**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**федеральное государственное бюджетное образовательное учреждение высшего профессионального образования**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**Кафедра Систем обработки данных**

**КУРСОВОЙ ПРОЕКТ**

по дисциплине \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование учебной дисциплины согласно учебному плану)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

Тема проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Автор: студент гр. \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_/

(шифр группы) (подпись) (Ф.И.О.)

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

руководитель проекта \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_/

(должность) (подпись) (Ф.И.О.)

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

ОСНОВНАЯ ЧАСТЬ

ЗАМЫСЕЛ

Основная задача заключается в разработке программного обеспечения, которое поможет людям в организации поиска пропавших домашних животных.

**Направление развития:**

Сопоставление объявлений о пропаже и находке.

Создание базы данных для служб помощи животным.

**Границы реализации:**

Поддержка основных видов питомцев: собаки, кошки.

Локальное хранение данных.

Консольный интерфейс управления.

**Ключевые возможности системы:**

Автоматическое определение вероятных совпадений.

Статистический анализ эффективности поиска.

СОСТАВ ПРОЕКТА

Файловая структура всего проекта показана на рисунке 1.

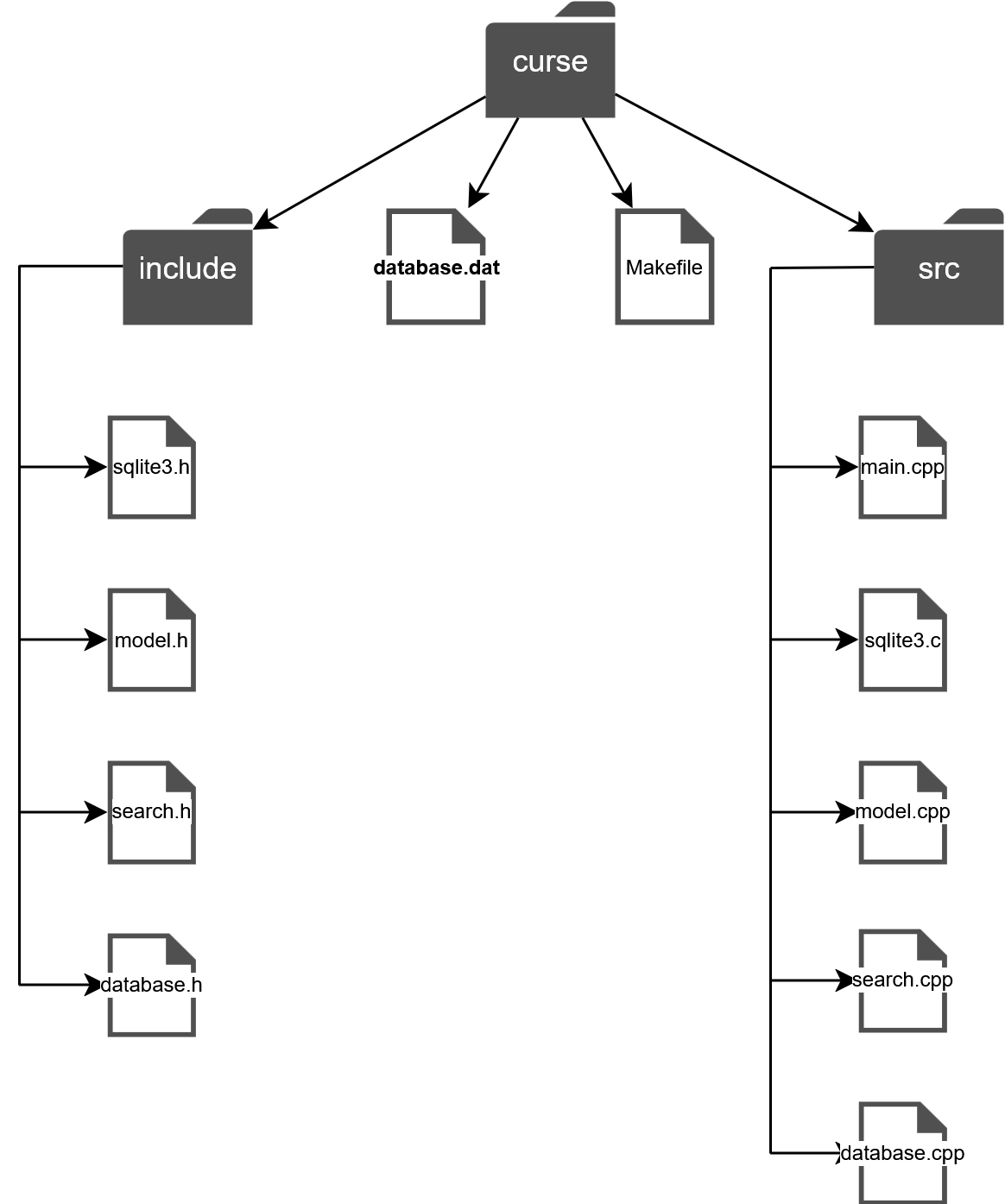


Рисунок 1 – файловая структура проекта

Диаграмма классов для файла model.h представлена в соответствии с рисунком 2. В ней отражена иерархия классов Animal, Shelter, VetClinic и Announcement с использованием наследования, инкапсуляции и полиморфизма, что обеспечивает гибкую и расширяемую модель предметной области.

