МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
**Высшая школа электроники и компьютерных наук**
**Кафедра системного программирования**

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

# Разработка GUI библиотеки на языке C++

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.926-013.ВКР

Научный руководитель,
профессор кафедры СП,
д.ф.-м.н., доцент
_____ Т.А. Макаровских

Автор работы,
студент группы КЭ-402
_____ А.А. Носков

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет**
**(национальный исследовательский университет)»**
**Высшая школа электроники и компьютерных наук**
**Кафедра системного программирования**

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

# ЗАДАНИЕ
## на выполнение выпускной квалификационной работы бакалавра

студенту группы КЭ-402
Носкову Артему Александровичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12) Разработка GUI библиотеки на языке C++.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Human Interface Guidelines. [Электронный ресурс] URL: https://developer.apple.com/design/human-interface-guidelines/guidelines/overview (дата обращения: 12.02.2024 г.).
3.2. Skia Documentation. [Электронный ресурс] URL: https://skia.org/docs (дата обращения: 12.02.2024 г.).
3.3. Flutter's Rendering Pipeline. [Электронный ресурс] URL: https://www.youtube.com/watch?v=UUfXWzp0-DU (дата обращения: 12.02.2024 г.).
3.4. Flutter architectural overview. [Электронный ресурс] URL: https://docs.flutter.dev/resources/architectural-overview (дата обращения: 12.02.2024 г.).

4. **Перечень подлежащих разработке вопросов**

4.1. Изучить существующие библиотеки для разработки графических пользовательских приложений (GUI), изучить архитектуры и паттерны, на которых они спроектированы.
4.2. Привести описание требований к разрабатываемой GUI библиотеке на основе диаграмм вариантов использования UML.
4.3. Спроектировать структуру библиотеки и разработать необходимые модули.

4.4. Осуществить тестирование библиотеки.
4.5. Создать приложения для демонстрации возможностей разработанной библиотеки.
4.6. Опубликовать библиотеку и приложения на GitHub.
**5. Дата выдачи задания:** 29.01.2024 г.

**Научный руководитель,**
профессор кафедры СП, д.ф.-м.н., доцент Т.А. Макаровских

**Задание принял к исполнению** А.А. Носков

**ОГЛАВЛЕНИЕ**

# ВВЕДЕНИЕ

## Актуальность

В современном мире растет потребность в разработке графических настольных приложений с высокой производительностью, гибкими возможностями настройки и удобным пользовательским интерфейсом. Использование удобных и интуитивно понятных пользовательских интерфейсов важно для успешной реализации программных продуктов в различных областях. В связи с этим, разработка высококачественных GUI библиотек, способных удовлетворять динамично изменяющимся потребностям и ожиданиям пользователей, растущей сложность и масштабу задач, которые решаются с использованием современного программного обеспечения, является одной из ключевых задач в области программной инженерии. Отсутствие универсальных решений, способных обеспечить высокий уровень производительности и удобство использования для пользователей и создателей графических приложений, актуализирует необходимость создания новых технологий в данной сфере.

## Постановка задачи

Целью выпускной квалификационной работы является разработка GUI библиотеки на языке C++. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести обзор существующих библиотек;
2) привести описание требований к разрабатываемой библиотеке;
3) спроектировать и разработать библиотеку;
4) осуществить тестирование библиотеки;
5) создать демонстрационные приложения;
6) опубликовать библиотеку и приложения на GitHub.

**Структура и содержание работы**

Работа состоит из введения, семи глав, заключения и списка литературы. Объем работы составляет 41 страница, объем списка литературы – 27 источников.

В первой главе описывается анализ предметной области, существующих работ по данной тематике и обзор аналогичных проектов.

Вторая глава посвящена описанию функциональных и нефункциональных требований к разрабатываемой библиотеке, вариантов использования библиотеки.

В третьей главе производится проектирование разрабатываемой библиотеки, описание разработанной архитектуры.

В четвертой главе описаны детали реализации библиотеки, листинги функционала библиотеки.

Пятая глава посвящена тестированию библиотеки.

В шестой главе описано создание демонстрационных приложений.

Седьмая глава посвящается процессу публикации библиотеки на общедоступных платформах.

В приложении А содержится спецификация вариантов использования.

В приложении Б содержатся скриншоты демонстрационных приложений.

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1. Предметная область проекта

Целью данной работы является разработка библиотеки для разработки приложений с графическим пользовательским интерфейсом на языке C++. Разрабатываемая библиотека должна быть модульной, расширяемой и легкой в использовании с языками более высокого уровня, например: JavaScript, Python. Библиотека должна иметь достаточный набор возможностей для создания любых элементов современных графических, таких как: текст с форматированием, выбор даты и времени, выпадающий список, радиокнопка и другие.

Существует множество подходов к проектированию библиотек для создания графических пользовательских интерфейсов. По режиму работы их можно разделить на два типа: непосредственного режима (Immediate mode) и абстрактного режима (Retained mode), сравнении двух режимов приведено в статье [1]. Непосредственный режим заключается в процедурной отрисовке элементов на каждый кадр, без использования дополнительных структур. Абстрактный режим предполагает хранение в памяти дерева элементов, для последующей отрисовки. С помощью абстрактного режима спроектировано множество библиотек Qt [2] и GTK [3], также для проектирования библиотек может применяться смешение двух режимов Flutter [4], Jetpack Compose [5].

Для непосредственной отрисовки графических элементов используется программные интерфейсы для взаимодействия с графическим ускорителем. Наиболее распространенным вариантом является OpenGL, разработанный и поддерживаемый компанией «Khronos Group». В 2016 году эта компания опубликовала спецификацию нового программного интерфейса Vulkan, являющегося идейным наследником OpenGL. Также отдельные системы работают более производительно с другими интерфейсам: на системах с операционной системой Windows это Direct3D, а на системах,

произведенных компанией Apple, это Metal. Сравнение работы с разными интерфейсами приведены в [6]. Для наиболее эффективной работы графическая библиотека должна быть модульной и обеспечивать возможность использования наиболее эффективного программного интерфейса для каждой системы. Реализацию этого принципа можно найти в Qt [7].

С целью упрощения разработки графических приложений была разработана библиотека Skia [8]. Skia – это библиотека 2D-графики с открытым исходным кодом, которая предоставляет общие API-интерфейсы, работающие на различных аппаратных и программных платформах. Skia используется в таких проектах как: Google Chrome и ChromeOS, Android, Flutter.

Другой аспект графических приложений – их взаимодействие с операционной системой и ее графической оболочкой. Операционные системы предоставляют свои собственные интерфейсы для создания окон и отрисовки графики: Windows – WinAPI [9], MacOS – Carbon [10]. Для кроссплатформенной разработки применяются «обертки», использующие нужные интерфейсы на каждой системе. Наиболее распространенными являются: GLFW [11], SDL [12], Winit [13].

### 1.2. Анализ аналогичных проектов

Множество работ [14, 15] сравнивали и систематизировали разные библиотеки для построения графических интерфейсов. Наиболее часто встречающиеся проекты: Qt, GTK и Flutter.

Qt – это фреймворк на языке программирования C++, предназначенным для создания графических приложений для множества платформ. Многие языки программирования могут использовать преимущества Qt, благодаря наличию соответствующих библиотек. Особенность Qt заключается в использовании метаобъектного компилятора, который является предварительной системой обработки исходного кода. Кроме того, Qt расширяет возможности программиста путем предоставления системы плагинов, которые

могут быть размещены прямо на панели визуального редактора. Также есть возможность расширения функциональности виджетов, связанной с их размещением, отображением и перерисовкой при изменении размеров окна.

GTK (GNU Toolkit) – это библиотека для создания графических пользовательских интерфейсов на различных операционных системах, включая Linux, Windows и MacOS. GTK используется на многих известных проектах с открытым исходным кодом, таких как GNOME, GIMP и Inkscape. Он написан на языке программирования C, но имеет привязки к другим языкам, таким как Python, Perl и Ruby. GTK имеет ряд отличительных возможностей: он может быть использован на различных операционных системах; широкий выбор привязок к языкам программирования – GTK имеет привязки ко многим языкам программирования, таким как C, Python, Perl и Ruby; распространяется на условиях лицензии GNU LGPL, что позволяет использовать его в свободных и открытых проектах; обеспечивает поддержку CSS [16], что обеспечивает гибкость в настройке внешнего вида приложения.

Flutter – это фреймворк для разработки мобильных, веб- и настольных приложений с использованием языка программирования Dart [17]. Его отличительные черты Flutter: кроссплатформенный; основан на Skia; имеет большие возможности для гибкой настройки и расширения элементов.

Сравнение перечисленных аналогов с разрабатываемой библиотекой по выделенным критериям приведено в таблице 1.

