

Documentație

Planificator în funcție de condițiile meteo

Realizat de:
Boacă Mădălina-Elena
Bulai Iustina-Bianca
Ivanov Alexandru
Pojoga Gabriel
Grupa 1307A

Univeristatea Tehnică “Gheorghe Asachi”, Iași
Facultatea de Automatică și Calculatoare
Mai, 2023

Cuprins

1. SRS	3
I. Introducere	4
A. Scop	
B. Publicul vizat și sugestii de lectură	
C. Scopul produsului	
D. Referințe	
II. Descriere generală	5
A. Perspectiva produsului	
B. Funcțiile produsului	
C. Clasele și caracteristicile utilizatorilor	
D. Mediul de operare	
E. Constrângeri de proiectare și implementare	
F. Documentația utilizatorului	
G. Ipoteze și dependențe	
III. Cerințe pentru interfața externă	6
A. Interfețe cu utilizatorul	
B. Interfețe hardware	
C. Interfețe software	
D. Interfețe de comunicații	
IV. Caracteristicile sistemului	6
V. Alte cerințe nefuncționale	7
A. Cerințe de performanță	
B. Cerințe de siguranță și securitate	
C. Atributele calității software	
D. Reguli de afaceri	
2. Diagrame UML	8
3. Mod de utilizare al aplicației	17
4. Exemple de execuție ale aplicației	22
Anexa 1: Exemple ale codului sursă asociat	27
A. Metode reprezentative pentru realizarea interfeței	27
B. Modulul de persistență	30
C. Modulul pentru preluarea datelor de la API	34
Anexa 2: Etapa de testare	37

1. Software Requirements Specification

Pentru

**Planificator în funcție
de condițiile meteo**

I. Introducere

A. Scop

Scopul proiectului nostru este acela de a oferi utilizatorului posibilitatea de a-și planifica cât mai util timpul, având la dispoziție și condițiile meteo. Acesta constă într-o aplicație desktop în care pot fi realizate mai multe operații, precum adăugarea unei notițe, editarea și ștergerea ei, sau a tuturor notițelor și vizualizarea vremii în ziua în care se dorește a se programa o activitate. Este necesară conexiunea la internet pentru a putea prelua datele meteo din API, dar în cazul în care nu pot fi încărcate, există totuși posibilitatea de a salva notița fără acestea.

B. Publicul vizat și sugestii de lectură

Acest document nu este destinat utilizatorului final deoarece reprezintă o specificație detaliată a modului în care a fost implementat software-ul. Din moment ce un utilizator dorește să știe cum să utilizeze aplicația, nu cum a fost ea realizată, acest document este destinat mai degrabă unui tester și mai ales dezvoltatorului aplicației.

Documentul începe cu o prezentare generală a funcțiilor și specificațiilor aplicației în secțiunea 2, iar mai apoi, în secțiunea 3, sunt descrise cerințele pentru interfața externă. Secțiunea 4 prezintă informații cu privire la aplicație, iar secțiunea 5 pune în evidență caracteristicile nefuncționale. În secțiunea 6, sunt prezentate date foarte utile, precum diagramele UML și cea a cazului de utilizare.

C. Scopul produsului

Planificatorul în funcție de condițiile meteo este o aplicație dezvoltată pe platforma .NET. Este creată în așa fel încât să poată fi utilizată de către oricine, datorită butoanelor sugestive. Se adresează persoanelor care doresc să își organizeze mai bine timpul în funcție de condițiile meteo preconizate.

D. Referințe

Pentru a putea accesa cu ușurință API-ul necesar condițiilor meteorologice, au fost utile informațiile prezente pe următoarele site-uri:

1. https://api-ninjas.com/api/geocoding?fbclid=IwAR0E69bMuAL3MTs4_fwJwogG34i4sE4GegpVY1FDiThL1N83rvyvjVwldFU
2. https://open-meteo.com/?fbclid=IwAR1EgtJ-5D_66V8GQUM9NdYO1_YaK63RCDooCuacp_uK-omzpjTP-MiTUOk

II. Descriere generală

A. Perspectiva produsului

Proiectul ales, intitulat Planificator în funcție de condițiile meteo, este implementat ca o aplicație Windows Forms, utilizând C# ca limbaj de programare și Framework-ul .NET 4.7.2. Datele pentru această aplicație sunt reprezentate de ID-ul unic, titlul notiței, conținutul acesteia, data aleasă, locația și vremea, toate fiind stocate într-o bază de date SQLite, aceasta fiind cea mai optimă variantă pentru aceste specificații.

Aplicația principală utilizează două DLL-uri (Dynamic Link Library), unul utilizat pentru a putea lua datele meteo din API, numit Weather, iar al doilea este folosit pentru a menține notițele pe termen lung, intitulat Persistence.

Sunt utilizate 3 module de testare a unităților, unul pentru interfață, unul pentru Persistence și unul pentru Weather API.

B. Funcțiile produsului

Aplicația prezintă următoarele funcționalități pentru utilizatori:

- Posibilitatea de a adăuga o notiță într-o anumită zi, fie prin accesarea unei date din calendarul vizibil în partea dreaptă a interfeței, fie prin accesarea butoanelor de Home-New;
- Posibilitatea de a vizualiza notițele adăugate într-o singură zi la apăsarea unei date din calendar, fie a tuturor notițelor, prin accesarea butoanelor Home-ViewAll;
- Posibilitatea de editare sau ștergere a unei notițe, opțiunea alegându-se în momentul în care se dă click-dreapta pe aceasta;
- Posibilitatea de ștergere a tuturor notițelor prin accesarea butoanelor de Home-DeleteAll.

C. Clasele și caracteristicile utilizatorilor

Aplicația este una ușor de utilizat, fiind accesibilă pentru orice tip de utilizator, datorită butoanelor sugestive și a indicațiilor prezente în interfață.

D. Mediul de operare

Din moment ce Planificatorul în funcție de condițiile meteo este dezvoltat cu ajutorul Framework-ului .NET, acesta este compatibil doar cu versiunile specifice sistemului de operare Windows, începând cu Windows 7. Prin urmare, este necesară instalarea versiunii .NET Framework 4.7.2 sau a uneia mai noi pe sistemul de operare.

E. Constrângeri de proiectare și implementare

Aplicația noastră a fost proiectată și implementată pe parcursul unui singur semestru, în cadrul facultății, la materia intitulată Ingineria Programării. Din acest motiv, unul dintre factorii care a limitat ciclul de dezvoltare al proiectului a fost limita de timp, precum și constrângerea tehnologică de a implementa totul cu ajutorul Framework-ului .NET. O altă problemă a fost lipsa noastră de experiență cu limbajul C#, cu proiectarea software-ului și cu crearea documentației.

F. Documentația utilizatorului

Planificatorul conține și un buton special pentru funcția de Help, în care se pot găsi informații cu privire la modul de utilizare al aplicației și al funcționalităților acesteia.

G. Ipoteze și dependențe

- **Ipoteze:** testarea aplicației s-a făcut pe 4 sisteme de operare diferite, mai exact Windows 10, respectiv 11, comerciale sau academice. Estimăm totuși că dat fiind specificul limbajului C#, nu ar exista probleme cu executarea aplicației pe alte versiuni de Windows dacă acestea depășesc versiunea Windows 7.
- **Dependențe:** funcția de determinare a condițiilor meteo depinde de disponibilitatea API-ului. La nivel local există o dependență a aplicației de sistemul de baze de date SQLite.

III. Cerințe pentru interfața externă

A. Interfețe cu utilizatorul

Când se lansează aplicația, va apărea o interfață ce conține meniul în partea stângă și un calendar în partea dreaptă. Există posibilitatea de minimizare și maximizare a meniului și inițial sunt vizibile două butoane, de Home și Help. Mai apoi, butonul de Home se poate extinde, urmând să apară și butoanele New, ViewAll, respectiv DeleteAll, asemănător și cel de Help, din care se pot accesa butoanele About și Info. Interfața mai conține încă două butoane în interiorul calendarului, pentru a face posibilă afișarea mai multor luni și ani.

B. Interfețe hardware

Interfața hardware trebuie să includă un mouse, o tastatură și un monitor pentru afișare. Nu este necesară o altă componentă hardware, deoarece aplicația funcționează pe orice computer cu Framework-ul .NET, versiunea 4.7.2 sau mai nouă. Pentru a putea introduce o notiță, este necesară utilizarea tastaturii, iar mouse-ul este utilizat atunci când se apasă pe un buton, sau când asupra unei notițe se folosește click-dreapta pentru a apărea posibilitatea de editare/ștergere a notiței respective.

C. Interfețe software

Pentru a lansa în execuție aplicația, utilizatorul trebuie să aibă sistemul de operare Windows 7 sau o variantă mai actuală și să aibă instalat Framework-ul .NET, versiunea 4.7.2 sau una mai nouă.

D. Interfețe de comunicații

În cadrul aplicației, în momentul în care se dorește adăugarea unei notițe, se realizează o cerere HTTP. Totuși, există și cazul în care cererea nu poate fi efectuată cu succes, dar aplicația va funcționa în continuare.

IV. Caracteristicile sistemului

Caracteristicile sistemului și diagrama de clase sunt prezentate în detaliu într-o secțiune ulterioară.

V. Alte caracteristici nefuncționale

A. Cerințe de performanță

Aplicația poate fi lansată în execuție de pe orice computer cu sistem de operare Windows, începând cu versiunea Windows 7 până la cea mai recentă, și care are o versiune a Framework-ului .NET de 4.7.2 sau una mai nouă.

B. Cerințe de siguranță și securitate

Dată fiind executarea locală a aplicației, nu este necesară autentificarea sau protecția datelor cu caracter personal. Cererile HTTP ar putea duce la probleme de securitate, dar acestea nu țin de implementarea aplicației, ci de comunicarea în rețea. Nu este necesar un sistem de recuperare datorită existenței nivelului de persistență și a commit-urilor după fiecare operație în baza de date (dacă se presupune că nivelul de persistență nu poate fi afectat), iar backup-ul se poate face prin editarea/ștergerea unei notițe.

Aplicația a fost proiectată astfel încât să reziste la eșecuri legate de API, prin urmare, erorile sale nu vor determina oprirea aplicației.

C. Atributele calității software

Aplicația încearcă să fie fiabilă, dorind să diminueze erorile cauzate de API și să păstreze funcțiile de bază, dacă apar. Fiind o aplicație care generează un executabil nativ pentru platforma Windows, eficiența este crescută. Aplicația este portabilă pe sisteme Windows cu versiuni mai mari decât 7 și nu apar dependențe hardware. Produsul are un grad de utilizabilitate foarte mare deoarece a fost conceput prin analiza unui număr mare de aplicații, oferind astfel un mod de utilizare similar cu celelalte produse de pe piață.

Aplicația este împărțită în 3 module principale: API, persistență și interfață grafică pentru a fi ușor extensibilă și reutilizabilă. Folosind modelul de proiectare fațadă, am putut asigura un nivel de legătură și siguranță crescută între aceste module.

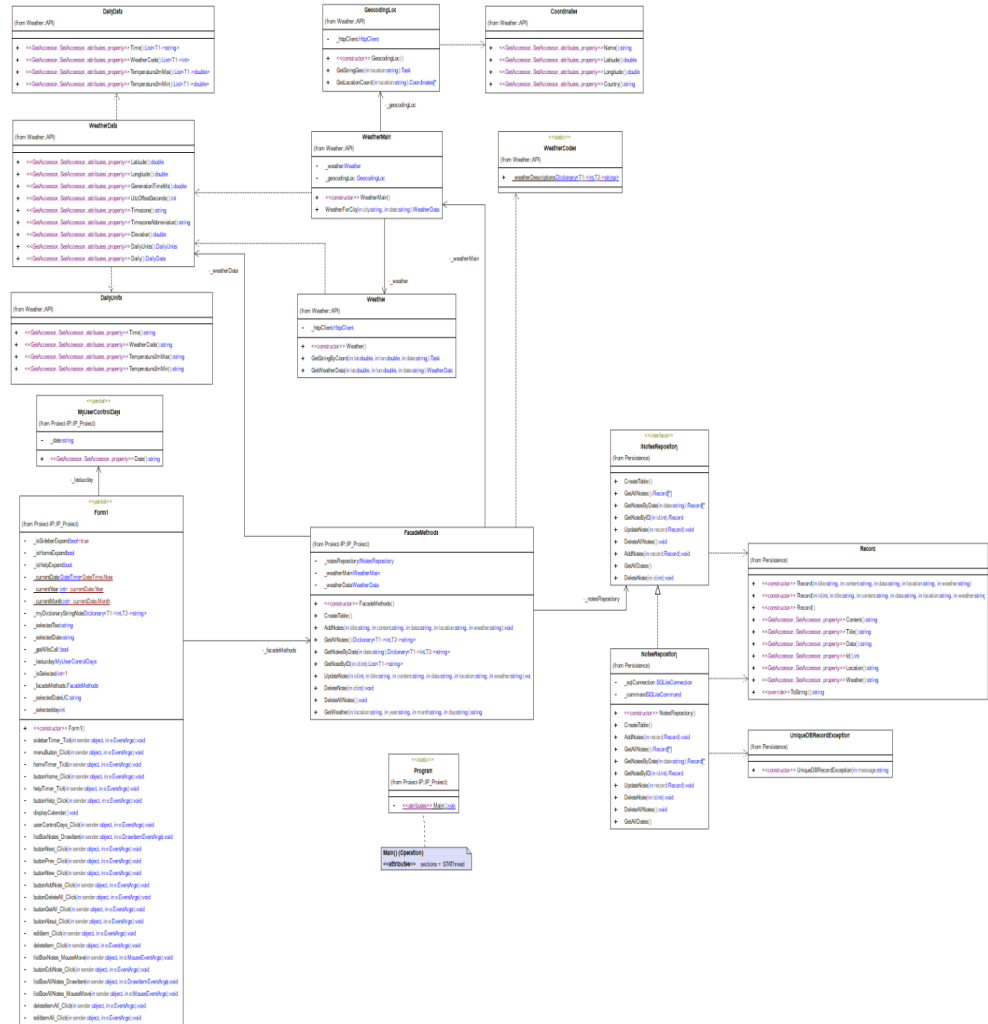
D. Reguli de afaceri

Echipa care a realizat acest proiect trebuie să respecte toate codurile de conduită stabilite de Universitate.

