Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ПО ПРОГРАММЕ БАКАЛАВРИАТА

09.03.04 Программная инженерия

(код, наименование ОПОП ВО: направление подготовки, направленность (профиль))

«Разработка программно-информационных систем»

Программная платформа для создания элементов графического пользовательского интерфейса

(название темы)

Дипломный проект

(вид ВКР: дипломная работа или дипломный проект)

Автор ВКР _____ Д. Г. Шлифер

(подпись, дата) (инициалы, фамилия)

Группа ПО-02б

Руководитель ВКР _____ Т. М. Белова

(подпись, дата) (инициалы, фамилия)

Нормоконтроль _____ А. А. Чаплыгин

(подпись, дата) (инициалы, фамилия)

ВКР допущена к защите:

Заведующий кафедрой _____ А. В. Малышев

(подпись, дата) (инициалы, фамилия)

Курск 2024 г.

Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

УТВЕРЖДАЮ:

Заведующий кафедрой

_____

(подпись, инициалы, фамилия)

«____» _____ 20____ г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ ПО ПРОГРАММЕ БАКАЛАВРИАТА

Студента Шлифера Д.Г., шифр 20-08-0080, группа ПО-02б

1. Тема «Программная платформа для создания элементов графического пользовательского интерфейса» утверждена приказом ректора ЮЗГУ от «04» апреля 2024 г. № 1616-с.

2. Срок предоставления работы к защите «11» июня 2024 г.

3. Исходные данные для создания программной системы:

3.1. Перечень решаемых задач:

1) провести анализ предметной области;

2) разработать концептуальную модель программы для создания графических пользовательских интерфейсов;

3) спроектировать программную систему для создания графических пользовательских интерфейсов;

4) сконструировать и протестировать программную систему для создания графических пользовательских интерфейсов.

3.2. Входные данные и требуемые результаты для программы:

1) Входными данными для программной системы являются: код, изображения, файлы настроек, нужные для приложения данные.

2) Выходными данными для программной системы являются: сформированное приложение с элементами управления и обработчиками событий.

4. Содержание работы (по разделам):

4.1. Введение

4.1. Анализ предметной области

4.2. Техническое задание: основание для разработки, назначение разработки, требования к программной системе, требования к оформлению документации.

4.3. Технический проект: общие сведения о программной системе, проект данных программной системы, проектирование архитектуры программной системы, проектирование пользовательского интерфейса программной системы.

4.4. Рабочий проект: спецификация компонентов и классов программной системы, тестирование программной системы, сборка компонентов программной системы.

4.5. Заключение

4.6. Список использованных источников

5. Перечень графического материала:

Лист 1. Сведения о ВКРБ.

Лист 2. Цель и задачи разработки.

Лист 3. Концептуальная модель фреймворка.

Лист 4. Интерфейс окна.

Лист 5. Диаграмма прецидентов.

Лист 6. Схема обмена данных.

Лист 7. Тестирование.

Лист 8. Заключение.

Руководитель ВКР		Т. М. Белова
	(подпись, дата)	(инициалы, фамилия)
Задание принял к исполнению		Д. Г. Шлифер
	(подпись, дата)	(инициалы, фамилия)

РЕФЕРАТ

Объем работы равен 135 страницам. Работа содержит 21 иллюстрацию, 21 таблицу, 23 библиографических источников и 8 листов графического материала. Количество приложений – 2. Графический материал представлен в приложении А. Фрагменты исходного кода представлены в приложении Б.

Перечень ключевых слов: фреймворк, графический интерфейс, интерфейс, библиотека, классы, элементы управления, событие, окно, компонент, метод.

Объектом разработки является фреймворк для создания элементов графического пользовательского интерфейса.

Целью выпускной квалификационной работы является создание удобной библиотеки для создания элементов пользовательского интерфейса.

В процессе создания фреймворка были выделены основные классы путем создания информационных блоков, использованы функции и методы модулей, обеспечивающие работу с сущностями предметной области.

При разработке фреймворка использовалась среда разработки «Microsoft Visual Studio».

ABSTRACT

The volume of work is 135 pages. The work contains 21 illustration, 21 table, 23 bibliographic sources and 8 sheets of graphic material. The number of applications is 2. The graphic material is presented in annex A. The layout of the site, including the connection of components, is presented in annex B.

List of keywords: framework, GUI, library, classes, control, event, window, interface, method.

The object of development is a framework for creating elements of graphical user interface.

The objective of the final qualification work is to create a convenient library for creating user interface elements.

In the process of creating the framework, the main classes were identified by creating information blocks, functions and methods of modules that provide work with the entities of the subject area were used.

The development environment ”<Microsoft Visual Studio”> was used during the framework development.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ГИП – графическекский интерфейс пользователя,

GUI(Graphic User Interface) – графический полльзовательский интерфейс

ИС – информационная система.

ИТ – информационные технологии.

ПО – программное обеспечение.

РП – рабочий проект.

ТЗ – техническое задание.

ТП – технический проект.

UML (Unified Modelling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения.

ВВЕДЕНИЕ

С развитием современных технологий и повышением требований к функциональности и производительности приложений, использование фреймворков становится все более актуальным для разработки программного обеспечения. Фреймворк – это набор готовых компонентов, библиотек и шаблонов, который позволяет ускорить процесс разработки, повысить надежность и безопасность приложения, а также облегчить его поддержку и масштабируемость.

Использование фреймворков позволяет разработчикам сосредоточиться на решении бизнес-задач, минуя необходимость повторного написания базового функционала. Это позволяет сократить время разработки, снизить затраты на обучение и поддержку разработчиков, а также повысить качество и уровень безопасности разрабатываемого приложения.

В связи с этим, фреймворки становятся неотъемлемой частью современного мира. Они обеспечивают разработчиков мощными инструментами и инфраструктурой, ускоряют процесс разработки и обеспечивают высокую гибкость и масштабируемость приложений. Именно поэтому использование фреймворков становится все более популярным среди разработчиков и компаний, стремящихся к созданию современных и эффективных приложений.

Цель настоящей работы – разработка фреймворка для создания графического пользовательского интерфейса. Для достижения поставленной цели необходимо решить *следующие задачи:*

– провести анализ предметной области;

– разработать концептуальную модель фреймворка;

– спроектировать фреймворк;

– протестировать полученные с помощью фреймворка приложения.

Структура и объем работы. Отчет состоит из введения, 4 разделов основной части, заключения, списка использованных источников, 2 приложений. Текст выпускной квалификационной работы равен 135 страницам.

Во введении сформулирована цель работы, поставлены задачи разработки, описана структура работы, приведено краткое содержание каждого из разделов.

В первом разделе на стадии описания технической характеристики предметной области приводится сбор информации.

Во втором разделе на стадии технического задания приводятся требования к разрабатываемому фреймворку.

В третьем разделе на стадии технического проектирования представлены проектные решения для фреймворка.

В четвертом разделе приводится список классов и их методов, использованных при разработке фреймворка, производится тестирование разработанного фреймворка.

В заключении излагаются основные результаты работы, полученные в ходе разработки.

В приложении А представлен графический материал. В приложении Б представлены фрагменты исходного кода.

1 Анализ предметной области

1.1 Характеристика графического пользовательского интерфейса

Первая концепция графического пользовательского интерфейса была создана ученым Дагом Энгельбартом в 1960-х годах, затем данную концепцию переняли ученые из лаборатории Xerox Parc и в последствии появилась система WIMP (Windows, Icons, Menus, Pointers) в 1973 г., но коммерческое воплощение данная концепция получила только в 1984 благодаря продукции Apple Computers. Сейчас мы можем видеть GUI во всех операционных системах.

Термин «графическеский интерфейс пользователя» (ГИП) означает использование графических элементов для взаимодействия с компьютером [2]. Функциональные элементы интерфейса должны отображаться в соответствии с их назначением и свойствами. Сегодня практически невозможно найти программу без использования меню и кнопок, благодаря которым пользователь может взаимодействовать с ПК.

При проектировании ГИП используется концепция «Do what I mean» (DWIM). Согласно концепции требуется, чтобы система пыталась предугадать намерения пользователя, автоматически исправляя тривиальные ошибки, а не слепо выполняя явные, но потенциально неверные действия пользователя.

ГИП предполагает построение окна методом добавления элементов для взаимодействия (Control) и обработчиков событий (Event).

Графические интерфейсы можно классифицировать следующим образом:

1. Простые: стандартные экранные формы и элементы интерфейса, предоставляемые самой GUI.

2. Истинно-графические, двумерные: нестандартные элементы интерфейса и оригинальные метафоры, созданные с использованием собственных возможностей приложения или сторонней библиотеки.

3. Трехмерные.

В современном мире все большее количество приложений оснащаются графическими интерфейсами. Применение графических интерфейсов программ (ГИП) способствует улучшению наглядности, удобства использования и интуитивного понимания функционала приложений.

1.2 Характеристика фреймворков

История появления фреймворков для создания графического пользовательского интерфейса (GUI) началась в 1980-х годах с развитием персональных компьютеров и появлением операционных систем, поддерживающих GUI, таких как Macintosh System Software и Windows.

Программисты столкнулись с необходимостью создания удобных и интуитивно понятных пользовательских интерфейсов для своих приложений, и в ответ на это начали появляться различные библиотеки и фреймворки, упрощающие и ускоряющие процесс создания GUI-приложений.

Со временем фреймворки становились все более мощными и универсальными, позволяя разработчикам создавать кросс-платформенные приложения с современными и стильными интерфейсами. Количество и разнообразие фреймворков для GUI продолжает расти, отвечая на потребности разработчиков в удобных и эффективных инструментах для создания приложений.

Фреймворк [3] - это набор структурированных и готовых к использованию компонентов, библиотек и инструментов, который предоставляет разработчику рамочную структуру для создания приложения. Фреймворк облегчает процесс разработки, предоставляя готовые решения для часто встречающихся задач, таких как управление данными, обработка событий, взаимодействие с пользовательским интерфейсом и многое другое.

Использование фреймворка позволяет ускорить процесс разработки, повысить стабильность и масштабируемость приложения, а также упростить поддержку и обновление кода. Фреймворк также обеспечивает стандартизацию процесса разработки, что помогает разработчикам лучше ориентироваться в коде и улучшить его качество.

Фреймворки могут быть общими, предназначенными для разработки различных видов приложений, или специализированными, ориентированными на конкретные типы приложений или технологии.

1.3 Сравнение популярных GUI фреймворков

1.3.1 Qt

Qt [4] является мощным фреймворком для разработки кросс-платформенных десктопных приложений с графическим интерфейсом. Он предоставляет широкий набор компонентов и инструментов для создания современных и качественных интерфейсов.

Плюсы:

– поддержка множества платформ (Windows, macOS, Linux, Android, iOS);

– богатый набор виджетов и инструментов;

– высокая производительность;

– активное сообщество и хорошая документация.

Минусы:

– лицензирование может быть дорогим для коммерческих приложений;

– может иметь крутой порог вхождения для начинающих.

1.3.2 Electron

Electron [5] позволяет разработчикам создавать кросс-платформенные десктопные приложения с использованием веб-технологий, таких как HTML, CSS и JavaScript. Он обеспечивает возможность упаковки веб-приложений в исполняемые файлы для различных операционных систем.

Плюсы:

– использует веб-технологии (HTML, CSS, JavaScript), что упрощает разработку для веб-разработчиков;

– кросс-платформенность (Windows, macOS, Linux);

– большое сообщество и множество библиотек.

Минусы:

– высокое потребление ресурсов и большой размер конечного приложения;

– меньшая производительность по сравнению с нативными решениями.

1.3.3 JavaFX

JavaFX [6] - это фреймворк, предоставляемый Java для создания кросс-платформенных десктопных и веб-приложений с графическим интерфейсом. Он поддерживает различные стили и эффекты для дизайна интерфейса.

Плюсы:

– хорошо интегрируется с экосистемой Java;

– поддержка мультимедийных и 3D-графических возможностей;

– кросс-платформенность (Windows, macOS, Linux).

Минусы:

– менее популярный по сравнению с другими решениями, что может усложнить поиск ресурсов и поддержки;

– весомый размер приложений по сравнению с нативными решениями.

1.3.4 GTK+

GTK [7] - это набор библиотек и инструментов для создания графического пользовательского интерфейса в Linux и других UNIX-подобных операционных системах. Он предоставляет различные виджеты и стили для разработки качественных интерфейсов.

Плюсы:

– открытый исходный код и бесплатность;

– хорошая поддержка Linux;

– легкость интеграции с языками, такими как Python.

Минусы:

– меньшая поддержка Windows и macOS по сравнению с другими фреймворками;

– интерфейс может выглядеть не так современно.

1.3.5 WPF (Windows Presentation Foundation)

WPF [8] предоставляет инструменты для создания десктопных приложений на платформе Windows с использованием .NET Framework. Он поддерживает создание богатых и интерактивных пользовательских интерфейсов с помощью XAML (Extensible Application Markup Language).

Плюсы:

– отличная интеграция в экосистему Windows;

– богатый набор функций для создания сложных интерфейсов;

– поддержка XAML для декларативного описания интерфейсов.

Минусы:

– привязанность к платформе Windows;

– крутая кривая обучения для новичков.

1.3.6 Перспектива развития

Проанализировав самые популярные фреймворки для создания графических интерфейсов, можно сделать вывод, что возникают сложности с лицензированием, высокого потребления ресурсов и меньшей производительности, крутого порога вхождения и усложненного поиска ресурсов.

К перспективам развития GUI фрейворков можно отнести:

1. Улучшение производительности: разработчики фреймворков будут стремиться увеличить скорость работы приложений, оптимизировать процессы отрисовки интерфейсов и уменьшить нагрузку на процессор.

2. Поддержка новых технологий: фреймворки будут активно интегрировать новые технологии, такие как искусственный интеллект, виртуальная реальность и др., для создания интерактивных и инновационных пользовательских интерфейсов.

3. Улучшение удобства использования: фреймворки будут работать над улучшением удобства использования пользовательского интерфейса, добавляя новые функции, улучшая навигацию и дизайн, а также упрощая процесс создания интерфейсов для разработчиков.

4. Развитие мобильной адаптации: с увеличением количества мобильных устройств, фреймворки будут активно развивать механизмы адаптации пользовательских интерфейсов под различные размеры экранов и разрешения.

5. Повышение безопасности: разработчики фреймворков будут усиливать уровень безопасности пользовательских интерфейсов, защищая их от взломов, утечек данных и других угроз.

6. Поддержка кроссплатформенности: фреймворки будут продолжать развивать возможности кроссплатформенной разработки, позволяя создавать приложения с единным кодом для различных операционных систем и устройств.

7. Интеграция с новыми технологиями интерфейсов: разработчики фреймворков будут работать над интеграцией новых технологий интерфейсов, таких как голосовые команды, жесты и распознавание лиц, для создания более удобного и интуитивного пользовательского опыта.

2 Техническое задание

2.1 Основание для разработки

Полное наименование системы: «Программная платформа для создания элементов графического пользовательского интерфейса». Основанием для разработки программы является приказ ректора ЮЗГУ от «04» апреля 2024 г. №1616-с «Об утверждении тем выпускных квалификационных работ».

2.2 Назначение разработки

Программно-информационная система предназначена для облегчения создания графических пользовательских интерфейсов в десктопных приложениях различного назначения. В реализованном фреймворке должна быть обеспечена возможность создания новых элементов графического интерфейса сторонними разработчиками для своих приложений с различными функциональными возможностями.

2.3 Требования к программной системе

2.3.1 Требования к данным программной системы

На рисунке 2.1 представлена концептуальная модель данных программной системы в виде UML-диаграммы сущность-связь.

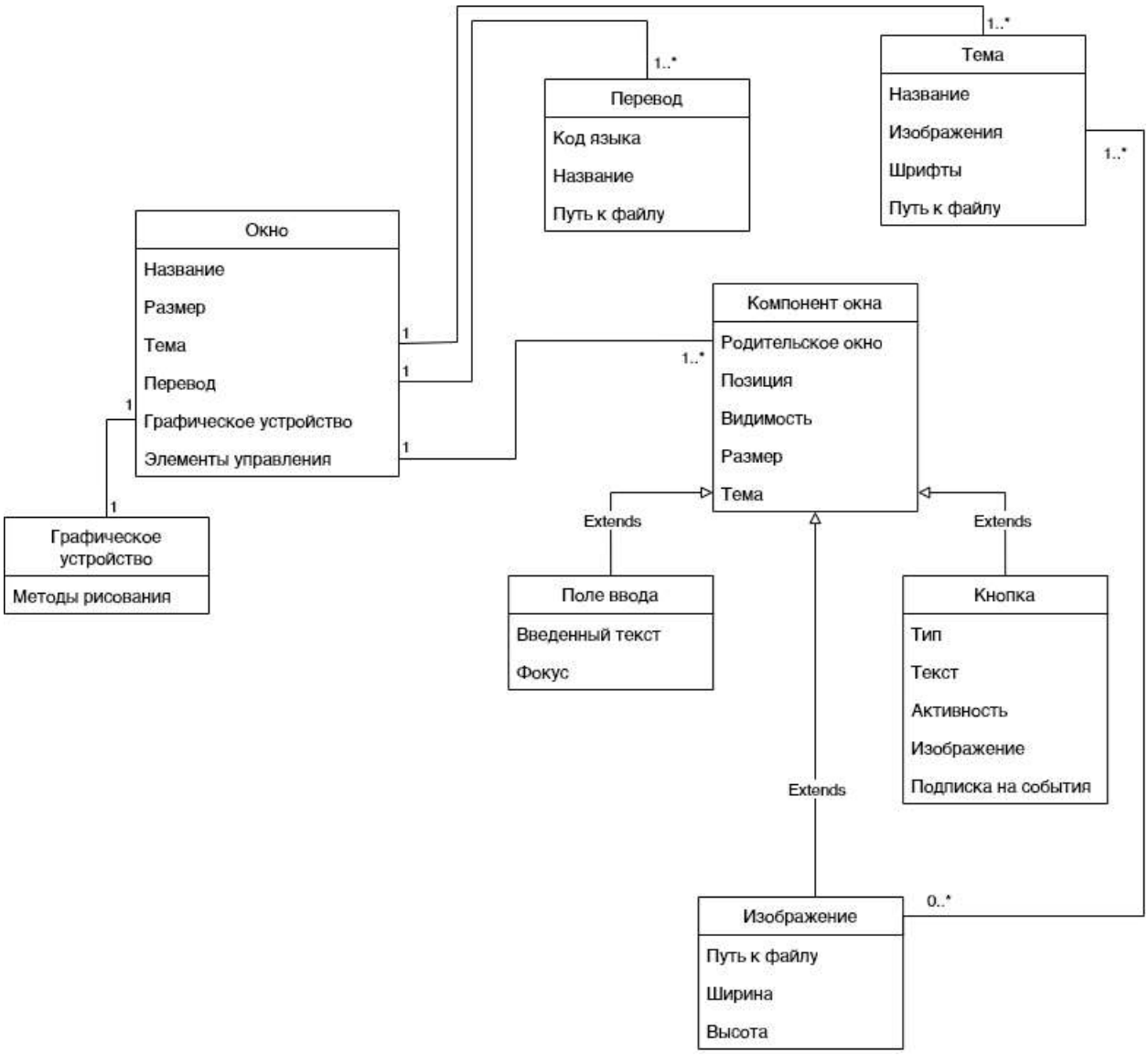


Рисунок 2.1 – Концептуальная модель данных

Входными данными для фреймворка являются: изображения, файлы настроек, код программы, регистры и названия ресурсов.

Выходными данными являются: десктопное приложение.

2.3.2 Функциональные требования к программной системе

На основании анализа предметной области в программе должны быть реализованы следующие прецеденты:

1. Добавление изображения.
2. Добавление кнопки.
3. Создание темы.
4. Создание функции смены языка.

2.3.3 Моделирование вариантов использования

Для разрабатываемого фреймворка была реализована модель, которая обеспечивает наглядное представление вариантов использования фреймворка.

Она помогает в физической разработке и детальном анализе взаимосвязей объектов. При построении диаграммы вариантов использования применяется унифицированный язык визуального моделирования UML[9].(рисунок 2.2).

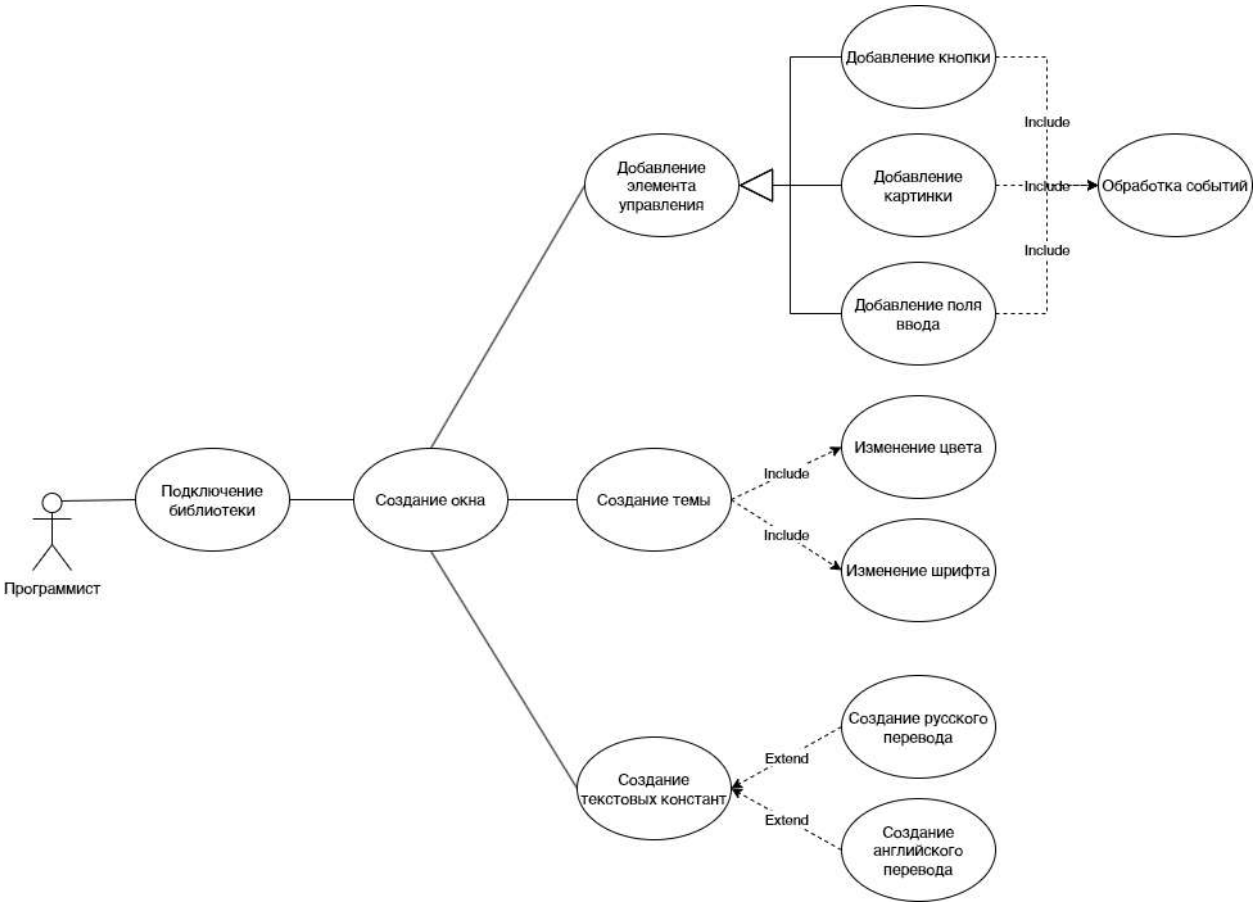


Рисунок 2.2 – Варианты использования фреймворка

2.3.4 Вариант использлвания «Добавление изображения»

Заинтересованные лица и их требования: пользователь желает добавить изображение. Предусловие: пользователь создал папку images в папке res и создал нужные темы. Постусловие: Изображение повилось в окне.

Основной успешный сценарий:

1. Пользователь создал несколько вариантов изображений.
2. Пользователь включил изображения в реестр.
3. Пользователь добавил переменную изображения и приписал к ней реестр в файле заголовка.
4. Пользователь прописал название ресурса и путь к изображению в файле ресурсов.
5. Пользователь прописал изображение в файле темы.
6. Пользователь добавил элемент управления с помощью метода add_control() в коде.
7. Пользователь прописал позицию в методе UpdateControlsPosition().

2.3.5 Вариант использлвания «Добавление кнопки»

Заинтересованные лица и их требования: пользователь желает добавить кнопку. Предусловие: пользователь создал файл окна и файл темы. Постусловие: создается ini файл в корневой папке проекта.

Основной успешный сценарий:

1. Пользователь добавил переменную кнопки в файле заголовка.
2. Пользователь прописал событие и связал его с помощью команды bind().
3. Пользователь создал элемент управления с помощью метода add_control() в коде.
4. Пользователь прописал позицию в методе UpdateControlsPosition().
5. Пользователь прописал кнопку в json-файле темы.

2.3.6 Вариант использлвания «Создание темы»

Заинтересованные лица и их требования: пользователь желает создать тему для своего проекта. Предусловие: пользователь создал json-файл в папке res. Постусловие: при нажатии кнопки сменяется тема.

Основной успешный сценарий:

1. Пользователь создал несколько вариантов тем.
2. Пользователь прописал название тем в реестре файла «resources.h».

3. Пользователь прописал тип ресурса и путь к темам в файле формата «.rc».

4. Пользователь добавил темы в методе set_app_themes() в файле «main.cpp».

5. Пользователь выбрал первичную тему с помошью config.

6. Пользователь создал элемент управления switch_theme_button.

2.3.7 Вариант испольлвания «Создание функции смены языка»

Заинтересованные лица и их требования: пользователь желает создать файл для быстрой настройки окна и других переменных. Предусловие: пользователь ввел команду создания файла конфигураций. Постусловие: при нажатии на кнопку меняется язык.

Основной успешный сценарий:

1. Пользователь создал несколько вариантов языка.

2. Пользователь прописал название языка в реестре файла «resources.h».

3. Пользователь прописал тип ресурса и путь к темам в файле формата «.rc».

