```
###############################################################################
./addroutedialog.cpp
###############################################################################
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#include "addroutedialog.h"
#include "ui_addroutedialog.h"
#include <QCompleter>

AddRouteDialog::AddRouteDialog(QWidget *parent) : QDialog(parent),
                                                  ui(new Ui::AddRouteDialog)
{
    ui->setupUi(this);
    initGUI();
}

AddRouteDialog::~AddRouteDialog()
{
    delete ui;
}

void AddRouteDialog::initGUI()
{
    DbManager database = DbManager::getInstance();
    QStringList airportList;
    for (auto &airport : database.airports)
    {
        airportList << airport.name + " (" + airport.iata + ")";
    }
    QCompleter *aiportCompleter = new QCompleter(airportList, this);
    aiportCompleter->setCaseSensitivity(Qt::CaseInsensitive);
    ui->StartAirport->setCompleter(aiportCompleter);
    ui->EndAirport->setCompleter(aiportCompleter);

    // Airlines
    QStringList airlineList;
    for (auto &airline : database.airlines)
    {
        airlineList << airline.name;
    }
    QCompleter *airlineCompleter = new QCompleter(airlineList, this);
    airlineCompleter->setCaseSensitivity(Qt::CaseInsensitive);
    ui->Airline->setCompleter(airlineCompleter);
}

void AddRouteDialog::on_buttonBox_accepted()
{
    DbManager database = DbManager::getInstance();
    QString departure = ui->StartAirport->text().simplified();
    QString destination = ui->EndAirport->text().simplified();
    QString airline = ui->Airline->text().simplified();

    departure = departure.left(departure.indexOf("(") - 1);
    destination = destination.left(destination.indexOf("(") - 1);

    int airport1 = database.getAirportId(departure);
    int airport2 = database.getAirportId(destination);
    int airlineId = database.getAirlineId(airline);

    if (airport1 == 0 || airport2 == 0)
    {
        qDebug() << "Using debug mode";
        airport1 = 4908; // Vienna
        airport2 = 3699; // Palm Spings
    }

    DbManager::getInstance().addRoute(airport1, airport2, airlineId);

    ui->StartAirport->setText("");
    ui->EndAirport->setText("");
    ui->Airline->setText("");
}

void AddRouteDialog::on_buttonBox_rejected()
{
}
###############################################################################
./drawablemapwidget.cpp
###############################################################################
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#include "drawablemapwidget.h"

#include "mainwindow.h"
#include <algorithm>

#include <cmath>

#include <QStringList>
#include <QPainter>
#include <QApplication>

DrawableMapWidget::DrawableMapWidget(QWidget *parent) : QWidget(parent)
{
    resetPic();
}

void DrawableMapWidget::paintEvent(QPaintEvent *)
{
    QPainter painter{this};
    painter.drawPixmap(0, 0, pic);
}

QPoint DrawableMapWidget::airportToImg(Airport airport)
{
    return latLonToPoint(airport.latitude, airport.longitude);
}

QPoint DrawableMapWidget::latLonToPoint(double lat, double lon)
{
    return QPoint((lon + 180.0) * 4.0,
                  (lat - 90.0) * -4.0);
}
void DrawableMapWidget::connectAirports(Airport from, Airport to, QColor color, QPainter &painter)
{
    auto fromPoint = airportToImg(from);
    auto toPoint = airportToImg(to);

    auto directDistance = abs(from.longitude - to.longitude);
    auto roundDistance = 360 - directDistance;

    if (directDistance < roundDistance)
    {
        painter.setPen(QPen{QBrush{color}, 1});
        painter.drawLine(fromPoint, toPoint);
    }
    else
    {
        painter.setPen(QPen{QBrush{color}, 1});

        auto leftSide = latLonToPoint((from.latitude + to.latitude) / 2, -180.0);
        auto rightSide = latLonToPoint((from.latitude + to.latitude) / 2, 180.0);

        if (from.longitude > 0)
        {
            painter.drawLine(fromPoint, rightSide);
            painter.drawLine(leftSide, toPoint);
        }
        else
        {
            painter.drawLine(fromPoint, leftSide);
            painter.drawLine(rightSide, toPoint);
        }
    }
}

void DrawableMapWidget::connectTheDots(std::vector<tuple<int, int>> routes, QColor color)
{
    auto airports = DbManager::getInstance().airports;
    QPainter painter{&pic};

    // Need to go twice, as lines would be over text
    for (auto route : routes)
    {
        auto fromAirport = airports[std::get<0>(route)];
        auto toAirport = airports[std::get<1>(route)];

        connectAirports(fromAirport, toAirport, color, painter);
    }

    for (auto route : routes)
    {
        auto fromAirport = airports[std::get<0>(route)];
        auto toAirport = airports[std::get<1>(route)];

        auto fromPoint = airportToImg(fromAirport);
        auto toPoint = airportToImg(toAirport);

        painter.setPen(QPen{QBrush{QColor{0, 0, 0}}, 6});
        painter.drawText(fromPoint, fromAirport.iata);
        painter.drawText(toPoint, toAirport.iata);
    }

    update();
}

void DrawableMapWidget::resetPic()
{
    QPixmap l_pic{1440, 720};
    QPainter painter(&l_pic);
    QPixmap map(":/static/static/Earthmap.jpg");
    map = map.scaled(1440, 720, Qt::KeepAspectRatioByExpanding, Qt::SmoothTransformation);
    painter.drawPixmap(0, 0, map);

    painter.setPen(QPen{QBrush{QColor{0, 255, 0, 200}}, 3});

    for (auto airport : DbManager::getInstance().airports)
    {
        painter.drawPoint(airportToImg(airport));
    }

    pic = l_pic;
}
```

```
#############################################################################
./main.cpp
#############################################################################
```

```
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

