ПОЯСНЮВАЛЬНА ЗАПИСКА

ЗАВДАННЯ

КАЛЕНДАРНИЙ ПЛАН

АНОТАЦІЯ

Гадзера І. С.«Розроблення системи збору даних про дорожні маршрути». Дипломнийпроект. Спеціальність 7.05010103 – «Системне проектування». НУ «Львівська політехніка», кафедра САП, Львів 2015.

Робота складається з 99 сторінок, 47 рисунків, 7 таблиць, 5 додатків, 30 використаних джерел.

Ключовими словами є: інформація, база даних, веб-додаток, інтерфейс, маршрут, відгук, MVC.

У першому розділі роботи проводиться аналіз сучасного стану методів та засобів розробки веб-додатків та проектування бази даних, а також огляд літературних джерел. У другому розділі розглядаються існуючі засоби для розробки веб-додатку, а також вибір засобів для проектування системи для збору даних про дорожні маршрути. Починаючи з третього розділу описано процес програмної реалізації дипломного проекту, а саме: проектування бази даних, реалізації доступу до даних, деталі розроблення архітектури, розробка адміністративної панелі, пошук існуючий та створення нових маршрутів за допомогою користувацької панелі веб-додатку.

Четвертий розділ присвячений економічній оцінці проектного рішення, що доводить можливість позитивних економічних ефектів. Для перевірки цих ефектів було розраховано витрати на розроблення і впровадження проектного рішення.

ABSTRACT

Hadzera I.S. «Development of the road routes data collection system». Diploma project. Specialty 7.05010103 - "System Design". Lviv Polytechnic National University, Department of CAD, Lviv 2015.

The work consists of 99 pages, 47 figures, 7 tables, 5 applications, 30 sources used.

The key words are: information, database, web application, interface, route, feedback, MVC.

In the first chapter analyzes the current state of methods and tools for web application development and database design, and review of the literature. The second section deals with the existing tools for web application development and design to the choice of system for collecting on road routes data. Since the third section describes the software implementation of the diploma project, namely, database design, implement data access details of architecture development, development administration panel, search for existing and create new routes by using a custom web application panel.

The fourth chapter is devoted to economic evaluation of the project decision that brings the possibility of positive economic effects. To verify these effects the costs of project development and realization were calculated.

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ, СКОРОЧЕНЬ, ПОЗНАЧЕНЬ

Визначення	Поняття
MVC	Model-View-Controller
JS	JavaScript
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
БД	База даних
СКБД	Система керування базами даних
НФ	Нормальна форма
ПЗ	Програмне забезпечення
SS	Smart Search
ПО	Предметна область
IT	Інформаційні технології
SQL	Structured query language
CLR	Common Language Runtime
EF	Entity Framework
REST	Representational State Transfer
WCF	Windows Communication Fundation
XML	Extensible Markup Language
LINQ	Language Integrated Query
URL	Uniform Resource Locator
GUI	Graphical user interface
JSON	Language Integrated Query
DF	Database First

ВСТУП

Інформація відіграє колосальну роль в сучасному житті. За останнє десятиліття, доступність та важливість інформації зросла в сотні разів, що в свою чергу спричинило появу безлічі сервісів та ресурсів, які полегшують як наукові дослідження, торгівельні зв’язки та промислові потужності, так і повсякденне життя кожної людини.

Процес перетворення даних в інформацію включає 2 основних етапи: збір якісних даних та аналіз зібраних даних. Успішному проходженню цих етапів може сприяти завчасна підготовка опитувальних листів або анкет, та розробка алгоритмів опрацювання отриманих даних з метою формування висновків та статистичної інформації.

Метою даної роботи є проектування та розробка автоматизованої системи для збору даних про дорожні маршрути.

Об’єктом даного наукового дослідження є методи збору даних про дорожні маршрути та способи аналізу цих даних для подальшої обробки, а саме складання маршрутів.

Наукова та практична цінність роботи полягає в розробці інформаційної системи для аналізу отриманої від респондента інформації, що дозволить фільтрувати контент відповідно до потреб користувача.

Система збору даних орієнтована на збереження та аналіз значних обсягів інформації, з можливістю доповнення цієї інформації користувачем. Також система дозволить створювати нові дані на основі аналізу отриманої від користувача інформації.

Отже, можна сказати, що розроблення автоматизованої системи збору даних про дорожні маршрути є економічно доцільною і має маркетингові перспективи, так як сьогодні, не існує повністю подібних аналогів на ринку інформаційних технологій в Україні.

РОЗДІЛ 1.ТЕОРЕТИЧНІ ОСНОВИ ПРОЕКТУВАННЯ АВТОМАТИЗОВАНИХ СИСТЕМ ДЛЯ ЗБОРУ ДАНИХ ПРО ДОРОЖНІ МАРШРУТИ

1.1. Опис та актуальність поставленої задачі

Метою дипломної роботи є дослідження, проектування та створення автоматизованої системи для збору та аналізу інформації про дорожні маршрути. Актуальність створення системи з`являється в контексті із актуальністю даної предметної області. Адже у сучасному світі технології і мобільність, кардинально змінюють наше життя.

Робота присвячена розв’язанню актуальної проблеми розвитку нового напряму –системи автоматизації збору даних про дорожні маршрути, що ставить на надійну основу вирішення важливої науково-технічної та соціальної задачі створення нових засобів для збирання, введення, записування, перетворення, зчитування, зберігання, знищення інформації, які б забезпечили автоматизовану обробку даних та генерацію результуючої інформації та висновків.

У таких засобах особливо зацікавлені автомобілісти, які можуть в такий спосіб зменшити час на прокладання оптимального маршруту за власними критеріями, за рахунок отримання результатів аналізу на основі статистичних даних інших мандрівників. Як показують дослідження, такі засоби мають певні переваги перед традиційними засобами, такими як картографічні сервіси. Серед таких основних переваг є можливість перегляду існуючих даних та побудова маршрутів на основі відгуків інших респондентів.

Відомо, що на сьогоднішній день близько 70% всіх подорожей у світі відбуваються особистими автомобілями - це зручно, відносно швидко і недорого. Для подорожей автомобілем не потрібно враховувати розклад руху або наявність квитків, а лише вибрати оптимальний маршрут і розпочати подорож. Зазвичай мандрівник перед подорожжю відкриває картографічний сервіс для планування маршруту і цей спосіб є виправданим якщо метою є добратись з одного пункту в інший, лише з урахуванням відстані. Але жоден з

існуючих сервісів не прокладає маршрут з огляду на красу оточуючої природи, чи якості дорожнього покриття (принаймні в Україні), або ж просто на основі відгуків інших людей, що їздили даним маршрутом.

Для вирішення даної задачі необхідно спроектувати базу для збереження даних, побудувати автоматизовану систему на основі нових технологій, яка буде збирати, зберігати, аналізувати та пропонувати користувачеві маршрути згідно проаналізованої інформації.

У роботі поставлені наступні завдання:

- дослідити існуючі підходи та виділити основні напрямки їх розвитку;
- проаналізувати шляхи побудови систем збору даних про дорожні маршрути і сформулювати методологію створення автоматизованих систем;
- дослідити проблеми забезпечення ефективності процесів аналізу даних про маршрути;
- синтезувати та критично проаналізувати функціонування даної системи;
- розробити структурну модель функціонування програми та алгоритми аналізу статистичних даних.
- проаналізувати шляхи розробки системи, використовуючи технології .Net Framework, мови програмування C#, технології розробки веб додатківASP.NETMVC, JS, CSS, HTML 5.0, AngularJs і сформулювати методологію створення даної системи.
- провести теоретичні розрахунки і експериментальні дослідження різних конструкцій для зручного та точного отримання інформації від респондента;
- провести теоретичний аналіз та експериментальну перевірку програми і визначити шляхи її покращення.
- виконати узагальнення отриманих результатів.

1.2. Огляд літературних джерел

Питання збору даних та перетворення їх в інформацію за допомогою обробки та аналізу давно є об’єктом пильної уваги наукових досліджень і методичних розробок. Такі науковці як Волович Ю. П., Батигін В.С., Панін Н.В. [1-3] і багато інших, внесли вагомий вклад в розвиток та покращення процесів збору та аналізу інформації. Принципи, що закладені у роботах цих авторів стали фундаментом для подальших досліджень у даній сфері.

Збір даних можна виконувати за допомогою методів первинного і вторинного дослідження, які зображено на рисунку 1.1. До методів первинного дослідження можна віднести: опитування, спостереження, моделювання, періодичні дослідження та збір даних у процесі практичного експерименту. Ці методи застосовуються для вирішення задач різного рівня складності. Водночас кожний з методів може використовуватися в поєднанні з іншими [4].

Рис. 1.1. Методи збору даних

За допомогою опитування можна одержати інформацію, яка не завжди присутня в документальних джерелах чи доступна прямому спостереженню. До опитування вдаються, коли необхідним, а часто і єдиним джерелом інформації є людина - безпосередній учасник, представник, носій досліджуваних явища чи

процесу. Перевагою методу опитування є його універсальність. Вона полягає в тому, що при опитуванні реєструють і мотиви діяльності індивідів, і результати їх діяльності. Все це забезпечує методові опитування переваги, не властиві ні методові спостереження, ні методові аналізу документів[3].

Спостереження — це метод, що використовується для збору інформації, яка важко піддається запису. Респондент проводить спостерігає за перебігом подій і робить власні висновки щодо певної ситуації чи явища [1].

Пропонують такі види спостережень: розвідувальні, описові, аналітичні, експериментальні. Як різновид аналітичного спостереження виділяють експеримент [2].

Вибір методів збору даних залежить від кінцевої мети, співвідношення відповідних витрат та очікуваних результатів. Результат багатьох експериментів різних галузей свідчить, що вторинні дослідження дозволяють досягати мети з меншими витратами порівняно з первинними.

Збір даних вторинними методами дозволяє створити основу для аналізу наукової діяльності та виявлення тенденцій її розвитку. У тих випадках, коли вторинне дослідження не дає потрібного результату, проводять первинне дослідження. Високі витрати первинного дослідження мають бути компенсовані важливістю завдання, що вирішується, та очікуваними результатами [4], однак первинні дослідження дають більш точні статистичні дані, хоч і більшою ціною.

Пошук, збір та аналіз інформації є дуже важливими в епоху інформаційних технологій. У сучасному світі стільки інформації, що час задуматися над тим, як правильно її використовувати. Усе знання цього світу здається доступним. Але що є справді новим, що достовірним? Через надлишок інформації більшість людей не мають змоги розгледіти суттєве. Ключ до вирішення проблеми називається «методичний пошук і збір інформації».

На сьогоднішній день автоперевезення та туризм автомобілем є невід’ємною частиною сучасного суспільства. Подорожувати автомобілем зручно та відносно дешево. Головною задачею при подорожі автомобілем є вибір оптимального маршруту. Для кожного окремого випадку критерії

оптимальності будуть різними, тому маршрут від точки А в точку Б також може бути різним.

Для побудови маршрутів використовується теорія графів, що є самостійним розділом математики, що вивчає властивості графів. Наочно граф можна уявити як геометричну конфігурацію, яка складається з точок сполучених лініями.

Роком виникнення теорії графів одностайно вважається рік 1736, коли Леонард Ейлер опублікував розв’язок так званої задачі про кенігсберзькі мости, а також знайшов загальний критерій існування ейлерового циклу в графі. Отримання суттєвих результатів у цій галузі датують серединою XIX століття. Однак початок проведення активних систематичних досліджень та становлення теорії графів як окремого авторитетного розділу сучасної математики відбулося ще майже 100 років по тому, тобто всередині XX століття. Саме з цього часу граф стає однією з найпоширеніших і найпопулярніших математичних моделей у багатьох сферах науки і техніки. Картинка у вигляді набору точок на площині та ліній,проведених між деякими з них, стала зручною і наочною формою зображення найрізноманітніших об’єктів, процесів та явищ.

Із суто формальної точки зору граф можна розглядати як один з різновидів алгебраїчної системи (а саме, як модель), а отже, і всю теорію графів (як розділ сучасної алгебри). Справді, результати та методи алгебри широко використовуються в теорії графів. Однак за останні півстоліття активного інтенсивного та екстенсивного розвитку і поширення електронних обчислювальних машин, теорія графів виробила свою достатньо специфічну власну проблематику і методологію. На сьогодні теорія графів є однією зі складових математичного апарату кібернетики та картографії,важливим розділом дискретної математики.

1.3. Опис існуючих методів і рішень проектування бази даних

Поняття бази даних (далі – БД) як засобу опрацювання інформації з'явилось на етапі застосування комп'ютерних систем у сфері бізнесу, фінансів та управління. Особливостями сучасних баз даних є:

- структурування і класифікація даних за певною множиною формальних та змістовних ознак;

- наявність спеціального програмного забезпечення - системи управління базами даних;

- незалежність методів і засобів зберігання даних (технологій фізичного рівня) від методів та засобів опрацювання та сприйняття даних (технологій логічного рівня);

- незалежність способів подання і оброблення даних від їхнього змісту та галузі застосування;

- незалежність методів і процедур опрацювання від обсягів даних;

- можливість застосування однієї бази даних для розв'язання різноманітних задач.

Отже, база даних - це множина взаємопов'язаних даних, об'єднаних спільним середовищем зберігання, спільним застосуванням, єдиною формою подання, єдиними методами і засобами керування [6].

Головним завданням БД є збереження великих обсягів інформації (записи даних) та надання доступу до неї користувачеві або ж прикладній програмі. Таким чином, база даних складається з двох частин: збереженої інформації та системи управління нею. З метою забезпечення ефективності доступу записи даних організовують як множину фактів (елементи даних). Отже, до головних властивостей БД належать такі:

− цілісність означає, що в будь-який момент часу відомості в БД повинні бути несуперечливі;

− безпека означає, що виконується захист даних від санкціонованого і несанкціонованого доступу;

− відновлюваність означає можливість відновлення БД після збоїв роботи системи [6].

Основою бази даних є модель даних — фіксована система понять і правил для представлення даних структури, стану і динаміки проблемної області в базі даних [7]. Найбільш поширені такі моделі баз даних: ієрархічна, мережева, реляційна та об'єктно-орієнтована.

Ієрархічна модель даних будується на основі принципу підпорядкованості поміж елементами даних і представляє собою деревоподібну структуру, яка складається із вузлів (сегментів) і дуг (гілок). Дерево у ієрархічній структурі упорядковане за існуючими правилами розташування його сегментів і гілок [8]. Приклад графічного зображення простої ієрархічної схеми даних наведений на рисунку 1.2. До переваг ієрархічної моделі належать:

- розвинені низько рівневі засоби керування даними в зовнішній пам’яті;
- можливість побудови ефективних прикладних систем;
- економне використання пам’яті.

Слід зазначити, що ієрархічна модель має також певні недоліки, оскільки не всі предметні області мають чітко виражену ієрархічну структуру. А саме:

- асиметрія пошуку за симетричними даними;
- залежність між пошуком та відповідністю ієрархічної структури наявним зв’язкам у предметній області;
- низький рівень мови запитів і маніпулювання даними;
- аномалії додавання, видалення та оновлення даних;
- дублювання даних.

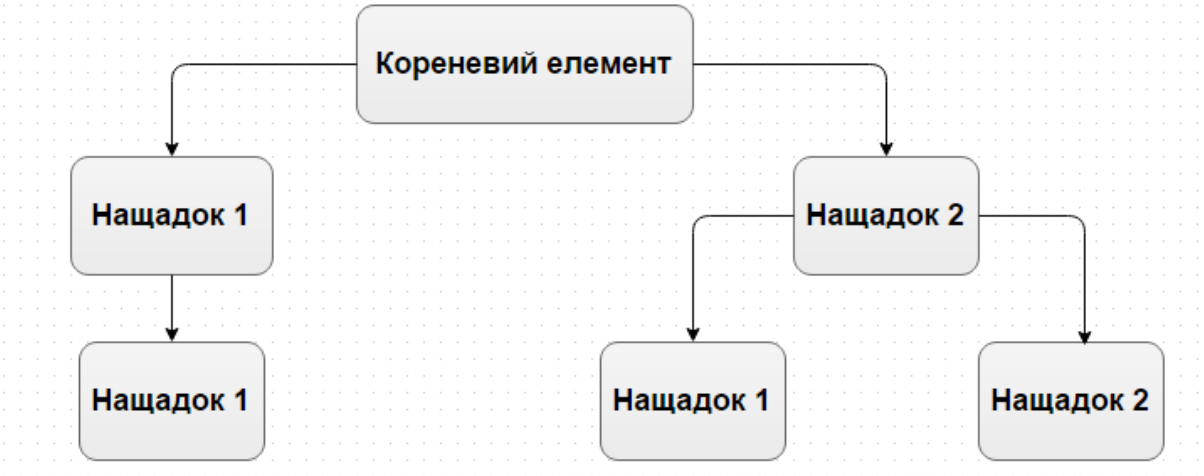

Рис. 1.2. Схема відношень між об’єктами в ієрархічній БД

Крім згаданих основних недоліків ієрархічної моделі слід зазначити також складність реалізації гнучких механізмів захисту даних, цілісності та несуперечливості й “дружніх” інтерфейсів користувача [8].

Мережева модель даних представляє собою орієнтований граф з іменованими вершинами і дугами. Вершини графа - записи, які представляють собою іменовану сукупність логічних взаємозв'язаних елементів даних або агрегатів даних. Під агрегатом даних розуміють сукупність елементів даних, які є в середині запису. Для кожного типу записів може бути кілька екземплярів конкретних значень його інформаційних елементів. Два записи, взаємозв'язані дугою, створюють набір даних [9]. Така структура набагато гнучкіша і виразніша від попередньої і придатна для моделювання більш широкого класу завдань (рис.1.3.).

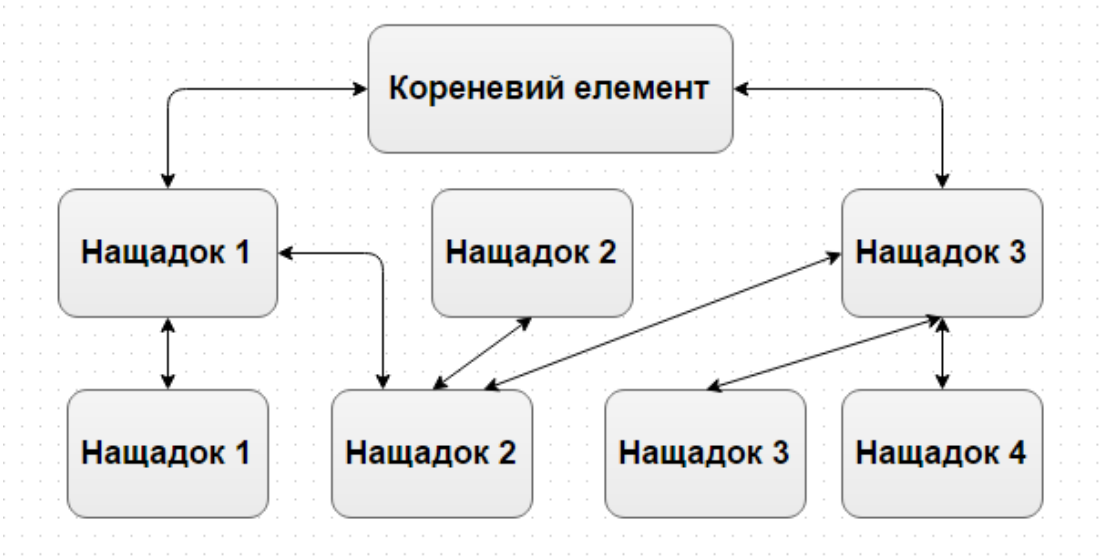

Рис. 1.3. Схема відношень між об’єктами в мережевій БД

Реляційна модель даних (рис. 1.4.) являє собою набір двомірних плоских таблиць, що складаються з рядків і стовпців. Первинний документ або лінійний масив являє собою плоску двомірну таблицю. Така таблиця називається відношенням, кожний стовбець - атрибутом, сукупність значень одного типу (стовпця) – доменом, а рядка – кортежем. Таким чином, стовпці таблиці являються традиційними елементами даних, а рядки – записами. Таблиці (відношення) мають імена. Імена також присвоюються і стовпцям таблиці. Кожний запис відношення має ключ [9].

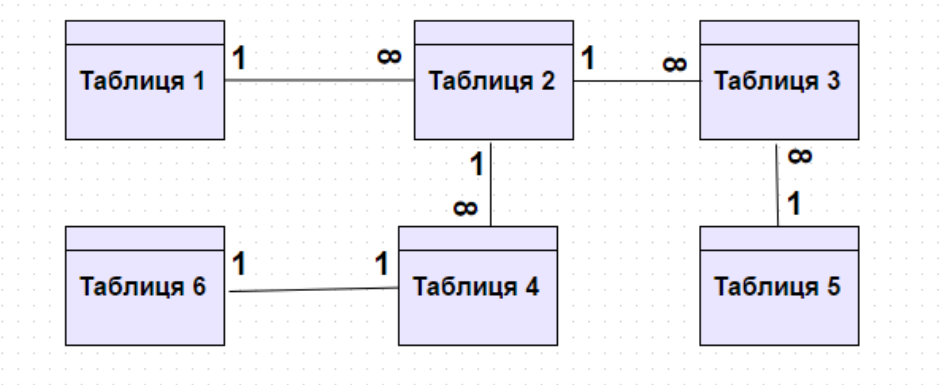

Рис. 1.4. Схема відношень між об’єктами в реляційній БД

Основною відмінністю пошуку даних в ієрархічних, мережевих і реляційних базах даних є те, що ієрархічні і мережеві моделі даних здійснюють зв'язок і пошук між різними об’єктами за структурою, а реляційні – за значенням ключових атрибутів.

Найбільш популярними СКБД для створення реляційних баз даних є MicrosoftAccess, FoxPro, Paradox, Clipper, MicrosoftSqlServer, MySql та ін.

Об'єктна СКБД ідеально підходить для інтерпретації складних даних, на відміну від реляційних СКБД, де додавання нового типу даних досягається ціною втрати продуктивності або за рахунок різкого збільшення термінів і вартості розробки додатків. Об'єктна база, на відміну від реляційної, не вимагає модифікації ядра при додаванні нового типу даних. Новий клас і його екземпляри просто надходять у зовнішні структури бази даних. Система управління ними залишається без змін.

Об'єктно-орієнтована база даних (ООБД) - база даних, в якій дані оформлені у вигляді моделей об'єктів, що включають прикладні програми, які управляються зовнішніми подіями. Результатом поєднання можливостей (особливостей) баз даних і можливостей об'єктно-орієнтованих мов програмування є об'єктно-орієнтовані системи управління базами даних (ООСУБД). ООСУБД дозволяють працювати з об'єктами баз даних також, як з об'єктами у програмуванні в об'єктно-орієнтованих мовах програмування. ООСУБД розширює мови програмування, прозоро вводячи довготривалі дані,

управління паралелізмом, відновлення даних, асоційовані запити й інші можливості.

