Файл Transport.h

#include "../database/Database.h"

#include <iostream>

class Transport

{

private:

std::string brand;

std::string model;

std::string color;

EngineType engineType;

public:

enum EngineType

{

PETROL,

DIESEL,

HYBRID,

ELECTRIC

};

Transport();

Transport(std::string brand, std::string model, std::string color, EngineType engineType);

~Transport();

std::string getBrand();

void setBrand(std::string brand);

std::string getModel();

void setModel(std::string model);

std::string getColor();

void setColor(std::string color);

EngineType getEngineType();

void setEngineType(EngineType engineType);

std::string getEngineTypeString();

static EngineType stringToEngineType(const std::string &engineTypeStr);

void isIdValid(int &id, const std::string &transportName, Database \*Db);

};

Файл Transport.cpp

#include "Transport.h"

Transport::Transport() : brand(""), model(""), color(""), engineType(EngineType::DIESEL) {}

Transport::Transport(std::string brand, std::string model, std::string color, EngineType engineType) : brand(brand), model(model), color(color), engineType(engineType) {}

Transport::~Transport() {}

std::string Transport::getBrand()

{

return brand;

}

void Transport::setBrand(std::string brand)

{

this->brand = brand;

}

std::string Transport::getModel()

{

return model;

}

void Transport::setModel(std::string model)

{

this->model = model;

}

std::string Transport::getColor()

{

return color;

}

void Transport::setColor(std::string color)

{

this->color = color;

}

EngineType Transport::getEngineType()

{

return engineType;

}

void Transport::setEngineType(EngineType engineType)

{

this->engineType = engineType;

}

std::string Transport::getEngineTypeString()

{

switch (engineType)

{

case EngineType::PETROL:

return "PETROL";

case EngineType::DIESEL:

return "DIESEL";

case EngineType::HYBRID:

return "HYBRID";

case EngineType::ELECTRIC:

return "ELECTRIC";

default:

return "Unknown";

}

}

EngineType Transport::stringToEngineType(const std::string &engineTypeStr) {

if (engineTypeStr == "PETROL") {

return EngineType::PETROL;

} else if (engineTypeStr == "DIESEL") {

return EngineType::DIESEL;

} else if (engineTypeStr == "HYBRID") {

return EngineType::HYBRID;

} else if (engineTypeStr == "ELECTRIC") {

return EngineType::ELECTRIC;

} else {

std::cerr << "Error: Invalid engine type - unknown value: " << engineTypeStr << std::endl;

throw std::runtime\_error("Invalid engine type: " + engineTypeStr);

}

}

void Transport::isIdValid(int &id, const std::string &transportName, Database \*Db)

{

while (true)

{

try

{

std::cout << "Enter the " << transportName << " ID: ";

std::cin >> id;

std::string checkQuery = "SELECT \* FROM " + transportName + " WHERE " + transportName + "\_id = " + std::to\_string(id);

pqxx::result result = Db->executeQuery(checkQuery);

if (!result.empty())

{

return;

}

else

{

std::cerr << transportName << " with entered value does not exist. Try again." << std::endl;

}

}

catch (const std::exception &e)

{

std::cerr << "Error: " << e.what() << std::endl;

}

}

}

Файл PublicTransport.h

#include <iostream>

#include "Transport.h"

#include "ticket/TransportTicket.h"

class PublicTransport : public Transport {

private:

int transportId;

int capacity;

public:

enum TransportType {

BUS,

TROLLEYBUS,

};

PublicTransport();

PublicTransport(int transportId, std::string brand, std::string model, std::string color, EngineType engineType, int capacity);

PublicTransport(std::string brand, std::string model, std::string color, EngineType engineType, int capacity);

int getTransportId();

void setTransportId(int id);

int getCapacity();

void setCapacity(int capacity);

std::string getCurrentTimestampAsString();

int insertTicket(Database& db, TransportTicket ticket);

void bookTransport(Database& Db, PublicTransport::TransportType transportType);

~PublicTransport();

};

Файл PublicTransport.cpp

#include "PublicTransport.h"

#include "Customer.h"

#include "price/RoutePrice.h"

#include "ticket/TransportTicket.h"

#include "../action/InputUtils.h"

PublicTransport::PublicTransport() : transportId(0), capacity(0) {}

PublicTransport::PublicTransport(int transportId, std::string brand, std::string model, std::string color, EngineType engineType, int capacity)

: Transport(brand, model, color, engineType), transportId(transportId), capacity(capacity) {}

PublicTransport::PublicTransport(std::string brand, std::string model, std::string color, EngineType engineType, int capacity)

: Transport(brand, model, color, engineType), transportId(0), capacity(capacity) {}

int PublicTransport::getTransportId() {

return transportId;

}

void PublicTransport::setTransportId(int transporId) {

this->transportId = transporId;

}

int PublicTransport::getCapacity() {

return capacity;

}

void PublicTransport::setCapacity(int capacity) {

this->capacity = capacity;

}

std::string PublicTransport::getCurrentTimestampAsString() {

auto now = std::chrono::system\_clock::now();

std::time\_t currentTime = std::chrono::system\_clock::to\_time\_t(now);

std::stringstream ss;

ss << std::ctime(&currentTime);

return ss.str();

}

int PublicTransport::insertTicket(Database& db, TransportTicket ticket) {

try {

std::string currentTimestamp = getCurrentTimestampAsString();

std::string insertQuery = "INSERT INTO ticket (transport\_id, customer\_id, price, purchase\_time, transport\_type) VALUES ("

+ std::to\_string(ticket.getTransportId()) + ", "

+ std::to\_string(ticket.getCustomerId()) + ", "

+ std::to\_string(ticket.getPrice()) + ", '"

+ currentTimestamp + "', '"

+ (ticket.getTransportType() == TransportType::BUS ? "BUS" : "TROLLEYBUS") + "') RETURNING ticket\_id";

pqxx::result ticketIdResult = db.executeQuery(insertQuery);

int ticketId = ticketIdResult[0]["ticket\_id"].as<int>();

return ticketId;

} catch (const std::exception& e) {

std::cerr << e.what() << std::endl;

throw std::runtime\_error("Failed to insert ticket into the database");

}

}

void PublicTransport::bookTransport(Database& Db, PublicTransport::TransportType transportType) {

std::string transportTypeName = (transportType == PublicTransport::BUS) ? "bus" : "trolleybus";

try {

Customer customer;

customer.setName(InputUtils::getStringInput("Enter your name: "));

customer.setSurname(InputUtils::getStringInput("Enter your surname: "));

customer.setContactInformation(InputUtils::getStringInput("Enter your contact information: "));

std::string addCustomerQuery = "INSERT INTO customer (name, surname, contact\_information) VALUES ('" +

customer.getName() + "','" +

customer.getSurname() + "','" +

customer.getContactInformation() + "') RETURNING customer\_id";

pqxx::result customerIdResult = Db.executeQuery(addCustomerQuery);

customer.setCustomerId(customerIdResult[0][0].as<int>());

int routeId = InputUtils::getPositiveInput<int>("Enter the route ID: ");

RoutePrice routePrice;

try {

routePrice = RoutePrice().getTicketPrice(Db, routeId);

std::cout << "The price for the ticket on Route ID: " << routeId

<< " is $" << routePrice.getPrice() << std::endl;

} catch (const std::exception& e) {

std::cerr << "Error: " << e.what() << std::endl;

return;

}

int transportId = InputUtils::getPositiveInput<int>("Enter the " + transportTypeName + " ID: ");

std::string checkTransportQuery = "SELECT EXISTS(SELECT 1 FROM " + transportTypeName + " WHERE " + transportTypeName + "\_id = " + std::to\_string(transportId) + ")";

pqxx::result transportExistsResult = Db.executeQuery(checkTransportQuery);

bool transportExists = transportExistsResult[0][0].as<bool>();

if (!transportExists) {

std::cout << "No " << transportTypeName << " with ID " << transportId << " found. Please enter a valid ID." << std::endl;

return;

}

TransportTicket ticket;

ticket.setTransportId(transportId);

ticket.setCustomerId(customer.getCustomerId());

ticket.setPrice(routePrice.getPrice());

int ticketId = insertTicket(Db, ticket);

std::cout << transportTypeName << " booked successfully! Ticket ID: " << ticketId << std::endl;

} catch (const std::exception& e) {

std::cerr << e.what() << std::endl;

}

}

PublicTransport::~PublicTransport() {}

Файл Bus.h

#include <iostream>

#include <string>

#include "PublicTransport.h"

#include "pqxx/pqxx"

class Bus : public PublicTransport {

private:

bool hasContactlessPayment;

public:

Bus();

Bus(const std::string& brand, const std::string& model, const std::string& color, EngineType engineType, int capacity, bool hasContactlessPayment);

Bus(int transportId, std::string brand, std::string model, std::string color, EngineType engineType, int capacity, bool hasContactlessPayment);

bool isHasContactlessPayment();

void setHasContactlessPayment(bool hasContactlessPayment);

void displayBusDetails(const pqxx::result::const\_iterator& row);

void displayAllBuses(Database& Db);

};

Файл Bus.cpp

#include "bus.h"

Bus::Bus() : PublicTransport(0, "", "", "", EngineType::DIESEL, 0), hasContactlessPayment(false) {}

Bus::Bus(const std::string& brand, const std::string& model, const std::string& color, EngineType engineType, int capacity, bool hasContactlessPayment)

: PublicTransport(brand, model, color, engineType, capacity), hasContactlessPayment(hasContactlessPayment) {}

Bus::Bus(int transportId, std::string brand, std::string model, std::string color, EngineType engineType, int capacity, bool hasContactlessPayment)

