*Федеральное государственное бюджетное образовательное учреждение высшего образования*

*«Владимирский государственный университет*

*имени Александра Григорьевича и Николая Григорьевича Столетовых»*

*Кафедра информационных систем и программной инженерии*

***ПОЯСНИТЕЛЬНАЯ ЗАПИСКА***

***к курсовому проекту по дисциплине   
"Технологии программирования"***

***на тему***

*Проектирование и разработка программной системы   
«Помощник повара»*

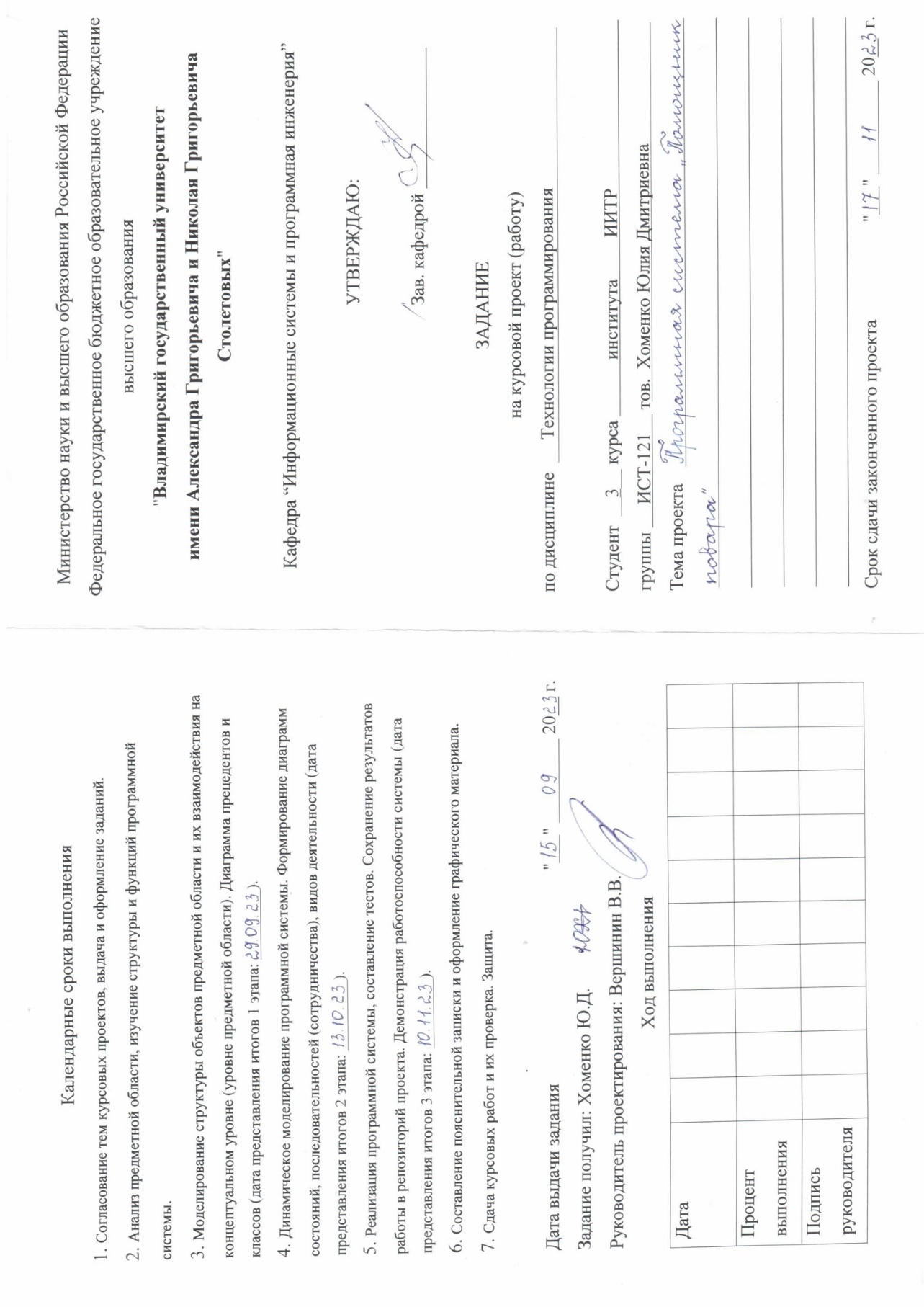
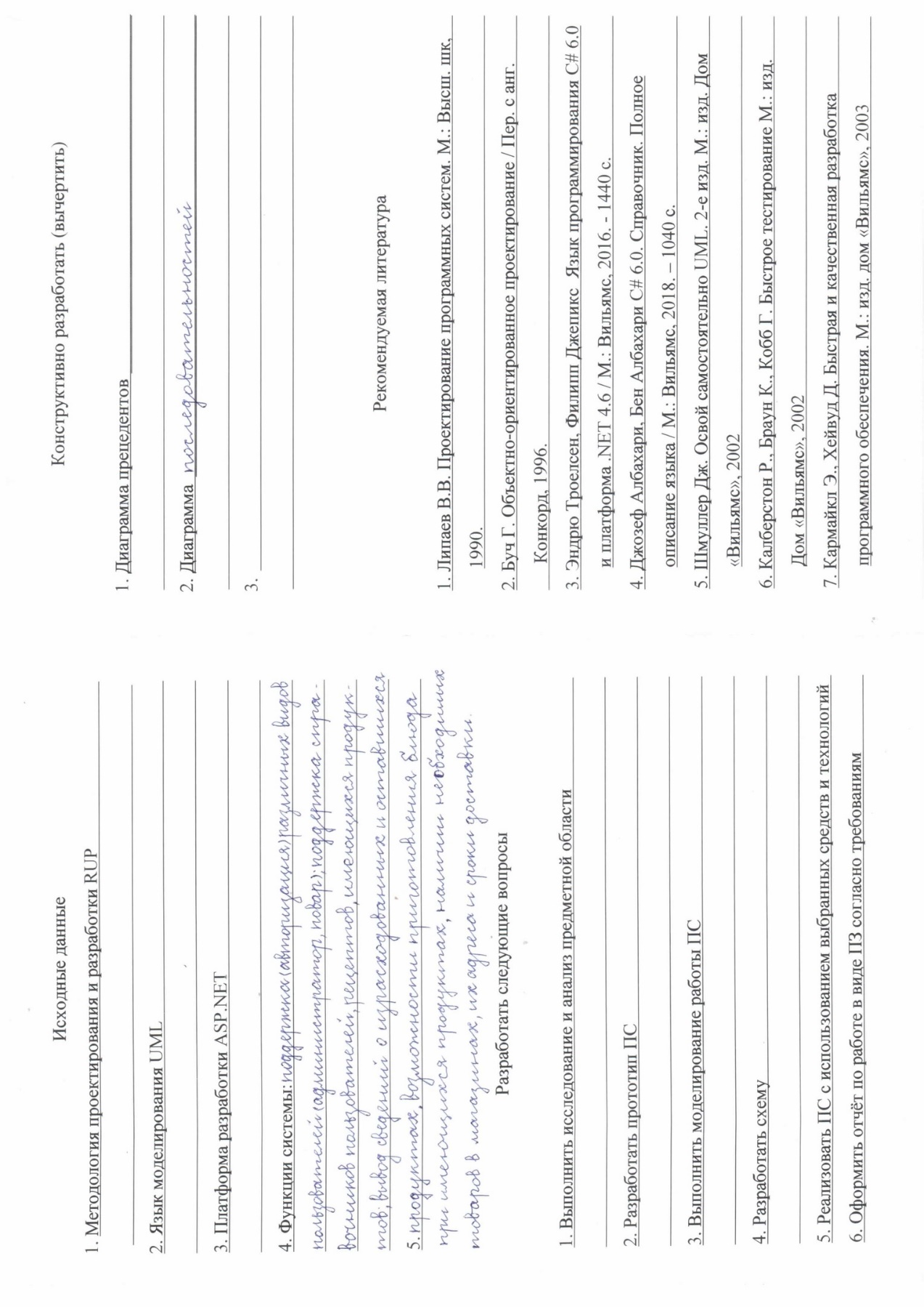
*Выполнила: ст. гр. ИСТ-121*

*Хоменко Ю.Д.*

*Принял: доц. кафедры ИСПИ*

*Вершинин В.В.*

*Владимир, 2023*

****

**Аннотация**

Данная курсовая работа посвящена разработке программной системы «Помощник повара», предназначенной для оптимизации и улучшения процесса приготовления пищи. В работе проведен анализ предметной области, выделены основные понятия и сущности, определены требования к системе. Произведено проектирование структуры с использованием диаграмм прецедентов, классов, видов деятельности, последовательностей, состояний и развёртывания.