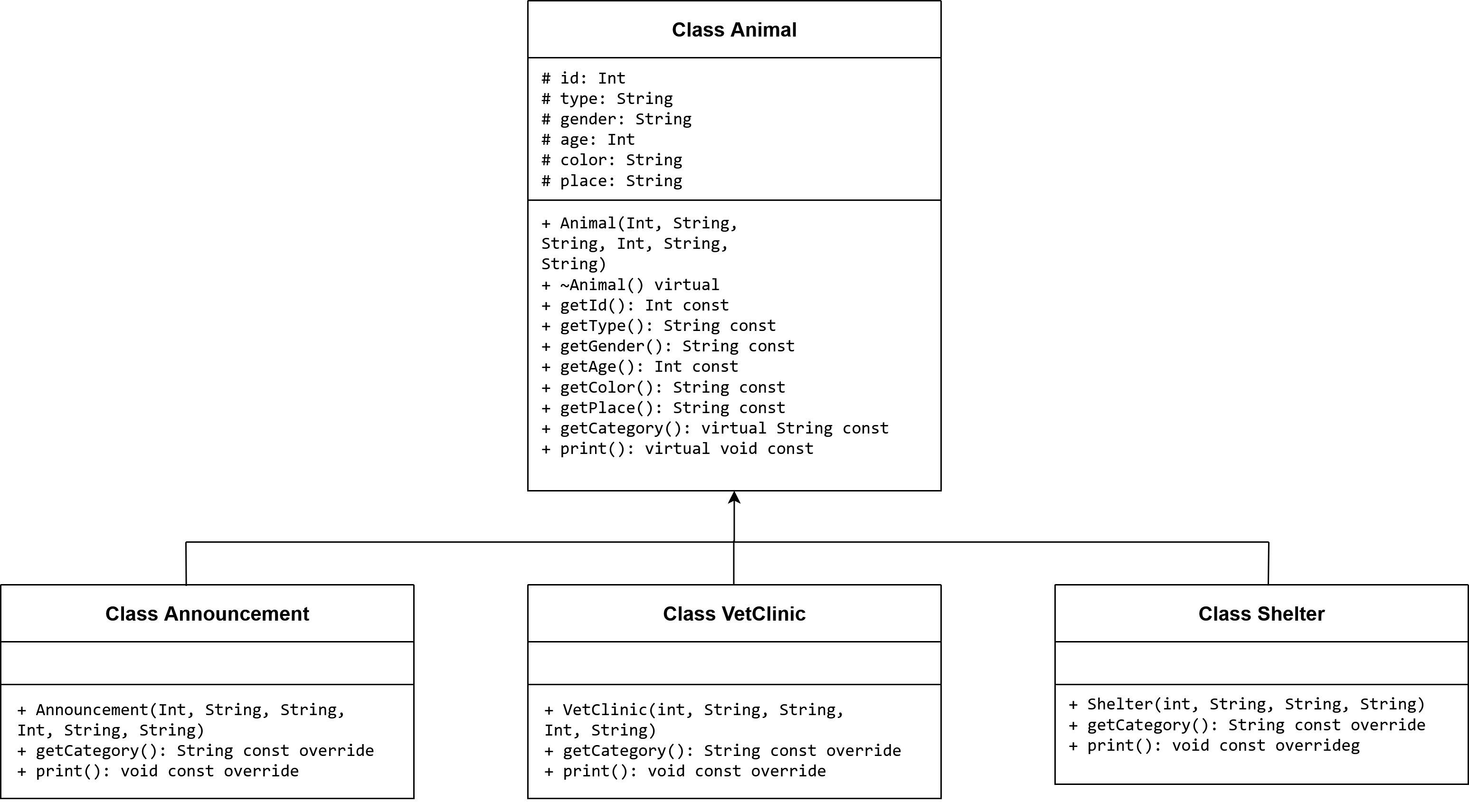


Рисунок 2 – Диаграмма классов

В database.h реализована работа с базой данных (sqlite3) в соответствии с рисунком 3. Класс Database инкапсулирует логику взаимодействия с SQLite3, включая открытие и закрытие базы, выполнение SQL-запросов, а также загрузку и сохранение данных в виде объектов модели. Используется управление ресурсами и callback-функции для обработки результатов запросов.

