УДК 004

РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ДЛЯ ПРОГРАММНОГО ПРОДУКТА «КАЛЬКУЛЯТОР ДЛЯ ВЕТЕРИНАРНОЙ КЛИНИКИ»

Бужинская Н.В., Ведерников Н.А.

Нижнетагильский государственный социально-педагогический институт (филиал) ФГАОУ ВО «Российский государственный профессионально-педагогический университет», Нижний Тагил, e-mail: nadezhda_v_a@mail.ru

На современном этапе развития общества пользуются спросом программные продукты для автоматизации различных аспектов деятельности человека. С помощью данных программных продуктов можно знакомить пользователей с важной информацией, производить необходимые расчеты, оформлять необходимую документацию. В статье рассматриваются основные этапы разработки пользовательского интерфейса для калькулятора ветеринарной клиники. С помощью калькулятора клиенты знакомятся с услугами клиники, делают заказ и получают информацию о стоимости заказа и скидках. Основной частью калькулятора является база данных, которая включает перечень услуг и их стоимость. Поскольку стоимость услуги для того или иного животного различна, на языке программирования C# был создан шаблон базы данных, которая может принимать определенные значения в зависимости от действий пользователя. В статье представлен алгоритм создания такого шаблона. Кроме того, в статье описаны результаты этапа проектирования. С помощью диаграммы прецедентов были определены требования к калькулятору, а диаграмма деятельности показывает ход действий пользователя. На основе полученной информации был создан прототип интерфейса. На заключительном этапе работы в Microsoft Visual Studio 2019 на языке C# был разработан программный продукт.

Ключевые слова: программирование, интерфейс, пользовательский интерфейс, программный продукт, калькулятор, база данных

DEVELOPMENT OF A USER INTERFACE FOR THE SOFTWARE PRODUCT «CALCULATOR FOR VETERINARY CLINIC»

Buzhinskaya N.V., Vedernikov N.A.

Nizhny Tagil State Socio-Pedagogical Institute (branch) of Federal State Autonomous educational institution «Russian state vocational pedagogical University», Nizhny Tagil, e-mail: nadezhda_v_a@mail.ru

Abstract: At the present stage of development of society, software products are in demand for the automation of various aspects of human activity. Using these software products, you can acquaint users with important information, make the necessary calculations, draw up the necessary documentation. The article discusses the main stages of developing a user interface for a calculator of a veterinary clinic. With the help of a calculator, clients get acquainted with the clinic services, place an order and receive information about the cost of the order and discounts. The main part of the calculator is a database, which includes a list of services and their cost. Since the cost of the service for a particular animal is different, a database template was created in the C # programming language, which can take on certain values depending on the user's actions. The article presents an algorithm for creating such a template. In addition, the article describes the results of the design phase. Using the use case diagram, the requirements for the calculator were determined, and the activity diagram shows the course of action of the user. Based on the information received, a prototype interface was created. At the final stage of work at Microsoft Visual Studio 2019 in C #, a software product was developed.

Keywords: programming, interface, user interface, software product, calculator, database

Любой программный продукт можно оценить с точки зрения надежности, безопасности, удобства предлагаемых функций для пользователя. Эти показатели напрямую зависят от качества пользовательского интерфейса. Пользовательский интерфейс должен валидировать ввод данных, быть максимально простым в управлении, а также понятным для пользователя. Хорошо разработанный пользовательский интерфейс влияет на успех предлагаемого продукта на рынке программного обеспечения.

Интерфейс – комплекс программных и технических средств, посредством которых осуществляется взаимодействие оператора-человека с различными информационными системами в процессе их функционирования. В свою очередь, пользовательский интерфейс включает в себя компоненты, которые определяют характер данного взаимодействия. К ним относятся информационные поля, различные панели, диалоговое меню, сетки, списки и др. [1].

К особенностям пользовательского интерфейса можно отнести [2]:

– наличие графических средств управления программой;

– осуществление удобной навигации по функционалу программы;

– соответствие интерфейса бизнес-логики требуемой модели, а не бизнес-логика модели под интерфейс;

– обеспечение валидации входных данных и безопасности использования программы.

Рис. 1. Диаграмма прецедентов

Материалы и методы исследования

В данном исследовании рассматриваются этапы разработки пользовательского интерфейса для программы, которая предоставляет услуги ветеринарной клиники и делает расчёт их стоимости. Данный проект реализует базовый функционал любой информационной системы. Разработанные для проекта алгоритмы поиска работают независимо от бизнес-логики приложения, следовательно, проект не теряет актуальности даже в долгосрочной перспективе.