Таблица 1 – Сравнение аналогов и разрабатываемой библиотеки

| **Критерий** | **Qt** | **GTK** | **Flutter** | **Разрабатываемая библиотека** |
|---|---|---|---|---|
| Режим работы | Абстрактный (Retained) | Абстрактный (Retained) | Смешанный (Immediate и Retained) | Смешанный |
| Платформы | Windows, MacOS, Linux, Android, IoT | Windows, MacOS, Linux | Windows, Linux, Android, IOS, IoT | Windows, MacOS, Linux |

Окончание таблицы 1

| Критерий | Qt | GTK | Flutter | Разрабатываемая библиотека |
|---|---|---|---|---|
| Язык написания | C++ | C | C++, Dart | C++ |
| Интеграции для других языков | Python, Java, Rust, Zig, Nim, Go, .NET, JavaScript | D, Go, JavaScript, Perl, Python, Rust, Vala, .NET | Python, ClojureDart | JavaScript |
| Используемый графический интерфейс | Vulkan, Metal, OpenGL, Direct3D | OpenGL, OpenGL ES, Vulkan, Metal | Skia | Skia |
| Интерфейс взаимодействия с операционной системой | WinAPI, Cocoa, Wayland, XOrg, Android, IOS | WinAPI, Cocoa, Wayland, XOrg | Android, IOS, WinAPI, GTK, Cocoa | GLFW |
| Содержит состояние логики приложения | Нет [18] | Нет | Да [19] | Нет |

**Выводы по первой главе**

В результате обзора литературы были выявлены следующие недостатки существующих решений.

Такие решения как Qt и Flutter являются комплексными, они предоставляют полный набор средств для разработки приложений. Недостатком в этом подходе является невозможность отделение отдельных компонентов и их замена в системе. Например, для управления состоянием виджетов во Flutter используется паттерн Компоненты Бизнес-Логики («Business logic components», BLoC). Данный подход неразрывно связывает код «бизнес логики» с кодом фреймворка Flutter, а также принуждает разрабатывать бизнес-логику исключительно в ООП стиле. Разрабатываемое решение должно влиять исключительно на часть кода, связанное с отображением пользовательского интерфейса.

Следующее рассмотренное решение, GTK, хоть и отвечает выдвинутому ранее требованию, имеет другой недостаток. А именно, нестабильное отображение пользовательского интерфейса на разных операционных

системах. Одно приложение, разработанное в одной ОС, будет иметь совершенно другой вид в иных операционных системах. Это связано с использованием системных элементов пользовательского интерфейса. Также это приводит к другому недостатку – невозможность низкоуровневого изменения отображения элементов, предоставляемые элементы предоставляются «как есть» и изменения многих аспектов стиля оформления может стать невозможным.

Подводя итоги анализа следует выделить, что разрабатываемая библиотека не должна оказывать влияние на другие слои приложения, а также должна предоставлять стабильное отображение на разных операционных системах, с возможностью низкоуровневого изменения отображения элементов.

Также в результате обзора литературы был выявлен набор инструментальных средств для реализации поставленной задачи, наиболее полно удовлетворяющий требованиям к подобному рода системам. Библиотека будет реализована на языке C++ с использованием графической библиотеки Skia и библиотекой для взаимодействия с операционной системой GLFW.

# 2. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОЙ СИСТЕМЕ

## 2.1. Требования к проектируемой системе

### Функциональные требования

Функциональные требования к библиотеке для разработки приложений с графическим пользовательским интерфейсом на языке C++ представлены ниже.

1. Библиотека должна предоставлять коллекцию стандартных элементов для построения графического пользовательского интерфейса. Список необходимых к реализации элементов.

2. Элементы представляемой коллекции должны проектироваться, основываясь на следующих принципах: один элемент решает одну задачу; элементы можно комбинировать друг с другом.

3. Пользователь библиотеки должен иметь возможность создать свои элементы.

### Нефункциональные требования

Нефункциональные требования к разрабатываемой библиотеке представлены ниже.

1. Библиотека должна поддерживать работу в таких операционных системах, как Windows 10/11 и Linux.

2. Библиотека должна быть написана на языке C++, с использованием версии языка – C++23 и современных стандартов разработки.

3. Отображение графики в библиотеке должна быть реализована с помощью библиотеки Skia.

4. Взаимодействие с операционной системой должно осуществляться с помощью библиотеки GLFW.

5. Архитектура библиотеки должна позволять добавлять собственные модули интеграции с операционной системой.

## 2.2. Диаграмма вариантов использования

Для проектирования приложения был использован язык графического описания для объектного моделирования UML. Была построена модель взаимодействия актера «Пользователь библиотеки» с разрабатываемой библиотекой. Диаграмма вариантов использования приведена на рисунке 1.

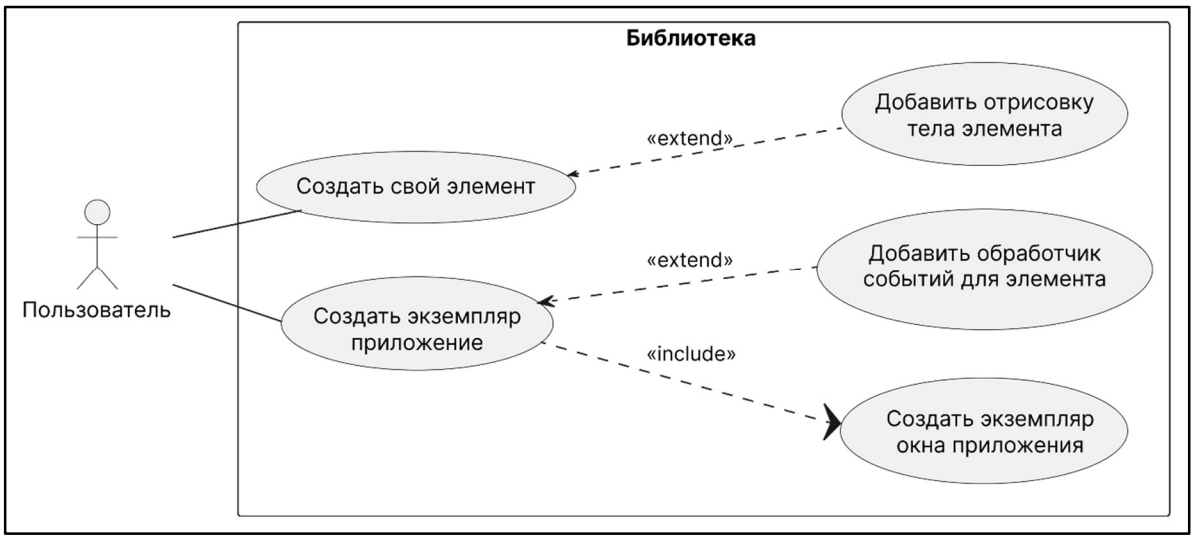


Рисунок 1 – Диаграмма вариантов использования

Элемент – единица построения графического пользовательского интерфейса. В коде библиотеки элемент представлен абстрактным классом. Пользователь библиотеки может создавать свои собственные элементы, путем определения класса, наследующегося от абстрактного класса `Element`. Разработчик изменить часть поведения элементов с помощью переопределения определенных абстрактных методов. Таким образом может быть изменено отображение графики элемента, обработка возникающих в приложении событий, а также управление дочерними элементами.

Элементы способны определять свои дочерние элементы, это позволяет быть объединять элементы с использованием композиции. Множество объединённых элементов составляют дерево элементов. Созданное дерево отображается в окнах операционной системы конечного пользователя.

В разрабатываемой библиотеке основным актером, взаимодействующим с системой, является «Пользователь библиотеки» – разработчик, использующий библиотеку для создания графического приложения.

Краткое описание вариантов использования приведено в таблице 2. Спецификация основных вариантов использования приведена в приложении А.

Таблица 2 – Описание вариантов использования

| **Имя прецедента** | **Краткое описание** |
|---|---|
| Создать элемент | Пользователь создает новый элемент – часть дерева элементов |
| Добавить отрисовку элемента | Пользователь переопределяет методы отрисовки элемента |
| Добавить обработчик событий элемента | Пользователь переопределяет методы обработки событий элемента |
| Создать экземпляр приложения | Пользователь создает приложение с заданным окном и деревом элементов |
| Создать дерево элементов | Пользователь создает дерево элементов, которое будет отображаться |
| Создать экземпляр окна приложения | Пользователь создает экземпляр окно, в котором отображается приложение |

**Выводы по второй главе**

В результате анализа требований к разрабатываемой системе был выделен ряд функциональных и нефункциональных требований. С использованием язык графического описания для объектного моделирования была разработана диаграмма вариантов использования. Были приведены варианты использования разрабатываемой библиотеки. Составлена спецификация вариантов использования, а также их краткое описание.

# 3. АРХИТЕКТУРА СИСТЕМЫ

## 3.1. Описание архитектуры системы

Архитектура библиотеки для разработки приложений с графическим пользовательским интерфейсом состоит из двух компонентов: «Ядро» и «Интеграция ядра». Классы из компонента «Ядро» взаимодействуют с операционной системой опосредованно, через классы компонента «Интеграция ядра».

На рисунке 2 представлена диаграмма компонентов разрабатываемой библиотеки.