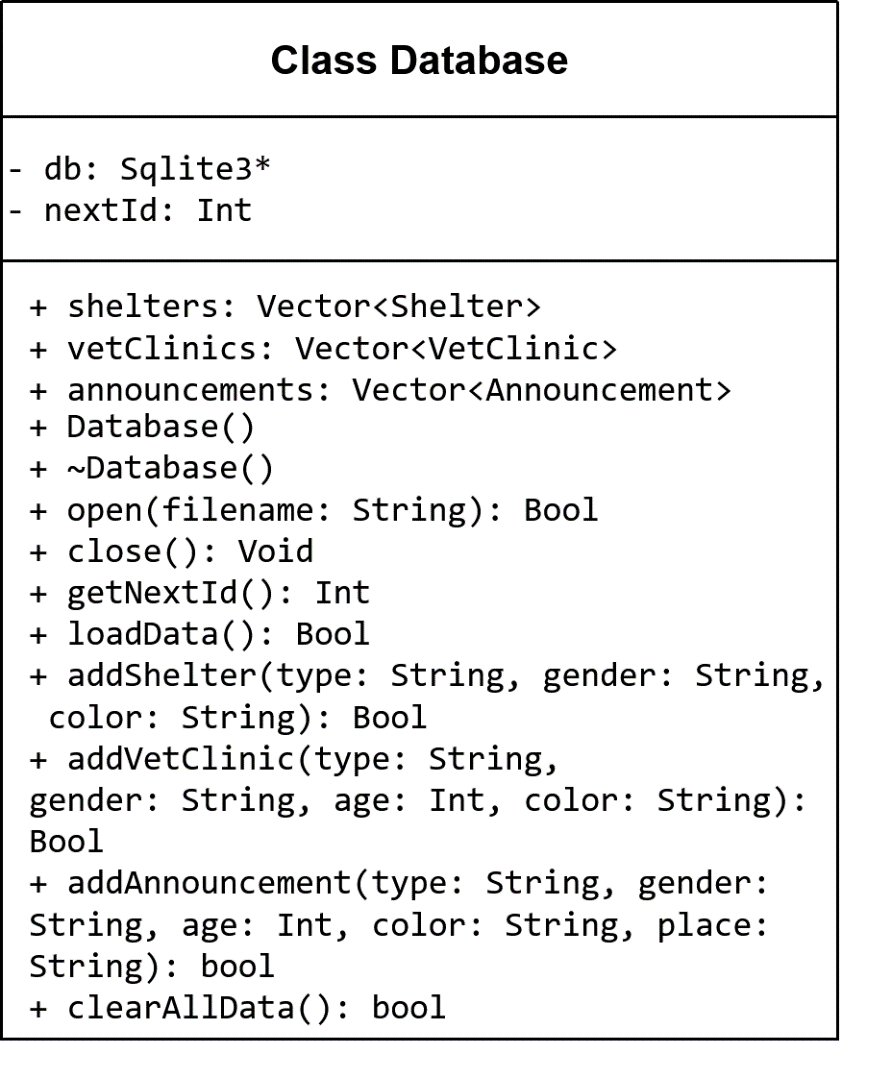


Рисунок 3 – Класс базы данных

В файле search.h был реализован основной функционал поиска, в соответствии с рисунком 4, потерянных животных по различным базам данных. Функции поиска используют сравнение объектов Animal из разных категорий с помощью вычисления процента сходства по нескольким параметрам.

