RĪGAS VALSTS TEHNIKUMS

PROJEKTS

Tēma: Informācijas uzziņu sistēma  
 “Satiksmes vadības sistēma”

PIKC “Rīgas valsts tehnikums”  
“Programmēšanas tehniķis”  
DP2-1 kursa students  
Gundars Rēbuks

Darba vadītājs: Oksana Roslova

Rīga 2021

# SATURS

[SATURS 2](#_Toc102698681)

[Ievads 4](#_Toc102698682)

[1. Projekta uzdevums 5](#_Toc102698683)

[1.1. Projekta tēma 5](#_Toc102698684)

[1.2. Projekta mērķis 5](#_Toc102698685)

[1.3. Uzdevumi 5](#_Toc102698686)

[1.4. Programmas prasības 5](#_Toc102698687)

[2. Priekšmeta jomas analīze 6](#_Toc102698688)

[2.1. Priekšmeta jomas apraksts 6](#_Toc102698689)

[2.2. Objektu apraksts 6](#_Toc102698690)

[3. Projektēšana 7](#_Toc102698691)

[3.1. Programmas struktūras projektēšana 7](#_Toc102698692)

[3.1.1. Galvenā forma 8](#_Toc102698693)

[3.1.2. Tabulas/informācijas avota izvēlnes forma 8](#_Toc102698694)

[3.1.3. Datu apskates forma 9](#_Toc102698695)

[3.1.4. Datu rediģēšanas forma 9](#_Toc102698696)

[3.2. Metodes apraksts 10](#_Toc102698697)

[3.2.1. Datu rediģēšanas metode. 10](#_Toc102698698)

[3.2.2. Datu skatīšanas metodes. 10](#_Toc102698699)

[3.2.3. Izvēlnes izvades metodes. 10](#_Toc102698700)

[3.2.4. Izvēlnes un ievades apstrādes metodes. 10](#_Toc102698701)

[3.3. Programmu algoritmu projektēšana 11](#_Toc102698702)

[3.3.1. Datu pievienošanas metodes algoritms 11](#_Toc102698703)

[3.3.2. Tabulas izvēlnes metodes algoritms 11](#_Toc102698704)

[4. Kodēšana 13](#_Toc102698705)

[4.1. Galveno mainīgo apraksts 13](#_Toc102698706)

[4.2. Metodes apraksts 14](#_Toc102698707)

[4.3. Datu bāzes apraksts 18](#_Toc102698708)

[5. Testēšana 19](#_Toc102698709)

[5.1. Tests 1. Liela izmēra datu izvade 19](#_Toc102698710)

[5.2. Tests 2. Dažādu izmēru datu izvade tabulā 19](#_Toc102698711)

[5.3. Tests 3. Informācijas saglabāšana datu bāzē 19](#_Toc102698712)

[6. Programmas darbības rezultāti 19](#_Toc102698713)

[7. Secinājums 23](#_Toc102698714)

[8. Literatūras avoti 24](#_Toc102698715)

[Pielikumi 24](#_Toc102698716)

[Pielikums 1: Klases *Program* kods: 24](#_Toc102698717)

[Pielikums 2: Klases *DBConnection* kods 24](#_Toc102698718)

[Pielikums 3: Klases *TableBuilder* kods 24](#_Toc102698719)

[Pielikums 4: Klases *UserInterface* kods 24](#_Toc102698720)

[Pielikums 5: Klases *Information* kods 24](#_Toc102698721)

[Pielikums 6: klases *Driver* kods 24](#_Toc102698722)

[Pielikums 6: Klases *Transport* kods 24](#_Toc102698723)

[Pielikums 7: Klases *Route* kods 24](#_Toc102698724)

# Ievads

Projekta rakstīšanas mērķis ir izstrādāt uzziņu un informācijas sistēmu, kas ļauj uzglabāt informāciju datu bāzē un apstrādāt informāciju satiksmes vadības sistēmā. Mērķis tika sasniegts, analizējot mācību priekšmetu jomu, izstrādājot, kodējot un testējot lietojumprogrammu.

Projekta darbs tika veikts C# programmēšanas valodā, izmantojot Microsoft Visual Studio 2019 IDE.

Izstrādātajai lietojumprogrammai ir skaidrs lietotāja interfeiss, un tā ļauj veikt šādus uzdevumus:

1. Skatīt esošo datu bāzi.
2. Datu rediģēšana (datu pievienošana, dzēšana).
3. Datu kārtošana pēc noteikta kritērija.

Atskaitē ir vairākas daļas.

Atskaite pirmajā daļā ir aprakstīti mērķi, uzdevumi un prasības projekta izstrādei par

izvēlēto tēmu.

Otrajā daļā tiek analizēta priekšmeta joma (priekšmeta jomas un objektu apraksts).

Trešajā daļā ir aprakstīta programmas struktūras veidošana (datu pievienošanas, apskatīšanas, dzēšanas, meklēšanas, šķirošanas formas), programmas algoritmu projektēšana.

Ceturtajā daļā ir kodēšana. Ir aprakstīti galvenie mainīgie, metodes un faili.

Piektā daļa ir testēšana. Tiek prezentēti īstenotie testi, kā arī radušās problēmas.

Sestajā daļā ir parādīti programmas rezultāti ekrānuzņēmumu veidā ar aprakstu.

Noslēgumā ir aprakstīti projekta darba rezultāti un secinājumi. Darbā ir arī atsauču un pielikumu saraksts ar programmas kodu. Atskaite sastāv no 80 lapām. Tajā ir 3 tabulas un 6 attēli ar blokshēmām.

# 1. Projekta uzdevums

## 1.1. Projekta tēma

Projekta tēma ir pilsētas sabiedriskā transporta satiksmes vadības sistēma vadības sistēma, kuru izmantos satiksmes pārvalde.

## 1.2. Projekta mērķis

Mērķis ir izstrādāt funkcionālu un ērti lietojamu vadības sistēmu, ar kuras palīdzību var pievienot, dzēst, mainīt, apskatīt informāciju par transportiem, vadītājiem, maršrutiem.

## 1.3. Uzdevumi

1. Savienojuma izveide ar datu bāzi, informācijas saglabāšana datu bāzē;
2. Izstrādāt lietojumprogrammas struktūru. Izstrādāt datu struktūras. Katram objektam jābūt savai struktūrai.
3. Izstrādāt lietotāja interfeisu – attēlojumu, kā tiks attēloti dati, kā tiks atlasīta tā vai cita darbība ar datiem, izvēlnes utt.
4. Izstrādāt algoritmus, kas veic programmas galvenās metodes (meklēšana, aprēķinātie raksturlielumi, dzēšana, kārtošana utt.)
5. Uzrakstīt programmas kodu.
6. Testēt programmu.
7. Uzrakstīt atskaiti.

## 1.4. Programmas prasības

1. Iekļautas metodes:

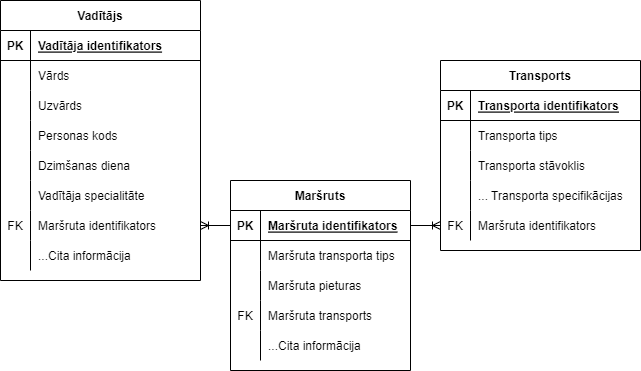
* Pievienot jaunus datu ierakstus;
* Saglabātās informācijas datu attēlošana;
* Datu dzēšana pēc noteikta kritērija;
* Datu meklēšana un filtrēšana ar iespēju uzstādīt dažādus kritērijus (vismaz 3);
* Datu kārtošana pēc vairākiem kritērijiem.

1. Atbilstība pēc dotajiem dokumentiem

# 2. Priekšmeta jomas analīze

## 2.1. Priekšmeta jomas apraksts

Priekšmeta joma ir satiksmes vadības sistēma, kuru izmantos satiksmes pārvalde. Tas dos iespēju mainīt, rediģēt, pievienot un dzēst maršrutus, kurus vada nodarbinātie vadītāji un katram maršrutam ir nozīmēti savi transportlīdzekļi.

Projektā tiks izmantotas 3 atsevišķas bāzes klases – Vadītājs, Maršruts un Transports. Transporta klase tiks sadalīta 4 apakšklasēs, kas norādīs katra transporta veidu, t.i. Autobuss, Tramvajs, Trollejbuss, Mikroautobuss.