Об'єктно-орієнтовані бази даних звичайно рекомендовані для тих випадків, коли потрібна високопродуктивна обробка даних, які мають складну структуру [10].

В проектуванні баз даних розрізняють два варіанти розміщенні даних: локальний і віддалений. Локальний варіант передбачає розміщення даних безпосередньо на комп'ютері користувача. В цьому випадку користувач володіє монопольним доступом до даних, доступ до даних інших користувачів неможливий. Віддалений варіант передбачає розміщення даних поза комп'ютерами користувачів – на файловому сервері мережі або на спеціально виділеному комп'ютері [11].

На сьогодні існує велика кількість різноманітних засобів роботи з базами даних. Одними з найпоширеніших систем керування базами даних є продукти компаній Microsoft (MSAccess, MSSQLServer) та Oracle (MySQL, OracleDatabase). Попри конкуренцію цих двох фірм їх продукти зайняли різні ніші в сфері розробки програмного забезпечення.

MySQL - система керування базами даних, щорозповсюджується як за вільною ліцензією GNU другого покоління, так і під власною комерційною ліцензією. Входить до складу значної кількості серверів: WAMP, AppServ, LAMP і портативні збірки серверів Денвер, XAMPP. Зазвичай MySQL використовується в якості сервера, до якого звертаються локальні або віддалені клієнти, але в дистрибутив входить бібліотека, що дозволяє працювати з автономними програмами. Завдяки таким властивостям даного продукту він здобув популярність при розробці сайтів, веб-додатків [12]. Для некомерційного використання MySQL є безкоштовною. Можливості сервера MySQL:

- простота у встановленні та використанні;
- підтримується необмежена кількість користувачів, що одночасно працюють із БД;
- кількість рядків у таблицях може досягати 50 млн.;

- висока швидкість виконання команд;
- наявність простої та ефективної системи безпеки [13].

Microsoft SQL Server — система управління базами даних, особливістю якої є мова запитів Transact-SQL, створена спільно Microsoft та Sybase. Transact-SQL є реалізацією стандарту ANSI/ISO відносно мови структурованих запитів (SQL) з розширеннями. Використовується для роботи з базами даних від персональних до великих баз даних масштабу підприємства [14].

При взаємодії з мережею Microsoft SQL Server і Sybase ASE використовують протокол рівня додатків під назвою Tabular Data Stream (TDS, протокол передачі табличних даних). Протокол TDS також був реалізований у проекті FreeTDS з метою забезпечити різні додатки можливістю взаємодії з базами даних Microsoft SQL Server і Sybase.

Для забезпечення доступу до даних Microsoft SQL Server підтримує Open Database Connectivity (ODBC) - інтерфейс взаємодії додатків з СКБД. SQL Server надає можливість підключення користувачів через веб-сервіси, що використовують протокол SOAP. Це дозволяє клієнтським програмам, не призначеним для Windows, кросплатформно з'єднуватися з SQL Server [13].

Компанія Microsoftмає свої аналоги технологій для роботи з БД:

– ADO.NET;

– LINQ to SQL;

– ADO.NET Entity Framework.

ADO.NET - це набір класів, що надають служби доступу до даних програмісту, що працює на платформі .NET Framework. ADO.NET має широкий набір компонентів для створення розподілених додатків, що сумісно використовують дані. Це є невід’ємною частиною платформи .NET Framework, що надає доступ до реляційних даних, XML-даних та даних додатків. ADO.NET задовольняє різні потреби розробників, включаючи створення клієнтських додатків баз даних, а також бізнес-об’єктів середнього рівня, що використовуються додатками, мовами, браузерами [15].

LINQ to SQL є компонентом .NET Framework, починаючи з версії 3.5, що надає інфраструктуру підчас виконання для керування реляційними базами

даних як об’єктами. Реляційні дані відображаються в вигляді колекції двохвимірних таблиць, в яких загальні стовбці зв’язують таблиці одна з одною. В LINQ to SQL модель даних реляційної бази даних ставиться у відповідність об’єктній моделі, що виражається в мові програмування розробника [16].

Можна виділити основні переваги і недоліки технологій ADO.NET і LINQ.

Головні перевагиADO.NET:

- дозволяє працювати з різними джерелами даних, розробник застосування може і не знати, яке СКБД буде у бази даних, з яким працюватиме його застосування, йому досить буде поміняти постачальника даних;

- наявність автономних об'єктів дозволяють підвищити продуктивність і понизити навантаження на СКБД.

Головні недоліки ADO.NET:

- обмежені можливості для роботи із запитами одного з найбільш широко використовуваного компоненту - DataSet.

- необхідність написання SQLкоду, або виконання процедур, що зберігаються на сервері, що збільшує вірогідність помилок в синтаксисі.

Головні переваги LINQ:

- за певних умов дозволяє провести глибоку інтеграцію бази даних і застосування;

- значно прискорює процес написання запитів до бази даних, за рахунок, розширення синтаксису мов C# і VisualBasic;

- надає компоненти для зручної роботи не тільки з базами даних, але і об'єктами, XML-документами і т.д.

Головні недоліки LINQ:

- довший час виконання, в порівнянні з ADO.NET, за рахунок одного додаткового шару.

- менший контроль над виконанням запитів.

В основі Entity Framework лежить об’єктна модель даних (EDM). У моделі EDM визначаються типи сутностей, відносини і контейнери, а розробник взаємодіє з усім цим за допомогою коду. Платформа Entity

Framework будує відповідності між згаданими елементами і схемою даних, яку надає реляційна база даних. Модель EDM платформа Entity Framework використовує через XML, в якому визначається концептуальна модель додатку. Визначатися вона може як самостійно, так і разом з кодом XML, визначальним схему сховищ, і з кодом XML, визначальним відношення між ними [18].

Тобто, платформа Entity Framework дозволяє розробникам створювати додатки для доступу до даних, що працюють з концептуальною моделлю, а не відразу з реляційною схемою зберігання. Метою є зменшення об’єму коду і зниження витрат на супроводження додатків, орієнтованих на обробку даних.

У Entity Framework розробники дістають можливість працювати з даними, представленими у формі об'єктів, що відносяться до конкретних доменів, і властивостей, таких як клієнти і їх адреси, не будучи вимушеними звертатися до базових таблиць і стовпців бази даних, де зберігаються ці дані [17].

1.4. Огляд засобів для програмної реалізації веб-додатка

Веб-додаток отримує запит від клієнта і виконує обчислення, після цього формує відповідь (веб-сторінку) і надсилає її клієнту через мережу з використанням протоколу HTTP. Веб-додаток може бути клієнтом інших служб, наприклад, бази даних або іншого веб-додатку, розташованого на іншому сервері.

Тенденція бурхливого розвитку мережі Інтернет - зберігається протягом останніх років. База даних для системи збору даних з подальшим її аналізом повинна зберігатись централізовано, що не можливо без використання клієнт-серверної архітектури. Клієнти не залежать від конкретної операційної системи користувача, тому веб-додатки є багатоплатформовими сервісами. Найважливішою перевагою веб-додатків є простота розгортання та оновлення (не потрібно перевстановлювати програмних модулів на робочих станціях користувачів).

Для розробки веб-ресурсу можна використовувати безліч технологій та засобів, тут вже справа за розробником. На сьогоднішній день одними з

найпоширеніших засобів є: Asp.Net MVC, WCF, Json.net, AngularJs, jQuery, Bootstrap та багато інших.

Microsoft .NET Framework — програмна технологія, запропонована компанією Microsoft як платформа для створення як звичайних програм, так і веб-додатків. Microsoft .NET Framework має наступні переваги:

– повна підтримка принципів ООП;

– незалежність від мови – завдяки .NET, код всіх мов, тобто VisualBasic .NET, C #, F#, J # керованого C + +, компілюється в спільну мову проміжного рівня - IL (Рис. 1.5.);

– краща підтримка динамічних веб-сторінок;

– ефективний доступ до даних - набір компонентів .NET, відомий під загальною назвою ADO.NET, надає ефективний доступ до реляційних баз даних і широкої різноманітності інших джерел даних;

– поділ коду - середовище .NET повністю змінила спосіб поділу коду між додатками, ввівши концепцію збірки (assembly), яка замінила традиційні бібліотеки DLL;

– підвищена безпека - кожна збірка також може містити вбудовану інформацію безпеки, яка точно описує, кому і яким категоріям користувачів або процесів які методи яких класів дозволено викликати, що забезпечує дуже високу ступінь контролю за тим, як можуть використовуватися встановлені збірки;

– підтримка Web-служб. .NET пропонує повністю інтегровану підтримку розробки Web-служб – аналогічно з створенням додатків будь-якого іншого типу [19].

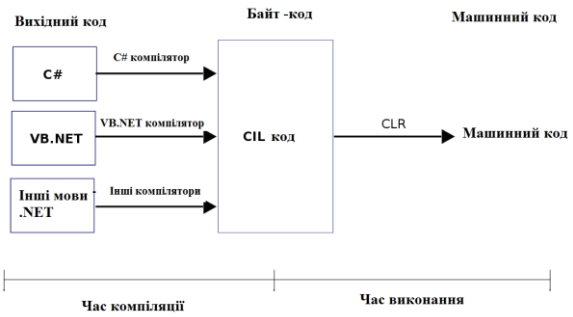

Рис. 1.5. Принципова схема роботи Microsoft .NET Framework

Модель-представлення-контролер(англ. Model-view-controller, MVC) - архітектурний шаблон (рис.1.6), який використовується під час проектування та розробки програмного забезпечення. Цей шаблон поділяє систему на три частини: модель даних, вигляд даних та керування. Застосовується для відокремлення даних (модель) від інтерфейсу користувача (представлення) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача [20].

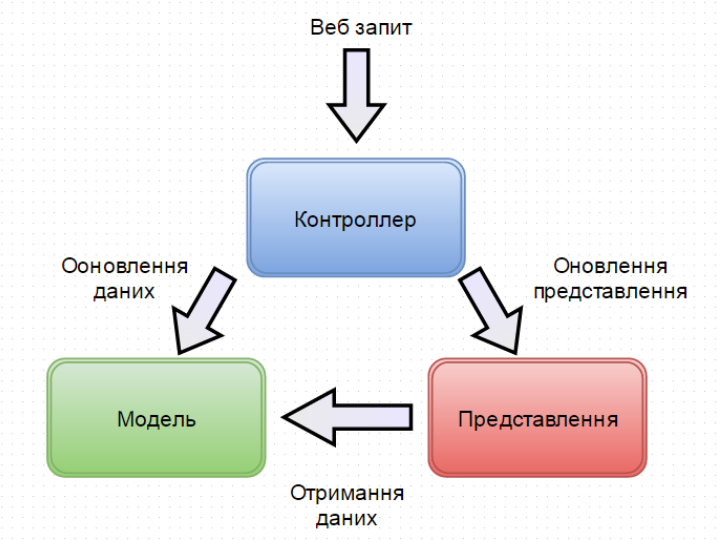

Рис. 1.6. Архітектурний шаблон MVC

Платформа ASP.NET MVC є альтернативою схемі веб- форм ASP.NET при створенні веб-додатків. ASP.NET MVC - легковагова платформа відображення з широкими можливостями тестування і, подібно додаткам на основі веб-форм, інтегрована з існуючими функціями ASP.NET. Платформа MVC визначається в збірці System.Web.Mvc[21].

За допомогою технології WCF (Windows Communication Foundation), можна забезпечити уніфікований доступ до інформації та бізнес логіки, незалежно від операційної системи. Та розробляти системи за допомогою будь-яких сторонніх технологій та практик. WCF являє собою набір клієнтських бібліотек, що дозволяють застосункам на базі відкритої платформи .NET Core взаємодіяти з сервісами WCF, відправляючи повідомлення між сервісами в асинхронному режимі. WCF робить можливою

побудову безпечних, надійних і транзакційних систем через спрощену уніфіковану програмну модель міжплатформової взаємодії. Концептуальна схема роботи WCF сервісів, зображена на рисунку 1.7.

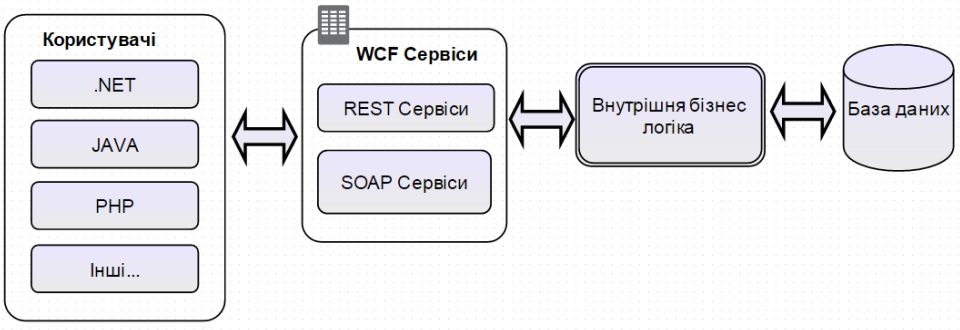

Рис. 1.7. Концептуальна схема роботи WCF.

Комбінуючи функціональність поточних технологій .NET з розробки розподілених застосувань WCF надає єдину інфраструктуру розробки, що підвищує продуктивність і знижує витрати на створення веб-служб.

ASP.NET Identity - система авторизації і аутентифікації в .NET. Крім логінів і паролів система авторизації і аутентифікації пропонує ще такий компонент для розмежування доступу, як ролі. Ролі дозволяють створити групи користувачів з певними правами і залежно від приналежності до тієї чи іншої групи, розмежувати доступ до ресурсів аплікації [22].

JSON (JavaScript Object Notation) —текстовий формат обміну даними. JSON базується на тексті, і може бути з легкістю прочитаним людиною. Формат дозволяє описувати об'єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією). JSON виступає як альтернатива XML під час асинхронної передачі структурованої інформації між клієнтом та сервером. Якщо говорити про веб-застосунки, JSON доречний в задачах обміну даними як між браузером і сервером , так і між самими серверами (програмні HTTP-інтерфейси). Формат JSON так само добре підходить для зберігання складних динамічних структур в реляційних базах даних або файловому кеші[23].

AngularJS - JavaScript-фреймворк з відкритим програмним кодом, який розробляє Google. Призначений для розробки односторінкових застосунків, що складаються з однієї HTML сторінки з CSS і JavaScript. Його мета — розширення браузерних застосунків на основі шаблону Model-View-Whatever (MVW), а також спрощення їх тестування та розробки. Фреймворк працює зі сторінкою HTML, що містить додаткові атрибути і пов'язує області вводу або виводу сторінки з моделлю, яка являє собою звичайні змінні JavaScript. Значення цих змінних задаються вручну або отримуються зі статичних або динамічних JSON-даних [24].

За минулі роки бібліотека зазнала безліч змін і на поточний день містить функціонал, корисний для максимально широкого кола завдань. Вона має невеликий розмір і не засмічує глобальний простір імен тривіальними ідентифікаторами. Приголомшливі можливості механізму селекторів, що дозволяють легко отримати доступ до будь-якого елементу об'єктної моделі документа, зробили бібліотеку jQuery дуже популярною.

В основному бібліотеку jQuery використовують для прикраси сайту, яка вже давно замінила flash. Якщо раніше жвавий сайт можна було зробити тільки за допомогою флеш елементів, то в даний час все це можна зробити за допомогою jQuery [25].

Bootstrap є найбільш популярним HTML, CSS, та JS фреймворком для зовнішнього вигляду та інтерактивності веб-сторінок. Bootstrap допомагає швидше та простіше розробляти зовнішний вигляд веб-сторінок. Він підходить для людей з будь-яким рівнем досвіду, для пристроїв будь-яких форматів, та проектів будь-якого розміру. Bootstrap автоматично адаптує перегляд сторінок під декілька розмірів екранів. Bootstrap спроектовано для найкращої роботи в нових браузерах, тобто старі браузери можуть відображати стилі по-іншому, або ж можуть мати неповну функціональність при показі певного компонента. Елементи Bootstrap повинні досить добре виглядати та поводитись в Chromium та Chrome для Linux, Firefox для Linux, та Internet Explorer 7, хоча ці браузери підтримуються не офіційно [27].

HTML (англ. HyperText Markup Language — Мова розмітки гіпертексту) — стандартна мова розмітки документів у всесвітній мережі. Більшість веб-сторінок створюються за допомогою мови HTML. Документ HTML оброблюється браузером та відтворюється на екрані у звичному для людини вигляді.

Каскадні таблиці стилів (англ. CascadingStyleSheets або скороченоCSS) — спеціальна мова, що використовується для опису сторінок, написаних мовами розмітки даних. CSS використовується авторами та відвідувачами веб-сторінок, щоб визначити кольори, шрифти, верстку та інші аспекти вигляду сторінки. Одна з головних переваг — можливість розділити зміст сторінки (або контент, наповнення, зазвичай HTML) від вигляду документу (що описується в CSS).

1.5. Висновок

У результаті огляду сучасних інструментів проектування бази даних, було представлено основі сучасні СКБД, найпоширеніші сфери їхнього застосування, переваги та недоліки. Також було розглянуто основні методології побудови бази даних, доцільність використання кожної з них та гнучкість самої структури про побудові та використанні.

Щодо програмної реалізації, то в даному розділі було представлено різні програмно-архітектурні рішення для побудови сучасних веб-додатків та сервісів, на основі яких створюються складні системи та програмні комплекси.

Також був проведений детальний аналіз існуючих програмних платформ для розробників, що забезпечують комплексні рішення для вирішення складних програмних задач різного рівня, починаючи від серверної частини, питань продуктивності та швидкодії, закінчуючи високорівневими програмними конструкціями для швидкого та зручного досягнення результату.

Окрему увагу було представлено технологіям та програмним бібліотекам, за допомогою яких відбувається розробка користувацького інтерфейсу та аспектів, що безпосередньо впливають на стиль зовнішнього відображення програмного продукту користувачеві.

РОЗДІЛ 2. ВИБІР ЗАСОБІВ ДЛЯ РОЗРОБКИ СИСТЕМИ ДЛЯ ЗБОРУ ДАНИХ ПРО ДОРОЖНІ МАРШРУТИ

2.1. Обґрунтування вибору методів та засобів для розробки системи

Серед розглянутих СКБД і технологій для розробки конструкторської бази даних було обрано MSSQL Server, та технологію Entity Framework, які в поєднанні надають широкі можливості управління даними.

Після аналізу існуючих СКБД і технологій взаємодії з базами даних, було обрано MS SQL Server як основною системою керування базами даних, Database Project як інструмент розгортання та початкового наповнення бази даних та Entity Framework як механізм програмної взаємодії.

SQL Server є найбільш інтегрованою СКБД до сімейства технологій .NET і являє собою надійну, захищену та стабільну платформу для роботи програмного забезпечення та дозволяє значно спростити розробку та підтримку такого ПЗ, знижуючи затрати часу на керування даними різних рівнів. До основних його переваг належать:

- Простота використання. Легкість в експлуатації дозволяє знизити операційні витрати і витрати на розробку рішень на основі платформи.

- Керованість. Інтуїтивно зрозумілі засоби керування та автоматизованого адміністрування допомагають ефективно управляти програмними додатками.

- Створення звітів і аналітика. За допомогою вбудованих технологій аналізу та створення звітності можна легко і швидко отримувати практичну і значиму інформацію і приймати обгрунтовані рішення [28].

За допомогою .NET Framework, є можливість створити проект бази даних (DataBase Project), що забезпечує гнучку конфігурацію та швидкий процес розгортання бази даних. Використання проекту бази даних надає такі переваги як:

- Можливість розгорнути проект бази даних з віддаленого місця розташування.

- Можливість зробити додаткові оновлення до існуючої бази даних.

- Можливість включити сценарії заздалегідь розгортання або сценарії після розгортання.

- Можливість адаптувати розгортання для декількох середовищ.

- Можливість розгорнути проект бази даних, як частину більшого, сценарію.

Entity Framework реалізує два підходи до синхронізації змін в коді і БД:

- Автоматичне оновлення. Entity Framework бере на себе задачу визначення відмінностей у версіях БД та моделі і самостійно може оновлювати БД. Цей метод корисний на початкових етапах розробки, коли модель БД часто змінюється.

- Міграції. Entity Framework дає можливість вимкнути опцію автоматичного оновлення БД, після чого за оновленням БД повинен слідкувати розробник. Проте і в цьому випадку Entity Framework забезпечує розробника потужним функціоналом, що дає змогу порівнювати поточну версію БД і моделі, а також генерувати скрипти оновлення БД. Фактично все, що потрібно робити розробнику, - це слідкувати за правильним і послідовним виконанням скриптів міграції. Цей метод використовується коли перестворення БД і втрата даних є критичним і тому необхідно чітко контролювати процес оновлення БД.

Тому можна виділити такі переваги додатків Entity Framework:

1.Додатки можуть працювати з концептуальною моделлю в термінах предметної області – в тому числі з типами, що унаслідуються, складними комбінованими елементами та зв’язками.

2.Додаток є вільним від жорстких залежностей і конкретного ядра СКБД чи схеми зберігання

3.Зіставлення між концептуальною моделлю і схемою, специфічною для конкретного сховища можуть мінятися без зміни коду додатку.

4.Декілька концептуальних моделей можуть бути узгоджені з єдиною схемою зберігання і навпаки – одна модель може консолідувати декілька схем з різнотипних джерел даних.

5.Підтримка запитів LINQ забезпечує перевірку синтаксису під час компіляції для запитів до концептуальної моделі [17].

Для керування SQL Server було обрано MS SQL Server Management Studio 2012. SQL Management Studio працює з усіма версіями SQL Server, починаючи з сьомої, і підтримує всі новітні можливості SQL Server, включаючи табличні типи і параметри табличних значень, тригери входу, резервні копії з стиском і багато іншого. Програма включає в себе безліч інструментів, таких як:

- Візуальний конструктор баз даних;
- Сучасний графічний інтерфейс;
- Гнучкі інструменти для моніторингу та аналізу швидкодії;
- Широкі можливості налаштувань та набір опцій.

За допомогою інструментальних засобів для розробки програмного забезпечення – Microsoft Visual Studio 2013 і мову програмування високого рівня C#, яка спеціально призначена для роботи у середовищі Microsoft .NET Framework.

Для роботи з базами даних в середовищі MS SQL Server та середовищі розробки MS Visual Studioвикористаємо три підходи об’єктно–реляційній проекції Entity Framework, а саме Database First, Model First та Code First.

Підхід Database First найчастіше використовують коли БД вже розроблена і необхідно розробити програмну логіку для роботи з нею. Використовуючи такі інструменти Visual Studio, як Entity Framework Designer, можна згенерувати C# класи, що будуть відповідати структурі існуючої БД. Після створення проекції з метою оптимізації моделі можна змінювати зв’язки чи структуру за допомогою дизайнера (чи змінюючи XML файли проекцій). Пріоритетом в даному підході є БД – код і модель виносяться на задній план.

