Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка та інтеграція модуля обслуговування автопарку в програмне забезпечення логістичної компанії

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент гр. ІПЗ-23-1м

_____ /Б.В. Заіка/

Керівник кваліфікаційної роботи _____ / А. А.Трачук /

Економіко-організаційна частина _____ /_____/

Нормоконтроль _____ /_____/

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг

2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: магістр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А. М. Стрюк

«___» _____ 20__ р

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-23-1м Заіці Богдану Володимировичу

1. Тема: Розробка та інтеграція модуля обслуговування автопарку в програмне забезпечення логістичної компанії
   затверджено наказом по КНУ № 277с від «14» квітня 2024 р
2. Термін подання студентом закінченої роботи: «5» грудня 2024р.
3. Вихідні дані по роботі: огляд та дослідження ефективності використання програмного забезпечення для управління обслуговуванням транспортних засобів; серверний застосунок, що збирає та аналізує дані, зберігає їх у базі даних; веб застосунок що виконує роль клієнтської частини розробленої системи та допомагає візуалізувати та спростити роботу з модулем.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): виконати аналіз предметної області та існуючих аналогів, сформувати результати дослідження, сформувати список функціональних вимог до застосунку, розробити структуру баз даних, спроектувати архітектуру системи, здійснити програмну реалізацію розробленої системи.
5. Перелік ілюстративного матеріалу: діаграма бази даних, діаграма використання, діаграма розгортання.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літератури за тематикою та збір даних	14.04.2024-25.04.2024
2	Проведення аналізу предметної області та існуючих аналогів	26.04.2024-10.05.2024
3	Проектування функціональної частини додатку, його архітектури, вибір інструментів	11.05.2024-25.05.2024
4	Підготовка матеріалів першого розділу роботи	26.05.2024-15.06.2024
5	Підготовка матеріалів другого розділу роботи	16.06.2024-25.06.2024
6	Розробка програмного забезпечення	26.06.2024-29.09.2024
7	Підготовка матеріалів третього розділу роботи	30.09.2024-10.10.2024
8	Підготовка матеріалів четвертого розділу роботи	11.10.2024-21.10.2024
9	Оформлення пояснювальної записки	22.10.2024-30.11.2024

Дата видачі завдання «__» _____ 2024 р.

Студент _____ / Б. В. Заіка/

Керівник роботи _____ / А. А.Трачук /

РЕФЕРАТ

АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, МІКРОСЕРВІСИ, АНАЛІЗ ВИМОГ, РОЗРОБКА СЕРВЕРІВ, ЛОГІСТИЧНА КОМПАНІЯ

Пояснювальна записка: 70с., 2 табл., 12 рис., 1 дод., 5 джерел.

У кваліфікаційній роботі виконано дослідження, розробку та інтеграцію модуля обслуговування автопарку в програмне забезпечення логістичної компанії.

У першому розділі проведено аналіз предметної області та існуючих програм-конкурентів. Описано функції та задачі систем обслуговування автопарків як підвиду систем підтримки прийняття рішень. Проведено огляд популярних рішень, із зазначенням їх сильних і слабких сторін.

У другому розділі визначено вимоги до системи та створено математичні моделі. Розроблено схему бази даних, Описано математичні моделі для прогнозування витрат палива та оптимальної вартості використання транспортних засобів.

Третій розділ присвячено реалізації програмного забезпечення. Реалізовано функції обліку транспортних засобів і моніторингу їх технічного стану, інструменти прогнозування витрат на обслуговування та розрахунку вартості життєвого циклу транспортного засобу, користувацький інтерфейс.

У четвертому розділі проведено аналіз економічної ефективності розробки. Розраховано зниження витрат на ремонти та простої, оптимізацію використання автопарку. Оцінено вартість виконання розробки й загальний економічний ефект від впровадження модуля, що виявився значним.

Результатом роботи стало впровадження модуля обслуговування автопарку, який підвищує ефективність управління транспортом, знижує витрати та сприяє конкурентоспроможності логістичної компанії. Система демонструє високий потенціал для подальшого вдосконалення, зокрема в напрямі інтеграції додаткових аналітичних інструментів.

ABSTRACT

SOFTWARE ARCHITECTURE, MICROSERVICES, REQUIREMENTS ANALYSIS, SERVER DEVELOPMENT, LOGISTICS COMPANY

Explnatory note: 70p, 2 tab., 12 fig., 1 app., 5 references

The qualification work involves the research, development, and integration of a fleet maintenance module into the logistics company's software.

In the first chapter, the subject area and existing competing programs are analyzed. The functions and tasks of fleet maintenance systems as a subtype of decision support systems are described. A review of popular solutions is conducted, highlighting their strengths and weaknesses.

The second chapter defines the system requirements and develops mathematical models. A database schema is designed, and mathematical models for fuel cost forecasting and optimal vehicle usage cost estimation are described.

The third chapter focuses on software implementation. Functions for vehicle accounting and monitoring of their technical condition have been implemented, along with tools for maintenance cost forecasting and life cycle cost (LCC) calculation, and a user-friendly interface.

The fourth chapter analyzes the economic efficiency of the development. Calculations are made to estimate the reduction in repair costs and downtimes, as well as the optimization of fleet utilization. The cost of development and the overall economic impact of the module implementation, which proved significant, are assessed.

The result of the work is the implementation of a fleet maintenance module that enhances transport management efficiency, reduces costs, and boosts the competitiveness of the logistics company. The system demonstrates strong potential for further improvement, particularly in the integration of additional analytical tools.

ЗМІСТ

ВСТУП

Розвиток сучасного бізнесу неймовірно сильно залежить від якісної системи логістики. Успіх «Нової пошти», який почався ще в Україні і зараз поступово розповсюджується на країни Європи, дозволяє зрозуміти, що старі підходи до транспортування не працюють, клієнт не хоче чекати тиждень і готовий доплачувати за швидку доставку. Швидкість обробки замовлень у «Нової пошти» зумовлена в першу чергу завдяки впровадженню сучасних підходів до створення рейсів та плануванню завантаження вантажівок на ходу, реагуючи на кожну нову посилку. Цей досвід можна масштабувати на будь-яку частину логістичної компанії: починаючи від обробки замовлень, створення рейсів, так і до роботи з позаштатними ситуаціями та управлінням станом автопарку. Саме про розробку подібного модуля для вже існуючої логістичної компанії і буде йтись в даній роботі. Модуль, що буде розроблено, виконується в контексті вже готової системи для логістичної компанії. Це означає, що більшість загального функціоналу, такого як маршрутизація, прийом замовлень, системи моніторингу, вже реалізовані і тепер стоїть задача розширити функціонал програмного забезпечення. Окрім вже готового функціоналу у системі заготовлений певний стиль оформлення кодової бази, архітектура застосунку, підходи до комунікації між модулями, вибір інструментів для розробки, тощо. Це спрощує розробку нового модуля, скоротивши час на планування, але в той же час обмежує у виборі.

