POLITEHNICA UNIVERSITY OF TIMIȘOARA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE
MASTER'S DEGREE PROGRAM: AUTOMOTIVE EMBEDDED SOFTWARE

Multimedia system for cars

PROFESSORS:

Dr. ing. Stoicu-Tivadar Vasile
Dr. ing. Crișan-Vida Mihaela Marcella
.

STUDENTS:

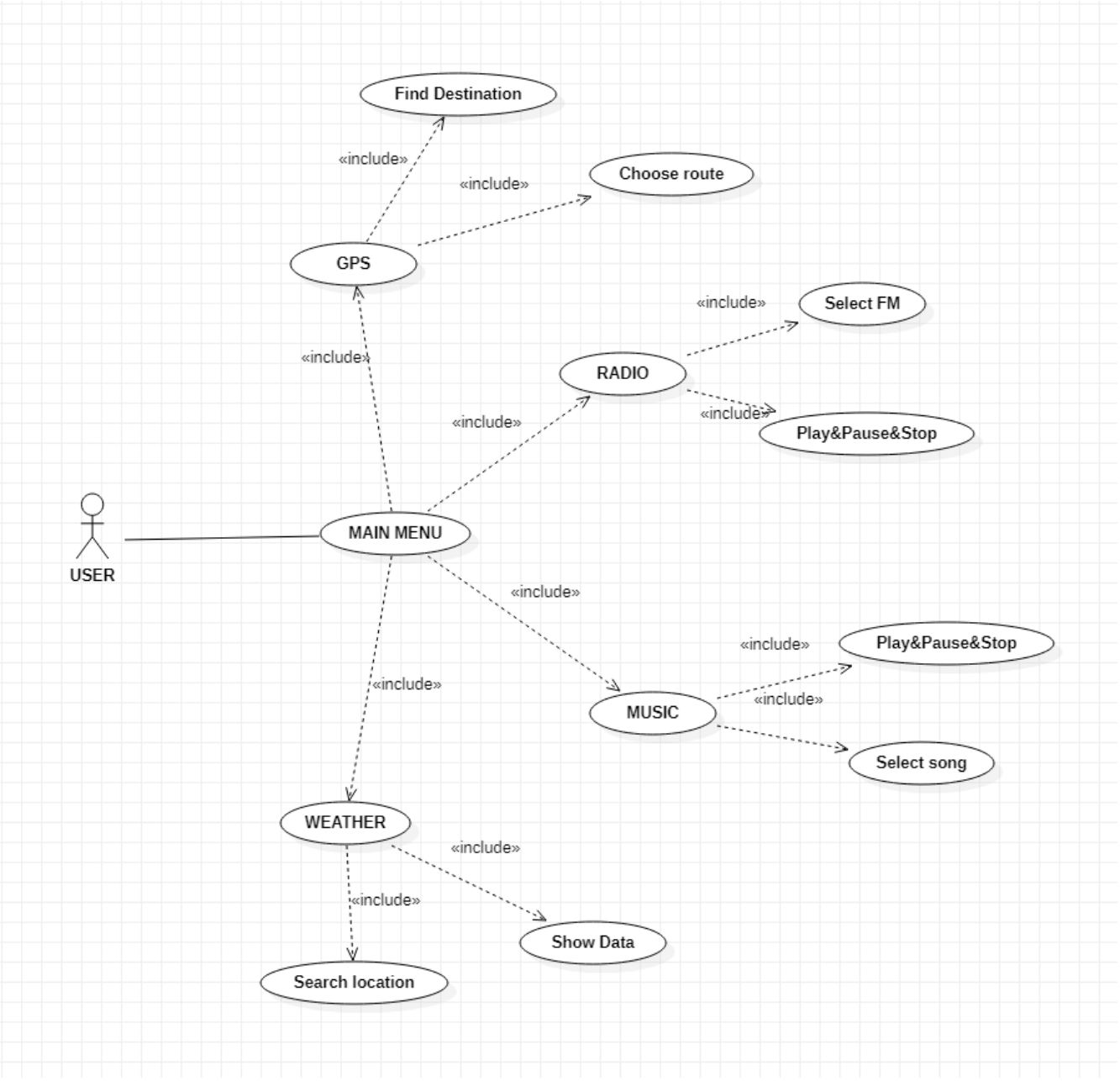
Deleu Anca
Igna Dianora
Nica Andrei

TIMIȘOARA
2021

Contents

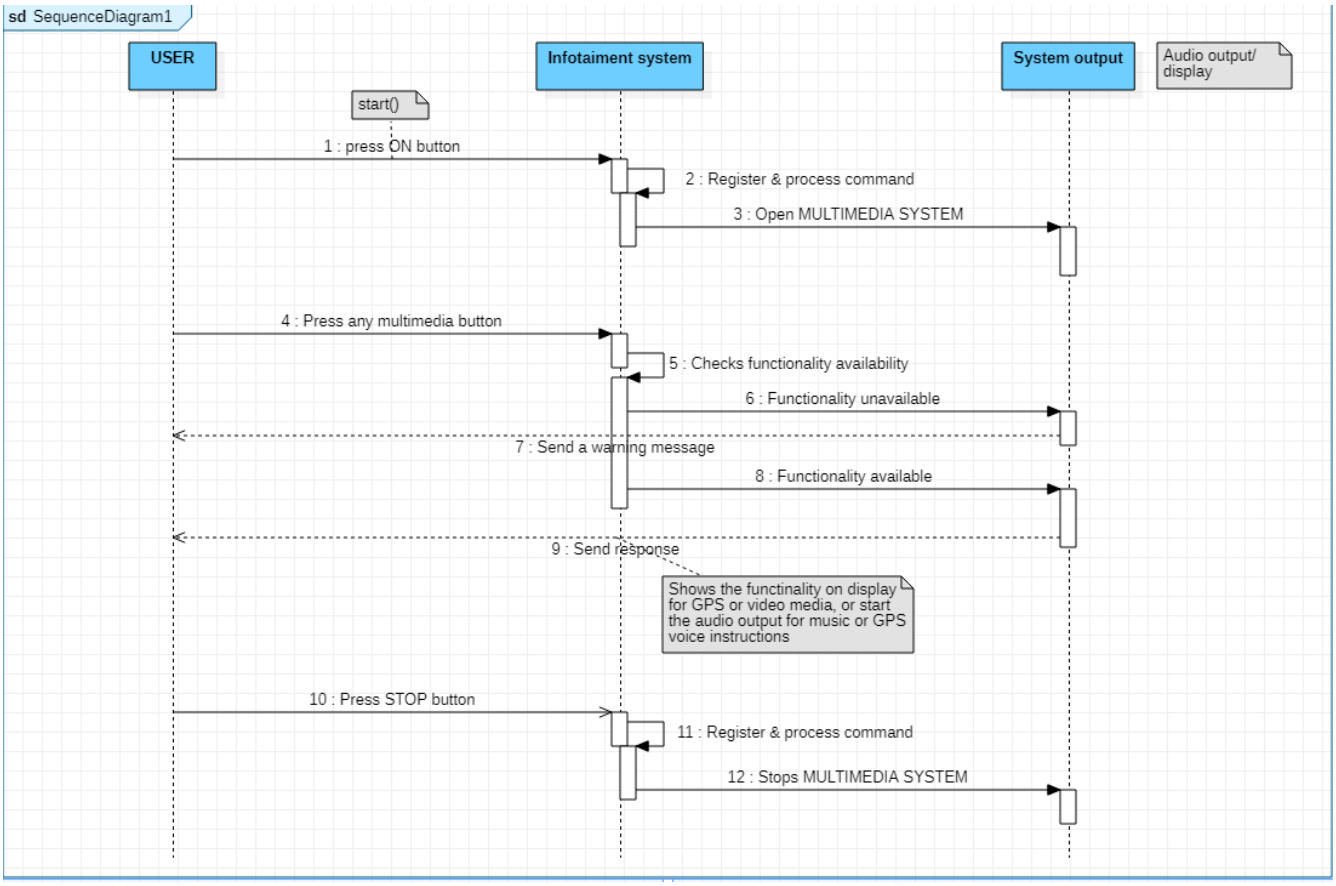
Chapter 1

Use Case Reports



Chapter 2

UML - Sequence Diagrams Reports



Interactions	Description
Interaction 1	User press ON/START button which sends a messange to the infotainment system
Interaction 2	The message send is register and precess, if everything is ok, infotainment system send a command to open Multimedia system
Interaction 3	When the command is send, the multimedia meniu will be shown on display
Interaction 4	Now, the user can choose any multimedia feature and press the button for that specify functionality
Interaction 5	The infotainment system will check if that functionality is available
Interaction 6	If the functionality is available, will show it on display and send a respons to the user, respons that could be visual or audio, depends on the functinality
Interaction 7	If the functionality is unavailable, will show an warning message on the display
Interaction 8	The user press STOP button which sends a messange to the infotainment system
Interaction 9	The message send is register and precess, if everything is ok, infotainment system will shut down and close the display.

