controller.h

```
#ifndef CONTROLLER_H
#define CONTROLLER_H

#include <QMainWindow>
#include <QtSql>
#include <QSqlDatabase>
#include <QSqlDriver>
#include <QSqlQuery>
#include <QSqlError>
#include <QtDebug>
#include <QMessageBox>
#include <QKeyEvent>
#include <QFileDialog>
#include<QItemDelegate>
#include<QSpinBox>
#include<QSize>
#include<QTableWidgetItem>
#include <QSpinBox>
#include "trip.h"
using namespace std;

class Controller : public QObject
{
  Q_OBJECT

public:
  explicit Controller(QObject *parent = nullptr);
  ~Controller();

  QSqlQueryModel *getDistancesQueryModel(QString query);
  QSqlQueryModel *getFoodsQueryModel(QString query);
  void editFoodCostQuery(QString city, QString food, double cost);
  void deleteFoodQuery(QString city, QString food, double cost);
  void addFoodQuery(QString city, QString food, double cost);
  void uploadCitiesFile();
  void uploadFoodsFile();
  void getCityCount();

  // For planning trips

  int cityCount;
```

```
    void createTripList();
    void createCustomTripList();
    void resetTripList();
    void displayTripList();
    void createTrip(QString startCity, int numberOfCities);
    void resetTrip();
    void displayTrip();
    void createFoodList();

    QVector<Trip*> tripList;
    QVector<Trip*> completedTripList;
    QVector<food*> foodList;
    QVector<QString> customTripListCities;

private:
    QSqlDatabase m_database;
};

#endif // CONTROLLER_H
```

food.h

```
#ifndef FOOD_H
#define FOOD_H

#include <QObject>
#include <QString>

class food : public QObject
{
    Q_OBJECT

private:
    QString city;
    QString name;
    double cost;

public:
    explicit food(QObject *parent = nullptr);
    QString getCity() const;
    QString getName() const;
    double getCost() const;
    void setCity(const QString &item);
    void setName(const QString &item);
```

```
    void setCost(double &item);
};

#endif // FOOD_H
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "controller.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(Controller *controller, QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_clear_pushButton_clicked();

    void on_login_pushButton_clicked();

    void on_viewCitiesAndFoods_pushButton_clicked();

    void fillCitiesComboBoxes();

    void on_returnToUserPage_pushButton_clicked();

    void on_userLogout_pushButton_clicked();

    void on_adminEdit_pushButton_clicked();

    void on_adminViewFoods_tableView_activated(const QModelIndex &index);

    void on_contactUs_pushButton_clicked();

    void on_submit_pushButton_clicked();
```

```
void on_return_pushButton_clicked();

void on_userLogout_Pushbutton_clicked();

void on_adminLogout_pushButton_clicked();

void on_userSelectCities_comboBox_currentTextChanged(const QString &arg1);

void on_adminChooseCities_comboBox_currentTextChanged(const QString &arg1);

void on_editPrice_pushButton_clicked();

void resetAdminEditPage();

void on_returnToAdminPage_pushButton_clicked();

void on_deleteFood_pushButton_clicked();

void on_addNewFood_pushButton_clicked();

void on_resetEditPage_pushButton_clicked();

void on_adminUpload_pushButton_clicked();

void on_returnToAdminPage_pushButton2_clicked();

void on_adminUploadChooseCities_comboBox_currentTextChanged(const QString &arg1);

void on_adminUploadCities_pushButton_clicked();

void on_adminUploadFoods_pushButton_clicked();

void on_planTrip_pushButton_clicked();

void on_parisTrip_pushButton_clicked();

void on_planTripPageBack_pushButton_clicked();

void on_pickTripBack_pushButton_clicked();

void on_autoPlanner_pushButton_clicked();

void on_autoTripReset_pushButton_clicked();
```

```
    void on_purchaseFoods_pushButton_clicked();

    void on_customPlanner_pushButton_clicked();

    void on_customTripBack_pushButton_clicked();

    void on_customTripSelect_tableView_doubleClicked(const QModelIndex &index);

    void on_customTripReset_pushButton_clicked();

    void on_createCustomTrip_pushButton_clicked();

    void on_customPurchaseFoods_pushButton_clicked();

private:
    Ui::MainWindow *ui;
    Controller *m_controller;
};

#endif // MAINWINDOW_H
```

trip.h

```
#ifndef TRIP_H
#define TRIP_H

#include <QObject>
#include <QString>
#include "food.h"

class Trip : public QObject
{
    Q_OBJECT

signals:

public slots:

private:
    QString startCity;
    QString endCity;
    int distance;
```

```
public:
    explicit Trip(QObject *parent = nullptr);
    QString getStartCity() const;
    QString getEndCity() const;
    int getDistance() const;
    void setStartCity(const QString &item);
    void setEndCity(const QString &item);
    void setDistance(int);
};

#endif // TRIP_H
```

