ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ НЕФТИ И ГАЗА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
ИМЕНИ И.М. ГУБКИНА»

ФАКУЛЬТЕТ КОМПЛЕКСНОЙ БЕЗОПАСНОСТИ ТЭК
КАФЕДРА БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Лабораторная работа №1

по дисциплине «Специализированные языки и технологии программирования»

на тему «Основы работы с виджетами в Qt»

Выполнил студент:
группы КА-22-06
Воронин Алексей Дмитриевич

Преподаватель:
Греков Владимир Сергеевич

Москва, 2025

Оглавление

Цель работы:

- Изучить основные виджеты, доступные в Qt.
- Научиться разрабатывать графический интерфейс пользователя (GUI) с использованием виджетов.
- Разработать простое приложение с GUI, используя виджеты Qt.

Задание:

Создать приложение "Персональный органайзер", в котором пользователь сможет добавлять, просматривать и удалять записи о своих задачах. Приложение должно содержать следующие элементы управления: - Список задач (QListWidget или QTreeView). - Кнопки для добавления, удаления и просмотра деталей задач. - Форма для добавления новой задачи с полями: - Название задачи (QLineEdit) - Описание (QTextEdit) - Дата выполнения (QDateEdit)

Часть 1 — Разработка интерфейса

1. Запустите Qt Creator и создайте новый проект "Приложение Qt Widgets".

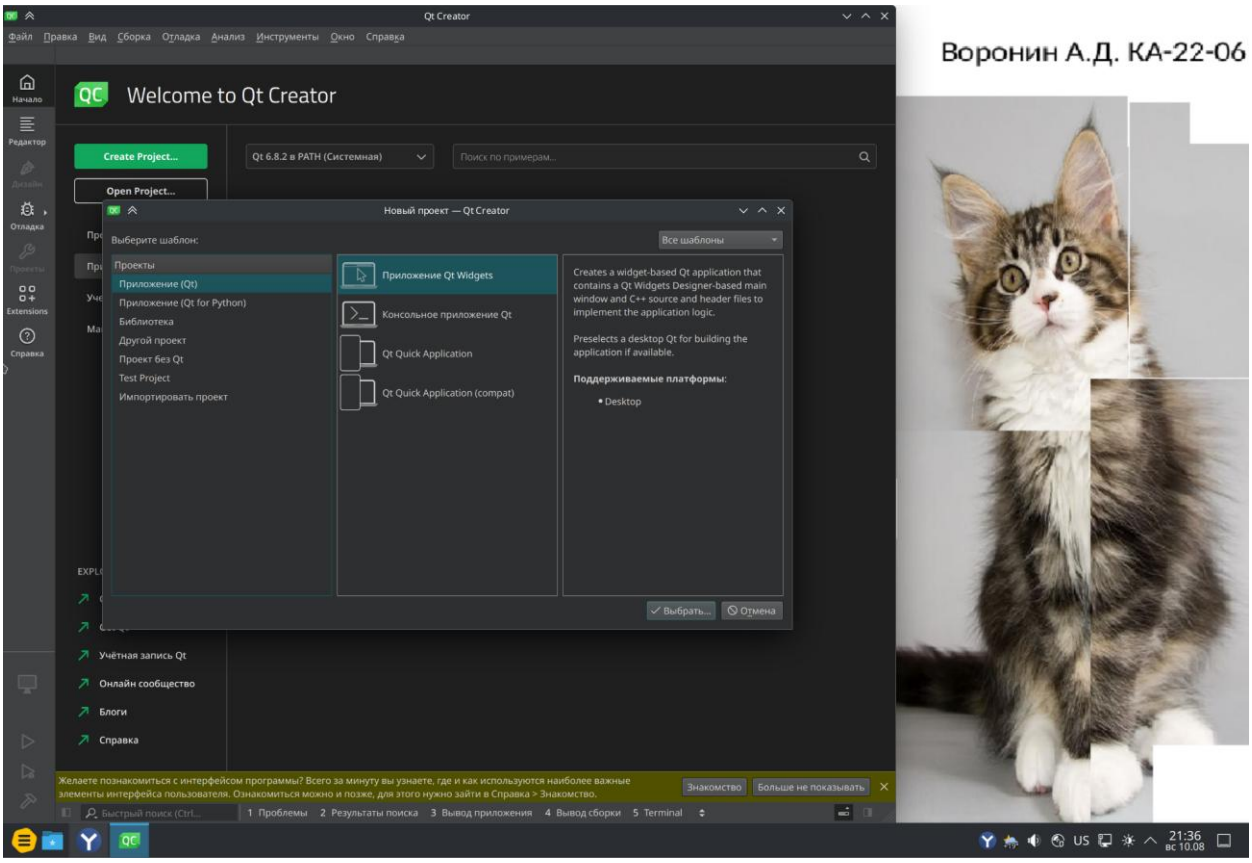

Рисунок 1 - Создание проекта

В мастере проектов укажите название проекта "PersonalOrganizer" и выберите директорию для проекта.

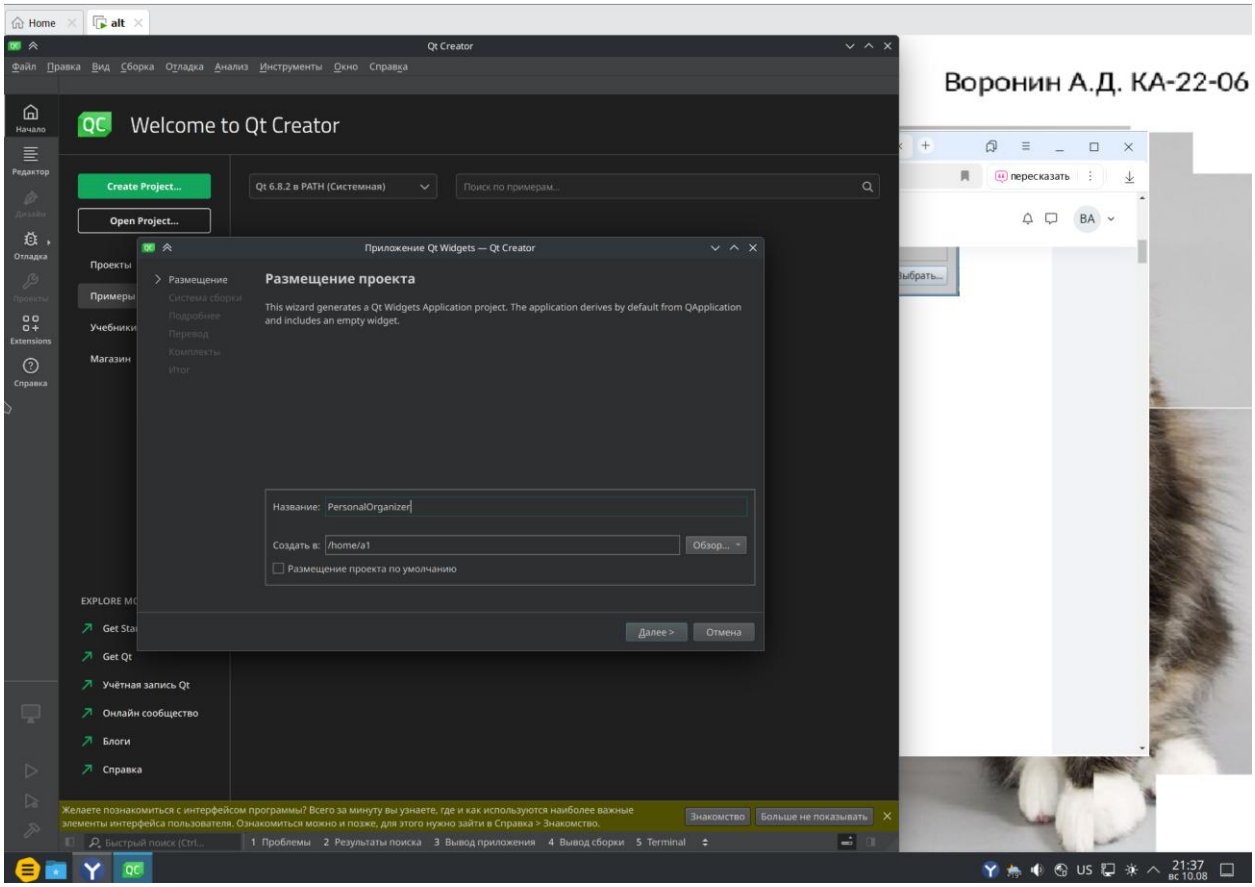

Рисунок 2 - Создание проекта

Выберите систему сборки CMake.