## 2.2. Objektu apraksts

Apakšā tiek dota informācija par objektu glabātiem metadatiem un to datu tipiem. Šie objekti iekļauj:

1. Vadītājs – Informācija par transporta vadītāju jeb šoferi. Vadītājam tiks piešķirts maršruts.

* Vārds – *string* datu tips, glabā informāciju par vadītāja vārdu;
* Uzvārds – *string* datu tips, glabā informāciju par vadītāja uzvārdu;
* Personas kods – *string* datu tips, glabā informāciju par vadītāja personas kodu;
* Dzimšanas diena – *date* datu tips, glabā informāciju par vadītāja dzimšanas dienu;
* Vadītāja specialitāte – *List<string>* datu tips, glabā informāciju par vadītāja specialitāti. Vadītājs varēs vadīt tikai tos transportlīdzekļus, uz kuriem vadītājam ir specialitāte.
* Maršruta identifikators\* – *int* datu tips, glabā piešķirtā maršruta identifikatoru.

2. Maršruts – Informācijas par maršrutu. Pie katra maršruta tiks pievienoti atbilstošie vadītāji un transporti.

* Maršruta transporta tips: *List<string>* datu tips, glabā informāciju par maršruta transporta tipu, norāda maršruta tipus, kas var veikt šo maršrutu. Piemēram, vienu maršrutu var veikt gan autobuss, gan mikroautobuss, taču citu maršrutu var veikt tikai tramvajs;
* Maršruta pieturas: *List<string>* datu tips, glabā informāciju par maršruta veicamo pieturu nosaukumiem.

3. Transports – Informācija par transportu. Transportam tiks piešķirts veicamais maršruts.

* Transporta tips: *string* datu tips, glabā informāciju par transporta tipu. Transportu varēs piešķirt tikai pie tāda veicamā maršruta, kas atbilstīs ar transporta tipiem. Piemēram, tramvajs nevarēs veikt autobusa maršrutus.
* Transporta stāvoklis: *string* datu tips, glabā informāciju par transporta stāvokli. Tas var būt braucamā stāvoklī, bojāts, tiek remontēts u.t.t.
* Maršruta identifikators\*: *int* datu tips, glabā informāciju par maršruta identifikatoru, kas ir piešķirts dotajam transportlīdzeklim.

\*- Identifikatori tiks saglabāti datu bāzē. Kodā, iespējams, maršruta objektam tiks piešķirts saraksts ar atbilstošā maršruta vadītājiem un transportlīdzekļiem.

# 3. Projektēšana

## 3.1. Programmas struktūras projektēšana

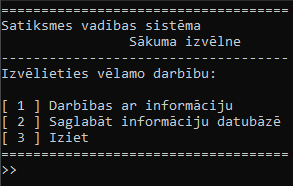
Programmas struktūra satur 7 moduļus:

1. Galvenā forma;
2. Tabulas/informācijas avota izvēles forma
3. Datu apskates forma
4. Datu rediģēšanas forma

### 3.1.1. Galvenā forma

Formā ir šādas izvēlnes:

1. Darbības ar informāciju – veikt darbības ar informāciju (apskatīt, pievienot, dzēst, labot utt.) Atver 2. formu – tabulas/informācijas avota izvēles formu;
2. Saglabāt informāciju datu bāzē – izmantojot izveidoto savienojumu ar datu bāzi, saglabā visas veiktās informācijas izmaiņas datu bāzē;
3. Iziet – aptur aplikācijas darbību.

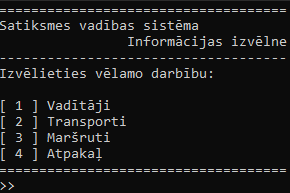


2.Att. Sākuma izvēlne konsolē

### 3.1.2. Tabulas/informācijas avota izvēlnes forma

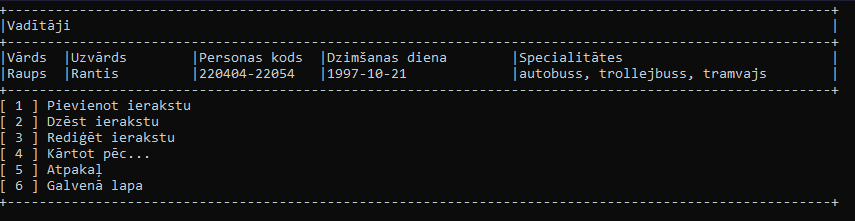
Formā ir šādas izvēlnes:

1. Vadītāji – Izvada tabulu ar visu vadītāju informāciju;
2. Transporti – Izvada tabulu ar visu transportu informāciju;
3. Maršruti – Izvada tabulu ar visu maršrutu informāciju;
4. Atpakaļ – Atgriež lietotāju atpakaļ uz sākuma izvēlni.

  
3.Att. Informācijas izvēlne konsolē

### 3.1.3. Datu apskates forma

Formā ir šādas izvēlnes:

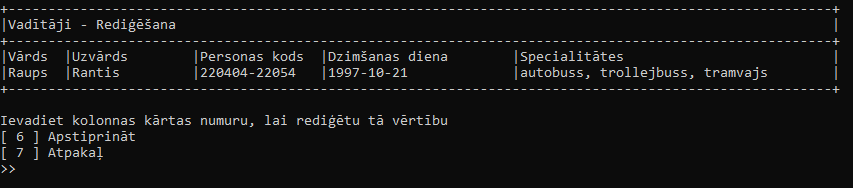
1. Pievienot ierakstu – Izveido jaunu ierakstu tabulā, kuru pēc tam lietotājs rediģē, izmantojot datu rediģēšanas formu;
2. Dzēst ierakstu – Jautā lietotājam, kuru ierakstu tas vēlas izdzēst, pēc tam šo ierakstu izdzēšot;
3. Rediģēt ierakstu – Pārvieto lietotāju uz datu rediģēšanas formu;
4. Kārtot pēc... – Kārto tabulas ierakstus pēc noteiktās kolonnas augošā secībā;
5. Atpakaļ – Pārvieto lietotāju uz formu, kura bija atvērta iepriekš;
6. Galvenā lapa – Pārvieto lietotāju uz galveno lapu.

4.Att. Datu apskates forma

### 3.1.4. Datu rediģēšanas forma

Formā ir šādas izvēlnes:

1. Kolonnas kārtas numura ievade – nosaka kolonnas numuru, kuru lietotājs vēlas rediģēt. Pēc tā ir iespējams rediģēt izvēlētās kolonnas vērtību izvēlētajam ierakstam.
2. Apstiprināt – Apstiprina izmaiņas un atgriežas atpakaļ uz datu apskates formu;
3. Atpakaļ – Dzēš izmaiņas un atgriežas atpakaļ uz datu apskates formu.

  
5.Att. Datu rediģēšanas forma

## 3.2. Metodes apraksts

Rakstot programmu, tiek izmantotas *void*, *List<string>*, *List<dynamic>*, *List<List<dynamic>>*, *MySqlDataReader*, *string*, *int* tipa metodes. Visas, izņemot *void* tipa metodes, atgriež kādu noteiktu vērtību un izmanto *return* paziņojumu.

### 3.2.1. Datu rediģēšanas metode.

*void EditObject();* - Rediģē objekta datus. Tiek izmantots arī objekta izveides fāzē. Tiek izveidots jauns objekts, un tiek izsaukta šī metode, lai mainītu objekta mainīgās vērtības. Datu rediģēšanai, pievienošanai tiek izmantota tikai viena, universāla metode, lai nebūtu nepieciešams saukt trīs dažādas metodes vienas vietā.

### 3.2.2. Datu skatīšanas metodes.

*void PrintInformation();* - izvada informāciju par objektu sarakstu tabulas veidā, ņemot vērā parametros norādīto objektu sarakstu un izvēlnes sarakstu. Strādā kā tilts starp klasēm *UserInterface* un *TableBuilder* – klase, kas izveidota tieši informācijas izvadei.

*string BuildTable();* - veic matemātiskus aprēķinus ar mazāku metožu palīdzību, kas beigās izveido tabulu un atgriež to *string* vērtībā tālākai izvadei. Tā veic aprēķinus tabulas optimālajam platumam, katras šūnas optimālam izmēram, kā arī, nepieciešamības gadījumā, izveido izvēlni.

### 3.2.3. Izvēlnes izvades metodes.

*void MainMenu();* - izvada galveno izvēlni.

*void TableSelection();* - izvada klases/tabulas izvēlni apskatei.

### 3.2.4. Izvēlnes un ievades apstrādes metodes.

*void AskInput(); -* pieprasa ievadi no lietotāja izsaucot *Input()* metodi, sagaidot kādu no izvēlnes kārtas skaitļiem kā ievadi, izsaucot atbilstošās metodes un veicot atbilstošās darbības. Šajā metodē tiek iekļauta datu dzēšana un datu pievienošana, jo *AskInput()* metode ir vienīgā vieta, kur parādās datu dzēšanas un pievienošanas funkcionalitāte un tā nesatur daudz rindas.

*int Input(); -* Saīsināta metode *Console.ReadLine()* funkcijai, ar papildus pārbaudi, vai ievadīta pareiza vērtība, kā arī vai ievades skaitlis nepārsniedz izvēlnes opciju skaitu.

