Файл main.cpp:

```
#include <QApplication>
#include <QPushButton>
#include "ui/mainwindow/mainwindow.h"
int main(int argc, char *argv[]) {
  QApplication a(argc, argv);
  MainWindow w;
  QIcon appIcon(":icon_app/image/icon2.png");
  a.setWindowIcon(appIcon);
  w.show();
  return a.exec();
}
```

Файл mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include "../secondscreen/secondscreen.h"
#include<QStackedWidget>
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
class MainWindow : public QMainWindow {
Q_OBJECT
public:
  MainWindow(QWidget *parent = nullptr);
  ~MainWindow();
private slots:
  void on_pushButton_clicked();
private:
  Ui::MainWindow *ui;
  SecondScreen *secondScreen;
  QStackedWidget *stackedWidget;
};
#endif // MAINWINDOW_H
```

Файл mainwindow.cpp:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::MainWindow) {
  ui->setupUi(this);
  QString new_title = "ASCII-ART";
  setWindowTitle(new_title);
  QLinearGradient gradient(0, 0, width(), height());
```

```
  gradient.setColorAt(1, QColor(255, 236, 210, 215));
  gradient.setColorAt(0, QColor(252, 182, 159, 200));
  QBrush brush(gradient);
  QPalette palette = this->palette();
  palette.setBrush(QPalette::Window, brush);
  this->setPalette(palette);
  stackedWidget = new QStackedWidget(this);
  stackedWidget->addWidget(ui->centralwidget);
  secondScreen = new SecondScreen(this);
  stackedWidget->addWidget(secondScreen);
  setCentralWidget(stackedWidget); {
void MainWindow::on_pushButton_clicked() {
  stackedWidget->setCurrentWidget(secondScreen);
}
MainWindow::~MainWindow() {
  delete ui;
}
```

Файл secondscreen.h:

```
#ifndef TERM_QT_SECONDSCREEN_H
#define TERM_QT_SECONDSCREEN_H
#include <QWidget>
#include<QStackedWidget>
#include <QGraphicsScene>
#include <QGraphicsPixmapItem>
#include <QFileDialog>
#include <QDir>
#include <QDebug>
#include <QListWidget>
QT_BEGIN_NAMESPACE
namespace Ui { class SecondScreen; }
QT_END_NAMESPACE
class SecondScreen : public QWidget {
Q_OBJECT
public:
  explicit SecondScreen(QWidget *parent = nullptr);
  ~SecondScreen() override;
public slots:
  void remove_item(const QString& text);
private slots:
  void on_list_item_clicked(QListWidgetItem *item);
  void on_pushButton_clicked();
  void on_pushButton_2_clicked();
private:
  bool is_image(const QString &filePath);
  void load_image(const QString &file_path);
  Ui::SecondScreen *ui;
  QGraphicsScene *scene;
```

```
};
#endif //TERM_QT_SECONDSCREEN_H
```

