Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление подготовки: 09.03.04 - «Программная инженерия»

Отчёт о творческой работе

на тему «Разработка приложения для туристов города Пермь»

Пермь 2025

**СОДЕРЖАНИЕ**

[СПИСОК ИСПОЛНИТЕЛЕЙ 3](#_Toc19974)

[ВВЕДЕНИЕ 4](#_Toc19577)

[ОТЧЕТ 5](#_Toc24516)

[1. Функционал приложения 5](#_Toc31976)

[2. Интерфейс приложения 7](#_Toc22434)

[3. Технология реализации 9](#_Toc19319)

[4. Тестирование продукта 10](#_Toc21659)

[ЗАКЛЮЧЕНИЕ 13](#_Toc31031)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 14](#_Toc16763)

[ПРИЛОЖЕНИЯ 15](#_Toc9172)

# СПИСОК ИСПОЛНИТЕЛЕЙ

Студентка группы РИС-24-1б Н. Морозова

подпись, дата

Студент группы РИС-24-1б С. Иванов

подпись, дата

# ВВЕДЕНИЕ

Современные города, такие как Пермь, обладают богатым культурным и историческим наследием, однако многие жители и туристы сталкиваются с трудностями при самостоятельном изучении достопримечательностей. Существующие решения часто перегружены избыточным функционалом, что затрудняет знакомство с городами.

Актуальность работы заключается в необходимости создания удобного инструмента для навигации по достопримечательностям города Перми, который объединит точность геолокации, лаконичный интерфейс и актуальную информацию о местах.

Цель работы: создание мобильного приложения «GeoTourist», автоматизирующего процесс знакомства с достопримечательностями Перми на основе местоположения пользователя.

Задачи:

1. Определить функционал приложения.
2. Спроектировать базовый пользовательский интерфейс.
3. Реализовать алгоритм в коде.
4. Протестировать работоспособность продукта.

# ОТЧЕТ

# Функционал приложения

Приложение должно считывать местоположение пользователя, который в свою очередь разрешает доступ к геолокации. При обнаружении ближайшей достопримечательности в радиусе, который настроит пользователь, приложение показывает список мест. Подразумевается, что пользователь будет двигаться непрерывно, поэтому список ближайших достопримечательностей может меняться с неизвестной скоростью, для этого необходимо, чтобы местоположение обновлялось с регулярной скоростью.

Местоположение ближайшей достопримечательности будет видно на карте относительно пользователя, а из списка можно выбрать любую другую ближайшую и ознакомиться с её названием, кратким описанием и адресом.

Чтобы выводить достопримечательности, нужно подготовить базу данных для их использования, то есть к каждому названию места привязано его описание, адрес и координаты. Координаты нужны для высчитывания расстояния до пользователя, а всё остальное для вывода на экран устройства.

Пройденные пользователем достопримечательности в приложении не запоминаются, поэтому предлагаемые места будут появляться каждый раз, как человек будет проходить мимо памятника или мемориала.

Приложение разрабатывается для устройств Android (рис. 1).

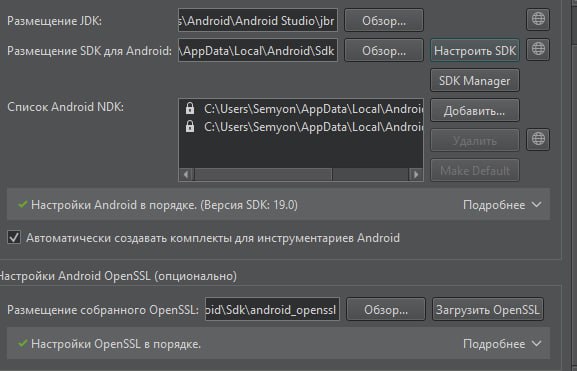


Рисунок 1 - Сборка под Android

**Вывод**

Определены необходимые функции для приложения, которые будут реализованы в дальнейшем.

# Интерфейс приложения

Примерное расположение элементов программы изображены на рисунке 2

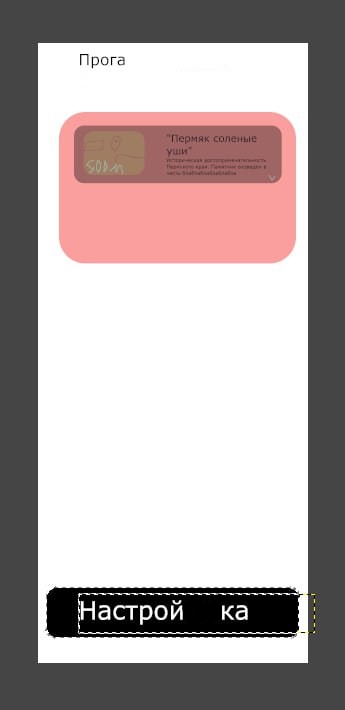


Рисунок 2 - Базовый интерфейс

Где на месте слова «Прога» будет название приложения; в основном поле будет отображаться список ближайших достопримечательностей, их краткое описание, фото и расстояние от пользователя в метрах; в самом низу будет кнопка настроек, где как раз и будет регулироваться радиус.

Но в результате разработки приложения интерфейс значительно поменялся. Итоговый интерфейс находится в приложении А.

Мы убрали фото и добавили карту, которая будет находиться над описанием достопримечательности. Реализация карты показана на рисунке 3.