Підхід Model First як правило застосовується коли схема бази даних ще не реалізована і ще невідомо який СКБД буде обраний для збереження та керування даними. При використанні цього підходу створюється діаграма об’єктів та їх відношень, ця діаграма дуже нагадує схему бази даних. Єдиною і суттєвою відмінністю є те, що бази даних фактично не існує, а є лише її схема, за допомогою якої можна створити базу даних і згенерувати об’єктно-реляційну модель.

Підхід Code First дає можливість відмовитися від використання схем дизайнера чи файлів XML проекцій, натомість БД створюється на основі коду. Програмісти можуть створювати програмні класи для опису потрібних об’єктів, а Entity Framework забезпечить їх використання з БД і моделлю. Entity Framework забезпечує коректну комунікацію і роботу з БД; всі класи, які необхідні для проекції, генеруються автоматично. При використанні цього підходу конфігурація проекції БД не зчитується з XML файлу, натомість використовується об’єкт DbContext, який забезпечує представлення структури БД в пам’яті [29].

Для забезпечення «дружнього» інтерфейсу, а саме для створення веб-сторінки буде використовуватись html5 та css3, а також для забезпечення адаптивності під різні розміри екранів – Bootstrap фрейворк. Особливістю використання CSS3 є використання його як засобу опису та оформлення зовнішнього вигляду веб-сторінок, написаних за допомогою мов розмітки HTML і XHTML, але може також застосовуватися до будь XML-документів.

Для реалізації архітектури системи доцільно буде використати ASP.NET MVC та WCF сервіси. Оскільки, платформа ASP.NET MVC базується на взаємодії трьох компонентів: контролера, моделі і представлення. Контролер приймає запити, обробляє користувацьке введення даних, взаємодіє з моделлю і представленням і повертає користувачу результат обробки запиту. Сама модель являє шар, що описує логіку організації даних у додатку. Модель представлення отримує дані з контролера і генерує елементи користувацького інтерфейсу для відображення інформації. В свою чергу WCF сервіс, буде забезпечувати гнучкість і незалежність від операційної системи та програмної платформи, що дозволить в майбутньому розробити різні додатки та інтерактивні портали, незалежно від технологій використання.

Важливим етапом при розробці будь-якого веб-додатку є процес авторизації, який найдоцільніше в сімействі технологій .NET Framework реалізовувати за допомогою ASP.NET Identity. Ця технологія являє собою готовий захищений механізм авторизації та автентифікації, і є тісно інтегрованим в сучасні програмні платформи і рішення від Microsoft.

2.2. Концептуальне та логічне моделювання предметної області

Для подальшої реалізації програмного продукту, необхідно провести концептуально моделювання бази даних та предметної області. Концептуальне проектування бази даних не залежить від подробиць її реалізації, таких як тип обраної СКБД, набір створюваних прикладних програм, використовувані мови програмування, тип обраної обчислювальної платформи, а також від будь-яких інших особливостей фізичної реалізації [30].

Одним з засобів формалізованого представлення предметної області (ПО) є модель «сутність-зв'язок», що являє собою діаграму об'єктів і їхніх залежностей. Моделювання предметної області базується на використанні ER-діаграм. Базовими поняттями моделі є сутність, зв'язок та атрибут.

Перш за все необхідно визначити основні типи сутностей. Після проведення аналізу предметної області, були виявлені 2 основні групи сутностей в системі:

До першої групи відносяться такі сутності:

- Користувач;
- Поля з яких складається форма відгуку;
- Інформація про форму відгуку;
- Інформація про поле в формі відгуку;
- Населений пункт та його регіон;
- Сегмент шляху;
- Маршрут;
- Інформація про налаштування;

До другої групи відносяться такі сутності:

- Переклади населених пунктів;
- Переклади статичних елементів інтерфейсу;
- Переклади динамічних елементів інтерфейсу;
- Мова

Нижче детальніше розглянуто сутності, що беруть участь в побудові системи для збору даних про дорожні маршрути.

Атрибутами користувача (рис. 2.1.) є ім'я, пароль та тип користувача.

Users
Id
Name
Password
User Type

Рис. 2.1. Сутність «Користувач»

Частина відгуків складається з чотирьох різних сутностей, а саме:

- Модель відгуку (рис. 2.2.) – відповідає за збереження метадати про відображення відгуку і володіє такими атрибутами як ім’я моделі, HTML та javascript код.
- Налаштування моделі відгуку (рис. 2.3.) – відповідає за збереження даних про налаштування моделі відгуку, та володіє такими атрибутами як обов’язковість заповнення, порядковий номер, переклад, статистичність даних в відгуку.
- Дані відгуку (рис. 2.4.) – відповідає за збереження фактичних даних відгуку, тобто інформацію що ввів респондент.
- Відгук (рис. 2.5.) – сутність, яка поєднує в собі всі вище зазначені сутності, та являє собою комплексний об’єкт відгуку. Містить такі атрибути як час отримання відгуку, приналежність до сегменту маршруту та інформацію про користувача який залишив відгук.

Кожна з описаних вище сутностей виконує свою частину обов’язків по збереженню та опрацюванню даних.

Feedback Model
Id
HtmlCode
JavascriptCode
Model Name

Рис. 2.2. Сутність «Модель відгуку»

Feedback Item
Id
Mandatory
Sort Number
Is Numeric
Model
Translation

Рис. 2.3. Сутність «Налаштування моделі відгуку»

Feedback Value
Id
Value
Feedback Id
Feedback Item

Рис. 2.4. Сутність «Дані відгуку»

Feedback
Id
Submit Time
Route Node
User

Рис. 2.5. Сутність «Відгук»

Атрибутами сутності «Населений пункт» (рис. 2.6.) є її назва та інформація про сутність «Регіон» (рис. 2.7.), атрибутом якої є назва.

CityNode
Id
Name
Region

Рис. 2.6. Сутність «Населений пункт»

Region
Id
Name

Рис. 2.7. Сутність «Регіон»

Атрибутами сутності «Сегмент шляху» (рис. 2.8.) є ідентифікатор початкового та кінцевого населених пунктів, щодо самого шляху, сутності «Маршрут» (рис. 2.9.), то тут атрибутами будуть хеш значення маршруту, перелік всіх точок маршруту, точки початку та кінця маршруту, кількість населених пунктів в маршруті, дата коли був маршрут був зареєстрований.

Route Node
Id
Origin City Node
Destination City Node

Рис. 2.8. Сутність«Сегмент шляху»

Trek
Id
Hash
Track
Origin City Node
Destination City Node
Nodes Count
Trek Date

Рис. 2.9. Сутність«Маршрут»

Атрибутами сутності «Інформація про налаштування» (рис. 2.10) будуть її назва та значення налаштувань.

Setting
Id
Setting Name
Setting Value

Рис. 2.10. Сутність «Інформація про налаштування»

До другої групи належать сутності що відповідають за збереження та відображення інформації перекладів об’єктів, а саме географічних об’єктів, статичних елементів інтерфейсу та динамічних елементів інтерфейсу. Всі вони мають схожу структуру і головні атрибути сутності, а саме ключ об’єкта перекладу, ідентифікатор мови та безпосереднє значення (переклад).

Dynamic Translations	Static Translations	
Id	Id	MapObject Translation
Value	Contol Type Key	Id
Description Value	Value	Value
Language	Language	Language
Dynamic Key	Static Key	Language Key

Рис. 2.11.Сутності перекладів динамічних, статичних елементів інтерфейсу та переклади географічних об’єктів (населених пунктів та регіонів)

Атрибутами сутності «Мова» (рис. 2.12), є назва мови та ідентифікатор чи є ця мова вибраною за замовчуванням.

Language
Id
Name
IsDefault

Рис. 2.12.Сутність «Мова»

Дана сутність фігурує в вище описаних сутностях перекладу і в деяких інших сутностях в яких необхідно зберігати інформацію про мову відображення елементів на інтерфейсі користувача.

В результаті було отримано концептуальну модель системи для збору даних про дорожні маршрути (рис. 2.13).

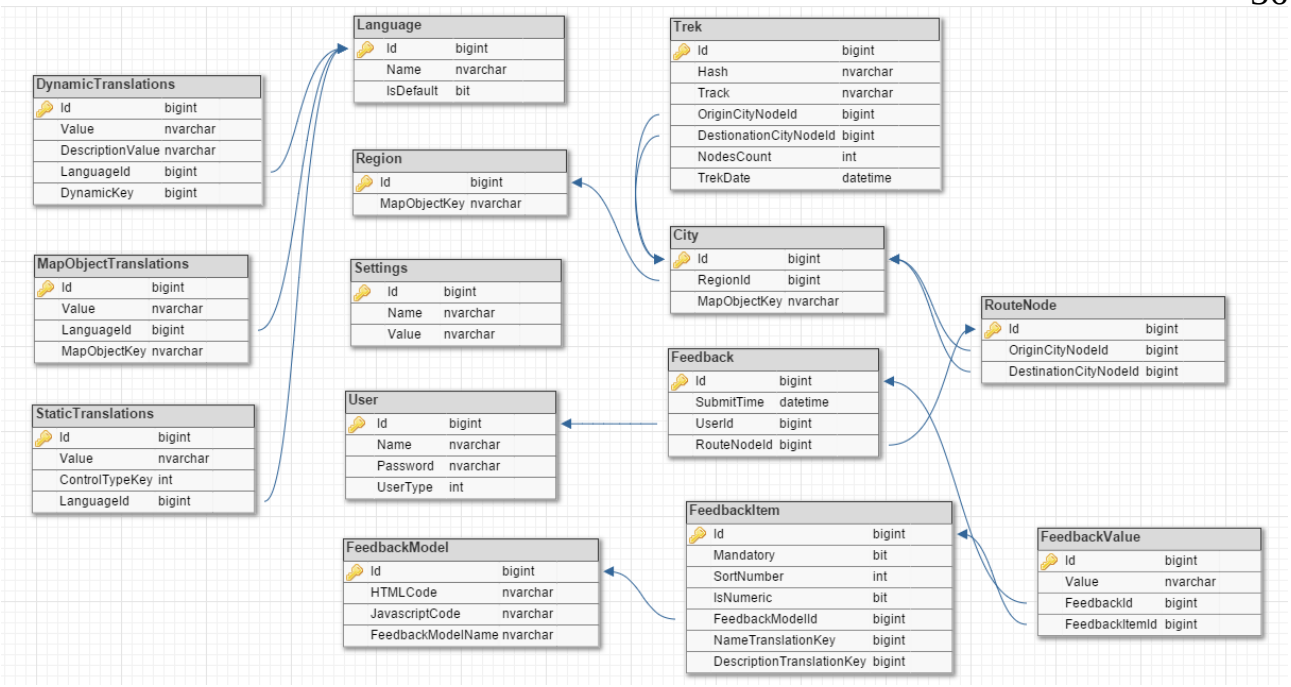

Рис. 2.13. Концептуальна модель

На схемі можна помітити що деякі з зв’язків задано неявним чином (наприклад всі посилання на переклади), це створює більші складнощі при вибірці даних, але робить структуру бази даних більш гнучкою і забезпечує уникання зайвих обмежень і зв’язків.

Створена концептуальна модель даних є початковою точкою для фази логічного проектування бази даних. Після проведення нормалізації яка дозволяє переконатися в структурній узгодженості, логічній цілісності і мінімальній збитковості прийнятої моделі даних.

Логічна модель є основним джерелом інформації на етапі фізичного проектування, а також полегшує експлуатацію і супровід БД, тому що логічна модель дозволяє наочно представити будь-які внесені в БД зміни й оцінити їхній вплив на прикладні програми і використання даних в цих програмах. В свою чергу логічна модель не прив’язана до певної технології і демонструє основні архітектурні рішення в сфері структури об’єктів та їхньої взаємодії незалежно від платформи.

2.3. Розроблення прототипу інтерфейсу користувача та загальні відомості про роботу системи

Прототипіювання є незамінною частиною програмного продукту, який призначений для використання в інтерактивному режимі. Прототип користувацького інтерфейсу об'єднує в собі всі елементи і компоненти програми, які здатні впливати на взаємодію користувача з програмним забезпеченням. Тому побудова адаптивного користувацького інтерфейсу для веб-додатку є важливим етапом процесу розробки.Для успішної побудови інтерфейсу, потрібно зважати на основні вимоги, а саме:

- відповідність завдань, що вирішуються користувачем;
- легкість застосування;
- керованість;
- відповідність очікуванням користувача;
- стійкість до помилок;
- адаптованість;
- легкість вивчення.

За допомогою середовища Balsamiq Mockups створимо прототип інтерфейсу системи. Перш за все необхідно створити сторінку для авторизації (рис. 2.14), яка скрадатиметься з двох Textbox, двох Labelта одиніїButton.

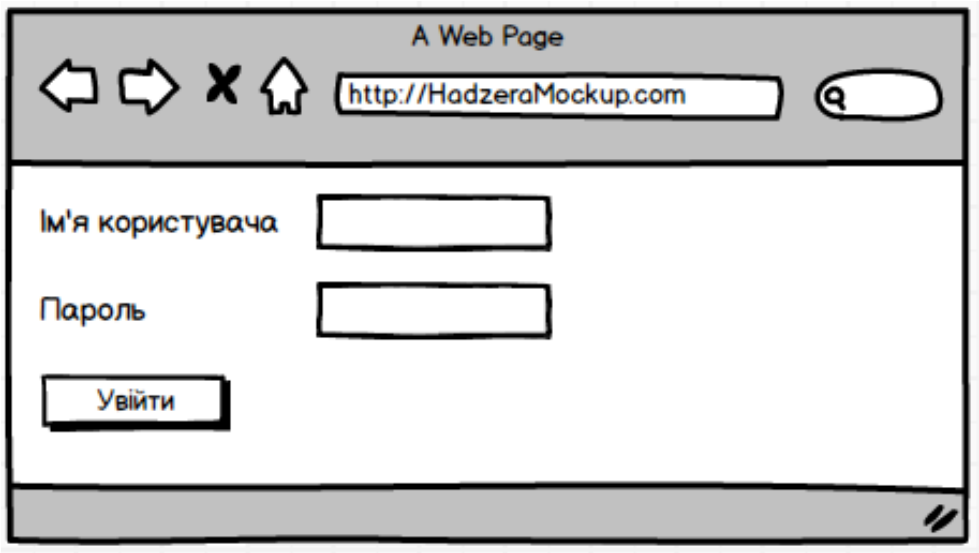

Рис. 2.14. Прототип сторінки авторизації

Наступною буде сторінка пошуку маршрутів (рис. 2.15), по замовчуванню це початкова сторінка. Вона склададиметься з двох TextBox, Button та DataGrid для відображення результатів.

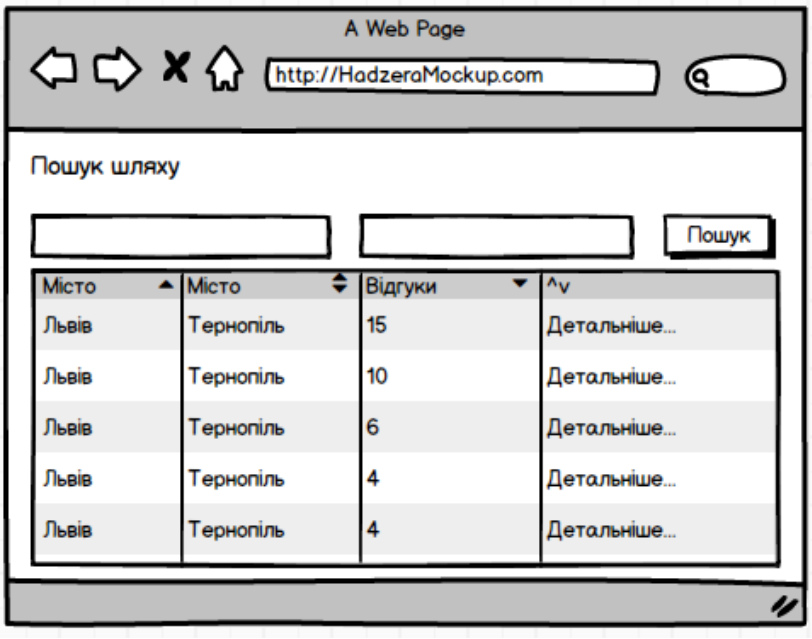

Рис. 2.15. Прототип сторінки пошуку маршрутів.

Наступною після пошуку, є сторінка детального перегляду інформації про маршрут (2.16), вона скрадатиметься з масиву елементів Button і Label ліворуч, для навігації між сегментами, там динамічною формою відгуку праворуч, в верхній частині буде знаходитись середні числові значення по відгукам за весь маршрут.

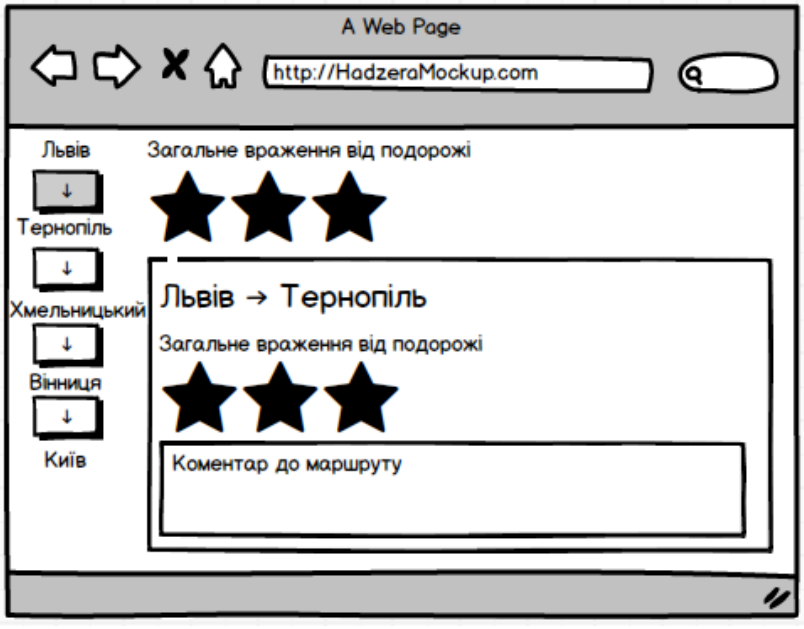

Рис. 2.16. Прототип сторінки перегляду інформації про маршрут.

Сторінка додання нового маршруту (рис. 2.17), буде містити масив з елементів Label і Button, кожний елемент відповідатиме за новий населений пункт в маршруті, також буде присутній елемент Button, натискання на який переходитиме на наступний крок створення маршруту.

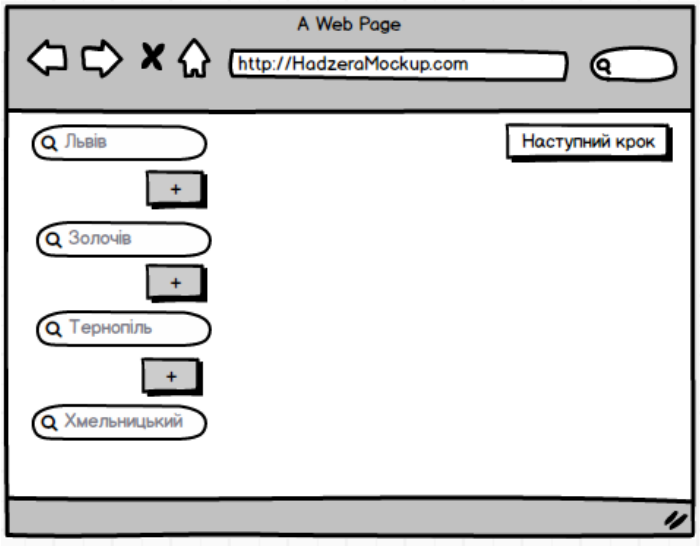

Рис. 2.17. Прототип сторінкиствореннянового маршруту.

Наступним кроком додання нового маршруту є додання нових коментарів до сегментів (рис. 2.18). Ця формабуде складатись з групи елементів Button і Label ліворуч, ця група необхідна для навігації між сегментами та динамічна форма відгуків праворуч з елементом Button, для збереження відгуку.

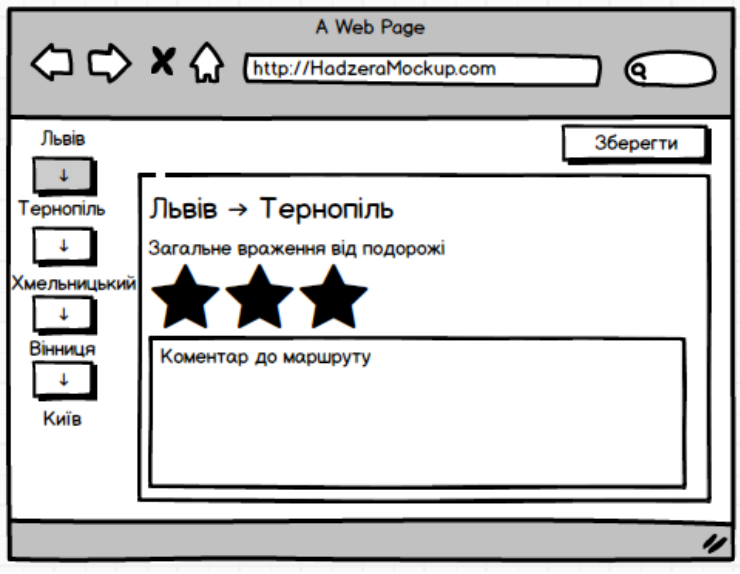

Рис. 2.18. Прототип вікна для заповнення відгуку про маршрут.

2.4. Висновок

Після проведення детального аналізу існуючих програмних засобів, платформ та інструментів розробника, було обрано такі основні засоби для реалізації бази даних та веб-додатку:

- MS SQL Server як основну систему керування базами даних;
- технологію Entity Framework для реалізації об’єктно реляційної моделі та взаємодії з базою даних;
- .Net Database Project для гнучкої конфігурації та швидкого розгортання бази даних;
- Microsoft Visual Studio 2013, як основне середовище програмної розробки;
- платформу Asp.Net MVC, як архітектурне та програмне рішення реалізації інтерактивної системи;
- бібліотеку UI Bootstrap для реалізації адаптивної розмітки веб застосування;
- Windows Communication Foundation, як механізм для кросплатформенного доступу до даних та бізнес логіки;
- бібліотеку Angular.jsдля полегшення розробки користувацького інтерфейсу та управлінням об’єктів, за допомогою концепції подвійної прив’язки даних.

Під час концептуального моделювання предметної області визначено сутності, зв’язки між ними, атрибути сутностей. На основі концептуальної моделі створено відношення, визначено потенційні первинні і вторинні ключі. Таким чином, отримано логічну модель даних, за допомогою якої можна виконати фізичне проектування бази даних.

