

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

<i>Факультет</i>	<i>Кафедра</i>
вычислительной техники	МОиПЭВМ
Направление подготовки	09.03.04 «Программная инженерия»
Направленность	Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Клиентская часть комплекса программных
средств для автоматизации написания игровых
скриптов**

Студент	_____	Рязанский Максим Андреевич
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	Козлов А.Ю.
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой	_____	Козлов А.Ю.
	(подпись)	(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК	_____	Попова Н.А.
	(подпись)	(фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Рязанский Максим Андреевич гр. 22ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>
2. Тема работы Клиентская часть комплекса программных средств для автоматизации
написания игровых скриптов

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы д.т.н., профессор, заведующий кафедрой «МО и ПЭВМ» Козлов
А.Ю.

4. Задание на работу (назначение разработки, исходные данные и
т.п.)

1) Назначение: Программное обеспечение автоматизации написания скриптов

2) Среда выполнения: операционная система Windows 11

Языковые средства разработки: C#, XAML, UML

Инструментальные средства: MS Visual Studio 2022, Visual Paradigm

Технологии разработки: ООП, паттерн проектирования MVVM.

3) Основные функции приложения:

а) визуальное конструирование игровых рецептов;

б) локальное извлечение, декодирование и отображение игровых метаданных;

в) поиск, фильтрация и навигация по базе предметов;

г) сохранение сгенерированных скриптов в файл и итеративная запись;

- 1) Анализ предметной области и постановка задачи на разработку;
- 2) Проектирование клиентской части;
- 3) Реализация программного обеспечения;
- 4) Тестирование графического интерфейса;
- 5) Оформление пояснительной записки.

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 85 листов, 21 рисунок, 14 таблиц, 21 использованный источник, 2 приложения.

АВТОМАТИЗАЦИЯ ГЕНЕРАЦИИ СКРИПТОВ, UX ДИЗАЙН, ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС.

Объект исследования – программное обеспечение автоматизации игровых скриптов.

Цель выпускной квалификационной работы – спроектировать и разработать программные средства, предоставляющие пользователю доступ к удобным и практичным средствам генерации игровых скриптов для изменения игры.

Технологии разработки – разработка средств визуализации производится в среде разработки Microsoft Visual Studio 2022 на объектно-ориентированном языке программирования C#. Развертывание основного приложения осуществляется на операционных системах семейства Windows версии 10 и выше.

Результаты работы – в ходе выполнения выпускной квалификационной работы были спроектированы и разработаны программные средства, позволяющие автоматизировать написание игровых скриптов.

					ПГУ 09.03.04 – 08ВП221.15 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Клиентская часть комплекса программных средств для автоматизации написания игровых скриптов. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Рязанский М.А.					4	85
Пров.		Козлов А.Ю.				Группа 22ВП1		
Н. Контр.		Такташкин Д.В.						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и постановка задачи на разработку системы автоматизации написания KubeJS-скриптов	9
1.1 Анализ предметной области.....	9
1.2 Анализ программного обеспечения аналогичного назначения.....	10
1.3 Постановка задачи на разработку программных средств	12
1.4 Анализ функциональных требований	13
1.5 Анализ нефункциональных требований	16
1.6 Планирование разработки и оценка бюджета	17
2 Проектирование клиентской части.....	21
2.1 Проектирование архитектуры приложения	21
2.2 Проектирование пользовательского интерфейса.....	22
2.3 Проектирование программных средств	27
3 Реализация программного обеспечения.....	30
3.1 Реализация интерфейса/уровня представлений	35
3.2 Реализация архитектурной части приложения/уровня модели представлений	39
4 Тестирование графического интерфейса добавления рецептов.....	44
4.1 Тестирование модуля добавления рецепта методом черного ящика	44
4.1.1 Выбор методики тестирования	44
4.1.2 Функциональная диаграмма добавления рецепта	44
4.1.3 Таблица решений для добавления рецепта	46
4.2 Тестирование графического интерфейса для отображения данных	48

4.2.1	Выбор средств тестирования графического интерфейса	48
4.2.2	Результаты тестирования графического интерфейса	50
4.3	Анализ качества кода	54
4.3.1	Количественные метрики кода (SLOC)	54
4.3.2	Метрика Холстеда	56
	Заключение	60
	Список использованных источников	62
	Приложение А (справочное) Глоссарий	65
	Приложение Б (обязательное) Код программы.....	67

Введение

Современная разработка и поддержание сборок для Minecraft усложняется с каждым днем по мере появления новых модификаций и изменения старых. Из-за этого поддержка сервера с помощью платформы KubeJS становится крайне трудоёмким и подверженным ошибкам процессом. Ручное написание скриптов характеризуется необходимостью глубокого знания синтаксиса JavaScript и специфики API игровой среды, а также критической уязвимостью к синтаксическим и логическим ошибкам при масштабировании проектов. Разработка специализированного комплекса программных средств для автоматизации написания игровых скриптов позволяет перевести процесс конфигурации в визуальную среду, абстрагируя пользователя от рутинного кодирования.

Целью данной работы является разработка пользовательского интерфейса для автоматизации генерации корректных KubeJS-скриптов, что позволит сократить трудоёмкость разработки и минимизировать количество ошибок ручного ввода. Разработка будет проводиться на языке программирования C# с использованием платформы WinUI 3, основанном на языке разметки XAML.

Для достижения поставленной цели необходимо решить следующие задачи:

- а) разработать WinUI 3 приложение с современным графическим интерфейсом, обеспечивающим интуитивное управление рецептами без необходимости написания кода;
- б) реализовать компонент поиска и выбора игровых предметов с фильтрацией по названию, идентификатору и принадлежности к модификации, а также отображением иконок предметов;
- в) разработать визуальный конструктор рецептов с поддержкой сложных параметров: вероятностных выходов, жидкостных ингредиентов, тегов предметов и модификаторов цепочек вызовов;

- d) обеспечить взаимодействие клиентской части с серверным API для получения игровых данных и передачи конфигураций рецептов на генерацию;
- e) провести всестороннее тестирование клиентских компонентов для обеспечения стабильной работы и корректного отображения данных.

В ходе проектирования будут рассмотрены основные аспекты разработки подобных систем, включая проектирование архитектуры клиентского приложения, разработка пользовательского интерфейса и интеграцию различных компонентов с игровыми данными и внешними источниками.

1 Анализ предметной области и постановка задачи на разработку системы автоматизации написания KubeJS-скриптов

1.1 Анализ предметной области

Предметная область разработки охватывает сферу разработки модификаций для игры «Minecraft», в частности – процесс настройки игровых механик, скриптов и баланса с помощью платформы KubeJS [1]. Современные сборки модификаций представляют собой сложные программные конфигурации, включающие десятки модификаций и сотни сложных рецептов, событий и условий.

Платформа KubeJS, основанная на JavaScript, предоставляет мощный и гибкий инструмент для серверной настройки игровых сборок, однако её использование требует от разработчика глубоких знаний синтаксиса языка, специфики API и внутренней логики игровой среды. Ручное написание и поддержка программного кода характеризуется высокой трудоёмкостью, подверженностью синтаксическим и логическим ошибкам, а также сложностью масштабирования при увеличении количества конфигураций.

Ключевыми сущностями предметной области являются: рецепты (объекты с типом, входами, выходами и модификаторами), игровые объекты (предметы, жидкости, блоки с их метаданными: идентификаторами, тегами, локализованными названиями) и сами скрипты – файлы .js, содержащие вызовы `ServerEvents.recipes(event => {...})` [1].

Наиболее эффективным подходом для реализации клиентской части системы является разработка специализированного пользовательского интерфейса на базе языка C# с использованием библиотеки WinUI 3 платформы .NET. Подобное решение значительно упростит процесс создания игровых рецептов, снизит порог входа для начинающих разработчиков, обеспечит возможность расширения под новые модификации и минимальные накладные расходы при интеграции в существующие инструменты разработки, оставаясь при этом независимым от конкретной среды отображения данных.

Таким образом, предметная область характеризуется высокой сложностью данных, строгими требованиями к корректности генерируемого кода и выраженной потребностью в инструментах, способных абстрагировать пользователя от рутинного кодирования и повысить надёжность процесса разработки игровых сборок.

1.2 Анализ программного обеспечения аналогичного назначения

В настоящее время существует несколько подходов к решению задачи настройки игровых скриптов, однако специализированных инструментов с полноценным графическим интерфейсом практически нет. Для проведения сравнительного анализа были рассмотрены наиболее релевантные решения, предоставляющие различные возможности для генерации игровых рецептов: KubeJS, CraftTweaker и MCreator [1 – 3].

KubeJS – современное решение для создания игровых скриптов на языке JavaScript [1]. Позволяет динамически добавлять/удалять рецепты, теги, события, NBT-данные и многое другое. KubeJS поддерживает сложную логику: условные проверки, циклы, модификацию свойств предметов без перезагрузки системы и интеграцию со множеством других модификаций через официальные дополнения. Экосистема активно развивается, а сообщество регулярно обновляет документацию под новые версии игры.

Преимущества KubeJS:

- a) наличие горячей перезагрузки командой без перезапуска сервера/клиента;
- b) кроссплатформенность и адаптация под актуальные версии Minecraft;
- c) стабильный api с поддержкой механик create, mekanism, ae2 и других модификаций;
- d) гибкая архитектура: можно писать, как простые рецепты, так и сложные скрипты.

Недостатки:

- a) требует базового понимания javascript и внутренней структуры игровой среды;
- b) отсутствие графического интерфейса, разработка ведётся исключительно в текстовых редакторах;
- c) диагностика ошибок часто ограничивается общими сообщениями;
- d) api может меняться между версиями kubejs и minecraft, что требует постоянного обновления шаблонов генерации.

CraftTweaker представляет собой модификацию, которая позволяет добавлять новые рецепты, редактировать и удалять старые и изменять другие аспекты игрового процесса [2]. CraftTweaker использует собственный язык ZenScript и весь код прописывается пользователем вручную на этом языке.

Преимущества CraftTweaker:

- a) стабильный и хорошо документированный api для исторических версий;
- b) обширная библиотека готовых скриптов и решений;
- c) встроенная поддержка множества модификаций через отдельные дополнения и надстройки.

Недостатки:

- a) полное отсутствие поддержки для новых версий minecraft, ограниченная поддержка современных механик и новых api загрузчиков;
- b) синтаксис zenscript менее гибок по сравнению с javascript, что усложняет реализацию сложной логики;
- c) отсутствие графического интерфейса, необходимость ручной проверки логики и синтаксиса скриптов.

MCreator – программное обеспечение с открытым кодом для создания модификаций без написания кода [3]. Включает встроенный редактор рецептов с графическим интерфейсом и автоматической генерацией Java-кода, который компилируется в готовую модификацию через встроенный Gradle-скрипт. Программа скрывает сложность разработки за графическим слоем,

автоматически управляя зависимостями, сборкой и регистрацией контента в игре.

Преимущества MCreator:

- a) интуитивный графический интерфейс с предварительным просмотром рецептов и автоматической проверкой конфликтов;
- b) поддержка множества модификаций через плагины;
- c) отсутствие необходимости изучать java или настраивать рабочее окружение;
- d) генерация полноценной модификации «в один клик» с корректной регистрацией в игре.

Недостатки:

- a) генерирует избыточный java-код, который сложно редактировать вручную;
- b) архитектурно замкнут на собственной реализации;
- c) слабая совместимость с механиками более масштабных модификаций;
- d) высокая нагрузка на систему из-за встроенного компилятора и большого количества зависимостей.

Таким образом, существующие решения либо требуют знания языков программирования, либо слишком сильно нагружают систему. Разрабатываемое приложение предлагает простой и понятный графический интерфейс, не требующий глубоких знаний языков программирования, визуальное конструирование любых типов рецептов KubeJS, поддержкой синтаксического анализа существующих скриптов и валидацией на уровне предметной области.

1.3 Постановка задачи на разработку программных средств

Разрабатываемое программное средство представляет собой клиентское приложение на платформе WinUI 3 (.NET 8) [4 – 6], предназначенное для визуального проектирования и автоматической генерации скриптов KubeJS. Его основная цель – предоставить авторам игровых сборок интуитивно понятный

визуальный инструмент, не требующий от пользователя знания JavaScript или иных языков программирования.

Система должна предоставить удобный графический интерфейс, позволяющий создавать и настраивать различные типы рецептов, поддерживаемых платформой KubeJS, механизм автоматической генерации валидного JavaScript-кода с учётом синтаксических норм KubeJS через взаимодействие с серверным API. Интерфейс должен обеспечивать строгую визуальную валидацию входных данных и автоматическое сопоставление игровых предметов с их локализованными названиями и иконками. Также необходима поддержка импорта рецептов из .js-файлов, их преобразование в понятный редактируемый вид и дальнейшее их изменение с повторной генерацией.

Приложение ориентировано на работу в среде Windows 10/11 и должно обеспечивать отзывчивый интерфейс даже при работе с большим объемом игровых объектов.

1.4 Анализ функциональных требований

Построение диаграммы вариантов использования позволяет наглядно отобразить функциональные требования к системе или программному продукту с точки зрения конечного пользователя, абстрагируясь от внутренней реализации. Она позволяет определить границы системы, идентифицировать внешних актеров и установить перечень действий, которые должна выполнять система для удовлетворения потребностей пользователей. Для разрабатываемого приложения главным актёром является разработчик игровых сборок, а варианты использования включают создание, редактирование, удаление и импорт рецептов KubeJS.

Диаграмма вариантов использования для клиентской части комплекса программных средств для автоматизации написания игровых скриптов приведена на рисунке 1 [7, 8].

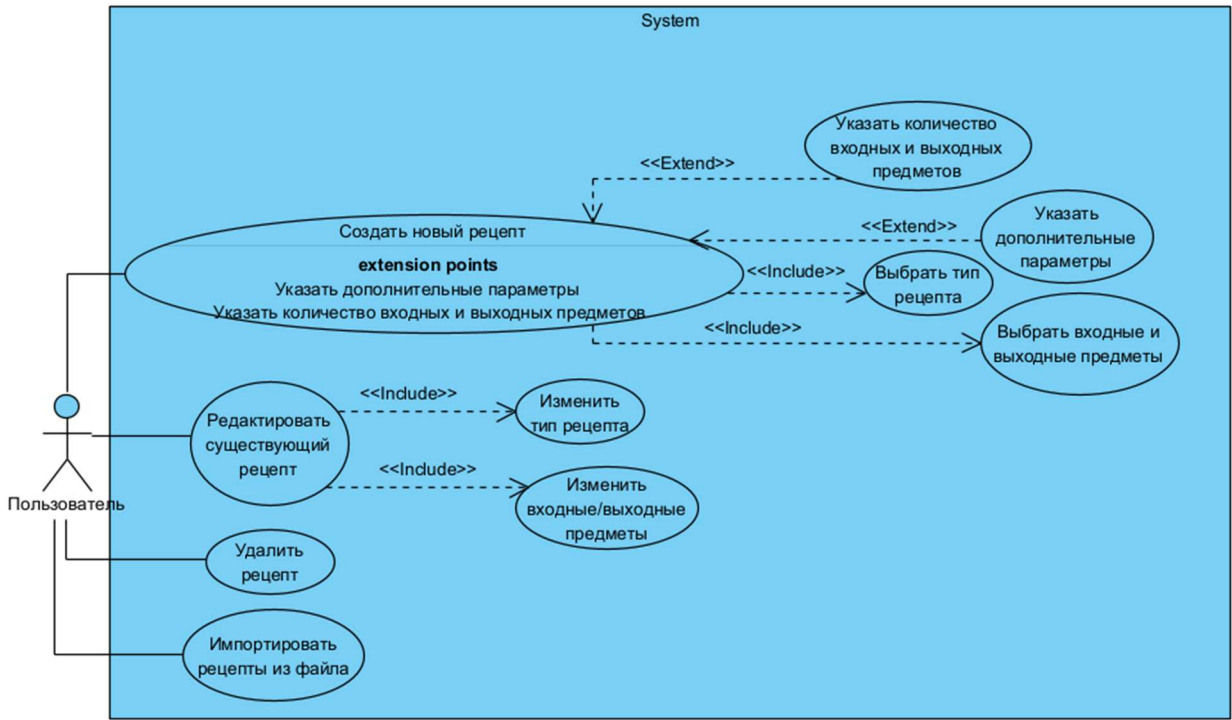


Рисунок 1 – Диаграмма вариантов использования приложения

Спецификация варианта использования «Создать новый рецепт» приведена на рисунке 2 [8].

Наименование: Создать новый рецепт
ID: 1
Краткое описание: Пользователь создает новый игровой рецепт
Действующие лица: Пользователь
Предусловие: -
Основной поток: 1. Пользователь выбирает тип нового рецепта. 2. В зависимости от типа рецепта, пользователь устанавливает модификаторы рецепта и количество входных/выходных предметов. 3. Пользователь выбирает входные предметы. 4. Пользователь выбирает выходные предметы. 5. Пользователь указывает файл для сохранения и инициирует создание рецепта. 6. Система проверяет данные на корректность. 7. Система добавляет рецепт в указанный файл.
Постусловие: Создан новый игровой рецепт

Рисунок 2 – Спецификация варианта использования «Создать новый рецепт»

Спецификация варианта использования «Редактировать существующий рецепт» приведена на рисунке 3 [8].

Наименование: Редактировать существующий рецепт
ID: 2
Краткое описание: Пользователь редактирует существующий игровой рецепт
Действующие лица: Пользователь
Предусловие: Пользователь импортировал список рецептов из файла
Основной поток: 1. Пользователь выбирает рецепт из списка для изменения 2. По желанию пользователь выбирает новый тип рецепта и <u>модификаторы</u> . 3. Пользователь заменяет входные предметы. 4. Пользователь заменяет выходные предметы. 5. Пользователь инициирует сохранение рецепта. 6. Система проверяет данные на корректность. 7. Система заменяет старый рецепт на новый.
Постусловие: Изменен игровой рецепт

Рисунок 3 – Спецификация варианта использования «Редактировать существующий рецепт»

Спецификация варианта использования «Удалить рецепт» приведена на рисунке 4 [8].

