Министерство образования, науки и молодежной политики
Краснодарского края
Государственное бюджетное профессиональное образовательное учреждение
Краснодарского края «Ейский полипрофильный колледж»

РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

ЛАБОРАТОРНАЯ РАБОТА №17

по теме:

Приложение Windows Forms, использующее Entity Framework Core (EF Core) для соединения с базой данных SQLite

Выполнил:
студент ЕПК, группа
ФИО

Проверил:
преподаватель дисциплины
«Разработка программных
модулей»
Фомин А. Т.

Ейск 2024

1 Цель работы

Создание простого приложения Windows Forms, с поддержкой базы данных SQLite. Приложение использует ***Entity Framework Core (EF Core)*** для загрузки данных из базы данных, отслеживание внесенных изменений данных и сохранение этих изменений в DB.

Примечание. ***Microsoft.EntityFrameworkCore.Sqlite*** *— это пакет поставщика базы данных для использования EF Core с базой данных SQLite. Аналогичные пакеты доступны для других систем баз данных. Установка пакета поставщика базы данных автоматически приводит все зависимости, необходимые для использования EF Core с этой системой базы данных. Сюда входит базовый* ***пакет Microsoft.EntityFrameworkCore*** .

2 Теоретические сведения

Создание приложения

1. Запустите Visual Studio
2. На начальном экране выберите **Создать проект**.
3. Выберите **приложение** Windows Forms (Майкрософт) и нажмите кнопку **"Далее"**.

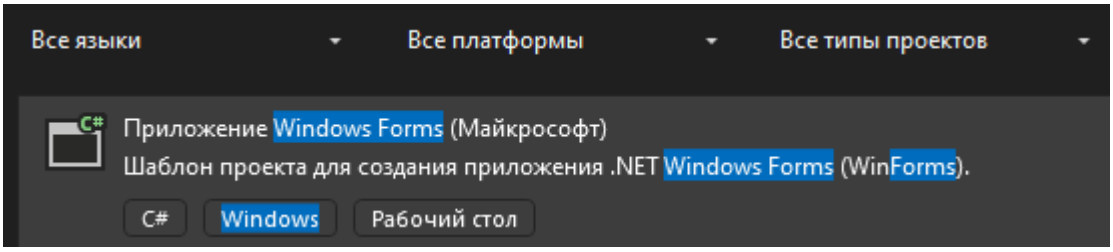

Рис. 1. Выбор типа проекта

4. На следующем экране укажите имя проекта, например **Lab17** и нажмите кнопку **"Далее"**.
5. На следующем экране выберите используемую версию .NET. Этот пример был создан с помощью .NET 8. Выберите **Создать**.

Установка пакетов NuGet EF Core

6. Щелкните правой кнопкой мыши решение и выберите **Управление пакетами NuGet для решения;**
7. Перейдите на вкладку "Обзор" и найдите **"Microsoft.EntityFrameworkCore.Sqlite"**.
8. Выберите пакет **Microsoft.EntityFrameworkCore.Sqlite**.

9. Проверьте проект **Lab17** в правой области.
10. Выберите последнюю версию.
11. Щелкните **Установить**.

