Traveller’s app

Симеон Христов Христов 72072

**1. Увод:**

1.1. Създаване на социална мрежа, в която потребителите могат да споделят впечатленията си от различни пътешествия. Приложението също дава възможност на потребителите да получат информация относно дадена дестинация както от приятели, така и от непознати, дали своята оценка за съответното място.

1.2. Основна цел на приложението е да е максимално опростено и интуитивно за всеки потребител, който го използва. Друг важен аспект е самостоятелното написване на някои от библиотеките, свързани с главното функциониране на проекта, което естествено води до по-малко ресурси и по-бърза производителност. Приложението също така съдържа цялата си информация във файлове, които играят ролята на бази от данни, позволяващи по-лесна обработка на данните.

**2. Преглед на дефиниционната област:**

2.1. За да разберем как работи приложението, ще е добре да се запознаем със следните дефиниции:

Потребител – Лице, използващо програмата, характеризирано чрез потребителско име, имейл, парола и списък от приятели, където приятелите са други потребители.

Пътешествие – Характеризира се чрез име на дестинация, период, в който е посетена, оценка за посетеното място, кратък коментар по него и снимки.

2.2. Дефиниране на проблеми и сложност на поставената задача.

Системата трябва да има възможността за:

* Споделяне на информация с други потребители
* Намиране на информация по подадена дестинация
* Въвеждане и складиране на информация за потребители и дестинации
* Автентикация и оторизация за използване и модифициране на информацията за потребители и дестинации

2.3. Подходи и методи за решаване на проблемите:

За решаване на проблемите, описани по-горе, се използва 1 база от данни, която съхранява основните данни за отделните потребители. Също така всеки потребител има своя лична база от данни, в която съхранява своите пътешествия. Изградена е *login* система, която дава право на потребителите да използват съхранената информация. *Search* система, която дава възможност на потребителите да намират информация по подадена дестинация.

2.4  Потребителски ( функционални) и качествени изисквания.

Функционални изисквания:

* При стартиране на програмата, когато потребителят избере опцията за *login,* програмата изисква въвеждане на потребителско име и парола.
* По време на автентикацияпри правилно въведени данни системата позволява достъп до модификация и четене на личните данни на съответния потребител.
* При стартиране на програмата, когато потребителят избере опцията за регистрация,програмата изисква въвеждане на потребителско име, имейл и парола.
* При успешна автентикация, след като потрребителят избере опцията за добавяне на приятел, програмата изисква въвеждане на име на съществуващ потребител, при правилно въведено име потребителят бива добавен като приятел, при грешно въведено име – програмата извежда съобщение за грешка.
* При успешна автентикация, след като потребителят избере опцията за добавяне на пътешествие, програмата изисква въвеждане на дестинация, период, оценка за пътешествието, кратък коментар и снимки.
* По време на въвеждане на данни, ако данните са коректни, дестинацията бива добавена към личната база от данни на потребителя.
* При успешна автентикация, след като потребителят избере опцията за намиране на информация относно дадена дестинация, програмата изисква въвеждане на дестинация и при наличие на информация, тя се извежда.
* При успешна автентикация, след като потребителят избере опцията за преглеждане на профил на приятел, програмата изисква въвеждане на потребителско име, при правилно въведено и съществуващо име програмата изкарва списък с пътуванията на потребителя.

**3. Проектиране.**

3.1. Архитектура

Програмата се разделя на 5 основни класа *User, String, Date, Authenticator, UserFactory*. Основен клас е клас User, който използва всички останали по определен начин.

Класовете *Authenticator* и *UserFactory* използват клас *User,* за да инициализират текущ потребител по подадените данни. *Authenticator* се използва, когато потребител с вече съществуващ профил иска да използва програмата, валидирайки подадените от него данни и подавайки ги към клас *User. UserFactory* се използва, когато нов потребител иска да създаде свой профил, като валидира подадените от него данни и проверява дали те вече не се използват от друг потребител, след което ги подава на клас *User.*

Клас User инициализира текущия потребител с 6 член данни *username, email, password, friendsList, numOfFriends, currentLimitOfFriends,* като първите 4 използват клас String като контейнер, управляващ динамичната памет. Класът позволява на вече инициализирания потребител да използва програмата, предоставяйки му различни опции. Методът *manualAdd()* позволява добавяне на приятел към списъка от приятели, *addJourney(),* дава възможност за добавяне на пътешествие към личната база от данни на текущия потребител, този метод използва клас *Date* при въвеждането на период, за да потвърди, че датите съществуват и периодът е валиден (т.е. крайната дата не е преди началната), а частният метод *validFile()* проверява дали въведените снимки са с допустими разширения и коректно име. С *destinationGradeByAll()* текущият потребител може да потърси информация относно въведена от него дестинация. Частният метод *splitString() и addFriend()* се използва при инициализация на потребителя, за да се създаде списъкът му от приятели, a *isExistingUser(), isFriend* се използват от *manualAdd(),* за да се провери дали въведеният потребител съществува и дали той не е вече приятел. Преди приключване на програмата се използва методът *updateDB(),* който актуализира базата от данни с потребителите.