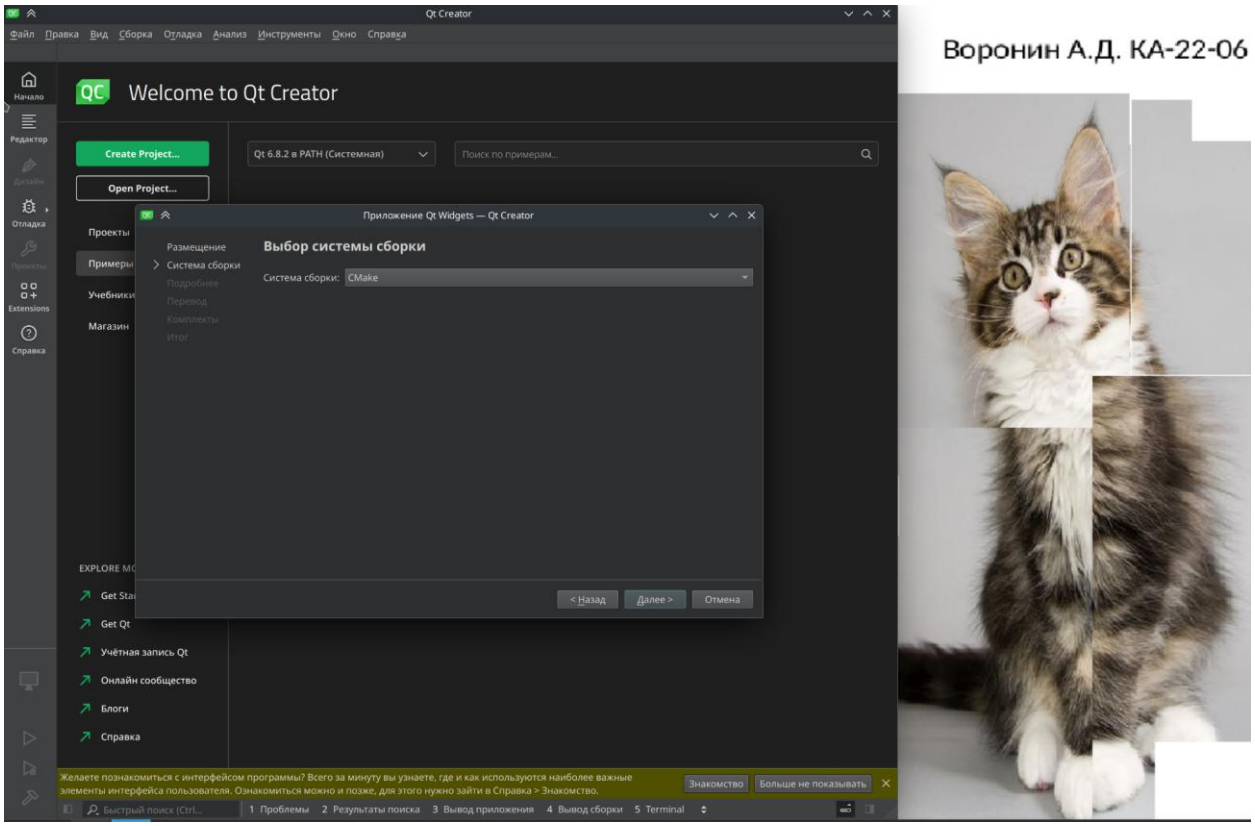

Рисунок 3 - Создание проекта

Заполните информацию о классах, данные оставьте по умолчанию.

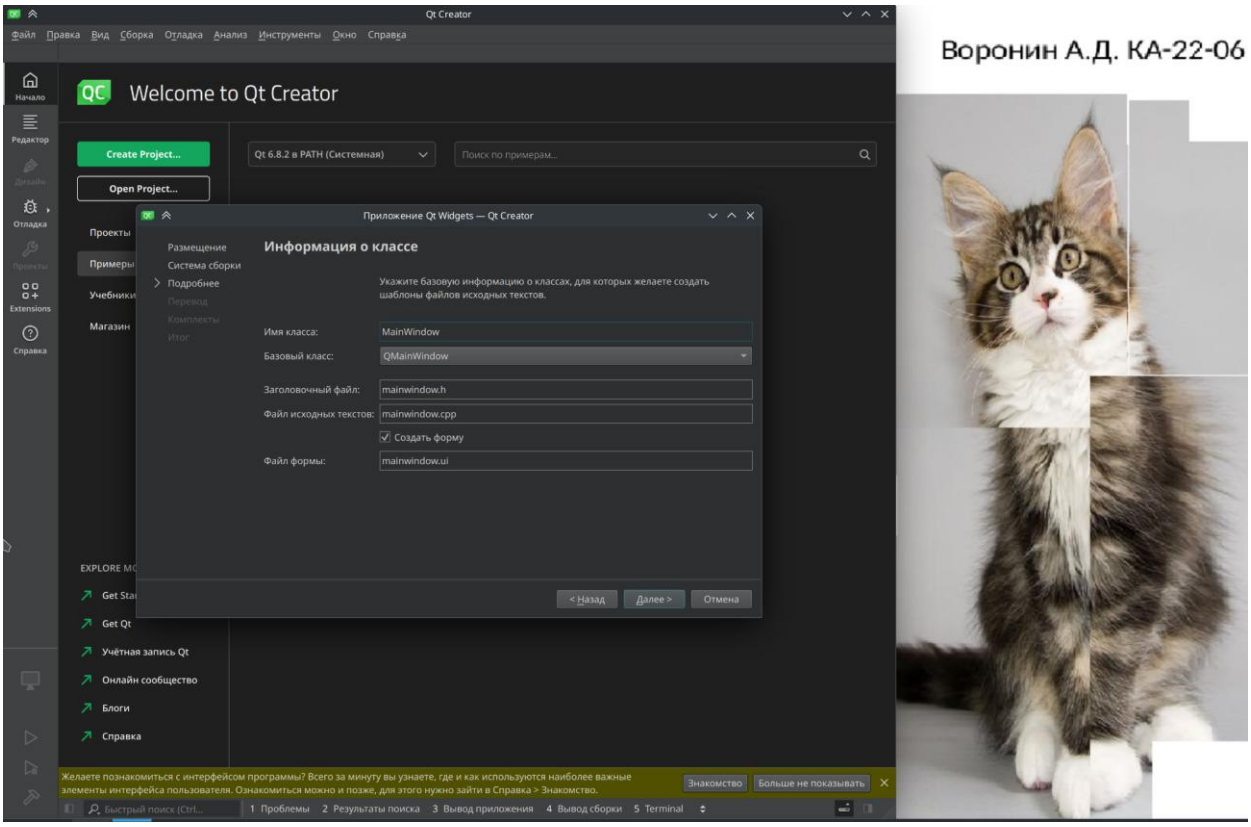

Рисунок 4 - Создание проекта

Файл перевода можно оставить по умолчанию пустым.

