Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Южно-Уральский государственный университет (национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

# Разработка GUI библиотеки на языке C++

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

Автор работы:
Студент группы КЭ-402
Носков Артем Александрович

Челябинск
2024

# Актуальность

- Графические приложения – основной способ взаимодействия человека с компьютером
- Графический интерфейс – необходимая часть современного программного обеспечения
- Разработчики выдвигают множество требований к решениям для создания графических интерфейсов
- Имеются недостатки у существующих решений для создания графических интерфейсов

# Цель и задачи

**Цель:**

Разработка GUI библиотеки на языке C++

**Задачи:**

1) Провести обзор существующих библиотек
2) Привести описание требований к разрабатываемой библиотеке
3) Спроектировать и разработать библиотеку
4) Осуществить тестирование библиотеки
5) Создать демонстрационные приложения
6) Опубликовать библиотеку и приложения на GitHub

# Предметная область проекта

1. Программные интерфейсы для отрисовки графических элементов: OpenGL, Vulkan, Direct3D и Metal
2. Skia - библиотека 2D-графики с открытым исходным кодом, используется в проектах Chromium, Android и Flutter
3. Взаимодействие графических приложений с операционной системой и ее графической оболочкой
4. Использование интерфейсов операционных систем для создания окон и отрисовки графики: WinAPI, Carbon
5. Кроссплатформенные обертки интерфейсов операционных систем: GLFW, SDL и Winit

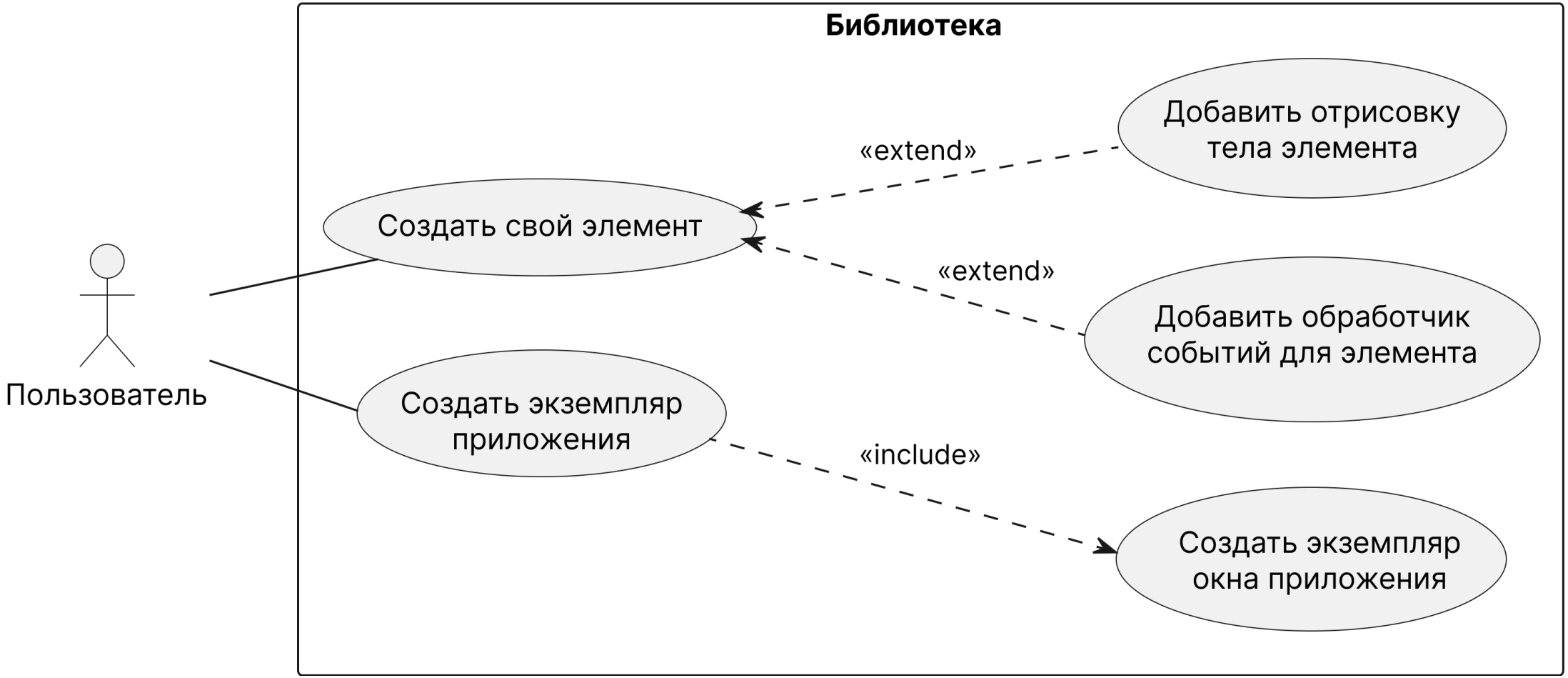
# Обзор существующих решений

| **Решение** | **Описание** | **Язык, на котором разработан фреймворк** | **Язык, на котором разрабатываются приложения** |
|---|---|---|---|
| Qt | Фреймворк для создания графических приложений на множестве платформ | C++ | C++, Python, QML, C#, F#, Java, PHP, Go и другие |
| GTK | Библиотека с открытым исходных кодом для создания GUI-приложений на различных операционных системах | C | C, C++, Python, Go, D, Erlang, Fortran, Java, Haskell и другие |
| Flutter | Фреймворк для разработки приложений (мобильные, web, настольные) | C++, Dart | Dart |

# Практическая значимость

1. Библиотека, а не фреймворк
2. Переиспользование элементов:
   создать свою кнопку из базовой кнопки
3. Переиспользование отдельных свойств:
   Focusable, Inputable, Hoverable и т.д. для собственного поля ввода
4. Легко создавать свои элементы
   1. Любое расположение элементов, например: Flex, Grid, Constraints и т.д.
   2. Любое отображение – все что есть в Skia или в отдельном графическом интерфейсе

# Диаграмма вариантов использования



# Создание элемента

1. Объявить класс, наследуемый от абстрактного класса `Element`
2. Переопределить необходимые методы класса `Element`
    1) Определение размера: `getSize`
    2) Отрисовка: `paintBackground`
    3) Определение дочерних элементов: `getChildren`
3. Добавить к наследуемым классам класс `WithEventsHandlers`
4. Переопределить метод `getEventsHandlers` класса `WithEventsHandlers` для обработки событий