За допомогою Balsamiq Mockups було побудовано базовий прототип інтерфейсу системи (додаток А) та визначено основний її функціонал.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ АВТОМАТИЗОВАНОГО ЗБОРУ ДАНИХ ПРО ДОРОЖНІ МАРШРУТИ

3.1. Проектування бази даних

Фізичний етап проектування забезпечує вибір раціональної структури збереження даних і методів доступу до них, виходячи з вибраних засобів розробника та способів взаємодії з конкретною СКБД [17].

На даному етапі проектування, для наглядності, доцільно використати SQL Server Management Studio, як інструмент для пректування структури бази даних. При створенні таблиць, необхідно вказувати головні атрибути полів які містяться в цих таблицях, а саме: тип даних колонки, максимальна довжина або розмір колонки, обов'язковість заповнення (Allow Nulls), приналежність до ключа, індексація поля та ін.

При виборі типу даних для поля, необхідно спланувати які дані там будуть зберігатись, в якому форматі та якого розміру. Для прикладу можна навести такі цілочисельні типи, як tinyint, діапазон якого від 0 до 255, та розмір займаємої пам'яті дорівнює 1 байту, та bіpint, діапазон цього типу складає від -2^63 до 2^63-1 і розмір пам'яті, що займає дорівнює 8 байтам. Для однієї чи двох сотень записів в таблиці це не зіграє відчутної різниці, тому що розміри сучасних носії інформації вже давно досяг сотень гігабайт та десятків петайбайт. Але якщо число записів в таблиці невпинно росте, то для порівняння різниця займаної пам'яті між 10 000 000 рядків поля tinyint та bigint складе майже 66 Мб (65.94 Мб), і це лише різниця для одного поля, але якщо таблиця містить багато таких полів, а база даних містить багато таких таблиць, то розмір пам'яті, що займає база даних буде рости в геометричній прогресії.

Якщо проблему зростання фізичного розміру бази даних можна до певної пори ігнорувати, то як відомо, розмір оперативної пам'яті має набагато менші об'єми та вищу вартість. Всі сучасні системи керування базами даних підтримують кешування даних, для швидкого доступу до сегментів, що часто використовуються, а тому питання об'ємів пам'яті стає ще гострішим, тому що

на сьогоднішній день об’єми оперативної пам’яті навіть на сучасних супер комп’ютерах вимірюються в десятках, максимум сотнях гігабайтів, що значно менше аніж на пристроях зовнішньої пам’яті. Отже при вибори типів даних, необхідно точно визначитись з розміром та форматом даних, які будуть зберігатись в тій чи іншій таблиці і дотримуватись принципу «потрібно виділяти стільки пам’яті, скільки необхідно, і ні бітом більше».

Після створення структури таблиць, необхідно задати відношення між таблицями, для цього необхідно вибрати пункт меню під назвою «Відношення» та вибрати один з трьох видів зв’язків («один до одного», «один до багатьох» та відношення«багато до багатьох», який реалізується за допомогою пари зв’язків «один до багатьох»). Інколи інструменти розробника дозволяють створювати додаткові відношення, такі як «один до нуля або одного» чи «один до нуля або багатьох», але як правило це є лише налаштуванням базових трьох відношень. Також є можливість задання зв’язків в неявному вигляді, в цьому випадку контроль над таблицями з даними та їхніми взаємозв’язками покладається на бізнес логіку клієнтського додатку.

Після створення всіх відношень між таблицями, схема даних буде мати вигляд (рис. 3.1). Більш детальний опис зв’язків між сутностями наведений на етапі концептуального проектування. Розширений варіант структури бази даних наведений в додатку Б.

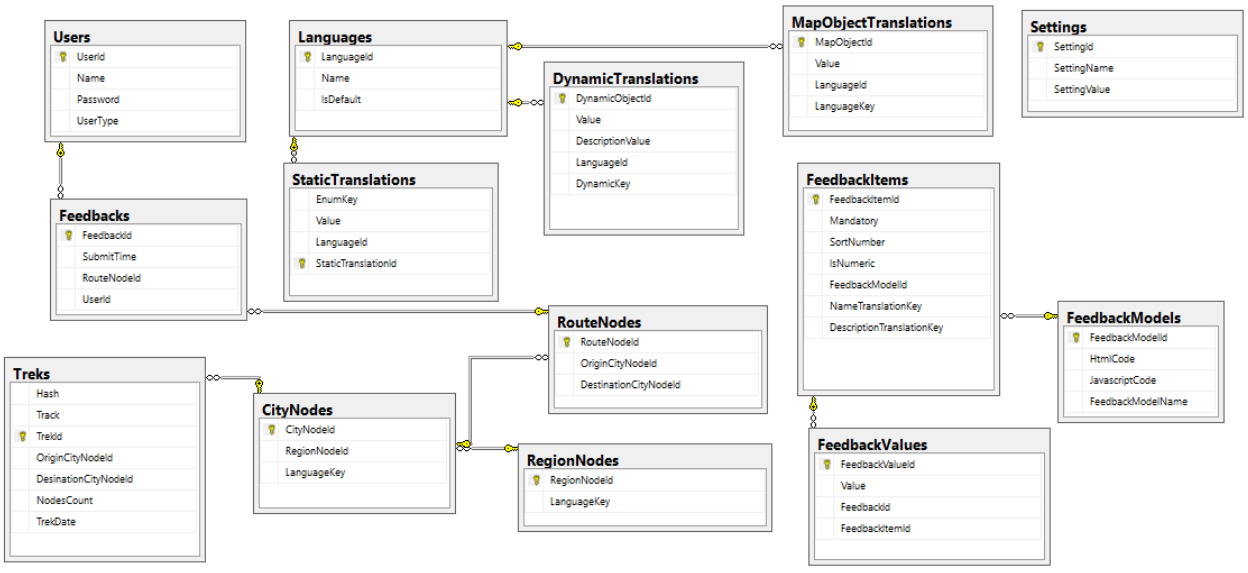

Рис. 3.1. Схема відношень в базі даних «RoadsDatabase»

3.2. Реалізація доступу до даних

Доступ з клієнтського додатку до бази даних, здійснюється за допомогою програмного провайдера та вказаного імені та адреси сервера з його налаштуваннями. Комплекс вище описаних аспектів, являє собою стрічку підключення, що є обов'язковою складовою для доступу до бази даних. З огляду на те, що було вибрано базу даних Microsoft SQL Server, провайдер матиме назву "System.Data.SqlClient», що є стандартним класом фреймоврку .NET.

Попередньо розглянувши більшість сучасних рішень побудови і використання об'єктно-реляційної моделі, було прийняте рішення використовувати технологію Entity Framework та підхід DataBase First. Для генерації об'єктно реляційної моделі даних та класів, що представляють таблиці в базі даних, необхідно скористатись майстром генерації об'єктів з готової бази даних (рис. 3.2., рис. 3.3., рис 3.4.).

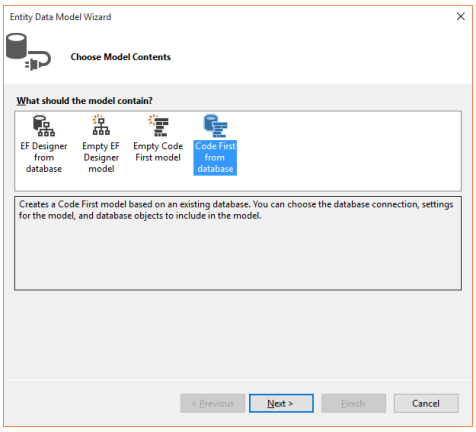

Рис 3.2. Генерація класів (крок 1).

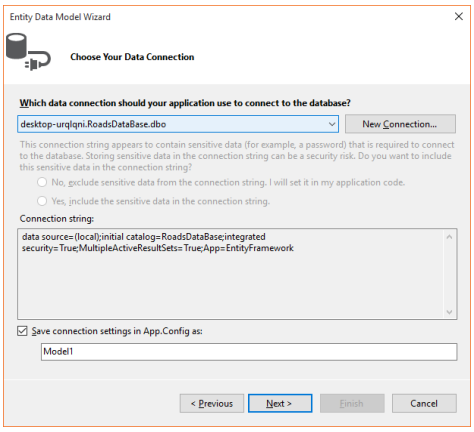

Рис 3.3. Генерація класів (крок 2).

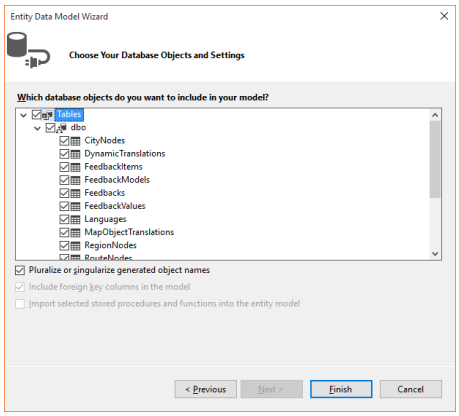

Рис 3.4. Генерація класів (крок 3). Попередній перегляд таблиць, які приймають участь в процесі генерації.

Після генерації класів, необхідно згенерувати об’єктну модель, яка наглядно демонструватиме структуру об’єктів та їхніх взаємозв’якзів. Для цього необхідно скористатись вище зазначеним методом генерації та вибрати «EF Desinger from database» як результуючий об’єкт.

Для зручності використання та більш чіткого розділення обов’язків об’єктів, було прийняте рішення створити 2 об’єктні моделі:

– Модель, що відповідатиме за переклади елементів інтерфейсу та географічних найменувань (рис. 3.5.);

– Модель, що відповідатиме за збір інформації, збереження сегментів шляху та генерацію маршрутів (рис. 3.6.).

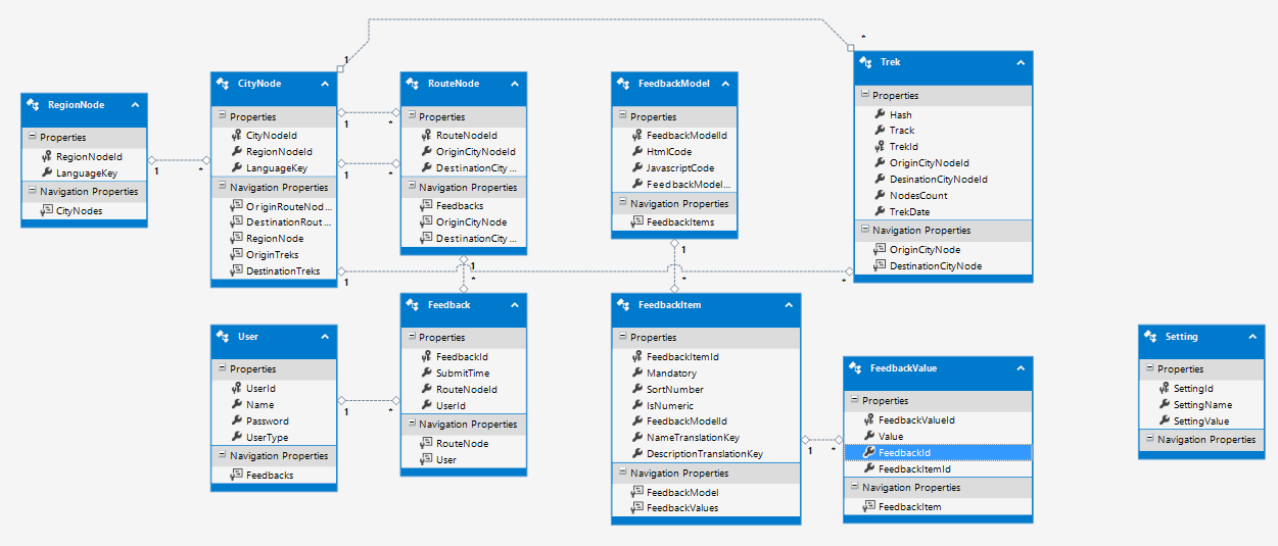

Рис. 3.5. Об’єктна модель відгуків та маршрутів бази даних «RoadsDatabase».

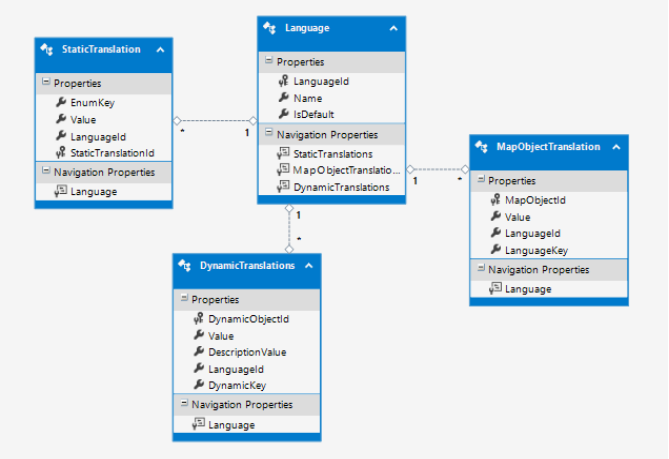

Рис. 3.6. Об’єктна модель перекладів бази даних «RoadsDatabase».

Отже, з рисунку 3.5 помітно те, що відношення «багато до багатьох», реалізується без створення проміжного зв'язувального об'єкту, що полегшує розуміння структури відношень між елементами.

Для забезпечення ізольованості клієнтського додатку від бази даних, прийнято рішення, реалізувати патерн «Репозиторій» (рис. 3.7.). Цей патерн є фасадом для доступу до бази даних. Весь код клієнтського додатку за межами репозиторію, працює з базою даних тільки через репозиторій. Таким чином, ця програмна конструкція інкапсулює в собі логіку роботи з базою даних. Основними перевагами використання цього патерну є:

- Можливість заміни технології доступу до даних не змінюючи при цьому рівень бізнес логіки;
- Можливість створення методів, які інкапсулюють певний алгоритм роботи, що зменшує кількість повторюваного коду (наприклад метод отримання списку присутніх на парі студентів та ін.);
- Спрощення структури системи доступу до бази даних, шляхом більшої декомпозиції на окремі модулі;
- Полегшення тестування, шляхом підміни реалізації певного репозиторія, тестовим «моком» з тестовими даними.

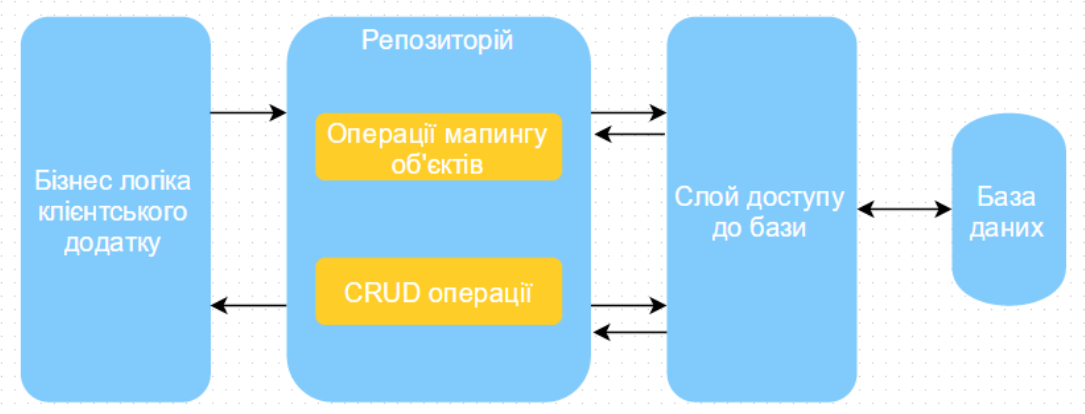

Рис. 3.7. Діаграма патерна «Репозиторій».

Взаємодія бази даних з сучасними підходами доступу програмних комплексів до даних, забезпечило зручний механізм збереження та аналізу даних про дорожні маршрути, роблячи реалізацію програмної частини системи незалежною від зовнішніх факторів та зміни деяких сервісних процесів.

3.3. Архітектура автоматизованої системи збору даних про дорожні маршрути

Побудова архітектури проекту має вирішальне значення для цілої низки аспектів реалізації, термінів, технологій, розміру команди та ін. Архітектура програмного забезпечення, являє собою структуру програми, яка містить програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними. Проектування архітектури системи, здійснюється шляхом визначення цілей системи, її вхідних і вихідних даних, декомпозиції системи на підсистеми, компоненти або модулі та розроблення її загальної структури. Правильно вибрана та повністю продумана архітектура програмного продукту, дозволить скоротити час на імплементацію, мінімізувати фінансові витрати та полегшити додання нової або модифікацію існуючої програмної логіки системи.

Існують тисячі порад щодо побудови «ідеальної» архітектури системи, але жодна з них не є ключем до успіху, а лише рекомендаційними порадами в певних ситуаціях і при певних обставинах. Не дивлячись на це, існує ряд принципів, які описують правила взаємодії об’єктів та їхньої побудови. Набір цих правил називається SOLID. Цей набір складається з п’яти основних принципів:

- Принцип єдиного обов’язку - кожен об’єкт виконувати лише один обов’язок;
- Принцип відкритості/закритості - програмні сутності повинні бути відкритими для розширення, але закритими для змін;
- Принцип підстановки Барбари Лісков - об'єкти в програмі можуть бути заміненими їх нащадками без зміни коду програми;
- Принцип розділення інтерфейсу- багато спеціалізованих інтерфейсів краще за один універсальний;
- Принцип інверсії залежностей - залежності всередині системи будуються на основі абстракцій, що не повинні залежати від деталей; навпаки,

деталі мають залежати від абстракцій. Модулівищих рівнів не залежать від модулів нижчих рівнів.

З огляду на вище описані принципи, та інші рекомендації щодо побудови, було прийняте рішення, розробити архітектуру системи, яку логічно можна було б розділити на незалежні модулі. Архітектуру програмного продукту (рис 3.8.) було побудовано таким чином, що частинам системи невідомо про внутрішню реалізацію інших частин, а публічним є лише контракт взаємодії. Це називається слабким зв'язування (Loose coupling). Цей принцип дозволяє змінювати реалізацію певних модулів системи, при цьому не змінювати логіку взаємодії з цими модулями.

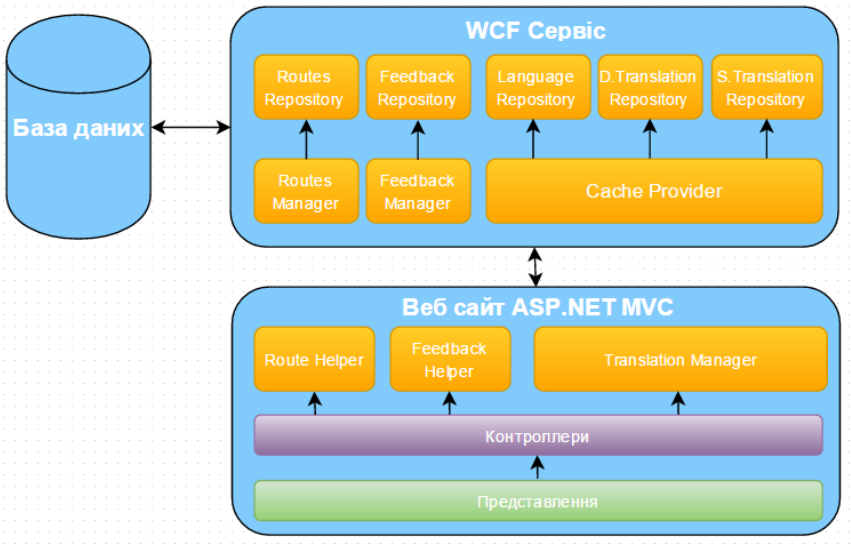

Рис. 3.8. Архітектурна схема автоматизованої системи збору даних про дорожні маршрути.

Представлене архітектурне рішення складається з трьох окремих модулів:

1. Модуль бази даних, в якому фактично зберігаються дані. Завдяки такому розділенню, в системі немає жорсткої залежності на певну технологію бази даних (MS SQL Server, MySQL та ін.), що робить зміну платформи бази даних швидкою та такою, що не потребує великої кількості змін;

2. Модуль бізнес логіки. Ця частина побудована на основі WCF сервісу, що дозволяє інтегруватись з будь-якими технологіями програмування, незалежно від платформи. Цей модуль складається з двох рівнів:

2.1. Рівень доступу до бази даних (репозиторії). Цей рівень містить логіку безпосередньої взаємодії з базою даних та перетворення даних з таблиць в об’єктні моделі. До цього рівня входять такі репозиторії як:

2.1.1. Routes Repository – репозиторій об’єктами якого є маршрути та сегменти шляху;

2.1.2. Feedback Repository – репозиторій відгуків;

2.1.3. Language, D. Translation, S. Translation Repository – репозиторії, об’єктами якого є переклади (географічних точок, динамічних та статичних елементів інтерфейсу).

2.2. Рівень бізнес логіки, який містить логіку збору даних про відгуки, сегменти шляхів та алгоритми формування маршрутів. До цього рівня входять такі елементи:

2.2.1. Routes Manager –програмний елемент, який містить логіку, що відноситься до маршрутів та їхніх сегментів;

2.2.2. Feedback Manager – програмний елемент, який містить логіку роботи з відгуками, створення та модифікацію форм для відгуків;

2.2.3. Cache Provider – програмний елемент, що містить віртуальне сховище перекладів та містить логіку взаємодії з різними типами перекладів.

3. Модуль представлення. Ця частина реалізована за допомогою технології ASP.NET MVC, хоча і може бути реалізованою за допомогою будь-якої іншої технології розробки, що підтримує протокол HTTP. Даний модуль містить спосіб представлення системи користувачу та визначає спосіб його взаємодії з системою. Дана реалізація містить такі архітектурні шари:

3.1. Шар помічників (helpers), він дозволяє створити додатковий прошарок між контроллером та основною бізнес логікою, також ізолює деталі взаємодії клієнтської та сервісної частини;

3.2. Шар контроллерів;

3.3. Шар представлень.

Кожен з описаних вище шарів, було розбито на додаткові абстракції, аби слідувати принципу інверсії залежностей та зробити систему слабко зв’язаною. Отримані абстракції були згруповані по різним проектам, що

забезпечило чітке розділення обов'язків різних частин проекту. Після проведення розділення, система складається з 11 окремих проектів (рис 3.9.).

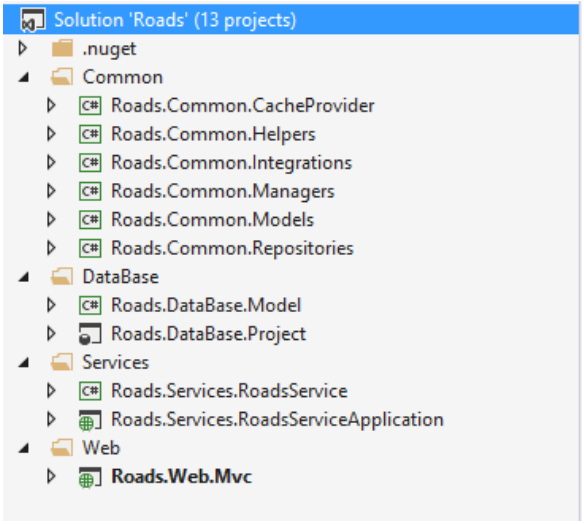

Рис. 3.9. Проектна структуру автоматизованої системи збору даних про дорожні маршрути.