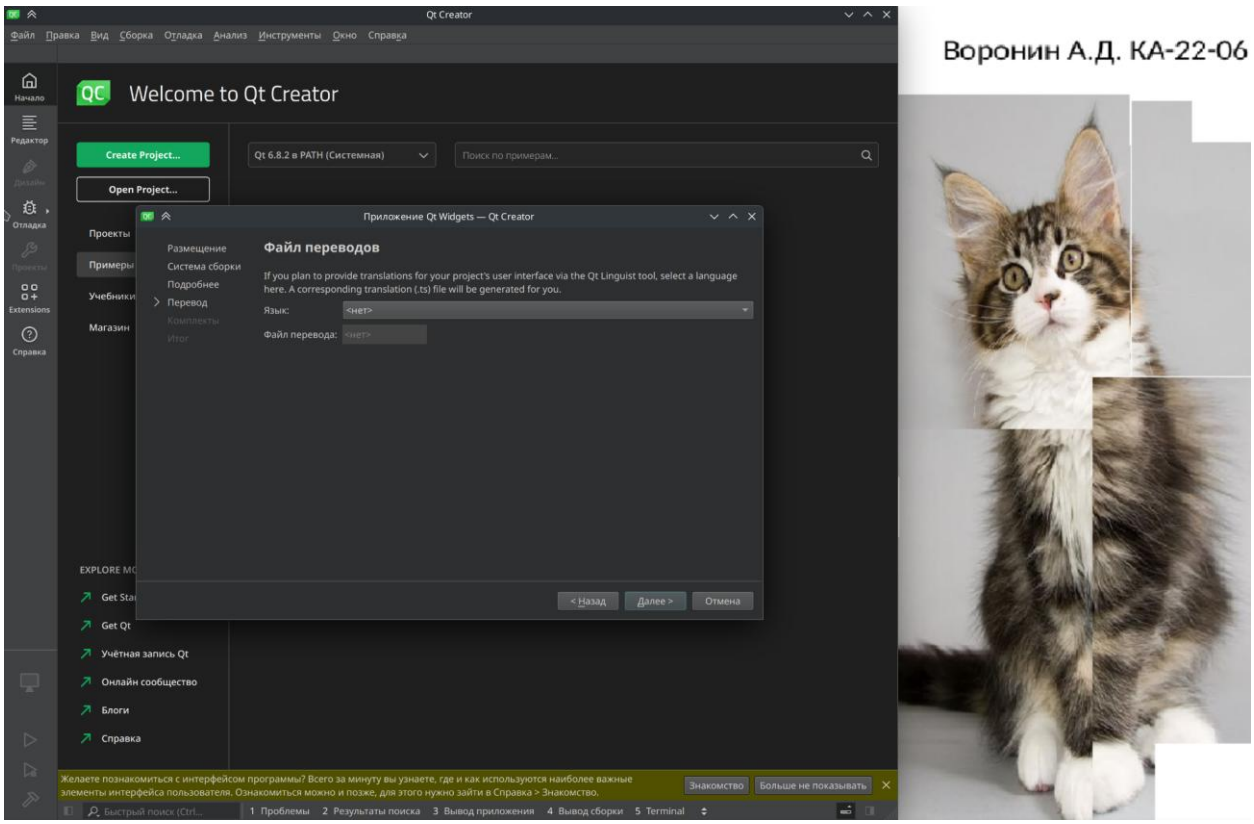

Рисунок 5 - Создание проекта

Выберите комплект для сборки (он должен быть автоматически определён на основе установленной версии Qt).

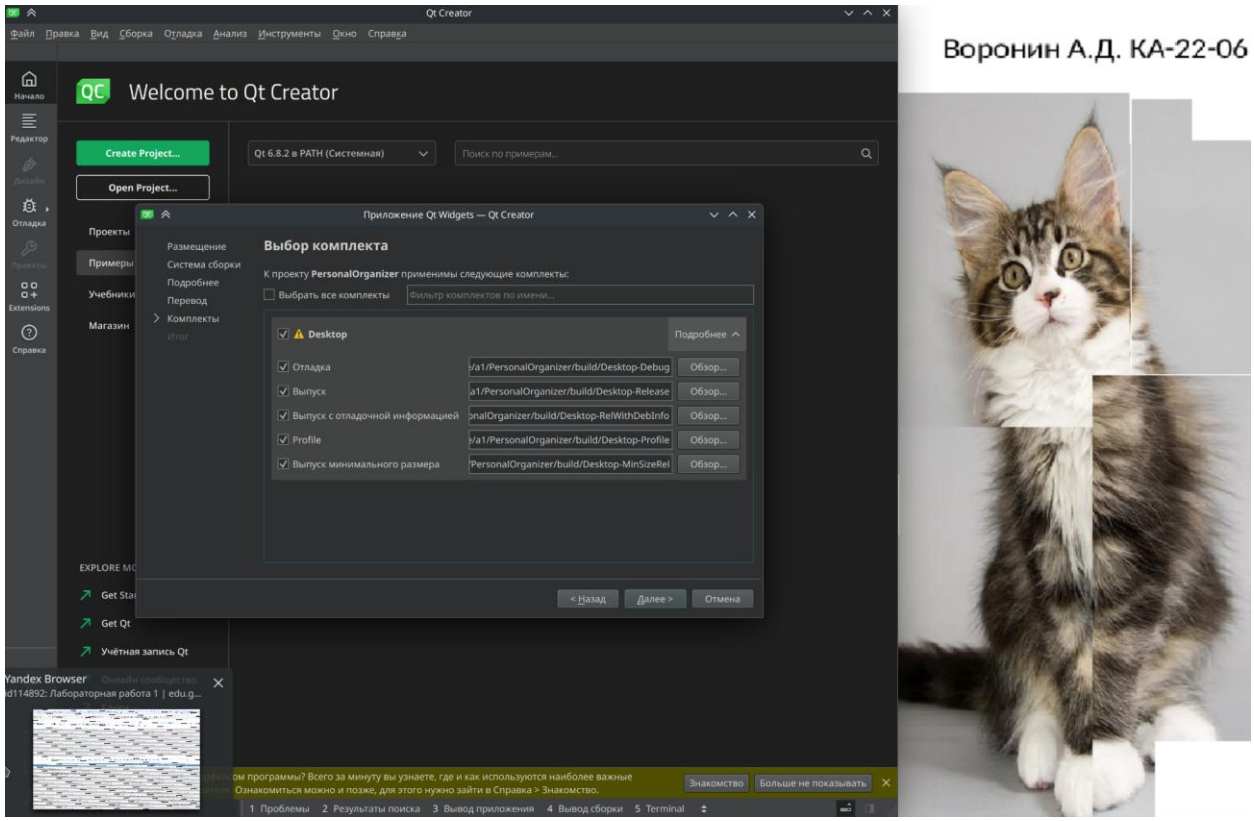

Рисунок 6 - Создание проекта

Завершите создание проекта, нажав кнопку "Завершить".

Шаг 2. Дизайн интерфейса.

Откройте файл mainwindow.ui в редакторе форм Qt Creator.

Используя визуальный редактор, разместите на главном окне следующие виджеты:

1. QListWidget для отображения списка задач.

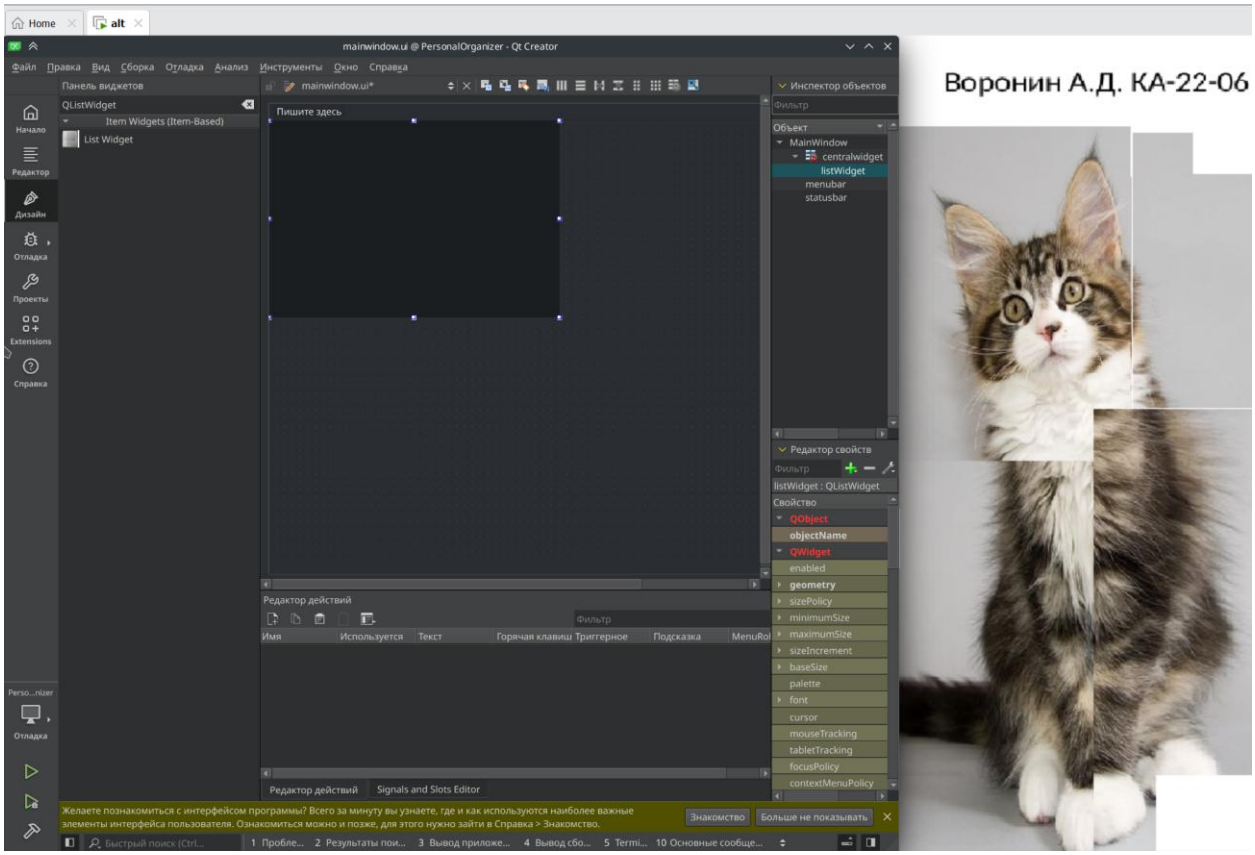

Рисунок 7 – Вставка виджета QListWidget

Так как изначально цвет виджета и окна будет серым (что не очень приятно для просмотра итогового приложения), его следует изменить:

Нажмите по окну MainWindow и перейдите в редактор свойств (расположен в правой части Qt Creator).

Найдите свойство styleSheet, которое позволяет задавать стили для графических элементов (аналог CSS в веб-разработке).

Для установки фона используйте строку:

background-color: white;

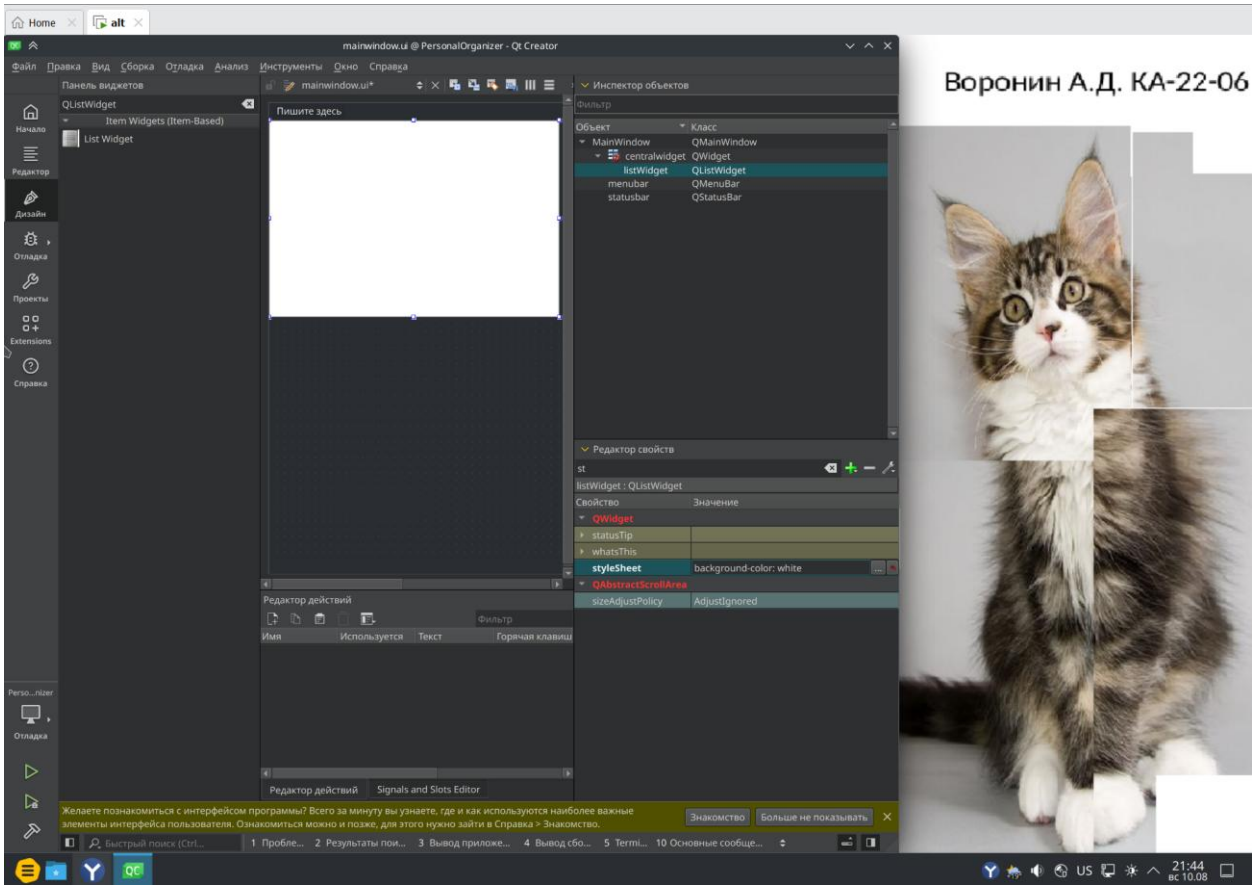

Рисунок 8 – Смена цвета

2. QPushButton для кнопок:

"Добавить задачу"

"Удалить задачу"

"Подробности задачи"