2. Diagrame UML

1. Diagrama de clase

Această diagramă cuprinde toate modulele (interfață, API, Persistence). În ea se poate observa implementarea șablonului Fațadă care leagă interfața de celelalte două module.



Pentru a le putea vizualiza mai ușor, vom atașa diagramele pe secțiuni:

→ Interfață

Diagrama pentru interfață cuprinde atât clasa Form1 (cu funcții de callback pentru elementele din interfața grafică), cât și o clasă ce reprezintă un contor extins folosit la afișarea calendarului și clasa pentru FacadeMethods (utilizată la legarea interfeței de celelalte module - se observă câmpurile private `_weatherMain`, `_weatherData` și `_notesRepository`).

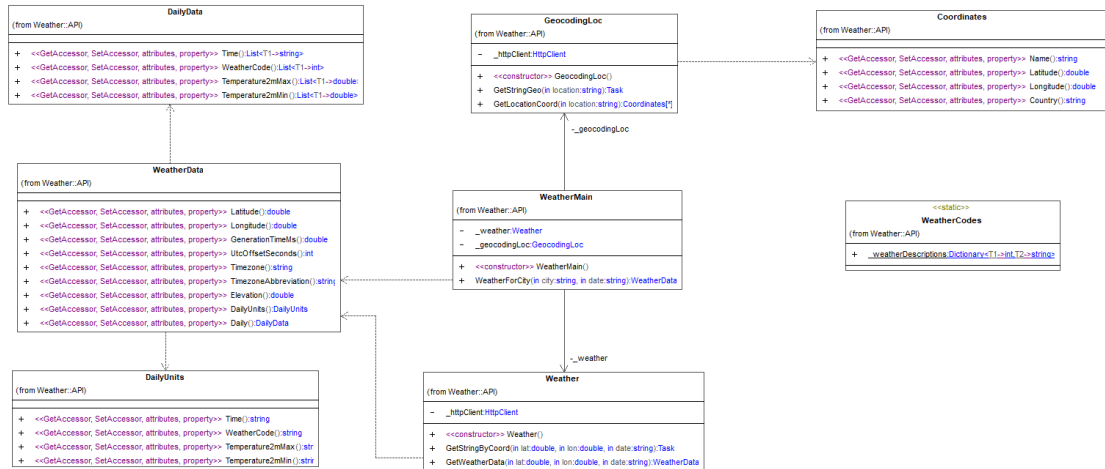


Generated by UModel

www.altova.com

→ API

Diagrama pentru API (modulul care se ocupă de extragerea prognozei meteo), cuprinde atât clase ce extrag efectiv date (`Weather`, `WeatherData`, `GeocodingLoc`), cât și clase folosite pentru deserializarea răspunsurilor HTTP (`DailyUnits`, `WeatherData`, `DailyData`, `Coordinates`) sau pentru afișarea descrierii meteo ca șir (`WeatherCodes`).

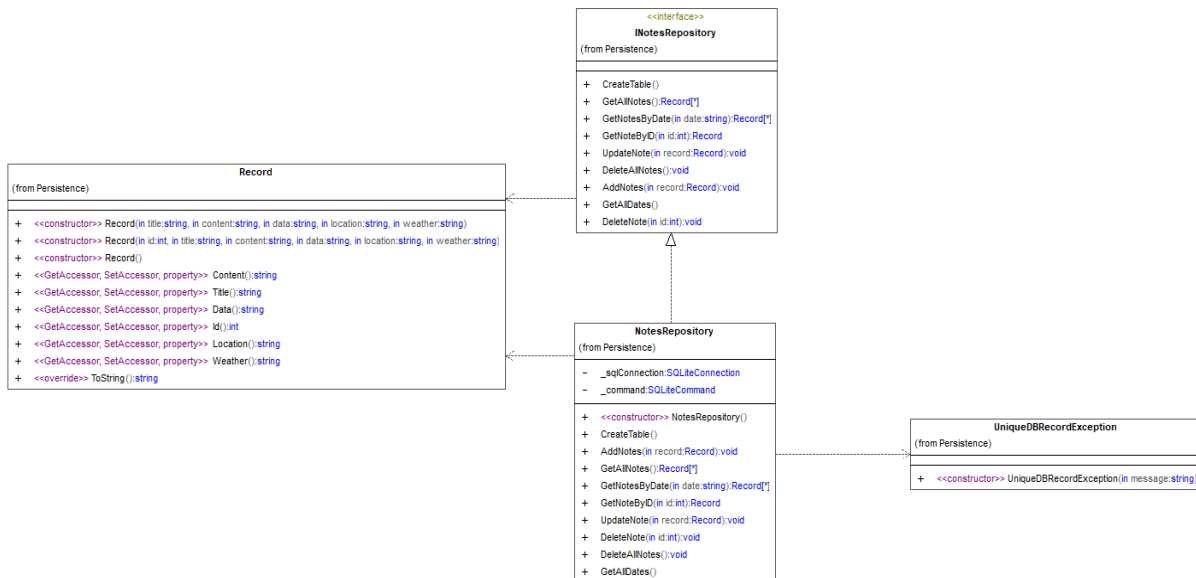


Generated by UModel

www.altova.com

➔ Persistence

Clasele din nivelul de persistență constau dintr-o interfață care este extinsă de clasa concretă pentru implementare. Se folosesc obiecte Record (clasa cu același nume pentru parametrii în funcții sau ca tipuri de return). În cazul în care este deja prezentă o înregistrare cu aceleași valori, este aruncată o instanță a clasei UniqueDBException. Se observă de asemenea implementarea operațiilor CRUD (Create, Read, Update, Delete).

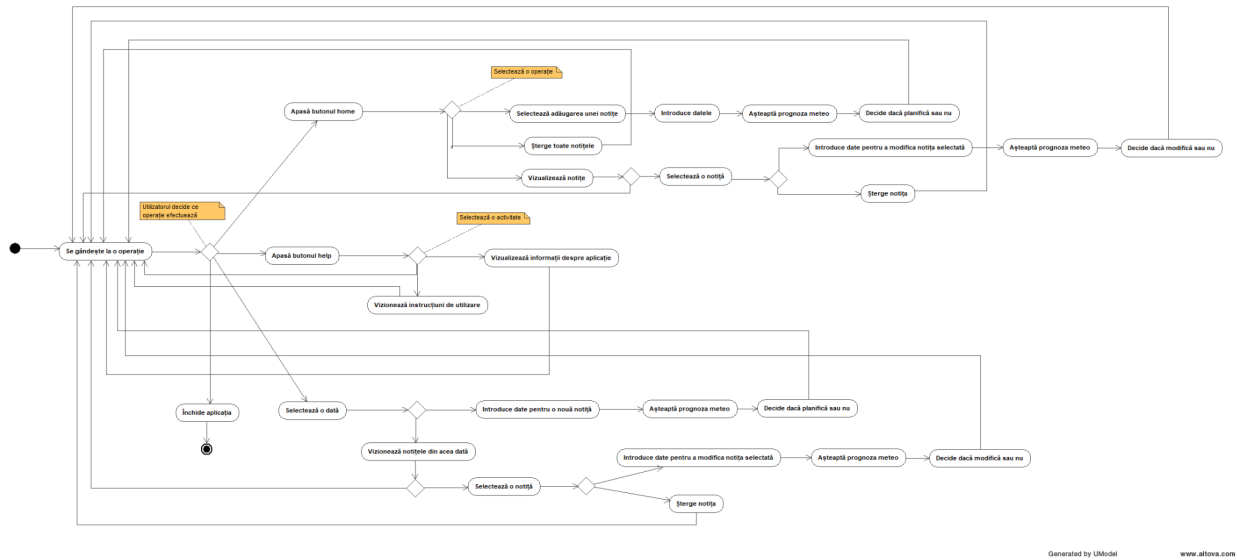


Generated by UModel

www.altova.com

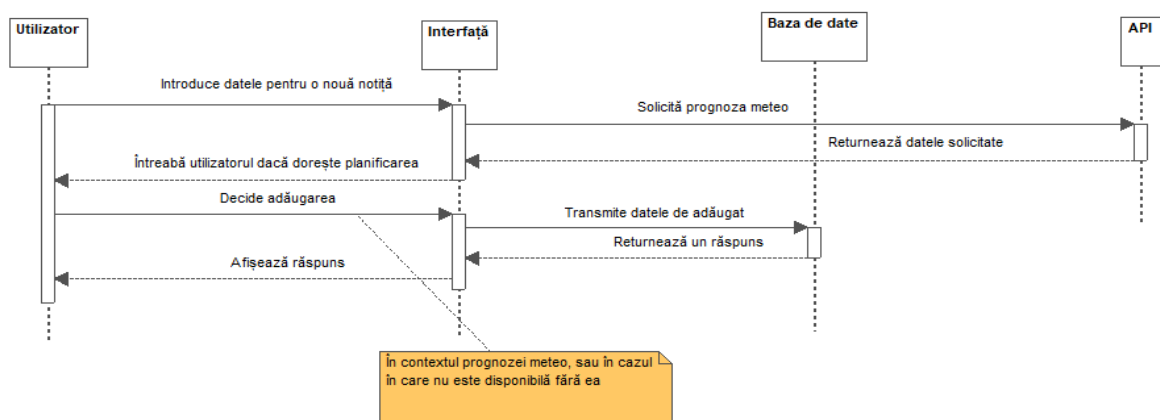
2. Diagrama de activități

Diagrama de activități cuprinde toate acțiunile posibile ale utilizatorului. Se observă în ea o tendință ciclică întrucât utilizatorul este pus aproape mereu în fața interfeței pentru a alege o operație. Starea finală este atinsă doar la părăsirea aplicației.

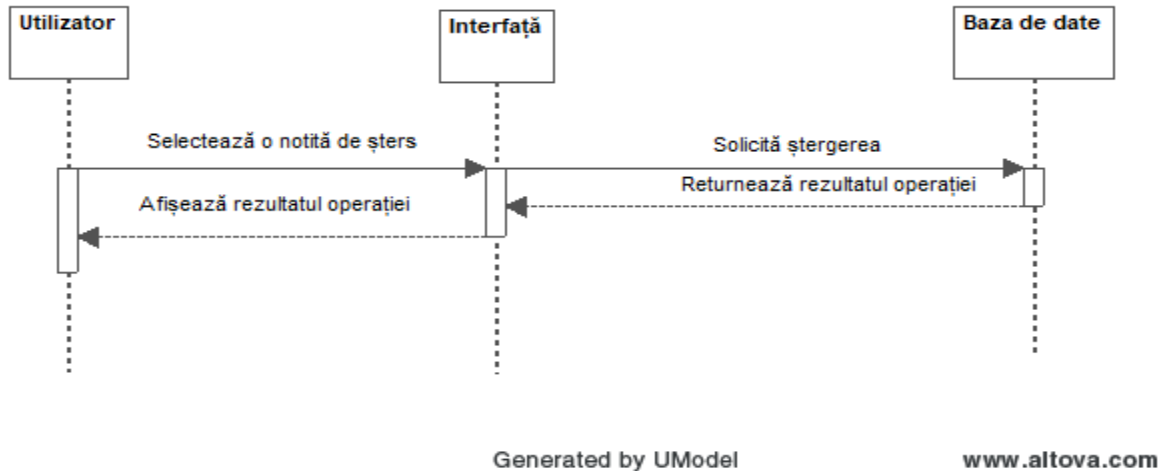


3. Diagrama de secvență

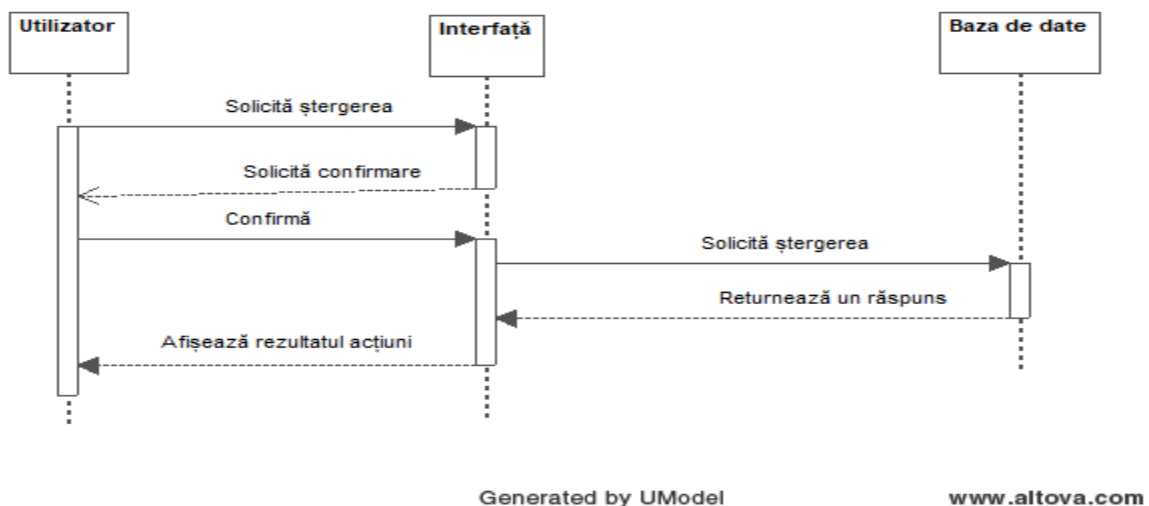
- Pentru accesarea operației de adăugare a notiței în calendar:
Se observă în următoarea diagramă o comunicare între utilizator și program (completare date, prezentare prognoză, alegere adăugare sau nu).