Файл secondscreen.cpp:

```
#include "secondscreen.h"
#include "ui_secondscreen.h"
#include "src_back/data_model/get_bmp_data/BMPImageProcess.h"
#include "src_back/data/header/bmp_image.h"
#include "src_back/data/header/ppm_image.h"
#include "../customwidget/customwidget.h"
#include "src_back/data/header/other_image.h"
#include <QImageReader>
#include <QFileDialog>
#include <QDesktopServices>
#include <QMessageBox>
SecondScreen::SecondScreen(QWidget *parent) :
    QWidget(parent), ui(new Ui::SecondScreen) {
  ui->setupUi(this);
  QLinearGradient gradient(0, 0, width(), height());
  gradient.setColorAt(1, QColor(255, 236, 210, 215));
  gradient.setColorAt(0, QColor(252, 182, 159, 200));
  QBrush brush(gradient);
  QPalette palette = this->palette();
  palette.setBrush(QPalette::Window, brush);
  this->setPalette(palette);
  ui->listWidget->setMinimumSize(400, 600);
  ui->graphicsView->setMinimumSize(600, 600);
        connect(ui->listWidget,    &QListWidget::itemClicked,    this,
&SecondScreen::on_list_item_clicked);
}
SecondScreen::~SecondScreen() {
  delete ui;
}
void SecondScreen::load_image(const QString &file_path) {
  QPixmap pixmap(file_path);
  if (!pixmap.isNull()) {
    scene = new QGraphicsScene;
    scene->clear();
                  QGraphicsPixmapItem     *graphicsItem     =     new
QGraphicsPixmapItem(pixmap);
    ui->graphicsView->setScene(scene);
    scene->addItem(graphicsItem);
            ui->graphicsView->fitInView(graphicsItem->boundingRect(),
Qt::KeepAspectRatio);
             qDebug()   <<   "Opened   file:   "   <<   file_path;
 }
}
```

```
void SecondScreen::on_pushButton_clicked() {
   QString file_path = QFileDialog::getOpenFileName(this, "Выберите
файл", QDir::homePath(),
                                    "Изображения(*.png *.jpg *.jpeg *.bmp
*.gif *.ppm)");
  if (!file_path.isEmpty()) {
    QImageReader imageReader(file_path);
    imageReader.setFormat("PPM");
    QImage qImage=imageReader.read();
    if (!qImage.isNull()) {
      for (int i = 0; i < ui->listWidget->count(); ++i) {
        QListWidgetItem *existingItem = ui->listWidget->item(i);
                        QString   existingFilePath   =   existingItem-
>data(Qt::UserRole).toString();
        if (existingFilePath == file_path) {
          QMessageBox::information(this, "Предупреждение", "Этот файл
уже добавлен.");
          return;
 }
}
      CustomWidget *customWidget = new CustomWidget(this);
      customWidget->set_text(QFileInfo(file_path).fileName());
      QListWidgetItem *item_list = new QListWidgetItem();
      item_list->setSizeHint(QSize(40, 40));
      item_list->setIcon(QIcon(QPixmap(file_path)));
      item_list->setData(Qt::UserRole, file_path);
      ui->listWidget->addItem(item_list);
      ui->listWidget->setItemWidget(item_list, customWidget);
      load_image(file_path);
    } else {
           QMessageBox::warning(this,  "Ошибка",  "Выбранный  файл  не
является изображением.");
  }
 }
}
void SecondScreen::on_list_item_clicked(QListWidgetItem *item) {
  QString filePath = item->data(Qt::UserRole).toString();
  if (!filePath.isEmpty()) {
    load_image(filePath);
 }
}
void SecondScreen::on_pushButton_2_clicked() {
  QListWidgetItem *selected_item = ui->listWidget->currentItem();
  if (selected_item) {
    int view_width = 100;
    int view_height = 100;
    Image *image;
    QString file_path = selected_item->data(Qt::UserRole).toString();
    std::vector<std::vector<Pixel>> pixels;
```

```
    if (!is_image(file_path)) {
      OtherImage otherImage;
      pixels=otherImage.read(file_path);
      otherImage.set_pixels(pixels);
      image = dynamic_cast<Image *>(new OtherImage(otherImage));
    } else if (file_path.toLower().endsWith(".bmp")) {
      BMPImageProcess bmpImageProcess;
      bmpImageProcess.read_image(file_path.toStdString());
      pixels = bmpImageProcess.get_pixels();
      BMPImage bmpImage;
      bmpImage.set_width(bmpImageProcess.get_width());
      bmpImage.set_height(bmpImageProcess.get_height());
      bmpImage.set_pixels(pixels);
      image = dynamic_cast<Image *>(new BMPImage(bmpImage));
    } else {
      PPMImage ppmImage;
      ppmImage.read_image(file_path.toStdString());
      image = dynamic_cast<Image *>(new PPMImage(ppmImage));
    }
    pixels = Convert::resized_image(image, view_height, view_width);
    image->set_pixels(pixels);
    Convert::grayscale(image);
    Convert::ascii(image, "test.txt");
       QString txt_file_path = "/home/egerin/Projects/term_qt/cmake-
build-debug/";
    QUrl fileUrl = QUrl::fromLocalFile(txt_file_path + "test.txt");
    QDesktopServices::openUrl(fileUrl);
    delete image;
     }
    }
void SecondScreen::remove_item(const QString &text) {
  for (int i = 0; i < ui->listWidget->count(); ++i) {
    auto item = ui->listWidget->item(i);
      auto itemWidget = qobject_cast<CustomWidget *>(ui->listWidget-
>itemWidget(item));
    if (itemWidget && itemWidget->get_text() == text) {
      delete itemWidget;
      delete ui->listWidget->takeItem(i);
                                                    break;
 }
}
  for (QGraphicsItem *scene_item: ui->graphicsView->scene()->items())
{
       if (scene_item->data(0).toString() == text && scene_item-
>isVisible()) {
      ui->graphicsView->scene()->removeItem(scene_item);
      break;
}
}
```

```
  ui->graphicsView->scene()->clear(); {
bool SecondScreen::is_image(const QString &file_path) {
  return file_path.toLower().endsWith(".bmp") ||
       file_path.toLower().endsWith(".ppm"); }
```