4. Пользователь добавил темы в методе set_app_locales() в файле «main.cpp».

5. Пользователь выбрал первичную тему с помошью config.

6. Пользователь создал элемент управления switch_lang_button.

7. Пользователь связал язык с нужными элементами.

2.3.8 Требования к интерфейсу

Фреймворк должен:

1. Обеспечивать создание новых компонентов.

2. Предоставлять общий интерфейс к подсистеме рисования.

3. Предоставлять общий интерфейс к событиям. Любой компонент или пользователь может подписаться на любую группу сообщений, в том числе

пользовательскую, с возможностью асинхронной отправки/получения сообщений.

4. Принимать системные сообщения, реагировать на мышь, клавиатуру и прочие события.

5. Открывать окна и отображать на них компоненты.

6. Предоставлять систему текстовых констант для заголовков и надписей в зависимости от выбранного языка.

7. Предоставлять систему тем для изменения шрифтов и цветовых акцентов в зависимости от выбранной темы.

Модель простого окна представлена на рисунке 2.3.

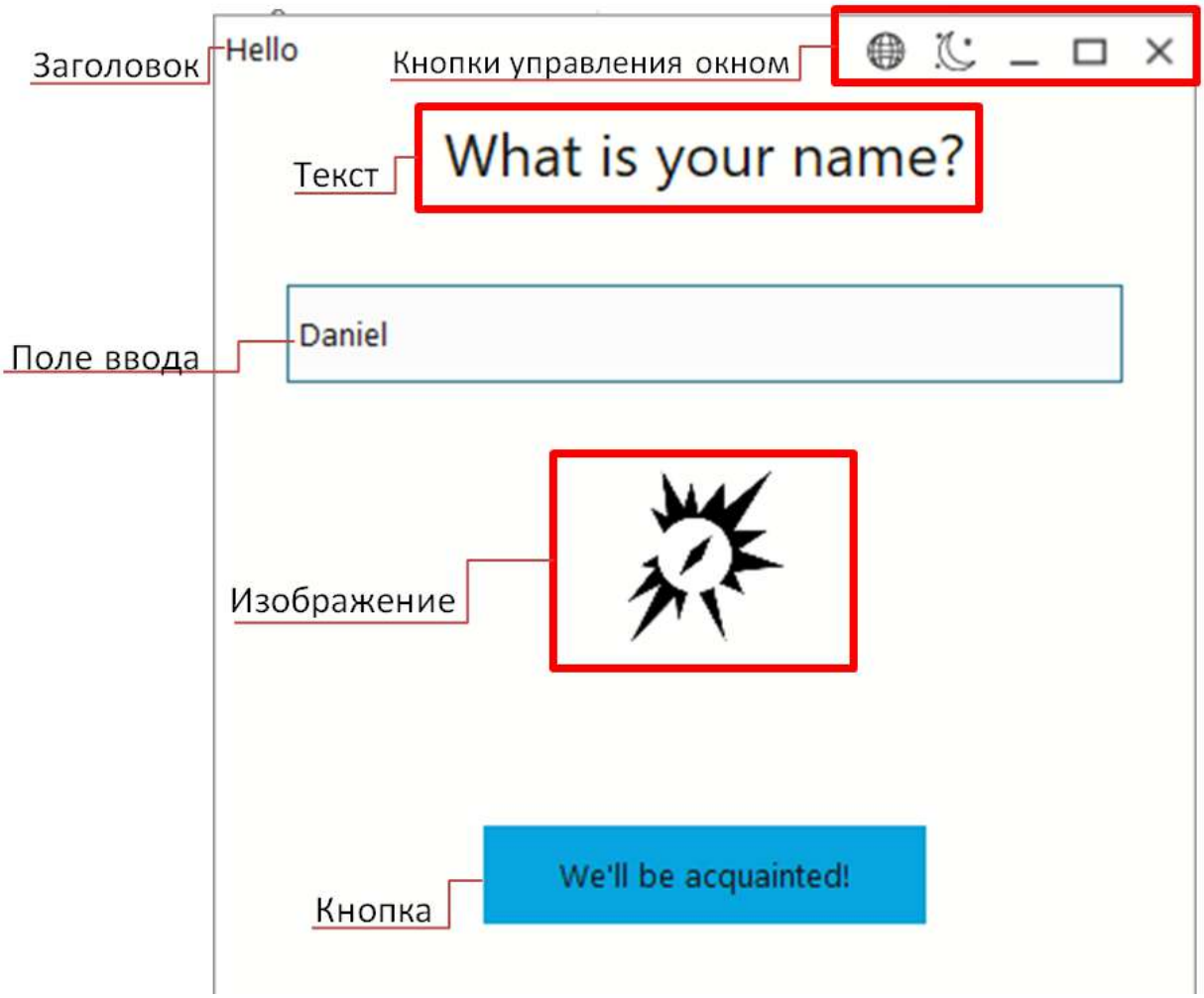


Рисунок 2.3 – Общая схема простого окна

В состав данного окна входят название программы, кнопки управления окном, такие как: кнопка смены темы, кнопка смены языка, кнопка свернуть

окно, кнопка развернуть окно, кнопка закрыть окно, и другие элементы взаимодействия, такие как: поле для ввода текста, кнопка подтверждения ввода, различные изображения и текст.

2.4 Нефункциональные требования к программной системе

2.4.1 Требования к надежности

Приложение должно функционировать на всех разработанных тестах. Тесты требуется разработать на этапе рабочего проекта. Приложение должно устойчиво функционировать при соблюдении гарантии устойчивого функционирования операционной системы. Под устойчивой работой ПК понимается непрерывное функционирование программы в отсутствии критических сбоев, приводящих к аварийному завершению.

2.4.2 Требования к аппаратному обеспечению

Для работы с фреймворком необходимо дисковое пространство не менее 38 Мб, свободная оперативная память в размере не менее 512 Мб, видеокарта с не менее 256 Мб видеопамяти, разрешение экрана не менее 1024*768, клавиатура, мышь.

2.5 Требования к оформлению документации

Требования к стадиям разработки программ и программной документации для вычислительных машин, комплексов и систем независимо от их назначения и области применения, этапам и содержанию работ устанавливаются ГОСТ 19.102–77.

Программная документация должна включать в себя:

1. Анализ предметной области.
2. Техническое задание.
3. Технический проект.
4. Рабочий проект.

3 Технический проект

3.1 Общая характеристика организации решения задачи

Необходимо спроектировать и разработать фреймворк, который должен ускорить разработку графических интерфейсов и оптимизировать емкостные характеристики приложения. Основое внимание следует уделить самым важным элементам взаимодействия с пользователем и выполняемого ими функционала.

3.1.1 Обоснование выбора технологии проектирования

На сегодняшний день информационный рынок, поставляющий программные решения в выбранной сфере, предлагает множество продуктов, позволяющих достигнуть поставленной цели – создание фреймворка. В процессе разработки фреймворка используются язык программирования C++. Также, значения основных переменных задавались в json-файлах.

3.1.2 Язык программирования C++

C++ – это высокоуровневый язык программирования общего назначения. Этот язык программирования зарекомендовал себя, ибо уже имеет множество библиотек и фреймворков.

Преимущества использования языка C++ при создании фреймворков:

1. Высокая производительность: C++ является компилируемым языком с низким уровнем абстракции, что позволяет создавать быстрые и эффективные фреймворки.

2. Возможность доступа к аппаратному обеспечению: C++ позволяет напрямую управлять памятью и ресурсами компьютера, что полезно при разработке фреймворков, работающих с аппаратным обеспечением.

3. Богатая функциональность: C++ обладает множеством возможностей и библиотек, что позволяет создавать мощные и гибкие фреймворки для различных целей.

Недостатки использования языка C++ при создании фреймворков:

1. Сложность: C++ – сложный и мощный язык программирования, который требует от разработчиков высокого уровня квалификации. Это может затруднить разработку и поддержку фреймворка, особенно для менее опытных программистов.

2. Низкая гибкость: Использование C++ может привести к более жесткой архитектуре фреймворка, что может затруднить его дальнейшее развитие и модификацию.

3. Сильная типизация: C++ имеет сильную статическую типизацию, что может потребовать более тщательного и длительного процесса разработки, особенно при работе с большими проектами.

Таким образом, использование языка C++ при разработке фреймворков обладает рядом преимуществ. Во-первых, C++ является высокопроизводительным языком программирования, что позволяет создавать быстрые и эффективные фреймворки. Во-вторых, C++ обладает богатыми возможностями для объектно-ориентированного программирования, что упрощает создание модульной и расширяемой архитектуры фреймворка. Кроме того, C++ поддерживает многопоточность, что позволяет улучшить производительность и масштабируемость фреймворка. В целом, использование языка C++ при разработке фреймворков позволяет создавать мощные, гибкие и эффективные инструменты для разработки программного обеспечения.

3.1.3 Сравнение методов отрисовки компонетов

GDI+ (Graphics Device Interface+) - это набор API для отрисовки 2D графики в операционной системе Windows. Он предоставляет набор простых методов для рисования примитивов, текста, изображений и других графических элементов. GDI+ поддерживает аппаратное ускорение графики и имеет удобный интерфейс для работы с изображениями. Однако он ограничен в возможностях 3D графики.

OpenGL[10] - это кросс-платформенная библиотека для отображения 2D и 3D графики. Она предоставляет низкоуровневый доступ к графическому аппарату компьютера и поддерживает широкий спектр возможностей, вклю-

чая шейдеры, текстурирование, освещение и другие эффекты. OpenGL позволяет создавать более сложные и реалистичные графические эффекты, чем GDI+, но требует более глубокого понимания работы с графикой.

Vulkan[11] - это новое поколение библиотеки для написания графических приложений, разработанное компанией Khronos Group. Она предоставляет низкоуровневый доступ к графическому аппарату и позволяет максимально эффективно использовать мощности современных GPU. Vulkan обеспечивает высокую производительность и позволяет параллельно обрабатывать большое количество графических команд. Однако для работы с ней требуется глубокое знание графического программирования.

При создании графических интерфейсов для приложений можно использовать любой из указанных методов в зависимости от требуемого уровня сложности и производительности. GDI+ подходит для простых 2D интерфейсов, OpenGL позволяет создавать более сложные 2D и 3D графику, а Vulkan обеспечивает высокую производительность и возможности для создания сложных 3D интерфейсов. Из-за своей простоты я выбрал Graphics Device Interface+.

3.1.4 Достоинства Json

JSON (JavaScript Object Notation) - это легкий формат обмена данными, основанный на тексте, который используется для передачи структурированных данных между программами.

Преимущества JSON:

1. Простота использования: JSON легко читаем и понятен людям, что делает его удобным для работы с данными.

2. Легкость: JSON файлы компактны и занимают мало места, что ускоряет передачу данных и уменьшает нагрузку на сеть.

3. Поддержка множества языков программирования: JSON поддерживается большинством языков программирования, что делает его универсальным форматом для обмена данными между различными системами.

4. Поддержка структурированных данных: JSON поддерживает различные типы данных, включая строки, числа, логические значения, массивы и объекты, что позволяет передавать сложные структуры данных.

5. Гибкость: JSON позволяет легко добавлять, изменять и удалять данные, а также манипулировать ими, что делает его удобным для работы с динамическими данными.

Таким образом, использование JSON упрощает обмен данными между различными системами и позволяет эффективно передавать информацию в формате, который легко читать и понять.

3.2 Диаграмма компонентов и схема обмена данными между классами

Диаграмма компонентов описывает особенности физического представления разрабатываемой системы. Она позволяет определить архитектуру системы, установив зависимости между программными компонентами, в роли которых может выступать как исходный, так и исполняемый код. Основными графическими элементами диаграммы компонентов являются классы, а также зависимости между ними. На рисунке 3.1 изображена диаграмма компонентов для проектируемой системы. Она базируется на двух сущностях - Window и Control. Окно может содержать контролы, также само окно является контролом.

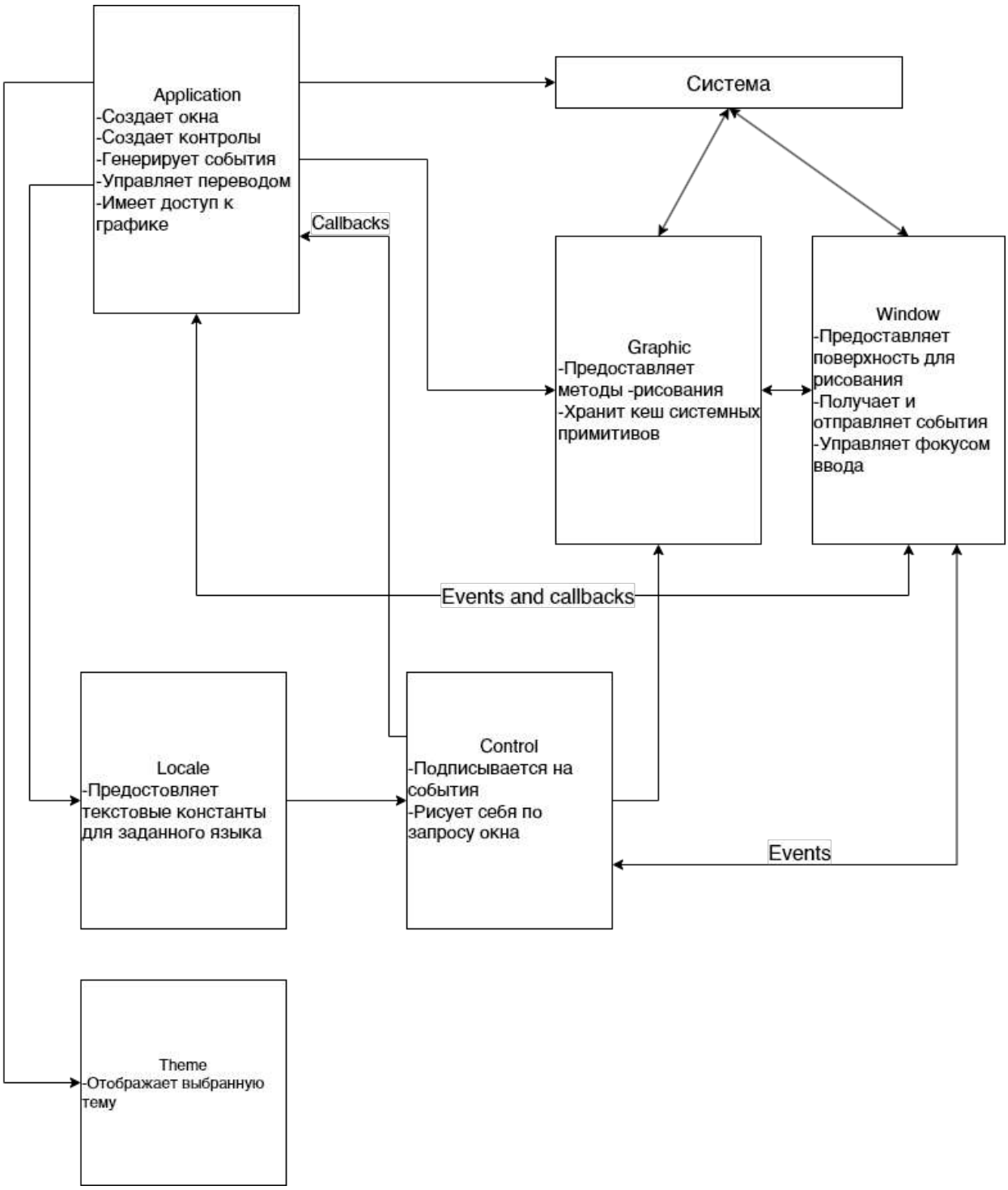


Рисунок 3.1 – Схема обмена данными между классами

3.3 Описание основных сущностей

Проанализировав требования, можно выделить девять основных сущностей:

1. «Фреймворк».
2. «Окно».

3. «Компонент».
4. «Кнопка».
5. «Изображение».
6. «Графика».
7. «Тема».
8. «Локаль».
9. «Ошибка».

В состав сущности «Фреймворк» можно включить атрибуты, представленные в таблице 3.1.

Таблица 3.1 – Атрибуты сущности «Фреймворк»

Поле	Тип	Обязательное	Описание
1	2	3	4
runned	bool	false	Флаг запуска приложения
err	error	false	Ошибка

В состав сущности «Ошибка» можно включить атрибуты, представленные в таблице 3.2.

Таблица 3.2 – Атрибуты сущности «Ошибка»

Поле	Тип	Обязательное	Описание
1	2	3	4
type	error_type	true	тип ошибки
component	string	true	указатель на компонент, вызвавший ошибку
message	string	true	Сообщение

В состав сущности «Графика» можно включить атрибуты, представленные в таблице 3.3.

Таблица 3.3 – Атрибуты сущности «Графика»

Поле	Тип	Обязательное	Описание
1	2	3	4
pc	primitive_container	true	Примитивный контейнер для хранения
max_size	rect	true	максимальный размер
background_color	color	true	Задний фон
mem_dc	HDC	true	Указатель на графический объект
mem_bitmap	HBITMAP	true	Указатель на битмап
err	error	false	Ошибка

В состав сущности «Компонент» можно включить атрибуты, представленные в таблице 3.4.

Таблица 3.4 – Атрибуты сущности «Компонент»

Поле	Тип	Обязательное	Описание
1	2	3	4
_position	rect	true	Позиция элемента
parent	window	true	Родительское окно
topmost	bool	true	Флаг «Поверх всех элементов»
showed	bool	true	Видимость элемента
enabled	bool	true	Включенность элемента
focused	bool	false	Возвращает true, если элемент управления сфокусирован

Продолжение таблицы 3.4

1	2	3	4
focusing	bool	false	Возвращает true, если элемент управления получает фокус
err	error	false	Ошибка

В состав сущности «Компонент Кнопка» можно включить атрибуты, представленные в таблице 3.5.

Таблица 3.5 – Атрибуты сущности «Кнопка»

Поле	Тип	Обяза-тельное	Описание
1	2	3	4
button_view_	button_view	true	Вид кнопки
caption	string	true	Надпись на кнопке
image_	string	false	Изображение на кнопке
image_size	int32_t	false	Размер изображения
tooltip_	tooltip	false	Подсказка при наведения на кнопку
theme_	i_theme	true	Тема кнопки
_position	rect	true	Позиция элемента
parent	window	true	Родительское окно
my_subscriber_id	string	true	Подписка на события
topmost	bool	true	Флаг «Поверх всех элементов»
pushed, turned	bool	false	Флаг «Включения кнопки»
showed	bool	true	Видимость элемента
enabled	bool	true	Включенность
focused	bool	false	Возвращает true, если элемент управления сфокусирован

Продолжение таблицы 3.5

1	2	3	4
focusing	bool	false	Возвращает true, если элемент управления получает фокус
err	error	false	Ошибка

В состав сущности «Компонент Изображение» можно включить атрибуты, представленные в таблице 3.6.

Таблица 3.6 – Атрибуты сущности «Изображение»

Поле	Тип	Обязательное	Описание
1	2	3	4
img_	Gdiplus::Image	true	Полотно изображения
file_name	string	true	Имя изображения
resource_index	int32_t	true	Индекс ресурса
theme_	i_theme	true	Тема изображения
_position	rect	true	Позиция изображения
parent	window	true	Родительское окно
topmost	bool	true	Флаг «Поверх всех элементов»
showed	bool	true	Видимость элемента
err	error	false	Ошибка

В состав сущности «Окно» можно включить атрибуты, представленные в таблице 3.7.

Таблица 3.7 – Атрибуты сущности «Окно»

Поле	Тип	Обязательное	Описание
1	2	3	4
graphic_	rect	true	Полотно для рисования
controls	std::vector	true	Компоненты окна
active_control	i_control	true	Активный компонент
caption	string	true	Название окна
_position, normal_position	rect	true	Позиция окна
min_width, min_height	int32_t	true	Минимальная высота/ширина окна
window_style_	window_style	true	Тип окна
window_state_, prev_window_state_	window_state	true	Состояние окна
theme_	i_theme	true	Тема окна
showed	bool	true	Видимость окна
skip_draw_	bool	true	Флаг пропуска отрисовки
focused	bool	false	Фокус

Продолжение таблицы 3.7

1	2	3	4
docked_	bool	false	Флаг прикрепления к окну
docked_control	i_control	false	Прикрепленный к окну элемент
subscribers_	std::vector	false	Подписанные на события элементы
mouse_tracked	bool	false	Флаг отслеживания мыши
x_click, y_click	int16_t	false	Координаты клика
switch_lang_button	button	true	Кнопка управления языком
switch_theme_button	button	true	Кнопка управления темой
pin_button	button	true	Кнопка управления закреплением
minimize_button	button	true	Кнопка «Свернуть»

Продолжение таблицы 3.7

1	2	3	4
expand_button	button	true	Кнопка «Расширить»
close_button	button	true	Кнопка «Закрыть»
err	error	false	Ошибка

В состав сущности «Локаль» можно включить атрибуты, представленные в таблице 3.8.

Таблица 3.8 – Атрибуты сущности «Локаль»

Поле	Тип	Обязательное	Описание
1	2	3	4
type	locale_type	true	Код языка
name	string	true	Название
strings	std::map	true	Строки перевода
dummy_string	string	true	Строка по умолчанию
err	error	false	Ошибка

В состав сущности «Тема» можно включить атрибуты, представленные в таблице 3.9.

Таблица 3.9 – Атрибуты сущности «Тема»

Поле	Тип	Обязательное	Описание
1	2	3	4
type	locale_type	true	Код языка
name	string	true	Название
ints	std::map	true	Коллекция чисел
strings	std::map	true	Строки
fonts	std::map	true	Коллекция шрифтов
imgs	std::map	true	Коллекция изображений
dummy_string	string	true	Строка по умолчанию
dummy_image	std::vector<uint8_t>	true	Изображение по умолчанию
err	error	false	Ошибка

В системе предусмотрен внутренний механизм связи между разделами и элементами информационных блоков, поэтому введения дополнительных идентификаторов при реализации связей между сущностями не предполагается.

Экземпляры сущностей реализуются в информационных блоках посредством элементов, атрибуты сущности – посредством полей и свойств элемента.

3.4 Описание сущностей для настройки окна

В состав сущности «Тема» можно включить набор основных полей JSON-документа и их описание, представленные в таблице 3.10.

Таблица 3.10 – Описание полей JSON-документа сущности «Тема»

Ключ	Тип	Описание
1	2	3
controls	map	Элементы управления
type	str	Тип элемента управления
background	str	Цвет заднего фона
border	str	Цвет обводки
border_width	number	Ширина обводки
round	number	Округление
text	str	Цвет текста
caption_font	map	Шрифт заголовка
name	str	Название шрифта
size	number	Размер шрифта
color	str	Цвет заливки
resource	str	Название ресурса
path	str	Путь к папке
calm	str	Основной цвет
active	str	Активный цвет
focused_border	str	Цвет обводки при фокусировке
disabled	str	Цвет при недоступности
anchor	str	Цвет ссылки
focusing	number	фокусировка
selection	str	Цвет при выборе поля ввода
button_calm	str	Основной цвет кпонки при выборе клавиатурой

Продолжение таблицы 3.10

1	2	3
button_active	str	Цвет активированной кнопки при выборе клавиатурой
scrollbar	str	Цвет полосы прокрутки
scrollbar_slider	str	Цвет ползунка полосы прокрутки
scrollbar_slider_active	str	Цвет активного ползунка полосы прокрутки
selected_item	str	Цвет выбранного элемента
active_item	str	Цвет активного элемента
title	str	Цвет заголовка листа
title_text	str	Цвет текста заголовка листа
slider	str	Цвет ползунка
slider_active	str	Цвет активного ползунка
disabled_text	str	Цвет неактивного текста в меню
text_indent	number	Отступ текста
meter	str	Цвет полосы прогресса
perform	str	Цвет активной части ползунка настройки
remain	str	Цвет неактивной части ползунка настройки
slider_width	number	Ширина ползунка настройки

Продолжение таблицы 3.10

1	2	3
slider_height	number	Высота ползунка настройки
images	map	Иконки окна

В состав сущности «Язык» можно включить набор основных полей JSON-документа и их описание, представленные в таблице 3.12.

Таблица 3.12 – Описание полей JSON-документа сущности «Тема»

Ключ	Тип	Описание
1	2	3
sections	map	Элементы управления
type	str	Тип элемента управления
ok	str	Подтверждение
yes	str	Да
no	str	Нет
cancel	number	Отмена
abort	str	Отказ
retry	str	Повторить
ignore	str	Пропустить
try_continue	str	Попробовать продолжить
complete	str	Готово
pin	str	Прикрепить окно
unpin	str	Открепить окно
dark_theme	str	Темная тема
light_theme	str	Светлая тема
switch_lang	str	Сменить язык
copy	str	Копировать

Продолжение таблицы 3.12

1	2	3
cut	str	Вырезать
paste	number	Вставить
undo	str	Отменить
redo	str	Вернуть отмену

4 Рабочий проект

4.1 Спецификация классов фреймворка

Класс Framework инициализирует фреймфорк. Запускает GDI+. Запускает или останавливает работу. (таблица 4.1).

Таблица 4.1 – Спецификация методов класса Framework

Название	Метод дрступа	Описание
1	2	3
init	public	Запуск GDI+
run	public	Запускает приложение
stop	public	Останавливает работу приложения

Класс Window создает окна для приложения. Принимает системные события и обеспечивает их рассылку подписчикам. Так же окно дает команду на перерисовку своих компонентов и предоставляет им свой graphic. Кроме этого, окно управляет фокусом ввода, может сделать модальность и отправить подписанному пользователю или в систему событие. (таблица 4.2).