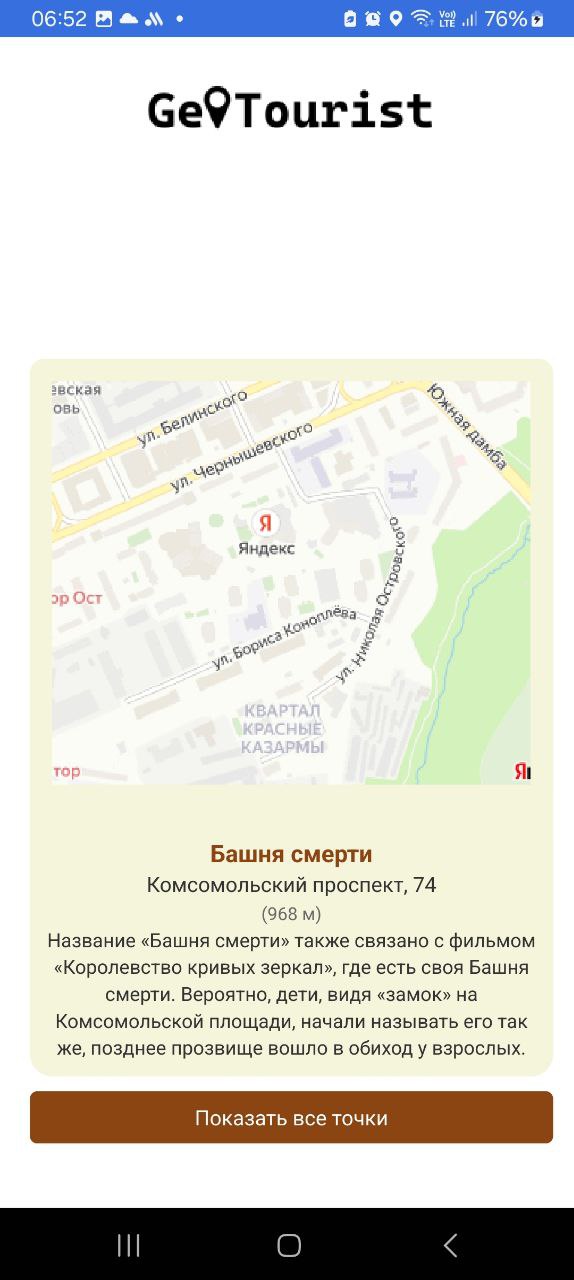


Рисунок 3 - Карта

**Вывод**

Разработан удобный интерфейс приложения.

# Технология реализации

Программа написана на языке C++, а интерфейс приложения создан с помощью Qt. Для работы приложения необходимо было заполнить базу данных с существующими достопримечательностями Перми, откуда будет использоваться информация с названием и точным местоположением. Программа определяет местоположение пользователя при помощи библиотек <QGeoPositionInfoSource> и <QGeoCoordinate>, находящихся в модуле QT positioning. Так как в приложениях на OC Android заблокирована возможность использования браузерных движков была использована статичная Яндекс.Карта, ссылка на которую реализовывается в функции loadYandexStaticMap вместе с двумя метками (красная – пользователь, зеленая - достопримечательность), позже, при помощи функции handleMapImageDownloaded изображение карты загружается в приложение, раз в несколько секунд карта обновляется в соответствии с актуальным местоположением пользователя.

Разработать уведомления нет возможности, так как они реализуются с помощью другого, нам ещё не известного, языка программирования - Kotlin / Java.

Код и диаграмма классов находятся в приложении Б и В соответственно.

**Вывод**

Реализовано приложение со всем задуманным функционалом.

# Тестирование продукта

Проверка на точность местоположения, его вывод: рисунки 4, 5.

Тест на корректность описания ближайшего места: рисунок 6.



Рисунок 4 - Данные в приложении

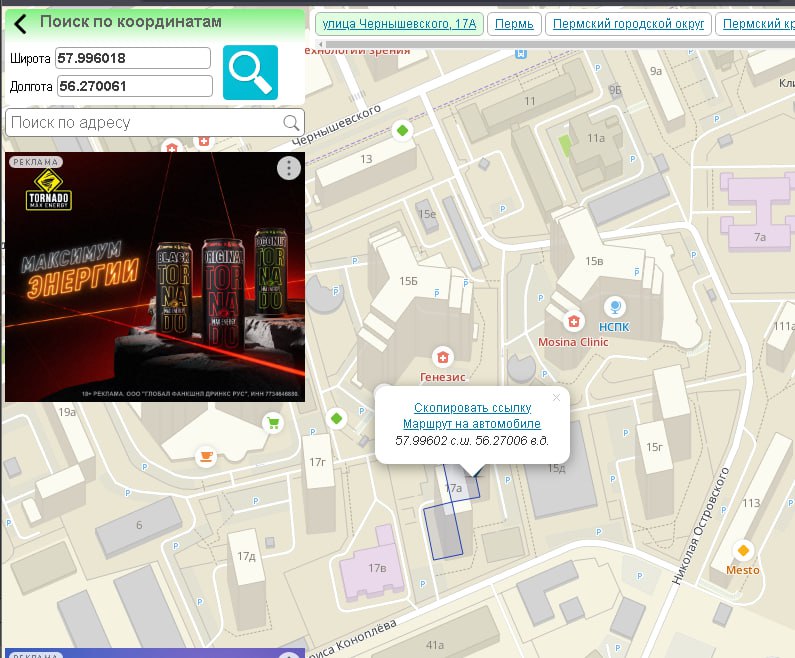


Рисунок 5 - Данные на другой карте



Рисунок 6 - Описание достопримечательности

**Вывод**

Приложение подтверждает корректность работы задуманных механизмов. Оно успешно обрабатывает базу данных.

# ЗАКЛЮЧЕНИЕ

Созданное приложение успешно справляется с поставленными задачами, его можно рекомендовать как жителям, так и посетителям нашего города для комфортного изучения культурного и исторического наследия Перми.

Практическая значимость заключается в том, что разработанное приложение помогает ориентироваться в рекомендованных местах.

Перспективой дальнейших разработок можно рассмотреть создание push-уведомлений для приложения и расширение базы данных достопримечательностей не только города Перми, но и других городов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. <https://yandex.ru/maps-api/docs/static-api/index.html>
2. <https://doc.qt.io/qt-6/qtsql-index.html>
3. <https://doc.qt.io/qt-6/qtnetwork-index.html>
4. <https://doc.qt.io/qt-6/qtpositioning-index.html>