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ ПРОГРАМ-КОНКУРЕНТІВ

1.1 Опис системи обслуговування автопарку

Система обслуговування автопарку представляє собою допоміжний інструмент для контролю технічного стану транспортних засобів, їх працездатності. Цей інструмент допомагає водіям, механікам та менеджерам контролювати витрати та зменшити документооборот на підприємстві. Також система допомагає покращити безпеку використання транспорту в автопарку та знизити ризики аварій, поломок в дорозі та крадіжок авто. Зниження ризиків використання автопарку допоможе здешевити кілометр шляху як напряму (у справного транспорту витрати палива нижчі), так і не напряму (ремонт у дорозі зазвичай менш якісний і дорожчий, ніж у пункті базування штатним механіком). Для механіків система працює більшою мірою як покрокова інструкція та нагадування про необхідність проведення тих чи інших перевірок. Також вона звільняє від заповнення великих об’ємів актів та підвищує ефективність їх роботи по прямим обов’язкам. Для менеджерів та аналітиків система дозволить краще контролювати стан автопарку та покращить прогнозованість роботи системи. Дозволить оптимізувати витрати на оновлення автопарку за рахунок прогнозування вартості кілометра. В загальному вигляді впровадження системи обслуговування автопарку має такі переваги для бізнесу:

- Зниження витрат на паливо завдяки кращому технічному стану автомобіля.
- Скорочення витрат на ремонти через своєчасне ТО.
- Швидке прийняття рішень на основі точних даних.
- Контроль технічного стану авто.
- Запобігання аваріям завдяки моніторингу водіїв, їх поведінки на дорозі.
- Чіткий облік всіх витрат і подій.
- Мінімізація несанкціонованого використання транспортних засобів.

- Контроль за витратами палива, уникнення зливання палива самими водіями.
- Менше ручної роботи.
- Мінімізація людських помилок у звітності та плануванні.

1.2 Задачі системи обслуговування автопарку

Система що розглядається повинна вирішувати цілий ряд задач. В першу чергу система повинна надавати доступ до інструментів для відслідковування технічного стану транспортних засобів. Пробіг, витрачене паливо, історія поломок та ремонтів.

Наступним етапом стануть інструменти для механіків: графіки зміни мастила, зміни шин на сезонні, інші планові ремонти, запити на діагностику або ремонт від водіїв, аналітична інформація на основі датчиків авто що може допомогти знайти проблему, електронні акти.

Менеджери повинні мати змогу ознайомитись з історією призначень водіїв, графіками пробігу та рівня пального, історією місцезнаходження, датчиками відкриття дверей, гаражі приписки вантажівок, дані по ефективності тих чи інших водіїв та вантажівок, тощо.

Крім того, програмне забезпечення такого рівня повинне мати змогу частково або повністю прибрати з життя людей бюрократію, дозволяючи ознайомлюватись з інформацією в зручному форматі, та, за необхідністю формувати звіти на основі даних з системи.

Підсумовуючи можна сказати що подібні системи мають такий функціонал:

- Відстеження місцезнаходження авто через GPS.
- Контроль пробігу, палива, технічного стану.
- Управління графіком використання транспортних засобів.
- Автоматичні нагадування про ТО.
- Планування ремонтних робіт.
- Ведення історії технічного обслуговування.

- Облік витрат на паливо, запчастини, ремонт.
- Аналіз ефективності використання автопарку.
- Аналіз поведінки водіїв (швидкість, різкі гальмування).
- Підтримка дисципліни серед персоналу.
- Генерація звітів про витрати, ефективність, аварійність.
- Інтеграція з бухгалтерськими системами.

1.3 Система обслуговування автопарку як підвид системи підтримки прийняття рішень

Найголовнішою задачею системи залишається бути в ролі системи підтримки прийняття рішень. З одного боку системи водії, механіки, оператори та датчики на транспорті для збору даних та оперативного прийняття рішень у випадку нагальної необхідності. З іншого боку фінансисти, менеджери можуть на основі отриманих даних приймати рішення щодо оновлення автопарку, слідкувати за виконанням планових ремонтів, аналізувати і виявляти причини виникнення проблем.

Система дозволяє отримувати багато важливих метрик: частка непланових ремонтів від загальної, динаміку витрат палива, загальні витрати