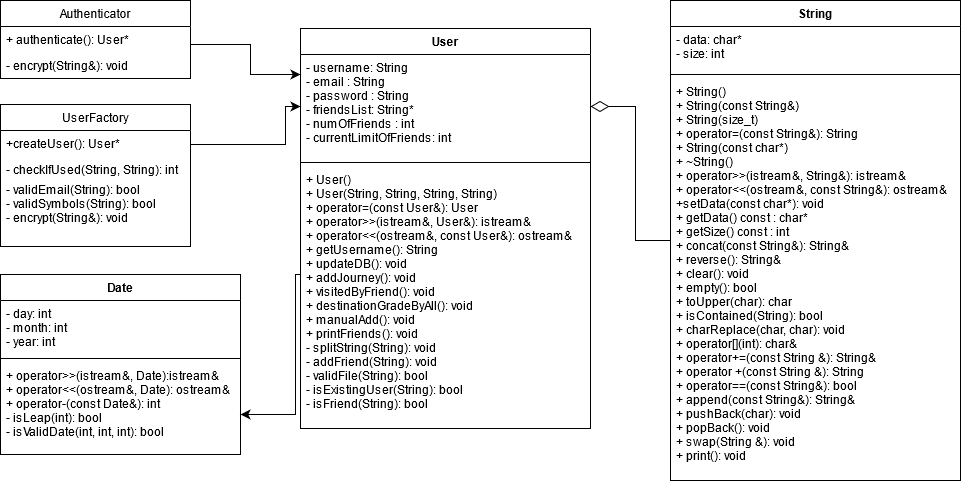


Diagram 1

**Глава 4. Реализация, тестване** (1–2 стр.)

4.1. Управление на паметта в класа *String*

Създаването на обект от клас String се случва чрез заделяне на динамична памет, като размерът се определя от дължината на входния низ.

void String::setData(const char \*data) {

    this->data = new char[strlen(data) + 1];

    for (int j = 0; j < strlen(data); j++) {

        this->data[j] = data[j];

    }

    this->data[strlen(data)] = '\0';

}

String::~String() {

    delete this->data;

}

Метод pushBack(char letter) заделя памет за временен низ, като му се присвоява стойността на текущия *String* и добавя нов елемент (буква).

void String::pushBack(char letter) {

    char \*temp = new char[this->getSize() + 2];

    for (size\_t i = 0; i < this->getSize(); i++)

    {

        temp[i] = this->data[i];

    }

    temp[this->getSize()] = letter;

    temp[this->getSize() + 1] = '\0';

    delete[] this->data;

    this->data = temp;

    this->size++;

}

Алгоритми

isContained (String) методът е в основата на *Search* системата, изградена в метода за намиране на информация относно дадена дестинация. Алгоритъмът работи чрез използване на 2 вложени цикъла, като външният обхожда даден низ, а вътрешният цикъл проверява дали буквите от даден индекс съвпадат.

bool String::isContained(String other) {

    int M = this->getSize();

    int N = other.getSize();

    if (M == N) {

        for (size\_t i = 0; i < M; i++) {

            if (toUpper(this->data[i]) != toUpper(other.data[i])) {

                return 0;

            }

        }

        return true;

    }

    for (int i = 0; i < N - M; i++) {

        int j;

        for (j = 0; j < M; j++) {

            if (toUpper(other[i + j]) != toUpper(this->data[j])) {

                break;

            }

        }

        if (j == M) {

            return 1;

        }

    }

    return 0;

}

аuthenticate() методът използва въведените от потребителя потребителско име и парола и ги сравнява с тези, които вече съществуват в базата от данни. При съвпадение потребителят бива инициализиран, в противен случай се изисква повторно въвеждане.

std::ifstream fin("users.db");

        if (fin.is\_open()){

            while (fin >> data1 >> data2 >> data3 >> friendsList) {

                if (data1 == username && data3 == password) {

                    // initialize class with this data...

                    User \*currentUser = new User(data1, data2, data3, friendsList);

                    returnUser = currentUser;

                    found = 1;

                    fin.close();

                    return currentUser;

                }

            }

            if (fin.eof()){

                std::cout << "No such username and password combination

found, please try again." << std::endl;

            }

        }

        fin.close();

encrypt (String&) алгоритъмът по подобие на ***Caesar Cipher*** съпоставя на буква 3 други, като първата е нейното отместване с 1 символ назад, втората е отместването ѝ с 6 символа напред, а третата отместването с 2 напред. При излизане от допустимите граници за символ алгоритъмът по математически начин съпоставя нов символ, определен от разликата на допустимите граници и съответният символ.

void Authenticator::encrypt(String &input) {

    String encrypted = "";

    for (int i = 0; i < input.getSize(); i++) {

        char a,b,c;

        if (input[i] - 1 == 32) {

            a = (char)(input[i] + 32);

        }

        else {

            a = (char)(input[i] - 1);

        }

        if (input[i] + 4 >= 127) {

            b = (char)((input[i] + 6) % 95 + 33);

        }

        еlse {

            b = (char)(input[i] + 6);

        }

        if (input[i] + 2 >= 127) {

            c = (char)((input[i] + 2) % 95 + 33);

        }

        else {

            c = (char)(input[i] + 2);

        }

        encrypted.pushBack(a);

        encrypted.pushBack(b);

        encrypted.pushBack(c);

    }

    input = encrypted;

}

**5. Заключение.**

5.1. Насоки за бъдещо развитие и усъвършенстване:

Имплементиране на *tag* система, която да позволи на различните потребители да могат да отбелязват други потребители, с които са били на дадено пътуване.

Система за коментиране, в която всеки потребител може да изкаже личното си мнение относно пътуването на друг потребител.

Подобряване на защитата на потребителските данни с цел сигурност.

**Ресурси:**

[**https://github.com/simeonhhristov/OOP-Project**](https://github.com/simeonhhristov/OOP-Project)