# Создание элемента CustomImage

```
struct CustomImageProps {
    std::optional<std::string> path = std::nullopt;
};

class CustomImage : public Element {
public:
  Image(
    const std::shared_ptr<ApplicationContext>& ctx,
    const ImageProps& props
  ) : Element(ctx) {
    setPath(props.path);
  }

  void setPath(std::optional<std::string> newPath) {
    markChanged();
    path = newPath;
  }

  [[nodiscard]] std::string getPath() const {
    return path;
  }

private:
    std::optional<std::string> path = std::nullopt;
};
```

# Изменение элемента CustomImage

```
class CustomImage : public Element {
public:
  ...

  Size getSize(const Boundaries& boundaries) override {
    float pixelScale = ctx->getPixelScale();
    float imageWidthScaled = skImage->width() * pixelScale;
    float imageHeightScaled = skImage->height() * pixelScale;

    Size size = { widthScaled, heightScaled };
    size = fitSizeInBoundaries(size, boundaries);

    return size;
  }

  void paintBackground(SkCanvas* canvas, const ElementRect& rect) override {
    SkRect skRect = SkRect::MakeXYWH(0, 0, rect.size.width, rect.size.height);
    canvas->drawImageRect(skImage, skRect, samplingOptions);
  }
```

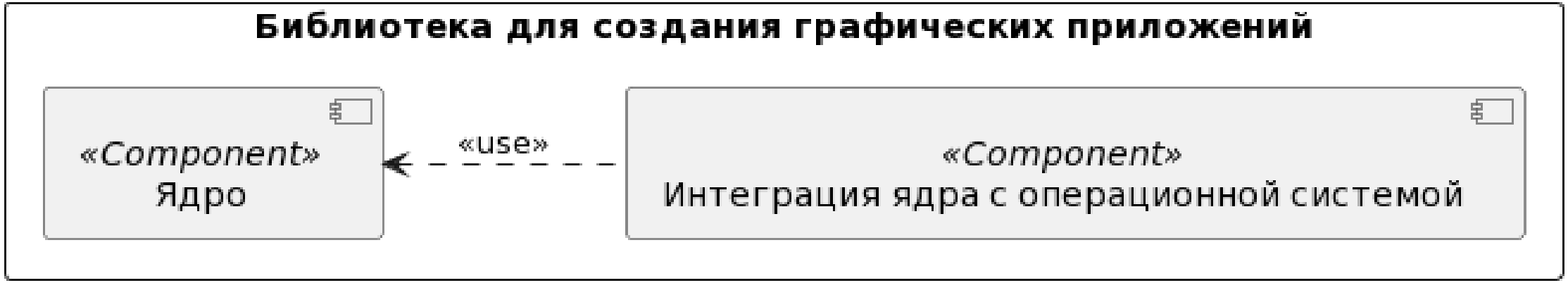
# Создание элемента CustomButton

```
class CustomButton : public Element, public WithEventsHandlers {
public:
   ...

  std::vector<std::shared_ptr<EventHandler>>
  getEventsHandlers() override {
    return {
      KeyboardEvent::Handle([this](const auto& event) {
        if (event->key == KeyboardKey::Escape &&
            event->action == KeyAction::Release) {
            blur();
            return EventCallbackResponse::StopPropagation;
        }

         ...

        return EventCallbackResponse::None;
      })
    };
  };
```

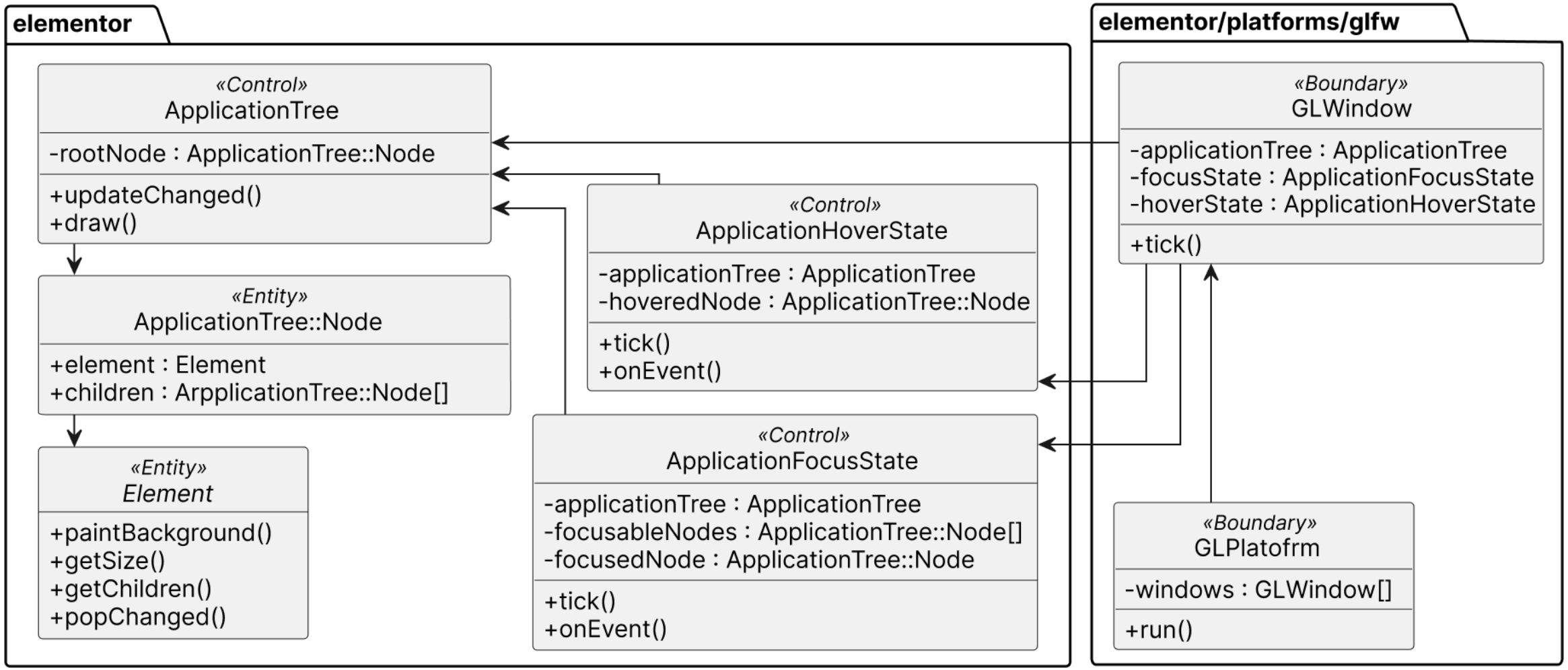
# Создание экземпляра приложения

```
1  int main() {
2      auto platform = std::make_shared<GLPlatform>();
3
4      // Создание окна приложения
5      auto window = std::make_shared<GLWindow>(platform);
6      window->setTitle("Application Window Title");
7      window->setSize({ 480, 720 });
8      window->setMinSize({ 240, 240 });
9      platform->addWindow(window);
10
11     // Создание экземпляр приложения
12     auto application = Todo::New(window, {
13         .filename = asset("todo.md"),
14     });
15
16     // Добавление приложения к окну
17     window->setRoot(application);
18
19     // Запуск приложения
20     platform->run();
21 }
```