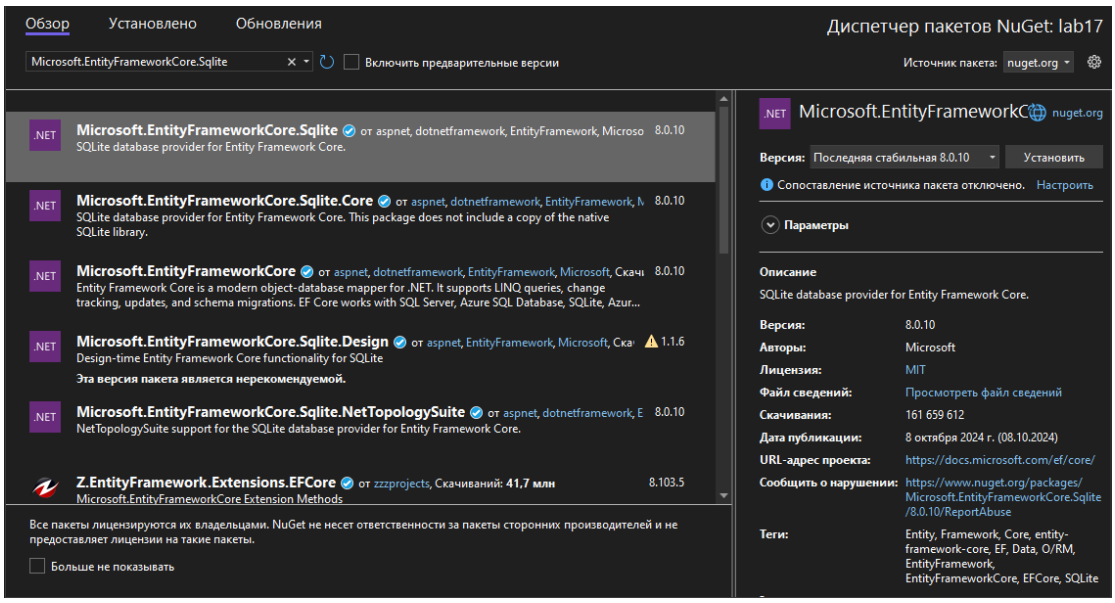

Рис. 2. Выбор нужного пакета

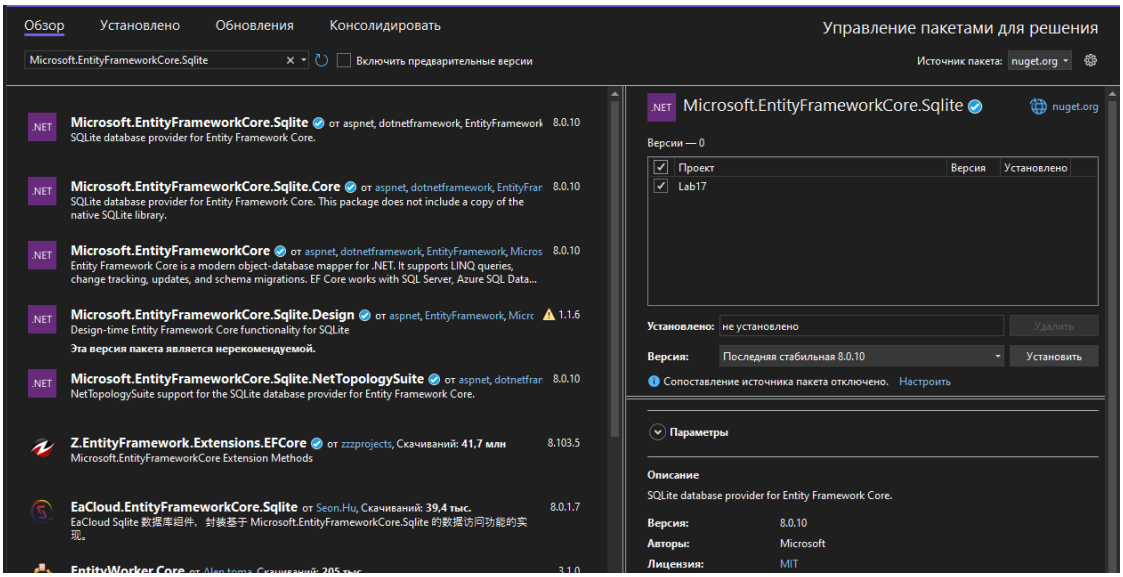

Рис. 3. Установка Microsoft.EntityFrameworkCore.Sqlite

Создание модели данных DB

12. Щелкните проект правой кнопкой мыши в обозревателе решений и нажмите кнопку "Добавить", а затем "Класс", чтобы добавить новый класс. (Рис. 4). Используйте имя файла Product.cs. Измените код для класса следующим образом:

Листинг 1. Файл Product.cs

```
using System.ComponentModel;

namespace GetStartedWinForms;

public class Product
{
    public int ProductId { get; set; }
    public string? Name { get; set; }
    public int CategoryId { get; set; }
    public virtual Category Category { get; set; } = null!;
}
```