Раздел «Реализация системы» включает в себя проектирование пользовательского интерфейса, реализацию классов и контроллеров.

Объем работы составляет 32 страницы, включая 16 рисунков, 7 таблиц, 4 использованных источника. Приложение включают структуру таблиц базы данных.

**ANNOTATION**

This course work is devoted to the development of the «Chef Assistant» software system, designed to optimize and improve the food preparation process. The work carried out an analysis of the subject area, highlighted the basic concepts and entities, and determined the requirements for the system. Designed the structure using use case, class, activity, sequence, state and deployment diagrams.

The «System Implementation» section includes designing the user interface, implementing classes and controllers.

The volume of work is 32 pages, including 16 figures, 7 tables, 4 sources used. The application includes the structure of the database tables.

**Содержание**

ВВЕДЕНИЕ 3

1 Описание предметной области 4

1.1. Словарь предметной области 4

1.2. Понятия предметной области 4

1.3. Описание предметной области 4

2 проектирование системы 6

2.1 Проектирование диаграммы прецедентов 6

2.2 Проектирование диаграммы классов 10

2.3 Проектирование диаграмм видов деятельности 12

2.4 Проектирование диаграмм последовательностей 13

2.5 Проектирование диаграмм состояний 14

2.6 Проектирование диаграмм развёртывания 15

заключение 18

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 19

ПРИЛОЖЕНИЕ А Исходный код программной системы 20

ПРИЛОЖЕНИЕ Б Структура таблиц БД 21

# ВВЕДЕНИЕ

С ростом темпов современной жизни многие люди сталкиваются с ограниченным временем на готовку пищи, а также с необходимостью постоянного улучшения кулинарных навыков. Программные системы призваны облегчить эти трудности, предоставляя пользователям удобный инструмент для поддержки в процессе приготовления блюд.

Целью данной курсовой работы является разработка программной системы «Помощник повара», направленной на оптимизацию процесса кулинарного творчества.

Для достижения поставленной цели необходимо решить следующие задачи:

– Разработать функциональные требования к программной системе, определить основные характеристики и возможности.

– Спроектировать диаграммы прецедентов, классов, деятельности и другие диаграммы.

– Реализовать программную систему с использованием языка C#.

– Провести тестирование системы.

Объектом исследования является процесс кулинарного творчества, а предметом исследования — программная система «Помощник повара».

# Описание предметной области

* 1. Словарь предметной области

*Повар* – тот, кто пользуется системой, выполняет закупки продуктов и другие доступные функции.

Атрибуты: имя, возраст, пол.

*Блюдо* – описание самого блюда с указанием набора ингредиентов, которые входят в его состав.

Атрибуты: название, ингредиенты, количество каждого ингредиента.

*Продукт* – описание ингредиентов, который находятся в наличии с указанием количества.

Атрибуты: наименование, количество на складе.

* 1. Понятия предметной области

Рецепт – набор шагов и инструкций для приготовления конкретного блюда.

Ингредиент – компонент, используемый при приготовлении блюда.

Меню – набор блюд.

Пользователь – лицо, использующее систему для приготовления пищи.

Список покупок – список ингредиентов, необходимых для приготовления выбранных блюд.

* 1. Описание предметной области

Технологии активно развиваются и влияют на различные сферы жизни, гастрономия не является исключением. Разнообразие кулинарных традиций и стремление к новым вкусовым открытиям создают потребность в средствах поддержки и оптимизации процесса приготовления пищи.

Современные требования к приготовлению пищи отличаются ориентированностью на сохранение времени повара. Растущий интерес к здоровому образу жизни также повлиял на тенденции в кулинарии, подчеркивая важность разнообразных блюд. Также, увеличивается спрос на персонализированные рецепты, адаптированные к диетам.

На рынке существует множество приложений и веб-сервисов, предназначенных для помощи в приготовлении блюд. Разнообразие функционала включает в себя управление рецептами с подробными инструкциями, возможность составления списка покупок, а также функции социального взаимодействия, например, обмен рецептами. Некоторые приложения также предоставляют функции планирования недельного рациона и подбора блюд в соответствии с диетическими предпочтениями.

На основе проведенного анализа предметной области, выделяются ключевые требования к программной системе «Помощник повара»:

– Система должна предоставлять возможность управления списками блюд, рецептов, поваров.

– Система должна предоставлять возможность создания закупок новых продуктов.

– Система должна предоставлять возможность получения сведений о запасах продуктов.

– Система должна предоставлять пользователю удобный, интуитивный интерфейс.

Нефункциональные требования:

– Реализация на платформе .NET.

– Использование технологии ASP.NET.

– Использование шаблона проектирования MVC.

– База данных MS SQL.

– Использование Entity Framework для работы с данными.

# проектирование системы

2.1 Проектирование диаграммы прецедентов

Согласно методологии UML, каждый прецедент тождественен бизнес-функции системы, выявленной на этапе анализа. На рисунке 1 представлена диаграмма прецедентов системы «Помощник повара».

