**Государственное бюджетное профессиональное образовательное учреждение «Нижегородский Губернский колледж»**

Методическая комиссия «Информатика и вычислительная техника»

Допущен к защите:

преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_ Л.В. Мухина,

\_\_\_\_\_\_\_\_\_\_\_\_ Е.П. Голубева,

«21»\_\_марта\_\_2025 г.

**ОТЧЕТ** **ПО**

**УЧЕБНОЙ ПРАКТИКЕ**

ПМ.01 РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ

Руководитель: \_\_\_\_\_\_\_\_\_\_ Мухина Л.В., Голубева Е.П. 21.03.2025 г.

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Климычева Анастасия Артёмовна 21.03.2025 г.

Специальность, группа: 09.02.07, 43П

Нижний Новгород

2025 г.

**Содержание**

[Введение 3](#_Toc192219819)

[Основная часть 4](#_Toc192219820)

[Разработка базы данных 4](#_Toc192219821)

[Архитектура проекта и используемые технологии 5](#_Toc192219822)

[Вывод списка партнёров 7](#_Toc192219823)

[Экран редактирования партнёра 9](#_Toc192219824)

[Экран просмотра истории продаж партнёра 12](#_Toc192219825)

[Экран расчёта количества материала 14](#_Toc192219826)

[Разработка Dll 17](#_Toc192219827)

[Тестирование 20](#_Toc192219828)

[Разработка UML диаграмм. 24](#_Toc192219829)

[Заключение 26](#_Toc192219830)

[Приложения 27](#_Toc192219831)

[Приложение А. 27](#_Toc192219832)

# **Введение**

Целью данной учебной практики является выполнение четырех сессий демонстрационного экзамена 2025 года, как указано в документе КОД 09.02.07-2-2025 Том 1. Эти сессии соответствуют следующим модулям:

1. Разработка, администрирование и защита баз данных;
2. Разработка модулей программного обеспечения для компьютерных систем;
3. Сопровождение и обслуживание программного обеспечения компьютерных систем;
4. Интеграция программных модулей.

Каждый из указанных модулей предполагает выполнение конкретных задач, включая анализ требований, разработку программных решений, создание модульных тестов, описание тестовых сценариев и проектирование диаграмм. В рамках третьей сессии особое внимание уделяется тестовым сценариям для процедуры редактирования партнера, которые демонстрируют различные исходы работы алгоритма. Четвертая сессия акцентирует внимание на реализации метода для расчета количества материалов в виде библиотеки, а также на разработке проекта модульных тестов, что является важным этапом в обеспечении надежности и корректности программного обеспечения.

# **Основная часть**

## **Разработка базы данных**

На основе описания предметной области создана база данных в СУБД Postgres для разрабатываемой системы, содержащая только нужные таблица и связи для реализации системы. База данных находится в 3 нормальной форме с обеспечением ссылочной целостности.

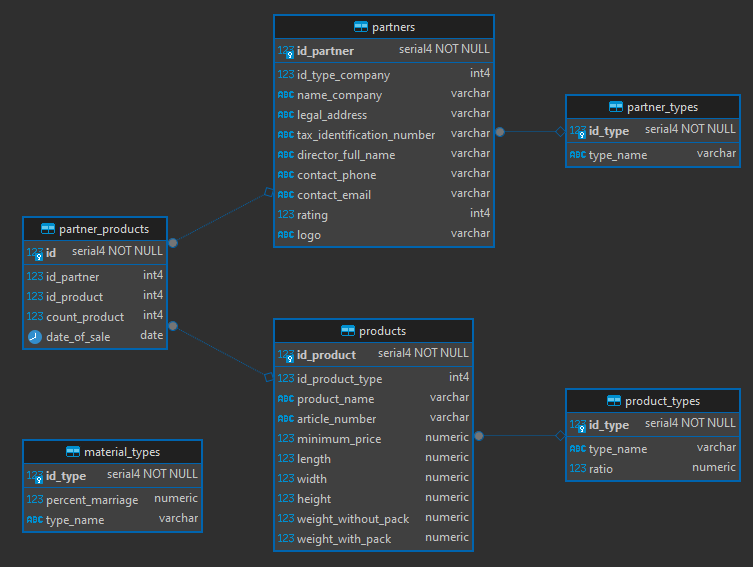


Рисунок №1 – Схема базы данных

Скрипт создания БД представлен в Приложении А, таблице 1.