```
#############################################################################
./mainwindow.cpp
#############################################################################
```

```
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <vector>
#include <future>
#include <QCompleter>
#include <chrono>
#include "customsearchalgorithm.h"
#include "breadthfirstsearchalgorithm.h"

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent),
                                          ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    initGUI();
}

void MainWindow::initGUI()
{
    QStringList airportList;
    for (auto &airport : database.airports)
    {
        airportList << airport.name + " (" + airport.iata + ")";
    }
    QCompleter *aiportCompleter = new QCompleter(airportList, this);
    aiportCompleter->setCaseSensitivity(Qt::CaseInsensitive);
    ui->FromSearch->setCompleter(aiportCompleter);
    ui->ToSearch->setCompleter(aiportCompleter);

    // Airlines
    QStringList airlineList;
    for (auto &airline : database.airlines)
    {
        airlineList << airline.name;
    }
    QCompleter *airlineCompleter = new QCompleter(airlineList, this);
```

```cpp
    airlineCompleter->setCaseSensitivity(Qt::CaseInsensitive);
    ui->AirlineSearch->setCompleter(airlineCompleter);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::fillFlightTable(vector<vector<int>> routes, bool sort)
{
    vector<QString> flights;
    for (auto &vec : routes)
    {
        QString flight;
        for (int i{0}; i <= vec.size() - 1; i++)
        {
            if (i != 0)
            {
                flight += " -> ";
            }
            QString airportName = database.getAirportName(vec[i]);
            flight += airportName;
        }
        flights.push_back(flight);
    }
    if (sort) {
        std::sort(flights.begin(), flights.end());
    }

    for (auto &flight : flights)
    {
        ui->flighttable->addItem(new QListWidgetItem(flight));
    }
}

void MainWindow::on_pushButton_clicked()
{
    ui->flighttable->clear();
    ui->map->resetPic();

    QString departure = ui->FromSearch->text().simplified();
    QString destination = ui->ToSearch->text().simplified();
    QString airline = ui->AirlineSearch->text().simplified();

    departure = departure.left(departure.indexOf("(") - 1);
    destination = destination.left(destination.indexOf("(") - 1);

    int airport1 = database.getAirportId(departure);
    int airport2 = database.getAirportId(destination);
    int airlineId = database.getAirlineId(airline);

    if (airport1 == 0 || airport2 == 0)
    {
        qDebug() << "Using debug mode";
        airport1 = 4908; // Vienna
        airport2 = 3699; // Palm Spings
    }

