**Ministry of education and science of the Kyrgyz Republic**

**Kyrgyz State Technical University named after I.Razzakov**

**Faculty of Information Technologies**

**Department of Software of Computer Systems**

**Major: 710400 «Software Engineering»**

Report

Discipline: «**Object-Oriented Design**»

Software requirements

Done by**:** student of the SE(eng)- 1- 21

Shumkarbekov Sultan

Checked by: Musabaev E.B.

Bishkek 2024

Table of contents

[Introduction 3](#_Toc165013399)

[Functional requirements: 3](#_Toc165013400)

[Non-functional requirements: 3](#_Toc165013401)

[Use case 4](#_Toc165013402)

[Class diagram 5](#_Toc165013403)

[Code 5](#_Toc165013404)

# Introduction

The purpose of this document is to draw up a technical specification for the development of the information system "Buyer's Directory", designed for convenient access to information about retail outlets in the city. The system will contain a database with the main characteristics of the stores, such as name, address, contact numbers, specialization, form of ownership and opening hours. The main task of the system will be to provide users with the opportunity to select stores according to various criteria, which will allow them to quickly find the necessary stores according to their needs and preferences.

# Functional requirements:

Administrator Authentication:

The system must allow the administrator to log in by entering his username and password.

Adding a store:

The administrator should be able to add a new store, specifying its name, address, phone numbers, specialization, form of ownership and working hours.

Deleting a store:

The administrator should be able to delete an existing store from the list.

Editing the store:

The administrator must be able to edit information about the store, including its name, address, phone numbers, specialization, form of ownership and working hours.

Viewing the list of stores:

The user (administrator or regular user) should be able to view the list of all stores.

Search for stores :

The user should be able to search for stores by a given template (for example, by name, address, specialization, etc.).

Saving data:

The system must save information about stores after they are added, deleted or edited.

# Non-functional requirements:

Safety:

The system must provide data protection, including administrator authentication.

Efficiency:

The system must be efficient and provide quick access to store data.

Interface:

The system interface should be intuitive and user-friendly for use by the administrator and ordinary users.

Reliability:

The system must be reliable and ensure the safety of data in all situations, including program failures.

Scalability:

The system must be scalable and capable of handling a large number of stores if necessary.

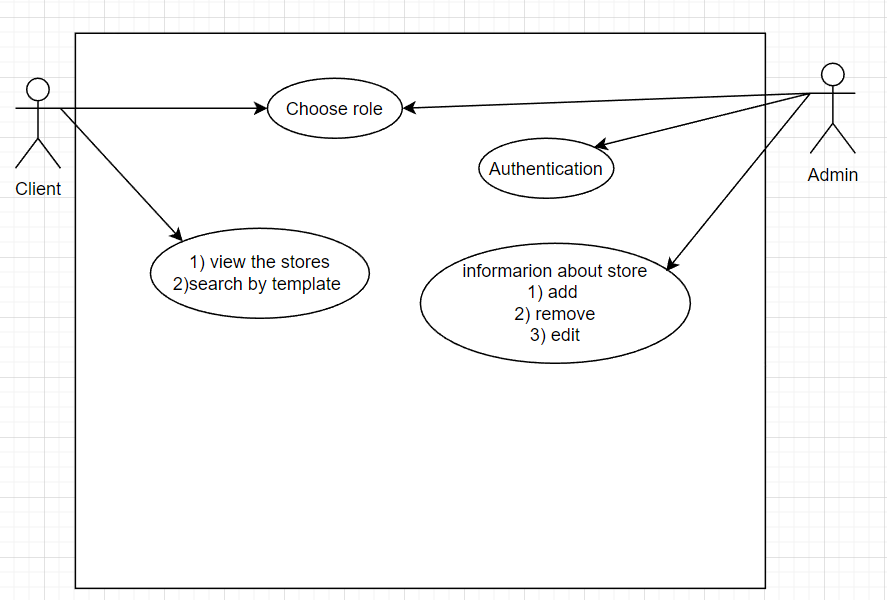
Availability:

The system must be available for use at any time and from any place where there is internet access.