## 3.3. Programmu algoritmu projektēšana

### 3.3.1. Datu pievienošanas metodes algoritms

Datu pievienošanas metodes algoritms programmā tiek izmantots metodē *void AskInput()* kā jauna ieraksta pievienošana kopējam objektu sarakstam, un tas turpinās *void EditObject()* metodē kā objekta vērtību aizpildīšana. Šo metožu algoritmi darbojas šādi:

1. *void AskInput()* metode:

* Tiek izveidots jauns objekts pēc saglabātā objekta tipa mainīgajā *Type viewingType*;
* Izveidotais objekts tiek saglabāts kā abstraktās bāzes klases *Information* reference;
* Reference tiek ievietota kopējā referenču sarakstē *List<Information> info*;
* Tiek izsaukta *EditObject()* metode, parametros norādot jaunizveidoto objektu.

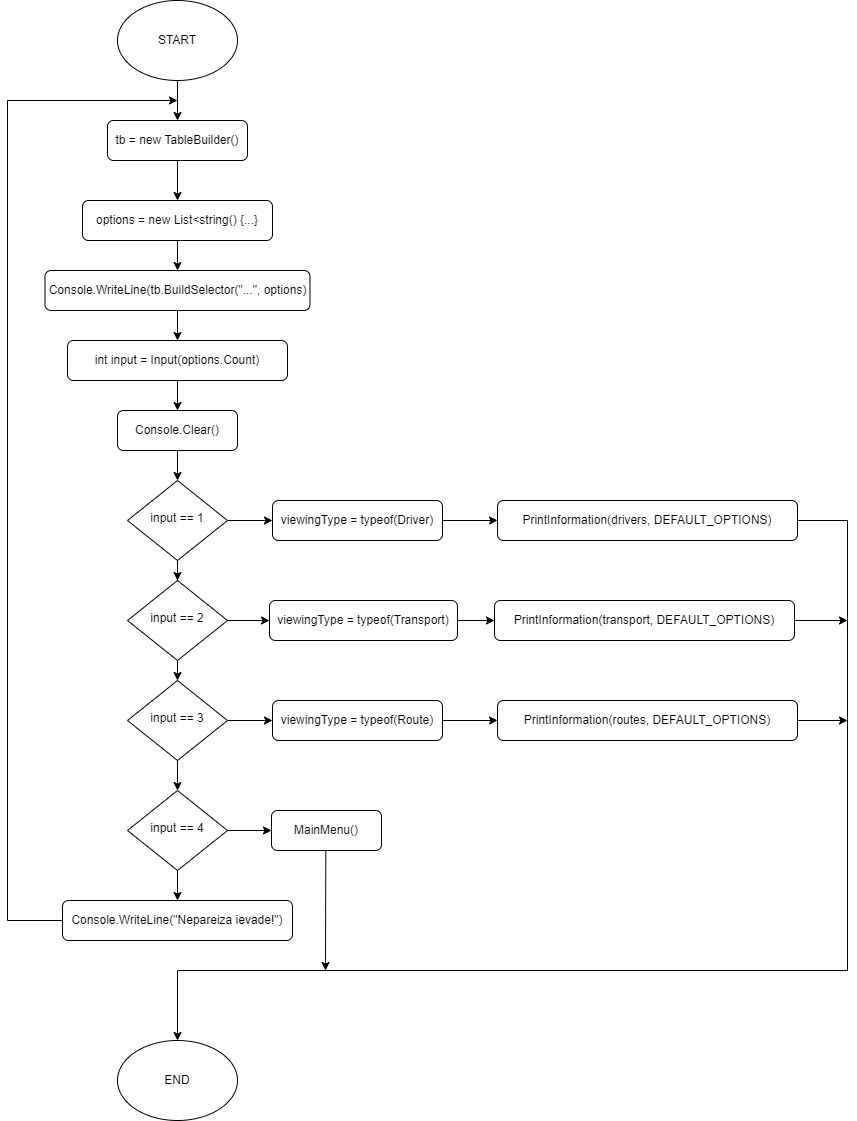
1. *void EditObject()* metode:

* Lietotājam tiek izvadītas pašreizējās sākuma objekta vērtības, kas tika piešķirtas objekta konstruktorā bez parametriem, izveidojot objektu;
* Lietotājam tiek pieprasīts ievadīt kolonnas kārtas skaitli noteiktas kolonnas vērtības rediģēšanai, vai ievadīt norādīto skaitļu vērtību (aprēķinātu pēc kolonnu skaita), lai atgrieztos atpakaļ vai apstiprinātu izmaiņas;’
* Ja lietotājs ir ievadījis skaitļu vērtību, kas apstiprina izmaiņas, lietotājs tiek atgriezts atpakaļ datu apskates metodē;
* Ja lietotājs ir ievadījis skaitļu vērtību, kas atgriež lietotāju atpakaļ, objekts tiek dzēsts un lietotājs tiek atgriezts atpakaļ datu apskates metodē;
* Ja lietotājs ir ievadījis kādu no kolonnu kārtas skaitļiem, no lietotāja tiek pieprasīta ievade šī kolonnas vērtības maiņai;
* Pēc ievades informācija tiek procesēta un saglabāta iekš objekta mainīgajiem;
* Lietotājam atkal tiek pieprasīta ievade, līdz brīdim, kad lietotājs ir ievadījis atgriešanās vai apstiprināšanas skaitli.

### 3.3.2. Tabulas izvēlnes metodes algoritms

Tabulas izvēlnes metodes algoritms tiek izmantots *void TableSelection()* metodē. Tas ļauj izvēlēties starp vairākām tabulām un ved uz datu izvades metodēm.

6. attēlā var redzēt tabulas izvēlnes metodes algoritma blokshēmu.

  
6.Att. metodes *void TableSelection();* algoritms;

# 4. Kodēšana

## 4.1. Galveno mainīgo apraksts

Tabula 1

**Galvenie mainīgie un to apraksts**

|  |  |  |
| --- | --- | --- |
| **N.p.k.** | **Mainīgais** | **Apraksts** |
| 1. | List<Information> drivers | Saglabā vadītāju objektu references sarakstē priekš izvades, tālākas apstrādes un saglabāšanas. |
| 2. | List<Information> transport | Saglabā transportu objektu references sarakstē priekš izvades, tālākas apstrādes un saglabāšanas. |
| 3. | List<Information> route | Saglabā maršrutu objektu references sarakstē priekš izvades, tālākas apstrādes un saglabāšanas. |
| 4. | DBConnection db | Saglabā datu bāzes “handle” un uztur savienojumu ar datu bāzi, tiek izmantots katru reizi, kad nepieciešamas darbības ar datu bāzi. |
| 5. | Type viewingType | Saglabā pašreiz skatāmo objektu tipu, pēc šīs informācijas programma veido jaunus objektus pēc tipa, kas saglabāts šajā mainīgajā. |
| 6. | List<string> DEFAULT\_OPTIONS | Saraksts ar pamata izvēlņu opcijām, kuras visbiežāk tiek izmantotas apskatot datus no tabulas. |
| 7. | string cs | Uztur savienojumam ar datu bāzi nepieciešamos datus. Tiek izmantoti katru reizi, kad tiek mēģināts izveidot savienojumu. |

## 4.2. Metodes apraksts

2. tabulā ir aprakstītas visas programmas kodā izmantotās metodes.

