МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«Новосибирский национальный исследовательский государственный университет»

(Новосибирский государственный университет, НГУ)

Структурное подразделение Новосибирского государственного университета – Высший колледж информатики Университета (ВКИ НГУ)

КАФЕДРА ИНФОРМАТИКИ

ОТЧЕТ КУРСОВОГО ПРОЕКТА

Разработка программных модулей

EGO Library

|  |  |
| --- | --- |
| Руководитель  Ассистент ВКИ НГУ | Пауль С.А.  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2025 г. |
| Студент 3 курса  гр. 2307б1 | Семьянова М.Е.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2025 г. |

Новосибирск

2025

**СОДЕРЖАНИЕ**

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ 4](#_Toc216686467)

[ВВЕДЕНИЕ 5](#_Toc216686468)

[1 ПОСТАНОВКА ЗАДАЧИ КП 7](#_Toc216686469)

[1.1 Пользовательские требования 8](#_Toc216686470)

[1.2 Системные требования 9](#_Toc216686471)

[1.3 Требования к графическому пользовательскому интерфейсу 10](#_Toc216686472)

[1.4 План-график выполнения КП 11](#_Toc216686473)

[2 АНАЛИЗ ТРЕБОВАНИЙ И ОПРЕДЕЛЕНИЕ СПЕЦИФИКАЦИЙ 12](#_Toc216686474)

[2.1 Описание предметной области задачи КП 12](#_Toc216686475)

[2.1.1 Информационные объекты предметной области и взаимосвязи между ними 13](#_Toc216686476)

[2.1.2 Информационные и функциональные потребности пользователей разрабатываемой ПС 14](#_Toc216686477)

[2.1.3 Обзор существующих программных реализаций решения задачи 15](#_Toc216686478)

[2.1.4 Концептуальное обоснование разработки 15](#_Toc216686479)

[2.2 Классы и характеристики пользователей 16](#_Toc216686480)

[2.3 Функциональные требования 16](#_Toc216686481)

[2.4 Нефункциональные требования 17](#_Toc216686482)

[3 ВЫБОР ПРОГРАММНЫХ СРЕД И СРЕДСТВ РАЗРАБОТКИ 19](#_Toc216686483)

[3.2 Стек технологий разработки 20](#_Toc216686484)

[4 АЛГОРИТМ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ 22](#_Toc216686485)

[4.1 Этапы реализации ПС 22](#_Toc216686486)

[4.2 Пользовательский интерфейс ПС 23](#_Toc216686487)

[4.2.1 Взаимодействие пользователя с ПС 23](#_Toc216686488)

[4.2.1.1 Диаграмма последовательности для прецедента «Поиск и фильтрация каталога EGO Gifts» 23](#_Toc216686489)

[4.2.1.2 Диаграмма активности для прецедента «Просмотр детальной информации о даре» 24](#_Toc216686490)

[4.2.1.3 Диаграмма состояний для объекта «Сессия работы с каталогом» 26](#_Toc216686491)

[4.2.2 Проектирование структуры экранов ПС и схемы навигации 27](#_Toc216686492)

[4.2.3 Определение операций пользователей 29](#_Toc216686493)

[4.3 Входные, выходные и промежуточные данные 31](#_Toc216686494)

[4.3.2 Входные данные 33](#_Toc216686495)

[4.4 Разработка базы данных, реализуемой в рамках ПС 33](#_Toc216686496)

[4.7 Архитектура и схема функционирования ПС (ПМ) 37](#_Toc216686497)

[5 ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ 39](#_Toc216686498)

[5.1 План тестирования 39](#_Toc216686499)

[5.2 Результаты тестирования 40](#_Toc216686500)

[6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 41](#_Toc216686501)

[6.1 Запуск и начальная настройка 41](#_Toc216686502)

[6.2 Обзор главного окна 43](#_Toc216686503)

[6.3 Последовательность действий для решения основных задач 44](#_Toc216686504)

[6.4 Действия в случае сбоя 45](#_Toc216686505)

[ЗАКЛЮЧЕНИЕ 46](#_Toc216686506)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 48](#_Toc216686507)

[ПРИЛОЖЕНИЯ 49](#_Toc216686508)

[Приложение А 49](#_Toc216686509)

# 

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ

Десктопное приложение – программа, которая устанавливается на компьютер пользователя и работает под управлением операционной системы.

WPF (Windows Presentation Foundation) – система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая подсистема в составе .NET Framework, использующая язык XAML.

XAML – язык разметки, используемый для инициализации объектов в технологиях на платформе .NET.

C# – объектно-ориентированный язык программирования, разработанный Microsoft для платформы .NET.

Entity Framework Core (EF Core) – объектно-реляционный преобразователь (ORM), фреймворк для работы с базами данных с помощью объектов .NET.

SQLite – встраиваемая кроссплатформенная система управления реляционными базами данных.

MVVM (Model-View-ViewModel) – архитектурный шаблон проектирования, разделяющий логику данных (Model), пользовательский интерфейс (View) и бизнес-логику (ViewModel).

База данных (БД) – организованная совокупность структурированных данных, хранящихся в электронном виде.

EGO gifts (в контексте Limbus Company) – тип специального снаряжения, предоставляющего уникальные пассивные эффекты в игре.

Mirror Dungeon – игровой режим в Limbus Company, в котором игроки собирают EGO gifts.

Пользовательский интерфейс (UI) – интерфейс, обеспечивающий передачу информации между пользователем и программным обеспечением.

# ВВЕДЕНИЕ

В современном мире, когда цифровые технологии проникают во все сферы деятельности человека, становится особенно важным создание удобных инструментов для обработки и систематизации информации. Одной из сфер, где это проявляется наиболее ярко, является игровая индустрия, где пользователю необходимо одновременно воспринимать и анализировать множество данных.

Компьютерные игры перестали быть просто развлечением — они превратились в сложные системы, требующие планирования, анализа и принятия решений. Одним из таких проектов является *Limbus Company*, в которой важную роль играют EGO gifts — особые предметы, наделяющие персонажей уникальными свойствами. Однако информация об этих предметах разбросана по различным источникам: форумам, базам данных, фанатских вики и видеообзорах. Игроку приходится тратить значительное время на поиск нужных данных, что снижает удобство и вовлеченность в процесс игры.

Традиционные способы хранения информации, такие как статичные таблицы или неофициальные сайты, часто оказываются неудобными и неполными. Отсутствие централизованного ресурса делает анализ и сравнение предметов трудоемким процессом. В связи с этим возникает необходимость в разработке современного решения, которое позволит игрокам быстро находить, сортировать и изучать сведения об артефактах, не отвлекаясь от игрового процесса.

EGO Library представляет собой настольное приложение, предназначенное для удобного взаимодействия с базой данных EGO gifts из режима *Mirror Dungeon* игры *Limbus Company*. С помощью данного приложения пользователь может просматривать предметы, фильтровать их по характеристикам, искать по названию и изучать подробные описания, включая эффекты, источники получения и возможные комбинации с другими предметами.

Такой подход позволяет не только сократить время на поиск информации, но и сделать взаимодействие с ней более интуитивным и приятным. *EGO Library* является примером применения современных технологий в сфере геймерских сервисов, демонстрируя возможности настольных приложений в повышении эффективности и удобства взаимодействия пользователей с цифровыми данными.

# 1 ПОСТАНОВКА ЗАДАЧИ КП

Цель работы – разработка desktop-приложения для операционной системы Windows под названием «EGO Library», предоставляющего игрокам Limbus Company удобный и функциональный инструмент для работы с каталогом предметов EGO gifts из режима «Mirror Dungeon».

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ предметной области игры Limbus Company и механики EGO gifts для формирования требований к приложению.
2. Выбрать и обосновать стек современных технологий для реализации desktop-приложения с графическим интерфейсом.
3. Спроектировать архитектуру приложения, включая структуру локальной базы данных, и разработать её с использованием Entity Framework Core и SQLite.
4. Спроектировать и реализовать интуитивно понятный пользовательский интерфейс с использованием WPF и XAML, применив паттерн MVVM.
5. Реализовать бизнес-логику приложения: загрузку данных, поиск, фильтрацию, навигацию и отображение детальной информации.
6. Провести тестирование работоспособности приложения и корректности выполнения основных функций.
7. Составить документацию по проекту.

Способ решения данных задач зависит от выбора рабочей среды и языка программирования. Для разработки программного средства будут использоваться следующие технологии:

* графический пользовательский интерфейс Windows Presentation Foundation (WPF);
* язык разметки XAML;
* язык программирования C#;
* архитектурный паттерн Model-View-ViewModel (MVVM);
* объектно-реляционный преобразователь Entity Framework Core;
* встраиваемая система управления базами данных SQLite.

### 1.1 Пользовательские требования

Для пользователя (игрока Limbus Company) необходимо, чтобы приложение позволяло:

1. Просматривать полный каталог EGO gifts: Видеть все доступные в игре предметы в виде сетки наглядных карточек, каждая из которых отображает ключевую информацию: иконку, название, уровень и краткое описание эффекта.
2. Осуществлять быстрый поиск: Находить конкретный предмет по его названию или части названия через текстовое поле поиска для экономии времени.
3. Фильтровать коллекцию по ключевым параметрам: Сужать список предметов с помощью фильтров по:
   * Уровню (Tier): Выбирать предметы определенного уровня мощности.
   * Типу эффекта: Выбирать предметы, накладывающие нужный статус (Burn, Bleed, Charge, Poise и другие).
4. Сбрасывать фильтры одним действием: Быстро возвращаться к просмотру полной коллекции после применения фильтров с помощью специальной кнопки.
5. Изучать детальную информацию о предмете: При выборе конкретного EGO gift открывать отдельное окно или панель с исчерпывающими данными, включающими:
   * Полное текстовое описание предмета.
   * Список всех связанных с ним статус-эффектов.
   * Информацию о всех возможных способах получения (из магазина, от боссов, через события).
6. Работать автономно: Использовать приложение без обязательного подключения к интернету после его установки и первичной загрузки данных.
7. Получать информацию в удобном и структурированном виде: Интерфейс приложения должен быть интуитивно понятен для игрока, знакомого с терминологией Limbus Company, и обеспечивать легкую навигацию между списком предметов и детальным просмотром.

### 1.2 Системные требования

Для корректной работы необходима операционная система Windows 10 или выше.