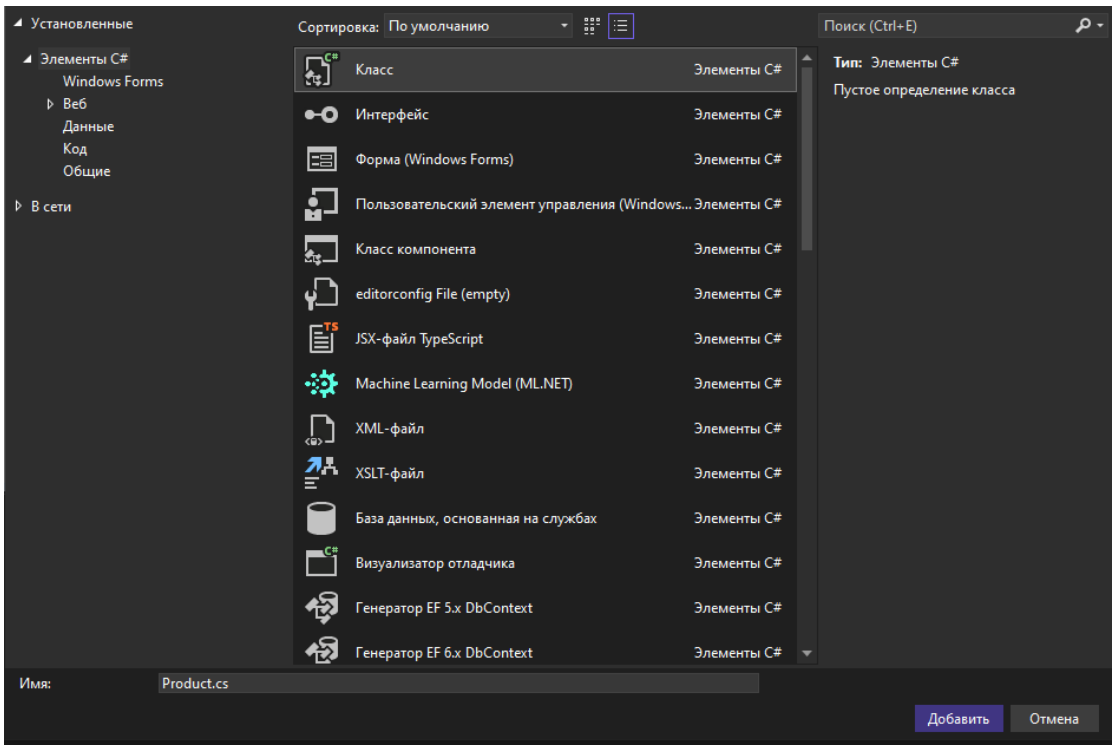

Рис. 4. Создание нового класса Product.cs

13. Аналогичные действия выполните по созданию класса Category.cs используя следующий код:

Листинг 2. Файл Category.cs

```
using Microsoft.EntityFrameworkCore.ChangeTracking;

namespace GetStartedWinForms;

```

```
5.  public class Category
6.  {
7.      public int CategoryId { get; set; }
8.      public string? Name { get; set; }
9.      public virtual ObservableCollectionListSource<Product>
    Products { get; } = new();
10. }
```

Примечание. Свойство Products Category класса и Category свойства класса Product называются навигациями. В EF Core навигации определяют связь между двумя типами сущностей. В этом случае навигация Product.Category ссылается на категорию, к которой принадлежит данный продукт. Аналогичным образом навигация Category.Products по коллекции содержит все продукты для данной категории.

Определение DbContext

14. В EF Core класс, производный от DbContext, используется для настройки типов сущностей в модели и выступает в качестве сеанса для взаимодействия с базой данных. В самом простом случае DbContext класс:
    - Содержит свойства DbSet для каждого типа сущности в модели.
    - Переопределяет метод OnConfiguring для настройки поставщика базы данных и строку подключения для соединения с DB.