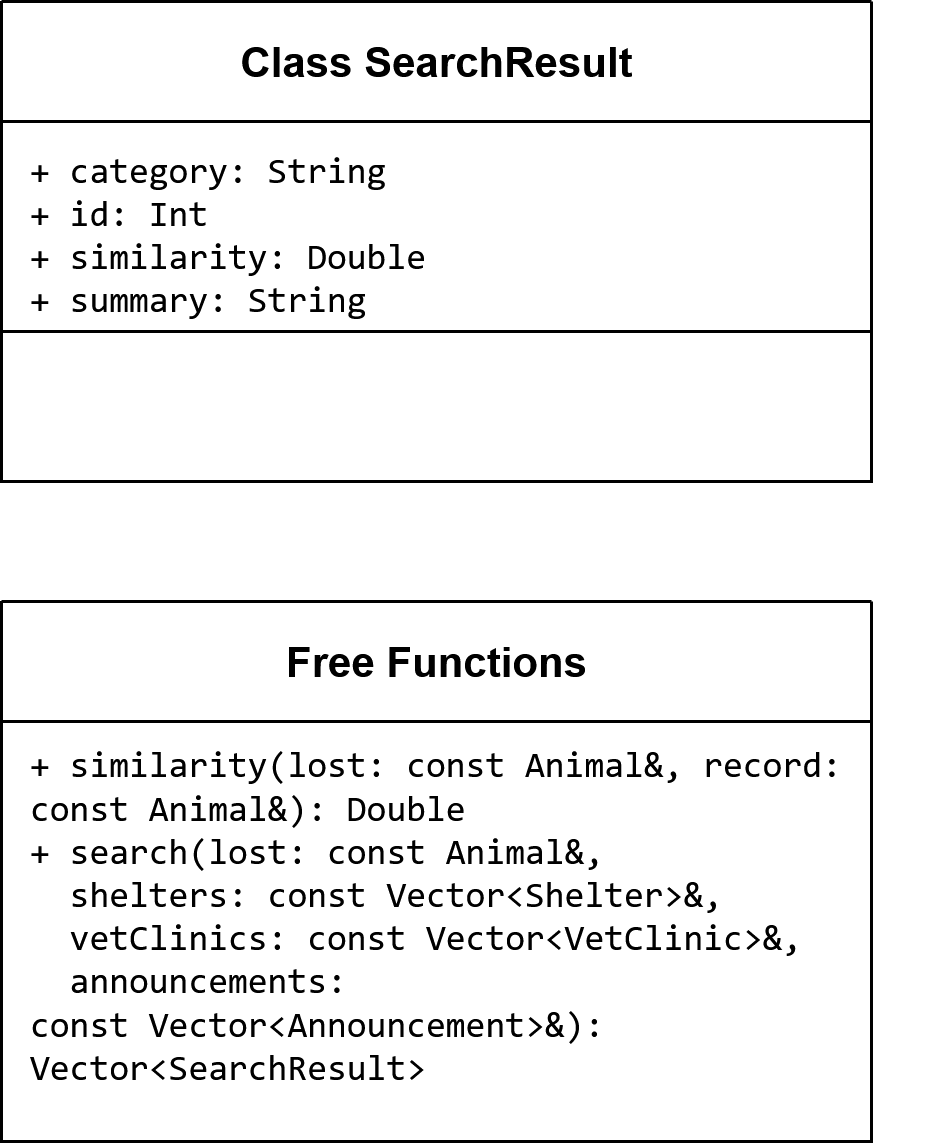


Рисунок 4 – Реализация поиска

В файле main.cpp реализована основная логика работы программы и взаимодействия с пользователем. Main.cpp – это связующее звено между пользователем и логикой программы. Используются базовые принципы структурного программирования и объектно-ориентированного подхода через создание и использование объектов классов.

ТЕКСТ ПРОГРАММЫ

**main.cpp**

Реализует пользовательское меню, обработку ввода, добавление и поиск записей, а также работу с базой данных через класс database.

#include <iostream>

#include <limits>

#include "model.h"

#include "database.h"

#include "search.h"

// Функция для удобного ввода подсказок.

std::string inputString(const std::string& prompt, const std::string& hint = "") {

std::string input;

while (true) {

std::cout << prompt;

if (!hint.empty()) std::cout << " (" << hint << ")";

std::cout << ": ";

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

if (!input.empty()) break; // Проверка ну пустую строку.

std::cout << "Input cannot be empty. Please try again.\n";

}

return input;

}

// Функция для ввода числового значения.

int inputInt(const std::string& prompt, const std::string& hint = "") {

int value;

while (true) {

std::cout << prompt;

if (!hint.empty()) std::cout << " (" << hint << ")";

std::cout << ": ";

std::cin >> value;

if (!std::cin.fail() && value >= 0) { // Проверка на корректность ввода + неотрецательность.

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

break;

}

std::cout << "Invalid number. Please enter a non-negative integer.\n";

std::cin.clear();

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

}

return value;

}

// Функция для вывода меню.

void printMenu() {

std::cout << "\n--- Menu ---\n";

std::cout << "1. Add Shelter record\n";

std::cout << "2. Add VetClinic record\n";

std::cout << "3. Add Announcement record\n";

std::cout << "4. Search lost animal\n";

std::cout << "5. Exit\n";

std::cout << "Choose option (1-5): ";

}

int main() {

Database db;

// Проверка успешного открытия бд.

if (!db.open("database.db")) {

std::cerr << "Failed to open database.\n";

return 1;

}

int choice = 0;

do {

printMenu();

std::cin >> choice;

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

// Реализация меню.

switch (choice) {

case 1: { // Приют.

std::string type = inputString("Type", "e.g. dog, cat");

std::string gender = inputString("Gender", "M or F");

std::string color = inputString("Color");

if (db.addShelter(type, gender, color))

std::cout << "Shelter record added.\n";

else

std::cout << "Failed to add Shelter record.\n";

break;

}

case 2: { // Ветклиника.