Для зменшення кількості посилань на реалізації, було прийняте рішення використовувати IoC (Inversion of Control) контейнер, що являє собою програмну бібліотеку або фреймворк, яка дозволяє спростити і автоматизувати написання коду з використанням принципу інверсії залежностей на стільки, на скільки це можливо. Використання IoC контейнера полегшує контроль за налаштуванням та використанням реалізацій при різних умовах в різних частинах проекту, зберігаючи всі залежності в одному місці, що робить підтримку та модифікацію проекту набагато простішою, а можливість завантаження списку залежностей та їхніх налаштувань з файлу або з віддаленого серверу забезпечує велику гнучкість без необхідності перекомпіляції проекту.

3.4. Програмна реалізація, описалгоритмів та деталей розробки

Для збору даних про дорожні маршрути для частини, що відповідає за збереження сегментів шляху та побудови маршрутів реалізовано клас RoadsManager (додаток Е). Тобто цей клас містить логіку додання нових сегментів шляху, додання нових відгуків про сегмент та побудову маршрутів на основі створених сегментів.

Для керуваннями сегментів шляху та відгуків по них, необхідно побудувати об’єкт, який міститиме необхідну інформацію. Даний об’єкт матиме такий вигляд:

```
public class RouteNodeWithFeedbacksData
{
	public int? RouteNodeId { get; set; }
	public int OriginCityNodeId { get; set; }
	public string OriginCityNode { get; set; }
	public int DestinationCityNodeId { get; set; }
	public string DestinationCityNode { get; set; }
	public int UserId { get; set; }
	public DateTime SubmitTime { get; set; }
	public int FeedbackId { get; set; }
	public List<FeedbackValueData> FeedbackValues;
}
```

Описана вище структура містить ідентифікатор сегменту, інформацію про точку відправлення та прибуття, ідентифікатор користувача, відгуку та час коли він був залишений. Також присутня колекція фактичних даних, що ввів користувач.

Отже це є стартовим об’єктом для опрацювання класом RoadsManager, та збереження нових відгуків. Нижче описаний алгоритм додання нових відгуків в систему:

1. Ініціалізувати репозиторії для збереження відгуків та сегментів;

2. Створити текстовий об’єкт, який міститиме повний маршрут. Виділити для нього необхідний розмір за формулою: $n * 5 + (n - 1)$, де n – кількість пар «старт-фініш», кожна з яких являє собою пару ідентифікаторів населених пунктів відбуття та прибуття (наприклад Львів - Тернопіль);

3. Для кожної з пар «старт-фініш» виконати наступні дії:

3.1. Спробувати отримати з бази існуючу пару «старт фініш»;

3.2. Якщо на кроці 3.1 отримано пустий об’єкт, тоді створити нову пару «старт-фініш», з відповідними атрибутами населених пунктів відбуття та прибуття;

3.3. Створити новий комплексний об’єкт відгуку, наповнити його даними та зв’язати з щойно створеним сегментом маршруту;

3.4. Додати поточну пару «старт-фініш» до текстового об’єкту створеного на кроці 2;

4. На основі початкової та кінцевої точки шляху, побудувати всі можливі в маршрути через існуючі в системі сегменти шляху (детальний алгоритм буде описаний нижче в цьому ж розділі);

5. Отримати Хеш-код маршруту за текстовим об’єктом (крок 2).

Блок схема алгоритму зображена на рисунку 3.10.

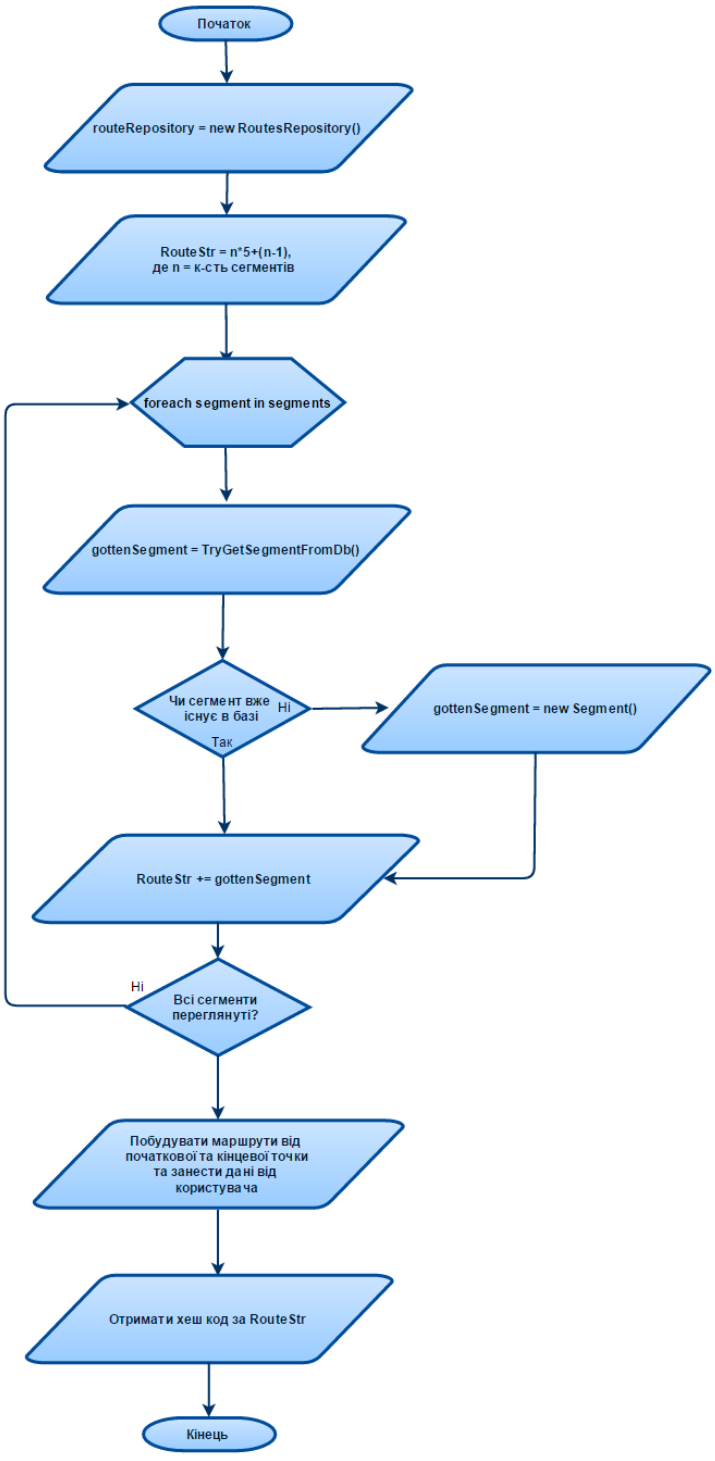

Рис. 3.10. Алгоритм додання нового відгуку про маршрут.

Метод який реалізує цей алгоритм має такий вигляд:

```
publicstring AddNewFeedbackAndGetUrlToRoute(
List<RouteNodeWithFeedbacksData> routesNodeWithFeedbacksData){
//Створити новий репозиторій маршрутів та відгуків
RoutesRepository routeRepository = new RoutesRepository();
FeedbackItemRepository feedbackItemRepository = newFeedbackItemRepository();
//Ініціалізація текстового об’єкту для маршруту
StringBuilder routeStr = newStringBuilder(
routesNodeWithFeedbacksData.Count * 5 +
routesNodeWithFeedbacksData.Count – 1
);
//Визначення порядку маршруту, на початку завжди місто з вищим ідентифікатором
bool backwardOrder = routesNodeWithFeedbacksData[0].OriginCityNodeId >
                     routesNodeWithFeedbacksData[routesNodeWithFeedbacksData
                     .Count - 1].DestinationCityNodeId;

            routesNodeWithFeedbacksData.ForEach(routeWithFeedback =>
            {
//Спроба отримання сегмент з бази
RouteNode routeNode = routeRepository.GetRouteNode(
       routeWithFeedback.OriginCityNodeId,
       routeWithFeedback.DestinationCityNodeId
) ??
//Якщо сегмент відсутній, тоді створити новий сегмент
newRouteNode(){
RouteNodeId = routeRepository.CreateRouteNode(
                         routeWithFeedback.OriginCityNodeId,
routeWithFeedback.DestinationCityNodeId
)
                     };
//Додання нового відгуку
int feedbackId = feedbackItemRepository.AddNewFeedback(
routeNode.RouteNodeId, routeWithFeedback.UserId,
                     routeWithFeedback.SubmitTime);

routeWithFeedback.FeedbackValues.ForEach(feedbackValue
=>feedbackItemRepository.AddNewFeedbackValue(newFeedbackValueData()
                 {
                     FeedbackId = feedbackId,
                     Value = feedbackValue.Value,
                     FeedbackItemId = feedbackValue.FeedbackItemId
                 }));

if (!backwardOrder)
                 {
                     track.Append(routeWithFeedback.OriginCityNodeId).Append("-");
if (routeWithFeedback.Equals(routesNodeWithFeedbacksData.Last()))
                     {
                         track.Append(routeWithFeedback.DestinationCityNodeId);
                     }
                 }
else
                 {
                     track.Insert(0, routeWithFeedback.OriginCityNodeId).Insert(0, "-");
if (routeWithFeedback.Equals(routesNodeWithFeedbacksData.Last()))
                     {
                         track.Insert(0, routeWithFeedback.DestinationCityNodeId);
                     }
                 }
            });
//Побудувати маршрути з початкової та кінцевої точки
            BuildRoutes(routesNodeWithFeedbacksData[0].OriginCityNodeId,
                routesNodeWithFeedbacksData[routesNodeWithFeedbacksData.Count - 1]
```

```
.DestinationCityNodeId);
//Отримати Хеш
string hash = routeRepository.GetHashByTrack(track.ToString());
//Згенерувати URL для маршруту
returnString.Format("{0}-{1}-To-{2}", hash,
                routesNodeWithFeedbacksData.First().OriginCityNode,
                routesNodeWithFeedbacksData.Last().DestinationCityNode)
                .Replace(" ", "-")
                .Replace(",", "");
}
```

Наступним важливим етапом розробки, є алгоритм пошуку та побудови маршрутів з наявними в системі сегментами. В основі цього принципу знаходження, лежить алгоритм пошуку в глибину. Об’єкт який містить інформацію про сегменти шляху, необхідну для здійснення пошуку, має такий вигляд:

```
publicclassPointTree
    {
publicint ParentId { get; set; }
publicList<int> ChildPointIds { get; set; }
}
```

Отже алгоритм побудови та пошуку маршрутів містить такі кроки:

1. Визначити ідентифікатори початкової та кінцевої точок;
2. Заповнити елемент PointTree таким чином:

2.1. Атрибут ParentId – це одна з точок (відправлення/прибуття).

2.2. Для задання атрибута ChildPointIds, необхідно здійснити в базі даних пошук всіх існуючих сегментів шляху, в яких фігурує parentId та після цього видалити повторення в колекції нащадків;

3. Рекурсивно запустити функцію пошуку, яка містить такі кроки:

3.1. Якщо ParentId = точці прибуття, тоді виконати перевірку (чи існує аналогічний маршрут в базі даних, якщо ні – додати), інакше перейти до кроку 3.2.

3.2. Перевірити глибину вкладення, якщо межа досягнута, перейти до наступного елемента-нащадка (ChildPoint), перейшовши до кроку 2.2. і задавши значення атрибуту ParentId, як поточний елемент ChildPoint. Інакше перейти до кроку 3.2.1.

3.2.1. Вибрати перший з елементів ChildPoint;

3.2.2. Заповнити новий елемент PointTree, виконавши крок 2, але зазначити ParentId значенням ChildPoint який був визначений на кроці 3.2.1.

3.2.3. Видалити перший елемент ChildPoint (той який був вибраний на кроці 3.2.1).

3.2.4. Виконати крок 3.

Блок схема зображена на рисунку 3.11.

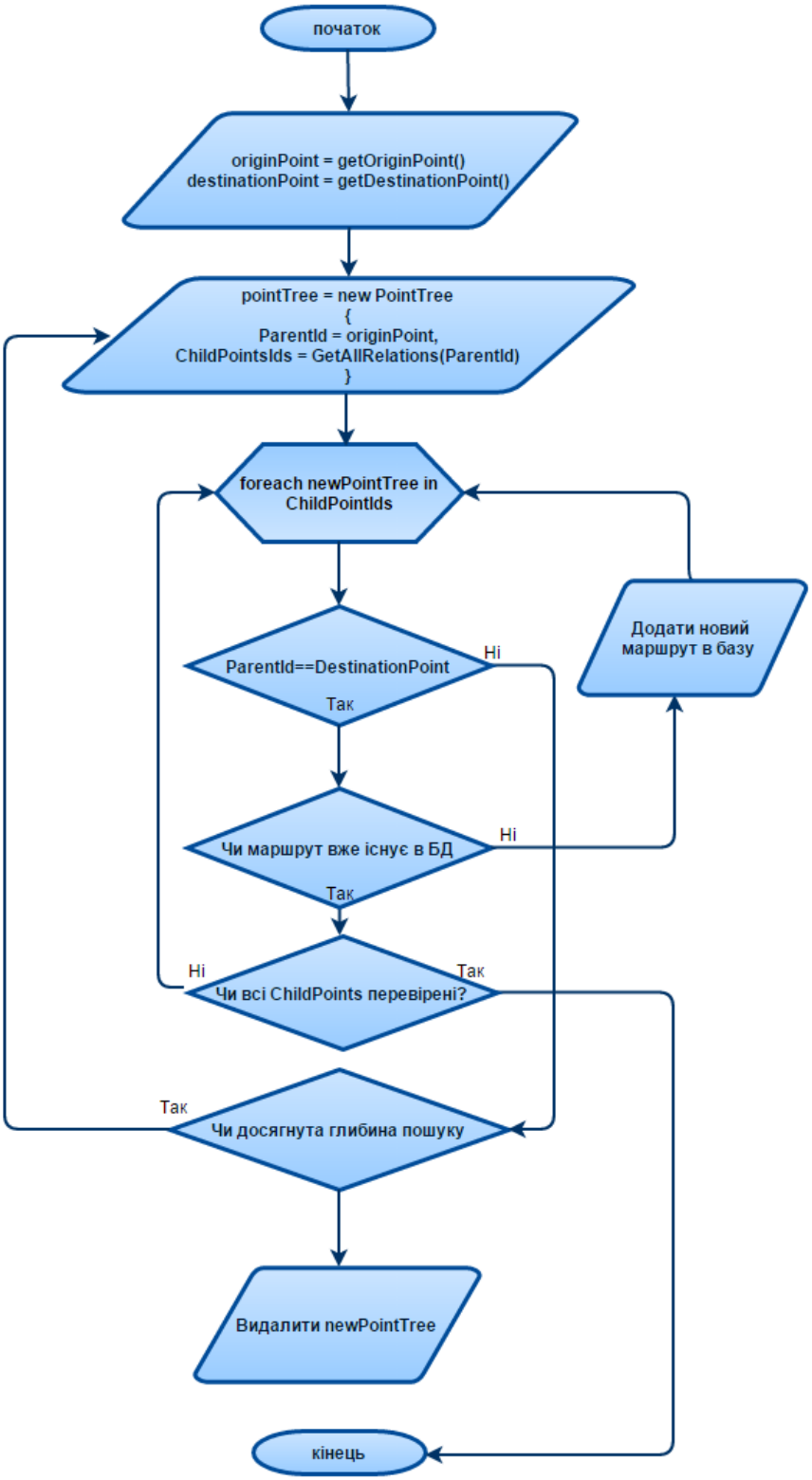

Рис. 3.11. Алгоритм додання нового відгуку про маршрут.

Рекурсивна функція з кроку 3, має такий вигляд:

```
privatevoid Build(string trek, ref PointTree step, long destinationCityId){

//Формування текстового об’єкту з переліком точок в маршруті
            trek = string.IsNullOrEmpty(trek) ?
             step.ParentId.ToString(CultureInfo.InvariantCulture)
             : string.Format("{0}-{1}", trek, step.ParentId);
//Якщо точка відправлення та прибуття однакова
if (step.ParentId == destinationCityId){
//І якщо такого маршруту в базі ще не існує
if (_roads.Count(s => s == trek) == 0){
//Додати новий маршрут
                    _roads.Add(trek);

                    SaveRoute(trek);
                }
            }
else{
//Якщо глибина пошуку не досягнута
if (CheckTrekDepth(trek)){
//Поки перевірені не всі ChildPoints
while (step.ChildPointIds.Count != 0){
//Створити новий PointTree елемент
var nexPointId = step.ChildPointIds.First();
var newLevel = newPointTree{
                            ParentId = nexPointId,
                            ChildPointIds = GetRelations(trek, nexPointId)
                        };
                        step.ChildPointIds.Remove(nexPointId);
//Виконати побудову рекурсивно
                        Build(trek, ref newLevel, destinationCityId);
                    }
                }
            }
}
```

Результатом роботи вище описаного алгоритму, буде список ідентифікаторів маршрутів, знайдених, або щойно створених.

Вище описаний алгоритм, повертає всі можливі маршрути, які можна скласти з наявних в системі сегментів. Але як показує аналіз даних, багато з сформованих маршрутів є недоцільними. Наприклад вірогідність того, що когось зацікавить маршрут Львів → Донецьк → Тернопіль, вкрай мала, і відображення цієї інформації на інтерфейсі користувачі є недоцільним. З огляду на цю проблему, було розроблено додатковий алгоритм фільтрації знайдених маршрутів. Даний алгоритм працює за допомогою сервісу Google Distance Matrix. Цей сервіс, допомагає отримати географічну відстань між двома точками (наприклад Львів → Тернопіль), також є можливість задання багатьох пар. Цей запитмає такий вигляд:

https://maps.googleapis.com/maps/api/distancematrix/json?origins=Vancouver+BC|Seattle&destinations=San+Francisco|Victoria+BC, де точки відправлення зазначаються після слова origins, а точки прибуття після слова destinations. Якщо елементів декілька, вони розділяються символом «|».Результатом запиту буде матриця відстаней, яка представлена в форматі JSON і матиме такий вигляд:

```
{
   "destination_addresses":[
      "San Francisco"
   ],
   "origin_addresses":[
      "Vancouver"
   ],
   "rows":[
      {
         "elements":[
            {
               "distance":{
                  "text":"1 707 km",
                  "value":1706929
               }
            }
         ]
      }
   ]
}
```

Отже головна ідея алгоритму фільтрації знайдених маршрутів, полягає в порівнянні відстані оптимального маршруту з точки зору Google Maps API, з маршрутом який представлений в системі. Фільтрація елемента відбувається, якщо наступне твердження є вірним: якщо відстань знайденого маршруту, є більш як вдвічі більшою, за відстань маршруту який запропонував Google Maps API, тоді виключити даний маршрут з списку для представлення користувачеві. Розроблений код алгоритму, має такий вигляд:

```
privateasyncTask<bool> ValidateTrack( string track ) {
      StringBuilder requestBuilder = newStringBuilder();
//Отримання ідентифікатор мови за замовчуванням
      int languageId=(await DataContext.Languages.SingleAsync( x => x.IsDefault )
      ).LanguageId;
      //Отримання списку назв точок маршруту, за їхніми ідентифікаторами та мовою
      List<string> citiesNodes = track.Split( '-' )
```

```
      .Select( trackNumber =>
            GetCityNameAndRegionName(
                  int.Parse( trackNumber ),
                  languageId
            )
      ).ToList();
      requestBuilder.Append(@"https://maps.googleapis.com/maps/api/distancematrix/json?
" );
      requestBuilder.Append( "origins=" );
//Додання місто відправлення до запиту
      for( int i = citiesNodes.Count - 2; i >= 0; i-- ) {
            requestBuilder.Append( citiesNodes[i] );
            if( i > 0 ) {
                  requestBuilder.Append( "|" );
            }
      }
//Додання місто прибуття до запиту
      requestBuilder.Append( "&destinations=" );
      for( int i = citiesNodes.Count - 1; i >= 1; i-- ) {
            requestBuilder.Append( citiesNodes[i] );
            if( i > 1 ) {
                  requestBuilder.Append( "|" );
            }
      }
//Задання методу пересування
      requestBuilder.Append( "&mode=driving" );
      requestBuilder.Append( "&key=AIzaSyDagV9wfy0HxBPjCZti7NdVIUrcdtQqgoU" );
//надсилання запиту на Google Distance Matrix Api
      using( var data = await new WebClient()
      .OpenReadTaskAsync( requestBuilder.ToString() ) ) {
            if( data == null ) return true;
            using( var reader = newStreamReader( data ) ) {
//десереалізація відгуку сервера
                  dynamic responce = JsonConvert.DeserializeObject(
                        await reader.ReadToEndAsync() );
                  if( responce.status == "OK" ) {
//визначення рекомендованої відстані маршруту
                        long reccomendedDistance = long.Parse(
                              responce
                              .rows[responce.rows.Count - 1]
                              .elements[0]
                              .distance.value.ToString()
                        );
//визначення фактичної відстані маршруту
                        long actualDistance = 0;
                        for( int i = 0; i < responce.rows.Count; i++ ) {
                              actualDistance += long.Parse(
                              responce
                              .rows[i]
                              .elements[i]
                              .distance.value.ToString()
                        );
                        }
//порівняння фактичної та рекомендованої відстаней маршруту
                        if( actualDistance < reccomendedDistance * 2 ) {
                              return true;
                        }
                  }
            }
      }
      return false;

}
```

Повертаючись до попереднього прикладу, маршрут Львів → Донецьк → Тернопіль, буде відфільтрований та не буде відображатись як результат запиту. Також буде можливість отримати результати пошуку з ввімкнутим та вимкнутим алгоритмом фільтрації маршрутів.

3.5. Результати виконання роботи

Коли користувач починає свою роботу з цим додатком, початковою є сторінка пошуку (рис. 3.12.), на якій йому необхідно ввести початкову та кінцеву точку маршруту та увімкнути чи вимкнути фільтрацію маршрутів.