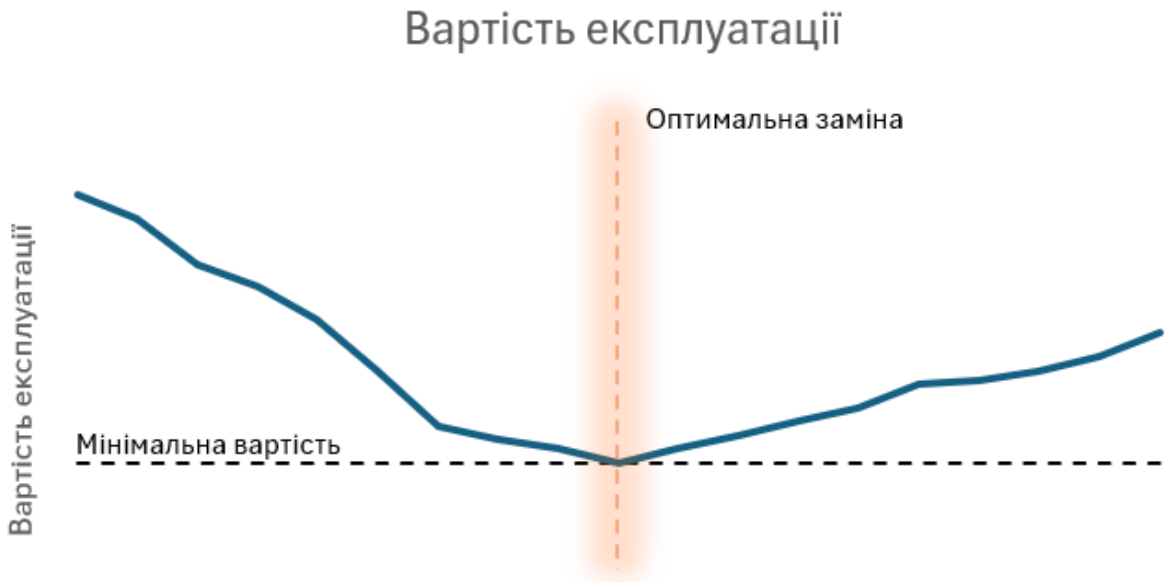

Рисунок 1.1. Графік оптимізації вартості експлуатації в залежності від пробігу

на транспортний засіб, розподілення цих витрат по часу, тощо. Але найголовнішою метрикою яка допомагає зменшити витрати і планувати бюджети залишається вартість експлуатації (Рисунок 1.1). Вона враховує загальні витрати на автомобіль в залежності від його загального пробігу. Метрика враховує ще й дохід від продажу ТЗ під час його заміни.

В зоні оптимальної заміни спостерігається найбільша вигода від використання транспортного засобу: компанія ще не встигла витратити багато грошей на ремонти і машина ще коштує досить багато на ринку вторинного транспорту. Але при цьому авто встигло принести багато користі при низьких ризиках позаштатних ситуацій.

В загальному вигляді маємо такі характеристики системи обслуговування автопарку які дозволяють нам казати про те що це є прикладною системою підтримки прийняття рішень:

Автоматизація прийняття рішень: система обробляє велику кількість даних (про транспорт, водіїв, витрати тощо) та пропонує оптимальні рішення на основі аналітичних моделей.

Інформаційна підтримка: система надає користувачам актуальну та релевантну інформацію, наприклад, стан транспортних засобів, рівень витрат на паливо, терміни проведення технічного обслуговування.

Аналіз і прогнозування: система дозволяє аналізувати історичні дані та прогнозувати можливі проблеми, як-от поломки автомобілів або перевищення витрат, що допомагає ухвалювати рішення на випередження.

Моделювання сценаріїв: система може моделювати різні сценарії, наприклад, вплив зміни графіку ТО на витрати та ефективність автопарку.

Підтримка стратегічного планування: система інтегрується з іншими бізнес-системами, допомагаючи узгоджувати плани розвитку автопарку із загальною стратегією підприємства.

Таким чином ми впевнились в тому що для розробки системи обслуговування автопарку є класичним прикладом системи підтримки

прийняття рішень і всі підходи до побудови такої системи збігаються з загальними.

1.4 Аналіз програм-конкурентів

Люди вже давно використовують транспорт для ведення підприємницької діяльності, тому побудова достатньо потужної системи обліку та обслуговування транспорту була нагальною ще з давніх часів.

Для базових функцій подібного програмного забезпечення цілком вистачить можливостей програми по типу Microsoft Excel, проте для більш складних задач без спеціалізованого програмного забезпечення не обійтись.

Більшість подібного програмного забезпечення великі компанії розробляють власними силами для себе, проте існують різноманітні аналоги за різними форматами надання доступу: від рішень під ключ, з повною передачею програмного забезпечення, аж до сучасних SaaS рішень.

1.4.1 Fleetio

Першою програмою-аналогом з якою є сенс ознайомитись є Fleetio. Сучасна програма яка включає в себе веб та мобільний застосунки і

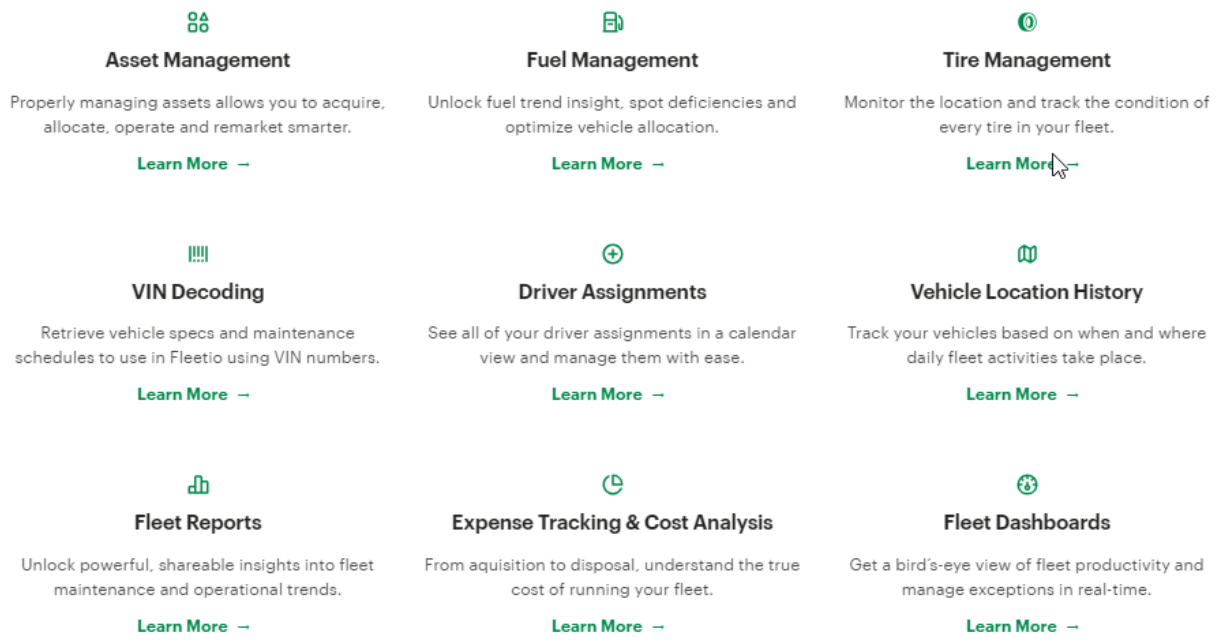

Рисунок 1.2. Частина з можливого функціоналу програми Fleetio

віддаленим сервером обслуговуванням якого займаються розробники компанії. Послуги надаються за схемою SaaS. В арсеналі цього застосунку є досить велика кількість інструментів (Рисунок 1.2): менеджмент регулярних ремонтів, нагадування про зміну мастил та шин, зчитування інформації про авто з його VIN коду, історія місцезнаходження ТЗ (для цих цілей компанія навіть надає GPS-трекери, що спрощує впровадження системи для кінцевого клієнта), контроль витрат, призначення водіїв на певні транспортні засоби, складання відповідних графіків.