controller.cpp

```
#include "controller.h"

Controller::Controller(QObject *parent) : QObject(parent)
{
    m_database = QSqlDatabase::addDatabase("QSQLITE");
    QString path = "..//CS1D-European-Vacation-Project.db";
    m_database.setDatabaseName(path);

    if (!m_database.open())
        qDebug() << "PROBLEM OPENING DATABASE.";
    else
        qDebug() << "DATABASE OPENED.";
}

Controller::~Controller()
{
    m_database.close();
}

QSqlQueryModel *Controller::getDistancesQueryModel(QString query)
{
    QSqlQueryModel* model = new QSqlQueryModel();
    model->setQuery(query);

    if (model->lastError().isValid())
        qDebug() << model->lastError();
    else
        qDebug() << model;

    return model;
```

```
}

QSqlQueryModel *Controller::getFoodsQueryModel(QString query)
{
  QSqlQueryModel* model = new QSqlQueryModel();
  model->setQuery(query);

  if (model->lastError().isValid())
     qDebug() << model->lastError();
  else
     qDebug() << model;

  return model;
}

void Controller::editFoodCostQuery(QString city, QString food, double cost)
{
  QString costAsString = "$" + QString::number(cost);
  QSqlQuery qry;

  qry.prepare("UPDATE [Foods] set   "
              "City       = ?,      "
              "Food       = ?,      "
              "Cost       = ?       "
              "where Food = ? and   "
              "City       = ?;      ");

  qry.addBindValue(city);
  qry.addBindValue(food);
  qry.addBindValue(costAsString);
  qry.addBindValue(food);
  qry.addBindValue(city);

  if (!qry.exec())
  {
     qDebug() << "ERROR IN editFoodPriceQueryModel(QString city, QString food, double
price)!!!!!!!!!!";
     qDebug() << food << " from " << city << "not updated to " << costAsString << "!";
  }
  else
     qDebug() << food << " from " << city << "updated to " << costAsString << "!";

  qry.clear();
}
```

```
void Controller::deleteFoodQuery(QString city, QString food, double cost)
{
  QString costAsString = "$" + QString::number(cost);
  QSqlQuery qry;

  qry.prepare("DELETE FROM Foods WHERE City = '"+city+"' AND Food = '"+food+"' AND
Cost = '"+costAsString+"';");

  if (!qry.exec())
    qDebug() << "ERROR IN deleteFoodQuery(QString city, QString food, double cost)!!!!!!!!!!!";
  else
    qDebug() << food << " FROM " << city << " REMOVED!";

  qry.clear();
}

void Controller::addFoodQuery(QString city, QString food, double cost)
{
  QString costAsString = "$" + QString::number(cost);
  QSqlQuery qry;

  qry.prepare("INSERT INTO Foods (City, Food, Cost) "
              "VALUES (?, ?, ?);");
  qry.addBindValue(city);
  qry.addBindValue(food);
  qry.addBindValue(costAsString);

  if (!qry.exec())
    qDebug() << "ERROR IN addFoodQuery(QString city, QString food, double cost)!!!!!!!!!!!!!!";
  else
    qDebug() << "ADDED " << city << ", " << food << ", " << costAsString;

  qry.clear();
}

void Controller::uploadCitiesFile()
{
  QString fileName = QFileDialog::getOpenFileName(nullptr, tr("Open File"),
"/home/CS1D-Project", tr("Text Files (*.txt)"));
  QFile file(fileName);

  if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
    qDebug() << "Error reading file.";
```

```
    else
    {
        QTextStream in(&file);

        while (!in.atEnd())
        {
            QSqlQuery qry;
            QString startCity = in.readLine();
            QString endCity = in.readLine();
            QString distance = in.readLine();

            qry.prepare("INSERT INTO Distances (StartCity, EndCity, Distance)"
                        "VALUES (?, ?, ?);");

            qry.addBindValue(startCity);
            qry.addBindValue(endCity);
            qry.addBindValue(distance);

            if (!qry.exec())
                qDebug() << "ERROR READING .TXT ON " << startCity << ", " << endCity << ", " <<
distance;
            else
            {
                qDebug() << "CITY DATA APPENDED TO  .DB: " << startCity << ", " << endCity << ", "
<< distance;
                qry.clear();

                Trip* entry = new Trip();
                entry->setStartCity(startCity);
                entry->setEndCity(endCity);
                entry->setDistance(distance.toInt());
                this->tripList.append(entry);
            }
        }
    }
}

void Controller::uploadFoodsFile()
{
    QString fileName = QFileDialog::getOpenFileName(nullptr, tr("Open File"),
"/home/CS1D-Project", tr("Text Files (*.txt)"));
    QFile file(fileName);

    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
```

```
        qDebug() << "Error reading file.";
    else
    {
        QTextStream in(&file);

        while (!in.atEnd())
        {
            QSqlQuery qry;
            QString city = in.readLine();
            QString food = in.readLine();
            QString cost = in.readLine();

            qry.prepare("INSERT INTO Foods (City, Food, Cost)"
                        "VALUES (?, ?, ?);");

            qry.addBindValue(city);
            qry.addBindValue(food);
            qry.addBindValue(cost);

            if (!qry.exec())
                qDebug() << "ERROR READING .TXT ON " << city << ", " << food << ", " << cost;
            else
            {
                qDebug() << "FOOD DATA APPENDED TO  .DB: " << city << ", " << food << ", " <<
cost;
                qry.clear();
            }
        }
    }
}

void Controller::getCityCount()
{
    QSqlQuery qry;
    qry.prepare("SELECT COUNT(DISTINCT StartCity) FROM Distances");

    if (!qry.exec())
        qDebug() << "ERROR GETTING CITY COUNT";
    else
    {
        qry.next();
        cityCount = qry.value(0).toInt() - 1;