    BreadthFirstSearchAlgorithm searchAlgorithm;

    auto start = std::chrono::high_resolution_clock::now();
    vector<vector<int>> routes = searchAlgorithm.getRoutes(airport1, airport2);
    auto finish = std::chrono::high_resolution_clock::now();
    auto microseconds = std::chrono::duration_cast<std::chrono::microseconds>(finish - start);

    qDebug() << "TIME: " << microseconds.count() * 0.001;
    fillFlightTable(routes, true);

    auto newRoutes = splitRoutes(routes, airlineId);
    if (airlineId == -1)
    {
        ui->map->connectTheDots(get<2>(newRoutes), QColor{82, 82, 255});
    }
    else
    {
        ui->map->connectTheDots(get<0>(newRoutes), QColor{255, 82, 82});
        ui->map->connectTheDots(get<1>(newRoutes), QColor{82, 82, 255});
        ui->map->connectTheDots(get<2>(newRoutes), QColor{82, 82, 82});
    }
}

std::tuple<vector<tuple<int, int>>, vector<tuple<int, int>>, vector<tuple<int, int>>> MainWindow::splitRoutes(vector<vector<int>> routes, int airline)
{
    vector<tuple<int, int>> airlineRoutes;
    vector<tuple<int, int>> allianceRoutes;
    vector<tuple<int, int>> otherRoutes;

    int alliance = database.airlines[airline].alliance;

    for (auto &route : routes)
    {
        for (int i{0}; i <= route.size() - 2; i++)
        {
            if (database.isConnected(route[i], route[i + 1], airline))
            {
                airlineRoutes.push_back(make_tuple(route[i], route[i + 1]));
            }
            else if (database.isConnectedViaAlliance(route[i], route[i + 1], alliance))
            {
                allianceRoutes.push_back(make_tuple(route[i], route[i + 1]));
            }
            else
            {
                otherRoutes.push_back(make_tuple(route[i], route[i + 1]));
            }
        }
    }

    return std::make_tuple(airlineRoutes, allianceRoutes, otherRoutes);
}

void MainWindow::on_actionAbout_triggered()
{
    QMessageBox qMessageBox;
    qMessageBox.setText("<h3>Name:</h3> Konstantin Lampalzer<br>"
                        "<h3>Klasse:</h3> 5BHIF");
    qMessageBox.exec();
}

void MainWindow::on_actionAdd_Route_triggered()
{
    addRouteDialog.show();
}

void MainWindow::on_multiSearchButton_clicked()
{
    QStringList airportNames = ui->multiSearch->text().split(';');
    QString airline = ui->AirlineSearch->text().simplified();
    int airlineId = database.getAirlineId(airline);

    vector<int> airports;
    for (QString airport : airportNames)
    {
        airport = airport.simplified();
        int airportId = database.getAirportId(airport);
        if (airportId == 0 || airportId == -1)
        {
            // TODO Add alert!
            qDebug() << "Error in aiport " << airport;
            return;
        }
        airports.push_back(airportId);
    }

    BreadthFirstSearchAlgorithm searchAlgorithm;
    vector<vector<int>> routes;

    for (int i{0}; i <= airports.size() - 2; i++)
    {
        qDebug() << airports[i] << " " << airports[i+1];
        auto singleSearch = searchAlgorithm.getRoutes(airports[i], airports[i+1])
;
        routes.push_back(singleSearch[0]);
    }
    auto singleSearch = searchAlgorithm.getRoutes(airports[airports.size() - 1],
airports[0]);
    routes.push_back(singleSearch[0]);

    fillFlightTable(routes, false);

    auto newRoutes = splitRoutes(routes, airlineId);
    if (airlineId == -1)
    {
        ui->map->connectTheDots(get<2>(newRoutes), QColor{82, 82, 255});
    }
    else
    {
        ui->map->connectTheDots(get<0>(newRoutes), QColor{255, 82, 82});
        ui->map->connectTheDots(get<1>(newRoutes), QColor{82, 82, 255});
        ui->map->connectTheDots(get<2>(newRoutes), QColor{82, 82, 82});
    }
}
#############################################################################
./addroutedialog.h
#############################################################################
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#ifndef ADDROUTEDIALOG_H
#define ADDROUTEDIALOG_H

#include <QDialog>
#include "dbmanager.h"

namespace Ui
{
class AddRouteDialog;
}

class AddRouteDialog : public QDialog
{
    Q_OBJECT

  public:
    explicit AddRouteDialog(QWidget *parent = 0);
    ~AddRouteDialog();

  private slots:
    void on_buttonBox_accepted();

    void on_buttonBox_rejected();

  private:
    Ui::AddRouteDialog *ui;

    void initGUI();
};

#endif // ADDROUTEDIALOG_H
#############################################################################
./airline.h
#############################################################################
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#ifndef AIRLINE_H
#define AIRLINE_H

#include <QString>

class Airline
{
public:
  int id;
  QString name;
  int alliance;

  Airline(int _id, QString _name, int _alliance)
  {
    id = _id;
    name = _name;
    alliance = _alliance;
  }