Крім того, програма дозволяє створювати або обирати з уже готових плани огляду транспортних засобів що спрощує життя механікам. Облік наявних запасних частин також зекономить час для механіків

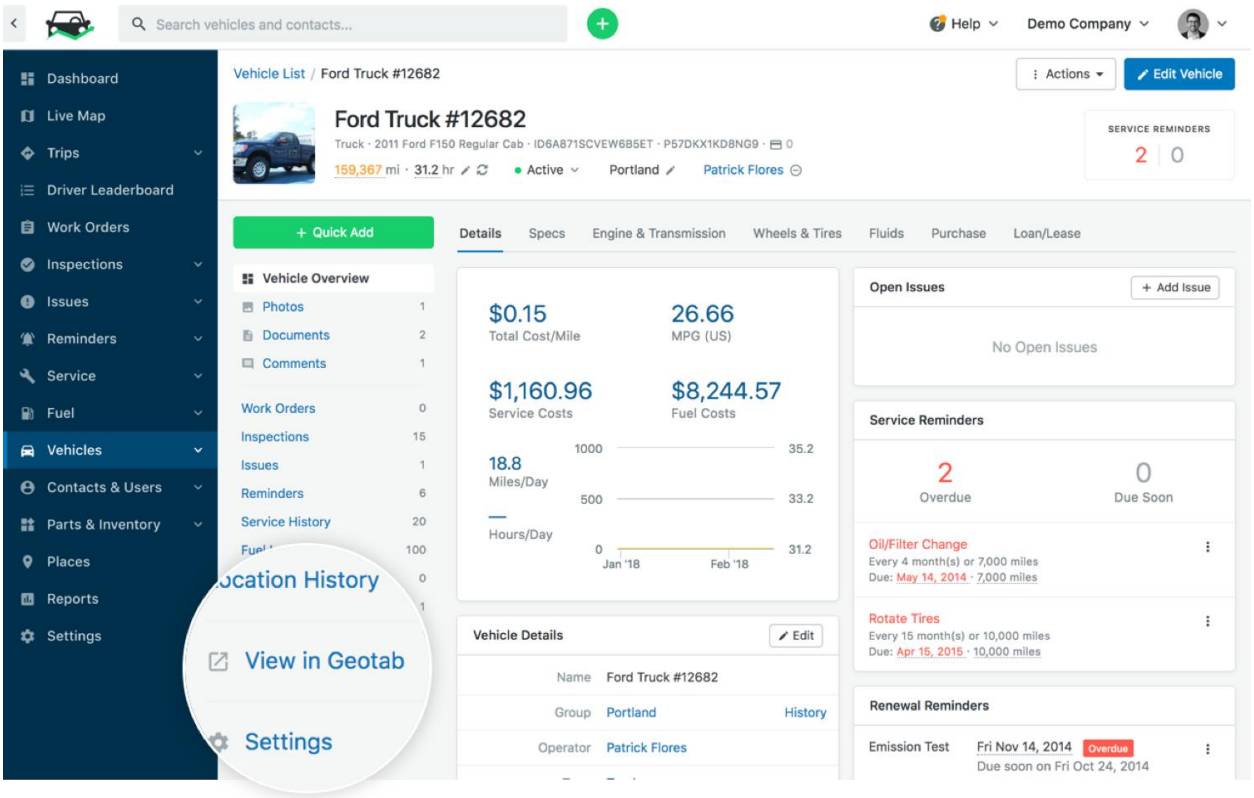

Рисунок 1.3. Приклад інтерфейсу програми Fleetio

Також розробники передбачили можливість інтеграції з програмним забезпеченням самих транспортних засобів при його наявності, що дозволяє знімати необхідні показники без участі водія чи механіка. Крім того, інтеграція з паливними картками допоможе вести облік витрат на паливо та знизить ризики зливу палива водієм чи використання менш якісного палива.

Також для програми доступна публічна специфікація API, яка дає змогу інтегрувати модуль у власні системи на програмному рівні.

Розробники програми пропонують демонстрацію роботи програми на замовлення, що дозволяє краще оцінити перспективи її використання перед прийняттям остаточного рішення.

1.4.2 Intellias Fleet Management solution

Це продукт від української IT-компанії Intellias, який представляє собою замкнену систему обслуговування автотранспорту (Рисунок 1.4). Клієнтом виступала провідна B2B платформа з майже сторічним досвідом роботи яка обслуговує 260 тисяч клієнтів в 50 країнах. На даний момент вона пропонує доступ до найбільшої енергетично-агностичної мережі в Європі, яка включає заправні станції, пункти зарядки електромобілів (як публічні, так і напівпублічні), а також станції альтернативного пального.