Требования к программному продукту:

– независимость слоёв представления программы – реализовать приложение на архитектуре MVVM. Это предпочтительная архитектура для программного обеспечения, использующего функционал WPF;

– наличие базы данных с услугами;

– наличие алгоритма поиска по названию и категории услуги;

– реализация функции для расчета стоимости выбранных услуг.

Результаты исследования и их обсуждение

Первым этапом в реализации нашего программного продукта является проектирование. На этапе проектирования определяются требования к нашей программе и основные функции [3–5]. Диаграмма прецедентов представлена на рис. 1.

Пользователь производит поиск интересующей его услуги посредством сортировки по названию или категории. После выбора услуг программа рассчитывает и показывает стоимость предоставляемых услуг.

Динамические аспекты поведения разрабатываемой системы описываются в виде диаграммы активностей [6]. На ней изображен алгоритм взаимодействия между пользователем и информационной системой. Пользователь с помощью инструмента поиска выбирает интересующие его услуги и узнаёт их суммарную стоимость (рис. 2).

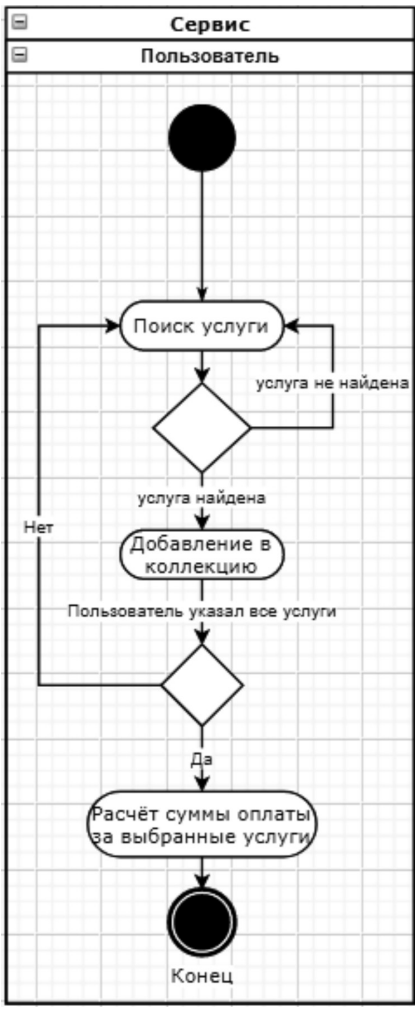

Рис. 2. Диаграмма деятельности UML

На следующем этапе был разработан прототип интерфейса, который должен удовлетворять требованиям заказчика и быть максимально удобным для пользователей.

База данных включает две таблицы. Первая таблица «Услуги» содержит информацию о коде услуги, ее названии и типе. Во второй таблице хранится информация о стоимости услуг, так как для каждого животного она различна. Поскольку одна и та же услуга может относиться ко многим животным, то в зависимости от типа животного изменяется цена и другие параметры. На C# был создан шаблон подобной базы данных, которая позволяет изменять значения в процессе выполнения программы, то есть в самой программе таблицы будут работать как коллекции.

Программный продукт будет создаваться в MicrosoftVisualStudio 2019 на языке C#. В процессе разработки нам необходимо создать основные модели, которые будут представлять базу данных.

Листинг 1

Описание таблицы «Услуги»

```
public class Service : IModel
{
public int Id { get; set; }
    public string Name { get; set; }
    public decimal Cost { get; set; }
    public PetCategory PetCategory { get; set; }

static public implicit operator ServiceViewModel?(Service model) =>
      new ServiceViewModel()
        {
      Id = model.Id,
            Cost = model.Cost,
            Name = model.Name,
            PetCategory = model.PetCategory
};
}
```

Данный код описывает таблицу «Услуги». Каждое свойство класса Service указывает, какие поля будут включены в таблицу. Данный класс также композирует в себе указатель на экземпляр типа PetCategory, что означает, что данная таблица будет связана со второй таблицей.