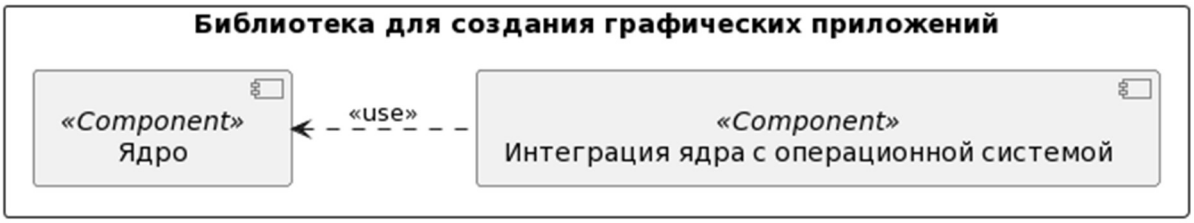


Рисунок 2 – Диаграмма компонентов

**Компонент «Ядро»**

Данный компонент включает в себя основные составляющие всей системы: Элемент, Приложение, Окно приложения и другие. Классы данного компонента не взаимодействуют напрямую с операционной системой. Составляющие компоненты позволяют создавать приложения без привязки к определенной операционной системе. Данное разделение позволяет заменять компонент интеграции для работы с другими операционными системами, без необходимости изменения классов ядра.

**Компонент «Интеграция ядра»**

Компонент включает в себя классы, реализующие взаимодействие классов и компонента ядро с операционной системой. Взаимодействие с операционной системой осуществляется с помощью библиотеки GLFW.

### 3.2. Описание реализации архитектуры системы

Для реализации выбранной архитектуры и составляющих ее модулей, были разработаны классы, изображенные на рисунке 3.

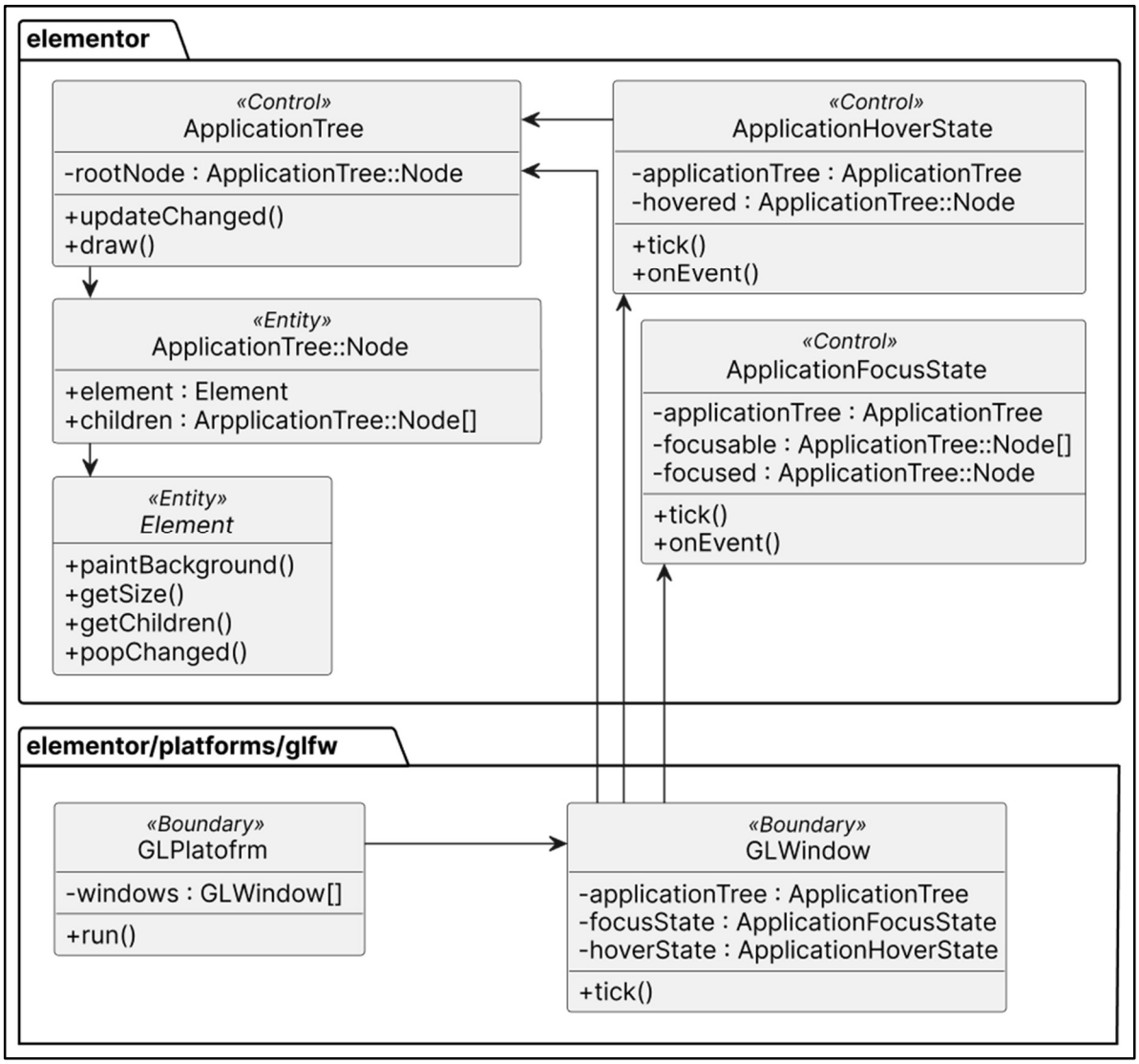


Рисунок 3 – Диаграмма классов

**Компонент «Ядро»**

Данный компонент содержит следующие классы.

1. Элемент (`Element`) – абстрактный класс, единица построения графического пользовательского интерфейса. Способен определять свой размер по заданным ограничениям, иметь потомков, отображаться на экране, обрабатывать возникающее в приложении события и уведомлять о своем изменении.

2. Составляющая дерева элементов (`ApplicationTree::Node`) – класс, содержащий ссылку на исходный элемент, список своих потомков, других составляющих дерева, кэш отображения элемента.

3. Дерево приложения (`ApplicationTree`) – класс, управляющий деревом элементов, отображаемых на экране. Содержит ссылку на главный элемент дерева. Данный класс способен отображать на экране дерево элементов, обновлять и удалять кэш измененных элементов, а также передавать события элементам.

4. Состояние фокуса приложения (`ApplicationFocusState`) – класс, содержащий ссылку на сфокусированный элемент и список элементов, которые могут быть сфокусированы. Осуществляет навигацию между сфокусированными элементами. Передает элементам события, связанные с изменением фокуса.

5. Состояние наведения приложения (`ApplicationHoverState`) – класс, содержащий ссылку на наведенный элемент. Передает элементам события, связанные с изменением текущего наведенного элемента.

**Компонент «Интеграция ядра»**

Данный компонент содержит следующие классы.

1. Платформа GLFW (`GLPlatform`) –класс, отвечающий за взаимодействие с операционной системой, посредством библиотеки GLFW, и отображение активных окон.

2. Окно приложения GLFW (`GLWindow`) – класс, отвечающий за взаимодействие с окна операционной системы, с помощью GLFW.

**Выводы по третьей главе**

В результате проектирования архитектура была разработана архитектура библиотеки, удовлетворяющая приведенным требованиям. Архитектура состоит из нескольких связанных модулей, каждый из которых содержит приведенные классы. Данное разделение облегчает поддержку кодовой базы, а также внесение изменений.

# 4. РЕАЛИЗАЦИЯ СИСТЕМЫ

## 4.1. Реализация компонентов системы

Разрабатываемая система состоит из двух компонентов: ядра и интеграции ядра с операционной системой.

### 4.1.1. Реализация компонента «Ядро»

Ядро библиотеки представляет собой набор классов, предназначенных для создания графических приложений. Он содержит классы для создания окон, элементов управления и обработки событий. В ядре реализован паттерн проектирования «Компоновщик», который позволяет организовывать элементы управления в древовидную структуру.

Все объекты данного компонента находятся в пространстве имен «elementor» и доступны всем дочерним пространствам имен.

В ходе разработки компонента были реализованы следующие объекты.

**Класс ApplicationTree**

Класс, содержащий дерево элементов. Он осуществляет поиск по дереву, отрисовку и управление кэшем отрисовки.

Отображение дерева элементов производится с помощью рекурсивного обхода дерева элементов. Реализация данного метода приведена в листинге 1.