Таблица 4.2 – Спецификация методов класса Window

Название	Метод дрступа	Описание
1	2	3
init	public	Создание окна
destroy	public	Уничтожение окна
add_control	public	Добавление контрола
remove_control	public	Удаление контрола
bring_to_front, move_to_back	public	Изменеие порядка слоев элементов
set_position	public	Изменеие позиции окна
position	public	Возвращение позиции окна
set_parent	public	Установление родительского окна

Продолжение таблицы 4.2

1	2	3
parent	public	Возвращение родительского окна
clear_parent	public	Очищение свойства родительского окна
update_theme	public	Обновление темы
redraw	public	Перерисовывает часть окна. Вызывается компонентом окна.
subscribe	public	Подписка на события
unsubscribe	public	Отписка от событий
emit_event	public	Посылает сообщение через системный шедулер сообщений
set_caption	public	Изменяет название окна
set_style	public	Изменяет стиль окна
set_min_size	public	Изменяет минимальный размер окна
switch_lang	public	Изменяет язык
switch_theme	public	Изменяет тему
pin	public	Прикрепляет окно
minimize	public	Сворачивает окно
expand	public	Расширяет окно
normal	public	Возвращает окно в исходное состояние
state	public	Изменяет состояние окна
disable_draw, enable_draw	public	Отключает/Включает рисование для повышения производительности при выполнении массовых операций
set_focused	public	Делает компонент окна сфокусированным
set_control_callback	public	Привязывает событие к компоненту

Продолжение таблицы 4.2

1	2	3
set_default_push_control	public	Нажатие кнопки по клавише Enter
receive_control_events	private	Принимает события от компонентов
send_event_to_control	private	Посылает событие компоненту
send_mouse_event	private	Посылает событие мыши
check_control_here	private	Проверяет компонент на месте клика
change_focus	private	Изменяет фокус
get_focused	private	Возвращает сфокусированый элемент
start_docking, end_docking	private	Включает/отключает привязку к окну
draw_border	private	Рисует границы графического контекста
send_system	private	Отправить системное сообщение

Класс Control - это любой визуальный элемент для взаимодействия с пользователем - кнопка, поле ввода, список, меню и т.д. Control знает, как обрабатывать события, поступающие от Window, хранит свои состояния и рисует себя на графическом контексте, который предоставляется содержащим его окном. (таблица 4.3).

Таблица 4.3 – Спецификация методов класса Control

Название	Метод дрступа	Описание
1	2	3
draw	public	Рисует компонент. Вызывается окном.

Продолжение таблицы 4.3

1	2	3
set_position	public	Изменяет положение контрола на окне. Координаты задаются в пикселях относительно левого верхнего угла родительского окна.
position	public	Возвращает положение контрола относительно окна
set_parent	public	Позволяет контролу получить указатель на свое родительское окно
parent	public	Возвращает указатель на родительское окно
clear_parent	public	Очищает указатель на родительское окно контрола
topmost	public	Сообщает родительскому окну, нужно ли рисовать контрол поверх всех остальных контролов
update_theme	public	Изменяет визуальную тему контрола
show,hide,showed	public	Методы управления видимостью
enable,disable,enabled	public	Методы управления включенностью
focused,focusing	public	Клавиатурный фокус ввода
get_error	public	Возвращает структуру, содержащую подробности последней ошибки

Класс Image создает изображение на экране. (таблица 4.4).

Таблица 4.4 – Спецификация методов класса Image

Название	Метод дрступа	Описание
1	2	3
image	public	Создает изображение
change_image	public	Изменяет изображение
width, height	public	Возвращает ширину/-высоту изображения
draw	public	Рисует изображение. Вызывается окном.
set_position	public	Изменяет положение контрола на окне. Координаты задаются в пикселях относительно левого верхнего угла родительского окна.
position	public	Возвращает положение контрола относительно окна
set_parent	public	Позволяет контролу получить указатель на свое родительское окно
parent	public	Возвращает указатель на родительское окно
clear_parent	public	Очищает указатель на родительское окно контрола
topmost	public	Сообщает родительскому окну, нужно ли рисовать контрол поверх всех остальных контролов
update_theme	public	Изменяет визуальную тему контрола
show,hide,showed	public	Методы управления видимостью
enable,disable,enabled	public	Методы управления включенностью

Продолжение таблицы 4.4

1	2	3
focused,focusing	public	Клавиатурный фокус ввода
get_error	public	Возвращает структуру, содержащую подробности последней ошибки

Класс Button - это любая кнопка для взаимодействия с пользователем. Кнопка знает, как обрабатывать события, поступающие от Window, хранит свои состояния и рисует себя на графическом контексте, который предоставляется содержащим его окном. (таблица 4.5).

Таблица 4.5 – Спецификация методов класса Button

Название	Метод дрступа	Описание
1	2	3
button	public	Функция создания кнопки.
draw	public	Рисует кнопку.
set_position	public	Изменяет положение контрола на окне. Координаты задаются в пикселях относительно левого верхнего угла родительского окна.
position	public	Возвращает положение контрола относительно окна
set_parent	public	Позволяет контролу получить указатель на свое родительское окно
parent	public	Возвращает указатель на родительское окно
clear_parent	public	Очищает указатель на родительское окно контрола

Продолжение таблицы 4.5

1	2	3
topmost	public	Сообщает родительскому окну, нужно ли рисовать контрол поверх всех остальных контролов
update_theme	public	Изменяет визуальную тему контрола
show,hide,showed	public	Методы управления видимостью
enable,disable,enabled	public	Методы управления включенностью
focused,focusing	public	Клавиатурный фокус ввода
get_error	public	Возвращает структуру, содержащую подробности последней ошибки
update_err	private	Обновляет структуру ошибки
set_caption	private	Устанавливает надпись на кнопке
set_button_view	private	Изменяет вид кнопки
set_image	private	Устанавливает изображение на кнопку
enable_focusing	private	Включает фокус
disable_focusing	private	Отключает фокус
turn	private	Включает кнопку
turned	private	Возвращает флаг включения кнопки
set_callback	private	Устанавливает функионал кнопки
receive_event	private	Получение события
redraw	private	Перерисовывает кнопку

Класс Graphic предоставляет интерфейс к системным методам рисования. В настоящий момент, реализовано рисование на Windows GDI/GDI+. (таблица 4.6).

Таблица 4.6 – Спецификация методов класса Graphic

Название	Метод дрступа	Описание
1	2	3
init	public	Инициализация
release	public	Деинициализации
set_background_color	public	Устанавливает цвет фона
clear	public	Заливает холст
flush	public	Отрисовка области на системный графический контекст
draw_pixel_	public	Рисует точку
draw_line	public	Рисует линию
measure_text	public	Измеряет размер текста с выбранным шрифтом
draw_text	public	Рисует текст
draw_rect	public	Рисует прямоугольник
draw_buffer	public	Рисует буфер RGB32
draw_graphic	public	Рисует содержимое другого графика

Класс Locale нужен для смены языка программы. Он загружает текстовые константы, меняет текущий язык, а также загружает json-файл для быстрой настройки. (таблица 4.7).

Таблица 4.7 – Спецификация методов класса Locale

Название	Метод дрступа	Описание
1	2	3
get_type()	public	Возвращает код языка
get_name	public	Возвращает название перевода
set	public	Меняет текущий язык

Продолжение таблицы 4.7

1	2	3
get	public	Возвращает данные перевода
load_resource	public	Загружает ресурсы перевода
load_json	public	Загружает json-файл перевода
load_locale	public	Загружает строки перевода

Класс Theme нужен для изменение цветовых акцентов окна. Он позволяет менять цвета, шрифты, изображение, а также загружать json-файлы для быстрой настройки. (таблица 4.8).

Таблица 4.8 – Спецификация методов класса Theme

Название	Метод дрступа	Описание
1	2	3
get_name	public	Возвращает название темы
set_color	public	Меняет цвет
get_color	public	Возвращает цвет
set_string	public	Меняет текст
get_string	public	Возвращает текст
set_font	public	Меняет шрифт
get_font	public	Возвращает шрифт
set_image	public	Меняет изображение
get_image	public	Возвращает изображение
load_resource	public	Загружает ресурсы темы
load_json	public	Загружает json-файл темы
load_theme	public	Загружает данные темы

Класс Config меняет значения регистра. Работает с определенными типами данных. (таблица 4.9)

Таблица 4.9 – Спецификация методов класса Config

Название	Метод дрступа	Описание
1	2	3
get_int	public	Возвращает целое число
set_int	public	Меняет целое число
get_int64	public	Возвращает большое число
set_int64	public	Меняет большое число
get_string	public	Возвращает строку
set_string	public	Меняет строку
delete_value	public	Удаляет значение из регистра
delete_key	public	Удаляет ключ из регистра

Класс Error выводит сообщения об ошибках. (таблица 4.10)

Таблица 4.10 – Спецификация методов класса Error

Название	Метод дрступа	Описание
1	2	3
is_ok	public	Присваевает ошибке тип «исправности»
reset	public	Очищает указатель на компонент и сообщение
str	public	Возвращает сообщение

4.2 Модульное тестирование разработанного фреймворка

Модульный тест для класса Text из модели данных представлен на рисунке 4.1.

```cpp
#include "pch.h"
#include "CppUnitTest.h"
#include "../include/wui/window/window.hpp"
#include "../include/wui/control/text.hpp"
#include "../include/wui/common/rect.hpp"
#include "../include/wui/common/alignment.hpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace GUITest
{
  TEST_CLASS(GUITest)
  {
    public:

    TEST_METHOD(Test_Text)
    {
      std::shared_ptr<wui::window> window = std::make_shared<wui::window>()
          ;
      const auto width = window->position().width(), height = window->
          position().height();
      const int32_t top = 40, element_height = 40, space = 30;

      std::shared_ptr<wui::text> Text = std::make_shared<wui::text>(
      "Test",
      wui::hori_alignment::center, wui::vert_alignment::center,
      "h1_text");
      wui::rect pos = { space, top, width - space, top + element_height };
      Text->set_position(pos);

      Assert::AreEqual(pos, Text->position());
    }
  };
}
```

Рисунок 4.1 – Модульный тест класса Text

4.3 Системное тестирование разработанного приложения с помошью фреймворка

В целях проверки работоспособности программно-информационной системы было проведено системное тестирование. Описание тестов,их результаты и скриншоты экрана представлены в данном разделе.

Функция: смена темы.

Условия: нажата кнопка смены темы.

Входные данные: данные светлой темы, данные темной темы.

Результат: изменение цвета окна и внутренних элементов (рисунок 4.2).

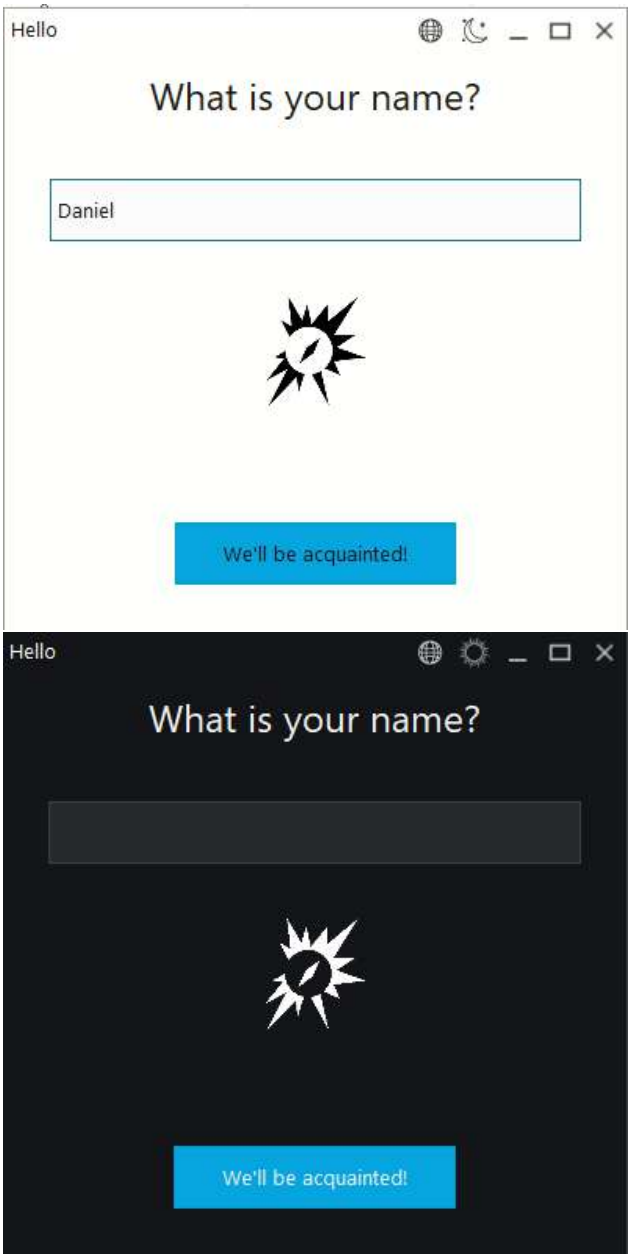


Рисунок 4.2 – Смена темы

Функция: смена языка.

Условия: нажата кнопка смены языка.

Входные данные: данные для русского языка, данные для английского языка.

Результат: изменение текста окна и внутренних элементов (рисунок 4.3).

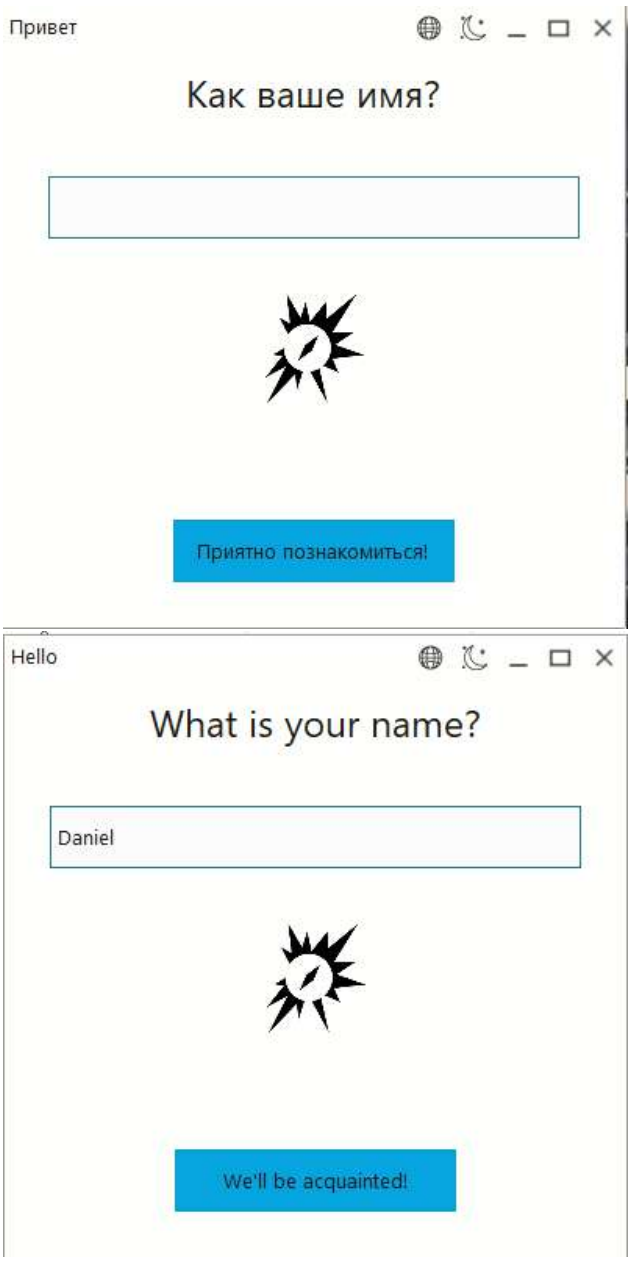


Рисунок 4.3 – Смена языка

Функция: всплывающее окно.

Условия: введен текст, нажата кнопка подтверждения.

Входные данные: текст из поля ввода.

Результат: появление окна приветствия (рисунок 4.4).

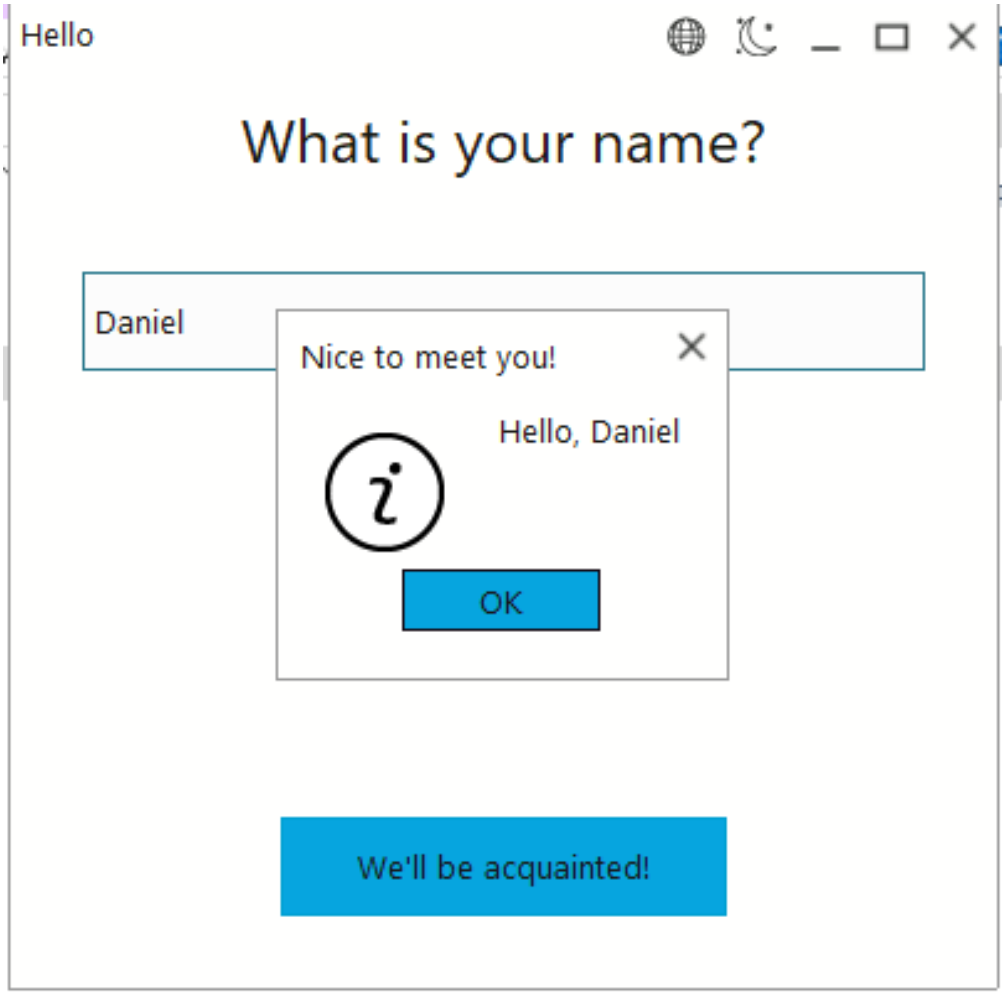


Рисунок 4.4 – Всплывающее окно

Функция: окно подтверждения.

Условия: нажата кнопка выхода из приложения.

Входные данные: событие нажатия кнопки мыши на кнопку закрытия.

Результат: появление окна закрытия (рисунок 4.5).

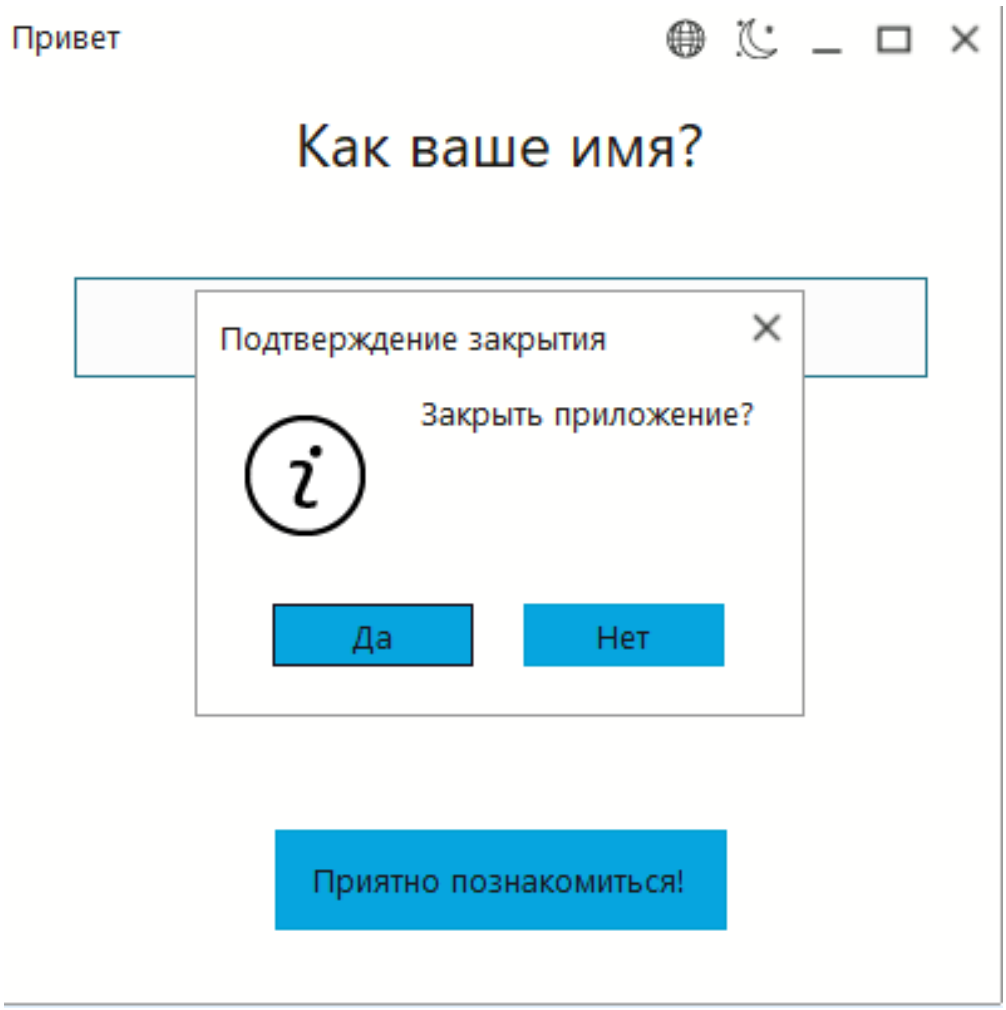


Рисунок 4.5 – Окно подтверждения закрытия

Функция: подсказка.

Условия: наведение курсора мыши на иконку кнопки.

Входные данные: положение курсора, обработчик события подсказки.

Результат: появление подсказки под кнопкой (рисунок 4.6).

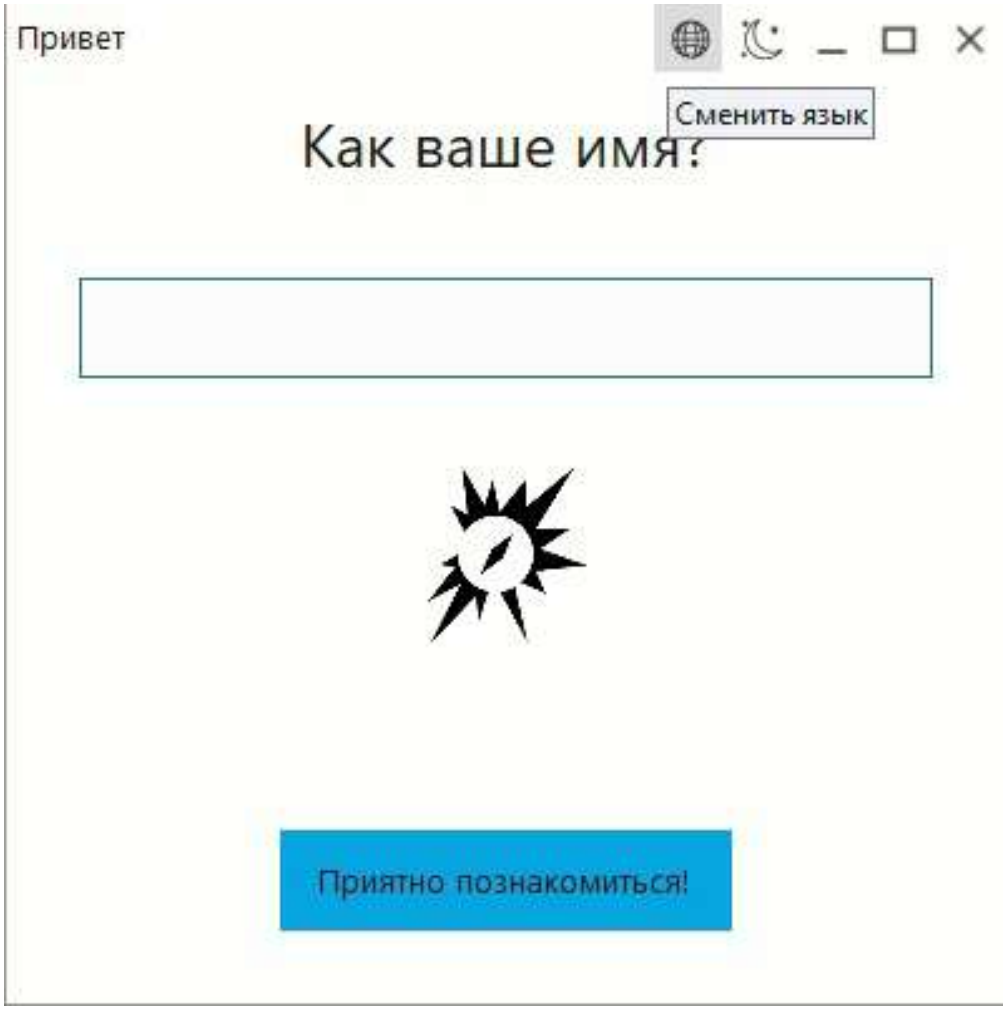


Рисунок 4.6 – Подсказка

Функция: развернуть окно во весь экран.

Условия: нажата кнопка «Развернуть».

Входные данные: размер экрана, обработчик события кнопки «Развернуть», состояние окна.

Результат: окно развернулось на полный экран, изменился размер внутренних элементов окна. (рисунок 4.7).

