Lucrare practică de atestare a cunoștințelor de informatică

**Aplicație**

**pentru**

**prognoză meteo**

**Cuprins**

1. Prezentarea lucrării……………………………………………………….. 2
2. Tehnologii utilizate………………………………………………………...2
   1. C++......................................................................................................2
   2. SFML...................................................................................................2
   3. TGUI………........................................................................................3
   4. WeatherAPI..........................................................................................3
3. [Schema conceptuală a lucrării……………………………………………](#_j1vrtz56lq01)...4
4. [Testarea în timp real………………………………………………………..](#_4s3ss7yxwhn0) .5
5. [Anexe………………………………………………………………………](#_p4vwq1d0x7gb).6
6. [Bibliografie……………..…………………………………………………..](#_dhmshktfgdrp)12
7. **Prezentarea lucrarii**

În contextul dezvoltării aplicațiilor interactive, proiectul de față explorează realizarea unei aplicații de prognoză meteo folosind tehnologii moderne de programare și interfață grafică. Aplicația oferă utilizatorilor posibilitatea de a căuta informații meteo în timp real pentru orice oraș din lume, prezentând date esențiale precum temperatura, condițiile meteo, viteza vântului și altele, printr-o interfață ușor de utilizat.

Aplicația este construită pe baza bibliotecii **SFML**, care asigură gestionarea ferestrei grafice și a evenimentelor, iar pentru interfața grafică, se utilizează **TGUI**. Aceasta include componente esențiale, precum câmpuri de text, butoane și etichete, pentru a permite interacțiunea cu utilizatorul într-un mod simplu și intuitiv.

Un aspect central al aplicației îl reprezintă integrarea cu **API-ul WeatherAPI**, prin intermediul bibliotecii **cpr** care permite realizarea unui apel HTTP GET pentru a obține datele meteo actuale. Răspunsul primit de la API este prelucrat și afișat utilizatorului într-un format clar și accesibil

1. **Tehonologii utilizate**

**II.1 C++**

C++ este un limbaj de programare de nivel mediu, creat de Bjarne Stroustrup în 1979 ca o extensie a limbajului C. Este un limbaj versatil, utilizat pentru dezvoltarea unor aplicații de înaltă performanță, inclusiv sisteme de operare, aplicații de jocuri, software embedded și aplicații ce necesită control direct asupra hardware-ului. C++ combină eficiența limbajului C cu caracteristicile programării orientate pe obiecte, oferind dezvoltatorilor control maxim asupra resurselor de sistem, dar și o flexibilitate sporită în alegerea paradigmei de programare (procedurală sau orientată pe obiecte). C++ a fost standardizat de ISO în mai multe versiuni (C++98, C++03, C++11, C++14, C++17, C++20, C++23), fiecare dintre acestea aducând îmbunătățiri și noi funcționalități care sprijină dezvoltarea rapidă a aplicațiilor eficiente și scalabile.

# II.2 SFML (Simple and Fast Multimedia Library)

SFML este o bibliotecă multimedia de tip open-source, destinată să ajute dezvoltatorii să creeze aplicații grafice, jocuri și programe interactive. Dezvoltată pentru a fi ușor de utilizat, SFML oferă o serie de funcționalități pentru manipularea ferestrelor, input-urilor, graficii 2D, sunetului și rețelelor. SFML se integrează perfect cu limbajul C++ și este adesea utilizată în dezvoltarea de jocuri 2D și aplicații vizuale, oferind un cadru simplu și rapid pentru rendering grafic, controlul evenimentelor de tastatură și mouse, precum și gestionarea sunetului și muzicii. De asemenea, SFML se poate utiliza pe mai multe platforme (Windows, Linux, macOS), oferind o soluție portabilă pentru dezvoltarea aplicațiilor grafice.

# II.3 TGUI

TGUI este o bibliotecă open-source pentru crearea interfețelor grafice de utilizator (GUI) pentru aplicații C++. Aceasta oferă o gamă largă de widgeturi, inclusiv butoane, etichete, panouri, câmpuri de text și multe altele, toate fiind ușor de integrat într-o aplicație. TGUI este construită pe baza SFML, oferind dezvoltatorilor o soluție simplă și eficientă pentru crearea de interfețe grafice, fără a fi necesare cunoștințe avansate despre grafica de bază. Bibliotecile TGUI sunt destinate pentru a sprijini dezvoltarea rapidă a aplicațiilor cu interfețe grafice, aducând la un loc funcționalitățile de control al elementelor vizuale și gestionarea evenimentelor. TGUI este extrem de flexibilă și poate fi folosită pentru o varietate de aplicații, de la jocuri la aplicații de birou.