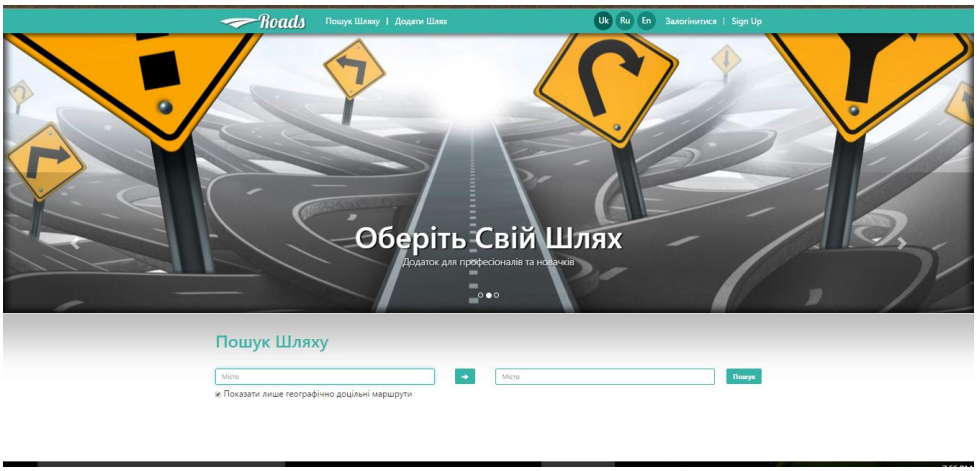

Рис. 3.12. Сторінка пошуку маршрутів.

Після введення інформації про початкову та кінцеву точку та натиснення кнопки «Пошук», на екрані з’явиться сторінка з таблицею результатів (рис. 3.13.).

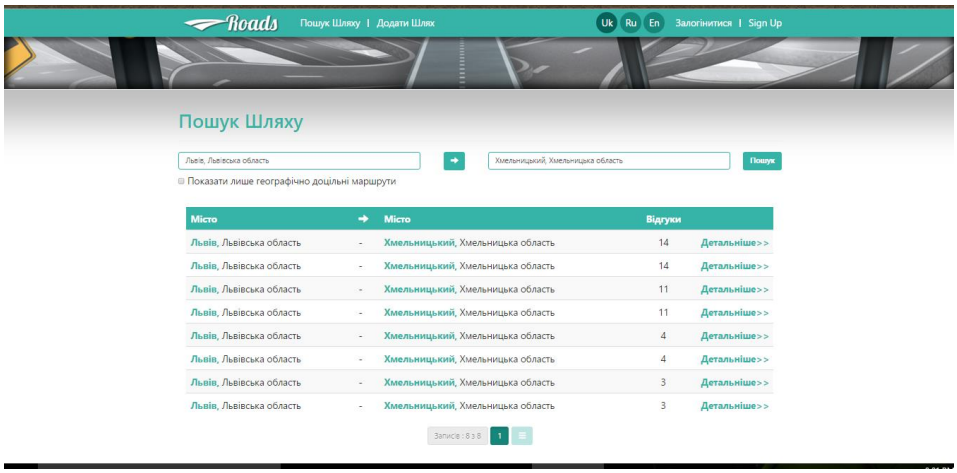

Рис. 3.13. Сторінка результатів пошуку маршрутів.

Після того як користувач натиснув на кнопку «Детальніше», що знаходиться навпроти одного з запропонованих маршрутів, він побачить сторінку детальної інформації про маршрут (рис 3.14.). На цій сторінці є можливість переглянути всі відгуки залишені на цьому сегменті маршруту, побачити середнє значення числових показників відгуку по сегменту та по всьому маршруту.

За допомогою панелі з лівої частини сторінки, можна вибирати різні сегменти маршруту, що являє собою деталізовану інформацію всього шляху.

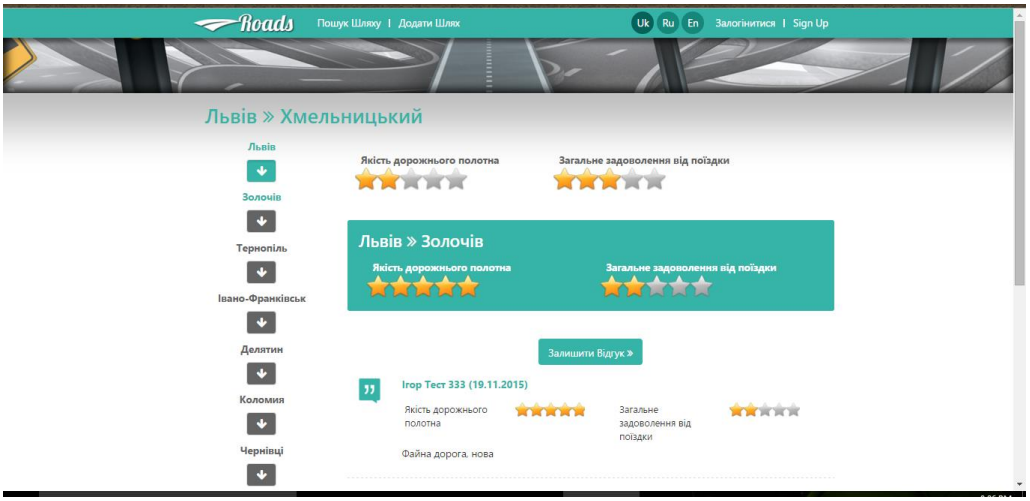

Рис. 3.14. Сторінка детальної інформації про маршрут.

На сторінці детальної інформації, в користувача є можливість додати власний відгук, для цього необхідно натиснути на кнопку «Залишити відгук», після чого з’явиться модальне вікно (рис. 3.15.), в якому можна буде залишити дані щодо конкретного сегменту маршруту.

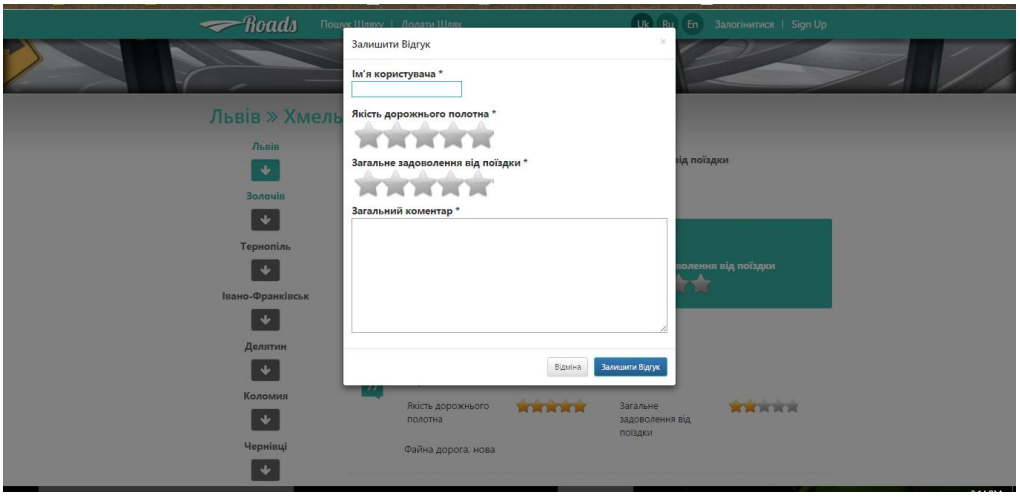

Рис. 3.15. Модальне вікно відгуку про сегмент дороги.

Якщо користувач не зміг знайти маршрут яким слідував останнього разу, в нього є можливість додати новий шлях. Для цього необхідно на верхній панелі сторінки, вибрати пункт «Додати шлях». Після цього, користувач побачить сторінку (рис. 3. 16.), на якій необхідно ввести всі ключові проміжні пункти, через які пролягає маршрут.

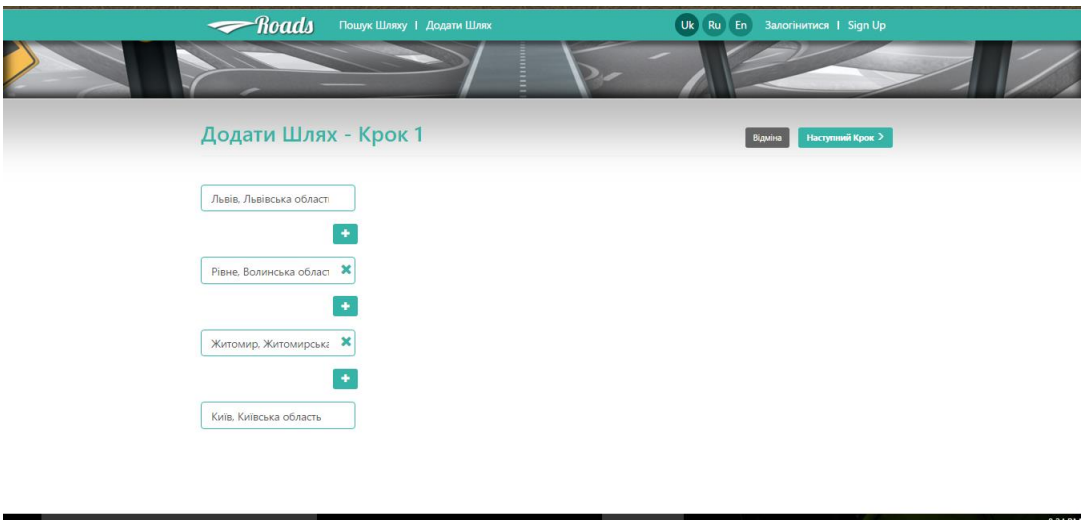

Рис. 3.16. Сторінка додання нового маршруту. Крок 1.

Після вибору проміжних географічних пунктів, необхідно натиснути на кнопку «Наступний крок». На рисунку 3.17. зображена сторінка створення нового маршруту, в якій необхідно ввести відгуки по кожному сегменту дороги, що був створений на попередньому кроці. Дана сторінка складається з 2 областей, панель зліва використовується для навігації між сегментами, а центральна панель призначена для введення даних про вибрану частину шляху.

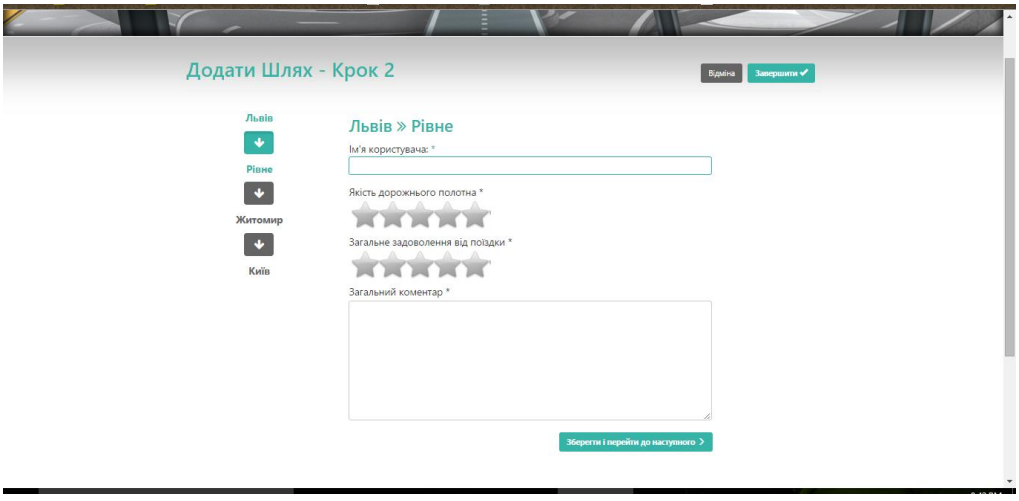

Рис. 3.17. Сторінка додання нового маршруту. Крок 2.

Даний веб-додаток є багатомовним, та дозволяє легко перемикатись між мовами інтерфейсу, для цього необхідно на верхній панелі (рис. 3. 18.) вибрати необхідну мову. Програмна архітектура системи побудована таким чином, що кількість підтримуваних мов є необмеженою, на даний момент число представлених дорівнює 3. А саме: українська, російська та англійська.

Рис. 3.18. Верхня панель веб-додатку.

Для налаштування системи, реалізована адміністративна частина. Для переходу в адміністративну частину, необхідно пройти попередній етап авторизації (рис. 3.19.) і вибрати пункт «Територія Адміна» на верхній панелі веб-додатку.

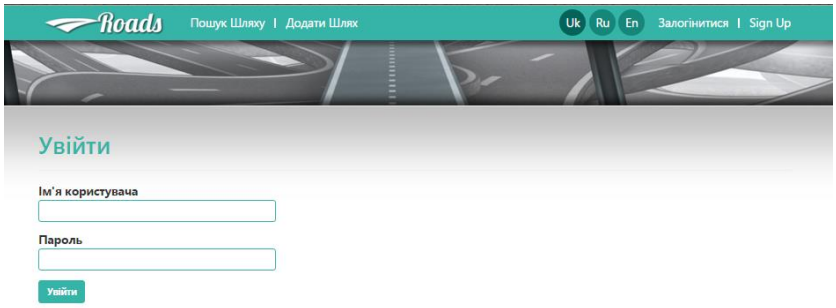

Рис. 3.19. Сторінка авторизації веб-додатка.

Адміністративна панель (рис 3.20.) системи збору даних про дорожні маршруту, включає в себе 5 основних зон керування:

- Загальні налаштування сайту– базові опції, такі як глибина пошуку та кількість відображуваних маршрутів при пошуку на одній сторінці;
- Відгуки(рис 3.21.) – конфігурація форм відгуків, складових цих форм та їхні переклади;
- Статичні переклади– конфігурація перекладів елементів інтерфейсу користувача;
- Динамічні переклади– конфігурація перекладів елементів форми відгуків;
- Переклади місць (рис 3.22.) – переклади міст, сіл та областей.

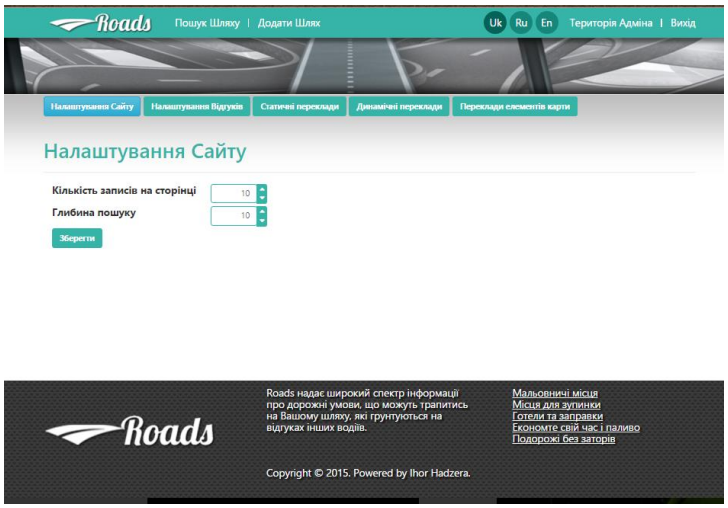

Рис. 3.20. Адміністративна панель системи.

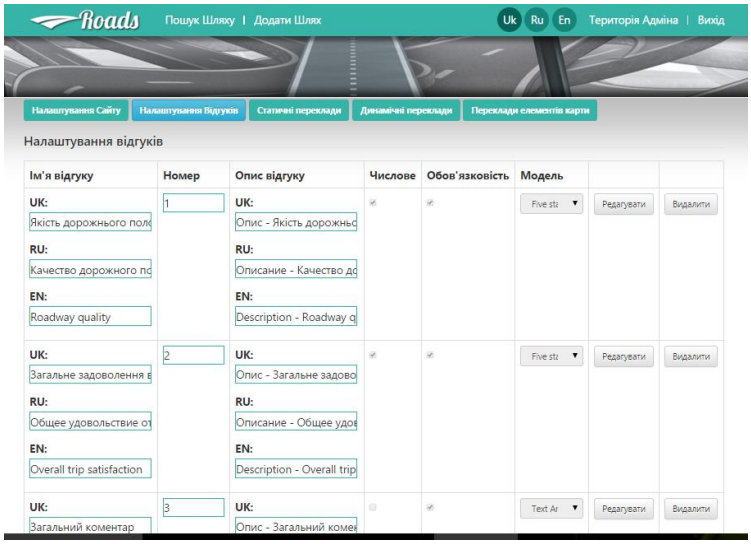

Рис. 3.21. Вкладка налаштувань відгуків.

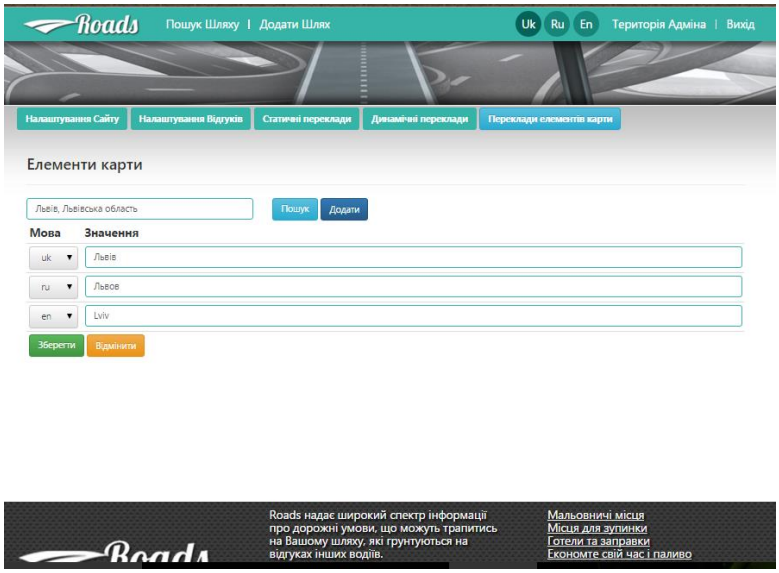

Рис. 3.22. Вкладка налаштувань перекладів географічних об’єктів.

3.6. Висновок

В процесі фізичного проектування бази даних для системи автоматизованої системи збору даних про дорожні маршрути, було спроектовано структуру бази даних, яка складається з реляційних таблиць та взаємозв’язків між ними. Також було визначено конкретні типи даних та інші атрибути, які забезпечують оптимальність та зручність збереження даних.

З огляду на специфіку системи, було розроблене рішення для доступу до даних, що забезпечує оптимальну продуктивність системи з базою даних, та високий рівень ізоляції інших частин системи від логіки взаємодії з даними. Також за допомогою технології Entity Framework був реалізований низькорівневий доступ до даних, що дозволило використовувати технологію LINQ для побудови запитів та отримання інформації з бази даних та маніпуляції даними як колекціями класів.

Особливу увагу було приділено розробці архітектури системи, що є вирішальним моментом всієї структури проекту і має визначний вплив на час та затрати щодо розробки проекту. В результаті було прийнято рішення щодо максимального розділення системи, що зробило архітектурні частини більш незалежними, а процес зміни менш клопітким. Це допомогло впровадити принцип інверсії залежностей та використати технологію IoC контейнера, для більш гнучкої конфігурації зв’язків між об’єктами в системі.

В процесі розробки, було використано ряд існуючих алгоритмічних рішень (наприклад алгоритм пошуку в глибину при формуванні маршруту) та створено ряд власних, таких як фільтрація згенерованих маршрутів за допомогою Google Distance Matrix API, що дозволило відсіяти результати, які як правило є малоцікавими користувачу.

На реальному прикладі було продемонстровано роботу автоматизованої системи для збору даних про дорожні маршрути та продемонстровано основні можливості даного програмного продукту.

РОЗДІЛ 4. ЕКОНОМІЧНА ОЦІНКА ПРОЕКТНОГО РІШЕННЯ

4.1. Економічна характеристика автоматизованої системи для збору даних про дорожні маршрути

Метою дипломної роботи є розроблення автоматизованої системи для збору даних про дорожні маршрути. Результатом роботи є система, яка базується на інтерактивному отриманні даних про шляхи, що дозволяє автомобілісту обрати оптимальний для себе маршрут.

На сьогоднішній день, всі сфери діяльності людини потребують сучасних рішень, що автоматизують повсякденні задачі. Однією з таких задач створення систем збору і обробки даних.

Сучасні автоматизовані системи, повинні відповідати багатьом суворим критеріям, особливо це стосується систем для збору даних від респондентів. Оскільки процес проектування та розробки є затратним в часі та ресурсах, а отже внесення змін у випадку незадовільних результатів потребує значних вкладень. Тому розробка методів та алгоритмів для підвищення ефективності автоматизованого збору даних про дорожні маршрути є актуальною задачею сьогодення.

Результати роботи дають можливість розробити систему, що оперуватиме алгоритмами аналізу та порівняння результатів, це в свою чергу дає змогу не витрачати багато ресурсів та часу на внесення змін.

Основною причиною поширення таких систем є відносно досить вагома економія часу, фінансових та людських ресурсів, у порівнянні з отриманням таких даних звичайними способами, такими як опитування та анкетування.

Потенційними користувачами автоматизованих систем для збору даних про дорожні маршрути можуть бути автомобілісти та туристи, що подорожують автомобільними шляхами автобусом або автостопом.

Яскравим аналогом розробленої системи є інтерактивний картографічний сервіс UaRoads. Даний ресурс дає змогу прокладати маршрути відповідно до багатьох характеристик, які задав користувач та деяких інших аспектів, таких як відстань та завантаженість дороги на момент запиту.

4.2. Розрахунок витрат на розробку та впровадження програмного рішення

1) Витрати на розробку і впровадження програмного засобу (К) визначаються як:

$$K_{заг} = K_1 + K_2 \quad (4.1)$$

де K_1 – витрати на розробку програмного засобу, грн.;

K_2 – витрати на відлагодження і дослідну експлуатацію програмного засобу на ЕОМ, грн.

Витрати на розробку програмного засобу включають в себе:

1. Витрати на оплату праці розробників ($B_{оп}$);
2. Єдиний соціальний внесок ($B_{єв}$);
3. Вартість додаткових виробів, що закуповуються ($B_{д}$);
4. Транспортно-заготівельні витрати ($B_{тр}$);
5. Витрати на придбання спецобладнання ($B_{со}$);
6. Накладні витрати ($B_{н}$);
7. Інші витрати ($B_{ін}$).

Для проведення розрахунків витрат на оплату праці необхідно визначити категорії працівників, які приймають участь в процесі проектування, їх чисельність, середньоденну заробітну плату спеціаліста відповідної категорії та трудомісткість робіт у людино-днях (людино-годинах).

Команда, що приймає участь в процесі розробки складається з двох чоловік:

1. Програміст (.NET Developer), який відповідає за увесь життєвий цикл продукту окрім етапу тестування.

2. Інженер з забезпечення якості (Quality Assurance engineer), на якому лежить відповідальність за тестування продукту.

Їхня заробітна плата становить:

- .NET Developer – 8 400 грн. (1 люд.);

- Quality Assurance engineer– 4 500 грн. (1 люд.).

Трудомісткість робіт над проектом становить 23 людино-днів для програміста та 10 людино-днів для тестера.