    }
}
```

```
void Controller::createTripList()
{
   QSqlTableModel model;
   model.setTable("Distances");
   model.select();

   for (int i = 0; i < model.rowCount(); i++)
   {
      Trip* entry = new Trip();
      entry->setStartCity(model.record(i).value("StartCity").toString());
      entry->setEndCity(model.record(i).value("EndCity").toString());
      entry->setDistance(model.record(i).value("Distance").toInt());

      this->tripList.append(entry);
   }
}

void Controller::createCustomTripList()
{
   QSqlTableModel model;
   model.setTable("Distances");
   model.select();

   for (int i = 0; i < customTripListCities.size(); i++)
   {
      for (int j = 0; j < model.rowCount(); j++)
      {
         if (model.record(j).value("StartCity").toString() == customTripListCities[i])
         {
            for (int k = 0; k < customTripListCities.size(); k++)
            {
               if (model.record(j).value("EndCity").toString() == customTripListCities[k])
               {
                  Trip* entry = new Trip();

                  entry->setStartCity(model.record(j).value("StartCity").toString());
                  entry->setEndCity(model.record(j).value("EndCity").toString());
                  entry->setDistance(model.record(j).value("Distance").toInt());

                  this->tripList.append(entry);
               }
            }
         }
```

```
    }
  }
}

void Controller::resetTripList()
{
  for (int i = 0; i < tripList.size(); i++)
  {
    delete tripList[i];
  }

  tripList.clear();

  customTripListCities.clear();
}

void Controller::displayTripList()
{
  for (int i = 0; i < tripList.size(); i++)
  {
    qDebug() << tripList[i]->getStartCity() << ", " << tripList[i]->getEndCity() << ", " <<
tripList[i]->getDistance();
  }
}

void Controller::createTrip(QString startCity, int numberOfCities)
{
  QString tempEndCity = "DEFAULT";
  int tempDistance = 99999;
  int distance = 99999;   // RE-INITIALIZES DISTANCE FOR DISTANCE SEARCH

  if (completedTripList.size() == numberOfCities)    // ENDS WHEN THE NUMBER OF CITIES
IS REACHED.
    return;

  else
  {
    for (int i = 0; i < tripList.size(); i++) // TRAVERSES tripList AND SEARCHES FOR THE CITY
WITH THE LOWEST DISTANCE RELATIVE TO startCity
    {
      if (tripList[i]->getStartCity() == startCity) // FINDS THE CITY WITH THE LOWEST
DISTANCE AND SAVES endCity AND distance
      {
        if (tripList[i]->getDistance() < distance)
```

```
            {
                tempEndCity = tripList[i]->getEndCity();
                tempDistance = tripList[i]->getDistance();
                distance = tripList[i]->getDistance();
            }

            tripList[i]->setStartCity("NULL");
            tripList[i]->setEndCity("NULL");
            tripList[i]->setDistance(100000);
            continue;
        }
    }

    for (int i = 0; i < tripList.size(); i++) // TRAVERSES tripList AND SEARCHES FOR THE CITY
WITH THE LOWEST DISTANCE RELATIVE TO startCity
    {
        if (tripList[i]->getEndCity() == startCity)
        {
            tripList[i]->setStartCity("NULL");
            tripList[i]->setEndCity("NULL");
            tripList[i]->setDistance(100000);
            continue;
        }
    }

    Trip *entry = new Trip();
    entry->setStartCity(startCity);
    entry->setEndCity(tempEndCity);
    entry->setDistance(tempDistance);

    this->completedTripList.append(entry);

    createTrip(tempEndCity, numberOfCities);
  }
}

void Controller::resetTrip()
{
  for (int i = 0; i < completedTripList.size(); i++)
  {
    delete completedTripList[i];
  }

  completedTripList.clear();
```

```
    for (int i = 0; i < foodList.size(); i++)
    {
        delete foodList[i];
    }

    foodList.clear();
}

void Controller::displayTrip()
{
    qDebug() << "++++++++++FINAL TRIP++++++++++++++++++";

    for (int i = 0; i < completedTripList.size(); i++)
    {
        qDebug() << completedTripList[i]->getStartCity() << "--->" <<
completedTripList[i]->getDistance() << "--->" << completedTripList[i]->getEndCity();
    }
}

void Controller::createFoodList()
{
    QSqlTableModel model;
    model.setTable("Foods");
    model.select();

    for (int i = 0; i < completedTripList.size(); i++)
    {
        for (int j = 0; j < model.rowCount(); j++)
        {
            if (model.record(j).value("City").toString() == completedTripList[i]->getStartCity())
            {
                food* entry = new food();
                entry->setCity(model.record(j).value("City").toString());
                entry->setName(model.record(j).value("Food").toString());

                QString costAsString = model.record(j).value("Cost").toString();
                costAsString.remove(0,1);
                double costAsDouble = costAsString.toDouble();

                entry->setCost(costAsDouble);

                this->foodList.append(entry);
            }
```

```
        }
    }

    for (int i = 0; i < model.rowCount(); i++)
    {
        if (model.record(i).value("City").toString() == completedTripList.back()->getEndCity())
        {
            food* entry = new food();
            entry->setCity(model.record(i).value("City").toString());
            entry->setName(model.record(i).value("Food").toString());

            QString costAsString = model.record(i).value("Cost").toString();
            costAsString.remove(0,1);
            double costAsDouble = costAsString.toDouble();

            entry->setCost(costAsDouble);

            this->foodList.append(entry);
        }
    }
}
```