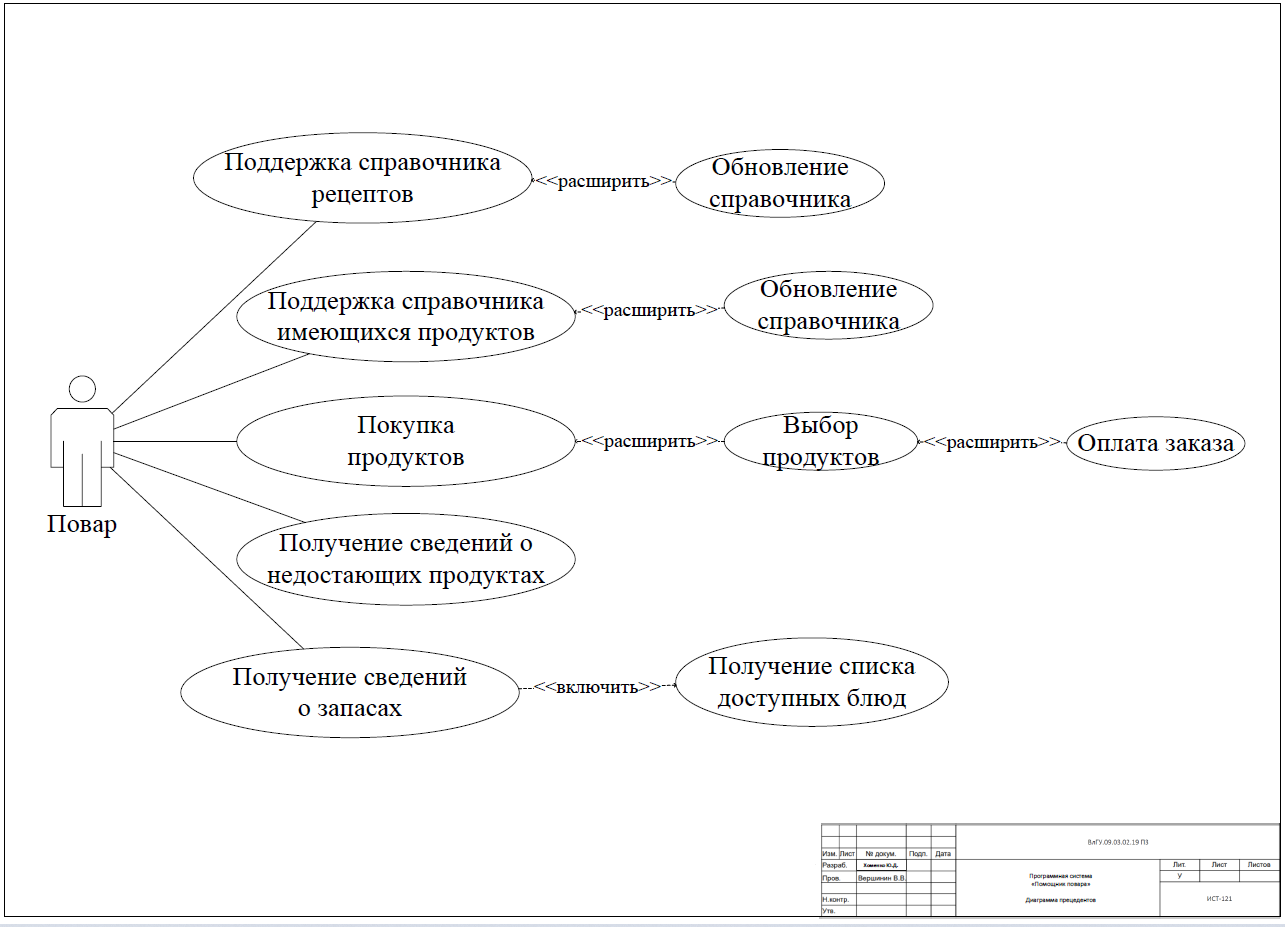


Рисунок 1. Диаграмма прецедентов системы «Помощник повара»

В таблицах 1-7 приведены спецификации прецедентов программной системы.

*Таблица 1*

Расширенное описание прецедента «Поддержка справочника рецептов»

**Название:** «Поддержка справочника рецептов»

**Предусловие:** повар зарегистрирован в ИС «Помощник повара» как повар, т.е. имеет присвоенный ему номер в справочнике системы, он авторизован в системе, блюдо выбрано.

**Действующее лицо:** повар

**Основной поток:** *Поддерживать справочник рецептов*

Повар запускает систему, открывает раздел «Блюда» и выбирает одно из блюд из списка.

Нажимает кнопку «Рецепт» и получает на экране выведенный список рецептов блюда.

**Альтернативный поток:** Ошибка при получении данных.

**Постусловие:** после нажатия последней кнопки список рецептов выводится на экран.

*Таблица 2*

Расширенное описание прецедента «Обновление справочника рецептов»

**Название:** «Обновление справочника рецептов»

**Предусловие:** повар зарегистрирован в ИС «Помощник повара» как повар, т.е. имеет присвоенный ему номер в справочнике системы, он авторизован в системе, рецепт выбран.

**Действующее лицо:** повар

**Основной поток:** *Обновлять справочник рецептов*

Повар запускает систему, открывает раздел «Блюда» и выбирает одно из блюд из списка.

Нажимает кнопку «Рецепт» и получает на экране выведенный список рецептов блюда.

Выбирает рецепт для обновления из списка или создаёт новый рецепт.

**Альтернативный поток:** Ошибка при получении данных.

**Постусловие:** после обновления все данные сохраняются.

*Таблица 3*

Расширенное описание прецедента «Поддержка справочника имеющихся продуктов»

**Название:** «Поддержка справочника имеющихся продуктов»

**Предусловие:** повар зарегистрирован в ИС «Помощник повара» как повар, т.е. имеет присвоенный ему номер в справочнике системы, он авторизован в системе.

**Действующее лицо:** повар

**Основной поток:** *Поддерживать справочник имеющихся продуктов*

Повар запускает систему и открывает раздел «Продукты».

**Альтернативный поток:** Ошибка при получении данных.

**Постусловие:** после нажатия последней кнопки список продуктов выводится на экран.

*Таблица 4*

Расширенное описание прецедента «Обновление справочника продуктов»

**Название:** «Обновление справочника продуктов»

**Предусловие:** повар зарегистрирован в ИС «Помощник повара» как повар, т.е. имеет присвоенный ему номер в справочнике системы, он авторизован в системе, продукт выбран.

**Действующее лицо:** повар

**Основной поток:** *Обновлять справочник продуктов*

Повар запускает систему и открывает раздел «Продукты».

Выбирает продукт для обновления из списка или вводит новый продукт.

**Альтернативный поток:** Ошибка при получении данных.

**Постусловие:** после обновления все данные сохраняются.

*Таблица 5*

Расширенное описание прецедента «Покупка продуктов»

**Название:** «Покупка продуктов»

**Предусловие:** повар зарегистрирован в ИС «Помощник повара» как повар, т.е. имеет присвоенный ему номер в справочнике системы, он авторизован в системе.

**Действующее лицо:** повар

**Основной поток:** *Покупать продукты*

Повар открывает раздел «Закупки», отображающий список продуктов, которые необходимы для выбранного блюда и отсутствуют на складе в данный момент.

Нажимает кнопку «Купить» и создаёт закупку, выбирая необходимые продукты и заполняя поля закупки.

Нажимает на кнопку «Оплатить заказ».

**Альтернативный поток:** Ошибка при получении данных.

**Постусловие:** когда покупка завершена, информация об имеющихся на складе продуктах в системе обновляется, а закупка сохраняется.

*Таблица 6*

Расширенное описание прецедента «Получение сведений о недостающих продуктах»

**Название:** «Получение сведений о недостающих продуктах»

**Предусловие:** повар зарегистрирован в ИС «Помощник повара» как повар, т.е. имеет присвоенный ему номер в справочнике системы, он авторизован в системе.

**Действующее лицо:** повар

**Основной поток:** *Получать сведения о недостающих продуктах*

Повар запускает систему и открывает раздел «Продукты».

**Альтернативный поток:** Ошибка при получении данных.

**Постусловие:** после нажатия последней кнопки список недостающих продуктов выводится на экран.

*Таблица 7*

Расширенное описание прецедента «Получение сведений о запасах продуктов»

**Название:** «Получение сведений о запасах продуктов»

**Предусловие:** повар зарегистрирован в ИС «Помощник повара» как повар, т.е. имеет присвоенный ему номер в справочнике системы, он авторизован в системе.

**Действующее лицо:** повар

**Основной поток:** *Получать сведения о запасах продуктов*

Повар запускает систему и открывает раздел «Продукты».

**Альтернативный поток:** Ошибка при получении данных.

**Постусловие:** после нажатия последней кнопки список запасов продуктов выводится на экран.

2.2 Проектирование диаграммы классов

Проектирование диаграммы классов начинается с анализа содержимого словаря предметной области, ТЗ и разработанной ранее диаграммы прецедентов. Результатом анализа является синтезированная диаграмма классов, которая ложится в основу разрабатываемой системы на этапе реализации.

На рисунке 2 изображена концептуальная диаграмма классов системы «Помощник повара».

