МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

ІКНІ
Кафедра **ПЗ**

КУРСОВА РОБОТА

на тему: “ Автомобільна дорога”
з дисципліни: “Об’єктно орієнтовне програмування”

Стедента групи ПЗ-24

спеціальності 6.121

“Інженерія програмного забезпечення”

Губика Артема

Керівник: доцент кафедри ПЗ,

к.т.н., доцент Коротєєва Т. О.

Національна шкала _____

Кількість балів __ Оцінка ECTS __

Члени комісії _____ _____

_____ _____

_____ _____

Львів – 2023

Зміст

1 Завдання до курсової роботи

Варіант 3: Створити клас Дорога

Автомобільна дорога | Тип (державна/регіональна/обласна/місцева) | Протяжність | Кількість смуг | Наявність пішохідної доріжки | Наявність розділювача посередині дороги

1. Впорядкувати дороги за протяжністю.
2. Знайти найкоротшу дорогу, де найбільша кількість смуг.
3. Знайти всі дороги, в яких наявні розділювачі посередині, кількість смуг >2 та згрупувати за типом.
4. Визначити типи автомобільних доріг, з найбільшою протяжністю та наявністю пішохідних доріжок.
5. Визначити автомобільні дороги з найбільшою кількістю смуг та наявними пішохідними доріжками які належать до регіональних.

Для класу створити:

1. Конструктор за замовчуванням;
2. Конструктор з параметрами;
3. конструктор копій;
4. перевизначити операції », « для зчитування та запису у файл. Для демонстрації роботи програми використати засоби візуального середовища програмування.

№ з/п	Зміст завдання	Дата
1	Здійснити аналітичний огляд літератури за заданою темою та обгрунтувати вибір інструментальних засобів реалізації.	08.10
2	Побудова UML діаграм	08.10
3	Розробка алгоритмів реалізації	11.10
4	Реалізація завдання (кодування)	21.10
5	Формування інструкції користувача	25.10
6	Оформлення звіту до курсової роботи згідно з вимогами Міжнародних стандартів, дотримуючись такої структури: • Оформлення звіту до курсової роботи згідно з вимогами Міжнародних стандартів, дотримуючись такої структури: • зміст; • алгоритм розв‘язку задачі у покроковому представленні; • діаграми UML класів, прецедентів, послідовності виконання; • код розробленої програми з коментарями; • протокол роботи програми для кожного пункту завдання • інструкція користувача та системні вимоги; • опис виняткових ситуацій; • структура файлу вхідних даних; • висновки; • список використаних джерел.	05.11

Завдання прийнято до виконання: _____ (Губик А. С.)
(підпис студента)

Керівник роботи: _____ (Коротєєва Т. О.)
Дата видачі завдання: 20.09.2023

2 Алгоритм розв‘язку задачі у покроковому представленні

2.1 Сортування доріг за протяжністю (Sort by Length – SBL)

SBL1 Порівняння елементів і обмін:

- SBL1.1 Порівняємо перший елемент з наступним.
- SBL1.2 Якщо довжина першої дороги більша за довжину наступної, міняємо їх місцями.
- SBL1.3 Перше порівняння: (Roads[0] > Roads[1])

SBL2 Перший прохід по списку:

- SBL2.1 Продовжуємо порівнювати та міняти місцями сусідні елементи.
- SBL2.2 Довжина доріг порівнюється і міняється за необхідності.
- SBL2.3 Кінець першого проходу: найбільший елемент (за довжиною) розміщується в кінці списку.

SBL3 Повторення ітерацій:

- SBL3.1 Процес повторюється для усього списку, крім вже відсортованих елементів.
- SBL3.2 З кожною ітерацією найбільший (за довжиною) елемент "спливає"до кінця списку.

SBL4 Фінальна ітерація:

- SBL4.1 Повний прохід відбувається до тих пір, поки всі елементи не будуть відсортовані за зростанням довжини.
- SBL4.2 Кінцевий результат: список доріг, відсортований за зростанням довжини.

2.2 Знайти найкоротшу дорогу, де найбільша кількість смуг. (Shortes road most lanes – SRML)

SRML1 Очистка списку RequestedRoads:

- SRML1.1 Починаємо з очищення списку RequestedRoads, щоб гарантувати, що він буде порожнім перед початком операцій.

SRML2 Перевірка умови наявності доріг:

SRML2.1 Перевірка наявності доріг в колекції Roads або її порожності.

SRML2.2 Якщо Roads не існує або він порожній, функція завершується і повертає порожній список RequestedRoads.

SRML3 Пошук дороги з найменшою довжиною та найбільшою кількістю смуг:

SRML3.1 Ініціалізація змінних для зберігання найменшої довжини та найбільшої кількості смуг дороги.

SRML3.2 Перебір всіх доріг у колекції та знаходження дороги з найменшою довжиною.

SRML3.3 Якщо зустрічаємо дорогу із меншою довжиною, оновлюємо значення довжини та кількості смуг, а також вибираємо цю дорогу.

SRML4 Додавання вибраної дороги до RequestedRoads:

SRML4.1 Додаємо вибрану дорогу із найменшою довжиною та, можливо, найбільшою кількістю смуг до списку RequestedRoads.

SRML5 Зміна візуальної видимості (SecondTaskVisible):

SRML5.1 Змінюємо властивість SecondTaskVisible для відображення або приховання пов'язаних з цим елементів інтерфейсу користувача.

2.3 Знайти всі дороги, в яких наявні розділювачі посередині, кількість смуг >2 та згрупувати за типом. (A Complex One – ACO)

ACO5 Очищення списку RequestedRoads:

ACO5.1 Починаємо з очищення списку RequestedRoads, щоб гарантувати, що він буде порожнім перед виконанням нових операцій.

ACO5 Перевірка наявності доріг:

ACO5.1 Перевірка, чи існують дороги в колекції Roads або чи вона порожня.

ACO5.1 Якщо Roads не існує або вона порожня, функція завершується, повертаючи порожній список RequestedRoads.

ACO5 Пошук доріг із певними умовами:

ACO5.1 Перебираємо всі дороги в колекції.

ACO5.1 Якщо зустрічаємо дорогу, де кількість смуг більша за 2 і є лінія, додаємо цю дорогу до списку RequestedRoads.

ACO5 Зміна візуальної видимості (SecondTaskVisible):

ACO5.1 Змінюємо властивість SecondTaskVisible для відображення або приховання пов'язаних з цим елементів інтерфейсу користувача.

2.4 Визначити тип автомобільних доріг, з найбільшою протяжністю та наявністю пішохідних доріжок. (Heighest length and pavement – HLP)

HLP1 Перевірка наявності доріг:

HLP1.1 Перевірка, чи Roads не порожній і не має значення null.

HLP1.2 Якщо Roads не існує або вона порожня, функція завершується, не виконуючи подальших дій.

HLP2 Пошук типу доріг з певними умовами:

HLP2.1 Визначаємо початкові значення: тип selectedType встановлено як RoadType.National (припускаючи значення за замовчуванням).

HLP2.2 Шукаємо серед всіх доріг ті, у яких є покриття (HasPavement) і довжина більша за maxLength.

HLP2.3 Якщо знаходимо дорогу, що відповідає умовам, оновлюємо selectedType на тип цієї дороги.

HLP3 Виведення результату в консоль:

HLP3.1 Після завершення пошуку виводимо у консоль тип доріг із найбільшою довжиною та наявним покриттям (HasPavement).

2.5 Визначити автомобільні дороги з найбільшою кількістю смуг та наявними пішохідними доріжками які належать до регіональних.(Regional most lane and pavement – RMLP)

RMLP1 Очищення списку RequestedRoads:

RMLP1.1 Починаємо з очищення списку RequestedRoads, щоб гарантувати його порожність перед подальшою роботою.

RMLP2 Перевірка наявності доріг:

RMLP2.1 Перевіряємо, чи Roads не порожній і не є null. Якщо це так, функція завершується, не виконуючи подальших дій.

RMLP3 Пошук доріг з певними умовами:

RMLP3.1 Шукаємо серед всіх доріг ті, які мають більше двох смуг та наявну пішохідну доріжку (pavement).

RMLP3.2 Додаємо ці дороги до списку RequestedRoads.

RMLP4 Зміна візуальної видимості SecondTaskVisible:

RMLP4.1 Змінюємо властивість SecondTaskVisible для відображення або приховання елементів інтерфейсу, що пов'язані з наступною дією користувача.

3 Діаграми UML класів, прецедентів, послідовності виконання.