Файл convert.h:

```
#ifndef TERM_WORK_CONVERT_H
#define TERM_WORK_CONVERT_H
#include "src_back/data/header/image.h"
#include <iostream>
#include <ui_secondscreen.h>
class Image;
class Pixel;
class Convert {
public:
  static void ascii(const Image *image,const std::string& file_path);
  static void grayscale(Image *image);
   static std::vector<std::vector<Pixel>> resized_image(Image *image,
int new_height, int new_width);
private:
   static float map(float value, float start1, float stop1, float
start2, float stop2);
};
#endif //TERM_WORK_CONVERT_H
```

Файл convert.cpp:

```
#include <QString>
#include <QProcess>
#include "convert.h"
#include<QDebug>
#include <QTextCodec>
#include <fstream>
#include <cmath>
void Convert::ascii(const Image *image, const std::string &file_path)
{
            const      std::string      ASCII_LIST      =      "$@B
%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/|()1{}[]?-_+~<>i!lI;:,\"^`'. ";
  std::vector<std::vector<Pixel>> pixels = image->get_pixels();
  std::ofstream output_file(file_path);
  if (output_file.is_open()) {
    for (int y = 0; y < image->get_height(); y++) {
      for (int x = 0; x < image->get_width(); x++) {
                                          int     ascii_index     =
static_cast<int>(map(static_cast<float>(pixels[y][x].m_red),  0,  255,
0,
                              static_cast<float>(ASCII_LIST.length())
- 1));
```

```
        char ascii_char = ASCII_LIST[ascii_index];
        output_file << ascii_char;
}
      output_file << std::endl;
}
  } else {
    std::cerr << "Unable to open the output file." << std::endl; {
  output_file.close(); {
float Convert::map(float value, float start1, float stop1, float
start2, float stop2) {
  return ((value - start1) / (stop1 - start1)) * (stop2 - start2) +
start2;
}
void Convert::grayscale(Image *image) {
  std::vector<std::vector<Pixel>> pixels = image->get_pixels();
          std::vector<std::vector<Pixel>>     grayscale_pixels(image-
>get_height(), std::vector<Pixel>(image->get_width()));
  for (int i = 0; i < image->get_height(); i++) {
    for (int j = 0; j < image->get_width(); j++) {
       auto gray = (unsigned char) (0.3 * (int) pixels[i][j].m_red +
0.12 * (int) pixels[i][j].m_blue +
                    0.58 * (int) pixels[i][j].m_green);
      grayscale_pixels[i][j].m_red = gray;
      grayscale_pixels[i][j].m_green = gray;
      grayscale_pixels[i][j].m_blue = gray;
 }
}
  image->set_pixels(grayscale_pixels); {
std::vector<std::vector<Pixel>> Convert::resized_image(Image *image,
int new_height, int new_width) {
  std::vector<std::vector<Pixel>> pixel = image->get_pixels();
         std::vector<std::vector<Pixel>>    resized_image(new_height,
std::vector<Pixel>(new_width));
  int old_h = image->get_height();
  int old_w = image->get_width();
  image->set_height(new_height);
  image->set_width(new_width);
        float    w_scale_factor    =    static_cast<float>(old_w)    /
static_cast<float>(new_width);
        float    h_scale_factor    =    static_cast<float>(old_h)    /
static_cast<float>(new_height);
  for (int i = 0; i < new_height; i++) {
    for (int j = 0; j < new_width; j++) {
      float x = i * h_scale_factor;
      float y = j * w_scale_factor;
      int x_floor = static_cast<int>(floor(x));
      int x_ceil = std::min(old_h - 1, static_cast<int>(ceil(x)));
      int y_floor = static_cast<int>(floor(y));
      int y_ceil = std::min(old_w - 1, static_cast<int>(ceil(y)));
```

```
      if (x_ceil == x_floor && y_ceil == y_floor) {
        resized_image[i][j] = pixel[x_floor][y_floor];
      } else if (x_ceil == x_floor) {
        Pixel q1 = pixel[x_floor][y_ceil];
        Pixel q2 = pixel[x_floor][y_floor];
        Pixel q;
        q = q1 * (y_ceil - y) + q2 * (y - y_floor);
        resized_image[i][j] = q;
      } else if (y_ceil == y_floor) {
        Pixel q1 = pixel[x_ceil][y_floor];
        Pixel q2 = pixel[x_floor][y_floor];
        Pixel q;
        q = q1 * (x_ceil - x) + q2 * (x - x_floor);
        resized_image[i][j] = q;
      } else {
        Pixel v1 = pixel[x_floor][y_floor];
        Pixel v2 = pixel[x_ceil][y_floor];
        Pixel v3 = pixel[x_floor][y_ceil];
        Pixel v4 = pixel[x_ceil][y_ceil];
        Pixel q1, q2, q;
        q1 = v1 * (x_ceil - x) + v2 * (x - x_floor);
        q2 = v3 * (x_ceil - x) + v4 * (x - x_floor);
        q = q1 * (y_ceil - y) + q2 * (y - y_floor);
        resized_image[i][j] = q;
   }
  }
}
  return resized_image;
}
```

