Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

**ХХ**

Курсовая работа

по дисциплине «Разработка мобильных приложений»

Пояснительная записка

Студент гр. 584-М

\_\_\_\_\_ Зорин Д. Л.

«\_\_» \_\_\_\_\_\_\_\_ 2025 г.

Руководитель:

преподаватель каф. КСУП

\_\_\_\_\_ \_\_\_\_\_ Мурзин Е. С.

«\_\_» \_\_\_\_\_\_\_\_ 2025 г.

Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

УТВЕРЖДАЮ

Заведующий кафедрой КСУП, д-р техн. наук, профессор

Ю.А. Шурыгин

“ ” 2025 г.

**ЗАДАНИЕ**

на курсовую работу по дисциплине «Разработка мобильных приложений» студенту Зорину Даниилу Леонидовичу группы 584-М факультета вычислительных систем.

* 1. Тема работы: **ХХ**
  2. Исходные данные к работе:
     1. Техническое задание
  3. Срок сдачи студентом законченной работы: до **ХХ** декабря 2025 г.
  4. Содержание курсовой работы:

Пояснительная записка должна быть оформлена в соответствии со стандартом ТУСУР.

В конверте на обложке приложить диск с БД, исходными текстами программы с соответствующими файлами, исполнительными файлами, пояснительной запиской.

6 Дата выдачи задания:

«16» апреля 2025 г.

Задание согласовано:

Руководитель работы Мурзин Евгений Сергеевич

“ ” 2025 г.

Задание принято к исполнению

“ ” 2025 г.

**Реферат**

**Сокращения, обозначения, термины и определения**

**Оглавление**

[**Введение** 8](#_Toc200320174)

[**1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ** 9](#_Toc200320175)

[**1.1 Функциональные требования** 9](#_Toc200320176)

[**1.3 Стек технологий** 9](#_Toc200320177)

[**1.3.1 Язык программирования** 9](#_Toc200320178)

[**1.3.2 Платформа разработки** 10](#_Toc200320179)

[**1.3.3 Разработка настольных кроссплатформенных приложений** 12](#_Toc200320180)

[**1.3.5 Обоснование выбора технологий** 12](#_Toc200320181)

[**2 РАЗРАБОТКА ПРИЛОЖЕНИЯ** 13](#_Toc200320182)

[**2.1 Бизнес-логика приложения** 13](#_Toc200320183)

[**2.1.1 Проектирование системы** 13](#_Toc200320184)

[**2.2 Пользовательский интерфейс** 13](#_Toc200320185)

[**2.3 Сборка и развертывание приложения** 16](#_Toc200320186)

[**3 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ** 17](#_Toc200320187)

[**3.1 Тестирование программного обеспечения** 17](#_Toc200320188)

[**3.2 Запуск приложения на требуемых ОС** 19](#_Toc200320189)

[**3.3 Unit-тестирование** 20](#_Toc200320190)

[**Заключение** 21](#_Toc200320191)

[**Список использованных источников** 22](#_Toc200320192)

**Введение**

**1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ**

**1.1 Функциональные требования**

**1.2 Стек технологий**

**1.2.1 Язык программирования**

C# – это современный объектно-ориентированный язык программирования, разработанный корпорацией Microsoft. Язык программирования был представлен в 2000 году и с тех пор стал одним из основных языков для создания приложений под платформу .NET [5].

Все элементы программы, такие как классы, объекты, наследование и инкапсуляция, используются для организации кода. C# является строго типизированным языком, что означает, что переменные должны быть объявлены с указанием их типа до использования, что способствует безопасности и уменьшает количество ошибок в процессе выполнения. Код на C# компилируется в промежуточный язык (IL), который затем выполняется с использованием виртуальной машины CLR. Это обеспечивает управление памятью, сборку мусора и другие важные аспекты, что делает C# языком с управляемым кодом. C# тесно связан с платформами .NET Framework и .NET Core, предоставляющими множество библиотек и инструментов для разработки приложений. Синтаксис C# напоминает синтаксис таких языков, как C++ и Java, что делает его относительно простым для изучения и использования [5].

C# предоставляет множество возможностей для разработчиков, охватывая широкий спектр сфер от веб-разработки и настольных приложений до создания мобильных приложений и игр.