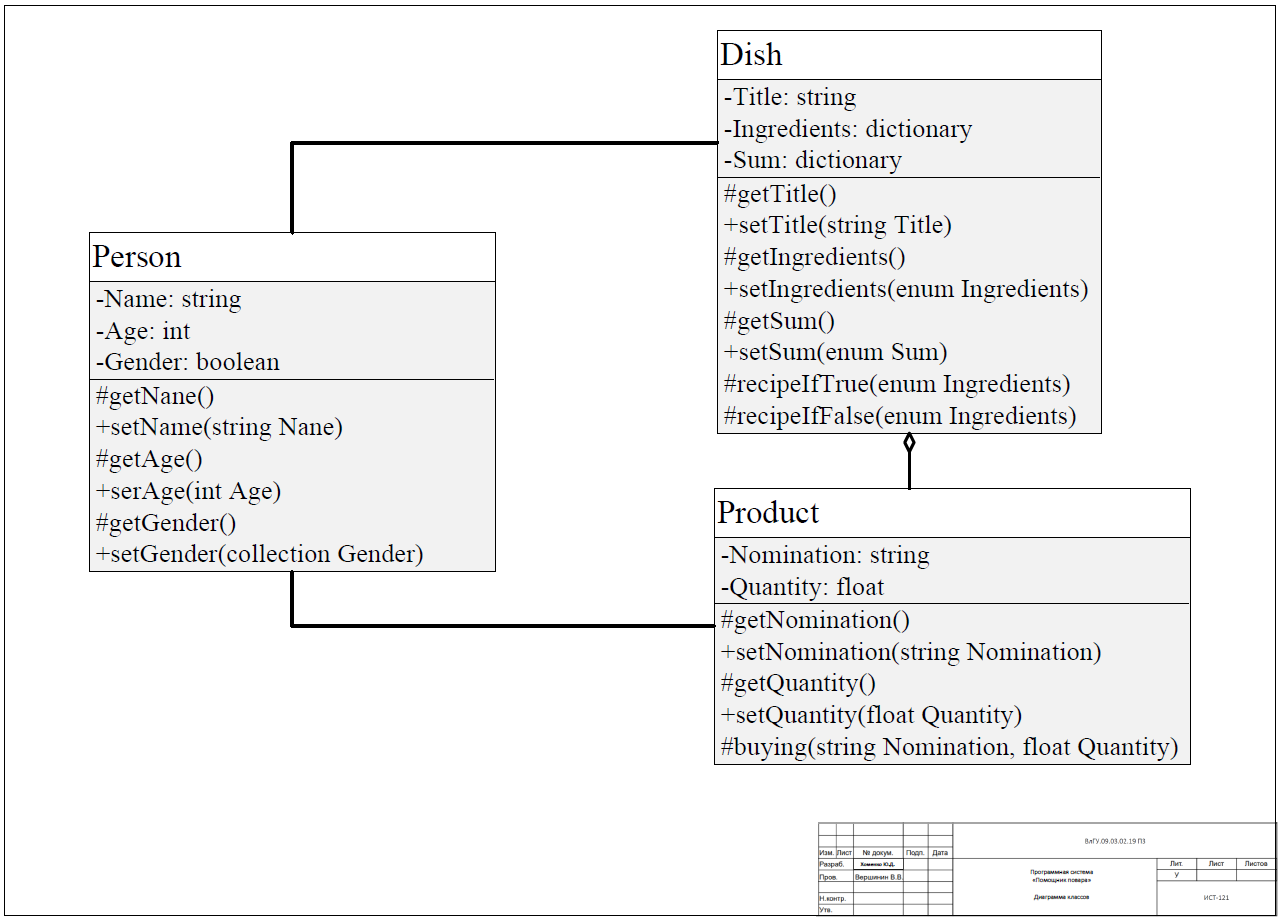


Рисунок 2. Концептуальная диаграмма классов системы «Помощник повара»

На диаграмме представлены следующие классы:

1) Person (человек или повар). Свойства класса: Name типа string, Age типа int и Gender типа bool. Методы класса: getName(), getAge(), getGender() для получения данных объекта.

2) Dish (блюдо). Свойства класса: Title типа string, Ingredients (список ингредиентов), Sum (количество ингредиентов). Методы класса: getTitle(), getIngredients(), getSum(), recipeIfTrue().

3) Product (продукт). Свойства класса: Nomination типа string, Quantity типа float. Методы класса: getNomination(), setNomination(), buying() для покупки проудкта.

На рисунке 3 изображена диаграмма классов системы «Помощник повара» уровня реализации.

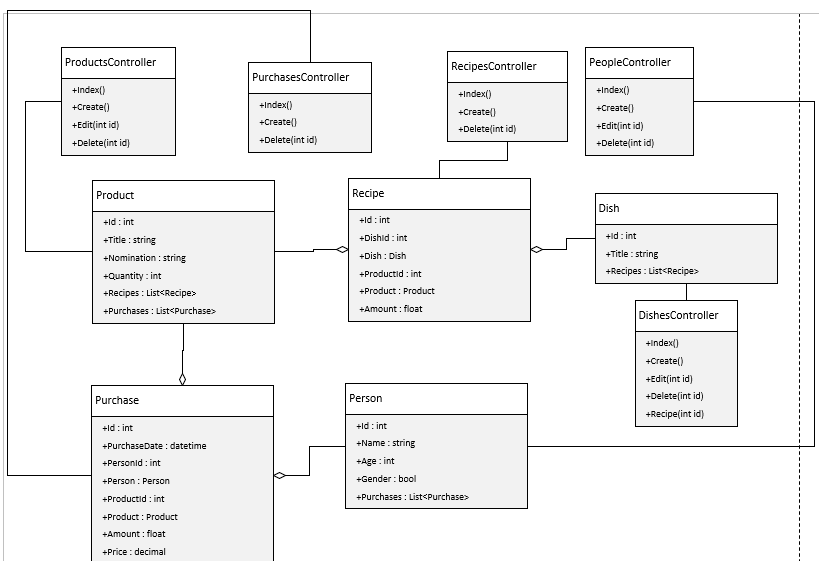


Рисунок 3. Диаграмма классов системы «Помощник повара» уровня реализации

На диаграмме классов уровня реализации отображаются также данные о закупках продуктов (Purchase) и рецептах блюд (Recipe). Для управления справочниками используются так называется контроллеры (Controllers). Для каждого объекта соответствующий контроллер.

2.3 Проектирование диаграмм видов деятельности

Диаграмма видов деятельности похожа на диаграмму последовательностей, но делает акцент на взаимодействии участников. Здесь также отображается динамическая сторона системы.

На рисунке 4 изображена диаграмма видов деятельности для прецедента «Покупка продуктов».

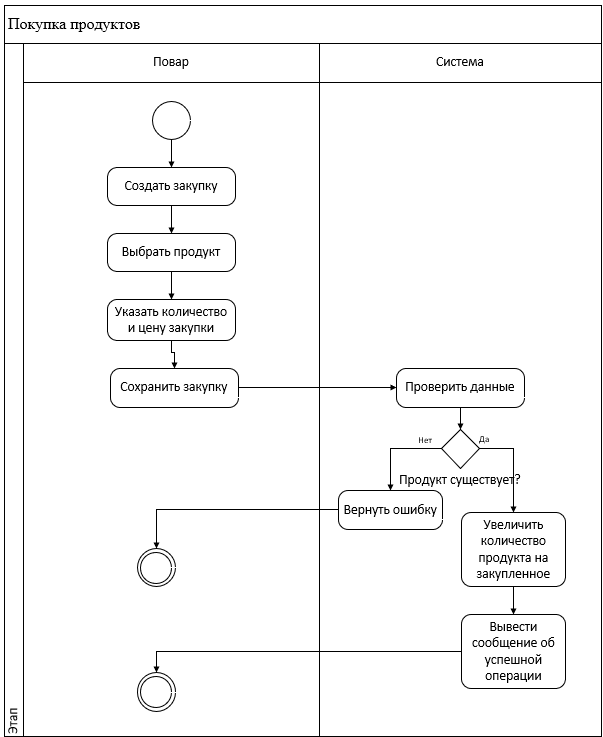


Рисунок 4. Диаграмма видов деятельности для прецедента «Покупка продуктов»

На рисунке 5 изображена диаграмма видов деятельности для прецедента «Получение сведений о запасах».