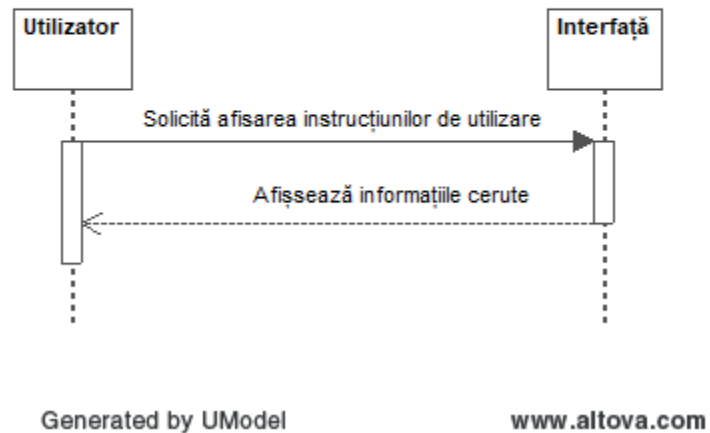
- Pentru accesarea operației de Delete a unei notițe:
În această diagramă se observă o secvență mai simplă de operații(doar succesiune de cerere și răspunsuri).



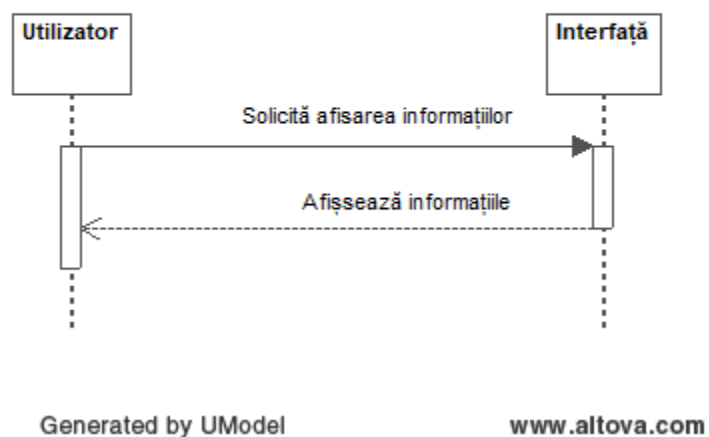
- Pentru accesarea operației de ștergere a tuturor notițelor:
Se observă în diagrama de mai jos că utilizatorul își dă încă o dată acordul pentru ștergere.



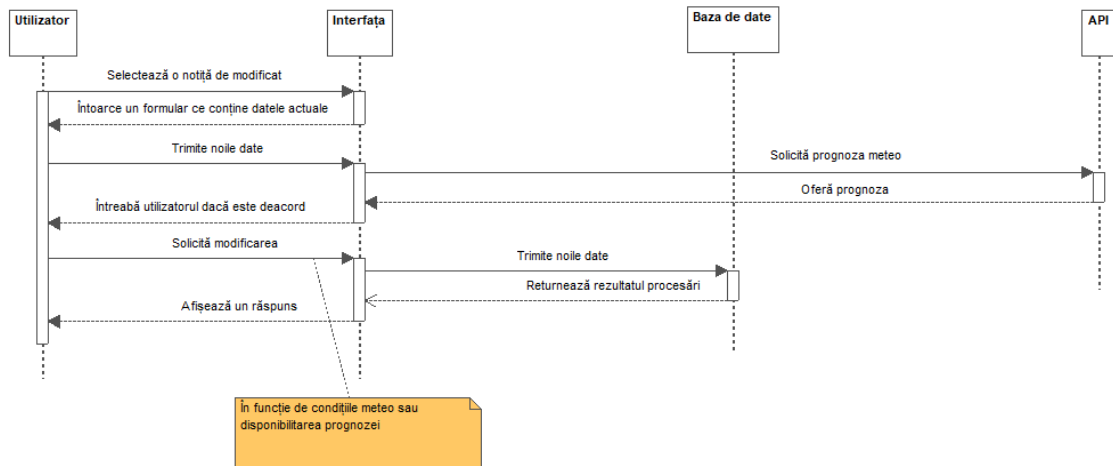
- Pentru accesarea butonului Help:
Diagrama evidențiază o operație simplă(se face o cerere pentru afișarea Help-ului și este returnat utilizatorului).



- Pentru accesarea butonului Info:
Se observă în această diagramă că solicitarea este urmată de afișarea datelor.



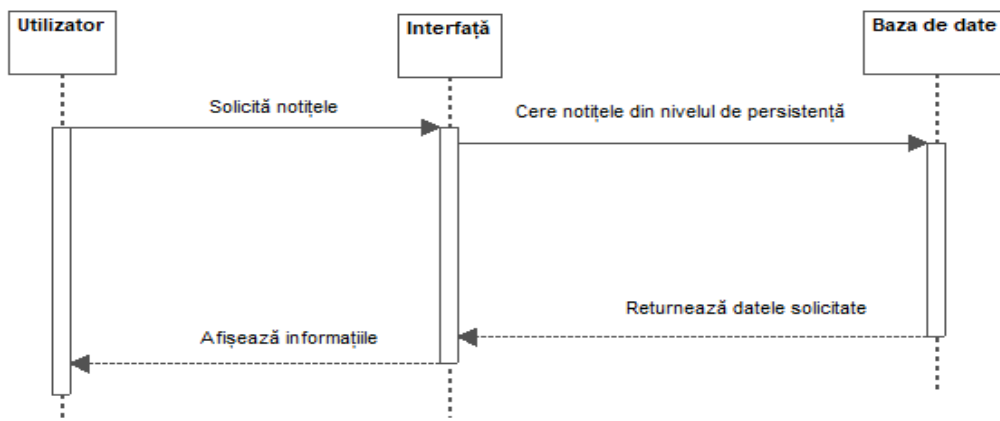
- Pentru accesarea operației de modificare a notiței:
- Această diagramă evidențiază un schimb de pași mai complex (afișare date inițiale, completare valori noi, solicitare vreme, afișare răspuns, confirmare/infirmare operație, primire status execuție operație).



Generated by UModel

www.altova.com

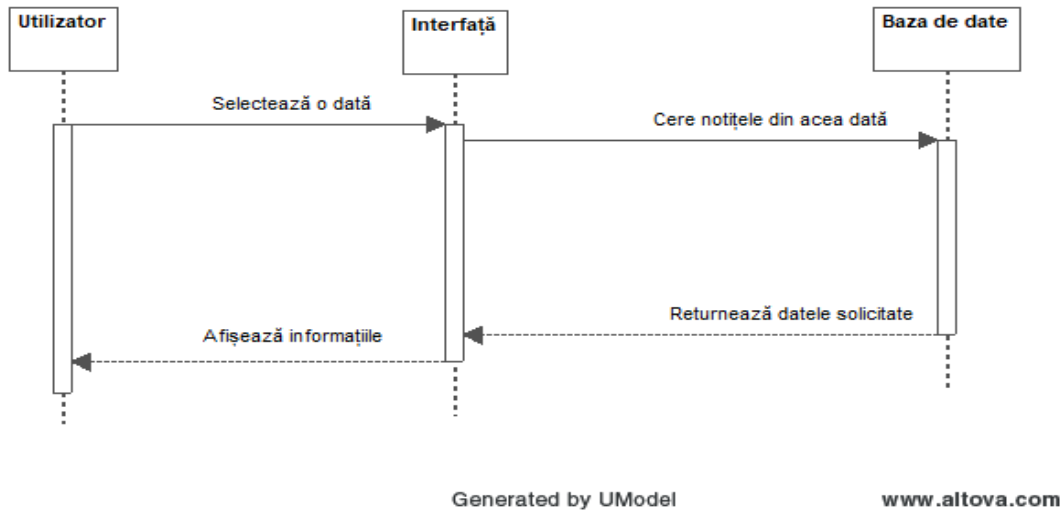
- Pentru a putea vedea toate notițele:
- Diagrama evidențiază o secvență simplă de pași (cereri și răspunsuri)



Generated by UModel

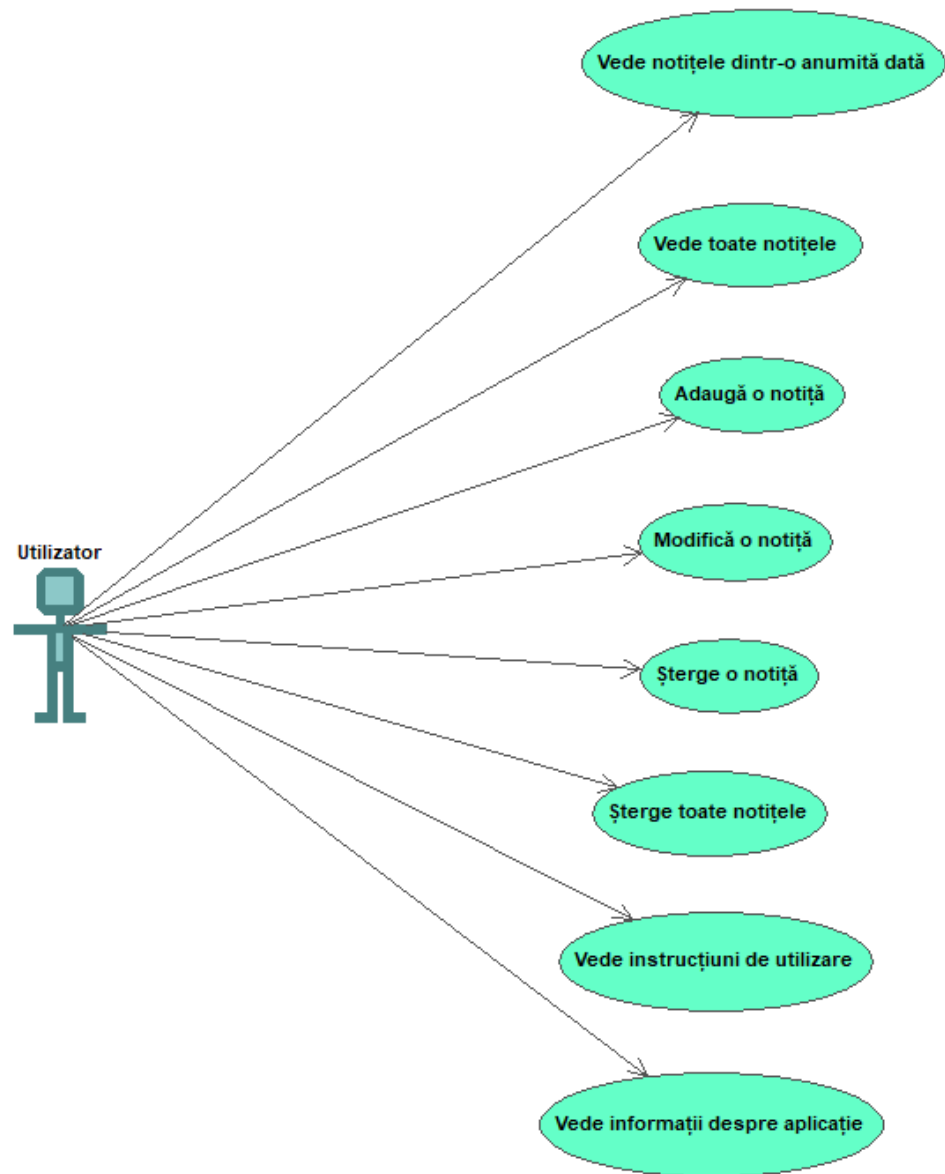
www.altova.com

- Pentru a putea vedea conținutul unei singure notițe:
Diagrama evidențiază o secvență simplă de pași (cereri și răspunsuri)



4. Diagrama de activități

Evidențiază toate operațiile pe care utilizatorul le poate efectua (Vizualizare date, adăugare, modificare, ștergere, vizualizare informații despre aplicație, vizualizare help).