Середньоденна заробітна плата і-го розробника ($ЗП_{Дi}$) обчислюється за формулою:

$$ЗП_{Д_i} = \frac{ЗП_i}{Фм} \quad (4.2)$$

де $ЗП_i$ - основна місячна заробітна плата розробника і-ої спеціальності, грн.;

$Фм$ – місячний фонд робочого часу, днів (24 дні).

$$ЗП_{Д1} = \frac{8400}{24} = 350 \text{ грн.}$$

Середня заробітна плата програміста становить 350 грн.

$$ЗП_{Д2} = \frac{4\,500}{24} = 187{,}5 \text{ грн.}$$

Середня заробітна плата тестера становить $187{,}50$ грн.

Розрахунок витрат на оплату праці усіх працівників обчислюємо за формулою:

$$В_{оп} = \sum_{i=1}^{N} n_i t_i ЗП_{Д_i} \quad (4.3)$$

де n_i – чисельність розробників проекту і-ої спеціальності, осіб;

t_i – час, витрачений на розробку проекту працівником і-ої спеціальності, дні;

$ЗП_{Дi}$ – денна заробітна плата розробника і-ої спеціальності, грн.;

$$В_{оп} = 1 * 23 * 350 + 1 * 10 * 187{,}5 = 8050 + 1875 = 9925 \text{ грн.}$$

Розрахунок витрат на оплату праці розробників зведено у табл. 4.1

Таблиця 4.1.

Розрахунок витрат на заробітну плату

№ з/п	Спеціальність розробника	Кількість розробників	Час роботи, Дні	Денна заробітна плата розробника, грн.	Витрати на оплату праці, грн.
1	.NET Developer	1	23	350	8050
2	Quality Assurance engineer	1	10	187,5	1875
Разом					9925

2)Витрати на оплату праці працівникам тягнуть за собою відрахування єдиного соціального внеску $B_{єв}$, ставка якого залежатиме від класу професійного ризику розробників програмного забезпечення та визначатиметься у відсотковому співвідношенні від фонду оплати праці. Підприємство зобов'язане здійснити відрахування до Пенсійного фонду, згідно 2-го класу ризику – 36,77 %.

$$В_{ЄВ} = 9925 * 0{,}3677 = 3649{,}42 \text{ грн.}$$

3) Витрати на додаткові вироби, що закупляються ($В_д$) (папір, диски тощо) визначаються за їхніми фактичними цінами з врахуванням найменування, номенклатури та необхідної їх кількості в проекті. Вихідні дані та результати розрахунків занесені в табл. 4.2. Транспортно-заготівельні витрати ($В_{тр}$) становлять 13% від суми витрат на додаткові вироби, що закупляються.

$В_{Д_1} = 68$ грн вартість упаковки паперу формату А4.

$В_{Д_2} = 8$ грн – вартість диску DVD-R.

$В_{Д_3} = 17$ грн – вартість кулькової ручки.

Розраховуємо суму з урахуванням транспортно-заготівельних витрат:

$$В_д = 68 * 1 * 1{,}13 + 8 * 4 * 1{,}13 + 17 * 2 * 1{,}13 = 76{,}84 + 36{,}16 + 38{,}42 = 151{,}42 \text{грн.}$$

Таблиця 4.2.

Розрахунок витрат на куповані вироби

№ з/п	Найменування купованих виробів	Марка, тип	Кількість на розробку, шт.	Ціна за одиницю, грн.	Сума витрат, грн.	Сума витрат з урахуванням транспортно-заготівельних
1	Папір формату А4	Lazer Copy	1	68	68	76,84
2	Диск DVD-R	Verbatim DVD-R 4,7 GB	4	8	32	36,16
3	Канцелярські вироби	Flair Writo-[illegible]	2	17	34	38,42
Разом						151,42

Згідно проведених обчислень, усі здійснені витрати на додаткові вироби становлять 151,42 грн.

4) Витрати на придбання спецобладнання ($В_{со}$) для проведення експериментальних робіт розраховуються в тому випадку, коли для розроблення та впровадження проектного рішення необхідне придбання додаткових технічних засобів. Оскільки при розробці проектного рішення потреби у придбанні спецобладнання не було, тому його вартість буде рівна 0.

5) Накладні витрати ($В_{н}$) проектних організацій передбачають витрати на управління, загальногосподарські, невиробничі витрати. Вони становлять 20-30% витрат на оплату праці.

Для розрахунку накладних витрат необхідно перемножити витрати на оплату праці на 25%:

$$В_{н} = 9925 * 0,25 = 2481,25 \text{ грн.}$$

6) Інші витрати ($В_{ін}$) – це усі ті витрати, які не були враховані в попередніх статтях витрат.

Їх розраховують за встановленими відсотками до витрат на оплату праці (становить 10%).

$В_{ін} = 9925 * 0,1 = 992,5$ грн.

7) Щодо витрат на розроблення проектного рішення, то їх обчислюємо за формулою:

$$K_1 = B_{оп} + B_{св} + B_д + B_{со} + B_н + B_{ін} \quad (4.4)$$

$К_1$=9925+3649,42+151,42+2481,25+992,5=17199,59 грн.

Також потрібно обрахувати витрати на налагодження і дослідну експлуатацію системи. Їх можна визначити згідно із встановленою для цих обчислень формулою:

$$K_2 = S_{м.г.} \cdot t_{від} \quad (4.5)$$

де $S_{м.г.}$ – вартість однієї години роботи ПК, грн./год.;

$t_{від}$ - кількість годин роботи ПК на налагодження програми, год.

При роботі ноутбук споживає 0.036 КВт/год., якщо вираховувати, що тариф на електроенергію становить 0,46 грн., то таким чином остаточна вартість однієї години роботи ПК ($S_{м.г.}$) становитиме 0,01656грн. (0,036 КВт/год.* 0,46грн.).

На написання та відлагодження системи було витрачено ($t_{від}$) 33 днів (33*8 = 264год.), тому:

$К_2 = 0,01656 * 264 = 4,37$грн.

Результати усіх здійснених розрахунків зведено і записано в єдину табл. 4.3, де вказано кошторис витрат на розробку проектного рішення, а також окремі статті витрат.

$К = 27003,59 + 4,37 = 27007,96$ грн.

Таблиця 4.3.

Кошторис витрат на розробку проектного рішення

Найменування елементів витрат	Сума витрат, грн.
Витрати на розробку проектного рішення, у т.ч.:	27003,59
витрати на оплату праці	9925
сплата єдиного соціального внеску	3649,42
витрати на додаткові вироби, що закуповуються	151,42
витрати на придбання спецобладнання	0
накладні витрати	2481,25
інші витрати	992,5
Витрати на відлагодження і дослідну експлуатацію системи	4,37
Всього	17203,96

Сума витрат на розробку програмного продукту становить 17203,96грн. В цю суму входить заробітна плата розробників, відрахування у єдиний соціальний внесок, витрати на додаткові вироби, які купуються, накладні витрати, а також інші витрати.

4.3. Визначення комплексного показника якості

Комплексний показник якості ($П_я$) визначається шляхом порівняння показників якості проектованої системи і вибраного аналогу.

Вибір показників якості здійснюється експертним методом. До основної групи показників обов’язково були включені наступні:

Показники призначення

1. Актуальність
2. Універсальність
3. Ступінь новизни

Показники надійності

1. Ймовірність помилки в проектуванні

2. Стійкість

Показники безпеки

3. Захищеність

Патентно-правові показники

4. Патентно-правовий статус

Ергономічні показники

5. Легкість експлуатації

Комплексний показник якості проектованої системи визначається методом пошуку арифметичного середньозваженого з формули:

$$П_{Я} = \sum_{i=1}^{m} Ci \times q_i \tag{4.6}$$

де m - кількість одиничних показників (параметрів), прийнятих для оцінки якості розробленого проектного рішення;

q_i - коефіцієнт вагомості кожного з параметрів щодо їхнього впливу на технічний рівень та якість проектованої системи (встановлюється експертним шляхом), причому:

$$\sum_{i=1}^{m} q_i = 1{,}0 \tag{4.7}$$

C_i - часткові показники якості, визначені порівнянням числових значень одиничних показників проектованої системи і аналога за формулами:

$$C_i = \frac{П_{пр\,i}}{П_{ai}} \quad \text{або} \quad C_i = \frac{П_{ai}}{П_{пр\,i}} \tag{4.8}$$

де $П_{прi}$, $П_{ai}$ - кількісні значення і-го одиничного показника якості відповідно проектованої системи і аналога.

З попередніх двох формул вибирається та, в якій збільшення відповідає покращенню показника якості проектованої системи. Результати розрахунку зводимо в табл. 4.4.

Таблиця 4.4

Визначення комплексного показника якості проекту або аналога

Показники	Числове значення показників, бали		Відносний показник якості, Сі	Коефіцієнт вагомості,	Сі ×qi
	Аналог	Розроблене проектне рішення		Qi	
Показники призначення					
Актуальність	4	9	2,3	0,1	0,23
Універсальність	3	8	2,7	0,2	0,54
Ступінь новизни	4	9	2,3	0,05	0,11
Показники надійності					
Ймовірність помилки в проектуванні	6	10	1,7	0,3	0,51
Стійкість	10	6	1,5	0,2	0,3
Показники безпеки					
Захищеність	9	9	1,0	0,05	0,05
Патентно-правові показники					
Патентно-правовий статус	10	10	1	0,05	0,05
Ергономічні показники					
Легкість експлуатації	7	9	1,3	0,05	0,06
Всього				**1**	**1,85**

Отже, комплексний показник якості дорівнює:

$$П_я = 0,23+0,54+0,11+0,51+0,3+0,05+0,05+0,06 = 1,85$$

Отриманий результат показує, що розробка автоматизованої системи збору даних про дорожні маршрути є кращою, в порівнянні з аналогом.

4.4. Визначення експлуатаційних витрат

При порівнянні програмних засобів в експлуатаційні витрати включають вартість підготовки даних (E_1) і вартість годин роботи ПК (E_2). Одноразові експлуатаційні витрати визначаються за формулою:

$$E_{П(А)} = E_{1П(А)} + E_{2П(А)} \quad (4.9)$$

де $E_{П(А)}$ - одноразові експлуатаційні витрати на проектне рішення (аналог), грн.;

$E_{1П(А)}$ - вартість підготовки даних для експлуатації проектного рішення (аналогу), грн.;

$E_{2П(А)}$ - вартість машино-годин роботи ПК для проектного рішення (аналогу), грн.

Річні експлуатаційні витрати визначаються за формулою:

$$B_{(е)П(А)} = E_{П(А)} * N_{П(А)} \quad (4.10)$$

де $B_{(е)П(А)}$ – експлуатаційні річні витрати проектного рішення, грн.;

$N_{П(А)}$ - періодичність експлуатації проектного рішення (аналогу), разів/рік.

Вартість підготовки даних для експлуатації проектного рішення (аналогу) (E_1) визначаються за формулою:

$$E_1 = \sum_{i=1}^{N} n_i \cdot t_i \cdot ЗПг_i \quad (4.11)$$

де i – номери категорій персоналу, які беруть участь у підготовці даних;

n_i – чисельність співробітників і-ї категорії, осіб;

t_i – трудомісткість роботи співробітників і-ї категорії, осіб;

$ЗПг_i$ – середньогодинна ставка робітника і-ї категорії з врахуванням відрахувань єдиного соціального внеску, грн./год.

Середньогодинна ставка оператора визначається за формулою:

$$ЗПг_i = \frac{ЗПг_{0i}(1+b)}{Фг} \quad (4.12)$$

де $ЗПг_{0i}$ – основна місячна зарплата працівника і-ї категорії, грн.;

b – коефіцієнт, який враховує єдиний соціальний внесок;

$Фг$ – місячний фонд робочого часу, год.

$$ЗПг_1 = \frac{8400 * (1 + 0{,}3677)}{24 * 8} = 45{,}36 \text{ грн.}$$

$$ЗПг_2 = \frac{4500 * (1 + 0{,}3677)}{24 * 8} = 18{,}32 \text{ грн.}$$

Розраховуємо вартість підготовки даних для експлуатації проектного рішення:

$$Е_1 = 1 * 23 * 45{,}36 + 1 * 10 * 18{,}32 = 1043{,}28 + 183{,}2 = 1226{,}48 \text{ грн.}$$

Вартість машино-годин роботи ПК для проектного рішення рівна 3,1 грн. Розраховуємо одноразові експлуатаційні витрати, вони становлять

$$Е_П = 1226{,}48 + 3{,}1 = 1229{,}58 \text{ грн.}$$

Періодичність експлуатації проектного рішення становить 3 рази/рік. Враховуючи цей параметр розраховуємо річні експлуатаційні витати:

$$В_{(е)П} = 1229{,}58 * 3 = 3688{,}74 \text{ грн.}$$

Над проектом-аналогом працює 1 керівник проекту, 1 програміст та 1 тестер. Розраховуємо середньогодинну ставку для кожного працівника враховуючи що заробітна плата становить для керівника проекту 9000 грн., для програміста8500грн.та для тестера 6000 грн. Час роботи над проектом 24 дні.

$$ЗПг_1 = \frac{9000 * (1 + 0{,}3677)}{24 * 8} = 64{,}11 \text{ грн.}$$

$$ЗПг_2 = \frac{8500 * (1 + 0{,}3677)}{24 * 8} = 60{,}54 \text{ грн.}$$

$$ЗПг_3 = \frac{6000 * (1 + 0{,}3677)}{24 * 8} = 42{,}74 \text{ грн.}$$

Розраховуємо вартість підготовки даних для експлуатації проектно-аналогового рішення:

$$Е_{1,А} = 1 * 24 * 64{,}11 + 1 * 24 * 60{,}54 + 1 * 24 * 42{,}74$$
$$= 1538{,}64 + 1452{,}96 + 1025{,}76 = 4017{,}36 \text{грн.}$$

Вхідні дані та отримані результати заносимо в таблицю 4.5

Таблиця 4.5.

Розрахунок витрат на підготовку даних для роботи на ПК

Категорія персоналу	Чисельність співробітникі в і–ої категорії, чол.	Час роботи співробітників і–ої категорії, год.	Середньогодинна ЗП співробітника і–ої категорії, грн.	Витрати на підготовку даних, грн.
Проектне рішення				
Програміст	1	184	45,36	1043,28
Тестер	1	80	18,32	183,2
Всього				1226,48
Аналог				
Керівник проекту	1	192	64,11	1538,64
Програміст	1	192	60,54	1452,96
Тестер	1	192	42,74	1025,76
Всього				4017,36

Вартість машино-годин роботи ПК для проектно-аналогового рішення рівна 6,4 грн. Розраховуємо одноразові експлуатаційні витрати для проектно-аналогового рішення, вони становлять :

$$E_{(A)} = 4017{,}36 + 6{,}4 = 4023{,}76 \text{ грн.}$$

Періодичність експлуатації проектного рішення становить 4 разів/рік. Враховуючи цей параметр розраховуємо річні експлуатаційні витрати для проектно-аналогового рішення:

$$B_{(e)A} = 4023{,}76 * 4 = 16095{,}04 \text{ грн.}$$

4.5. Розрахунок ціни споживання проектного рішення

Ціна споживання ($Ц_С$) – це витрати на придбання і експлуатацію проектного рішення за весь строк його служби:

$$Ц_{С(П)} = Ц_П + B_{(E)NPV} \quad (4.13)$$

де $Ц_П$ – ціна придбання проектного рішення, грн.;

$B_{(E)NPV}$ – теперішня вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації), грн.:

$$Ц_П = К * \left(1 + \frac{П_Р}{100}\right) \times (1 + С_{ПДВ}) + К_О + К_к \quad (4.14)$$

де $П_р$ – норматив рентабельності (19%);

$К_о$ – витрати на прив'язку та освоєння проектного рішення на конкретному об’єкті, 700грн.;

$К_к$ – витрати на доукомплектування технічних засобів на об'єкті, 1200грн.;

$С_{ПДВ}$ – ставка податку на додану вартість (20 %).

Ц$_п$=17203,96 *(1+0,19)*(1+0,2)+700+1200=26467,25 грн.

Ціна придбання аналогу:

Ц$_А$ = 49641,54 грн.

Теперішня вартість витрат на експлуатацію проектного рішення розраховується за формулою:

$$B_{(e)NPV} = \sum_{t=1}^{T} \frac{B_{(E)Nt}}{(1+R)^t} \quad (4.15)$$

де $B_{(E)Пt}$ - річні експлуатаційні витрати в t-ому році, грн.;

T - строк служби проектного рішення, років;

R - річна ставка проценту банків, (16%).

Зважаючи на те, що $B_{(e)Nt} = const$ протягом всього строку експлуатації розраховуємо $B_{(E)NPV}$:

$$B_{(E)NPV} = \sum_{i=1}^{T} B_{(T)nt} * \frac{1}{(1+R)^t} = PV * B_{(E)П} \quad (4.16)$$

Де PV - ставка дисконту на період T, яка визначається від річної процентної ставки R і періоду експлуатації T. При значенні $R = 16\%$, ставка дисконту буде:

T	0	1	2	3	4
PV	1,0	0,87	0,76	0,66	0,57

Термін експлуатації становить 4 роки, тому $PV = 0,57$. Теперішня вартість витрат на експлуатацію проектного рішення становить:

$$B_{(E)NPV} = PV * B_{(E)П} = 0,57 * 3688,74 = 2102,58 \text{ грн.}$$

Розраховуємо ціну споживання проектного рішення:

$Ц_{с(п)}$= 26467,25+2102,58=28569,83 грн.

Визначаємо ціну споживання для аналогу. Визначаємо теперішню вартість витрат на експлуатацію аналогу. Термін експлуатації аналогу становить 4 роки, тоді $PV = 0,57$. Оскільки $B_{(e)Nt} = const$, то значить:

$$B_{(E)NPV} = PV * B_{(E)A} = 0,57 * 16095,04 = 9174,17 \text{ грн.}$$

Тепер розраховуємо ціну споживання проекту аналогу:

$Ц_{С(А)}$= 49641,54+9174,17=58815,71 грн

4.6. Визначення показників економічної ефективності

Якщо базою для порівняння обрані відповідні програмні засоби, в даному розділі розраховуються такі показники:

1) Показник конкурентоспроможності:

$$Ккс = \frac{Цс(п) \cdot Пя}{Цс(а)} \quad (4.17)$$

$$Ккс = \frac{28569,83 \cdot 1,85}{58815,71} = 0,89$$

2) Економічний ефект в сфері експлуатації (грн.):

$$Еекс = B(e)a - B(e)n \quad (4.18)$$

$$\text{Е}_{\text{екс}} = 16095{,}04 - 3688{,}74 = 12406{,}3 \text{ грн.}$$

3) Економічний ефект в сфері проектування (грн.):

$$Епр = Ца - Цп \quad (4.19)$$

$\text{Е}_{\text{пр}}$=58815,71 -28569,83=30325,88 грн.

Якщо Епр>0 та Еекс>0, то розраховується:

4) Додатковий економічний ефект в сфері експлуатації (грн.):

$$E_{екс\,Д} = \sum_{i=1}^{T} E_{екс}(1+R)^{T-i}, \quad (4.20)$$

$$\text{Е}_{\text{ексД}} = 12406{,}3 * (1.16^1 + 1.16^0) = 14391{,}31 \text{ грн.}$$

5) Додатковий економічний ефект в сфері проектування (грн.):

$$Епр_Д = Епр \times (1+R)^T \quad (4.21)$$

$\text{Е}_{\text{прд}}$=30245,88*(1+0,16)^2=40698,86 грн.

6) Термін окупності витрат на програмний продукт(років)

$$Ток = \frac{К}{Еекс} \quad (4.22)$$

$$Ток = \frac{17203{,}96}{14391{,}31} = 1{,}19$$

Результуючі показники економічної ефективності зводяться в таблицю 4.6.

Таблиця 4.7

Показники економічної ефективності проектного рішення

Найменування показників	Одиниці вимірювання	Значення показників	
		Аналог	Проектне рішення
1	2	3	4
Капітальні вкладення	грн.		17203,96
Ціна придбання	грн.	49641,54	26467,25
Річні експлуатаційні витрати	грн.	16095,04	3688,74

Ціна споживання	грн.	58815,71	28569,83
Економічний ефект в сфері експлуатації	грн.	-	12406,3
Додатковий економічний ефект в сфері експлуатації	грн.	-	14391,31
Економічний ефект в сфері проектування	грн.	-	30325,88
Додатковий економічний ефект в сфері проектування	грн.	-	40698,86
Термін окупності витрат на проектування рішення	Роки	-	1,19
Коефіцієнт конкурентоспроможності	-	-	0,89

Після проведення розрахунків бачимо, що розроблений програмний засіб має переваги в порівнянні з аналогом тому, що його вартість є набагато меншою ніж вартість аналога.

4.7. Висновки

Розглянувши ринок програмного забезпечення та врахувавши всі фактори впливу, виконуємо створення нового програмного рішення автоматизованої системи збору даних про дорожні маршрути.

Після проведення оцінювання продукту з точки зору економіки, можна сказати про те що цей програмний засіб має перспективи на розвиток у майбутньому та зможе принести чималий дохід. Це в свою чергу підтверджується великою цільовою аудиторією та новизною цього продукту. Але на данному етапі конкурентоспроможність розробленого програмного продукту відстає від аналогу.

Головними факторами впливу на проект являються користувачі та конкуренти. Тому для успіху важливо зібрати максимальну кількість інформації про потреби та очікування потенційних користувачів, якими будуть розробники програмного забезпечення.

Отже, дуже важливо врахувати всі вимоги та потреби користувача. Ще одним важливим фактором буде правильний вибір технологій, тому що від цього залежатиме продуктивність даного програмного засобу. Відповідно до того як ми врахуємо всі аспекти і залежатиме якість даного програмного забезпечення.

Аналіз отриманих даних показав, що проект повністю окупить себе через 1 рік використання. Також після проведення розрахунків програмний засіб для підвищення якості індивідуального процесу розробки програмного забезпечення має переваги в порівнянні з аналогом тому, що є більш дешевим та зручним у використанні, гнучким та надійним. Одною з головних переваг можна назвати більшу продуктивність системи в порівнянні з аналогом.

ВИСНОВКИ

В даній дипломній роботі було розроблено автоматизовану систему для збору даних про дорожні маршрути.

Результатом детального огляду та аналізу сучасних методів проектування баз даних, технологій та методологій побудови новітніх програмних рішень для реалізації веб-додатків та автоматизованих систем, став висновок про те, що найкращим варіантом для реалізації бази даних автоматизованої системи для збору інформації про дорожні маршрути стане реляційна модель.

Після проведення аналізу новітніх програмних засобів було обрано такі основні засоби для розробки бази даних, серверної та клієнтської частини додатку: MS SQL Server 2012, середовище для розробника Microsoft Visual Studio 2013, технологію побудови об’єктно-реляційної моделі Entity Framework, платформу для побудови веб-застосунківASP.Net MVC, платформу для побудови веб-сервісів Windows Communication Foundation, Bootstrap, AngularJs.