Chapter 3

UML - Design Pattern

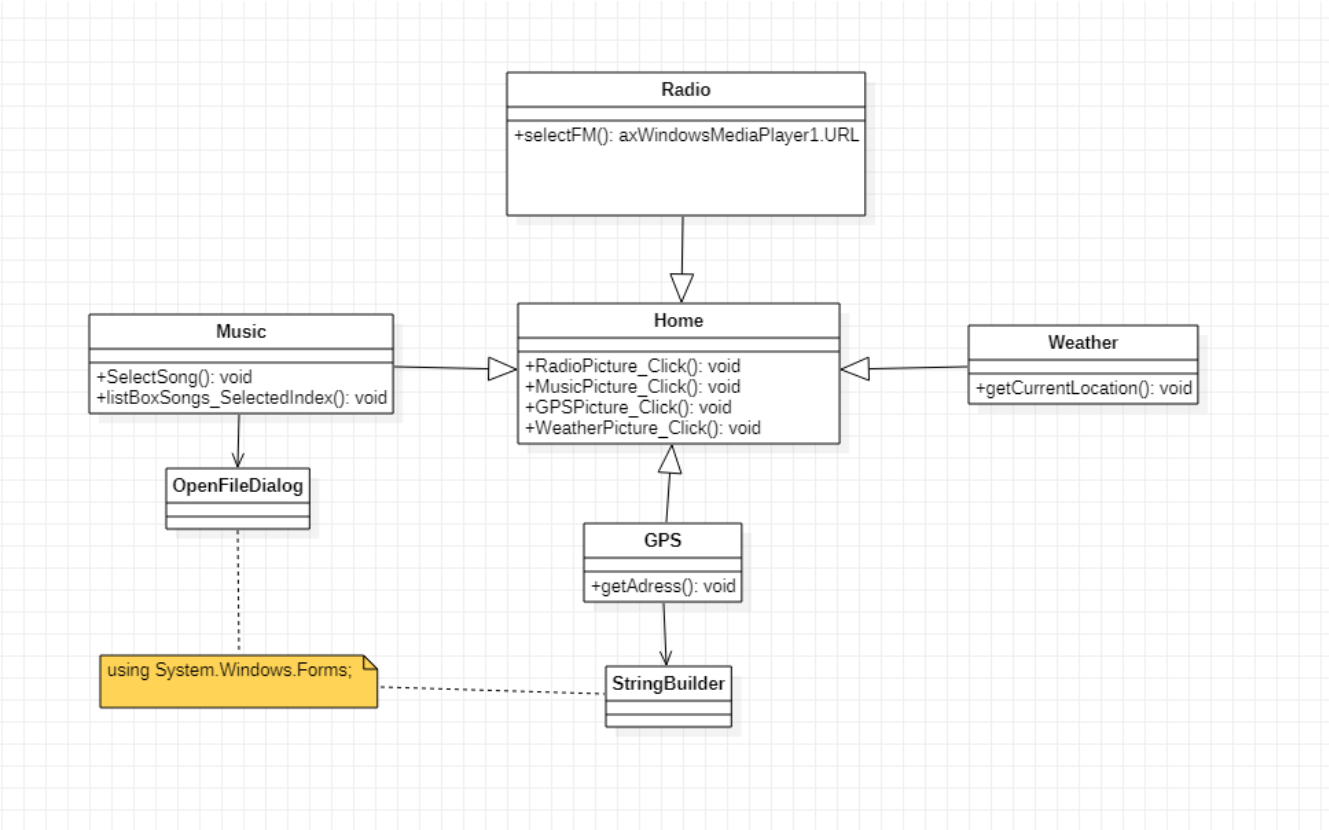
Chosen Design Pattern is **BUILD PATTERN**. We choose this pattern because we structure our project in individual classes, using C programing language and we find this pattern easier to implement for our project.

