INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

MATHEUS AUDIBERT

Projeto Final

CAMPOS DO JORDÃO

2025

RESUMO

Este trabalho apresenta o desenvolvimento de uma aplicação To-Do List em C++ com Qt Creator, criada como projeto final com o objetivo de aplicar e consolidar conhecimentos em programação orientada a objetos (POO). O sistema permite adicionar, marcar como concluídas e remover tarefas, com uma interface gráfica simples e funcional usando Qt Widgets. O projeto foi desenvolvido de forma modular, reforçando boas práticas de organização e reutilização de código, e mostrou-se adequado para o aprendizado prático de conceitos fundamentais de POO e interfaces gráficas.

Palavras-Chave: C++; Qt; POO; Lista de Tarefas; Interface Gráfica.

ABSTRACT

This work presents the development of a To-Do List application in C++ using Qt Creator, created as a final project with the goal of applying and consolidating knowledge in object-oriented programming (OOP). The system allows users to add tasks, mark them as completed, and remove them through a simple and functional graphical interface built with Qt Widgets. The project was developed in a modular way, reinforcing good practices of code organization and reuse, and proved to be suitable for the practical learning of fundamental OOP concepts and graphical interface development.

Keywords: C++; Qt; OOP; To-Do List; Graphical Interface.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de aplicações com interface gráfica é uma etapa fundamental no processo de formação em programação, por permitir a aplicação prática de conceitos de lógica, estruturação de código e interação com o usuário. Neste contexto, o presente trabalho tem como proposta a criação de uma aplicação do tipo To-Do List, desenvolvida em linguagem C++ com o uso da ferramenta Qt Creator e da biblioteca gráfica Qt. A escolha deste projeto se deu com o intuito de consolidar os conhecimentos adquiridos ao longo da disciplina, especialmente os relacionados à programação orientada a objetos (POO) e ao desenvolvimento de interfaces visuais.

O objetivo principal do projeto é criar uma aplicação funcional e intuitiva que permita ao usuário adicionar, marcar como concluídas e remover tarefas. Por meio dessa proposta, foram explorados conceitos importantes como manipulação de eventos, criação de componentes gráficos, controle de estados e organização modular do código. Dessa forma, o projeto não apenas reforça os fundamentos teóricos aprendidos em sala de aula, como também proporciona uma experiência prática significativa no desenvolvimento de aplicações desktop.

1.1 Objetivos

O objetivo deste projeto foi desenvolver uma aplicação de lista de tarefas (To-Do List) utilizando a linguagem C++ com a ferramenta Qt Creator. O foco foi praticar os conceitos aprendidos em sala de aula, principalmente lógica de programação e programação orientada a objetos. Além disso, buscou-se criar uma interface gráfica simples e funcional, onde o usuário pudesse adicionar tarefas, marcá-las como concluídas e removê-las com facilidade. Para isso, foram definidos os seguintes objetivos específicos:

- Aplicar os conceitos de programação orientada a objetos (POO) na estruturação do código-fonte da aplicação;
- Construir uma interface gráfica funcional e intuitiva utilizando os recursos da biblioteca Qt;

- Das funcionalidades principais de uma lista de tarefas: adicionar, concluir e remover itens da lista.

1.2 Justificativa

Este trabalho é importante porque ajuda a colocar em prática os conceitos de programação orientada a objetos e lógica de programação aprendidos em aula. Criar uma aplicação de To-Do List em C++ com Qt Creator permite aprender a construir um programa com interface gráfica, tratando eventos e interagindo com o usuário.

Além disso, o projeto é simples e focado, facilitando o aprendizado e ajudando a entender como organizar o código e usar componentes visuais.

Por fim, essa aplicação serve como um bom exemplo para outros estudantes que querem praticar programação com C++ e desenvolver interfaces gráficas de forma prática e aplicada.