Файл BMPImageProcess.h:

```
#ifndef TERM_WORK_BMPIMAGEPROCESS_H
#define TERM_WORK_BMPIMAGEPROCESS_H
#include <vector>
#include "../../data/header/bmp_header.h"
#include "../../data/header/pixel.h"
#include <iostream>
class BMPImageProcess {
public:
  void read_image(const std::string &file_name);
  std::vector<std::vector<Pixel>> get_pixels() const;
  int get_width() const;
  int get_height() const;
private:
    std::vector<std::vector<Pixel>>  read_data_of_pixels(std::ifstream
&file) const;
  BMPFileHeader m_bmpFileHeader;
  BMPImageHeader m_bmpImageHeader;
```

```
  std::vector<std::vector<Pixel>> m_pixels;
};
#endif //TERM_WORK_BMPIMAGEPROCESS_H
```

Файл BMPImageProcess.cpp:

```
#include "BMPImageProcess.h"
#include <fstream>
std::vector<std::vector<Pixel>>
BMPImageProcess::read_data_of_pixels(std::ifstream &file) const {
  int bytesPP = m_bmpImageHeader.m_bpp / 8;
  int row_size = m_bmpImageHeader.m_width * bytesPP;
  int row_padding = (4 - (row_size % 4)) % 4;
   int  pixel_offset  =  static_cast<int>(m_bmpFileHeader.m_offset)  +
row_padding;
  file.seekg(pixel_offset, std::ios::beg);
                                      std::vector<std::vector<Pixel>>
image_pixels(m_bmpImageHeader.m_height,
                         std::vector<Pixel>(m_bmpImageHeader.m_width)
);
  auto *buffer = new unsigned char[bytesPP];
  for (int i = m_bmpImageHeader.m_height - 1; i >= 0; i--) {
    for (int j = 0; j < m_bmpImageHeader.m_width; j++) {
      file.read(reinterpret_cast<char *>(buffer), bytesPP);
      image_pixels[i][j].m_blue = buffer[0];
      image_pixels[i][j].m_green = buffer[1];
      image_pixels[i][j].m_red = buffer[2];
 }
}
  delete[] buffer;
  return image_pixels;
}
void BMPImageProcess::read_image(const std::string &file_name) {
  std::ifstream file(file_name, std::ios::binary);
  if (!file.is_open()) {
    std::cerr << "unable to open file " << std::endl;
    file.close();
                                                  return;
}
            file.read(reinterpret_cast<char      *>(&m_bmpFileHeader),
sizeof(BMPFileHeader));
  if (m_bmpFileHeader.m_magic[0] != 'B' || m_bmpFileHeader.m_magic[1]
!= 'M') {
    std::cerr << "invalid file format" << std::endl;
    file.close();
    return;
}
           file.read(reinterpret_cast<char     *>(&m_bmpImageHeader),
sizeof(BMPImageHeader));
```

```
  m_pixels = read_data_of_pixels(file);
  if (file.fail()) {
    std::cerr << "Unable to read image data" << std::endl;
    file.close();
                                                    return;
}
                                       file.close();
}
std::vector<std::vector<Pixel>> BMPImageProcess::get_pixels() const {
                            return             m_pixels;
}
int BMPImageProcess::get_width() const {
                    return          m_bmpImageHeader.m_width;
}
int BMPImageProcess::get_height() const {
                   return          m_bmpImageHeader.m_height;
}
```