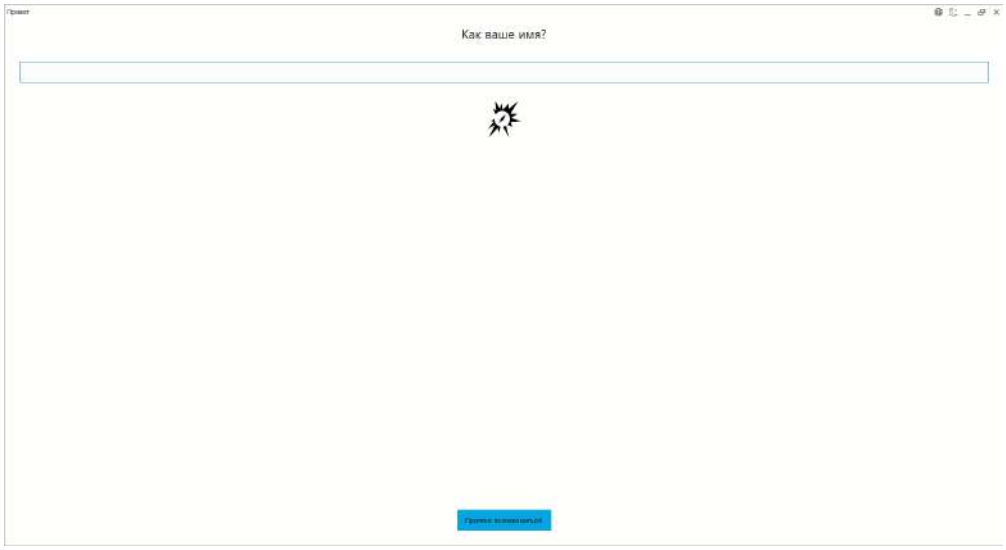


Рисунок 4.7 – Приложение, развернутое на полный экран

Функция: ползунок.

Условия: изменено положение ползунка.

Входные данные: позиция ползунка, обработчик события ползунка.

Результат: на экране появилась координата x, в случае изменения положения горизонтального ползунка, и координата y при изменении положения вертикального ползунка. . (рисунок 4.8).

Рисунок 4.8 – Ползунок

Функция: поле для ввода.

Условия: в поле для ввода введен текст.

Входные данные: фокус поля для ввода, обработчик события клавиатуры.

Результат: на экране появился текст. . (рисунок 4.9).

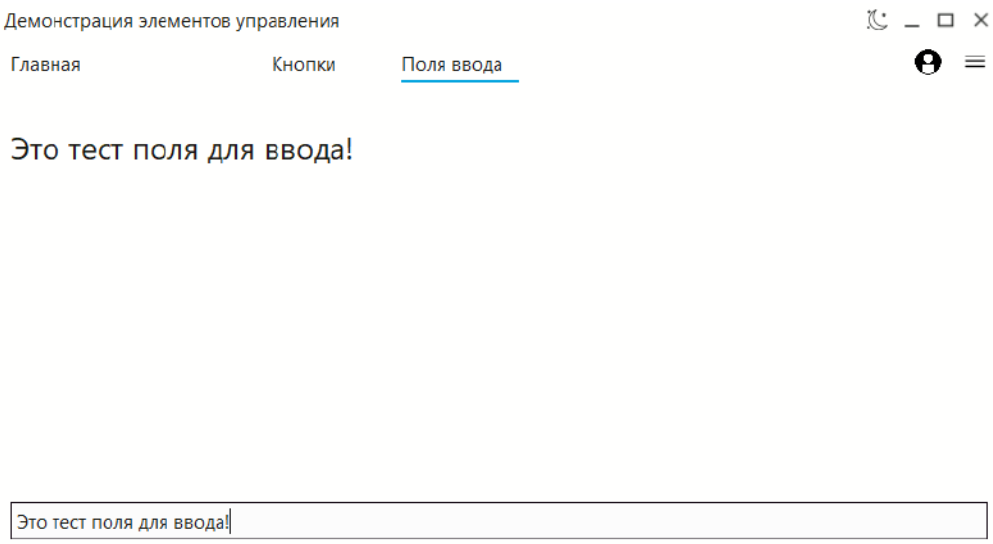


Рисунок 4.9 – Поле для ввода

Функция: нажатие кнопки.

Условия: нажатие кнопки мыши на кнопку.

Входные данные: событие нажатия кнопки мыши на кнопку, нажатая кнопка.

Результат: появилось текстовое сообщение о том какая кнопка нажата. (рисунок 4.10).

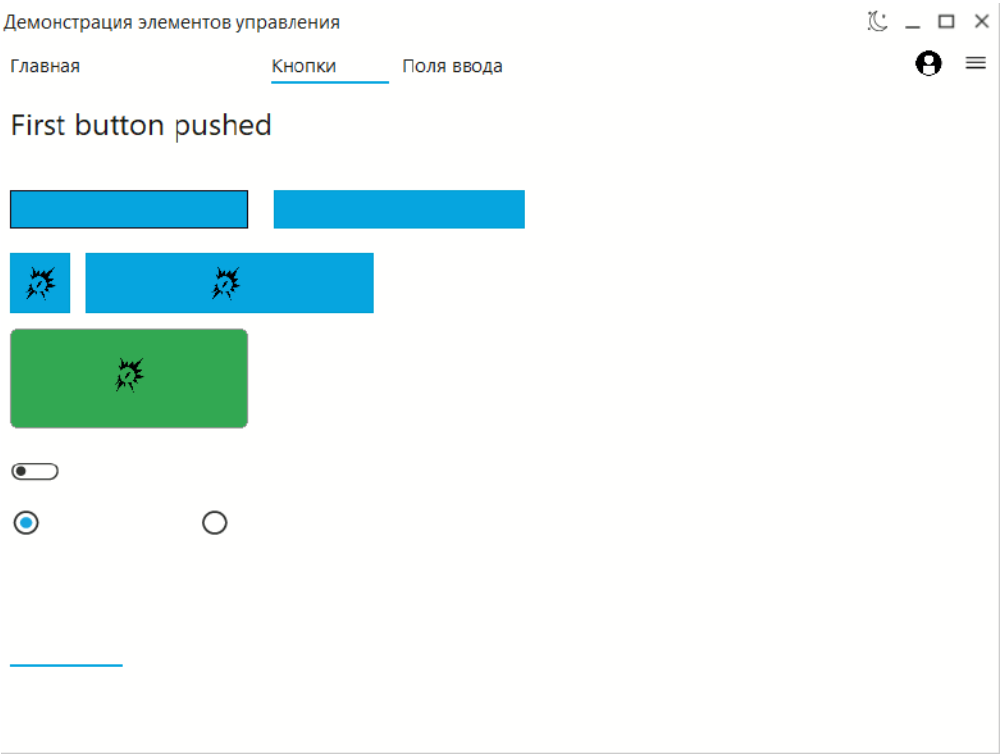


Рисунок 4.10 – Нажатие кнопки

Функция: нажатие переключателя.

Условия: нажатие кнопки мыши на переключатель.

Входные данные: событие нажатия кнопки мыши на переключатель, нажатый переключатель.

Результат: появилось текстовое сообщение о том, что нажат переключатель. (рисунок 4.11).

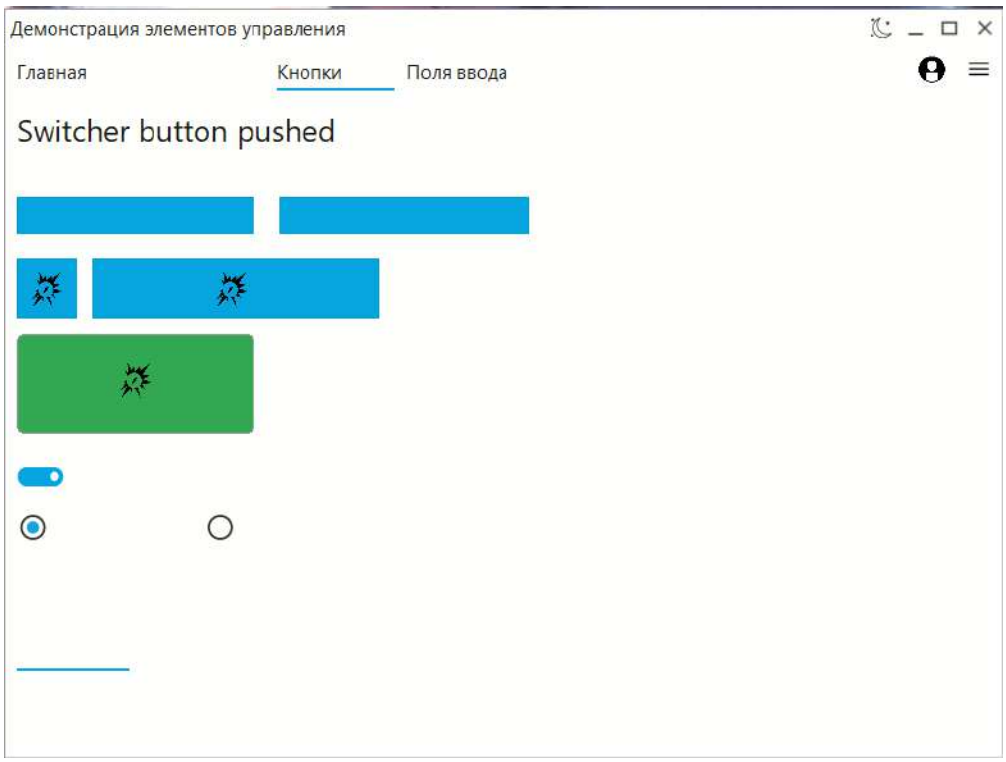


Рисунок 4.11 – Нажатие переключателя

Функция: радиокнопка.

Условия: нажатие кнопки мыши на радиокнопку.

Входные данные: событие нажатия кнопки мыши на радиокнопку, нажатая радиокнопка.

Результат: появилось текстовое сообщение о том, что нажата радиокнопка. (рисунок 4.12).

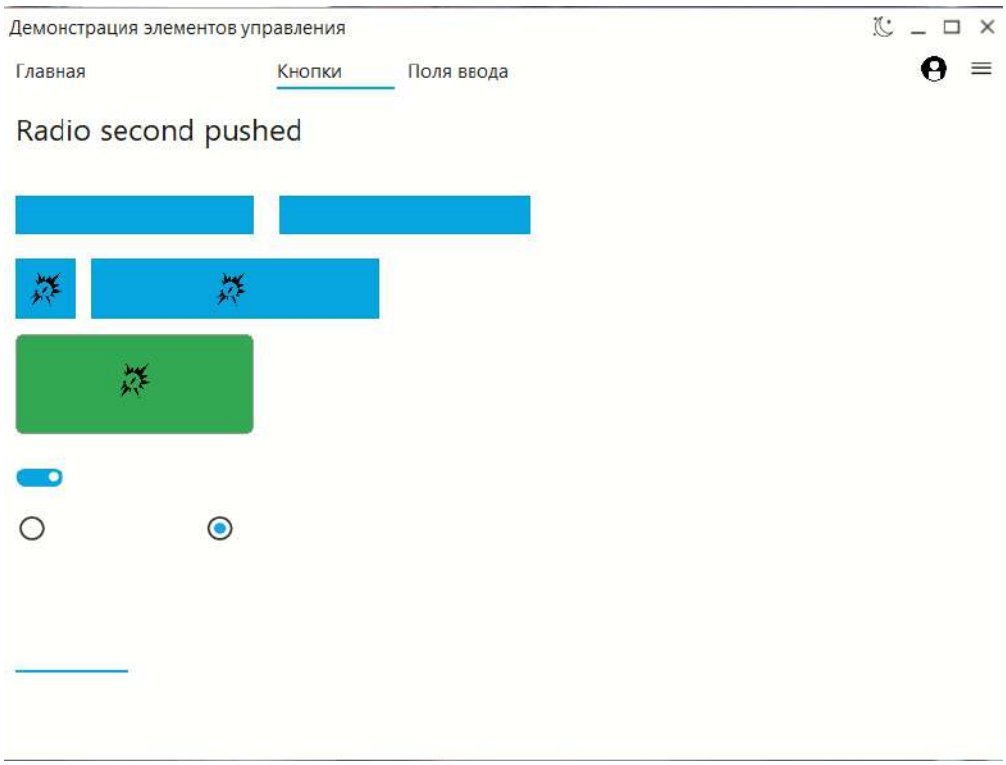


Рисунок 4.12 – Нажатие радиокнопки

Функция: смена страницы.

Условия: нажатие кнопки мыши на страницу.

Входные данные: событие нажатия кнопки мыши на страницу, нажатая страница.

Результат: появилось текстовое сообщение о том, что нажата страница. (рисунок 4.13).

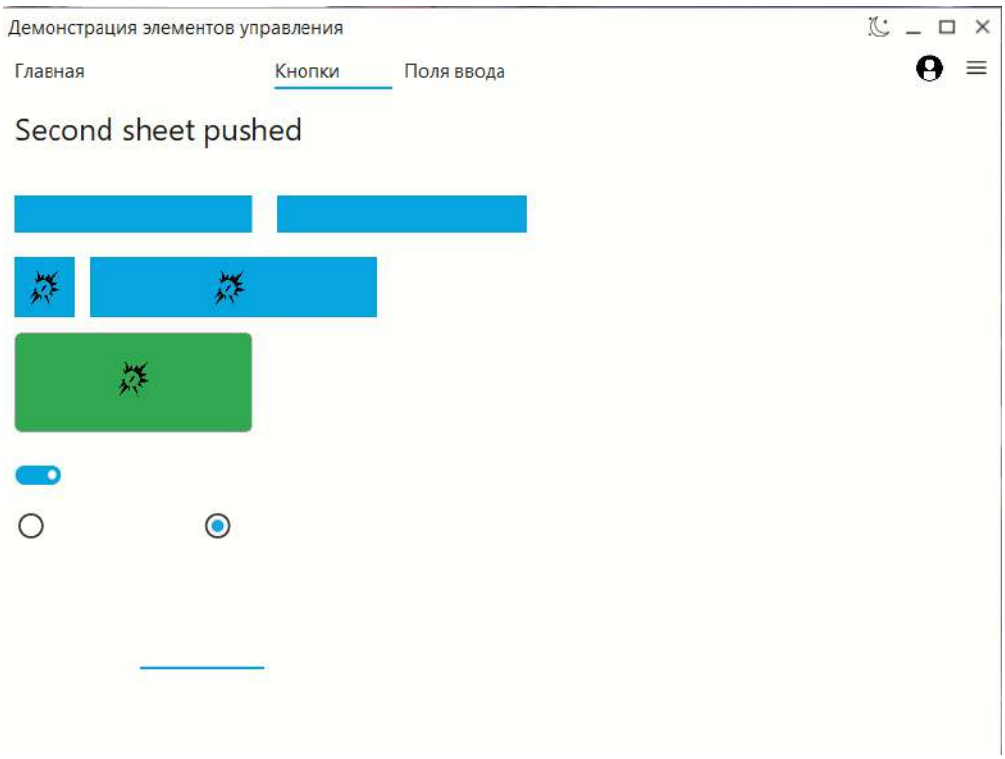


Рисунок 4.13 – Кнопка смены страницы

Функция: ползунок настройки и полоса прогресса.

Условия: передвинут ползунок настройки.

Входные данные: положение ползунка настроек, обработчик событий.

Результат: при изменении положения ползунка настроек заполняется полоса прогресса. (рисунок 4.14).

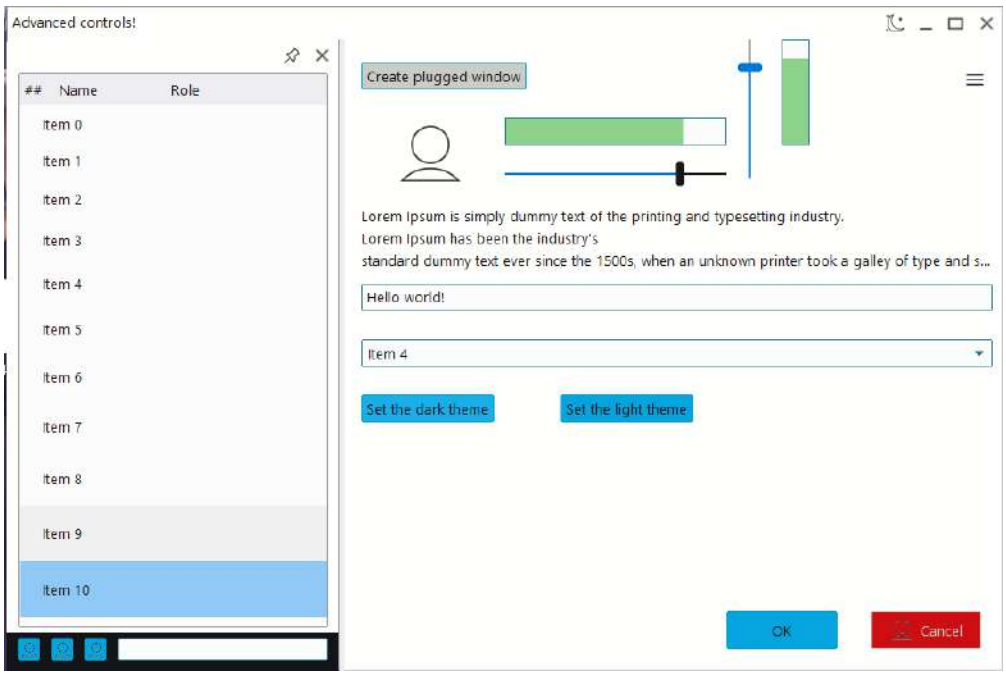


Рисунок 4.14 – Ползунок настройки и полоса прогресса

Функция: выпадающий список.

Условия: нажатие кнопки мыши на выпадающий список.

Входные данные: нажатие кнопки мыши на выпадающий список, обработчик события списка.

Результат: Раскрытие выпадающего списка. (рисунок 4.15).

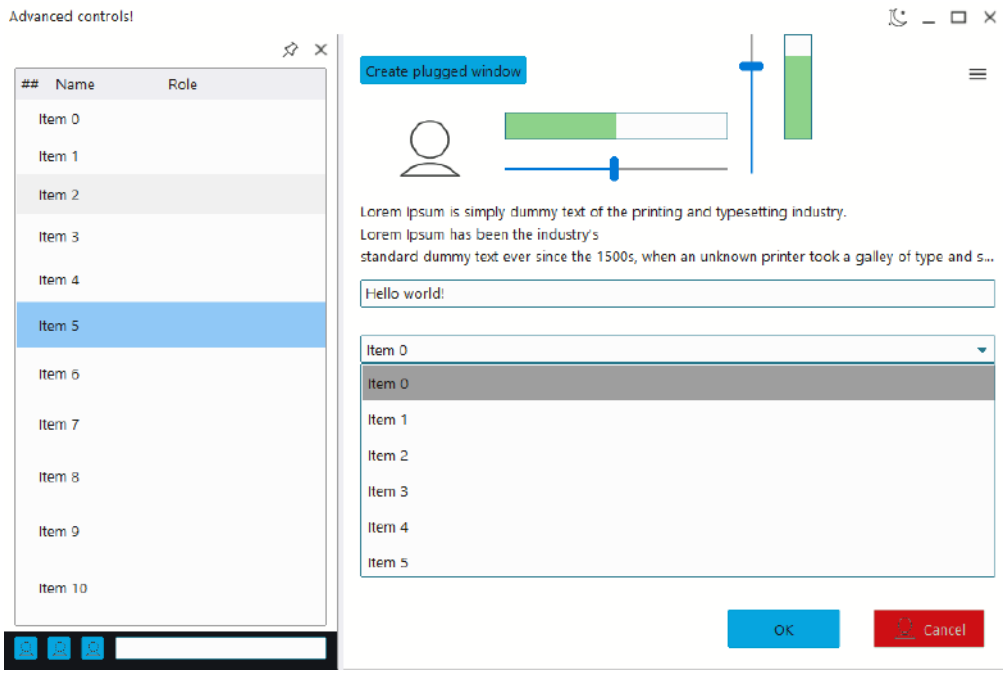


Рисунок 4.15 – Список

Функция: Меню.

Условия: нажата кнопка настроек.

Входные данные: нажатие кнопки мыши на кнопку настроек, обработчик меню настроек.

Результат: на экране появилось меню. (рисунок 4.16).

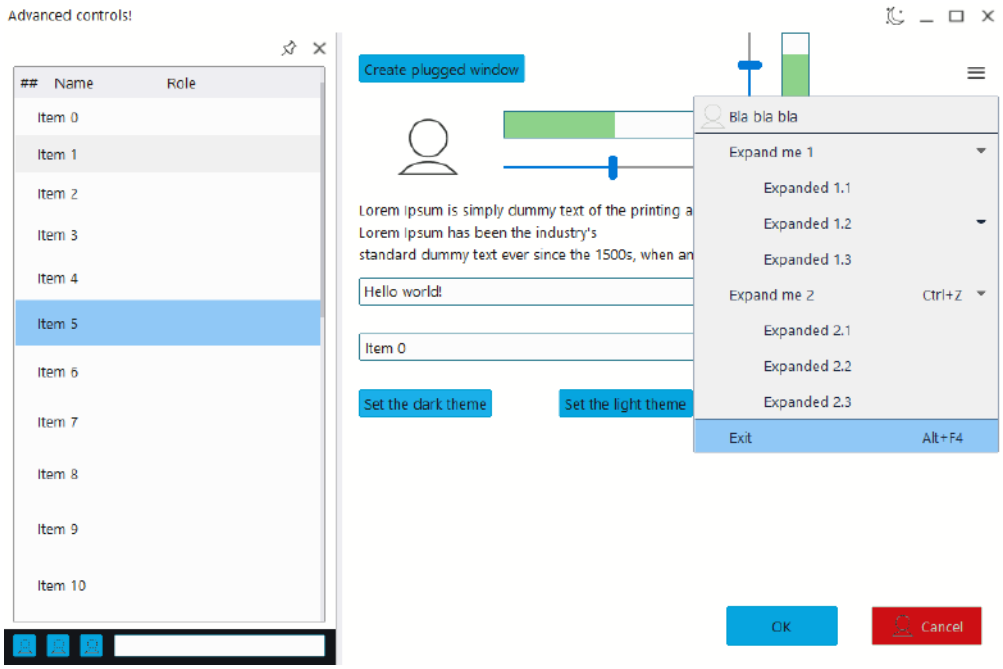


Рисунок 4.16 – Меню настроек

Функция: откреплление окна.

Условия: нажатие кнопки мыши на кнопку откреплления окна.

Входные данные: нажатие кнопки мыши на кнопку откреплления окна, позиция окна, состояние окна, обработчик события.

Результат: появилось откреплленное окно, которое можно двигать по экрану. (рисунок 4.17).

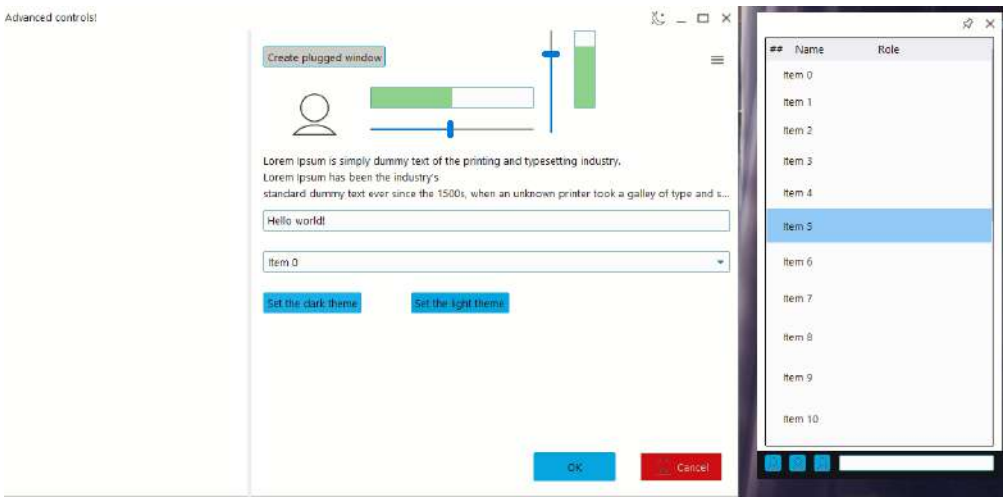


Рисунок 4.17 – Откребление окна

ЗАКЛЮЧЕНИЕ

В ходе исследования было выявлено, что проектирование фреймворка для создания графического пользовательского интерфейса играет ключевую роль в обеспечении удобства использования программного продукта. Разработка такого фреймворка позволяет значительно упростить процесс создания интерфейсов, повысить их качество и снизить затраты на разработку. Отличительной особенностью успешного фреймворка является его гибкость, расширяемость и простота в использовании. Дальнейшее развитие в этом направлении сможет улучшить опыт пользователей и повысить эффективность разработки программных продуктов.

К особенностям данного фреймворка можно отнести встроенную поддержку локалей и цветовых тем на основе json-схем позволяет легко создавать впечатляющие многоязычные приложения с разнообразными цветовыми и визуальными темами. Еще одним приимуществом является небольшой средний размер двоичного кода, что позволило значительно уменьшить итоговый вес приложений.

Основные результаты работы:

1. Проведен анализ предметной области. Выявлена необходимость использовать C++.

2. Разработана концептуальная модель фреймворка. Разработана модель данных системы. Определены требования к системе.

3. Спроектирована программная система для создания графических пользовательских интерфейсов

4. Реализован и протестирован фреймворк. Проведено модульное и системное тестирование.

Все требования, объявленные в техническом задании, были полностью реализованы, все задачи, поставленные в начале разработки проекта, были также решены.

Готовый рабочий проект представлен библиотекой. Фреймворк находится в публичном доступе, поскольку опубликован в сети Интернет.

Перспективой дальнейшей разработки является расширение списка поддерживаемых элементов управления и шаблонов программ.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Купер Алан. Интерфейс. Основы проектирования взаимодействия / А. Купер. – Санкт-Петербург: Питер, 2018. – 720 с. – ISBN 978-5-4461-0877-0. – Текст : непосредственный.

2. Машин В. А. Проектирование и дизайн пользовательского интерфейса / В.А. Машин, А.К. Гультяев. – Санкт-Петербург: Корона Принт, 2010. – 352 с. – ISBN 978-5-7931-0814-0. – Текст : непосредственный.

3. Gregory F. Rogers. Framework-Based Software Development in C++ / Gregory F. Rogers. – London: Pearson, 2008. – 400 с. – ISBN 978-0135333655. – Текст : непосредственный.