Tabula 2

**Metodes nosaukums un mērķis**

|  |  |  |  |
| --- | --- | --- | --- |
| **N.p.k.** | **Tips** | **Nosaukums** | **Mērķis** |
| **1. Interfeisa izstrādē izmantotās metodes (*TableBuilder* klase)** | | | |
| 1. | string | BuildTable | Metode pēc informācijas, kas pasniegta parametros, veido informācijas tabulu, kas veidota pēc aprēķiniem, lai noteiktu tabulas platumu, katras šūnas atkāpes, kolonnas lielumus u.t.t. Atgriež gatavu tabulu ar visu informāciju un izvēlni. |
| 2. | string | BuildSelector | Metode veido izvēlnes bez informācijas izvades, piem. galvenā izvēlne, tabulas izvēlne u.t.t. Parametros pieprasa virsrakstu un sarakstu ar izvēlnes opcijām, atgriež gatavu izvēlni kā string. |
| 3. | int | TableWidth | Aprēķina kopēju tabulas platumu, atgriež tabulas platumu simbolu izmērā. Pēc šī izmēra tiek veidotas tabulas BuildTable metodē. |
| 4. | int | MaxCellSize | Aprēķina maksimālo kolonnas izmēru pēc informācijas. Tā atgriež tuvāko vērtību, kas dalās ar 8, lai tabulā varētu izmantot tabulatorus (‘\t’), lai tabula izskatītos uniforma. Visu kolonnu maksimālie izmēri summējās TableWidth metodē. |
| 5. | void | BuildLine | Veido parastu līniju, kas sastāv no ‘-’ simboliem un ‘+’ simboliem galos pēc dotā izmēra parametros. |
| 6. | void | BuildCell | Veido vienu šūnu, kas ieraksta šūnas atdales līnijas, iekšā esošo informāciju un atbilstošo skaitu tabulatoru beigās. |
| 7. | void | FillEmpty | Šūnas vai vajadzīgajā vietā pievieno vajadzīgo tabulatoru skaitu, kas tiek aprēķināts no ievadītā izmēra parametros. |
| 8. | Void | BuildOptions | Veido izvēlni tabulas beigās vai pašā izvēlnē. |
| 9. | List<List<dynamic> | TransposeTable | Transponē tabulu, kolonnas samainot vietām ar rindām vieglākiem aprēķiniem. |
| **2. Interfeisa izstrādē izmantotās metodes (*UserInterface* klase)** | | | |
| 1. | void | MainMenu | Izvada galveno izvēlni |
| 2. | void | TableSelection | Izvada tabulas izvēlni |
| 3. | void | AskInput | Pieprasa informāciju no lietotāja, kas, izmantojot *switch*, pārbauda ievadīto skaitli un vai izsauc nepieciešamās metodes, vai pārvieto lietotāju tur, kur nepieciešams. |
| 4. | void | PrintInformation | Izvada tabulu ar informāciju, kas norādīts parametros – gan objektu sarakste, gan izvēlnes opcijas |
| 5. | void | EditObject | Izvada viena objekta informāciju, maina objekta informāciju pēc lietotāja pieprasījuma. |
| 6. | List<List<dynamic> | SortListByColumn | Kārto divdimensionālu saraksti pēc norādītās kolonnas alfabētiskā vai augošā secībā. |
| 7. | List<List<dynamic> | ObjectToDynamic (List<Information>) | Pārveido objektu sarakstu uz divdimensionālu nenoteiktu datu tipu saraksti, lai to pārvērstu it kā tabulas veidā. |
| 8. | List<List<dynamic> | ObjectToDynamic (Information) | Pārveido vienu objektu uz divdimensionālu nenoteiktu datu tipu saraksti. |
| 9. | List<List<dynamic> | ObjectToSpecificDynamic | Pārveido *Route* klases objektu specifiskā informācijas sarakstē, lai izvadītu detalizētu informāciju par individuālu maršrutu |
| 10. | int | Input | Jautā lietotājam ievadi un pārbauda, vai tas ir pareiza formāta un vai nepārsniedz izvēlnes opcijas skaitu – ja nepieciešams. |
| 11. | void | Redirect | Atgriež lietotāju iepriekšējā logā, izmantojot iepriekš saglabātus izsaukto metožu nosaukumus |
| **3. Datu bāzes apstrādes metodes** | | | |
| 1. | List<List<dynamic> | Select | Strādā kā SQL komanda SELECT, iegūst informāciju pēc parametriem no datu bāzes un saglabā divdimensionālā sarakstē |
| 2. | MySqlDataReader | Insert | Strādā kā SQL komanda INSERT INTO..., pēc parametriem saglabā informāciju iekš datu bāzes |
| 3. | MySqlDataReader | ReplaceAll | Pārraksta visu tabulas informāciju ar esošo informāciju, lieto datu saglabāšanai |
| 4. | void | GetInformation | Programmas darbības sākumā iegūst visu informāciju no datu bāzes. |

## 4.3. Datu bāzes apraksts

Darba gaitā programma iegūst informāciju un saglabā to datu bāzē, kas tiek servēta no lokālās mašīnas. 3. tabulā aprakstīta datu bāze.

Tabula 3

**Faili un struktūras**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N.p.k.** | **Tabulas nosaukums** | **Struktūra** | | | **Apraksts** |
| **Tips** | **Nosaukums** | **Izmērs** |
| 1. | driver | int | id | 11 | Vadītāju identifikators |
| varchar | name | 30 | Vadītāja vārds |
| varchar | surname | 30 | Vadītāja uzvārds |
| varchar | social\_number | 12 | Vadītāja personas kods |
| date | birth\_date | - | Vadītāja dzimšanas diena |
| varchar | speciality | 80 | Vadītāja specialitātes |
| 2. | route | int | id | 11 | Maršruta identifikators |
| varchar | name | 60 | Maršruta nosaukums |
| varchar | transport\_type | 60 | Maršruta transporta tips |
| text | stop\_list | 65535 | Pieturu saraksts |
| text | time\_displacement\_list | Pieturu laika starpības |
| text | start\_time\_list | Maršruta sākuma laiki |
| 3. | transport | int | id | 11 | Transporta identifikators |
| varchar | type | 40 | Transporta tips |
| varchar | transport\_condition | 40 | Transporta stāvoklis |

# 5. Testēšana

## 5.1. Tests 1. Liela izmēra datu izvade

Pārbaudot maršruta datu izvadi, radās problēmas ar pareizu tabulas izvadi. Konsoles ekrāns bija pārāk mazs, lai izvadītu visu pieturu sarakstu. Risinājums bija ieviest individuālu maršruta izvadi apskatei un kopējā maršruta saraksta izvades vienkāršošana.  
Šim nolūkam tika izmantota *GetSpecific()* metode, kas izvada viena objekta pilnu informāciju, taču ierastā metode *GetRow()* izvada saīsinātu informācijas apkopojumu – pieturu skaitu, nevis visu pieturu sarakstu.

## 5.2. Tests 2. Dažādu izmēru datu izvade tabulā

Pārbaudot vadītāju datu izvadi, radās problēmas ar tabulas izmēru aprēķiniem. Tabulas šūnas, kur informācijas izmērs, daloties ar 8, atlikums bija 7, 0, vai 1, tabula nepareizi izvadīja blakuslīnijas un tabulatorus. Risinājums bija ieviest specifisku pārbaudi pirms šūnu veidošanas, kas pārbauda informācijas izmēru, pielāgojot tabulu. Tas redzams metodē *BuildCell().*

## 5.3. Tests 3. Informācijas saglabāšana datu bāzē

Pārbaudot informācijas saglabāšanu datu bāže, radās problēmas ar datumu un laiku formātu saglabāšanu datubāzē. Lai saglabātu datumu vai laiku datu bāzē, tam ir jābūt specifiskā formātā, un sākotnēji datu bāze neņēma pretī ievadītās vērtības. Risinājums bija ieviest specifisku pārbaudi, vai ievadītie datumi ir ievadīti pareizā formātā.

# 6. Programmas darbības rezultāti

Kad tiek palaists Transport Management System.exe fails, tiek atvērta galvenā izvēlne (skat. 2. att.). Ievadot “1”, lietotājam ir iespēja veikt darbības ar informāciju. Ievadot “2”, lietotājam ir iespēja saglabāt visu mainīto informāciju datu bāzē. Ievadot “3”, programma beidz savu darbību.

Izvēloties 1. opciju, parādās tabulas izvēlne (skat. 3. att.). Lietotājam ir iespēja izvēlēties attiecīgi starp opciju “1” – apskatīt vadītāju tabulu, opciju “2” – apskatīt transportu tabulu, opciju “3” – apskatīt maršrutu tabulu. Izvēloties opciju “4”, lietotājs tiks aizvests atpakaļ uz galveno izvēlni.

Izvēloties pirmo opciju, tiks atvērta vadītāju tabula (Skat. 4. att.). Šajā tabulā tiek izvadīta visa informācija par visiem vadītājiem – vārds, uzvārds, personas kods, dzimšanas diena un viņu transportu specialitātes. Zem tabulas ir vairākas darbības/opcijas, ko var veikt ar šo informāciju. Opcija “1” ir datu pievienošanas opcija – lietotājam ir iespēja pievienot ierakstu. Opcija “2” ir datu dzēšanas opcija – lietotājam ir iespēja dzēst ierakstus. Opcija “3” ir datu rediģēšanas opcija – lietotājam ir iespēja kārtot visus tabulā redzamos datus pēc kādas no kolonnām augošā vai alfabētiskā secībā. Savukārt opcija “5” aizved lietotāju atpakaļ uz iepriekšējo logu, bet opcija “6” lietotāju aizved atpakaļ uz sākuma ekrānu.

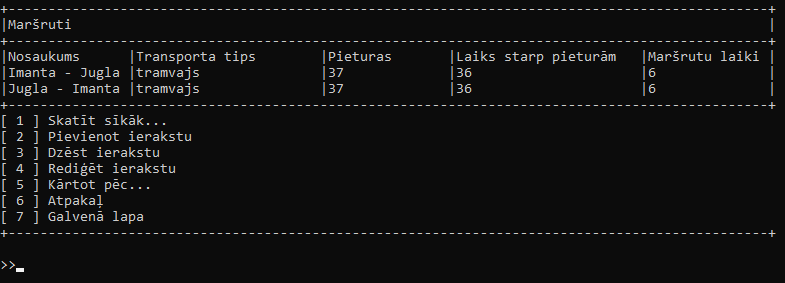
Izvēloties pirmo opciju – datu pievienošanas opciju (skat. 5. att.), programma ir izveidojusi jaunu ierakstu, un lietotājam to ir iespēja pielāgot datu rediģēšanas izvēlnē/logā. Lietotājam ir 2 opcijas – opcija (šajā gadījumā ar numuru “6”) apstiprināt izmaiņas un saglabāt jauno vadītāju, vai opcija ar numuru “7” atmest izmaiņas un atgriezties atpakaļ sākotnējā logā. Pirms atgriešanās sākotnējā logā, programma jautā apstiprinājumu – vai patiešām lietotājs vēlas atgriezties. Lietotājs maina informāciju, atzīmējot kolonnas kārtas numuru un mainot informāciju pēc vēlmēm (pievēršot uzmanību ievades formātam).