Минимальные системные требования:

* Операционная система: Windows 10
* Процессор: 1 ГГц или быстрее
* ОЗУ: 2 ГБ
* Свободное место на диске: 200 МБ
* Видеоадаптер: с поддержкой DirectX 9

### 1.3 Требования к графическому пользовательскому интерфейсу

Требования к внешнему виду пользовательского интерфейса:

* Понимание и удобство в использовании: интерфейс должен быть интуитивно понятным. Большинство пользователей должно с первого взгляда понять его функционал без необходимости обучения или чтения инструкций. Простые и понятные элементы управления (кнопки, списки, поля ввода) способствуют быстрому освоению приложения.
* Гибкость по отношению к различным разрешениям экрана: необходима поддержка разнообразных разрешений и размеров окон для обеспечения удобства использования приложения на экранах с различными параметрами. Это позволяет адаптировать интерфейс под разные условия эксплуатации, сохраняя логичную компоновку элементов.

Требования по доступу к функциональности системы:  
Графический пользовательский интерфейс должен обеспечивать полный и эффективный доступ ко всем функциям приложения. Это включает не только управление основными возможностями, но и предоставление пользователю интуитивного взаимодействия. Ключевые аспекты:

* Отзывчивость: интерфейс должен реагировать на действия пользователя (клики, ввод текста) визуальной обратной связью.
* Понятные элементы: каждый интерактивный элемент (например, поле поиска или выпадающий список) должен иметь ясное назначение и поведение, предсказуемое для пользователя.
* Эффективность: интерфейс должен помогать пользователю достигать целей (поиск, фильтрация, просмотр информации) максимально быстро и без лишних действий.

### 1.4 План-график выполнения КП

Чтобы разработать приложение был выбран следующий план-график:

Таблица 1 – План график выполнения КП

|  |  |  |  |
| --- | --- | --- | --- |
| № | Ожидаемое время | Реальное время | Задача |
| 1. | 1 неделя | 2 дня | Анализ требований к разработке и предметной области (Limbus Company) |
| 2. | 1 неделя | 3 дня | Разработка архитектуры приложения и проектирование структуры БД |
| 3. | 1.5 недели | 8 дней | Разработка БД (SQLite + EF Core) и её наполнение |
| 4. | 1 неделя | 2 дня | Проектирование пользовательского интерфейса |
| 5. | 2 недели | 6 дней | Реализация пользовательского интерфейса и внутренней логики программы |
| 6. | 1 неделя | 2 дня | Тестирование и отладка |
| 7. | 1.5 недели | 4 дня | Документирование проекта |

# 

# 2 АНАЛИЗ ТРЕБОВАНИЙ И ОПРЕДЕЛЕНИЕ СПЕЦИФИКАЦИЙ

### 2.1 Описание предметной области задачи КП

Предмет данной курсовой работы охватывает область разработки программных решений для игрового сообщества, а именно — создание справочного инструмента для игры «Limbus Company» (Project Moon).

«Limbus Company» — это игра в жанрах dungeon crawler и пошаговая RPG с элементами gacha-механик и отдельным roguelite-режимом («Mirror Dungeon»). Ключевым элементом игрового процесса в этом режиме является сбор EGO gifts — особых предметов, которые даруют персонажам уникальные пассивные способности и накладывают различные статус-эффекты (например, Burn, Bleed, Charge, Poise).

Основной проблемой, с которой сталкиваются игроки, является отсутствие в самой игре удобного встроенного инструмента для работы с обширным каталогом EGO gifts. Информация о предметах, их эффектах, источниках получения и рецептах улучшения оказывается разрозненной. Это приводит к значительным временным затратам на поиск и сравнение данных, сложностям в планировании стратегии развития отряда и общей неэффективности взаимодействия с игровой механикой.

Таким образом, разработка внешнего десктопного приложения, которое предоставляет централизованный, структурированный и функциональный доступ к полной базе знаний об EGO gifts, является актуальной практической задачей. Такое приложение позволит оптимизировать процесс взаимодействия с игровым контентом, повысив скорость принятия решений и общий комфорт игрового процесса.

## 2.1.1 Информационные объекты предметной области и взаимосвязи между ними

В приложении есть следующие информационные объекты:

* EGO Gift (Дар): Центральный объект, который представляет конкретный игровой предмет из режима Mirror Dungeon. У дара есть название, иконка, уровень (Tier), текстовое описание и список связанных с ним статус-эффектов (например, Burn, Bleed).
* Источник (Source): Объект, который описывает способ получения дара в игре (например, "победа над боссом", "покупка в магазине"). Один дар может быть получен из нескольких источников.
* Слияние (Recipe): Объект, который описывает рецепт создания одного дара из других. Определяет связи "компонент -> результат" между предметами.
* Пользователь: Объект, который представляет самого игрока, использующего справочное приложение. В контексте данной системы пользователь является потребителем информации и не имеет активных ролей. Его задача — просмотр, поиск и фильтрация информации.

#### 

## 2.1.2 Информационные и функциональные потребности пользователей разрабатываемой ПС

Приложение должно выполнять следующие задачи для удовлетворения потребностей игроков:

1. Предоставлять полный каталог: Отображать все существующие EGO gifts в удобном для просмотра и сравнения виде (карточки с ключевой информацией).
2. Обеспечивать эффективный поиск: Позволять быстро находить предметы по названию.
3. Реализовывать гибкую фильтрацию: Предоставлять инструменты для отбора предметов по уровню (Tier) и связанным с ними статус-эффектам.
4. Детализировать информацию: По запросу пользователя отображать полное описание выбранного предмета, включая все его эффекты, все возможные способы получения и схемы слияния (какие предметы требуются для его создания и что из него можно создать).

#### 

## 2.1.3 Обзор существующих программных реализаций решения задачи

Среди существующих способов получения информации об EGO gifts можно выделить:

* Официальные и неофициальные вики-сайты (например, Fandom, Wiki.gg). Они предоставляют полные, но зачастую разрозненные текстовые описания на отдельных страницах. Функциональные возможности таких ресурсов ограничены: отсутствует единое поле для сравнения нескольких предметов, фильтрация реализована слабо или отсутствует, навигация между связанными страницами (например, по рецептам слияния) неудобна. Кроме того, их использование требует постоянного подключения к интернету.
* Сообщества и фанатские таблицы в социальных сетях или на форумах. Эти источники часто неполны, не систематизированы и быстро устаревают после выхода обновлений игры.

Таким образом, несмотря на наличие информации, отсутствует автономное десктопное приложение, которое объединило бы полноту данных вики с удобством быстрого поиска, наглядной фильтрации и отображением связей между предметами в одном интерфейсе.

## 2.1.4 Концептуальное обоснование разработки

Разработка данного приложения обоснована необходимостью предоставить игровому сообществу Limbus Company специализированный, удобный и быстрый инструмент для работы с игровой механикой. Наличие такого справочника позволяет игрокам эффективнее планировать свои действия в режиме Mirror Dungeon, тратить меньше времени на рутинный поиск данных и больше — на непосредственную игру, что повышает общее качество и комфорт игрового процесса.

### 2.2 Классы и характеристики пользователей

Приложение содержит только одну роль — Пользователь (Игрок). Это конечный потребитель справочной информации, который использует приложение для получения сведений об игровых предметах.

Характеристики пользователя:

* Цель использования: Получение справочной информации об EGO gifts для оптимизации игрового процесса в Limbus Company.
* Ключевые задачи: Просмотр каталога, использование поиска и фильтров, изучение детальных описаний предметов (эффекты, источники получения, рецепты слияния).
* Уровень доступа: Полный доступ ко всем функциям просмотра и анализа данных, представленным в интерфейсе приложения.

### 2.3 Функциональные требования

Пользователь (Игрок) может запустить приложение, свернуть, развернуть или закрыть окно, а также использовать все функции для работы с каталогом EGO gifts, которые включают в себя:

1. Просмотр каталога предметов: Отображение списка всех EGO gifts в виде сетки карточек с базовой информацией (иконка, название, уровень).
2. Поиск по каталогу: Возможность найти конкретный предмет по его названию или части названия через текстовое поле поиска.
3. Фильтрация каталога: Возможность отфильтровать список предметов по уровню (Tier) и типу статус-эффекта (Burn, Bleed и др.) с помощью соответствующих элементов управления.
4. Сброс фильтров: Возможность одним действием очистить все примененные фильтры и строку поиска, чтобы вернуться к полному каталогу.
5. Просмотр детальной информации: При выборе конкретного предмета из списка система отображает отдельную панель или окно с полной информацией о нём: описание, список эффектов, источники получения и схемы слияния.

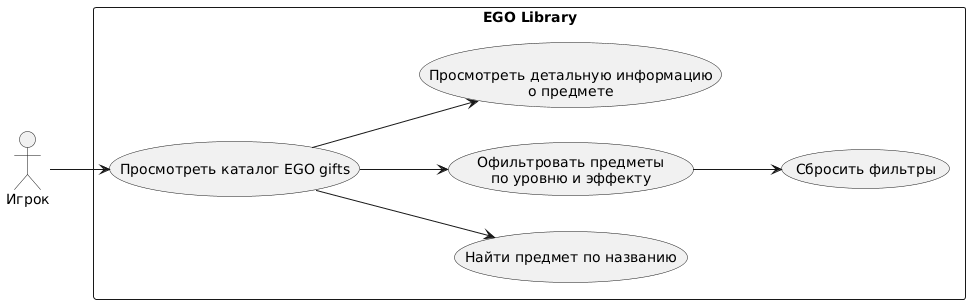


Рисунок 1 – UML-диаграмма для пользователя «Игрок»

### 2.4 Нефункциональные требования

Приложение должно соответствовать следующим нефункциональным требованиям и ограничениям:

1. Проверка целостности данных: При запуске приложение должно корректно загружать данные из локальной базы данных (SQLite). В случае повреждения файла БД или отсутствия необходимых таблиц должно выводиться понятное сообщение об ошибке.
2. Производительность интерфейса: Операции поиска предмета по названию и применения фильтров должны выполняться без видимых задержек, обеспечивая плавную работу с каталогом.
3. Устойчивость к вводу: Поле поиска и элементы интерфейса должны корректно обрабатывать различные пользовательские данные, не приводя к сбоям в работе приложения.
4. Адаптивность интерфейса: Компоновка главного окна должна сохранять читаемость и функциональность при изменении его размеров.

# 3 ВЫБОР ПРОГРАММНЫХ СРЕД И СРЕДСТВ РАЗРАБОТКИ

На основе требований к созданию современного десктопного приложения с графическим интерфейсом для ОС Windows, в качестве основной технологии был выбран фреймворк Windows Presentation Foundation (WPF), а следовательно, язык программирования C# и язык разметки XAML.

В качестве целевой платформы для приложения был выбран .NET 8.0 как актуальная стабильная версия с долгосрочной поддержкой (LTS), что обеспечивает высокую производительность, доступ к современным функциям и стабильность разработки.