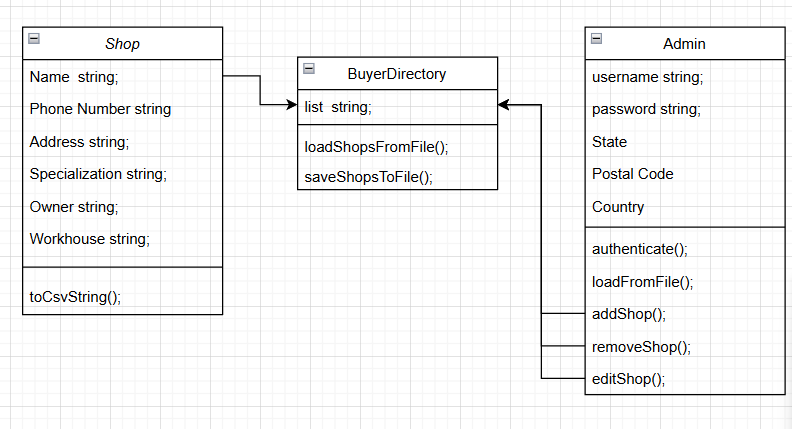
Support:

The system should be able to be updated and maintained to fix bugs and add new features.

# Use case



# Class diagram



# Code

#include <iostream>

#include <fstream>

#include <sstream>

#include <vector>

#include <string>

class Shop {

public:

std::string name;

std::string address;

std::vector<std::string> phones;

std::string specialization;

std::string ownership;

std::string workingHours;

Shop() = default;

Shop(const std::string& n, const std::string& addr, const std::vector<std::string>& ph,

const std::string& spec, const std::string& own, const std::string& wh)

: name(n), address(addr), phones(ph), specialization(spec), ownership(own), workingHours(wh) {}

void print() const {

std::cout << "Name: " << name << std::endl;

std::cout << "Address: " << address << std::endl;

std::cout << "Phones: ";

for (const auto& phone : phones) {

std::cout << phone << " ";

}

std::cout << std::endl;

std::cout << "Specialization: " << specialization << std::endl;

std::cout << "Ownership: " << ownership << std::endl;

std::cout << "Working Hours: " << workingHours << std::endl;

std::cout << std::endl;

}

std::string toCsvString() const {

std::stringstream ss;

ss << name << "," << address << ",";

for (const auto& phone : phones) {

ss << phone << ";";

}

ss << "," << specialization << "," << ownership << "," << workingHours;

return ss.str();

}

void fromCsvString(const std::string& csvString) {

std::stringstream ss(csvString);

std::getline(ss, name, ',');

std::getline(ss, address, ',');

phones.clear();

std::string phone;

std::getline(ss, phone, ',');

std::stringstream phoneStream(phone);

while (std::getline(phoneStream, phone, ';')) {

phones.push\_back(phone);

}

std::getline(ss, specialization, ',');

std::getline(ss, ownership, ',');

std::getline(ss, workingHours, ',');

}

};

class Admin {

private:

std::string username;

std::string password;

public:

Admin() = default;

Admin(const std::string& uname, const std::string& pwd)

: username(uname), password(pwd) {}

bool authenticate(const std::string& uname, const std::string& pwd) const {

return (username == uname && password == pwd);

}

void loadFromFile(const std::string& filename) {

std::ifstream file(filename);

if (file.is\_open()) {

std::getline(file, username);

std::getline(file, password);

file.close();

}

}

void saveToFile(const std::string& filename) const {

std::ofstream file(filename);

if (file.is\_open()) {

file << username << std::endl;

file << password << std::endl;

file.close();

}

}

void addShop(std::vector<Shop>& shops, const std::string& name, const std::string& address, const std::vector<std::string>& phones,

const std::string& specialization, const std::string& ownership, const std::string& workingHours) {

shops.push\_back(Shop(name, address, phones, specialization, ownership, workingHours));

}

void removeShop(std::vector<Shop>& shops, int index) {

if (index >= 0 && index < shops.size()) {

shops.erase(shops.begin() + index);

std::cout << "Shop successfully removed." << std::endl;

} else {

std::cout << "Invalid shop index." << std::endl;