Izvēloties otro opciju – datu dzēšanas opciju, no lietotāja tiek pieprasīts ieraksta kārtas skaitlis, kuru lietotājs vēlas dzēst. Pēc lietotāja ievadītā kārtas skaitļa, programma pārbauda, vai ieraksts ar tādu kārtas numuru eksistē. Ja jā, tad tas tiek dzēsts no saraksta.

Izvēloties trešo opciju, programma pieprasa lietotājam ievadīt ieraksta numuru pēc kārtas. Pēc ievades, programma pārbauda ievadi, un, ja ievade atbilst kādai no ierakstiem, tad lietotājs tiek pārvietots uz datu rediģēšanas logu (skat. 5. att.). Lietotājam ir 2 opcijas – opcija (šajā gadījumā ar numuru “6”) apstiprināt izmaiņas un saglabāt jauno vadītāju, vai opcija ar numuru “7” atmest izmaiņas un atgriezties atpakaļ sākotnējā logā. Pirms atgriešanās sākotnējā logā, programma jautā apstiprinājumu – vai patiešām lietotājs vēlas atgriezties. Lietotājs maina informāciju, atzīmējot kolonnas kārtas numuru un mainot informāciju pēc vēlmēm (pievēršot uzmanību ievades formātam).

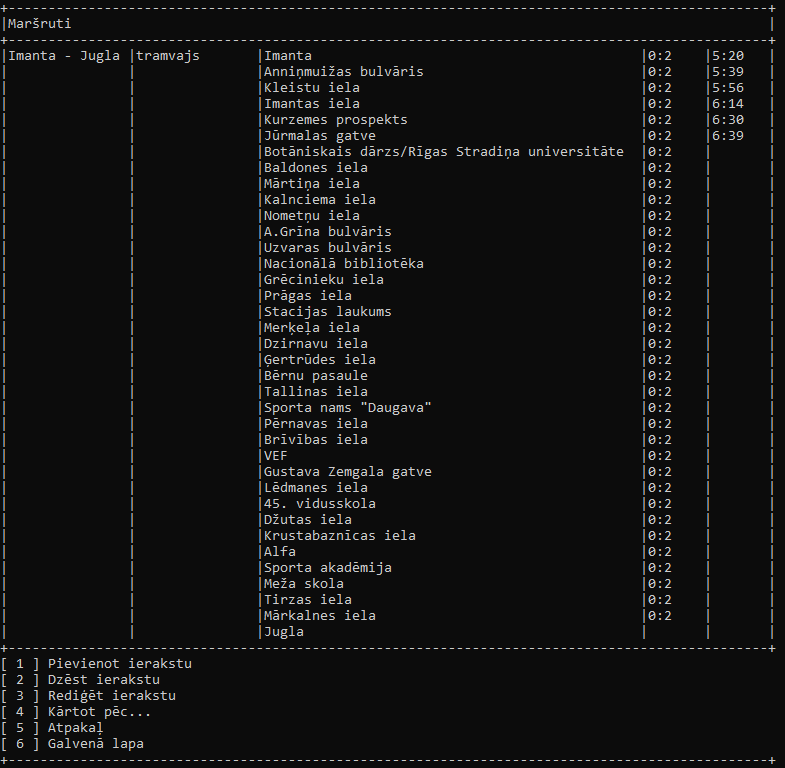
Izvēloties ceturto opciju, programma pieprasa lietotājam kolonnas kārtas numuru, pēc kura lietotājs vēlas kārtot visus ierakstus. Pēc ievades, programma pārbauda, vai ievadītais numurs atbilst kādai no kolonnām, un ja tas atbilst, tad tabula tiek kārtota alfabētiskā vai augošā secībā pēc izvēlētās kolonnas.

Izvēloties piekto opciju, lietotājs tiek pārvirzīts atpakaļ uz iepriekšējo logu, un sestā opcija pārvieto lietotāju uz galveno izvēlni.

Tabulas izvēlnē (3. att.) izvēloties otro opciju visas tālākās izvēlnes ir vienādas. Taču, izvēloties trešo opciju, parādās maršrutu tabula (skat. 7. att.), kur, izmēru ierobežojumu pēc, informācija par maršrutiem tiek sniegta saīsinātā veidā, t.i. tiek norādīti pieturu un laiku vērtību skaits. Ja lietotājs vēlas apskatīt kādu maršrutu detalizēti, tam ir nepieciešams izvēlēties opciju “1”, kas ir “Skatīt sīkāk...” opcija. Pārējās opcijas ir identiskas ar iepriekšējo tabulu opcijām.

7.Att. Maršrutu saīsināta izvade, maršrutu tabula

Izvēloties opciju “Skatīt sīkāk...”, programma pieprasa lietotājam ievadīt ieraksta numuru pēc kārtas, ko tas vēlas skatīt sīkāk. Pēc numura ievades, programma pārbauda, vai ievade sakrīt ar kādu no ierakstiem. Ja kāds ieraksts sakrīt, tiek atvērts sīkāks maršrutu paskaidrojums (skat. 8. att.), kur tiek izvadīta visa informācija par vienu ierakstu.

  
8.Att. Detalizēta maršruta izvade

# 7. Secinājums

Darba gaitā pie noteikta uzdevuma tika izstrādāta konsoles aplikācija par tēmu “Uzziņu un informācijas sistēmas izveide: “Satiksmes vadības sistēma””. Ir sasniegti un īstenoti mērķi kā savienojums ar datu bāzi, datu apskate, datu pievienošana, datu dzēšana, datu rediģēšana un datu kārtošana. Diemžēl netika sasniegti mērķi kā datu meklēšana un raksturlielumu skaitīšana.

Izstrādātā programma ļauj lietotājam pārvaldīt datus no datu bāzes, izmantojot ārpusēju programmatūru.

Projekta izpildes procesā bija arī kļūdas. Daudz laika tika pavadīts tabulas noformēšanas aprēķinos, datu pielāgošanā, lai to varētu ērti apstrādāt ar pēc iespējas mazāk dažādiem procesiem.

Izstrādāto aplikāciju var uzlabojot, pievienojot meklēšanas funkcionalitāti, raksturlielumu skaitīšanu, kā arī veicot vairākas minimālas izmaiņas programmas ērtai izmantošanai.

Labojot kļūdas, nācās apgūt jaunu materiālu, izmantojot pārsvarā interneta resursus, kā publiskos forumus, dokumentāciju un pamācības saitus. Visvairāk uzzināju, kā tieši .NET struktūra strādā ar “garbage collection”, datu tipiem, masīviem un sarakstiem. Pastiprināju savas zināšanas par objektu orientēto programmēšanu, izmantojot abstraktu klasi un metodes, virtuālās metodes un references.

# 8. Literatūras avoti

1. <https://stackoverflow.com/>

2. <https://www.w3schools.com/>

3. <https://docs.microsoft.com/en-us/dotnet/csharp/>

# Pielikumi

## Pielikums 1: Klases *Program* kods:

using System;

using System.Collections.Generic;

namespace Transport\_Management\_System

{

class Program

{

protected static List<Information> drivers = new List<Information>();

protected static List<Information> routes = new List<Information>();

protected static List<Information> transport = new List<Information>();

static void Main(string[] args)

{

Console.OutputEncoding = System.Text.Encoding.Unicode;

Console.InputEncoding = System.Text.Encoding.Unicode;

DBConnection db = new DBConnection();

GetInformation(db);

UserInterface.DB = db;

UserInterface.MainMenu();

}

static void GetInformation(DBConnection db)

{

List<List<dynamic>> data = new List<List<dynamic>>();

// Gets information about routes from database

data = db.Select("Route");

foreach (List<dynamic> row in data)

{

string name = row[1];

string transportType = row[2];

string stopString = row[3];

string stopTimeDifferenceString = row[4];

string routeStartTimeString = row[5];

routes.Add(new Route(name, transportType, stopString, stopTimeDifferenceString, routeStartTimeString));

}

// Gets information about transport from database

data = db.Select("Transport");

foreach (List<dynamic> row in data)

{

string transportType = row[1];

string condition = row[2];

Information new\_transport = new Transport(transportType, condition);

transport.Add(new\_transport);

}

// Gets information about drivers from database

data = db.Select("Driver");

foreach (List<dynamic> row in data)

{

string name = row[1];

string surname = row[2];

string socialNumber = row[3];

DateTime birthDate = row[4];

List<string> specialities = new List<string>(row[5].ToString().Split(", "));

Information driver = new Driver(name, surname, socialNumber, birthDate.Date, specialities);

drivers.Add(driver);