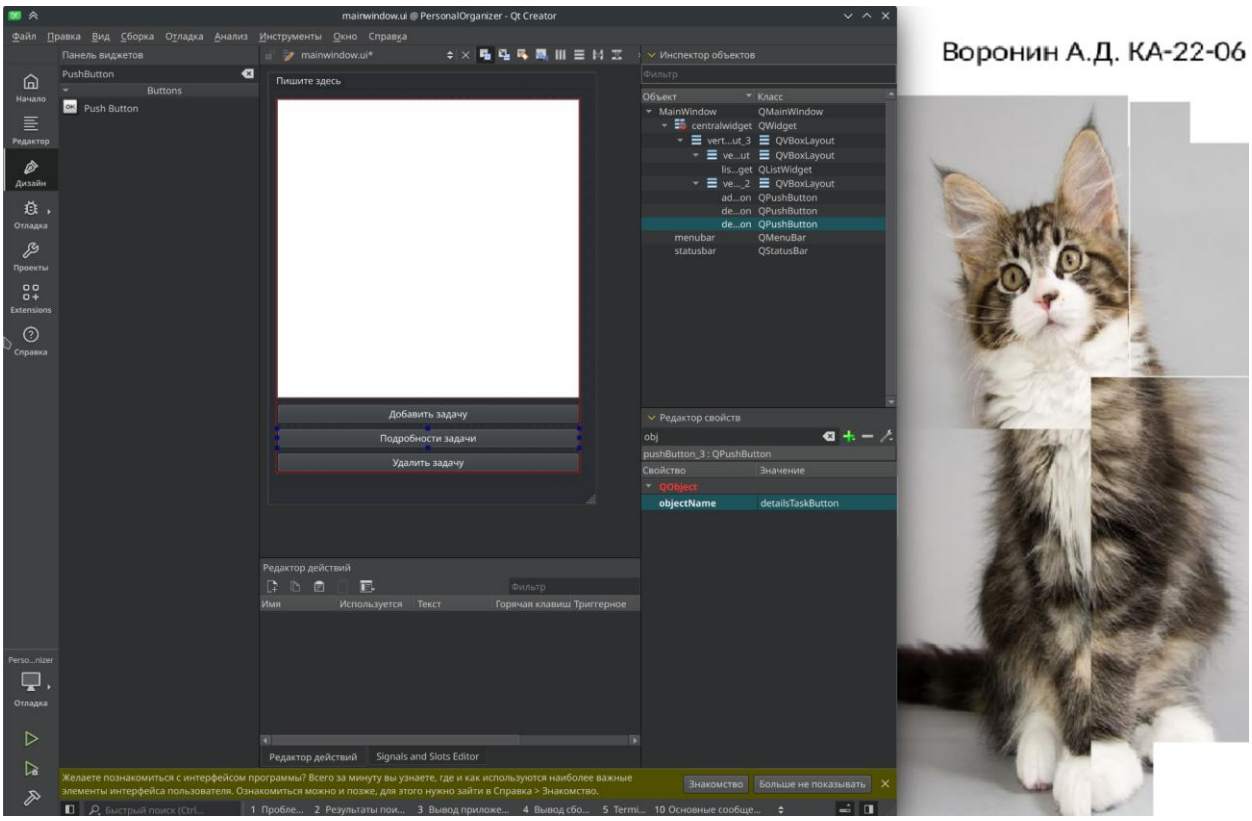

Рисунок 9 – Добавление кнопок для приложения

Шаг 3. Настройка виджетов.

Выберите QListWidget и в редакторе свойств измените значение objectName, например, на tasksListWidget.

Аналогично настройте имена для кнопок:

- addTaskButton
- deleteTaskButton
- detailsTaskButton

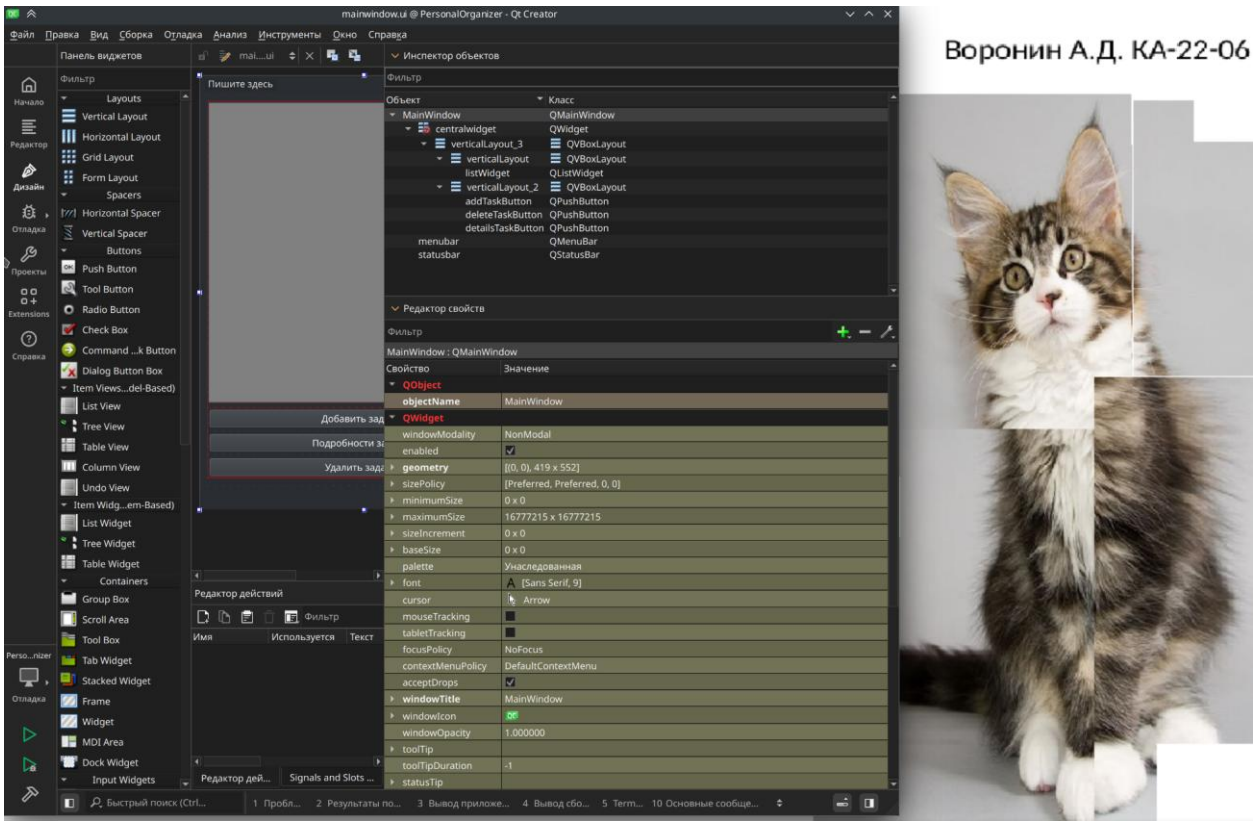

Рисунок 10 – Имена кнопок

Часть 2 — Реализация логики

Шаг 1. Обработка событий кнопок.

Создайте слоты для обработки нажатий на кнопки. Перейдите в mainwindow.ui и нажмите правой кнопкой мыши на любую из добавленных кнопок. Выберите опцию “Перейти к слоту” и затем обычный сигнал clicked(). После этого слот автоматически создастся в mainwindow.cpp.

Повторите для остальных кнопок.