: PublicTransport(transportId, brand, model, color, engineType, capacity), hasContactlessPayment(hasContactlessPayment) {}

bool Bus::isHasContactlessPayment(){

return hasContactlessPayment;

}

void Bus::setHasContactlessPayment(bool hasContactlessPayment) {

this->hasContactlessPayment = hasContactlessPayment;

}

void Bus::displayBusDetails(const pqxx::result::const\_iterator& row) {

int transportId = row["bus\_id"].as<int>();

int capacity = row["capacity"].as<int>();

std::string brand = row["brand"].as<std::string>();

std::string model = row["model"].as<std::string>();

std::string color = row["color"].as<std::string>();

EngineType engineType = Transport::stringToEngineType(row["engineType"].as<std::string>());

bool hasContactlessPayment = row["has\_contactless\_payment"].as<bool>();

std::unique\_ptr<Bus> bus = std::make\_unique<Bus>(

transportId, brand, model, color,

static\_cast<EngineType>(engineType),

capacity,

hasContactlessPayment

);

std::cout << "Bus Details:" << std::endl;

std::cout << "Transport ID: " << bus->getTransportId() << std::endl;

std::cout << "Brand: " << bus->getBrand() << std::endl;

std::cout << "Model: " << bus->getModel() << std::endl;

std::cout << "Color: " << bus->getColor() << std::endl;

std::cout << "Engine Type: " << bus->getEngineTypeString() << std::endl;

std::cout << "Capacity: " << bus->getCapacity() << std::endl;

std::cout << "Has Contactless Payment: " << (bus->isHasContactlessPayment() ? "Yes" : "No") << std::endl;

std::cout << std::endl;

}

void Bus::displayAllBuses(Database& Db) {

try {

pqxx::result busResult = Db.executeQuery("SELECT \* FROM bus");

std::cout << "Displaying Bus Details:" << std::endl;

for (const auto& row : busResult) {

displayBusDetails(row);

}

} catch (const std::exception &e) {

std::cerr << e.what() << std::endl;

}

}

Файл TrolleyBus.h

#include "../database/Database.h"

#include <iostream>

enum EngineType

{

PETROL,

DIESEL,

HYBRID,

ELECTRIC

};

class Transport

{

private:

std::string brand;

std::string model;

std::string color;

EngineType engineType;

public:

Transport();

Transport(std::string brand, std::string model, std::string color, EngineType engineType);

~Transport();

std::string getBrand();

void setBrand(std::string brand);

std::string getModel();

void setModel(std::string model);

std::string getColor();

void setColor(std::string color);

EngineType getEngineType();

void setEngineType(EngineType engineType);

std::string getEngineTypeString();

static EngineType stringToEngineType(const std::string &engineTypeStr);

void isIdValid(int &id, const std::string &transportName, Database \*Db);

};

Файл TrolleyBus.cpp

#include "TrolleyBus.h"

TrolleyBus::TrolleyBus() : PublicTransport(), hasSockets(false) {}

TrolleyBus::TrolleyBus(int transportId, std::string brand, std::string model, std::string color, EngineType engineType, int capacity, bool hasSockets)

: PublicTransport(transportId, brand, model, color, engineType, capacity), hasSockets(hasSockets) {}

TrolleyBus::TrolleyBus(const std::string& brand, const std::string& model, const std::string& color, EngineType engineType, int capacity, bool hasSockets)

: PublicTransport(brand, model, color, engineType, capacity), hasSockets(hasSockets) {}

bool TrolleyBus::getHasSockets(){

return hasSockets;

}

void TrolleyBus::setHasSockets(bool hasSockets) {

this->hasSockets = hasSockets;

}

void TrolleyBus::displayTrolleyBusDetails(const pqxx::result::const\_iterator& row) {

int transportId = row["trolleybus\_id"].as<int>();

int capacity = row["capacity"].as<int>();

std::string brand = row["brand"].as<std::string>();

std::string model = row["model"].as<std::string>();

std::string color = row["color"].as<std::string>();

EngineType engineType = stringToEngineType(row["engineType"].as<std::string>());

bool hasSockets = row["has\_sockets"].as<bool>();

std::unique\_ptr<TrolleyBus> trolleyBus = std::make\_unique<TrolleyBus>(

transportId, brand, model, color,

static\_cast<EngineType>(engineType),

capacity,

hasSockets

);

std::cout << "TrolleyBus Details:" << std::endl;

std::cout << "Transport ID: " << trolleyBus->getTransportId() << std::endl;

std::cout << "Brand: " << trolleyBus->getBrand() << std::endl;

std::cout << "Model: " << trolleyBus->getModel() << std::endl;

std::cout << "Color: " << trolleyBus->getColor() << std::endl;

std::cout << "Engine Type: " << trolleyBus->getEngineTypeString() << std::endl;

std::cout << "Capacity: " << trolleyBus->getCapacity() << std::endl;

std::cout << "Has Sockets: " << (trolleyBus->getHasSockets() ? "Yes" : "No") << std::endl;

std::cout << std::endl;

}

void TrolleyBus::displayAllTrolleyBuses(Database& db) {

try {

pqxx::result trolleyBusResult = db.executeQuery("SELECT \* FROM trolleybus");

std::cout << "Displaying TrolleyBus Details:" << std::endl;

for (const auto& row : trolleyBusResult) {

displayTrolleyBusDetails(row);

}

} catch (const std::exception &e) {

std::cerr << e.what() << std::endl;

}

}

Файл Taxi.h

#include "Transport.h"

#include "pqxx/pqxx"

#include "../database/Database.h"

enum RentCarTypes {

ECONOMY,

COMFORT,

BUSINESS

};

class Taxi : public Transport {

private:

int carId;

double pricePerKil;

bool hasDriver;

bool hasWiFi;

bool hasChildSeat;

RentCarTypes rentCarTypes;

public:

Taxi(std::string brand, std::string model, std::string color, EngineType engineType, int carId, double pricePerKil, bool hasDriver, bool hasWiFi, bool hasChildSeat, RentCarTypes rentCarTypes);

Taxi();

int getCarId();

void setCarId(int carId);

double getPricePerKil();

void setPricePerKil(double pricePerKil);

bool getHasDriver();

void setHasDriver(bool hasDriver);

bool getHasWiFi();

void setHasWiFi(bool hasWiFi);

bool getHasChildSeat();

void setHasChildSeat(bool hasChildSeat);

RentCarTypes getRentCarTypes();

void setRentCarTypes(RentCarTypes rentCarTypes);

~Taxi();

static RentCarTypes stringToRentCarTypes(const std::string& rentCarTypesStr);

static void displayTaxiDetails(const pqxx::result::const\_iterator& row);

void displayAllTaxis(Database& Db);

static void displayTaxiById(Database& Db, int taxiId);

static void displayTaxisByBrand(Database& Db, const std::string& brand);

static void displayTaxisByRentCarType(Database& Db, RentCarTypes rentCarType);

std::string to\_string(const int value);

void orderTaxi(Database& Db);

};

Файл Taxi.cpp

#include "Taxi.h"

#include "Order.h"

#include "../action/InputUtils.h"

Taxi::Taxi(std::string brand, std::string model, std::string color, EngineType engineType, int carId, double pricePerKil, bool hasDriver, bool hasWiFi, bool hasChildSeat, RentCarTypes rentCarTypes)

: Transport(brand, model, color, engineType), carId(carId), pricePerKil(pricePerKil), hasDriver(hasDriver), hasWiFi(hasWiFi), hasChildSeat(hasChildSeat), rentCarTypes(rentCarTypes) {}

Taxi::Taxi() : Transport("", "", "", EngineType::PETROL), carId(0), pricePerKil(0.0), hasDriver(false), hasWiFi(false), hasChildSeat(false), rentCarTypes(RentCarTypes::ECONOMY) {}

int Taxi::getCarId(){

return carId;

}

void Taxi::setCarId(int carId) {

this->carId = carId;

}

double Taxi::getPricePerKil() {

return pricePerKil;

}

void Taxi::setPricePerKil(double pricePerKil) {

this->pricePerKil = pricePerKil;

}

bool Taxi::getHasDriver() {

return hasDriver;

}

void Taxi::setHasDriver(bool hasDriver) {

this->hasDriver = hasDriver;

}

bool Taxi::getHasWiFi() {

return hasWiFi;

}

void Taxi::setHasWiFi(bool hasWiFi) {

this->hasWiFi = hasWiFi;

}

bool Taxi::getHasChildSeat() {

return hasChildSeat;

}

void Taxi::setHasChildSeat(bool hasChildSeat) {

this->hasChildSeat = hasChildSeat;

}

RentCarTypes Taxi::getRentCarTypes() {

return rentCarTypes;

}

void Taxi::setRentCarTypes(RentCarTypes rentCarTypes) {

this->rentCarTypes = rentCarTypes;

}

Taxi::~Taxi() {

}

RentCarTypes Taxi::stringToRentCarTypes(const std::string& rentCarTypesStr) {

if (rentCarTypesStr == "ECONOMY") {

return RentCarTypes::ECONOMY;

} else if (rentCarTypesStr == "COMFORT") {

return RentCarTypes::COMFORT;

} else if (rentCarTypesStr == "BUSINESS") {

return RentCarTypes::BUSINESS;

} else {

std::cerr << "Error: Invalid rent car type - unknown value: " << rentCarTypesStr << std::endl;

throw std::runtime\_error("Invalid rent car type: " + rentCarTypesStr);

}

}

void Taxi::displayTaxiDetails(const pqxx::result::const\_iterator& row) {

try {

int carId = row[0].as<int>();

std::string brand = row[1].as<std::string>();

std::string model = row[2].as<std::string>();

std::string color = row[3].as<std::string>();