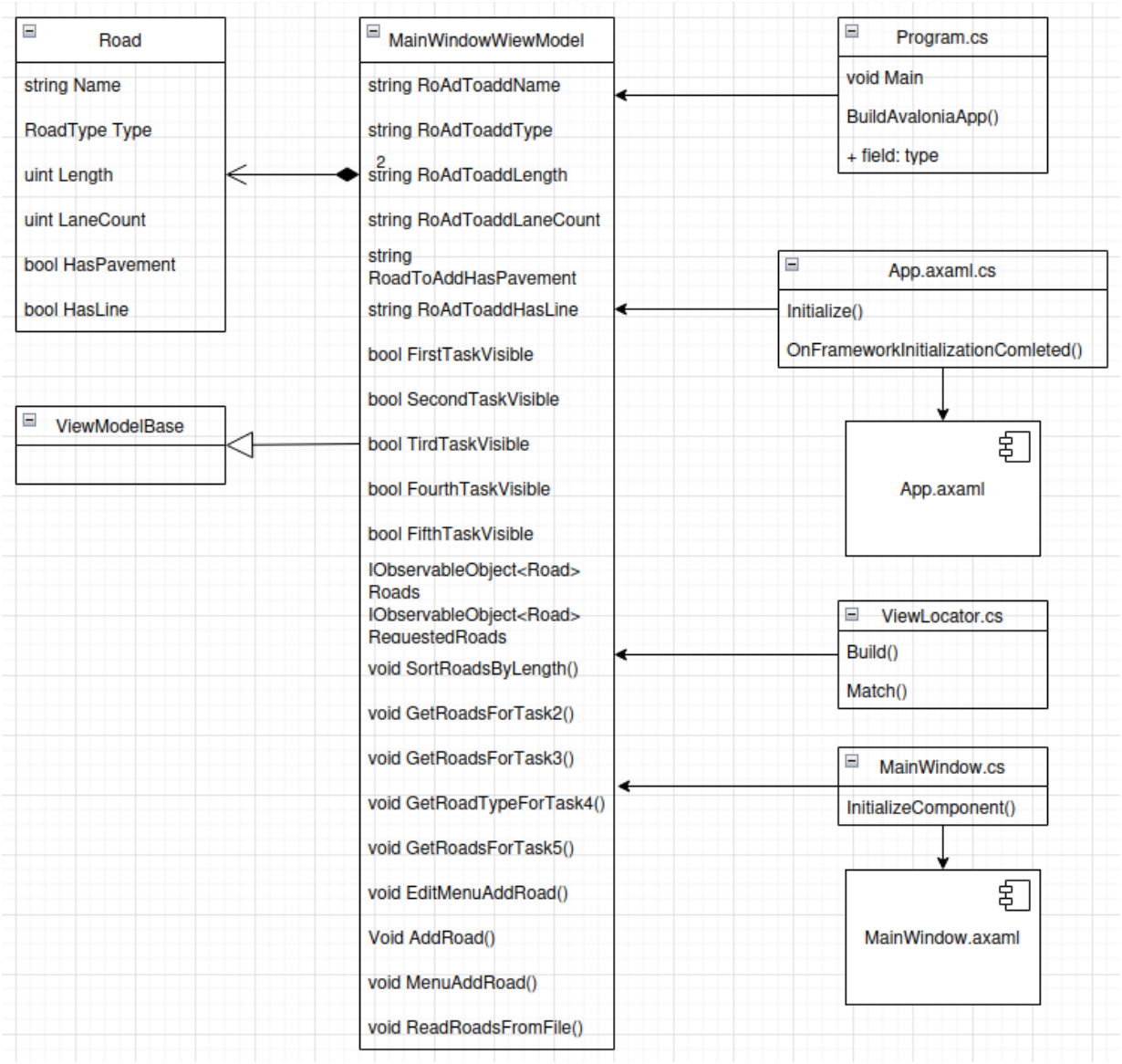


Рис. 1. UML діаграма класів

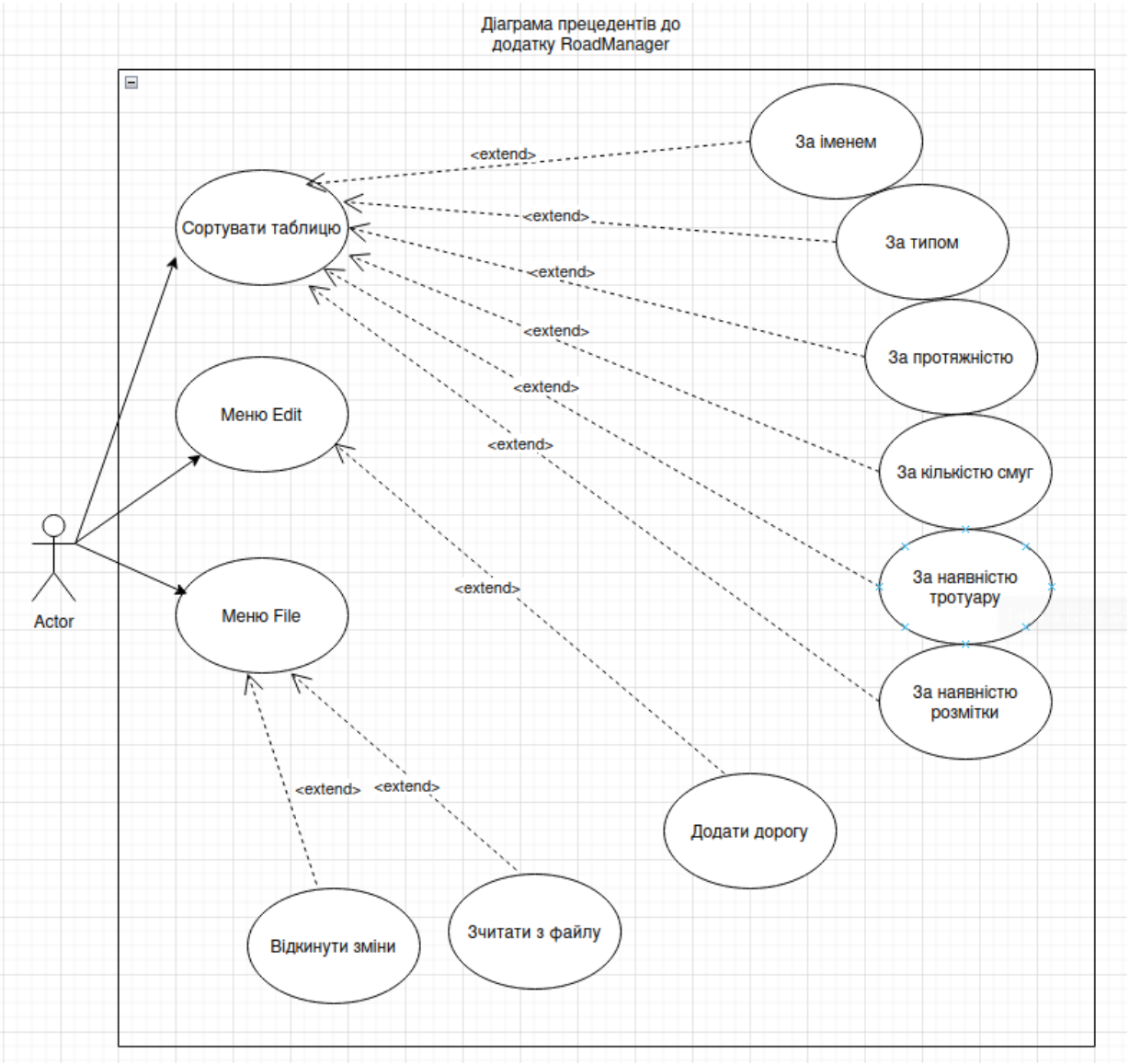
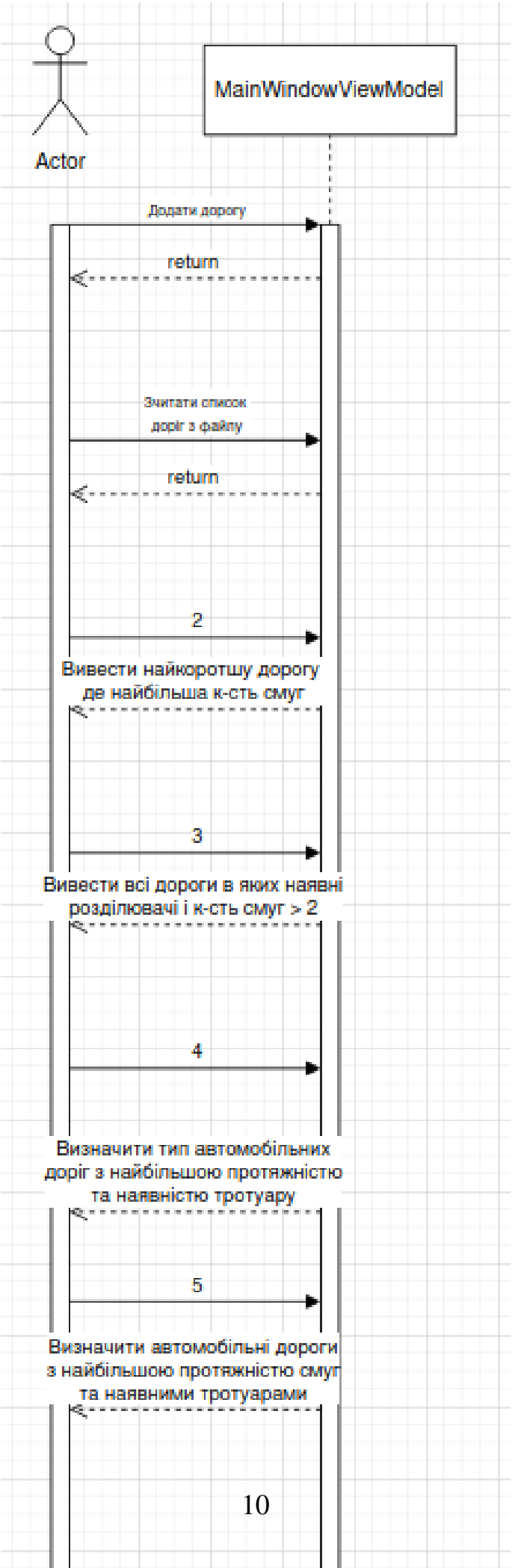


Рис. 2. Діаграма прецедентів


Actor
MainWindowViewModel
Додати дорогу
return
Зчитати список
доріг з файлу
return
2
Вивести найкоротшу дорогу
де найбільша к-сть смуг
3
Вивести всі дороги в яких наявні
розділювачі і к-сть смуг > 2
4
Визначити тип автомобільних
доріг з найбільшою протяжністю
та наявністю тротуару
5
Визначити автомобільні дороги
з найбільшою протяжністю смуг
та наявними тротуарами