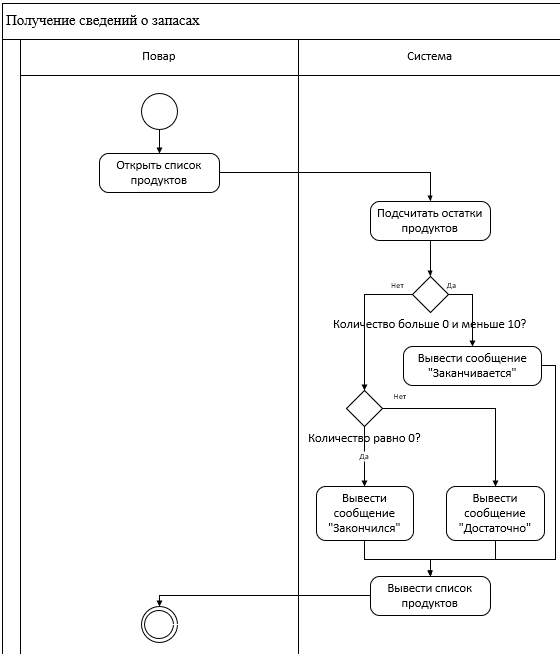


Рисунок 5. Диаграмма видов деятельности для прецедента «Получение сведений о запасах»

2.4 Проектирование диаграмм последовательностей

На диаграмме последовательностей система изображается в динамике. Она используется в том случае, если прецедент имеет неочевидное поведение.

На рисунке 6 изображена диаграмма последовательностей для прецедента «Покупка продуктов».

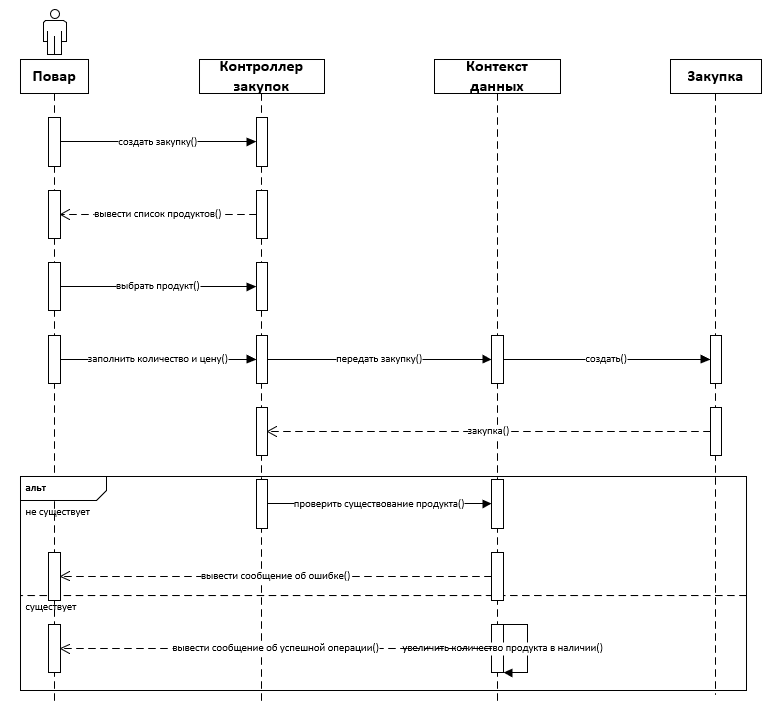


Рисунок 6. Диаграмма последовательностей для прецедента «Покупка продуктов»

На диаграмме приведено последовательность действий при выполнении прецедента «Покупка продуктов». Повар создает запрос на закупку, заполняет все поля на форме. Контроллер передает данные закупки, а контекст данных создает новый экземпляр класса «Закупка». Если продукт не существует, то система выведет ошибку. В ином случае, контекст данных добавляет закупку в БД и обновляет количество продукта в наличии в соответствии с закупленным.

2.5 Проектирование диаграмм состояний

Диаграмма состояний нужна для изображения состояний, в котором может находится бизнес-объект системы.

На рисунке 7 изображена диаграмма состояний продукта.

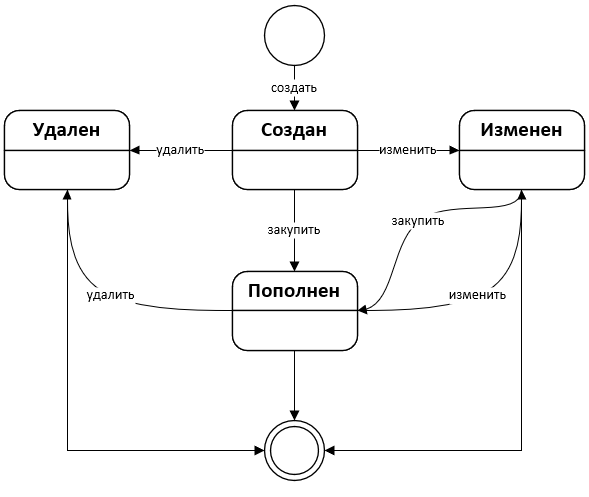


Рисунок 7. Диаграмма состояний продукта

Таким образом, продукт после создания в справочнике может быть изменен, пополнен при помощи закупки и удален. Удалить продукт можно из любого создания.

2.6 Проектирование диаграмм развёртывания

Диаграммы развертывания, или применения – это один из двух видов диаграмм, используемых при моделировании физических аспектов объектно-ориентированной системы. Такая диаграмма показывает конфигурацию узлов, где производится обработка информации, и то, какие компоненты размещены на каждом узле.

На рисунке 8 приведена диаграмма развёртывания системы «Помощник повара» на стороне клиента.

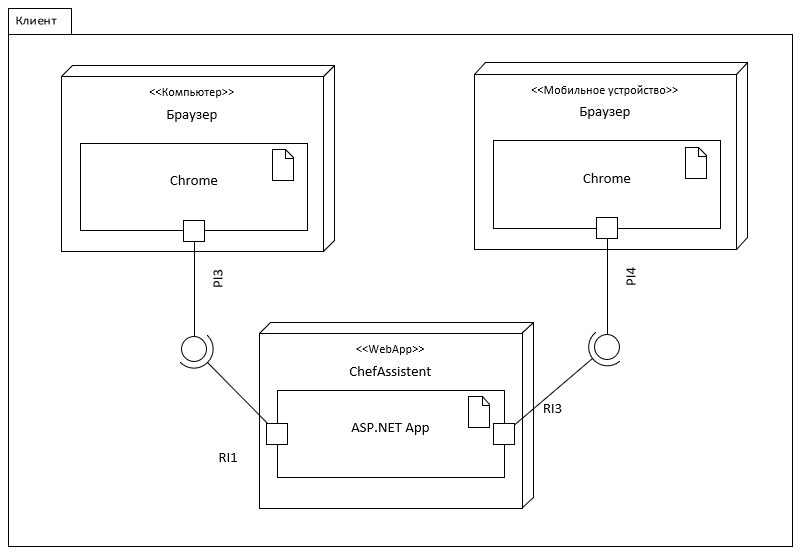


Рисунок 8. Диаграмма развёртывания системы «Помощник повара» стороне клиента

Так как система представляет собой веб-приложение, то при развертывании оно может быть запущено на любом устройстве, где есть веб-браузер.

На рисунке 9 приведена диаграмма развёртывания системы «Помощник повара» на сервере.