Для работы с данными был выбран объектно-реляционный преобразователь Entity Framework Core в сочетании с встраиваемой базой данных SQLite. Этот выбор обусловлен необходимостью создания автономного приложения, не требующего развертывания отдельного сервера СУБД. SQLite обеспечивает компактное и быстрое файловое хранение структурированных данных, а Entity Framework Core позволяет работать с ними через объектную модель C#, что значительно ускоряет разработку.

Архитектура приложения строится на паттерне Model-View-ViewModel (MVVM), который является стандартным для WPF и обеспечивает четкое разделение логики данных, бизнес-логики и пользовательского интерфейса, что повышает тестируемость и сопровождаемость кода.

Рассмотрим подробнее выбранные средства:

Выбор языка программирования C# обоснован его глубокой интеграцией с платформой .NET и мощными возможностями объектно-ориентированного программирования. C# является статически типизированным языком, что способствует высокой производительности и предотвращает множество потенциальных ошибок на этапе компиляции. Благодаря поддержке от Microsoft и активному сообществу разработчиков, C# постоянно обновляется и совершенствуется, предоставляя средства для создания эффективных и надежных приложений.

Выбор WPF обусловлен его мощными средствами для создания современных графических интерфейсов. WPF предоставляет разработчикам широкий набор инструментов для создания интерактивных и стильных пользовательских интерфейсов, что является важным фактором для обеспечения привлекательности и удобства использования приложения. Технология WPF также обладает гибкостью и возможностью масштабирования, что важно для разработки приложений с разнообразными требованиями.

### 3.2 Стек технологий разработки

Для наглядности, полный набор инструментов и технологий, использованных в проекте, представлен ниже:

* Язык программирования: C#
* Платформа: .NET 8.0 (LTS)
* Фреймворк для графического интерфейса: Windows Presentation Foundation (WPF)
* Язык разметки интерфейса: XAML
* Архитектурный шаблон: Model-View-ViewModel (MVVM)
* ORM для работы с данными: Entity Framework Core
* Система управления базами данных: SQLite
* Среда разработки: Microsoft Visual Studio 2022

Этот стек технологий обеспечивает сбалансированное сочетание производительности, скорости разработки и возможности создания удобного пользовательского интерфейса, полностью соответствующего поставленным задачам.

# 4 АЛГОРИТМ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

### 4.1 Этапы реализации ПС

Этапы реализации ПС:

1. Анализ требований к разработке. Изучение предметной области (игры Limbus Company, механики EGO gifts), формулирование пользовательских и функциональных требований.
2. Разработка архитектуры приложения. Определение компонентов системы (модуль данных, модуль бизнес-логики, модуль интерфейса), выбор паттерна проектирования (MVVM) и проектирование структуры базы данных.
3. Работа с базой данных:
   1. *Проектирование БД.* Создание ER-диаграммы, определение сущностей (EGO Gift, Sources, Recipes).
   2. *Реализация БД.* Создание классов-моделей Entity Framework Core, настройка контекста данных (AppDbContext) для SQLite и наполнение базы первоначальными данными.
4. Разработка приложения:
   1. *Проектирование пользовательского интерфейса.* Создание макетов и прототипов основных окон (главное окно, окно деталей) на языке XAML.
   2. *Реализация пользовательского интерфейса.* Вёрстка финальных окон и пользовательских элементов управления (карточек предметов, панелей фильтров).
   3. *Программирование логики программы.* Реализация ViewModel по паттерну MVVM, написание кода для загрузки данных из БД, реализации поиска, фильтрации и навигации.
5. Тестирование. Проверка работоспособности всех функций приложения, исправление ошибок, оптимизация производительности.

### 4.2 Пользовательский интерфейс ПС

## 4.2.1 Взаимодействие пользователя с ПС

Для описания ключевых сценариев взаимодействия пользователя с системой «EGO Library» используются UML-диаграммы: диаграммы последовательности, активности и состояний.

## 4.2.1.1 Диаграмма последовательности для прецедента «Поиск и фильтрация каталога EGO Gifts»

Данная диаграмма (Рисунок 2.1) иллюстрирует последовательность сообщений между объектами системы при выполнении пользователем наиболее частой операции — фильтрации списка даров.

* Основные объекты (акторы):
  + Пользователь: Инициирует действие.
  + View (Окно поиска/фильтров): Элементы UI (текстовое поле, выпадающие списки).
  + ViewModel (Модель представления каталога): Содержит логику фильтрации и коллекцию отображаемых даров.
  + Model (Сервис данных / БД): Обеспечивает доступ к хранилищу.
* Описание последовательности:
  + Пользователь вводит текст в поле поиска или выбирает параметры в выпадающих фильтрах (Tier, Эффект).
  + View уведомляет ViewModel об изменении параметров.
  + ViewModel отправляет асинхронный запрос к Model с новыми критериями фильтрации.
  + Model выполняет запрос к базе данных SQLite через EF Core и возвращает ViewModel отфильтрованную коллекцию объектов Ego Gift.
  + ViewModel обновляет свою публичную коллекцию Gifts.
  + Благодаря механизму привязки данных (Data Binding), View автоматически получает уведомление об изменении Gifts и перерисовывает сетку карточек, отображая новый результат.

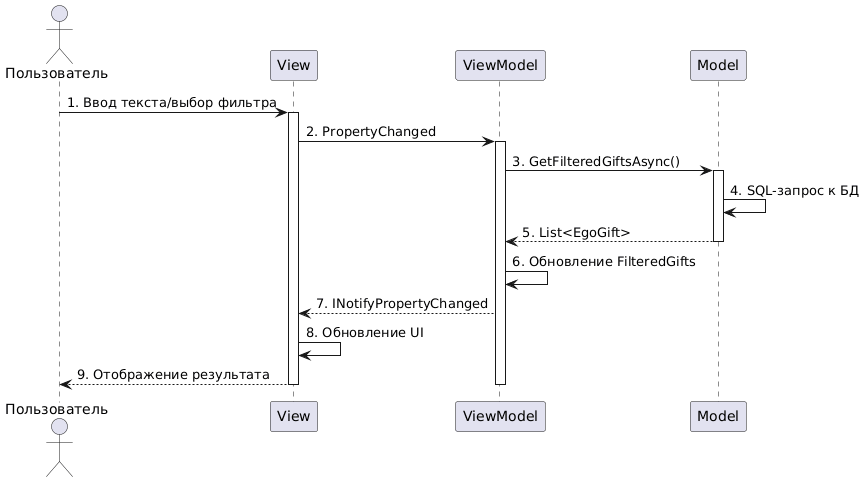


Рисунок 2.1 – Диаграмма последовательности для прецедента «Поиск и фильтрация каталога»

## 4.2.1.2 Диаграмма активности для прецедента «Просмотр детальной информации о даре»

Диаграмма активности (Рисунок 2.2) отображает поток операций, выполняемых системой при выборе пользователем конкретного EGO Gift для детального изучения.

* Начало: Пользователь кликает на карточку дара в основной сетке.
* Основной поток:
  1. Система фиксирует идентификатор (ID) выбранного дара.
  2. ViewModel запрашивает у Model не краткие, а полные данные для этого ID, включая связанные коллекции.
  3. Model загружает полный объект EgoGift с включенными данными (.Include()).
  4. Полный объект возвращается в ViewModel.
  5. ViewModel присваивает этот объект свойству SelectedGift и вычисляет производные данные.
  6. View панели деталей, привязанной к SelectedGift, автоматически обновляется, отображая всю полученную информацию: описание, иконку, таблицы источников и рецептов.
* Конец: Панель деталей полностью сформирована и отображена для пользователя.

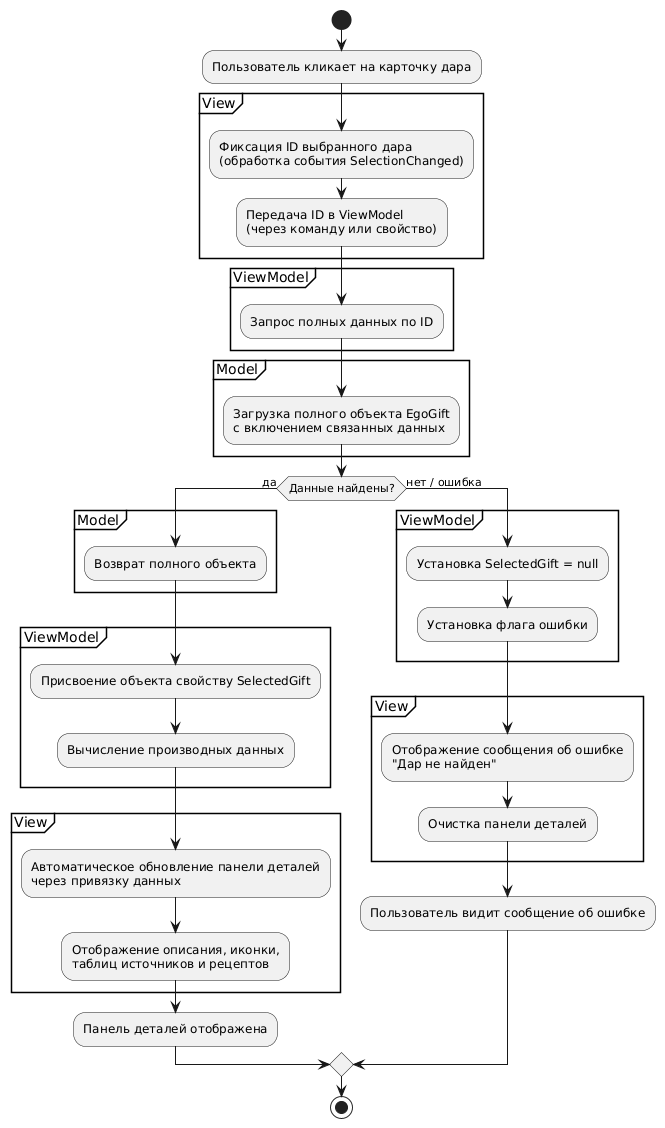


Рисунок 2.2 – Диаграмма активности для прецедента «Просмотр детальной информации»

## 4.2.1.3 Диаграмма состояний для объекта «Сессия работы с каталогом»

Диаграмма состояний (Рисунок 2.3) описывает жизненный цикл и возможные состояния основного модуля приложения — каталога EGO Gifts — и события, вызывающие переходы между ними.