# Архитектура системы



# Архитектура системы



# Алгоритм работы приложения

1. Планировщик событий вызывает метод `tick` у окна приложения
2. Проверяется размер окна ОС
   - При изменении вызывается метод `resize`
3. Вызывается метод `tick` у состояний приложения (`hover, focus`)
4. Применяются возникшие события
5. Проверяется наличие изменений в дереве элементов
   - При наличии изменений, вызывается метод обновления кэша дерева элементов
6. Происходит отрисовка приложения

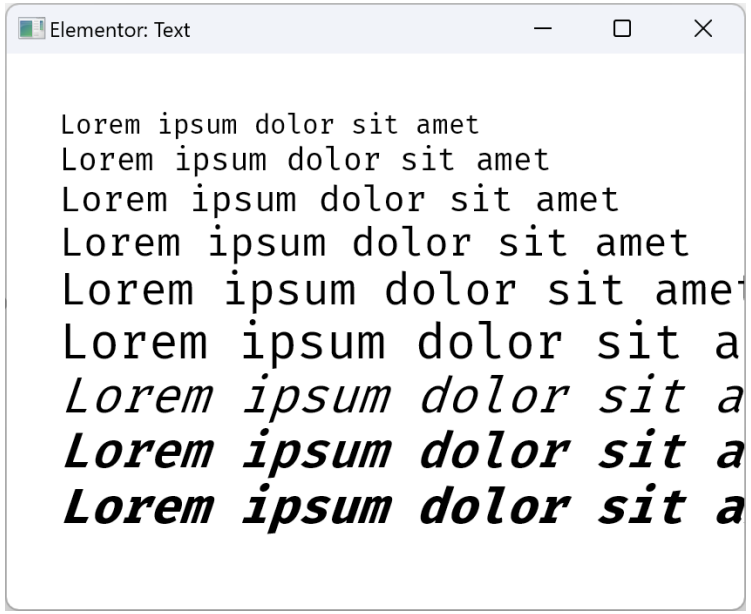
# Реализованные элементы

## Всего элементов: **33**

- Примитивные элементы (27):
  - Расположение элементов (12):
    `Align`, `Column`, `FitContain`, `FitCover`, `Flex`, `Flexible`, `Height`, `Padding`, `Ratio`, `Row`, `Stack`, `Width`
  - Пользовательское взаимодействие (8):
    `Clickable`, `ClickableOutside`, `Cursorable`, `Draggable`, `Eventable`, `Focusable`, `Hoverable`, `Scrollable`
  - Отрисовка графики (7):
    `Background`, `Border`, `Image`, `Paragraph`, `Paragraph Placeholder`, `SVG`, `Text`
- Составные элементы (6):
  `Button`, `TextButton`, `IconButton`, `Scrollbar`, `Scroll`, `FPSLabel`

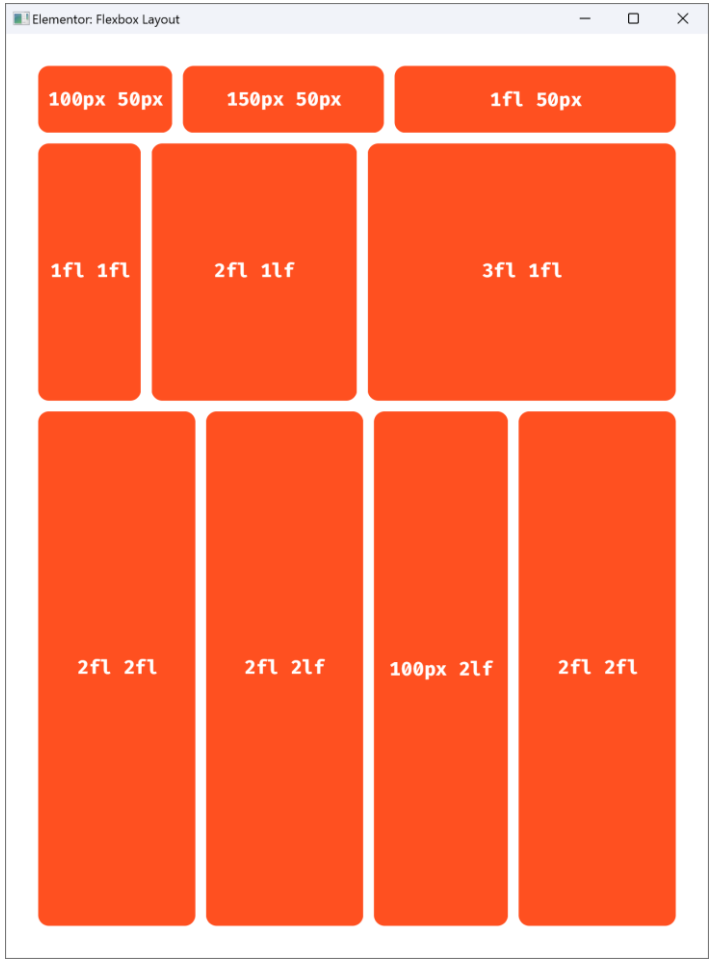
# Элемент «Текст»

```
Flex::New(ctx, {
  .spacing = 2,
  .direction = FlexDirection::Column,
  .children = {
    Text::New(ctx, {
      .text = lipsumText,
      .fontSize = 6,
      .fontFamily = "Fira Code",
    }),
    Text::New(ctx, {
      .text = lipsumText,
      .fontSize = 16,
      .fontSlant = FontSlant::Italic,
      .fontFamily = "Fira Code",
    }),
    Text::New(ctx, {
      .text = lipsumText,
      .fontSize = 16,
      .fontWeight = 1000,
      .fontSlant = FontSlant::Oblique,
      .fontFamily = "Fira Code",
    }),
    ...
```