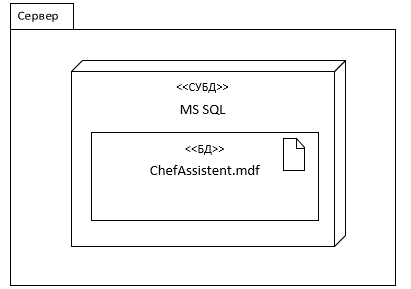


Рисунок 9. Диаграмма развёртывания системы «Помощник повара» на сервере

Таким образом, серверное развертывание включает в себя MDF-файл БД в СУБД MS SQL. Структура таблиц БД представлена в приложении А.

# 3 Реализация системы

3.1 Проектирование пользовательского интерфейса

Проектирование пользовательского интерфейса выполняется средствами ASP.NET. Для адаптивной верстки также используются возможности AJAX.

Для каждого класса предметной области разработан собственный блок представлений Index (список объектов), Create (добавить), Edit (изменить), Delete (удалить). На рисунке 10 представлен макет главной страницы системы.

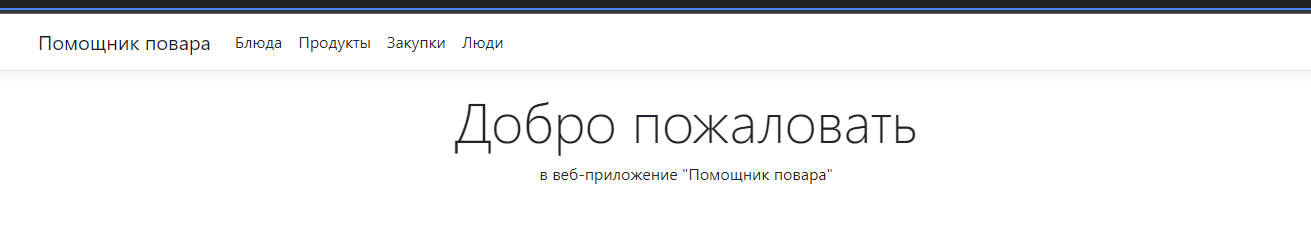


Рисунок 10. Макет главной страницы системы

На главной странице размещено приветственное сообщение. В шапке расположились разделы «Блюда», «Продукты», «Закупки», «Люди». При выборе раздела «Блюда» будет произведен переход на страницу блюд (рисунок 11).

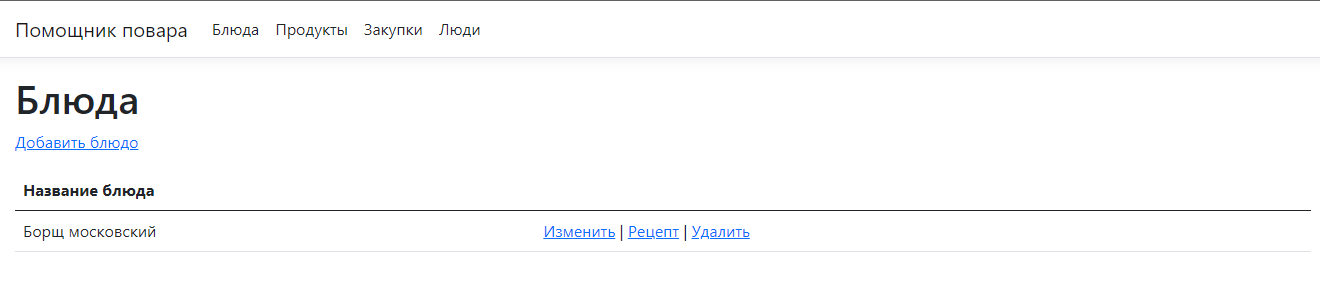


Рисунок 11. Макет страницы «Блюда»

На странице блюд выводится перечень названий блюд, рядом с которыми размещаются кнопки «Изменить», «Рецепт», «Удалить». При выборе пункта «Рецепт» будет произведен переход на страницу рецепта этого блюда (рисунок 12).

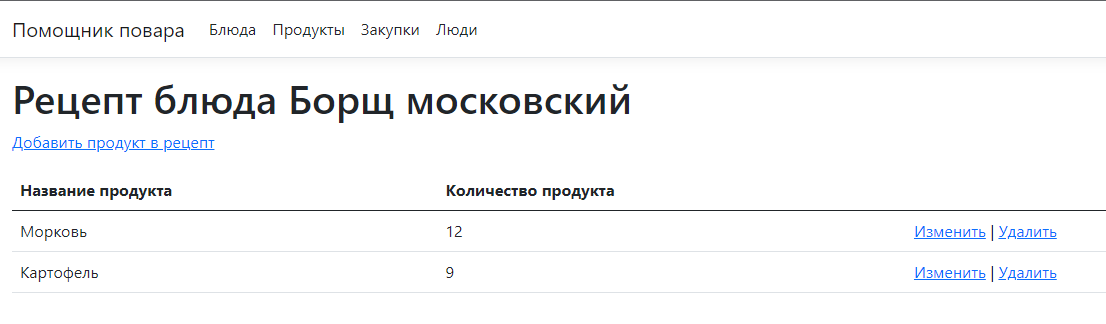


Рисунок 12. Макет страницы «Рецепт»

На странице рецепта выводится перечень названий и количества продуктов, который находятся в составе блюда. Каждую позицию рецепта можно изменить или удалить. Также можно добавить новый продукт в рецепт при помощи специальной страницы (рисунок 13).

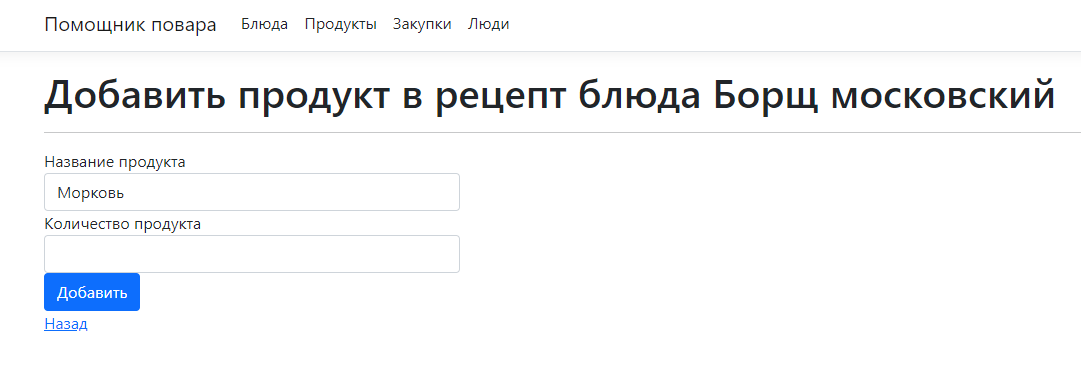


Рисунок 13. Макет страницы «Добавление продукта в рецепт»

На данной странице пользователь может указать в списке, какой именно продукта нужно добавить и в каком количестве.

К основным операция системы также относится возможность просмотра остатков продуктов. Эта информация выводится на специальной странице, представленной на рисунке 14.