food.cpp

```
#include "food.h"

food::food(QObject * parent) : QObject(parent) {}

QString food::getCity() const
{
    return city;
}

QString food::getName() const
{
    return name;
}

double food::getCost() const
{
    return cost;
}

void food::setCity(const QString &item)
```

```
{
  city = item;
}

void food::setName(const QString &item)
{
  name = item;
}

void food::setCost(double &item)
{
  cost = item;
}
```

main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  Controller controller;
  MainWindow w(&controller);
  w.show();
  return a.exec();
}
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QPixmap>
#include <QPalette>

MainWindow::MainWindow(Controller *controller, QWidget *parent)
  : QMainWindow(parent)
  , ui(new Ui::MainWindow),
    m_controller(controller)
{
  ui->setupUi(this);
  ui->stackedWidget->setCurrentWidget(ui->login_page);
  m_controller->getCityCount();
```

```
    fillCitiesComboBoxes();

    /*QPixMap and QPalette used to set the background.*/
    QPixmap background(":/images/backgroundwhite.png");
    background = background.scaled(this->size(), Qt::IgnoreAspectRatio);
    QPalette palette;
    palette.setBrush(QPalette::Window, background);
    this->setPalette(palette);
}

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

void MainWindow::fillCitiesComboBoxes()
{
    ui->userSelectCities_comboBox->setModel(m_controller->getDistancesQueryModel("select
DISTINCT StartCity from Distances ORDER BY StartCity ASC;"));

ui->adminChooseCities_comboBox->setModel(m_controller->getDistancesQueryModel("select
DISTINCT StartCity from Distances ORDER BY StartCity ASC;"));

ui->adminUploadChooseCities_comboBox->setModel(m_controller->getDistancesQueryModel("
select DISTINCT StartCity from Distances ORDER BY StartCity ASC;"));
    ui->autoTripCity_comboBox->setModel(m_controller->getDistancesQueryModel("select
DISTINCT StartCity from Distances ORDER BY StartCity ASC;"));
    ui->selectNumberOfCities_spinBox->setMaximum(m_controller->cityCount);
}

void MainWindow::on_clear_pushButton_clicked()
{
    this->ui->inputUsername_lineEdit->setText("");
    this->ui->inputPassword_lineEdit->setText("");
}

void MainWindow::on_login_pushButton_clicked()
{
    const QString ADMIN_USERNAME = "admin";
    const QString ADMIN_PASSWORD = "admin";
    const QString USER_USERNAME = "user";
    const QString USER_PASSWORD = "user";

    QString usernameInput = ui->inputUsername_lineEdit->text();
```

```
    QString passwordInput = ui->inputPassword_lineEdit->text();

    if (usernameInput == ADMIN_USERNAME && passwordInput == ADMIN_PASSWORD)
    {
        QMessageBox::information(this, "Success", "Username and Password are correct\nLogging into Admin");
        on_clear_pushButton_clicked();
        ui->stackedWidget->setCurrentWidget(ui->admin_page);
    }
    else if (usernameInput == USER_USERNAME && usernameInput == USER_PASSWORD)
    {
        QMessageBox::information(this, "Success", "Username and Password are correct\nLogging into User");
        on_clear_pushButton_clicked();
        ui->stackedWidget->setCurrentWidget(ui->user_page);
    }
    else
    {
        QMessageBox::information(this, "Invalid", "Username and Password are incorrect");
        on_clear_pushButton_clicked();
    }
}

void MainWindow::on_viewCitiesAndFoods_pushButton_clicked()
{
    ui->stackedWidget->setCurrentWidget(ui->viewCitiesAndFoods_page);
}

void MainWindow::on_userSelectCities_comboBox_currentTextChanged(const QString &arg1)
{
    ui->userViewCities_tableView->setModel(m_controller->getDistancesQueryModel("select EndCity, Distance from Distances where StartCity = '"+arg1+"';"));
    ui->userViewFoods_tableView->setModel(m_controller->getFoodsQueryModel("select Food, Cost from Foods where City = '"+arg1+"';"));
}

void MainWindow::on_returnToUserPage_pushButton_clicked()
{
    ui->stackedWidget->setCurrentWidget(ui->user_page);
    qDebug() << "BACK BUTTON PRESSED";
}

void MainWindow::on_contactUs_pushButton_clicked()
{
```

```
  QPixmap icon (":images/runtimeerror.png");
  ui->iconLabel->setPixmap(icon);
  ui->stackedWidget->setCurrentWidget(ui->contactUs_Page);
}

void MainWindow::on_submit_pushButton_clicked()
{
  QMessageBox::information(this, "Submitted", "Thank you for contacting Runtime Terror. Your
information will be forwarded to the proper team.");
  ui->userInputBox->clear();
  ui->stackedWidget->setCurrentWidget(ui->user_page);
}