* Состояния:
  + Загрузка: Начальное состояние после запуска. Система инициализирует контекст БД и загружает данные.
  + КаталогОтображен: Данные успешно загружены, отображается полный список. Это состояние по умолчанию.
  + Фильтрация: Пользователь применил один или несколько критериев (поиск, Tier, эффект). Список динамически обновляется.
  + ДетальныйПросмотр: Пользователь выбрал конкретный дар. Основной фокус интерфейса смещен на панель с полной информацией.
  + ОшибкаЗагрузки: Состояние сбоя.
* Ключевые переходы:
  + ДанныеУспешноЗагружены → из Загрузка в КаталогОтображен.
  + КритерийИзменен (ввод в поиск, выбор фильтра) → из КаталогОтображен в Фильтрация.
  + СбросФильтров → из Фильтрация в КаталогОтображен.
  + ДарВыбран → из КаталогОтображен или Фильтрация в ДетальныйПросмотр.
  + ЗакрытьДетали / ВыбранНовыйДар (внутри ДетальныйПросмотр) → переход в предыдущее состояние (КаталогОтображен или Фильтрация).
  + ОшибкаБД → из Загрузка в ОшибкаЗагрузки.

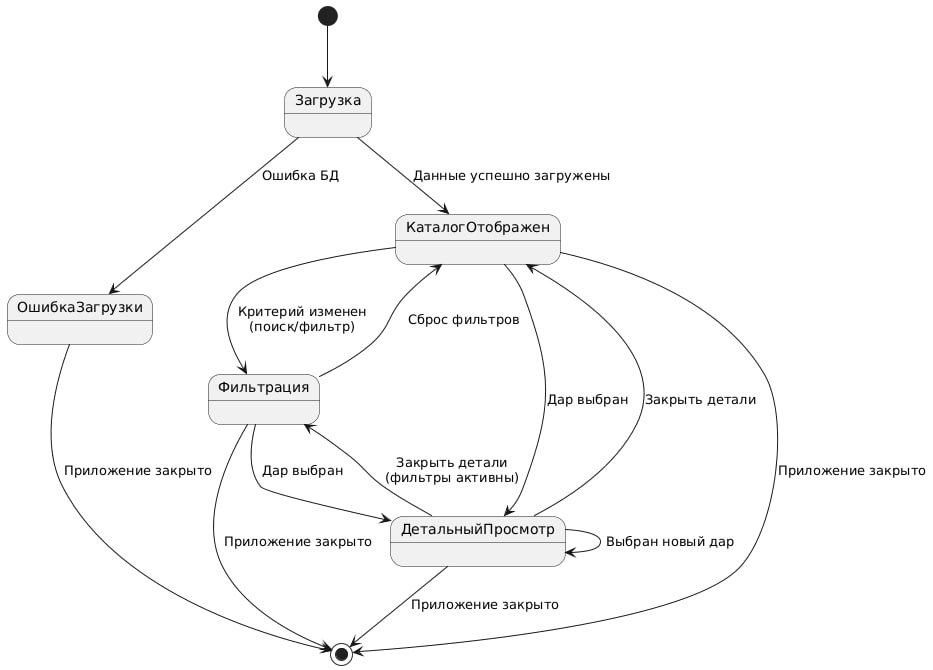


Рисунок 2.3 – Диаграмма состояний объекта «Сессия работы с каталогом»

## 4.2.2 Проектирование структуры экранов ПС и схемы навигации

В ПС представлено шесть основных экранов (окон), каждый из которых отвечает за определенный этап взаимодействия пользователя с приложением:

1. Окно авторизации (LoginView): Стартовый экран. Запрашивает имя пользователя и пароль для входа в систему. Содержит кнопку перехода к регистрации.
2. Окно регистрации (RegisterView): Позволяет новому пользователю создать учетную запись, введя необходимые данные (логин, пароль, email).
3. Главное окно (GifListView): Центральный экран приложения. Содержит полный каталог EGO gifts в виде сетки карточек, панель поиска и фильтров, а также кнопки для доступа к дополнительным разделам (Recipes, About, Help).
4. Окно детальной информации (GiftDetailView): Отображает исчерпывающие сведения о выбранном в каталоге EGO gift: описание, эффекты, стоимость, источники получения.
5. Окно рецептов (RecipeView): Предоставляет информацию о схемах слияния и создании даров.
6. Окно «О программе» (AboutView): Содержит сведения о приложении, авторе, версии и использованных технологиях.

Схема навигации между экранами является иерархической и представлена на Рисунке 3. Поток взаимодействия пользователя с системой выглядит следующим образом:

1. При запуске приложение открывает окно авторизации (LoginView).
2. Из окна авторизации пользователь может:
   * Успешно войти в систему, после чего произойдет переход к главному окну (GifListView).
   * Нажать кнопку «Зарегистрироваться», чтобы перейти к окну регистрации (RegisterView).
3. Из окна регистрации (RegisterView) пользователь может:
   * Завершить регистрацию, после чего автоматически произойдет переход к главному окну (GifListView).
4. Из главного окна (GifListView) пользователь может:
   * Нажать на любую карточку дара, чтобы открыть окно детальной информации (GiftDetailView).
   * Нажать кнопку «Recipes», чтобы открыть окно рецептов (RecipeView).
   * Нажать кнопку «About», чтобы открыть окно «О программе» (AboutView).
5. Все дочерние окна (GiftDetailView, RecipeView, AboutView) имеют возможность вернуться на главное окно (GifListView).

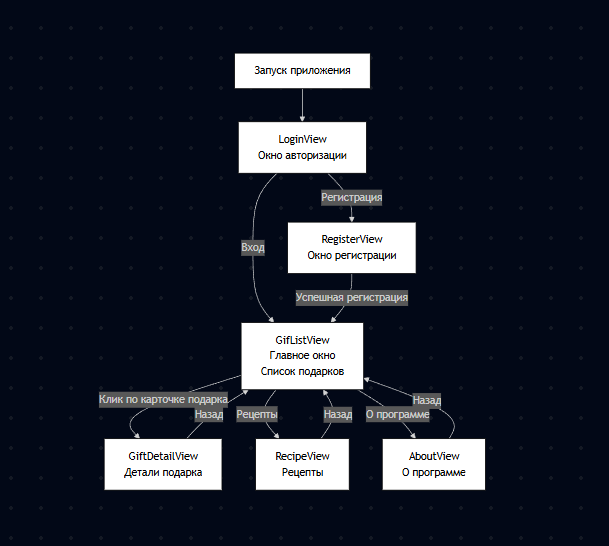


Рисунок 3 – Схема навигации в ПС EGO Library

## 4.2.3 Определение операций пользователей

В рамках определенных прецедентов пользователь (Игрок) может выполнять следующие операции с элементами графического интерфейса ПС «EGO Library».

Таблица 2 – Операции пользователя в рамках прецедентов

|  |  |  |  |
| --- | --- | --- | --- |
| № | Категория операции | Конкретные действия и элементы интерфейса | Цель операции |
| 1 | Общее управление | Запуск приложения; закрытие окна; свёртывание/развёртывание окна | Инициировать и завершить работу |
| 2 | Навигация по каталогу | Просмотр сетки карточек; выбор карточки (одинарный клик) | Ознакомиться и выбрать объект |
| 3 | Фильтрация и поиск | Ввод в «Search»; выбор в «Tier»; выбор в «Status»; «Clear Filters» | Сузить список или сбросить фильтры |
| 4 | Работа с детальной информацией | Открытие «Gift Detail View»; просмотр данных; кнопка «Back» | Получить сведения и вернуться в каталог |
| 5 | Дополнительная навигация | Кнопки: «Recipes», «About», «Help» | Перейти к рецептам, информации о программе или справке |

### 

### 4.3 Входные, выходные и промежуточные данные

Промежуточные данные хранятся в объектах следующих моделей, которые соответствуют сущностям в локальной базе данных SQLite:

* EGO Gift (Дар):
  + Id — уникальный идентификатор.
  + Name — название дара.
  + Tier — уровень.
  + Effect — описание основного эффекта.
  + ImagePath — путь к иконке.
  + Cost, SellPrice — игровая стоимость.
  + Sources — коллекция связанных источников получения.
  + ResultRecipes, RequiredInRecipes — коллекции связанных рецептов.
* Источник (Source):
  + Id — уникальный идентификатор.
  + Location, Type — место и тип источника.
  + DropRate — шанс выпадения.
  + EgoGiftId — ссылка на связанный дар.
* Рецепт (Recipe):
  + Id — уникальный идентификатор.
  + Name, Description — название и описание.
  + ResultGiftId — ссылка на дар, являющийся результатом.
  + RequiredGifts — коллекция даров, необходимых для создания.
* Пользователь (User):
  + Id — уникальный идентификатор.
  + Username, Password — данные для аутентификации.

Входные данные — это действия пользователя, которые инициируют обработку в системе:

* Поисковый запрос: Текст, введенный пользователем в поле поиска.
* Параметры фильтра: Выбранные пользователем значения для фильтров по уровню и эффекту.
* Команда навигации: Событие выбора пользователем конкретного дара из списка для просмотра деталей.
* Учетные данные: Логин и пароль, введенные в окнах авторизации и регистрации.

Выходные данные — это результат обработки, отображаемый пользователю:

* Отфильтрованный список даров: Коллекция объектов EgoGift, соответствующая критериям поиска и фильтров, отображаемая в виде сетки карточек.
* Детализированная информация: Полный набор данных о выбранном даре, отображаемый в отдельной панели.
* Результат аутентификации: Успешный переход в главное окно приложения или сообщение об ошибке.

### 4.3.2 Входные данные

Входные данные приложения можно разделить на два ключевых потока:

1. Исходные справочные данные (инициализация): При первом запуске приложения автоматически выполняется код инициализации. Этот код создает структуру локальной базы данных SQLite и заполняет таблицы (EgoGifts, Sources, Recipes) полным актуальным набором данных об EGO gifts из игры Limbus Company. Эти данные являются основой для работы приложения и хранятся в файле ego\_library.db в рабочей директории программы.

### 4.4 Разработка базы данных, реализуемой в рамках ПС

В качестве основного хранилища данных в ПС «EGO Library» используется встраиваемая реляционная база данных SQLite.