Файл image.cpp:

```
#include "header/image.h"
void Image::get_pixel(int x, int y, Pixel &pixel) {
  if (x < m_width && y < m_height) {
    pixel = m_pixels[x][y];
}
}
void Image::set_pixel(int x, int y, unsigned char red, unsigned char
green, unsigned char blue) {
  if (x < m_width && y < m_height) {
    m_pixels[x][y].m_red = red;
    m_pixels[x][y].m_green = green;
    m_pixels[x][y].m_blue = blue;
}
}
```

Файл bmp_image.cpp:

```
#include "header/bmp_image.h"
int BMPImage::get_width() const {
                             return              m_width;
}
int BMPImage::get_height() const {
                             return              m_height;
}
void BMPImage::set_width(const int &new_width) {
                                    m_width=new_width;
}
void BMPImage::set_height(const int &new_height) {
```

```
                                m_height=new_height;
}
std::vector<std::vector<Pixel>> BMPImage::get_pixels() const {
                            return              m_pixels;
}
void    BMPImage::set_pixels(const    std::vector<std::vector<Pixel>>
&pixels) {
  m_pixels=pixels;
}
```

Файл ppm_image.cpp:

```
#include "header/ppm_image.h"
#include <fstream>
void PPMImage::read_image(const std::string &file_name) {
  std::ifstream input_file(file_name, std::ios::binary);
  if (!input_file.is_open()) {
    std::cout << "error: Could not open input file" << std::endl;
    return;
}
   input_file >> m_magic >> m_width >> m_height >> m_max_color >>
std::noskipws >> m_last_line;
  if (m_magic != "P6") {
    std::cout << "error: Invalid PPM file format" << std::endl;
    input_file.close();
                                                    return;
}
  Pixel pixel;
               std::vector<std::vector<Pixel>>       pixels(m_height,
std::vector<Pixel>(m_width));
  for (int i = 0; i < m_height; i++) {
    for (int j = 0; j < m_width; j++) {
      input_file >> pixel.m_red >> pixel.m_green >> pixel.m_blue;
                                  pixels[i][j]     =     pixel;
}
}
  m_pixels=pixels;
                                  input_file.close();
}
int PPMImage::get_width() const {
                             return              m_width;
}
std::vector<std::vector<Pixel>> PPMImage::get_pixels() const {
                            return              m_pixels;
}
void    PPMImage::set_pixels(const    std::vector<std::vector<Pixel>>
&pixels) {
                     m_pixels          =          pixels;
}
```

```
int PPMImage::get_height() const {
                          return             m_height;
}
void PPMImage::set_width(const int &new_width) {
  m_width = new_width;
}
void PPMImage::set_height(const int &new_height) {
                   m_height          =          new_height;
}
```