# ПРИЛОЖЕНИЯ

**Приложение А**

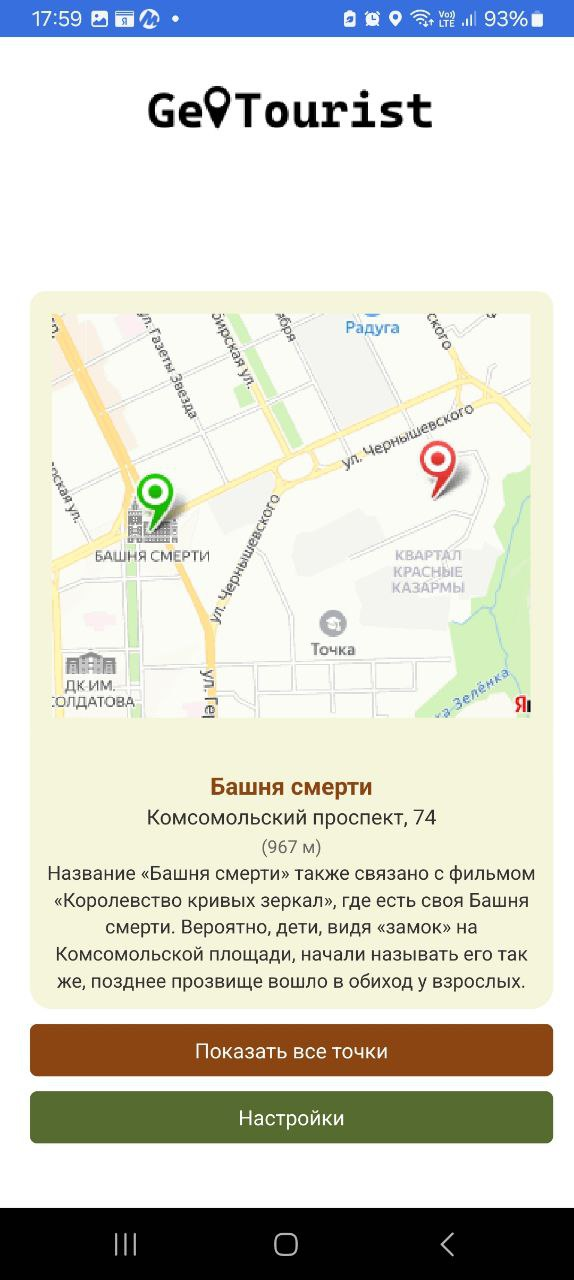


Рисунок 7 - Исходное окно

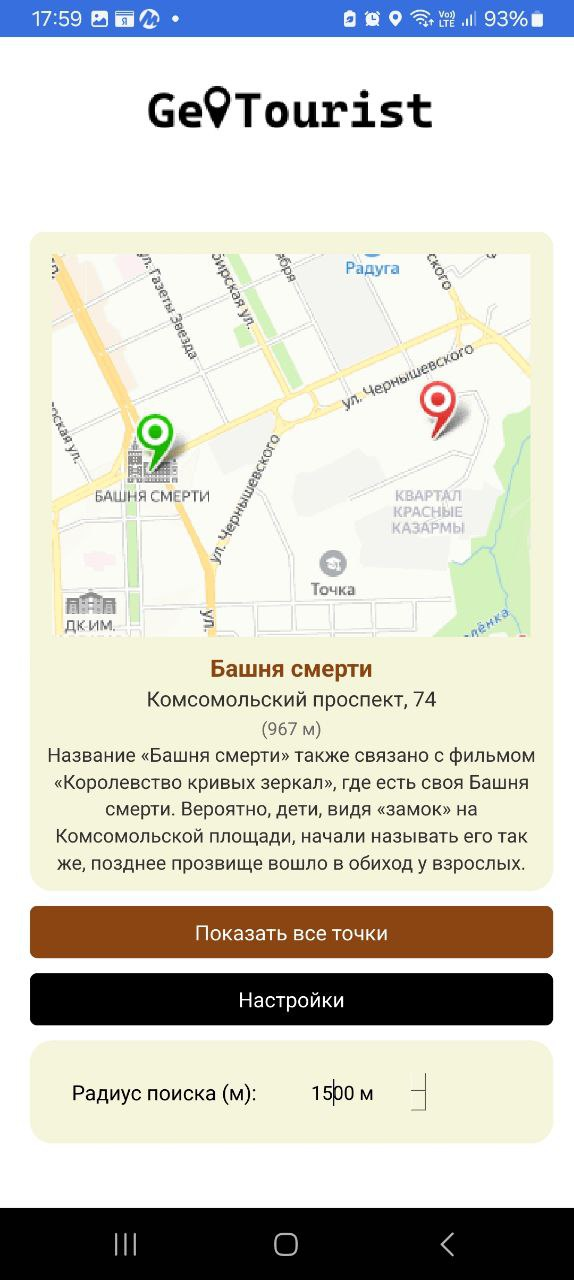


Рисунок 8 - Настройки радиуса

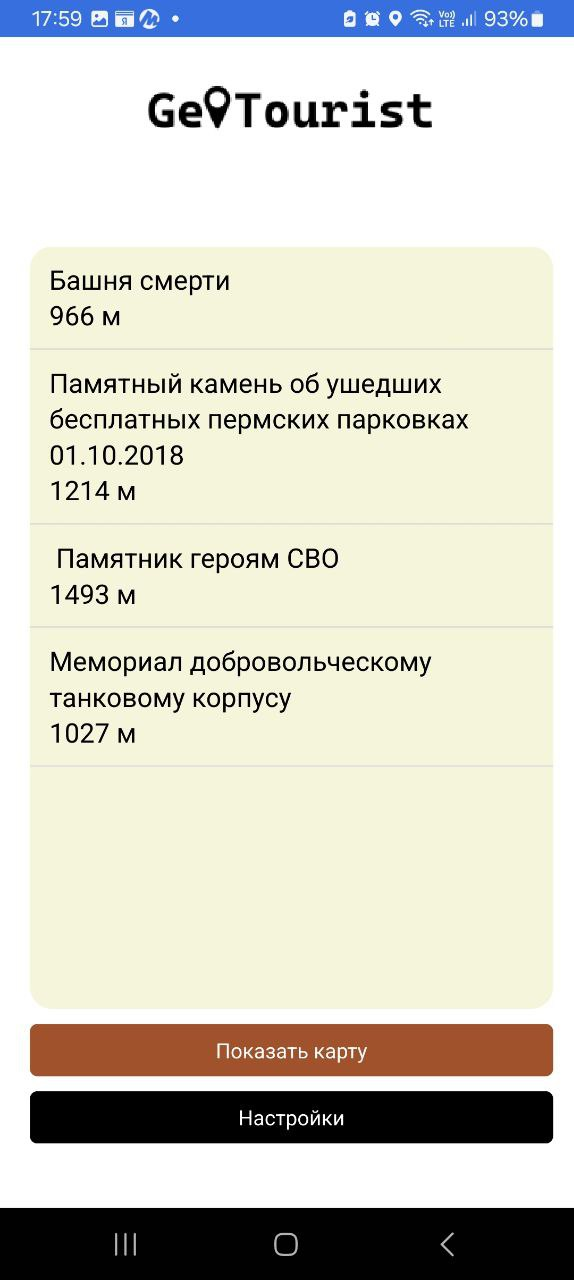


Рисунок 9 - Список ближайших достопримечательностей

**Приложение Б**

**Заголовочные файлы**

**Locationmanager.h**

#ifndef LOCATIONMANAGER\_H

#define LOCATIONMANAGER\_H

#include <QObject>

#include <QGeoPositionInfoSource>

#include <QGeoCoordinate>

class LocationManager : public QObject{

Q\_OBJECT

public:

explicit LocationManager(QObject \*parent = nullptr);

~LocationManager();

// Основные методы

void startLocationUpdates();

void stopLocationUpdates();

double calculateDistanceTo(double latitude, double longitude) const;

static double calculateDistanceTo(double fromLat, double fromLng, double toLat, double toLng);