The more complex an application is the complexity of classes and objects used increases. Complex objects are made of parts produced by other objects that need special care when being built. An application might need a mechanism for building complex objects that is independent from the ones that make up the object. If this is the problem you are being confronted with, you might want to try using the Builder (or Adaptive Builder) design pattern.

This pattern allows a client object to construct a complex object by specifying only its type and content, being shielded from the details related to the objects representation. This way the construction process can be used to create different representations. The logic of this process is isolated form the actual steps used in creating the complex object, so the process can be used again to create a different object form the same set of simple objects as the first one.

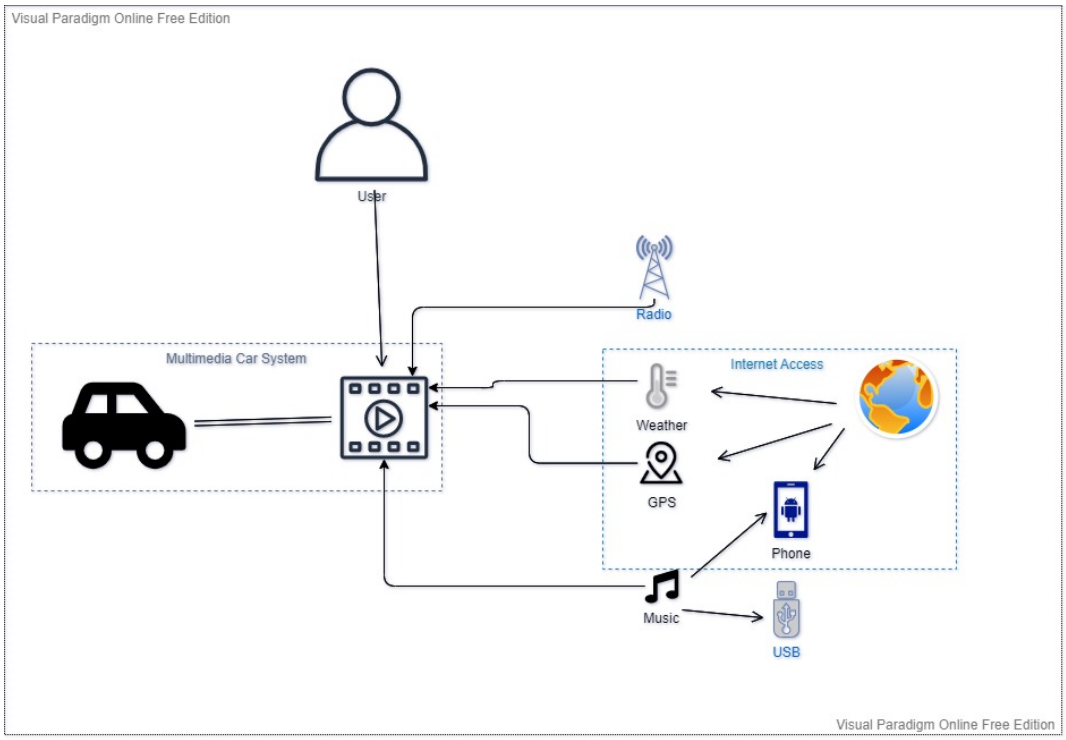
Chapter 4

UML - Class Diagram Report



Chapter 5

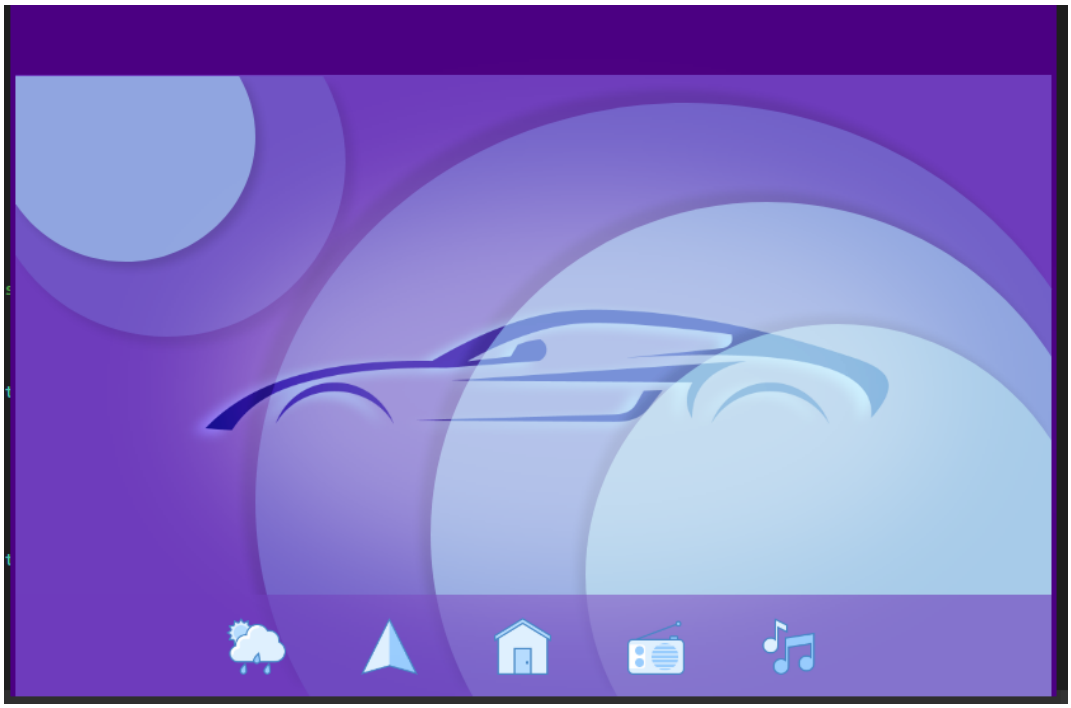
Application architecture



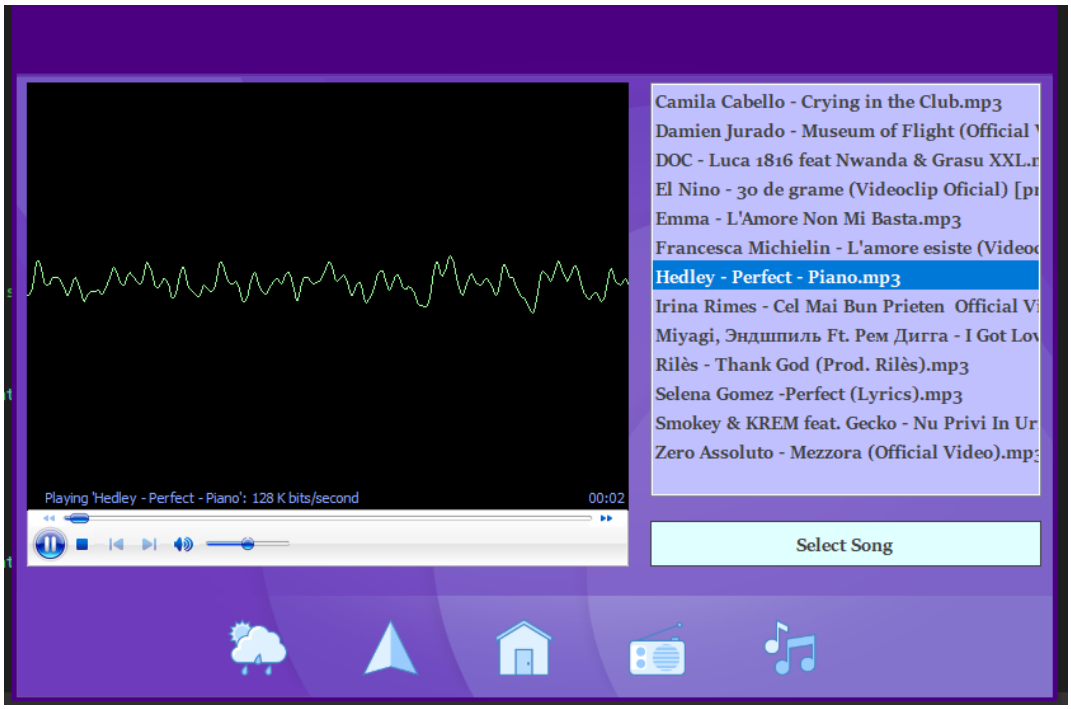
Weather Page

Chapter 6

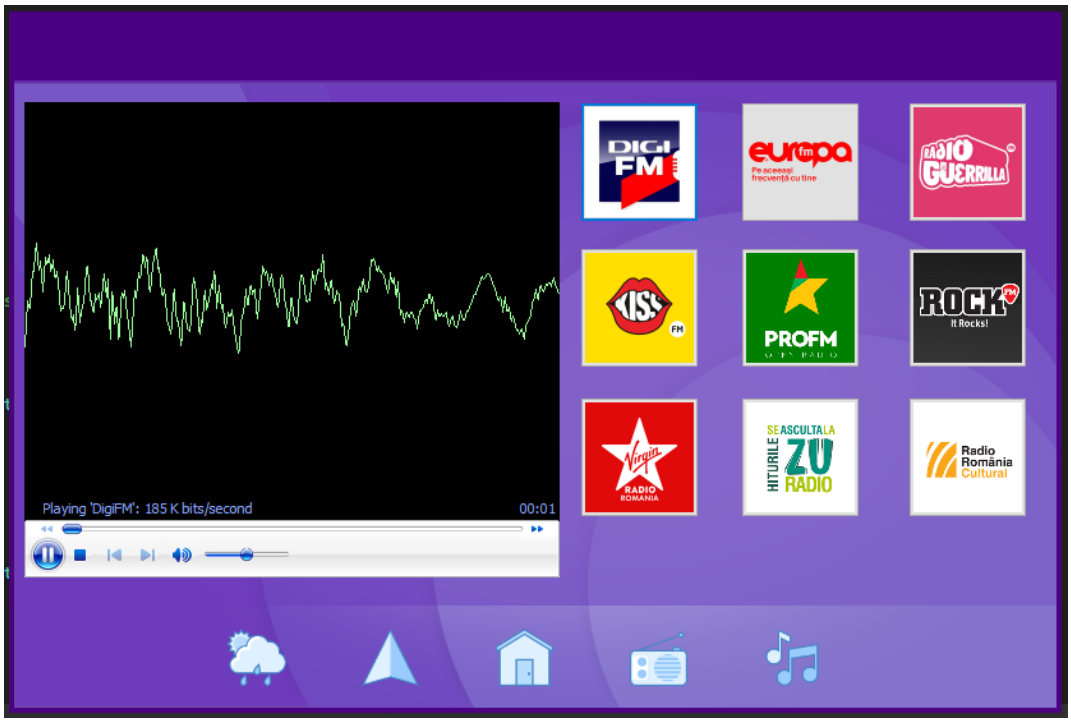
Application Design



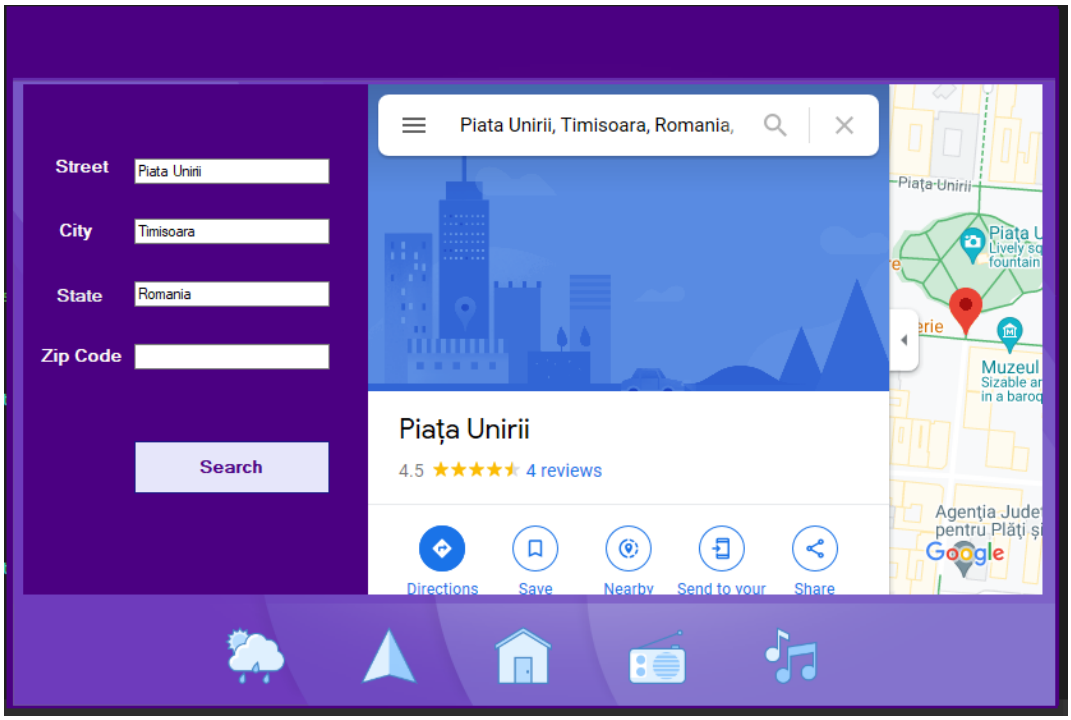
Home Page



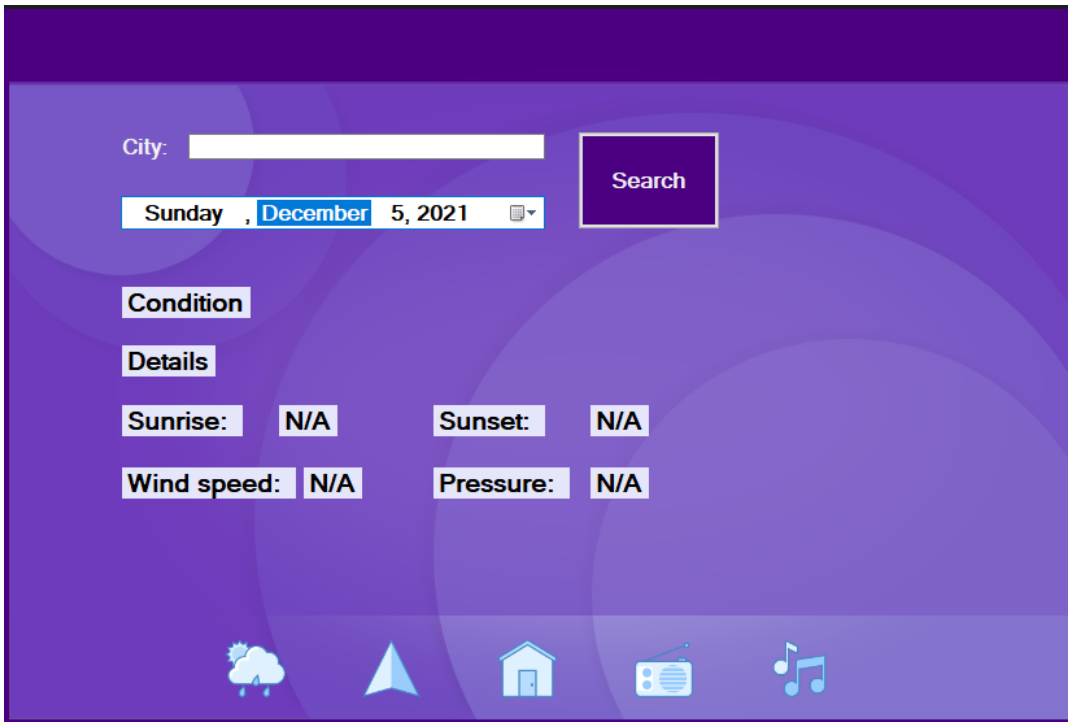
Music Page



Radio Page



GPS Page



Weather Page

Chapter 7

The generated code from class diagram