## **Архитектура проекта и используемые технологии**

В разработанном приложении используется архитектура MVVM (Model-View-ViewModel), которая является одной из самых популярных архитектур для создания приложений с пользовательским интерфейсом. MVVM разделяет логику приложения на три основных компонента:

1. Model (Модель): этот компонент отвечает за бизнес-логику и управление данными. В нашем приложении модель реализована с использованием Entity Framework, который обеспечивает удобный доступ к базе данных и управление данными через объектно-ориентированный подход. Entity Framework позволяет работать с данными как с объектами, что упрощает взаимодействие с базой данных и уменьшает количество кода, необходимого для выполнения операций CRUD (создание, чтение, обновление, удаление).
2. View (Представление): представление отвечает за отображение данных пользователю и взаимодействие с ним. В нашем случае мы используем Avalonia, кроссплатформенный фреймворк для создания пользовательских интерфейсов. Avalonia позволяет создавать современные и отзывчивые интерфейсы, поддерживающие различные платформы, такие как Windows, macOS и Linux. Это обеспечивает широкую доступность приложения для пользователей.
3. ViewModel (Модель представления): ViewModel служит связующим звеном между моделью и представлением. Он обрабатывает пользовательский ввод, взаимодействует с моделью и обновляет представление. ViewModel реализует интерфейсы INotifyPropertyChanged и ICommand, что позволяет автоматически обновлять интерфейс при изменении данных и обрабатывать команды от пользователя.

Используемые технологии:

1. Avalonia: это кроссплатформенный фреймворк для создания пользовательских интерфейсов на .NET. Он поддерживает XAML для разметки интерфейса и предоставляет мощные инструменты для создания отзывчивых и современных приложений.
2. Entity Framework: это ORM (Object-Relational Mapping) библиотека для .NET, которая упрощает работу с базами данных. Она позволяет разработчикам работать с данными в виде объектов, а не в виде таблиц и строк, что значительно упрощает код и повышает его читаемость.
3. MessageBoxAvalonia: эта библиотека предоставляет функциональность для отображения диалоговых окон сообщений в приложениях на Avalonia. Она позволяет легко создавать и настраивать диалоговые окна для отображения информации пользователю или запроса подтверждения.
4. .NET 8: это последняя версия платформы .NET, которая включает в себя множество улучшений производительности, новых возможностей и обновлений для разработки приложений. Использование .NET 8 позволяет нам использовать современные функции языка C# и улучшенные библиотеки.
5. Community Toolkit: это набор инструментов и библиотек, разработанных для упрощения разработки приложений на .NET. Он включает в себя различные вспомогательные классы и функции, которые помогают в реализации паттернов проектирования и упрощают работу с данными и пользовательским интерфейсом.

В результате использования архитектуры MVVM и современных технологий, таких как Avalonia и Entity Framework, приложение обладает высокой модульностью, простотой в сопровождении и расширяемостью. Это позволяет легко вносить изменения и добавлять новые функции, а также обеспечивает удобный и интуитивно понятный интерфейс для пользователей.

## **Вывод списка партнёров**

В данном разделе рассмотрим настройку экрана вывода списков продуктов. Экран реализован с использованием архитектуры MVVM и фреймворка Avalonia, что обеспечивает четкое разделение логики приложения и пользовательского интерфейса.

1. Структура ViewModel:

* Класс PartnersListVM: это основной класс ViewModel, который отвечает за управление данными и логикой, связанной со списком партнеров (в данном случае, продуктов).
* Свойство \_partners: это приватное поле, помеченное атрибутом [ObservableProperty], которое хранит список объектов типа Partner. Это свойство автоматически уведомляет представление об изменениях, что позволяет обновлять интерфейс при изменении данных.
* Конструктор PartnersListVM: в конструкторе происходит инициализация списка партнеров. Данные загружаются из контекста базы данных с использованием Entity Framework, включая связанные данные (например, тип компании и продукты партнера) с помощью метода Include.

|  |
| --- |
| public PartnersListVM()  {  Partners = MainWindowViewModel.Context.Partners  .Include(it => it.IdTypeCompanyNavigation)  .Include(it => it.PartnerProducts)  .ToList();  } |

1. Команды для взаимодействия с пользователем

Создание, редактирование и удаление партнеров: ViewModel содержит несколько команд, которые позволяют пользователю взаимодействовать с данными:

* CreatePartner: открывает экран для создания нового партнера.
* EditPartner: открывает экран редактирования выбранного партнера.
* DeletePartner: удаляет выбранного партнера после подтверждения от пользователя через диалоговое окно.

|  |
| --- |
| [RelayCommand]  public async void DeletePartner(Partner partner)  {  // Логика удаления партнера с подтверждением  } |

1. Обработка событий и взаимодействие с интерфейсом

Использование MessageBox: для подтверждения действий пользователя (например, удаление партнера) используется библиотека MessageBoxAvalonia, которая позволяет отображать диалоговые окна с сообщениями.