Наименование: Удалить рецепт
ID: 3
Краткое описание: Пользователь удаляет игровой рецепт
Действующие лица: Пользователь
Предусловие: Пользователь импортировал список рецептов из файла
Основной поток: 1. Пользователь выбирает рецепт из списка для удаления. 2. Пользователь инициирует удаление рецепта. 3. Система удаляет выбранный рецепт.
Постусловие: Удален игровой рецепт

Рисунок 4 – Спецификация варианта использования «Удалить рецепт»

Спецификация варианта использования «Импортировать рецепты из файла» приведена на рисунке 5 [8].

Наименование: Импортировать рецепты из файла
ID: 4
Краткое описание: Пользователь импортирует игровые рецепты из .js файла
Действующие лица: Пользователь
Предусловие: -
Основной поток: 1. Пользователь выбирает файл с рецептами. 2. Система читает и обрабатывает файл. 3. Система выводит обработанные данные на экран.
Постусловие: Импортированы рецепты из файла

Рисунок 5 – Спецификация варианта использования «Импортировать рецепты из файла»

Проведённый анализ функциональных требований позволил сформировать перечень возможностей клиентского приложения. Указанные требования служат объективными критериями для этапов разработки и тестирования, снижая риски расхождения между ожиданиями пользователей и реализованным функционалом.

Таким образом, проведенный анализ гарантирует, что разрабатываемое приложение будет обладать необходимым, достаточным функционалом для эффективной автоматизации написания игровых скриптов.

1.5 Анализ нефункциональных требований

Разработка клиентской части программного обеспечения осуществляется в интегрированной среде Microsoft Visual Studio 2026 на языке программирования C# с использованием платформы WinUI 3 и .NET 8. Система функционирует в операционных системах Windows 10/11.

Интерфейс приложения должен реагировать на действия пользователя не более чем через 200 миллисекунд. Загрузка и отображение списка игровых предметов, должны завершаться не более чем за 3 секунды. Фильтрация списка предметов по поисковому запросу должна выполняться не более чем за 100 миллисекунд.

Приложение должно корректно обрабатывать неподдерживаемые .js-файлы, не допуская аварийного завершения работы. При возникновении ошибок чтения или записи файлов пользователю должно отображаться сообщение с указанием причины ошибки. Целостность редактируемых файлов должна сохраняться: изменения вносятся только после явного подтверждения со стороны пользователя, допускается ввод новых данных в любое место файла.

Интерфейс должен быть доступен пользователям без навыков программирования: все операции с рецептами выполняются через графические элементы управления без необходимости ручного ввода кода. Приложение не требует установки дополнительных зависимостей помимо .NET 8 и поставляется как самодостаточный исполняемый файл.

1.6 Планирование разработки и оценка бюджета

Планирование разработки и оценка бюджета являются неотъемлемыми этапами управления жизненным циклом программного обеспечения. Тщательное планирование позволяет обеспечить контроль сроков, оптимальное распределение ресурсов и минимизацию финансовых и технических рисков.

Для оценки временных затрат и планирования разработки программного обеспечения была построена диаграмма Ганта [9]. Общий процесс разработки содержит пять ключевых этапов. Каждый этап включает в себя определенный набор подзадач, которые необходимо выполнить для перехода к следующему этапу:

- a) анализ предметной области и постановка задачи на разработку;
- b) проектирование программного обеспечения: разработка макетов экранов и модели данных интерфейса;

с) реализация программного обеспечения: разработка визуального конструктора рецептов, компонента поисковой строки для предметов и интеграция с серверным API;

d) контроль качества и тестирование пользовательского интерфейса;

e) оформление отчётной документации.

На рисунке 6 представлен календарный график проекта, который отражает выделенные задачи, время их выполнения и ресурсы, необходимые для выполнения каждой из них.

Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
Начало проекта	0 дней	Пт 08.05.26	Пт 08.05.26		
▢ Анализ предметной области и постановка задачи	4 дней	Пт 08.05.26	Ср 13.05.26		
Анализ предметной области	0,5 дней	Пт 08.05.26	Пт 08.05.26	1	Аналитик
Анализ ПО аналогичного назначения	0,5 дней	Пт 08.05.26	Пт 08.05.26	3	Аналитик
Постановка задачи	0,5 дней	Пн 11.05.26	Пн 11.05.26	4	Аналитик
Анализ функциональных требований	1 день	Пн 11.05.26	Вт 12.05.26	5	Аналитик
Анализ нефункциональных требований	0,5 дней	Вт 12.05.26	Вт 12.05.26	6	Аналитик
Планирование разработки и оценка бюджета	1 день	Ср 13.05.26	Ср 13.05.26	7	Аналитик
▢ Проектирование ПО	9 дней	Чт 14.05.26	Вт 26.05.26		
Проектирование архитектуры приложения	2 дней	Чт 14.05.26	Пт 15.05.26	8	Разработчик
Проектирование модели данных интерфейса	2 дней	Пн 18.05.26	Вт 19.05.26	10	Разработчик
Проектирование взаимодействия с серверным API	2 дней	Ср 20.05.26	Чт 21.05.26	11	Разработчик
Проектирование программных средств	3 дней	Пт 22.05.26	Вт 26.05.26	12	Разработчик
▢ Реализация ПО	21 дней	Ср 27.05.26	Ср 24.06.26		
Реализация отображения списка рецептов	3 дней	Ср 27.05.26	Пт 29.05.26	13	Разработчик
Реализация компонента поиска и выбора игровых предметов	5 дней	Пн 01.06.26	Пт 05.06.26	15	Разработчик
Реализация визуального конструктора рецептов	7 дней	Пн 08.06.26	Вт 16.06.26	16	Разработчик
Реализация открытия рецептов из файлов	2 дней	Ср 17.06.26	Чт 18.06.26	17	Разработчик
Реализация интеграции с серверным API	4 дней	Пт 19.06.26	Ср 24.06.26	18	Разработчик
▢ Тестирование ПО	7 дней	Чт 25.06.26	Пт 03.07.26		
Тестирование поиска предметов	2 дней	Чт 25.06.26	Пт 26.06.26	19	Тестирующий
Тестирование визуального конструктора рецептов	2 дней	Пн 29.06.26	Вт 30.06.26	21	Тестирующий
Тестирование открытия, изменения и удаления рецептов из .	2 дней	Ср 01.07.26	Чт 02.07.26	22	Тестирующий
Оценка производительности интерфейса	1 день	Пт 03.07.26	Пт 03.07.26	23	Тестирующий
Оформление документации	7 дней	Пн 06.07.26	Вт 14.07.26	24	Аналитик
Конец проекта	0 дней	Вт 14.07.26	Вт 14.07.26	25	

Рисунок 6 – Сведения об этапах разработки приложения

Первый этап направлен на изучение предметной области и формирование требований к разрабатываемому приложению. Проводятся анализ предметной области, изучение технологии настройки игровых скриптов KubeJS, анализ аналогичных решений, формирование функциональных и нефункциональных требований и планирование разработки с оценкой бюджета.

На втором этапе проводится планирование проектирования структуры приложения, модели данных интерфейса и взаимодействие с серверным API, выбираются программные средства реализации.

Третий этап включает в себя реализацию клиентской части приложения. Разрабатывается главное окно с отображением списка предметов, визуальный конструктор рецептов, компонент поиска предметов с поддержкой поиска по локализованным названиям и тегам, функционал импорта существующих рецептов из файла и добавления новых, а также интеграция с серверным API для генерации скриптов.

Четвёртый этап предполагает тестирование разработанного приложения. Проверяется корректность работы поиска предметов по названию и тегам, тестируется визуальный конструктор рецептов на всех поддерживаемых типах рецептов, проверяются операции изменения и удаления рецептов из .js-файлов, а также проводится оценка производительности интерфейса.

Последний этап направлен на оформление документации по приложению, описание порядка работы с приложением и финальную проверку соответствия полученных результатов исходным требованиям.

Для наглядного отражения временных характеристик разработки программного обеспечения была построена диаграмма Ганта, приведенная на рисунке 7.

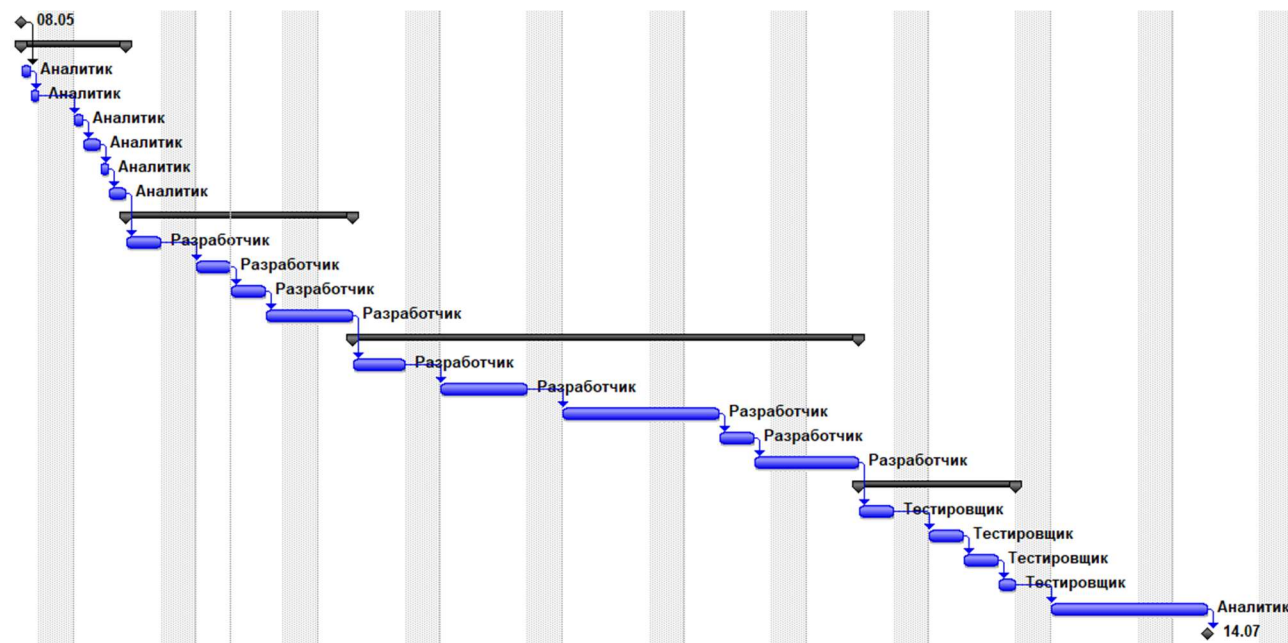


Рисунок 7 – Диаграмма Ганта

Для оценки финансовых затрат необходимо назначить цены ресурсов. Стоимость используемых ресурсов представлена на рисунке 8.

Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных	Затраты на исполыз.	Начисление	Базовый календарь
Аналитик	Трудовой		A		100%	800,00 Р/ч	0,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
Разработчик	Трудовой		P		100%	1 200,00 Р/ч	0,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
Тестировщик	Трудовой		T		100%	700,00 Р/ч	0,00 Р/ч	0,00 Р	Пропорциональное	Стандартный

Рисунок 8 – Стоимость задействованных ресурсов

Общая оценка финансовых затрат проекта представлена на рисунке 9.

	Длительность	Трудозатраты	Затраты
Текущие	48д	384ч	397 600,00 Р
Базовые	0д?	0ч	0,00 Р
Фактические	0д	0ч	0,00 Р
Оставшиеся	48д	384ч	397 600,00 Р

Рисунок 9 – Общая оценка финансовых затрат

По итогам оценки бюджета проекта общая стоимость разработки составила 397 600 руб., общая продолжительность работы составила 48 дней.

2 Проектирование клиентской части

2.1 Проектирование архитектуры приложения

Клиентская часть приложения для автоматизации написания игровых скриптов спроектирована с использованием паттерна MVVM (Model-View-ViewModel), что является стандартом для приложений на платформе WinUI 3 [10]. Данный подход позволяет строго разделить пользовательский интерфейс и бизнес-логику автоматизации скриптов обеспечивая слабую связанность компонентов и упрощая дальнейшее сопровождение кодовой базы.

Выбор паттерна MVVM обусловлен рядом практических преимуществ применительно к данному приложению. Логика работы с рецептами, синтаксический анализ .js-файлов и взаимодействие с серверным API не зависят от конкретного визуального представления, что позволяет тестировать их независимо от запущенного интерфейса. Механизм привязки данных, встроенный в WinUI 3, реализует связь между слоями View и ViewModel без необходимости написания избыточного кода [11].

Архитектура приложения состоит из трех логических уровней:

Уровень представления (View). Отвечает за отображение интерфейса, его элементов и обработку ввода пользователя. Содержит XAML-разметку интерфейса [11] и всех его вкладок: список предметов, строку поиска, визуальный конструктор рецептов, вкладки для удаления и открытия файлов с рецептами. Также здесь описана логика взаимодействия пользователя с приложением: обработка нажатий, выбор предметов из списка, навигация по вкладкам, работа с рецептами в файле.

Уровень модели представления (ViewModel). Файл MainViewModel.cs содержит информацию, отвечающую за управление состоянием пользовательского интерфейса и преобразование «сырых данные» в понятный пользователю формат. Реализует интерфейс INotifyPropertyChanged, который обеспечивает автоматическое обновление элементов интерфейса при изменении данных [12]. Здесь хранится список загруженных рецептов, состояние

поискового запроса и отфильтрованный список предметов, отображаемый в диалоге выбора.

Уровень модели (Model). Содержит классы предметной области, представляющие собой чистые данные без привязки к UI: конфигурации рецептов, описывающие их типы, ингредиенты и модификаторы, экстракторы данных, отвечающие за чтение и разбор существующих .js-файлов с рецептами, генераторы рецептов, преобразующие конфигурации в рабочий JavaScript-код, а также данные из внешней библиотеки. Модель не содержит ссылок на типы WinUI 3 и может использоваться и тестироваться полностью изолированно от интерфейса.

Взаимодействие между уровнями организовано следующим образом: View обращается к ViewModel через механизм привязки данных и команды; ViewModel обращается к Model через сервисный слой, не имея прямой зависимости от конкретных реализаций. Такое разделение обеспечивает возможность независимого тестирования уровня модели без запуска пользовательского интерфейса, а также упрощает поддержку и расширение приложения при появлении новых типов рецептов или изменении формата скриптов KubeJS.

2.2 Проектирование пользовательского интерфейса

Интерфейс приложения должен предоставлять доступ к основным возможностям программного обеспечения, быть интуитивно понятным и удобным в использовании, располагать простыми и эффективными инструментами взаимодействия с приложением.

Приложение состоит из сетки предметов, строки поиска и 3 основных вкладок: создать, удалить, файл. Эти элементы предоставляют пользователю удобный и доступный функционал для автоматизации генерации скриптов, минимизируя количество ручных действий при составлении скрипта. На рисунке 10 представлены список предметов и строка поиска.

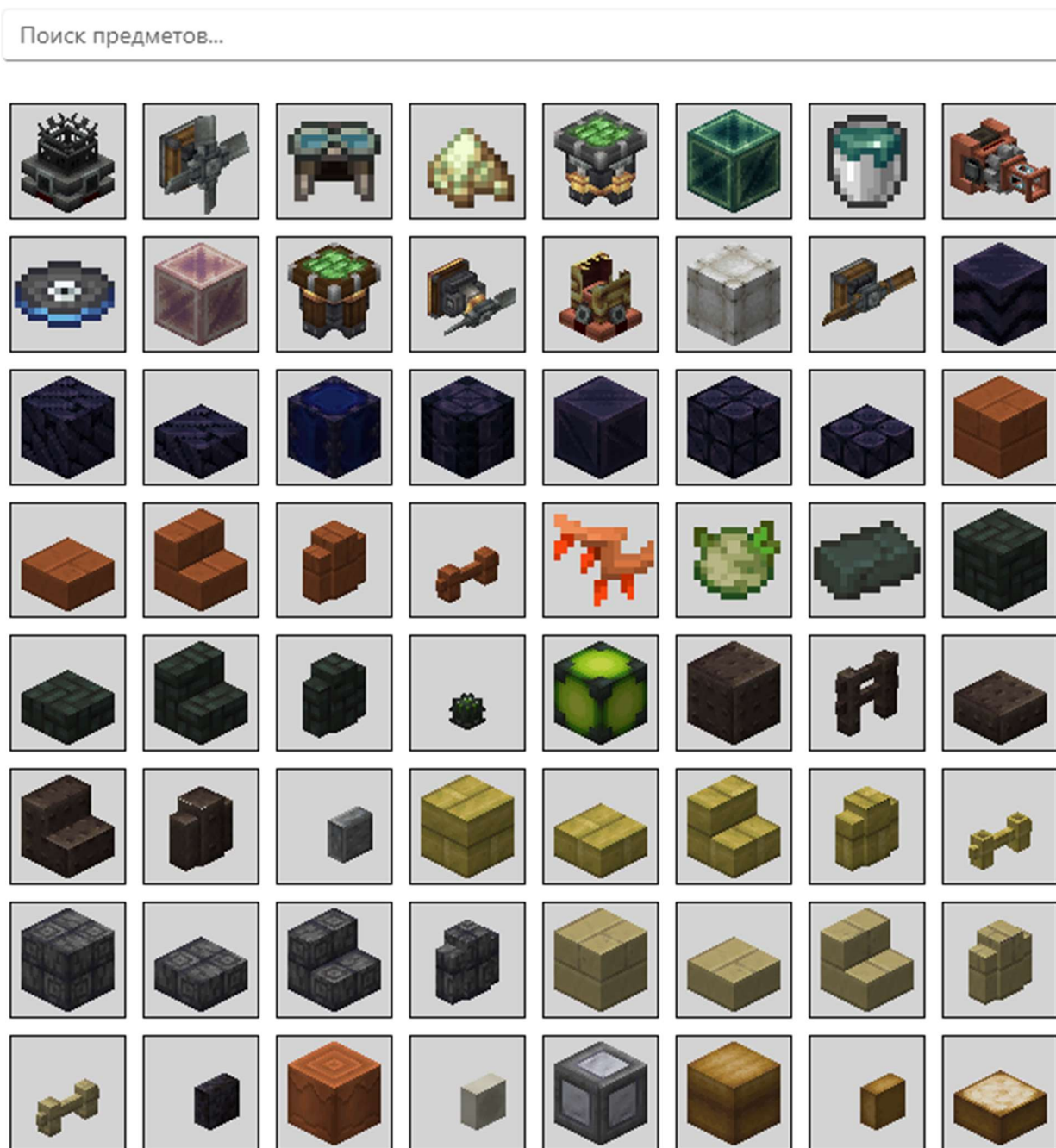


Рисунок 10 – Список предметов и строка поиска

Программа получает данные об игровых предметах и отображает в прокручиваемой сетке. Эти предметы будут использоваться для дальнейшего создания/удаления/изменения рецептов. Строка поиска предметов позволяет искать предметы по их идентификатору (который состоит из названия предмета и его модификации) и локализованному названию. В списке предметов отображаются все найденные совпадения со значением в строке поиска. На рисунке 11 представлена вкладка создания рецептов.