4 Код розробленої програми з коментарями

4.1 Road.cs:

```
namespace RoadClass
{
    public enum RoadType
    {
        National,
        Regional,
        Oblast,
        Local

    }
    public class Road
    {
        public string? Name{get; set;} = "Noname";

        public RoadType Type{get; set;} = RoadType.Local;
        public uint Length{get; set;}  = 1;
        public uint LaneCount{get; set;} = 1;
        public bool HasPavement{get; set;} = false;
        public bool HasLine{get; set;} = false;

        public Road(string name, RoadType type, uint length, uint lane
            Name = name;
            Type = type;
            Length = length;
            LaneCount = laneCount;
            HasPavement = hasPavement;
            HasLine = hasLine;
        }
        public Road(Road other){
            Name = other.Name;
            Type = other.Type;
            Length = other.Length;
            LaneCount = other.LaneCount;
            HasPavement = other.HasPavement;
            HasLine = other.HasLine;
        }


        public Road(){}
```

```
        }
    }
```

4.2 MainWindow.cs:

```
using Avalonia.Controls;
using RoadManager.ViewModels;

namespace RoadManager.Views;

public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        DataContext = new MainWindowViewModel();

    }
}
```

4.3 MainWindow.axaml:

```
<Window xmlns="https://github.com/avaloniaui"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"
x:Class="KurSUCH.MainWindow"
Title="KurSUCH"
Background="#dedede">

<Grid RowDefinitions="Auto, Auto, *">
    <Grid Grid.Row="0" Background="White" Height="30" ColumnDefinitions="A
        <Button Grid.Column="0" Content="File" Foreground="Black"/>
        <Button Grid.Column="1" Content="Edit" Foreground="Black"/>
        <Button Grid.Column="2" Content="Help" Foreground="Black"/>
        <Rectangle></Rectangle>
    </Grid>
    <Grid Grid.Row="1" Background="#dedede" Height="50" ColumnDefinitions=
        <Button Grid.Column="0">
            <Image Source="/Assets/plus.png" />
        </Button>
```

```
        <Button Grid.Column="1">
            <Image Source="/Assets/error.png" />
        </Button>
        <Button Grid.Column="2">
            <Image Source="/Assets/error.png" />
        </Button>
        <Rectangle Grid.Column="3" Fill="Black" />
    </Grid>
    <Grid ColumnDefinitions="*, Auto">
        <DataGrid Grid.Column="0" ItemsSource="Binding Roads">
                <DataGrid.Columns>
                <DataGridTextColumn Header="Name"
                                    Binding="{Binding Name}"
                                    Width="2*" />
                <DataGridTextColumn Header="Last Name"
                                    Binding="{Binding Type}"
                                    Width="*" />
                <DataGridTextColumn Header="Department"
                                    Binding="{Binding Length}"
                                    Width="*" />
                <DataGridTextColumn Header="Department"
                                    Binding="{Binding LaneCount}"
                                    Width="*" />
                <DataGridTextColumn Header="Department"
                                    Binding="{Binding HasPavement}"
                                    Width="*" />
                <DataGridTextColumn Header="Department"
                                    Binding="{Binding HasLine}"
                                    Width="*" />
                </DataGrid.Columns>
        </DataGrid>
    </Grid>
</Grid>

</Window>
```

4.4 MainWindowViewModel.cs:

```
using System;
using System.IO;
using System.Linq;
using System.Collections.Generic;
```

```
using System.Collections.ObjectModel;
using System.ComponentModel.DataAnnotations;

using System.Diagnostics;
using System.Reactive;
using System.Text;
using ReactiveUI;
using RoadClass;
using Avalonia;
using Avalonia.Controls;
using Avalonia.Markup.Xaml;
using RoadManager.Views;
namespace RoadManager.ViewModels;

public class MainWindowViewModel : ViewModelBase
{
    private string _RoadToAddName = "Name";
    private uint _RoadToAddTypeIndex;
    private RoadType _RoadToAddType;
    private uint _RoadToAddLength;
    private uint _RoadToAddLaneCount;
    private bool _RoadToAddHasPavement;
    private bool _RoadToAddHasLine;
    private string _MenuOpenFileText = "in.txt";

    [Required]
    public string MenuOpenFileText
    {
        get => _MenuOpenFileText;
        set => this.RaiseAndSetIfChanged(ref _MenuOpenFileText, value);
    }
    //private string _RoadToAddName;

    public uint RoadToAddTypeIndex
    {
        get => _RoadToAddTypeIndex;
        set => this.RaiseAndSetIfChanged(ref _RoadToAddTypeIndex, value);
    }
    public uint RoadToAddLaneCount
    {
        get => _RoadToAddLaneCount;
        set => this.RaiseAndSetIfChanged(ref _RoadToAddLaneCount, value);
    }
```

```
public uint RoadToAddLength
{
    get => _RoadToAddLength;
    set => this.RaiseAndSetIfChanged(ref _RoadToAddLength, value);
}
[Required]
public string RoadToAddName
{
    get => _RoadToAddName;
    set
    {
    if(value == "")
    {
        //throw new ArgumentNullException("", "This field is required"
    }
    this.RaiseAndSetIfChanged(ref _RoadToAddName, value);
    }
}
public bool RoadToAddHasPavement
{
    get => _RoadToAddHasPavement;
    set => this.RaiseAndSetIfChanged(ref _RoadToAddHasPavement,
                                     _RoadToAddHasPavement ^= true);
}
public bool RoadToAddHasLine
{
    get => _RoadToAddHasLine;
    set => this.RaiseAndSetIfChanged(ref _RoadToAddHasLine,
                                     _RoadToAddHasLine ^= true);
}

//private bool[] ItemVisibility = new bool[10];

public ObservableCollection<bool> ItemVisibility { get; } = new Observ


public bool AddRoadVisible
{
    get => ItemVisibility[0];
    set
    {
        for(int i = 0; i < ItemVisibility.Count; i++){
            ItemVisibility[i] = false;
```

```
        }
        ItemVisibility[0] = true;
    }
}
public bool SecondTaskVisible
{
    get => ItemVisibility[1];
    set
    {
        for(int i = 0; i < ItemVisibility.Count; i++){
            ItemVisibility[i] = false;
            //this.RaiseAndSetIfChanged(ref ItemVisibility[i], false);
        }
        ItemVisibility[1] = true;
        //this.RaiseAndSetIfChanged(ref ItemVisibility[1], true);

    }
}

public bool ThirdTaskVisible
{
    get => ItemVisibility[2];
    set
    {
        for(int i = 0; i < ItemVisibility.Count; i++){
            ItemVisibility[i] = false;
            //this.RaiseAndSetIfChanged(ref ItemVisibility[i], false);
        }
        ItemVisibility[2] = true;
        //this.RaiseAndSetIfChanged(ref ItemVisibility[1], true);

    }
}
public bool FourthTaskVisible
{
    get => ItemVisibility[3];
    set
    {
        for(int i = 0; i < ItemVisibility.Count; i++){
            ItemVisibility[i] = false;
            //this.RaiseAndSetIfChanged(ref ItemVisibility[i], false);
        }
        ItemVisibility[3] = true;
```

```
        //this.RaiseAndSetIfChanged(ref ItemVisibility[1], true);

    }
}
public bool FifthTaskVisible
{
    get => ItemVisibility[4];
    set
    {
        for(int i = 0; i < ItemVisibility.Count; i++){
            ItemVisibility[i] = false;
            //this.RaiseAndSetIfChanged(ref ItemVisibility[i], false);
        }
        ItemVisibility[4] = true;
        //this.RaiseAndSetIfChanged(ref ItemVisibility[1], true);

    }
}
public void SetRoadToAddType()
{
    switch(RoadToAddTypeIndex){
    case 0:
        _RoadToAddType = RoadType.National;
        return;
    case 1:
        _RoadToAddType = RoadType.Regional;
        return;
    case 2:
        _RoadToAddType = RoadType.Oblast;
        return;
    case 3:
        _RoadToAddType = RoadType.Local;
        return;
    }
}
public ObservableCollection<Road> Roads { get; } = new();
public ObservableCollection<Road> RequestedRoads { get; } = new();


    public void EditMenuAddRoad()
    {
        AddRoadVisible = true;
    }
```

```
public MainWindowViewModel()
{

    //Roads = new ObservableCollection<Road>(ReadRoadsFromFile("in
}

public void AddRoad()
{
    SetRoadToAddType();
    Roads.Add(new()
    {
        Name = _RoadToAddName,
        Type = _RoadToAddType,
        Length = _RoadToAddLength,
        LaneCount = _RoadToAddLaneCount,
        HasLine = _RoadToAddHasLine,
        HasPavement = _RoadToAddHasPavement
    });
}

public void SortRoadsByLength()
{
    if (Roads == null || Roads.Count <= 1)
    {
        // Return if the input list has 0 or 1 elements
        return;
    }

    // Bubble Sort implementation for sorting Roads by Length
    int n = Roads.Count;
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            // Swap if the current road's Length is greater than t
            if (Roads[j].Length > Roads[j + 1].Length)
            {
                Road temp = Roads[j];
                Roads[j] = Roads[j + 1];
                Roads[j + 1] = temp;
            }
        }
```

```
    }
}

public void GetRoadsForTask2()
{
    RequestedRoads.Clear();
    if (Roads == null || Roads.Count == 0)
    {
        // Return an empty list if the input list is null or empty
        return;
    }

    uint smallestLength = uint.MaxValue;
    uint largestLaneCount = 0;

    // Find the smallest length and largest lane count
    foreach (var road in Roads)
    {
        smallestLength = Math.Min(smallestLength, road.Length);
        largestLaneCount = Math.Max(largestLaneCount, road.LaneCou
    }

    // Find Roads with the smallest length and largest lane count
    foreach (var road in Roads)
    {
        if (road.Length == smallestLength || road.LaneCount == lar
        {
            RequestedRoads.Add(road);
        }
    }
    SecondTaskVisible = true;

}
public void GetRoadsForTask3()
{
    RequestedRoads.Clear();

    if (Roads == null || Roads.Count == 0)
    {
        // Return an empty list if the input list is null or empty
        return;
    }
```

```
    // Find Roads with LaneCount > 2 and HasLine is true
    foreach (var road in Roads)
    {
        if (road.LaneCount > 2 && road.HasLine)
        {
            RequestedRoads.Add(road);
        }
    }

    ThirdTaskVisible = true;
}

public void GetRoadTypeForTask4()
{
    RequestedRoads.Clear();
    if (Roads == null || Roads.Count == 0)
    {
        // Return if the input list is null or empty
        return;
    }

    // Find type of roads with the biggest length and HasPavement
    Road tmp = null;
    RoadType selectedType = RoadType.National; // Assuming a defau
    uint maxLength = 0;

    foreach (var road in Roads)
    {
        if (road.HasPavement && road.Length > maxLength)
        {
            maxLength = road.Length;
            selectedType = road.Type;
            tmp = road;
        }
    }
    if (tmp != null)
    {
        RequestedRoads.Add(tmp);
    }

    FourthTaskVisible = true;
    // Now 'selectedType' contains the type of roads with the bigg
    Console.WriteLine($"Type of roads with the biggest length and
```

```
}

public void GetRoadsForTask5()
{
    RequestedRoads.Clear();

    if (Roads == null || Roads.Count == 0)
    {
        // Return an empty list if the input list is null or empty
        return;
    }

    // Find Roads with the biggest LaneCount, HasPavement is true,
    Road selectedRoad = null;

    foreach (var road in Roads)
    {
        if (road.Type == RoadType.Regional && road.HasPavement)
        {
            if (selectedRoad == null || road.LaneCount > selectedR
            {
                selectedRoad = road;
            }
        }
    }

    if (selectedRoad != null)
    {
        RequestedRoads.Add(selectedRoad);
    }

    FifthTaskVisible = true;
}

public void MenuOpenFile()
{
    Roads.Clear();
    //Roads = new ObservableCollection<Road>(ReadRoadsFromFile(Men
    ReadRoadsFromFile(Roads, MenuOpenFileText);

}
public void MenuResetTable()
{
```

```
    Roads.Clear();
    //Roads = new ObservableCollection<Road>(ReadRoadsFromFile(Men
    ReadRoadsFromFile(Roads, "in.txt");
}


void  ReadRoadsFromFile(ObservableCollection<Road> roads, string f
{
    //List<Road> roads = new List<Road>();

    try
    {
        // Read all lines from the file
        string[] lines = File.ReadAllLines(filePath);

        foreach (var line in lines)
        {
            // Split each line based on the delimiter
            string[] roadProperties = line.Split(delimiter);

            // Ensure the line has enough elements
            if (roadProperties.Length >= 6)
            {
                // Parse properties and create a Road object
                string name = roadProperties[0].Trim();
                RoadType type = Enum.Parse<RoadType>(roadPropertie
                uint length = uint.Parse(roadProperties[2].Trim())
                uint laneCount = uint.Parse(roadProperties[3].Trim
                bool hasPavement = bool.Parse(roadProperties[4].Tr
                bool hasLine = bool.Parse(roadProperties[5].Trim()

                Road road = new Road(name, type, length, laneCount
                roads.Add(road);
            }
            else
            {
                // Log or handle cases where the line doesn't have
                Console.WriteLine($"Invalid line: {line}");
            }
        }
    }
    catch (Exception ex)
    {
```

```
                // Handle exceptions (e.g., file not found, format issues,
                Console.WriteLine($"Error reading file: {ex.Message}");
                MenuOpenFileText = "Could not open file!";
                //throw new ArgumentNullException(nameof(MenuOpenFileText)
            }

            //return roads;
        }
}
```