Файл other_image.cpp:

```
#include <QImageReader>
#include "header/other_image.h"
std::vector<std::vector<Pixel>> OtherImage::get_pixels() const {
  return m_pixels; {
void   OtherImage::set_pixels(const   std::vector<std::vector<Pixel>>
&pixels) {
m_pixels=pixels;
}
int OtherImage::get_width() const {
                           return             m_width;
}
int OtherImage::get_height() const {
                          return             m_height;
}
void OtherImage::set_width(const int &new_width) {
m_width=new_width;
}
void OtherImage::set_height(const int &new_height) {
m_height=new_height;
}
std::vector<std::vector<Pixel>>    OtherImage::read(const    QString
&file_path) {
  std::vector<std::vector<Pixel>> pixels;
  QImageReader image_reader(file_path);
  QImage q_image = image_reader.read();
                                    pixels.resize(q_image.height(),
std::vector<Pixel>(q_image.width()));
  for (int y = 0; y < q_image.height(); ++y) {
    for (int x = 0; x < q_image.width(); ++x) {
      QRgb pixel_color = q_image.pixel(x, y);
      Pixel pixel;
      pixel.m_red = qRed(pixel_color);
      pixel.m_green = qGreen(pixel_color);
      pixel.m_blue = qBlue(pixel_color);
      pixels[y][x] = pixel;
}
}
```

```
  set_height(q_image.height());
  set_width(q_image.width());
  return pixels;
}
```

Файл pixel.h:

```
#ifndef TERM_WORK_PIXEL_H
#define TERM_WORK_PIXEL_H
#include "../../../convert.h"
#pragma pack(push, 1)
struct Pixel {
  unsigned char m_blue;
  unsigned char m_green;
  unsigned char m_red;
  Pixel operator+(const Pixel& other) const {
    Pixel result;
                         result.m_red      =      static_cast<unsigned
char>( static_cast<int>(m_red) + static_cast<int>(other.m_red));
                        result.m_green     =      static_cast<unsigned
char>( static_cast<int>(m_green) + static_cast<int>(other.m_green));
                         result.m_blue     =      static_cast<unsigned
char>( static_cast<int>(m_blue) + static_cast<int>(other.m_blue));
                                       return         result;
}
  Pixel operator-(const Pixel& other) const {
    Pixel result;
    result.m_red = static_cast<unsigned char>(static_cast<int>(m_red)
- static_cast<int>(other.m_red));
                        result.m_green     =      static_cast<unsigned
char>( static_cast<int>(m_green) - static_cast<int>(other.m_green));
                         result.m_blue     =      static_cast<unsigned
char>( static_cast<int>(m_blue) - static_cast<int>(other.m_blue));
                                       return         result;
}
  Pixel operator*(float scalar) const {
    Pixel result;
      result.m_red  =  static_cast<unsigned  char>(  static_cast<float
>(m_red) * scalar);
     result.m_green  =  static_cast<unsigned  char>(  static_cast<float
>(m_green) * scalar);
      result.m_blue  =  static_cast<unsigned  char>(static_cast<float
>(m_blue) * scalar);
                                       return         result;
}
  bool operator==(const Pixel& other) const {
    return m_red == other.m_red && m_green == other.m_green && m_blue
== other.m_blue;
}
```

```
  bool operator!=(const Pixel& other) const {
                        return      !(*this      ==      other);
}
};
#pragma pack(pop)
#endif //TERM_WORK_PIXEL_H
```

Файл bmp_header.h:

```
#ifndef TERM_WORK_BMP_HEADER_H
#define TERM_WORK_BMP_HEADER_H
#pragma pack(push,1)
typedef struct  {
  char m_magic[2];
  unsigned int m_size;
  unsigned int m_reserved;
  unsigned int m_offset;
}BMPFileHeader;
typedef struct  {
  unsigned int m_header_size;
  int m_width;
  int m_height;
  unsigned short m_planes;
  unsigned short m_bpp;
  unsigned int m_compression;
  unsigned int m_image_size;
  int m_x_ppm;
  int m_y_ppm;
  unsigned int m_colors;
  unsigned int m_important_colors;
}BMPImageHeader;
#pragma pack(pop)
#endif //TERM_WORK_BMP_HEADER_H
```

Файл image.h:

```
#ifndef TERM_WORK_IMAGE_H
#define TERM_WORK_IMAGE_H
#include <vector>
#include "pixel.h"
class Pixel;
class Image {
public:
   Image()=default;
  virtual ~Image() = default;
  virtual std::vector<std::vector<Pixel>> get_pixels() const = 0;
    virtual  void  set_pixels(const  std::vector<std::vector<Pixel>>
&pixels) = 0;
  void get_pixel(int x,int y,Pixel& pixel);
```

```
    void  set_pixel(int  x,int  y,unsigned  char  red,unsigned  char
g,unsigned char b);
   virtual int get_width() const = 0;
  virtual int get_height() const = 0;
  virtual void set_width(const int &new_width) = 0;
  virtual void set_height(const int &new_height) = 0;
protected:
  int m_width;
  int m_height;
  std::vector<std::vector<Pixel>> m_pixels;
};
#endif //TERM_WORK_IMAGE_H
```