EngineType engineType = stringToEngineType(row["engineType"].as<std::string>());

double pricePerKil = row[5].as<double>();

bool hasDriver = row[6].as<bool>();

bool hasWiFi = row[7].as<bool>();

bool hasChildSeat = row[8].as<bool>();

RentCarTypes rentCarTypes = stringToRentCarTypes(row["rent\_car\_type"].as<std::string>());

std::unique\_ptr<Taxi> taxi = std::make\_unique<Taxi>(

brand, model, color,

engineType,

carId, pricePerKil,

hasDriver, hasWiFi, hasChildSeat,

rentCarTypes

);

std::cout << "Taxi Details:" << std::endl;

std::cout << "Car ID: " << taxi->getCarId() << std::endl;

std::cout << "Brand: " << taxi->getBrand() << std::endl;

std::cout << "Model: " << taxi->getModel() << std::endl;

std::cout << "Color: " << taxi->getColor() << std::endl;

std::cout << "Engine Type: " << taxi->getEngineTypeString() << std::endl;

std::cout << "Price per Kilometer: " << taxi->getPricePerKil() << std::endl;

std::cout << "Has Driver: " << (hasDriver ? "Yes" : "No") << std::endl;

std::cout << "Has WiFi: " << (hasWiFi ? "Yes" : "No") << std::endl;

std::cout << "Has Child Seat: " << (hasChildSeat ? "Yes" : "No") << std::endl;

std::cout << "Rent Car Type: " << taxi->getRentCarTypes() << std::endl;

std::cout << std::endl;

} catch (const std::exception &e) {

std::cerr << "Error displaying taxi details: " << e.what() << std::endl;

}

}

void Taxi::displayAllTaxis(Database& Db) {

try {

pqxx::result R = Db.executeQuery("SELECT \* FROM taxi");

if (R.empty()) {

std::cout << "No taxis found in the database." << std::endl;

return;

}

std::cout << "Printing query result:" << std::endl;

for (const auto& row : R) {

for (const auto& field : row) {

std::cout << field.name() << ": " << field.c\_str() << "\t";

}

std::cout << std::endl;

}

for (pqxx::result::const\_iterator c = R.begin(); c != R.end(); ++c) {

displayTaxiDetails(c);

}

} catch (const std::exception &e) {

std::cerr << "Error displaying all taxis: " << e.what() << std::endl;

}

}

void Taxi::displayTaxiById(Database& Db, int taxiId) {

try {

pqxx::result R = Db.executeQuery("SELECT \* FROM taxi WHERE taxi\_id = " + std::to\_string(taxiId));

if (R.empty()) {

std::cout << "No taxi found with ID " << taxiId << std::endl;

return;

}

for (const auto& row : R) {

displayTaxiDetails(row);

}

} catch (const std::exception &e) {

std::cerr << "Error displaying taxi by ID: " << e.what() << std::endl;

}

}

void Taxi::displayTaxisByBrand(Database& Db, const std::string& brand) {

try {

pqxx::result R = Db.executeQuery("SELECT \* FROM taxi WHERE brand = '" + brand + "'");

if (R.empty()) {

std::cout << "No taxis found with brand '" << brand << "'" << std::endl;

return;

}

for (const auto& row : R) {

displayTaxiDetails(row);

}

} catch (const std::exception &e) {

std::cerr << "Error displaying taxis by brand: " << e.what() << std::endl;

}

}

void Taxi::displayTaxisByRentCarType(Database& Db, RentCarTypes rentCarType) {

try {

std::string rentCarTypeStr;

switch (rentCarType) {

case ECONOMY:

rentCarTypeStr = "ECONOMY";

break;

case COMFORT:

rentCarTypeStr = "COMFORT";

break;

case BUSINESS:

rentCarTypeStr = "BUSINESS";

break;

default:

std::cerr << "Invalid rent car type." << std::endl;

return;

}

pqxx::result R = Db.executeQuery("SELECT \* FROM taxi WHERE rent\_car\_type = '" + rentCarTypeStr + "'");

for (const auto& row : R) {

displayTaxiDetails(row);

}

} catch (const std::exception &e) {

std::cerr << e.what() << std::endl;

}

}

std::string Taxi::to\_string(const int value) {

std::ostringstream oss;

oss << value;

return oss.str();

}

void Taxi::orderTaxi(Database& Db) {

try {

int carId = InputUtils::getPositiveInput<int>("Enter the car ID: ");

std::string checkTaxiQuery = "SELECT EXISTS(SELECT 1 FROM taxi WHERE taxi\_id = " + to\_string(carId) + ")";

pqxx::result taxiExistsResult = Db.executeQuery(checkTaxiQuery);

bool taxiExists = taxiExistsResult[0][0].as<bool>();

if (!taxiExists) {

std::cout << "Taxi with ID " << carId << " does not exist. Please enter a valid taxi ID." << std::endl;

return;

}

std::string name = InputUtils::getStringInput("Enter your name: ");

std::string surname = InputUtils::getStringInput("Enter your surname: ");

std::string contactInformation = InputUtils::getStringInput("Enter your contact information: ");

std::string addCustomerQuery = "INSERT INTO customer (name, surname, contact\_information) VALUES ('" +

name + "','" +

surname + "','" +

contactInformation + "')";

Db.executeQuery(addCustomerQuery);

pqxx::result customerIdResult = Db.executeQuery("SELECT MAX(customer\_id) FROM customer");

int customerId = customerIdResult[0][0].as<int>();

Order order;

order.setCustomerId(customerId);

order.setCarId(carId);

std::string addOrderQuery = "INSERT INTO orders (customer\_id, car\_id, order\_time) VALUES (" +

to\_string(customerId) + "," +

to\_string(carId) + ",'" +

order.getOrderTime() + "')";

Db.executeQuery(addOrderQuery);

std::cout << "Order placed successfully! Your taxi ID is " << carId << "." << std::endl;

} catch (const std::exception& e) {

std::cerr << e.what() << std::endl;

}

}

Файл Stop.h

#in clude <iostream>

#include "../database/Database.h"

class Stop {

private:

int stopId;

std::string stopName;

std::string address;

public:

Stop(int stopId, const std::string& stopName, const std::string& stopAddress);

Stop();

int getStopId();

void setStopId(int stopId);

std::string getStopName();

void setStopName(std::string stopName);

std::string getAddress();

void setAddress( std::string address);

static std::vector<Stop> findAllStops(Database& db);

static std::optional<Stop> findStopById(Database& db, int stopId);

};

Файл Stop.cpp

#include "Stop.h"

Stop::Stop(int stopId, const std::string& stopName, const std::string& stopAddress)

: stopId(stopId), stopName(stopName), address(stopAddress) {}

Stop::Stop() : stopId(0), stopName(""), address("") {

}

int Stop::getStopId() {

return stopId;

}

void Stop::setStopId(int stopId) {

this->stopId = stopId;

}

std::string Stop::getStopName() {

return stopName;

}

void Stop::setStopName(std::string stopName) {

this->stopName = stopName;

}

std::string Stop::getAddress() {

return address;

}

void Stop::setAddress(std::string address) {

this->address = address;

}

std::vector<Stop> Stop::findAllStops(Database& db) {

std::vector<Stop> stops;

pqxx::result R = db.executeQuery("SELECT \* FROM Stop;");

for (auto row : R) {

stops.emplace\_back(row["stop\_id"].as<int>(), row["stop\_name"].as<std::string>(), row["address"].as<std::string>());

std::cout << "Stop ID: " << row["stop\_id"].as<int>() << ", Name: " << row["stop\_name"].as<std::string>() << ", Address: " << row["address"].as<std::string>() << std::endl;

}

return stops;

}

std::optional<Stop> Stop::findStopById(Database& db, int stopId) {

pqxx::result R = db.executeQuery("SELECT \* FROM Stop WHERE stop\_id = " + std::to\_string(stopId) + ";");

if (!R.empty()) {

auto row = R[0];

std::cout << "Found Stop: " << row["stop\_name"].as<std::string>() << ", Address: " << row["address"].as<std::string>() << std::endl;

return std::optional<Stop>(Stop(row["stop\_id"].as<int>(), row["stop\_name"].as<std::string>(), row["address"].as<std::string>()));

} else {

std::cout << "No stop found with ID: " << stopId << std::endl;

return std::nullopt;

}

}

Файл Route.h

#include <iostream>

#include <vector>

#include "Stop.h"

#include "../database/Database.h"

#include "PublicTransport.h"

class Route {

private:

int routeId;

std::string routeName;

std::vector<Stop> stops;

public:

Route();

Route(int routeId, const std::string& routeName, const std::vector<Stop>& routeStops);

int getRouteId();

std::string getRouteName();

std::vector<Stop> getStops();

void setRouteId(int routeId);

void setRouteName(std::string routeName);

void setStops(std::vector<Stop> stops);

void getStopsForRoute(Database& db, int routeId);

void getRoutesForTransport(Database& db, int transportId, PublicTransport::TransportType transportType);

void displayAllRoutes(Database& db);

};

Файл Route.cpp

#include "Route.h"

Route::Route() : routeId(0) {}

Route::Route(int routeId, const std::string& routeName, const std::vector<Stop>& routeStops)

: routeId(routeId), routeName(routeName), stops(routeStops) {}

int Route::getRouteId(){ return routeId; }

std::string Route::getRouteName(){ return routeName; }

std::vector<Stop> Route::getStops(){ return stops; }

void Route::setRouteId(int routeId) {

this->routeId = routeId;

}

void Route::setRouteName(std::string routeName) {

this->routeName = routeName;