std::string type = inputString("Type", "e.g. dog, cat");

std::string gender = inputString("Gender", "M or F");

int age = inputInt("Age", "integer");

std::string color = inputString("Color");

if (db.addVetClinic(type, gender, age, color))

std::cout << "VetClinic record added.\n";

else

std::cout << "Failed to add VetClinic record.\n";

break;

}

case 3: { // Объявление.

std::string type = inputString("Type", "e.g. dog, cat");

std::string gender = inputString("Gender", "M or F");

int age = inputInt("Age", "integer");

std::string color = inputString("Color");

std::string place = inputString("Place");

if (db.addAnnouncement(type, gender, age, color, place))

std::cout << "Announcement record added.\n";

else

std::cout << "Failed to add Announcement record.\n";

break;

}

case 4: { // Поиск и сравнение.

std::string type = inputString("Type", "e.g. dog, cat");

std::string gender = inputString("Gender", "M or F");

int age = inputInt("Age", "integer");

std::string color = inputString("Color");

std::string place = inputString("Place");

// Внешная запись, чтобы не пресекалась с базой данных.

Animal lost(-1, type, gender, age, color, place);

auto results = search(lost, db.shelters, db.vetClinics, db.announcements);

if (results.empty()) {

std::cout << "No records found.\n";

} else { // вывод 10 больше всего подходящих.

std::cout << "\nSearch results (top 10):\n";

int count = (results.size() > 10) ? 10 : (int)results.size();

for (int i = 0; i < count; ++i) {

const auto& r = results[i];

std::cout << i + 1 << ". [" << r.category << "] ID: " << r.id

<< ", Similarity: " << r.similarity << "%, " << r.summary << std::endl;

}

}

break;

}

case 5: // Выход.

std::cout << "Goodbye!\n";

break;

case 6: { // Скрытая функция очистки базы данных.

if (db.clearAllData())

std::cout << "Database cleared successfully.\n";

else

std::cout << "Failed to clear database.\n";

break;

}

default:

std::cout << "Invalid option. Please try again.\n";

}

} while (choice != 5);

db.close();

return 0;

}

**database.cpp**

Реализация функций для работы с базой данных.

#include "database.h"

#include <iostream>

// Конструктор дб.

Database::Database() : db(nullptr), nextId(1) {}

// Деструктор дб.

Database::~Database() {

close();

}

// Открытие дб.

bool Database::open(const std::string& filename) {

if (sqlite3\_open(filename.c\_str(), &db) != SQLITE\_OK) {

std::cerr << "Can't open DB: " << sqlite3\_errmsg(db) << std::endl;

db = nullptr;

return false;

}

// Создание бд. Через Create table if not exists.

execute("CREATE TABLE IF NOT EXISTS Shelter (id INTEGER PRIMARY KEY, type TEXT, gender TEXT, color TEXT);");

execute("CREATE TABLE IF NOT EXISTS VetClinic (id INTEGER PRIMARY KEY, type TEXT, gender TEXT, age INTEGER, color TEXT);");

execute("CREATE TABLE IF NOT EXISTS Announcement (id INTEGER PRIMARY KEY, type TEXT, gender TEXT, color TEXT, place TEXT);");

return loadData();

}

// Закрытие дб.

void Database::close() {

if (db) {

sqlite3\_close(db);

db = nullptr;

}

}

// Проверка вызова запроса через sqlite3\_exec == SQLITE\_OK.

bool Database::execute(const std::string& sql) {

char\* errMsg = nullptr;

int rc = sqlite3\_exec(db, sql.c\_str(), nullptr, nullptr, &errMsg);

if (rc != SQLITE\_OK) {

std::cerr << "SQL error: " << (errMsg ? errMsg : "") << std::endl;

if (errMsg) sqlite3\_free(errMsg);

return false;

}

return true;

}

// Генерация индекса.

int Database::getNextId() {

return nextId++;

}

// Получить данные для shelter из дб.

static int loadShelterCallback(void\* data, int argc, char\*\* argv, char\*\* azColName) {

auto\* shelters = reinterpret\_cast<std::vector<Shelter>\*>(data);

int id = 0;

std::string type, gender, color;

for (int i = 0; i < argc; ++i) {

std::string col = azColName[i];

std::string val = argv[i] ? argv[i] : "";

if (col == "id") id = std::stoi(val);

else if (col == "type") type = val;

else if (col == "gender") gender = val;

else if (col == "color") color = val;

}

shelters->emplace\_back(id, type, gender, color);

return 0;

}

// Получить данные для vetcliniccallback из дб.