// Текущие координаты

double latitude() const { return m\_latitude; }

double longitude() const { return m\_longitude; }

bool hasValidLocation() const { return m\_hasValidLocation; }

signals:

// Сигналы для обновления позиции и ошибок

void positionUpdated(double latitude, double longitude);

void errorOccurred(const QString &error);

private slots:

void handlePositionUpdated(const QGeoPositionInfo &info);

void handlePositionError(QGeoPositionInfoSource::Error positioningError);

private:

QGeoPositionInfoSource \*m\_positionSource = nullptr;

double m\_latitude = 0.0;

double m\_longitude = 0.0;

bool m\_hasValidLocation = false;

};

#endif // LOCATIONMANAGER\_H

**Databasemanager.h**

#ifndef DATABASEMANAGER\_H

#define DATABASEMANAGER\_H

#include <QObject>

#include <QSqlDatabase>

#include <QSqlQuery>

#include <QSqlError>

#include <QList>

#include <QVariantMap>

class DatabaseManager : public QObject{

Q\_OBJECT

public:

explicit DatabaseManager(QObject \*parent = nullptr);

~DatabaseManager();

bool updateDatabaseSchema();

bool openDatabase();

void closeDatabase();

bool initializeDatabase(); // Добавлен новый метод

QList<QVariantMap> getNearbyPoints(double latitude, double longitude, double maxDistance);

private:

QSqlDatabase m\_database;

};

#endif // DATABASEMANAGER\_H

**Mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QLabel>

#include <QNetworkAccessManager>

#include <QListWidget>

#include <QListWidgetItem>

#include <QPushButton>

#include <QSpinBox>

#include <QTextEdit>

#include "databasemanager.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private:

Ui::MainWindow \*ui;

DatabaseManager \*m\_dbManager;

QTextEdit \*m\_pointInfoLabel; // Изменено с QLabel на QTextEdit

QLabel \*m\_mapLabel;

QNetworkAccessManager \*m\_networkManager;

QListWidget \*m\_pointsListWidget;

QPushButton \*m\_toggleViewButton;

QPushButton \*m\_settingsButton;

QWidget \*m\_settingsPanel;

double m\_currentLat = 0.0;

double m\_currentLng = 0.0;

QList<QVariantMap> m\_nearbyPoints;

double m\_searchRadius = 500.0;

int m\_selectedPointId = -1;

void setupUI();

void initSSL();

void showPointInfo(const QVariantMap &point);

void loadYandexStaticMap(double pointLat, double pointLng);

void updateMapImage(const QByteArray &imageData);

void setupPointsListView();

void showPointsList();

void showSinglePoint();

void initLocationManager();

void checkNearbyPoints(double lat, double lng);

private slots:

void updatePosition(double lat, double lng);

void showLocationError(const QString &error);

void handleMapImageDownloaded();

void onPointSelected(QListWidgetItem \*item);

void onRadiusChanged(int value);

};

#endif // MAINWINDOW\_H

**Исходники**

**Locationmanager.cpp**

#include "locationmanager.h"

#include <QGeoPositionInfoSource>

#include <QGeoCoordinate>

#include <QDebug>

LocationManager::LocationManager(QObject \*parent)

: QObject(parent), m\_positionSource(QGeoPositionInfoSource::createDefaultSource(this)),

m\_latitude(0.0),

m\_longitude(0.0),

m\_hasValidLocation(false){

if (m\_positionSource) {

// Правильное подключение сигналов

connect(m\_positionSource, &QGeoPositionInfoSource::positionUpdated,

this, &LocationManager::handlePositionUpdated);

// Исправленный сигнал таймаута (если доступен)

#if QT\_VERSION >= QT\_VERSION\_CHECK(5, 12, 0)

connect(m\_positionSource, &QGeoPositionInfoSource::errorOccurred,

this, &LocationManager::handlePositionError);

#else

connect(m\_positionSource, QOverload<QGeoPositionInfoSource::Error>::of(&QGeoPositionInfoSource::error), this, &LocationManager::handlePositionError);

#endif

} else {

qWarning() << "No position source available";

emit errorOccurred(tr("No position source available on this device"));

}

}

LocationManager::~LocationManager(){

stopLocationUpdates();

}

void LocationManager::startLocationUpdates(){

if (!m\_positionSource) {

emit errorOccurred(tr("Position source not available"));

return;

}

m\_positionSource->setUpdateInterval(10000);

m\_positionSource->startUpdates();

}

void LocationManager::stopLocationUpdates(){

if (m\_positionSource) {

m\_positionSource->stopUpdates();

}

}

double LocationManager::calculateDistanceTo(double latitude, double longitude) const{

if (!m\_hasValidLocation) {

return -1.0;

}

return QGeoCoordinate(m\_latitude, m\_longitude).distanceTo(QGeoCoordinate(latitude, longitude));

}

double LocationManager::calculateDistanceTo(double fromLat, double fromLng, double toLat, double toLng){

return QGeoCoordinate(fromLat, fromLng).distanceTo(QGeoCoordinate(toLat, toLng));

}

void LocationManager::handlePositionUpdated(const QGeoPositionInfo &info){

// Исправленная проверка валидности координат

if (info.coordinate().isValid()) {

m\_latitude = info.coordinate().latitude();

m\_longitude = info.coordinate().longitude();

m\_hasValidLocation = true;

emit positionUpdated(m\_latitude, m\_longitude);

}

}

void LocationManager::handlePositionError(QGeoPositionInfoSource::Error positioningError){

QString errorString;

switch (positioningError) {

case QGeoPositionInfoSource::AccessError:

errorString = tr("Access to position information denied");

break;

case QGeoPositionInfoSource::ClosedError:

errorString = tr("Position source closed");

break;

case QGeoPositionInfoSource::NoError:

return;

default:

errorString = tr("Unknown positioning error");

}

qWarning() << "Location error:" << errorString;

emit errorOccurred(errorString);

}

**Databasemanager.cpp**

#include "databasemanager.h"

#include "locationmanager.h"

#include <QFile>

#include <QDebug>

DatabaseManager::DatabaseManager(QObject \*parent) : QObject(parent) {

#ifdef QT\_DEBUG

QFile::remove("points.db"); // Удаляем в режиме отладки

#endif

m\_database = QSqlDatabase::addDatabase("QSQLITE");

m\_database.setDatabaseName("points.db");

qDebug() << "Database path:" << m\_database.databaseName();

}

DatabaseManager::~DatabaseManager() { closeDatabase(); }

bool DatabaseManager::openDatabase(){

if (!m\_database.open()) {

qWarning() << "Error opening database:" << m\_database.lastError().text();

return false;

}

return true;