    Класс DbContext также переопределяет метод OnModelCreating для предоставления некоторых примеров данных для приложения. Добавьте новый класс ProductsContext.cs в проект со следующим кодом:

Листинг 3. Файл ProductsContext.cs

```
1.  using Microsoft.EntityFrameworkCore;
2.
3.  namespace GetStartedWinForms;
4.
5.  public class ProductsContext : DbContext
6.  {
7.      public DbSet<Product> Products { get; set; }
8.      public DbSet<Category> Categories { get; set; }
9.
```

```
10.     protected override void
    OnConfiguring(DbContextOptionsBuilder optionsBuilder)
11.         => optionsBuilder.UseSqlite("Data Source=products.db");
12.
13.     protected override void OnModelCreating(ModelBuilder
    modelBuilder)
14.     {
15.         modelBuilder.Entity<Category>().HasData(
16.             new Category { CategoryId = 1, Name = "Cheese" },
17.             new Category { CategoryId = 2, Name = "Meat" },
18.             new Category { CategoryId = 3, Name = "Fish" },
19.             new Category { CategoryId = 4, Name = "Bread" });
20.
21.         modelBuilder.Entity<Product>().HasData(
22.             new Product { ProductId = 1, CategoryId = 1, Name =
    "Cheddar" },
23.             new Product { ProductId = 2, CategoryId = 1, Name =
    "Brie" },
24.             new Product { ProductId = 3, CategoryId = 1, Name =
    "Stilton" },
25.             new Product { ProductId = 4, CategoryId = 1, Name =
    "Cheshire" },
26.             new Product { ProductId = 5, CategoryId = 1, Name =
    "Swiss" },
27.             new Product { ProductId = 6, CategoryId = 1, Name =
    "Gruyere" },
28.             new Product { ProductId = 7, CategoryId = 1, Name =
    "Colby" },
29.             new Product { ProductId = 8, CategoryId = 1, Name =
    "Mozzela" },
30.             new Product { ProductId = 9, CategoryId = 1, Name =
    "Ricotta" },
31.             new Product { ProductId = 10, CategoryId = 1, Name =
    "Parmesan" },
32.             new Product { ProductId = 11, CategoryId = 2, Name =
    "Ham" },
33.             new Product { ProductId = 12, CategoryId = 2, Name =
    "Beef" },
```

```
34.             new Product { ProductId = 13, CategoryId = 2, Name =
    "Chicken" },
35.             new Product { ProductId = 14, CategoryId = 2, Name =
    "Turkey" },
36.             new Product { ProductId = 15, CategoryId = 2, Name =
    "Prosciutto" },
37.             new Product { ProductId = 16, CategoryId = 2, Name =
    "Bacon" },
38.             new Product { ProductId = 17, CategoryId = 2, Name =
    "Mutton" },
39.             new Product { ProductId = 18, CategoryId = 2, Name =
    "Pastrami" },
40.             new Product { ProductId = 19, CategoryId = 2, Name =
    "Hazlet" },
41.             new Product { ProductId = 20, CategoryId = 2, Name =
    "Salami" },
42.             new Product { ProductId = 21, CategoryId = 3, Name =
    "Salmon" },
43.             new Product { ProductId = 22, CategoryId = 3, Name =
    "Tuna" },
44.             new Product { ProductId = 23, CategoryId = 3, Name =
    "Mackerel" },
45.             new Product { ProductId = 24, CategoryId = 4, Name =
    "Rye" },
46.             new Product { ProductId = 25, CategoryId = 4, Name =
    "Wheat" },
47.             new Product { ProductId = 26, CategoryId = 4, Name =
    "Brioche" },
48.             new Product { ProductId = 27, CategoryId = 4, Name =
    "Naan" },
49.             new Product { ProductId = 28, CategoryId = 4, Name =
    "Focaccia" },
50.             new Product { ProductId = 29, CategoryId = 4, Name =
    "Malted" },
51.             new Product { ProductId = 30, CategoryId = 4, Name =
    "Sourdough" },
52.             new Product { ProductId = 31, CategoryId = 4, Name =
    "Corn" },
```

```
53.             new Product { ProductId = 32, CategoryId = 4, Name =
    "White" },
54.             new Product { ProductId = 33, CategoryId = 4, Name =
    "Soda" });
55.     }
56. }
```

15. **Выполните сборку** проекта обязательно (!).