static int loadVetClinicCallback(void\* data, int argc, char\*\* argv, char\*\* azColName) {

auto\* vetClinics = reinterpret\_cast<std::vector<VetClinic>\*>(data);

int id = 0, age = 0;

std::string type, gender, color;

for (int i = 0; i < argc; ++i) {

std::string col = azColName[i];

std::string val = argv[i] ? argv[i] : "";

if (col == "id") id = std::stoi(val);

else if (col == "type") type = val;

else if (col == "gender") gender = val;

else if (col == "age") age = std::stoi(val);

else if (col == "color") color = val;

}

vetClinics->emplace\_back(id, type, gender, age, color);

return 0;

}

// Получить данные для Announcement из дб.

static int loadAnnouncementCallback(void\* data, int argc, char\*\* argv, char\*\* azColName) {

auto\* announcements = reinterpret\_cast<std::vector<Announcement>\*>(data);

int id = 0;

std::string type, gender, color, place;

for (int i = 0; i < argc; ++i) {

std::string col = azColName[i];

std::string val = argv[i] ? argv[i] : "";

if (col == "id") id = std::stoi(val);

else if (col == "type") type = val;

else if (col == "gender") gender = val;

else if (col == "color") color = val;

else if (col == "place") place = val;

}

announcements->emplace\_back(id, type, gender, 0, color, place);

return 0;

}

// Проверка актуальности бд и вектора.

bool Database::loadData() {

shelters.clear();

vetClinics.clear();

announcements.clear();

char\* errMsg = nullptr;

int rc = sqlite3\_exec(db, "SELECT \* FROM Shelter;", loadShelterCallback, &shelters, &errMsg);

if (rc != SQLITE\_OK) {

std::cerr << "Failed to load Shelter: " << (errMsg ? errMsg : "") << std::endl;

if (errMsg) sqlite3\_free(errMsg);

return false;

}

rc = sqlite3\_exec(db, "SELECT \* FROM VetClinic;", loadVetClinicCallback, &vetClinics, &errMsg);

if (rc != SQLITE\_OK) {

std::cerr << "Failed to load VetClinic: " << (errMsg ? errMsg : "") << std::endl;

if (errMsg) sqlite3\_free(errMsg);

return false;

}

rc = sqlite3\_exec(db, "SELECT \* FROM Announcement;", loadAnnouncementCallback, &announcements, &errMsg);

if (rc != SQLITE\_OK) {

std::cerr << "Failed to load Announcement: " << (errMsg ? errMsg : "") << std::endl;

if (errMsg) sqlite3\_free(errMsg);

return false;

}

int maxId = 0;

for (const auto& s : shelters) if (s.getId() > maxId) maxId = s.getId();

for (const auto& v : vetClinics) if (v.getId() > maxId) maxId = v.getId();

for (const auto& a : announcements) if (a.getId() > maxId) maxId = a.getId();

nextId = maxId + 1;

return true;

}

// Функции для сохранения данных в память.

bool Database::addShelter(const std::string& type, const std::string& gender, const std::string& color) {

int id = getNextId();

Shelter s(id, type, gender, color);

std::string sql = "INSERT INTO Shelter (id, type, gender, color) VALUES (" +

std::to\_string(id) + ", '" + type + "', '" + gender + "', '" + color + "');";

if (!execute(sql)) return false;

shelters.push\_back(s);

return true;

}

bool Database::addVetClinic(const std::string& type, const std::string& gender, int age, const std::string& color) {

int id = getNextId();

VetClinic v(id, type, gender, age, color);

std::string sql = "INSERT INTO VetClinic (id, type, gender, age, color) VALUES (" +

std::to\_string(id) + ", '" + type + "', '" + gender + "', " + std::to\_string(age) + ", '" + color + "');";

if (!execute(sql)) return false;

vetClinics.push\_back(v);

return true;

}

bool Database::addAnnouncement(const std::string& type, const std::string& gender, int /\*age\*/, const std::string& color, const std::string& place) {

int id = getNextId();

Announcement a(id, type, gender, 0, color, place);

std::string sql = "INSERT INTO Announcement (id, type, gender, color, place) VALUES (" +

std::to\_string(id) + ", '" + type + "', '" + gender + "', '" + color + "', '" + place + "');";

if (!execute(sql)) return false;

announcements.push\_back(a);

return true;

}

// очистка дб.

bool Database::clearAllData() {

bool ok = true;

ok &= execute("DELETE FROM Shelter;");

ok &= execute("DELETE FROM VetClinic;");

ok &= execute("DELETE FROM Announcement;");

ok &= execute("VACUUM;");

if (ok) {

shelters.clear();

vetClinics.clear();

announcements.clear();

}

return ok;

}

**model.cpp**

Реализация функций для классов.