}

void Route::setStops(std::vector<Stop> stops) {

this->stops = stops;

}

void Route::getStopsForRoute(Database& db, int routeId) {

std::cout << "Attempting to get stops for Route ID: " << routeId << std::endl;

pqxx::result routeCheck = db.executeQuery("SELECT route\_id FROM Route WHERE route\_id = " + std::to\_string(routeId));

if (routeCheck.empty()) {

std::cout << "Route ID: " << routeId << " does not exist in the Route table." << std::endl;

return;

}

pqxx::result routeStopCheck = db.executeQuery("SELECT stop\_id FROM RouteStop WHERE route\_id = " + std::to\_string(routeId));

if (routeStopCheck.empty()) {

std::cout << "No stops are linked to Route ID: " << routeId << " in the RouteStop table." << std::endl;

return;

}

try {

pqxx::result result = db.executeQuery("SELECT s.stop\_name, s.address "

"FROM Stop s "

"JOIN RouteStop rs ON s.stop\_id = rs.stop\_id "

"WHERE rs.route\_id = " + std::to\_string(routeId));

if (!result.empty()) {

for (const auto& row : result) {

std::cout << "Stop Name: " << row[0].as<std::string>() << ", Address: " << row[1].as<std::string>() << std::endl;

}

} else {

std::cout << "No stops found for the specified route." << std::endl;

}

} catch (const std::exception& e) {

std::cerr << "Error: " << e.what() << std::endl;

}

}

void Route::getRoutesForTransport(Database& db, int transportId, PublicTransport::TransportType transportType) {

std::string transportTypeName = (transportType == PublicTransport::TransportType::BUS) ? "BUS" : "TROLLEYBUS";

std::cout << "Attempting to get routes for Transport: " << transportTypeName << " with ID: " << transportId << std::endl;

try {

pqxx::result result = db.executeQuery("SELECT r.route\_id, r.route\_name "

"FROM Route r "

"JOIN (SELECT route\_id FROM TransportRoute WHERE transport\_id = " +

std::to\_string(transportId) +

" AND transport\_type = '" + transportTypeName + "') tr ON r.route\_id = tr.route\_id");

if (!result.empty()) {

for (const auto& row : result) {

std::cout << "Route ID: " << row[0].as<int>() << ", Route Name: " << row[1].as<std::string>() << std::endl;

}

} else {

std::cout << "No routes found for the specified transport." << std::endl;

}

} catch (const std::exception& e) {

std::cerr << "Error: " << e.what() << std::endl;

}

}

void Route::displayAllRoutes(Database& db) {

try {

pqxx::result result = db.executeQuery("SELECT \* FROM Route");

if (!result.empty()) {

std::cout << "All Routes:" << std::endl;

for (const auto& row : result) {

int routeId = row[0].as<int>();

std::string routeName = row[1].as<std::string>();

std::cout << "Route ID: " << routeId << ", Route Name: " << routeName << std::endl;

std::cout << std::endl;

}

} else {

std::cout << "No routes found in the database." << std::endl;

}

} catch (const std::exception& e) {

std::cerr << "Error: " << e.what() << std::endl;

}

}

Файл Order.h

#include <iostream>

#include <string>

#include <chrono>

#include <sstream>

#include "../database/Database.h"

class Order {

private:

int orderId;

int customerId;

int carId;

std::string orderTime;

public:

Order();

Order(int orderId, int customerId, int carId, const std::string& orderTime);

int getOrderId();

void setOrderId(int orderId);

int getCustomerId();

void setCustomerId(int customerId);

int getCarId();

void setCarId(int carId);

std::string getOrderTime();

void setOrderTime(std::string orderTime);

std::string getCurrentTime();

static void printAllOrders(Database &db, int isLogged);

};

Файл Order.cpp

#include <vector>

#include "order.h"

#include "../database/Database.h"

Order::Order() : orderId(0), customerId(0), carId(0), orderTime(getCurrentTime()) {}

Order::Order(int orderId, int customerId, int carId, const std::string& orderTime)

: orderId(orderId), customerId(customerId), carId(carId), orderTime(orderTime) {}

int Order::getOrderId() {

return orderId;

}

void Order::setOrderId(int orderId) {

this->orderId = orderId;

}

int Order::getCustomerId() {

return customerId;

}

void Order::setCustomerId(int customerId) {

this->customerId = customerId;

}

int Order::getCarId() {

return carId;

}

void Order::setCarId(int carId){

this->carId = carId;

}

std::string Order::getOrderTime() {

return orderTime;

}

void Order::setOrderTime(std::string orderTime) {

this->orderTime = orderTime;

}

std::string Order::getCurrentTime() {

auto now = std::chrono::system\_clock::now();

std::time\_t currentTime = std::chrono::system\_clock::to\_time\_t(now);

std::stringstream ss;

ss << std::ctime(&currentTime);

return ss.str();

}

void Order::printAllOrders(Database& db, int isLogged) {

pqxx::result result = db.executeQuery("SELECT \* FROM orders");

std::cout << "All Orders:" << std::endl;

for (auto row : result) {

std::cout << "Order ID: " << row["order\_id"].as<int>() << ", Customer ID: " << row["customer\_id"].as<int>()

<< ", Car ID: " << row["car\_id"].as<int>() << ", Order Time: " << row["order\_time"].as<std::string>() << std::endl;

}

}

Файл Customer.h

#include <iostream>

class Customer {

private:

int customerId;

std::string name;

std::string surname;

std::string contactInformation;

public:

Customer();

int getCustomerId();

void setCustomerId(int customerId);

std::string getName();

void setName(std::string name);

std::string getSurname();

void setSurname(std::string surname);

std::string getContactInformation();

void setContactInformation(std::string contactInformation);

};

Файл Customer.cpp

#include "customer.h"

Customer::Customer() {

customerId = 0;

}

int Customer::getCustomerId() {

return customerId;

}

void Customer::setCustomerId(int customerId) {

this->customerId = customerId;

}

std::string Customer::getName() {

return name;

}

void Customer::setName(std::string name) {

this->name = name;

}

std::string Customer::getSurname() {

return surname;

}

void Customer::setSurname(std::string surname) {

this->surname = surname;

}

std::string Customer::getContactInformation() {

return contactInformation;

}

void Customer::setContactInformation(std::string contactInformation) {

this->contactInformation = contactInformation;

}

Файл TransportTicket.h

#include <iostream>

#include <chrono>

#include <sstream>

#include "../../database/Database.h"

enum TransportType{

BUS,

TROLLEYBUS

};

class TransportTicket {

private:

int ticketId;

int transportId;

int customerId;

double price;

std::string purchaseTime;

TransportType transportType;

Database\* db;

public:

TransportTicket();

TransportTicket(int ticketId, int transportId, int customerId, double price, const std::string& purchaseTime, TransportType transportType);

TransportTicket(Database\* db) : db(db) {}

int getTicketId();

int getTransportId();

int getCustomerId();

double getPrice();

std::string getPurchaseTime();

TransportType getTransportType();

void setCustomerId(int customerId);

void setTransportId(int transportId);

void setPrice(double price);

void displayTicketInfo();

};

Файл TransportTicket.cpp

#include "TransportTicket.h"

TransportTicket::TransportTicket()

: ticketId(0), transportId(0), customerId(0), price(0.0), purchaseTime(""), transportType(TransportType::BUS) {}

TransportTicket::TransportTicket(int ticketId, int transportId, int customerId, double price, const std::string& purchaseTime, TransportType transportType)

: ticketId(ticketId), transportId(transportId), customerId(customerId), price(price), purchaseTime(purchaseTime), transportType(transportType) {}

int TransportTicket::getTicketId() {

return ticketId;

}

int TransportTicket::getTransportId(){

return transportId;

}

int TransportTicket::getCustomerId(){

return customerId;

}

double TransportTicket::getPrice(){

return price;

}

std::string TransportTicket::getPurchaseTime(){

return purchaseTime;

}

TransportType TransportTicket::getTransportType(){

return transportType;

}

void TransportTicket::setCustomerId(int customerId) {

this->customerId = customerId;

}

void TransportTicket::setTransportId(int transportId) {

this->transportId = transportId;

}

void TransportTicket::setPrice(double price) {

this->price = price;

}

void TransportTicket::displayTicketInfo() {

std::cout << "Ticket ID: " << getTicketId() << std::endl;

std::cout << "Transport ID: " << getTransportId() << std::endl;

std::cout << "Customer ID: " << getCustomerId() << std::endl;

std::cout << "Price: " << getPrice() << std::endl;

std::cout << "Purchase Time: " << getPurchaseTime() << std::endl;

std::cout << "Transport Type: " << (getTransportType() == TransportType::BUS ? "Bus" : "Trolleybus") << std::endl;

std::cout << std::endl;

}

Файл Schedule.h

#include <vector>

#include <string>

#include "../../database/Database.h"

#include "../PublicTransport.h"

class Schedule {

private:

int scheduleId;

int routeId;

int stopId;

TransportType transportType;

int transportId;

std::string arrivalTime;

public:

Schedule();

Schedule(int scheduleId, int routeId, int stopId, TransportType transportType, int transportId, const std::string& arrivalTime);

int getScheduleId();

void setScheduleId(int scheduleId);

int getRouteId();

void setRouteId(int routeId);

int getStopId();

void setStopId(int stopId);

TransportType getTransportType();

void setTransportType(TransportType transportType);

int getTransportId();

void setTransportId(int transportId);

std::string getArrivalTime();

void setArrivalTime(std::string arrivalTime);

static std::vector<Schedule> getScheduleForRoute(Database& db, int routeId);

void printStopsForRoute(Database& db, int routeId);