Generated by UModel

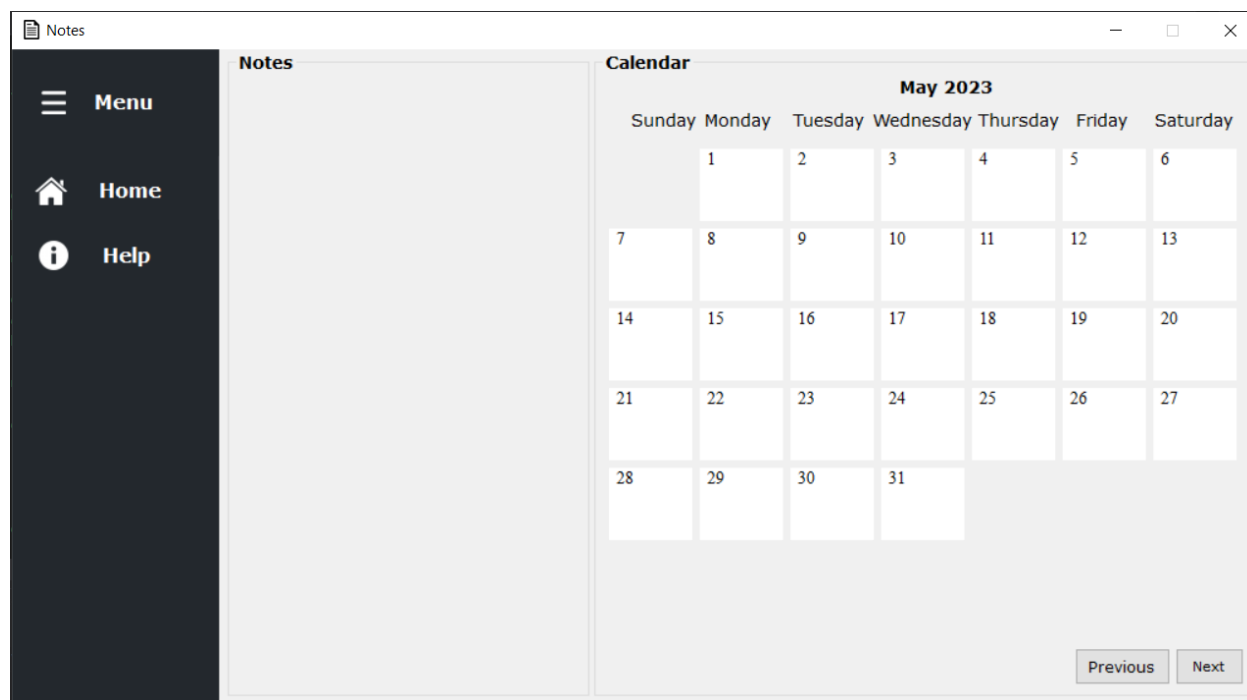
www.altova.com

3. Mod de utilizare al aplicației

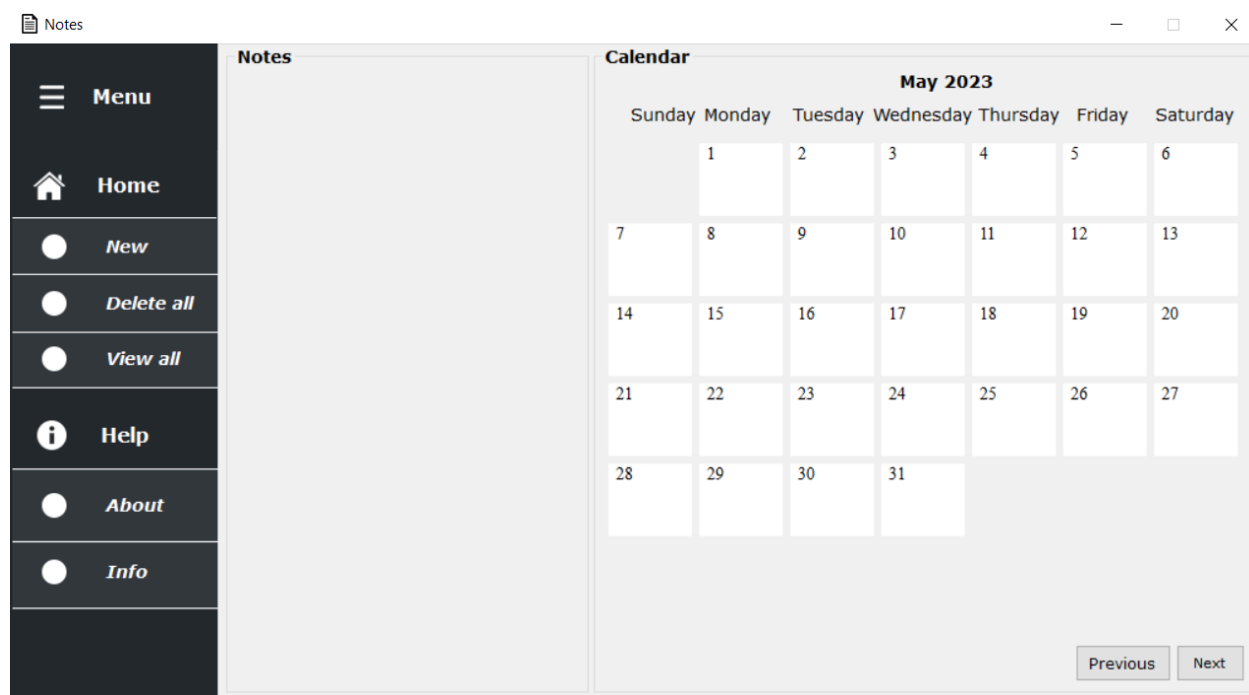
Planificatorul în funcție de condițiile meteo a fost creat așa încât să poată fi folosit de orice utilizator, indiferent de cunoștințe. Operațiile care pot fi realizate sunt foarte ușor de înțeles datorită butoanelor sugestive. Astfel, în cadrul aplicației, utilizatorul poate să:

- Adauge o notiță: prin accesarea butoanelor Home-New, sau printr-un click asupra unei zile din calendarul vizibil constant în partea dreaptă a interfeței.
- Vadă toate notițele: prin accesarea butoanelor Home-ViewAll. Astfel, acesta va găsi informațiile de care are nevoie.
- Șteargă toate notițele: cu un simplu click pe butoanele Home-DeleteAll, acesta poate să renunțe la conținutul de care nu mai are nevoie.
- Primească informații despre vreme: în momentul în care acesta accesează butonul de adăugare al unei notițe, o fereastră pop-up îl va informa cu privire la condițiile meteorologice din ziua aleasă. Există totuși posibilitatea de a salva notițele și dacă informațiile despre vreme nu pot fi transmise.
- Editeze sau să șteargă o notiță: în momentul în care apasă click-dreapta asupra acesteia. Astfel, dacă planurile sale s-au schimbat, acesta le poate modifica și în cadrul aplicației.
- Navigheze către diferite date ale calendarului: prin accesarea butoanelor sugestive, Next și Prev, cu ajutorul cărora acesta poate să își amintească evenimente trecute și să își planifice viitorul.

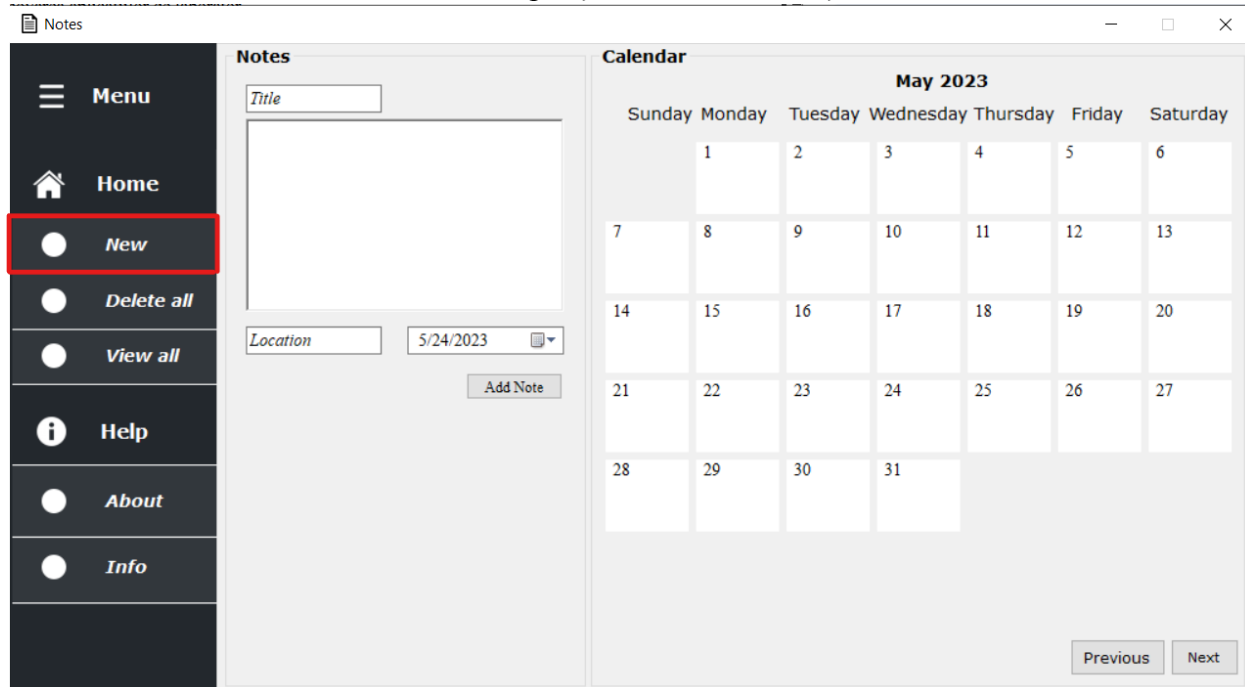
În cele ce urmează, sunt oferite câteva exemple de execuție ale aplicației. Astfel, la lansarea în execuție, utilizatorul va vedea următoarea interfață:



În următoarea imagine, sunt prezentate butoanele ce constau în operațiile pe care utilizatorul le poate realiza, fiecare având un nume sugestiv, pentru o mai bună înțelegere a funcționalităților:



Accesarea butonului New, va duce la apariția următorului conținut:



Astfel, utilizatorul poate introduce o notiță și poate alege și ziua în care să o salveze.

În momentul în care accesează butonul Add Note, în funcție de locația introdusă și data aleasă, va apărea o fereastră pop-up ce conține informațiile meteorologice. De exemplu, pentru locația Iași și data de 1 iunie 2023, conținutul pop-up-ului va fi:

Notes

Title: *Iesire*

Content: *Grădina botanică.*

Location: *Iasi*

Date: *6/ 1/2023*

Add Note

Calendar

May 2023

Sunday Monday Tuesday Wednesday Thursday Friday Saturday

1 2 3 4

7 8 9 10 11

18

25

Weather

The weather will have the following parameters:

Descriptions: Drizzle: Light

Maximum temperature: 22.1

Minimum temperature: 13.4

You want to add?

Yes **No**

Dacă se va apăsa butonul Yes, notița va fi adăugată în lista ce conține toate planificările și va putea fi vizualizată atât prin accesarea butonului ViewAll, cât și în momentul în care utilizatorul apasă în calendar pe data de 1.06.2023:

Notes

The number of notes is 4

Title: *Iesire*

Date: *2023-06-01*

Location: *Iasi*

Content: *Grădina botanică.*

Weather: Descriptions: Drizzle: Light

Maximum temperature: 22.1

Minimum temperature: 13.4

Calendar

May 2023

Sunday Monday Tuesday Wednesday Thursday Friday Saturday

1 2 3 4 5 6

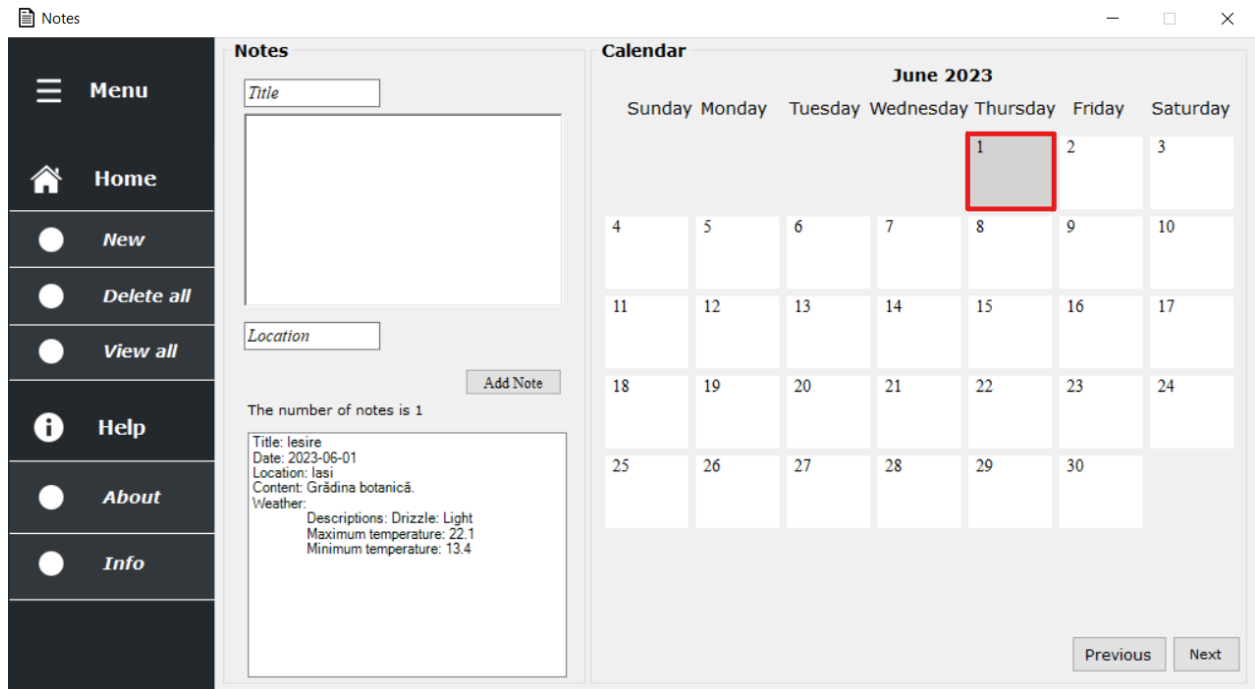
7 8 9 10 11 12 13

14 15 16 17 18 19 20

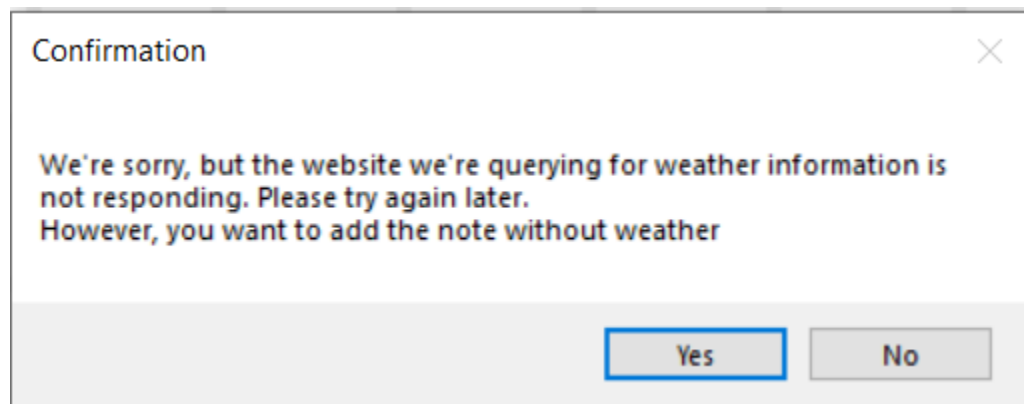
21 22 23 24 25 26 27

28 29 30 31

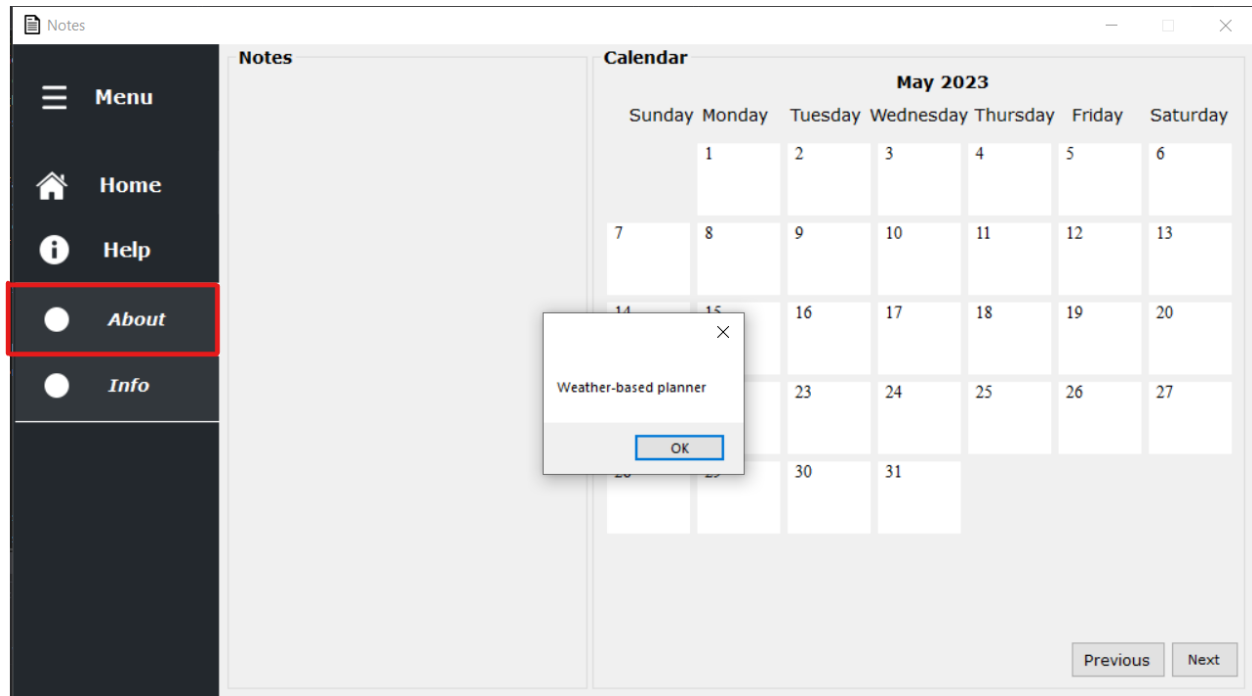
Previous **Next**



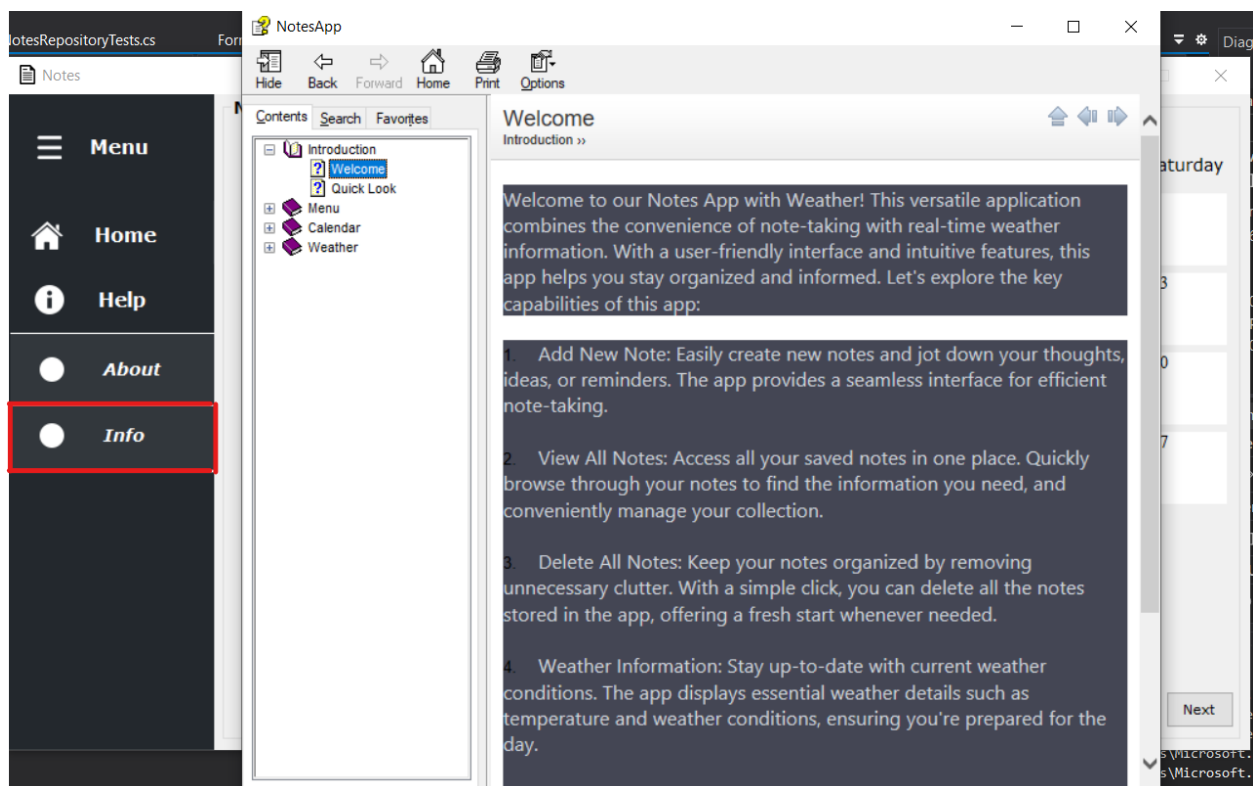
În cazul în care informațiile cu privire la datele meteorologice nu pot fi oferite, va exista totuși posibilitatea de a salva notița și fără acestea, acțiunea fiind posibilă prin apăsare butonului Yes în momentul în care va apărea următoarea fereastră pop-up:



La accesarea butonului About, pe interfață va apărea o fereastră pop-up cu următorul conținut:

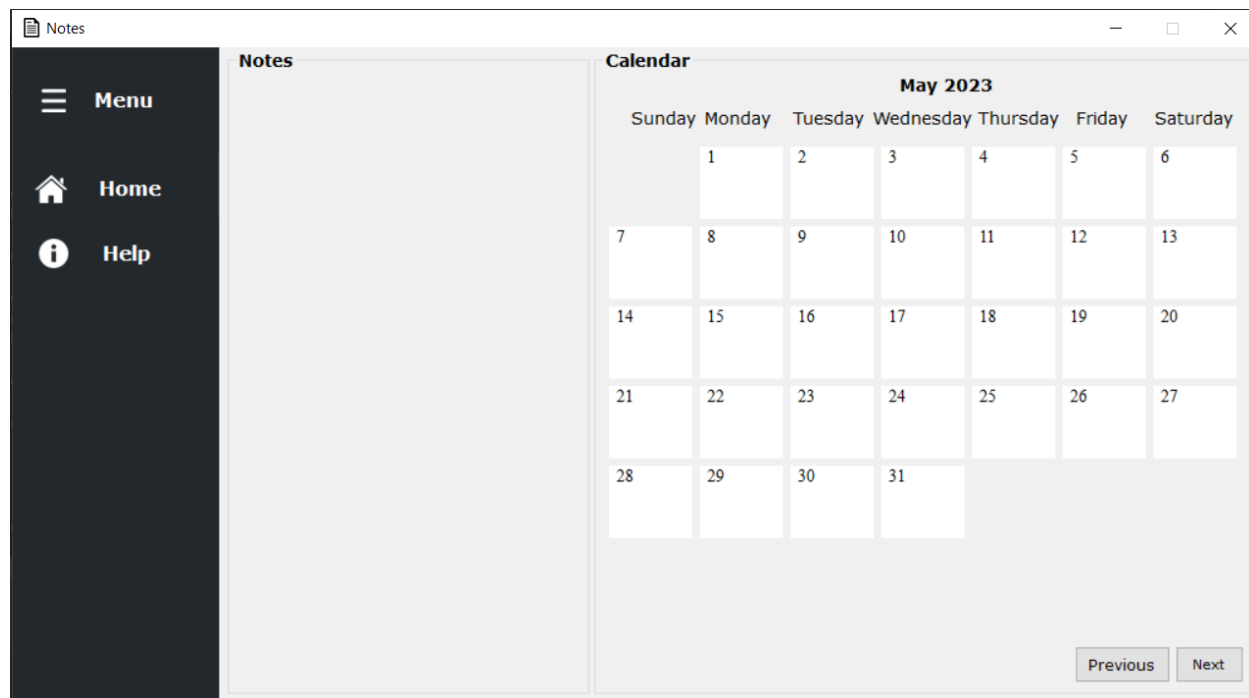


Accesarea butonului Info, va duce la deschiderea unui Help, realizat special pentru a oferi îndrumare utilizatorului în vederea utilizării aplicației.