#include "model.h"

// Animal

Animal::Animal(int id\_, const std::string& type\_, const std::string& gender\_, int age\_, const std::string& color\_, const std::string& place\_) : id(id\_), type(type\_), gender(gender\_), age(age\_), color(color\_), place(place\_) {}

int Animal::getId() const { return id; }

std::string Animal::getType() const { return type; }

std::string Animal::getGender() const { return gender; }

int Animal::getAge() const { return age; }

std::string Animal::getColor() const { return color; }

std::string Animal::getPlace() const { return place; }

std::string Animal::getCategory() const { return "Animal"; }

void Animal::print() const {

std::cout << "ID: " << id << ", Type: " << type << ", Gender: " << gender << ", Age: " << age << ", Color: " << color << ", Place: " << place << std::endl;

}

// Shelter

Shelter::Shelter(int id\_, const std::string& type\_, const std::string& gender\_,

const std::string& color\_)

: Animal(id\_, type\_, gender\_, 0, color\_, "") {}

std::string Shelter::getCategory() const { return "Shelter"; }

void Shelter::print() const {

std::cout << "Shelter ID: " << id << ", Type: " << type << ", Gender: " << gender << ", Color: " << color << std::endl;

}

// VetClinic

VetClinic::VetClinic(int id\_, const std::string& type\_, const std::string& gender\_, int age\_, const std::string& color\_) : Animal(id\_, type\_, gender\_, age\_, color\_, "") {}

std::string VetClinic::getCategory() const { return "VetClinic"; }

void VetClinic::print() const {

std::cout << "VetClinic ID: " << id << ", Type: " << type << ", Gender: " << gender << ", Age: " << age << ", Color: " << color << std::endl;

}

// Announcement

Announcement::Announcement(int id\_, const std::string& type\_, const std::string& gender\_,

int /\*age\_\*/, const std::string& color\_, const std::string& place\_)

: Animal(id\_, type\_, gender\_, 0, color\_, place\_) {}

std::string Announcement::getCategory() const { return "Announcement"; }

void Announcement::print() const {

std::cout << "Announcement ID: " << id << ", Type: " << type << ", Gender: " << gender

<< ", Color: " << color << ", Place: " << place << std::endl;

}

**database.h**

#pragma once

#include <vector>

#include <string>

#include <sqlite3.h>

#include "model.h"

class Database {

private:

sqlite3\* db;

int nextId;

bool execute(const std::string& sql);

public:

std::vector<Shelter> shelters;

std::vector<VetClinic> vetClinics;

std::vector<Announcement> announcements;

Database();

~Database();

bool open(const std::string& filename);

void close();

int getNextId();

bool loadData();

bool addShelter(const std::string& type, const std::string& gender, const std::string& color);

bool addVetClinic(const std::string& type, const std::string& gender, int age, const std::string& color);

bool addAnnouncement(const std::string& type, const std::string& gender, int age, const std::string& color, const std::string& place);

bool clearAllData();

};

**model.h**

#pragma once

#include <string>

#include <iostream>

class Animal {

protected:

int id;

std::string type;

std::string gender;

int age;

std::string color;

std::string place;

public:

Animal(int id\_, const std::string& type\_, const std::string& gender\_, int age\_,

const std::string& color\_, const std::string& place\_);

virtual ~Animal() = default;

int getId() const;

std::string getType() const;

std::string getGender() const;

int getAge() const;

std::string getColor() const;

std::string getPlace() const;

virtual std::string getCategory() const;

virtual void print() const;

};

class Shelter : public Animal {

public:

Shelter(int id\_, const std::string& type\_, const std::string& gender\_,

const std::string& color\_);

std::string getCategory() const override;

void print() const override;

};

class VetClinic : public Animal {

public:

VetClinic(int id\_, const std::string& type\_, const std::string& gender\_,

int age\_, const std::string& color\_);

std::string getCategory() const override;

void print() const override;

};

class Announcement : public Animal {

public:

Announcement(int id\_, const std::string& type\_, const std::string& gender\_,

int age\_, const std::string& color\_, const std::string& place\_);

std::string getCategory() const override;

void print() const override;

};

**search.h**

#pragma once

#include <vector>

#include "model.h"

struct SearchResult {

std::string category;

int id;

double similarity;

std::string summary;

};

double similarity(const Animal& lost, const Animal& record);

std::vector<SearchResult> search(const Animal& lost,

const std::vector<Shelter>& shelters,

const std::vector<VetClinic>& vetClinics,

const std::vector<Announcement>& announcements);

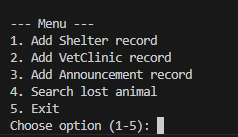
РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

Программа приветствует пользователя меню с выбором вариантов. Первые три пункта добавляют объявление о находке животного.

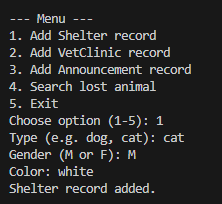
1 – Приют для животных.

2 – Ветеринарная клиника.