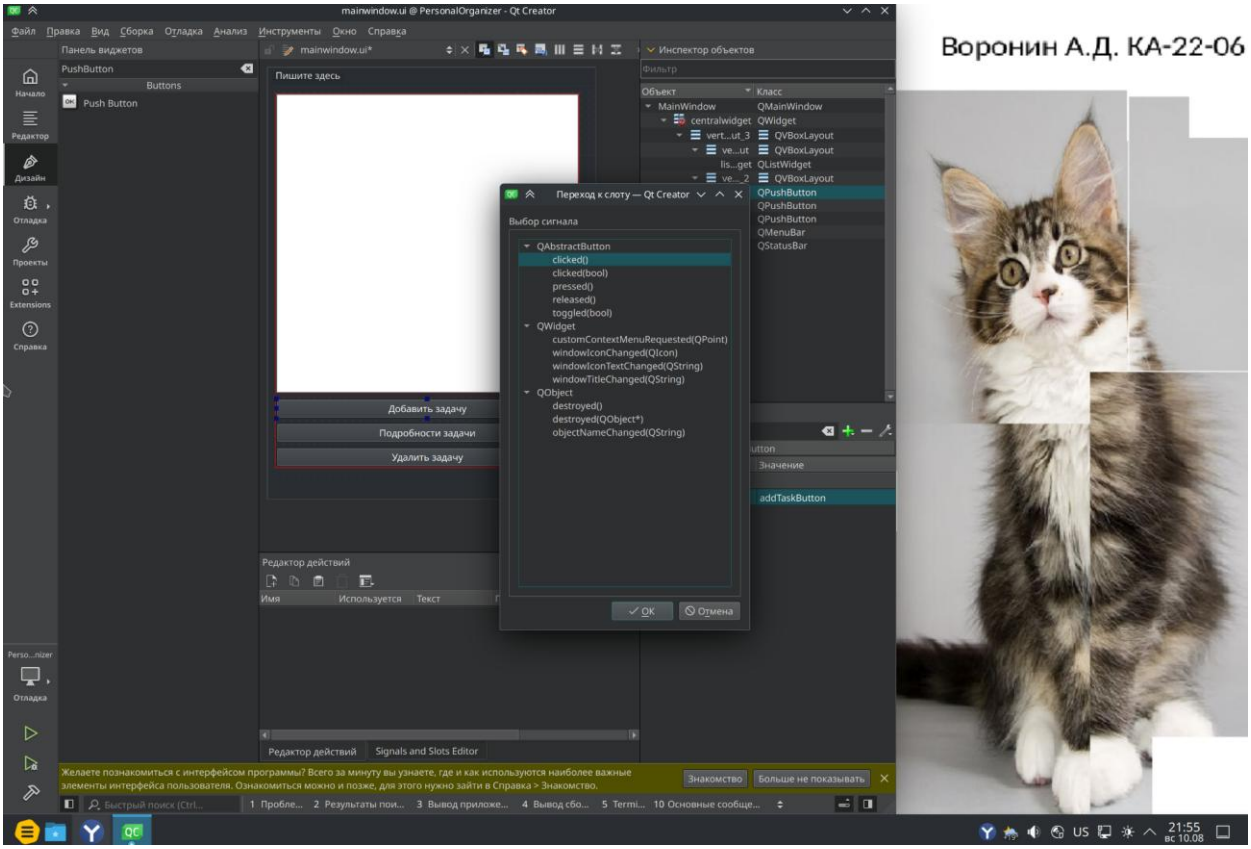

Рисунок 11 – Создание слотов для обработки нажатий

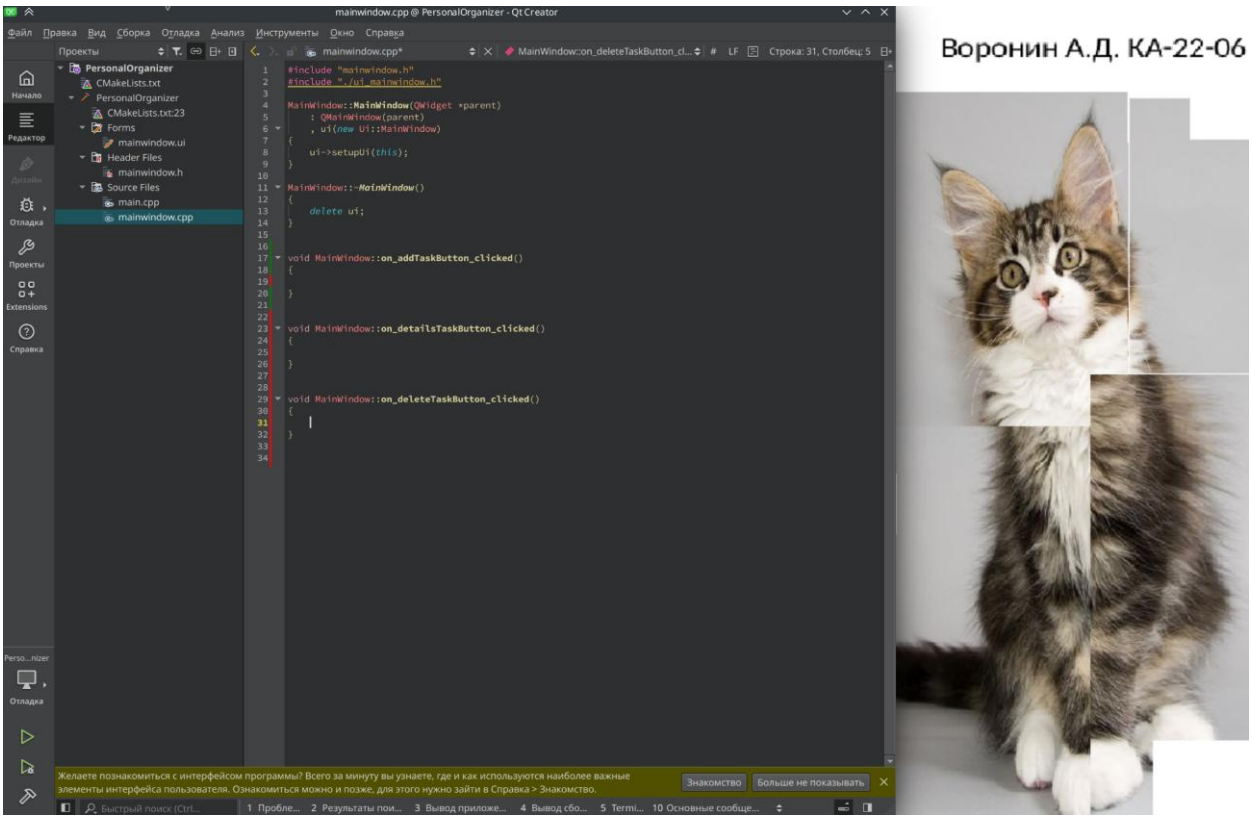

Рисунок 12 – Добавление кнопок для приложения

В конструкторе класса соедините сигнал кнопки со слотом с помощью команды:

connect(ui->addTaskButton, &QPushButton::clicked, this, &MainWindow::addButton)