Листинг 1 – Отображение дерева элементов в классе ApplicationTree

```
void ApplicationTree::Node::draw(SkCanvas* canvas, bool withChildrenCache){
  canvas->save();
  canvas->translate(rect.inParentPosition.x, rect.inParentPosition.y);
  clipCanvas(canvas);
  element->paintBackground(canvas, rect);
  for (const auto& child: children) {
    if (withChildrenCache) child->drawWithCache(canvas);
    else child->draw(canvas, false);
  }
  canvas->restore();
}

void ApplicationTree::Node::drawToCache(SkCanvas* canvas) {
  auto cacheSurface = canvas->makeSurface(canvas->imageInfo());
  auto cacheCanvas = cacheSurface->getCanvas();
  cacheCanvas->setMatrix(canvas->getTotalMatrix());
  draw(cacheCanvas, false);
```

```
  drawCachedImage = cacheSurface->makeImageSnapshot();
  drawCache(canvas);
}

void ApplicationTree::Node::drawCache(SkCanvas* canvas) {
  canvas->save();
  canvas->translate(rect.inParentPosition.x, rect.inParentPosition.y);
  clipCanvas(canvas);
  canvas->resetMatrix();
  canvas->drawImage(drawCachedImage, 0, 0);
  canvas->restore();
}

void ApplicationTree::Node::drawWithCache(SkCanvas* canvas) {
  if (drawCachedImage) drawCache(canvas);
  else if (!childrenCached) draw(canvas, true);
  else if (beforeCache > 0) { beforeCache -=1;draw(canvas, false);}
  else drawToCache(canvas);
}

void ApplicationTree::draw(SkCanvas* canvas) {
  root->drawWithCache(canvas);
}
```

**Класс ApplicationHoverState**

Класс, хранящий логику, касающуюся наведения курсора. Он хранит ссылку на активный элемент и его родителей, обеспечивает их обновление в зависимости от действий пользователя, а также передает необходимые события элементам.

**Класс ApplicationFocusState**

Класс, хранящий логику, касающуюся сфокусированного элемента. Он хранит ссылку на сфокусированный элемент, обеспечивает его обновление в зависимости от действий пользователя, а также передает необходимые события элементам.

Для более быстрого взаимодействия с интерфейсом, реализовано перемещение между элементами с помощью клавиатуры. Если пользователь сфокусирован на элементе и нажимает кнопку Tab, то он фокусируется на следующий элемент и обратное действие происходит при нажатии Shift+Tab. Реализация данного поведения приведена в листинге 2.

Листинг 2 – Обработка событий в классе `ApplicationFocusState`

```
EventCallbackResponse ApplicationFocusState::onEvent(const
std::shared_ptr<Event>& event) {
  if (focusedNode && isFocusRelatedEvent(event)) {
```

```
    auto bubbleEventResponse = focusedNode->bubbleEvent(event);
    if (bubbleEventResponse == EventCallbackResponse::StopPropagation) {
      return EventCallbackResponse::StopPropagation;
    }
  }

  if (isGoToNextFocusableEvent(event)) {
    focusNextNode();
  } else if (isGoToPreviousFocusableEvent(event)) {
    focusPreviousNode();
  }

  return EventCallbackResponse::None;
}

bool isGoToNextFocusableEvent(const std::shared_ptr<Event>& event) {
  auto keyboardEvent = std::dynamic_pointer_cast<KeyboardEvent>(event);
  return keyboardEvent != nullptr
     && (keyboardEvent->action == KeyAction::Press
      || keyboardEvent->action == KeyAction::Repeat)
     && !(keyboardEvent->mods & KeyModsShift)
     && keyboardEvent->key == KeyboardKey::Tab;
}

bool isGoToPreviousFocusableEvent(const std::shared_ptr<Event>& event) {
  auto keyboardEvent = std::dynamic_pointer_cast<KeyboardEvent>(event);
  return keyboardEvent != nullptr
     && (keyboardEvent->action == KeyAction::Press
      || keyboardEvent->action == KeyAction::Repeat)
     && (keyboardEvent->mods & KeyModsShift)
     && keyboardEvent->key == KeyboardKey::Tab;
}
```

**Класс ApplicationContext**

Абстрактный класс, посредством которого элементы взаимодействуют с приложением и операционной системой, например: получить язык системы пользователя, доступ к буферу обмена, информации об экране и т.д.

**Класс Element**

Абстрактный класс `Element` представляет собой единицу построения графического пользовательского интерфейса. Способен отображаться на экране и обрабатывать возникающее в приложении события.

### 4.1.2. Реализация компонента «Интеграция ядра»

Интеграция ядра с операционной системой обеспечивает взаимодействие ядра с оконной системой операционной системы. Этот компонент

обеспечивает доступ ядра к системным функциям, таким как создание и управление окнами, обработку ввода и вывода графики.

Все объекты данного компонента находятся в пространстве имен «elementor::platforms::gl».

В ходе разработки компонента были реализованы следующие объекты.

**Класс GLPlatform**

Класс, управляющий окнами приложения, обеспечивает их отображение. Также предоставляет доступ к ресурсам операционной системы не ограниченным одним определенным окном, например: буфер обмена, язык операционной системы, метрики производительности отрисовки. Данный класс взаимодействует с операционной системой с помощью библиотеки GLFW.

**Класс GLWindow**

Класс, наследующийся от класса окна приложения. Хранит ссылку на объекты окна и монитора, предоставляемые библиотекой GLFW. Данный класс взаимодействует с операционной системой с помощью библиотеки GLFW.

Конструктор класса `GLWindow`, содержащий инициализацию GLFW окна и создание обработчиков событий приведен в листинге 3.

Листинг 3 – Конструктор класса `GLWindow`

```
GLWindow::GLWindow(const std::shared_ptr<GLPlatformContext>& ctx)
  : ctx(ctx) {
  glWindow = glfwCreateWindow(1, 1, "Elementor", nullptr, nullptr);
  glfwMakeContextCurrent(glWindow);
  glfwSwapInterval(1);
  glfwSetWindowUserPointer(glWindow, this);
  glfwSetWindowRefreshCallback(glWindow, onWindowRefresh);
  glfwSetWindowPosCallback(glWindow, onWindowPosition);
  glfwSetWindowCloseCallback(glWindow, onWindowClose);
  glfwSetWindowFocusCallback(glWindow, onSetWindowFocus);
  glfwSetKeyCallback(glWindow, onWindowKey);
  glfwSetCharCallback(glWindow, onWindowChar);
  glfwSetMouseButtonCallback(glWindow, onWindowMouseButton);
  glfwSetCursorPosCallback(glWindow, onWindowCursorPosition);
  glfwSetScrollCallback(glWindow, onWindowScroll);
  updateMonitor();
  cursor = std::make_shared<GLCursor>(glWindow);
  skContext = GrDirectContext::MakeGL(GrGLMakeNativeInterface()).release();
}
```

### 4.2. Реализация интерфейса системы

Разрабатываемая библиотека предоставляет разработчикам удобный интерфейс для создания графических элементов, таких как кнопки, текстовые поля и т. д.

Интерфейс разрабатываемой системы предназначен для обеспечения взаимодействия с пользователем и управления элементами графического интерфейса. Он включает в себя классы для создания и управления окнами, элементами интерфейса и событиями.

Для реализации интерфейса системы используется объектно-ориентированный подход, что позволяет легко создавать и управлять элементами интерфейса. Классы взаимодействуют между собой с помощью интерфейсов и наследования, что обеспечивает удобную расширяемость и модульность системы.

Для разработки графических приложений с использованием разрабатываемой системы, разработчикам необходимо знать основные классы и методы, предоставляемые интерфейсом, и уметь создавать свои собственные элементы интерфейса на основе существующих классов.

#### 4.2.1. Реализация элемента текста

Главным графическим элементом приложения является текст. Для отображения текста был разработан элемент «Text». У данного элемента могут заданы различные атрибуты: содержание, стили оформления и локализация. Используя значения данных параметров, происходит отображение текста на экране. Код отображения текста приведен в листинге 4.

Листинг 4 – Отображение текста

```
SkFont Text::makeSkFont() const {
  float pixelScale = ctx->getPixelScale();
  float fontSizeScaled = fontSize * pixelScale;

  SkFont skFont;
  skFont.setTypeface(makeSkTypeface());
  skFont.setSize(fontSizeScaled);
  skFont.setScaleX(fontScale);
  skFont.setSkewX(fontSkew);
```

```
  skFont.setEdging(getSkFontEdging());
  return skFont;
}

SkPaint Text::makeSkPaint() const {
  SkPaint skPaint;
  skPaint.setColor(getFontColor());
  return skPaint;
}

void Text::paintBackground(SkCanvas* canvas, const ElementRect& rect) {
  float pixelScale = ctx->getPixelScale();

  if (!font.has_value() || pixelScale != lastPixelScale) font =
makeSkFont();
  if (!paint.has_value()) paint = makeSkPaint();
  lastPixelScale = pixelScale;

  canvas->drawString(text.data(), 0, rect.size.height, font.value(),
paint.value());
}
```

Элемент текст позволяет отображать строчку текста в одном оформлении. Элемент параграф позволяет объединять несколько элементов текста, сохраняя стили каждого из элементов. У каждого фрагмента текста запрашивается его оформление, код формирования структуры данных, определяющих оформление текста, приведен в листинге 5.