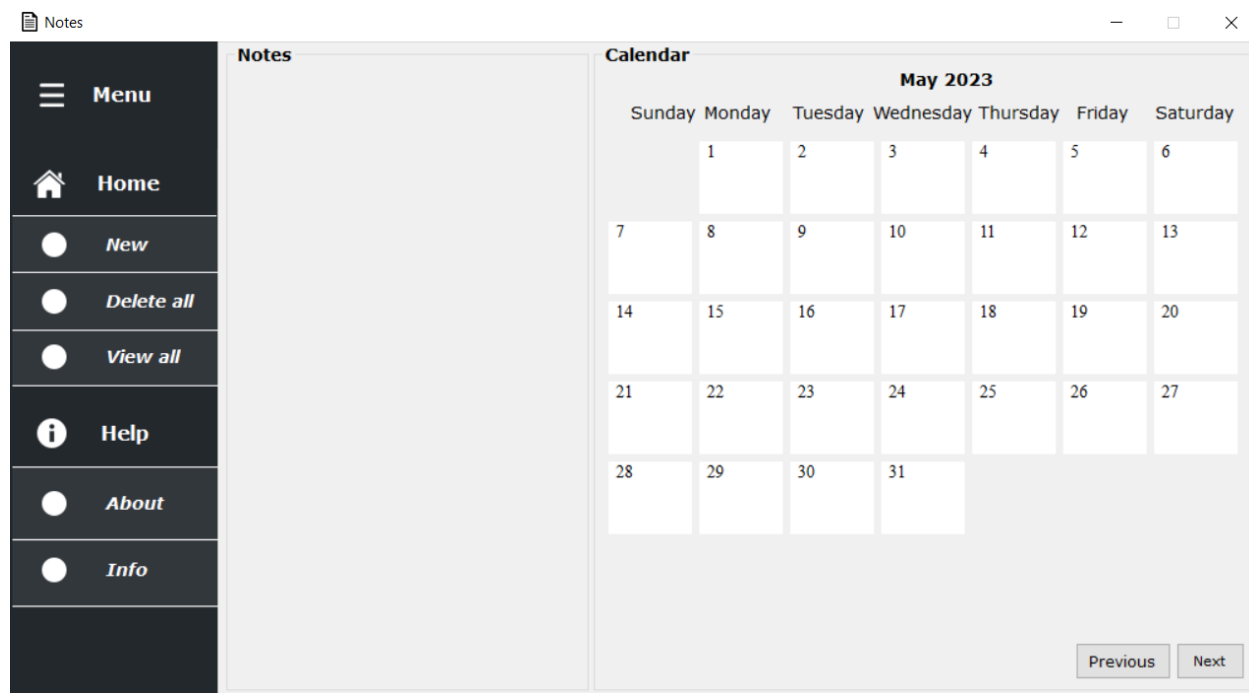


4. Exemple de execuție ale aplicației

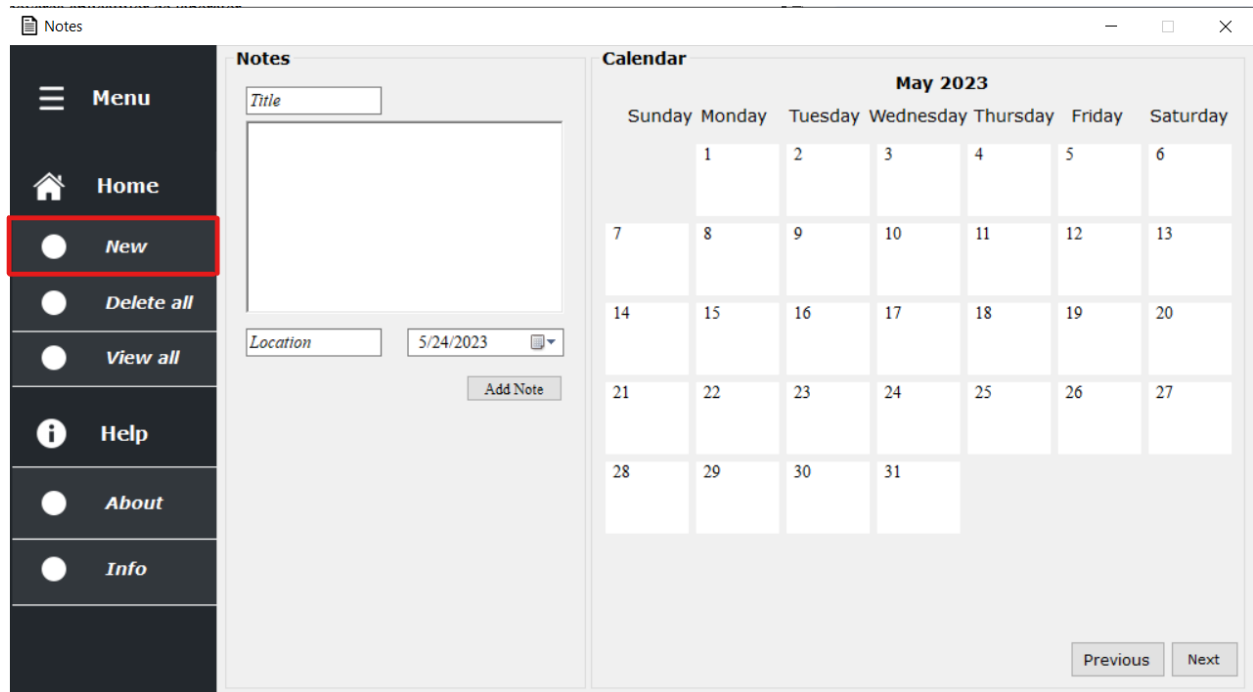
Interfața de la deschiderea aplicației



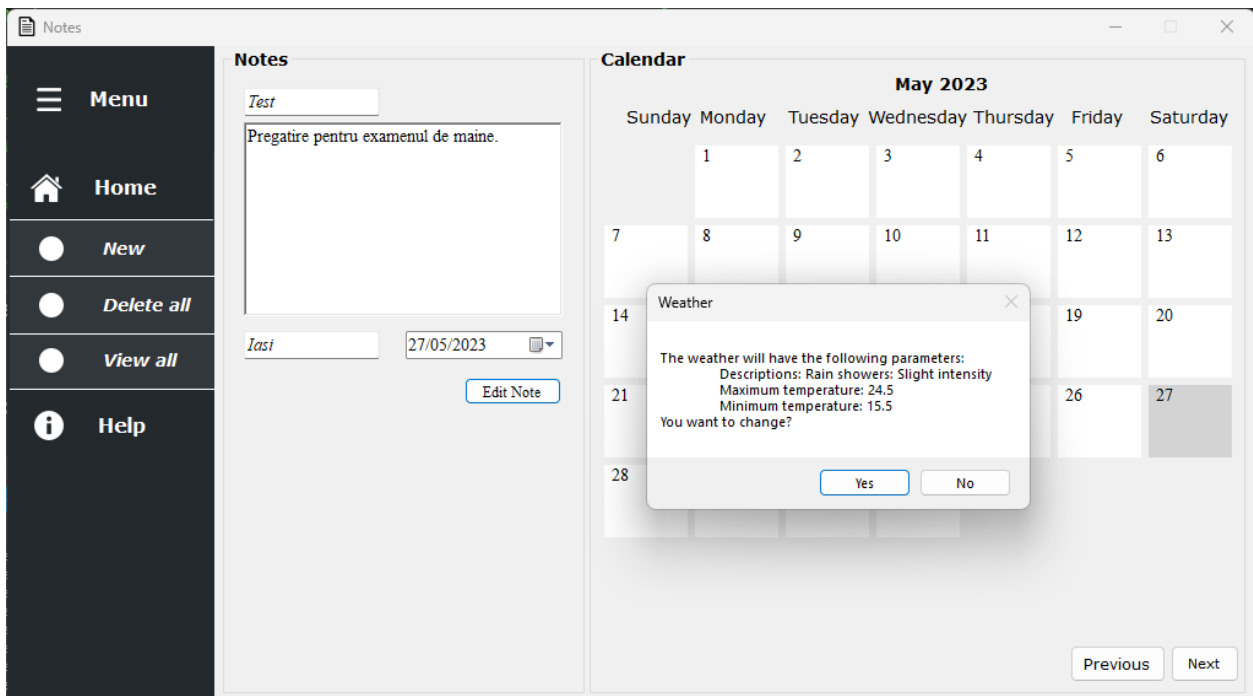
Meniul expandat



Adăugarea unei noi notițe



Editarea unei notițe



Mesaj pop-up cu prognoza meteo la adăugare

Notes

Iesire

Grădina botanică.

Iasi

6/ 1/2023

Add Note

Calendar

May 2023

Sunday	Monday	Tuesday	Wednesday	Thursday
	1	2	3	4
7	8	9	10	11
				18
				25

Weather

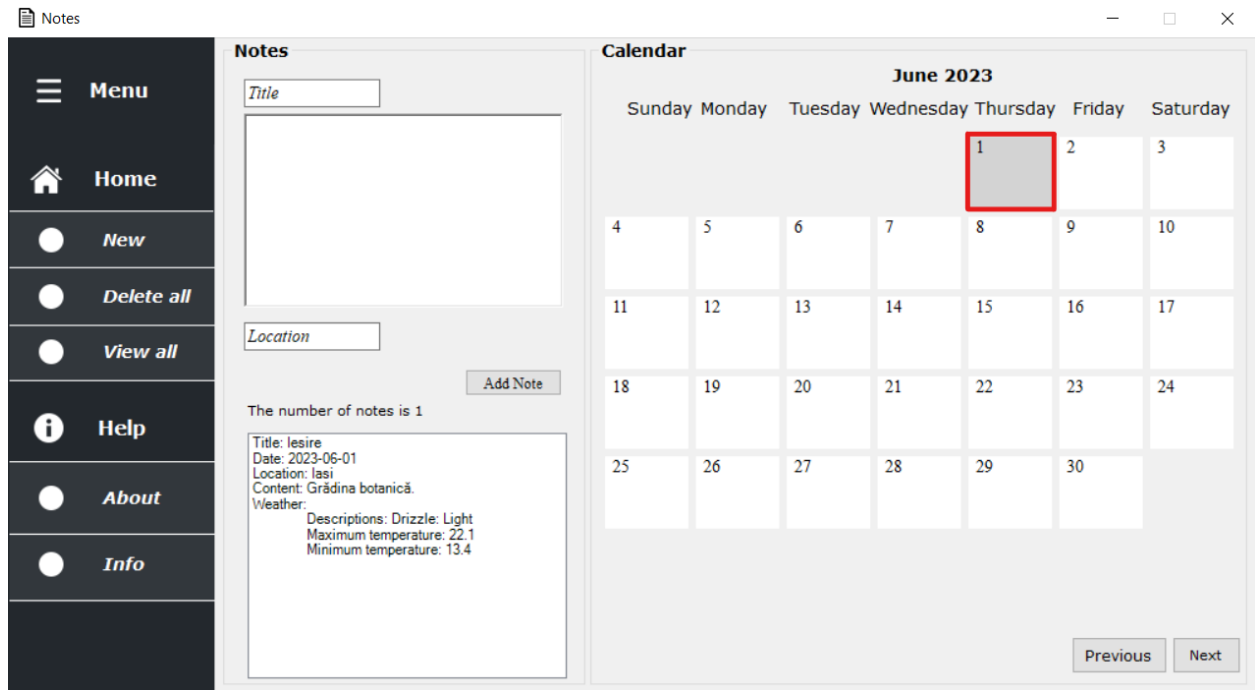
The weather will have the following parameters:
Descriptions: Drizzle: Light
Maximum temperature: 22.1
Minimum temperature: 13.4
You want to add?

YesNo

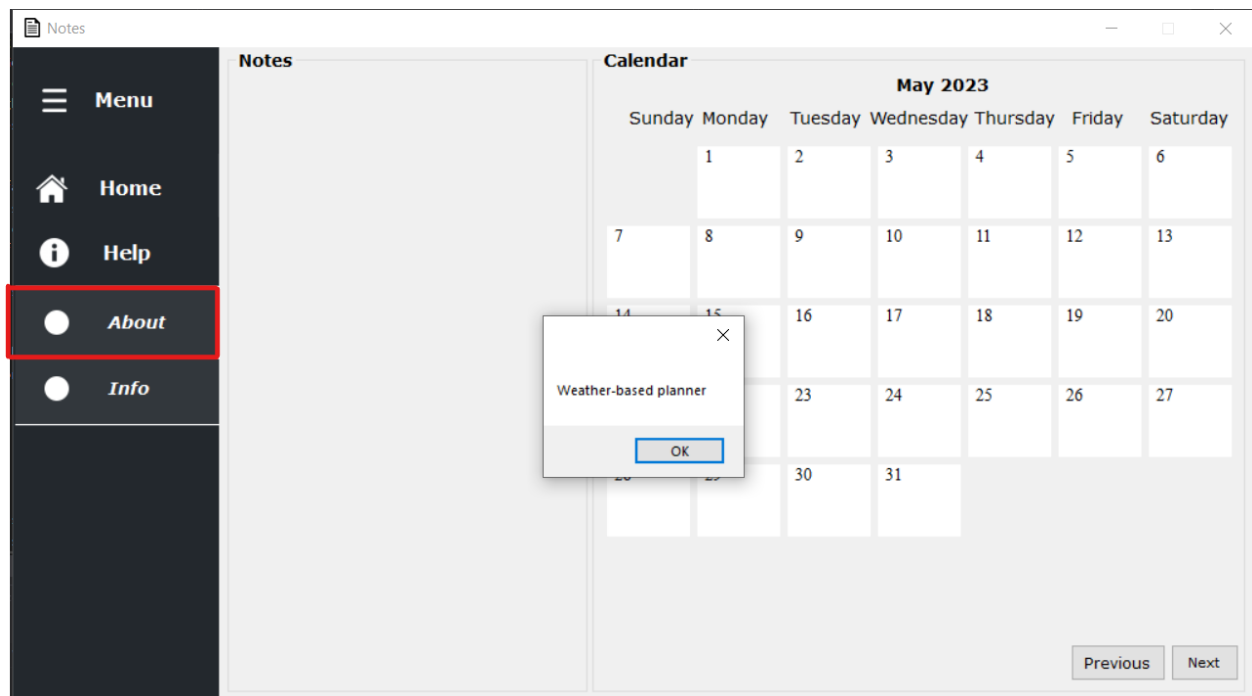
Vizualizarea tuturor notițelor

The screenshot displays a desktop application with a dark sidebar on the left containing a menu with options: Menu, Home, New, Delete all, View all (highlighted with a red rectangle), Help, About, and Info. The main area is divided into two panels. The left panel, titled 'Notes', shows a list of notes with the header 'The number of notes is 4'. The selected note details are: Title: Iesire, Date: 2023-06-01, Location: Iasi, Content: Grădina botanică, Weather: Descriptions: Drizzle: Light, Maximum temperature: 22.1, Minimum temperature: 13.4. The right panel, titled 'Calendar', shows a calendar for May 2023. The calendar grid has columns for Sunday through Saturday. The dates 1 through 31 are displayed in the grid. At the bottom right of the calendar panel, there are buttons for 'Previous' and 'Next'.