}

void DatabaseManager::closeDatabase(){

if (m\_database.isOpen()) {

m\_database.close();

}

}

bool DatabaseManager::initializeDatabase() {

if (!openDatabase()) return false;

#ifdef QT\_DEBUG

// В режиме отладки всегда пересоздаем структуру

QSqlQuery("DROP TABLE IF EXISTS points");

#endif

// Проверяем, есть ли таблица points

if (!m\_database.tables().contains("points")) {

// Таблицы нет - создаем новую из скрипта

QFile scriptFile(":/bd/init\_db.sql");

if (!scriptFile.open(QIODevice::ReadOnly)) {

qWarning() << "Failed to open SQL script";

return false;

}

QSqlQuery query;

QStringList scriptQueries = QString(scriptFile.readAll()).split(';', Qt::SkipEmptyParts);

for (const QString &queryStr : scriptQueries) {

if (queryStr.trimmed().isEmpty()) continue;

if (!query.exec(queryStr)) {

qWarning() << "Query failed:" << query.lastError().text();

return false;

}

}

} else {

// Таблица существует - обновляем схему

if (!updateDatabaseSchema()) {

return false;

}

}

return true;

}

QList<QVariantMap> DatabaseManager::getNearbyPoints(double latitude, double longitude, double maxDistance){

QList<QVariantMap> result;

if (!m\_database.isOpen()) return result;

QSqlQuery query;

query.prepare("SELECT id, name, description, address, latitude, longitude, image\_path FROM points");

if (!query.exec()) {

qWarning() << "Query error:" << query.lastError().text();

return result;

}

while (query.next()) {

double pointLat = query.value(4).toDouble();

double pointLng = query.value(5).toDouble();

double distance = LocationManager::calculateDistanceTo(latitude, longitude, pointLat, pointLng);

if (distance <= maxDistance) {

QVariantMap point;

point["id"] = query.value(0);

point["name"] = query.value(1);

point["description"] = query.value(2);

point["address"] = query.value(3); // Добавляем адрес

point["latitude"] = pointLat;

point["longitude"] = pointLng;

point["image\_path"] = query.value(6);

point["distance"] = distance;

result.append(point);

}

}

return result;

}

bool DatabaseManager::updateDatabaseSchema(){

if (!openDatabase()) return false;

// Проверяем, есть ли столбец address

QSqlQuery checkQuery("PRAGMA table\_info(points)");

bool hasAddressColumn = false;

while (checkQuery.next()) {

if (checkQuery.value(1).toString() == "address") {

hasAddressColumn = true;

break;

}

}

// Если столбца нет - добавляем

if (!hasAddressColumn) {

QSqlQuery alterQuery;

if (!alterQuery.exec("ALTER TABLE points ADD COLUMN address TEXT")) {

qWarning() << "Failed to add address column:" << alterQuery.lastError().text();

return false;

}

qDebug() << "Added address column to existing database";

}

return true;

}

**Mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "locationmanager.h"

#include "databasemanager.h"

#include <QVBoxLayout>

#include <QLabel>

#include <QPainter>

#include <QPixmap>

#include <QDebug>

#include <QMessageBox>

#include <QGeoCoordinate>

#include <QNetworkReply>

#include <QUrlQuery>

#include <QSslSocket>

#include <QNetworkAccessManager>

#include <QListWidgetItem>

#include <QPushButton>

#include <QJniObject>

#ifdef Q\_OS\_ANDROID

#include <QJniObject>

#include <QJniEnvironment>

void checkAndRequestPermissions(){

QStringList permissions = {

"android.permission.ACCESS\_FINE\_LOCATION",

"android.permission.ACCESS\_COARSE\_LOCATION",

"android.permission.INTERNET"

};

QJniEnvironment env;

jclass stringClass = env.findClass("java/lang/String");

jobjectArray jPermissions = env->NewObjectArray(permissions.size(), stringClass, nullptr);

for (int i = 0; i < permissions.size(); ++i) {

QJniObject jniString = QJniObject::fromString(permissions.at(i));

env->SetObjectArrayElement(jPermissions, i, jniString.object());

}

bool needRequest = false;

for (const QString &permission : permissions) {

auto result = QJniObject::callStaticMethod<jint>(

"androidx/core/content/ContextCompat",

"checkSelfPermission",

"(Landroid/content/Context;Ljava/lang/String;)I",

QNativeInterface::QAndroidApplication::context(),

QJniObject::fromString(permission).object<jstring>());

if (result != 0) {

needRequest = true;

break;

}

}

if (needRequest) {

QJniObject activity = QJniObject::callStaticObjectMethod(

"org/qtproject/qt/android/QtNative",

"activity",

"()Landroid/app/Activity;");

if (activity.isValid()) {

QJniObject::callStaticMethod<void>(

"androidx/core/app/ActivityCompat",

"requestPermissions",

"(Landroid/app/Activity;[Ljava/lang/String;I)V",

activity.object(),

jPermissions,

0);

}

}

}

#endif

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

, m\_dbManager(new DatabaseManager(this))

, m\_pointInfoLabel(nullptr)

, m\_mapLabel(new QLabel(this))

, m\_networkManager(new QNetworkAccessManager(this))

, m\_pointsListWidget(new QListWidget(this))

, m\_toggleViewButton(new QPushButton("Показать все точки", this))

, m\_settingsButton(new QPushButton("Настройки", this))

, m\_settingsPanel(new QWidget(this))

{

ui->setupUi(this);

setStyleSheet("background-color: white;");

initSSL();

m\_mapLabel->setAlignment(Qt::AlignCenter);

m\_mapLabel->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);

m\_mapLabel->setMinimumSize(400, 300);

m\_mapLabel->setStyleSheet("QLabel { background-color: #F0F0F0; }");

#ifdef Q\_OS\_ANDROID

checkAndRequestPermissions();

#endif

if (!m\_dbManager->openDatabase()) {

qWarning() << "Failed to open database";

ui->label->setText("Ошибка открытия базы данных");

return;

}

if (!m\_dbManager->initializeDatabase()) {

qWarning() << "Failed to initialize database";

ui->label->setText("Ошибка инициализации базы данных");

return;

}

setupUI();

setupPointsListView();

initLocationManager();

}

void MainWindow::initSSL(){

qDebug() << "SSL supported:" << QSslSocket::supportsSsl();

qDebug() << "SSL version:" << QSslSocket::sslLibraryVersionString();

QSslConfiguration sslConfig = QSslConfiguration::defaultConfiguration();

sslConfig.setPeerVerifyMode(QSslSocket::VerifyNone);

sslConfig.setProtocol(QSsl::TlsV1\_2OrLater);

QSslConfiguration::setDefaultConfiguration(sslConfig);