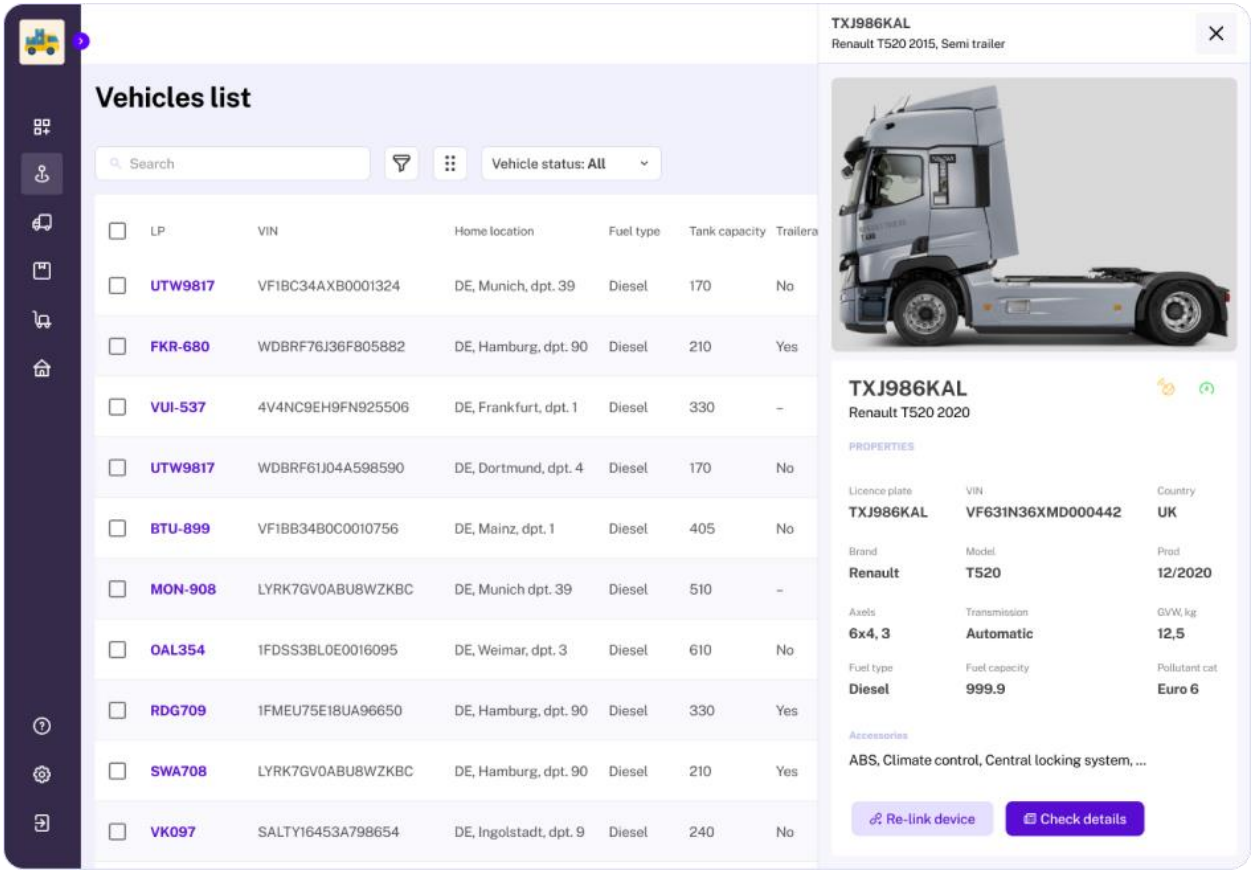

Рисунок 1.4. Інтерфейс застосунку від Intellias

Система включає в себе навігацію, ведення обліку технічного стану транспортних засобів, рівня їх палива тощо. Система робить ставку на різного роду датчики та пристрої ніж на дані від людей. Дозволяє аналізувати поведінку водія на дорозі, виявляти потенційні загрози та сповіщати центральний сервер про такі інциденти. Також наявна оптимізація вартості використання транспорту що дозволяє прогнозувати витрати на перевезення.

Система робить ставку на зручний і простий інтерфейс розроблений з розрахунком на швидкодію та мінімумом відволікаючих елементів. Також було реалізовано мобільний застосунок для водіїв який дозволяє отримувати необхідну інформацію напряму з кабіни (Рисунок 1.5).

Задачею стояло побудувати універсальну систему обслуговування автотранспорту яка б надавала користувачам цінну інформацію для транспортування вантажів на шляху до підвищення ефективності, продуктивності, безпеки та прибутковості.

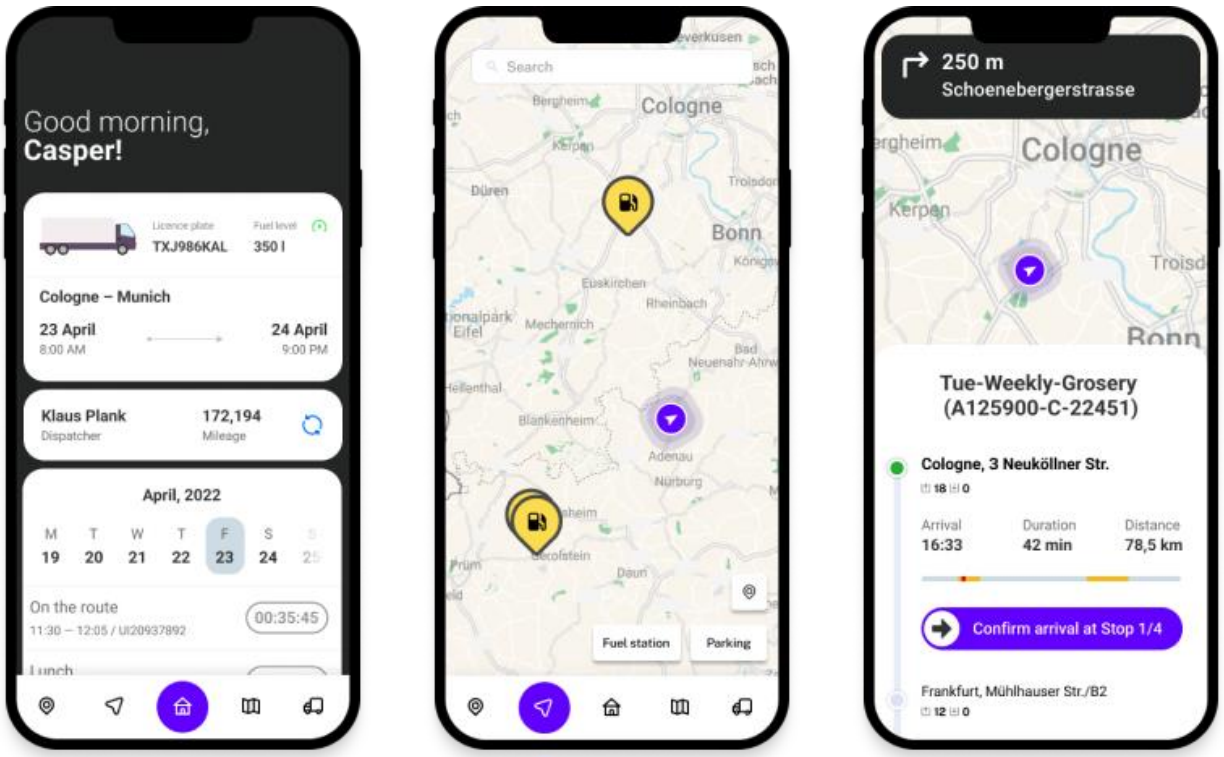

Рисунок 1.5. Мобільний застосунок програми від Intellias

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, СТВОРЕННЯ МАТЕМАТИЧНИХ МОДЕЛЕЙ