Добавление элементов управления на форму

16. Приложение отобразит список категорий и список продуктов. Если в первом списке выбрана категория, то второй список изменится, чтобы отобразить продукты для этой категории. Эти списки можно изменять, чтобы добавлять, удалить или изменить продукты и категории, и эти изменения можно сохранять в базе данных SQLite, нажав кнопку "Сохранить ".
    Измените имя основной формы Form1 на MainForm.
17. Измените заголовок на "Продукты и категории".
18. С помощью панели элементов добавьте два DataGridView элемента управления, расположенные рядом друг с другом.
19. В свойствах для первого DataGridView измените его имя на dataGridViewCategories. В войствах второго DataGridView измените его имя на dataGridViewProducts.
20. С помощью панели элементов добавьте элемент управления Button. Назовите кнопку buttonSave и присвойте ей текст "Сохранить". Форма должна выглядеть следующим образом:

Рис. 5. Внешний вид формы

Привязка данных

21. Следующим шагом является подключение типов Product и Category из модели DataGridView к элементам управления. Это привяжет данные, загруженные EF Core, к элементам управления, таким образом, что сущности,

отслеживаемые EF Core, синхронизируются с данными, отображаемыми в элементах управления. Щелкните глиф действия в конструкторе на первом DataGridView. (Это крошечная кнопка в правом верхнем углу элемента управления). Откроется список действий, из которого можно получить доступ к раскрывающемся списку для выбранного источника данных. Мы еще не создали источник данных, поэтому перейдите к нижнему краю и нажмите кнопку "Добавить новый источник данных объекта...".