void MainWindow::on_return_pushButton_clicked()
{
  ui->stackedWidget->setCurrentWidget(ui->user_page);
}

void MainWindow::on_userLogout_pushButton_clicked()
{
  ui->stackedWidget->setCurrentWidget(ui->login_page);
}

void MainWindow::on_adminEdit_pushButton_clicked()
{
  ui->stackedWidget->setCurrentWidget(ui->adminEdit_page);
}

void MainWindow::on_adminViewFoods_tableView_activated(const QModelIndex &index)
{
  QString currentFood;
  QString currentCity;

  if (index.isValid())
  {
     QSqlQuery qry;
     double price;
     currentFood = index.data().toString();
     ui->editFoodFood_label->setText(currentFood);
     currentCity = ui->adminChooseCities_comboBox->currentText();
     qry.prepare("select Cost from Foods where City = '"+currentCity+"' and Food =
'"+currentFood+"';");

     if (!qry.exec())
```

```
        qDebug() << "ERROR: on_adminViewFoods_tableView_activated(const QModelIndex
&index))";
    else
    {
        if (qry.first())
        {
            qDebug() << qry.value(0);
            QString test = qry.value(0).toString();
            test.remove(0,1);
            price = test.toDouble();
            ui->editPrice_doubleSpinBox->setValue(price);
        }
    }
  }
}

void MainWindow::on_userLogout_Pushbutton_clicked()
{
  ui->stackedWidget->setCurrentWidget(ui->login_page);
}

void MainWindow::on_adminLogout_pushButton_clicked()
{
  ui->stackedWidget->setCurrentWidget(ui->login_page);
}

void MainWindow::on_adminChooseCities_comboBox_currentTextChanged(const QString
&arg1)
{
  ui->adminViewFoods_tableView->setModel(m_controller->getFoodsQueryModel("select Food,
Cost from Foods where City = '"+arg1+"';"));
  ui->editFoodCity_label->setText(arg1);
}

void MainWindow::resetAdminEditPage()
{
  ui->editFoodCity_label->setText(ui->adminChooseCities_comboBox->currentText());
  ui->editFoodFood_label->setText("No Food Selected!");
  ui->editPrice_doubleSpinBox->clear();
  ui->newFood_lineEdit->clear();
  ui->newFoodPrice_doubleSpinBox->clear();

}
void MainWindow::on_editPrice_pushButton_clicked()
```

```
{
  if (ui->editFoodFood_label->text() == "No Food Selected!")
     QMessageBox::warning(this, "Invalid", "No food selected!");
  else if (ui->editPrice_doubleSpinBox->value() < 0.01)
     QMessageBox::warning(this, "Invalid", "Invalid price!");
  else
  {
     QString food = ui->editFoodFood_label->text();
     QString city = ui->editFoodCity_label->text();
     double cost = ui->editPrice_doubleSpinBox->value();

     QMessageBox::StandardButton reply =
           QMessageBox::question(this, "Edit", "Are you sure you want to edit " + food + "?",
                                 QMessageBox::Yes | QMessageBox::No);

     if (reply == QMessageBox::Yes)
     {
        m_controller->editFoodCostQuery(city, food, cost);
        resetAdminEditPage();
        on_adminChooseCities_comboBox_currentTextChanged(city);
     }
  }
}


void MainWindow::on_returnToAdminPage_pushButton_clicked()
{
  ui->stackedWidget->setCurrentWidget(ui->admin_page);
}


void MainWindow::on_deleteFood_pushButton_clicked()
{

  if (ui->editFoodFood_label->text() == "No Food Selected!")
     QMessageBox::warning(this, "Invalid", "No food selected!");
  else
  {
     QString food = ui->editFoodFood_label->text();
     QString city = ui->editFoodCity_label->text();
     double cost = ui->editPrice_doubleSpinBox->value();

     QMessageBox::StandardButton reply =
           QMessageBox::question(this, "Delete", "Are you sure you want to delete " + food + "?",
```

```
                                QMessageBox::Yes | QMessageBox::No);

        if (reply == QMessageBox::Yes)
        {
            m_controller->deleteFoodQuery(city, food, cost);
            resetAdminEditPage();
            on_adminChooseCities_comboBox_currentTextChanged(city);
        }
    }
}

void MainWindow::on_addNewFood_pushButton_clicked()
{

    if (ui->newFood_lineEdit->text() == "" && ui->newFoodPrice_doubleSpinBox->value() <= 0.00)
        QMessageBox::warning(this, "Invalid", "New food has no name and price!");
    else if (ui->newFoodPrice_doubleSpinBox->value() <= 0.00)
        QMessageBox::warning(this, "Invalid", "New food has no price!");
    else if (ui->newFood_lineEdit->text() == "")
        QMessageBox::warning(this, "Invalid", "New food has no name!");
    else if (ui->editFoodCity_label->text() == "No City Selected!")
        QMessageBox::warning(this, "Invalid", "No city selected for new food!");
    else
    {
        QString city = ui->editFoodCity_label->text();
        QString food = ui->newFood_lineEdit->text();
        double cost  = ui->newFoodPrice_doubleSpinBox->value();