Тип: filling

Параметры рецепта

Входы:

? не выбран

? не выбран

Выходы:

? не выбран

Файл: имя файла (без .js)

Создать рецепт

Рисунок 11 – Создание рецептов

Вкладка создания рецептов является основной рабочей областью. В верхней части вкладки расположен выпадающий список выбора типа рецепта, охватывающий стандартные рецепты Minecraft и рецепты модификации Create [13]. При выборе типа рецепта интерфейс динамически перестраивается. В зависимости от типа устанавливается нужное количество входных и выходных предметов, добавляется возможность установить произвольное их количество, возможность установить модификаторы рецепта (количество получаемого опыта для smelting, степень нагрева для compacting, время обработки для crushing и т.д.). Для рецепта верстака отображается сетка 3*3 с ячейками для предметов, которую необязательно заполнять полностью, и один выходной предмет. После указания всех данных пользователь выбирает нужный .js-файл и сохраняет готовый рецепт. На рисунке 12 представлена вкладка удаления рецептов.

Удаление рецептов

Заполните хотя бы одно поле фильтра (AND внутри фильтра).

Output:

?

не выбран

×

Input:

?

не выбран

×

Type:

не выбран

▼

Mod:

не выбран

▼

Id:

minecraft:glowstone

Файл:

имя файла (без .js)

Удалить рецепт

Рисунок 12 – Удаление рецептов

Вкладка удаления рецептов генерирует код типа event.remove, который отвечает за удаление стандартных рецептов игры. Поддерживается удаление по входным/выходным предметам, типу, модификации и идентификатору рецепта. Таким образом при указании только выходного предмета удалятся все рецепты, которые позволяли получить выбранный предмет, при указании типа – все рецепты этого типа и т.д. Выбор всех параметров позволяет точно удалять ненужные пользователю рецепты. Так же, как и с созданием, пользователь выбирает файл для сохранения генерируемого кода. На рисунке 13 представлена вкладка работы с файлами.

C:\Users\КОМП_2024\AppData\Roaming\Crea

Обзор...

Открыть

Обновить

Загружено 5 рецептов.

create_cutting: 'architects_palette:abyssaline_tile_slab' → CreateIt

[<+]

[+>]

Удалить

smelting: minecraft:diamond → 12x architects_palette:entwine_rod

[<+]

[+>]

Удалить

smoking: aeronautics:propeller_bearing → aeronautics:aviators_goggle

[<+]

[+>]

Удалить

stonecutting: aeronautics:pearlescent_levitite → aeronautics:wooden_

[<+]

[+>]

Удалить

shaped: create:belt_connector ×3

[<+]

[+>]

Удалить

Рисунок 13 – Список рецептов

Вкладка «Файл» позволяет открыть .js-файл с существующими рецептами и отобразить их единым списком. Кнопка обзор предназначена для выбора нужного файла, кнопка обновить – для отображения изменений. У каждого рецепта есть опции его удаления, вставки нового рецепта до и после указанного (для поддержания целостной структуры файла) и редактирования рецепта путем нажатия на сам рецепт. На рисунке 14 представлена вкладка редактирования рецептов.

Тип: smelting

Параметры рецепта

Входы:

 minecraft:diamond

Выходы:

 architects_palette:entwine_rod Кол-во: 12

Редактирование: smelting: minecraft:diamond → 12x architects_palette:entwine_rod

Сохранить изменения

Отмена

Рисунок 14 – Редактирование рецепта

Вкладка изменения рецепта похожа на вкладку создания, за исключением отсутствия выбора файла. При инициации редактирования данные рецепта автоматически выбираются и отображаются. Пользователь может также изменить тип, предметы и модификаторы рецепта. Затем можно сохранить изменения или отменить редактирование.

Спроектированный интерфейс обеспечивает эффективный рабочий процесс за счет разделения задач на вкладки и минимизации ручного ввода данных. Использование визуальных иконок и контекстных меню снижает нагрузку на пользователя, позволяя сосредоточиться на логике рецептов, а не на синтаксисе кода.

2.3 Проектирование программных средств

Диаграмма классов позволяет отобразить структуру системы через классы, их свойства, методы и взаимосвязи между ними. Способствует четкой формализации архитектурных решений и облегчает процесс проектирования приложения. С помощью визуализации упрощается переход от проектирования к программной реализации. На основе архитектуры приложения была спроектирована диаграмма классов, приведенная на рисунке 15 [14]. Диаграмма отражает структуру слоя представления, построенного по паттерну MVVM с использованием платформы WinUI 3.

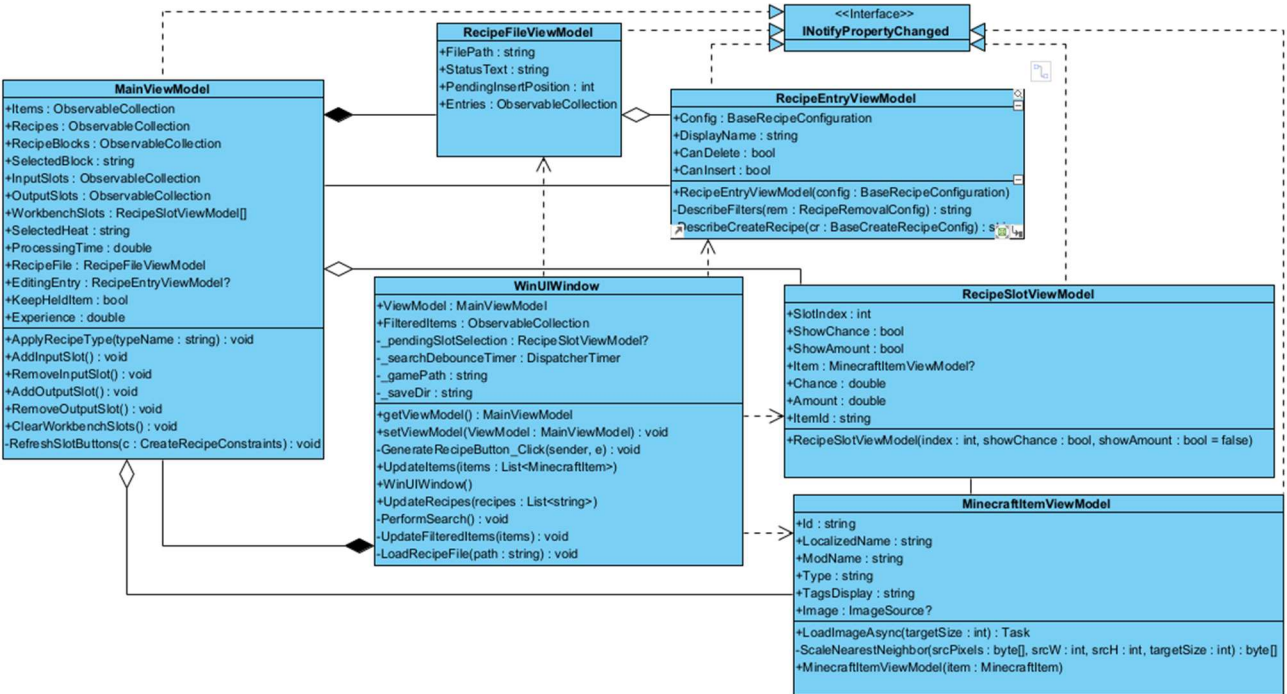


Рисунок 15 – Диаграмма классов

Спецификации приведенных классов представлены в таблице 1.

Таблица 1 – Спецификации классов

Наименование сущности	Назначение класса
RecipeSlotViewModel	Модель представления одной ячейки для предмета рецепта. Хранит ссылку на выбранный предмет, его количество и дополнительные модификаторы.

Продолжение таблицы 1

Наименование сущности	Назначение класса
MinecraftItemViewModel	Модель представления одного предмета из игры. Хранит данные о предмете (идентификатор, локализованное название, теги и т.д.) и его изображение. Отвечает за асинхронную загрузку и масштабирование изображения для правильного отображения моделей предметов.
RecipeEntryViewModel	Модель представления одной записи рецепта в списке загруженного файла. Преобразует данные рецепта в приемлемый формат и отображает в интерфейсе. Определяет доступность операций удаления и вставки рецептов.
RecipeFileViewModel	Модель представления загружаемого .js-файла с рецептами. Хранит путь к файлу, статусное сообщение с количеством рецептов, список рецептов (RecipeEntryViewModel) и позицию для вставки нового рецепта между существующими.
WinUIWindow	Главное окно приложения WinUI 3. Связывает интерфейс с MainViewModel, обрабатывает все пользовательские события: выбор предмета в ячейку, переключение типа рецепта, генерацию и сохранение кода рецепта, поиск предметов с автодополнением, загрузку и редактирование файлов рецептов. Запускает извлечение данных о предметах и рецептах из игровой директории.
MainViewModel	Центральный класс модели представления, управляет состоянием всего интерфейса. Хранит список предметов, рецептов, список доступных типов рецептов, ячейки входных и выходных предметов, модификаторы, режим редактирования. Содержит логику переключения между типами рецептов и предоставляет команды добавления/удаления ячеек предметов.
INotifyPropertyChanged	Интерфейс, уведомляющий реализующие его классы об изменении состояний элементов окна.

Разработанная диаграмма классов демонстрирует чёткое разделение ответственности между компонентами уровня представления. ViewModel-

классы инкапсулируют состояние и логику отображения элементов, не завися от конкретной реализации интерфейса, в то время как WinUIWindow отвечает за обработку пользовательских событий и связывание данных. Такая архитектура обеспечивает независимость бизнес-логики от UI и упрощает последующее сопровождение и расширение приложения.

3 Реализация программного обеспечения

Разработка пользовательского интерфейса проводилась на языке программирования C# с использованием платформы .NET 8 и WinUI 3, основанной на декларативном языке разметки XAML, в среде программирования Visual Studio 2026. Пользовательский интерфейс предназначен для визуального конструирования, редактирования и управления рецептами KubeJS для модифицированной сборки Minecraft.

Для отображения основных вариантов использования были составлены диаграммы деятельности. Диаграммы деятельности представляют собой графическое моделирование бизнес-процессов и алгоритмов функционирования системы, позволяя визуализировать последовательность выполнения действий, условия переходов между состояниями, параллельные потоки выполнения и зоны ответственности компонентов. Они представляют из себя блок-схемы, адаптированные для объектно-ориентированного анализа и проектирования.

Для моделирования взаимодействия компонентов во времени были составлены диаграммы последовательности, которые фокусируются на обмене сообщениями между объектами в хронологическом порядке. Данный тип диаграмм является ключевым инструментом для проектирования архитектуры уровня взаимодействия, позволяя выявить зависимости между компонентами, определить необходимые интерфейсы и методы, а также спроектировать механизмы асинхронной обработки данных.

Применение данных диаграмм позволило выявить потенциальные архитектурные проблемы до начала программирования, определить структуру взаимодействия компонентов, минимизировать связанность модулей и обеспечить соответствие разрабатываемой системы принципам SOLID и паттерну MVVM.

На рисунке 16 представлена диаграмма деятельности создания рецепта.

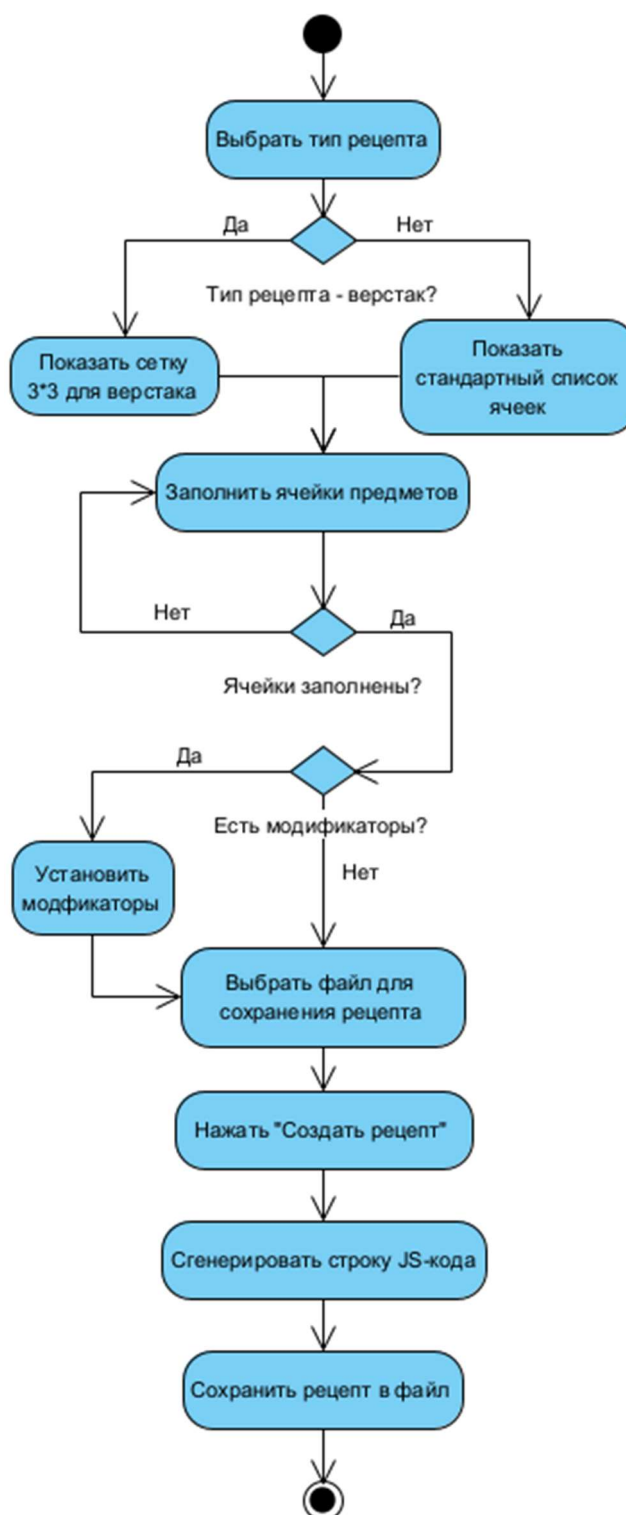


Рисунок 16 – Диаграмма деятельности создания рецепта

Диаграмма описывает бизнес-процесс создания нового игрового рецепта. Процесс создания начинается с выбора типа рецепта, после чего система определяет, требуется ли сетка 3*3 (для рецептов верстака) или стандартный список ячеек. Пользователь заполняет ячейки предметами, при необходимости

устанавливает модификаторы (нагрев, время обработки и т.д.), выбирает файл для сохранения и инициирует генерацию. Система генерирует JavaScript-код для KubeJS и сохраняет его в указанный файл.

На рисунке 17 представлена диаграмма деятельности работы с существующим списком рецептов.

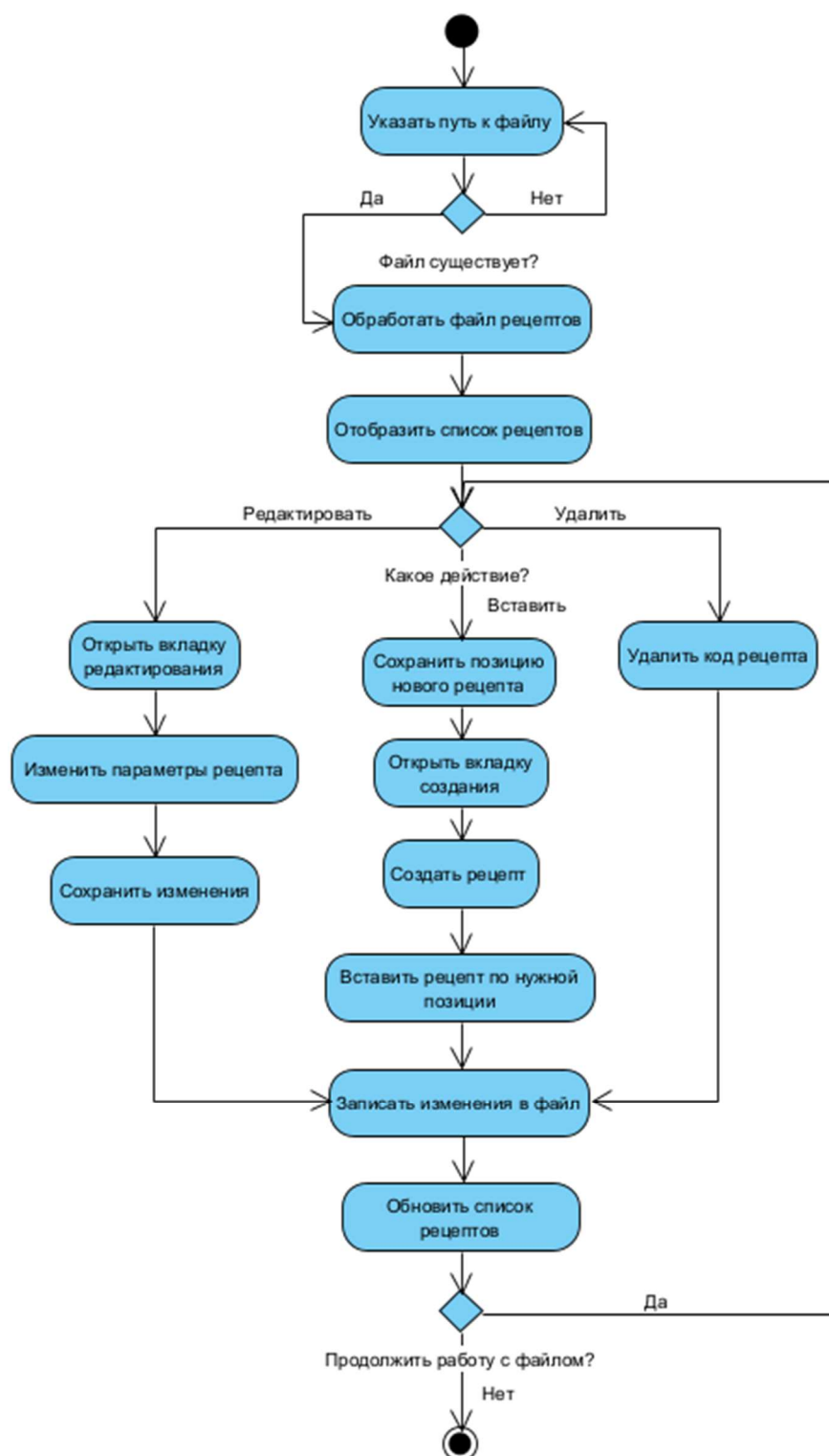


Рисунок 17 – Диаграмма деятельности работы с файлом

Диаграмма моделирует процесс управления существующими файлами рецептов. После указания пути к файлу система проверяет его существование и извлекает список рецептов. Пользователю предоставляется три варианта действий: редактирование существующего рецепта, вставка нового рецепта в указанную позицию или удаление рецепта. Все изменения записываются в файл с последующим обновлением списка.