  Airline() {}
};

#endif // AIRLINE_H
#############################################################################
./airport.h
#############################################################################
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#ifndef AIRPORT_H
#define AIRPORT_H

#include <QString>

class Airport
{
public:
  int id;
  double latitude;
  double longitude;
  QString name;
  QString iata;

  Airport(int _id, double _latitude, double _longitude, QString _name, QString _iata)
  {
    id = _id;
    latitude = _latitude;
    longitude = _longitude;
    name = _name;
    iata = _iata;
  };

  Airport() {}
};

#endif // AIRPORT_H
```

```
################################################################################
./breadthfirstsearchalgorithm.h
################################################################################
```

```
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#ifndef BREADTHFIRSTSEARCHALGORITHM_H
#define BREADTHFIRSTSEARCHALGORITHM_H

#include "searchalgorithm.h"
#include "queue"
#include "map"

class BreadthFirstSearchAlgorithm : public SearchAlgorithm
{
  public:
    BreadthFirstSearchAlgorithm() {}
    std::vector<std::vector<int>> getRoutes(int start, int end)
    {
        // Possible solution:
        // Find shortest amount of hops
        // Do BFS till depth = min - 1
        //  Keep Track of the predecessors of all neighbours
        // Take pred. of end node and backtrack all routes recursively
        //  AKA. recusively go into all pred. .
        // This works, as pred. have to have some route to the start node

        std::vector<std::vector<int>> routes;
        int depth = getShortestDepth(start, end);

        auto predecessors = generatePredecessors(start, depth);
        routes = backtrack({end}, depth, end, start, predecessors);

        for (auto &route : routes)
        {
            std::reverse(route.begin(), route.end());
        }

        return routes;
    }

  private:
    std::vector<std::vector<int>> backtrack(std::vector<int> prev, int depth, int start, int end, std::map<int, std::vector<int>> &predecessors)
    {
        std::vector<std::vector<int>> ret;

        if (depth == 0)
        {
            if (start == end)
            {
                std::vector<int> newPrev = prev;
                //newPrev.push_back(end);
                ret.push_back(newPrev);
            }
        }
        else
        {
            auto nbs = predecessors[start];
            for (auto &airport : nbs)
            {
                if (std::find(prev.begin(), prev.end(), airport) != prev.end())
                {
                    continue;
                }

                std::vector<int> newPrev = prev;
                newPrev.push_back(airport);

                auto toConcat = backtrack(newPrev, depth - 1, airport, end, predecessors);
                if (toConcat.size() != 0)
                {
                    ret.insert(ret.end(), toConcat.begin(), toConcat.end());
                }
            }
        }
        return ret;
    }

    std::map<int, std::vector<int>> generatePredecessors(int start, int goalDepth)
    {
        int depth{0};
        std::map<int, std::vector<int>> result;
        std::queue<int> currentLayer;
        std::queue<int> nextLayer;
        std::map<int, bool> visited;

        currentLayer.push(start);
        int next = start;

        while (depth != goalDepth)
        {
            if (!visited[next])
            {
                visited[next] = true;
                for (auto &airport : database.getNeighbours(next))
                {
                    if (visited[airport] != true)
                    {
                        nextLayer.push(airport);
                        result[airport].push_back(next);
                    }
                }
            }

            currentLayer.pop();
            if (currentLayer.empty())
            {
                depth += 1;
                currentLayer = nextLayer;
                if (currentLayer.empty())
                {
                    return result;
                }
            }
            next = currentLayer.front();
        }

        return result;
    }

    int getShortestDepth(int start, int end)
    {
        int depth{0};
        std::queue<int> currentLayer;
        std::queue<int> nextLayer;
        std::map<int, bool> visited;

        currentLayer.push(start);
        visited[start] = true;
        int next = start;
        while (next != end)
        {
            for (auto &airport : database.getNeighbours(next))
            {
                if (visited[airport] != true)
                {
                    visited[airport] = true;
                    nextLayer.push(airport);
                }
            }

            currentLayer.pop();
            if (currentLayer.empty())
            {
                depth += 1;
                currentLayer = nextLayer;
                if (currentLayer.empty())
                {
                    return -1;
                }
            }
            next = currentLayer.front();
        }
        return depth;
    }
};

#endif // BREADTHFIRSTSEARCHALGORITHM_H
```

```
#############################################################################
./customsearchalgorithm.h
#############################################################################
```