}

void MainWindow::setupUI(){

QWidget \*container = new QWidget(this);

QVBoxLayout \*mainLayout = new QVBoxLayout(container);

mainLayout->setContentsMargins(20, 20, 20, 20);

mainLayout->setSpacing(10);

QLabel \*imageLabel = new QLabel(this);

QPixmap logo(":/logo/logo.png");

if(!logo.isNull()) {

imageLabel->setPixmap(logo.scaledToWidth(200, Qt::SmoothTransformation));

} else {

imageLabel->setText("Логотип");

}

imageLabel->setAlignment(Qt::AlignCenter);

imageLabel->setStyleSheet("margin-top: 0; padding-top: 10px;");

ui->label->setAlignment(Qt::AlignCenter);

ui->label->setStyleSheet("QLabel { color: black; font-size: 16px; margin: 10px; }");

QWidget \*infoContainer = new QWidget(this);

infoContainer->setStyleSheet(

"QWidget {"

" background-color: #F5F5DC;"

" border-radius: 15px;"

" padding: 15px;"

"}"

);

QVBoxLayout \*infoLayout = new QVBoxLayout(infoContainer);

infoLayout->setContentsMargins(0, 0, 0, 0);

infoLayout->setSpacing(10);

m\_mapLabel = new QLabel(infoContainer);

m\_mapLabel->setAlignment(Qt::AlignCenter);

m\_mapLabel->setStyleSheet("QLabel { background-color: #F5F5DC; border-radius: 10px; }");

m\_mapLabel->setMinimumSize(350, 300);

m\_mapLabel->setMaximumSize(350, 300);

m\_mapLabel->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);

m\_mapLabel->setScaledContents(false);

// Заменяем QLabel на QTextEdit

m\_pointInfoLabel = new QTextEdit(infoContainer);

m\_pointInfoLabel->setReadOnly(true);

m\_pointInfoLabel->setAlignment(Qt::AlignCenter);

m\_pointInfoLabel->setStyleSheet(

"QTextEdit {"

" color: #333333;"

" font-size: 14px;"

" padding: 10px;"

" border: none;"

" background: transparent;"

"}");

m\_pointInfoLabel->setFrameShape(QFrame::NoFrame);

m\_pointInfoLabel->setVerticalScrollBarPolicy(Qt::ScrollBarAsNeeded);

m\_pointInfoLabel->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

infoLayout->addWidget(m\_mapLabel);

infoLayout->addWidget(m\_pointInfoLabel);

mainLayout->addWidget(imageLabel, 0, Qt::AlignTop | Qt::AlignHCenter);

mainLayout->addWidget(ui->label);

mainLayout->addWidget(infoContainer);

setCentralWidget(container);

}

void MainWindow::setupPointsListView(){

m\_pointsListWidget->setStyleSheet(

"QListWidget {"

" background-color: #F5F5DC;"

" border-radius: 15px;"

" color: black;"

"}"

"QListWidget::item {"

" border-bottom: 1px solid #DDD;"

" padding: 10px;"

" color: black;"

"}"

"QListWidget::item:hover {"

" background-color: #EEE;"

"}"

);

m\_pointsListWidget->setWordWrap(true);

m\_pointsListWidget->setTextElideMode(Qt::ElideRight); m\_pointsListWidget->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

m\_pointsListWidget->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);

m\_pointsListWidget->setMaximumWidth(400);

m\_pointsListWidget->hide(); qobject\_cast<QVBoxLayout\*>(centralWidget()->layout())->addWidget(m\_pointsListWidget);

m\_toggleViewButton->setStyleSheet(

"QPushButton {"

" background-color: #8B4513;"

" color: white;"

" border-radius: 5px;"

" padding: 8px;"

"}"

"QPushButton:hover {"

" background-color: #A0522D;"

"}"

); qobject\_cast<QVBoxLayout\*>(centralWidget()->layout())->addWidget(m\_toggleViewButton);

m\_settingsButton->setStyleSheet(

"QPushButton {"

" background-color: #000000;"

" color: white;"

" border-radius: 5px;"

" padding: 8px;"

"}"

"QPushButton:hover {"

" background-color: #556B2F;"

"}"

); qobject\_cast<QVBoxLayout\*>(centralWidget()->layout())->addWidget(m\_settingsButton);

m\_settingsPanel->setStyleSheet(

"QWidget {"

" background-color: #F5F5DC;"

" border-radius: 15px;"

" padding: 15px;"

"}"

);

m\_settingsPanel->hide();

QVBoxLayout \*settingsLayout = new QVBoxLayout(m\_settingsPanel);

settingsLayout->setContentsMargins(10, 10, 10, 10);

settingsLayout->setSpacing(10);

QHBoxLayout \*radiusLayout = new QHBoxLayout();

QLabel \*radiusLabel = new QLabel("Радиус поиска (м):", m\_settingsPanel);

radiusLabel->setStyleSheet("QLabel { color: black; font-size: 14px; }");

QSpinBox \*radiusSpinBox = new QSpinBox(m\_settingsPanel);

radiusSpinBox->setRange(100, 20000);

radiusSpinBox->setSingleStep(100);

radiusSpinBox->setValue(m\_searchRadius);

radiusSpinBox->setSuffix(" м");

radiusSpinBox->setStyleSheet(

"QSpinBox {"

" padding: 5px;"

" color: black;"

"}"

);

radiusLayout->addWidget(radiusLabel);

radiusLayout->addWidget(radiusSpinBox);

radiusLayout->addStretch();

settingsLayout->addLayout(radiusLayout); qobject\_cast<QVBoxLayout\*>(centralWidget()->layout())->addWidget(m\_settingsPanel);

connect(m\_toggleViewButton, &QPushButton::clicked, this, [this]() {

if (m\_pointsListWidget->isVisible()) {

showSinglePoint();

} else {

showPointsList();

}

});

connect(m\_settingsButton, &QPushButton::clicked, this, [this]() {

setUpdatesEnabled(false);

m\_settingsPanel->setVisible(!m\_settingsPanel->isVisible());

setUpdatesEnabled(true);

});

connect(radiusSpinBox, QOverload<int>::of(&QSpinBox::valueChanged),

this, [this](int value) {

m\_searchRadius = value;

checkNearbyPoints(m\_currentLat, m\_currentLng);

});

connect(m\_pointsListWidget, &QListWidget::itemClicked, this, &MainWindow::onPointSelected);

}

void MainWindow::onRadiusChanged(int value){

m\_searchRadius = value;

checkNearbyPoints(m\_currentLat, m\_currentLng);