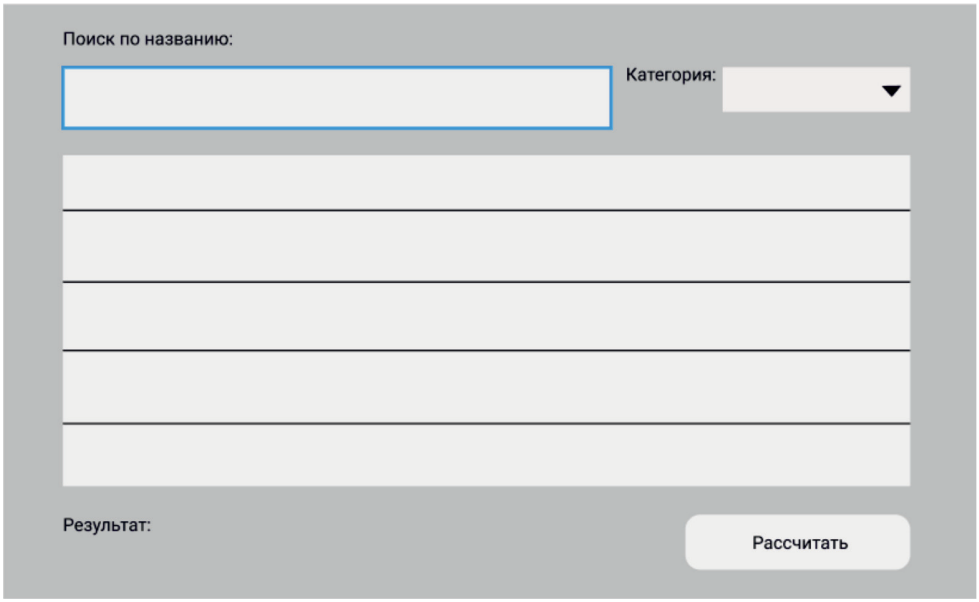

Рис. 3. Прототип интерфейса

В листинге 2 объявляется класс PetCategory, который описывает таблицу категорий животных. Данный класс содержит в себе коллекцию экземпляров типа Service, что указывает на связь между таблицами «Один ко многим».

Листинг 2

Описание таблицы категорий животных

```
public class PetCategory : IModel
{
public int Id { get; set; }
public string Value { get; set; }
    public List<Service> Services { get; set; }
    public PetCategory()
    {
Services = new List<Service>();
}

public static implicit operator PetCategoryViewModel?(PetCategory model) =>
new PetCategoryViewModel()
{
      Id = model.Id,
           Value = model.Value
};
}
```

Класс, представленный на листинге 3, VeterinaryDbContext, наследовавший класс DbContext, определяет настройки подключения к базе данных в методе OnConfiguring(), конфигурацию модели базы данных в методе OnModelCreating(), две дженерик-коллекции DbSet от классов PetCategory и Services, а также конструктор класса, реализующий DependencyInjection (встраиваемые зависимости), где в роли зависимости выступает конфигурация подключения к базе данных.

Листинг 3

Настройки подключения к базе данных

```
public class VeterinaryDbContext : DbContext
   {
      IConfiguration _configuration;
      public DbSet<PetCategory> PetCategories { get; set; }
      public DbSet<Service> Services { get; set; }
      public VeterinaryDbContext() => Database.EnsureCreatedAsync();
       public VeterinaryDbContext(DbContextOptions<VeterinaryDbContext> options) : base(options) =>
Database.EnsureCreatedAsync();
      public VeterinaryDbContext(IConfiguration configuration)
      {
         _configuration = configuration;
         Database.EnsureCreated();
      }
      protected override void OnModelCreating(ModelBuilder modelBuilder)
      {
         modelBuilder.Entity<Service>()
            .ToTable("services_tbl")
            .HasKey(key => key.Id);
         modelBuilder.Entity<PetCategory>()
            .ToTable("petCategories_tbl")
            .HasKey(key => key.Id);
         modelBuilder.Entity<PetCategory>()
            .HasMany(s => s.Services)
            .WithOne(p => p.PetCategory)
            .HasForeignKey("FK_to_pet");
      }
      protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) =>
         optionsBuilder.UseSqlServer(_configuration["connectionString"]);
   }
```