На рисунке 18 представлена диаграмма последовательности загрузки изображений.

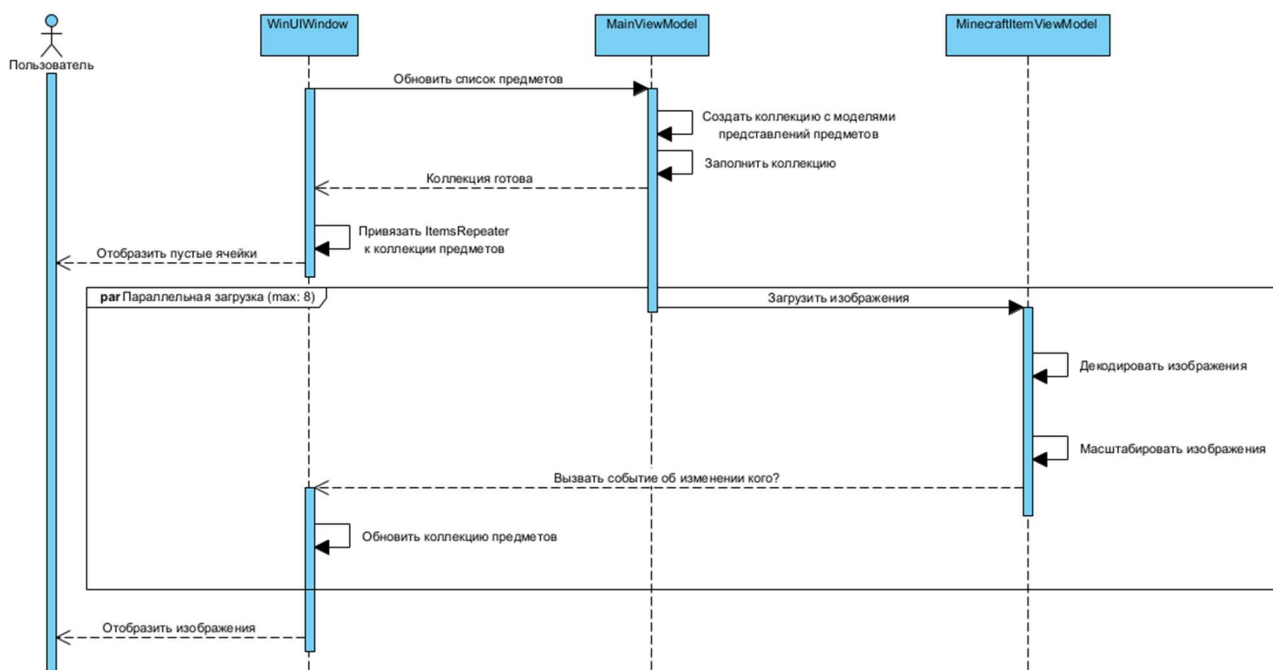


Рисунок 18 – Диаграмма последовательности загрузки изображений

Диаграмма иллюстрирует асинхронный процесс загрузки изображений предметов. Пользователь инициирует обновление списка, после чего WinUIWindow передаёт управление MainViewModel. ViewModel создаёт коллекцию MinecraftItemViewModel для каждого предмета и заполняет её данными. ItemsRepeater привязывается к коллекции, отображая пустые ячейки. В параллельном блоке с ограничением в 8 одновременных потоков каждый MinecraftItemViewModel декодирует и масштабирует своё изображение. По завершении загрузки вызывается событие PropertyChanged, уведомляющее

WinUIWindow об обновлении коллекции, что приводит к отображению иконок в интерфейсе.

На рисунке 19 представлена диаграмма последовательности генерации кода.

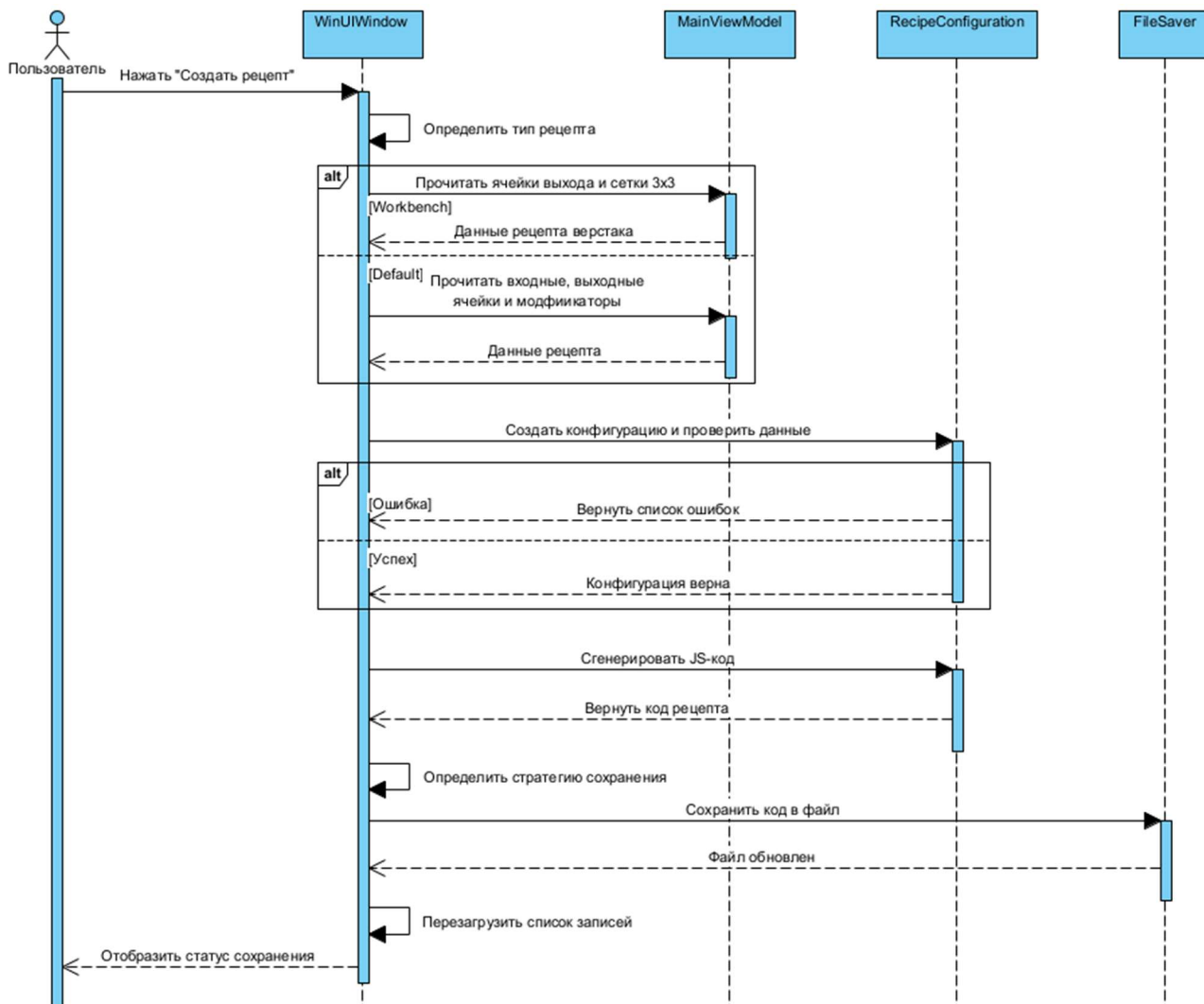


Рисунок 19 – Диаграмма последовательности генерации кода

Диаграмма демонстрирует взаимодействие компонентов при создании рецепта. Пользователь нажимает кнопку «Создать рецепт», WinUIWindow определяет тип рецепта и выбирает стратегию чтения данных: для Workbench считываются ячейки сетки 3×3, для остальных типов – входные/выходные ячейки и модификаторы. MainViewModel передаёт данные в RecipeConfiguration для валидации. В случае ошибок возвращается список, при успехе – конфигурация подтверждается. Затем генерируется JS-код, WinUIWindow

определяет стратегию сохранения (новый файл, вставка или перезапись) и через FileSaver записывает код в файл. После обновления файла список рецептов перезагружается, и пользователю отображается статус операции.

3.1 Реализация интерфейса/уровня представлений

Для отображения основных элементов были созданы файлы WinUIWindow.xaml и WinUIWindow.xaml.cs, код которых приведен в приложении Б. Первый файл содержит разметку всего интерфейса, то что непосредственно видит пользователь, в то время как второй отвечает за инициализацию данных, обработчики нажатий и событий, манипуляцию элементов управления интерфейса.

В файле разметки реализованы сетка предметов, строка поиска предметов и 3 вкладки для создания, удаления и открытия рецептов из файла.

Сетка предметов – это элемент ItemsRepeater, который служит для отображения коллекции данных. Он отображает связанную с ним коллекцию предметов FilteredItems, которая содержит список моделей представлений предметов (MinecraftItemViewModel), предметы в коллекции фильтруются в зависимости от значения в строке поиска. Визуально каждая ячейка представлена элементом Border внутри которого находится Image размером 64×64. При наведении на ячейку отображается всплывающая подсказка с названием предмета, идентификатором и названием модификации. Вся сетка предметов обернута в прокручиваемый элемент ScrollViewer, который обеспечивает вертикальную прокрутку списка, если количество предметов превышает высоту панели.

Поле поиска предметов представляет собой TextBox, с привязанным к нему событием, которое запускает таймер при каждом изменении текста. Сделано это для того, чтобы уменьшить количество ненужных вычислений фильтрации предметов при вводе. Во правой части строки находится кнопка «Clear», которая при нажатии сбрасывает текст поиска, очищает подсказки и возвращает полный список предметов.

Вкладки реализованы через элемент `TabView`, который позволяет отобразить набор вкладок и их содержимое. Он полезен для отображения нескольких страниц с возможностью переупорядочивания, открытия или закрытия новых вкладок.

Во вкладке создания тип рецепта выбирается через `ComboBox` – выпадающий список, который связан со всеми доступными типами рецептов. Ячейки для входных и выходных предметов рецепта реализованы через `ItemsRepeater`. Кнопками «+» и «-» можно добавлять и удалять ячейки (если это позволяет тип рецепта). Модификаторы рецептов появляются динамически только для определенных типов рецептов, реализованы через `RadioButton`, `CheckBox`, `NumberBox`. Также есть поле для выбора файла и кнопка для создания рецепта.

Вкладка удалить служит для создания `event.remove` команд, которые удаляют стандартные рецепты игры. Содержит те же ячейки для выбора предметов и выпадающие списки для указания типа и модификации рецептов. Также есть текстовое поле для ввода `Id` рецепта, если нужно удалить конкретный рецепт и поле для выбора файла.

Во вкладке файл также есть поле для ввода файла и набор кнопок. «Обзор» открывает системный диалог выбора файла, «Открыть» загружает файл по пути, в текстовом поле, «Обновить» перечитывает файл с диска, если он был изменен внешне. При загрузке файла список рецептов реализован через `ListView`, поскольку отображение рецептов не требует гибкой настройки. Каждый рецепт в списке представляется строкой с кнопками действий. Используется `Grid` с 4 колонками: кнопка редактирования рецепта, удаления рецепта и вставки нового рецепта до и после выбранного.

Файл `WinUIWindow.xaml.cs` является связующим звеном между визуальной частью и уровнем модели представлений. Здесь создается экземпляр класса `MainViewModel` для сохранения состояния интерфейса. В этом файле реализованы необходимые для работы интерфейса поля: `FilteredItems` – отфильтрованная коллекция предметов для отображения в сетке;

`_searchDebounceTimer` – таймер задержки перед выполнением фильтрации при поиске; `_pendingSlotSelection` – выбранная ячейка для назначения предмета; `_gamePath` и `_saveDir` – пути к игре и папке сохранения рецептов.

Обработчик `SearchBox_TextChanged` перезапускает таймер задержки, а за логику поиска предметов отвечает метод `PerformSearch`, который ищет совпадения с предметами по `Id`, названию предмета или модификации и обновляет коллекцию `FilteredItems`.

Во вкладке «Создать» обработчик `BlockComboBox_SelectionChanged` реагирует на изменение типа рецепта в выпадающем списке. Этот метод обновляет свойство `SelectedBlock` в `ViewModel`, вызывает метод `ApplyRecipeType` для динамической перестройки интерфейса под выбранный тип рецепта и сбрасывает настройки модификаторов на значения по умолчанию. При нажатии на любую ячейку для предмета метод `Slot_PointerPressed` устанавливает значение поля `_pendingSlotSelection`, затем при выборе предмета из сетки обработчик `ItemBorder_PointerPressed` извлекает `MinecraftItemViewModel` из `Tag` элемента и вызывает метод `AssignItemToSlot` выполняет его назначение в указанную ячейку. Метод `Slot_RightTapped` очищает данные ячейки по нажатию правой кнопки мыши. Управление динамическими ячейками входов и выходов реализовано через простые методы-делегаты, такие как `AddInput_Click` и `RemoveOutput_Click`, которые перенаправляют вызовы соответствующим методам в `ViewModel`.

Главный метод создания рецептов `GenerateRecipeButton_Click` определяет тип выбранного рецепта и вызывает соответствующий метод для формирования рецепта: `BuildWorkbenchRecipeCode` для рецептов верстака, `BuildCreateRecipeCode` для рецептов `Create` или `BuildMinecraftRecipeCode` для стандартных рецептов `Minecraft`. Каждый из этих методов собирает данные из интерфейса и формирует строку JavaScript-кода.

Вспомогательные методы построения кода реализуют сложную логику преобразования данных интерфейса в синтаксис `KubeJS`. Метод `BuildWorkbenchRecipeCode` анализирует данные в сетке 3×3, создаёт карту

соответствия предметов буквенным обозначениям, формирует строки паттерна рецепта и создаёт объект конфигурации для генерации финального кода. Метод `BuildMinecraftRecipeCode` работает со стандартными рецептами, выполняя проверку данных через специализированный генератор и учитывая специфичные параметры, такие как опыт для рецепта копчения. Наиболее сложным является `BuildCreateRecipeCode`, который обрабатывает многообразные рецепты мода `Create`, включая модификаторы нагрева, времени обработки и сохранения предметов в руке.

Логика вкладки «Удалить» реализована в методе `RemoveRecipeButton_Click`. Он собирает данные из полей фильтрации, создаёт объект фильтра `RecipeFilter` и конфигурацию удаления `RecipeRemovalConfig`, выполняет проверку и генерирует код для удаления рецептов. Результаты генерации отображаются в отдельном поле, а сохранение осуществляется через стратегию `FileSaverToEnd`.

Вкладка «Файл» предоставляет возможность работы с существующими рецептами. Метод `FileTab_Browse_Click` открывает системный диалог выбора файла, используя специальную процедуру инициализации `WinRT.Interop.InitializeWithWindow`, необходимую для корректной работы диалогов в WinUI 3. Полный путь к файлу отображается в поле `FileTabPathBox`. Метод `FileTab_OpenFile_Click` берёт путь из `FileTabPathBox`, проверяет на пустоту и вызывает `LoadRecipeFile`. Метод `FileTab_Reload_Click` берёт путь из `ViewModel.RecipeFile.FilePath` и обновляет вывод рецептов если файл был изменен из вне. Метод `LoadRecipeFile` загружает и обрабатывает выбранный файл с помощью класса `JsRecipeParser`, заполняя коллекцию рецептов в `ViewModel`. Метод редактирования `FileTab_EditEntry_Click` определяет тип выбранного рецепта, находит соответствующие предметы и заполняет интерфейс вкладки «Создать» данными для модификации, переключая приложение в режим редактирования. Методы вставки `FileTab_InsertBefore_Click` и `FileTab_InsertAfter_Click` сохраняют позицию в файле для последующей вставки нового рецепта, а `FileTab_DeleteEntry_Click`

удаляет код рецепта из файла, обрабатывая переносы строк для сохранения форматирования.

Все обновления пользовательского интерфейса из асинхронных операций выполняются через `DispatcherQueue.TryEnqueue` для соблюдения потоковой безопасности. Загрузка иконок предметов в методе `UpdateItems` использует семафор `SemaphoreSlim` для ограничения параллелизма, что предотвращает перегрузку системы при обработке больших объемов данных. Для обеспечения безопасности данных при редактировании используется паттерн записи во временный файл с последующей заменой оригинала.