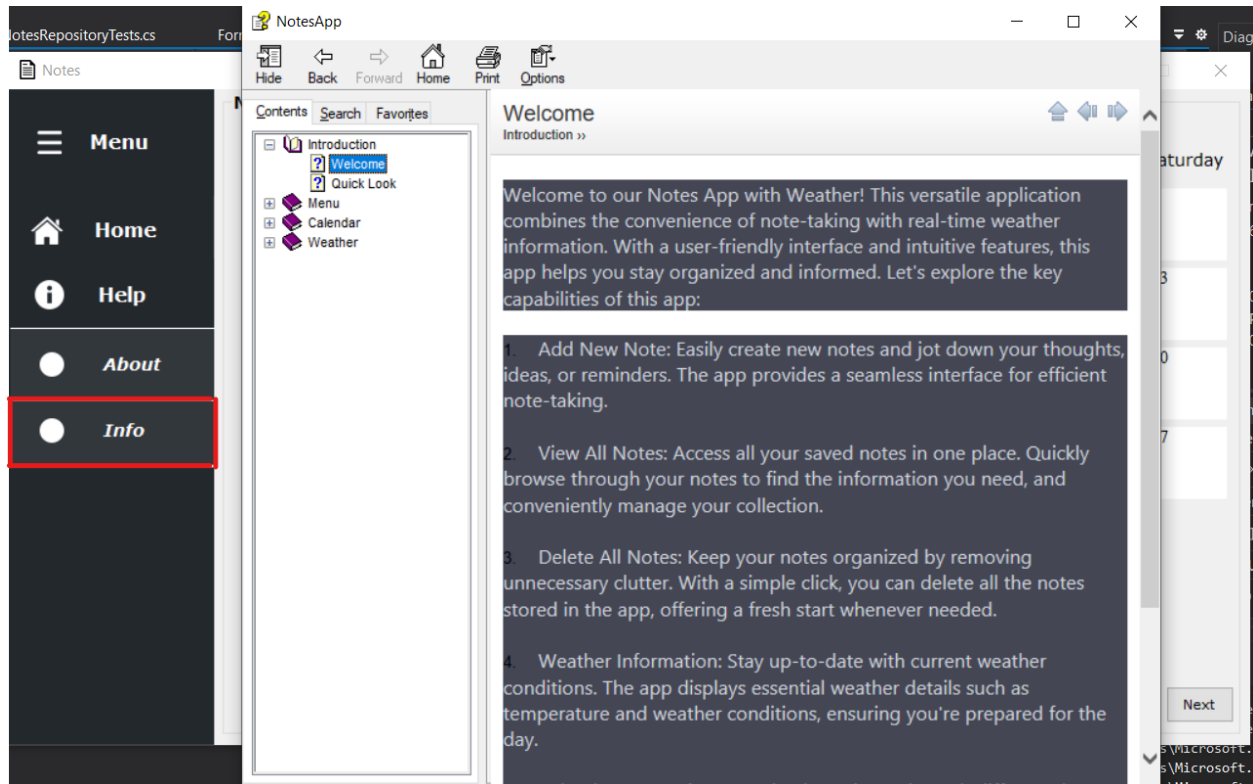
Vizualizarea notițelor dintr-o anumită dată



Informații despre aplicație



Help-ul pentru utilizator



Anexa 1: Exemple ale codului sursă asociat

A. Metode reprezentative pentru realizarea interfeței

1. Pentru a putea realiza calendarul din interfață, s-a utilizat componenta UserControl. Astfel, cu ajutorul unui control gol în care s-a introdus un label, s-au putut crea zilele pentru fiecare lună și an. Acest lucru a fost implementat cu ajutorul funcției displayCalendar()

```
private void displayCalendar()
{
    //afisare luna, an
    String monthName =
DateTimeFormatInfo.CurrentInfo.GetMonthName(_currentMonth);
    labelLunaAn.Text = monthName + " " + _currentYear;

    //prima zi din luna
    DateTime firstday = new DateTime(_currentYear, _currentMonth, 1);

    //numarul de zile din luna curenta
    int days = DateTime.DaysInMonth(_currentYear, _currentMonth);

    //ziua curenta convertita la int
    int currentday = Convert.ToInt32(firstday.DayOfWeek.ToString("d"))
+ 1;

    for (int i = 1; i < currentday; i++)
    {
        //se adauga obiecte de tip usercontrol goale(aceeasi culoare ca
background ul)
        UserControlEmpty ucempty = new UserControlEmpty();

        panelDay.Controls.Add(ucempty);
    }

    for (int i = 1; i <= days; i++)
    {
        //se adauga obiectele de tip usercontrol care vor fi efectiv
panel urile cu zilele lunii
        MyUserControlDays ucdays = new MyUserControlDays();
        ucdays.Days(i);

        ucdays.Date = _currentYear + "-" +
_currentMonth.ToString().PadLeft(2, '0') + "-" + i.ToString().PadLeft(2, '0');

        //inregistrare callback pentru apasare pe o data
        ucdays.Click += new EventHandler(UserControlDays_Click);
        panelDay.Controls.Add(ucdays);
    }
}
```

2. Pentru a putea realiza o animație în cazul barei de meniu, s-a utilizat evenimentul “Tick”. În acest sens, s-a utilizat o variabilă booleană, pentru a ști în ce stadiu se află animația (meniu maximizat sau minimizat). În momentul în care meniul este minimizat, nu se pot accesa celelalte butoane pe care le conține, motiv pentru care s-a utilizat proprietatea Enabled. Pentru a putea porni animația, la apelul funcției de callback menuButton_Click, se va apela metoda sidebarTimer.Start().

```
private void sidebarTimer_Tick(object sender, EventArgs e)
{
    //metoda pentru extinderea barei de meniu (se apasa pe icon
    ul de langa meniu)
    if (!_isSidebarExpand)
    {
        //bara de meniu expandata, se va minimiza
        sidebarMenu.Width -= 10;
        if (sidebarMenu.Width == sidebarMenu.MinimumSize.Width)
        {
            _isSidebarExpand = false;
            sidebarTimer.Stop();
            //cand se minimizeaza meniul, se va muta si pozitia
            groupBoxNote.Location = new Point(124, 2);
            buttonHome.Enabled = false;
            buttonHelp.Enabled = false;
            buttonAbout.Enabled = false;
            buttonDelete.Enabled = false;
            buttonNew.Enabled = false;
            buttonModify.Enabled = false;
            buttonInfo.Enabled = false;
        }
    }
    else
    {
        //bara de meniu minimizata, se va expanda la atingerea
        icon-ului pentru meniu
        sidebarMenu.Width += 10;
        if (sidebarMenu.Width == sidebarMenu.MaximumSize.Width)
        {
            _isSidebarExpand = true;
            sidebarTimer.Stop();
            groupBoxNote.Location = new Point(176, 2);
            buttonHome.Enabled = true;
            buttonHelp.Enabled = true;
            buttonAbout.Enabled = true;
            buttonDelete.Enabled = true;
            buttonNew.Enabled = true;
            buttonModify.Enabled = true;
            buttonInfo.Enabled = true;
        }
    }
}
```

În aceeași idee, au fost implementate metodele și pentru butoanele Home și About:

1. Home (cu apelul homeTimer.Start() în funcția de callback buttonHome_Click):

```
private void homeTimer_Tick(object sender, EventArgs e)
{
    //metoda pentru extinderea butonului Home
    if (_isHomeExpand)
    {
        panelHome.Height -= 10;
        if (panelHome.Height ==
panelHome.MinimumSize.Height)
        {
            _isHomeExpand = false;
            homeTimer.Stop();
        }
    }
    else
    {
        panelHome.Height += 10;
        if (panelHome.Height ==
panelHome.MaximumSize.Height)
        {
            _isHomeExpand = true;
            homeTimer.Stop();
        }
    }
}
```

2. About (cu apelul helpTimer.Start() în funcția de callback buttonHelp_Click):

```
private void helpTimer_Tick(object sender, EventArgs e)
{
    if (_isHelpExpand)
    {
        //se minimizeaza
        panelHelp.Height -= 10;
        if (panelHelp.Height ==
panelHelp.MinimumSize.Height)
        {
            _isHelpExpand = false;
            helpTimer.Stop();
        }
    }
    else
    {
        //se extinde
        panelHelp.Height += 10;
        if (panelHelp.Height ==
panelHelp.MaximumSize.Height)
        {
            _isHelpExpand = true;
            helpTimer.Stop();
        }
    }
}
```

B. Modulul de persistență

1. Nivelul de persistență a presupus folosirea unei baze de date SQLite, în contextul cerințelor (datele accesate doar de pe sistemul pe care este executată aplicația) nu se justifică folosirea unor soluții care au și caracter distribuit.

```
public NotesRepository()
{
    //inițializarea conexiuni cu baza de date SQLite
    _sqlConnection = new SQLiteConnection(@"Data Source =
Resources\notes.db; Version = 3; New = True; Compress = True;");
    //variabila pentru comandă, pentru executarea uneia explicite se
vor modifica anumiți parametri pentru această variabilă
    _command = _sqlConnection.CreateCommand();
}
```

2. Funcția pentru crearea tabelului (dacă ea nu există). Au fost incluse constrângeri de unicitate a unei înregistrări și indexarea automată. Totodată s-a avut în vedere tratarea excepțiilor, acestea fiind aruncate spre un nivel mai apropiat de utilizator (sunt afișate dacă apar ca mesaje popup).

```
public void CreateTable()
{
    try
    {
        _sqlConnection.Open();

        //comanda SQL pentru crearea tabelului dacă ea nu există
        //inclusiv constrângerea aferentă
        string sqlString = "CREATE TABLE IF NOT EXISTS
NotesWeather (id INTEGER PRIMARY KEY AUTOINCREMENT," + " title
VARCHAR(50), content VARCHAR(200), data VARCHAR(20), location
VARCHAR(20),weather TEXT, UNIQUE(title,content,data,location))";
        _command.CommandText = sqlString;
        _command.ExecuteNonQuery();
    }

    catch (Exception exception)
    {
        //excepția va fi tratată la un alt nivel
        throw exception;
    }
    finally
    {
        //închiderea conexiuni indiferent dacă au fost sau nu
        //excepții
        _sqlConnection.Close();
    }
}
```

3. Inserarea în baza de date.

```
public void AddNotes(in Record record)
```

```

    {
        try
        {
            _sqlConnection.Open();

            //Pentru inserare sa folosit "SQL injection" astfel încât
            //eventualele caractere speciale din cod
            //să nu introducă erori funcționale.
            string sqlString = "INSERT INTO NotesWeather (id, title,
content, data, location,weather)
VALUES (null,@title,@content,@data,@location,@weather)";
            _command.CommandText = sqlString;
            _command.Parameters.Add(new SQLiteParameter("@title",
record.Title));
            _command.Parameters.Add(new SQLiteParameter("@content",
record.Content));
            _command.Parameters.Add(new SQLiteParameter("@data",
record.Data));
            _command.Parameters.Add(new SQLiteParameter("@location",
record.Location));
            _command.Parameters.Add(new SQLiteParameter("@weather",
record.Weather));
            _command.ExecuteNonQuery();
        }
        catch (SQLiteException exception)
        {
            //Aplicația nu permite crearea unei înregistrări cu
aceiași parametri
            if (exception.Message.Contains("UNIQUE"))
            {
                throw new UniqueDBRecordException("Exista deja o
astfel de înregistrare în baza de date!");
            }
            else
            {
                throw exception;
            }
        }
        catch (Exception exception)
        {
            throw exception;
        }
        finally
        {
            //închiderea conexiuni indiferent dacă au fost sau nu
excepții
            _sqlConnection.Close();
        }
    }
}

```

4. O altă operație strict necesară în contextul nostru este extragerea datelor din baza de date (fie dintr-o anumită dată fie toate înregistrările). În acest cod este exemplificată extragerea tuturor înregistrărilor.

```

public Record[] GetAllNotes()