Файл bmp_image.h:

```
#ifndef TERM_WORK_BMP_IMAGE_H
#define TERM_WORK_BMP_IMAGE_H
#include "image.h"
class BMPImage : public Image {
public:
  BMPImage() :Image() {};
  std::vector<std::vector<Pixel>> get_pixels() const override;
    void  set_pixels(const  std::vector<std::vector<Pixel>>  &pixels)
override;
  int get_width() const override;
  int get_height() const override;
  void set_width(const int &new_width) override;
  void set_height(const int &new_height) override;
};
#endif //TERM_WORK_BMP_IMAGE_H
```

Файл ppm_image.h:

```
#ifndef TERM_WORK_PPM_IMAGE_H
#define TERM_WORK_PPM_IMAGE_H
#include "image.h"
class PPMImage : public Image {
public:
  PPMImage():Image(){}
  void read_image(const std::string &file_name);
  int get_width() const override;
  std::vector<std::vector<Pixel>> get_pixels() const override;
    void  set_pixels(const  std::vector<std::vector<Pixel>>  &pixels)
override;
  int get_height() const override;
  void set_width(const int &new_width) override;
  void set_height(const int &new_height) override;
private:
  int m_max_color;
```

```
  std::string m_magic;
  char m_last_line;
};
#endif //TERM_WORK_PPM_IMAGE_H
```

Файл other_image.h:

```
#ifndef TERM_QT_OTHER_IMAGE_H
#define TERM_QT_OTHER_IMAGE_H
#include "image.h"
class OtherImage: public Image{
public:
  OtherImage():Image(){};
  std::vector<std::vector<Pixel>> get_pixels() const override;
    void  set_pixels(const  std::vector<std::vector<Pixel>>  &pixels)
override;
  int get_width() const override;
  int get_height() const override;
  void set_width(const int &new_width) override;
  void set_height(const int &new_height) override;
  std::vector<std::vector<Pixel>> read(const QString &filePath);
};
#endif //TERM_QT_OTHER_IMAGE_H
```

Файл customwidget.cpp:

```
#include "customwidget.h"
#include "ui_customwidget.h"
#include "../secondscreen/secondscreen.h"
#include <QPushButton>
CustomWidget::CustomWidget(QWidget* parent)
    : QWidget(parent)
    , ui(new Ui::CustomWidget) {
  ui->setupUi(this);
  ui->closeButton->setWindowIcon(QIcon(":icon_app/image/close_icon_4.
png"));
                connect(this,        &CustomWidget::send_remove_item,
qobject_cast<SecondScreen*>(parent), &SecondScreen::remove_item);
         connect(ui->closeButton,    &QPushButton::clicked,    this,
&CustomWidget::close_button_clicked);
}
CustomWidget::~CustomWidget() { }
void CustomWidget::set_text(const QString& text) {
                             ui->label->setText(text);
}
QString CustomWidget::get_text() {
                       return           ui->label->text();
}
void CustomWidget::close_button_clicked()
```

```
{   qDebug()<<"click"<<ui->label->text();
              emit       send_remove_item(ui->label->text());
}
```

Файл customwidget.h:

```
#ifndef TERM_QT_CUSTOMWIDGET_H
#define TERM_QT_CUSTOMWIDGET_H
#include <QWidget>
QT_BEGIN_NAMESPACE
namespace Ui { class CustomWidget; }
QT_END_NAMESPACE
class CustomWidget : public QWidget {
Q_OBJECT
public:
  explicit CustomWidget(QWidget* parent = nullptr);
  ~CustomWidget();
  void set_text(const QString& text);
  QString get_text();
signals:
  void send_remove_item(const QString& text);
private slots:
  void close_button_clicked();
private:
  Ui::CustomWidget *ui;
};
#endif //TERM_QT_CUSTOMWIDGET_H
```