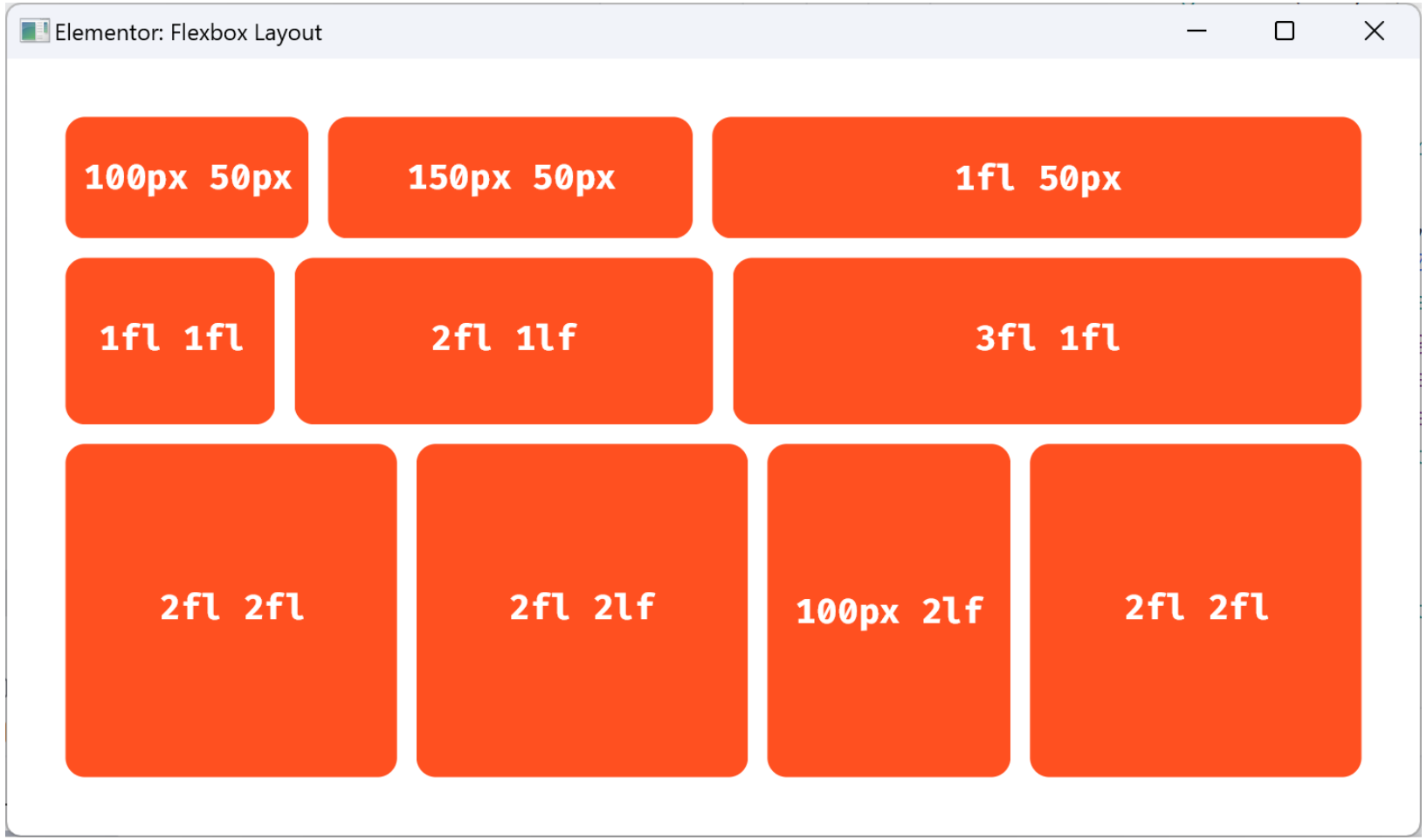


# Элемент «Flex»

```
Flex::New(ctx, {
  .spacing = 8,
  .direction = FlexDirection::Column,
  .children = {
    Height::New(ctx, {
      .height = 50,
      .child = Flex::New(ctx, {
        .spacing = 8,
        .alignment = FlexAlignment::Stretch,
        .children = {
          Width::New(ctx, {
            .width = 100,
            .child = Box(ctx, "100px 50px"),
          }),
          Width::New(ctx, {
            .width = 150,
            .child = Box(ctx, "150px 50px"),
          }),
          Flexible::New(ctx, {
            .grow = 1,
            .child = Box(ctx, "1fl 50px"),
          })
        }
      }),
    }),
    Flexible::New(ctx, {
      .grow = 1,
      .child = Flex::New(ctx, {
        ...
```