}

}

void editShop(std::vector<Shop>& shops, int index, const std::string& name, const std::string& address, const std::vector<std::string>& phones,

const std::string& specialization, const std::string& ownership, const std::string& workingHours) {

if (index >= 0 && index < shops.size()) {

Shop& shop = shops[index];

shop.name = name;

shop.address = address;

shop.phones = phones;

shop.specialization = specialization;

shop.ownership = ownership;

shop.workingHours = workingHours;

std::cout << "Shop successfully edited." << std::endl;

} else {

std::cout << "Invalid shop index." << std::endl;

}

}

};

class BuyerDirectory {

public:

std::vector<Shop> shops;

void loadShopsFromFile(const std::string& filename) {

std::ifstream file(filename);

if (file.is\_open()) {

shops.clear();

std::string line;

while (std::getline(file, line)) {

Shop shop;

shop.fromCsvString(line);

shops.push\_back(shop);

}

file.close();

}

}

void saveShopsToFile(const std::string& filename) const {

std::ofstream file(filename);

if (file.is\_open()) {

for (const auto& shop : shops) {

file << shop.toCsvString() << std::endl;

}

file.close();

}

}

void addShop(const Shop& shop) {

shops.push\_back(shop);

}

void removeShop(int index) {

if (index >= 0 && index < shops.size()) {

shops.erase(shops.begin() + index);

std::cout << "Shop successfully removed." << std::endl;

} else {

std::cout << "Invalid shop index." << std::endl;

}

}

void editShop(int index) {

if (index >= 0 && index < shops.size()) {

Shop& shop = shops[index];

std::string name, address, specialization, ownership, workingHours;

std::vector<std::string> phones;

std::cout << "Enter new shop name: ";

std::cin >> name;

shop.name = name;

std::cout << "Enter new shop address: ";

std::cin >> address;

shop.address = address;

std::cout << "Enter new shop specialization: ";

std::cin >> specialization;

shop.specialization = specialization;

std::cout << "Enter new shop ownership: ";

std::cin >> ownership;

shop.ownership = ownership;

std::cout << "Enter new shop working hours: ";

std::cin >> workingHours;

shop.workingHours = workingHours;

std::cout << "Enter new shop phones (type 'end' to finish): ";

std::string phone;

while (true) {

std::cin >> phone;

if (phone == "end") {

break;

}

phones.push\_back(phone);

}

shop.phones = phones;

std::cout << "Shop successfully edited." << std::endl;

} else {

std::cout << "Invalid shop index." << std::endl;

}

}

std::vector<Shop> selectShopsByTemplate(const std::string& templateStr) const {

std::vector<Shop> selectedShops;

for (const auto& shop : shops) {

if (shop.name.find(templateStr) != std::string::npos ||

shop.address.find(templateStr) != std::string::npos ||

shop.specialization.find(templateStr) != std::string::npos ||

shop.ownership.find(templateStr) != std::string::npos ||

shop.workingHours.find(templateStr) != std::string::npos) {

selectedShops.push\_back(shop);

}

}

return selectedShops;

}

void printAllShops() const {

for (const auto& shop : shops) {

shop.print();

}

}

};

int main() {

BuyerDirectory directory;

directory.loadShopsFromFile("C:\\Users\\Lenovo\\OneDrive\\Документы\\shops.txt");

Admin admin("admin", "admin123");

admin.loadFromFile("C:\\Users\\Lenovo\\OneDrive\\Документы\\admin.txt");

while (true) {

std::string role;

std::cout << "Enter your role (user-1 or admin-2): ";

std::cin >> role;

if (role == "1") {

if (directory.shops.empty()) {

std::cout << "The list of shops is currently empty." << std::endl;

} else {

int currentShopIndex = 0;

while (true) {

if (currentShopIndex < directory.shops.size()) {

std::cout << "Information about shop " << currentShopIndex + 1 << ":" << std::endl;

directory.shops[currentShopIndex].print();

std::string choice;

std::cout << "Do you want to continue viewing (yes-1/no-0)? ";

std::cin >> choice;

if (choice != "1") {

break;

}

currentShopIndex++;

} else {

std::cout << "You have viewed all shops." << std::endl;

break;