3.2 Реализация архитектурной части приложения/уровня модели представлений

В соответствии с паттерном MVVM был разработан файл `MainViewModel.cs`, который представляет собой уровень модели представлений. Этот файл хранит данные о состоянии интерфейса и подготавливает информацию для отображения. Он содержит коллекции данных, выбранные пользователем предметы и модификаторы и определения классов.

Класс `MainViewModel` является корневым управляющим компонентом, хранящим данные о состоянии всего окна приложения. Класс управляет несколькими наблюдаемыми коллекциями, которые привязываются к элементам интерфейса.

`Items` хранит полный список всех доступных предметов, загруженных из игры. Эта коллекция служит источником данных для сетки предметов.

`Recipes` содержит список идентификаторов рецептов, используемый для отображения в соответствующей панели.

`RecipeBlocks` – это статический список всех поддерживаемых типов рецептов, получаемый из дескриптора `RecipeTypeDescriptor`. Используется для заполнения выпадающего списка выбора типа рецепта.

`InputSlots` / `OutputSlots` – это динамические коллекции, представляющие ячейки входов и выходов текущего рецепта. Их содержимое автоматически пересоздаётся при смене типа рецепта.

`WorkbenchSlots` – массив из 9 элементов, инициализированный при создании объекта. Представляет фиксированную сетку 3×3 из ячеек для точной настройки рецепта верстака.

`RemoveModOptions` список уникальных названий модификаций, извлечённых из загруженных предметов, для использования в фильтрах вкладки «Удалить».

Для управления динамическим интерфейсом конструктора в классе определены свойства, контролирующие видимость и доступность элементов.

`SelectedBlock` хранит выбранный пользователем тип рецепта.

`ShowWorkbenchGrid` / `ShowInputSlots`: свойства, которые переключают отображение между сеткой рецепта 3×3 и списком ячеек в зависимости от типа рецепта.

`ShowInputButtons` / `ShowOutputButtons`: управляют видимостью кнопок добавления/удаления ячеек.

`CanAddInput` / `CanRemoveInput` / `CanAddOutput` / `CanRemoveOutput`: логические флаги, определяющие доступность кнопок управления ячейками на основе ограничений текущего типа рецепта.

Свойства `ShowModifierHeated`, `ShowModifierProcessingTime`, `ShowModifierKeepHeldItem`, `ShowModifierExperience` отвечают за отображение полей ввода модификаторов для рецептов (нагрев, время обработки, опыт и т.д.).

Значения модификаторов: `SelectedHeat`, `ProcessingTime`, `KeepHeldItem`, `Experience` хранят пользовательский ввод для генерации кода.

Ключевые методы управления.

Метод `ApplyRecipeType` перестраивает интерфейс под выбранный тип рецепта. Он запрашивает ограничения рецепта через `RecipeTypeDescriptor.GetConstraints`, очищает старые ячейки, создаёт новые в

нужном количестве, настраивает видимость модификаторов и сбрасывает значения по умолчанию.

`AddInputSlot` / `RemoveInputSlot` / `AddOutputSlot` / `RemoveOutputSlot`: методы для динамического управления количеством ячеек, которые учитывают ограничения типа рецепта и обновляют состояние кнопок.

`RefreshSlotButtons`: вспомогательный метод, вычисляющий доступность кнопок «+» и «-» на основе текущих ограничений и количества ячеек.

`ClearWorkbenchSlots`: очищает все 9 ячеек сетки рецепта.

`RefreshRemoveModOptions`: обновляет список модификаций для вкладки «Удалить» на основе загруженных предметов.

Класс предоставляет несколько свойств для поддержки редактирования существующих рецептов.

`EditingEntry` хранит ссылку на редактируемый рецепт. Его изменение автоматически обновляет вычисляемые свойства `IsEditMode`, `CancelEditVisibility`, `FileNameVisibility` и `CreateButtonLabel`, которые переключают интерфейс между режимами создания и редактирования.

`RemovalOutputSlot` / `RemovalInputSlot` – это специализированные ячейки для вкладки «Удалить», позволяющие выбирать предметы через тот же механизм, что и в конструкторе.

`RemoveTypeOptions` – это список типов рецептов для фильтрации при удалении.

`RecipeFile` – это вложенный объект, управляющий состоянием открытого файла скриптов.

Класс `RecipeSlotViewModel` представляет одну ячейку входа или выхода рецепта и инкапсулирует всю логику ее отображения.

`SlotIndex`, `ShowChance`, `ShowAmount` – неизменяемые параметры, определяющие поведение ячейки.

Свойство `Item` хранит ссылку на выбранный предмет (`MinecraftItemViewModel`). При его изменении автоматически обновляются вычисляемые свойства `ItemId`, `PlaceholderVisibility`, `ImageVisibility`.

Параметры рецепта: Amount (количество) и Chance (шанс выпадения) с двусторонней привязкой для ввода пользователем.

Управление видимостью: свойства ChanceVisibility, AmountVisibility динамически показывают или скрывают соответствующие поля ввода в зависимости от типа рецепта.

Класс `MinecraftItemViewModel` преобразует доменную модель `MinecraftItem` из библиотеки `LogExtractorLibrary` в формат, удобный для отображения в интерфейсе:

Свойства `Id`, `LocalizedName`, `ModName`, `Type`, `TagsDisplay` копируются из исходной модели и предоставляют информацию для отображения и поиска.

Асинхронная загрузка изображений: свойство `Image` типа `ImageSource` и метод `LoadImageAsync` отвечают за декодирование, масштабирование и отображение иконки предмета. Метод использует `BitmapDecoder` для декодирования изображения из байтового массива. Масштабирование выполняется алгоритмом `Nearest Neighbor` в фоновом потоке, что сохраняет чёткость пиксельной графики. Результат применяется в UI-потоке через присваивание свойству `Image`, что автоматически обновляет привязанный элемент управления.

Класс `RecipeEntryViewModel` представляет обработанный рецепт из существующего файла для отображения в списке и редактирования:

`Config` – это ссылка на исходную конфигурацию (`BaseRecipeConfiguration`), содержащую все данные рецепта.

`DisplayName` – вычисляемое свойство, формирующее описание рецепта с использованием сопоставления с образцом (`pattern matching`) для разных типов конфигураций.

`CanDelete` / `CanInsert` – флаги, определяющие возможность удаления или вставки рядом с этим рецептом на основе наличия метаданных о позиции в исходном файле (`HasSourcePosition`).

Вспомогательные методы `DescribeFilters` и `DescribeCreateRecipe` форматируют информацию о рецептах удаления и Create-рецептах для отображения.

Класс `RecipeFileViewModel` управляет состоянием открытого файла рецептов.

`FilePath` – путь к файлу на диске с двусторонней привязкой для отображения и редактирования.

`StatusText` – текст статуса для информирования пользователя о результатах операций (загрузка, сохранение, ошибки).

`PendingInsertPosition` – позиция в файле, куда будет вставлен новый рецепт при использовании функции вставки.

`Entries` – коллекция `RecipeEntryViewModel`, представляющая все рецепты, найденные в файле.

4 Тестирование графического интерфейса добавления рецептов

4.1 Тестирование модуля добавления рецепта методом черного ящика

4.1.1 Выбор методики тестирования

Тестирование методом черного ящика – это вид тестирования приложения вне зависимости от его внутренней структуры, архитектуры и реализации [15]. На вход системы подаются данные, а выходные данные анализируются и сравниваются с ожидаемым результатом. Метод «черного ящика» включает в себя набор методик, которые позволяют максимально эффективно проверить работоспособность системы [16].

Объектом тестирования является графический интерфейс добавления рецепта. Функционал предполагает ввод и проверку следующих параметров: тип рецепта, предметы, количество предметов и опциональные модификаторы. Ключевой особенностью системы является наличие жестких зависимостей между параметрами (например, доступные модификаторы рецепта зависят от типа рецепта) и числовых диапазонов с четкими границами (1 – 64 для предметов, 0 – 100 для получаемого опыта).

Для проведения тестирования графического интерфейса была выбрана методика таблицы решений.

4.1.2 Функциональная диаграмма добавления рецепта

Выделим причины и следствия событий для добавления рецепта (таблицы 2 – 3).

Таблица 2 – Причины событий

Номер	Причина
П1	Тип рецепта выбран
П2	Выбраны входные предметы
П3	Выбраны выходные предметы
П4	Корректное количество входных предметов
П5	Корректное количество выходных предметов

Продолжение таблицы 2

Номер	Причина
П6	Корректные модификаторы рецепта
П7	Файл выбран

Таблица 3 – Следствия событий

Номер	Следствие
С1	Ошибка: Тип рецепта не выбран
С2	Ошибка: Входные предметы не выбраны
С3	Ошибка: Выходные предметы не выбраны
С4	Ошибка: Неверное количество входных предметов
С5	Ошибка: Неверное количество выходных предметов
С6	Ошибка: Неверные модификаторы рецепта
С7	Ошибка: Файл не выбран
С8	Рецепт успешно добавлен

Для этих событий была составлена функциональная диаграмма (рисунок 20) [17].

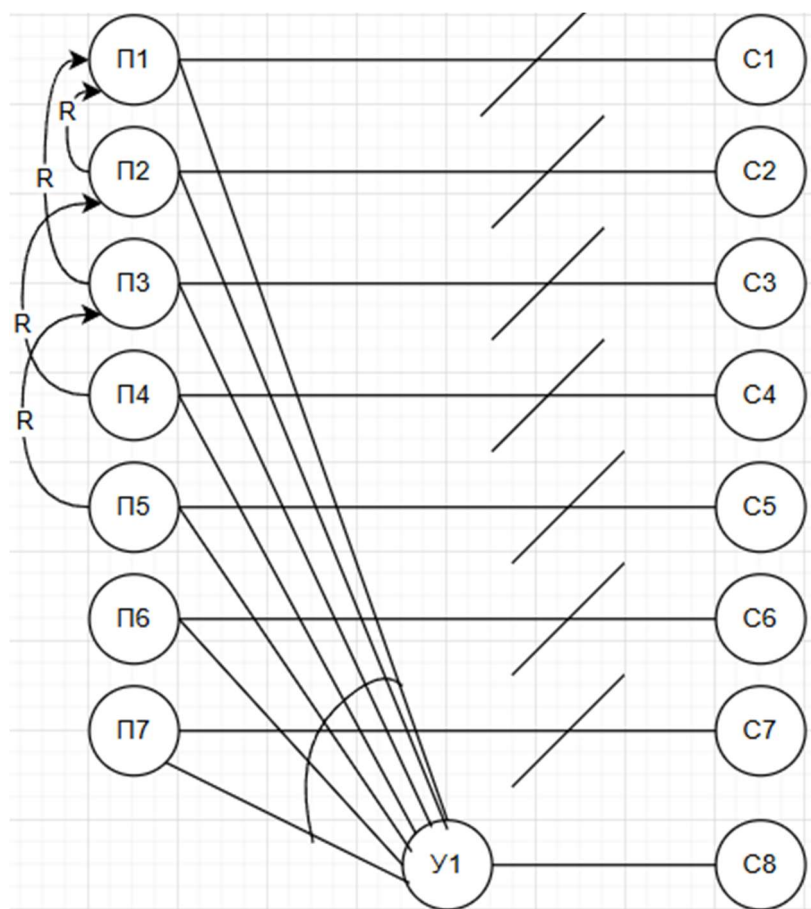


Рисунок 20 – Функциональная диаграмма добавления рецепта

4.1.3 Таблица решений для добавления рецепта

Для ранее выделенных событий, на основе составленной диаграммы, была составлена таблица решений (таблица 4) и проведены соответствующие тесты (таблица 5) [16].

Таблица 4 – Таблица решений

	1	2	3	4	5	6	7	8	9	10	11	12
П1	0	1	1	1	1	1	1	1	1	1	1	1
П2	-	0	1	1	0	0	1	1	1	1	1	1
П3	-	0	0	0	1	1	1	1	1	1	1	1
П4	-	-	0	1	-	-	0	0	1	1	1	1
П5	-	-	-	-	0	1	0	1	0	1	1	1
П6	-	0	0	0	0	0	0	0	0	0	1	1
П7	-	0	0	0	0	0	0	0	0	0	0	1
С1	1	0	0	0	0	0	0	0	0	0	0	0
С2	-	1	0	0	1	1	0	0	0	0	0	0
С3	-	1	1	1	0	0	0	0	0	0	0	0
С4	-	-	1	0	-	-	1	1	0	0	0	0
С5	-	-	-	-	1	0	1	0	1	0	0	0
С6	-	1	1	1	1	1	1	1	1	1	0	0
С7	-	1	1	1	1	1	1	1	1	1	1	0
С8	0	0	0	0	0	0	0	0	0	0	0	1

Таблица 5 – Тест-кейсы

№	Входные данные	Выходные данные
1	Тип: - Входные предметы: - Выходные предметы: - Модификаторы: - Файл: -	Ошибка: Тип рецепта не выбран.
2	Тип: smelting Входные предметы: - Выходные предметы: - Модификаторы: - Файл: -	Ошибка: Входные и выходные предметы не выбраны.
3	Тип: smelting Входные предметы: Iron Ore, 100 Выходные предметы: - Модификаторы: - Файл: -	Ошибка: Выходные предметы не выбраны, неверное количество входных предметов.

Продолжение таблицы 5

№	Входные данные	Выходные данные
4	Тип: smelting Входные предметы: Iron Ore, 32 Выходные предметы: - Модификаторы: - Файл: -	Ошибка: Выходные предметы не выбраны.
5	Тип: smelting Входные предметы: - Выходные предметы: Iron Ingot, 100 Модификаторы: - Файл: -	Ошибка: Входные предметы не выбраны, неверное количество выходных предметов.
6	Тип: smelting Входные предметы: - Выходные предметы: Iron Ingot, 32 Модификаторы: - Файл: -	Ошибка: Входные предметы не выбраны.
7	Тип: Stonecutting Входные предметы: Stone, 2000 Выходные предметы: Stone Bricks, 400 Модификаторы: - Файл: -	Ошибка: Неверное количество входных и выходных предметов.
8	Тип: Stonecutting Входные предметы: Stone, 2000 Выходные предметы: Stone Bricks, 32 Модификаторы: - Файл: -	Ошибка: Неверное количество входных предметов.
9	Тип: Stonecutting Входные предметы: Stone, 32 Выходные предметы: Stone Bricks, 400 Модификаторы: - Файл: -	Ошибка: Неверное количество выходных предметов.
10	Тип: Compacting Входные предметы: Coal block, 1 Выходные предметы: Diamond, 1 Модификаторы: - Файл: -	Ошибка: Модификатор не указан.
11	Тип: Compacting Входные предметы: Coal block, 1 Выходные предметы: Diamond, 1 Модификаторы: Heated Файл: -	Ошибка: Файл не выбран.
12	Тип: Blasting Входные предметы: Coal block, 1 Выходные предметы: Diamond, 1 Модификаторы: Heated Файл: CustomRecipes.js	Рецепт успешно добавлен.

Все разработанные тест-кейсы были полностью выполнены и показали полное совпадение с ожидаемыми результатами.

4.2 Тестирование графического интерфейса для отображения данных

4.2.1 Выбор средств тестирования графического интерфейса

Объектом тестирования является графический интерфейс, для создания рецептов. Интерфейс реализован на платформе WinUI 3 и предполагает ввод и проверку следующих параметров: тип рецепта, предметы, количество предметов и опциональные модификаторы.

Интерфейс добавления рецепта представлен на рисунке 21.

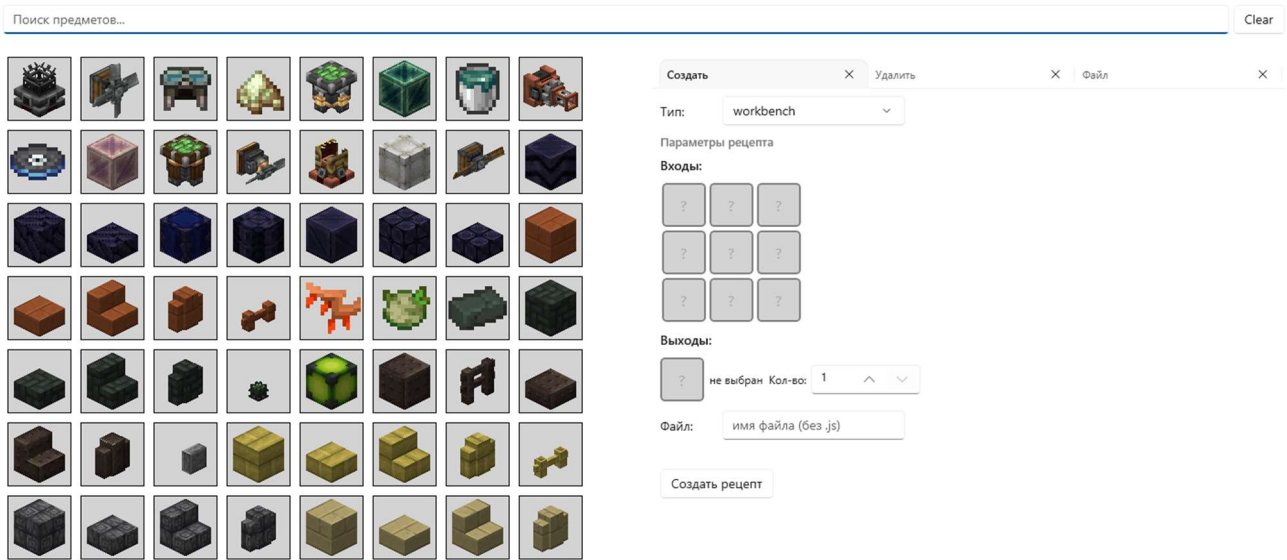


Рисунок 21 – Интерфейс добавления рецепта

Для оценки качества разработанного интерфейса был проведен анализ существующих методов тестирования UX, приведенный в таблице 6.