4. Прохоренок Николай. Qt 6. Разработка оконных приложений на C++ / Прохоренок Н.А. – Санкт-Петербург: БХВ, 2022. – 512 с. – ISBN 978-5-9775-1180-3. – Текст : непосредственный.

5. Скотт Адам Д. Разработка на JavaScript. Построение кроссплатформенных приложений с помощью GraphQL, React, React Native и Electron / Скотт А. Д. – Санкт-Петербург: Питер, 2021. – 320 с. – ISBN 978-5-4461-1462-7. – Текст : непосредственный.

6. Прохоренок Николай. JavaFX / Прохоренок Н.А. – Санкт-Петербург: БХВ, 2019. – 768 с. – ISBN 978-5-9775-4072-8. – Текст : непосредственный.

7. Андрей Костельцов. GTK+. Разработка переносимых графических интерфейсов / Костельцов А.В. – Санкт-Петербург: БХВ, 2002. – 362 с. – ISBN 5-94157-161-5. – Текст : непосредственный.

8. Мак-Дональд Мэтью. WPF: Windows Presentation Foundation в .NET 4.5 с примерами на C# 5.0 для профессионалов. / Мак-Дональд М. – Москва: Вильямс, 2013. – 1024 с. – ISBN 978-5-8459-1854-3. – Текст : непосредственный.

9. Максимчук Р.А. UML для простых смертных /Максимчук Р.А., Нейбург Э.Дж. – Москва: Лори, 2024. – 300 с. – ISBN 978-5-85582-434-6. – Текст : непосредственный.

10. Гинсбург Дэн. OpenGL ES 3.0. Руководство разработчика / Гинсбург Д. – Санкт-Петербург: ДМК Пресс, 2015. – 449 с. – ISBN 978-5-97060-256-0. – Текст : непосредственный.

11. Селлерс Грэхем. Vulkan. Руководство разработчика / Селлерс Г. – Санкт-Петербург: ДМК Пресс, 2017. – 394 с. – ISBN 978-5-97060-486-1. – Текст : непосредственный.

12. E. Balagurusamy. Object Oriented Programming With C++ / Balagurusamy E. – New York Sity : McGraw-Hill Education (India) Pvt Limited, 2008 – 637 с. – ISBN 978-0070669079. – Текст : непосредственный.

13. Paul Deitel C++ How to Program / Deitel P., Deitel H. – London: Pearson, 2016. – 1080 с. – ISBN 978-0134448237. – Текст : непосредственный.

14. Мейерс Скотт. Эффективный и современный C++:42 рекомендации по использованию C++11 и C++14 / Мейерс С. – Москва : Вильямс, 2019. – 304 с. – ISBN 978-5-907114-67-8. – Текст : непосредственный.

15. Дьюхерст Стивен. Скользкие места C++. Как избежать проблем при проектировании и компиляции ваших программ / Дьюхерст С. – Москва : ДМК Пресс, 2017. – 264 с. – ISBN 978-5-97060-475-5. – Текст : непосредственный.

16. Грегори Кейт. Красивый C++: 30 главных правил чистого, безопасного и быстрого кода / Кейт Г., Дэвидсон Дж. Г. – Санкт-Петербург : Питер, 2023. – 368 с. – ISBN 978-5-4461-2272-1. – Текст : непосредственный.

17. Спрол Антон. Думай как программист. Креативный подход к созданию кода. C++ версия / Спрол А. – Москва : Эксмо, 2018. – 272 с. – ISBN 978-5-04-089838-1. – Текст : непосредственный.

18. Полухин Антон. Разработка приложений на C++ с использованием Boost. Рецепты, упрощающие разработку вашего приложения / Полухин А. - Санкт-Петербург : ДМК Пресс, 2020. - 346 с. -ISBN 978-5-97060-868-5. – Текст : непосредственный.

19. Шилдт Герберт. C++ для начинающих / Шилдт Г. – Санкт-Петербург: Питер, 2024. – 608 с. – ISBN 978-5-4461-1821-2. – Текст : непосредственный.

20. Евдокимов Петр. C++ на примерах. Практика, практика и только практика / Евдокимов П. В., Орленко П. А. – Санкт-Петербург: Наука и техника, 2022. – 288 с. – ISBN 978-5-94387-772-8. – Текст : непосредственный.

21. Страуструп Бьёрн. Дизайн и эволюция языка-C++. Второе издание / Страуструп Б. – Санкт-Петербург: ДМК Пресс, 2016. – 448 с. – ISBN 978-5-97060-419-9. – Текст : непосредственный.

22. Пахомов Б.И. C/C++ и MS Visual C++ 2010 для начинающих / Пахомов Б.И. – Санкт-Петербург: БХВ, 2011. – 736 с. – ISBN 978-5-9775-0599-4. – Текст : непосредственный.

23. Побегайло А. П. C/C++ для студента / Побегайло А. П. – Санкт-Петербург: БХВ, 2006. – 525 с. – ISBN 978-5-94157-647-1. – Текст : непосредственный.

ПРИЛОЖЕНИЕ А

Представление графического материала

Графический материал, выполненный на отдельных листах, изображен на рисунках А.1–А.8.

Сведения о ВКРБ

Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ПО ПРОГРАММЕ БАКАЛАВРИАТА

«Программная платформа для создания элементов графического пользовательского интерфейса»

Руководитель ВКРБ
к.т.н, доцент
Белова Татьяна Михайловна

Автор ВКРБ
студент группы ПО-02б
Шлифер Даниил Георгиевич

				ВКРБ 2068443.09.03.04.24.013		
	Фамилия И. О.	Подпись	Дата	Сведения о ВКРБ	Лит.	Масса	Масштаб
Автор работы	Шлифер Д.Г.						
Руководитель	Белова Т.М.				Лист 1	Листов 6
Нормоконтроль	Чаплыгин А.А.					
				Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-026	

Рисунок А.1 – Сведения о ВКРБ

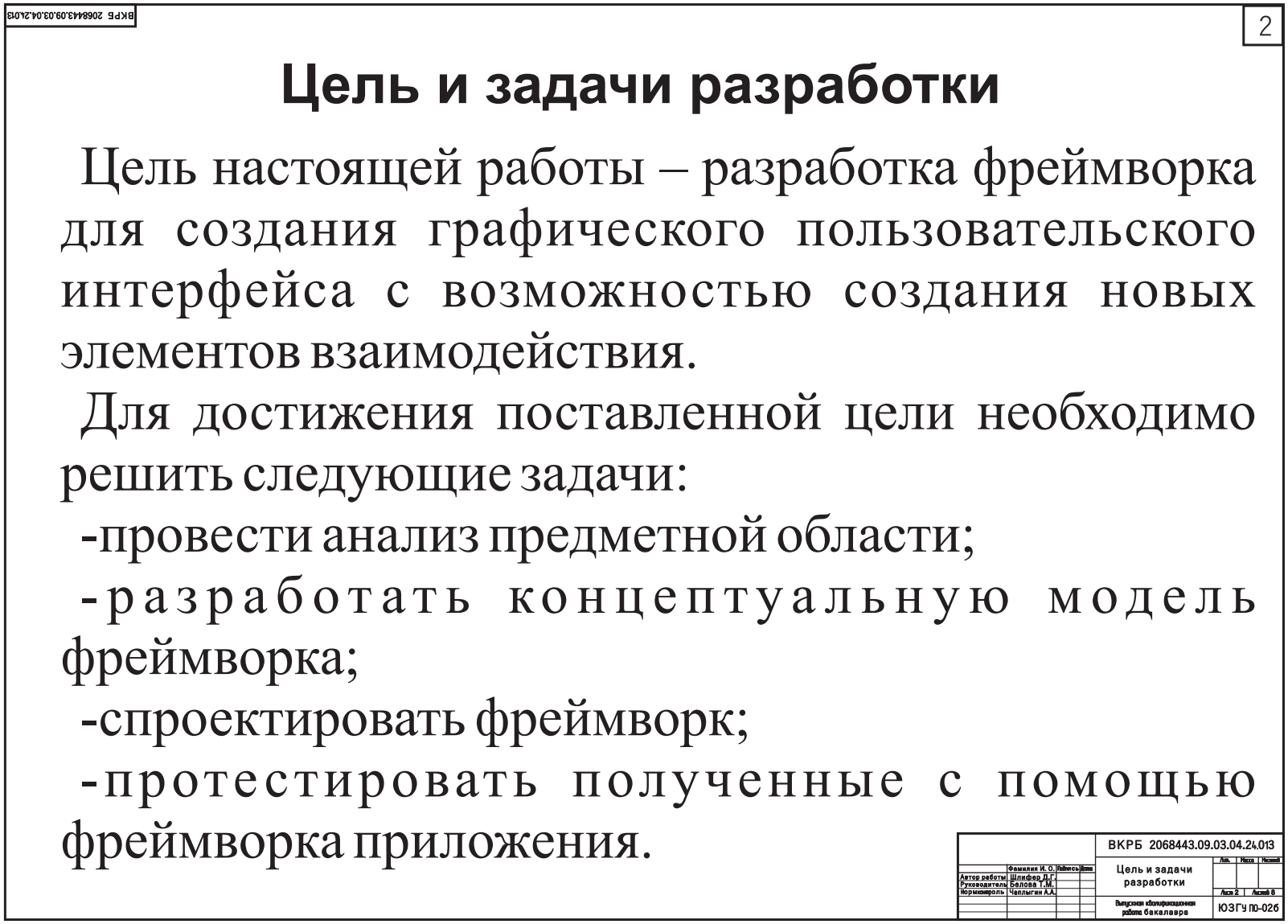
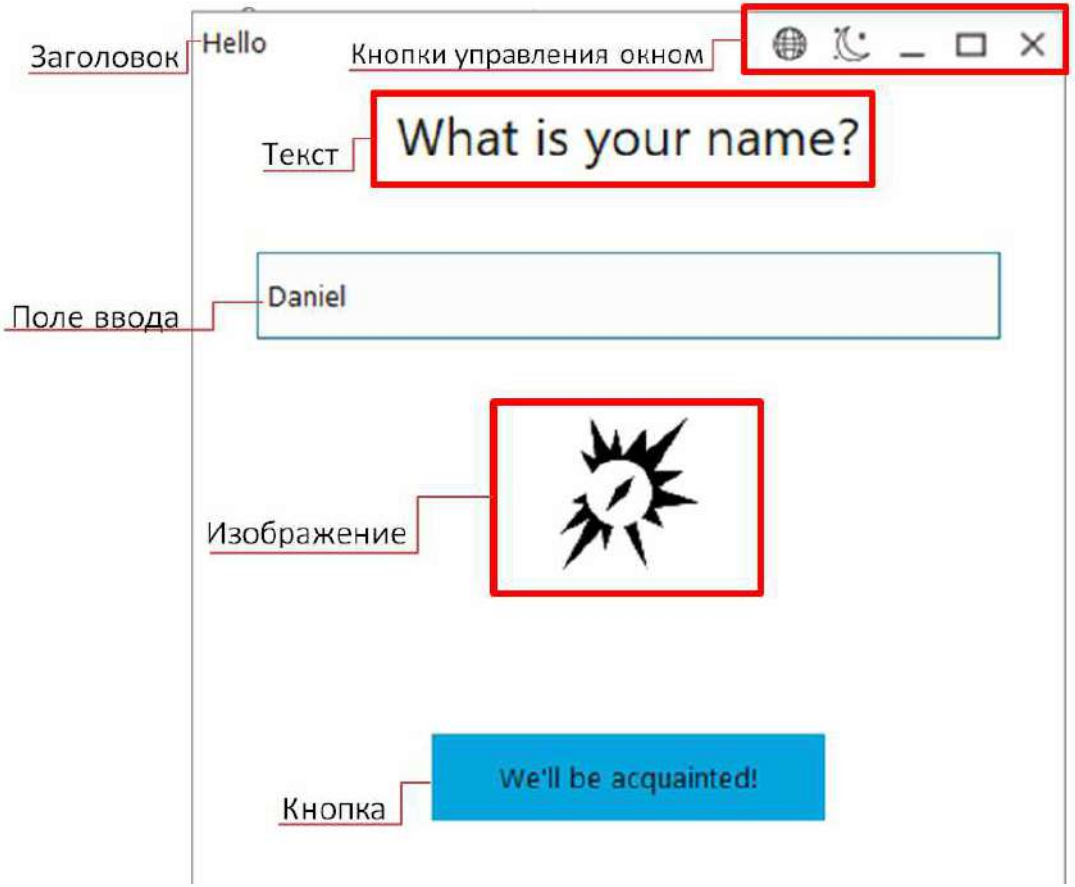


Рисунок А.2 – Цель и задачи разработки

Рисунок А.3 – Концептуальная модель фреймворка

Интерфейс окна



				ВКРБ 2068443.09.03.04.24.013	
	Фамилия И. О.	Подпись	Дата		
Автор работы	Шлифер Д.Г.			Интерфейс окна	Лит. / Масса / Масштаб
Руководитель	Белова Т.М.				
Нормоконтроль	Чаплыгин А.А.				Лист 4 / Листов 8
				Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-026

Рисунок А.4 – Интерфейс окна

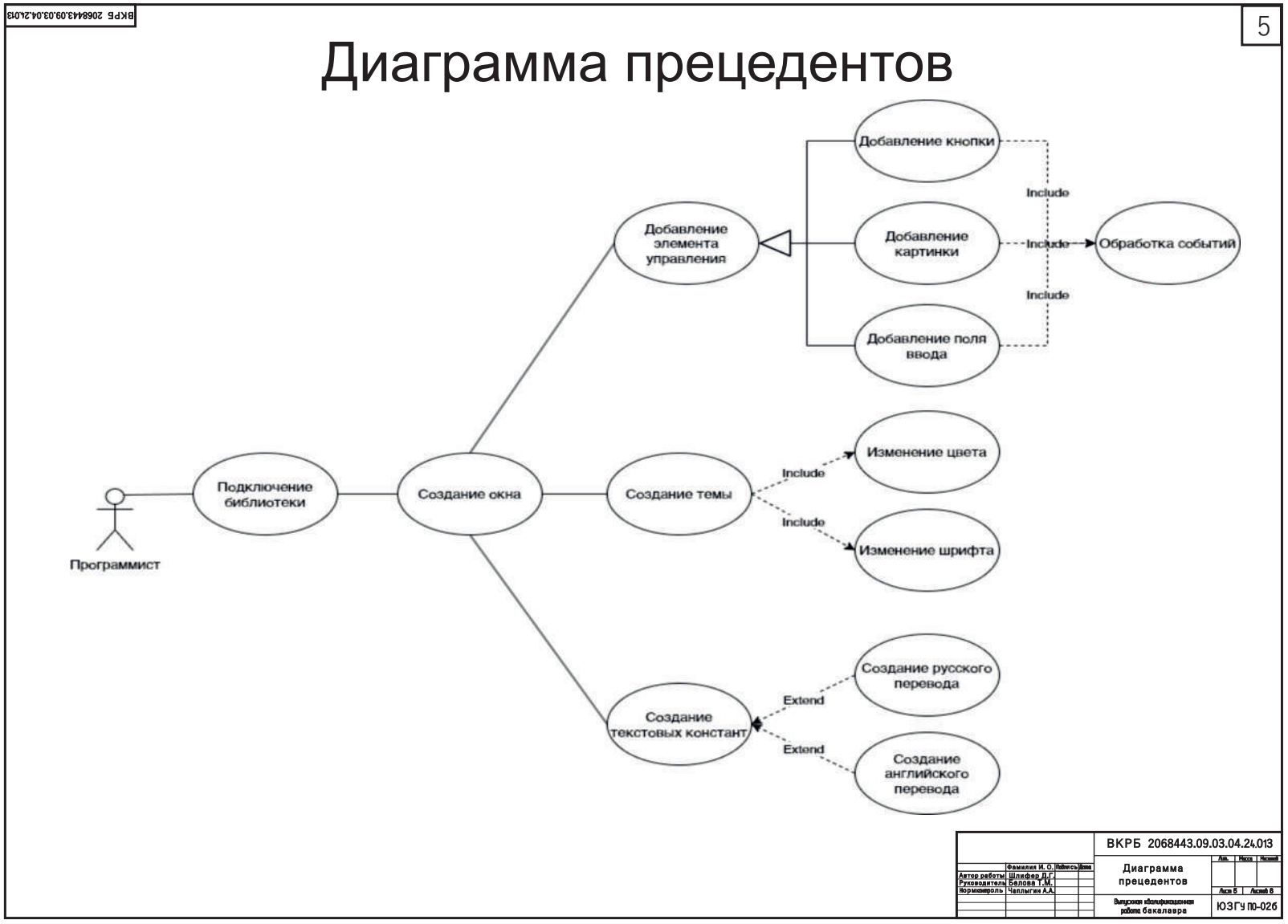
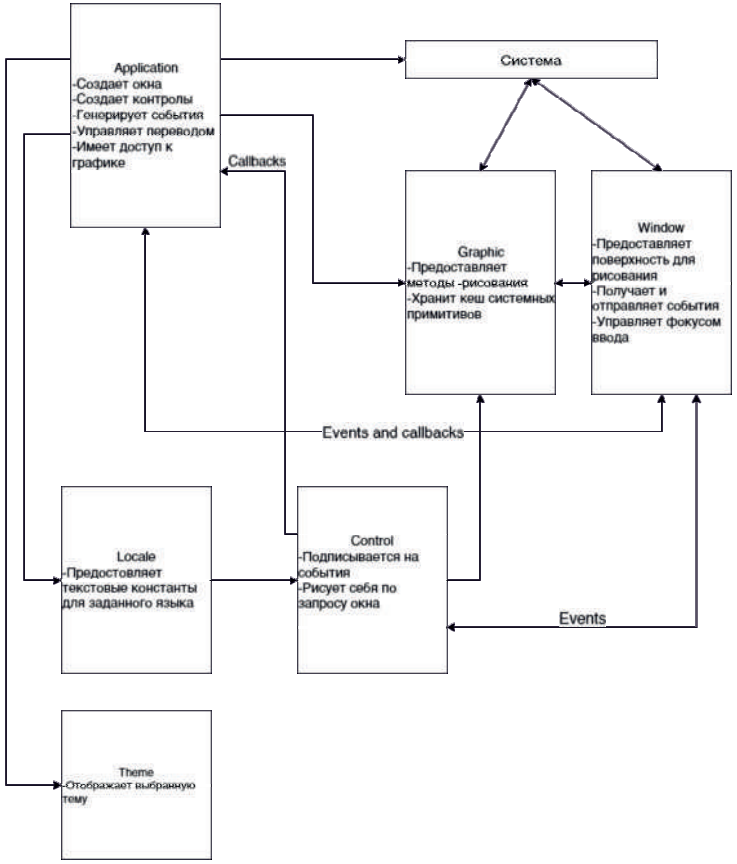


Рисунок А.5 – Диаграмма прецидентов

Схема обмена данных

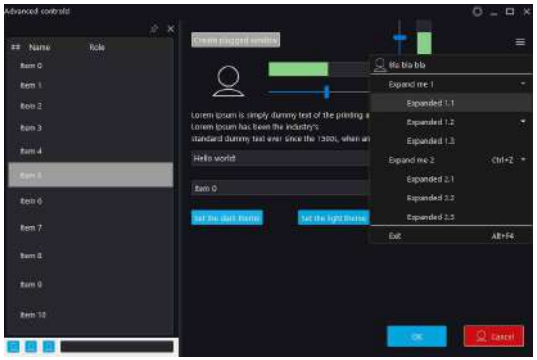


				ВКРБ 2068443.09.03.04.24.013	
	Фамилия И. О.	Подпись	Дата		
Автор работы	Шлифер Д.Г.			Схема обмена данных	Лит. / Масса / Масштаб
Руководитель	Белова Т.М.				
Нормоконтроль	Чаплыгин А.А.				Лист 6 / Листов 6
				Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-026

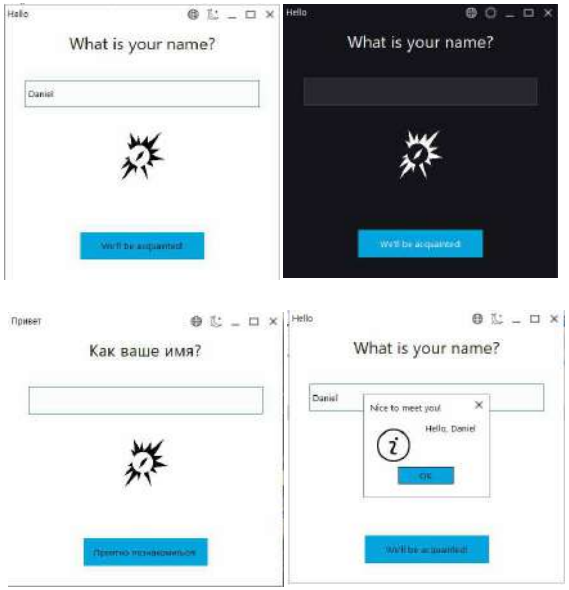
Рисунок А.6 – Схема обмена данных

Тестирование фреймворка

Тестирование кнопок



Тестирование меню



Тестирование смены темы и языка.
Тестирование сообщений

				ВКРБ 2068443.09.03.04.24.013		
	Фамилия И. О.	Подпись	Дата	Тестирование	Лит.	Масса	Масштаб
Автор работы	Шлифер Д.Г.						
Руководитель	Белова Т.М.				Лист 7	Листов 8	
Нормоконтроль	Чаплыгин А.А.						
				Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-026		

Рисунок А.7 – Тестирование

Заключение

В ходе исследования было выявлено, что проектирование фреймворка для создания графического пользовательского интерфейса играет ключевую роль в обеспечении удобства использования программного продукта. Разработка такого фреймворка позволяет значительно упростить процесс создания интерфейсов, повысить их качество и снизить затраты на разработку. Отличительной особенностью успешного фреймворка является его гибкость, расширяемость и простота в использовании. Дальнейшее развитие в этом направлении сможет улучшить опыт пользователей и повысить эффективность разработки программных продуктов.

К особенностям данного фреймворка можно отнести встроенную поддержку локалей и цветовых тем на основе json-схем позволяет легко создавать впечатляющие многоязычные приложения с разнообразными цветовыми и визуальными темами. Еще одним приимуществом является небольшой средний размер двоичного кода, что позволило значительно уменьшить итоговый вес приложений.

Основные результаты работы:

- Проведен анализ предметной области. Выявлена необходимость использовать C++.
- Разработана концептуальная модель фреймворка. Разработана модель данных системы. Определены требования к системе.
- Спроектирована программная система для создания графических пользовательских интерфейсов
- Реализован и протестирован фреймворк. Проведено модульное и системное тестирование.

Все требования, объявленные в техническом задании, были полностью реализованы, все задачи, поставленные в начале разработки проекта, были также решены.