Обновление текущей страницы: после выполнения команд, таких как создание или удаление партнера, ViewModel обновляет текущую страницу в MainWindowViewModel, что позволяет пользователю видеть актуальный список партнеров.

1. Привязка данных к представлению

XAML (представление): в XAML-файле используется привязка данных к свойству Partners из ViewModel. Это позволяет автоматически отображать список партнеров в интерфейсе, используя элемент управления ListBox.

|  |
| --- |
| <ListBox ItemsSource="{Binding Partners}">  <ListBox.ItemTemplate>  <DataTemplate>  <!-- Определение шаблона для отображения данных -->  </DataTemplate>  </ListBox.ItemTemplate>  </ListBox> |

## **Экран редактирования партнёра**

В данном разделе рассмотрим настройку экрана редактирования партнера. Экран реализован с использованием архитектуры MVVM и фреймворка Avalonia, что обеспечивает удобное взаимодействие пользователя с данными.

1. Структура ViewModel:

Класс PartnerEditorVM: это ViewModel, который отвечает за управление данными и логикой, связанной с созданием и редактированием партнера.

Свойства:

* NameScreen: строка, определяющая заголовок экрана. В зависимости от контекста (создание или редактирование) она может принимать значения "Добавление партнёра" или "Редактирование партнёра".
* NameButton: строка, определяющая текст на кнопке. Она также меняется в зависимости от контекста.
* \_partner: приватное поле, помеченное атрибутом [ObservableProperty], которое хранит объект типа Partner. Это свойство автоматически уведомляет представление об изменениях.
* \_partnerTypes: список типов партнеров, доступных для выбора.
* \_selPartnerType: выбранный тип партнера, который также помечен как наблюдаемое свойство.

|  |
| --- |
| [ObservableProperty]  private Partner \_partner = new Partner(); |

1. Конструкторы ViewModel

Конструктор без параметров: инициализирует данные для нового партнера, устанавливая заголовок экрана и текст кнопки на "Добавление партнёра" и "Добавить" соответственно. Также загружает доступные типы партнеров из базы данных.

|  |
| --- |
| public PartnerEditorVM()  {  InitData();  SelPartnerType = PartnerTypes.FirstOrDefault();  } |

Конструктор с параметром idPartner: используется для редактирования существующего партнера. Заголовок экрана и текст кнопки изменяются на "Редактирование партнёра" и "Редактировать". Данные партнера загружаются из базы данных по его идентификатору.

|  |
| --- |
| public PartnerEditorVM(int idPartner)  {  NameScreen = "Редактирование партнёра";  NameButton = "Редактировать";  InitData();  Partner = MainWindowViewModel.Context.Partners.First(it => it.IdPartner == idPartner);  SelPartnerType = Partner.IdTypeCompanyNavigation;  } |

1. Инициализация данных

Метод InitData: загружает список типов партнеров из базы данных и инициализирует свойство PartnerTypes. Это позволяет пользователю выбирать тип партнера из доступных вариантов.

|  |
| --- |
| private void InitData()  {  PartnerTypes = MainWindowViewModel.Context.PartnerTypes.ToList();  } |

1. Команды для взаимодействия с пользователем

Save: эта команда обрабатывает сохранение данных партнера. Если идентификатор партнера равен 0, это означает, что создается новый партнер, и он добавляется в базу данных. В противном случае, данные существующего партнера обновляются. После успешного сохранения пользователь получает уведомление через диалоговое окно.

|  |
| --- |
| [RelayCommand]  public async void Save()  {  try  {  Partner.IdTypeCompanyNavigation = SelPartnerType;  if (Partner.IdPartner == 0)  {  MainWindowViewModel.Context.Partners.Add(Partner);  MainWindowViewModel.Context.SaveChanges();  MainWindowViewModel.Instance.CurrentPage = new PartnersListView();  await MessageBoxManager.GetMessageBoxStandard("Сообщение", $"Партнёр '{Partner.NameCompany}' был успешно добавлен", MsBox.Avalonia.Enums.ButtonEnum.Ok).ShowAsync();  }  else  {  MainWindowViewModel.Context.SaveChanges();  MainWindowViewModel.Instance.CurrentPage = new PartnersListView();  await MessageBoxManager.GetMessageBoxStandard("Сообщение", $"Партнёр '{Partner.NameCompany}' был изменён", MsBox.Avalonia.Enums.ButtonEnum.Ok).ShowAsync();  }  }  catch (Exception ex)  {  await MessageBoxManager.GetMessageBoxStandard("Ошибка", $"{ex.Message}", MsBox.Avalonia.Enums.ButtonEnum.Ok).ShowAsync();  }  } |