}

void MainWindow::showPointsList(){

layout()->setEnabled(true);

setUpdatesEnabled(true);

m\_pointInfoLabel->hide();

m\_mapLabel->hide();

m\_pointsListWidget->show();

m\_toggleViewButton->setText("Показать карту");

m\_pointsListWidget->clear();

for (const QVariantMap &point : m\_nearbyPoints) {

QString text = QString("%1\n%2 м").arg(point["name"].toString()).arg(point["distance"].toInt());

QListWidgetItem \*item = new QListWidgetItem(text, m\_pointsListWidget);

item->setData(Qt::UserRole, point["id"]);

item->setTextAlignment(Qt::AlignLeft);

item->setToolTip(point["name"].toString());

}

layout()->setEnabled(true);

setUpdatesEnabled(true);

}

void MainWindow::showSinglePoint(){

layout()->setEnabled(true);

setUpdatesEnabled(true);

m\_pointsListWidget->hide();

m\_pointInfoLabel->show();

m\_mapLabel->show();

m\_toggleViewButton->setText("Показать все точки");

if (!m\_nearbyPoints.isEmpty()) {

showPointInfo(m\_nearbyPoints.first());

}

layout()->setEnabled(true);

setUpdatesEnabled(true);

}

void MainWindow::onPointSelected(QListWidgetItem \*item){

m\_selectedPointId = item->data(Qt::UserRole).toInt();

for (const QVariantMap &point : m\_nearbyPoints) {

if (point["id"].toInt() == m\_selectedPointId) {

showPointInfo(point);

break;

}

}

}

void MainWindow::showPointInfo(const QVariantMap &point){

QString info = QString("<div style='text-align: center;'>"

"<b style='font-size: 16px; color: #8B4513;'>%1</b><br>"

"<span style='font-size: 14px;'>%2</span><br>"

"<span style='font-size: 12px; color: #696969;'>(%3 м)</span><br>"

"<span style='font-size: 13px;'>%4</span>"

"</div>")

.arg(point["name"].toString())

.arg(point["address"].toString())

.arg(point["distance"].toInt())

.arg(point["description"].toString());

m\_pointInfoLabel->setHtml(info); // Используем setHtml вместо setText

loadYandexStaticMap(point["latitude"].toDouble(), point["longitude"].toDouble());

}

void MainWindow::loadYandexStaticMap(double pointLat, double pointLng){

const QString apiKey = "a5183891-5776-4fa6-a843-fac9320c41f3";

QUrl url("http://static-maps.yandex.ru/1.x/");

QUrlQuery query;

double centerLat = (m\_currentLat + pointLat) / 2;

double centerLng = (m\_currentLng + pointLng) / 2;

query.addQueryItem("ll", QString("%1,%2")

.arg(centerLng, 0, 'f', 6)

.arg(centerLat, 0, 'f', 6));

query.addQueryItem("z", "14");

query.addQueryItem("size", "400,300");

QString marks = QString("%1,%2,pm2rdl~%3,%4,pm2gnl")

.arg(m\_currentLng, 0, 'f', 6)

.arg(m\_currentLat, 0, 'f', 6)

.arg(pointLng, 0, 'f', 6)

.arg(pointLat, 0, 'f', 6);

query.addQueryItem("pt", marks);

query.addQueryItem("apikey", apiKey);

query.addQueryItem("l", "map");

url.setQuery(query);

QNetworkRequest request(url);

request.setHeader(QNetworkRequest::UserAgentHeader, "Mozilla/5.0");

request.setAttribute(QNetworkRequest::RedirectPolicyAttribute, QNetworkRequest::NoLessSafeRedirectPolicy);

QSslConfiguration sslConfig = request.sslConfiguration();

sslConfig.setProtocol(QSsl::TlsV1\_2OrLater);

request.setSslConfiguration(sslConfig);

QNetworkReply\* reply = m\_networkManager->get(request);

connect(reply, &QNetworkReply::finished, this, &MainWindow::handleMapImageDownloaded);

connect(reply, &QNetworkReply::errorOccurred, this, [this](QNetworkReply::NetworkError error) {

qDebug() << "Network error occurred:" << error;

m\_mapLabel->setText("Ошибка загрузки карты");

});

}

void MainWindow::handleMapImageDownloaded(){

QNetworkReply\* reply = qobject\_cast<QNetworkReply\*>(sender());

if (!reply) return;

QScopedPointer<QNetworkReply, QScopedPointerDeleteLater> replyDeleter(reply);

if (reply->error() != QNetworkReply::NoError) {

qWarning() << "Network error:" << reply->errorString();

m\_mapLabel->setText("Ошибка сети: " + reply->errorString());

return;

}

const QByteArray imageData = reply->readAll();

if (imageData.isEmpty()) {

m\_mapLabel->setText("Пустой ответ от сервера");

return;

}

QPixmap map;

if (!map.loadFromData(imageData)) {

m\_mapLabel->setText("Неверный формат изображения");

return;

}

QSize labelSize = m\_mapLabel->size();

QPixmap scaled = map.scaled(labelSize, Qt::KeepAspectRatioByExpanding, Qt::SmoothTransformation);

QPixmap rounded(scaled.size());

rounded.fill(Qt::transparent);

QPainter painter(&rounded);

painter.setRenderHint(QPainter::Antialiasing);

painter.setBrush(QBrush(scaled));

painter.setPen(Qt::NoPen);

painter.drawRoundedRect(rounded.rect(), 10, 10);

m\_mapLabel->setPixmap(rounded);

}

void MainWindow::checkNearbyPoints(double lat, double lng){

m\_nearbyPoints = m\_dbManager->getNearbyPoints(lat, lng, m\_searchRadius);

if (!m\_nearbyPoints.isEmpty()) {

bool selectedPointStillAvailable = false;

QVariantMap pointToShow;

if (m\_selectedPointId != -1) {

for (const QVariantMap &point : m\_nearbyPoints) {

if (point["id"].toInt() == m\_selectedPointId) {

selectedPointStillAvailable = true;

pointToShow = point;

break;

}

}

}

if (!selectedPointStillAvailable) {

m\_selectedPointId = -1;

double minDistance = m\_searchRadius;

for (const QVariantMap &point : m\_nearbyPoints) {

double distance = point["distance"].toDouble();

if (distance < minDistance) {

minDistance = distance;

pointToShow = point;

}

}

}

if (!m\_pointsListWidget->isVisible()) {

showPointInfo(pointToShow);

}

} else {

m\_selectedPointId = -1;

m\_pointInfoLabel->setText("Рядом нет интересных точек");

m\_mapLabel->setText("Карта не доступна");