Таблица 6 – Анализ методов тестирования

Метод	Применимость к проекту	Трудоемкость	Стоимость
Тестирование удобства использования с пользователями	Низкая (сложно набрать целевую аудиторию)	Высокая	Высокая
А/В тестирование	Не применимо (одна версия интерфейса)	Средняя	Средняя

Продолжение таблицы 6

Метод	Применимость к проекту	Трудоемкость	Стоимость
Автоматизированное UI-тестирование	Средняя (WinUI 3 имеет ограниченную поддержку)	Высокая	Высокая
Эвристическая оценка [18]	Высокая (быстрое выявление проблем)	Низкая	Низкая
Ручное функциональное тестирование	Высокая (полное покрытие, гибкость)	Средняя	Низкая

На основании проведенного анализа выбран метод ручного функционального тестирования с элементами эвристической оценки [18, 19]. Данный выбор обоснован следующими факторами:

- a) стадия разработки;
- b) характер приложения;
- c) отсутствие необходимости в дорогостоящих инструментах.

Приложение находится на стадии прототипа, поэтому интерфейс подвержен частым изменениям. Ручное тестирование обеспечивает гибкость и не требует поддержки автоматических тестов. Функциональная корректность генерации валидных KubeJS-скриптов в данном случае приоритетнее визуальной эстетики. Метод не требует дорогостоящих инструментов, инфраструктуры для автоматизации или набора фокус-группы пользователей.

Процесс тестирования включает:

- a) разработку тест-кейсов на основе таблицы решений;
- b) выполнение тест-кейсов для всех сценариев использования;
- c) оценку интерфейса по эвристикам Нильсена [18];
- d) устранение дефектов и реализацию предложений по улучшению.

Тест-кейсы разрабатываются для всех сценариев использования, включая позитивные и негативные сценарии. Оценка интерфейса проводится по эвристикам Нильсена [18], включая видимость состояния системы, соответствие реальному миру и контроль пользователя. Такой подход обеспечивает баланс между полнотой проверки и эффективностью использования ресурсов, позволяя выявить критические проблемы до выпуска приложения.

4.2.2 Результаты тестирования графического интерфейса

Для оценки качества пользовательского интерфейса и эффективности взаимодействия с ним был проведен набор ручных функциональных тестов, охватывающих основные сценарии использования приложения: создание рецептов различных типов, обработку ошибок валидации, проверку граничных значений и работу с режимом редактирования.

В качестве количественных показателей эффективности взаимодействия были выбраны три метрики: время выполнения задачи, количество кликов мышью и количество нажатий клавиш. Шкалы оценки метрик приведены в таблицах 7 – 9.

Таблица 7 – Шкала оценки времени

Время выполнения	Оценка
< 5 сек	Отличное
5-15 сек	Хорошее
15-30 сек	Приемлемое
30-60 сек	Удовлетворительное
> 60 сек	Критическое

Таблица 8 – Шкала оценки нажатий мышки

Количество нажатий	Оценка
1-3	Минимальное
4-7	Оптимальное
8-12	Приемлемое
13-20	Избыточное
> 20	Критическое

Таблица 9 – Шкала оценки нажатий клавиш

Количество нажатий	Оценка
<10	Минимальное
10-30	Оптимальное
31-50	Приемлемое
50-100	Избыточное

Продолжение таблицы 9

Количество нажатий	Оценка
> 100	Критическое

В таблице 10 представлен набор проведенных ручных тестов, их результаты и измеренные метрики.

Таблица 10 – Ручные тесты

№	Наименование	Действия	Результат	Метрики
1	Создание стандартного рецепта плавки (smelting)	1) нажатие по combobox выбора типа; 2) выбор smelting из списка; 3) нажатие по ячейке «вход»; 4) ввод в поиске «iron_ore»; 5) нажатие по нужному предмету; 6) нажатие по ячейке «выход»; 7) ввод в поиске «iron_ingot»; 8) нажатие по нужному предмету; 9) нажатие по полю «имя файла»; 10) ввод названия файла; 11) нажатие кнопки «Создать рецепт».	Код сгенерирован и сохранен, статус отображен пользователю	Время – 9,5 сек Нажатия мышки – 8 Нажатия клавиш – 38
2	Обработка ошибки при отсутствии выбора типа рецепта	1) пропуск выбора типа рецепта; 2) нажатие по ячейке «вход»; 3) выбор предмета из списка; 4) нажатие по ячейке «выход»; 5) выбор предмета; 6) ввод имени файла; нажатие кнопки «Создать рецепт».	Система вывела сообщение о необходимости выбора типа рецепта, создание файла заблокировано	Время – 5,5 сек Нажатия мышки – 8 Нажатия клавиш – 22

Продолжение таблицы 10

№	Наименование	Действия	Результат	Метрики
3	Валидация пустой входной ячейки	1) выбор типа рецепта blasting; 2) пропуск заполнения входной ячейки; 3) заполнение выходной ячейки предметом gold_ingot; 4) ввод имени файла; 5) нажатие кнопки «Создать рецепт».	Система подсветила пустую ячейку, вывела сообщение об ошибке, генерация кода не произведена	Время – 6,5 сек Нажатия мышки – 7 Нажатия клавиш – 29
4	Проверка валидации минимального количества предметов	1) выбор типа рецепта smoking; 2) заполнение входной и выходной ячеек; 3) ввод значения 0 в поле количества; 4) ввод имени файла; 5) нажатие кнопки «Создать рецепт».	Система заблокировала ввод нулевого значения, вывела сообщение об ошибке, генерация кода не произведена	Время – 9,7 сек Нажатия мышки – 11 Нажатия клавиш – 44
5	Проверка валидации максимального количества предметов	1) выбор типа рецепта smelting; 2) заполнение входной и выходной ячеек; 3) ввод значения 100 в поле количества; 4) ввод имени файла; 5) нажатие кнопки «Создать рецепт».	Система заблокировала ввод неверного значения, вывела сообщение об ошибке, генерация кода не произведена	Время – 9,8 сек Нажатия мышки – 11 Нажатия клавиш – 46
6	Создание рецепта верстака (workbench)	1) выбор типа рецепта workbench; 2) последовательное заполнение пяти ячеек сетки крафта; 3) выбор выходного предмета wooden_pickaxe; 4) ввод имени файла test_craft; 5) нажатие кнопки «Создать рецепт».	Сгенерирован корректный код с шаблоном рецепта верстака, файл успешно сохранен	Время – 15,5 сек Нажатия мышки – 17 Нажатия клавиш – 34

Окончание таблицы 10

№	Наименование	Действия	Результат	Метрики
7	Создание рецепта с модификаторами	1) выбор типа рецепта mixing; 2) заполнение двух входных ячеек; 3) выбор выходного предмета dough; 4) активация модификатора нагрева; 5) ввод времени обработки 50; 6) ввод имени файла; 7) нажатие кнопки «Создать рецепт».	Код сгенерирован с учетом примененных модификаторов, файл сохранен корректно	Время – 13,8 сек Нажатия мышки – 15 Нажатия клавиш – 58
8	Проверка граничного значения минимального количества	1) заполнение стандартного рецепта; 2) ввод значения 1 в поле количества; 3) сохранение файла.	Система приняла минимальное допустимое значение, файл успешно сохранен	Время – 9,8 сек Нажатия мышки – 11 Нажатия клавиш – 44
9	Проверка граничного значения максимального количества	1) заполнение стандартного рецепта; 2) ввод значения 64 в поле количества; 3) сохранение файла.	Система приняла максимальное допустимое значение, файл успешно сохранен	Время – 9,8 сек Нажатия мышки – 11 Нажатия клавиш – 45
10	Проверка функциональности отмены редактирования	1) переход на вкладку управления файлами; 2) активация режима редактирования существующего рецепта; 3) изменение параметра количества; 4) нажатие кнопки «Отмена»; 5) проверка сброса формы.	Интерфейс вернулся в исходное состояние, изменения не применены, файл не модифицирован	Время – 3,7 сек Нажатия мышки – 5 Нажатия клавиш – 2

Проведенное ручное тестирование подтвердило работоспособность базового функционала приложения. Интерфейс не требует излишних действий от пользователя, демонстрирует высокую скорость отклика и корректную обработку основных пользовательских сценариев. Выявленные области для оптимизации не являются критическими и могут быть улучшены в последующих итерациях разработки без изменения архитектурных решений.

4.3 Анализ качества кода

4.3.1 Количественные метрики кода (SLOC)

Метрики кода являются совокупностью оценок ПО, служащих рекомендательными характеристиками и представляющих более полную информацию о программном коде. Применение метрик позволяет руководителям проектов и предприятий изучить сложность разработанного или разрабатываемого проекта, оценить объем работ, стилистику разрабатываемой программы и усилия, потраченные каждым разработчиком для реализации того или иного решения.

Количество строк кода (SLOC) – это метрика программного обеспечения, используемая для измерения его объёма с помощью подсчёта количества строк в тексте исходного кода [20]. Как правило, этот показатель используется для прогноза трудозатрат на разработку конкретной программы на конкретном языке программирования, либо для оценки производительности труда уже после того, как программа написана.

Была рассчитана метрика SLOC по методу UpdateItems, который обновляет и отображает коллекцию предметов в левой панели интерфейса. Код метода приведен в листинге 1.

Листинг 1 – Метод UpdateItems

```
// Метод для обновления предметов (вызывается из логики)
public void UpdateItems(List<MinecraftItem> items)
{
    this.DispatcherQueue.TryEnqueue(() =>
    {
        ViewModel.Items.Clear();
        var viewModels = new
List<MinecraftItemViewModel>(items.Count);
        foreach (var item in items)
        {
            var vm = new MinecraftItemViewModel(item);
            viewModels.Add(vm);
            ViewModel.Items.Add(vm);
        }

        ViewModel.RefreshRemoveModOptions();
    });
}
```

```

if (!string.IsNullOrEmpty(SearchBox.Text))
{
    _searchDebounceTimer.Stop();
    PerformSearch();
}
else
{
    UpdateFilteredItems(ViewModel.Items);
}

// Загружаем изображения в фоне, не блокируя UI-поток
var semaphore = new System.Threading.SemaphoreSlim(8);
_ = Task.WhenAll(viewModels.Select(async vm =>
{
    await semaphore.WaitAsync();
    try { await vm.LoadImageAsync(64); }
    finally { semaphore.Release(); }
})));
});
}

```

Логическими строками считаются все строки, которые содержат выполняемые команды или объявления. Измеряя логические строки кода, предпринимается попытка посчитать количество операторов в программе что позволяет абстрагироваться от стиля форматирования кода разработчиком.

В данном методе имеем:

- 36 физических строк кода.
- 18 логических строк кода.
- 2 строки комментариев.

В таблице 11 приведена шкала оценки размера метода по количеству логических строк.

Таблица 11 – Шкала оценки размера метода

Количество логических строк	Размер
1-10	Очень малый
11-30	Оптимальный
31-50	Приемлемый
51-100	Большой
>100	Критический

Данные показатели свидетельствуют о том, что метод выполняет четко ограниченную задачу, обладает высокой читаемостью и тестируемостью.

4.3.2 Метрика Холстеда

Метрика Холстеда – это набор показателей для количественной оценки сложности программного обеспечения [21]. Эти показатели основаны на количестве различных операторов и операндов в программе и используются для оценки усилий, необходимых для разработки и сопровождения программы. Они включают в себя такие показатели, как продолжительность программы, словарный запас, объем, сложность и трудоемкость, рассчитываемые на основе количества уникальных операторов и операндов в программе.

Расчет метрики Холстеда и определения комплексного показателя качества кода производится для ранее указанного метода обновления коллекции предметов. Операнды и операторы приведены в таблицах 12 – 13.

Таблица 12 – Операторы

№	Оператор	Количество вхождений
1	.	15
2	(	19
3	)	19
4	{	9
5	}	9
6	;	14
7	=	3
8	var	3
9	new	3
10	await	2
11	=>	2
12	<	2
13	>	2
14	public	1
15	void	1
16	foreach	1
17	in	1
18	if	1
19	else	1
20	async	1

Продолжение таблицы 12

№	Оператор	Количество вхождений
21	try	1
22	finally	1
23	!	1

Уникальные операторы (n1) = 23.

Общее количество операторов (N1) = 112.

Таблица 13 – Операнды

№	Операнд	Количество вхождений
1	ViewModel	4
2	vm	4
3	items	3
4	Items	3
5	viewModels	2
6	item	2
№	Операнд	Количество вхождений
7	semaphore	2
8	MinecraftItemViewModel	2
9	Add	2
10	UpdateItems	1
11	List	1
12	MinecraftItem	1
13	Clear	1
14	Count	1
15	RefreshRemoveModOptions	1
16	string	1
17	IsNullOrEmpty	1
18	SearchBox	1
19	Text	1
20	_searchDebounceTimer	1
21	Stop	1
22	PerformSearch	1
23	UpdateFilteredItems	1
24	System	1
25	Threading	1
26	SemaphoreSlim	1
27	8	1
28	_	1
29	Task	1
30	WhenAll	1

Продолжение таблицы 13

№	Операнд	Количество вхождений
31	Select	1
32	WaitAsync	1
33	LoadImageAsync	1
34	64	1
35	Release	1
36	this	1
37	DispatcherQueue	1
38	TryEnqueue	1

Уникальные операнды (n_2) = 38.

Общее количество операндов (N_2) = 53.

Длина программы рассчитывается по формуле:

$$N = N_1 + N_2 = 112 + 53 = 165,$$

где N_1 – общее количество операторов,

N_2 – общее количество операндов.

Словарь программы рассчитывается по формуле:

$$n = n_1 + n_2 = 23 + 38 = 61,$$

где n_1 – уникальные операторы,

n_2 – уникальные операнды.

Объем программы рассчитывается по формуле:

$$V = N * \log_2(n) = 165 * \log_2(61) \approx 980,$$

где N – длина программы,

n – словарь программы.

Сложность программы рассчитывается по формуле:

$$D = (n_1/2) * (N_2/n_2) = (23/2) * (53/38) = 16,04,$$

где n_1 – уникальные операторы,

n_2 – уникальные операнды,

N_2 – общее количество операндов.

Трудоемкость программы рассчитывается по формуле:

$$E = D * V = 16,04 * 980 \approx 15719,$$

где D – сложность программы,

V – объем программы.

В таблице 14 представлены шкалы интерпретации метрик Холстеда.

Таблица 14 – Шкалы интерпретации метрик Холстеда

Метрика	Малое значение	Среднее значение	Высокое значение	Критическое значение
Объем (V)	<1000	1000-3000	3000-10000	>10000
Сложность (D)	<5	5-15	15-30	>30
Трудоемкость (E)	<10000	10000-50000	50000-100000	>100000

Малый объем метода указывает на его компактность и высокую информационную плотность. Он не перегружен излишними данными, что снижает вероятность наличия скрытых ошибок.

Высокое значение сложности обусловлено существенным соотношением общего количества вхождений операндов к их уникальному числу ($N_2/n_2 \approx 1,39$) при значительном разнообразии операторов ($n_1 = 23$), что свидетельствует о насыщенной логике управления с активным использованием методов доступа к членам классов и асинхронных конструкций.

Среднее значение трудоемкости указывает на умеренные затраты интеллектуальных ресурсов. Метод не является тривиальным и не требует значительных усилий на отладку и сопровождение.

Данные значения характеризуют метод как умеренно сложный, однако требующий внимательного подхода к тестированию из-за обилия внешних зависимостей и асинхронных операций, которые повышают когнитивную нагрузку при отладке.

Заключение

В ходе выполнения выпускной квалификационной работы была разработана клиентская часть комплекса программных средств для автоматизации написания игровых скриптов в виде WinUI 3 приложения с современным графическим интерфейсом. Решены все поставленные задачи: проведён анализ предметной области, сформулированы функциональные и нефункциональные требования, выполнено функциональное моделирование и проектирование архитектуры с применением методологии UML.

На этапе проектирования архитектуры и пользовательского интерфейса применён паттерн MVVM, обеспечивающий строгое разделение ответственности между визуальным слоем, логикой представления и бизнес-правилами генерации конфигураций. Разработана модульная структура клиентской части, включающая компоненты управления состоянием, адаптеры доменных моделей и абстракции сервисов валидации данных. Проектирование визуального интерфейса выполнено в соответствии с принципами Fluent Design System, что обеспечило адаптивность разметки, единообразие элементов управления и соответствие стандартам доступности. Созданные модели потоков данных и диаграммы взаимодействия компонентов формализовали логику реактивных привязок, механизмы обновления интерфейса и маршрутизацию пользовательских команд, что минимизировало риски архитектурных противоречий на стадии разработки.

Программная реализация клиентского приложения выполнена на платформе WinUI 3 с применением языка C# и декларативной разметки XAML. Внедрены механизмы компилируемых привязок данных, гарантирующие высокую производительность рендеринга и статическую проверку типов на этапе сборки. Для обработки асинхронных операций экстракции игровых ресурсов и парсинга конфигурационных файлов применён подход на основе задач с управлением параллелизмом через семафоры, что исключает блокировку основного потока пользовательского интерфейса. Интеграция внешних библиотек анализа синтаксиса и валидации структур данных осуществлена через

изолированные сервисные интерфейсы, обеспечивающие слабую связанность модулей и возможность масштабирования функциональности без модификации ядра системы.

Верификация разработанного программного обеспечения проведена методом ручного функционального и эвристического тестирования. Контроль качества осуществлялся по количественным метрикам эффективности взаимодействия, включающим временные затраты на выполнение типовых сценариев, количество интерактивных воздействий и коэффициент успешного завершения задач. Устранение критических отклонений в механизмах валидации входных параметров и обработки файловой системы подтвердило стабильность работы приложения, корректность генерации целевых скриптов и его техническую готовность к эксплуатации.