};

Файл Schedule.cpp

#include "Schedule.h"

#include <iostream>

Schedule::Schedule(int scheduleId, int routeId, int stopId, TransportType transportType, int transportId, const std::string& arrivalTime)

: scheduleId(scheduleId), routeId(routeId), stopId(stopId), transportType(transportType), transportId(transportId), arrivalTime(arrivalTime) {}

Schedule::Schedule()

: scheduleId(0), routeId(0), stopId(0), transportType(TransportType::BUS), transportId(0), arrivalTime("") {}

int Schedule::getScheduleId() {

return scheduleId;

}

void Schedule::setScheduleId(int scheduleId) {

this->scheduleId = scheduleId;

}

int Schedule::getRouteId() {

return routeId;

}

void Schedule::setRouteId(int routeId) {

this->routeId = routeId;

}

int Schedule::getStopId() {

return stopId;

}

void Schedule::setStopId(int stopId) {

this->stopId = stopId;

}

TransportType Schedule::getTransportType() {

return transportType;

}

void Schedule::setTransportType(TransportType transportType) {

this->transportType = transportType;

}

int Schedule::getTransportId() {

return transportId;

}

void Schedule::setTransportId(int transportId) {

this->transportId = transportId;

}

std::string Schedule::getArrivalTime() {

return arrivalTime;

}

void Schedule::setArrivalTime(std::string arrivalTime) {

this->arrivalTime = arrivalTime;

}

std::vector<Schedule> Schedule::getScheduleForRoute(Database& db, int routeId) {

pqxx::result R = db.executeQuery("SELECT \* FROM Schedule WHERE route\_id = " + std::to\_string(routeId) + ";");

std::vector<Schedule> scheduleList;

for (auto row: R) {

scheduleList.emplace\_back(row["schedule\_id"].as<int>(),

row["route\_id"].as<int>(),

row["stop\_id"].as<int>(),

static\_cast<TransportType>(row["transport\_type"].as<int>()), // Assuming transport\_type is stored as an integer in the database

row["transport\_id"].as<int>(),

row["arrival\_time"].as<std::string>());

}

return scheduleList;

}

void Schedule::printStopsForRoute(Database& db, int routeId) {

pqxx::result R = db.executeQuery("SELECT s.stop\_id, s.stop\_name, s.address, sch.arrival\_time "

"FROM Stop s "

"JOIN StopRoute sr ON s.stop\_id = sr.stop\_id "

"JOIN Schedule sch ON sr.route\_id = sch.route\_id AND sr.stop\_id = sch.stop\_id "

"WHERE sr.route\_id = " + std::to\_string(routeId) + " "

"ORDER BY sr.stop\_id;");

std::cout << "Stops for route ID " << routeId << ":" << std::endl;

for (auto row : R) {

std::cout << "Stop ID: " << row["stop\_id"].as<int>() << ", Stop Name: " << row["stop\_name"].as<std::string>()

<< ", Address: " << row["address"].as<std::string>() << ", Arrival Time: " << row["arrival\_time"].as<std::string>() << std::endl;

}

}

Файл RoutePrice.h

#ifndef COURSEWORK\_ROUTEPRICE\_H

#define COURSEWORK\_ROUTEPRICE\_H

#include "pqxx/pqxx"

#include "../../database/Database.h"

class RoutePrice {

private:

int routeId;

double price;

public:

RoutePrice();

RoutePrice(int id, double price);

int getRouteId();

double getPrice();

static RoutePrice fromSQLResult(const pqxx::row& row);

RoutePrice getTicketPrice(Database& db, int routeId);

void displayTicketPrice(Database& db, int routeId);

};

Файл RoutePrice.cpp

#include "routeprice.h"

RoutePrice::RoutePrice() : routeId(0), price(0.0) {}

RoutePrice::RoutePrice(int id, double price) : routeId(id), price(price) {}

int RoutePrice::getRouteId() {

return routeId;

}

double RoutePrice::getPrice() {

return price;

}

RoutePrice RoutePrice::fromSQLResult(const pqxx::row& row) {

int id = row["route\_id"].as<int>();

double price = row["price"].as<double>();

return RoutePrice(id, price);

}

RoutePrice RoutePrice::getTicketPrice(Database& db, int routeId) {

pqxx::result result = db.executeQuery("SELECT route\_id, price FROM RoutePrice WHERE route\_id = " + std::to\_string(routeId));

if (!result.empty()) {

return RoutePrice::fromSQLResult(result[0]);

} else {

throw std::runtime\_error("No price found for the specified route.");

}

}

void RoutePrice::displayTicketPrice(Database& db, int routeId) {

try {

RoutePrice routePrice = getTicketPrice(db, routeId);

std::cout << "The price for the ticket on Route ID: " << routePrice.getRouteId()

<< " is $" << routePrice.getPrice() << std::endl;

} catch (const std::exception& e) {

std::cerr << "Error: " << e.what() << std::endl;

}

}

Файл Database.h

class Database {

private:

pqxx::connection C;

public:

Database() : C("dbname=CourseWorkDb user=postgres password=aboba hostaddr=127.0.0.1 port=5432") {

if (C.is\_open()) {

std::cout << "Opened database successfully: " << C.dbname() << std::endl;

} else {

std::cout << "Can't open database" << std::endl;

throw std::runtime\_error("Failed to open database");

}

}

~Database() {

}

pqxx::result executeQuery(const std::string& query) {

pqxx::nontransaction N(C);

return N.exec(query);

}

};

Файл Admin.h

#include <iostream>

#include <pqxx/pqxx>

#include <stdexcept>

#include "../database/Database.h"

#include "../models/Taxi.h"

#include "../models/Bus.h"

#include "../models/Trolleybus.h"

class Admin {

private:

int adminId;

std::string username;

std::string hashedPassword;

Database &db;

std::string hashPassword(const std::string &password);

public:

Admin(Database &db, const std::string &username, const std::string &password);

bool login();

bool checkAuthorization(int isLogged);

bool isValidRouteID(Database &db, int routeId);

std::string engineTypeToString(EngineType engineType);

std::string rentCarTypesToString(RentCarTypes rentCarTypes);

bool registerAdmin(Database &db, const std::string &username, const std::string &password);

void addBus(const std::string &brand, const std::string &model, const std::string &color, EngineType engineType,

int capacity, bool hasContactlessPayment, int isLogged);

void addTrolleyBus(const std::string &brand, const std::string &model, const std::string &color,

EngineType engineType, int capacity, bool hasSockets, int isLogged);

void addTaxi(const std::string &brand, const std::string &model, const std::string &color, EngineType engineType,

double pricePerKilometer, bool hasDriver, bool hasWiFi, bool hasChildSeat, RentCarTypes rentCarTypes,

int isLogged);

void addStop(const std::string &stopName, const std::string &address, int isLogged);

void addRoute(const std::string &routeName, int isLogged);

void addSchedule(Database& db, int routeId, TransportType transportType, int transportId, int isLogged);

void setRoutePrice(Database &Db, int routeId, double price, int isLogged);

void linkTransportToRoute(Database &Db, int route\_id, TransportType transport\_type, int transport\_id, int isLogged);

void linkStopToRoute(Database &db, int routeId, int stopId, int isLoggedIn);

bool adminLogin(Database &db, const std::string &username, const std::string &password);

};

Файл Admin.cpp

#include <regex>

#include "Admin.h"

Admin::Admin(Database& db, const std::string& username, const std::string& password)

: db(db), username(username), hashedPassword(hashPassword(password)) {

}

std::string Admin::hashPassword(const std::string &password) {

std::string hashed;

for (char c: password) {

hashed += std::to\_string((c + 3) % 256);

}

return hashed;

}

bool Admin::checkAuthorization(int isLogged) {

if (!isLogged) {

std::cerr << "Error: You must be logged in as an admin to link transport to a route." << std::endl;

return false;

}

return true;

}

bool Admin::login() {

std::string query =

"SELECT admin\_id FROM Admins WHERE username = '" + username + "' AND password = '" + hashedPassword +

"'";

pqxx::result result = db.executeQuery(query);

if (!result.empty()) {

adminId = result[0]["admin\_id"].as<int>();

std::cout << "Login successful for admin ID: " << adminId << std::endl;

return true;

} else {

std::cout << "Login failed for username: " << username << std::endl;

return false;

}

}

std::string Admin::engineTypeToString(EngineType engineType) {

switch (engineType) {

case DIESEL:

return "DIESEL";

case PETROL:

return "PETROL";

case ELECTRIC:

return "ELECTRIC";

case HYBRID:

return "HYBRID";

default:

return "UNKNOWN";

}

}

std::string Admin::rentCarTypesToString(RentCarTypes rentCarTypes) {

switch (rentCarTypes) {

case ECONOMY:

return "ECONOMY";

break;

case COMFORT:

return "COMFORT";

break;

case BUSINESS:

return "BUSINESS";

break;

default:

return "UNKNOWN";

break;

}

};

bool Admin::isValidRouteID(Database &db, int routeId) {

std::string query = "SELECT EXISTS(SELECT 1 FROM Route WHERE route\_id = " + std::to\_string(routeId) + ");";

auto result = db.executeQuery(query);

return !result.empty() && result[0][0].as<bool>();

}

bool Admin::adminLogin(Database &db, const std::string &username, const std::string &password) {

Admin admin(db, username, password);

return admin.login();

}

bool Admin::registerAdmin(Database &db, const std::string &username, const std::string &password) {

std::string hashedPassword;

for (char c: password) {

hashedPassword += std::to\_string((c + 3) % 256);