}

}

std::string searchChoice;

std::cout << "Do you want to search for shops by template (yes-1/no-0)? ";

std::cin >> searchChoice;

if (searchChoice == "1") {

std::string templateStr;

std::cout << "Enter search template: ";

std::cin.ignore(); // ignore the newline character left in the buffer

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

std::vector<Shop> selectedShops = directory.selectShopsByTemplate(templateStr);

if (selectedShops.empty()) {

std::cout << "No shops found matching the template." << std::endl;

} else {

std::cout << "Shops found matching the template:" << std::endl;

for (const auto& shop : selectedShops) {

shop.print();

}

}

}

}

}

else if (role == "2") {

std::string username, password;

std::cout << "Enter admin username: ";

std::cin >> username;

std::cout << "Enter password: ";

std::cin >> password;

if (admin.authenticate(username, password)) {

std::cout << "Authentication successful!" << std::endl;

do {

std::string action;

std::cout << "Choose action (add-1, remove-2, edit-3, search-4): ";

std::cin >> action;

if (action == "1") {

std::string name, address, specialization, ownership, workingHours;

std::vector<std::string> phones;

std::cout << "Enter shop name: ";

std::cin >> name;

std::cout << "Enter shop address: ";

std::cin >> address;

std::cout << "Enter shop specialization: ";

std::cin >> specialization;

std::cout << "Enter shop ownership: ";

std::cin >> ownership;

std::cout << "Enter shop working hours: ";

std::cin >> workingHours;

std::cout << "Enter shop phones (type 'end' to finish): ";

std::string phone;

while (true) {

std::cin >> phone;

if (phone == "end") {

break;

}

phones.push\_back(phone);

}

admin.addShop(directory.shops, name, address, phones, specialization, ownership, workingHours);

directory.saveShopsToFile("C:\\Users\\Lenovo\\OneDrive\\Документы\\shops.txt");

} else if (action == "2") {

int index;

std::cout << "Enter shop index to remove: ";

std::cin >> index;

admin.removeShop(directory.shops, index);

directory.saveShopsToFile("C:\\Users\\Lenovo\\OneDrive\\Документы\\shops.txt");

} else if (action == "3") {

int index;

std::cout << "Enter shop index to edit: ";

std::cin >> index;

std::string name, address, specialization, ownership, workingHours;

std::vector<std::string> phones;

std::cout << "Enter new shop name: ";

std::cin >> name;

std::cout << "Enter new shop address: ";

std::cin >> address;

std::cout << "Enter new shop specialization: ";

std::cin >> specialization;

std::cout << "Enter new shop ownership: ";

std::cin >> ownership;

std::cout << "Enter new shop working hours: ";

std::cin >> workingHours;

std::cout << "Enter new shop phones (type 'end' to finish): ";

std::string phone;

while (true) {

std::cin >> phone;

if (phone == "end") {

break;

}

phones.push\_back(phone);

}

admin.editShop(directory.shops, index, name, address, phones, specialization, ownership, workingHours);

directory.saveShopsToFile("C:\\Users\\Lenovo\\OneDrive\\Документы\\shops.txt");

} else if (action == "4") {

std::string templateStr;

std::cout << "Enter search template: ";

std::cin.ignore(); // ignore the newline character left in the buffer

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

std::vector<Shop> selectedShops = directory.selectShopsByTemplate(templateStr);

if (selectedShops.empty()) {

std::cout << "No shops found matching the template." << std::endl;

} else {

std::cout << "Shops found matching the template:" << std::endl;

for (const auto& shop : selectedShops) {

shop.print();

}

}

} else {

std::cout << "Invalid action." << std::endl;

}

std::string addMore;

std::cout << "Do you want to continue editing shops (yes-1/no-0)? ";

std::cin >> addMore;

if (addMore != "1") {

break;

}

} while (true);

} else {

std::cout << "Invalid admin username or password." << std::endl;

}

} else {

std::cout << "Invalid role." << std::endl;

}

directory.saveShopsToFile("C:\\Users\\Lenovo\\OneDrive\\Документы\\shops.txt");

std::string continueChoice;

std::cout << "Do you want to continue (yes-1/no-0)? ";

std::cin >> continueChoice;

if (continueChoice != "1") {

break;

}

}

directory.saveShopsToFile("C:\\Users\\Lenovo\\OneDrive\\Документы\\shops.txt");

return 0;

}