4.5 ViewModelBase.cs:

```
using ReactiveUI;

namespace RoadManager.ViewModels;

public class ViewModelBase : ReactiveObject
{
}
```

4.6 Program.cs:

```
using Avalonia;
using Avalonia.ReactiveUI;
using System;

namespace KurSUCH;

class Program
{
    // Initialization code. Don't use any Avalonia, third-party APIs or an
    // SynchronizationContext-reliant code before AppMain is called: thing
    // yet and stuff might break.
    [STAThread]
    public static void Main(string[] args) => BuildAvaloniaApp()
        .StartWithClassicDesktopLifetime(args);

    // Avalonia configuration, don't remove; also used by visual designer.
    public static AppBuilder BuildAvaloniaApp()
```

```
        => AppBuilder.Configure<App>()
            .UsePlatformDetect()
            .WithInterFont()
            .LogToTrace()
            .UseReactiveUI();
}
```

4.7 ViewLocator.cs:

```
using System;
using Avalonia.Controls;
using Avalonia.Controls.Templates;
using KurSUCH.ViewModels;

namespace KurSUCH;

public class ViewLocator : IDataTemplate
{
    public Control Build(object data)
    {
        var name = data.GetType().FullName!.Replace("ViewModel", "View");
        var type = Type.GetType(name);

        if (type != null)
        {
            return (Control)Activator.CreateInstance(type)!;
        }

        return new TextBlock { Text = "Not Found: " + name };
    }

    public bool Match(object data)
    {
        return data is ViewModelBase;
    }
}
```

4.8 App.axaml.cs:

```
using Avalonia;
using Avalonia.Controls.ApplicationLifetimes;
using Avalonia.Markup.Xaml;
```

```
using RoadManager.ViewModels;
using RoadManager.Views;

namespace RoadManager;

public partial class App : Application
{
    public override void Initialize()
    {
        AvaloniaXamlLoader.Load(this);
    }

    public override void OnFrameworkInitializationCompleted()
    {
        if (ApplicationLifetime is IClassicDesktopStyleApplicationLifetime
        {
            desktop.MainWindow = new MainWindow
            {
                DataContext = new MainWindowViewModel(),
            };
        }

        base.OnFrameworkInitializationCompleted();
    }
}
```

4.9 App.axaml:

```
<Application xmlns="https://github.com/avaloniaui"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
x:Class="RoadManager.App"
xmlns:local="using:RoadManager"
RequestedThemeVariant="Light">
<!-- "Default" ThemeVariant follows system theme variant. "Dark" or "Light

<Application.DataTemplates>
<local:ViewLocator/>
</Application.DataTemplates>

<Application.Styles>
<FluentTheme>
   <FluentTheme.Palettes>
```

```
        <!-- Palette for Light theme variant -->
        <!-- Palette for Dark theme variant -->
        <ColorPaletteResources x:Key="Light"
                               Accent="Green"
                               RegionColor="White"
                               ErrorText="Red" />
    </FluentTheme.Palettes>
    <StyleInclude Source="avares://Avalonia.Controls.DataGrid/Themes/Fluent
</FluentTheme>
</Application.Styles>
</Application>
```