        QMessageBox::StandardButton reply =
                QMessageBox::question(this, "Add", "Are you sure you want to add " + food + " to " +
city + "?",
                                QMessageBox::Yes | QMessageBox::No);

        if (reply == QMessageBox::Yes)
        {
            m_controller->addFoodQuery(city, food, cost);

            resetAdminEditPage();
            on_adminChooseCities_comboBox_currentTextChanged(city);
        }
    }
}
```

```
void MainWindow::on_resetEditPage_pushButton_clicked()
{
  QString defaultCity = "Amsterdam";
  resetAdminEditPage();
  on_adminChooseCities_comboBox_currentTextChanged(defaultCity);
  ui->adminChooseCities_comboBox->setCurrentIndex(0);
}


void MainWindow::on_adminUpload_pushButton_clicked()
{
  ui->stackedWidget->setCurrentWidget(ui->adminUpload_page);
}


void MainWindow::on_returnToAdminPage_pushButton2_clicked()
{
  ui->stackedWidget->setCurrentWidget(ui->admin_page);
}


void MainWindow::on_adminUploadChooseCities_comboBox_currentTextChanged(const
QString &arg1)
{

ui->adminUploadViewCities_tableView->setModel(m_controller->getDistancesQueryModel("sel
ect EndCity, Distance from Distances where StartCity = '"+arg1+"';"));

ui->adminUploadViewFoods_tableView->setModel(m_controller->getFoodsQueryModel("select
Food, Cost from Foods where City = '"+arg1+"';"));
}


void MainWindow::on_adminUploadCities_pushButton_clicked()
{
  m_controller->uploadCitiesFile();
  fillCitiesComboBoxes();
}


void MainWindow::on_adminUploadFoods_pushButton_clicked()
{
  m_controller->uploadFoodsFile();
  fillCitiesComboBoxes();
```

```
    ui->selectNumberOfCities_spinBox->setMaximum(12);
}

void MainWindow::on_planTrip_pushButton_clicked()
{
    ui->stackedWidget->setCurrentWidget(ui->pickTrip_Page);
}

void MainWindow::on_parisTrip_pushButton_clicked()
{
    int totalDistance = 0;
    m_controller->createTripList();
    m_controller->displayTripList();
    m_controller->createTrip(ui->autoTripCity_comboBox->currentText(),
ui->selectNumberOfCities_spinBox->value());
    m_controller->displayTrip();

    ui->autoTrip_tableWidget->setRowCount(m_controller->completedTripList.size());
    ui->autoTrip_tableWidget->setColumnCount(3);

    for (int i = 0; i < m_controller->completedTripList.size(); i++)
    {
        QTableWidgetItem *startCity = new QTableWidgetItem();
        QTableWidgetItem *endCity = new QTableWidgetItem();
        QTableWidgetItem *distance = new QTableWidgetItem();

        startCity->setText(m_controller->completedTripList[i]->getStartCity());
        endCity->setText(m_controller->completedTripList[i]->getEndCity());
        distance->setText(QString::number(m_controller->completedTripList[i]->getDistance()));

        ui->autoTrip_tableWidget->setItem(i, 0, startCity);
        ui->autoTrip_tableWidget->setItem(i, 1, endCity);
        ui->autoTrip_tableWidget->setItem(i, 2, distance);

        totalDistance = totalDistance + m_controller->completedTripList[i]->getDistance();
    }

    ui->autoTrip_tableWidget->resizeColumnsToContents();
    ui->autoTripTotalDistance_label->setText(QString::number(totalDistance));

    // -FOOD SECTION-

    m_controller->createFoodList();
```

```
    ui->purchaseFoods_tableWidget->setRowCount(m_controller->foodList.size());
    ui->purchaseFoods_tableWidget->setColumnCount(4);

    for (int i = 0; i < m_controller->foodList.size(); i++)
    {
        QTableWidgetItem *city = new QTableWidgetItem();
        QTableWidgetItem *food = new QTableWidgetItem();
        QTableWidgetItem *cost = new QTableWidgetItem();

        city->setText(m_controller->foodList[i]->getCity());
        food->setText(m_controller->foodList[i]->getName());
        cost->setText(QString::number(m_controller->foodList[i]->getCost()));

        ui->purchaseFoods_tableWidget->setItem(i, 0, city);
        ui->purchaseFoods_tableWidget->setItem(i, 1, food);
        ui->purchaseFoods_tableWidget->setItem(i, 2, cost);
        ui->purchaseFoods_tableWidget->setCellWidget(i, 3, new QSpinBox);
    }

    ui->purchaseFoods_tableWidget->resizeColumnsToContents();
}

void MainWindow::on_planTripPageBack_pushButton_clicked()
{
    on_autoTripReset_pushButton_clicked();
    ui->stackedWidget->setCurrentWidget(ui->user_page);
}


void MainWindow::on_pickTripBack_pushButton_clicked()
{
    ui->stackedWidget->setCurrentWidget(ui->user_page);
}

void MainWindow::on_autoPlanner_pushButton_clicked()
{
    ui->stackedWidget->setCurrentWidget(ui->planAutoTrip_Page);
}

void MainWindow::on_autoTripReset_pushButton_clicked()
{
    // -CLEARING TRIP DATA-
    m_controller->resetTripList();
    m_controller->resetTrip();
```