Листинг 5 – Создания структуры оформления текста

```
skia::textlayout::TextStyle Text::makeSkTextStyle() const {
  float pixelScale = ctx->getPixelScale();
  float fontSizeScaled = fontSize * pixelScale;

  auto defaultLocale = ctx->getLocale();
  auto localeOrDefault = locale.value_or(defaultLocale);

  skia::textlayout::TextStyle textStyle;
  textStyle.setFontSize(fontSizeScaled);
  textStyle.setFontFamilies({ SkString(fontFamily) });
  textStyle.setLocale(SkString(localeOrDefault));
  textStyle.setForegroundColor(makeSkPaint());
  textStyle.setFontStyle(makeSkFontStyle());
  textStyle.setDecoration(getSkTextDecoration());
  textStyle.setDecorationMode(getSkTextDecorationMode());
  textStyle.setDecorationStyle(getSkTextDecorationStyle());
  textStyle.setDecorationColor(getDecorationColor());
  textStyle.setDecorationThickness (getDecorationThickness ());

  return textStyle;
}
```

Полученные структуры оформления текста используются при создании объекта параграфа и его дальнейшем отображении. Код создания объекта параграфа приведен в листинге 6.

Листинг 6 – Создания объекта параграфа

```
std::unique_ptr <skia::textlayout::Paragraph>
Paragraph::makeSkParagraph() const {
  skia::textlayout::ParagraphBuilderImpl builder = makeBuilder(ctx);

  for (const std::shared_ptr <Element>& child: children) {
    auto textChild = std::dynamic_pointer_cast<Text>(child);
    if (textChild) {
      auto textChildValue = textChild->getText();
      builder.pushStyle(textChild->makeSkTextStyle());
      builder.addText(textChildValue.data(), textChildValue.size());
      builder.pop();
    } else {
      builder.addPlaceholder(makeChildPlaceholderStyle(child));
    }
  }

  std::unique_ptr <skia::textlayout::Paragraph> paragraph =
    builder.Build();
  paragraph->updateTextAlign(getSkTextAlign());

  return paragraph;
}
```

### 4.2.2. Реализация элемента кнопки

Другим важным графическим элементом приложения является кнопка. Для отображения текста был разработан элемент «Button».

Данные элемент отличается от рассмотренных ранее, он определяется с использованием композиции других элементов. Все отдельные свойства кнопки представляют собой отдельные элементы, каждый из которых выполняют свою отдельную задачу. Например: обработку нажатия осуществляет элемент – «Clickable», наведения курсора – «Hoverable», отображение заливки – «Background», обработка фокуса кнопки и событий, возникающих при фокусе – «Focusable». Подобное разбиение функциональности уменьшает связность кода, так как каждое отдельное свойство разрабатывается отдельно. Также это позволяет использовать отдельные свойства в отрыве от других, создавая новую функциональность.

Композиция элементов происходит в конструкторе элемента. Исходный код конструктора элемента «Button» приведен в листинге 7.

Листинг 7 – Конструктор элемента «Button»

```
explicit Button(
  const std::shared_ptr<ApplicationContext>& ctx,
  const ButtonProps& props
) {
  element = Outline::New(ctx, {
    .border = {
      radius = 6,
      .width = 3,
      .color = props.outlineColor,
      .style = BorderStyle::Dashed
    },
    .offset = 6,
    .child = Focusable::New(ctx, focusable, {
      .child = Cursorable::New(ctx, {
        .cursorShape = CursorShape::Hand,
        .child = ClickableOutside::New(ctx, {
          .onClickOutside = [this]() {
            blur();
          },
          .child = Clickable::New(ctx, {
            .onClick = [this](KeyMod _) {
              focus();
            },
            .child = Rounded::New(ctx, {
            .all = 4,
              .child = Background::New(ctx, {
                .color = props.backgroundColor,
                .child = Padding::New(ctx, {
                  .all = 8,
                  .child = Text::New(ctx, text, {
                    .text = props.text,
                    .fontColor = props.fontColor,
                    .fontSize = 16,
                    .fontFamily = "Arial",
                  }) }) }) }) }) }) }) }) }); }
```

**Выводы по четвертой главе**

В результате реализации системы была разработана программная реализация модулей библиотеки. Была произведена разработка компонентов модулей «Ядро» и «Интеграция ядра», а также основных элементов библиотеки текст и кнопка.

# 5. ТЕСТИРОВАНИЕ СИСТЕМЫ

Для тестирования разрабатываемой библиотеки на языке C++ можно использовать различные методы тестирования, такие как автоматизированное -тестирование и ручное тестирование.

## 5.1. Автоматизированное тестирование

Алгоритм автоматизированного тестирования состоит из нескольких шагов: запуск тестового приложения, создание снимка окна приложение, сравнение этого снимка с эталонным результатом. При расхождении эталонного и текущего результата тест считается проваленным. Код для создания снимков приложений приведен в листинге 8.

Листинг 8 – Создание снимков приложений

```
int main() {
  auto platform = std::make_shared<GLPlatform>();

  auto window = std::make_shared<GLWindow>(platform);
  platform->addWindow(window);

  for (const auto& example: Examples) {
    window->setSize(example.size);
    window->setRoot(example.make(window));
    platform->__T_tick(2);
    saveImageToFile(
      window->__T_screenshot(),
      "../../../../tests/screenshots_new/" + example.name + ".png"
    );
  }

  window->close();
}
```

Созданные снимки сравниваются эталонными и определяется их расхождение. Если расхождение больше порогового значения, то тест считается проваленным. Результат сравнения также сохраняется в визуальном формате. Код для сохранения расхождения в формате изображения на языке Python приведен в листинге 9.

Листинг 9 – Сохранение расхождения изображений

```
def image_with_diff(img, diff):
  diff = ImageOps.invert(diff.convert('L'))

  img_darken = ImageEnhance.Brightness(img)
```

```
  img_darken = img_darken.enhance(0.64)

  red = Image.new('RGBA', img.size, color='#F00')
  img_diffed = Image.composite(img_darken, red, diff)
  return img_diffed
old = Image.open(path.join(old_path, screenshot))
new = Image.open(path.join(new_path, screenshot))
diff = ImageChops.difference(old, new)
image_with_diff(old, diff).save(path.join(diff_path, screenshot))
```

Пример расхождения изображений приведен на рисунке 4. В данном примере был изменен шрифт текста, из-за снимки экрана не совпали и это было отображено в изображении расхождения.

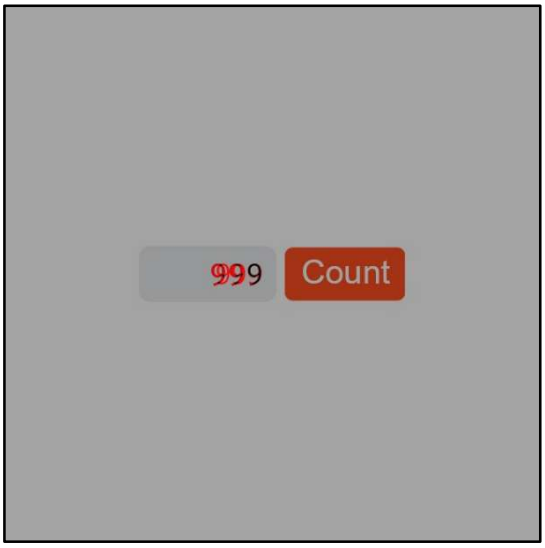


Рисунок 4 – Расхождения в тесте «Счетчик»

Полный набор разработанных автоматизированных тестов приведен в таблице 3.

Таблица 3 – Автоматизированное тестирование системы

| № | Название теста | Процент расхождения | Тест пройден? |
|---|---|---|---|
| 1 | Основной | 0% | Да |
| 2 | Счетчик | 0% | Да |
| 3 | CRUD | 0% | Да |
| 4 | Конвертер температуры | 0% | Да |
| 5 | Список дел | 0% | Да |

## 5.2. Ручное тестирование

Процесс ручного тестирования разработанной библиотеки представлен в таблице 4.

Таблица 4 – Ручное тестирование системы

| № | Название теста | Шаги | Ожидаемый результат | Тест пройден? |
|---|---|---|---|---|
| 1 | Создание платформы | Создать экземпляр `GLPlatform` | `GLPatform` создан | Да |
| 2 | Создание экземпляра элемента | 1. Выбрать класс элемент, наследующегося от класса `Element`<br>2. Создать его экземпляр | Экземпляр `Element` создан | Да |
| 3 | Обработка событий | 1. Создать экземпляр элемента Кнопка<br>2. Навести курсор на данный элемент<br>3. Убрать курсор с элемента | Элемент изменил свой цвет, а потом вернулась в обычное состояние | Да |
| 4 | Очистка элементов из памяти | 1. Открыть приложение галерею примеров<br>2. Открыть одну страницу<br>3. Перейти на другую страницу | После перехода на вторую страницу память предыдущей страницы была освобождена | Да |
| 5 | Создание нового элемента | 1. Создать файл класса<br>2. Написать класс, наследующийся от класса `Element`<br>3. Переопределить нужные методы | Создан класс элемента. Элемент отображается без ошибок | Да |

### Выводы по пятой главе

В результате тестирования системы были разработаны процессы для автоматического и ручного тестирования библиотеки. Разработанные тесты покрывают большинство функционала системы и облегчают исправление кода и разработку нового функционала.

# 6. СОЗДАНИЕ ДЕМОНСТРАЦИОННЫХ ПРИЛОЖЕНИЙ

Для облегченного тестирования визуальных элементов при разработке системы, а также после его завершения, было разработаны демонстрационные приложения. Они содержат примеры использования реализованных элементов, а также примеры их совместного использования. Все демонстрационные приложения выполнены с использованием разрабатываемой библиотеки. Всего было реализовано 6 демонстрационных приложений.