Під час концептуального моделювання предметної області визначено сутності, зв’язки між ними, атрибути сутностей.

Етап прототипіювання інтерфейсу системи був виконаний за допомогою програми Balsamiq Mockups, що дозволило визначити основний функціонал та дані що мають відображатись екрані користувача.

За допомогою платформи WCF, було реалізовано серверну частину додатку, у вигляду веб-сервісу, що дозволяє зробити зв’язок між клієнтом на сервером мультиплатформенним та уніфікувати формати запитів та відповідей.

На реальному прикладі було продемонстровано роботу системи наглядно показано, що реалізація даної системи є актуальною та значно полегшує роботу автомобілістів, та інших осіб, що використовують для пересування автомобільні шляхи загального користування.

Результати економічного аналізу свідчать про те, що розробка системи є доцільною, оскільки забезпечується економія капіталовкладень в розробку програмного продукту та значно покращується річний економічний ефект, який отримується користувачами даної системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Городяненко В. Г. Социологический практикум. Учебно-методическое пособие.—М.: Изд.центр 'Академия', 1999. – 160с.

2. Горшков М.К., Шереги Ф.Э. Прикладная социология: методология и методы. Учебное пособие. - М.: Институт социологии РАН, 2011. 372 с.

3. Фон Нейман Дж., Моргенштерн Е. Теорія ігор і економічна поведінка. -М.: Наука, 1970. - 708 с.

4. ІвановаВ. Ф. та Коль. А. Пошук і збір інформації: Навчальний посібник/ За загал. ред.– К.: Академія Української Преси, 2006. – 308 с

5. Кононенко О.Т. Методи пошуку та обробки інформації //Вісник ДНУ ім. М. Туган-Барановського - № 3 (59), 2013. – 30 с

6. Стаття. Microsoft про новітні розробки в веб світі.[Електронний ресурс]. – Режим доступу:http://ua.korrespondent.net/journal/1231586

7. Берко А.Ю., Верес О.М. Застосування баз даних: Навч. посібник. - Львів: Ліга-Прес, 2011. -210 с.

8. Гайна Г.А. Основи проектування баз даних: Навчальний посібник. − К.: КНУБА, 2010. – 204 с.

9. Світличний О.О., Плотницький С.В. Основи геоінформатики: Навчальний посібник. − Суми: ВТД «Університетська книга», 2006. -256 с.

10. Лекція 5. Бази даних. [Електронний ресурс]. – Режим доступу:http://lib.mdpu.org.ua/e-book/vstup/L5.htm

11. Пасічник В.В., Резніченко В.А. Організація баз даних та знань. – К.: видавнича група BHV, 2011. – 384 с.

12. М’якшило О.М. організація баз даних та знань. Навчальний посібник – К.: НУХТ, 2013 – 148 с.

13. MySQL. [Електронний ресурс]. – Режим доступу:http://uk.wikipedia.org/wiki/MySQL

14. Коротка характеристика деяких СУБД. [Електронний ресурс]. – Режим доступу: http://lib.mdpu.org.ua/e-book/vstup/L6.htm

15. Виейра Р. Программирование баз данных Microsoft SQL Server 2008 для профессионалов. – Диалектика, 2012. – 1066 с.

16. ADO.NET. [Електронний ресурс]. – Режим доступу: http://uk.wikipedia.org/wiki/ADO.NET

17. LINQ to SQL. [Електронний ресурс]. –Режим доступу:http://msdn.microsoft.com/ru-ru/library/bb397926.aspx

18. Лосєв М.Ю., Скорін Ю.І. Особливості використання Entity Framework// Вісник ХНУ,Інформаційні технології в системах – 2012. – № 616. – С. 30–35.

19. Разработка сущностной модели данных с помощью Entity Framework. [Електронний ресурс]. – Режим доступу: http://www.cyberguru.ru/microsoft-net/ado-net/entity-dev.html

20. Microsoft .NET. [Електронний ресурс]. – Режим доступу: http://uk.wikipedia.org/wiki/dotNet

21. Model-View-Controller. [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Model-View-Controller

22. Общие сведения о ASP.NET MVC. [Електронний ресурс]. – Режим доступу: https://msdn.microsoft.com/ru-ru/library/dd381412%28v=vs.108%29.aspx

23. Общие сведения о WCF. [Електронний ресурс]. – Режим доступу:https:// msdn.microsoft.com/library/ms735119(v=vs.90).aspx

24. JSON. [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/JSON

25. Angular JS. [Електронний ресурс]. – Режим доступу: http://angular.ru/

26. Бібліотека jQuery. [Електронний ресурс]. – Режим доступу: http://victoria.lviv.ua/html/gim/4.html

27. Google Maps Distance Matrix. [Електронний ресурс]. – Режим доступу: https://developers.google.com/maps/documentation/distance-matrix/intro?hl=en

28. .Bootstrap.[Електронний ресурс]. – Режим доступу: http://twbs.docs.org.ua/

29. Обзор платформы Microsoft SQL Server 2012. [Електронний ресурс]. – Режим доступу:http://www.osp.ru/resources/izones/mssql/platform/platform_1.html

30. Концептуальне та логічне проектування. [Електронний ресурс]. – Режим доступу: http://bibliofond.ru/view.aspx?id=655892

Додаток А. Об’єктна модель даних

Додаток Б. Структура бази даних

Додаток В. Прототип графічного інтерфейсу користувача

Додаток Г. Графічний інтерфейс користувача

Додаток Д. Концептуальна модель

Додаток Е. Лістинг програми (клас RoadsManager)

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Runtime.InteropServices.ComTypes;
using System.Text;
using System.Threading.Tasks;
using Roads.Common.Models;
using Roads.Common.Models.DataContract;
using Roads.Common.Repositories;
using Roads.DataBase.Model.Models;

namespace Roads.Common.Managers
{
   /// <summary>
   /// The manager for Find Road functionality.
   /// </summary>
   public class RoadsManager : IDisposable
   {
      #region Private fields and constants

      private int _searchingDepth;

      private const char Separator = '-';

      private List<string> _roads;

      /// <summary>
      /// The _map object translations repository
      /// </summary>
      private readonly RoutesRepository _routeRepository;

      #endregion

      #region Constructor and Destructor

      /// <summary>
      /// Initializes a new instance of the <see cref="SmartSuggestionsManager"/> class.
      /// </summary>
      /// <param name="repository">The repository.</param>
      public RoadsManager(RoutesRepository repository)
      {
         _routeRepository = repository;

         Initialize();
      }

      /// <summary>
      /// Initializes a new instance of the <see cref="SmartSuggestionsManager"/> class.
      /// </summary>
      public RoadsManager()
      {
```

```
    _routeRepository = new RoutesRepository();

    Initialize();
}

#endregion

#region Public Methods

/// <summary>
/// Gets the searching depth.
/// </summary>
/// <returns>Returns searching depth</returns>
public int GetSearchingDepth()
{
    return _routeRepository.GetSearchingDepth();
}

/// <summary>
/// Gets the routes.
/// </summary>
/// <param name="startedCityId">The started city identifier.</param>
/// <param name="destinationCityId">The destination city identifier.</param>
public List<string> BuildRoutes(int startedCityId, int destinationCityId)
{
    _roads = new List<string>();

    if (startedCityId == destinationCityId || startedCityId == 0 || destinationCityId == 0)
    {
        return new List<string>();
    }

    var beginPointId = startedCityId < destinationCityId ? startedCityId : destinationCityId;

    var endPointId = destinationCityId > startedCityId ? destinationCityId : startedCityId;

    var pointTree = new PointTree
    {
        ParentId = beginPointId,
        ChildPointIds = _routeRepository.GetAllRelationsIds(beginPointId)
    };

    Build(string.Empty, ref pointTree, endPointId);

    return _roads;
}

/// <summary>Gets the route estimation.</summary>
/// <param name="route">The route string value.</param>
/// <returns>The <see cref="RouteEstimation"/> result.</returns>
public RouteEstimation GetRouteEstimation(string route)
{
    //To do: route string value can be changed on hash value and gat the route from data base.
    var estimation = new RouteEstimation
```

```
        {
            CityPointsIds = route.Split(Separator).Select(int.Parse).ToArray(),
        };

        for (var i = 0; i < estimation.CityPointsIds.Count() - 1; i++)
        {
            var routeNode = _routeRepository.GetRouteNode(
                GetPointId(true, estimation.CityPointsIds[i], estimation.CityPointsIds[i + 1]),
                GetPointId(false, estimation.CityPointsIds[i], estimation.CityPointsIds[i + 1])
                );

            if (routeNode != null)
            {
                estimation.NodesEstimations.Add(GetEstimationForRouteNode(routeNode));
            }
        }

        return estimation;
    }

    /// <summary>
    /// Gets the feedback controls for new feedback.
    /// </summary>
    /// <returns>GetLanguageById list of feedback controls for new feedback</returns>
    public List<HolisticFeedback> GetFeedbackControlsForNewFeedback()
    {
        var feebackControls = new List<HolisticFeedback>();

        foreach (FeedbackItem feedbackItem in
_routeRepository.GetFeedbackControlsForNewFeedback())
        {
            var feedbackControlItem = new HolisticFeedback();

            feedbackControlItem.Value = null;
            feedbackControlItem.UserName = String.Empty;
            feedbackControlItem.FeedbackItem = new FeedbackItemData()
            {
                FeedbackItemId = feedbackItem.FeedbackItemId,
                IsNumeric = feedbackItem.IsNumeric,
                Mandatory = feedbackItem.Mandatory,
                SortNumber = feedbackItem.SortNumber,
                FeedbackModelId = feedbackItem.FeedbackModelId,
                DescriptionTranslationKey = feedbackItem.DescriptionTranslationKey,
                NameTranslationKey = feedbackItem.NameTranslationKey
            };
            feedbackControlItem.FeedbackModel = new FeedbackModelData()
            {
                FeedbackModelId = feedbackItem.FeedbackModel.FeedbackModelId,
                HtmlCode = feedbackItem.FeedbackModel.HtmlCode,
                JavascriptCode = feedbackItem.FeedbackModel.JavascriptCode
            };
            feebackControls.Add(feedbackControlItem);
        }
        return feebackControls;
```

```
        }

        /// <summary>
        /// Gets the route details by hash.
        /// </summary>
        /// <param name="hash">The hash.</param>
        /// <param name="language"></param>
        /// <returns>Return RouteDetailsData object.</returns>
        public RouteDetailsData GetRouteDetailsByHash(string hash, string language)
        {
            if (hash == null)
            {
                throw new ArgumentNullException("hash");
            }
            if (language == null)
            {
                throw new ArgumentNullException("language");
            }

            var routeDetailsData = new RouteDetailsData();
            var trek = _routeRepository.GetTrekByHash(hash);
            if (trek != null)
            {
                string[] cityPoints = trek.Track.Split('-');
                var nodes = new List<RouteNode>();
                for (int i = 0; i < (cityPoints.Length - 1); i++)
                {
                    long pointOrigin;
                    long pointDestination;
                    if (Int64.TryParse(cityPoints[i], out pointOrigin) &&
                        Int64.TryParse(cityPoints[i + 1], out pointDestination))
                    {
                        var newItem = _routeRepository.GetRouteNode(pointOrigin, pointDestination);
                        if (newItem == null)
                        {
                            newItem = _routeRepository.GetRouteNode(pointDestination, pointOrigin);
                            if (newItem != null)
                            {
                                var destCityNode = new CityNode
                                {
                                    CityNodeId = newItem.DestinationCityNode.CityNodeId,
                                    DestinationRouteNodes =
newItem.DestinationCityNode.DestinationRouteNodes,
                                    DestinationTreks = newItem.DestinationCityNode.DestinationTreks,
                                    LanguageKey = newItem.DestinationCityNode.LanguageKey,
                                    OriginRouteNodes = newItem.DestinationCityNode.OriginRouteNodes,
                                    OriginTreks = newItem.DestinationCityNode.OriginTreks,
                                    RegionNode = newItem.DestinationCityNode.RegionNode,
                                    RegionNodeId = newItem.DestinationCityNode.RegionNodeId
                                };
                                int destCityNodeId = newItem.DestinationCityNodeId;
                                var origenCityNode = new CityNode
                                {
                                    CityNodeId = newItem.OriginCityNode.CityNodeId,
```

```
                        DestinationRouteNodes =
newItem.OriginCityNode.DestinationRouteNodes,
                        DestinationTreks = newItem.OriginCityNode.DestinationTreks,
                        LanguageKey = newItem.OriginCityNode.LanguageKey,
                        OriginRouteNodes = newItem.OriginCityNode.OriginRouteNodes,
                        OriginTreks = newItem.OriginCityNode.OriginTreks,
                        RegionNode = newItem.OriginCityNode.RegionNode,
                        RegionNodeId = newItem.OriginCityNode.RegionNodeId
                    };
                    int origenCityNodeId = newItem.OriginCityNodeId;

                    newItem.OriginCityNodeId = destCityNodeId;
                    newItem.OriginCityNode = destCityNode;
                    newItem.DestinationCityNodeId = origenCityNodeId;
                    newItem.DestinationCityNode = origenCityNode;
                }
            }
            nodes.Add(newItem);
        }
        else
        {
            nodes.Add(null);
        }
    }
    foreach (var routeNode in nodes)
    {
        var rdFeedbacks = _routeRepository.GetRoutDetailsFeedbackFor(routeNode,
language);

        routeDetailsData.RouteDetailsItems.Add(rdFeedbacks);
    }
  }
  return routeDetailsData;
}

/// <summary>
/// Gets the hash for route.
/// </summary>
/// <param name="trek">The trek.</param>
/// <returns>Hash value for route.</returns>
public string GetHashForRoute(string trek)
{
  return string.Format("{0:X8}", trek.GetHashCode());
}

/// <summary>
/// Gets the routes search result.
/// </summary>
/// <param name="startedCityId">The started city identifier.</param>
/// <param name="destinationCityId">The destination city identifier.</param>
/// <param name="page">The page.</param>
/// <param name="languageName">Name of the language.</param>
/// <returns>
/// The <see cref="RoutesSearchResultData" /> result.
```

```
        /// </returns>
                    public async Task<RoutesSearchResultData> GetRoutesSearchResult( int
startedCityId, int destinationCityId, int page, string languageName, bool isRouteValidation )
        {
            BuildRoutes(startedCityId, destinationCityId);

            var result = await _routeRepository.GetRoutesSearchResultData(
                GetPointId(true, startedCityId, destinationCityId),
                GetPointId(false, startedCityId, destinationCityId),
                page,
                                        languageName, isRouteValidation );

            return result;
        }

        /// <summary>
        /// Adds the new feedback and get URL to route.
        /// </summary>
        /// <param name="routesNodeWithFeedbacksData">The routes node with feedbacks
data.</param>
        /// <returns>Return URL to route with new feedbacks</returns>
        public string AddNewFeedbackAndGetUrlToRoute(List<RouteNodeWithFeedbacksData>
routesNodeWithFeedbacksData)
        {
            RoutesRepository routeRepository = new RoutesRepository();
            FeedbackItemRepository feedbackItemRepository = new FeedbackItemRepository();
            StringBuilder track = new StringBuilder(routesNodeWithFeedbacksData.Count * 5 +
routesNodeWithFeedbacksData.Count - 1);

            bool backwardOrder = routesNodeWithFeedbacksData[0].OriginCityNodeId >
                    routesNodeWithFeedbacksData[routesNodeWithFeedbacksData
                    .Count - 1].DestinationCityNodeId;

            routesNodeWithFeedbacksData.ForEach(routeWithFeedback =>
            {
                RouteNode routeNode = routeRepository.GetRouteNode(
                routeWithFeedback.OriginCityNodeId, routeWithFeedback.DestinationCityNodeId) ??
                    new RouteNode()
                    {
                        RouteNodeId =
routeRepository.CreateRouteNode(routeWithFeedback.OriginCityNodeId,
                            routeWithFeedback.DestinationCityNodeId)
                    };

                int feedbackId = feedbackItemRepository.AddNewFeedback(routeNode.RouteNodeId,
routeWithFeedback.UserId,
                    routeWithFeedback.SubmitTime);

                routeWithFeedback.FeedbackValues.ForEach(feedbackValue =>
feedbackItemRepository.AddNewFeedbackValue(new FeedbackValueData()
                {
                    FeedbackId = feedbackId,
                    Value = feedbackValue.Value,
                    FeedbackItemId = feedbackValue.FeedbackItemId
```

```
            }));

            if (!backwardOrder)
            {
               track.Append(routeWithFeedback.OriginCityNodeId).Append("-");
               if (routeWithFeedback.Equals(routesNodeWithFeedbacksData.Last()))
               {
                  track.Append(routeWithFeedback.DestinationCityNodeId);
               }
            }
            else
            {
               track.Insert(0, routeWithFeedback.OriginCityNodeId).Insert(0, "-");
               if (routeWithFeedback.Equals(routesNodeWithFeedbacksData.Last()))
               {
                  track.Insert(0, routeWithFeedback.DestinationCityNodeId);
               }
            }
         });
         BuildRoutes(routesNodeWithFeedbacksData[0].OriginCityNodeId,
            routesNodeWithFeedbacksData[routesNodeWithFeedbacksData.Count -
1].DestinationCityNodeId);

         string hash = routeRepository.GetHashByTrack(track.ToString());
         return String.Format("{0}-{1}-To-{2}", hash,
            routesNodeWithFeedbacksData.First().OriginCityNode,
            routesNodeWithFeedbacksData.Last().DestinationCityNode)
            .Replace(" ", "-")
            .Replace(",", "");
      }

      /// <summary>
      /// Creates the feedback.
      /// </summary>
      /// <param name="feedbacksData">The feedbacks data.</param>
      public void CreateFeedback(RouteNodeWithFeedbacksData feedbacksData)
      {
         var feedbackItemRepository = new FeedbackItemRepository();
         var routeRepository = new RoutesRepository();
         RouteNode routeNode = routeRepository.GetRouteNode(
            feedbacksData.OriginCityNodeId, feedbacksData.DestinationCityNodeId);

         int feedbackId = feedbackItemRepository.AddNewFeedback(routeNode.RouteNodeId,
feedbacksData.UserId,
               feedbacksData.SubmitTime);

         feedbacksData.FeedbackValues.ForEach(feedbackValue =>
feedbackItemRepository.AddNewFeedbackValue(new FeedbackValueData()
         {
            FeedbackId = feedbackId,
            Value = feedbackValue.Value,
            FeedbackItemId = feedbackValue.FeedbackItemId
         }));
      }
```

```
#endregion

#region Private Methods

/// <summary>
/// Saves the route.
/// </summary>
/// <param name="trek">The trek.</param>
private void SaveRoute(string trek)
{
   _routeRepository.AddTrekIfNotExist(new Trek
   {
      Track = trek,
      Hash = GetHashForRoute(trek),
      OriginCityNodeId = GetPointId(true, trek),
      DesinationCityNodeId = GetPointId(false, trek),
      NodesCount = (byte)trek.Split('-').Count(),
      TrekDate = DateTime.Now
   });
}

/// <summary>
/// Gets the point identifier.
/// </summary>
/// <param name="isBegin">if set to <c>true</c> [is begin].</param>
/// <param name="pointId1">The point id1.</param>
/// <param name="pointId2">The point id2.</param>
/// <returns>Point Id.</returns>
private int GetPointId(bool isBegin, int pointId1, int pointId2)
{
   if (isBegin)
   {
      return pointId1 < pointId2
         ? pointId1
         : pointId2;
   }
   return pointId1 > pointId2
      ? pointId1
      : pointId2;
}

/// <summary>
/// Gets the point identifier.
/// </summary>
/// <param name="isBegin">if set to <c>true</c> [is begin].</param>
/// <param name="trek">The trek.</param>
/// <returns>Point Id.</returns>
private int GetPointId(bool isBegin, string trek)
{
   if (!string.IsNullOrEmpty(trek))
   {
      var points = trek.Split(Separator).Select(int.Parse).ToList();
```

```
            return isBegin ? points.First() : points.Last();
        }
        return 0;
    }

    /// <summary>
    /// Gets the estimation for route node.
    /// </summary>
    /// <param name="node">The node.</param>
    /// <returns>The <see cref="RouteNodeEstimation"/> estimation.</returns>
    private RouteNodeEstimation GetEstimationForRouteNode(RouteNode node)
    {
        return new RouteNodeEstimation
        {
            RouteNode = new RouteNodeData
            {
                DestinationCityNodeId = node.DestinationCityNodeId,
                OriginCityNodeId = node.OriginCityNodeId,
                RouteNodeId = node.RouteNodeId
            },

            HolisticFeedbacks = _routeRepository.GetHolisticFeedbackFor(node, null)
        };
    }

    /// <summary>
    /// Build the specified trek.
    /// </summary>
    /// <param name="trek">The string formatted trek.</param>
    /// <param name="step">The <see cref="PointTree"/> instance.</param>
    /// <param name="destinationCityId">The destination city identifier.</param>
    private void Build(string trek, ref PointTree step, long destinationCityId)
    {
        trek = string.IsNullOrEmpty(trek) ? step.ParentId.ToString(CultureInfo.InvariantCulture) :
string.Format("{0}-{1}", trek, step.ParentId);

        if (step.ParentId == destinationCityId)
        {
            if (_roads.Count(s => s == trek) == 0)
            {
                _roads.Add(trek);

                SaveRoute(trek);
            }
        }
        else
        {
            if (CheckTrekDepth(trek))
            {
                while (step.ChildPointIds.Count != 0)
                {
                    var nexPointId = step.ChildPointIds.First();

                    var newLevel = new PointTree
```

```
                {
                    ParentId = nexPointId,
                    ChildPointIds = GetRelations(trek, nexPointId)
                };

                step.ChildPointIds.Remove(nexPointId);

                Build(trek, ref newLevel, destinationCityId);
            }
        }
    }
}

/// <summary>
/// Gets GetRelations.
/// </summary>
/// <param name="trek">The trek.</param>
/// <param name="id">The id.</param>
/// <returns>List of relation id's.</returns>
private List<int> GetRelations(string trek, int id)
{
    var visited = trek.Split(Separator).ToList().Select(int.Parse).ToList();
    var relations = _routeRepository.GetAllRelationsIds(id);
    relations = relations.Where(e => !visited.Contains(e)).ToList();
    return relations;
}

/// <summary>
/// Checks the trek deeps.
/// </summary>
/// <param name="trek">The trek.</param>
/// <returns> The <see cref="bool"/> value true if trek deep no biggest then <see
cref="_searchingDepth"/> value.</returns>
private bool CheckTrekDepth(string trek)
{
    return trek.Split(Separator).Count() <= _searchingDepth;
}

/// <summary>
/// Initializes this instance.
/// </summary>
private void Initialize()
{
    _searchingDepth = _routeRepository.GetSearchingDepth();
}
#endregion

#region IDisposable implemetation
public void Dispose()
{
}
#endregion
  }
}
```