GoBack: Эта команда позволяет пользователю вернуться на предыдущий экран (список партнеров) без сохранения изменений.

|  |
| --- |
| [RelayCommand]  public void GoBack()  {  MainWindowViewModel.Instance.CurrentPage = new PartnersListView();  } |

1. Привязка данных к представлению

XAML (представление): в XAML-файле используется привязка данных к свойствам ViewModel. Это позволяет автоматически отображать данные партнера и типы партнеров в интерфейсе, используя элементы управления, такие как TextBox для ввода данных и ComboBox для выбора типа партнера.

|  |
| --- |
| <TextBox Text="{Binding Partner.NameCompany}" />  <ComboBox ItemsSource="{Binding PartnerTypes}" SelectedItem="{Binding SelPartnerType}" /> |

## **Экран просмотра истории продаж партнёра**

В данном разделе рассмотрим настройку экрана истории партнера.

1. Структура ViewModel

Класс HistoryPartnerVM: это ViewModel, который отвечает за управление данными и логикой, связанной с отображением истории реализации продукции для конкретного партнера.

* Свойство NameScreen: Строка, определяющая заголовок экрана. По умолчанию устанавливается значение "История".
* \_listPartProds: Приватное поле, помеченное атрибутом [ObservableProperty], которое хранит список объектов типа PartnerProduct. Это свойство автоматически уведомляет представление об изменениях, что позволяет обновлять интерфейс при изменении данных.

|  |
| --- |
| [ObservableProperty]  private List<PartnerProduct> \_listPartProds = new(); |

1. Конструктор ViewModel

Конструктор с параметром id: этот конструктор используется для инициализации ViewModel с идентификатором партнера. Он загружает историю реализации продукции для данного партнера из базы данных и формирует заголовок экрана, включая название компании партнера.

|  |
| --- |
| public HistoryPartnerVM(int id)  {  ListPartProds = MainWindowViewModel.Context.PartnerProducts  .Where(it => it.IdPartner == id)  .Include(it => it.IdProductNavigation)  .ToList();  NameScreen = $"История реализации продукции партнёра \"{MainWindowViewModel.Context.Partners.FirstOrDefault(it => it.IdPartner == id).NameCompany}\"";  } |

1. Загрузка данных

Загрузка истории продукции: в конструкторе происходит выборка данных о продукции, связанной с конкретным партнером, из базы данных. Используется метод Include для загрузки связанных данных о продуктах, что позволяет получить полную информацию о каждой записи.

1. Команды для взаимодействия с пользователем

GoBack: эта команда позволяет пользователю вернуться на предыдущий экран (список партнеров) без каких-либо изменений. Она обновляет текущую страницу в MainWindowViewModel.

|  |
| --- |
| [RelayCommand]  public void GoBack()  {  MainWindowViewModel.Instance.CurrentPage = new PartnersListView();  } |

1. Привязка данных к представлению

XAML (представление): в XAML-файле используется привязка данных к свойству ListPartProds из ViewModel. Это позволяет автоматически отображать список истории реализации продукции в интерфейсе, используя элемент управления ListBox.

|  |
| --- |
| <ListBox ItemsSource="{Binding ListPartProds}">  <!-- Определение шаблона для отображения данных о продукции -->  </ ListBox > |

## **Экран расчёта количества материала**

В данном разделе рассмотрим настройку экрана расчета материалов. Экран реализован с использованием архитектуры MVVM и фреймворка Avalonia, что обеспечивает удобное взаимодействие пользователя с данными для расчета необходимых материалов.

1. Структура ViewModel

Класс CalculationMaterialVM: это ViewModel, который отвечает за управление данными и логикой, связанной с расчетом количества материалов, необходимых для производства продукции.

Свойства:

* \_quantityProduct: Приватное поле, помеченное атрибутом [ObservableProperty], которое хранит количество продукции, для которой будет производиться расчет.
* \_materialTypes: Список типов материалов, доступных для выбора.
* \_products: Список продуктов, для которых будет производиться расчет.
* \_materialType: Выбранный тип материала, который также помечен как наблюдаемое свойство.
* \_product: Выбранный продукт, для которого будет производиться расчет.
* \_result: Результат расчета, который будет отображаться пользователю.

|  |
| --- |
| [ObservableProperty]  private int \_quantityProduct = 0; |

1. Конструктор ViewModel

Конструктор CalculationMaterialVM: в этом конструкторе происходит инициализация данных для расчета. Загружаются доступные типы материалов и продукты из базы данных с использованием Entity Framework. Также устанавливаются значения по умолчанию для выбранного типа материала и продукта.