}

}

void MainWindow::initLocationManager(){

LocationManager \*locationManager = new LocationManager(this);

connect(locationManager, &LocationManager::positionUpdated,

this, &MainWindow::updatePosition);

connect(locationManager, &LocationManager::errorOccurred,

this, &MainWindow::showLocationError);

locationManager->startLocationUpdates();

}

void MainWindow::updatePosition(double lat, double lng){

m\_currentLat = lat;

m\_currentLng = lng;

checkNearbyPoints(lat, lng);

}

void MainWindow::showLocationError(const QString &error){

ui->label->setText("Ошибка: " + error);

QMessageBox::warning(this, "Ошибка геолокации", error);

}

MainWindow::~MainWindow(){

delete ui;

delete m\_dbManager;

delete m\_networkManager;

delete m\_mapLabel;

}

**Main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <QLocale>

#include <QTranslator>

int main(int argc, char \*argv[]){

QApplication a(argc, argv);

QTranslator translator;

const QStringList uiLanguages = QLocale::system().uiLanguages();

for (const QString &locale : uiLanguages) {

const QString baseName = "GeoTourist\_12\_" + QLocale(locale).name();

if (translator.load(":/i18n/" + baseName)) {

a.installTranslator(&translator);

break;

}

}

MainWindow w;

w.show();

return a.exec();

}

**Приложение В**

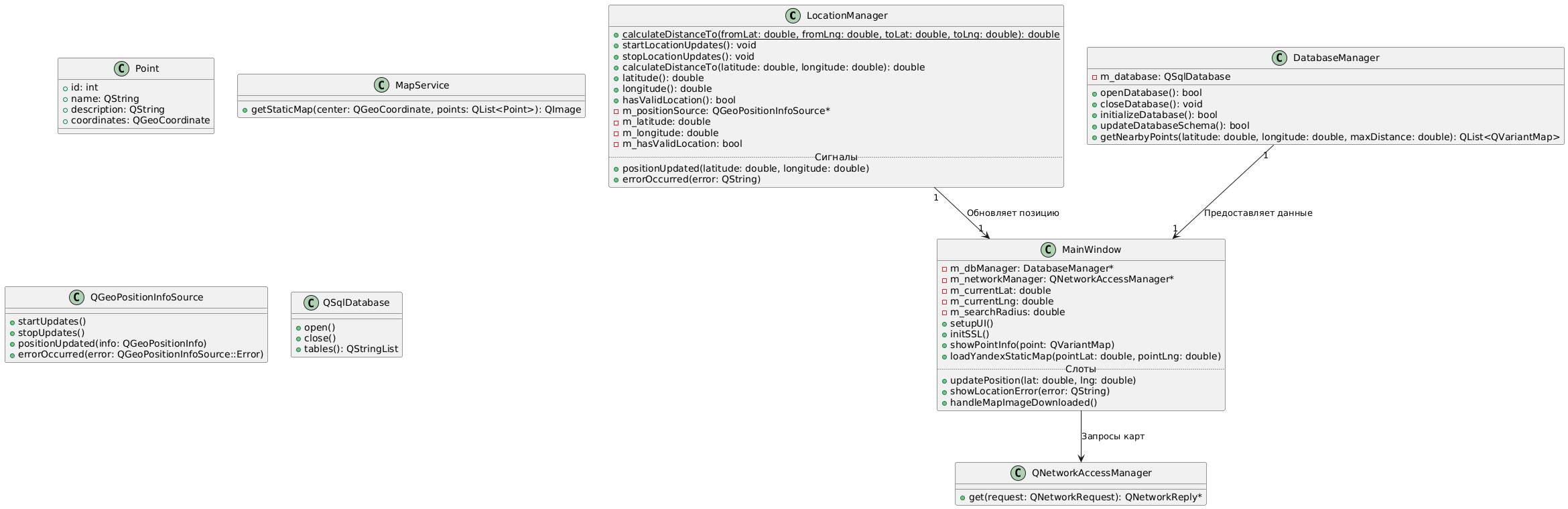


Рисунок 10 - Диаграмма классов, 1 часть

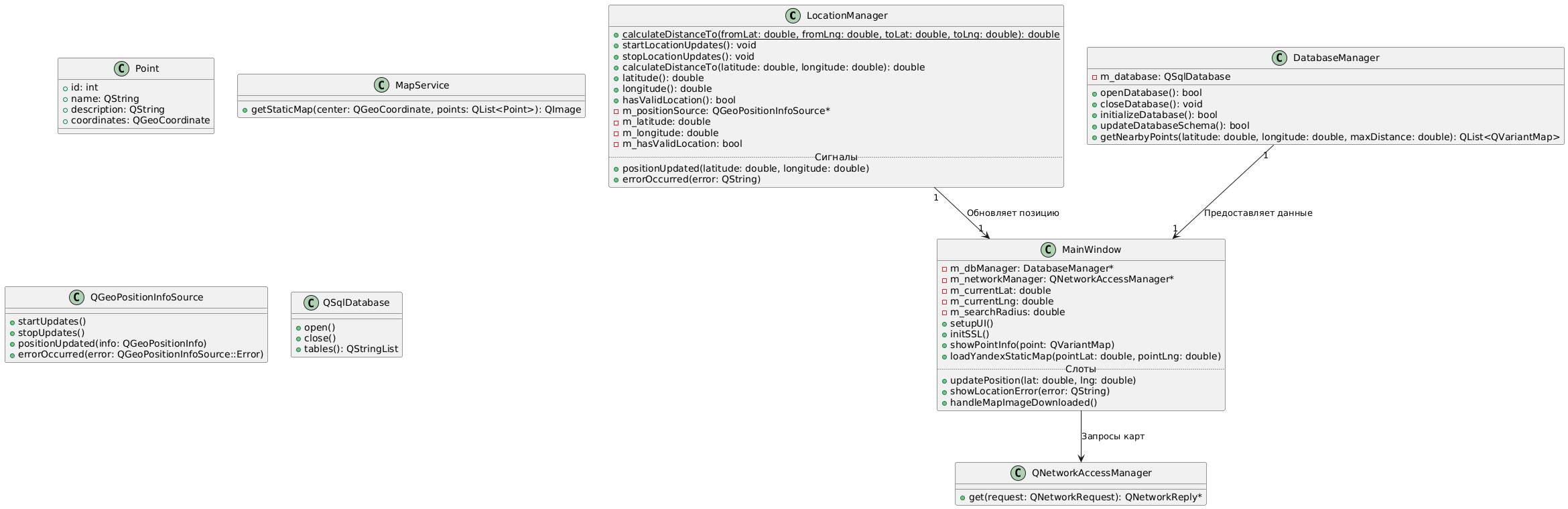


Рисунок 11 - Диаграмма классов, 2 часть