*Таблица 3 – Поля таблицы* Ego Gifts

|  |  |  |
| --- | --- | --- |
| Название поля | Тип | Описание |
| Id | int / INTEGER | Уникальный идентификатор записи (первичный ключ). |
| Name | string / TEXT | Название EGO gift. |
| Tier | int / INTEGER | Уровень дара. |
| Status | string / TEXT | Статус/категория. |
| ImagePath | string / TEXT | Путь к иконке. |
| Effect | string / TEXT | Описание эффекта. |
| Description | string / TEXT | Развернутое описание. |
| Keywords | string / TEXT | Ключевые слова. |
| Cost | int / INTEGER | Стоимость покупки. |
| SellPrice | int / INTEGER | Цена продажи. |
| Acquisition | string / TEXT | Способ получения. |

*Таблица 4 – Поля таблицы* Recipes

|  |  |  |
| --- | --- | --- |
| Название поля | Тип | Описание |
| Id | int / INTEGER | Уникальный идентификатор рецепта. |
| Name | string / TEXT | Название рецепта. |
| Description | string / TEXT | Описание рецепта. |
| ResultGiftId | int / INTEGER | FK на EgoGift. |
| Location | string / TEXT | Локация. |
| Difficulty | string / TEXT | Сложность. |

## 

*Таблица 5 – Поля таблицы* Sources

|  |  |  |
| --- | --- | --- |
| Название поля | Тип | Описание |
| Id | int / INTEGER | Уникальный идентификатор источника. |
| Location | string / TEXT | Локация, где находится источник. |
| Type | string / TEXT | Тип источника. |
| Floor | int / INTEGER | Этаж, на котором расположен источник. |
| DropRate | number / REAL | Шанс выпадения предмета. |
| EgoGiftId | int / INTEGER | Внешний ключ на EgoGifts. |

*Таблица 6 – Поля таблицы* User

|  |  |  |
| --- | --- | --- |
| Название поля | Тип | Описание |
| Id | int / INTEGER | Уникальный идентификатор пользователя. |
| Username | string / TEXT | Имя пользователя. |
| Password | string / TEXT | Пароль |
| Email | string / TEXT | Email. |
| CreatedDate | DateTime / TEXT | Дата регистрации. |
| LastLogin | DateTime / TEXT | Дата последнего входа. |

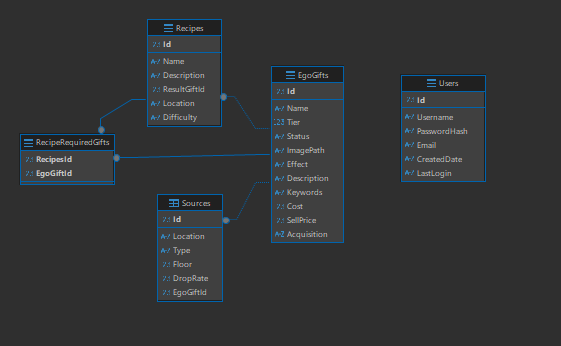


Рисунок 4 – ERD схема базы данных

### 4.7 Архитектура и схема функционирования ПС (ПМ)

Паттерн MVVM обеспечивает четкое разделение ответственностей между компонентами приложения (Model, View, ViewModel), облегчает тестирование и повторное использование кода благодаря независимости ViewModel от конкретной реализации View, обеспечивает гибкость при масштабировании и изменении интерфейса, автоматически обновляет View при изменении данных в Model, а также позволяет разработчикам работать параллельно, увеличивая производительность и уменьшая время разработки. Этот подход способствует поддерживаемости приложений и улучшает их структуру.

* Model (Модель):
  + Отвечает за представление данных и бизнес-логики приложения.
  + Не содержит информации о том, как эти данные будут отображены или как пользователь будет взаимодействовать с ними.
* View (Представление):
  + Отвечает за отображение данных пользователю и за взаимодействие с пользователем.
  + Не содержит логики приложения и не знает о существовании Model.
* ViewModel (Модель представления):
  + Служит связующим звеном между Model и View.
  + Отвечает за обработку пользовательских действий и передачу соответствующих команд Model.
  + Не зависит напрямую от конкретной реализации View, что облегчает тестирование и повторное использование кода.

Для реализации MVVM использовалась следующая файловая структура:



Рисунок 5 – Файловая структура проекта

# 5 ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ

### 5.1 План тестирования

Основываясь на ключевых функциях приложения, для проверки работоспособности был составлен следующий план тестирования:

1. Корректность загрузки данных: Проверка, что при запуске приложения из локальной базы данных корректно загружается полный каталог EGO gifts, включая связанную информацию (источники получения, рецепты).
2. Корректность работы поиска: Проверка функции поиска даров по названию. Должны учитываться частичные совпадения, поиск должен быть регистронезависимым и выполняться без задержек при вводе.
3. Корректность работы фильтрации: Проверка фильтров по уровню (Tier) и статус-эффекту (Effect). При выборе одного или нескольких критериев список должен соответствующим образом обновляться.
4. Корректность отображения детальной информации: Проверка, что при выборе дара из списка в отдельной панели или окне отображаются все его атрибуты: описание, эффект, стоимость, а также списки связанных источников и рецептов.
5. Устойчивость к некорректным данным: Проверка реакции приложения на потенциально проблемные ситуации, например, на отсутствие файла базы данных или на некорректные значения в полях, которые могут быть не заполнены.

## 

### 5.2 Результаты тестирования

В результате функционального тестирования по плану были сделаны следующие выводы:

1. Основные функции приложения работают корректно. Каталог даров загружается, поиск и фильтрация выполняются, детальная информация отображается в полном объеме.
2. В процессе тестирования была выявлена и устранена типичная ошибка, связанная с целостностью данных. В ранней версии приложения возникало исключение при попытке отфильтровать дары, если у некоторых записей в базе не были заполнены обязательные для отображения поля (например, не указан Tier или Effect).
3. Для устранения данной проблемы была проведена оптимизация кода:
   * В классах ViewModel были добавлены проверки на null и значения по умолчанию для свойств, участвующих в фильтрации.
   * Логика фильтрации была доработана таким образом, чтобы записи с отсутствующими (null) данными корректно обрабатывались и не вызывали сбоев, а просто не отображались при применении соответствующего фильтра.

В результате всех исправлений приложение демонстрирует стабильную работу, а его интерфейс корректно реагирует на все действия пользователя, предусмотренные функциональными требованиями.

# 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

### 6.1 Запуск и начальная настройка

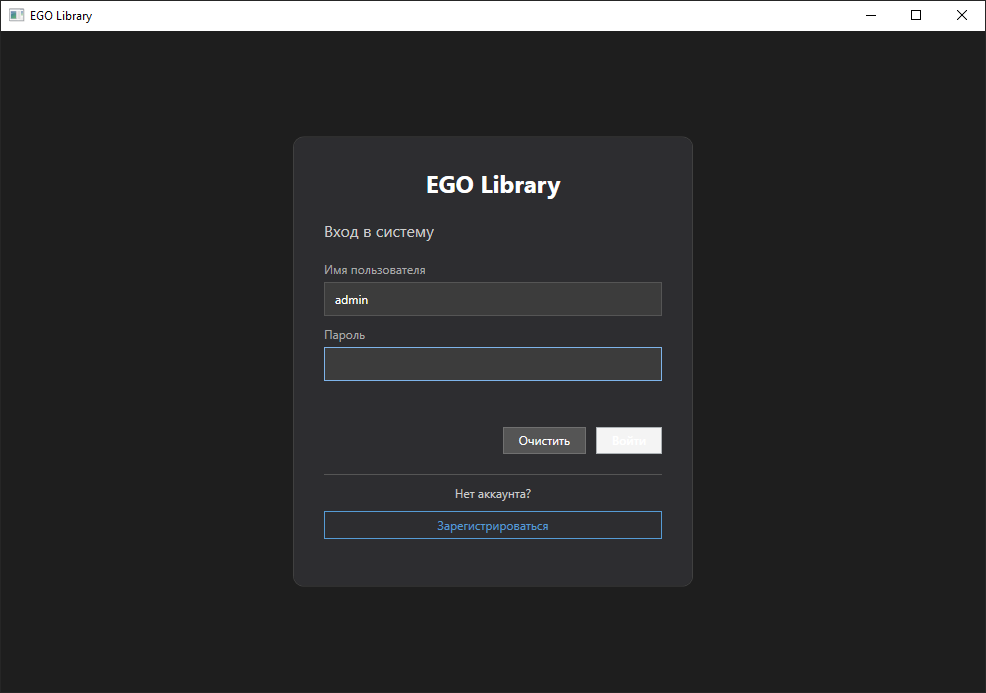
Приложение «EGO Library» является автономным. После установки или распаковки архива для его запуска необходимо запустить исполняемый файл EGO\_Library.exe. Программа автоматически создаст и подключится к локальному файлу базы данных (egolibrary.db), содержащему актуальный каталог EGO gifts. Никакой дополнительной настройки не требуется.  


Рисунок 6 – Окно входа в систему

Для начала работы необходимо:

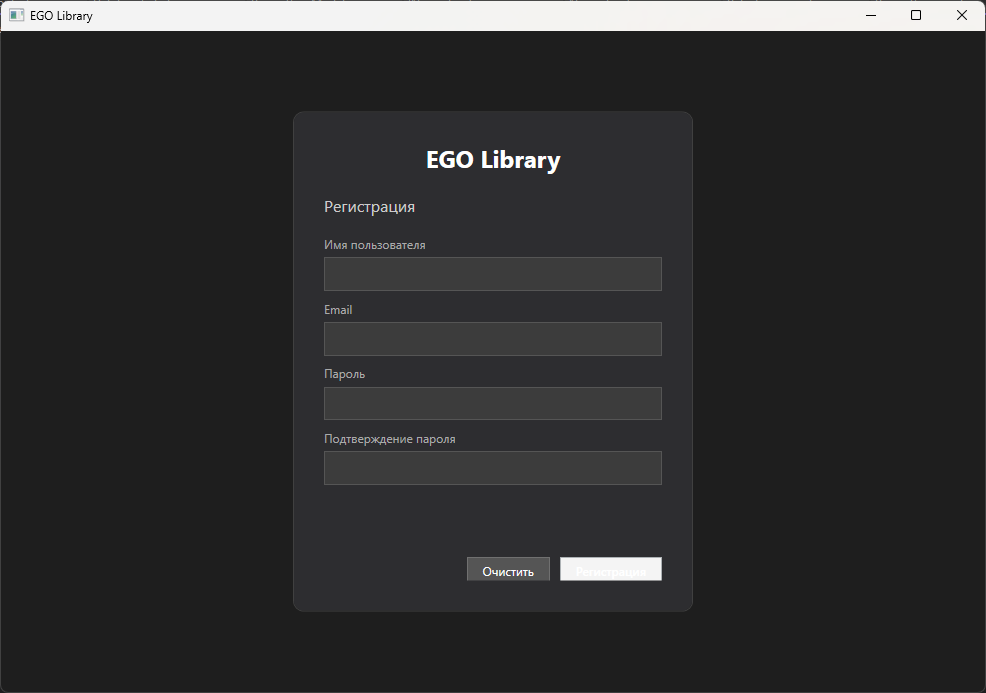
1. Ввести зарегистрированное Имя пользователя (Username) и Пароль (Password) в соответствующие поля.
2. Нажать кнопку «Войти» (Login).
3. Если учетная запись отсутствует, нажать кнопку «Зарегистрироваться», чтобы перейти к созданию нового профиля (Рисунок 7).