```
//HOME
public class Home {
    public Home() {
    }
    public void RadioPicture_Click() {
        // TODO implement here
        return null;
    }
    public void MusicPicture_Click() {
        // TODO implement here
        return null;
    }
    public void GPSPicture_Click() {
        // TODO implement here
        return null;
    }
    public void WeatherPicture_Click() {
        // TODO implement here
        return null;
    }
}
//MUSIC
public class Music extends Home {

    public Music() {
    }
    public void SelectSong() {
        // TODO implement here
        return null;
    }
    public void listBoxSongs_SelectedIndex() {
        // TODO implement here
        return null;
    }
}
//RADIO
public class Radio extends Home {

    public Radio() {
    }
    public axWindowsMediaPlayer1.URL selectFM() {
        // TODO implement here
        return null;
    }
}
//GPS
public class GPS extends Home {
    public GPS() {
    }
```

```
50      public void getAdress() {
51          // TODO implement here
52          return null;
53      }
54  }
55  //WEATHER
56  public class Weather extends Home {
57
58      public Weather() {
59      }
60      public void getCurrentLocation() {
61          // TODO implement here
62          return null;
63      }
64  }
65
66  //StringBuilder
67  public class StringBuilder {
68
69      public StringBuilder() {
70      }
71
72  }
```