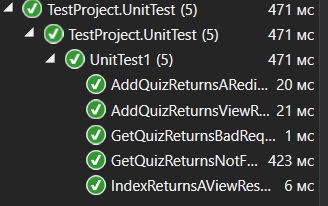


Рисунок 5.1 – Результаты выполнения тестов

В проекте сделана валидация на случай неправильного поведения пользователя, к примеру некорректный ввод данных или добавления продукта с пустыми полями. В 4 главе приведены примеры валидации на клиентской стороне.

В данном курсовом проекте все поля ввода проходят валидацию, где в случае некорректного ввода, может произойти ошибка или непредвиденное поведение программы. Валидация приложения – это один из основных гарантов надёжности приложения и всякий программист должен предусмотреть и предотвратить непредвиденное поведение пользователя.

# **Заключение**

В ходе выполнения курсовой работы было разработано приложение, являющееся помощником в автоматизации и контроле использования реального холодильника несколькими пользователи, контроле продуктов и корзины покупок. Во время выполнения данной курсовой работы было изучено немало теоретического материала, а также просмотрено и разобрано большое количество уже готовых решений тех или иных задач. Разработка приложения­ – трудоемкое занятие, выполнение которого требует всестороннего знания той предметной области, к которой относится тема разрабатываемого приложения. Программная система имеет простой, интуитивно понятный интерфейс пользователя. Также была разработана возможность изменения данных пользователя и его статистики.

Данное программное средство использует Angular, bootstrap и fontawesome для реализации front-end части и платформа ASP.NET Core для разработки back-end. В качестве базы данных использовалась MSSql – реляционная база данных. На back-end реализована n-layer архитектура.

Для удалённого доступа пользователей приложение было опубликовано на Azure. Ссылка перехода: [http://fridgekey.azurewebsites.net](http://fridgekey.azurewebsites.net/)/.

Для спецификации RESTful API был выбран фреймворк Swagger. При отображении документации, существует форма для отправки тестового запроса со своими параметрами.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

# **Список литературы**

1. Цыганенко, Н. П. Курс лекций по предмету «Программирование серверных кроссплатформенных приложений» - 2019.
2. N-layer архитектура [Электронный ресурс] / Github – Режим доступа : <https://accetone.github.io/cwp/10/#/> - Дата доступа : 16.05.2019.
3. MSDN сеть разработчиков в Microsoft [Электронный ресурс] / Режим доступа: <http://msdn.microsoft.com/library/rus/> – Дата доступа: 29.04.2019
4. METANIT.COM Сайт о программировании[Электронный ресурс] / Режим доступа: <https://metanit.com> Дата доступа 15.04.2019
5. apidoc [Электронный ресурс] / npm – Режим доступа : <https://www.npmjs.com/package/apidoc> - Дата доступа : 16.05.2019.
6. PROFESSORWEB.RU Сайт о программировании[Электронный ресурс] / Режим доступа: [http://professorweb.ru](http://professorweb.ru/) Дата доступа 15.04.2019
7. ASP.NET Core [Электронный ресурс]. Режим доступа: <https://www.asp.net> - Дата доступа : 10.05.2019.
8. Stackoverflow [Электронный ресурс]. Режим доступа: <https://ru.stackoverflow.com/>- Дата доступа : 16.05.2019.
9. Habr [Электронный ресурс]. Режим доступа: <https://habr.com/ru/post/312226/>- Дата доступа : 28.05.2019.

# **Приложение A**

Листинг 1. Контекст базы данных приложения

|  |
| --- |
| namespace FK.DAL  {  public class AppDbContext : IdentityDbContext<ApplicationUser>  {  public AppDbContext(DbContextOptions<AppDbContext> options)  : base(options)  {  }  public DbSet<Sticker> Stickers { get; set; }  public DbSet<Fridge> Fridges { get; set; }  public DbSet<FridgeProduct> FridgeProducts { get; set; }  public DbSet<CartProduct> CartProducts { get; set; }  public DbSet<Product> Products { get; set; }  public DbSet<ProductInfo> ProductInfos { get; set; }  public DbSet<UserFridge> UserFridges { get; set; }  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  {  optionsBuilder  .UseLazyLoadingProxies();  }  }  } |