}

std::string query = "INSERT INTO Admins (username, password) VALUES ('"

+ username + "', '" + hashedPassword + "')";

try {

db.executeQuery(query);

std::cout << "Admin registered successfully." << std::endl;

return true;

} catch (const std::exception &e) {

std::cerr << "Error registering admin: " << e.what() << std::endl;

return false;

}

}

void Admin::addBus(const std::string &brand, const std::string &model, const std::string &color, EngineType engineType,

int capacity, bool hasContactlessPayment, int isLogged) {

if (!checkAuthorization(isLogged)) {

return;

}

std::string query =

"INSERT INTO bus (brand, model, color, engineType, capacity, has\_contactless\_payment) VALUES ('"

+ brand + "', '" + model + "', '" + color + "', '" + engineTypeToString(engineType) + "', "

+ std::to\_string(capacity) + ", " + (hasContactlessPayment ? "TRUE" : "FALSE") + ")";

try {

db.executeQuery(query);

std::cout << "Bus added successfully." << std::endl;

} catch (const std::exception &e) {

std::cerr << "Error adding bus: " << e.what() << std::endl;

}

}

void Admin::addTrolleyBus(const std::string &brand, const std::string &model, const std::string &color, EngineType engineType,

int capacity, bool hasSockets, int isLogged) {

if (!checkAuthorization(isLogged)) {

return;

}

std::string query = "INSERT INTO trolleybus (brand, model, color, engineType, capacity, has\_sockets) VALUES ('"

+ brand + "', '" + model + "', '" + color + "', '" + engineTypeToString(engineType) + "', "

+ std::to\_string(capacity) + ", " + (hasSockets ? "TRUE" : "FALSE") + ")";

try {

db.executeQuery(query);

std::cout << "TrolleyBus added successfully." << std::endl;

} catch (const std::exception &e) {

std::cerr << "Error adding trolleybus: " << e.what() << std::endl;

}

}

void Admin::addTaxi(const std::string &brand, const std::string &model, const std::string &color, EngineType engineType,

double pricePerKilometer, bool hasDriver, bool hasWiFi, bool hasChildSeat, RentCarTypes rentCarTypes,

int isLogged) {

if (!checkAuthorization(isLogged)) {

return;

}

std::string rentCarTypeStr = rentCarTypesToString(rentCarTypes);

std::string engineTypeStr = engineTypeToString(engineType);

std::string query =

"INSERT INTO taxi (brand, model, color, engineType, price\_per\_kil, has\_driver, has\_wifi, has\_child\_seat, rent\_car\_type) VALUES ('"

+ brand + "', '" + model + "', '" + color + "', '" + engineTypeStr + "', " +

std::to\_string(pricePerKilometer) + ", "

+ (hasDriver ? "TRUE" : "FALSE") + ", " + (hasWiFi ? "TRUE" : "FALSE") + ", " +

(hasChildSeat ? "TRUE" : "FALSE") + ", '"

+ rentCarTypeStr + "')";

try {

db.executeQuery(query);

std::cout << "Taxi added successfully." << std::endl;

} catch (const std::exception &e) {

std::cerr << "Error adding taxi: " << e.what() << std::endl;

}

}

void Admin::addStop(const std::string &stopName, const std::string &address, int isLogged) {

if (!checkAuthorization(isLogged)) {

return;

}

std::string query = "INSERT INTO Stop (stop\_name, address) VALUES ('"

+ stopName + "', '" + address + "')";

try {

db.executeQuery(query);

std::cout << "Stop added successfully." << std::endl;

} catch (const std::exception &e) {

std::cerr << "Error adding stop: " << e.what() << std::endl;

}

}

void Admin::addRoute(const std::string &routeName, int isLogged) {

if (!checkAuthorization(isLogged)) {

return;

}

std::string query = "INSERT INTO Route (route\_name) VALUES ('"

+ routeName + "')";

try {

db.executeQuery(query);

std::cout << "Route added successfully." << std::endl;

} catch (const std::exception &e) {

std::cerr << "Error adding route: " << e.what() << std::endl;

}

}

void Admin::addSchedule(Database& db, int routeId, TransportType transportType, int transportId, int isLogged) {

if (!checkAuthorization(isLogged)) {

return;

}

std::string transportTypeStr = (transportType == BUS) ? "BUS" : "TROLLEYBUS";

std::string transportTable = (transportType == BUS) ? "bus" : "trolleybus";

std::string transportIdColumn = (transportType == BUS) ? "bus\_id" : "trolleybus\_id";

std::string checkTransportRouteQuery = "SELECT EXISTS(SELECT 1 FROM TransportRoute WHERE route\_id = " +

std::to\_string(routeId) + " AND transport\_type = '" +

transportTypeStr + "' AND transport\_id = " +

std::to\_string(transportId) + ");";

auto transportRouteExists = db.executeQuery(checkTransportRouteQuery);

if (transportRouteExists.empty() || !transportRouteExists[0][0].as<bool>()) {

std::cerr << "Transport is not linked to the route." << std::endl;

return;

}

std::string getStopsQuery = "SELECT stop\_id FROM StopRoute WHERE route\_id = " + std::to\_string(routeId) + ";";

auto stops = db.executeQuery(getStopsQuery);

std::regex timeRegex(R"((2[0-3]|[01]?[0-9]):([0-5]?[0-9]):([0-5]?[0-9]))");

for (const auto& stop : stops) {

int stopId = stop[0].as<int>();

std::string arrivalTime;

do {

std::cout << "Enter arrival time for stop ID " << stopId << " (HH:MM:SS): ";

std::cin >> arrivalTime;

if (!std::regex\_match(arrivalTime, timeRegex)) {

std::cerr << "Error: Invalid time format. Please enter the time in HH:MM:SS format." << std::endl;

}

} while (!std::regex\_match(arrivalTime, timeRegex));

std::string insertScheduleQuery = "INSERT INTO Schedule (route\_id, stop\_id, transport\_type, transport\_id, arrival\_time) VALUES (" +

std::to\_string(routeId) + ", " +

std::to\_string(stopId) + ", '" +

transportTypeStr + "', " +

std::to\_string(transportId) + ", '" +

arrivalTime + "');";

db.executeQuery(insertScheduleQuery);

}

std::cout << "Schedule added successfully for all linked stops with individual arrival times." << std::endl;

}

void Admin::setRoutePrice(Database &Db, int routeId, double price, int isLogged) {

if (!checkAuthorization(isLogged)) {

return;

}

if (!isValidRouteID(Db, routeId)) {

std::cerr << "Invalid route ID provided." << std::endl;

return;

}

std::string checkQuery = "SELECT price FROM RoutePrice WHERE routeId = " + std::to\_string(routeId);

auto checkResult = Db.executeQuery(checkQuery);

if (!checkResult.empty()) {

std::string updateQuery = "UPDATE RoutePrice SET price = " + std::to\_string(price) + " WHERE routeId = " +

std::to\_string(routeId);

Db.executeQuery(updateQuery);

std::cout << "Route price updated successfully." << std::endl;

} else {

std::string insertQuery =

"INSERT INTO RoutePrice (routeId, price) VALUES (" + std::to\_string(routeId) + ", " +

std::to\_string(price) + ")";

Db.executeQuery(insertQuery);

std::cout << "Route price set successfully." << std::endl;

}

}

void Admin::linkTransportToRoute(Database &Db, int routeId, TransportType transportType, int transport\_id, int isLogged) {

if (!checkAuthorization(isLogged)) {

return;

}

if (!isValidRouteID(Db, routeId)) {

std::cerr << "Invalid route ID provided." << std::endl;

return;

}

std::string transportTypeStr = (transportType == BUS) ? "BUS" : "TROLLEYBUS";

std::string transportTable = (transportType == BUS) ? "bus" : "trolleybus";

std::string transportIdColumn = (transportType == BUS) ? "bus\_id" : "trolleybus\_id";

std::string checkTransportQuery = "SELECT EXISTS(SELECT 1 FROM " + transportTable +

" WHERE " + transportIdColumn + " = " + std::to\_string(transport\_id) + ");";

auto transportExists = Db.executeQuery(checkTransportQuery);

if (transportExists.empty() || !transportExists[0][0].as<bool>()) {

std::cerr << "Transport ID does not exist in the database." << std::endl;

return;

}

std::string checkQuery = "SELECT 1 FROM TransportRoute WHERE route\_id = " + std::to\_string(routeId) +

" AND transport\_type = '" + transportTypeStr + "' AND transport\_id = " +

std::to\_string(transport\_id);

auto checkResult = Db.executeQuery(checkQuery);

if (!checkResult.empty()) {

std::cerr << "This transport is already linked to the route." << std::endl;

return;

}

try {

std::string insertQuery = "INSERT INTO TransportRoute (route\_id, transport\_type, transport\_id) VALUES (" +

std::to\_string(routeId) + ", '" + transportTypeStr + "', " +

std::to\_string(transport\_id) + ")";

Db.executeQuery(insertQuery);

std::cout << "Transport linked to route successfully." << std::endl;

} catch (const pqxx::unique\_violation &e) {

std::cerr << "Error: Transport is already linked to this route." << std::endl;

} catch (const std::exception &e) {

std::cerr << "Error: " << e.what() << std::endl;

}

}

void Admin::linkStopToRoute(Database &db, int routeId, int stopId, int isLogged) {

if (!checkAuthorization(isLogged)) {

return;

}

std::string checkRouteQuery = "SELECT EXISTS(SELECT 1 FROM Route WHERE route\_id = " + std::to\_string(routeId) + ");";

auto routeExists = db.executeQuery(checkRouteQuery);

if (routeExists.empty() || !routeExists[0][0].as<bool>()) {

std::cerr << "Invalid route ID provided." << std::endl;

return;