Рисунок А.8 – Заключение

ПРИЛОЖЕНИЕ Б

Фрагменты исходного кода программы

framework.cpp

```
#include <wui/framework/framework.hpp>

#include <wui/framework/framework_win_impl.hpp>

#include <wui/framework/i_framework.hpp>

#ifdef _WIN32
#include <windows.h>
#include <gdiplus.h>
#endif

#include <memory>
#include <iostream>

namespace wui
{

namespace framework
{

static std::shared_ptr<i_framework> instance = nullptr;

/// Interface

void init()
{
        Gdiplus::GdiplusStartupInput gdiplusStartupInput;
        ULONG_PTR gdiplusToken;
        Gdiplus::GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);
}

void run()
{
    if (instance)
    {
        return;
    }
    instance = std::make_shared<framework_win_impl>();

    instance->run();
}

void stop()
{
    if (instance)
    {
        instance->stop();
    }
    instance.reset();
```

```
}

bool runned()
{
    return instance != nullptr;
}

error get_error()
{
    if (instance)
    {
        return instance->get_error();
    }
    return {};
}

}

}
```

window.cpp

```
#include <wui/window/window.hpp>

#include <wui/graphic/graphic.hpp>

#include <wui/theme/theme.hpp>
#include <wui/locale/locale.hpp>

#include <wui/control/button.hpp>

#include <wui/common/flag_helpers.hpp>

#include <wui/system/tools.hpp>
#include <wui/system/wm_tools.hpp>

#include <boost/nowide/convert.hpp>

#include <algorithm>
#include <set>
#include <random>

#include <windowsx.h>

// Some helpers

void center_horizontally(wui::rect &pos, wui::system_context &context)
{
    RECT work_area;
    SystemParametersInfo(SPI_GETWORKAREA, 0, &work_area, 0);
    auto screen_width = work_area.right - work_area.left;
    pos.left = (screen_width - pos.right) / 2;
    pos.right += pos.left;
}