|  |
| --- |
| public CalculationMaterialVM()  {  MaterialTypes = MainWindowViewModel.Context.MaterialTypes.ToList();  Products = MainWindowViewModel.Context.Products.Include(it => it.IdProductTypeNavigation).ToList();  MaterialType = MaterialTypes.FirstOrDefault();  Product = Products.FirstOrDefault();  } |

1. Команды для взаимодействия с пользователем

Calculate: эта команда выполняет расчет необходимого количества материалов на основе введенных данных. Она использует класс CalculationClass, который содержит логику для вычисления количества материалов, необходимых для производства заданного количества продукции. Результат расчета сохраняется в свойстве Result.

|  |
| --- |
| [RelayCommand]  public void Calculate()  {  Result = \_calculation.GetQuantityForProduct((int)Product.IdProductType, MaterialType.IdType, QuantityProduct, (double)Product.Height, (double)Product.Width);  } |

GoBack: эта команда позволяет пользователю вернуться на предыдущий экран (список партнеров) без каких-либо изменений. Она обновляет текущую страницу в MainWindowViewModel.

|  |
| --- |
| [RelayCommand]  public void GoBack()  {  MainWindowViewModel.Instance.CurrentPage = new PartnersListView();  } |

1. Инициализация расчетного класса

Класс CalculationClass: Этот класс содержит логику для расчета количества материалов на основе различных параметров, таких как тип продукта, тип материала, количество продукции и размеры продукта. Он использует списки для хранения коэффициентов и процентов брака, что позволяет гибко управлять расчетами.

1. Привязка данных к представлению

XAML (представление): в XAML-файле используется привязка данных к свойствам ViewModel. Это позволяет автоматически отображать данные о типах материалов, продуктах и результатах расчетов в интерфейсе, используя элементы управления, такие как ComboBox для выбора типа материала и продукта, а также TextBox для ввода количества продукции.

|  |
| --- |
| <ComboBox ItemsSource="{Binding MaterialTypes}" SelectedItem="{Binding MaterialType}" />  <ComboBox ItemsSource="{Binding Products}" SelectedItem="{Binding Product}" />  <TextBox Text="{Binding QuantityProduct}" />  <TextBlock Text="{Binding Result}" /> |

## **Разработка Dll**

В данном разделе описывается процесс разработки динамической библиотеки (DLL), которая включает класс CalculationClass. Этот класс предназначен для выполнения расчетов необходимого количества материалов для производства продукции с учетом возможного брака. Библиотека используется в приложении для обеспечения точности расчетов.

1. Цели и задачи разработки DLL

Целью разработки данной DLL является создание модуля, который предоставляет функционал для расчета необходимого количества материалов на основе различных параметров. Основные задачи включают:

* Реализация класса CalculationClass, который содержит метод для расчета количества материалов.
* Обеспечение корректной работы метода с учетом возможных ошибок и исключительных ситуаций.
* Подготовка библиотеки к интеграции с основным приложением.

1. Структура DLL

DLL состоит из одного основного класса — CalculationClass, который включает:

* Списки типов материалов и типов продукции.
* Метод GetQuantityForProduct, который выполняет расчеты.

1. Реализация класса CalculationClass

Класс CalculationClass реализован следующим образом:

|  |
| --- |
| public class CalculationClass  {  // Список типов материалов с процентом брака  private List<(int IdType, double PercentMarriage)> \_materialTypes = new()  {  (1, 0.1), // 10% брака  (2, 0.95), // 95% брака  (3, 0.28), // 28% брака  (4, 0.55), // 55% брака  (5, 0.34) // 34% брака  };  // Список типов продукции с коэффициентами  private List<(int IdType, double Ratio)> \_productTypes = new()  {  (1, 2.35),  (2, 5.15),  (3, 4.34),  (4, 1.5)  };  // Конструктор класса  public CalculationClass() { }  // Метод для расчета необходимого количества материала  public int GetQuantityForProduct(int typeProduct, int typeMaterial, int quantityProduct, double weight, double height)  {  int quantityMaterial = 0;  double percent\_marriage = 0, ratio = 0;  // Проверка на существование типов и корректность входных данных  if (!\_materialTypes.Any(it => it.IdType == typeMaterial) ||  !\_productTypes.Any(it => it.IdType == typeProduct) ||  quantityProduct < 0 || weight < 0 || height < 0)  {  return -1; // Возврат -1 в случае ошибок  }  // Получение процента брака и коэффициента типа продукции  percent\_marriage = \_materialTypes.First(it => it.IdType == typeMaterial).PercentMarriage;  ratio = \_productTypes.First(it => it.IdType == typeProduct).Ratio;  // Расчет количества материала для одной единицы продукции  int quantityFor1Product = (int)((weight \* height) \* ratio);  // Общий расчет необходимого материала с учетом брака  quantityMaterial = (int)(quantityFor1Product \* quantityProduct \* (1.0 + percent\_marriage));  return quantityMaterial; // Возврат рассчитанного количества материала  }  } |

1. Логика метода GetQuantityForProduct

Проверка входных данных: метод проверяет, существуют ли указанные типы продукции и материалов, а также корректны ли значения для количества продукции и параметров (вес и высота). Если данные некорректны, метод возвращает -1.