2 FUNDAMENTAÇÃO TEÓRICA

O desenvolvimento da aplicação To-Do List baseou-se na aplicação prática dos conceitos de programação orientada a objetos, manipulação de eventos e criação de interfaces gráficas. A ferramenta Qt Creator foi usada como IDE, oferecendo suporte ao desenvolvimento em C++ com a biblioteca Qt, facilitando o design da interface, gerenciamento do projeto e depuração. A biblioteca Qt Widgets permitiu construir uma interface funcional e responsiva, enquanto a combinação de Qt Creator e POO garantiu um código modular e de fácil manutenção.

2.1 Ferramentas e Bibliotecas

A aplicação foi desenvolvida em C++ utilizando o Qt Creator, um ambiente de desenvolvimento integrado (IDE) que oferece ferramentas completas para criação, edição, compilação e depuração de projetos. O Qt Creator facilita o desenvolvimento ao integrar recursos visuais para o design da interface gráfica, gerenciamento de código e configuração do projeto.

Para a construção da interface gráfica, foi utilizada a biblioteca Qt Widgets, que disponibiliza componentes visuais prontos, como botões, caixas de texto e listas, permitindo criar interfaces funcionais e responsivas de forma prática e organizada. O uso dos Widgets facilita o gerenciamento de eventos e a interação do usuário, simplificando o desenvolvimento de aplicações desktop com interfaces gráficas.

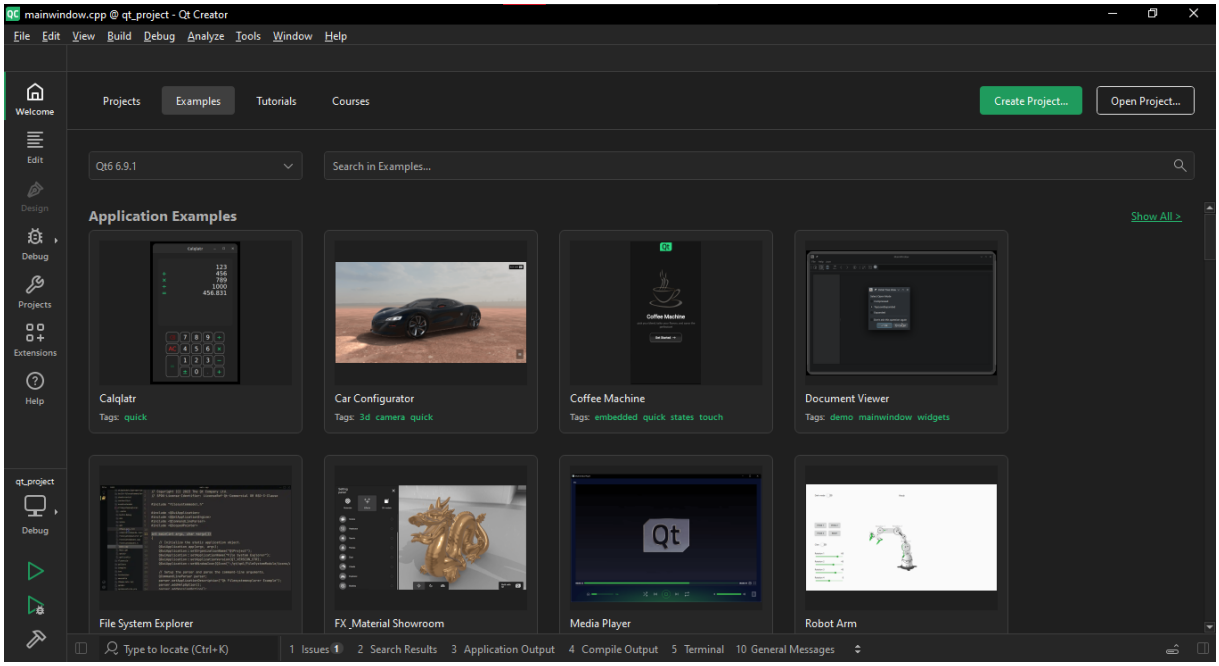

Figura 1 – Tela inicial QtCreator (Autor, 2025)

3 PROJETO PROPOSTO

O projeto proposto foi desenvolver uma aplicação To-Do List em C++ com interface gráfica usando Qt Creator. A ideia principal é criar um programa que permita ao usuário adicionar, marcar como concluídas e remover tarefas de forma simples e prática.