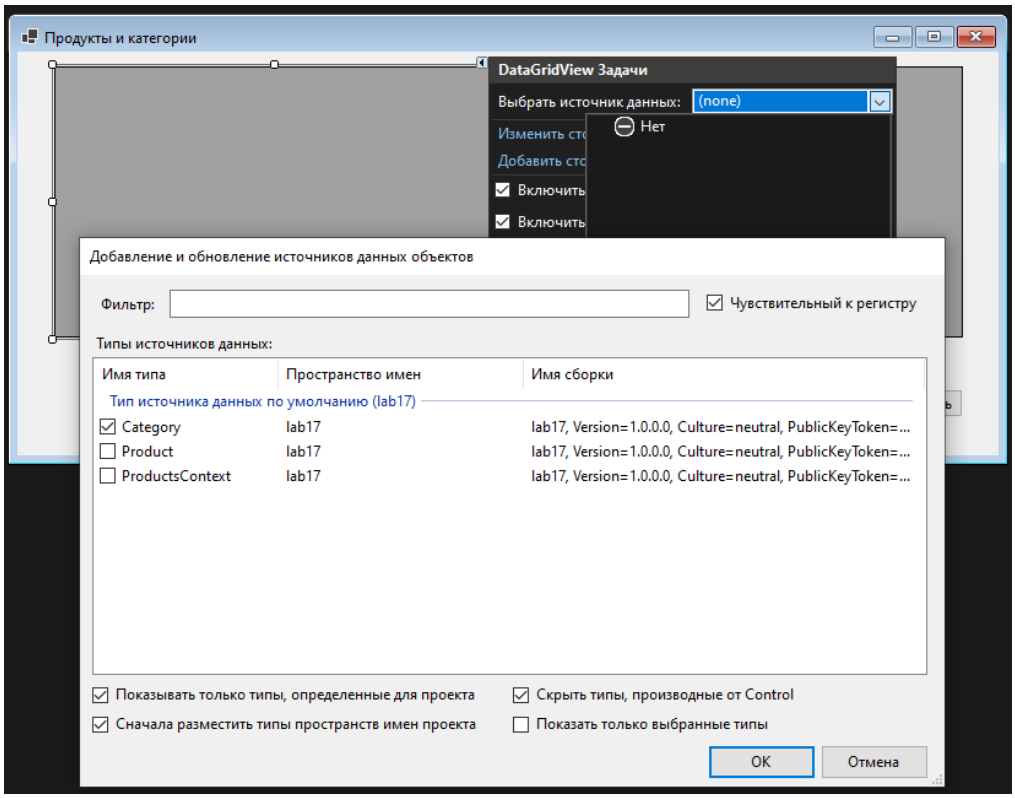

Рис. 6. Добавление источника данных

22. Второй DataGridView будет привязан к продуктам. Однако вместо привязки к типу верхнего уровня Product она будет привязана к Products навигации из Category привязки первой DataGridView. Это означает, что при выборе категории в первом представлении продукты для этой категории будут автоматически использоваться во втором представлении.
23. Используя глиф действия в конструкторе во втором DataGridView, выберите "Выбрать источник данных", а затем разверните categoryBindingSource и выберите Products.

Настройка отображаемых элементов

24. По умолчанию столбецы в DataGridView создаются для каждого свойства привязанных типов. Кроме того, значения для каждого из этих свойств может

быть изменено пользователем. Однако некоторые значения, такие как значения первичного ключа, концептуально доступны только для чтения, и поэтому их не следует изменять. Кроме того, некоторые свойства, такие как свойство CategoryId внешнего ключа и Category навигации, не полезны для пользователя, и поэтому их следует скрыть.

Щелкните правой кнопкой мыши первый DataGridView и выберите пункт "Изменить столбцы...". CategoryId Сделайте столбец, представляющий первичный ключ, только для чтения и нажмите кнопку "OK".

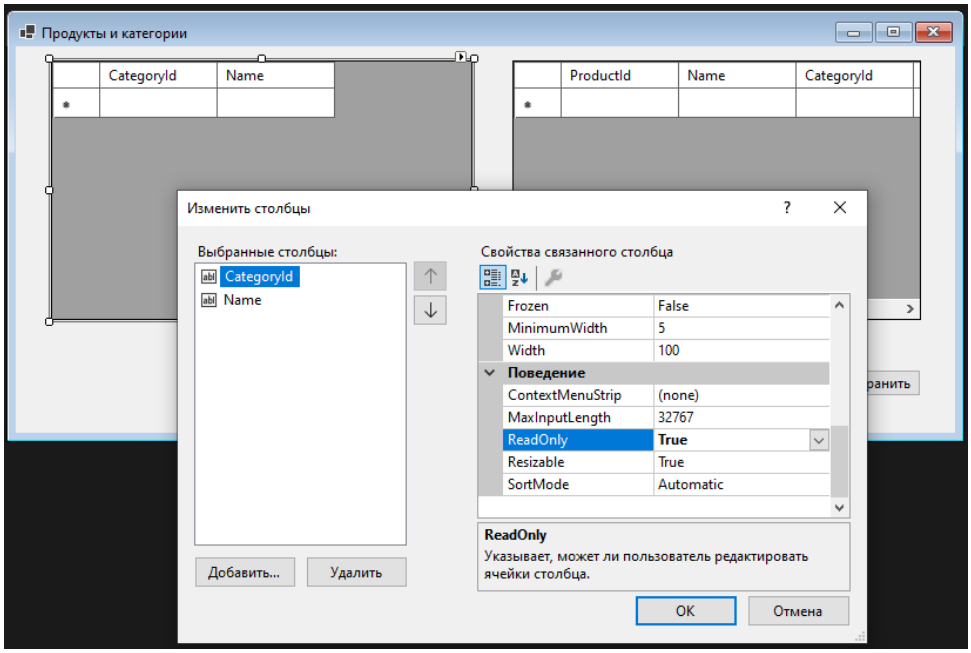

Рис. 7. Управление отображением

25. Щелкните правой кнопкой мыши на второй DataGridView и выберите пункт "Изменить столбцы...". Сделайте столбец ProductId доступным только для чтения и удалите столбцы Category и CategoryId, а затем нажмите кнопку "OK".