Получение данных: если входные данные корректны, метод извлекает процент брака и коэффициент типа продукции из соответствующих списков.

Расчет: метод вычисляет количество материала, необходимое для производства одной единицы продукции, и затем умножает это значение на количество продукции с учетом процента брака.

1. Интеграция DLL в приложение

После завершения разработки DLL, она была интегрирована в основное приложение. Для этого:

* Библиотека была скомпилирована в DLL.
* Ссылка на созданную DLL была добавлена в проект приложения.
* Класс CalculationClass и его метод GetQuantityForProduct использовались для выполнения расчетов.

## **Тестирование**

В данном разделе описываются разработанные юнит-тесты для класса CalculationClass, который отвечает за расчет необходимого количества материалов для производства продукции. Тесты проверяют корректность работы метода GetQuantityForProduct в различных сценариях, включая как нормальные, так и крайние случаи.

1. Структура юнит-тестов

Юнит-тесты организованы в классе UnitTests, который использует атрибуты из библиотеки Microsoft.VisualStudio.TestTools.UnitTesting. Каждый тест проверяет определенный аспект работы метода GetQuantityForProduct.

1. Тесты на некорректные входные данные

Тест на несуществующий тип продукции:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_NonExistentProductType()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(8, 1, 15, 20, 45);  int correctResult = -1;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на несуществующий тип материала:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_NonExistentMaterialType()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 8, 15, 20, 45);  int correctResult = -1;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на отрицательное количество продукции:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_NegativeProductQuantity()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 4, -4, 20, 45);  int correctResult = -1;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на отрицательную ширину продукции:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_NegativeProductWidth()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 4, 15, -20, 45);  int correctResult = -1;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на отрицательную длину продукции:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_NegativeProductLength()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 4, 15, 20, -45);  int correctResult = -1;  Assert.AreEqual(actualResult, correctResult);  } |

1. Тесты на корректные входные данные

Тесты на корректные расчеты для различных типов продукции и материалов:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_MaterialType1ProductType1\_CorrectResult()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(1, 1, 15, 20, 45);  int correctResult = 34897;  Assert.AreEqual(actualResult, correctResult);  }  [TestMethod]  public void GetQuantityForProduct\_MaterialType2ProductType1\_CorrectResult()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(2, 1, 15, 20, 45);  int correctResult = 76477;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на случай, когда ширина и длина равны нулю:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_ZeroWidthAndLength\_CorrectResult()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 2, 15, 0, 0);  int correctResult = 0;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на малые значения:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_SmallValues\_CorrectResult()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 2, 1, 1, 1);  int correctResult = 7;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на большие значения:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_LargeProduct\_CorrectResult()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 2, 2, 500, 500);  int correctResult = 4231500;  Assert.AreEqual(actualResult, correctResult);  } |

1. Тесты на тип возвращаемого значения

Тест на возврат значения -1 при передаче null:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_NullValue\_ReturnMinusOne()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(Convert.ToInt32(null), Convert.ToInt32(null),  Convert.ToInt32(null), Convert.ToInt32(null), Convert.ToInt32(null));  int correctResult = -1;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на тип возвращаемого значения:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_ReturnInt()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(2, 1, 15, 20, 45);  Assert.IsInstanceOfType<int>(actualResult);  } |

Также был протестирован экран редактирования партнёра.

Список тестов:

1. Редактирование партнера с указанием отрицательного рейтинга;
2. Редактирование партнера с полной информацией;
3. Редактирование партнера с указанием рейтинга нечисловыми значениями;
4. Редактирование партнера без информации;
5. Кнопка редактирования изменяет цвет при нажатии;

Подробнее с ui-тестами можно ознакомиться в Приложении \*

## **Разработка UML диаграмм.**

Для системы фитнес-центра были разработаны две диаграммы.