```
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#ifndef CUSTOMSEARCHALGORITHM_H
#define CUSTOMSEARCHALGORITHM_H

#include "searchalgorithm.h"
#include <future>

class CustomSearchAlgorithm : public SearchAlgorithm
{
  public:
    CustomSearchAlgorithm(){};

    std::vector<std::vector<int>> getRoutes(int start, int end)
    {
        std::vector<std::vector<int>> routes;
        int depth{0};
        do
        {
            routes = getRoutesInternal({start}, depth, start, end);
            qDebug() << depth;
            depth += 1;
        } while (routes.size() == 0 && depth <= 4);
        return routes;
    }

  private:
    std::vector<std::vector<int>> getRoutesInternal(std::vector<int> prev, int depth, int start, int end)
    {
        std::vector<std::vector<int>> ret;

        if (depth == 0)
        {
            if (database.isConnected(start, end))
            {
                std::vector<int> newPrev = prev;
                newPrev.push_back(end);
                ret.push_back(newPrev);
            }
        }
        else
        {
            auto nbs = database.getNeighbours(start);
            std::vector<std::future<std::vector<std::vector<int>>>> futs;

            for (auto &airport : nbs)
            {
                if (std::find(prev.begin(), prev.end(), airport) != prev.end())
                {
                    continue;
                }

                std::vector<int> newPrev = prev;
                newPrev.push_back(airport);

                if (depth == 4)
                {
                    futs.push_back(std::async(&CustomSearchAlgorithm::getRoutesInternal, this, newPrev, depth - 1, airport, end));
                }
                else
                {
                    auto toConcat = getRoutesInternal(newPrev, depth - 1, airport, end);
                    if (toConcat.size() != 0)
                    {
                        ret.insert(ret.end(), toConcat.begin(), toConcat.end());
                    }
                }
            }

            for (auto &fut : futs)
            {
                auto toConcat = fut.get();
                ret.insert(ret.end(), toConcat.begin(), toConcat.end());
            }
        }
        return ret;
    }
};

#endif // CUSTOMSEARCHALGORITHM_H
```
```
################################################################################
./dbmanager.h
################################################################################
```
```
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#ifndef DBMANAGER_H
#define DBMANAGER_H

#include <QString>
#include <QSqlDatabase>
#include <QSqlQuery>
#include <QSqlRecord>
#include <QtDebug>
#include <vector>
#include <route.h>
#include <airport.h>
#include <airline.h>
#include <unordered_set>

class DbManager
{
  public:
    std::vector<std::vector<Route>> routes;
    std::vector<std::map<int, bool>> connections;

    std::vector<Airport> airports;
    std::vector<Airline> airlines;

    static DbManager &getInstance()
    {
        static DbManager instance;
        return instance;
    }

    int getAirportCount()
    {
        QSqlQuery query("select count(*) as amount from Airport");
        int amount = query.record().indexOf("amount");

        while (query.next())
        {
            return query.value(amount).toInt();
        }
    }

    int getAirlineCount()
    {
        QSqlQuery query("select count(*) as amount from Airline");
        int amount = query.record().indexOf("amount");

        while (query.next())
        {
            return query.value(amount).toInt();
        }
    }

    int getAirportId(QString name)
    {
        std::vector<Airport> possibleAirports;
        for (auto &airport : airports)
        {
            if (airport.name.contains(name) || airport.iata.contains(name))
            {
                possibleAirports.push_back(airport);
            }
        }

        if (possibleAirports.size() != 1)
        {
            return -1;
        } else {
            return possibleAirports[0].id;
        }
    }

    int getAirlineId(QString name)
    {
        for (auto &airline : airlines)
        {
            if (airline.name == name && airline.name != "")
            {
                return airline.id;
            }
        }
        return -1;
    }

    bool isConnected(int id1, int id2)
    {
        if (connections[id1][id2] == true)
        {
            return true;
        }
        return false;
    }

    bool isConnected(int id1, int id2, int airline)
    {
        for (auto &route : routes[id1])
        {
            if (route.end == id2 && route.airline == airline)
            {
                return true;
            }
        }
        return false;
    }

    bool isConnectedViaAlliance(int id1, int id2, int alliance)
    {
        for (auto &route : routes[id1])
        {
            if (route.end == id2 && airlines[route.airline].alliance == alliance)
            {
                return true;
            }
        }
        return false;
    }