Подключение к EF Core

26. Осуществим подключение привязанных к данных в EF Core к элементам управления. Откройте код MainForm, щелкнув файл правой кнопкой мыши на форме: "Просмотреть код". Добавьте приватное поле для хранения сеанса DbContext и добавьте переопределение методов для OnLoad и OnClosing методов. Код должен выглядеть следующим образом:

Листинг 4. Файл Form1.cs

```
1.  using Microsoft.EntityFrameworkCore;
```

```
using System.ComponentModel;

namespace GetStartedWinForms
{
    public partial class MainForm : Form
    {
        private ProductsContext? dbContext;

        public MainForm()
        {
            InitializeComponent();
        }

        protected override void OnLoad(EventArgs e)
        {
            base.OnLoad(e);
            this.dbContext = new ProductsContext();
 // Uncomment the line below to start fresh with a new database.
            this.dbContext.Database.EnsureCreated();
            this.dbContext.Categories.Load();
            this.categoryBindingSource.DataSource =
dbContext.Categories.Local.ToBindingList();
        }

        protected override void OnClosing(CancelEventArgs e)
        {
            base.OnClosing(e);
            this.dbContext?.Dispose();
            this.dbContext = null;
        }
    }
}
```

- Метод OnLoad вызывается при загрузке формы.
- Экземпляр ProductsContext использоваться для загрузки и отслеживания изменений в продуктах и категориях, отображаемых приложением.
- EnsureCreated вызывается DbContext для создания базы данных SQLite, если она еще не существует. Это быстрый способ создания базы данных при

создании прототипов или тестировании приложений. **Если модель изменяется, необходимо удалить базу данных, чтобы ее можно было создать еще раз.** (Строка EnsureDeleted может быть незакомментирована, чтобы легко удалить и повторно создать базу данных при запуске приложения.) Для изменения и обновления схемы базы данных без потерь можно использовать миграцию данных EF Core: https://learn.microsoft.com/ru-ru/ef/core/managing-schemas/migrations/.

- EnsureCreated заполняет новую базу данных данными, определенными в методе ProductsContext.OnModelCreating.
- Метод Load используется для загрузки всех категорий из базы данных в DbContext базу данных. Эти сущности будут отслеживаться DbContext, который покажет любые изменения, внесенные пользователем при изменении категорий.
- Свойство categoryBindingSource.DataSource инициализировано в категории, отслеживаемых данных DbContext. Это реализуется путем вызова свойства Categories DbSet Local.ToBindingList(). Local предоставляет доступ к локальному представлению отслеживаемых категорий с событиями, которые подключены к локальным данным, чтобы обеспечить синхронизацию локальных данных с отображаемыми данными и наоборот. ToBindingList() предоставляет эти данные в виде IbindingList привязки данных Windows Forms.
- Метод OnClosing вызывается при закрытии формы. Он удаляет DbContext. Это гарантирует, что все ресурсы базы данных будут освобождены, а поле dbContext переведено в значение NULL и его нельзя будет использовать снова.

Заполнение представления "Продукты"

27. Чтобы заполнить таблицу продуктов, EF Core необходимо загрузить продукты из базы данных для выбранной категории. Чтобы достичь этого, выполните действия:
    1. В конструкторе основной формы выберите dataGridViewCategories.
    2. В разделе "Свойства" выберите события (кнопка молнии) **и дважды щелкните событие SelectionChanged**.
28. Это создаст заглушку в основном коде формы для события, которое будет запущено всякий раз, когда изменяется выбор категории. Введите код для события:

Листинг 5. Фрагмент кода в файле Form1.cs

```
1.  private void dataGridViewCategories_SelectionChanged(object
    sender, EventArgs e)
2.  {
3.      if (this.dbContext != null)
4.      {
5.          var category =
    (Category)this.dataGridViewCategories.CurrentRow.DataBoundItem;
6.
7.          if (category != null)
8.          {
9.              this.dbContext.Entry(category).Collection(e =>
    e.Products).Load();
10.         }
11.     }
12. }
```