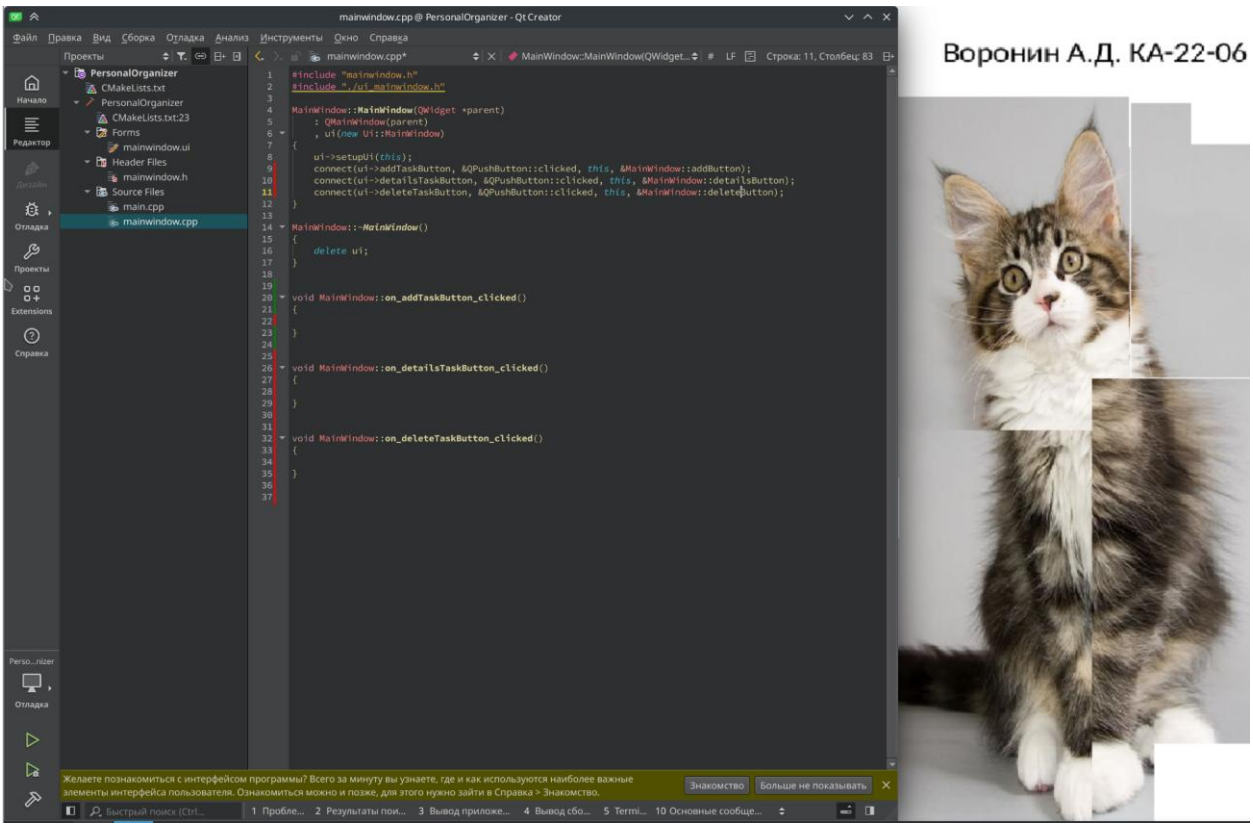

Рисунок 13 – Соединение сигнала

Реализуйте открытие диалогового окна с формой для ввода данных о новой задаче. При добавлении задачи должно открываться диалоговое окно с полями:

```
#include <QDialog>
#include <QVBoxLayout>
#include <QLineEdit>
#include <QTextEdit>
#include <QDateEdit>
#include <QPushButton>
#include <QMessageBox>

void MainWindow::addButton()
{
   QDialog dialog(this);
   dialog.setWindowTitle("Добавить задачу");
   QVBoxLayout layout(&dialog);

   QLineEdit titleEdit;
   titleEdit.setPlaceholderText("Название задачи");
   layout.addWidget(&titleEdit);

   QTextEdit descriptionEdit;
   descriptionEdit.setPlaceholderText("Описание задачи");
   layout.addWidget(&descriptionEdit);

   QDateEdit dateEdit;
   dateEdit.setCalendarPopup(true);
   layout.addWidget(&dateEdit);
```

```
    QPushButton addButton("Добавить");
    layout.addWidget(&addButton);

    connect(&addButton, &QPushButton::clicked, [&]() {
        QString title = titleEdit.text().trimmed();
        QString description = descriptionEdit.toPlainText().trimmed();
        QDate date = dateEdit.date();
        QListWidgetItem *newItem = new QListWidgetItem(title);
        newItem->setData(Qt::UserRole, description);
        newItem->setData(Qt::UserRole + 1, date);
        ui->tasksListWidget->addItem(newItem);
        dialog.accept();
    });

    dialog.exec();
}
```

Реализуйте удаление выбранной задачи из списка:

```
void MainWindow::deleteButton()
{
    QListWidgetItem *item = ui->tasksListWidget->currentItem();
    if (item) {
        QMessageBox msgBox(this);
        msgBox.setWindowTitle("Удаление задачи");
        msgBox.setText("Вы уверены, что хотите удалить задачу?");
        QPushButton* yesButton = msgBox.addButton(QMessageBox::Yes);
        QPushButton* noButton = msgBox.addButton(QMessageBox::No);
        yesButton->setText("Да");
        noButton->setText("Нет");
        msgBox.exec();
        if (msgBox.clickedButton() == yesButton) {
            delete item;
        }
    } else {
        QMessageBox::warning(this, "Удаление задачи", "Выберите задачу для
удаления.");
    }
}
```

Реализуйте отображение информации о выбранной задаче, например, с помощью QMessageBox:

```
void MainWindow::showDetailsButton()
{
    QListWidgetItem *item = ui->tasksListWidget->currentItem();
    if (item) {
        QString taskDetails = item->text();
        QString description = item->data(Qt::UserRole).toString();
        QDate taskDate = item->data(Qt::UserRole + 1).toDate();
        QString details = QString("<b>Задача:</b> %1\n<b>Описание:</b> %2\n<b>Дата
выполнения:</b> %3")
                          .arg(taskDetails)
                          .arg(description)
                          .arg(taskDate.toString("dd.MM.yyyy"));
        QMessageBox::information(this, "Детали задачи", details);
    } else {
        QMessageBox::warning(this, "Просмотр задачи", "Выберите задачу для
просмотра.");
```

```
    }
}
```

Шаг 2. Тестирование приложения.

Соберите и запустите приложение.