Создана система клиентского WinUI 3 приложения, включающая: главное окно управления рецептами со списком предметов, визуальный конструктор игровых рецептов, компонент поиска по названиям и тегам и выбора игровых предметов с фильтрацией, страницу импорта скриптов из существующих файлов. Выполнены планирование и расчёт бюджета: трудоёмкость составила 48 календарных дней, ориентировочная стоимость – 397 600 руб. Разработанное клиентское программное обеспечение позволяет снизить трудоёмкость создания скриптов, исключить необходимость написания кода пользователем, минимизировать ошибки ввода и повысить доступность инструментария создания и изменения игровых скриптов для широкой аудитории.

Список использованных источников

1. KubeJS Official Documentation // KubeJS [Электронный ресурс]. URL: <https://kubejs.com/> (дата обращения: 01.06.2026).
2. CraftTweaker Documentation // docs.blamejared.com [Электронный ресурс]. URL: <https://docs.blamejared.com/> (дата обращения: 01.06.2026).
3. MCreator Official Website // MCreator [Электронный ресурс]. URL: <https://mcreator.net/> (дата обращения: 01.06.2026).
4. WinUI 3 Documentation // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/windows/apps/winui/winui3/> (дата обращения: 01.06.2026).
5. C# Documentation // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 01.06.2026).
6. .NET 8 Documentation // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/dotnet/core/whats-new/dotnet-8/overview> (дата обращения: 01.06.2026).
7. UML. Официальная спецификация Unified Modeling Language // Object Management Group [Электронный ресурс]. URL: <https://www.omg.org/spec/UML/> (дата обращения: 01.06.2026).
8. Приложение М. Оформление схем и диаграмм // StudFiles [Электронный ресурс]. URL: <https://studfile.net/preview/11104650/page:24/> (дата обращения: 01.06.2026).
9. Диаграмма Ганта. Общие сведения // Microsoft Support [Электронный ресурс]. URL: <https://support.microsoft.com/ru-ru/project/work-with-the-gantt-chart-view> (дата обращения: 01.06.2026).
10. Model-View-ViewModel (MVVM) pattern for .NET MAUI // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm> (дата обращения: 02.06.2026).

11. XAML overview for WinUI 3 // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/windows/apps/winui/winui3/xaml-overview> (дата обращения: 02.06.2026).
12. INotifyPropertyChanged Interface (.NET) // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/dotnet/api/system.componentmodel.inotifypropertychanged> (дата обращения: 02.06.2026).
13. Create Mod Official Website // Create Mod [Электронный ресурс]. URL: <https://createmod.com/> (дата обращения: 02.06.2026).
14. Введение в UML 2.0. Лекция 4: Диаграмма классов // НОУ ИНТУИТ [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/1007/209/lecture/5264> (дата обращения: 02.06.2026).
15. ГОСТ Р 56920–2024. Системная и программная инженерия. Тестирование программного обеспечения. Общие положения. – Введ. 2024 – 30– 09. – Москва : Стандартиформ, 2024. – 27 с.
16. Култыгин О. П. Тестирование программного обеспечения. Учебное пособие // Intuit.ru [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/1007/209/info> (дата обращения: 02.06.2026).
17. Функциональная диаграмма (причинно-следственная диаграмма) в тестировании // Software-testing.ru [Электронный ресурс]. URL: <https://software-testing.ru/> (дата обращения: 02.06.2026).
18. Правила (эвристики) Якоба Нильсена // Uprock.ru [Электронный ресурс]. URL: <https://www.uprock.ru/journal/lab/10-jakob-nielsen-heuristics/> (дата обращения: 02.06.2026).
19. Функциональное тестирование. Ручное тестирование. QA evolution // Tquality.ru [Электронный ресурс]. URL: <https://tquality.ru/blog/funktsionalnoe-testirovanie/> (дата обращения: 02.06.2026).
20. SLOC. Source Lines of Code // Habr.com [Электронный ресурс]. URL: <https://habr.com/ru/articles/104756/> (дата обращения: 02.06.2026).

21. Halstead's Software Metrics // GeeksforGeeks.com [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Холстед,_Морис_Х. (дата обращения: 02.06.2026).

Приложение А
(справочное)
Глоссарий

KubeJS – модификация для Minecraft, предоставляющая API на языке JavaScript для создания и модификации игровых рецептов, событий и механик без необходимости написания кода.

Рецепт – игровая механика, определяющая правила получения одних предметов из других посредством.

Модификация – стороннее дополнение к Minecraft, расширяющее игровой функционал новыми предметами, механиками и рецептами.

Скрипт – файл, содержащий инструкции на языке JavaScript для KubeJS, определяющий поведение игровых рецептов.

Модификатор рецепта – дополнительный параметр, специфичный для определённых типов рецептов (например, время обработки, температура нагрева, опыт).

MVVM (Model-View-ViewModel) – архитектурный паттерн, разделяющий приложение на три слоя: модель данных, представление (UI) и модель представления (прослойка с логикой и состоянием).

Привязка данных – механизм автоматической синхронизации данных между источником (ViewModel) и элементом управления (View).

UX (User Experience) – пользовательский опыт, совокупность ощущений и впечатлений от взаимодействия с интерфейсом приложения.

Валидация – проверка введённых пользователем данных на соответствие заданным ограничениям и правилам.

Приложение Б
(обязательное)
Код программы

```

<Window
    x:Class="KubeAutomation.WinUIWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:KubeAutomation"
    Title="KubeJS Automation - WinUI 3">

    <Grid>
        <Grid.Resources>
            <local:BoolToVisibilityConverter
x:Key="BoolToVisibility"/>
            <local:NullToVisibilityConverter
x:Key="NullToVisibility"/>
        </Grid.Resources>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*/>
        </Grid.RowDefinitions>

        <!-- Панель поиска -->
        <Grid Grid.Row="0" Margin="10">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*/>
                <ColumnDefinition Width="Auto"/>
            </Grid.ColumnDefinitions>
            <TextBox x:Name="SearchBox" Grid.Column="0"
PlaceholderText="Поиск предметов..."
TextChanged="SearchBox_TextChanged"/>
            <Button Grid.Column="1" Content="Clear"
Click="ClearSearch_Click" Margin="5,0,0,0"/>
        </Grid>

        <!-- Основной контент -->
        <Grid Grid.Row="1">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*/ />
                <ColumnDefinition Width="*/ />
            </Grid.ColumnDefinitions>

            <!-- Левая панель: Список предметов -->
            <ScrollViewer Grid.Column="0" Margin="10">
                <ItemsRepeater x:Name="ItemsRepeater"
ItemsSource="{x:Bind FilteredItems, Mode=OneWay}">
                    <ItemsRepeater.Layout>
                        <UniformGridLayout MinItemWidth="80"
MinItemHeight="80" />
                    </ItemsRepeater.Layout>
                    <ItemsRepeater.ItemTemplate>
                        <DataTemplate
x:DataType="local:MinecraftItemViewModel">
                            <Border Margin="5" Padding="1"
BorderBrush="Black" BorderThickness="1" Background="LightGray"

```

```

Tag="{x:Bind}"

PointerPressed="ItemBorder_PointerPressed">
    <Image Source="{x:Bind Image,
Mode=OneWay}" Width="64" Height="64" Stretch="Uniform" />
    <ToolTipService.ToolTip>
        <StackPanel>
            <TextBlock Text="{x:Bind
LocalizedName}" FontWeight="Bold"/>
            <TextBlock Text="{x:Bind
ModName}"/>
            <TextBlock Text="{x:Bind
Id}"/>
        </StackPanel>
    </ToolTipService.ToolTip>
</Border>
</DataTemplate>
</ItemsRepeater.ItemTemplate>
</ItemsRepeater>
</ScrollView>

<!-- Правая панель: вкладки -->
<TabView x:Name="MainTabView" Grid.Column="1"
Margin="10"
    IsAddTabButtonVisible="False"
    TabWidthMode="Equal">

    <!-- Вкладка: Создать -->
    <TabViewItem Header="Создать">
        <ScrollView Padding="8">
            <StackPanel Spacing="8">

                <!-- Выбор типа рецепта -->
                <StackPanel Orientation="Horizontal"
Spacing="8">
                    <TextBlock Text="Тип:"
VerticalAlignment="Center" Width="60"/>
                    <ComboBox x:Name="BlockComboBox"
ItemsSource="{x:Bind
ViewModel.RecipeBlocks, Mode=OneWay}"
Width="200"
SelectionChanged="BlockComboBox_SelectionChanged"/>
                </StackPanel>

                <!-- ===== Единая секция рецепта =====
-->
                <StackPanel Spacing="8">
                    <TextBlock Text="Параметры
рецепта" FontWeight="SemiBold" FontSize="13"
Foreground="Gray"/>

                    <!-- Входы -->

```

```

FontWeight="SemiBold"/>
<TextBlock Text="Входы:"
x:Name="InputSlotsRepeater"
Visibility="{x:Bind
ViewModel.ShowInputSlots, Mode=OneWay}"
ItemsSource="{x:Bind ViewModel.InputSlots, Mode=OneWay}">
    <ItemsRepeater.ItemTemplate>
        <DataTemplate
x:DataType="local:RecipeSlotViewModel">
            <StackPanel
Orientation="Horizontal" Spacing="6" Margin="0,2">
                <Border Width="48"
Height="48"
BorderBrush="Gray" BorderThickness="2"
Background="LightGray" CornerRadius="4"
Tag="{x:Bind}"
PointerPressed="Slot_PointerPressed"
RightTapped="Slot_RightTapped"
ToolTipService.ToolTip="ЛКМ - выбрать, ПКМ - очистить">
                    <Grid>
                        <Image
Source="{x:Bind Item.Image, Mode=OneWay}"
Width="40" Height="40" Stretch="Uniform"
Visibility="{x:Bind ImageVisibility, Mode=OneWay}"/>
                        <TextBlock
Text="?" HorizontalAlignment="Center"
VerticalAlignment="Center"
FontSize="16" Foreground="DarkGray"
Visibility="{x:Bind PlaceholderVisibility, Mode=OneWay}"/>
                    </Grid>
                </Border>
                <TextBlock
Text="{x:Bind ItemId, Mode=OneWay}"
VerticalAlignment="Center" MaxWidth="180"
TextWrapping="Wrap" FontSize="12"/>
            </StackPanel>
        </DataTemplate>
    </ItemsRepeater.ItemTemplate>

```

```

        </ItemsRepeater>
        <StackPanel
Orientation="Horizontal" Spacing="4"
                                Visibility="{x:Bind
ViewModel.ShowInputButtons, Mode=OneWay}">
                                <Button Content="+ Вход"
Click="AddInput_Click"
                                IsEnabled="{x:Bind
ViewModel.CanAddInput, Mode=OneWay}"/>
                                <Button Content="- Вход"
Click="RemoveInput_Click"
                                IsEnabled="{x:Bind
ViewModel.CanRemoveInput, Mode=OneWay}"/>
        </StackPanel>

        <!-- Вектор 3x3 -->
        <Grid Visibility="{x:Bind
ViewModel.ShowWorkbenchGrid, Mode=OneWay}"
                                HorizontalAlignment="Left">
                                <Grid.RowDefinitions>
                                <RowDefinition
Height="Auto"/>
                                <RowDefinition
Height="Auto"/>
                                <RowDefinition
Height="Auto"/>
                                </Grid.RowDefinitions>
                                <Grid.ColumnDefinitions>
                                <ColumnDefinition
Width="Auto"/>
                                <ColumnDefinition
Width="Auto"/>
                                <ColumnDefinition
Width="Auto"/>
                                </Grid.ColumnDefinitions>

                                <!-- Row 0 -->
                                <Border Grid.Row="0"
Grid.Column="0" Width="48" Height="48" Margin="2"
                                BorderBrush="Gray"
BorderThickness="2" Background="LightGray" CornerRadius="4"
                                Tag="{x:Bind
ViewModel.WorkbenchSlots[0]}"
                                PointerPressed="Slot_PointerPressed"
                                RightTapped="Slot_RightTapped"
                                ToolTipService.ToolTip="ЛКМ - выбрать, ПКМ - очистить">
                                <Grid>
                                <Image Source="{x:Bind
ViewModel.WorkbenchSlots[0].Item.Image, Mode=OneWay}" Width="40"
Height="40" Stretch="Uniform"

```

```

Visibility="{x:Bind ViewModel.WorkbenchSlots[0].ImageVisibility,
Mode=OneWay}"/>
                                <TextBlock Text="?"
HorizontalAlignment="Center" VerticalAlignment="Center"

FontSize="16" Foreground="DarkGray"

Visibility="{x:Bind
ViewModel.WorkbenchSlots[0].PlaceholderVisibility, Mode=OneWay}"/>
                                </Grid>
                                </Border>
                                <Border Grid.Row="0"
Grid.Column="1" Width="48" Height="48" Margin="2"
                                BorderBrush="Gray"
BorderThickness="2" Background="LightGray" CornerRadius="4"
                                Tag="{x:Bind
ViewModel.WorkbenchSlots[1]}"

PointerPressed="Slot_PointerPressed"

RightTapped="Slot_RightTapped"

ToolTipService.ToolTip="ЛКМ - выбрать, ПКМ - очистить">
                                <Grid>
                                <Image Source="{x:Bind
ViewModel.WorkbenchSlots[1].Item.Image, Mode=OneWay}" Width="40"
Height="40" Stretch="Uniform"

Visibility="{x:Bind ViewModel.WorkbenchSlots[1].ImageVisibility,
Mode=OneWay}"/>
                                <TextBlock Text="?"
HorizontalAlignment="Center" VerticalAlignment="Center"

FontSize="16" Foreground="DarkGray"

Visibility="{x:Bind
ViewModel.WorkbenchSlots[1].PlaceholderVisibility, Mode=OneWay}"/>
                                </Grid>
                                </Border>
                                <Border Grid.Row="0"
Grid.Column="2" Width="48" Height="48" Margin="2"
                                BorderBrush="Gray"
BorderThickness="2" Background="LightGray" CornerRadius="4"
                                Tag="{x:Bind
ViewModel.WorkbenchSlots[2]}"

PointerPressed="Slot_PointerPressed"

RightTapped="Slot_RightTapped"

ToolTipService.ToolTip="ЛКМ - выбрать, ПКМ - очистить">
                                <Grid>

```

```

<Image Source="{x:Bind
ViewModel.WorkbenchSlots[2].Item.Image, Mode=OneWay}" Width="40"
Height="40" Stretch="Uniform"

Visibility="{x:Bind ViewModel.WorkbenchSlots[2].ImageVisibility,
Mode=OneWay}"/>

<TextBlock Text="?"
HorizontalAlignment="Center" VerticalAlignment="Center"

FontSize="16" Foreground="DarkGray"

Visibility="{x:Bind
ViewModel.WorkbenchSlots[2].PlaceholderVisibility, Mode=OneWay}"/>
</Grid>
</Border>

<!-- Row 1 -->
<Border Grid.Row="1"
Grid.Column="0" Width="48" Height="48" Margin="2"
BorderBrush="Gray"
BorderThickness="2" Background="LightGray" CornerRadius="4"
Tag="{x:Bind
ViewModel.WorkbenchSlots[3]}"

PointerPressed="Slot_PointerPressed"

RightTapped="Slot_RightTapped"

ToolTipService.ToolTip="ЛКМ - выбрать, ПКМ - очистить">
<Grid>
<Image Source="{x:Bind
ViewModel.WorkbenchSlots[3].Item.Image, Mode=OneWay}" Width="40"
Height="40" Stretch="Uniform"

Visibility="{x:Bind ViewModel.WorkbenchSlots[3].ImageVisibility,
Mode=OneWay}"/>

<TextBlock Text="?"
HorizontalAlignment="Center" VerticalAlignment="Center"

FontSize="16" Foreground="DarkGray"

Visibility="{x:Bind
ViewModel.WorkbenchSlots[3].PlaceholderVisibility, Mode=OneWay}"/>
</Grid>
</Border>
<Border Grid.Row="1"
Grid.Column="1" Width="48" Height="48" Margin="2"
BorderBrush="Gray"
BorderThickness="2" Background="LightGray" CornerRadius="4"
Tag="{x:Bind
ViewModel.WorkbenchSlots[4]}"

PointerPressed="Slot_PointerPressed"

```

```

RightTapped="Slot_RightTapped"

ToolTipService.ToolTip="ЛКМ - выбрать, ПКМ - очистить">
    <Grid>
        <Image Source="{x:Bind
ViewModel.WorkbenchSlots[4].Item.Image, Mode=OneWay}" Width="40"
Height="40" Stretch="Uniform"

Visibility="{x:Bind ViewModel.WorkbenchSlots[4].ImageVisibility,
Mode=OneWay}"/>
        <TextBlock Text="?"
HorizontalAlignment="Center" VerticalAlignment="Center"

FontSize="16" Foreground="DarkGray"

Visibility="{x:Bind
ViewModel.WorkbenchSlots[4].PlaceholderVisibility, Mode=OneWay}"/>
    </Grid>
</Border>
<Border Grid.Row="1"
Grid.Column="2" Width="48" Height="48" Margin="2"
BorderBrush="Gray"
BorderThickness="2" Background="LightGray" CornerRadius="4"
Tag="{x:Bind
ViewModel.WorkbenchSlots[5]}"

PointerPressed="Slot_PointerPressed"

RightTapped="Slot_RightTapped"

ToolTipService.ToolTip="ЛКМ - выбрать, ПКМ - очистить">
    <Grid>
        <Image Source="{x:Bind
ViewModel.WorkbenchSlots[5].Item.Image, Mode=OneWay}" Width="40"
Height="40" Stretch="Uniform"

Visibility="{x:Bind ViewModel.WorkbenchSlots[5].ImageVisibility,
Mode=OneWay}"/>
        <TextBlock Text="?"
HorizontalAlignment="Center" VerticalAlignment="Center"

FontSize="16" Foreground="DarkGray"

Visibility="{x:Bind
ViewModel.WorkbenchSlots[5].PlaceholderVisibility, Mode=OneWay}"/>
    </Grid>
</Border>

<!-- Row 2 -->
<Border Grid.Row="2"
Grid.Column="0" Width="48" Height="48" Margin="2"