2.1 Огляд функціоналу системи

Система ставить на меті збільшення прибутку користувача за рахунок більшої прогнозованості витрат на автопарк та зниження ризиків використання транспорту. Основною перевагою впровадження системи є зменшення впливу людського фактору на виконання робіт, переведення більшої частки ремонтів з термінових в регулярні, збільшення контролю за витратами палива, підвищення безпеки перевезень за рахунок датчиків в транспортних засобах та відслідковування їх геопозиції.

На основі даних що збираються системою створюються відповідні графіки, таблиці та звіти що допомагають уповноваженим особам приймати рішення щодо проведення тих чи інших ремонтів, проведення позачергових діагностик, вчасної заміни транспорту.

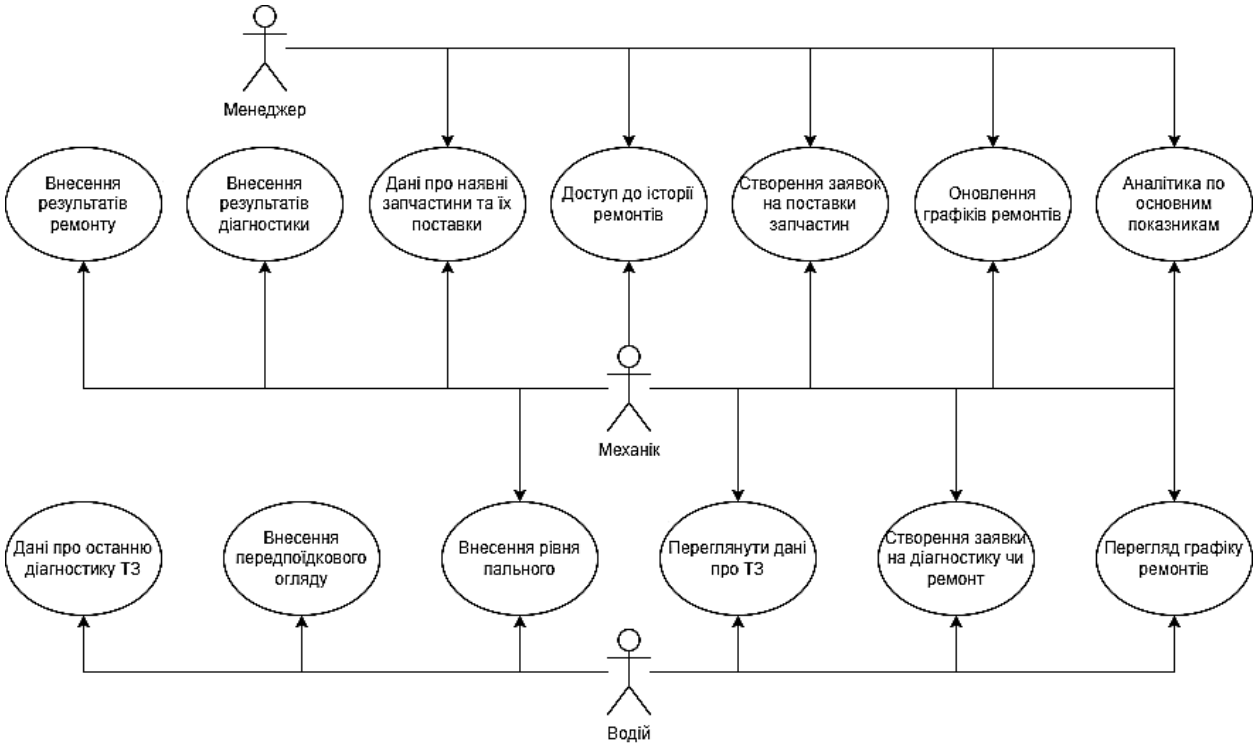

Рисунок 2.6. Діаграма використання

Зі сторони менеджера найбільш важливим функціоналом має стати доступ до аналітики по витратам і доходам, прогнози по витратам, загальна вартість експлуатації транспортного засобу. Крім того, він має мати доступ до всіх транспортних засобів, історію їх обслуговувань та рейсів, моніторинг витрат палива та технічного стану, запасів запчастин.

Механіки мають мати доступ до запитів на обслуговування, результати попередніх діагностик, ремонтів та планів обслуговування.

Водії в рамках даного модуля мають мати можливість переглянути результати попередніх діагностик транспортного засобу що їм призначено, історію ремонтів, вносити рівні палива та запити на ремонт чи діагностику.

На основі сформульованих вимог можна спроектувати діаграму використання (Рисунок 2.6) яка опише необхідний функціонал і дозволить створити подальших план проектування.

2.2 Формування вимог до системи

При формуванні вимог необхідно покладатись на попередньо проведений аналіз предметної області та вироби-аналоги. Крім того, весь запропонований функціонал також диктує вимоги до нього. Окрім вимог викладений в діаграмі використання (Рисунок 2.6) також можна виділити такі вимоги:

- Кросплатформенність застосунку, тобто можливість роботи з ним з якомога більшої кількості платформ.
- Спокійний виважений дизайн. Так як наша розробка буде використовуватись всередині компанії, для нас не настільки важлива візуальна частина застосунку, анімації та прикраси для сторінок. Для нас більш важливим є зручність використання та ненав’язливість дизайну. Нашою програмою люди можуть користуватись впродовж довгого часу, для менеджерів взагалі цілий робочий день пов’язаний з програмою. Крім того, важливо передбачити використання темної теми, адже очікується використання програми вночі. В найкращому випадку можна

взагалі використовувати довільну зміну кольорів інтерфейсу користувачем задля задоволення його потреб. Крім того, елементи інтерфейсу можуть бути довільно масштабовані, адже для різних груп користувачів буде комфортним різний масштаб.