# II.4 WeatherAPI

**WeatherAPI** este un serviciu de API dedicat furnizării de date meteo actualizate în timp real pentru orice locație din lume. Acesta permite accesul la informații precum temperatura, umiditatea, condițiile meteo, viteza vântului, presiunea atmosferică, radiațiile UV și multe altele. WeatherAPI oferă date detaliate atât pentru condițiile curente, cât și pentru prognoza meteo pe termen scurt și lung.

API-ul este ușor de integrat în diverse aplicații, fiind folosit frecvent în dezvoltarea de aplicații mobile, desktop sau aplicații web care necesită informații meteo. Acesta utilizează un protocol HTTP pentru a răspunde la cererile de tip GET, returnând datele într-un format JSON. Pentru a utiliza API-ul, este necesară o cheie de autentificare, care asigură accesul la serviciile sale. WeatherAPI oferă o gamă variată de planuri, inclusiv unul gratuit cu limitări privind numărul de cereri zilnice.

Un avantaj important al acestui API este acuratețea și actualizarea frecventă a datelor, ceea ce îl face potrivit pentru aplicații care necesită informații precise și recente, precum aplicațiile de prognoză meteo sau sistemele de monitorizare a condițiilor meteorologice.

# Schema conceptuală a lucrării

.

└── Weather app

├── include

│ └── main.h

├── src

│ └── main.cpp

├── out

│ └── build

│ └── x64-debug

│ └── Weather\_app.exe

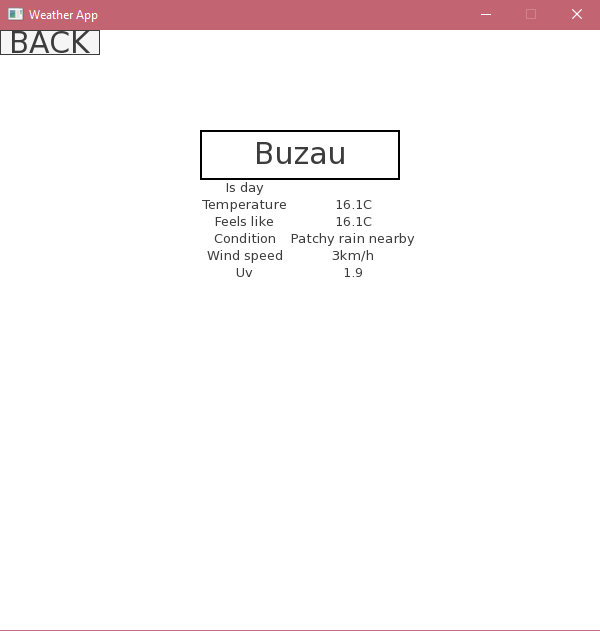
└── CMakeLists.t

# Testarea în timp real

Aplicația **WeatherApp** este o aplicație de desktop destinată să furnizeze informații meteo în timp real pentru utilizatori. La deschiderea aplicației, utilizatorul poate introduce numele unui oraș într-un câmp de căutare. După ce utilizatorul apasă butonul de căutare, aplicația trimite o cerere către **WeatherAPI**, un API care furnizează date meteo actualizate, și preia informațiile meteo pentru orașul respectiv.

După ce datele sunt preluate cu succes, aplicația trece la un ecran secundar care prezintă următoarele informații:

* **Temperatura actuală** a orașului, afișată într-un format clar și concis.
* **Temperatura resimțită** (sau "feels like"), care indică cum se simte temperatura reală pentru utilizator.
* **Condițiile meteo**, de exemplu, "cer senin", "ploaie ușoară", etc.
* **Viteza vântului**, exprimată în km/h sau altă unitate relevantă.
* **Nivelul radiațiilor UV**, un indicator al intensității radiațiilor solare.
* **Starea zilei** (zi/noapte), care indică dacă orașul se află într-o perioadă de zi sau noapte.

Un buton de navigare permite utilizatorului să revină la ecranul principal pentru a efectua o altă căutare, oferind o experiență de utilizare fluidă și ușor de navigat

# Anexe

**Codul proiectului**

* + main.hpp

#include <SFML/Window.hpp>

#include <TGUI/TGUI.hpp>

#include <TGUI/Backend/SFML-Graphics.hpp>

#include <iostream>

#include <cpr/cpr.h>

#include <nlohmann/json.hpp>

using namespace std;

int scene = 0;

std::string API\_KEY = "22e0e7bff53d4cd4bd8140117250902";

std::string city;

nlohmann::json city\_stats;

nlohmann::json search\_city(std::string API\_KEY, std::string city)

{

cpr::Parameters parameters{

{"key", API\_KEY},

{"q", city}

};

cpr::Response response = cpr::Get(cpr::Url{ "http://api.weatherapi.com/v1/current.json" },

parameters);

return nlohmann::json::parse(response.text);

}

int main()