}

}

}

}

## Pielikums 2: Klases *DBConnection* kods

using System;

using System.Collections.Generic;

using MySql.Data.MySqlClient;

namespace Transport\_Management\_System

{

class DBConnection

{

private MySqlConnection conn;

private string cs = @"server=localhost;port=3306;userid=root;password=;database=satiksmes\_vadiba";

public DBConnection()

{

try

{

conn = new MySqlConnection(cs);

conn.Open();

}

catch (Exception e)

{

Console.WriteLine("Error: Could not connect to database.");

Console.WriteLine("Error message: " + e.Message);

}

}

public List<List<dynamic>> Select(string table, List<string> columns = null, List<string> options = null)

{

// If no columns specified

if (columns == null)

{

columns = new List<string>() { "\*" };

}

// If no options specified

if (options == null)

{

options = new List<string>() { "1" };

}

MySqlCommand cmd = new MySqlCommand($"SELECT {String.Join(", ", columns)} FROM {table} WHERE {String.Join(", ", options)}", conn);

MySqlDataReader read = cmd.ExecuteReader();

List<List<dynamic>> data = new List<List<dynamic>>();

dynamic[] currentRow = new dynamic[read.FieldCount];

while (read.Read())

{

read.GetValues(currentRow);

data.Add(new List<dynamic>(currentRow));

}

read.Close();

return data;

}

// If there are no options called

public List<List<dynamic>> Select(string table, List<string> columns)

{

List<string> options = new List<string>() {"1"};

return Select(table, columns, options);

}

// Select method if there are no columns or options called called

public List<List<dynamic>> Select(string table)

{

List<string> columns = new List<string>() { "\*" };

List<string> options = new List<string>() { "1" };

return Select(table, columns, options);

}

public MySqlDataReader Insert(string table, List<List<dynamic>> data)

{

string query = $"INSERT INTO {table} VALUES ";

List<string> subquery = new List<string>();

List<string> subqueryGroup = new List<string>();

subquery.Add("null");

foreach (List<dynamic> row in data)

{

foreach (dynamic cell in row)

{

subquery.Add($"\"{cell}\"");

}

subqueryGroup.Add($"({String.Join(",", subquery)})");

subquery = new List<string>();

subquery.Add("null");

}

query += String.Join(",", subqueryGroup);

MySqlCommand cmd = new MySqlCommand(query, conn);

MySqlDataReader read = cmd.ExecuteReader();

read.Close();

return read;

}

public MySqlDataReader ReplaceAll(string table, List<List<dynamic>> data)

{

MySqlCommand cmd = new MySqlCommand($"DELETE FROM {table} WHERE 1", conn);

MySqlDataReader read = cmd.ExecuteReader();

read.Close();

return Insert(table, data);

}

~DBConnection() {

conn.Close();

}

}

}

## Pielikums 3: Klases *TableBuilder* kods

using System;

using System.Text;

using System.Collections.Generic;

/// <summary>

/// Used to create output tables for printing out information

/// </summary>

namespace Transport\_Management\_System

{

class TableBuilder

{

private const int TAB\_SIZE = 8;

private const int ADD\_CHAR\_SIZE = 2;

private const string TITLE = "Satiksmes vadības sistēma";

private readonly List<string> DEFAULT\_OPTIONS = new List<string> {"Galvenā lapa", "Atpakaļ"};

private string \_tableHeader;

private List<List<dynamic>> \_tableCells;

private List<int> \_columnWidths = new List<int>();

private List<string> \_options;

// Data filtration and sorting

private List<string> criteria = new List<string>();

// Main method of building a table string

public string BuildTable(string tableHeader, List<List<dynamic>> tableCells, List<string> options = null)

{

\_tableHeader = tableHeader;

\_tableCells = tableCells;

\_options = options;

int width = TableWidth();

StringBuilder sb = new StringBuilder();

// Table header cell

BuildLine(width, ref sb);

BuildCell(\_tableHeader, width, ref sb);

sb.AppendLine("|");

BuildLine(width, ref sb);

// Table cell output

int max = 0;

foreach(List<dynamic> cell in \_tableCells)

{

int length = TransposeTable().ToArray().GetLength(0);

if (length > max)

{

max = length;

}

}

for (int i = 0; i < \_tableCells.ToArray().GetLength(0); i++)

{

for (int j = 0; j < max; j++)

{

try

{

BuildCell(\_tableCells[i][j], \_columnWidths[j], ref sb);

}

catch (ArgumentOutOfRangeException)

{

BuildCell("", \_columnWidths[j], ref sb);

System.Diagnostics.Debug.WriteLine("BT");

}

}

sb.AppendLine("|");

}

BuildLine(width, ref sb);

if(options != null)

{

BuildOptions(\_options, ref sb);

BuildLine(width, ref sb);

}

\_columnWidths = new List<int>();

return sb.ToString();

}

// Builds a selector table - doesn't output any data

public string BuildSelector(string tableHeader, List<string> options)

{

\_tableHeader = tableHeader;

\_options = options;

StringBuilder sb = new StringBuilder();

// Table header cell

sb.AppendLine("====================================");

sb.AppendLine($"{TITLE}\n\t\t{tableHeader}");

sb.AppendLine("------------------------------------");

sb.AppendLine("Izvēlieties vēlamo darbību:\n");

BuildOptions(options, ref sb);

sb.Append("====================================");

return sb.ToString();

}

// Calculates and returns the total width of a table

private int TableWidth()

{

int width = 0;

foreach (List<dynamic> columnCells in TransposeTable())

{

int columnSize = MaxCellSize(columnCells);

width += columnSize;

\_columnWidths.Add(columnSize);

}

return width;

}

// Calculates the maximum cell size of each column

// Iterates through the list to find the max value

private int MaxCellSize(List<dynamic> cells)

{

int max = 0;

foreach (dynamic cell in cells)

{

// Calculates the nearest tab size, using TAB\_SIZE const for correct table output

// ADD\_CHAR\_SIZE const used for the number of characters that surround the cell value

int nearestTab = (int)Math.Ceiling((double)(cell.ToString().Length + ADD\_CHAR\_SIZE) / TAB\_SIZE) \* TAB\_SIZE;

if (nearestTab > max)

{

max = nearestTab;

}

}

return max;

}

// Builds a basic line with the specified width

private void BuildLine(int width, ref StringBuilder sb)

{

for (int i = 1; i <= width+1; i++)

{

if (i == 1 || i == width+1)

{

sb.Append('+');

continue;

}

sb.Append('-');

}

sb.Append("\n");

}

// Builds a complete cell in a table, adding necessary tabs according to width

private void BuildCell(dynamic cell, int width, ref StringBuilder sb)

{

sb.Append($"|{cell}");

int difference = width - cell.ToString().Length;

// When these values are specific, the table breaks out of shape

// These checks prevent that from happening, excluding ADD\_CHAR\_SIZE const from the difference

if (cell.ToString().Length % 8 == 0 || cell.ToString().Length % 8 == 0 || difference % 8 == 1)

FillEmpty(difference - ADD\_CHAR\_SIZE, ref sb);

else

FillEmpty(difference, ref sb);

}

// Calculates and fills the empty spaces with tabs according to the specified width

private void FillEmpty(int width, ref StringBuilder sb)

{

for (int i = 0; i < Math.Ceiling((decimal)width / TAB\_SIZE); i++)

{

sb.Append("\t");

}

}

private void BuildOptions(List<string> options, ref StringBuilder sb)

{

int i = 1;

foreach (string option in options)

{

sb.Append($"[ {i} ] {option}\n");

i++;

}

}

// Returns a transposed table for size calculations

private List<List<dynamic>> TransposeTable()

{

List<List<dynamic>> transposedTable = new List<List<dynamic>>();

for (int i = 0; i < \_tableCells[0].Count; i++)

{

List<dynamic> col = new List<dynamic>();

for (int j = 0; j < \_tableCells.Count; j++)

{

try

{

col.Add(\_tableCells[j][i]);

}

catch(ArgumentOutOfRangeException)

{

col.Add("");

System.Diagnostics.Debug.WriteLine("transpose");

}

}

transposedTable.Add(col);

}

return transposedTable;

}

}

}

## Pielikums 4: Klases *UserInterface* kods

using System;

using System.Collections.Generic;

using System.Linq; // .Cast<dynamic>(), .ToList()

using System.Runtime.CompilerServices;

namespace Transport\_Management\_System

{

class UserInterface : Program

{

// Defines the default options and their count

private const int DEFAULT\_OPTION\_COUNT = 6;

private static readonly List<string> DEFAULT\_OPTIONS = new List<string>() { "Pievienot ierakstu", "Dzēst ierakstu", "Rediģēt ierakstu", "Kārtot pēc...", "Atpakaļ", "Galvenā lapa" };

// Holds the current viewing type so the program knows which class to create an object for

private static Type viewingType;

// Holds the database connection variable

private static DBConnection db;

// Prints the main menu of the application

public static void MainMenu()