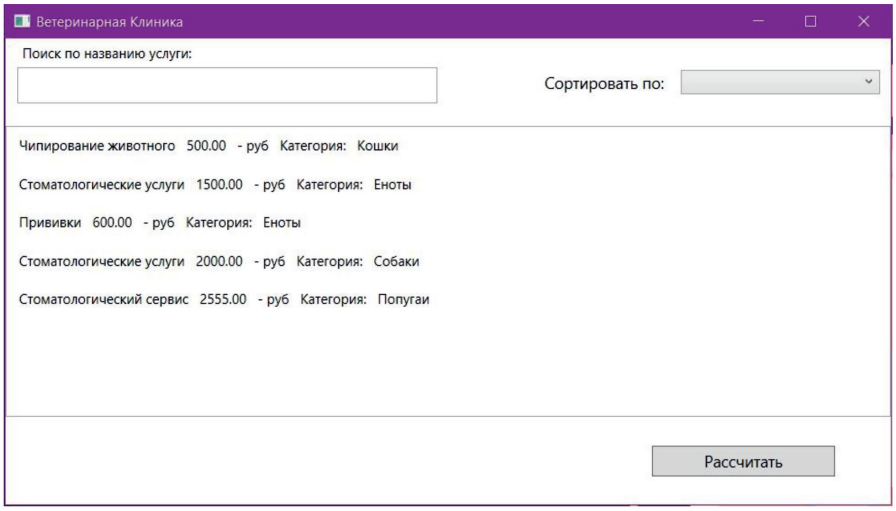

Рис. 4. Интерфейс программного продукта

После введённой команды, в консоли диспетчера пакетов Add-Migration Init-Migration с помощью библиотеки Entity-Framework Core создаётся класс миграции, в котором описываются правила создания базы данных на хосте. Далее вводится команда Update-Database, начинается процесс миграции и создания базы данных на узле, указанном в строке конфигурации (в данном случае localhost).

Для создания интерфейса была выбрана технология WPF. Особенность данной технологии в преимущественном использовании разметки XAML. Для привязки данных к интерфейсу используется контекст данных. Контекст данных позволяет не заботиться об обновлении данных на стороне хранилища данных, загруженного в адресное пространство программы. Слой View (представление) связывается с ViewModel (модель представления), который в свою очередь обращается к абстрактным репозиториям (паттерн для обращения к данным, который представляет общий интерфейс для любой реализации связи с базой данных посредством полиморфизма). Конструктор инициализирует коллекции для отображения PetCategories и Services через реализации абстрактного репозитория, которые в свой конструктор для инициализации принимают экземпляр контекста базы данных.

В классе главного окна MainWindow инициализируется конфигурация и основной экземпляр класса ViewModel слоя VeterinaryClinicViewModel. Экземпляр ViewModel слоя присваивается к свойству DataContext для того, чтобы все элементы представления имели доступ к свойствам и коллекциям экземпляра VeterinaryClinicViewModel.

Результат разработки представлен на рис. 4.

Пользователь может видеть список услуг ветеринарной клиники, осуществлять поиск и расчет нужной ему услуги.

Заключение

Программный продукт разработан и удовлетворяет всем требованиям заказчика. В дальнейшем планируется:

– оптимизировать поиск сервисов путем кеширования и повторного использования плана запроса к БД;

– осуществить переход на документо-ориентированную базу данных, поскольку данная сложность модели позволяет сделать этот переход, например MongoDB;

– оптимизировать алгоритм с помощью отказа от LINQ-запросов, которые относительно медленно работают со структурами данных. Если загружать данные с БД «порциями» (метод загрузки chunkcache) и эти данные сортировать оптимизированным алгоритмом, то скорость клиента возрастёт, а нагрузка на сервер снизится.

Список литературы

1. Сергеев С.Ф., Падерно П.И., Назаренко Н.А. Введение в проектирование интеллектуальных интерфейсов: учебное пособие. СПб.: СПбГУ ИТМО, 2011. 108 с.

2. Мандел Т. Разработка пользовательского интерфейса: Пер. с англ. М.: ДМК Пресс. 416 с.

3. Бедердинова О.И., Коряковская Н.В., Бойцова Ю.А. Статическая модель системы оценивания качества программных продуктов // Вестник Северного (Арктического) федерального университета. Серия: Естественные науки. 2015. № 3. С. 87–96.

4. Васева Е.С., Матис П.С. Проектирование базы данных учета сотрудников образовательного учреждения // Наука и перспективы. 2017. № 2. С. 35–41.

5. Попов А.А., Винтова Т.А. Объектно-ориентированный анализ предметной области «Управление многоквартирными домами» на основе зарубежного опыта автоматизации управления недвижимостью // Современные наукоемкие технологии. 2018. № 2. С. 74–82.

6. Каюмова А.В. Визуальное моделирование систем в StarUML: учебное пособие. Казань: Казанский федеральный университет, 2013. 104 с.