    QString getAirportName(int id)
    {
        return airports[id].name;
    }

    std::vector<int> getNeighbours(int id)
    {
        std::unordered_set<int> set;
        std::vector<int> results;

        for (auto &route : routes[id])
        {
            set.insert(route.end);
        }
        results.assign(set.begin(), set.end());

        return results;
    }

    void addRoute(int startAirport, int endAirport, int airline)
    {
        routes[startAirport].push_back(Route(startAirport, endAirport, airline));
        connections[startAirport][endAirport] = true;

        //QSqlQuery query;
        //query.prepare("insert into routes values (?, ?, ?)");
        //query.bindValue(startAirport, endAirport, airline);
        //query.exec();
    }

  private:
    QSqlDatabase m_db;

    DbManager()
    {
        m_db = QSqlDatabase::addDatabase("QSQLITE");
        m_db.setDatabaseName("./AirlineRoutes.db");

        if (!m_db.open())
        {
            qDebug() << "Error: connection with database fail";
        }
        else
        {
            qDebug() << "Database: connection ok";
        }

        loadRoutes();
        loadAirports();
        loadAirlines();
    }

    void loadRoutes()
    {
        routes.resize(getAirportCount());
        connections.resize(getAirportCount());

        QSqlQuery query;
        query.prepare("select * from route order by airport1");
        query.exec();

        auto a1Col = query.record().indexOf("airport1");
        auto a2Col = query.record().indexOf("airport2");
        auto airlineCol = query.record().indexOf("airline");

        while (query.next())
        {
            int a1{query.value(a1Col).toInt()};
            int a2{query.value(a2Col).toInt()};
            routes[a1].push_back(Route(a1, a2, query.value(airlineCol).toInt()));
            connections[a1][a2] = true;
        }
    }

    void loadAirports()
    {
        airports.resize(getAirportCount() + 1);
        QSqlQuery query;
        query.prepare("select * from airport");
        query.exec();

        auto idCol = query.record().indexOf("id");
        auto latitudeCol = query.record().indexOf("latitude");
        auto longitudeCol = query.record().indexOf("longitude");
        auto nameCol = query.record().indexOf("name");
        auto iataCol = query.record().indexOf("iata");

        while (query.next())
        {
            auto id{query.value(idCol).toInt()};
            airports.at(id) = Airport(id,
                                      query.value(latitudeCol).toDouble(), query.value(longitudeCol).toDouble(),
                                      query.value(nameCol).toString().simplified(), query.value(iataCol).toString());
        }
    }

    void loadAirlines()
    {
        airlines.resize(getAirlineCount() + 1);
        QSqlQuery query;
        query.prepare("select * from airline");
        query.exec();

        auto idCol = query.record().indexOf("id");
        auto nameCol = query.record().indexOf("name");
        auto allianceCol = query.record().indexOf("alliance");

        while (query.next())
        {
            auto id{query.value(idCol).toInt()};
            airlines.at(id) = Airline(id, query.value(nameCol).toString(), query.value(allianceCol).toInt());
        }
    }
};

#endif // DBMANAGER_H
#############################################################################
./dijkstrasearchalgorithm.h
#############################################################################
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#ifndef DIJKSTRASEARCHALGORITHM_H
#define DIJKSTRASEARCHALGORITHM_H

#include "searchalgorithm.h"
#include <map>

class DijkstraSearchAlgorithm : public SearchAlgorithm
{
    DijkstraSearchAlgorithm() {}
    std::vector<std::vector<int>> getRoutes(int start, int end)
    {
    }
}
```

```
#endif // DIJKSTRASEARCHALGORITHM_H
#############################################################################
./drawablemapwidget.h
#############################################################################
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#ifndef DRAWABLEMAPWIDGET_H
#define DRAWABLEMAPWIDGET_H

#include <QWidget>
#include "dbmanager.h"
#include <QApplication>
#include <QPoint>

class DrawableMapWidget : public QWidget
{
  Q_OBJECT
public:
  explicit DrawableMapWidget(QWidget *parent = nullptr);

  void paintEvent(QPaintEvent *e);
  QPoint airportToImg(Airport airport);
  QPoint latLonToPoint(double lat, double lon);

  void connectAirports(Airport from, Airport to, QColor color, QPainter &painter)
;
  void connectTheDots(std::vector<std::tuple<int, int>> routes, QColor color);

  void resetPic();
signals:

public slots:

private:
  QPixmap pic;
};