5 Протокол роботи програми для кожного пункту завдання

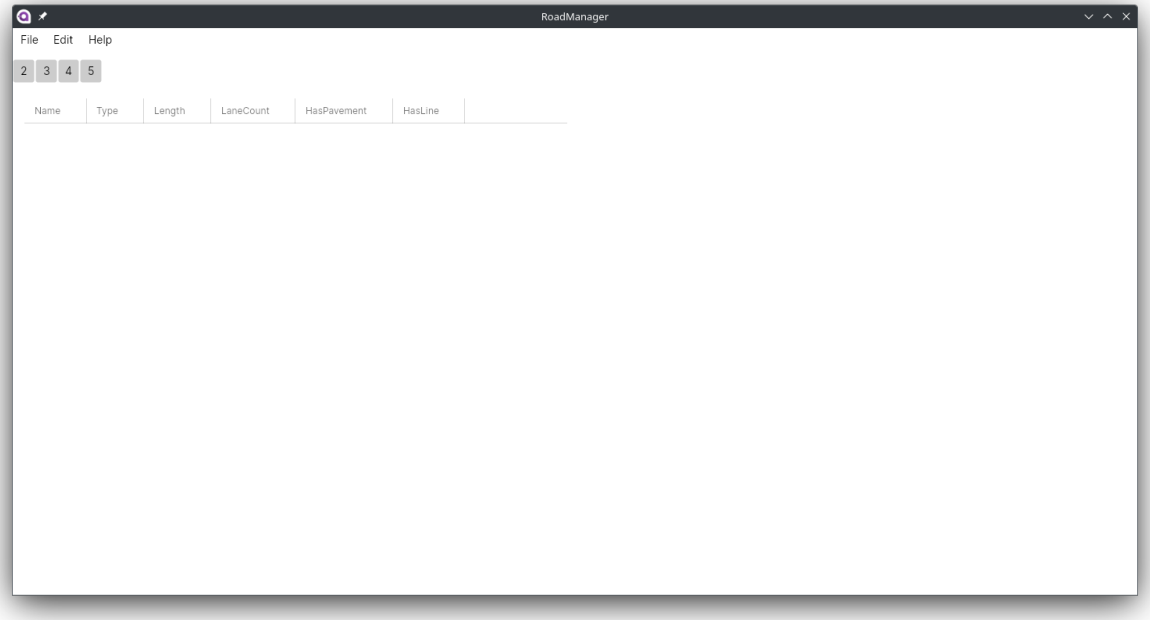


Рис. 4. Стартовий екран / Головне вікно

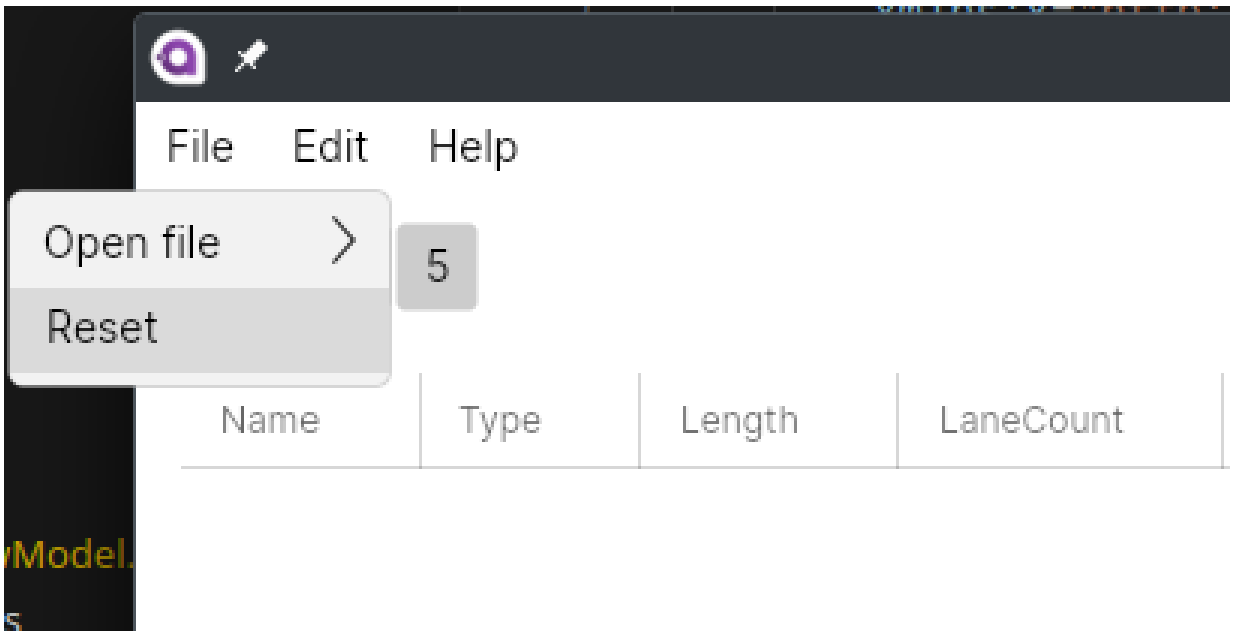


Рис. 5. Натискання на кнопку File

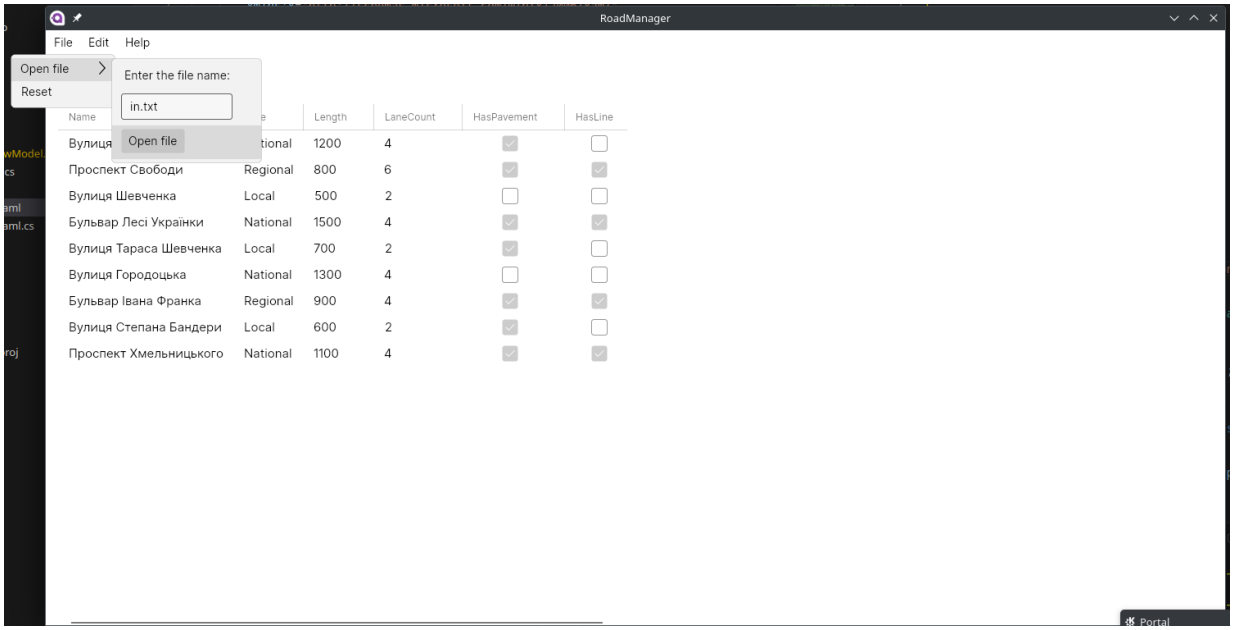


Рис. 6. Натискання на Open file

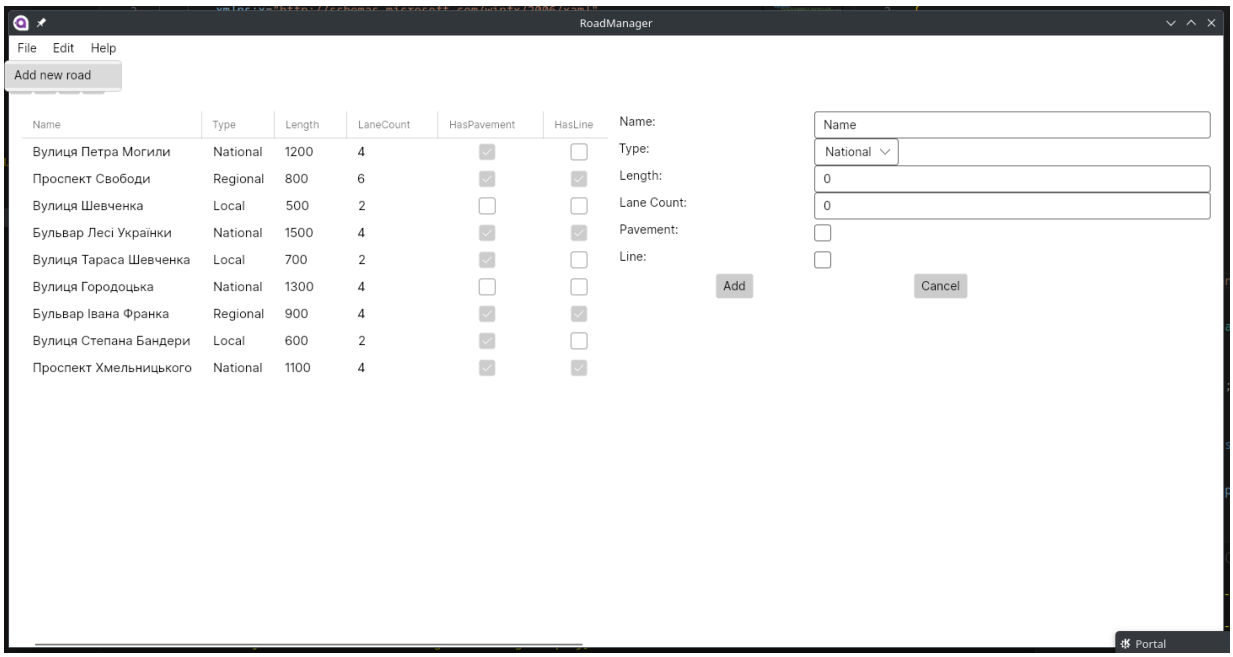


Рис. 7. Натискання на меню Edit і кнопку Add new road

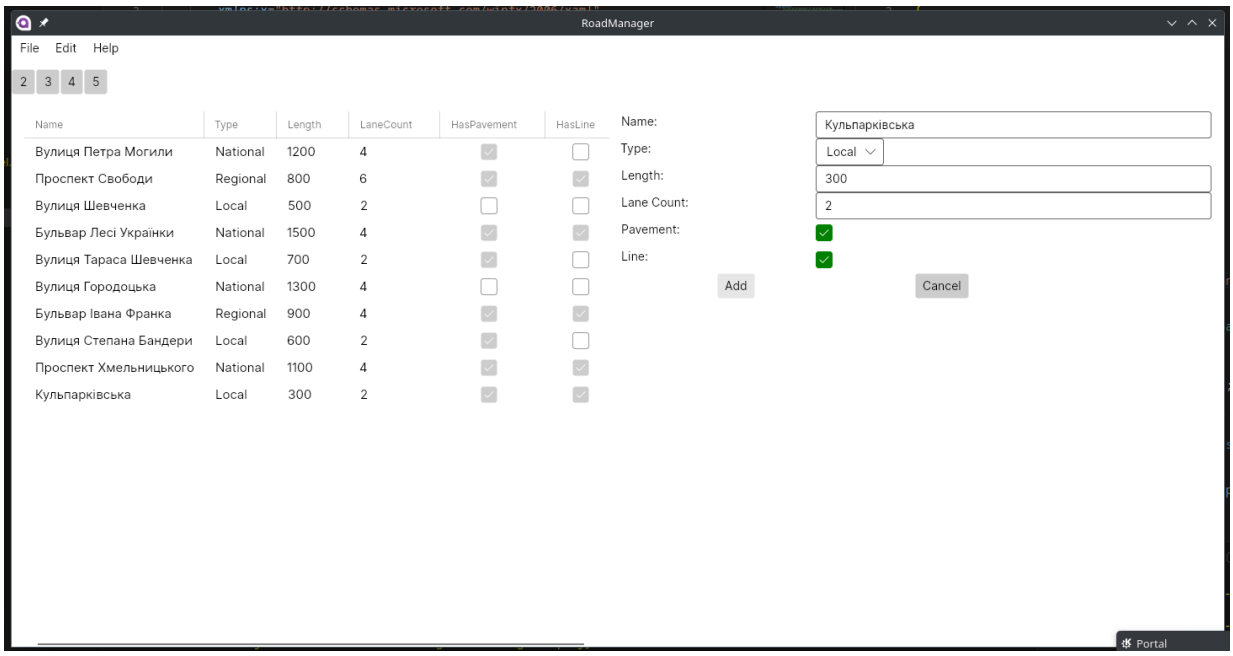


Рис. 8. Введення даних в поля зправа

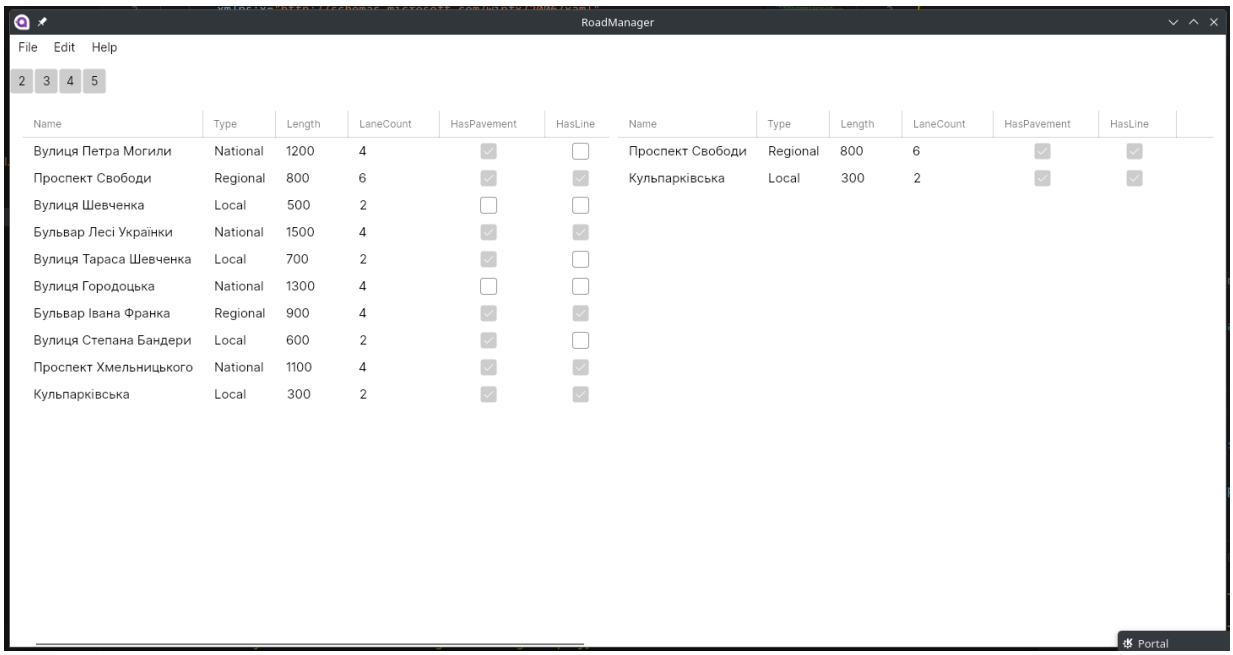


Рис. 9. Натискання на кнопку 2

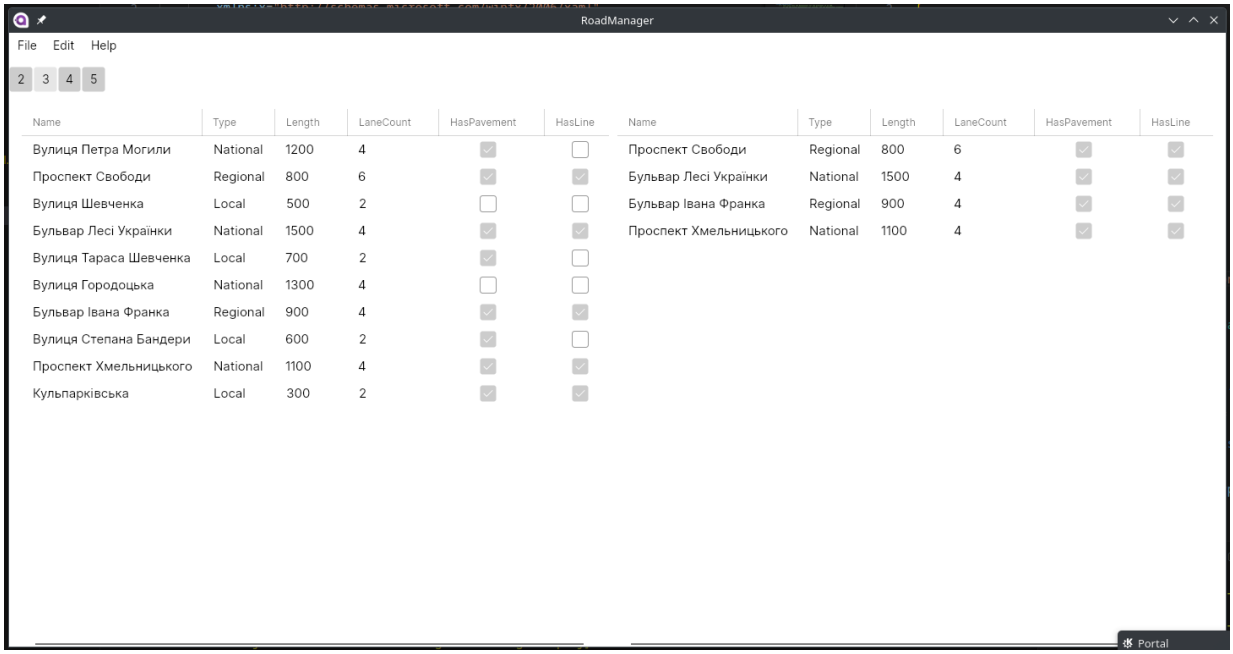


Рис. 10. Натискання на кнопку 3

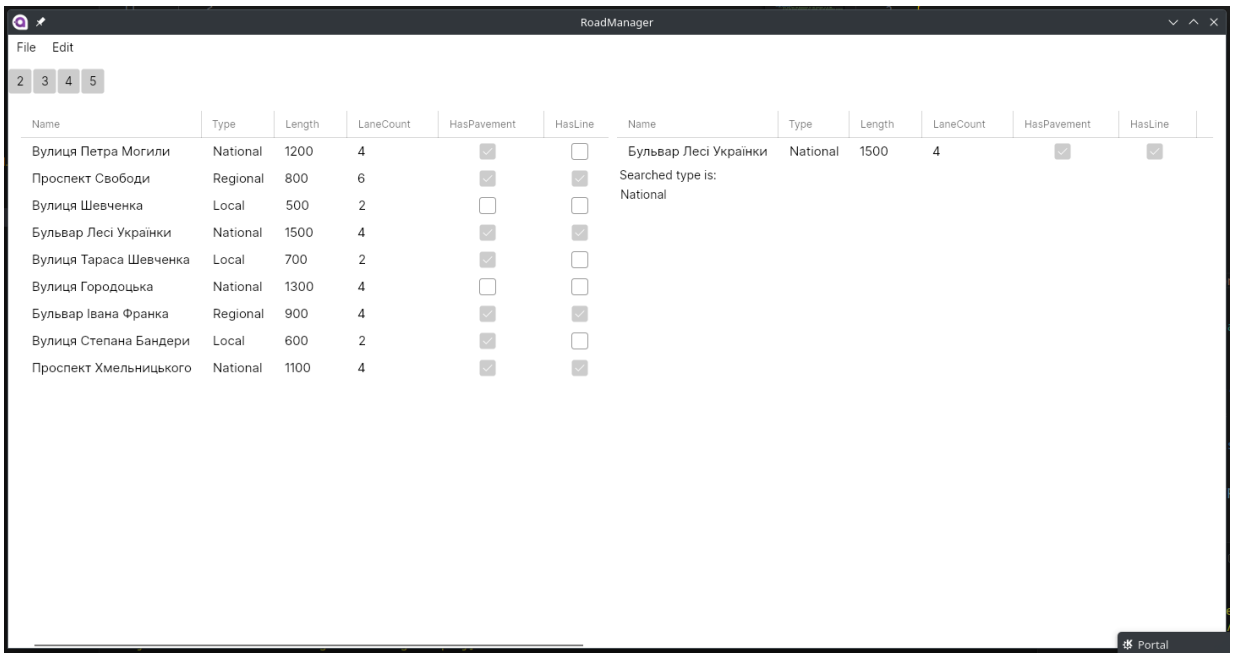


Рис. 11. Натискання на кнопку 4

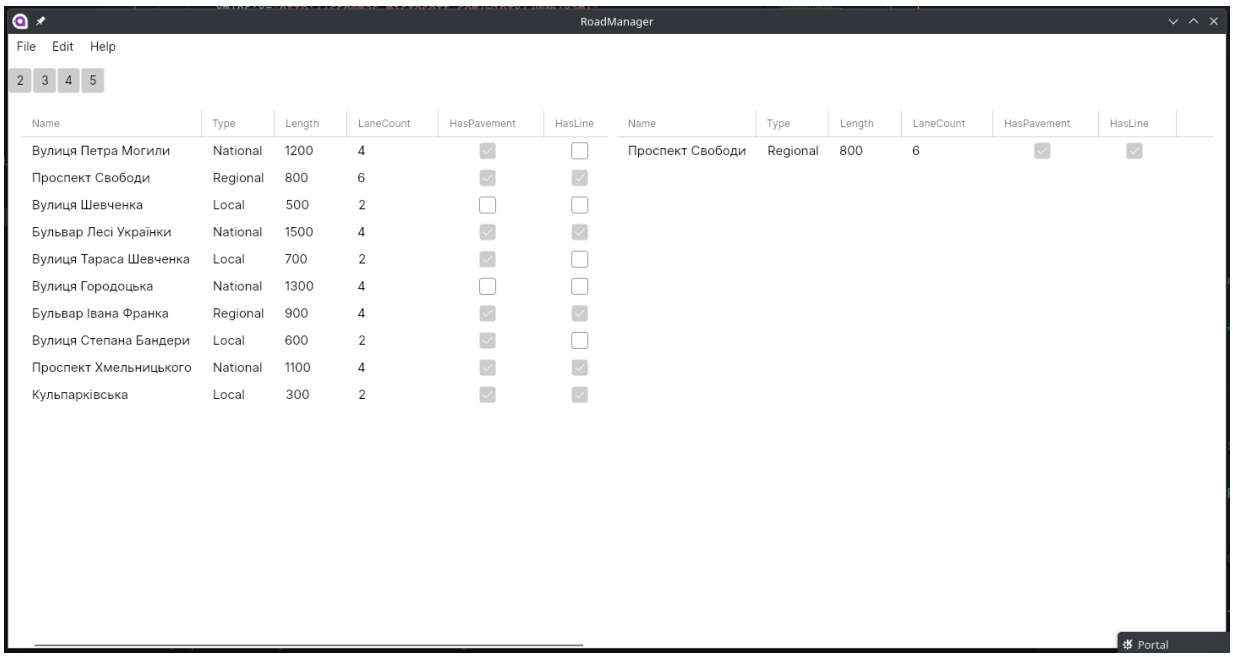


Рис. 12. Натискання на кнопку 5

6 Інструкція користувача та системні вимоги

6.1 Встановлення ПЗ

Додаток створений за допомогою .NET тому для його запуску достатньо мати об’єктний файл програми та встановлений dotnet-runtime або ASPNET.

6.2 Налаштування ПЗ

Налаштування додатку користувачем не передбачається.