```

```

BorderThickness="2" Background="LightGray" BorderBrush="Gray" CornerRadius="4"
Tag="{x:Bind
ViewModel.WorkbenchSlots[6]}"

PointerPressed="Slot_PointerPressed"

RightTapped="Slot_RightTapped"

ToolTipService.ToolTip="ЛКМ - выбрать, ПКМ - очистить">
    <Grid>
        <Image Source="{x:Bind
ViewModel.WorkbenchSlots[6].Item.Image, Mode=OneWay}" Width="40"
Height="40" Stretch="Uniform"

Visibility="{x:Bind ViewModel.WorkbenchSlots[6].ImageVisibility,
Mode=OneWay}"/>
        <TextBlock Text="?"
HorizontalAlignment="Center" VerticalAlignment="Center"

FontSize="16" Foreground="DarkGray"

Visibility="{x:Bind
ViewModel.WorkbenchSlots[6].PlaceholderVisibility, Mode=OneWay}"/>
    </Grid>
</Border>
<Border Grid.Row="2"
Grid.Column="1" Width="48" Height="48" Margin="2"
BorderBrush="Gray"
BorderThickness="2" Background="LightGray" CornerRadius="4"
Tag="{x:Bind
ViewModel.WorkbenchSlots[7]}"

PointerPressed="Slot_PointerPressed"

RightTapped="Slot_RightTapped"

ToolTipService.ToolTip="ЛКМ - выбрать, ПКМ - очистить">
    <Grid>
        <Image Source="{x:Bind
ViewModel.WorkbenchSlots[7].Item.Image, Mode=OneWay}" Width="40"
Height="40" Stretch="Uniform"

Visibility="{x:Bind ViewModel.WorkbenchSlots[7].ImageVisibility,
Mode=OneWay}"/>
        <TextBlock Text="?"
HorizontalAlignment="Center" VerticalAlignment="Center"

FontSize="16" Foreground="DarkGray"

Visibility="{x:Bind
ViewModel.WorkbenchSlots[7].PlaceholderVisibility, Mode=OneWay}"/>
    </Grid>

```

```

        </Border>
        <Border Grid.Row="2"
Grid.Column="2" Width="48" Height="48" Margin="2"
        BorderBrush="Gray"
BorderThickness="2" Background="LightGray" CornerRadius="4"
        Tag="{x:Bind
ViewModel.WorkbenchSlots[8]}"

PointerPressed="Slot_PointerPressed"

RightTapped="Slot_RightTapped"

ToolTipService.ToolTip="ЛКМ - выбрать, ПКМ - очистить">
        <Grid>
            <Image Source="{x:Bind
ViewModel.WorkbenchSlots[8].Item.Image, Mode=OneWay}" Width="40"
Height="40" Stretch="Uniform"

Visibility="{x:Bind ViewModel.WorkbenchSlots[8].ImageVisibility,
Mode=OneWay}"/>
            <TextBlock Text="?"
HorizontalAlignment="Center" VerticalAlignment="Center"

FontSize="16" Foreground="DarkGray"

Visibility="{x:Bind
ViewModel.WorkbenchSlots[8].PlaceholderVisibility, Mode=OneWay}"/>
        </Grid>
    </Border>
</Grid>

<!-- Выходы -->
<TextBlock Text="Выходы:"

FontWeight="SemiBold"/>
    <ItemsRepeater
x:Name="OutputSlotsRepeater"

ItemsSource="{x:Bind ViewModel.OutputSlots, Mode=OneWay}">
        <ItemsRepeater.ItemTemplate>
            <DataTemplate
x:DataType="local:RecipeSlotViewModel">
                <StackPanel
Orientation="Horizontal" Spacing="6" Margin="0,2">
                    <Border Width="48"
Height="48"

BorderBrush="Gray" BorderThickness="2"

Background="LightGray" CornerRadius="4"

Tag="{x:Bind}"

PointerPressed="Slot_PointerPressed"

```

```

RightTapped="Slot_RightTapped"

ToolTipService.ToolTip="ЛКМ - выбрать, ПКМ - очистить">
    <Grid>
        <Image
Source="{x:Bind Item.Image, Mode=OneWay}"
Width="40" Height="40" Stretch="Uniform"
Visibility="{x:Bind ImageVisibility, Mode=OneWay}"/>
    <TextBlock
Text="?" HorizontalAlignment="Center"
VerticalAlignment="Center"
FontSize="16" Foreground="DarkGray"
Visibility="{x:Bind PlaceholderVisibility, Mode=OneWay}"/>
    </Grid>
</Border>
<TextBlock
Text="{x:Bind ItemId, Mode=OneWay}"
VerticalAlignment="Center" MaxWidth="120"
TextWrapping="Wrap" FontSize="12"/>
    <!-- Количество --
>
    <StackPanel
Orientation="Horizontal" Spacing="4"
Visibility="{x:Bind AmountVisibility}"
VerticalAlignment="Center">
    <TextBlock
Text="Кол-во:" VerticalAlignment="Center" FontSize="12"/>
    <NumberBox
Value="{x:Bind Amount, Mode=TwoWay}"
Minimum="1" Maximum="64" SmallChange="1"
Width="Auto" SpinButtonPlacementMode="Inline"
FontSize="12"/>
    </StackPanel>
    <!-- Шанс
выпадения -->
    <StackPanel
Orientation="Horizontal" Spacing="4"
Visibility="{x:Bind ChanceVisibility}"

```

```

VerticalAlignment="Center">
Text="Шанс:" VerticalAlignment="Center" FontSize="12"/>
Value="{x:Bind Chance, Mode=TwoWay}"
Minimum="0.01" Maximum="1.0" SmallChange="0.05"
Width="90" SpinButtonPlacementMode="Inline"
FontSize="12"/>
</StackPanel>
</StackPanel>
</DataTemplate>
</ItemsRepeater.ItemTemplate>
</ItemsRepeater>
<StackPanel
Orientation="Horizontal" Spacing="4"
Visibility="{x:Bind
ViewModel.ShowOutputButtons, Mode=OneWay}">
<Button Content="+ Выход"
Click="AddOutput_Click"
IsEnabled="{x:Bind
ViewModel.CanAddOutput, Mode=OneWay}"/>
<Button Content="- Выход"
Click="RemoveOutput_Click"
IsEnabled="{x:Bind
ViewModel.CanRemoveOutput, Mode=OneWay}"/>
</StackPanel>
<!-- Модификаторы -->
<StackPanel Visibility="{x:Bind
ViewModel.ShowModifierHeated, Mode=OneWay}"
Spacing="4">
<TextBlock Text="Нагрев:"
FontWeight="SemiBold"/>
<StackPanel
Orientation="Horizontal" Spacing="8">
<RadioButton
x:Name="HeatNoneRadio" Content="Нет"
GroupName="HeatGroup" IsChecked="True"
Checked="HeatRadio_Checked" Tag="none"/>
<RadioButton
Content="Heated"
GroupName="HeatGroup"
Checked="HeatRadio_Checked" Tag="heated"/>
<RadioButton
Content="Superheated"

```

```

GroupName="HeatGroup"

Checked="HeatRadio_Checked" Tag="superheated"/>
    </StackPanel>
</StackPanel>

    <StackPanel Visibility="{x:Bind
ViewModel.ShowModifierProcessingTime, Mode=OneWay}"

Orientation="Horizontal" Spacing="8">
    <TextBlock Text="Время
(тики):" VerticalAlignment="Center"/>
    <NumberBox Value="{x:Bind
ViewModel.ProcessingTime, Mode=TwoWay}"
        Minimum="1"
Maximum="10000" SmallChange="10"
        Width="120"
SpinButtonPlacementMode="Inline"/>
    </StackPanel>

    <CheckBox Visibility="{x:Bind
ViewModel.ShowModifierKeepHeldItem, Mode=OneWay}"
        Content="keepHeldItem"
        IsChecked="{x:Bind
ViewModel.KeepHeldItem, Mode=TwoWay}"/>

    <StackPanel Visibility="{x:Bind
ViewModel.ShowModifierExperience, Mode=OneWay}"

Orientation="Horizontal" Spacing="8">
    <TextBlock Text="Опыт:"
VerticalAlignment="Center"/>
    <NumberBox Value="{x:Bind
ViewModel.Experience, Mode=TwoWay}"
        Minimum="0"
Maximum="100" SmallChange="0.1"
        Width="Auto"
SpinButtonPlacementMode="Inline"/>
    </StackPanel>
</StackPanel>

<!-- Имя файла -->
<StackPanel Orientation="Horizontal"
Spacing="8"
    Visibility="{x:Bind
ViewModel.FileNameVisibility, Mode=OneWay}">
    <TextBlock Text="Файл:"
VerticalAlignment="Center" Width="60"/>
    <TextBox
x:Name="CreateFileNameBox"
        PlaceholderText="имя
файла (без .js)"

```

```

Width="200"/>
</StackPanel>

<TextBlock x:Name="SaveStatusBlock"
Text="" FontSize="12"
TextWrapping="Wrap"/>

<Button x:Name="GenerateRecipeButton"
Content="{x:Bind
ViewModel.CreateButtonLabel, Mode=OneWay}"
Click="GenerateRecipeButton_Click"
HorizontalAlignment="Left"/>
<Button x:Name="CancelEditButton"
Content="Отмена"
Click="CancelEditButton_Click"
HorizontalAlignment="Left"
Visibility="{x:Bind
ViewModel.CancelEditVisibility, Mode=OneWay}"/>

<TextBlock x:Name="GeneratedCodeBlock"
Text=""
FontFamily="Consolas" FontSize="12"
TextWrapping="Wrap"
IsTextSelectionEnabled="True"
Foreground="DarkGreen"/>
</StackPanel>
</ScrollViewer>
</TabViewItem>

<!-- Вкладка: Удалить -->
<TabViewItem Header="Удалить">
<ScrollViewer Padding="8">
<StackPanel Spacing="8">
<TextBlock Text="Удаление рецептов"
FontWeight="SemiBold" FontSize="14"/>
<TextBlock Text="Заполните хотя бы
одно поле фильтра (AND внутри фильтра)."
Foreground="Gray"
FontSize="12" TextWrapping="Wrap"/>

<!-- Output: ячейка выбора предмета -->
<StackPanel Orientation="Horizontal"
Spacing="8">
<TextBlock Text="Output:"
VerticalAlignment="Center" Width="80"/>
<Border Width="48" Height="48"
BorderBrush="Gray"
BorderThickness="2"
Background="LightGray"
CornerRadius="4"

```

```

Tag="{x:Bind
ViewModel.Re movalOutputSlot}"

PointerPressed="Slot_PointerPressed"

RightTapped="Slot_RightTapped"

ToolTipService.ToolTip="ЛКМ - выбрать, ПКМ - очистить">
    <Grid>
        <Image Source="{x:Bind
ViewModel.Re movalOutputSlot.Item.Image, Mode=OneWay}"
            Width="40"
            Height="40" Stretch="Uniform"
            Visibility="{x:Bind
ViewModel.Re movalOutputSlot.ImageVisibility, Mode=OneWay}"/>
        <TextBlock Text="?"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"
            FontSize="16"
            Foreground="DarkGray"
            Visibility="{x:Bind
ViewModel.Re movalOutputSlot.PlaceholderVisibility, Mode=OneWay}"/>
    </Grid>
</Border>
<TextBlock Text="{x:Bind
ViewModel.Re movalOutputSlot.ItemId, Mode=OneWay}"
    VerticalAlignment="Center" MaxWidth="180"
    TextWrapping="Wrap"
    FontSize="12"/>
</StackPanel>

<!-- Input: ячейка выбора предмета -->
<StackPanel Orientation="Horizontal"
Spacing="8">
    <TextBlock Text="Input:"
    VerticalAlignment="Center" Width="80"/>
    <Border Width="48" Height="48"
        BorderBrush="Gray"
        BorderThickness="2"
        Background="LightGray"
        CornerRadius="4"
        Tag="{x:Bind
ViewModel.Re movalInputSlot}"
        PointerPressed="Slot_PointerPressed"
        RightTapped="Slot_RightTapped"
        ToolTipService.ToolTip="ЛКМ - выбрать, ПКМ - очистить">
        <Grid>

```

```

        <Image Source="{x:Bind
ViewModel.RemovalInputSlot.Item.Image, Mode=OneWay}"
                                Width="40"
Height="40" Stretch="Uniform"
                                Visibility="{x:Bind
ViewModel.RemovalInputSlot.ImageVisibility, Mode=OneWay}"/>
        <TextBlock Text="?"
HorizontalAlignment="Center"
VerticalAlignment="Center"
                                FontSize="16"
Foreground="DarkGray"
Visibility="{x:Bind
ViewModel.RemovalInputSlot.PlaceholderVisibility, Mode=OneWay}"/>
    </Grid>
</Border>
    <TextBlock Text="{x:Bind
ViewModel.RemovalInputSlot.ItemId, Mode=OneWay}"
VerticalAlignment="Center" MaxWidth="180"
                                TextWrapping="Wrap"
FontSize="12"/>
</StackPanel>

    <!-- Type: выпадающий список -->
    <StackPanel Orientation="Horizontal"
Spacing="8">
        <TextBlock Text="Type:"
VerticalAlignment="Center" Width="80"/>
        <ComboBox x:Name="RemoveTypeCombo"
ItemsSource="{x:Bind
ViewModel.RemoveTypeOptions, Mode=OneWay}"
PlaceholderText="не
выбран"
Width="220"/>
    </StackPanel>

    <!-- Mod: выпадающий список -->
    <StackPanel Orientation="Horizontal"
Spacing="8">
        <TextBlock Text="Mod:"
VerticalAlignment="Center" Width="80"/>
        <ComboBox x:Name="RemoveModCombo"
ItemsSource="{x:Bind
ViewModel.RemoveModOptions, Mode=OneWay}"
PlaceholderText="не
выбран"
Width="220"/>
    </StackPanel>

    <!-- Id: текстовое поле -->

```

```

        <StackPanel Orientation="Horizontal"
Spacing="8">
            <TextBlock Text="Id:"
VerticalAlignment="Center" Width="80"/>
            <TextBox x:Name="RemoveIdBox"
PlaceholderText="minecraft:glowstone" Width="220"/>
        </StackPanel>

        <!-- Имя файла -->
        <StackPanel Orientation="Horizontal"
Spacing="8">
            <TextBlock Text="Файл:"
VerticalAlignment="Center" Width="80"/>
            <TextBox
x:Name="RemoveFileNameBox"
PlaceholderText="имя
файла (без .js)"
Width="220"/>
        </StackPanel>

        <Button x:Name="RemoveRecipeButton"
Content="Удалить рецепт"

Click="RemoveRecipeButton_Click"
HorizontalAlignment="Left"/>

        <TextBlock x:Name="RemoveStatusBlock"
Text="" FontSize="12"
TextWrapping="Wrap"/>

        <TextBlock x:Name="RemoveCodeBlock"
Text=""
FontFamily="Consolas" FontSize="12"
TextWrapping="Wrap"
IsTextSelectionEnabled="True"
Foreground="DarkGreen"/>
    </StackPanel>
</ScrollViewer>
</TabViewItem>

<!-- Вкладка: Файл -->
<TabViewItem Header="Файл">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*/>
        </Grid.RowDefinitions>

        <!-- Шапка: выбор файла -->
        <StackPanel Grid.Row="0" Spacing="8"
Margin="8">
            <StackPanel Orientation="Horizontal"
Spacing="8">

```

```

        <TextBox x:Name="FileTabPathBox"
                PlaceholderText="Путь к
.js файлу"
                Width="300"
                Text="{x:Bind
ViewModel.RecipeFile.FilePath, Mode=TwoWay}"/>
        <Button Content="Обзор..."
Click="FileTab_Browse_Click"/>
        <Button Content="Открыть"
Click="FileTab_OpenFile_Click"/>
        <Button Content="Обновить"
Click="FileTab_Reload_Click"/>
    </StackPanel>
    <TextBlock Text="{x:Bind
ViewModel.RecipeFile.StatusText, Mode=OneWay}"
                FontSize="12"
TextWrapping="Wrap"/>
</StackPanel>

    <!-- Список рецептов -->
    <ListView Grid.Row="1"
                ItemsSource="{x:Bind
ViewModel.RecipeFile.Entries, Mode=OneWay}"
                SelectionMode="None">
        <ListView.ItemTemplate>
            <DataTemplate
x:DataType="local:RecipeEntryViewModel">
                <Grid Margin="0,4">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition
Width="*" />
                        <ColumnDefinition
Width="Auto" />
                        <ColumnDefinition
Width="Auto" />
                        <ColumnDefinition
Width="Auto" />
                    </Grid.ColumnDefinitions>
                    <Button Grid.Column="0"
                        Content="{x:Bind
DisplayName}"
                        HorizontalAlignment="Stretch"
                        HorizontalContentAlignment="Left"
                        FontFamily="Consolas" FontSize="12"
                        Background="Transparent"
                        BorderThickness="0"
                        Tag="{x:Bind}"

```

```

Click="FileTab_EditEntry_Click"

ToolTipService.ToolTip="Редактировать рецепт"/>
<Button Grid.Column="1"
        Content=" [<+]"
        Margin="8,0,0,0"
        IsEnabled="{x:Bind
CanInsert}"
        Tag="{x:Bind}"

Click="FileTab_InsertBefore_Click"

ToolTipService.ToolTip="Вставить рецепт до"/>
<Button Grid.Column="2"
        Content=" [+>]"
        Margin="4,0,0,0"
        IsEnabled="{x:Bind
CanInsert}"
        Tag="{x:Bind}"

Click="FileTab_InsertAfter_Click"

ToolTipService.ToolTip="Вставить рецепт после"/>
<Button Grid.Column="3"
        Content="Удалить"
        Margin="4,0,0,0"
        IsEnabled="{x:Bind
CanDelete}"
        Tag="{x:Bind}"

Click="FileTab_DeleteEntry_Click"/>
        </Grid>
    </DataTemplate>
</ListView.ItemTemplate>
</ListView>
</Grid>
</TabViewItem>

</TabView>
</Grid>

</Grid>
</Window>

```

Полный код приложения приведен на [GitHub](https://github.com/Coatllicence/KubejsAutomation):
<https://github.com/Coatllicence/KubejsAutomation>.