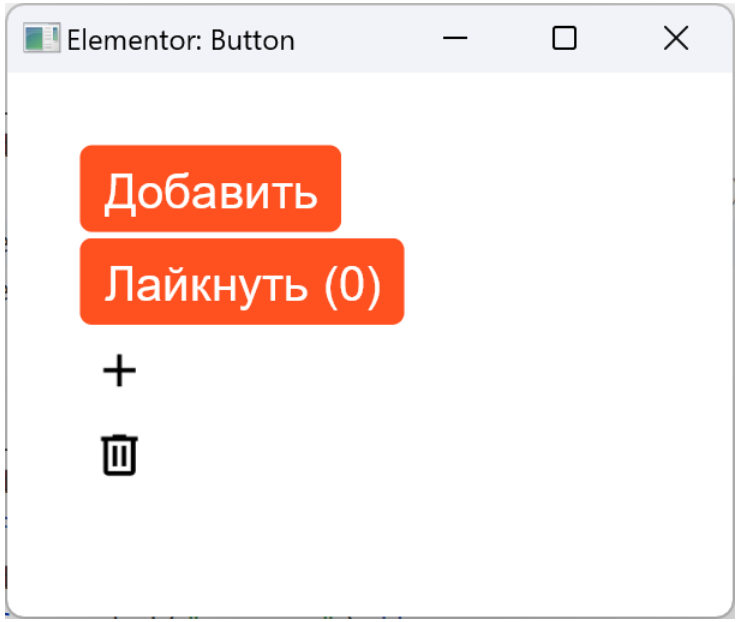


# Элемент «Flex»



# Элемент «Кнопка»

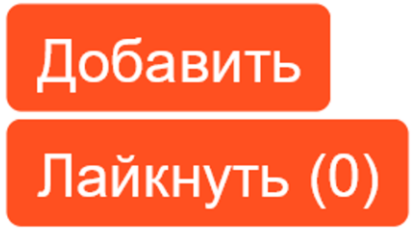
```
Flex::New(ctx, {
  .spacing = 2,
  .direction = FlexDirection::Column,
  .children = {
    TextButton::New(ctx, {
      .text = "Добавить",
      .fontColor = "#fff",
      .backgroundColor = "#ff5020",
    }),
    LikeButton::New(ctx),
    IconButton::New(ctx, {
      .src = asset("add.svg"),
    }),
    IconButton::New(ctx, {
      .src = asset("delete.svg"),
    })
  }
})
```



# Реализация элемента «Кнопка»

```
// Button
Outline::New(ctx, {
  .border = {
    .radius = 6,
    .width = 3,
    .color = "#21CFFF",
    .style = BorderStyle::Dashed,
  },
  .offset = 6,
  .child = Focusabled::New(ctx, {
    .child = Cursorable::New(ctx, {
      .cursorShape = CursorShape::Hand,
      .child = ClickableOutside::New(ctx, {
        .onClickOutside = [this]() {
          blur();
        },
        .child = Clickable::New(ctx, {
          .onButton = [this]( ... ) { ... },
          .child = props.child,
        })
      })
    })
  })
})
```

# Реализация элемента «Кнопка»




```
// TextButton
Button::New(ctx, {
  .onClick = props.onClick,
  .child = Rounded::New(ctx, {
    .all = 4,
    .child = Background::New(ctx, {
      .color = props.backgroundColor,
      .child = Padding::New(ctx, {
        .all = 8,
        .top = 6,
        .child = Text::New(ctx, {
          .text = props.text,
          .fontColor = props.fontColor,
          .fontSize = 16,
          .fontFamily = "Arial",
        })
      })
    })
  })
})
```

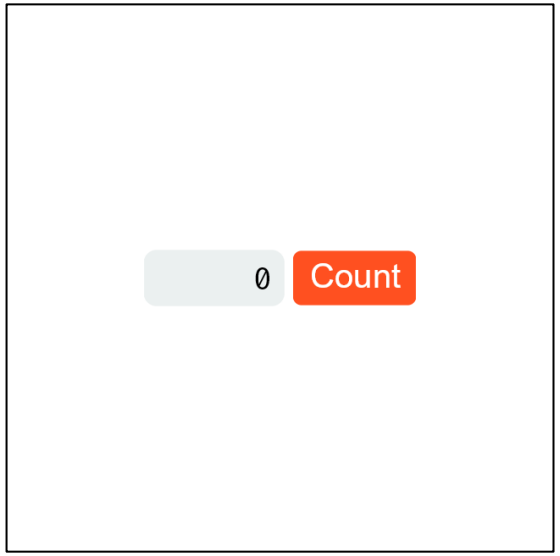
# Реализация элемента «Кнопка»

```
// IconButton
Button::New(ctx, {
  .onClick = props.onClick,
  .child = Rounded::New(ctx, {
    .all = props.size / 2,
    .child = Background::New(ctx, {
      .color = props.backgroundColor,
      .child = Padding::New(ctx, {
        .all = 4,
        .child = Width::New(ctx, {
          .width = props.size,
          .child = Height::New(ctx, {
            .height = props.size,
            .child = SVG::New(ctx, {
              .src = props.src
            })
          })
        })
      })
    })
  })
})
```

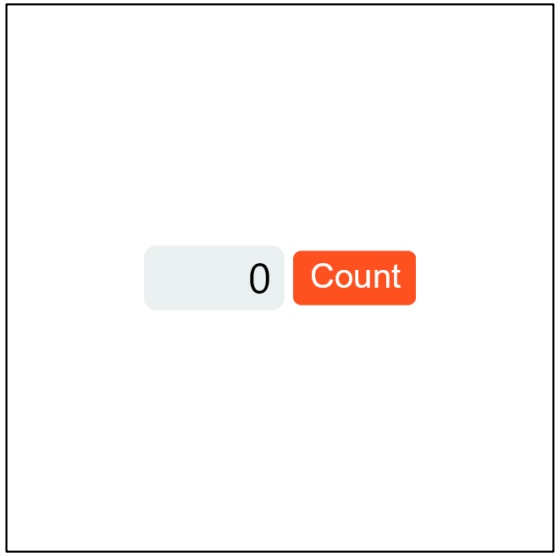
# Алгоритм тестирования

1. Создается эталонный скриншот приложения, который проверяется пользователем
2. Вносятся изменения в кодовую базу
3. Создается новый скриншот приложения
4. Вычисляется разница между новым и эталонным скриншотами
    1. Если изображения идентичны, то тест пройден
    2. Если изображения расходятся, то:
        1. Разница сохраняется в виде изображения
        2. Разница выводится в консоль
        3. Тест не пройден