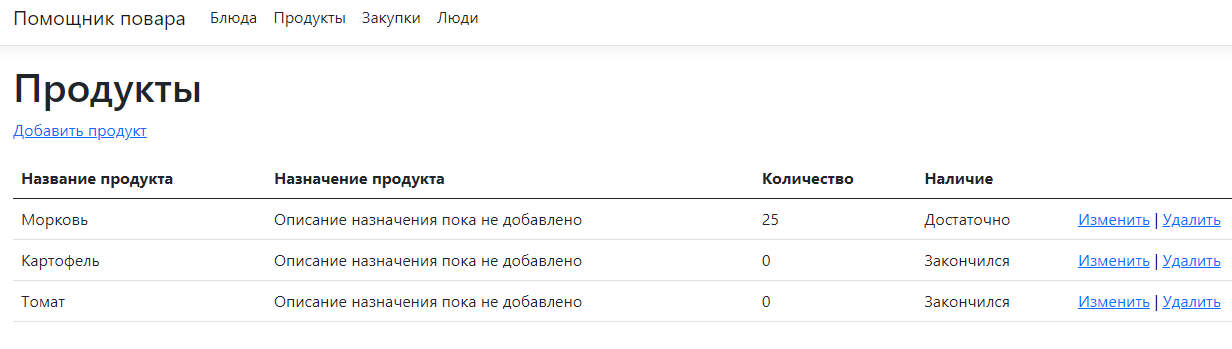


Рисунок 14. Макет страницы «Продукты»

Если количество продукта меньше 10 и больше 0, то система выведет текст «Заканчивается». Если количество продукта равно 0, то система выведет текст «Закончился». В иное случае, текст будет принимать значение «Достаточно».

Для закупки продуктов используется отдельная форма (рисунок 15).

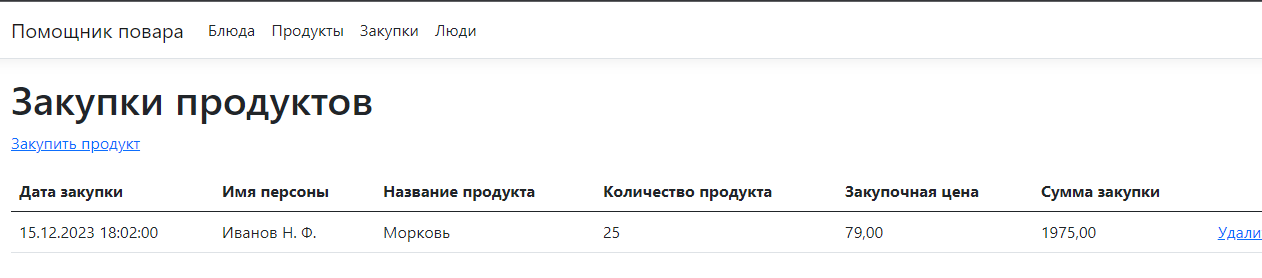


Рисунок 15. Макет страницы «Закупки»

На данной странице выводится перечень закупок с подсчетом итоговой суммы для каждой из них. Изменить закупку нельзя, но можно удалить или добавить. Чтобы добавить закупку, необходимо воспользоваться специальной страницей, представленной на рисунке 16.

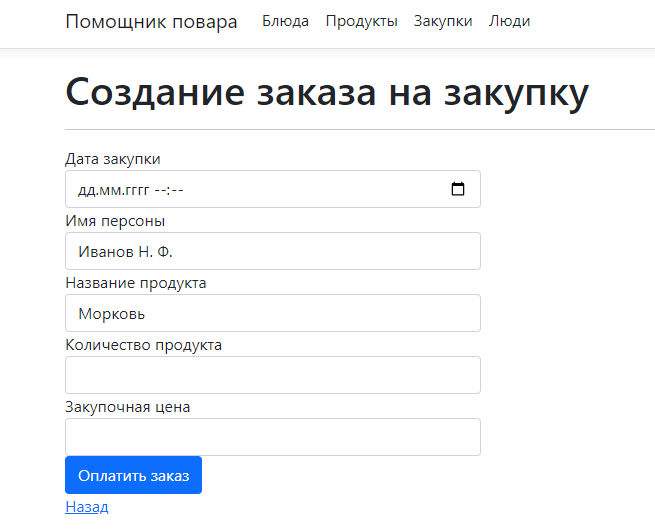


Рисунок 16. Макет страницы «Добавление закупки»

После создания заказа на закупку и его оплаты количество выбранного продукта на складе пополнится.

3.2 Реализация классов

Согласно диаграмме классов уровня реализации, было создано пять основных моделей: блюдо, персона, продукт, закупка и рецепт.

Структуру класса «Dish» можно увидеть ниже:

public class Dish

{

[Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[Required]

[MaxLength(255)]

[DisplayName("Название блюда")]

public string Title { get; set; }

public List<Recipe> Recipes { get; set; } = new List<Recipe>();

}

При помощи аннотаций для поля Id задается первичный ключ и автоинкремент. MaxLength позволяет задать максимальную длину строки. DisplayName подписывает имя поля так, как оно должно выглядеть на странице. Required помечает поле, как обязательное для заполнения.

Структура класса «Person»:

public class Person

{

[Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[Required]

[MaxLength(255)]

[DisplayName("Имя")]

public string Name { get; set; }

[Required]

[DisplayName("Возраст")]

public int Age { get; set; }

[Required]

[DisplayName("Пол")]

public bool Gender { get; set; }

[DisplayName("Пол")]

public string GenderName => Gender ? "Муж." : "Жен.";

public List<Purchase> Purchases { get; set; } = new List<Purchase>();

}

В данном классе есть свойство «Purchases», которое является списком закупок. В дальнейшем, это свойство будет использовано для создания связи между таблицами «Person» и «Purchase» при помощи Fluent API.

Структура класса « Product»:

public class Product

{

[Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[Required]

[MaxLength(255)]

[DisplayName("Название продукта")]

public string Title { get; set; }

[Required]

[MaxLength(255)]

[DisplayName("Назначение продукта")]

public string Nomination { get; set; }

[Required]

[DisplayName("Количество")]

public float Quantity { get; set; }

[DisplayName("Наличие")]

public string InStock => Quantity <= 10 && Quantity > 0 ? "Заканчивается" : Quantity == 0 ? "Закончился" : "Достаточно";

public List<Recipe> Recipes { get; set; } = new List<Recipe>();

public List<Purchase> Purchases { get; set; } = new List<Purchase>();

}

В данном классе свойства «Recipes» и «Purchases» также связывают таблицу «Product» с таблицами «Recipe» и «Purchase» соответственно.

Структура класса « Product»:

public class Purchase

{

[Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[Required]

[DisplayName("Дата закупки")]

public DateTime PurchaseDate { get; set; }

[Required]

[DisplayName("Имя персоны")]

public int PersonId { get; set; }

[Required]

[DisplayName("Имя персоны")]

public Person Person { get; set; }

[Required]

[DisplayName("Название продукта")]

public int ProductId { get; set; }

[Required]

[DisplayName("Название продукта")]

public Product Product { get; set; }

[Required]

[DisplayName("Количество продукта")]

public float Amount { get; set; }

[Required]

[DisplayName("Закупочная цена")]

public decimal Price { get; set; }

[DisplayName("Сумма закупки")]

public decimal TotalSum => (decimal)Amount \* Price;

}

В данном классе навигационными являются свойства «Product» и «Person» для обеспечения связи между таблицами. Свойство «TotalSum» является вычисляемым и отображает общую стоимость закупки.

Структура класса « Recipe»:

public class Recipe

{

[Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[Required]

[DisplayName("Название блюда")]

public int DishId { get; set; }

[Required]

[DisplayName("Название блюда")]

public Dish Dish { get; set; }

[Required]

[DisplayName("Название продукта")]

public int ProductId { get; set; }

[Required]

[DisplayName("Название продукта")]

public Product Product { get; set; }

[Required]

[DisplayName("Количество продукта")]

public float Amount { get; set; }

}

В данном классе навигационными являются свойства «Dish» и «Product» для обеспечения связи между таблицами.