Рисунок 7 – Окно регистрации нового пользователя

После успешной аутентификации откроется главное окно программы.

#### 

### 6.2 Обзор главного окна

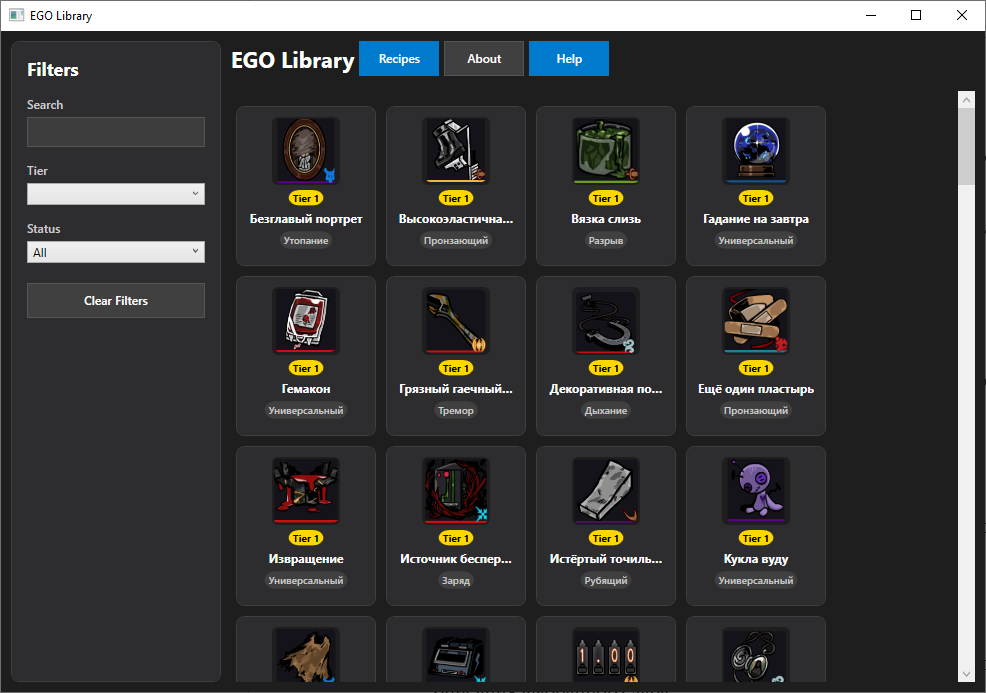
Главное окно программы представлено на Рисунке 8.

Рисунок 8 – Главный экран приложения "EGO Library" (каталог EGO gifts)

Основные элементы интерфейса:

1. Панель поиска и фильтров: Содержит поле поиска по названию (Search), фильтры по уровню (Tier) и эффекту (Effect), а также кнопку сброса (Reset).
2. Область каталога: Сетка карточек, отображающая EGO gifts (иконка, название, уровень).
3. Область детальной информации: Панель для просмотра полного описания выбранного дара.

### 

### 6.3 Последовательность действий для решения основных задач

1. Поиск конкретного дара:

* 1. В поле «Search» введите название.
  2. Список карточек обновится в реальном времени.

2. Фильтрация каталога:

* 1. В списках «Tier» и «Effect» выберите нужные критерии.
  2. Для возврата к полному каталогу нажмите кнопку «Сбросить».

3. Просмотр детальной информации:

* 1. В каталоге нажмите на карточку дара.
  2. В панели детальной информации (Рисунок 9) загрузится полное описание, эффекты, стоимость, источники.

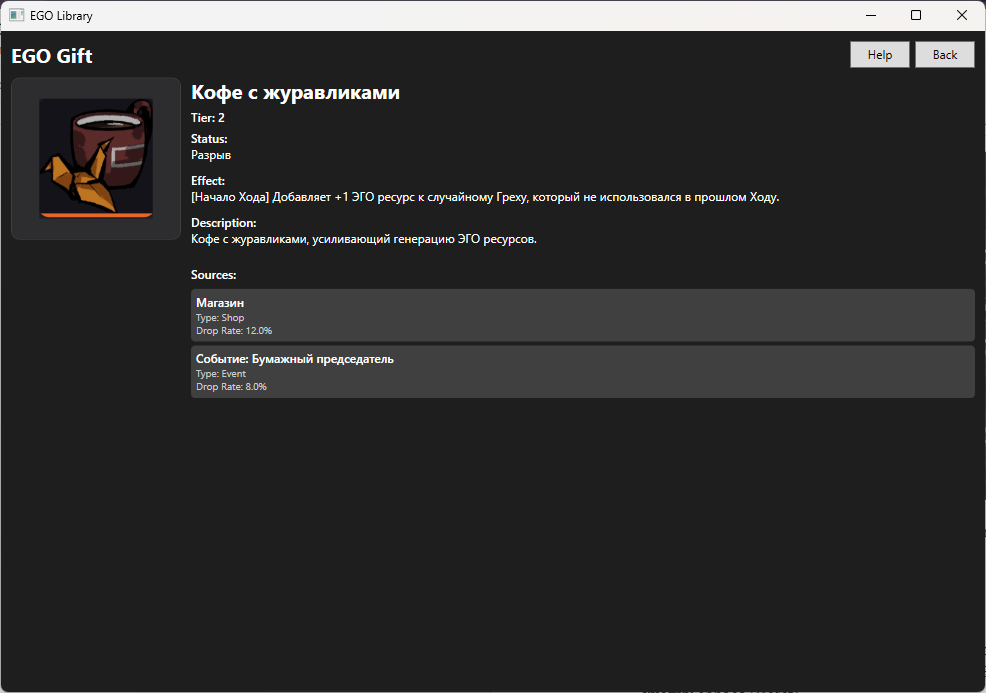


Рисунок 9 – Панель детальной информации о даре

### 6.4 Действия в случае сбоя

* Ошибка входа: Убедитесь в правильности логина и пароля. При необходимости используйте функцию восстановления или регистрации новой учетной записи.
* Ошибка загрузки данных: Проверьте наличие файла базы данных egolibrary.db в папке с программой и права на его чтение.
* Приложение не отвечает: Закройте программу через Диспетчер задач и перезапустите.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано десктопное приложение «EGO Library» — специализированный справочник по предметам EGO gifts из игры Limbus Company. Приложение реализовано с использованием платформы WPF, языка разметки XAML и языка программирования C# на платформе .NET 8.0 с применением архитектурного паттерна MVVM, объектно-реляционного преобразователя Entity Framework Core и встраиваемой базы данных SQLite.

В результате выполнения проекта были успешно решены задачи, поставленные в начале работы:

1. Проведен анализ предметной области игры Limbus Company и механики EGO gifts, на основе которого были сформулированы функциональные и нефункциональные требования.
2. Выбран и обоснован современный стек технологий, оптимальный для разработки отзывчивого desktop-приложения с графическим интерфейсом и локальным хранением данных.
3. Спроектирована и реализована структура реляционной базы данных, состоящей из четырех основных таблиц (EgoGifts, Sources, Recipes, Users), и произведено её первоначальное наполнение.
4. Спроектирован и реализован интуитивно понятный пользовательский интерфейс, состоящий из шести экранов (авторизация, регистрация, главный каталог, детальный просмотр, рецепты, информация о программе), с четкой схемой навигации между ними.
5. Реализована бизнес-логика приложения, включающая механизмы поиска по названию, фильтрации по уровню и эффекту, авторизации пользователей и навигации.
6. Проведено функциональное тестирование ключевых сценариев использования, в ходе которого была выявлена и устранена проблема целостности данных, что повысило общую надежность приложения.

Практическая значимость работы заключается в создании удобного, автономного инструмента, который решает конкретную проблему игрового сообщества — неэффективный доступ к разрозненной справочной информации. Приложение позволяет игрокам быстро находить, сравнивать и изучать игровые предметы, что способствует оптимизации стратегического планирования и улучшает игровой опыт.

Возможные направления развития (точки роста) системы:

1. Расширение функционала: Добавление возможности пользователям создавать персональные коллекции или списки избранных даров, добавлять заметки.
2. Обновление данных: Реализация механизма безопасного централизованного обновления файла базы данных через сеть для синхронизации с новыми игровыми патчами.
3. Углубление анализа: Внедрение инструментов для сравнения нескольких даров, построения более сложных графических схем зависимостей между ними.

Таким образом, в результате выполнения курсовой работы было полностью разработано работоспособное настольное программное средство, которое соответствует всем поставленным функциональным требованиям, обладает четкой архитектурой и имеет практическую ценность для целевой аудитории.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. WPF [Электронный ресурс] / Microsoft Learn – URL: https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/overview/

(Дата обращ. 12.12.25).

1. XAML [Электронный ресурс] / Microsoft Learn – URL: https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/xaml/

(Дата обращ. 12.12.25).

1. C# [Электронный ресурс] / Microsoft Learn – URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/tutorials/oop (Дата обращ. 12.12.25).
2. Entity Framework Core [Электронный ресурс] / Microsoft Learn – URL: https://learn.microsoft.com/ru-ru/ef/core/ (Дата обращ. 12.12.25).
3. SQLite [Электронный ресурс] / SQLite — URL: https://www.sqlite.org/index.html (Дата обращ. 12.12.25).
4. Обучение по C# [Электронный ресурс] / Metanit – URL: https://metanit.com/sharp/tutorial/ (Дата обращ. 12.12.25).
5. Обучение по WPF [Электронный ресурс] / Metanit – URL: https://metanit.com/sharp/wpf/ (Дата обращ. 12.12.25).
6. Limbus Company [Электронный ресурс] / Project Moon — URL: https://limbuscompany.com/ (Дата обращ. 12.12.25).
7. Limbus Company Wiki [Электронный ресурс] / Fandom — URL: https://limbuscompany.fandom.com/wiki/Limbus\_Company\_Wiki (Дата обращ. 12.12.25).
8. Исходный код [Электронный ресурс] / GitHub — URL: https://github.com/AntoSubo/EGO-Library (Дата обращ. 12.12.25).