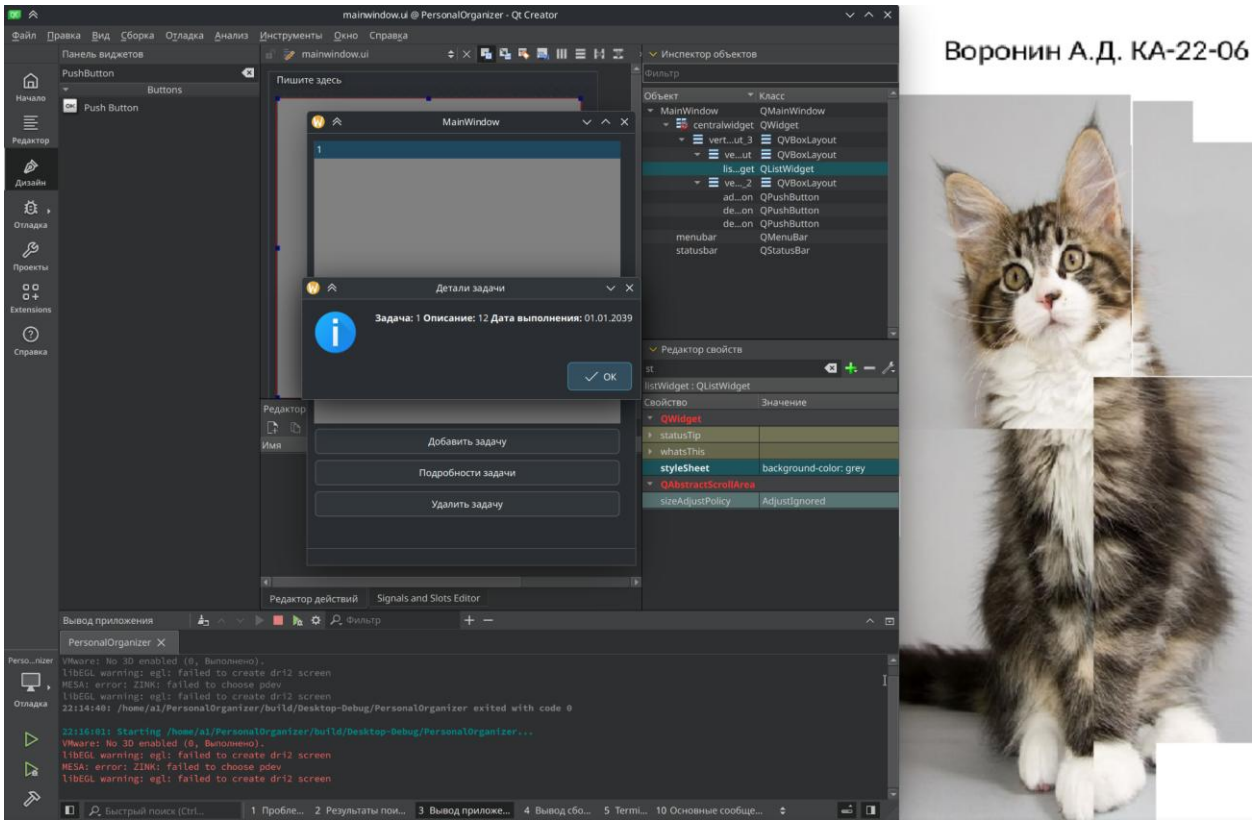

Рисунок 14 – Тестирование

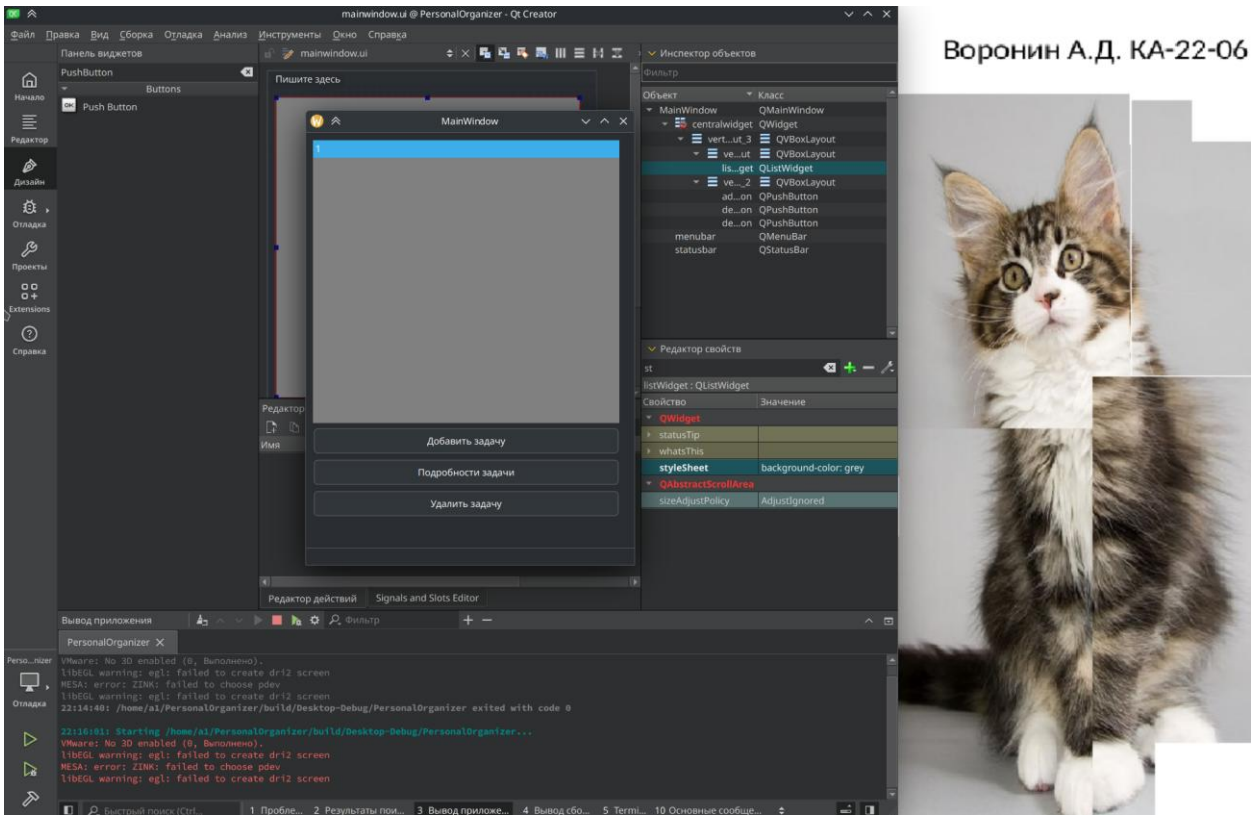

Рисунок 15 – Тестирование

Задание для самостоятельной работы

Добавьте валидацию данных формы добавления задачи (например, проверку на пустое название).

Добавим проверку на пустое название:

```
if (title.isEmpty()) {
    QMessageBox::warning(&dialog, "Ошибка ввода", "Введите название задачи!");
    return;
}
if (description.isEmpty()) {
    QMessageBox::warning(&dialog, "Ошибка ввода", "Введите описание задачи!");
    return;
}
```

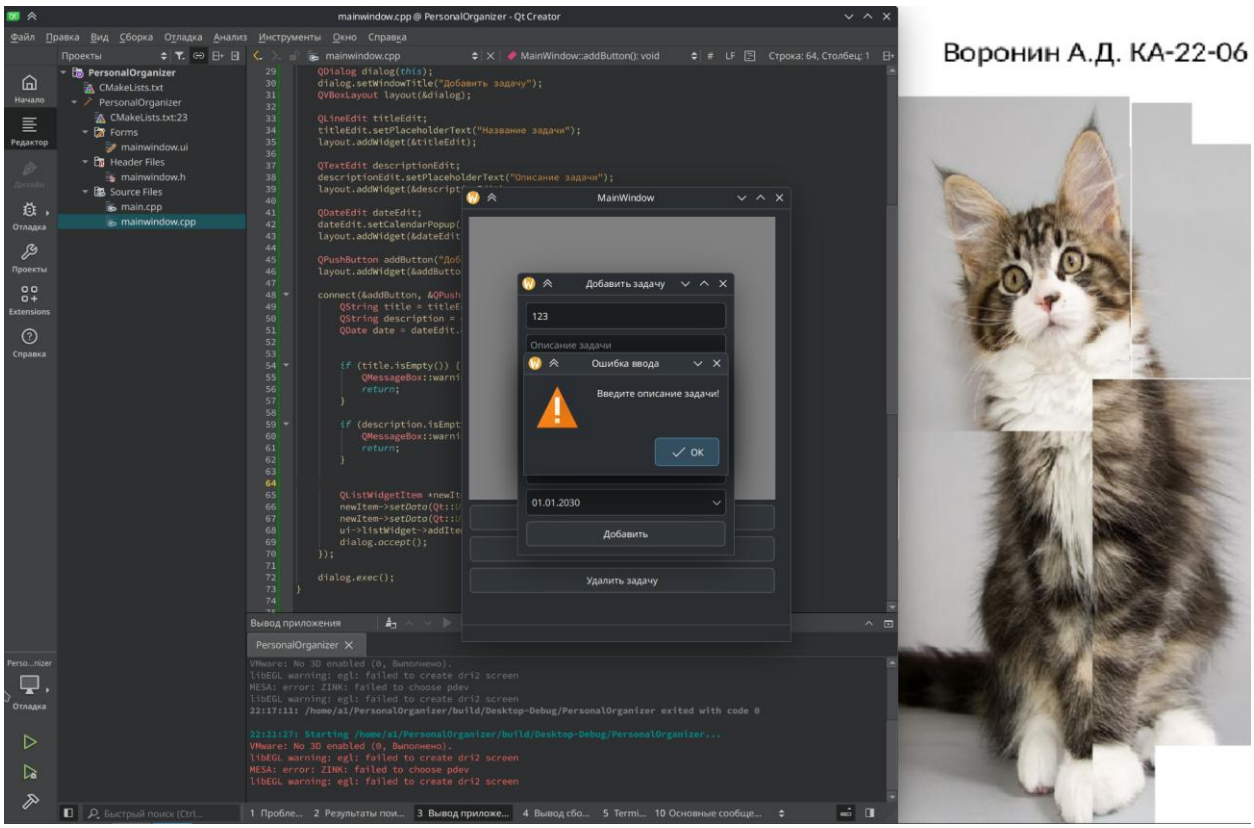

Рисунок 16 – Проверка на пустой ввод

Добавим сохранение списка между сеансами.

Подключим QSettings

```
#include <QSettings>
```

Загружаем задачи при запуске:

В конструкторе, после ui->setupUi(this);:

```
loadTasks();
```

Сохраняем задачи при закрытии окна:

Переопределяем closeEvent:

```
void MainWindow::closeEvent(QCloseEvent *event)
{
    saveTasks();
    QMainWindow::closeEvent(event);
}
```

Реализация saveTasks():

```
void MainWindow::saveTasks()
{
    QSettings settings("MyCompany", "MyTodoApp");
    settings.beginWriteArray("tasks");

    for (int i = 0; i < ui->listWidget->count(); ++i) {
        settings.setArrayIndex(i);
        QListWidgetItem *item = ui->listWidget->item(i);
        settings.setValue("title", item->text());
        settings.setValue("description", item->data(Qt::UserRole).toString());
        settings.setValue("date", item->data(Qt::UserRole + 1).toDate());
    }
```

```
    settings.endArray();
}
```

Реализация loadTasks():

```
void MainWindow::loadTasks()
{
    QSettings settings("MyCompany", "MyTodoApp");
    int size = settings.beginReadArray("tasks");

    for (int i = 0; i < size; ++i) {
        settings.setArrayIndex(i);
        QString title = settings.value("title").toString();
        QString description = settings.value("description").toString();
        QDate date = settings.value("date").toDate();

        QListWidgetItem *item = new QListWidgetItem(title);
        item->setData(Qt::UserRole, description);
        item->setData(Qt::UserRole + 1, date);
        ui->listWidget->addItem(item);
    }

    settings.endArray();
}
```

Таким образом после добавления задачи, закрытия приложения и повторном его открытии добавленная ранее задача остается в списке задач.

ЗАКЛЮЧЕНИЕ

В заключение, подводя итог проделанной работе, все поставленные в начале цели и задания данной лабораторной работы были достигнуты в полном объеме.

Контрольные вопросы

1. Как добавить виджеты на форму в Qt Creator?

В файле .ui в панели инструментов выбрать нужный виджет и перетащить на форму.

2. Как создать слот для обработки нажатия на кнопку?

Открываем файл .ui, затем выбираем нужную кнопку, нажимаем ПКМ на нее → Перейти к слотам→clicked().

3. Как можно реализовать сохранение данных между сессиями приложения в Qt.

- QSettings
- Использование файлов (JSON, XML и т.п.)
- База данных SQLite