Контекст данных отвечает за первоначальное создание таблиц на указанном сервере и позволяет работать с объектами класса, как с записями в таблице БД при помощи Entity Framework. Так выглядит создание таблиц и инициализация БД:

public MyContext()

{

Database.EnsureCreated();

}

public DbSet<Dish> Dishes { get; set; }

public DbSet<Person> People { get; set; }

public DbSet<Product> Products { get; set; }

public DbSet<Recipe> Recipes { get; set; }

public DbSet<Purchase> Purchases { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

base.OnConfiguring(optionsBuilder);

var connectionStringBuilder = new SqlConnectionStringBuilder();

connectionStringBuilder.DataSource = "DESKTOP-3HKK228";

connectionStringBuilder.InitialCatalog = "ChefAssistant";

connectionStringBuilder.IntegratedSecurity = true;

connectionStringBuilder.TrustServerCertificate = true;

optionsBuilder.UseSqlServer(connectionStringBuilder.ToString());

}

Также, при создании БД создаются внешние ключи в методе OnModelCreating:

model.Entity<Dish>()

.HasMany(\_ => \_.Recipes)

.WithOne(\_ => \_.Dish)

.HasForeignKey(\_ => \_.DishId)

.OnDelete(DeleteBehavior.Cascade);

model.Entity<Product>()

.HasMany(\_ => \_.Recipes)

.WithOne(\_ => \_.Product)

.HasForeignKey(\_ => \_.ProductId)

.OnDelete(DeleteBehavior.Cascade);

model.Entity<Product>()

.HasMany(\_ => \_.Purchases)

.WithOne(\_ => \_.Product)

.HasForeignKey(\_ => \_.ProductId);

model.Entity<Person>()

.HasMany(\_ => \_.Purchases)

.WithOne(\_ => \_.Person)

.HasForeignKey(\_ => \_.PersonId);

В качестве ограничений было созданы несколько уникальных индексов для запрета ввода одинаковых наименований блюд и продуктов, а также составной индекс для проверки на существование записи в таблице «Рецепт»:

model.Entity<Recipe>()

.HasIndex(\_ => new { \_.DishId, \_.ProductId })

.IsUnique();

model.Entity<Product>()

.HasIndex(\_ => \_.Title)

.IsUnique();

model.Entity<Dish>()

.HasIndex(\_ => \_.Title)

.IsUnique();

При первом запуске приложения некоторые таблицы будут заполнены тестовым набором данных:

model.Entity<Product>()

.HasData(

new Product() { Id = 1, Title = "Морковь", Quantity = 0, Nomination = "Описание назначения пока не добавлено" },

new Product() { Id = 2, Title = "Картофель", Quantity = 0, Nomination = "Описание назначения пока не добавлено" },

new Product() { Id = 3, Title = "Томат", Quantity = 0, Nomination = "Описание назначения пока не добавлено" }

);

model.Entity<Dish>()

.HasData(

new Dish() { Id = 1, Title = "Борщ московский" }

);

model.Entity<Recipe>()

.HasData(

new Recipe() { Id = 1, DishId = 1, ProductId = 1, Amount = 12 },

new Recipe() { Id = 2, DishId = 1, ProductId = 2, Amount = 9 }

);

model.Entity<Person>()

.HasData(

new Person() { Id = 1, Name = "Иванов Н. Ф.", Gender = true, Age = 28 }

);

## 3.3 Реализация контроллеров

Приведем пример работы одного контроллера, так как остальные работают аналогичным образом. Например, контроллер «Products» позволяет при помощи метода «Index» получать список продуктов из БД и выводить их на странице:

public async Task OnGetAsync()

{

if (\_context.Products != null)

{

Product = await \_context.Products.ToListAsync();

}

}

Получение данных из БД делается следующим образом:

1) Обращаемся к контексту данных.

2) Проверяем, если контекст существует, то возвращаем набор продуктов, как список.

Также, контроллер «Products» позволяет создавать новые продукты в справочнике при помощи метода «Create»:

public IActionResult OnGet()

{

return Page();

}

[BindProperty]

public Product Product { get; set; } = default!;

public async Task<IActionResult> OnPostAsync()

{

Product.Quantity = 0;

\_context.Products.Add(Product);

await \_context.SaveChangesAsync();

return RedirectToPage("./Index");

}

В методе «OnPostAsync» производится основная работа по созданию объекта.

Контроллер «Products» позволяет изменять данные существующих продуктов при помощи метода «Edit»:

public async Task<IActionResult> OnPostAsync()

{

\_context.Attach(Product).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ProductExists(Product.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToPage("./Index");

}

Если требуется удалить продукт из БД, то необходимо воспользоваться методом «Delete» и передать в него код продукта:

public async Task<IActionResult> OnPostAsync(int? id)

{

if (id == null || \_context.Products == null)

{

return NotFound();

}

var product = await \_context.Products.FindAsync(id);

if (product != null)

{

Product = product;

\_context.Products.Remove(Product);

await \_context.SaveChangesAsync();

}

return RedirectToPage("./Index");

}

Контекст данных в таком случае попытается найти продукт по его Id и в случае успешного нахождения удалит его.

Таким образом, была выполнена практическая реализация системы «Помощник повара».

# заключение

В ходе данной работы был проведен анализ предметной области, определены понятия и основные сущности программной системы «Помощник повара».

На этапе проектирования структуры системы были разработаны диаграммы прецедентов, классов, видов деятельности, последовательностей, состояний и развёртывания. Эти диаграммы являются ключевым инструментом для понимания взаимодействия компонентов системы, её архитектуры и логики работы.

В разделе «Реализация системы» представлены этапы создания системы «Помощник повара». Было выполнено проектирование пользовательского интерфейса, реализация классов и контроллеров.

Разработка программной системы «Помощник повара» направлена на решение современных проблем в сфере кулинарии. Уникальные возможности, такие как управление рецептами, планирование закупок делают систему удобным инструментом для любого пользователя.

Завершая эту работу, можно утверждать, что программная система «Помощник повара» обладает потенциалом и её функционал в дальнейшем может быть расширен за счёт внедрения возможности ведения личного кабинета повара с целью управления собственными рецептами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гради Буч, Рамбо Джеймс, Якобсон Ивар. Введение в UML от создателей языка. - М.: ДМК Пресс, 2015. - 496 c.

2. Камерон Р. ASP.NET 3.5, компоненты AJAX и серверные элементы управления для профессионалов. М.: Диалектика / Вильямс, 2016. - 587 c.

3. Пирогов В. Ю. SQL Server 2005: программирование клиент-серверных приложений. СПб.: BHV-Петербург, 2006. 336 стр.

4. Фримен Адам. ASP.NET 4.5 с примерами на C# 5.0 для профессионалов. - М.: Диалектика / Вильямс, 2014. - 278 c.

# Приложение А Структура таблиц БД

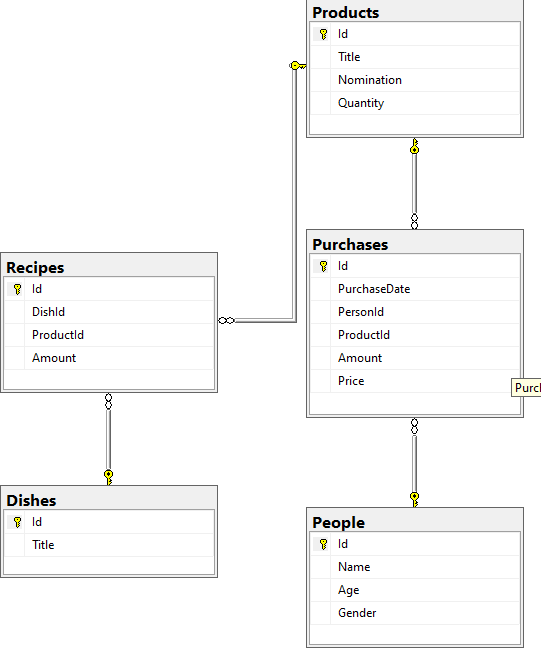


Рисунок А.1. Структура таблиц БД «Помощник повара»