```

```

{
    SQLiteDataReader sqlitedatareader = null;
    int id;
    string title;
    string data;
    string location;
    string content;
    string weather;
    //Lista cu înregistrările de returnat
    List<Record> listOfRecord = new List<Record>();

    try
    {
        _sqlConnection.Open();

        //comanda SQL
        _command.CommandText = "SELECT * FROM NotesWeather";

        sqlitedatareader = _command.ExecuteReader();
        while (sqlitedatareader.Read())
        {
            //Preluarea datelor
            //id, title, content, data, location
            id = sqlitedatareader.GetInt32(0);
            title = sqlitedatareader.GetString(1);
            content = sqlitedatareader.GetString(2);
            data = sqlitedatareader.GetString(3);
            location = sqlitedatareader.GetString(4);
            weather = sqlitedatareader.GetString(5);
            Record readRecord = new Record(id, title, content,
data, location, weather);
            listOfRecord.Add(readRecord);
        }
    }
    catch (Exception e)
    {
        throw e;
    }
    finally
    {
        sqlitedatareader.Close();
        _sqlConnection.Close();
    }

    return listOfRecord.ToArray();
}

```


5. Operația de actualizare a unei notițe folosește și ea tehnica "SQL injection"

```
public void UpdateNote(in Record record)
{
    try
    {
        _sqlConnection.Open();
        //Stringul pentru update
        //S-a optat pentru actualizarea tuturor câmpurilor
        string sqlString = "UPDATE NotesWeather set
title=@title,content=@content,data=@data,location=@location,weather=@weather where id=@id";
        _command.CommandText = sqlString;
        _command.Parameters.Add(new
SQLiteParameter("@id", record.ID));
        _command.Parameters.Add(new SQLiteParameter("@title",
record.Title));
        _command.Parameters.Add(new SQLiteParameter("@content",
record.Content));
        _command.Parameters.Add(new SQLiteParameter("@data",
record.Data));
        _command.Parameters.Add(new SQLiteParameter("@location",
record.Location));
        _command.Parameters.Add(new SQLiteParameter("@weather",
record.Weather));
        _command.ExecuteNonQuery();
    }
    catch (System.Data.SQLite.SQLiteException exception)
    {
        if (exception.Message.Contains("UNIQUE"))
        {
            throw new UniqueDBRecordException("Exista deja o
astfel de înregistrare în baza de date!");
        }
    }
    finally
    {
        _sqlConnection.Close();
    }
}
```

6. Ștergerea unei notițe se face după un ID. Singura diferență pentru ștergerea tuturor este eliminarea clauzei "WHERE", din acest motiv atașăm doar un exemplu pentru ștergerea unei singure notițe.

```
public void DeleteNote(in int id)
{
    try
    {
        _sqlConnection.Open();
        string sqlString = "DELETE FROM NotesWeather WHERE
id=@id";
        _command.CommandText = sqlString;
        _command.Parameters.Add(new SQLiteParameter("@id", id));
        _command.ExecuteNonQuery();
    }
}
```

```

    }
    catch (Exception exception)
    {
        throw exception;
    }
    finally
    {
        _sqlConnection.Close();
    }
}

```

C. Modulul pentru preluarea datelor de la API

1. Conectarea la clientul HTTP și introducerea cheii pentru API-ul de GeoLocatie.

```

private HttpClient _httpClient;

public GeocodingLoc()
{
    _httpClient = new HttpClient();
    _httpClient.DefaultRequestHeaders.Add("X-API-Key",
"er4kKN1GFWVTZbhbqUH9wA==rxDpclohWicLlua");
}

```

2. Preluarea datelor din răspunsul HTTP ca și string și deserializarea sa într-un obiect JSON.

```

public async Task<string> GetStringGeo(string location)
{
    string responseBody = " ";
    try
    {
        HttpResponseMessage response =
        _httpClient.GetAsync("https://api.api-ninjas.com/v1/geocoding?city=" +
        location).Result;

        if (response.IsSuccessStatusCode)
        {
            responseBody =
            response.Content.ReadAsStringAsync().Result;
        }
        else
        {
            throw new System.Exception($"Geocoding API call
            failed with status code {response.StatusCode}");
        }
    }
    catch (System.Exception ex)
    {
        throw new System.Exception($"An error occurred:
        {ex.ToString()}");
    }
}

```

```

        _httpClient.Dispose();
        // return responseBody;
        // Varianta alternativa pentru test successful
        return await Task.FromResult(responseBody);
    }

    public Coordinates[] GetLocationCoord(string strLocation)
    {
        Task<string> responseAsString = GetStringGeo(strLocation);

        return
        JsonConvert.DeserializeObject<Coordinates[]>(responseAsString.Result);
    }

```

3. Construirea request-ului HTTP pentru cel de-al doilea API, utilizând latitudinea și longitudinea obținute din obiectul JSON preluat anterior și preluarea datelor legate de vremea pentru o dată formatată astfel "20xx-xx-xx" și deserializarea datelor în JSON.

```

    public async Task<string> GetStringByCoord(double lat, double lon, string
date)
    {
        string responseBody = " ";
        try
        {
            HttpResponseMessage response =
            _httpClient.GetAsync("https://api.open-meteo.com/v1/forecast?"
+ "latitude=" + lat
+ "&longitude=" + lon
+
"&daily=weathercode,temperature_2m_max,temperature_2m_min"
+ "&start_date=" + date
+ "&end_date=" + date
+ "&timezone=auto").Result;

            if (response.IsSuccessStatusCode)
            {
                responseBody =
response.Content.ReadAsStringAsync().Result;
            }
            else
            {
                Console.WriteLine($"WeatherAPI call failed with
status code {response.StatusCode}");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
        _httpClient.Dispose();
        return responseBody;
    }

```

```

public WeatherData GetWeatherData(double lat, double lon, string date)
{
    Task<string> stringResponse = GetStringByCoord(lat, lon,
date);

    return
JsonConvert.DeserializeObject<WeatherData>(stringResponse.Result);
}

```

4. Realizarea comunicației dintre cele 2 API-uri pentru returnarea stării vremii pentru o dată și o locație anume.

```

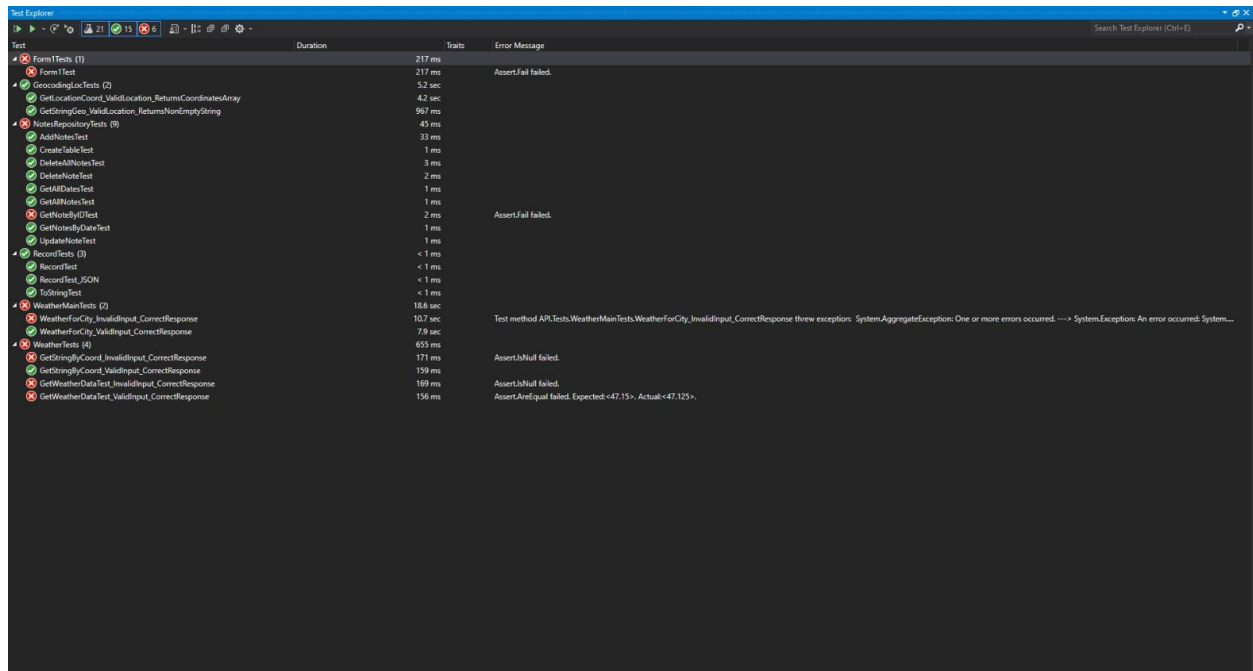
public WeatherData WeatherForCity(string city, string date)
{
    Coordinates[] coordinates =
_geocodingLoc.GetLocationCoord(city);

    return _weather.GetWeatherData(coordinates[0].Latitude,
coordinates[0].Longitude, date);
}

```

Anexa 2: Etapa de testare

Cele 21 de teste:



Test	Duration	Traits	Error Message
Form1Tests (1)	217 ms		
Form1Test	217 ms		Assert.Fail failed.
GeocodingLocTests (2)	52 sec		
GetLocationCoord_ValidLocation_ReturnsCoordinatesArray	42 sec		
GetStringGeo_ValidLocation_ReturnsNonEmptyString	967 ms		
NotesRepositoryTests (8)	45 ms		
AddNotesTest	33 ms		
CreateTableTest	1 ms		
DeleteAllNotesTest	3 ms		
DeleteNoteTest	2 ms		
GetAllDatesTest	1 ms		
GetAllNotesTest	1 ms		
GetNoteByIdTest	2 ms		Assert.Fail failed.
GetNotesByDateTest	1 ms		
UpdateNoteTest	1 ms		
RecordTests (3)	< 1 ms		
RecordTest	< 1 ms		
RecordTest_JSON	< 1 ms		
RoutingTest	< 1 ms		
WeatherMainTests (2)	184 sec		
WeatherForCity_InvalidInput_CorrectResponse	10.7 sec		Test method AP.Tests.WeatherMainTests.WeatherForCity_InvalidInput_CorrectResponse threw exception: System.AggregateException: One or more errors occurred. -> System.Exceptions An error occurred: System...
WeatherForCity_ValidInput_CorrectResponse	7.9 sec		
WeatherTests (4)	655 ms		
GetStringByCoord_InvalidInput_CorrectResponse	171 ms		Assert.IsNull failed.
GetStringByCoord_ValidInput_CorrectResponse	199 ms		
GetWeatherDataTest_InvalidInput_CorrectResponse	169 ms		Assert.IsNotNull failed.
GetWeatherDataTest_ValidInput_CorrectResponse	156 ms		Assert.AreEqual failed. Expected:<47.15>. Actual:<47.125>.

1. Teste necesare pentru modulul care folosește conexiune SQL

1.1. Inițializarea testelor

În contextul în care câmpurile “_sqlConnection” și “_command” sunt membri privați și aceștia nu pot fi inițializați normal, se accesează folosind o metodă de reflexie cunoscându-se existența lor. Această abordare a fost necesară pentru a putea crea conexiunea necesară folosindu-se un tipar de tipul “WhiteBox” în care codul de testat este cunoscut.

```
[TestInitialize]
public void TestInitialize()
{
    _notesRepository = new NotesRepository();
    // Acceseaza si modifica campurile private folosind
    System.Reflection
    var connectionField =
    typeof(NotesRepository).GetField("_sqlConnection",
    BindingFlags.Instance | BindingFlags.NonPublic);
    var commandField = typeof(NotesRepository).GetField("_command",
    BindingFlags.Instance | BindingFlags.NonPublic);

    // Creeaza o conexiune SQL si o atribuie campului
    _sqlConnection din clasa de testare
    var newConnection = new SQLiteConnection("Data
    Source=your_database_file.db");
```

```

        connectionField.SetValue(_notesRepository,
newConnection);

        // Creeaza o comanda SQL si o atribuie campului _command
din clasa de testare
        var newCommand = newConnection.CreateCommand();
        commandField.SetValue(_notesRepository, newCommand);

    }

```

- 1.2. În vederea finalizării testelor, este necesară închiderea conexiunilor la baza de dată în caz că aceasta nu a fost terminată prin testele utilizate. Această metodă folosește o abordare de tip “WhiteBox” în care codul de testat este cunoscut, astfel, conexiunile pot fi terminate într-un mod corect.

```

[TestCleanup]
public void TestCleanup()
{
    var connectionField =
typeof(NotesRepository).GetField("_sqlConnection",
BindingFlags.Instance | BindingFlags.NonPublic);
    var connection =
(SQLiteConnection)connectionField.GetValue(_notesRepository);

    connection?.Dispose();
}

```

2. Testarea metodelor asincrone

În cadrul proiectului au fost utilizate task-uri pentru extragerea datelor folosind servicii API. Codul de mai jos evidențiază modul în care se pot testa astfel de metode, întrucât acestea sunt asincrone este nevoie de tratarea lor ca task-uri și folosind clasa Task și metodele Run și Wait pentru ca testele respective să poată fi executate conform modului în care metodele testate funcționează în cadrul proiectului.

```

[TestMethod()]
public void GetStringByCoord_ValidInput_CorrectResponse()
{
    // Date de intrare
    double latitude = 47.15;
    double longitude = 27.58;
    string date = "2023-05-25";

    // Functia de testat
    string response;
    // Implementat astfel intrucat functia de testat este un Task
asincron
    var task = Task.Run(() => _weather.GetStringByCoord(latitude,
longitude, date));
    task.Wait();

    response = task.Result;

    // Verificare
}

```

```
Assert.IsNotNull(response);  
Assert.IsFalse(string.IsNullOrEmpty(response));  
}
```