6.3 Посібник Користувача Додатку Менеджер Доріг

1. Відкриття Додатку:
    - Запустіть додаток "Менеджер Доріг".
2. Інтерфейс Головного Вікна:
    - При відкритті ви побачите головне вікно програми.
    - Воно відображає різні елементи, такі як кнопки, сітки і, можливо, таблицю з інформацією про дороги.
3. Додавання Дороги:
    - Для додавання дороги натисніть на меню "Edit".
    - З випадаючого меню оберіть "Add Road". Ця дія може викликати вспливаюче вікно або область введення, де ви можете ввести деталі про дорогу, такі як її назва, тип, довжина, кількість смуг, наявність покриття і лінії. Введіть необхідну інформацію і натисніть "Save"або еквівалент для додавання дороги.
4. Перегляд Доріг:
    - Основний інтерфейс може відображати список або таблицю, де показана інформація про дороги.
    - Кожен запис про дорогу може відображати її назву, тип, довжину, кількість смуг, наявність покриття і лінії у окремих колонках.
5. Пошук Конкретних Доріг:
    - Можуть бути кнопки або опції для знаходження доріг за конкретними умовами, наприклад, найдовша дорога із найбільшою кількістю смуг, дороги, що відповідають певним критеріям або найдовша дорога з покриттям.
    - Клацніть на ці опції, щоб виконати відповідний пошук.
6. Завантаження Доріг з Файлу:
    - Може бути опція для завантаження доріг із файлу.
    - Клацніть на "Load from File"або схожу кнопку, щоб відкрити діалогове вікно вибору файлу.

- Виберіть відповідно структурований файл із інформацією про дороги. Це може бути файл у форматі CSV.

7. Вихід з Додатку:
   - Для виходу з програми просто закрийте головне вікно або використайте будь-яку доступну кнопку "Exit"або "Close".
8. Додаткові Функції:
   - Додаток може мати додаткові функції, такі як сортування доріг, фільтрація або відображення конкретних деталей доріги при кліці на запис.
9. Обробка Помилок:
   - У випадку помилок або несподіваної поведінки додаток може показувати повідомлення про помилки або попередження, щоб користувач зміг зрозуміти, як вирішити проблему.

Запам'ятайте, кроки можуть трохи відрізнятися в залежності від конкретної реалізації та дизайну інтерфейсу додатку "Менеджер Доріг який ви використовуєте.