# ПРИЛОЖЕНИЯ

### Приложение А

#### ****XAML-разметка главного окна приложения (GifListView.xaml)****

**Назначение:** Определяет структуру, внешний вид и основные элементы управления главного окна приложения. Реализует привязку данных к свойствам ViewModel (GifListViewModel) для отображения списка даров, работы поиска и фильтров.

<UserControl x:Class="EGO\_Library.Views.Controls.GiftListView"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:converters="clr-namespace:EGO\_Library.Converters"

mc:Ignorable="d"

d:DesignHeight="600" d:DesignWidth="1000">

<UserControl.Resources>

<converters:ImagePathConverter x:Key="ImagePathConverter"/>

<Style x:Key="GiftCardStyle" TargetType="Border">

<Setter Property="Background" Value="#2D2D30"/>

<Setter Property="BorderBrush" Value="#404040"/>

<Setter Property="BorderThickness" Value="1"/>

<Setter Property="CornerRadius" Value="8"/>

<Setter Property="Margin" Value="5"/>

<Setter Property="Padding" Value="10"/>

<Setter Property="Cursor" Value="Hand"/>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="BorderBrush" Value="#569CD6"/>

<Setter Property="Background" Value="#3E3E42"/>

</Trigger>

</Style.Triggers>

</Style>

<Style x:Key="DarkButtonStyle" TargetType="Button">

<Setter Property="Background" Value="#404040"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="BorderBrush" Value="#555555"/>

<Setter Property="BorderThickness" Value="1"/>

<Setter Property="Padding" Value="10,5"/>

<Setter Property="FontWeight" Value="SemiBold"/>

<Setter Property="Cursor" Value="Hand"/>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Background" Value="#505050"/>

<Setter Property="BorderBrush" Value="#569CD6"/>

</Trigger>

<Trigger Property="IsPressed" Value="True">

<Setter Property="Background" Value="#2D2D30"/>

</Trigger>

</Style.Triggers>

</Style>

<Style x:Key="PrimaryButtonStyle" TargetType="Button" BasedOn="{StaticResource DarkButtonStyle}">

<Setter Property="Background" Value="#007ACC"/>

<Setter Property="BorderBrush" Value="#007ACC"/>

<Setter Property="Foreground" Value="White"/>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Background" Value="#1C97EA"/>

<Setter Property="BorderBrush" Value="#1C97EA"/>

</Trigger>

</Style.Triggers>

</Style>

</UserControl.Resources>

<Grid Margin="10">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="220"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Border Grid.Column="0"

Background="#2D2D30"

BorderBrush="#404040"

BorderThickness="1"

CornerRadius="8"

Padding="15"

Margin="0,0,10,0">

<StackPanel>

<TextBlock Text="Filters"

FontSize="18"

FontWeight="Bold"

Foreground="White"

Margin="0,0,0,15"/>

<TextBlock Text="Search"

FontWeight="SemiBold"

Foreground="#CCCCCC"

Margin="0,0,0,5"/>

<TextBox Text="{Binding SearchText, UpdateSourceTrigger=PropertyChanged}"

Margin="0,0,0,15"

Height="30"

Padding="5"

Background="#3C3C3C"

Foreground="White"

BorderBrush="#555555"/>

<TextBlock Text="Tier"

FontWeight="SemiBold"

Foreground="#CCCCCC"

Margin="0,0,0,5"/>

<ComboBox ItemsSource="{Binding AvailableTiers}"

SelectedItem="{Binding SelectedTier}"

Margin="0,0,0,15"

Background="#3C3C3C"

Foreground="White"

BorderBrush="#555555">

<ComboBox.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding StringFormat='Tier {0}'}" Foreground="Black"/>

</DataTemplate>

</ComboBox.ItemTemplate>

</ComboBox>

<TextBlock Text="Status"

FontWeight="SemiBold"

Foreground="#CCCCCC"

Margin="0,0,0,5"/>

<ComboBox ItemsSource="{Binding AvailableStatuses}"

SelectedItem="{Binding SelectedStatus}"

Margin="0,0,0,20"

Background="#3C3C3C"

Foreground="Black"

BorderBrush="#555555"/>

<Button Content="Clear Filters"

Height="35"

Command="{Binding ClearFiltersCommand}"

Style="{StaticResource DarkButtonStyle}"

Margin="0,0,0,10"/>

</StackPanel>

</Border>

<Grid Grid.Column="1">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<DockPanel Grid.Row="0" Margin="0,0,0,15">

<TextBlock Text="EGO Library"

FontSize="22"

FontWeight="Bold"

Foreground="White"

VerticalAlignment="Center"/>

<StackPanel Orientation="Horizontal" DockPanel.Dock="Right">

<Button Content="Recipes"

Width="80"

Height="35"

Margin="5,0,0,0"

Style="{StaticResource PrimaryButtonStyle}"

Command="{Binding NavigateToRecipesCommand}"/>

<Button Content="About"

Width="80"

Height="35"

Margin="5,0,0,0"

Style="{StaticResource DarkButtonStyle}"

Command="{Binding NavigateToAboutCommand}"/>

<Button Content="Help"

Width="80"

Height="35"

Margin="5,0,0,0"

Style="{StaticResource PrimaryButtonStyle}"

Command="{Binding ShowHelpCommand}"/>

</StackPanel>

</DockPanel>

<ScrollViewer Grid.Row="1" VerticalScrollBarVisibility="Auto">

<ItemsControl ItemsSource="{Binding Gifts}">

<ItemsControl.ItemsPanel>

<ItemsPanelTemplate>

<WrapPanel Margin="0,10,0,0"

Orientation="Horizontal"/>

</ItemsPanelTemplate>

</ItemsControl.ItemsPanel>

<ItemsControl.ItemTemplate>

<DataTemplate>

<Border Style="{StaticResource GiftCardStyle}"

Width="140"

Height="160"

MouseLeftButtonDown="Border\_MouseLeftButtonDown">

<StackPanel>

<Border Background="#1E1E1E"

CornerRadius="6"

Padding="2"

HorizontalAlignment="Center"

Margin="0,0,0,5">

<Image Source="{Binding ImagePath, Converter={StaticResource ImagePathConverter}}"

Width="64"

Height="64"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

<Border Background="#FFD700"

CornerRadius="10"

Padding="4,1"

HorizontalAlignment="Center"

Margin="0,0,0,5">

<TextBlock Text="{Binding Tier, StringFormat='Tier {0}'}"

FontSize="10"

FontWeight="Bold"

Foreground="Black"

TextAlignment="Center"/>

</Border>

<TextBlock Text="{Binding Name}"

FontWeight="Bold"

FontSize="12"

TextAlignment="Center"

TextTrimming="CharacterEllipsis"

Foreground="White"

Margin="0,0,0,5"

MaxHeight="32"/>

<Border Background="#404040"

CornerRadius="10"

Padding="4,2"

HorizontalAlignment="Center">

<TextBlock Text="{Binding Status}"

FontSize="10"

FontWeight="SemiBold"

Foreground="#CCCCCC"

TextAlignment="Center"/>

</Border>

</StackPanel>

</Border>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</ScrollViewer>

</Grid>

</Grid>

</UserControl>

#### ****Класс модели EgoGift.cs****

**Назначение:** Основной класс-сущность, представляющий объект предметной области — EGO Gift. Содержит все свойства дара и навигационные свойства для связей с источниками и рецептами. Используется Entity Framework Core для маппинга на таблицу базы данных.

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace EGO\_Library.Models

{

public class EgoGift

{

public int Id { get; set; }

[Required]

[MaxLength(100)]

public string Name { get; set; } = string.Empty;

public int Tier { get; set; }

[MaxLength(50)]

public string Status { get; set; } = string.Empty;

[MaxLength(10)]

public string ImagePath { get; set; } = string.Empty;

public string Effect { get; set; } = string.Empty;

public string Description { get; set; } = string.Empty;

public string Keywords { get; set; } = string.Empty;

public int Cost { get; set; }

public int SellPrice { get; set; }

public string Acquisition { get; set; } = string.Empty;

public virtual ICollection<Sources> Sources { get; set; } = new List<Sources>();

public virtual ICollection<Recipes> ResultRecipes { get; set; } = new List<Recipes>();

public virtual ICollection<Recipes> RequiredInRecipes { get; set; } = new List<Recipes>();

}

}

#### ****ViewModel главного окна (GifListViewModel.cs)****

**Назначение:** Ключевой класс бизнес-логики по паттерну MVVM для главного окна. Отвечает за загрузку данных из базы, реализацию поиска и фильтрации, предоставляет свойства и команды для привязки данных в XAML-разметке (GifListView.xaml).

using EGO\_Library.Models;

using EGO\_Library.Services;

using System.Collections.ObjectModel;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Input;

namespace EGO\_Library.ViewModels

{

public class GiftListViewModel : BaseViewModel

{

private readonly DataService \_dataService;

private readonly INavigationService \_navigationService;

private ObservableCollection<EgoGift> \_gifts;

private string \_searchText;

private int? \_selectedTier;

private string \_selectedStatus = "All";

public ObservableCollection<EgoGift> Gifts

{

get => \_gifts;

set { \_gifts = value; OnPropertyChanged(); }

}

public string SearchText

{

get => \_searchText;

set { \_searchText = value; OnPropertyChanged(); \_ = LoadGiftsAsync(); }

}

public int? SelectedTier

{

get => \_selectedTier;

set { \_selectedTier = value; OnPropertyChanged(); \_ = LoadGiftsAsync(); }

}

public string SelectedStatus

{

get => \_selectedStatus;

set { \_selectedStatus = value; OnPropertyChanged(); \_ = LoadGiftsAsync(); }

}

public List<int> AvailableTiers { get; private set; } = new List<int>();

public List<string> AvailableStatuses { get; private set; } = new List<string> { "All" };

// Команды

public ICommand ClearFiltersCommand { get; }

public ICommand NavigateToAboutCommand { get; }

public ICommand ShowHelpCommand { get; }

public ICommand SelectGiftCommand { get; }

public ICommand NavigateToRecipesCommand { get; }

public GiftListViewModel(DataService dataService, INavigationService navigationService)

{

\_dataService = dataService;

\_navigationService = navigationService;

ClearFiltersCommand = new RelayCommand(\_ => ClearFilters());

NavigateToAboutCommand = new RelayCommand(\_ => \_navigationService.NavigateToAbout());

ShowHelpCommand = new RelayCommand(\_ => ShowHelp());

SelectGiftCommand = new RelayCommand(SelectGift);

NavigateToRecipesCommand = new RelayCommand(\_ => \_navigationService.NavigateToRecipes());

\_ = InitializeAsync();

}

private async Task InitializeAsync()

{

await LoadAvailableFiltersAsync();

await LoadGiftsAsync();