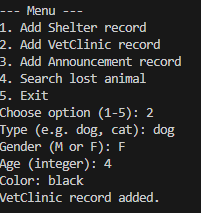
3 – Объявление, оставленное случайными людьми.



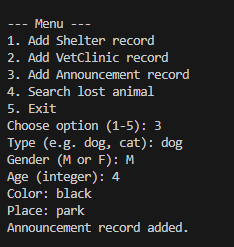
При каждом выборе пользователь может дополнить определенного кол-во данных о найденном животном. Как видно на примере, где выбрали 1, в основном это: вид животного, пол и цвет.



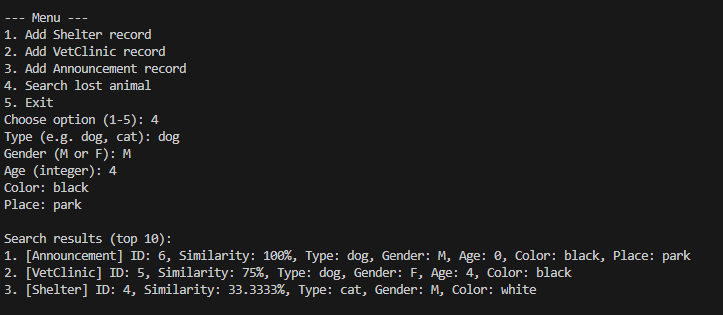
На 2 примере, видно, что к основным характеристикам добавился возраст.



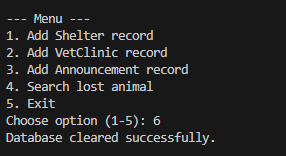
При вызове 3 пункта пользователь сможет внести примерное кол-во лет потерянного питомца.



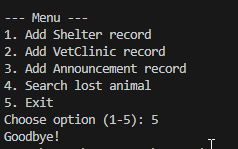
4 пункт служит для поиска пропавшего животного. Выводится сходство в % со всей базой данных.



6 пункт – это скрытая функция для очистки базы данных.



5 пункт – выход из программы.



ЗАКЛЮЧЕНИЕ

В ходе работы была решена задача создания консольного приложения для хранения, поиска и управления информацией о животных в приютах, ветеринарных клиниках и объявлениях. При программировании использовался объектно-ориентированный подход, что позволило эффективно организовать данные и логику через классы и наследование

1. Решенная задача и подход к программированию

Задача: Создать консольное приложение для работы с бд, по теме «поиск домашнего животного».

Подход: Использование объектно-ориентированный подход (ООП), то позволило эффективно организовать код.

2. Основные составляющие проекта

Основными составляющими проекта являются классы Animal и его потомки Shelter, VetClinic, Announcement, а также класс Database, который отвечает за взаимодействие с базой данных SQLite. Программа позволяет добавлять записи, искать похожих животных, очищать базу и работать с данными как в оперативной памяти, так и в базе.

3. Особенности программы

Особенностью программы является использование стандартных средств языка C++ и библиотеки SQLite для работы с базой данных. Для удобства пользователя реализован дружественный текстовый интерфейс с понятными инструкциями и проверкой вводимых данных, что повышает надёжность и удобство работы с программой

4. Какие задачи оказались сложными и почему

Сложными задачами оказались обеспечение синхронизации данных между базой и оперативной памятью, а также корректная обработка ошибок при работе с SQLite. Реализация callback-функций для загрузки данных потребовала аккуратного подхода к преобразованию строковых данных в объекты.

5. Техническая характеристика разработанной программы

Вся программа была написана на языке программирования C++.

Среда разработки: Visual Studio Code.

Библиотеки: string, iostream, limits, sqlite3.h, vector. Я выбрал в качестве бд – lite версию, потому что, как мне показалось, она менее сложная для понимая, а также проста в установке.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Habr, Объектно – ориентированное программирование на C++ в примерах - https://habr.com/ru/sandbox/189562. (03.05.2025)

2. Введение в использование SQLite C / C++ интерфейса. — 2012 г. - https://uploads.7bit.net.ru/2012-04/1334612461-C%20interface%20SQLite%20russian.pdf

3. Бьерн Страуструп. Язык программирования С++. — 2013 г. (ориентировочно) - <http://8361.ru/6sem/books/Straustrup-Yazyk_programmirovaniya_c.pdf>

4. Роберт Лафоре. Объектно-ориентированное программирование в С++. — 2015 г. - https://psv4.userapi.com/s/v1/d/PYIx5b3VKeQtYO8a3zCePeN85hlTNSUj\_bbsQFuN0dXBtXhHLLJKpL2lYzcJ3WHGx-Fca9cOH-kY5nB56-O6NLDsv3IVv3oE6HkrPA9oFmRq-ijuhAArJQ/Lafore\_R\_-\_Obektno-orientirovannoe\_programmir.pdf