{

TableBuilder tb = new TableBuilder();

List<string> options = new List<string>() { "Darbības ar informāciju", "Saglabāt informāciju datubāzē", "Iziet" };

Console.WriteLine(tb.BuildSelector("Sākuma izvēlne", options));

int input = Input(optionCount: options.Count);

Console.Clear();

switch (input)

{

case 1:

TableSelection();

break;

case 2:

db.ReplaceAll("driver", ObjectToDynamic(drivers, false));

db.ReplaceAll("route", ObjectToDynamic(routes, false, true));

db.ReplaceAll("transport", ObjectToDynamic(transport, false));

MainMenu();

break;

case 3:

Environment.Exit(0);

break;

default:

throw new NotImplementedException();

}

}

// Selection table so that the user can choose which table to output

static void TableSelection()

{

TableBuilder tb = new TableBuilder();

List<string> options = new List<string>() { "Vadītāji", "Transporti", "Maršruti", "Atpakaļ"};

Console.WriteLine(tb.BuildSelector("Informācijas izvēlne", options));

int input = Input(optionCount: options.Count);

Console.Clear();

switch (input)

{

case 1:

viewingType = typeof(Driver);

PrintInformation(drivers, DEFAULT\_OPTIONS);

break;

case 2:

viewingType = typeof(Transport);

PrintInformation(transport, DEFAULT\_OPTIONS);

break;

case 3:

viewingType = typeof(Route);

List<string> route\_options = new List<string>();

route\_options.Add("Skatīt sīkāk...");

route\_options.AddRange(DEFAULT\_OPTIONS);

PrintInformation(routes, route\_options);

break;

case 4:

MainMenu();

break;

default:

Console.WriteLine("Nepareiza ievade!");

TableSelection();

break;

}

}

// Asks the user input and compares them to the default options

static void AskInput(List<Information> info, List<string> options = null, [CallerMemberName] string memberName = "", List<List<dynamic>> sortedList = null)

{

TableBuilder tb = new TableBuilder();

int input = Input(optionCount: DEFAULT\_OPTIONS.Count);

int defaultOptionPosition = options.Count - DEFAULT\_OPTIONS.Count + 1;

switch (input)

{

// var value used to insert non-constant values in switch cases

case var value when value == defaultOptionPosition:

Console.Clear();

Information newObject = (Information)Activator.CreateInstance(viewingType);

info.Add(newObject);

EditObject(newObject, true, info);

PrintInformation(info, options, memberName);

break;

case var value when value == defaultOptionPosition + 1:

Console.WriteLine("Ievadiet kārtas numuru rindai, kuru vēlaties dzēst:");

input = Input(false);

Console.Clear();

if(sortedList == null)

info.RemoveAt(input - 1);

else

{

// PROBĻEMA - Es nezinu kāpēc neiet, bet ja fix tad sorted delete = done

info.RemoveAt(ObjectToDynamic(info).IndexOf(sortedList[input - 1]));

sortedList.RemoveAt(input - 1);

}

PrintInformation(info, options, memberName);

break;

case var value when value == defaultOptionPosition + 2:

Console.WriteLine("Ievadiet kārtas numuru rindai, kuru vēlaties rediģēt:");

input = Input(false);

Console.Clear();

if (sortedList == null)

EditObject(info[input - 1]);

else

{

Console.WriteLine(ObjectToDynamic(info).IndexOf(sortedList[input - 1]));

Console.WriteLine(input);

EditObject(info[ObjectToDynamic(info).IndexOf(sortedList[input - 1])]);

}

PrintInformation(info, options, memberName);

break;

case var value when value == defaultOptionPosition + 3:

Console.WriteLine("Ievadiet kolonnas kārtas numuru, pēc kuras vēlaties kārtot datus:");

input = Input(false);

Console.Clear();

List<List<dynamic>> sortList = SortListByColumn(ObjectToDynamic(info, false), input - 1);

Console.WriteLine(tb.BuildTable(info[0].Title, sortList, options));

AskInput(info, options, memberName, sortList);

break;

case var value when value == defaultOptionPosition + 4:

Redirect(memberName);

break;

case var value when value == defaultOptionPosition + 5:

Console.Clear();

MainMenu();

break;

default:

// Explicit case operation

if (options != DEFAULT\_OPTIONS)

{

if (viewingType == typeof(Route) && input == 1)

{

Console.WriteLine("Ievadiet kārtas numuru rindai, kuru vēlaties apskatīt:");

input = Input(false);

Console.Clear();

Console.WriteLine(tb.BuildTable(info[0].Title, info[input - 1].GetSpecific(), DEFAULT\_OPTIONS));

AskInput(info, DEFAULT\_OPTIONS, memberName);

}

}

else

{

Console.WriteLine("Nepareiza ievade!");

}

break;

}

}

// Outputs all table information in a table format

static void PrintInformation(List<Information> info, List<string> options = null, [CallerMemberName] string memberName = "")

{

TableBuilder tb = new TableBuilder();

Console.WriteLine(tb.BuildTable(info[0].Title, ObjectToDynamic(info), options));

AskInput(info, options, memberName);

}

// Edits a single object of class Information, is used in creating new objects as well

static void EditObject(Information editObject, bool isNewObject = false, List<Information> info = null, [CallerMemberName] string memberName = "")

{

TableBuilder tb = new TableBuilder();

bool editing = true;

int count = editObject.ColumnHeaders.Count;

dynamic[] values;

if (editObject is Route)

{

values = editObject.GetValues().ToArray();

}

else

{

values = editObject.GetRow().ToArray();

}

// Saves information in case the user cancels all changes

dynamic[] initialValues = new dynamic[values.Length];

Array.Copy(values, initialValues, values.Length);

do

{

Console.Clear();

if (editObject is Route)

{

Console.WriteLine(tb.BuildTable(editObject.Title + " - Rediģēšana", ObjectToSpecificDynamic(editObject)));

}

else

{

Console.WriteLine(tb.BuildTable(editObject.Title + " - Rediģēšana", ObjectToDynamic(editObject)));

}

Console.WriteLine("Ievadiet kolonnas kārtas numuru, lai rediģētu tā vērtību");

Console.WriteLine($"[ {count + 1} ] Apstiprināt");

Console.WriteLine($"[ {count + 2} ] Atpakaļ");

int input = Input(false);

if (input == count + 1)

{

editing = false;

}

else if (input == count + 2)

{

Console.WriteLine("Vai esiet pārliecināts ka vēlaties atgriezties? [Y/N]");

string answer = Console.ReadLine();

if (answer.ToLower() == "y")

{

if (isNewObject)

{

info.Remove(editObject);

Redirect(memberName);

return;

}

editObject.SetValues(initialValues);

Redirect(memberName);

return;

}

else continue;

}

else if (input <= count)

{

Console.WriteLine($"Ievadiet kolonnas '{editObject.ColumnHeaders[input - 1]}' jauno vērtību:");

// Expected a Date entry in DateTime column

if (viewingType == typeof(Driver) && input == 4)

{

try

{

Console.WriteLine($"Datums tiek ievadīts formātā: yyyy-MM-dd!");

values[input - 1] = Convert.ToDateTime(Console.ReadLine());

editObject.SetValues(values);

}

catch (FormatException) { }

continue;

}

values[input - 1] = Console.ReadLine();

editObject.SetValues(values);

continue;

}

} while (editing);

Console.Clear();

}

// Sorts the tables components by ascending or by alphabetical order

// Must specify the column index to sort by

static List<List<dynamic>> SortListByColumn(List<List<dynamic>> sortList, int columnIndex)

{

int numberOfChanges = 0;

for(int i = 1; i < sortList.Count() - 1; i++)

{

if (sortList[i][columnIndex] is string)

{

if (string.Compare(sortList[i][columnIndex], sortList[i + 1][columnIndex]) < 0

|| string.Compare(sortList[i][columnIndex], sortList[i + 1][columnIndex]) == 0)

{

continue;

}

}

else

{

if (sortList[i][columnIndex] >= sortList[i + 1][columnIndex])

{

continue;

}

}

List<dynamic> temp = sortList[i];

sortList[i] = sortList[i + 1];

sortList[i + 1] = temp;

numberOfChanges++;

}

if(numberOfChanges != 0)

{

SortListByColumn(sortList, columnIndex);

}

return sortList;

}

// Creates a dynamic two-dimensional list from list of Information class objects to be used elsewhere

static List<List<dynamic>> ObjectToDynamic(List<Information> objectTable, bool includeHeaders = true, bool isSpecific = false)

{

List<List<dynamic>> dynamicTable = new List<List<dynamic>>();

if(includeHeaders)

{

dynamicTable.Add(objectTable[0].ColumnHeaders.Cast<dynamic>().ToList());

}

foreach (Information obj in objectTable)

{

if (isSpecific)

{

dynamicTable.Add(obj.GetValues());

}

else

{

dynamicTable.Add(obj.GetRow());

}

}

return dynamicTable;

}

// Creates a dynamic two-dimensional list with one row from an Information class object to be used elsewhere

static List<List<dynamic>> ObjectToDynamic(Information obj)