```

```
34 void center_vertically(wui::rect &pos, wui::system_context &context)
35 {
36     RECT work_area;
37     SystemParametersInfo(SPI_GETWORKAREA, 0, &work_area, 0);
38     auto screen_height = work_area.bottom - work_area.top;
39     pos.top = (screen_height - pos.bottom) / 2;
40     pos.bottom += pos.top;
41 }
42
43 namespace wui
44 {
45
46 window::window(std::string_view theme_control_name, std::shared_ptr<i_theme>
       theme_)
47     : context_{ 0 },
48     graphic_(context_),
49     controls(),
50     active_control(),
51     caption(),
52     position_(), normal_position(),
53     min_width(0), min_height(0),
54     window_style_(window_style::frame),
55     window_state_(window_state::normal), prev_window_state_(window_state_),
56     tcn(theme_control_name),
57     theme_(theme_),
58     showed_(true), enabled_(true), skip_draw_(false),
59     focused_index(0),
60     parent_(),
61     my_control_sid(), my_plain_sid(),
62     transient_window(), docked_(false), docked_control(),
63     subscribers_(),
64     moving_mode_(moving_mode::none),
65     x_click(0), y_click(0),
66     err{},
67     close_callback(),
68     control_callback(),
69     default_push_control(),
70   switch_lang_button(std::make_shared<button>(locale(tcn, cl_switch_lang),
         std::bind(&window::switch_lang, this), button_view::image, theme_image(
         ti_switch_lang), 24, button::tc_tool)),
71     switch_theme_button(std::make_shared<button>(locale(tcn, cl_light_theme),
           std::bind(&window::switch_theme, this), button_view::image,
         theme_image(ti_switch_theme), 24, button::tc_tool)),
72     pin_button(std::make_shared<button>(locale(tcn, cl_pin), std::bind(&
         window::pin, this), button_view::image, theme_image(ti_pin), 24,
         button::tc_tool)),
73     minimize_button(std::make_shared<button>("", std::bind(&window::minimize,
           this), button_view::image, theme_image(ti_minimize), 24, button::
         tc_tool)),
74     expand_button(std::make_shared<button>("", [this]() { window_state_ ==
         window_state::normal ? expand() : normal(); }, button_view::image,
         window_state_ == window_state::normal ? theme_image(ti_expand) :
         theme_image(ti_normal), 24, button::tc_tool)),
```

```
75      close_button(std::make_shared<button>("", std::bind(&window::destroy,
            this), button_view::image, theme_image(ti_close), 24, button::
            tc_tool_red)),
76      mouse_tracked(false)
77  {
78    switch_lang_button->disable_focusing();
79      switch_theme_button->disable_focusing();
80      pin_button->disable_focusing();
81      minimize_button->disable_focusing();
82      expand_button->disable_focusing();
83      close_button->disable_focusing();
84  }
85
86  window::~window()
87  {
88      auto parent__ = parent_.lock();
89      if (parent__)
90      {
91          parent__->remove_control(shared_from_this());
92      }
93
94      if (context_.hwnd)
95      {
96          DestroyWindow(context_.hwnd);
97      }
98  }
99
100 void window::add_control(std::shared_ptr<i_control> control, const rect &
        control_position)
101 {
102     if (std::find(controls.begin(), controls.end(), control) == controls.end
            ())
103     {
104         control->set_parent(shared_from_this());
105         control->set_position(control_position, false);
106         controls.emplace_back(control);
107
108         redraw(control->position());
109     }
110 }
111
112 void window::remove_control(std::shared_ptr<i_control> control)
113 {
114     if (!control)
115     {
116         return;
117     }
118
119     auto exists = std::find(controls.begin(), controls.end(), control);
120     if (exists != controls.end())
121     {
122         controls.erase(exists);
123     }
124
```

```
125     if (control == docked_control)
126     {
127         docked_control.reset();
128     }
129 
130     auto clear_pos = control->position();
131     control->clear_parent();
132     redraw(clear_pos, true);
133 }
134 
135 void window::bring_to_front(std::shared_ptr<i_control> control)
136 {
137     auto size = controls.size();
138     if (size > 1)
139     {
140         auto it = std::find(controls.begin(), controls.end(), control);
141         if (it != controls.end())
142         {
143             controls.erase(it);
144         }
145         controls.emplace_back(control);
146     }
147 }
148 
149 void window::move_to_back(std::shared_ptr<i_control> control)
150 {
151     auto size = controls.size();
152     if (size > 1)
153     {
154         auto it = std::find(controls.begin(), controls.end(), control);
155         if (it != controls.end())
156         {
157             controls.erase(it);
158         }
159         controls.insert(controls.begin(), control);
160     }
161 }
162 
163 void window::redraw(const rect &redraw_position, bool clear)
164 {
165     if (redraw_position.is_null() || skip_draw_)
166     {
167         return;
168     }
169 
170     auto parent__ = parent_.lock();
171     if (parent__)
172     {
173         parent__->redraw(redraw_position, clear);
174     }
175     else
176     {
177         RECT invalidatingRect = { redraw_position.left > 0 ? redraw_position.
            left : 0,
```

```
178             redraw_position.top > 0 ? redraw_position.top : 0,
179             redraw_position.right > 0 ? redraw_position.right : 0,
180             redraw_position.bottom > 0 ? redraw_position.bottom : 0 };
181         InvalidateRect(context_.hwnd, &invalidatingRect, clear ? TRUE : FALSE
                );
182     }
183 }
184
185 std::string window::subscribe(std::function<void(const event&)>
        receive_callback_, event_type event_types_, std::shared_ptr<i_control>
        control_)
186 {
187     std::random_device rd;
188     std::mt19937 gen(rd());
189     std::uniform_int_distribution<> uid(0, 61);
190
191     auto randchar = [&uid, &gen]() -> char
192     {
193         const char charset[] =
194             "0123456789"
195             "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
196             "abcdefghijklmnopqrstuvwxyz";
197         const size_t max_index = (sizeof(charset) - 1);
198         return charset[uid(gen)];
199     };
200
201     std::string id(20, 0);
202     std::generate_n(id.begin(), 20, randchar);
203
204     subscribers_.emplace_back(event_subscriber{ id, receive_callback_,
            event_types_, control_ });
205     return id;
206 }
207
208 void window::unsubscribe(std::string_view subscriber_id)
209 {
210     auto it = std::find_if(subscribers_.begin(), subscribers_.end(), [&
            subscriber_id](const event_subscriber &es) {
211         return es.id == subscriber_id;
212     });
213     if (it != subscribers_.end())
214     {
215         subscribers_.erase(it);
216     }
217 }
218
219 system_context &window::context()
220 {
221     auto parent__ = parent_.lock();
222     if (!parent__)
223     {
224         return context_;
225     }
226     else
```

```
227     {
228         return parent__->context();
229     }
230 }
231
232 void window::draw(graphic &gr, const rect &paint_rect)
233 {
234     /// drawing the child window
235
236     if (!showed_)
237     {
238         return;
239     }
240
241     auto window_pos = position();
242
243     gr.draw_rect(window_pos,
244         theme_color(tcn, tv_background, theme_),
245         theme_color(tcn, tv_background, theme_),
246         theme_dimension(tcn, tv_border_width, theme_),
247         theme_dimension(tcn, tv_round, theme_)
248     );
249
250     if (flag_is_set(window_style_, window_style::title_showed))
251     {
252         gr.draw_text({ window_pos.left + 10, window_pos.top + 10, 0, 0 },
253             caption,
254             theme_color(tcn, tv_text, theme_),
255             theme_font(tcn, tv_caption_font, theme_));
256     }
257
258     std::vector<std::shared_ptr<i_control>> topmost_controls;
259
260     for (auto &control : controls)
261     {
262         if (control->position().in(paint_rect))
263         {
264             if (!control->topmost())
265             {
266                 control->draw(gr, paint_rect);
267             }
268             else
269             {
270                 topmost_controls.emplace_back(control);
271             }
272         }
273     }
274
275     for (auto &control : topmost_controls)
276     {
277         control->draw(gr, paint_rect);
278     }
279
280     if (flag_is_set(window_style_, window_style::border_left) &&
```

```
281         flag_is_set(window_style_, window_style::border_top) &&
282         flag_is_set(window_style_, window_style::border_right) &&
283         flag_is_set(window_style_, window_style::border_bottom))
284     {
285         gr.draw_rect(window_pos,
286             theme_color(tcn, tv_border, theme_),
287             make_color(0, 0, 0, 255),
288             theme_dimension(tcn, tv_border_width, theme_),
289             theme_dimension(tcn, tv_round, theme_)
290         );
291     }
292     else
293     {
294         draw_border(gr);
295     }
296 }
297
298 void window::receive_control_events(const event &ev)
299 {
300     /// Here we receive events from the parent window and relay them to our
        child controls
301
302     if (!showed_)
303     {
304         return;
305     }
306
307     switch (ev.type)
308     {
309         case event_type::mouse:
310             send_mouse_event(ev.mouse_event_);
311         break;
312         case event_type::keyboard:
313         {
314             auto control = get_focused();
315             if (control)
316             {
317                 send_event_to_control(control, ev);
318             }
319         }
320         break;
321         case event_type::internal:
322             switch (ev.internal_event_.type)
323             {
324                 case internal_event_type::set_focus:
325                     change_focus();
326                 break;
327                 case internal_event_type::remove_focus:
328                 {
329                     size_t focusing_controls = 0;
330                     for (const auto &control : controls)
331                     {
332                         if (control->focused())
333                         {
```

```
                            event ev;
                            ev.type = event_type::internal;
                            ev.internal_event_ = internal_event{
                                internal_event_type::remove_focus };
                            send_event_to_control(control, ev);

                            ++focused_index;
                        }

                        if (control->focusing())
                        {
                            ++focusing_controls;
                        }
                    }

                    if (focused_index > focusing_controls)
                    {
                        focused_index = 0;
                    }
                }
                break;
                case internal_event_type::execute_focused:
                    execute_focused();
                break;
            }
        break;
        default: break;
    }
}

void window::receive_plain_events(const event &ev)
{
    if (ev.type == event_type::internal && ev.internal_event_.type == wui::
        internal_event_type::size_changed)
    {
        int32_t w = ev.internal_event_.x, h = ev.internal_event_.y;
        if (docked_)
        {
            int32_t left = (w - position_.width()) / 2;
            int32_t top = (h - position_.height()) / 2;

            auto new_position = position_;
            new_position.put(left, top);
            set_position(new_position, false);
        }

        return;
    }

    send_event_to_plains(ev);
}

void window::set_position(const rect &position__, bool redraw_)
{
```

```
386     auto old_position = position_;
387     auto position___ = position__;
388 
389     if (context_.hwnd)
390     {
391         if (position___.left == -1)
392         {
393             center_horizontally(position___, context_);
394         }
395         if (position___.top == -1)
396         {
397             center_vertically(position___, context_);
398         }
399 
400         position_ = position___;
401 
402     SetWindowPos(context_.hwnd, NULL, position___.left, position___.top,
        position___.width(), position___.height(), NULL);
403         redraw({ 0, 0, position___.width(), position___.height() }, true);
404     }
405 
406     auto parent__ = parent_.lock();
407     if (parent__)
408     {
409         auto parent_pos = parent__->position();
410 
411         auto left = position___.left;
412         if (left == -1)
413         {
414             left = (parent_pos.width() - position___.width()) / 2;
415         }
416         auto top = position___.top;
417         if (top == -1)
418         {
419             top = (parent_pos.height() - position___.height()) / 2;
420         }
421 
422         position_ = { left, top, left + position___.width(), top +
            position___.height() };
423 
424         skip_draw_ = true;
425         send_internal(internal_event_type::size_changed, position_.width(),
            position_.height());
426 
427         if (old_position.width() != position_.width())
428         {
429             update_buttons();
430         }
431         skip_draw_ = false;
432 
433         if (showed_ && redraw_)
434         {
435             rect update_field {
```

```
436                 old_position.left < position_.left ? old_position.left :
                        position_.left,
437                 old_position.top < position_.top ? old_position.top :
                        position_.top,
438                 old_position.right > position_.right ? old_position.right :
                        position_.right,
439                 old_position.bottom > position_.bottom ? old_position.bottom
                        : position_.bottom
440             };
441             parent__->redraw(update_field, true);
442         }
443     }
444 }
445
446 rect window::position() const
447 {
448     return get_control_position(position_, parent_);
449 }
450
451 void window::set_parent(std::shared_ptr<window> window)
452 {
453     parent_ = window;
454
455     if (window)
456     {
457         if (context_.hwnd)
458         {
459             DestroyWindow(context_.hwnd);
460         }
461
462         my_control_sid = window->subscribe(std::bind(&window::
                receive_control_events, this, std::placeholders::_1), event_type::
                all, shared_from_this());
463         my_plain_sid = window->subscribe(std::bind(&window::
                receive_plain_events, this, std::placeholders::_1), event_type::
                all);
464
465         pin_button->set_caption(locale(tcn, cl_unpin));
466     }
467 }
468
469 std::weak_ptr<window> window::parent() const
470 {
471     return parent_;
472 }
473
474 void window::clear_parent()
475 {
476     auto parent__ = parent_.lock();
477     if (parent__)
478     {
479         parent__->unsubscribe(my_control_sid);
480         parent__->unsubscribe(my_plain_sid);
481     }
```

```
482
483     parent_.reset();
484 }
485
486 void window::set_topmost(bool yes)
487 {
488     if (yes)
489     {
490         set_style(static_cast<window_style>(static_cast<uint32_t>(
                window_style_) | static_cast<uint32_t>(window_style::topmost)));
491     }
492     else
493     {
494         set_style(static_cast<window_style>(static_cast<uint32_t>(
                window_style_) & ~static_cast<uint32_t>(window_style::topmost)));
495     }
496 }
497
498 bool window::topmost() const
499 {
500     return docked_ || parent_.lock() || flag_is_set(window_style_,
            window_style::topmost);
501 }
502
503 bool window::focused() const
504 {
505     for (const auto &control : controls)
506     {
507         if (control->focused())
508         {
509             return true;
510         }
511     }
512
513     return false;
514 }
515
516 bool window::focusing() const
517 {
518     for (const auto &control : controls)
519     {
520         if (control->focusing())
521         {
522             return true;
523         }
524     }
525
526     return false;
527 }
528
529 void window::update_theme_control_name(std::string_view theme_control_name)
530 {
531     tcn = theme_control_name;
532     update_theme(theme_);
```

```
533 }
534
535 void window::update_theme(std::shared_ptr<i_theme> theme__)
536 {
537     if (theme_ && !theme__)
538     {
539         return;
540     }
541     theme_ = theme__;
542
543     if (context_.valid() && !parent_.lock())
544     {
545         graphic_.set_background_color(theme_color(tcn, tv_background, theme_)
              );
546
547         RECT client_rect;
548         GetClientRect(context_.hwnd, &client_rect);
549         InvalidateRect(context_.hwnd, &client_rect, TRUE);
550     }
551
552     for (auto &control : controls)
553     {
554         control->update_theme(theme_);
555     }
556
557     update_button_images();
558 }
559
560 void window::show()
561 {
562     showed_ = true;
563
564     if (!parent_.lock())
565     {
566         ShowWindow(context_.hwnd, SW_SHOW);
567     }
568     else
569     {
570         for (auto &control : controls)
571         {
572             control->show();
573         }
574     }
575 }
576
577 void window::hide()
578 {
579     showed_ = false;
580
581     auto parent__ = parent_.lock();
582     if (!parent__)
583     {
584         ShowWindow(context_.hwnd, SW_HIDE);
585     }
```

```
586     else
587     {
588         for (auto &control : controls)
589         {
590             control->hide();
591         }
592 
593         parent__->redraw(position(), true);
594     }
595 }
596 
597 bool window::showed() const
598 {
599     return showed_;
600 }
601 
602 void window::enable()
603 {
604     enabled_ = true;
605 
606     EnableWindow(context_.hwnd, TRUE);
607     SetWindowPos(context_.hwnd, HWND_TOPMOST, 0, 0, 0, 0, SWP_SHOWWINDOW |
        SWP_NOSIZE | SWP_NOMOVE);
608     SetWindowPos(context_.hwnd, HWND_NOTOPMOST, 0, 0, 0, 0, SWP_SHOWWINDOW |
        SWP_NOSIZE | SWP_NOMOVE);
609 }
610 
611 void window::disable()
612 {
613     enabled_ = false;
614 
615     EnableWindow(context_.hwnd, FALSE);
616 }
617 
618 bool window::enabled() const
619 {
620     return enabled_;
621 }
622 
623 error window::get_error() const
624 {
625     return err;
626 }
627 
628 void window::switch_lang()
629 {
630   if (control_callback)
631   {
632     std::string tooltip_text;
633     bool continue_ = true;
634     control_callback(window_control::lang, tooltip_text, continue_);
635     switch_lang_button->set_caption(tooltip_text);
636   }
637 }
```

```
638
639 void window::switch_theme()
640 {
641     if (control_callback)
642     {
643         std::string tooltip_text;
644         bool continue_ = true;
645         control_callback(window_control::theme, tooltip_text, continue_);
646         switch_theme_button->set_caption(tooltip_text);
647     }
648 }
649
650 void window::pin()
651 {
652     if (active_control)
653     {
654         mouse_event me{ mouse_event_type::leave };
655         send_event_to_control(active_control, { event_type::mouse, me });
656         active_control.reset();
657     }
658
659     if (control_callback)
660     {
661         std::string tooltip_text;
662         bool continue_ = true;
663         control_callback(window_control::pin, tooltip_text, continue_);
664         pin_button->set_caption(tooltip_text);
665     }
666 }
667
668 void window::minimize()
669 {
670     if (window_state_ == window_state::minimized)
671     {
672         return;
673     }
674
675     if (control_callback)
676     {
677         std::string text = "minimize";
678         bool continue_ = true;
679         control_callback(window_control::state, text, continue_);
680         if (!continue_)
681         {
682             return;
683         }
684     }
685
686     normal_position = position();
687
688     prev_window_state_ = window_state_;
689
690     ShowWindow(context_.hwnd, SW_MINIMIZE);
691
```

```
692     window_state_ = window_state::minimized;
693
694     send_internal(internal_event_type::window_minimized, -1, -1);
695 }
696
697 void window::expand()
698 {
699     if (window_state_ == window_state::maximized)
700     {
701         return;
702     }
703
704     if (control_callback)
705     {
706         std::string text = "expand";
707         bool continue_ = true;
708         control_callback(window_control::state, text, continue_);
709         if (!continue_)
710         {
711             return;
712         }
713     }
714
715     window_state_ = window_state::maximized;
716     auto screenSize = wui::get_screen_size(context());
717     if (position().width() != screenSize.width() && position().height() !=
        screenSize.height())
718     {
719         normal_position = position();
720     }
721
722     MONITORINFO mi = { sizeof(mi) };
723     if (GetMonitorInfo(MonitorFromWindow(context_.hwnd,
        MONITOR_DEFAULTTOPRIMARY), &mi))
724     {
725         if (flag_is_set(window_style_, window_style::title_showed) &&
            flag_is_set<DWORD>(mi.dwFlags, MONITORINFOF_PRIMARY)) // normal
            window maximization
726         {
727             RECT work_area;
728             SystemParametersInfo(SPI_GETWORKAREA, 0, &work_area, 0);
729             SetWindowPos(context_.hwnd, NULL, work_area.left, work_area.top,
                work_area.right, work_area.bottom, NULL);
730         }
731         else // expand to full screen
732         {
733             SetWindowPos(context_.hwnd,
734                 HWND_TOP,
735                 mi.rcMonitor.left, mi.rcMonitor.top,
736                 mi.rcMonitor.right - mi.rcMonitor.left,
737                 mi.rcMonitor.bottom - mi.rcMonitor.top,
738                 SWP_NOOWNERZORDER | SWP_FRAMECHANGED);
739         }
740     }
```

```
741     expand_button->set_image(theme_image(ti_normal, theme_));
742 }
743
744 void window::normal()
745 {
746     if (control_callback)
747     {
748         std::string text = "normal";
749         bool continue_ = true;
750         control_callback(window_control::state, text, continue_);
751         if (!continue_)
752         {
753             return;
754         }
755     }
756
757     SetWindowPos(context_.hwnd, HWND_TOPMOST, 0, 0, 0, 0, SWP_SHOWWINDOW |
        SWP_NOSIZE | SWP_NOMOVE);
758     SetWindowPos(context_.hwnd, HWND_NOTOPMOST, 0, 0, 0, 0, SWP_SHOWWINDOW |
        SWP_NOSIZE | SWP_NOMOVE);
759
760     ShowWindow(context_.hwnd, SW_RESTORE);
761
762     window_state_ = window_state::normal;
763
764     if (!normal_position.is_null())
765     {
766         set_position(normal_position, false);
767     }
768
769     expand_button->set_image(theme_image(ti_expand, theme_));
770
771     update_buttons();
772
773     send_internal(internal_event_type::window_normalized, position_.width(),
        position_.height());
774
775     redraw({ 0, 0, position_.width(), position_.height() }, true);
776 }
777
778 window_state window::state() const
779 {
780     return window_state_;
781 }
782
783 #ifndef _WIN32
784 void set_wm_name(system_context &context_, std::string_view caption)
785 {
786     xcb_icccm_set_wm_name(context_.connection, context_.wnd,
787         XCB_ATOM_STRING, 8,
788         caption.size(), caption.data());
789
790     xcb_icccm_set_wm_icon_name(context_.connection, context_.wnd,
791         XCB_ATOM_STRING, 8,
```

```
792         caption.size(), caption.data());
793
794     std::string class_hint = std::string(caption) + '\0' + std::string(
        caption) + '\0';
795     xcb_icccm_set_wm_class(context_.connection, context_.wnd, class_hint.size
        (), class_hint.c_str());
796 }
797 #endif
798
799 void window::set_caption(std::string_view caption_)
800 {
801     caption = caption_;
802
803     if (flag_is_set(window_style_, window_style::title_showed) && !parent_.
        lock())
804     {
805         SetWindowText(context_.hwnd, boost::nowide::widen(caption).c_str());
806
807         redraw({ 0, 0, position_.width(), 30 }, true);
808     }
809 }
810
811 void window::set_style(window_style style)
812 {
813     window_style_ = style;
814
815     update_buttons();
816
817     if (topmost())
818     {
819         SetWindowPos(context_.hwnd, HWND_TOPMOST, 0, 0, 0, 0, SWP_NOMOVE |
            SWP_NOSIZE);
820         if (window_state_ == window_state::maximized)
821         {
822             MONITORINFO mi = { sizeof(mi) };
823             if (GetMonitorInfo(MonitorFromWindow(context_.hwnd,
                MONITOR_DEFAULTTOPRIMARY), &mi))
824             {
825                 SetWindowPos(context_.hwnd,
826                     HWND_TOP,
827                     mi.rcMonitor.left, mi.rcMonitor.top,
828                     mi.rcMonitor.right - mi.rcMonitor.left,
829                     mi.rcMonitor.bottom - mi.rcMonitor.top,
830                     SWP_NOOWNERZORDER | SWP_FRAMECHANGED);
831             }
832         }
833     }
834     else
835     {
836         SetWindowPos(context_.hwnd, HWND_NOTOPMOST, 0, 0, 0, 0, SWP_NOMOVE |
            SWP_NOSIZE);
837     }
838 }
839
```

```
840 void window::set_min_size(int32_t width, int32_t height)
841 {
842     min_width = width;
843     min_height = height;
844 }
845
846 void window::set_transient_for(std::shared_ptr<window> window_, bool docked__
    )
847 {
848     transient_window = window_;
849     docked_ = docked__;
850 }
851
852 void window::start_docking(std::shared_ptr<i_control> control)
853 {
854     enabled_ = false;
855
856     docked_control = control;
857
858     send_internal(internal_event_type::remove_focus, 0, 0);
859
860     set_focused(control);
861 }
862
863 void window::end_docking()
864 {
865     enabled_ = true;
866
867     docked_control.reset();
868 }
869
870 void window::disable_draw()
871 {
872     skip_draw_ = true;
873 }
874
875 void window::enable_draw()
876 {
877     skip_draw_ = false;
878 }
879
880 bool window::draw_enabled() const
881 {
882     return !skip_draw_;
883 }
884
885 void window::emit_event(int32_t x, int32_t y)
886 {
887     auto parent__ = parent_.lock();
888     if (!parent__)
889     {
890         PostMessage(context_.hwnd, WM_USER, x, y);
891     }
892     else
```

```
893     {
894         parent__->emit_event(x, y);
895     }
896 }
897
898 void window::set_control_callback(std::function<void(window_control control,
        std::string &text, bool &continue_)> callback_)
899 {
900     control_callback = callback_;
901 }
902
903 void window::set_default_push_control(std::shared_ptr<i_control> control)
904 {
905     default_push_control = control;
906 }
907
908 void window::send_event_to_control(const std::shared_ptr<i_control> &control_
        , const event &ev)
909 {
910     auto it = std::find_if(subscribers_.begin(), subscribers_.end(), [
            control_, ev](const event_subscriber &es) {
911         return flag_is_set(es.event_types, ev.type) && es.control == control_
                ;
912     });
913
914     if (it != subscribers_.end())
915     {
916         it->receive_callback(ev);
917     }
918 }
919
920 void window::send_event_to_plains(const event &ev)
921 {
922     auto subscribers__ = subscribers_; // This is necessary to be able to
            remove the subscriber in the callback
923     for (auto &s : subscribers__)
924     {
925         if (!s.control && flag_is_set(s.event_types, ev.type) && s.
                receive_callback)
926         {
927             s.receive_callback(ev);
928         }
929     }
930 }
931
932 void window::send_mouse_event(const mouse_event &ev)
933 {
934     if (!enabled_ && !docked_control)
935     {
936         return;
937     }
938
939     if (active_control && !active_control->position().in(ev.x, ev.y))
940     {
```

```
941             mouse_event me{ mouse_event_type::leave };
942             send_event_to_control(active_control, { event_type::mouse, me });
943 
944             active_control.reset();
945         }
946 
947         send_event_to_plains({ event_type::mouse, ev });
948 
949         auto send_mouse_event_to_control = [this](std::shared_ptr<wui::i_control>
                &send_to_control,
950             const mouse_event &ev_) noexcept -> void
951         {
952             if (active_control == send_to_control)
953             {
954                 if (send_to_control->focusing() && ev_.type == mouse_event_type::
                        left_down)
955                 {
956                     set_focused(send_to_control);
957                 }
958 
959                 return send_event_to_control(send_to_control, { event_type::mouse
                        , ev_ });
960             }
961             else
962             {
963                 if (active_control)
964                 {
965                     mouse_event me{ mouse_event_type::leave };
966                     send_event_to_control(active_control, { event_type::mouse, me
                            });
967                 }
968 
969                 if (ev_.y < 5 || (ev_.y < 24 && ev_.x > position().width() - 5))
                        /// control buttons border
970                 {
971                     return active_control.reset();
972                 }
973                 else
974                 {
975                     active_control = send_to_control;
976 
977                     mouse_event me{ mouse_event_type::enter };
978                     return send_event_to_control(send_to_control, { event_type::
                            mouse, me });
979                 }
980             }
981         };
982 
983         if (enabled_)
984         {
985             auto end = controls.rend();
986             for (auto control = controls.rbegin(); control != end; ++control)
987             {
```

```
988             if (*control && (*control)->topmost() && (*control)->showed() &&
                    (*control)->position().in(ev.x, ev.y))
989             {
990                 return send_mouse_event_to_control(*control, ev);
991             }
992         }
993 
994         for (auto control = controls.rbegin(); control != end; ++control)
995         {
996             if (*control && (*control)->showed() && (*control)->position().in
                    (ev.x, ev.y))
997             {
998                 return send_mouse_event_to_control(*control, ev);
999             }
1000        }
1001    }
1002    else
1003    {
1004        for (auto &control : controls)
1005        {
1006            if (control && control->position().in(ev.x, ev.y) && control ==
                    docked_control)
1007            {
1008                return send_mouse_event_to_control(control, ev);
1009            }
1010        }
1011    }
1012}
1013
1014bool window::check_control_here(int32_t x, int32_t y)
1015{
1016    for (auto &control : controls)
1017    {
1018        if (control->showed() &&
1019            control->position().in(x, y) &&
1020            std::find_if(subscribers_.begin(), subscribers_.end(), [&control
                    ](const event_subscriber &es) { return es.control == control;
                    }) != subscribers_.end())
1021        {
1022            return true;
1023        }
1024    }
1025    return false;
1026}
1027
1028void window::change_focus()
1029{
1030    if (controls.empty())
1031    {
1032        return;
1033    }
1034
1035    for (auto &control : controls)
1036    {
```

```
1037            if (control->focused() && control != docked_control)
1038            {
1039                event ev;
1040                ev.type = event_type::internal;
1041                ev.internal_event_ = internal_event{ internal_event_type::
                        remove_focus };
1042                send_event_to_control(control, ev);
1043
1044                if (!control->focused()) /// need to change the focus inside the
                        internal elements of the control
1045                {
1046                    ++focused_index;
1047                }
1048                else
1049                {
1050                    return;
1051                }
1052                break;
1053            }
1054        }
1055
1056        size_t focusing_controls = 0;
1057        for (const auto &control : controls)
1058        {
1059            if (control->focusing())
1060            {
1061                ++focusing_controls;
1062            }
1063        }
1064
1065        if (focused_index >= focusing_controls)
1066        {
1067            focused_index = 0;
1068        }
1069
1070        set_focused(focused_index);
1071    }
1072
1073    void window::execute_focused()
1074    {
1075        std::shared_ptr<wui::i_control> control;
1076
1077        if (docked_control)
1078        {
1079            control = docked_control;
1080        }
1081        else if (default_push_control)
1082        {
1083            control = default_push_control;
1084        }
1085        else
1086        {
1087            control = get_focused();
1088        }
```

```
1089
1090        if (control)
1091        {
1092            event ev;
1093            ev.type = event_type::internal;
1094            ev.internal_event_ = internal_event{ internal_event_type::
                    execute_focused };
1095
1096            send_event_to_control(control, ev);
1097        }
1098    }
1099
1100    void window::set_focused(std::shared_ptr<i_control> control)
1101    {
1102        size_t index = 0;
1103        for (auto &c : controls)
1104        {
1105            if (c == control)
1106            {
1107                if (c->focused())
1108                {
1109                    return;
1110                }
1111                focused_index = index;
1112            }
1113
1114            if (c->focused())
1115            {
1116                event ev;
1117                ev.type = event_type::internal;
1118                ev.internal_event_ = internal_event{ internal_event_type::
                        remove_focus };
1119                send_event_to_control(c, ev);
1120            }
1121
1122            if (c->focusing())
1123            {
1124                ++index;
1125            }
1126        }
1127
1128        if (control)
1129        {
1130            event ev;
1131            ev.type = event_type::internal;
1132            ev.internal_event_ = internal_event{ internal_event_type::set_focus
                    };
1133            send_event_to_control(control, ev);
1134        }
1135    }
1136
1137    void window::set_focused(size_t focused_index_)
1138    {
1139        size_t index = 0;
```

```
1140     for (auto &control : controls)
1141     {
1142         if (control->focusing())
1143         {
1144             if (index == focused_index_)
1145             {
1146                 event ev;
1147                 ev.type = event_type::internal;
1148                 ev.internal_event_ = internal_event{ internal_event_type::
                        set_focus };
1149                 send_event_to_control(control, ev);
1150 
1151                 break;
1152             }
1153 
1154             ++index;
1155         }
1156     }
1157 }
1158 
1159 std::shared_ptr<i_control> window::get_focused()
1160 {
1161     for (auto &control : controls)
1162     {
1163         if (control->focused())
1164         {
1165             return control;
1166         }
1167     }
1168 
1169     return nullptr;
1170 }
1171 
1172 void window::update_button_images()
1173 {
1174   switch_lang_button->set_image(theme_image(ti_switch_lang, theme_));
1175     switch_theme_button->set_image(theme_image(ti_switch_theme, theme_));
1176     pin_button->set_image(theme_image(ti_pin, theme_));
1177     minimize_button->set_image(theme_image(ti_minimize, theme_));
1178     expand_button->set_image(theme_image(window_state_ == window_state::
            normal ? ti_expand : ti_normal, theme_));
1179     close_button->set_image(theme_image(ti_close, theme_));
1180 }
1181 
1182 void window::update_buttons()
1183 {
1184   auto border_height = flag_is_set(window_style_, window_style::border_top) ?
          theme_dimension(tcn, tv_border_width, theme_) : 0;
1185     auto border_width = flag_is_set(window_style_, window_style::border_right
            ) ? theme_dimension(tcn, tv_border_width, theme_) : 0;
1186 
1187     auto btn_size = 28;
1188     auto left = position_.width() - btn_size - border_width;
1189     auto top = border_height;
```

```
1190
1191        if (flag_is_set(window_style_, window_style::close_button))
1192        {
1193            close_button->set_position({ left, top, left + btn_size, top +
                btn_size }, false);
1194            close_button->show();
1195
1196            left -= btn_size;
1197        }
1198        else
1199        {
1200            close_button->hide();
1201        }
1202
1203        if (flag_is_set(window_style_, window_style::expand_button) ||
            flag_is_set(window_style_, window_style::minimize_button))
1204        {
1205            expand_button->set_position({ left, top, left + btn_size, top +
                btn_size }, false);
1206            expand_button->show();
1207
1208            left -= btn_size;
1209
1210            if (flag_is_set(window_style_, window_style::expand_button))
1211            {
1212                expand_button->enable();
1213            }
1214            else
1215            {
1216                expand_button->disable();
1217            }
1218        }
1219        else
1220        {
1221            expand_button->hide();
1222        }
1223
1224        if (flag_is_set(window_style_, window_style::minimize_button))
1225        {
1226            minimize_button->set_position({ left, top, left + btn_size, top +
                btn_size }, false);
1227            minimize_button->show();
1228
1229            left -= btn_size;
1230        }
1231        else
1232        {
1233            minimize_button->hide();
1234        }
1235
1236        if (flag_is_set(window_style_, window_style::pin_button))
1237        {
1238            pin_button->set_position({ left, top, left + btn_size, top + btn_size
                }, false);
```

```
1239            pin_button->show();
1240
1241            left -= btn_size;
1242        }
1243        else
1244        {
1245            pin_button->hide();
1246        }
1247
1248        if (flag_is_set(window_style_, window_style::switch_theme_button))
1249        {
1250            switch_theme_button->set_position({ left, top, left + btn_size, top +
                    btn_size }, false);
1251            switch_theme_button->show();
1252
1253            left -= btn_size;
1254        }
1255        else
1256        {
1257            switch_theme_button->hide();
1258        }
1259
1260    if (flag_is_set(window_style_, window_style::switch_lang_button))
1261    {
1262        switch_lang_button->set_position({ left, top, left + btn_size, top +
               btn_size }, false);
1263        switch_lang_button->show();
1264
1265        left -= btn_size;
1266    }
1267    else
1268    {
1269        switch_lang_button->hide();
1270    }
1271 }
1272
1273 void window::draw_border(graphic &gr)
1274 {
1275     auto c = theme_color(tcn, tv_border, theme_);
1276     auto x = theme_dimension(tcn, tv_border_width, theme_);
1277
1278     int32_t l = 0, t = 0, w = 0, h = 0;
1279
1280     auto pos = position();
1281
1282     if (parent_.lock())
1283     {
1284         l = pos.left;
1285         t = pos.top;
1286         w = pos.right - x;
1287         h = pos.bottom - x;
1288
1289     }
1290     else
```

```
1291        {
1292            w = position_.width() - x;
1293            h = position_.height() - x;
1294        }
1295
1296        if (flag_is_set(window_style_, window_style::border_left))
1297        {
1298            gr.draw_line({ l, t, l, h }, c, x);
1299        }
1300        if (flag_is_set(window_style_, window_style::border_top))
1301        {
1302            gr.draw_line({ l, t, w, t }, c, x);
1303        }
1304        if (flag_is_set(window_style_, window_style::border_right))
1305        {
1306            gr.draw_line({ w, t, w, h }, c, x);
1307        }
1308        if (flag_is_set(window_style_, window_style::border_bottom))
1309        {
1310            gr.draw_line({ l, h, w, h }, c, x);
1311        }
1312    }
1313
1314    void window::send_internal(internal_event_type type, int32_t x, int32_t y)
1315    {
1316        event ev_;
1317        ev_.type = event_type::internal;
1318        ev_.internal_event_ = internal_event{ type, x, y };
1319        send_event_to_plains(ev_);
1320    }
1321
1322    void window::send_system(system_event_type type, int32_t x, int32_t y)
1323    {
1324        event ev_;
1325        ev_.type = event_type::system;
1326        ev_.system_event_ = system_event{ type, x, y };
1327        send_event_to_plains(ev_);
1328    }
1329
1330    std::shared_ptr<window> window::get_transient_window()
1331    {
1332        return transient_window.lock();
1333    }
1334
1335    bool window::init(std::string_view caption_, const rect &position__,
        window_style style, std::function<void(void)> close_callback_)
1336    {
1337        err.reset();
1338
1339        caption = caption_;
1340
1341        if (!position__.is_null())
1342        {
1343            position_ = position__;
```

```
1344        }
1345
1346        window_style_ = style;
1347        close_callback = close_callback_;
1348
1349    add_control(switch_lang_button, { 0 });
1350        add_control(switch_theme_button, { 0 });
1351        add_control(pin_button, { 0 });
1352        add_control(minimize_button, { 0 });
1353        add_control(expand_button, { 0 });
1354        add_control(close_button, { 0 });
1355
1356        update_button_images();
1357        update_buttons();
1358
1359        auto transient_window_ = get_transient_window();
1360        if (transient_window_)
1361        {
1362            if (transient_window_->window_state_ == window_state::minimized)
1363            {
1364                transient_window_->normal();
1365            }
1366
1367            if (docked_ && transient_window_->position_ > position_)
1368            {
1369          int32_t left = (transient_window_->position().width() - position_.width
                ()) / 2;
1370                int32_t top = (transient_window_->position().height() - position_
                    .height()) / 2;
1371
1372          transient_window_->add_control(shared_from_this(), { left, top, left +
                position_.width(), top + position_.height() });
1373                transient_window_->start_docking(shared_from_this());
1374            }
1375            else
1376            {
1377                auto transient_window_pos = transient_window_->position();
1378
1379                int32_t left = 0, top = 0;
1380
1381                if (transient_window_pos.width() > 0 && transient_window_pos.
                    height() > 0)
1382                {
1383                    left = transient_window_pos.left + ((transient_window_pos.
                        width() - position_.width()) / 2);
1384                    top = transient_window_pos.top + ((transient_window_pos.
                        height() - position_.height()) / 2);
1385                }
1386                else
1387                {
1388                    RECT work_area;
1389                    SystemParametersInfo(SPI_GETWORKAREA, 0, &work_area, 0);
1390                    auto screen_width = work_area.right - work_area.left,
1391                         screen_height = work_area.bottom - work_area.top;
```

```
1392                left = (screen_width - position_.width()) / 2;
1393                top = (screen_height - position_.height()) / 2;
1394            }
1395
1396            position_.put(left, top);
1397
1398            transient_window_->disable();
1399        }
1400    }
1401
1402    auto parent__ = parent_.lock();
1403    if (parent__)
1404    {
1405        send_internal(internal_event_type::window_created, 0, 0);
1406
1407        send_internal(internal_event_type::size_changed, position_.width(),
            position_.height());
1408
1409        parent__->redraw(position());
1410
1411        return true;
1412    }
1413
1414    auto h_inst = GetModuleHandle(NULL);
1415
1416    WNDCLASSEXW wcex = { 0 };
1417
1418    wcex.cbSize = sizeof(WNDCLASSEX);
1419
1420    wcex.style = CS_DBLCLKS;
1421    wcex.lpfnWndProc = window::wnd_proc;
1422    wcex.cbClsExtra = 0;
1423    wcex.cbWndExtra = sizeof(*this);
1424    wcex.hInstance = h_inst;
1425    wcex.hbrBackground = NULL;
1426    wcex.lpszClassName = L"WUI Window";
1427
1428    RegisterClassExW(&wcex);
1429
1430    if (position_.left == -1)
1431    {
1432        center_horizontally(position_, context_);
1433    }
1434    if (position_.top == -1)
1435    {
1436        center_vertically(position_, context_);
1437    }
1438
1439    context_.hwnd = CreateWindowEx(!topmost() ? 0 : WS_EX_TOPMOST,
1440        wcex.lpszClassName,
1441        L"",
1442        WS_VISIBLE | WS_MINIMIZEBOX | WS_POPUP | (window_state_ ==
            window_state::minimized ? WS_MINIMIZE : 0),
1443        position_.left,
```

```
1444            position_.top,
1445            position_.width(),
1446            position_.height(),
1447            nullptr,
1448            nullptr,
1449            h_inst,
1450            this);
1451
1452        if (!context_.hwnd)
1453        {
1454            return false;
1455        }
1456
1457        if (window_state_ == window_state::maximized)
1458        {
1459            expand();
1460        }
1461
1462        SetWindowText(context_.hwnd, boost::nowide::widen(caption).c_str());
1463
1464        UpdateWindow(context_.hwnd);
1465
1466        send_internal(internal_event_type::size_changed, position_.width(),
                position_.height());
1467
1468        if (!showed_)
1469        {
1470            ShowWindow(context_.hwnd, SW_HIDE);
1471        }
1472
1473        return true;
1474    }
1475
1476    void window::destroy()
1477    {
1478        if (control_callback)
1479        {
1480            std::string text = "close";
1481            bool continue_= true;
1482            control_callback(window_control::close, text, continue_);
1483            if (!continue_)
1484            {
1485                return;
1486            }
1487        }
1488
1489        std::vector<std::shared_ptr<i_control>> controls_;
1490        controls_ = controls; /// This is necessary to solve the problem of
                removing child controls within a control
1491
1492        for (auto &control : controls_)
1493        {
1494            if (control)
1495            {
```

```
1496                control->clear_parent();
1497            }
1498        }
1499
1500        if (active_control)
1501        {
1502            mouse_event me{ mouse_event_type::leave };
1503            send_event_to_control(active_control, { event_type::mouse, me });
1504        }
1505
1506        active_control.reset();
1507
1508        controls.clear();
1509
1510        auto parent__ = parent_.lock();
1511        if (parent__)
1512        {
1513            parent__->remove_control(shared_from_this());
1514
1515            auto transient_window_ = get_transient_window();
1516            if (transient_window_)
1517            {
1518                transient_window_->end_docking();
1519            }
1520
1521            if (close_callback)
1522            {
1523                close_callback();
1524            }
1525        }
1526        else
1527        {
1528            DestroyWindow(context_.hwnd);
1529        }
1530    }
1531
1532    /// Windows specified code
1533
1534    uint8_t get_key_modifier()
1535    {
1536        if (GetKeyState(VK_SHIFT) < 0)
1537        {
1538            return vk_lshift;
1539        }
1540        else if (GetKeyState(VK_CAPITAL) & 0x0001)
1541        {
1542            return vk_capital;
1543        }
1544        else if (GetKeyState(VK_MENU) < 0)
1545        {
1546            return vk_alt;
1547        }
1548        else if (GetKeyState(VK_LCONTROL) < 0)
1549        {
```

```
1550            return vk_lcontrol;
1551        }
1552        else if (GetKeyState(VK_RCONTROL) < 0)
1553        {
1554            return vk_rcontrol;
1555        }
1556        else if (GetKeyState(VK_INSERT) & 0x0001)
1557        {
1558            return vk_insert;
1559        }
1560        else if (GetKeyState(VK_NUMLOCK) & 0x0001)
1561        {
1562            return vk_numlock;
1563        }
1564        return 0;
1565    }
1566
1567    LRESULT CALLBACK window::wnd_proc(HWND hwnd, UINT message, WPARAM w_param,
        LPARAM l_param)
1568    {
1569        switch (message)
1570        {
1571            case WM_CREATE:
1572            {
1573                SetWindowLongPtr(hwnd, GWLP_USERDATA, reinterpret_cast<LONG_PTR>(
                    reinterpret_cast<CREATESTRUCT*>(l_param)->lpCreateParams));
1574
1575                window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                    GWLP_USERDATA));
1576
1577                wnd->graphic_.init(get_screen_size(wnd->context_), theme_color(
                    wnd->tcn, tv_background, wnd->theme_));
1578
1579                wnd->send_internal(internal_event_type::window_created, 0, 0);
1580            }
1581            break;
1582            case WM_PAINT:
1583            {
1584                window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                    GWLP_USERDATA));
1585
1586                PAINTSTRUCT ps;
1587                auto bpdc = BeginPaint(hwnd, &ps);
1588
1589                if (bpdc == NULL)
1590                {
1591                    return 0;
1592                }
1593
1594                const rect paint_rect{ ps.rcPaint.left,
1595                    ps.rcPaint.top,
1596                    ps.rcPaint.right,
1597                    ps.rcPaint.bottom };
1598
```

```
1599            if (ps.fErase)
1600            {
1601                wnd->graphic_.clear(paint_rect);
1602            }
1603            if (flag_is_set(wnd->window_style_, window_style::title_showed)
                && !wnd->parent_.lock())
1604            {
1605                auto caption_font = theme_font(wnd->tcn, tv_caption_font, wnd
                    ->theme_);
1606
1607                auto caption_rect = wnd->graphic_.measure_text(wnd->caption,
                    caption_font);
1608                caption_rect.move(5, 5);
1609
1610                if (caption_rect.in(paint_rect))
1611                {
1612                    wnd->graphic_.draw_rect(caption_rect, theme_color(wnd->
                        tcn, tv_background, wnd->theme_));
1613                    wnd->graphic_.draw_text(caption_rect,
1614                        wnd->caption,
1615                        theme_color(wnd->tcn, tv_text, wnd->theme_),
1616                        caption_font);
1617                }
1618            }
1619
1620            wnd->draw_border(wnd->graphic_);
1621
1622            std::vector<std::shared_ptr<i_control>> topmost_controls;
1623
1624            for (auto &control : wnd->controls)
1625            {
1626                if (control->position().in(paint_rect))
1627                {
1628                    if (!control->topmost())
1629                    {
1630                        control->draw(wnd->graphic_, paint_rect);
1631                    }
1632                    else
1633                    {
1634                        topmost_controls.emplace_back(control);
1635                    }
1636                }
1637            }
1638
1639            for (auto &control : topmost_controls)
1640            {
1641                control->draw(wnd->graphic_, paint_rect);
1642            }
1643
1644            wnd->graphic_.flush(paint_rect);
1645
1646            EndPaint(hwnd, &ps);
1647        }
1648        break;
```

```
1649        case WM_MOUSEMOVE:
1650        {
1651            window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                GWLP_USERDATA));
1652
1653            RECT window_rect;
1654            GetWindowRect(hwnd, &window_rect);
1655
1656            int16_t x_mouse = GET_X_LPARAM(l_param);
1657            int16_t y_mouse = GET_Y_LPARAM(l_param);
1658
1659            static bool cursor_size_view = false;
1660
1661            if (flag_is_set(wnd->window_style_, window_style::resizable) &&
                wnd->window_state_ == window_state::normal)
1662            {
1663                if ((x_mouse > window_rect.right - window_rect.left - 5 &&
                    y_mouse > window_rect.bottom - window_rect.top - 5) ||
1664                    (x_mouse < 5 && y_mouse < 5))
1665                {
1666                    set_cursor(wnd->context_, cursor::size_nwse);
1667                    cursor_size_view = true;
1668                }
1669                else if ((x_mouse > window_rect.right - window_rect.left - 5
                    && y_mouse < 5) ||
1670                    (x_mouse < 5 && y_mouse > window_rect.bottom -
                        window_rect.top - 5))
1671                {
1672                    set_cursor(wnd->context_, cursor::size_nesw);
1673                    cursor_size_view = true;
1674                }
1675                else if (x_mouse > window_rect.right - window_rect.left - 5
                    || x_mouse < 5)
1676                {
1677                    set_cursor(wnd->context_, cursor::size_we);
1678                    cursor_size_view = true;
1679                }
1680                else if (y_mouse > window_rect.bottom - window_rect.top - 5
                    || y_mouse < 5)
1681                {
1682                    set_cursor(wnd->context_, cursor::size_ns);
1683                    cursor_size_view = true;
1684                }
1685                else if (cursor_size_view &&
1686                    x_mouse > 5 && x_mouse < window_rect.right - window_rect.
                        left - 5 &&
1687                    y_mouse > 5 && y_mouse < window_rect.bottom - window_rect
                        .top - 5)
1688                {
1689                    set_cursor(wnd->context_, cursor::default_);
1690                    cursor_size_view = false;
1691                }
1692            }
1693
```

```
1694            if (!wnd->mouse_tracked)
1695            {
1696                TRACKMOUSEEVENT track_mouse_event;
1697
1698                track_mouse_event.cbSize = sizeof(track_mouse_event);
1699                track_mouse_event.dwFlags = TME_LEAVE;
1700                track_mouse_event.hwndTrack = hwnd;
1701
1702                TrackMouseEvent(&track_mouse_event);
1703
1704                wnd->mouse_tracked = true;
1705            }
1706
1707            if (wnd->moving_mode_ != moving_mode::none)
1708            {
1709                switch (wnd->moving_mode_)
1710                {
1711                    case moving_mode::move:
1712                    {
1713                        int32_t x_window = window_rect.left + x_mouse - wnd->
                               x_click;
1714                        int32_t y_window = window_rect.top + y_mouse - wnd->
                               y_click;
1715
1716                        SetWindowPos(hwnd, NULL, x_window, y_window, 0, 0,
                               SWP_NOSIZE | SWP_NOZORDER);
1717                    }
1718                    break;
1719                    case moving_mode::size_we_left:
1720                    {
1721                        POINT scr_mouse = { 0 };
1722                        GetCursorPos(&scr_mouse);
1723
1724                        int32_t width = window_rect.right - window_rect.left
                               - x_mouse;
1725                        int32_t height = window_rect.bottom - window_rect.top
                               ;
1726
1727                        if (width > wnd->min_width && height > wnd->
                               min_height)
1728                        {
1729                            SetWindowPos(hwnd, NULL, scr_mouse.x, window_rect
                                   .top, width, height, SWP_NOZORDER);
1730                        }
1731                    }
1732                    break;
1733                    case moving_mode::size_we_right:
1734                    {
1735                        int32_t width = x_mouse;
1736                        int32_t height = window_rect.bottom - window_rect.top
                               ;
1737                        if (width > wnd->min_width && height > wnd->
                               min_height)
1738                        {
```

```
1739                SetWindowPos(hwnd, NULL, 0, 0, width, height,
                        SWP_NOMOVE | SWP_NOZORDER);
1740            }
1741        }
1742        break;
1743        case moving_mode::size_ns_top:
1744        {
1745            POINT scr_mouse = { 0 };
1746            GetCursorPos(&scr_mouse);
1747
1748            int32_t width = window_rect.right - window_rect.left;
1749            int32_t height = window_rect.bottom - window_rect.top
                        - y_mouse;
1750            if (width > wnd->min_width && height > wnd->
                    min_height)
1751            {
1752                SetWindowPos(hwnd, NULL, window_rect.left,
                        scr_mouse.y, width, height, SWP_NOZORDER);
1753            }
1754        }
1755        break;
1756        case moving_mode::size_ns_bottom:
1757        {
1758            int32_t width = window_rect.right - window_rect.left;
1759            int32_t height = y_mouse;
1760            if (width > wnd->min_width && height > wnd->
                    min_height)
1761            {
1762                SetWindowPos(hwnd, NULL, 0, 0, width, height,
                        SWP_NOMOVE | SWP_NOZORDER);
1763            }
1764        }
1765        break;
1766        case moving_mode::size_nesw_top:
1767        {
1768            POINT scr_mouse = { 0 };
1769            GetCursorPos(&scr_mouse);
1770
1771            int32_t width = x_mouse;
1772            int32_t height = window_rect.bottom - window_rect.top
                        - y_mouse;
1773            if (width > wnd->min_width && height > wnd->
                    min_height)
1774            {
1775                SetWindowPos(hwnd, NULL, window_rect.left,
                        scr_mouse.y, width, height, SWP_NOZORDER);
1776            }
1777        }
1778        break;
1779        case moving_mode::size_nwse_bottom:
1780        {
1781            int32_t width = x_mouse;
1782            int32_t height = y_mouse;
```

```
1783                        if (width > wnd->min_width && height > wnd->
                                min_height)
1784                        {
1785                            SetWindowPos(hwnd, NULL, 0, 0, width, height,
                                    SWP_NOMOVE | SWP_NOZORDER);
1786                        }
1787                    }
1788                    break;
1789                    case moving_mode::size_nwse_top:
1790                    {
1791                        POINT scrMouse = { 0 };
1792                        GetCursorPos(&scrMouse);
1793
1794                        int32_t width = window_rect.right - window_rect.left
                                - x_mouse;
1795                        int32_t height = window_rect.bottom - window_rect.top
                                 - y_mouse;
1796                        if (width > wnd->min_width && height > wnd->
                                min_height)
1797                        {
1798                            SetWindowPos(hwnd, NULL, scrMouse.x, scrMouse.y,
                                    width, height, SWP_NOZORDER);
1799                        }
1800                    }
1801                    break;
1802                    case moving_mode::size_nesw_bottom:
1803                    {
1804                        POINT scr_mouse = { 0 };
1805                        GetCursorPos(&scr_mouse);
1806
1807                        int32_t width = window_rect.right - window_rect.left
                                - x_mouse;
1808                        int32_t height = y_mouse;
1809                        if (width > wnd->min_width && height > wnd->
                                min_height)
1810                        {
1811                            SetWindowPos(hwnd, NULL, scr_mouse.x, window_rect
                                    .top, width, height, SWP_NOZORDER);
1812                        }
1813                    }
1814                    break;
1815                }
1816            }
1817            else
1818            {
1819                wnd->send_mouse_event({ mouse_event_type::move, x_mouse,
                        y_mouse });
1820            }
1821        }
1822        break;
1823        case WM_LBUTTONDOWN:
1824        {
1825            window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                    GWLP_USERDATA));
```

```
1826
1827            SetCapture(hwnd);
1828
1829            RECT window_rect;
1830            GetWindowRect(hwnd, &window_rect);
1831
1832            wnd->x_click = GET_X_LPARAM(l_param);
1833            wnd->y_click = GET_Y_LPARAM(l_param);
1834
1835            wnd->send_mouse_event({ mouse_event_type::left_down, wnd->x_click
                , wnd->y_click });
1836
1837            if (wnd->window_state_ == window_state::normal)
1838            {
1839                if (flag_is_set(wnd->window_style_, window_style::moving) &&
1840                    !wnd->check_control_here(wnd->x_click, wnd->y_click))
1841                {
1842                    wnd->moving_mode_ = moving_mode::move;
1843                }
1844
1845                if (flag_is_set(wnd->window_style_, window_style::resizable))
1846                {
1847                    if (wnd->x_click > window_rect.right - window_rect.left -
                         5 && wnd->y_click > window_rect.bottom - window_rect.
                        top - 5)
1848                    {
1849                        wnd->moving_mode_ = moving_mode::size_nwse_bottom;
1850                    }
1851                    else if (wnd->x_click < 5 && wnd->y_click < 5)
1852                    {
1853                        wnd->moving_mode_ = moving_mode::size_nwse_top;
1854                    }
1855                    else if (wnd->x_click > window_rect.right - window_rect.
                        left - 5 && wnd->y_click < 5)
1856                    {
1857                        wnd->moving_mode_ = moving_mode::size_nesw_top;
1858                    }
1859                    else if (wnd->x_click < 5 && wnd->y_click > window_rect.
                        bottom - window_rect.top - 5)
1860                    {
1861                        wnd->moving_mode_ = moving_mode::size_nesw_bottom;
1862                    }
1863                    else if (wnd->x_click > window_rect.right - window_rect.
                        left - 5)
1864                    {
1865                        wnd->moving_mode_ = moving_mode::size_we_right;
1866                    }
1867                    else if (wnd->x_click < 5)
1868                    {
1869                        wnd->moving_mode_ = moving_mode::size_we_left;
1870                    }
1871                    else if (wnd->y_click > window_rect.bottom - window_rect.
                        top - 5)
1872                    {
```

```
1873                        wnd->moving_mode_ = moving_mode::size_ns_bottom;
1874                    }
1875                    else if (wnd->y_click < 5)
1876                    {
1877                        wnd->moving_mode_ = moving_mode::size_ns_top;
1878                    }
1879                }
1880            }
1881        }
1882        break;
1883        case WM_LBUTTONUP:
1884        {
1885            ReleaseCapture();
1886
1887            window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                GWLP_USERDATA));
1888
1889            wnd->moving_mode_ = moving_mode::none;
1890
1891            wnd->send_mouse_event({ mouse_event_type::left_up, GET_X_LPARAM(
                l_param), GET_Y_LPARAM(l_param) });
1892        }
1893        break;
1894        case WM_RBUTTONDOWN:
1895        {
1896            window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                GWLP_USERDATA));
1897            wnd->send_mouse_event({ mouse_event_type::right_down,
                GET_X_LPARAM(l_param), GET_Y_LPARAM(l_param) });
1898        }
1899        break;
1900        case WM_RBUTTONUP:
1901        {
1902            window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                GWLP_USERDATA));
1903            wnd->send_mouse_event({ mouse_event_type::right_up, GET_X_LPARAM(
                l_param), GET_Y_LPARAM(l_param) });
1904        }
1905        break;
1906        case WM_LBUTTONDBLCLK:
1907        {
1908            ReleaseCapture();
1909
1910            window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                GWLP_USERDATA));
1911
1912            wnd->moving_mode_ = moving_mode::none;
1913
1914            wnd->send_mouse_event({ mouse_event_type::left_double,
                GET_X_LPARAM(l_param), GET_Y_LPARAM(l_param) });
1915        }
1916        break;
1917        case WM_MOUSELEAVE:
1918        {
```

```
1919            window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                    GWLP_USERDATA));
1920            wnd->mouse_tracked = false;
1921            wnd->send_mouse_event({ mouse_event_type::leave });
1922        }
1923        break;
1924        case WM_MOUSEWHEEL:
1925        {
1926            window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                    GWLP_USERDATA));
1927            POINT p = { GET_X_LPARAM(l_param), GET_Y_LPARAM(l_param) };
1928            ScreenToClient(hwnd, &p);
1929            wnd->send_mouse_event({ mouse_event_type::wheel, p.x, p.y,
                    GET_WHEEL_DELTA_WPARAM(w_param) });
1930        }
1931        break;
1932        case WM_SIZE:
1933        {
1934            window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                    GWLP_USERDATA));
1935
1936            auto width = LOWORD(l_param), height = HIWORD(l_param);
1937
1938            wnd->position_ = { wnd->position_.left, wnd->position_.top, wnd->
                    position_.left + width, wnd->position_.top + height };
1939
1940            wnd->update_buttons();
1941
1942            wnd->send_internal(wnd->window_state_ != window_state::maximized
                    ? internal_event_type::size_changed : internal_event_type::
                    window_expanded, width, height);
1943
1944            RECT invalidatingRect = { 0, 0, width, height };
1945            InvalidateRect(hwnd, &invalidatingRect, FALSE);
1946        }
1947        break;
1948        case WM_MOVE:
1949        {
1950            window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                    GWLP_USERDATA));
1951
1952            RECT window_rect = { 0 };
1953            GetWindowRect(hwnd, &window_rect);
1954            wnd->position_ = rect{ window_rect.left, window_rect.top,
                    window_rect.right, window_rect.bottom };
1955
1956            wnd->send_internal(internal_event_type::position_changed,
                    window_rect.left, window_rect.top);
1957        }
1958        break;
1959        case WM_SYSCOMMAND:
1960            if (w_param == SC_RESTORE)
1961            {
```

```
1962                window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd
                        , GWLP_USERDATA));
1963                wnd->window_state_ = wnd->prev_window_state_;
1964            }
1965            return DefWindowProc(hwnd, message, w_param, l_param);
1966        break;
1967        case WM_KEYDOWN:
1968        {
1969            window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                    GWLP_USERDATA));
1970
1971            if (w_param == VK_TAB)
1972            {
1973                wnd->change_focus();
1974            }
1975            else if (w_param == VK_RETURN)
1976            {
1977                wnd->execute_focused();
1978            }
1979
1980            event ev;
1981            ev.type = event_type::keyboard;
1982            ev.keyboard_event_ = keyboard_event{ keyboard_event_type::down,
                    get_key_modifier(), 0 };
1983            ev.keyboard_event_.key[0] = static_cast<uint8_t>(w_param);
1984
1985            auto control = wnd->get_focused();
1986            if (control)
1987            {
1988                wnd->send_event_to_control(control, ev);
1989            }
1990            wnd->send_event_to_plains(ev);
1991        }
1992        break;
1993        case WM_KEYUP:
1994        {
1995            window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                    GWLP_USERDATA));
1996
1997            event ev;
1998            ev.type = event_type::keyboard;
1999            ev.keyboard_event_ = keyboard_event{ keyboard_event_type::up,
                    get_key_modifier(), 0 };
2000            ev.keyboard_event_.key[0] = static_cast<uint8_t>(w_param);
2001
2002            auto control = wnd->get_focused();
2003            if (control)
2004            {
2005                wnd->send_event_to_control(control, ev);
2006            }
2007            wnd->send_event_to_plains(ev);
2008        }
2009        break;
2010        case WM_CHAR:
```

```
2011            if (w_param != VK_ESCAPE && w_param != VK_BACK)
2012            {
2013                window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd
                        , GWLP_USERDATA));
2014
2015                event ev;
2016                ev.type = event_type::keyboard;
2017                ev.keyboard_event_ = keyboard_event{ keyboard_event_type::key
                        , get_key_modifier(), 0 };
2018                auto narrow_str = boost::nowide::narrow(reinterpret_cast<
                        const wchar_t*>(&w_param));
2019                memcpy(ev.keyboard_event_.key, narrow_str.c_str(), narrow_str
                        .size());
2020                ev.keyboard_event_.key_size = static_cast<uint8_t>(narrow_str
                        .size());
2021
2022                auto control = wnd->get_focused();
2023                if (control)
2024                {
2025                    wnd->send_event_to_control(control, ev);
2026                }
2027                wnd->send_event_to_plains(ev);
2028            }
2029        break;
2030        case WM_USER:
2031            reinterpret_cast<window*>(GetWindowLongPtr(hwnd, GWLP_USERDATA))
                    ->send_internal(internal_event_type::user_emitted, static_cast
                    <int32_t>(w_param), static_cast<int32_t>(l_param));
2032        break;
2033        case WM_DEVICECHANGE:
2034            reinterpret_cast<window*>(GetWindowLongPtr(hwnd, GWLP_USERDATA))
                    ->send_system(system_event_type::device_change, static_cast<
                    int32_t>(w_param), static_cast<int32_t>(l_param));
2035        break;
2036        case WM_DESTROY:
2037        {
2038            window* wnd = reinterpret_cast<window*>(GetWindowLongPtr(hwnd,
                    GWLP_USERDATA));
2039
2040            wnd->graphic_.release();
2041
2042            auto transient_window_ = wnd->get_transient_window();
2043            if (transient_window_)
2044            {
2045                transient_window_->enable();
2046            }
2047
2048            if (wnd->close_callback)
2049            {
2050                wnd->close_callback();
2051            }
2052
2053            wnd->context_.hwnd = 0;
2054        }
```

```
2055            break;
2056            default:
2057                return DefWindowProc(hwnd, message, w_param, l_param);
2058        }
2059        return 0;
2060    }
2061    }
```