```
    ui->autoTrip_tableWidget->clearContents();
    ui->autoTrip_tableWidget->clear();
    ui->autoTrip_tableWidget->setRowCount(0);
    ui->autoTrip_tableWidget->setColumnCount(0);
    ui->autoTripTotalDistance_label->clear();

    // -CLEARING FOOD DATA-
    ui->purchaseFoods_tableWidget->clearContents();
    ui->purchaseFoods_tableWidget->clear();
    ui->purchaseFoods_tableWidget->setRowCount(0);
    ui->purchaseFoods_tableWidget->setColumnCount(0);

    ui->foodReceipt_tableView->clear();
    ui->totalCost_label->clear();
}


void MainWindow::on_purchaseFoods_pushButton_clicked()
{
    ui->foodReceipt_tableView->clear();

    double totalCost = 0;
    double totalCostOfCity = 0;
    QTableWidgetItem *city = ui->purchaseFoods_tableWidget->item(0,0);
    QString tempCity = city->text();

    for (int i = 0; i < ui->purchaseFoods_tableWidget->rowCount(); i++)
    {
        int val = static_cast<QSpinBox*>(ui->purchaseFoods_tableWidget->cellWidget(i,
3))->value();
        QTableWidgetItem *cost = ui->purchaseFoods_tableWidget->item(i,2);
        totalCost = totalCost + (cost->text().toDouble() * val);

        int quantity = static_cast<QSpinBox*>(ui->purchaseFoods_tableWidget->cellWidget(i,
3))->value();

        if (quantity > 0)
        {
            if (tempCity != ui->purchaseFoods_tableWidget->item(i,0)->text()) {

//              ui->foodReceipt_tableView->append("----------------");
            ui->foodReceipt_tableView->append(tempCity + " total: $" +
QString::number(totalCostOfCity));
            ui->foodReceipt_tableView->append("----------------");
```

```
            totalCostOfCity = 0;
        }

        QString name = ui->purchaseFoods_tableWidget->item(i,1)->text();
        double costs = cost->text().toDouble() * quantity;

        ui->foodReceipt_tableView->append(name);
        ui->foodReceipt_tableView->append(cost->text() + " x" + QString::number(quantity));
        ui->foodReceipt_tableView->append(QString::number(costs));
        ui->foodReceipt_tableView->append("----------");

        totalCostOfCity += costs;
        tempCity = ui->purchaseFoods_tableWidget->item(i,0)->text();
    }
  }
  ui->foodReceipt_tableView->append(tempCity + " total: $" +
QString::number(totalCostOfCity));
  ui->foodReceipt_tableView->append("----------");

  ui->totalCost_label->setText("$" + QString::number(totalCost));
}


void MainWindow::on_customPlanner_pushButton_clicked()
{
  ui->stackedWidget->setCurrentWidget(ui->customTrip_page);
  ui->customTripSelect_tableView->setModel(m_controller->getDistancesQueryModel("select
DISTINCT StartCity from Distances ORDER BY StartCity ASC;"));
}


void MainWindow::on_customTripBack_pushButton_clicked()
{
  ui->stackedWidget->setCurrentWidget(ui->user_page);
}


void MainWindow::on_customTripSelect_tableView_doubleClicked(const QModelIndex &index)
{
  bool found = false;

  for (int i = 0; i < m_controller->customTripListCities.size(); i++)
  {
```

```
        if (m_controller->customTripListCities[i] == index.data())
        {
            found = true;
            QMessageBox::information(this, "Invalid", "Please select a different city!");
            break;
        }
    }

    if (!found)
    {
        m_controller->customTripListCities.append(index.data().toString());
        ui->customTrip_textBrowser->append(index.data().toString());
    }
}

void MainWindow::on_customTripReset_pushButton_clicked()
{
    // -CLEARING TRIP DATA-
    m_controller->resetTripList();
    m_controller->resetTrip();

    ui->customTripSelect_tableView->reset();
    ui->customTripDisplay_tableWidget->clearContents();
    ui->customTripDisplay_tableWidget->clear();
    ui->customTripDisplay_tableWidget->setRowCount(0);
    ui->customTripDisplay_tableWidget->setColumnCount(0);

    ui->customTripSelect_tableView->setModel(m_controller->getDistancesQueryModel("select
DISTINCT StartCity from Distances ORDER BY StartCity ASC;"));
    ui->customTrip_textBrowser->clear();
    ui->customTripTotalDistance_label->clear();