{

List<List<dynamic>> dynamicTable = new List<List<dynamic>>();

dynamicTable.Add(obj.ColumnHeaders.Cast<dynamic>().ToList());

dynamicTable.Add(obj.GetRow());

return dynamicTable;

}

// Creates a dynamic two-dimensional list specifically for Route class, used to output specific information of a single Route object

static List<List<dynamic>> ObjectToSpecificDynamic(Information obj)

{

List<List<dynamic>> dynamicTable = new List<List<dynamic>>();

dynamicTable.Add(obj.ColumnHeaders.Cast<dynamic>().ToList());

dynamicTable.AddRange(obj.GetSpecific());

return dynamicTable;

}

// Asks the user for input and checks if input is valid

static int Input(bool optionCountDependant = true, int optionCount = DEFAULT\_OPTION\_COUNT)

{

int input = 0;

do

{

try

{

Console.Write(">>");

input = Convert.ToInt32(Console.ReadLine());

if (input > optionCount && optionCountDependant)

{

throw new FormatException();

}

}

catch (FormatException)

{

Console.WriteLine("Kļūda! Nepareiza ievade - mēģiniet vēlreiz");

}

}

while (input == 0);

return input;

}

// Redirects user to the previous window/method

static void Redirect(string methodName)

{

switch (methodName)

{

case "MainMenu":

Console.Clear();

MainMenu();

break;

case "TableSelection":

Console.Clear();

TableSelection();

break;

default:

Console.Clear();

MainMenu();

break;

}

}

// Properties

public static DBConnection DB

{

set { db = value; }

}

}

}

## Pielikums 5: Klases *Information* kods

using System.Collections.Generic;

namespace Transport\_Management\_System

{

abstract class Information

{

public abstract string Title { get; }

public abstract List<string> ColumnHeaders { get; }

public abstract List<dynamic> GetRow();

public virtual List<List<dynamic>> GetSpecific()

{

return new List<List<dynamic>>();

}

public virtual List<dynamic> GetValues()

{

return new List<dynamic>();

}

public abstract void SetValues(dynamic[] values);

}

}

## Pielikums 6: klases *Driver* kods

using System;

using System.Collections.Generic;

using System.Linq;

namespace Transport\_Management\_System

{

/// <summary>

/// Contains information about a driver.

/// </summary>

class Driver : Information

{

// Private variables

private static string title = "Vadītāji";

public static List<string> columnHeaders = new List<string>() {"Vārds", "Uzvārds", "Personas kods", "Dzimšanas diena", "Specialitātes"};

private string name;

private string surname;

private string socialNumber;

private DateTime birthDate;

private List<string> specialities;

// Constructor

public Driver(string name, string surname, string socialNumber, DateTime birthDate, List<string> specialities)

{

this.name = name;

this.surname = surname;

this.socialNumber = socialNumber;

this.birthDate = birthDate.Date;

this.specialities = specialities;

}

public Driver()

{

name = "Name";

surname = "Surname";

socialNumber = "Social number";

birthDate = DateTime.MinValue.Date;

specialities = new List<string>() {""};

}

public override List<dynamic> GetRow()

{

List<dynamic> row = new List<dynamic>();

row.Add(name);

row.Add(surname);

row.Add(socialNumber);

row.Add(birthDate.ToString("yyyy-MM-dd"));

row.Add(String.Join(", ", specialities));

return row;

}

public override void SetValues(dynamic[] values)

{

name = values[0];

surname = values[1];

socialNumber = values[2];

birthDate = Convert.ToDateTime(values[3]);

string[] specs = values[4].Split(", ");

specialities = specs.ToList();

}

// Deconstructor

~Driver()

{

System.Diagnostics.Debug.WriteLine("A driver has been deconstructed.");

}

// Properties

public override string Title

{

get { return title; }

}

public override List<string> ColumnHeaders

{

get { return columnHeaders; }

}

}

}

## Pielikums 6: Klases *Transport* kods

using System.Collections.Generic;

namespace Transport\_Management\_System

{

/// <summary>

/// Contains basic information of a transport.

/// Base class to derived classes: Tram, Trolleybus, Bus, Minibus

/// </summary>

class Transport : Information

{

// Private variables

private static string title = "Transporti";

public static List<string> columnHeaders = new List<string>() {"Tips", "Stāvoklis" };

private string type;

private string condition;

// Constructor

public Transport (string type, string condition = "N/A")

{

this.type = type;

this.condition = condition;

}

public Transport()

{

type = "Type";

condition = "N/A";

}

// Methods

public override List<dynamic> GetRow()

{

List<dynamic> row = new List<dynamic>();

row.Add(type);

row.Add(condition);

return row;

}

public override void SetValues(dynamic[] values)

{

type = values[0];

condition = values[1];

}

// Properties

public string Type { get; set; }

public string Condition { get; set; }

public override string Title

{

get { return title; }

}

public override List<string> ColumnHeaders

{

get { return columnHeaders; }

}

}

}

## Pielikums 7: Klases *Route* kods

using System;

using System.Collections.Generic;

using System.Linq;

namespace Transport\_Management\_System

{

/// <summary>

/// Contains information about available routes.

/// Contains assigned drivers and transport to the specific route object.

/// </summary>

class Route : Information

{

// Private variables

private static string title = "Maršruti";

private static List<string> columnHeaders = new List<string>() { "Nosaukums", "Transporta tips", "Pieturas", "Laiks starp pieturām", "Maršrutu laiki"};

private string name;

private string transportType;

private List<string> stops;

private List<DateTime> stopTimeDifference = new List<DateTime>();

private List<DateTime> routeStartTime = new List<DateTime>();

// Constructor

public Route(string name, string transportType, string stopString, string stopTimeDifferenceString, string routeStartTimeString)

{

this.name = name;

this.transportType = transportType;

stops = stopString.Split(", ").ToList();

foreach (string timeDifference in stopTimeDifferenceString.Split(", "))

{

stopTimeDifference.Add(Convert.ToDateTime(timeDifference));

}

foreach (string startTime in routeStartTimeString.Split(","))

{

routeStartTime.Add(Convert.ToDateTime(startTime));

}

}

public Route()

{

name = "Sākuma pietura - galamērķis";

transportType = "Transporta tips";

stops = new List<string>();

stopTimeDifference = new List<DateTime>();

stopTimeDifference.Add(DateTime.MinValue);

routeStartTime = new List<DateTime>();

routeStartTime.Add(DateTime.MinValue);

}

public override List<dynamic> GetRow()

{

List<dynamic> row = new List<dynamic>();

row.Add(name);

row.Add(transportType);

row.Add(stops.Count);

row.Add(stopTimeDifference.Count);

row.Add(routeStartTime.Count);

return row;

}

public override List<dynamic> GetValues()

{

List<dynamic> row = new List<dynamic>();

row.Add(name);

row.Add(transportType);

row.Add(String.Join(", ", stops));

List<string> timeDiffString = new List<string>();

foreach (DateTime timeDiff in stopTimeDifference)

{

timeDiffString.Add(timeDiff.ToString("H:m"));

}

row.Add(String.Join(", ", timeDiffString));

timeDiffString = new List<string>();

foreach (DateTime timeDiff in routeStartTime)

{

timeDiffString.Add(timeDiff.ToString("H:m"));

}

row.Add(String.Join(", ", timeDiffString));

return row;

}

public override List<List<dynamic>> GetSpecific()

{

List<List<dynamic>> table = new List<List<dynamic>>();

List<dynamic> row = new List<dynamic>();

row.Add(name);

row.Add(transportType);

for (int i = 0; i < Math.Max(stops.Count, Math.Max(stopTimeDifference.Count, routeStartTime.Count)); i++)

{

if (i < stops.Count)

row.Add(stops[i]);

else row.Add("");

if (i < stopTimeDifference.Count)

row.Add(stopTimeDifference[i].ToString("H:m"));

else row.Add("");

if (i < routeStartTime.Count)

row.Add(routeStartTime[i].ToString("H:m"));

else row.Add("");

table.Add(row);

row = new List<dynamic>();

row.Add("");

row.Add("");

}

foreach (string stop in stops)

{

System.Diagnostics.Debug.WriteLine(stop);

}

System.Diagnostics.Debug.WriteLine($"Count = {stops.Count}, Length = {stops.ToArray().GetLength(0)}");

return table;

}

public override void SetValues(dynamic[] values)

{

this.name = values[0];

this.transportType = values[1];

stops = new List<string>(values[2].ToString().Split(", "));

stopTimeDifference = new List<DateTime>();

routeStartTime = new List<DateTime>();

foreach (string timeDifference in values[3].Split(", "))

{

stopTimeDifference.Add(Convert.ToDateTime(timeDifference));

}

foreach (string startTime in values[4].Split(","))

{

routeStartTime.Add(Convert.ToDateTime(startTime));

}

}

// Properties

public override string Title

{

get { return title; }

}

public override List<string> ColumnHeaders

{

get { return columnHeaders; }

}

}

}