#endif // DRAWABLEMAPWIDGET_H
#############################################################################
./mainwindow.h
#############################################################################
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QFont>
#include "dbmanager.h"
#include <QMessageBox>
#include <vector>
#include <QTableWidgetItem>
#include <memory>
#include "addroutedialog.h"

using namespace std;

namespace Ui
{
class MainWindow;
}

class MainWindow : public QMainWindow
{
  Q_OBJECT

public:
  explicit MainWindow(QWidget *parent = nullptr);
  void initGUI();
  ~MainWindow();

private slots:
  void on_pushButton_clicked();

  void on_actionAbout_triggered();

  void on_actionAdd_Route_triggered();

  void on_multiSearchButton_clicked();

private:
  Ui::MainWindow *ui;
  AddRouteDialog addRouteDialog;
  DbManager database = DbManager::getInstance();

  QFont titleFont{"Helvetica", 18, QFont::Bold};
  QFont standardFont{"Helvetica", 18};

  void fillFlightTable(vector<vector<int>> routes, bool sort);
  std::tuple<vector<tuple<int, int>>, vector<tuple<int, int>>, vector<tuple<int,
int>>> splitRoutes(vector<vector<int>> routes, int airline);
};

#endif // MAINWINDOW_H
#############################################################################
./route.h
#############################################################################
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#ifndef ROUTE_H
#define ROUTE_H

#include <string>

class Route
{
public:
  int start;
  int end;
  int airline;

  Route(int _start, int _end, int _airline)
  {
    start = _start;
    end = _end;
    airline = _airline;
  }

  Route() {}
};
#endif // ROUTE_H
#############################################################################
```

./searchalgorithm.h
##
```
/*
 * Author: Lampalzer Konstantin
 * Class: 5BHIF
 * Date: 16.02.2019
 */

#ifndef SEARCHALGORITHM_H
#define SEARCHALGORITHM_H

#include <vector>
#include "dbmanager.h"

class SearchAlgorithm
{
  public:
    DbManager database = DbManager::getInstance();
    virtual std::vector<std::vector<int>> getRoutes(int start, int end) = 0;
};

#endif // SEARCHALGORITHM_H
```
##
./addroutedialog.ui
##
```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>AddRouteDialog</class>
 <widget class="QDialog" name="AddRouteDialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>400</width>
    <height>300</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Dialog</string>
  </property>
  <layout class="QGridLayout" name="gridLayout">
   <item row="0" column="0">
    <layout class="QFormLayout" name="formLayout">
     <item row="0" column="0">
      <widget class="QLabel" name="label">
       <property name="text">
        <string>Start Flughafen</string>
       </property>
      </widget>
     </item>
     <item row="1" column="0">
      <widget class="QLabel" name="label_2">
       <property name="text">
        <string>End Flughafen</string>
       </property>
      </widget>
     </item>
     <item row="0" column="1">
      <widget class="QLineEdit" name="StartAirport"/>
     </item>
     <item row="1" column="1">
      <widget class="QLineEdit" name="EndAirport"/>
     </item>
     <item row="2" column="0">
      <widget class="QLabel" name="label_3">
       <property name="text">
        <string>Fluglinie</string>
       </property>
      </widget>
     </item>
     <item row="2" column="1">
      <widget class="QLineEdit" name="Airline"/>
     </item>
    </layout>
   </item>
   <item row="1" column="0">
    <widget class="QDialogButtonBox" name="buttonBox">
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
     <property name="standardButtons">
      <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
     </property>
    </widget>
   </item>
  </layout>
 </widget>
 <resources/>
 <connections>
  <connection>
   <sender>buttonBox</sender>
   <signal>accepted()</signal>
   <receiver>AddRouteDialog</receiver>
   <slot>accept()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>248</x>
     <y>254</y>
    </hint>
    <hint type="destinationlabel">
     <x>157</x>
     <y>274</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>buttonBox</sender>
   <signal>rejected()</signal>
   <receiver>AddRouteDialog</receiver>
   <slot>reject()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>316</x>
     <y>260</y>
    </hint>
    <hint type="destinationlabel">
     <x>286</x>
     <y>274</y>
    </hint>
   </hints>
  </connection>
 </connections>
</ui>
```
##
./mainwindow.ui
##
```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
```