# **Приложение Б**

Листинг 2. Модели приложения

|  |  |
| --- | --- |
| public class ApplicationUser : IdentityUser  {  public ApplicationUser()  {  Stickers = new HashSet<Sticker>();  FridgeProducts = new HashSet<FridgeProduct>();  Products = new HashSet<Product>();  UserFridges = new HashSet<UserFridge>();  }  [StringLength(450)]  public string FullName { get; set; }  public virtual ICollection<Sticker> Stickers { get; set; }  public virtual ICollection<FridgeProduct> FridgeProducts { get; set; }    public virtual ICollection<Product> Products { get; set; }  public virtual ICollection<UserFridge> UserFridges { get; set; }  }  [Table("CartProduct")]  public partial class CartProduct  {  [Key]  public int Id { get; set; }  [Required]  public int FridgeId { get; set; }  [Required]  public int ProductId { get; set; }  [StringLength(450)]  public virtual string UserId { get; set; }  [Required]  public double Amount { get; set; }  [Required]  [StringLength(50)]  public string EI { get; set; }  [StringLength(255)]  public string Description { get; set; }  public virtual Fridge Fridge { get; set; }  public virtual Product Product { get; set; }  public virtual ApplicationUser User { get; set; }  }  [Table("Fridge")]  public partial class Fridge  {  public Fridge()  {  FridgeProducts = new HashSet<FridgeProduct>();  UserFridges = new HashSet<UserFridge>();  Stickers = new HashSet<Sticker>();  }  [Key]  public int Id { get; set; }  [Required]  [StringLength(50)]  public string PasswordHash { get; set; }  [Required]  [StringLength(50)]  public string Name { get; set; }  [StringLength(50)]  public string Description { get; set; }  public DateTime? DateCreate { get; set; }  public virtual ICollection<Sticker> Stickers { get; set; }  public virtual ICollection<FridgeProduct> FridgeProducts { get; set; }    public virtual ICollection<UserFridge> UserFridges { get; set; }  }  [Table("Sticker")]  public partial class Sticker  {  [Key]  public int Id { get; set; }  [Required]  public int FridgeId { get; set; }  [StringLength(450)]  public virtual string UserId { get; set; }  [Required]  [StringLength(450)]  public string Text { get; set; }  [Required]  [DataType(DataType.DateTime)]  public DateTime DateAdd { get; set; }  public virtual ApplicationUser User { get; set; }  public virtual Fridge Fridge { get; set; }  } | [Table("FridgeProduct")]  public partial class FridgeProduct  {  [Key]  public int Id { get; set; }  [Required]  public int FridgeId { get; set; }  [Required]  public int ProductId { get; set; }  [StringLength(450)]  public virtual string UserId { get; set; }  [Required]  public double Amount { get; set; }  [Required]  [StringLength(50)]  public string EI { get; set; }  [StringLength(255)]  public string Description { get; set; }  public decimal? Price { get; set; }  public DateTime? DateBuy { get; set; }  public DateTime? DateValid { get; set; }  public virtual Fridge Fridge { get; set; }  public virtual Product Product { get; set; }  public virtual ApplicationUser User { get; set; }  }  [Table("Product")]  public partial class Product  {  public Product()  {  FridgeProducts = new HashSet<FridgeProduct>();  }    [Key]  public int Id { get; set; }  [StringLength(450)]  public virtual string UserId { get; set; }  [Required]  [StringLength(450)]  public string Name { get; set; }    [StringLength(450)]  public string Description { get; set; }  [Required]  [StringLength(255)]  public string ImgURL { get; set; }  public virtual ICollection<FridgeProduct> FridgeProducts { get; set; }  public virtual ProductInfo ProdInfo { get; set; }  public virtual ApplicationUser User { get; set; }  }  [Table("Product")]  public partial class Product  {  public Product()  {  FridgeProducts = new HashSet<FridgeProduct>();  }    [Key]  public int Id { get; set; }  [StringLength(450)]  public virtual string UserId { get; set; }  [Required]  [StringLength(450)]  public string Name { get; set; }    [StringLength(450)]  public string Description { get; set; }  [Required]  [StringLength(255)]  public string ImgURL { get; set; }  public virtual ICollection<FridgeProduct> FridgeProducts { get; set; }  public virtual ProductInfo ProdInfo { get; set; }  public virtual ApplicationUser User { get; set; }  }  [Table("UserFridge")]  public partial class UserFridge  {  [Key]  public int Id { get; set; }  [Required]  public int FridgeId { get; set; }  [StringLength(450)]  public virtual string UserId { get; set; }  public virtual Fridge Fridge { get; set; }  public virtual ApplicationUser User { get; set; }  } |