}

private async Task LoadAvailableFiltersAsync()

{

AvailableTiers = await \_dataService.GetAvailableTiersAsync();

var statuses = await \_dataService.GetAvailableStatusesAsync();

AvailableStatuses.AddRange(statuses);

OnPropertyChanged(nameof(AvailableTiers));

OnPropertyChanged(nameof(AvailableStatuses));

}

private async Task LoadGiftsAsync()

{

var gifts = await \_dataService.GetGiftsAsync(SearchText, SelectedTier,

SelectedStatus == "All" ? null : SelectedStatus);

Gifts = new ObservableCollection<EgoGift>(gifts);

}

private void ClearFilters()

{

SearchText = string.Empty;

SelectedTier = null;

SelectedStatus = "All";

}

private static void ShowHelp()

{

System.Windows.MessageBox.Show(

"Для просмотра деталей дара просто кликните на него в списке.\n\n" +

"Фильтры:\n- Поиск: по названию, описанию или статусу\n- Уровень: фильтр по Tier\n- Статус: фильтр по эффекту",

"Помощь");

}

private void SelectGift(object parameter)

{

if (parameter is EgoGift gift)

{

\_navigationService.NavigateToGiftDetail(gift);

}

}

}

}

#### ****Класс AppDbContext для работы с базой данных****

**Назначение:** Контекст базы данных Entity Framework Core. Определяет структуру базы данных SQLite, настраивает связи между таблицами и предоставляет точки доступа к данным через свойства DbSet<T>.

using Microsoft.EntityFrameworkCore;

using EGO\_Library.Models;

using System.IO;

using Microsoft.Extensions.Logging;

namespace EGO\_Library.Data

{

public class AppDbContext : DbContext

{

public DbSet<EgoGift> EgoGifts { get; set; }

public DbSet<Sources> Sources { get; set; }

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

public DbSet<User> Users { get; set; }

// Путь к БД

public string DbPath { get; }

public AppDbContext()

{

// БД будет рядом с приложением

var appPath = AppDomain.CurrentDomain.BaseDirectory;

DbPath = Path.Combine(appPath, "ego\_library.db");

}

// Дополнительный конструктор с параметром (чтобы не было ошибки CS1729)

public AppDbContext(string dbPath)

{

DbPath = dbPath;

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (!optionsBuilder.IsConfigured)

{

var fullPath = Path.GetFullPath(DbPath);

Console.WriteLine($"Configuring DB at: {fullPath}");

optionsBuilder.UseSqlite($"Data Source={fullPath}");

optionsBuilder.EnableSensitiveDataLogging();

optionsBuilder.LogTo(message => System.Diagnostics.Debug.WriteLine(message),

LogLevel.Information);

}

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<EgoGift>(entity =>

{

entity.HasKey(e => e.Id);

entity.HasIndex(e => e.Name).IsUnique();

entity.HasMany(e => e.Sources)

.WithOne(s => s.EgoGift)

.HasForeignKey(s => s.EgoGiftId)

.OnDelete(DeleteBehavior.Cascade);

});

modelBuilder.Entity<Recipes>(entity =>

{

entity.HasKey(r => r.Id);

entity.HasOne(r => r.ResultGift)

.WithMany(g => g.ResultRecipes)

.HasForeignKey(r => r.ResultGiftId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasMany(r => r.RequiredGifts)

.WithMany(g => g.RequiredInRecipes)

.UsingEntity<Dictionary<string, object>>(

"RecipeRequiredGifts",

j => j.HasOne<EgoGift>().WithMany().HasForeignKey("EgoGiftId"),

j => j.HasOne<Recipes>().WithMany().HasForeignKey("RecipesId"),

j => j.HasKey("RecipesId", "EgoGiftId"));

});

}

}

#### }

#### ****Базовый класс BaseViewModel и RelayCommand****

**Назначение:**

BaseViewModel — реализует интерфейс INotifyPropertyChanged, что позволяет автоматически обновлять пользовательский интерфейс при изменении свойств ViewModel.

using System.ComponentModel;

using System.Runtime.CompilerServices;

using System.Collections.Generic;

namespace EGO\_Library.ViewModels

{

public abstract class BaseViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler? PropertyChanged;

protected virtual void OnPropertyChanged([CallerMemberName] string? propertyName = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

protected bool SetProperty<T>(ref T field, T value, [CallerMemberName] string? propertyName = null)

{

if (EqualityComparer<T>.Default.Equals(field, value))

return false;

field = value;

OnPropertyChanged(propertyName);

return true;

}

}

}

RelayCommand — реализация паттерна Command, позволяющая привязывать действия пользователя (клики по кнопкам) к методам в ViewModel.

using System;

using System.Windows.Input;

namespace EGO\_Library.Services

{

public class RelayCommand : ICommand

{

private readonly Action<object> \_execute;

private readonly Func<object, bool> \_canExecute;

public RelayCommand(Action<object> execute, Func<object, bool> canExecute = null)

{

\_execute = execute ?? throw new ArgumentNullException(nameof(execute));

\_canExecute = canExecute;

}

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public bool CanExecute(object parameter)

{

return \_canExecute == null || \_canExecute(parameter);

}

public void Execute(object parameter)

{

\_execute(parameter);

}

}

}

#### ****Сервис навигации (NavigationService.cs)****

**Назначение:** Реализует механизм переключения между различными View (окнами/пользовательскими контролами) в приложении. Является ключевым компонентом для поддержания чистоты архитектуры MVVM, так как позволяет ViewModel управлять навигацией без прямых ссылок на View.

using EGO\_Library.Models;

using EGO\_Library.ViewModels;

using EGO\_Library.Views.Controls;

namespace EGO\_Library.Services

{

public class NavigationService : INavigationService

{

private readonly MainViewModel \_mainViewModel;

private readonly DataService \_dataService;

private readonly IAuthService \_authService;

public NavigationService(MainViewModel mainViewModel, IAuthService authService, DataService dataService)

{

\_mainViewModel = mainViewModel;

\_authService = authService;

\_dataService = dataService;

}

// основная навигация

public void NavigateToGiftList()

{

var giftListView = new GiftListView();

giftListView.DataContext = new GiftListViewModel(\_dataService, this);

\_mainViewModel.CurrentView = giftListView;

}

public void NavigateToRecipes()

{

var recipeView = new RecipeView();

recipeView.DataContext = new RecipeViewModel(this, \_dataService);

\_mainViewModel.CurrentView = recipeView;

}

public void NavigateToAbout()

{

var aboutView = new AboutView();

aboutView.DataContext = new AboutViewModel(this);

\_mainViewModel.CurrentView = aboutView;

}

public void NavigateToGiftDetail(EgoGift gift)

{

var giftDetailView = new GiftDetailView();

giftDetailView.DataContext = new GiftDetailViewModel(gift, this, \_dataService);

\_mainViewModel.CurrentView = giftDetailView;

}

// атризаця

public void NavigateToLogin()

{

\_mainViewModel.ShowLoginView();

}

public void NavigateToRegister()

{

\_mainViewModel.ShowRegisterView();

}

public void GoBack()

{

if (\_authService.IsAuthenticated)

{

NavigateToGiftList();

}

else

{

NavigateToLogin();

}

}

}

}

#### 

#### ****Окно детальной информации о даре (GiftDetailView.xaml)****

**Назначение:** XAML-разметка окна, отображающего полную информацию о выбранном EGO Gift. Содержит привязки ко всем свойствам дара: описание, эффекты, стоимость, а также списки связанных источников получения и рецептов.

<UserControl x:Class="EGO\_Library.Views.Controls.GiftDetailView"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:converters="clr-namespace:EGO\_Library.Converters"

xmlns:sys="clr-namespace:System;assembly=mscorlib">

<UserControl.Resources>

<converters:ImagePathConverter x:Key="ImagePathConverter"/>

</UserControl.Resources>

<Grid Margin="10">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<DockPanel Grid.Row="0" LastChildFill="False" Margin="0,0,0,10">

<TextBlock Text="EGO Gift"

FontSize="20"

FontWeight="Bold"

DockPanel.Dock="Left"

VerticalAlignment="Center"/>

<StackPanel Orientation="Horizontal" DockPanel.Dock="Right">

<Button Content="Help"

Width="60"

Margin="0,0,5,0"

Command="{Binding HelpCommand}"/>

<Button Content="Back"

Width="60"

Command="{Binding GoBackCommand}"/>

</StackPanel>

</DockPanel>

<Grid Grid.Row="1">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="180"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Border Background="#2D2D30"

BorderBrush="#404040"

BorderThickness="1"

CornerRadius="8"

Padding="20"

Margin="0,0,10,0"

VerticalAlignment="Top">

<Image Source="{Binding Gift.ImagePath, Converter={StaticResource ImagePathConverter}}"

Width="120"

Height="120"

HorizontalAlignment="Center"

VerticalAlignment="Center"/>

</Border>

<StackPanel Grid.Column="1">

<TextBlock Text="{Binding Gift.Name}"

FontSize="20"

FontWeight="Bold"

Margin="0,0,0,5"/>

<TextBlock Text="{Binding Gift.Tier, StringFormat='Tier: {0}'}"

FontWeight="SemiBold"

Margin="0,0,0,5"/>

<TextBlock Text="Status:"

FontWeight="SemiBold"/>

<TextBlock Text="{Binding Gift.Status}"

Margin="0,0,0,10"/>

<TextBlock Text="Effect:"

FontWeight="SemiBold"/>

<TextBlock Text="{Binding Gift.Effect}"

TextWrapping="Wrap"

Margin="0,0,0,10"/>

<TextBlock Text="Description:"

FontWeight="SemiBold"/>

<TextBlock Text="{Binding Gift.Description}"

TextWrapping="Wrap"

Margin="0,0,0,10"/>

<TextBlock Text="Sources:"

FontWeight="SemiBold"

Margin="0,10,0,5"/>

<ItemsControl ItemsSource="{Binding Gift.Sources}">

<ItemsControl.ItemTemplate>

<DataTemplate>

<Border Background="#404040"

CornerRadius="4"

Padding="5"

Margin="0,2">

<StackPanel>

<TextBlock Text="{Binding Location}"

FontWeight="SemiBold"/>

<TextBlock Text="{Binding Type, StringFormat='Type: {0}'}"

FontSize="10"

Foreground="#CCCCCC"/>

<TextBlock Text="{Binding DropRate, StringFormat='Drop Rate: {0:P1}'}"

FontSize="10"

Foreground="#CCCCCC"/>

</StackPanel>

</Border>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</StackPanel>

</Grid>

</Grid>

</UserControl>