7 Опис виняткових ситуацій

7.1 FileNameException

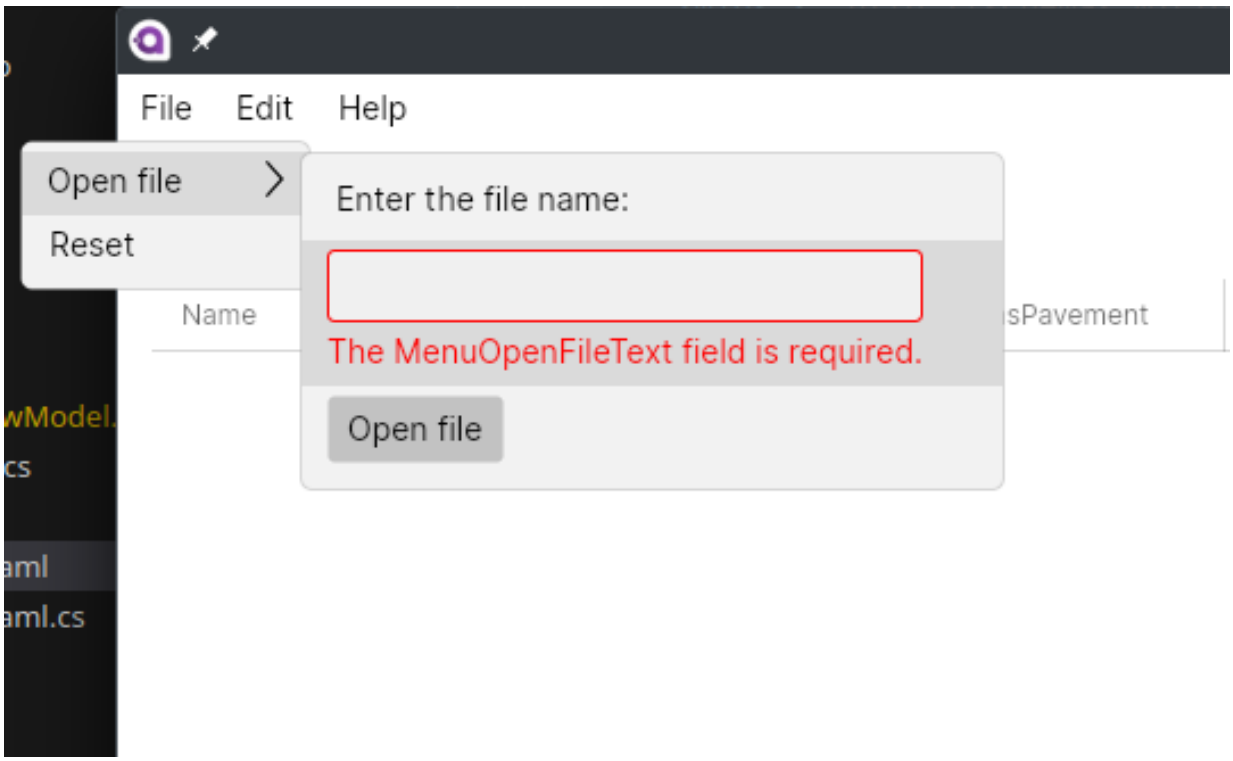


Рис. 13. Поле імені має бути обов'язково введено

7.2 NameException

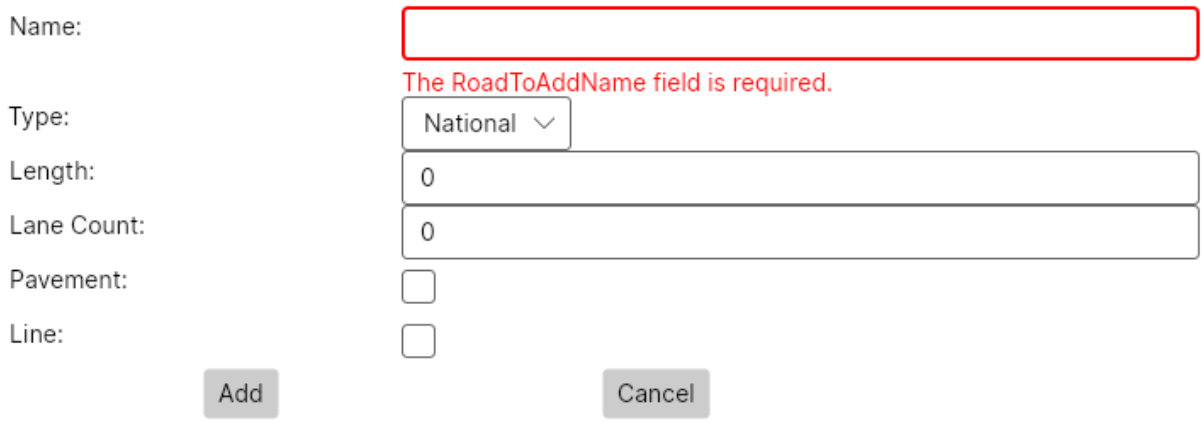


Рис. 14. Поле імені має бути обов’язково введено

7.3 LengthException

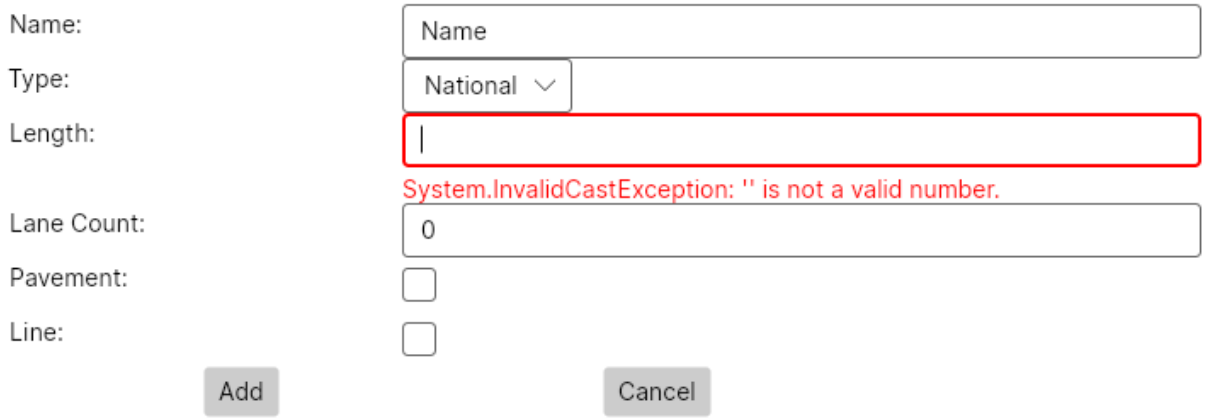


Рис. 15. Поле протяжності має бути більше нуля

7.4 LaneCountException

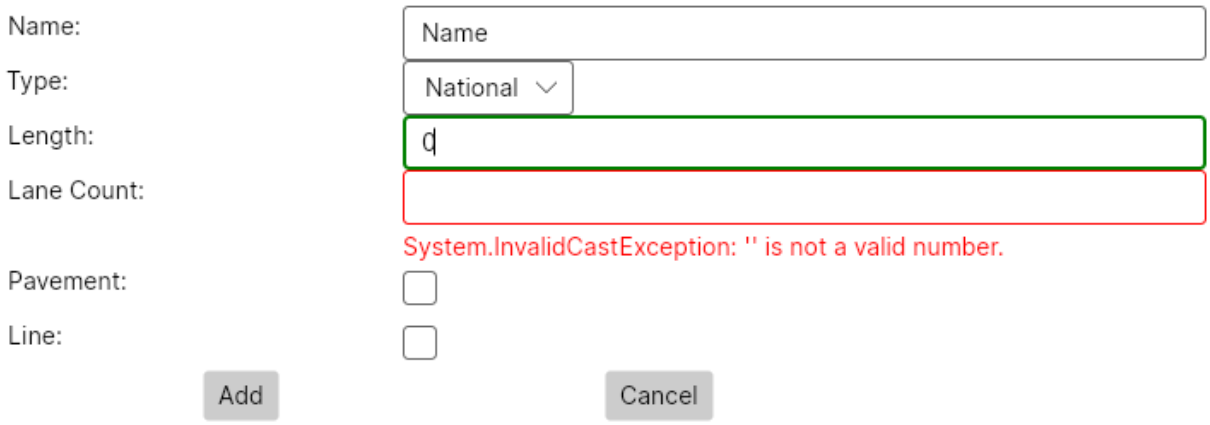


Рис. 16. Поле кількості смуг має бути більше нуля

8 Структура файлу вхідних даних

Програма має змогу зчитувати інформацію з двох типів файлів - .csv (Comma-separated Values) та .txt (Text). Система вибору файлу на ввід дозволяє користувачу вибрати файл через системний інтерфейс. Програма сприймає кожен рядок як дані для окремої машини, при тому, що поля машини розділені наступним чином:

Назва,Тип,Довжина,КількістьСмуг,НаявністьТротуару, НаявністьРозділювальноїСмуги

Зчитування передбачає, що після коми відсутні пробіли, якщо ж присутні – програма автоматично їх вилучає.

Назва і тип це стрічки, довжина і кількість смуг цілі числа, наявність тротуару та розділювальної смуги це булеві значення що задаються ключовими словами false і true

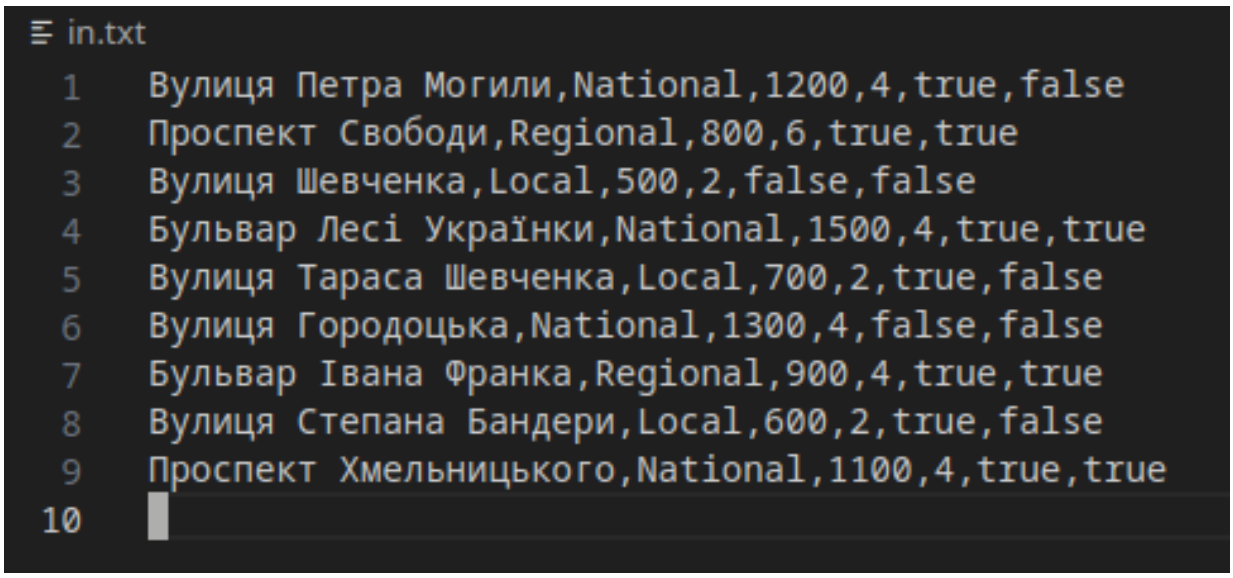


Рис. 17. Поле кількості смуг має бути більше нуля

9 Висновки

У результаті виконання курсової роботи мною було закріплено основні принципи та можливості об'єктно орієнтованого програмування. Для цього я спроєктував та розробив програмне забезпечення для класу Машина. Для проєктування програми мною було застосовано знання з дисципліни «Інженерія програмного забезпечення», а саме: евристики написання програмного інтерфейсу, UML-діаграми для зображення архітектури програми, стандарти написання коду та тестування програми. У реалізації задумки було використано мову програмування C Sharp, що ідеально підходить для вивчення та застосування принципів об'єктно орієнтованого програмування. Для побудови зовнішнього вигляду програми та взаємодії користувача з нею мною було використано Avalonia UI, це кросплатформенний фреймворк для графічних інтерфейсів. Серед особливостей об'єктно орієнтованого підходу у моєму програмному забезпеченні було використано зокрема: створення користувацьких класів, полів, методів, конструкторів, статичних варіацій цих частин програми. Також, особливістю C Sharp є те, що всі стандартні та користувацькі змінні повинні бути або структурами, або класами, тобто навіть найменші змінні у програмі підтримують об'єктно орієнтований підхід, мають власні методи, тощо. Я ознайомився з такою особливістю і повністю застосував її у програмі. У підсумку поступового проєктування та реалізації програмного забезпечення було створено фінальну програму з повноцінною взаємодією к ористувача; передбаченими за допомогою механізму обробки об'єктів «Винятків» обробкою критичних ситуацій; можливістю отримати інструкцію користувача через інтерфейс програми; реалізованими всіма передбаченими варіантом функціями взаємодії з Машиною. Також, для зручності вводу було створено два способи зчитування даних: через відповідні поля інтерфейсу у програмі через клавіатуру та за допомогою файлового зчитування.

10 Списки використаних джерел

1. Методичні вказіви до виконання курсової роботи з дисципліни «Об'єктно-орієнтоване програмування» для студентів першого (бакалаврського) рівня вищої освіти, напряму 121 «Інженерія програмного забезпечення» / Укл.: Коротєєва Т.О., Дяконюк Л.М., 27 стор., Сам.видав № 9347, 2020.

2. Є. В. Левус, Т. А. Марусенкова, О. О. Нитребич. Життєвий цикл програмного забезпечення / Є. В. Левус, Т. А. Марусенкова, О. О. Нитребич. - Львів : Видавництво Львівської політехніки, 2017.- 208 с.

3. C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development: Build applications with C#, .NET Core, Entity Framework Core, ASP.NET Core, and ML.NET using Visual Studio Code, 4th Edition/Mark J. Price.- Packt Publishing,2019.-818c.

4. C# For Beginners: An Introduction to C# Programming with Tutorials and Hands-On Examples/Nathan Metzler.,2018.-73p.

5. C# in Depth: Fourth Edition 4th Edition/Jon Skeet-Manning Publications.,2019.-528 c

6. Scott Meyers. Effective STL / AddisonWesley, an imprint of Pearson Education, Inc. – 60p.