graphic.cpp

```
1
2   #include <wui/graphic/graphic.hpp>
3   #include <wui/common/flag_helpers.hpp>
4   #include <wui/system/tools.hpp>
5
6   #include <boost/nowide/convert.hpp>
7
8   namespace wui
9   {
10
11  graphic::graphic(system_context &context__)
12      : context_(context__),
13        pc(context_),
14        max_size(),
15        background_color(0)
16      , mem_dc(0),
17        mem_bitmap(0),
18
19      err{}
20  {
21  }
22
23  graphic::~graphic()
24  {
25      release();
26  }
27
28  bool graphic::init(const rect &max_size_, color background_color_)
29  {
30      max_size = max_size_;
31      background_color = background_color_;
32
33      if (mem_dc)
34      {
35          err.type = error_type::already_runned;
36          err.component = "graphic::init()";
37          return false;
38      }
39
40      err.reset();
41
42      auto wnd_dc = GetDC(context_.hwnd);
43
44      mem_dc = CreateCompatibleDC(wnd_dc);
45
```

```
46      mem_bitmap = CreateCompatibleBitmap(wnd_dc, max_size.width(), max_size.
            height());
47      if (!mem_bitmap)
48      {
49          err.type = error_type::no_handle;
50          err.component = "graphic::init()";
51          err.message = "CreateCompatibleBitmap returns null";
52
53          ReleaseDC(context_.hwnd, wnd_dc);
54
55          return false;
56      }
57
58      SelectObject(mem_dc, mem_bitmap);
59
60      SetMapMode(mem_dc, MM_TEXT);
61
62      RECT filling_rect = { 0, 0, max_size.width(), max_size.height() };
63      FillRect(mem_dc, &filling_rect, pc.get_brush(background_color));
64
65      ReleaseDC(context_.hwnd, wnd_dc);
66
67      pc.init();
68
69      return true;
70  }
71
72  void graphic::release()
73  {
74      DeleteObject(mem_bitmap);
75      mem_bitmap = 0;
76
77      DeleteDC(mem_dc);
78      mem_dc = 0;
79
80      pc.release();
81  }
82
83  void graphic::set_background_color(color background_color_)
84  {
85      background_color = background_color_;
86
87      clear({ 0, 0, max_size.width(), max_size.height() });
88  }
89
90  void graphic::clear(const rect &position)
91  {
92      if (!mem_dc)
93      {
94          return;
95      }
96
97      RECT filling_rect = { position.left, position.top, position.right,
            position.bottom };
```

```
98      FillRect(mem_dc, &filling_rect, pc.get_brush(background_color));
99  }
100
101 void graphic::flush(const rect &updated_size)
102 {
103     auto wnd_dc = GetDC(context_.hwnd);
104
105     if (wnd_dc)
106     {
107         BitBlt(wnd_dc,
108             updated_size.left,
109             updated_size.top,
110             updated_size.width(),
111             updated_size.height(),
112             mem_dc,
113             updated_size.left,
114             updated_size.top,
115             SRCCOPY);
116     }
117
118     ReleaseDC(context_.hwnd, wnd_dc);
119 }
120
121 void graphic::draw_pixel(const rect &position, color color_)
122 {
123     SetPixel(mem_dc, position.left, position.top, color_);
124 }
125
126 void graphic::draw_line(const rect &position, color color_, uint32_t width)
127 {
128     auto old_pen = (HPEN)SelectObject(mem_dc, pc.get_pen(PS_SOLID, width,
        color_));
129
130     MoveToEx(mem_dc, position.left, position.top, (LPPOINT)NULL);
131     LineTo(mem_dc, position.right, position.bottom);
132
133     SelectObject(mem_dc, old_pen);
134 }
135
136 rect graphic::measure_text(std::string_view text_, const font &font__)
137 {
138     auto old_font = (HFONT)SelectObject(mem_dc, pc.get_font(font__));
139
140     RECT text_rect = { 0 };
141     auto wide_str = boost::nowide::widen(text_);
142     DrawTextW(mem_dc, wide_str.c_str(), static_cast<int32_t>(wide_str.size())
        , &text_rect, DT_CALCRECT);
143
144     SelectObject(mem_dc, old_font);
145
146     return {0, 0, text_rect.right, text_rect.bottom};
147 }
148
```

```
149 void graphic::draw_text(const rect &position, std::string_view text_, color
    color_, const font &font__)
150 {
151     auto old_font = (HFONT)SelectObject(mem_dc, pc.get_font(font__));
152 
153     SetTextColor(mem_dc, color_);
154     SetBkMode(mem_dc, TRANSPARENT);
155 
156     auto wide_str = boost::nowide::widen(text_);
157     TextOutW(mem_dc, position.left, position.top, wide_str.c_str(),
        static_cast<int32_t>(wide_str.size()));
158 
159     SelectObject(mem_dc, old_font);
160 }
161 
162 void graphic::draw_rect(const rect &position, color fill_color)
163 {
164     RECT position_rect = { position.left, position.top, position.right,
        position.bottom };
165     FillRect(mem_dc, &position_rect, pc.get_brush(fill_color));
166 }
167 
168 void graphic::draw_rect(const rect &position, color border_color, color
    fill_color, uint32_t border_width, uint32_t rnd)
169 {
170     auto old_pen = (HPEN)SelectObject(mem_dc, pc.get_pen(border_width != 0 ?
        PS_SOLID : PS_NULL, border_width, border_color));
171 
172     auto old_brush = (HBRUSH)SelectObject(mem_dc, pc.get_brush(fill_color));
173 
174     RoundRect(mem_dc, position.left, position.top, position.right, position.
        bottom, rnd, rnd);
175 
176     SelectObject(mem_dc, old_brush);
177 
178     SelectObject(mem_dc, old_pen);
179 }
180 
181 void graphic::draw_buffer(const rect &position, uint8_t *buffer, int32_t
    left_shift, int32_t top_shift)
182 {
183     auto source_bitmap = pc.get_bitmap(position.width(), position.height(),
        buffer, mem_dc);
184     auto source_dc = CreateCompatibleDC(mem_dc);
185     SelectObject(source_dc, source_bitmap);
186 
187     BitBlt(mem_dc,
188         position.left,
189         position.top,
190         position.width(),
191         position.height(),
192         source_dc,
193         left_shift,
194         top_shift,
```

```
195         SRCCOPY);
196
197     DeleteDC(source_dc);
198 }
199
200 void graphic::draw_graphic(const rect &position, graphic &graphic_, int32_t
        left_shift, int32_t top_shift)
201 {
202     if (graphic_.drawable())
203     {
204         BitBlt(mem_dc,
205             position.left,
206             position.top,
207             position.right,
208             position.bottom,
209             graphic_.drawable(),
210             left_shift,
211             top_shift,
212             SRCCOPY);
213     }
214 }
215
216 HDC graphic::drawable()
217 {
218     return mem_dc;
219 }
220
221 error graphic::get_error() const
222 {
223     return err;
224 }
225
226 }
```

locale.cpp

```
1
2 #include <wui/locale/locale.hpp>
3 #include <wui/locale/locale_impl.hpp>
4 #include <wui/locale/locale_selector.hpp>
5
6 namespace wui
7 {
8
9 static std::shared_ptr<i_locale> instance = nullptr;
10 static std::string dummy_string;
11 static std::vector<uint8_t> dummy_image;
12
13 /// Interface
14
15 bool set_locale_from_resource(locale_type type, std::string_view name,
        int32_t resource_index, std::string_view resource_section)
16 {
17     instance.reset();
18     instance = std::make_shared<locale_impl>(type, name);
```

```
19      instance->load_resource(resource_index, resource_section);
20
21      return instance->get_error().is_ok();
22  }
23
24  bool set_locale_from_json(locale_type type, std::string_view name, std::
        string_view json)
25  {
26      instance.reset();
27      instance = std::make_shared<locale_impl>(type, name);
28      instance->load_json(json);
29
30      return instance->get_error().is_ok();
31  }
32
33  bool set_locale_from_file(locale_type type, std::string_view name, std::
        string_view file_name)
34  {
35      instance.reset();
36      instance = std::make_shared<locale_impl>(type, name);
37      instance->load_file(file_name);
38
39      return instance->get_error().is_ok();
40  }
41
42  void set_locale_empty(locale_type type, std::string_view name)
43  {
44      instance.reset();
45      instance = std::make_shared<locale_impl>(type, name);
46  }
47
48  bool set_locale_from_type(locale_type type, error &err)
49  {
50      auto locale_params = wui::get_app_locale(type);
51
52      bool ok = wui::set_locale_from_resource(locale_params.type, locale_params
            .name, locale_params.resource_id, "JSONS");
53
54      err = instance->get_error();
55      return ok;
56  }
57
58  error get_locale_error()
59  {
60      if (instance)
61      {
62          instance->get_error();
63      }
64      return {};
65  }
66
67  std::shared_ptr<i_locale> get_locale()
68  {
69      return instance;
```

```
}

void set_locale_value(std::string_view section, std::string_view value, std::
    string_view str)
{
    if (instance)
    {
        instance->set(section, value, str);
    }
}

const std::string &locale(std::string_view section, std::string_view value)
{
    if (instance)
    {
        return instance->get(section, value);
    }
    return dummy_string;
}

}
```

theme.cpp

```
#include <wui/theme/theme.hpp>
#include <wui/theme/theme_impl.hpp>
#include <wui/theme/theme_selector.hpp>

namespace wui
{

static std::shared_ptr<i_theme> instance = nullptr;
static std::string dummy_string;
static std::vector<uint8_t> dummy_image;

/// Interface

bool set_default_theme_from_resource(std::string_view name, int32_t
    resource_index, std::string_view resource_section)
{
    instance.reset();
    instance = std::make_shared<theme_impl>(name);
    instance->load_resource(resource_index, resource_section);

    return instance->get_error().is_ok();
}

bool set_default_theme_from_json(std::string_view name, std::string_view json
    )
{
    instance.reset();
    instance = std::make_shared<theme_impl>(name);
    instance->load_json(json);

    return instance->get_error().is_ok();
```

```
}

bool set_default_theme_from_file(std::string_view name, std::string_view
    file_name)
{
    instance.reset();
    instance = std::make_shared<theme_impl>(name);
    instance->load_file(file_name);

    return instance->get_error().is_ok();
}

void set_default_theme_empty(std::string_view name)
{
    instance.reset();
    instance = std::make_shared<theme_impl>(name);
}

bool set_default_theme_from_name(std::string_view name, error &err)
{
    auto theme_params = wui::get_app_theme(name);

    bool ok = wui::set_default_theme_from_resource(name, theme_params.
        resource_id, "JSONS");

    err = instance->get_error();
    return ok;
}

error get_theme_error()
{
    if (instance)
    {
        instance->get_error();
    }
    return {};
}

std::shared_ptr<i_theme> get_default_theme()
{
    return instance;
}

std::shared_ptr<i_theme> make_custom_theme(std::string_view name)
{
    return std::make_shared<theme_impl>(name);
}

std::shared_ptr<i_theme> make_custom_theme(std::string_view name, std::
    string_view json)
{
    auto ct = std::make_shared<theme_impl>(name);
    ct->load_json(json);
    return ct;
```

```
81 }
82
83 color theme_color(std::string_view control, std::string_view value, std::
       shared_ptr<i_theme> theme_)
84 {
85     if (theme_)
86     {
87         return theme_->get_color(control, value);
88     }
89     else if (instance)
90     {
91         return instance->get_color(control, value);
92     }
93     return 0;
94 }
95
96 int32_t theme_dimension(std::string_view control, std::string_view value, std
       ::shared_ptr<i_theme> theme_)
97 {
98     if (theme_)
99     {
100        return theme_->get_dimension(control, value);
101    }
102    else if (instance)
103    {
104        return instance->get_dimension(control, value);
105    }
106    return 0;
107 }
108
109 const std::string &theme_string(std::string_view control, std::string_view
       value, std::shared_ptr<i_theme> theme_)
110 {
111    if (theme_)
112    {
113        return theme_->get_string(control, value);
114    }
115    else if (instance)
116    {
117        return instance->get_string(control, value);
118    }
119    return dummy_string;
120 }
121
122 font theme_font(std::string_view control, std::string_view value, std::
       shared_ptr<i_theme> theme_)
123 {
124    if (theme_)
125    {
126        return theme_->get_font(control, value);
127    }
128    else if (instance)
129    {
130        return instance->get_font(control, value);
```

```
131     }
132     return font();
133 }
134 
135 const std::vector<uint8_t> &theme_image(std::string_view name, std::
        shared_ptr<i_theme> theme_)
136 {
137     if (theme_)
138     {
139         return theme_->get_image(name);
140     }
141     else if (instance)
142     {
143         return instance->get_image(name);
144     }
145 
146     return dummy_image;
147 }
148 
149 }
```

i_control.hpp

```
1 
2 #pragma once
3 
4 #include <wui/common/error.hpp>
5 
6 #include <memory>
7 #include <string>
8 
9 namespace wui
10 {
11 
12 struct rect;
13 class graphic;
14 class window;
15 class i_theme;
16 
17 class i_control
18 {
19 public:
20     virtual void draw(graphic &gr, const rect &paint_rect) = 0;
21 
22     virtual void set_position(const rect &position, bool redraw = true) = 0;
23     virtual rect position() const = 0;
24 
25     virtual void set_parent(std::shared_ptr<window> window_) = 0;
26     virtual std::weak_ptr<window> parent() const = 0;
27     virtual void clear_parent() = 0;
28 
29     virtual void set_topmost(bool yes) = 0;
30     virtual bool topmost() const = 0;
31 
```

```
32      virtual void update_theme_control_name(std::string_view
           theme_control_name) = 0;
33      virtual void update_theme(std::shared_ptr<i_theme> theme_ = nullptr) = 0;
34
35      virtual void show() = 0;
36      virtual void hide() = 0;
37      virtual bool showed() const = 0;
38
39      virtual void enable() = 0;
40      virtual void disable() = 0;
41      virtual bool enabled() const = 0;
42
43      virtual bool focused() const = 0;  /// Returns true if the control is
           focused
44      virtual bool focusing() const = 0; /// Returns true if the control
           receives focus
45
46      virtual error get_error() const = 0;
47
48      friend class window;
49
50  protected:
51      ~i_control() {}
52
53  };
54
55  }
```

example.cpp

```
1  #include <wui/framework/framework.hpp>
2
3  #include <wui/config/config.hpp>
4
5  #include <wui/theme/theme.hpp>
6  #include <wui/theme/theme_selector.hpp>
7
8  #include <wui/locale/locale.hpp>
9  #include <wui/locale/locale_selector.hpp>
10
11 #include <MainFrame/MainFrame.h>
12
13 #include <Resource.h>
14
15 #include <iostream>
16
17 int APIENTRY wWinMain(_In_ HINSTANCE,
18     _In_opt_ HINSTANCE,
19     _In_ LPWSTR     lpCmdLine,
20     _In_ int        nCmdShow)
21 {
22     wui::framework::init();
23
24     auto ok = wui::config::create_config("hello_world.ini", "Software\\wui\\
           hello_world");
```

```
    if (!ok)
    {
        std::cerr << wui::config::get_error().str() << std::endl;
        return -1;
    }

    wui::error err;

    wui::set_app_locales({
        { wui::locale_type::eng, "English", "res/en_locale.json",
           TXT_LOCALE_EN },
        { wui::locale_type::rus, "Русский", "res/ru_locale.json",
           TXT_LOCALE_RU },
    });

    auto current_locale = static_cast<wui::locale_type>(wui::config::get_int(
       "User", "Locale",
        static_cast<int32_t>(wui::get_default_system_locale())));

    wui::set_current_app_locale(current_locale);

    wui::set_locale_from_type(current_locale, err);
    if (!err.is_ok())
    {
        std::cerr << err.str() << std::endl;
        return -1;
    }

    wui::set_app_themes({
        { "dark",  "res/dark.json",  TXT_DARK_THEME },
        { "light", "res/light.json", TXT_LIGHT_THEME }
    });

    auto current_theme = wui::config::get_string("User", "Theme", "dark");
    wui::set_default_theme("dark");
    wui::set_current_app_theme(current_theme);

    wui::set_default_theme_from_name(current_theme, err);
    if (!err.is_ok())
    {
        std::cerr << err.str() << std::endl;
        return -1;
    }

    MainFrame mainFrame;
    mainFrame.Run();

    wui::framework::run();

    return 0;
}
```

Место для диска