C# автоматически управляет памятью с помощью механизма сборки мусора, что облегчает разработку, уменьшая риск утечек памяти и снижая сложности работы с динамической памятью. C# поддерживает события и делегаты, что позволяет разработчикам легко реализовывать паттерн "наблюдатель" и обрабатывать события в приложениях, например, в графических интерфейсах пользователя. Используя ключевые слова "async" и "await", C# обеспечивает эффективное асинхронное программирование, что особенно полезно для разработки приложений с высокой производительностью и отзывчивостью. LINQ предоставляет выразительный синтаксис для запросов к данным, позволяя разработчикам удобно работать с коллекциями данных, базами данных и другими источниками данных. C# предоставляет механизм обработки исключений, позволяющий писать безопасный код, который может обрабатывать ошибки и неожиданные ситуации. C# может взаимодействовать с другими языками, библиотеками и технологиями, такими как C++, COM, Win32 API, JavaScript и другими, благодаря различным механизмам интероперабельности.

**1.2.2 Платформа разработки**

.NET — это бесплатная, кроссплатформенная и открытая платформа разработки для создания приложений. Она поддерживает множество языков программирования, среди которых наиболее популярен C#. Платформа обеспечивает высокую производительность и широко используется для масштабируемых приложений в рабочей среде [6].

.NET разработана для обеспечения высокой производительности, безопасности и надежности. Платформа управляет памятью с помощью сборщика мусора (GC) и использует строгие компиляторы для обеспечения типобезопасности. Включает обширные библиотеки, оптимизированные для работы на различных операционных системах и архитектурах.

.NET включает высокую производительность за счет полного набора средств выполнения, библиотек и инструментов, обеспечивает безопасность кода с возможностью ручной оптимизации, существует поддержка статического и динамического кода для различных сценариев, имеет встроенное взаимодействие с аппаратными компонентами и низкоуровневыми API, обеспечивает портируемость кода на разные платформы и адаптивность для различных областей, таких как облачные решения, клиентские приложения и игры.

Платформа поддерживается корпорацией Майкрософт и сообществом, регулярно обновляется для обеспечения безопасности и надежности.

Основные компоненты .NET:

* среда выполнения, исполняющая код приложения.
* библиотеки, предоставляющие различные функции, такие как синтаксический анализ JSON.
* компиляторы, преобразующие исходный код на C# и других языках в исполняемый.
* SDK и инструменты для разработки и мониторинга приложений.
* стек приложений, таких как ASP.NET Core и Windows Forms, для создания различных типов приложений.

C# является основным языком программирования для .NET, и значительная часть платформы написана на этом языке. Библиотеки предоставляют множество типов, интегрированных с C#, например, оператор «foreach» для перечисления коллекций и оптимизации работы с обобщениями типа «List<T>».

Поддержка многозадачности является ключевой для .NET, что позволяет приложениям эффективно обрабатывать множество одновременных задач. Система типов поддерживает объектно-ориентированное программирование с использованием наследования, интерфейсов и обобщенных типов.

Сборщик мусора (GC) в .NET автоматически управляет памятью и оптимизирует её использование для различных рабочих нагрузок. Типы значений и выделение памяти в стеке позволяют более низкоуровневый контроль над данными. Рефлексия позволяет программам динамически запрашивать и вызывать другие части программы.

Исключения используются для обработки ошибок, что упрощает управление ими и повышает надежность приложений. Современные инструменты разработки, такие как SDK, обеспечивают удобный процесс разработки и поддержки непрерывной интеграции (CI). NuGet, диспетчер пакетов для .NET, предоставляет доступ к тысячам пакетов для различных функциональных возможностей, поддерживаемых корпорацией Майкрософт.

**1.2.3 Разработка настольных кроссплатформенных приложений**

**1.2.4 Обоснование выбора технологий**

Для реализации проекта был выбран .NET Core 6.0 и язык программирования C# с фреймворками Avalonia UI и ReactiveUI по причинам наличия кроссплатформенности, высокой производительности, возможности расширяемости и гибкости создания графического интерфейса пользователя с использованием «axaml», что делает его привлекательным выбором для разработки современных и интерактивных пользовательских интерфейсов, наличия возможности реактивного программирования, а также имеющегося опыта работы с этой платформой. Данный стек технологий имеет активные сообщества разработчиков, большую экосистему инструментов и библиотек, что облегчает разработку, поддержку и масштабирование проекта.