Диаграмма использования иллюстрирует взаимодействие пользователей (клиентов, тренеров и администраторов) с системой, как показано на рисунке №\*. Неавторизованный пользователь имеет возможность только авторизоваться в системе. Клиент, в свою очередь, может просмотреть список тренеров и оставить заявку, которая обязательно должна содержать указание цели. После этого клиент может просматривать свои заявки, доступные занятия и отслеживать свой прогресс. Администратор имеет возможность добавлять новых тренеров в систему, записывать их на курсы и предоставлять скидки постоянным клиентам. Тренер может просматривать список своих текущих клиентов, а также новые заявки, которые он может принять или отклонить с указанием причины.

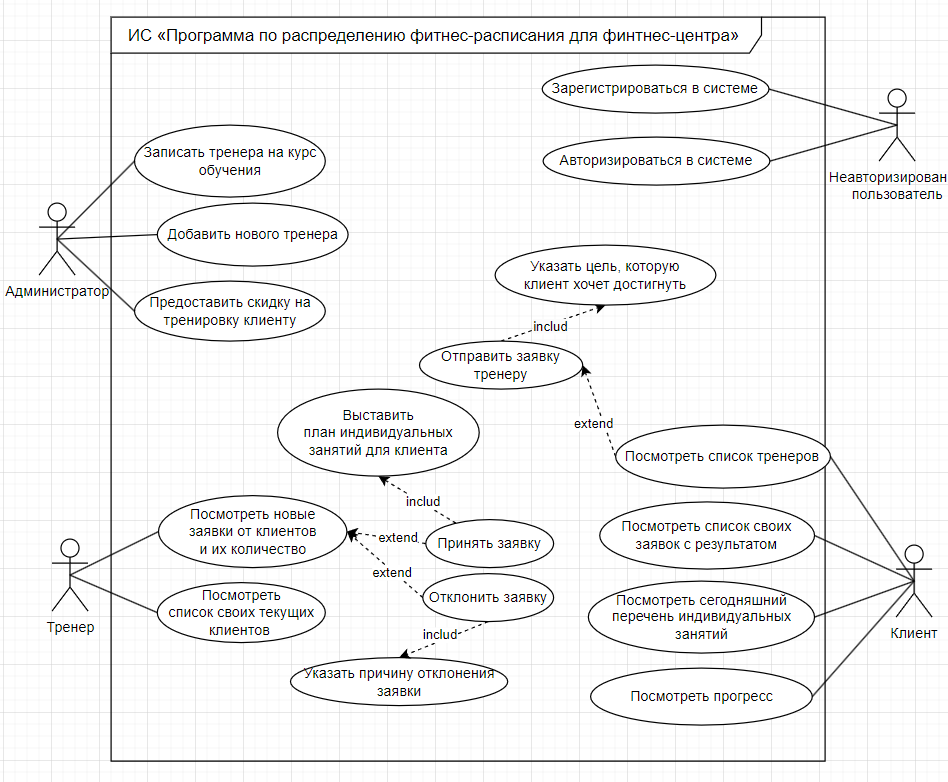


Рисунок №\* – Диаграмма использования.

Диаграмма деятельности иллюстрирует процессы, такие как выбор тренера и получение плана тренировок, и представлена на рисунке №\*. После авторизации в системе клиент может выбрать тренера и отправить ему заявку. В ответ на это действие система одновременно отображает список заявок для клиента и отправляет заявку тренеру, который может ее просмотреть. Тренер принимает решение либо отклонить заявку, указав причину отказа, после чего система обновляет список заявок у клиента, либо принять заявку и выставить план тренировок. В этом случае система отображает пользователю две новые вкладки, позволяющие ознакомиться с перечнем индивидуальных занятий или планом тренировок.

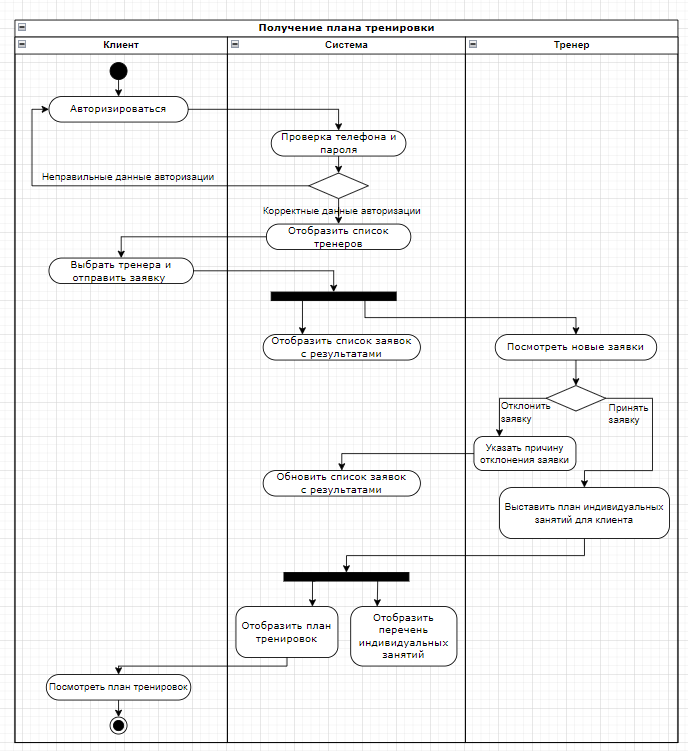


Рисунок №\* – Диаграмма деятельности.