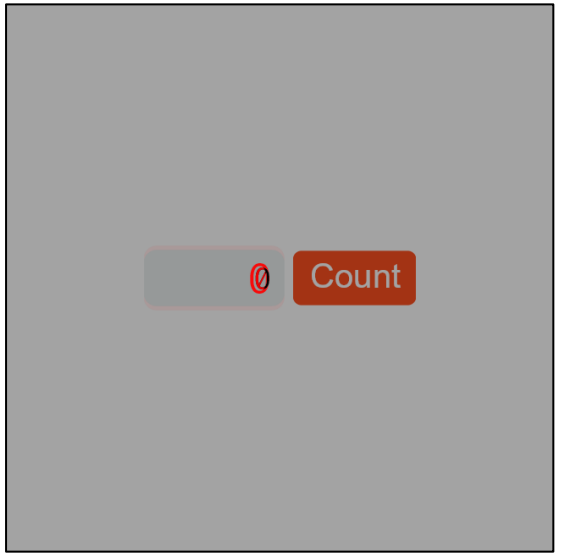
# Пример тестирования




Было

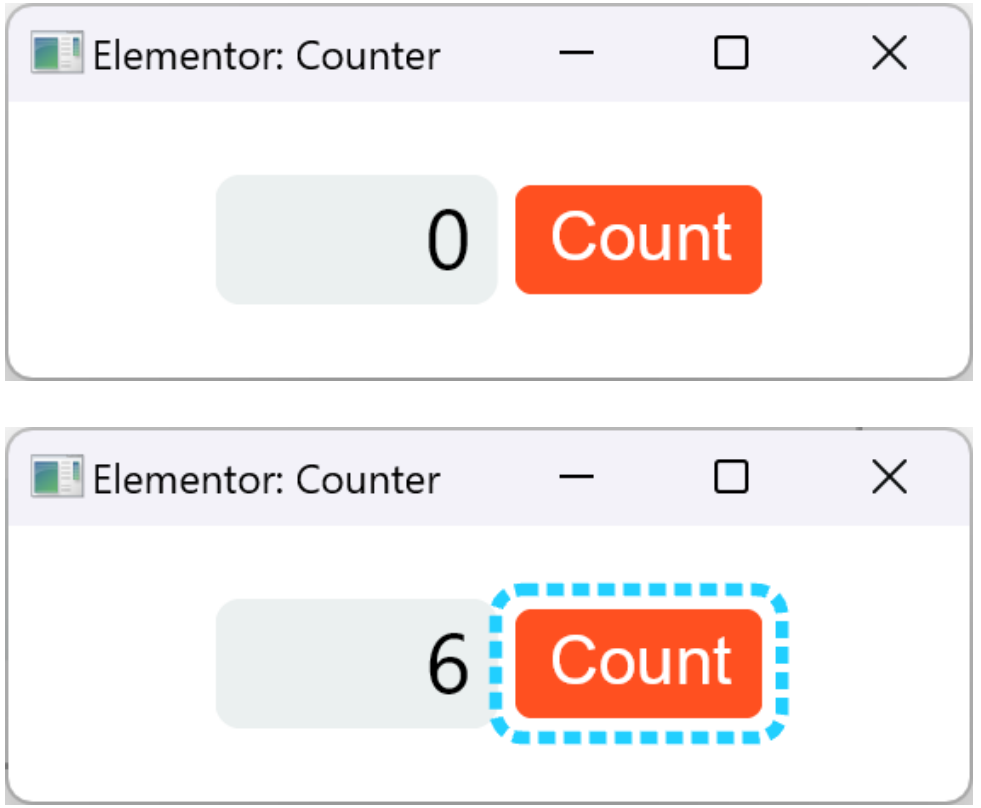



Стало

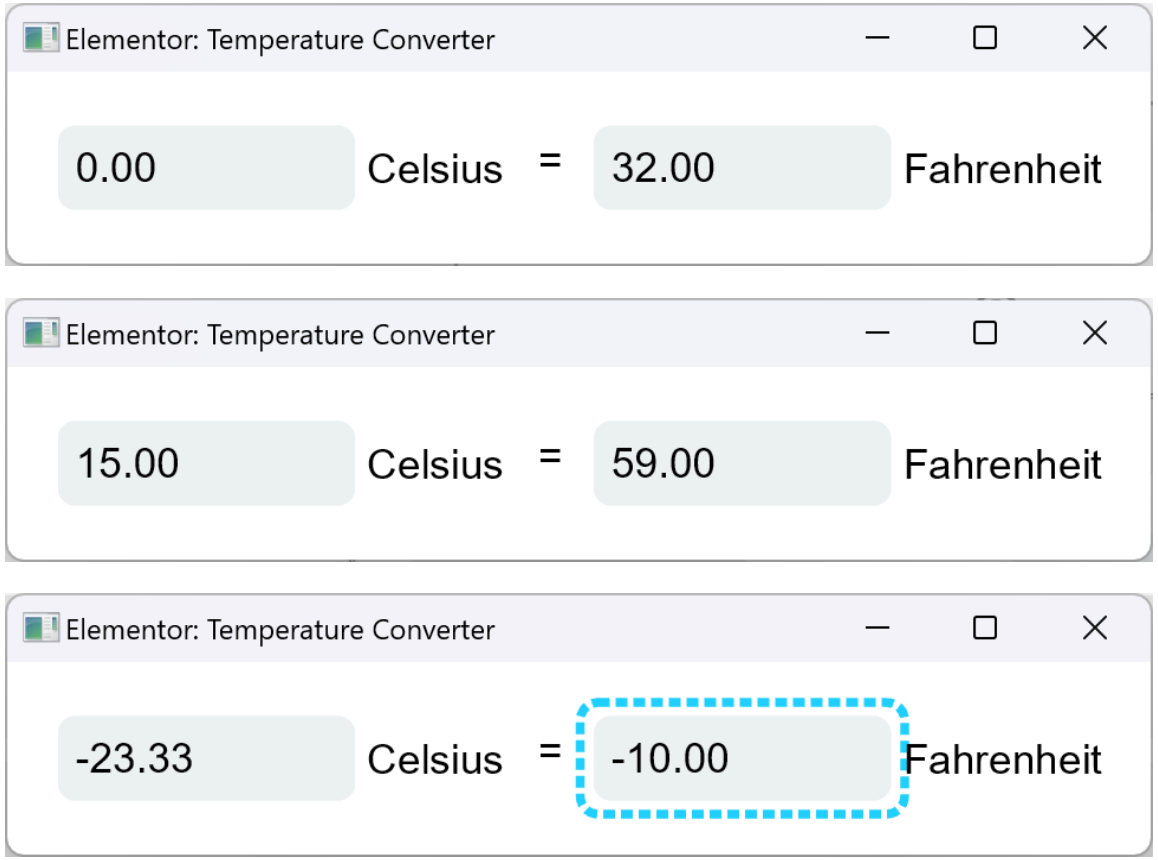



Что изменилось

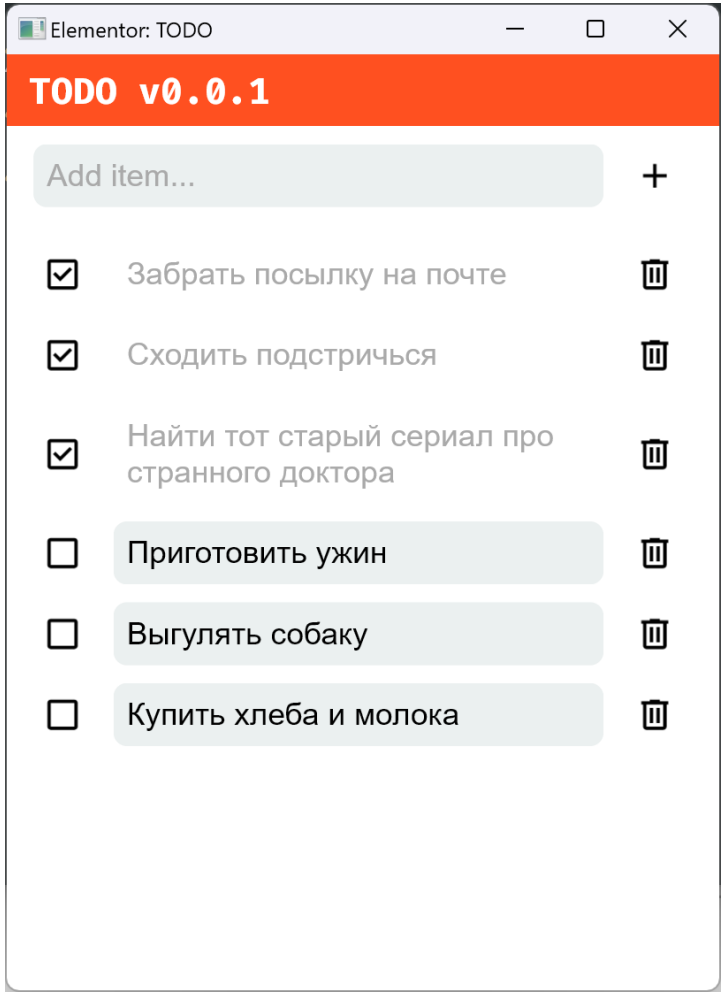
# Демонстрационные приложения

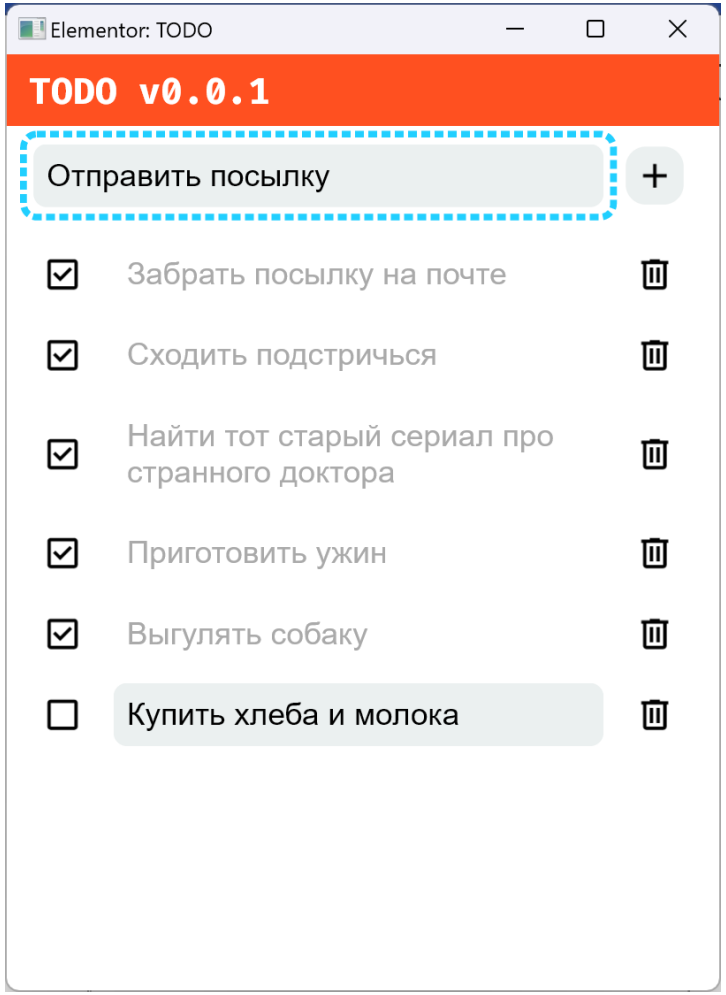