**2 РАЗРАБОТКА ПРИЛОЖЕНИЯ**

**2.1 Бизнес-логика приложения**

**2.1.1 Проектирование системы**

**2.2 Пользовательский интерфейс**

Общую архитектуру приложения задает шаблон проектирования Модель-Представление-Модель Представления (далее, MVVM). MVVM — это архитектурный шаблон, широко используемый в разработке приложений с графическим пользовательским интерфейсом, особенно в таких фреймворках, как WPF, Xamarin, .NET MAUI и Avalonia [16]. Паттерн разделяет логику приложения на три основные компоненты: модель (Model), представление (View) и модель представления (ViewModel), рисунок 2.16.

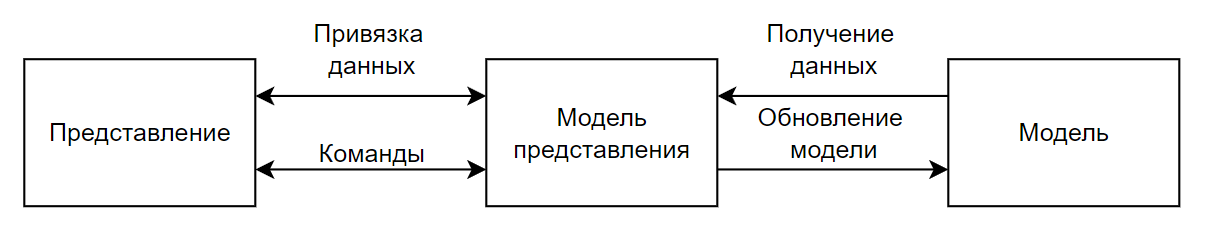


Рисунок 2.16 – схема MVVM

Модель – это объектное представление имеющихся данных. В паттерне MVVM модели концептуально совпадают с моделями внутри DAL. Данными являются классы бизнес-логики.

Представление – это пользовательский интерфейс приложения, который спроектирован так, чтобы быть легковесным. В Avalonia представлением являются «axaml» файлы.

Модель представления связывает модель и представление. Модель представления содержит логику представления и команды, необходимые для взаимодействия с представлением. Она выступает посредником, обрабатывая пользовательские действия и обновляя модель и представление соответственно. Модель представления использует механизм привязки данных для автоматического обновления интерфейса в ответ на изменения в модели.

Также используется паттерн «Наблюдатель» (Observer), это поведенческий шаблон проектирования, который определяет зависимость типа "один ко многим" между объектами, так что когда один объект изменяет своё состояние, все зависимые от него объекты оповещаются и обновляются автоматически [17]. Этот паттерн используется для реализации механизма подписки на события.

Основные компоненты паттерна "Наблюдатель":

* **IObservable, это о**бъект, за состоянием которого наблюдают. Он хранит список наблюдателей и предоставляет методы для добавления и удаления наблюдателей. При изменении состояния субъект уведомляет всех своих наблюдателей.
* **IObserver, это** интерфейс или абстрактный класс, который определяет метод обновления, вызываемый субъектом для оповещения наблюдателей о изменении состояния.
* **ConcreteObservable, это к**онкретная реализация субъекта. Хранит состояние, представляющее интерес для наблюдателей, и оповещает их при изменении этого состояния.
* **ConcreteObserver, это к**онкретная реализация наблюдателя. Хранит ссылку на конкретный субъект и обновляет своё состояние в соответствии с изменениями состояния субъекта.

UML диаграмма данного шаблона представлена на рисунке 2.17.

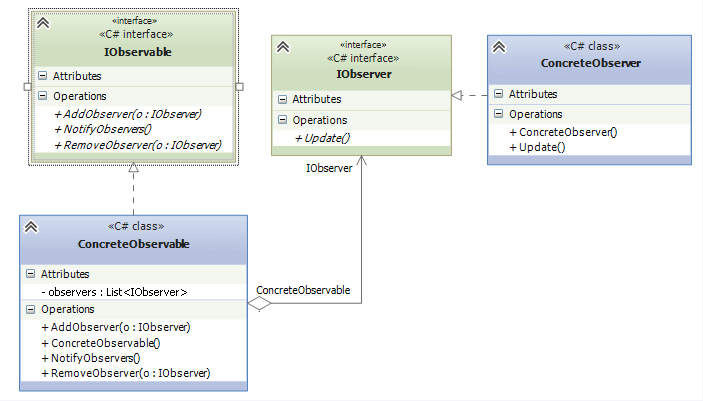


Рисунок 2.17 – Паттерн "Наблюдатель" (Observer)

Для верстки пользовательского интерфейса были составлены блок-схема решения основных задач пользователя, показанная на рисунке 2.18, а также сценарий использования, представленный на рисунке 2.19.