3.1 Desenvolvimento do Projeto

O desenvolvimento da aplicação foi feito passo a passo, focando nas funções principais e na organização do código. Para mostrar como o projeto funciona, serão apresentados trechos do código que mostram como as tarefas são adicionadas, marcadas e removidas.

Primeiro, foi criada a interface gráfica do projeto usando o arquivo mainwindow.ui no Qt Creator, onde foram posicionados os elementos visuais como botões, caixas de texto e listas para facilitar a interação do usuário.

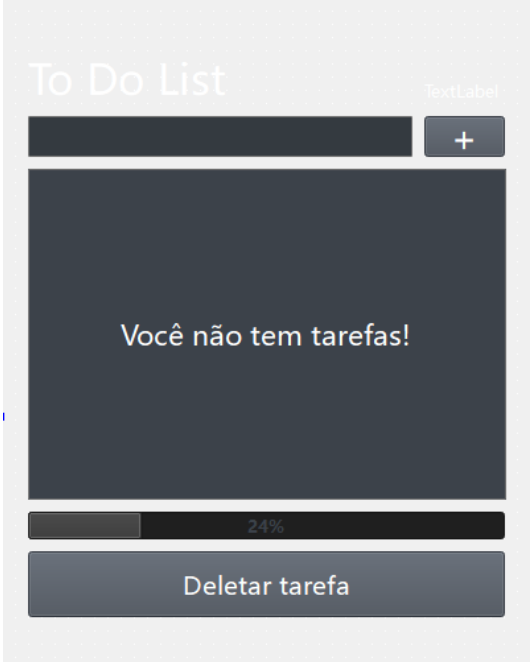

Figura 2 – Interface gráfica (UI) (Autor, 2025)

Depois, foram desenvolvidos os métodos da aplicação, responsáveis por adicionar, remover e atualizar as tarefas, além de gerenciar a interface e o comportamento dos elementos visuais.

```
private slots:
    void adicionarTarefa(); //Adiciona uma tarefa
    void removerTarefas(); // Remove X tarefas
    void atualizarHora(); // Atualiza o horário
    void atualizarProgresso(); // Atualiza a barra de progresso
    void atualizarMensagemVazio(); // Mensagem inicial/final
    void atualizarEstadoBotoes(); // Desabiliar/Abilitar botão

private:
    Ui::MainWindow *ui;
    QWidget* criarTarefa(const QString& texto);
};
```

Figura 3 – Métodos (Autor, 2025)

```
QWidget* MainWindow::criarTarefa(const QString& texto) {
    QWidget* widget = new QWidget();
    QHBoxLayout* layout = new QHBoxLayout(widget);
    layout->setContentsMargins(8, 8, 8, 8);

    QCheckBox* checkbox = new QCheckBox(texto);
    QFont font = checkbox->font();
    font.setPointSize(14);
    checkbox->setFont(font);

    // Estilo
    checkbox->setStyleSheet(R"(
        QCheckBox {
            spacing: 12px;
            color: white;
        }
        QCheckBox::indicator {
            width: 18px;
            height: 18px;
            border-radius: 7px;
            border: 2px solid #3498db;
            background-color: white;
        }
        QCheckBox::indicator:checked {
            border: 2px solid #2ecc71;
            background-color: #25a059;
        }
    )");

    layout->addWidget(checkbox, 0, Qt::AlignVCenter);

    connect(checkbox, &QCheckBox::stateChanged, this, &MainWindow::atualizarProgresso);

    return widget;
}
```