## 6.1. Приложение «Список дел»

Данное приложение состоит из формы создания задачи и списка задач. Форма создания задачи состоит из текстового поля и кнопки «Добавить». Задачи могут отредактированы, отмечены как выполненные и удалены. Выполненные задачи не могут быть отредактированы. Данные задач синхронизируются с заданным текстовым файлом в формате Markdown [22].

Внешний вид разработанного приложения приведен в приложении Б.

В листинге 10 приведен фрагмент кода приложения, код конструктора формы создания задачи.

Листинг 10 – Конструктор формы создания задачи

```
MakeTodoForm(const std::shared_ptr<ApplicationContext>& ctx, const Props&
props): Component(ctx) {
  submitCallback = props.onSubmit;
  element = Flex::New(ctx, {
     .spacing = 10,
     .alignment = FlexAlignment::Center,
       .children = {
         Flexible::New(ctx, {
           .child = TextInput::New(ctx, textInput, {
             .placeholder = U"Add item...",
             .onSubmit = [this](const std::u32string& newValue) {
               onSubmit();
             }
           }),
         }),
         IconButton::New(ctx, {
           .src = asset("add.svg"),
           .onClick = [this]() {
             onSubmit();
             return EventCallbackResponse::None;
           }
       })
     }}
   });}
```

### 6.2. Приложения «7GUIs Tasks»

7GUIs [23] – это бенчмарк для GUI библиотек. Он содержит ряд рекомендаций, а также список из 7 приложений [24], на примере которых можно сравнивать различные библиотеки. В рамках данной работы были реализованы 3 из 7 приложений.

Внешний вид реализованных приложений приведен в приложении Б.

В листинге 8 приведен фрагмент кода приложения «Счетчик», конструктор класса элемента «Counter».

Листинг 11 – Конструктор класса элемента «Counter»

```
explicit Counter(const std::shared_ptr<ApplicationContext>& ctx) {
  element = Background::New(ctx, {
    .color = "#fff",
    .child = Flex::New(ctx, {
    .spacing = 4,
    .alignment = FlexAlignment::Center,
    .crossAlignment = FlexCrossAlignment::Center,
    .children = {
      Rounded::New(ctx, {
        .all = 6,
        .child = Background::New(ctx, {
          .color = "#ebf0f0",
          .child = Padding::New(ctx, {
            .all = 8,
            .child = Width::New(ctx, {
              .width = 50,
              .child = Align::New(ctx, {
                .width = { 1, 1 },
                .child = Text::New(ctx, countText, {
                  .text = std::to_string(count),
                  .fontSize = 14,
                  .fontFamily = "Fira Code"
                })
              })
            })
          })
        })
      }),
      TextButton::New(ctx, {
        .text = "Count",
        .fontColor = "#fff",
        .backgroundColor = "#ff5020",
        .onClick = [this] {
          incCount();
        }
      })
    }
  });
}
```

В листинге 9 приведен фрагмент кода приложения «Конвертер температуры», конструктор класса элемента «TempConv».

Листинг 9 – Конструктор класса элемента «TempConv»

```
explicit TempConv(const std::shared_ptr<ApplicationContext>& ctx) {
  element = Background::New(ctx, {
    .color = "#fff",
    .child = Flex::New(ctx, {
      .spacing = 12,
      .alignment = FlexAlignment::Center,
      .crossAlignment = FlexCrossAlignment::Center,
        .children = {
          TextInput::New(ctx, {
            .onInput = [this](const std::u32string& value) {
              setTempC(parseFloat(value));
            }
          }),
          Text::New(ctx, {
            .text = "Celsius = "
          }),
          TextInput::New(ctx, {
            .onInput = [this](const std::u32string& value) {
              setTempF(parseFloat(value));
            }
          }),
          Text::New(ctx, {
            .text = "Fahrenheit"
          }),
        }
      })
    });
}
```

**Выводы по шестой главе**

В рамках данной главы разработаны три приложения, на примере которых может быть продемонстрировано большинство функционала библиотеки и его корректная работа.

# 7. ПУБЛИКАЦИЯ БИБЛИОТЕКИ

Размещение репозитория на платформе GitHub [25] является неотъемлемой частью современной открытой разработки. Причины размещения библиотеки на данной платформе представлены ниже.

1. Размещенные на GitHub репозитории сохраняются на множестве серверов по всему миру. Это уменьшает риски потери исходного кода.

2. Опубликованные в открытый доступ репозитории на платформе GitHub доступны разработчикам со всего мира.

3. Такие возможности GitHub как Issues [26] и Pull Requests [27], позволяют совместную разработку со множеством участников со всего мира.

Исходный код библиотеки на платформе GitHub был размещен по адресу «https://github.com/noartem/elementor». Также вместе с исходным кодом были размещены демонстрационные приложения и документация.

На рисунке 5 представлен скриншот страницы репозитория библиотеки.

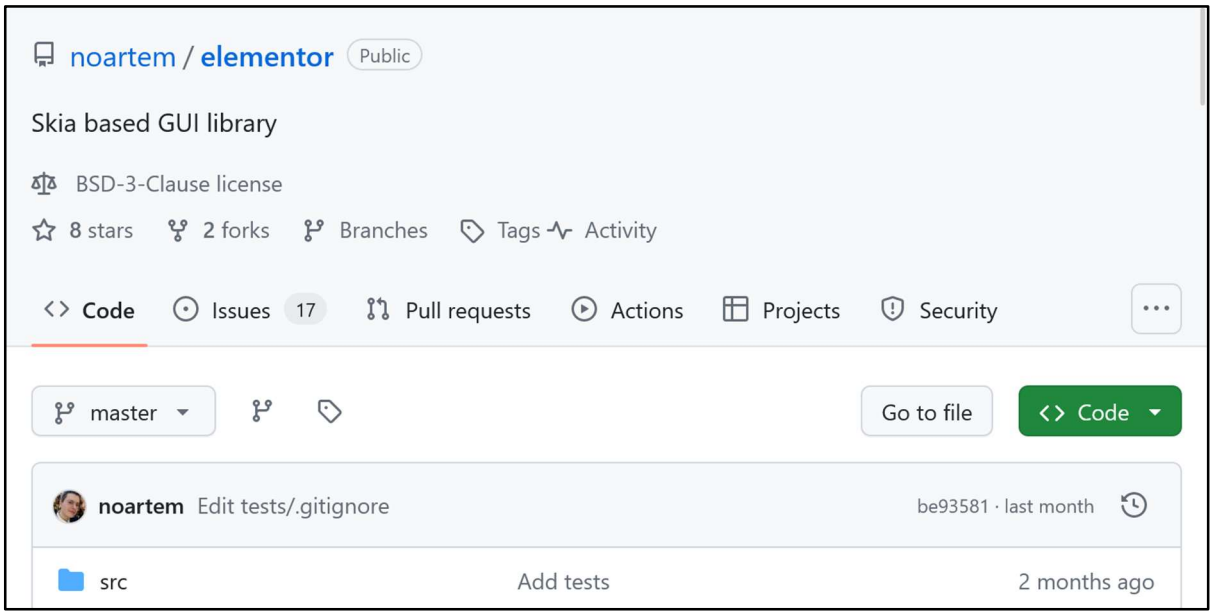


Рисунок 5 – Страница библиотеки на GitHub

**Выводы по седьмой главе**

В рамках данной главы библиотека была опубликована в общий доступ и теперь может использоваться в сторонних проектах.

## ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана библиотека для создания графических пользовательских интерфейсов на языке C++. При этом были решены следующие задачи.

1. Произведен анализ предметной области, обзор существующих работ по данной тематике и обзор аналогичных проектов.
2. Приведены требования к разрабатываемой библиотеке.
3. Выполнено проектирование архитектуры библиотеки и ее реализация.
4. Осуществлено тестирование библиотеки.
5. Разработаны демонстрационные приложения.
6. Библиотека и приложения опубликованы на GitHub.

Разработанная библиотека используется для создания графических пользовательских приложений, упрощает разработку приложений и объединяет преимущества существующих решений (Qt, GTK и Flutter). Корректность и работоспособность библиотеки подтверждается проведенными тестами. Также библиотека содержит примеры использования и демонстрационные приложения.

В дальнейшем планируется расширение функционала библиотеки, добавление большего числа примеров, а также написание документации.

**ЛИТЕРАТУРА**

1. Retained Mode Versus Immediate Mode | Microsoft Learn. [Электронный ресурс] URL: https://learn.microsoft.com/en-us/windows/win32/learnwin32/retained-mode-versus-immediate-mode (дата обращения: 12.02.2024 г.).

2. Qt Framework | Qt. [Электронный ресурс] URL: https://www.qt.io/product/framework (дата обращения: 12.02.2024 г.).

3. Docs | The GTK Project. [Электронный ресурс] URL: https://www.gtk.org/docs/ (дата обращения: 12.02.2024 г.).

4. Flutter documentation | Flutter. [Электронный ресурс] URL: https://docs.flutter.dev/ (дата обращения: 12.02.2024 г.).

5. Официальный сайт проекта Jetpack Compose. [Электронный ресурс] URL: https://developer.android.com/jetpack/compose (дата обращения: 12.02.2024 г.).