- Швидкість відгуку системи. Система має надавати інформацію якомога швидше і, бажано, в реальному часі для деяких категорій (геолокація та датчики).
- Інтеграція з третіми системами: попередньо створеним ПЗ логістичної компанії та програмним забезпеченням в самих транспортних засобах
- Опціональним є реалізація мобільного застосунку що б дозволило водіям мати більше свободи в використанні ресурсу і не прив’язуватись до ноутбука чи планшета.

2.3 Розробка схеми бази даних

Для виконання проекту нам необхідно обрати потрібний вид бази даних який дозволить зберігати великий об’єм інформації та обробляти його достатньо швидко для великих об’ємів зв’язаних даних. Реляційна база даних підходить для цих цілей достатньо добре.

Основними сутностями в базі даних є:

- Вантажівки (Trucks) – таблиця що містить в собі інформацію про самі транспортні засоби, їх статус, призначеного водія, тощо.
- Обслуговування (Maintenance) – таблиця що містить дані про обслуговування транспортних засобів, як і заплановані ремонти, так і позапланові.
- Роботи (Service) – таблиця що зберігає дані про роботи що можуть бути виконані в рамках обслуговування, наприклад: заміна мастила, діагностика підвіски, заміна сайлентблока.
- Запчастина (Part) – таблиця що зберігає дані про запчастини на складі. Має 2 дочірні сутності: тип деталі та сумісність деталі, що дозволяють краще класифікувати деталі на складі.

Створену схему бази даних наведено нижче (Рисунок 2.7).

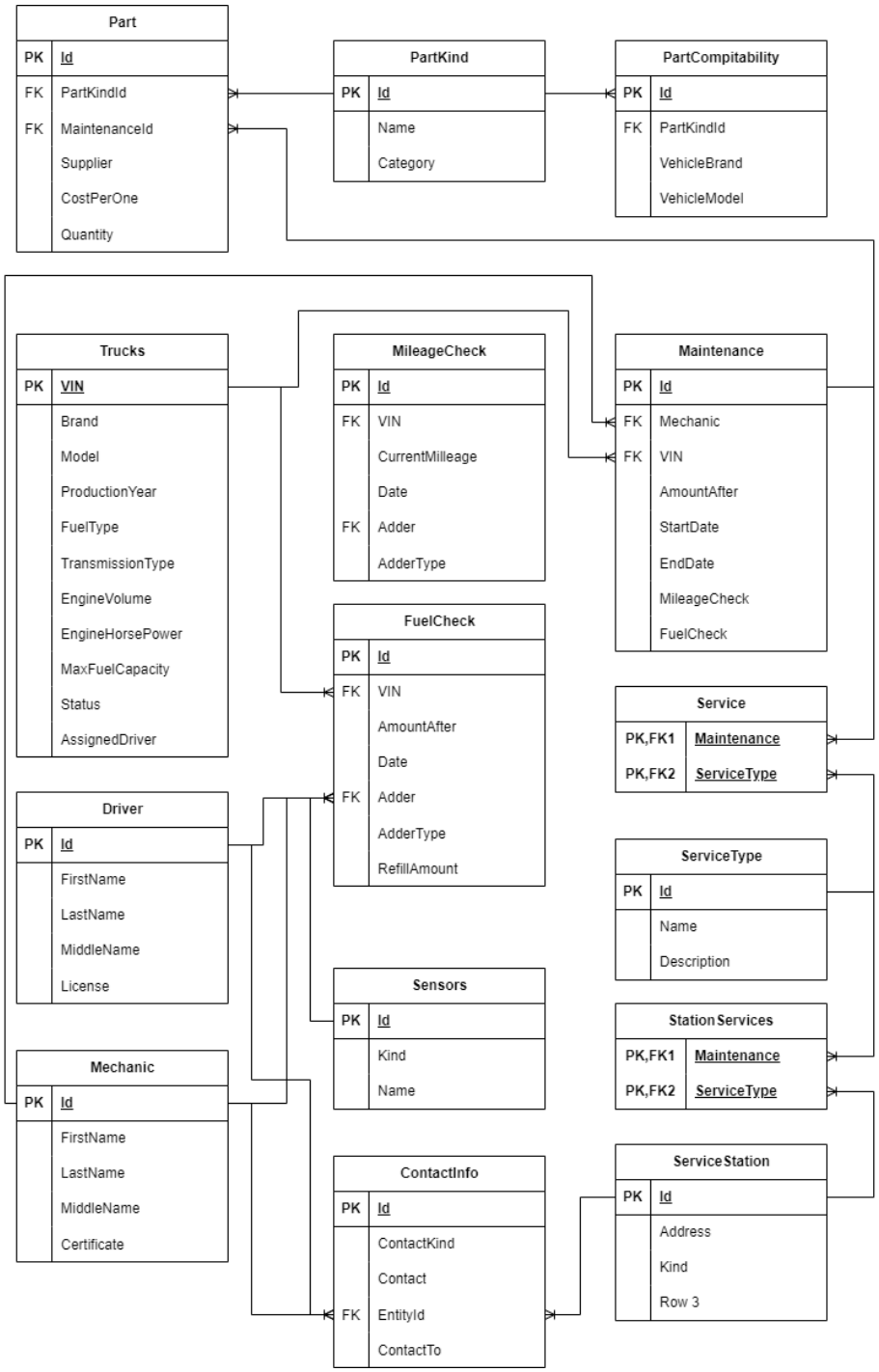

Рисунок 2.7. Схема бази даних для сервісу

2.4 Розгляд математичних моделей що застосовуються в системі

В системі може бути використано досить велику кількість моделей для вирішення тих чи інших задач. Більшість з них можуть бути описані як сума певних показників або знаходження середніх, що не є цікавим для висвітлення.

Тому розглянемо 2 більш цікавих моделі для оптимізації 2 величин: витрати палива та витрати на 1 км шляху.

2.4.1 Прогнозування витрат палива

Прогнозування витрат палива для великогабаритного автотранспорту — це складне завдання, яке враховує велику кількість факторів: технічні характеристики транспортних засобів, умови експлуатації, маршрут, погодні умови тощо. Нижче наведені кілька підходів, які можуть бути використані для вирішення цієї задачі.