Figura 4 – Método criarTarefa (Autor, 2025)

```cpp
void MainWindow::adicionarTarefa() {
    QString texto = ui->lineEdit->text().trimmed();
    if (!texto.isEmpty()) {
        QListWidgetItem* item = new QListWidgetItem();
        QWidget* widget = criarTarefa(texto);

        item->setSizeHint(widget->sizeHint());
        ui->listWidget->addItem(item);
        ui->listWidget->setItemWidget(item, widget);

        ui->lineEdit->clear();
        atualizarProgresso();
        atualizarMensagemVazio();
        atualizarEstadoBotoes();
    }
}
```

Figura 5 – Método adicionarTarefa (Autor, 2025)

```cpp
void MainWindow::removerTarefas() {
    for (int i = ui->listWidget->count() - 1; i >= 0; --i) {
        QListWidgetItem* item = ui->listWidget->item(i);
        QWidget* widget = ui->listWidget->itemWidget(item);
        QCheckBox* checkbox = widget->findChild<QCheckBox*>();

        if (checkbox && checkbox->isChecked()) {
            delete ui->listWidget->takeItem(i);
        }
    }
    atualizarProgresso();
    atualizarMensagemVazio();
    atualizarEstadoBotoes();
}
```

Figura 6 – Método removerTarefa (Autor, 2025)

```cpp
void MainWindow::atualizarHora() {
    QString horaAtual = QDateTime::currentDateTime().toString("HH:mm:ss");
    ui->labelHora->setText(horaAtual);
}
```

Figura 7 – Método atualizarHora (Autor, 2025)

```cpp
void MainWindow::atualizarProgresso() {
    int total = ui->listWidget->count();
    if (total == 0) {
        ui->progressBar->setValue(0);
        return;
    }

    int feitas = 0;
    for (int i = 0; i < total; ++i) {
        QWidget* widget = ui->listWidget->itemWidget(ui->listWidget->item(i));
        QCheckBox* checkbox = widget->findChild<QCheckBox*>();
        if (checkbox && checkbox->isChecked()) {
            feitas++;
        }
    }

    int progresso = static_cast<int>((feitas * 100.0) / total);
    ui->progressBar->setValue(progresso);
}
```

Figura 8 – Método atualizarProgresso (Autor, 2025)

```cpp
void MainWindow::atualizarMensagemVazio() {
    bool listaVazia = (ui->listWidget->count() == 0);
    ui->emptyLabel->setVisible(listaVazia);
}
```

Figura 9 – Método atualizarMensagemVazio (Autor, 2025)

```cpp
void MainWindow::atualizarEstadoBotoes() {
    bool temTarefas = (ui->listWidget->count() > 0);
    ui->removeButton->setEnabled(temTarefas);
}
```

Figura 10 – Método atualizarEstadoBotoes (Autor, 2025)

4 AVALIAÇÃO

A aplicação To-Do List mostrou-se funcional, com as principais funcionalidades implementadas corretamente. A adição, marcação e remoção de tarefas funcionaram conforme o esperado, proporcionando uma experiência simples e eficiente para o usuário.

Durante os testes, o programa apresentou boa estabilidade e resposta rápida aos comandos. A interface gráfica foi considerada clara e intuitiva, facilitando a interação mesmo para usuários sem experiência prévia.

De modo geral, o projeto atingiu os objetivos propostos, oferecendo uma aplicação prática para o aprendizado de programação orientada a objetos e desenvolvimento de interfaces. Futuras melhorias podem incluir a persistência dos dados e personalização da interface para tornar o sistema ainda mais completo e atrativo.

4.1 Resultados

A aplicação To-Do List foi desenvolvida com uma organização clara do código, separando a interface gráfica e a lógica do programa. Durante os testes, todas as funcionalidades principais funcionaram corretamente: adicionar tarefas, marcar como concluídas e removê-las.

A interface mostrou-se estável e responsiva, permitindo ao usuário interagir facilmente com os elementos visuais. A barra de progresso atualiza conforme o andamento das tarefas, e mensagens indicativas aparecem quando a lista está vazia, melhorando a usabilidade.

De modo geral, a aplicação apresentou bom desempenho e atendeu aos objetivos propostos, demonstrando ser uma ferramenta simples e eficiente para o gerenciamento básico de tarefas.