}

std::string checkStopQuery = "SELECT EXISTS(SELECT 1 FROM Stop WHERE stop\_id = " + std::to\_string(stopId) + ");";

auto stopExists = db.executeQuery(checkStopQuery);

if (stopExists.empty() || !stopExists[0][0].as<bool>()) {

std::cerr << "Invalid stop ID provided." << std::endl;

return;

}

std::string checkLinkQuery = "SELECT 1 FROM StopRoute WHERE route\_id = " + std::to\_string(routeId) +

" AND stop\_id = " + std::to\_string(stopId) + ";";

auto checkLinkResult = db.executeQuery(checkLinkQuery);

if (!checkLinkResult.empty()) {

std::cerr << "This stop is already linked to the route." << std::endl;

return;

}

std::string insertQuery = "INSERT INTO StopRoute (route\_id, stop\_id) VALUES (" +

std::to\_string(routeId) + ", " +

std::to\_string(stopId) + ");";

db.executeQuery(insertQuery);

std::cout << "Stop linked to route successfully." << std::endl;

}

Файл MainMenu.h

#include "../database/Database.h"

#include "../models/PublicTransport.h"

void displayMenu();

void bookTransport(Database& Db, PublicTransport::TransportType transportType);

void orderTaxi(Database& Db);

void handleTaxiSelect(Database& Db);

void handleBusSelect(Database& Db);

void handleTrolleyBusSelect(Database& Db);

void createAndAddBus(Database& Db, int isLogged);

void createAndAddTrolleyBus(Database& Db, int isLogged);

void createAndAddTaxi(Database& Db, int isLogged);

void createAndAddStop(Database& Db, int isLogged);

void createAndAddRoute(Database& Db, int isLogged);

void createAndAddSchedule(Database& Db, int isLogged);

void createAndSetRoutePrice(Database& Db, int isLogged);

void createAndLinkTransportToRoute(Database& Db, int isLogged);

void createAndLinkStopToRoute(Database& Db, int isLogged);

void handleAdminActions(Database& Db);

Файл MainMenu.cpp

#include "routeprice.h"

#include <string>

#include <iostream>

#include "pqxx/pqxx"

#include "../database/Database.h"

#include "../models/Taxi.h"

#include "../models/Order.h"

#include "../models/Bus.h"

#include "../models/Route.h"

#include "../models/Trolleybus.h"

#include "../models/price/RoutePrice.h"

#include "../admin/Admin.h"

#include "../models/schedule/Schedule.h"

#include "InputUtils.h"

using namespace std;

void displayMenu() {

cout << "Choose the type of transport:" << endl;

cout << "1. Taxi" << endl;

cout << "2. Bus" << endl;

cout << "3. Trolleybus" << endl;

cout << "4. Admin panel" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice (1/2/3/4/0): ";

}

RentCarTypes parseEnumRentCarType(int rentCarTypeInput) {

RentCarTypes rentCarTypes;

switch (rentCarTypeInput) {

case 1:

rentCarTypes = ECONOMY;

break;

case 2:

rentCarTypes = COMFORT;

break;

case 3:

rentCarTypes = BUSINESS;

break;

default:

cout << "Invalid choice" << endl;

break;

}

return rentCarTypes;

}

EngineType parseEnumEngineType(int engineTypeInput) {

EngineType engineType;

switch (engineTypeInput) {

case 1:

engineType = DIESEL;

break;

case 2:

engineType = PETROL;

break;

case 3:

engineType = HYBRID;

break;

case 4:

engineType = ELECTRIC;

break;

default:

std::cerr << "Invalid engine type selected." << std::endl;

break;

}

return engineType;

}

void handleTaxiSelect(Database& Db) {

bool exitMenu = false;

while (!exitMenu) {

Taxi taxi;

cout << "Choose an action:" << endl;

cout << "1. View all taxis" << endl;

cout << "2. Call taxi" << endl;

cout << "3. Display taxi by ID" << endl;

cout << "4. Display taxis by brand" << endl;

cout << "5. Display taxis by RentCarType" << endl;

cout << "0. Back to the main menu" << endl;

int actionChoice;

cin >> actionChoice;

cin.ignore();

switch (actionChoice) {

case 1: {

taxi.displayAllTaxis(Db);

break;

}

case 2: {

Taxi taxi;

taxi.orderTaxi(Db);

break;

}

case 3: {

int taxiId = InputUtils::getPositiveInput<int>("Enter taxi ID: ");

taxi.displayTaxiById(Db, taxiId);

break;

}

case 4: {

string brand = InputUtils::getStringInput("Enter brand: ");

taxi.displayTaxisByBrand(Db, brand);

break;

}

case 5: {

cout << "Choose RentCarType:" << endl;

cout << "1. Economy" << endl;

cout << "2. Comfort" << endl;

cout << "3. Business" << endl;

int rentCarTypeChoice;

cin >> rentCarTypeChoice;

cin.ignore();

RentCarTypes rentCarType;

Taxi::displayTaxisByRentCarType(Db, rentCarType);

break;

}

case 0: {

exitMenu = true;

break;

}

default: {

cout << "Invalid choice" << endl;

break;

}

}

}

}

void handleBusSelect(Database& Db) {

bool exitMenu = false;

while (!exitMenu) {

cout << "Choose an action:" << endl;

cout << "1. View all buses" << endl;

cout << "2. View all routes" << endl;

cout << "3. View schedule" << endl;

cout << "4. Get ticket price for a route" << endl;

cout << "5. Stop case" << endl;

cout << "6. Book a ticket" << endl;

cout << "0. Back to main menu" << endl;

cout << "Enter your choice (1/2/3/4/5/6/0): ";

int actionChoice;

cin >> actionChoice;

cin.ignore();

switch (actionChoice) {

case 1: {

Bus bus;

bus.displayAllBuses(Db);

break;

}

case 2: {

int busId = InputUtils::getPositiveInput<int>("Enter bus ID to find linked routes: ");

Route route;

route.getRoutesForTransport(Db, busId, PublicTransport::BUS);

int choice = InputUtils::getPositiveInput<int>("Wanna see all routes? (1 - yes, 2 - no): ");

if (choice == 1) {

route.displayAllRoutes(Db);

}

break;

}

case 3: {

int routeId = InputUtils::getPositiveInput<int>("Enter route ID: ");

Schedule transportSchedule;

transportSchedule.printStopsForRoute(Db, routeId);

break;

}

case 4: {

int routeId = InputUtils::getPositiveInput<int>("Enter the route ID to get the ticket price: ");

RoutePrice routePrice;

routePrice.displayTicketPrice(Db, routeId);

break;

}

case 5: {

Stop::findAllStops(Db);

int stopId = InputUtils::getPositiveInput<int>("Enter stop ID: ");

Stop::findStopById(Db, stopId);

break;

}

case 6: {

PublicTransport publicTransport;

publicTransport.bookTransport(Db, PublicTransport::BUS);

break;

}

case 0: {

exitMenu = true;

break;

}

default:

cout << "Invalid choice. Please try again." << endl;

}

}

}

void handleTrolleyBusSelect(Database& Db) {

bool exitMenu = false;

while (!exitMenu) {

cout << "Choose an action:" << endl;

cout << "1. View all trolleybuses" << endl;

cout << "2. View all routes" << endl;

cout << "3. View schedule" << endl;

cout << "4. Get ticket price for a route" << endl;

cout << "5. Stop case" << endl;

cout << "6. Book a ticket" << endl;

cout << "0. Back to main menu" << endl;

cout << "Enter your choice (1/2/3/4/5/6/0): ";

int actionChoice;

cin >> actionChoice;

cin.ignore();

switch (actionChoice) {

case 1: {

TrolleyBus trolleyBus;

trolleyBus.displayAllTrolleyBuses(Db);

break;

}

case 2: {

int trolleybusId = InputUtils::getPositiveInput<int>("Enter trolleybus ID to find linked routes: ");

Route route;

route.getRoutesForTransport(Db, trolleybusId, PublicTransport::TROLLEYBUS);

int choice = InputUtils::getPositiveInput<int>("Wanna see all routes? (1 - yes, 2 - no): ");

if (choice == 1) {

route.displayAllRoutes(Db);

}

break;

}

case 3: {

int routeId = InputUtils::getPositiveInput<int>("Enter route ID: ");

Schedule transportSchedule;

transportSchedule.printStopsForRoute(Db, routeId);

break;

}

case 4: {

int routeId = InputUtils::getPositiveInput<int>("Enter the route ID to get the ticket price: ");

RoutePrice routePrice;

routePrice.displayTicketPrice(Db, routeId);

break;

}

case 5: {

Stop::findAllStops(Db);

int stopId = InputUtils::getPositiveInput<int>("Enter stop ID: ");

Stop::findStopById(Db, stopId);

break;

}

case 6: {

PublicTransport publicTransport;

publicTransport.bookTransport(Db, PublicTransport::TROLLEYBUS);

break;

}

case 0: {

exitMenu = true;

break;

}

default:

cout << "Invalid choice. Please try again." << endl;

}

}

}

void createAndAddBus(Database& Db, int isLogged) {

Admin admin(Db, "admin\_username", "admin\_password");

std::string brand, model, color;