В этом коде, если существует активный (непустой) сеанс DbContext , мы получаем экземпляр Category, привязанный к выбранной строке текущей DataViewGrid строки (м. б. null , если выбрана последняя строка в представлении, которая используется для создания новых категорий). Если выбрана категория, необходимо загрузить продукты, связанные с этой категорией. Способы выполнения этих целей:

1. Получить экземпляр EntityEntry Category (dbContext.Entry(category)).
2. Предоставить EF Core сведения, что мы хотим работать с навигацией Products по коллекции Category (.Collection(e => e.Products)).
3. И, наконец, сообщить EF Core, что мы хотим загрузить коллекцию продуктов из базы данных (.Load();).

Сохранение изменений

29. Подключим кнопку "Сохранить"к EF Core, чтобы все изменения, внесенные в продукты и категории, сохранялись в базе данных. В конструкторе основной формы нажмите кнопку "Сохранить ". В разделе "Свойства" Button выберите события (кнопка молнии) и дважды щелкните событие Click. Введите код для события:

Листинг 6. Фрагмент кода в файле Form1.cs

```
private void buttonSave_Click(object sender, EventArgs e)
{
    this.dbContext!.SaveChanges();
    this.dataGridViewCategories.Refresh();
    this.dataGridViewProducts.Refresh();
}
```

Этот код вызывает SaveChanges DbContextобъект, который сохраняет все изменения, внесенные в базу данных SQLite. Если изменения не были внесены, то и вызов базы данных не выполняется. После сохранения DataGridView элементы управления обновляются. Это связано с тем, что EF Core считывает созданные значения первичного ключа для любых новых продуктов и категорий из базы данных. Вызов Refresh обновляет отображение с этими созданными значениями.

30. Теперь приложение можно запускать, а продукты и категории можно добавлять, удалять и изменять. Обратите внимание, что если кнопка "Сохранить " нажимается перед закрытием приложения, все внесенные изменения будут храниться в базе данных и повторно загружаться при повторном запуске приложения. Если элемент "Сохранить " не щелкается, все изменения теряются при повторном запуске приложения.

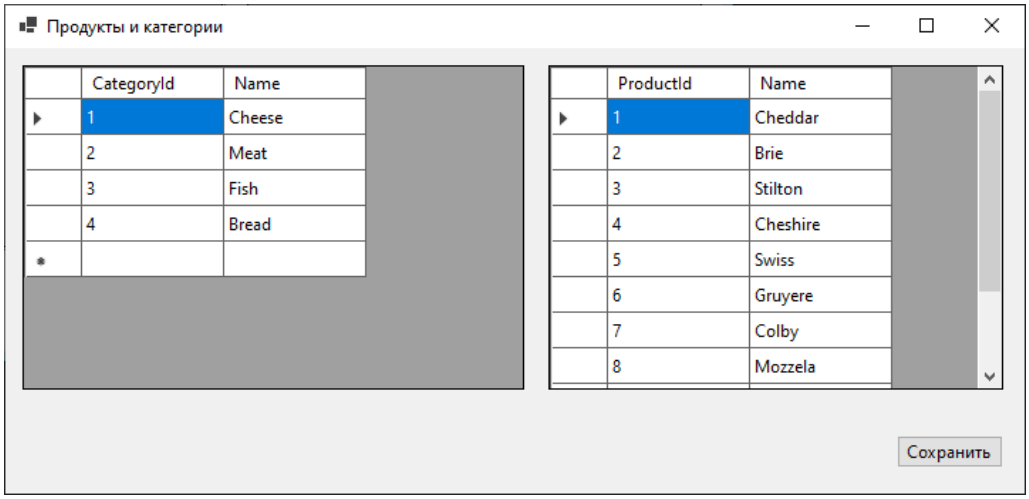

Рис. 8. Внешний вид окна приложения

5 Содержание отчета

1. Постановка задачи;
2. Описание элементов UI;
3. Исходный код (файлы Form1.cs и Form1.Designers.cs);
4. Скриншоты состояния окна программы;
5. Вывод, содержащий рекомендации по совершенствованию программы.