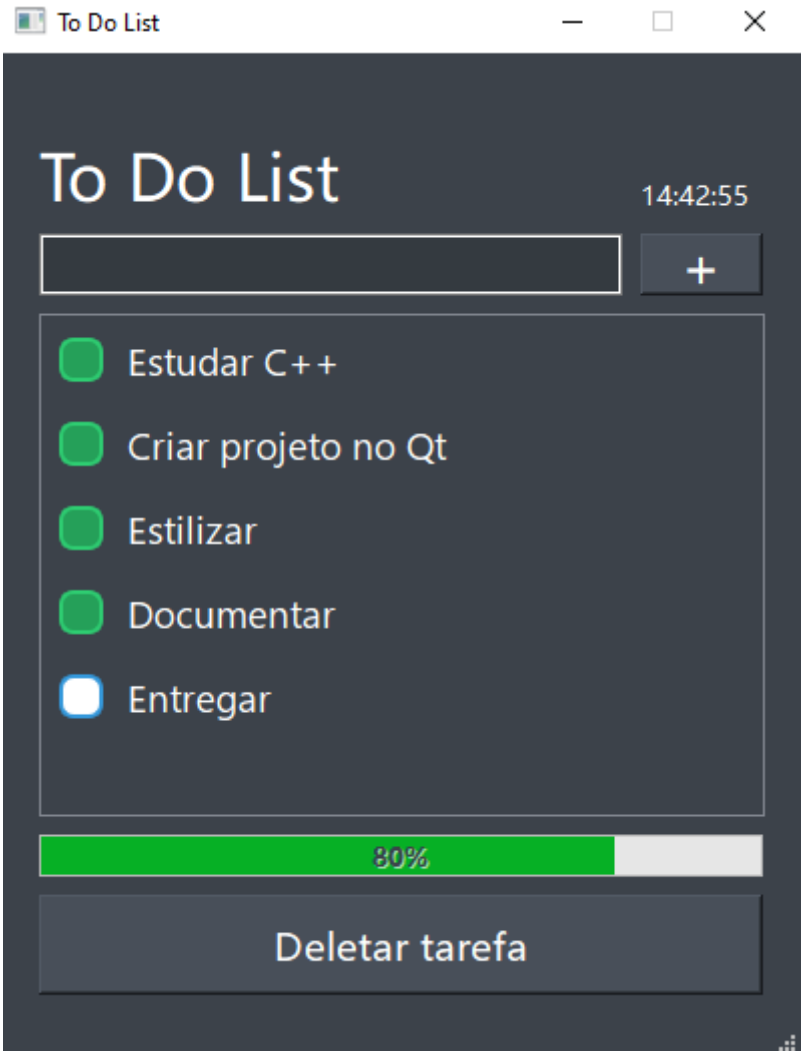

Figura 11 – Programa em execução (Autor, 2025)

5 CONCLUSÃO

O projeto da aplicação To-Do List foi concluído com sucesso, alcançando os objetivos propostos. Através do desenvolvimento, foi possível aplicar conceitos importantes de programação orientada a objetos e criar uma interface gráfica funcional com Qt.

6 REFERÊNCIAS

A. ONLINE:

ProgrammingKnowledge. Qt Tutorials For Beginners 4 - First Qt GUI widget Application. [S.l.]: YouTube, 2016. Disponível em: https://www.youtube.com/watch?v=Y1c-ieVO-UY&list=PLS1QulWo1RIZiBcTr5urECberTITj7gjA&index=4. Acesso em: 28 jul. 2025.

ProgrammingKnowledge. Qt Tutorials For Beginners 20 - QListWidget. [S.l.]: YouTube, 2016. Disponível em: https://www.youtube.com/watch?v=2YRAJt-LbkM&list=PLS1QulWo1RIZiBcTr5urECberTITj7gjA&index=22.Acesso em: 28 jul. 2025.

QT Company. Qt Creator Manual. Disponível em: https://doc.qt.io/qtcreator/. Acesso em: 28 jul. 2025.