6. A Comparison of Modern Graphics APIs | Alain Galvan Blog. [Электронный ресурс] URL: https://alain.xyz/blog/comparison-of-modern-graphics-apis (дата обращения: 12.02.2024 г.).

7. Qt Quick on Vulkan, Metal, and Direct3D | Qt Blog. [Электронный ресурс] URL: https://www.qt.io/blog/qt-quick-on-vulkan-metal-direct3d (дата обращения: 12.02.2024 г.).

8. Welcome to Skia: The 2D Graphics Library | Skia. [Электронный ресурс] URL: https://skia.org/ (дата обращения: 12.02.2024 г.).

9. Programming reference for the Win32 API | Microsoft Learn. [Электронный ресурс] URL: https://learn.microsoft.com/en-us/windows/win32/api/ (дата обращения: 12.02.2024 г.).

10. Apple Developer | Carbon Core. [Электронный ресурс] URL: https://developer.apple.com/documentation/coreservices/carbon_core (дата обращения: 12.02.2024 г.).

11. An OpenGL Library // GLFW. [Электронный ресурс] URL: https://www.glfw.org/ (дата обращения: 12.02.2024 г.).

12. About SDL | Simple DirectMedia Layer. [Электронный ресурс] URL: https://www.libsdl.org/ (дата обращения: 12.02.2024 г.).

13. Winit: Window handling library in pure Rust | GitHub. [Электронный ресурс] URL: https://github.com/rust-windowing/winit (дата обращения: 12.02.2024 г.).

14. Грузин Н.А., Голубничий А.А. Обзор и сравнение библиотек пользовательских интерфейсов: gtk, Qt, wxWidgets. // E-Scio. 2020. – №2(41). – С. 23–28.

15. Викулина Д.А., Макаров С.Н., Кунгурцева К.В., Гаранина Е.А. Современные технологии создания desktop-приложений. // Наука и современность, 2012. –№ 18. – С. 51–54.

16. CSS | MDN Web Docs. [Электронный ресурс] URL: https://developer.mozilla.org/en-US/docs/Glossary/CSS (дата обращения: 12.02.2024 г.).

17. Dart programming language | Dart. [Электронный ресурс] URL: https://dart.dev/ (дата обращения: 12.02.2024 г.).

18. Separate UI from Logic | Qt Documentation. [Электронный ресурс] URL: https://doc.qt.io/qt-5/qtquick-bestpractices.html#separate-ui-from-logic (дата обращения: 12.02.2024 г.).

19. State management | Flutter documentation. [Электронный ресурс] URL: https://docs.flutter.dev/development/data-and-backend/state-mgmt/intro (дата обращения: 12.02.2024 г.).

20. Doctest | GitHub. [Электронный ресурс] URL: https://github.com/doctest/doctest (дата обращения: 12.02.2024 г.).

21. CSS Flexbox | MDN Web Docs. [Электронный ресурс] URL: https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox (дата обращения: 12.02.2024 г.).

22. Markdown | Daring Fireball. [Электронный ресурс] URL: https://daringfireball.net/projects/markdown (дата обращения: 18.02.2024 г.).

23. 7GUIs: GUI Programming Benchmark | 7GUIs. [Электронный ресурс] URL: https://eugenkiss.github.io/7guis (дата обращения: 18.02.2024 г.).

24. The 7 Tasks | 7GUIs. [Электронный ресурс] URL: https://eugenkiss.github.io/7guis/tasks (дата обращения: 18.02.2024 г.).

25. Elementor: Skia based GUI library | GitHub. [Электронный ресурс] URL: https://github.com/noartem/elementor (дата обращения: 18.02.2024 г.).

26. GitHub Issues. Project planning for developers | GitHub. [Электронный ресурс] URL: https://github.com/features/issues (дата обращения: 18.02.2024 г.).

27. Features. Code review | GitHub. [Электронный ресурс] URL: https://github.com/features/code-review (дата обращения: 18.02.2024 г.).

# ПРИЛОЖЕНИЯ

## Приложение А. Спецификация вариантов использования

Спецификация вариантов использования (ВИ) системы приведена в таблицах 1–4.

Таблица 1 – Спецификация ВИ «Создать элемент»

| |
|---|
| Прецедент: Создать элемент |
| ID: 1 |
| Краткое описание:<br>Пользователь создает новый элемент |
| Главные актеры:<br>Пользователь библиотеки |
| Второстепенные актеры:<br>Нет |
| Предусловия:<br>Отсутствуют |
| Основной поток:<br>1. Пользователь создает новый класс, наследуемый от абстрактного класса «Element».<br>2. Пользователь может добавить переопределения для методов класса «Element». |
| Постусловия:<br>Был создан новый элемент, экземпляры которого будут частью дерева элементов. |
| Альтернативные потоки:<br>I. Добавить отрисовку элемента (ID: 1.1)<br>II. Добавить обработчик событий (ID: 1.2) |

Таблица 2 – Спецификация ВИ «Добавить отрисовку элемента»

| |
|---|
| Прецедент: Добавить отрисовку элемента |
| ID: 1.1 |
| Краткое описание:<br>Пользователь добавляет отрисовку к созданному элементу |
| Главные актеры:<br>Пользователь библиотеки |
| Второстепенные актеры:<br>Нет |
| Предусловия:<br>Пользователь создал свой элемент |
| Основной поток.<br>1. Пользователь переопределяет метод для отрисовки графической части элемента.<br>2. Пользователь переопределяет метод возвращающий список элементов потомков. |
| Постусловия:<br>1. Переопределенные методы отрисовки элемента будут вызываться при отрисовке дерева элементов. |
| Альтернативные потоки:<br>Отсутствуют |

Таблица 3 – Спецификация ВИ «Добавить обработчик событий»

| Прецедент: Добавить обработчик событий |
|---|
| ID: 1.2 |
| Краткое описание:<br>Пользователь добавляет обработку событий для созданного элемента. |
| Главные актеры:<br>Пользователь библиотеки |
| Второстепенные актеры:<br>Отсутствуют |
| Предусловия:<br>1. Пользователь создал свой элемент |
| Основной поток:<br>1. К списку наследуемых классов элемента пользователь добавляет класс «WithEventsHandlers», содержащий метод, возвращающий обработчики событий.<br>2. Пользователь переопределяет метод обработки события и добавляет свою реализацию, включающую передачу обработчика для необходимого события. |
| Постусловия:<br>1. Переопределенный методы обработки события будет вызываться при возникновении события в приложении. |
| Альтернативные потоки:<br>Отсутствуют |

Таблица 4 – Спецификация ВИ «Создать экземпляр приложения»

| Прецедент: Создать экземпляр приложения |
|---|
| ID: 2 |
| Краткое описание:<br>Пользователь создает экземпляр приложения. |
| Главные актеры:<br>Пользователь библиотеки |
| Второстепенные актеры:<br>Отсутствуют |
| Предусловия:<br>Отсутствует |
| Основной поток:<br>1. Пользователь, пользуясь доступными элементами, объединяет их в дерево элементов с помощью композиции.<br>2. Пользователь создает экземпляр класса «GlPlatform».<br>3. Пользователь создает экземпляр класса «GlWindow», задает необходимые параметры: имя окна, начальный размер, минимальный и максимальный размер, ссылку на дерево элементов.<br>4. Пользователь добавляет экземпляр класса «GlWindow» к экземпляру класса «GlPlatform» с помощью метода «addWindow» |
| Постусловия:<br>1. Был создан экземпляр приложения |
| Альтернативные потоки:<br>Отсутствуют |

**Приложение Б. Скриншоты демонстрационных приложений**

Скриншоты демонстрационного приложения «Список дел» приведены на рисунках 1–2.