# **Заключение**

В процессе выполнения задания были успешно решены все поставленные задачи, соответствующие четырем модулям демонстрационного экзамена.

В рамках первой сессии был проведен анализ требований к разработке, администрированию и защите баз данных, что способствовало формированию четкого понимания структуры и функциональности системы.

Во второй сессии были разработаны модули программного обеспечения, соответствующие требованиям предметной области.

Третья сессия включала описание пяти тестовых сценариев для процедуры редактирования партнера, охватывающих различные варианты исходов работы алгоритма.

В четвертой сессии был реализован метод для расчета количества материалов в виде библиотеки, а также разработан проект модульных тестов, состоящий из 15 методов (10 с низкой сложностью и 5 с высокой сложностью).

На основе анализа файла «Задание на проектирование» были составлены диаграммы использования и деятельности, что позволило визуализировать процессы и взаимодействия в системе.

Все результаты работы были систематизированы и представлены в репозитории с подробным описанием в файле README.md.

Выполнение данного задания способствовало закреплению навыков работы с базами данных, разработки программных модулей, модульного тестирования и проектирования UML-диаграмм.

# **Приложения**

## **Приложение А.**

Таблица №1 – Скрипт БД

|  |
| --- |
| -- public.partner\_types определение  CREATE TABLE public.partner\_types (  id\_type serial4 NOT NULL,  type\_name varchar NULL,  CONSTRAINT partner\_types\_pk PRIMARY KEY (id\_type)  );  -- public.product\_types определение  CREATE TABLE public.product\_types (  id\_type serial4 NOT NULL,  type\_name varchar NULL,  ratio numeric NULL,  CONSTRAINT product\_types\_pk PRIMARY KEY (id\_type)  );  -- public.material\_types определение  CREATE TABLE public.material\_types (  id\_type serial4 NOT NULL,  percent\_marriage numeric NULL,  type\_name varchar NULL,  CONSTRAINT material\_types\_pk PRIMARY KEY (id\_type)  );  -- public.partners определение  CREATE TABLE public.partners (  id\_partner serial4 NOT NULL,  id\_type\_company int4 NULL,  name\_company varchar NULL,  legal\_address varchar NULL,  tax\_identification\_number varchar NULL,  director\_full\_name varchar NULL,  contact\_phone varchar NULL,  contact\_email varchar NULL,  rating int4 NULL,  logo varchar NULL,  CONSTRAINT partners\_pk PRIMARY KEY (id\_partner),  CONSTRAINT partners\_partner\_types\_fk FOREIGN KEY (id\_type\_company) REFERENCES public.partner\_types(id\_type) ON DELETE RESTRICT ON UPDATE RESTRICT  );  -- public.products определение  CREATE TABLE public.products (  id\_product serial4 NOT NULL,  id\_product\_type int4 NULL,  product\_name varchar NULL,  article\_number varchar NULL,  minimum\_price numeric DEFAULT 0 NULL,  length numeric DEFAULT 0 NULL,  width numeric DEFAULT 0 NULL,  height numeric DEFAULT 0 NULL,  weight\_without\_pack numeric DEFAULT 0 NULL,  weight\_with\_pack numeric DEFAULT 0 NULL,  CONSTRAINT products\_pk PRIMARY KEY (id\_product),  CONSTRAINT products\_product\_types\_fk FOREIGN KEY (id\_product\_type) REFERENCES public.product\_types(id\_type) ON DELETE CASCADE ON UPDATE CASCADE  );  -- public.partner\_products определение  CREATE TABLE public.partner\_products (  id serial4 NOT NULL,  id\_partner int4 NULL,  id\_product int4 NULL,  count\_product int4 NULL,  date\_of\_sale date NULL,  CONSTRAINT partner\_products\_pk PRIMARY KEY (id),  CONSTRAINT partner\_products\_partners\_fk FOREIGN KEY (id\_partner) REFERENCES public.partners(id\_partner) ON DELETE CASCADE ON UPDATE CASCADE,  CONSTRAINT partner\_products\_products\_fk FOREIGN KEY (id\_product) REFERENCES public.products(id\_product) ON DELETE CASCADE ON UPDATE CASCADE  ); |