    // -CLEARING FOOD DATA-
    ui->customPurchaseFoods_tableWidget->clearContents();
    ui->customPurchaseFoods_tableWidget->clear();
    ui->customPurchaseFoods_tableWidget->setRowCount(0);
    ui->customPurchaseFoods_tableWidget->setColumnCount(0);

    ui->customFoodReceipt_tableView->clear();
    ui->customTotalCost_label->clear();
}

void MainWindow::on_createCustomTrip_pushButton_clicked()
{
```

```
if (m_controller->customTripListCities.size() > 1)
{
    QString startCity = m_controller->customTripListCities[0];
    int numberOfCities = m_controller->customTripListCities.size() - 1;
    int totalDistance = 0;

    m_controller->createCustomTripList();
    m_controller->displayTripList();
    m_controller->createTrip(startCity, numberOfCities);
    m_controller->displayTrip();

    ui->customTripDisplay_tableWidget->setRowCount(m_controller->completedTripList.size());
    ui->customTripDisplay_tableWidget->setColumnCount(3);

    for (int i = 0; i < m_controller->completedTripList.size(); i++)
    {
        QTableWidgetItem *startCity = new QTableWidgetItem();
        QTableWidgetItem *endCity = new QTableWidgetItem();
        QTableWidgetItem *distance = new QTableWidgetItem();

        startCity->setText(m_controller->completedTripList[i]->getStartCity());
        endCity->setText(m_controller->completedTripList[i]->getEndCity());
        distance->setText(QString::number(m_controller->completedTripList[i]->getDistance()));

        ui->customTripDisplay_tableWidget->setItem(i, 0, startCity);
        ui->customTripDisplay_tableWidget->setItem(i, 1, endCity);
        ui->customTripDisplay_tableWidget->setItem(i, 2, distance);

        totalDistance = totalDistance + m_controller->completedTripList[i]->getDistance();
    }

    ui->customTripTotalDistance_label->setText(QString::number(totalDistance));
    ui->customTripDisplay_tableWidget->resizeColumnsToContents();

    // -FOOD SECTION-

    m_controller->createFoodList();

    ui->customPurchaseFoods_tableWidget->setRowCount(m_controller->foodList.size());
    ui->customPurchaseFoods_tableWidget->setColumnCount(4);

    for (int i = 0; i < m_controller->foodList.size(); i++)
    {
```

```
            QTableWidgetItem *city = new QTableWidgetItem();
            QTableWidgetItem *food = new QTableWidgetItem();
            QTableWidgetItem *cost = new QTableWidgetItem();

            city->setText(m_controller->foodList[i]->getCity());
            food->setText(m_controller->foodList[i]->getName());
            cost->setText(QString::number(m_controller->foodList[i]->getCost()));

            ui->customPurchaseFoods_tableWidget->setItem(i, 0, city);
            ui->customPurchaseFoods_tableWidget->setItem(i, 1, food);
            ui->customPurchaseFoods_tableWidget->setItem(i, 2, cost);
            ui->customPurchaseFoods_tableWidget->setCellWidget(i, 3, new QSpinBox);
        }

        ui->customPurchaseFoods_tableWidget->resizeColumnsToContents();
    }

    else
        QMessageBox::information(this, "Invalid", "Please select at least 2 cities!");
}

void MainWindow::on_customPurchaseFoods_pushButton_clicked()
{
    ui->customFoodReceipt_tableView->clear();

    double totalCost = 0;
    double totalCostOfCity = 0;
    QTableWidgetItem *city = ui->customPurchaseFoods_tableWidget->item(0,0);
    QString tempCity = city->text();

    for (int i = 0; i < ui->customPurchaseFoods_tableWidget->rowCount(); i++)
    {
        int val = static_cast<QSpinBox*>(ui->customPurchaseFoods_tableWidget->cellWidget(i,
3))->value();
        QTableWidgetItem *cost = ui->customPurchaseFoods_tableWidget->item(i,2);
        totalCost = totalCost + (cost->text().toDouble() * val);

        int quantity =
static_cast<QSpinBox*>(ui->customPurchaseFoods_tableWidget->cellWidget(i, 3))->value();

        if (quantity > 0)
        {
            if (tempCity != ui->customPurchaseFoods_tableWidget->item(i,0)->text()) {
```

```
//              ui->foodReceipt_tableView->append("----------");
            ui->customFoodReceipt_tableView->append(tempCity + " total: $" +
QString::number(totalCostOfCity));
            ui->customFoodReceipt_tableView->append("----------");

            totalCostOfCity = 0;
        }
        QString name = ui->customPurchaseFoods_tableWidget->item(i,1)->text();
        double costs = cost->text().toDouble() * quantity;

        ui->customFoodReceipt_tableView->append(name);
        ui->customFoodReceipt_tableView->append(cost->text() + " x" +
QString::number(quantity));
        ui->customFoodReceipt_tableView->append(QString::number(costs));
        ui->customFoodReceipt_tableView->append("----------");

        totalCostOfCity += costs;
        tempCity = ui->customPurchaseFoods_tableWidget->item(i,0)->text();
    }
  }

  ui->customFoodReceipt_tableView->append(tempCity + " total: $" +
QString::number(totalCostOfCity));
  ui->customFoodReceipt_tableView->append("----------");

  ui->customTotalCost_label->setText("$" + QString::number(totalCost));
}
```

trip.cpp

```
#include "trip.h"

Trip::Trip(QObject *parent) : QObject(parent) {}

QString Trip::getStartCity() const
{
  return startCity;
}

QString Trip::getEndCity() const
{
  return endCity;
}
```

```
int Trip::getDistance() const
{
  return distance;
}

void Trip::setStartCity(const QString &temp)
{
  startCity = temp;
}

void Trip::setEndCity(const QString &temp)
{
  endCity = temp;
}

void Trip::setDistance(int temp)
{
  distance = temp;
}
```