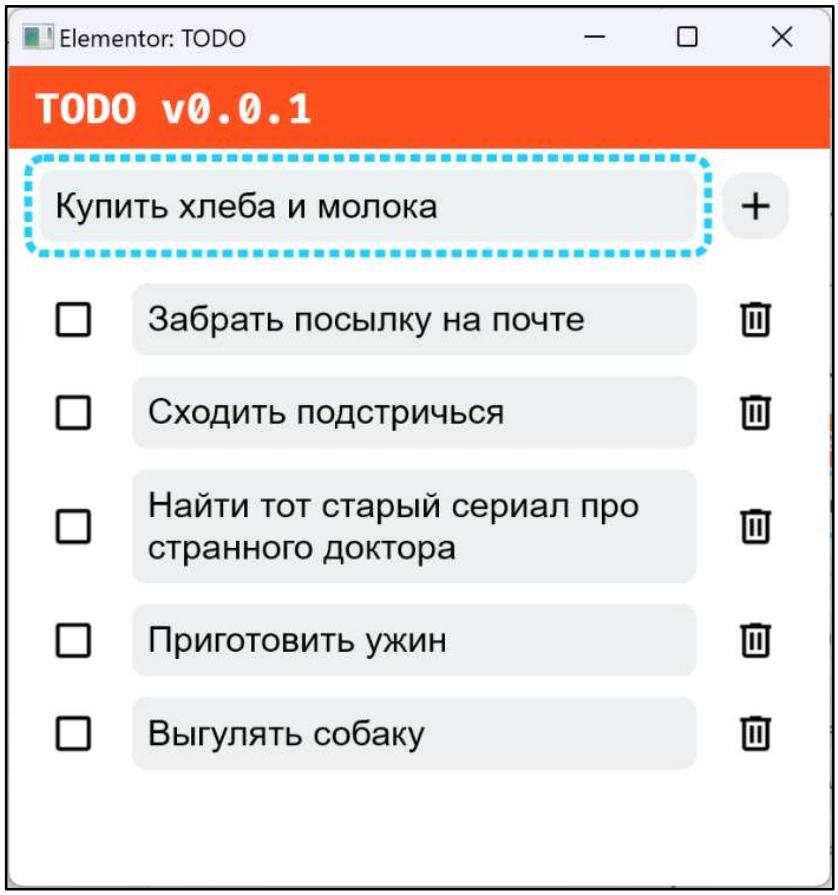


Рисунок 1 – Использование формы создания задачи

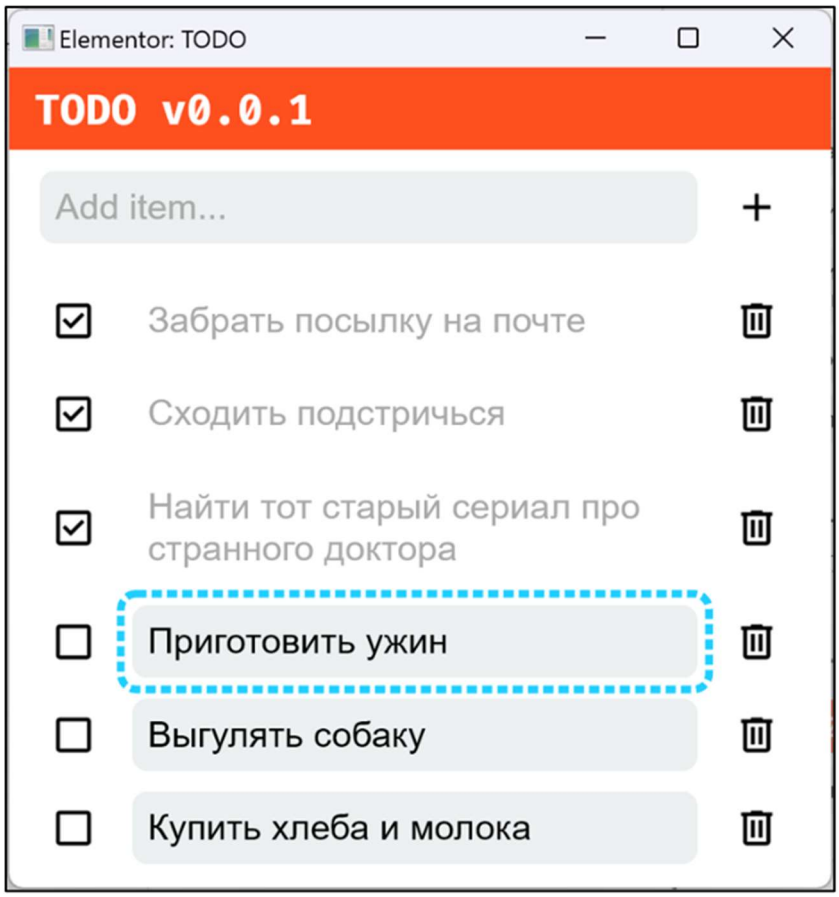


Рисунок 2 – Список задач

Скриншоты демонстрационных приложений из списка 7GUIs Tasks приведены на рисунках 3–5.

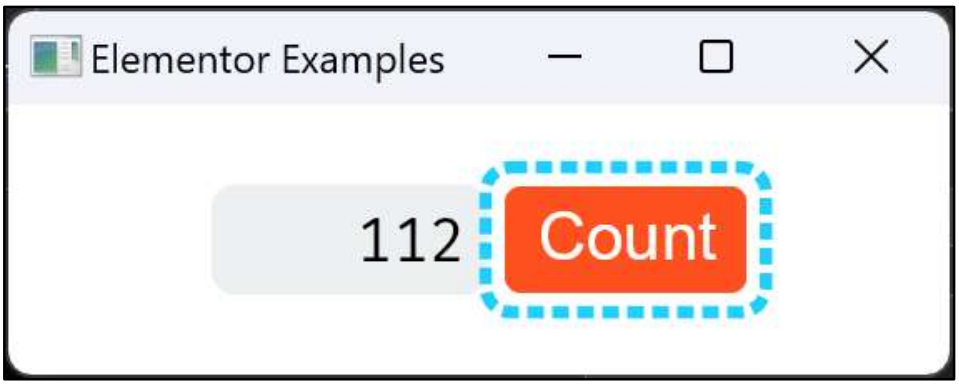


Рисунок 3 – Приложение «Счетчик»

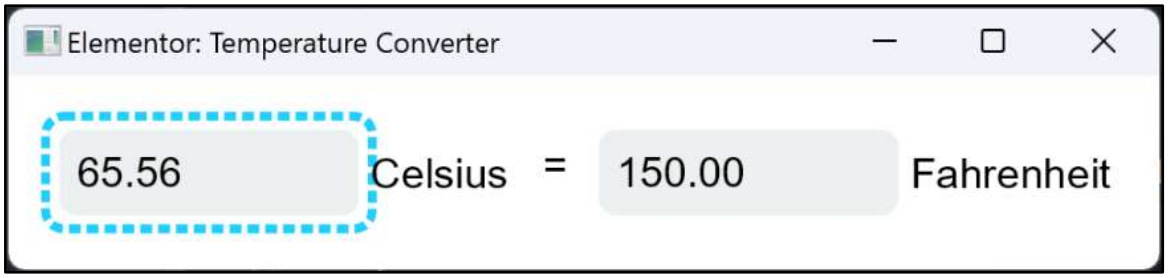


Рисунок 4 – Приложение «Конвертер температуры»

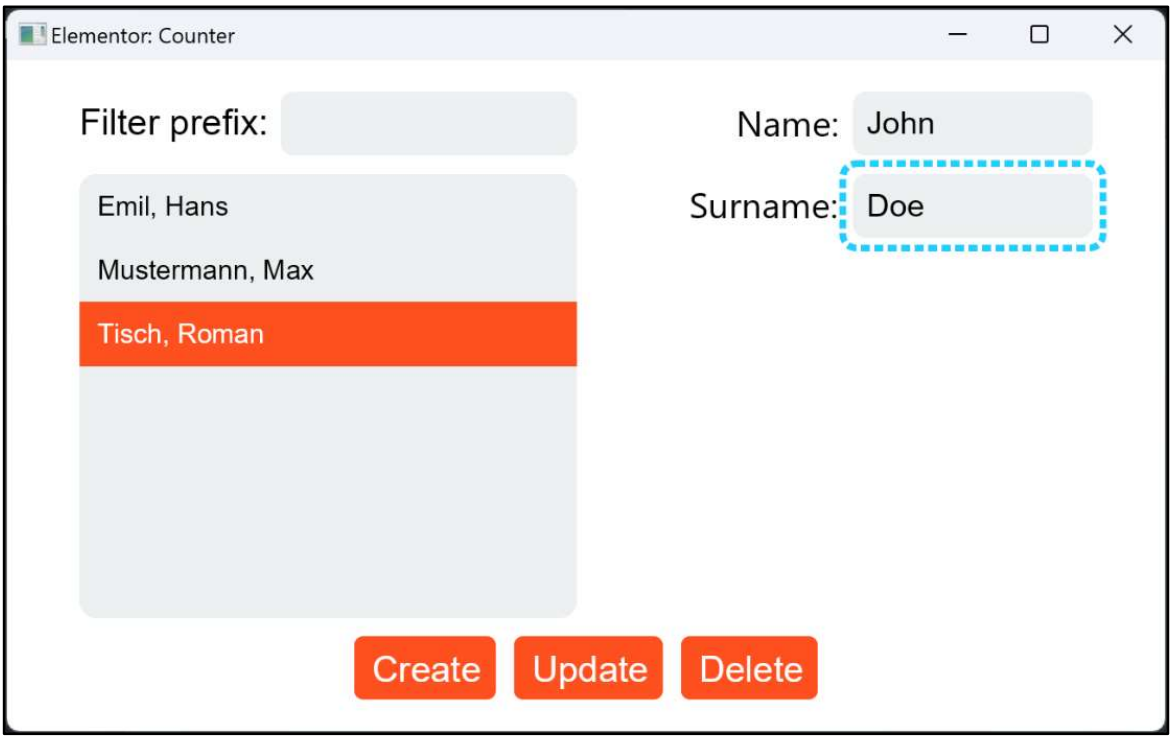


Рисунок 5 – Приложение «CRUD»