```xml
    <width>1720</width>
    <height>1100</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="minimumSize">
   <size>
    <width>1720</width>
    <height>1100</height>
   </size>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <property name="sizePolicy">
    <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
     <horstretch>0</horstretch>
     <verstretch>0</verstretch>
    </sizepolicy>
   </property>
   <layout class="QGridLayout" name="gridLayout">
    <item row="0" column="0">
     <layout class="QHBoxLayout" name="horizontalLayout" stretch="0,720">
      <item>
       <layout class="QFormLayout" name="formLayout">
        <item row="2" column="0">
         <widget class="QLabel" name="fromLabel">
          <property name="minimumSize">
           <size>
            <width>30</width>
            <height>0</height>
           </size>
          </property>
          <property name="text">
           <string>From:</string>
          </property>
         </widget>
        </item>
        <item row="2" column="1">
         <widget class="QLineEdit" name="FromSearch">
          <property name="text">
           <string/>
          </property>
         </widget>
        </item>
        <item row="3" column="0">
         <widget class="QLabel" name="toLabel">
          <property name="minimumSize">
           <size>
            <width>30</width>
            <height>0</height>
           </size>
          </property>
          <property name="text">
           <string>To:</string>
          </property>
         </widget>
        </item>
        <item row="3" column="1">
         <widget class="QLineEdit" name="ToSearch">
          <property name="text">
           <string/>
          </property>
         </widget>
        </item>
        <item row="4" column="0">
         <widget class="QLabel" name="airlineLabel">
          <property name="minimumSize">
           <size>
            <width>30</width>
            <height>0</height>
           </size>
          </property>
          <property name="text">
           <string>Airline:</string>
          </property>
         </widget>
        </item>
        <item row="4" column="1">
         <widget class="QLineEdit" name="AirlineSearch">
          <property name="text">
           <string/>
          </property>
         </widget>
        </item>
        <item row="5" column="0">
         <widget class="QLabel" name="label">
          <property name="text">
           <string/>
          </property>
         </widget>
        </item>
        <item row="5" column="1">
         <widget class="QPushButton" name="pushButton">
          <property name="text">
           <string>Search</string>
          </property>
         </widget>
        </item>
        <item row="7" column="0">
         <widget class="QLabel" name="label_2">
          <property name="text">
           <string>Multisearch</string>
          </property>
         </widget>
        </item>
        <item row="7" column="1">
         <widget class="QLineEdit" name="multiSearch">
          <property name="text">
           <string>VIE; JFK</string>
          </property>
         </widget>
        </item>
        <item row="8" column="1">
         <widget class="QPushButton" name="multiSearchButton">
          <property name="text">
           <string>Multi-search</string>
          </property>
         </widget>
        </item>
        <item row="6" column="0" colspan="2">
         <spacer name="horizontalSpacer">
          <property name="orientation">
           <enum>Qt::Horizontal</enum>
          </property>
          <property name="sizeHint" stdset="0">
           <size>
            <width>40</width>
```

```
             <height>20</height>
            </size>
           </property>
          </spacer>
         </item>
        </layout>
       </item>
       <item>
        <layout class="QVBoxLayout" name="verticalLayout_2" stretch="0,0">
         <item>
          <widget class="DrawableMapWidget" name="map" native="true">
           <property name="minimumSize">
            <size>
             <width>1440</width>
             <height>720</height>
            </size>
           </property>
          </widget>
         </item>
         <item>
          <widget class="QListWidget" name="flighttable">
           <property name="minimumSize">
            <size>
             <width>720</width>
             <height>360</height>
            </size>
           </property>
          </widget>
         </item>
        </layout>
       </item>
      </layout>
     </item>
    </layout>
   </widget>
   <widget class="QMenuBar" name="menuBar">
    <property name="geometry">
     <rect>
      <x>0</x>
      <y>0</y>
      <width>1720</width>
      <height>22</height>
     </rect>
    </property>
    <widget class="QMenu" name="menuMenu">
     <property name="title">
      <string>Menu</string>
     </property>
     <addaction name="actionAbout"/>
     <addaction name="actionAdd_Route"/>
    </widget>
    <addaction name="menuMenu"/>
   </widget>
   <action name="actionAbout">
    <property name="text">
     <string>About</string>
    </property>
   </action>
   <action name="actionAdd_Route">
    <property name="text">
     <string>Add Route</string>
    </property>
   </action>
  </widget>
  <layoutdefault spacing="6" margin="11"/>
  <customwidgets>
   <customwidget>
    <class>DrawableMapWidget</class>
    <extends>QWidget</extends>
    <header>drawablemapwidget.h</header>
    <container>1</container>
   </customwidget>
  </customwidgets>
  <resources/>
  <connections/>
 </ui>
```