{

sf::RenderWindow window(sf::VideoMode(600, 600), "Weather App", sf::Style::Close | sf::Style::Titlebar);

tgui::Gui search\_window(window);

tgui::Gui stats\_window(window);

//Stats window

auto panel = tgui::Panel::create({ 200, 50 });

panel->getRenderer()->setBorders({ 2, 2, 2, 2 }); // Border thickness

panel->getRenderer()->setBorderColor(tgui::Color::Black);

panel->setPosition(200, 100);

stats\_window.add(panel);

auto city\_name = tgui::Label::create();

city\_name->setSize(200, 50);

city\_name->setTextSize(30);

city\_name->setPosition(300-100, 100);

city\_name->getRenderer()->setBorderColor(tgui::Color::Black);

city\_name->setVerticalAlignment(tgui::VerticalAlignment::Center);

city\_name->setHorizontalAlignment(tgui::HorizontalAlignment::Center);

stats\_window.add(city\_name);

auto grid = tgui::Grid::create();

stats\_window.add(grid);

// Set the grid's position

grid->setPosition(200, 150);

auto back\_button = tgui::Button::create("BACK");

back\_button->setSize(100, 25);

back\_button->setTextSize(30);

back\_button->setPosition(0, 0);

back\_button->onClick([&]() {

scene = 0;

grid->removeAllWidgets();

});

stats\_window.add(back\_button);

//Search Window

auto textBox = tgui::TextArea::create();

textBox->setSize(200, 50);

textBox->setTextSize(30);

textBox->setPosition(300 - 100, 300 - 50 - 150);

textBox->getRenderer()->setBackgroundColor(tgui::Color::Blue);

textBox->getRenderer()->setTextColor(tgui::Color::White);

search\_window.add(textBox);

auto button = tgui::Button::create("SEARCH");

button->setSize(100, 100);

button->setPosition(300 - 50, 300 - 50);

button->onClick([&](){

city\_stats = search\_city(API\_KEY, textBox->getText().toStdString());

if (city\_stats.contains("location") && city\_stats.contains("current")){

scene = 1;

city\_name->setText(city\_stats["location"]["name"].get<std::string>());

float temp\_real = city\_stats["current"]["temp\_c"].get<float>();

std::ostringstream temp\_stream;

temp\_stream << std::fixed << std::setprecision(1) << temp\_real;

float temp\_feel = city\_stats["current"]["feelslike\_c"].get<float>();

std::ostringstream temp\_feel\_stream;

temp\_feel\_stream << std::fixed << std::setprecision(1) << temp\_feel;

float wind\_speed = city\_stats["current"]["wind\_kph"].get<float>();

std::ostringstream wind\_speed\_stream;

wind\_speed\_stream << (int)wind\_speed;

float uv = city\_stats["current"]["uv"].get<float>();

std::ostringstream uv\_stream;

uv\_stream << uv;

int isDay = city\_stats["current"]["is\_day"].get<int>();

if(isDay == 1)

grid->addWidget(tgui::Label::create("Is day"), 0, 0);

else grid->addWidget(tgui::Label::create("Is night"), 0, 0);

grid->addWidget(tgui::Label::create("Temperature"), 1, 0);

grid->addWidget(tgui::Label::create(temp\_stream.str() + "C"), 1, 1);

grid->addWidget(tgui::Label::create("Feels like"), 2, 0);

grid->addWidget(tgui::Label::create(temp\_feel\_stream.str() + "C"), 2, 1);

grid->addWidget(tgui::Label::create("Condition"), 3, 0);

grid->addWidget(tgui::Label::create(city\_stats["current"]["condition"]["text"].get<std::string>()), 3, 1);

grid->addWidget(tgui::Label::create("Wind speed"), 4, 0);

grid->addWidget(tgui::Label::create(wind\_speed\_stream.str() + "km/h"), 4, 1);

grid->addWidget(tgui::Label::create("Uv"), 5, 0);

grid->addWidget(tgui::Label::create(uv\_stream.str()), 5, 1);

}

textBox->setText("");

});

search\_window.add(button);

while (window.isOpen())

{

std::cout << "Scene: " << scene << std::endl;

sf::Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

window.close();

if (scene == 0)

search\_window.handleEvent(event);

else if (scene == 1)

stats\_window.handleEvent(event);

}

window.clear(sf::Color::White);

switch (scene)

{

case 0:

search\_window.draw();

break;

case 1:

stats\_window.draw();

break;

}

window.display();

}

return 0;

}

# Bibliografie

* <https://www.weatherapi.com/docs/>
* <https://en.wikipedia.org/wiki/C%2B%2B>
* <https://en.cppreference.com/w/>
* <https://www.sfml-dev.org/>
* <https://tgui.eu/>
* <https://github.com/libcpr/cpr>