# Демонстрационные приложения



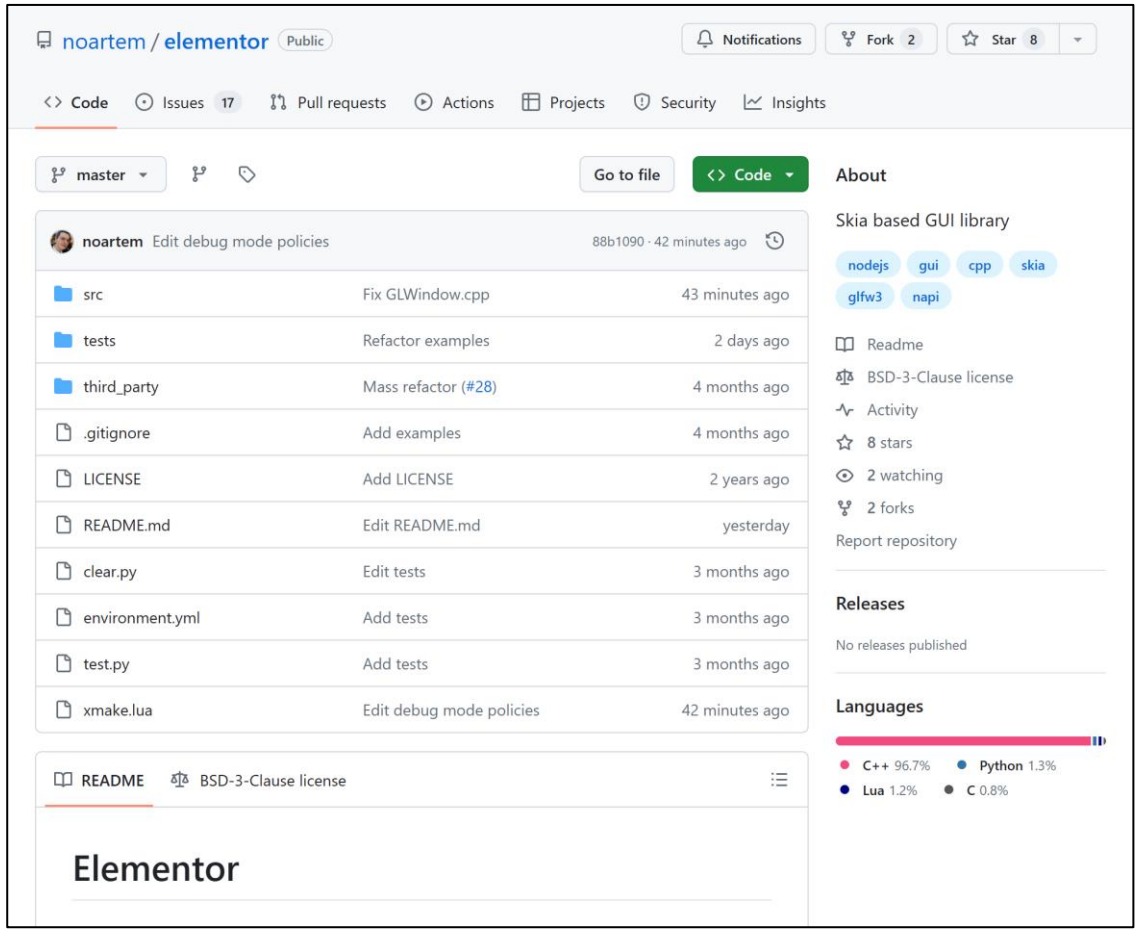
# Демонстрационные приложения



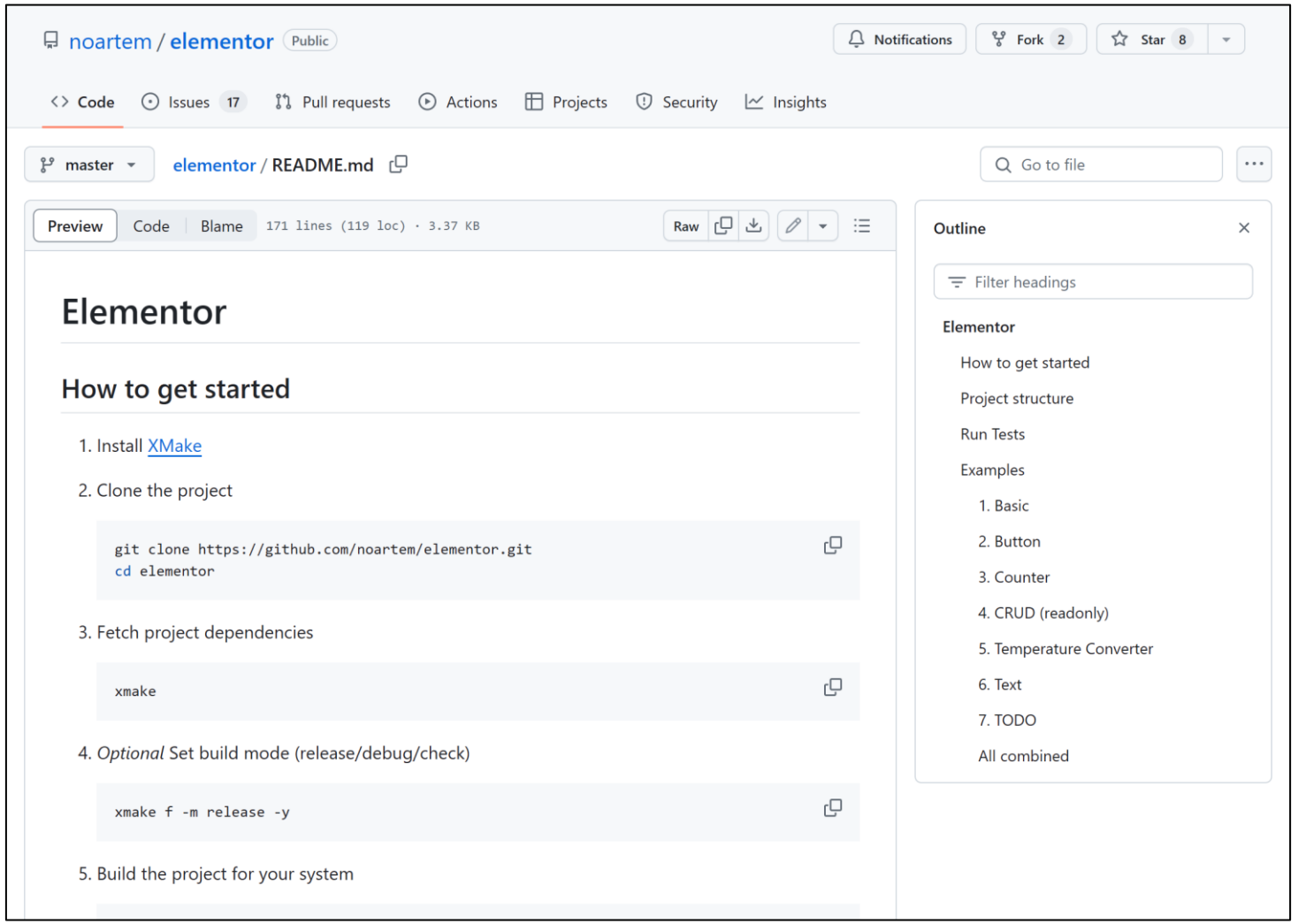



# Публикация проекта на GitHub

github.com/noartem/elementor



# Инструкция для проекта на GitHub



# Основные результаты

1. Произведен анализ предметной области
2. Приведены требования к разрабатываемой библиотеке
3. Выполнено проектирование архитектуры библиотеки и ее реализация
4. Осуществлено тестирование библиотеки
5. Разработаны демонстрационные приложения
6. Библиотека и приложения опубликованы на GitHub: github.com/noartem/elementor.