EngineType engineType;

int capacity;

bool hasContactlessPayment;

brand = InputUtils::getStringInput("Enter bus brand: ");

model = InputUtils::getStringInput("Enter bus model: ");

color = InputUtils::getStringInput("Enter bus color: ");

int engineTypeInput = InputUtils::getPositiveInput<int>("Enter engine type (1 for DIESEL, 2 for PETROL, etc.): ");

engineType = parseEnumEngineType(engineTypeInput);

capacity = InputUtils::getPositiveInput<int>("Enter bus capacity: ");

int hasContactlessPaymentInput = InputUtils::getPositiveInput<int>("Does the bus have contactless payment? (1 for YES, 0 for NO): ");

hasContactlessPayment = hasContactlessPaymentInput != 0;

admin.addBus(brand, model, color, engineType, capacity, hasContactlessPayment, isLogged);

}

void createAndAddTrolleyBus(Database& Db, int isLogged) {

Admin admin(Db, "admin\_username", "admin\_password");

std::string brand, model, color;

EngineType engineType;

int capacity;

bool hasSockets;

brand = InputUtils::getStringInput("Enter trolleybus brand: ");

model = InputUtils::getStringInput("Enter trolleybus model: ");

color = InputUtils::getStringInput("Enter trolleybus color: ");

int engineTypeInput = InputUtils::getPositiveInput<int>("Enter engine type (1 for DIESEL, 2 for PETROL, etc.): ");

engineType = parseEnumEngineType(engineTypeInput);

capacity = InputUtils::getPositiveInput<int>("Enter trolleybus capacity: ");

int hasSocketsInput = InputUtils::getPositiveInput<int>("Does the trolleybus have electrical sockets? (1 for YES, 0 for NO): ");

hasSockets = hasSocketsInput != 0;

admin.addTrolleyBus(brand, model, color, engineType, capacity, hasSockets, isLogged);

}

void createAndAddTaxi(Database& Db, int isLogged) {

Admin admin(Db, "admin\_username", "admin\_password");

std::string brand, model, color;

EngineType engineType;

double pricePerKilometer;

bool hasDriver, hasWiFi, hasChildSeat;

RentCarTypes carType;

brand = InputUtils::getStringInput("Enter taxi brand: ");

model = InputUtils::getStringInput("Enter taxi model: ");

color = InputUtils::getStringInput("Enter taxi color: ");

int engineTypeInput = InputUtils::getPositiveInput<int>("Enter car type (0 for PETROL, 1 for DIESEL, 2 for HYBRID, 3 for ELECTRIC): ");

engineType = parseEnumEngineType(engineTypeInput);

pricePerKilometer = InputUtils::getPositiveInput<double>("Enter price per kilometer: ");

int hasDriverInput = InputUtils::getPositiveInput<int>("Does the taxi have a driver? (1 for YES, 0 for NO): ");

hasDriver = hasDriverInput != 0;

int hasWiFiInput = InputUtils::getPositiveInput<int>("Does the taxi have WiFi? (1 for YES, 0 for NO): ");

hasWiFi = hasWiFiInput != 0;

int hasChildSeatInput = InputUtils::getPositiveInput<int>("Does the taxi have a child seat? (1 for YES, 0 for NO): ");

hasChildSeat = hasChildSeatInput != 0;

int carTypeInput = InputUtils::getPositiveInput<int>("Enter car type (0 for ECONOMY, 1 for COMFORT, 2 for BUSINESS): ");

carType = parseEnumRentCarType(carTypeInput);

admin.addTaxi(brand, model, color, engineType, pricePerKilometer, hasDriver, hasWiFi, hasChildSeat, carType, isLogged);

}

void createAndAddStop(Database& Db, int isLogged) {

Admin admin(Db, "admin\_username", "admin\_password");

std::string stopName = InputUtils::getStringInput("Enter stop name: ");

std::string address = InputUtils::getStringInput("Enter address: ");

admin.addStop(stopName, address, isLogged);

}

void createAndAddRoute(Database& Db, int isLogged) {

Admin admin(Db, "admin\_username", "admin\_password");

std::string routeName = InputUtils::getStringInput("Enter route name: ");

admin.addRoute(routeName, isLogged);

}

void createAndAddSchedule(Database& db, int isLogged) {

Admin admin(db, "admin\_username", "admin\_password");

int transportId;

int routeId;

std::string arrivalTime;

PublicTransport::TransportType transportType;

transportId = InputUtils::getPositiveInput<int>("Enter transport ID: ");

int transportTypeInput = InputUtils::getPositiveInput<int>("Enter transport type (1 for BUS, 2 for TROLLEYBUS): ");

switch (transportTypeInput) {

case 1:

transportType = PublicTransport::BUS;

break;

case 2:

transportType = PublicTransport::TROLLEYBUS;

break;

default:

std::cerr << "Invalid transport type selected." << std::endl;

return;

}

routeId = InputUtils::getPositiveInput<int>("Enter route ID: ");

admin.addSchedule(db, routeId, static\_cast<TransportType>(transportType), transportId, isLogged);

}

void createAndSetRoutePrice(Database& db, int isLogged) {

Admin admin(db, "admin\_username", "admin\_password");

int routeId;

double price;

routeId = InputUtils::getPositiveInput<int>("Enter route ID: ");

price = InputUtils::getPositiveInput<double>("Enter price: ");

admin.setRoutePrice(db, routeId, price, isLogged);

}

void createAndLinkTransportToRoute(Database& db, int isLogged) {

Admin admin(db, "admin\_username", "admin\_password");

int routeId;

PublicTransport::TransportType transportType;

int transportId;

routeId = InputUtils::getPositiveInput<int>("Enter route ID: ");

int transportTypeInput = InputUtils::getPositiveInput<int>("Enter transport type (1 for BUS, 2 for TROLLEYBUS): ");

switch (transportTypeInput) {

case 1:

transportType = PublicTransport::BUS;

break;

case 2:

transportType = PublicTransport::TROLLEYBUS;

break;

default:

std::cerr << "Invalid transport type selected." << std::endl;

return;

}

transportId = InputUtils::getPositiveInput<int>("Enter transport ID: ");

admin.linkTransportToRoute(db, routeId, static\_cast<TransportType>(transportType), transportId, isLogged);

}

void createAndLinkStopToRoute(Database& db, int isLogged) {

Admin admin(db, "admin\_username", "admin\_password");

int routeId, stopId;

routeId = InputUtils::getPositiveInput<int>("Enter route ID: ");

stopId = InputUtils::getPositiveInput<int>("Enter stop ID: ");

admin.linkStopToRoute(db, routeId, stopId, isLogged);

}

void handleAdminActions(Database& Db) {

Admin admin(Db, "admin\_username", "admin\_password");

std::cout << "1. Login\n2. Register\nChoose an option: ";

int choice;

std::cin >> choice;

cin.ignore();

std::string username, password;

int isLogged = 0;

switch (choice) {

case 1:

username = InputUtils::getStringInput("Enter username: ");

password = InputUtils::getStringInput("Enter password: ");

if (admin.adminLogin(Db, username, password)) {

std::cout << "Logged in successfully." << std::endl;

isLogged = 1;

} else {

std::cout << "Login failed." << std::endl;

}

break;

case 2:

username = InputUtils::getStringInput("Enter username for registration: ");

password = InputUtils::getStringInput("Enter password for registration: ");

if (admin.registerAdmin(Db, username, password)) {

std::cout << "Registered successfully." << std::endl;

isLogged = 1;

} else {

std::cout << "Registration failed." << std::endl;

}

break;

default:

std::cout << "Invalid option selected." << std::endl;

break;

}

while (isLogged == 1) {

int action = InputUtils::getPositiveInput<int>("Select an action:\n"

"1. Add Bus\n"

"2. Add Trolleybus\n"

"3. Add Taxi\n"

"4. Set Stop\n"

"5. Set Route\n"

"6. Set Schedule\n"

"7. Set Price\n"

"8. Set link transport to route\n"

"9. createAndLinkStopToRoute\n"

"10. Get orders\n"

"0. Logout\n"

"Enter your choice: ");

switch (action) {

case 1:

createAndAddBus(Db, isLogged);

break;

case 2:

createAndAddTrolleyBus(Db, isLogged);

break;

case 3:

createAndAddTaxi(Db, isLogged);

break;

case 4:

createAndAddStop(Db, isLogged);

break;

case 5:

createAndAddRoute(Db, isLogged);

break;

case 6:

createAndAddSchedule(Db, isLogged);

break;

case 7:

createAndSetRoutePrice(Db, isLogged);

break;

case 8:

createAndLinkTransportToRoute(Db, isLogged);

break;

case 9:

createAndLinkStopToRoute(Db, isLogged);

break;

case 10:

Order::printAllOrders(Db, isLogged);

break;

case 0:

isLogged = false;

std::cout << "Logged out successfully." << std::endl;

break;

default:

std::cout << "Invalid action selected. Please try again." << std::endl;

break;

}

}

}

Файл InputUtils.h

#include <string>

#include <iostream>

#include <numeric>

class InputUtils {

public:

template <typename T>

static T getPositiveInput(const std::string &prompt) {

T value;

std::string input;

while (true) {

std::cout << prompt;

std::getline(std::cin, input);

try {

size\_t pos = 0;

value = std::stod(input, &pos);

if (pos == input.length() && value >= 0) {

break;

} else {

std::cout << "Error: Input is not a positive number. Please try again." << std::endl;

}

} catch (std::invalid\_argument&) {

std::cout << "Error: Input is not a number. Please try again." << std::endl;

} catch (std::out\_of\_range&) {

std::cout << "Error: Input is too large. Please try again." << std::endl;

}

std::cin.clear();

//std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

return value;

}

static std::string getStringInput(const std::string &prompt) {

std::string input;

while (true) {

std::cout << prompt;

std::getline(std::cin, input);

if (!std::cin.fail()) {

break;

} else {

std::cout << "Error: Input failed. Please try again." << std::endl;

}

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

return input;

}

};