Рисунок 2.18 – Блок схема действий пользователя

Рисунок 2.19 – Сценарий использования

Далее создаются представления и модели представления для окон…

UML диаграмма классов пользовательского интерфейса показана на рисунке 2.27.

Рисунок 2.27 – Концептуальная диаграмма классов модели представления и представления

После отрисовки всех окон реализуется логика взаимодействия между элементами и окнами в модели представления и «Code Behind». Для этого используется библиотека ReactiveUI, создаются подписки на ввод текста в окна поиска элементов внутри «ListBox» и на изменения визуального вида элементов отображения данных, результат представлен на рисунке 2.28.

**2.3 Сборка и развертывание приложения**

**3 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ**

**3.1 Тестирование программного обеспечения**

Тестирование программного обеспечения — это процесс анализа или использования ПО с целью выявления дефектов. Оно включает два основных типа деятельности: статическое и динамическое тестирование [24].

Статическое тестирование заключается в анализе и проверке кода без его запуска, что включает ревью кода и контрольные проверки. Динамическое тестирование предполагает запуск программы и проверку её работы в реальных условиях.

Оба типа тестирования дополняют друг друга, используя разные подходы к обнаружению ошибок. Ошибка в программном обеспечении — это дефект, который вызывает расхождение между ожидаемыми и фактическими результатами.

Тестирование программного обеспечения выполняет две основные функции: верификацию и валидацию. Верификация обеспечивает соответствие результатов разработки на каждой стадии процесса заданным требованиям. Валидация гарантирует, что программное обеспечение удовлетворяет всем системным требованиям. Цель валидации — убедиться, что каждая функция программы соответствует конкретным требованиям заказчика и может быть отслежена до них.

Тестирование при разработке программного обеспечивания позволяет выявлять и исправлять ошибки до выпуска продукта, уменьшить вероятность дефектов в рабочей среде, гарантировать, что продукт соответствует ожиданиям и требованиям пользователей, а также обеспечить стабильность и производительность продукта.

Существует множество видов тестирования, такие как:

* функциональное тестирование используется для проверки правильности работы функциональных требований системы. Включает тестирование на уровне компонентов, интеграции и приемочные тесты;
* нефункциональное тестирование, проверяются аспекты, не связанные с конкретными функциями, такие как производительность, безопасность, удобство использования и совместимость;
* ручное тестирование, тесты выполняют тестировщики вручную без использования автоматизированных инструментов.
* автоматизированное тестирование, для выполнения тестов используются скрипты и инструменты. Повышает эффективность и точность, особенно при выполнении регрессионного тестирования.

Существуют различные автоматизированные инструменты тестирования, такие как:

* Selenium, инструмент для автоматизации веб-приложений.
* JUnit и NUnit, инструменты для модульного тестирования в Java и .NET соответственно.
* Appium, инструмент для автоматизации тестирования мобильных приложений.
* JMeter, инструмент для нагрузочного тестирования.

**3.2 Запуск приложения на требуемых ОС**

**3.3 Unit-тестирование**

Unit-тестирование – тестирование минимальных модулей архитектуры, максимально изолированных друг от друга. Так как минимальными модулями архитектуры как правило являются классы, юнит-тестирование фактически является тестированием исходного кода. В отличии от регрессионного тестирования, юнит-тесты пишутся на языке программирования программы в том же проекте [25]. Написание юнит-тестов – обязанность разработчиков.

В отличии от других видов тестов, юнит-тесты как правило составляются на основе разработанных алгоритмов, а не на основе технического задания. Разработчик, написавший некоторый алгоритм, определяет необходимое количество тестов, проверяющих все возможные варианты исходных данных и результатов выполнения. Степень покрытия исходного кода юнит-тестами в различных компаниях может отличаться и зависит от многих организационных факторов.

Unit-тестированию подвергается поведение класса, доступное через модификаторы доступа «public» или «protected». Тестовые случаи определяются на основе цикломатической сложности.

Цикломатическая сложность алгоритма (функции, метода) — это количество линейно независимых маршрутов через программный код. Например, если исходный код не содержит ветвлений или циклов, его сложность равна единице, так как существует только один маршрут. При наличии оператора «if» с простым условием существуют два маршрута: один, когда условие истинно, и другой, когда ложно. Значение цикломатической сложности можно определить с помощью блок-схемы или ориентированного графа алгоритма.

Тестирование проводится для модели бизнес-логики с использованием фреймворка «NUnit» [26], которые находятся в проекте …

**Заключение**

**Список использованных источников**