Для вирішення задачі можуть бути використані математичні моделі, серед яких варто виділити такі:

Регресійні моделі: Лінійна або нелінійна регресія може бути використана для моделювання взаємозв'язку між витратами палива і факторами, такими як маса вантажу, швидкість, дистанція, тип місцевості тощо.

Поліноміальна регресія: Використовується для врахування нелінійних залежностей між різними факторами (наприклад, вплив швидкості і нахилу дороги на витрати палива).

Іншим способом вирішення завдання є машинне навчання. Серед багатьох підходів найоптимальнішим серед алгоритмів є використання нейронних мереж: Глибинне навчання (deep learning) може бути використане для прогнозування на основі великих наборів даних з високою кількістю факторів. Такі моделі можуть автоматично виявляти складні залежності між параметрами. Додатково такі моделі можуть прогнозувати витрати на ходу використовуючи датчики з самого автомобіля для збору даних в реальному часі (швидкість, оберти двигуна, крутний момент) і будувати прогнози на основі аналізу великих об’ємів даних.

Прогнозування витрат палива може бути інтегроване з геоінформаційними системами, які аналізують маршрут та дорожні умови.

Цей підхід дозволяє враховувати такі фактори, як підйоми, спуски, трафік, які впливають на витрати палива.

Отримані дані можуть бути використані для оптимізації маршрутів і поведінки водія. Алгоритми знаходження оптимального маршруту, які мінімізують витрати палива, можуть бути використані для вибору найбільш економного маршруту з урахуванням типу дороги, підйомів, спусків і зупинок.

Для створення моделі прогнозування витрат палива необхідно накопичити достатньо багато даних по певному транспортному засобу. Серед основних показників можна виділити: тип палива (бензин, дизель), об’єм двигуна (літри), маса автомобіля (тони), тип трансмісії (механічна, автоматична), вік автомобіля, аеродинамічний коефіцієнт, різні параметри двигуна (кількість циліндрів, кінські сили), стан доріг (міська дорога, траса), поведінка водія (середня швидкість, різке гальмування, прискорення), температура довкілля (сезонні коливання).

Для деяких з цих показників можна брати дані з відкритих джерел (стан доріг та температура довкілля може бути взята відповідно з OpenStreetMap та прогнозів погоди). Ці показники та поведінка водія динамічні від поїздки до поїздки, проте модель можна налаштувати для прогнозування витрат по конкретній поїздці маючи дані по ній.

Всі ці показники буде доречно використати в якості параметрів нейронної мережі або для лінійної регресії.

2.4.2 Прогнозування оптимальної вартості використання транспортного засобу

Вартість володіння транспортним засобом – це загальні фінансові витрати, які несе власник протягом усього періоду використання автомобіля. Вона включає як прямі, так і непрямі витрати, пов'язані з придбанням, обслуговуванням та експлуатацією транспортного засобу. Повна вартість володіння (Total Cost of Ownership, TCO) допомагає покупцям автомобілів і

менеджерам автопарків оцінити не лише початкову ціну, але й усі витрати з часом.

Основними компонентами вартості володіння транспортним засобом є:

Знецінення – це зниження вартості автомобіля з часом через зношування, старіння та попит на ринку. Зазвичай автомобілі втрачають приблизно 15-20% своєї вартості щороку.

Витрати на паливо – вартість палива є значною частиною ТСО і залежить від паливної ефективності, типу палива (бензин, дизель або електрика) та цін на пальне.

Технічне обслуговування та ремонти регулярне техобслуговування, заміна запчастин (шини, гальма тощо) і ремонти — це постійні витрати. Дорожчі або складніші автомобілі зазвичай мають вищі витрати на ремонт.

Страхування – страхові премії є обов'язковими для законного використання автомобіля і залежать від типу транспортного засобу, місця використання, та варіантів покриття.

Фінансування (кредит/лізинг) – якщо компанії фінансують покупку через кредити або лізинг, несучи витрати на відсотки або лізингові платежі протягом часу.

Залишкова вартість – ціна, за яку можна продати або обміняти автомобіль. Вона впливає на загальну вартість володіння, частково компенсуючи знецінення.

Для оптимізації вартості володіння транспортним засобом можна створювати моделі, які мінімізують ТСО шляхом вибору найкращого транспортного засобу, стратегій експлуатації та обслуговування.

Лінійне програмування: Цей метод використовує лінійну цільову функцію та обмеження для мінімізації або максимізації певних витрат на володіння. Наприклад, можна мінімізувати загальні витрати, вибравши автомобіль із нижчими показниками знецінення, витратами на пальне та обслуговування при дотриманні бюджету.

Аналіз вартості життєвого циклу (Life-Cycle Cost, LCC). Цей підхід враховує весь життєвий цикл транспортного засобу, від покупки до утилізації. Він оцінює загальні витрати з часом, включаючи ціну покупки, експлуатаційні витрати, техобслуговування та утилізацію. Використовуючи LCC, можна приймати обґрунтовані рішення щодо того, який автомобіль матиме найнижчі ТСО протягом певного періоду (наприклад, 5-10 років).

Моделювання прогнозованого технічного обслуговування. Використовуючи алгоритми машинного навчання, можна прогнозувати, коли деталі, ймовірно, вийдуть з ладу або потребуватимуть ремонту на основі історичних даних. Ця модель знижує ризик несподіваних витрат на ремонт шляхом оптимізації графіків обслуговування та запобігання дорогим поломкам.

2.5 Підбір інструментів та технологій для вирішення поставленої задачі

На вибір інструментів для вирішення задачі найбільший вплив має вибір технологій вже існуючої системи, адже це дозволяє залучити тих самих розробників для виконання завдання, крім того в використанні цих інструментів вже набута експертиза. Все це дозволяє спростити та прискорити написання коду, а також підвищує його якість.

При виборі бази даних треба враховувати необхідність створення процедур та представлень. Тому найбільш правильним вибором стане одна з таких баз даних: Oracle, Microsoft SQL Server, PostgreSQL Pro. Так як для розробки попередньої версії застосунку використовувалась Microsoft SQL Server, то і для цього сервісу буде використана вона ж.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз технічних можливостей для інтеграції з існуючою системою

Основою для розробки нового модуля стане вже готовий проект програмного забезпечення логістичної компанії: її діаграма розгортання (Рисунок 3.8) наведена нижче.

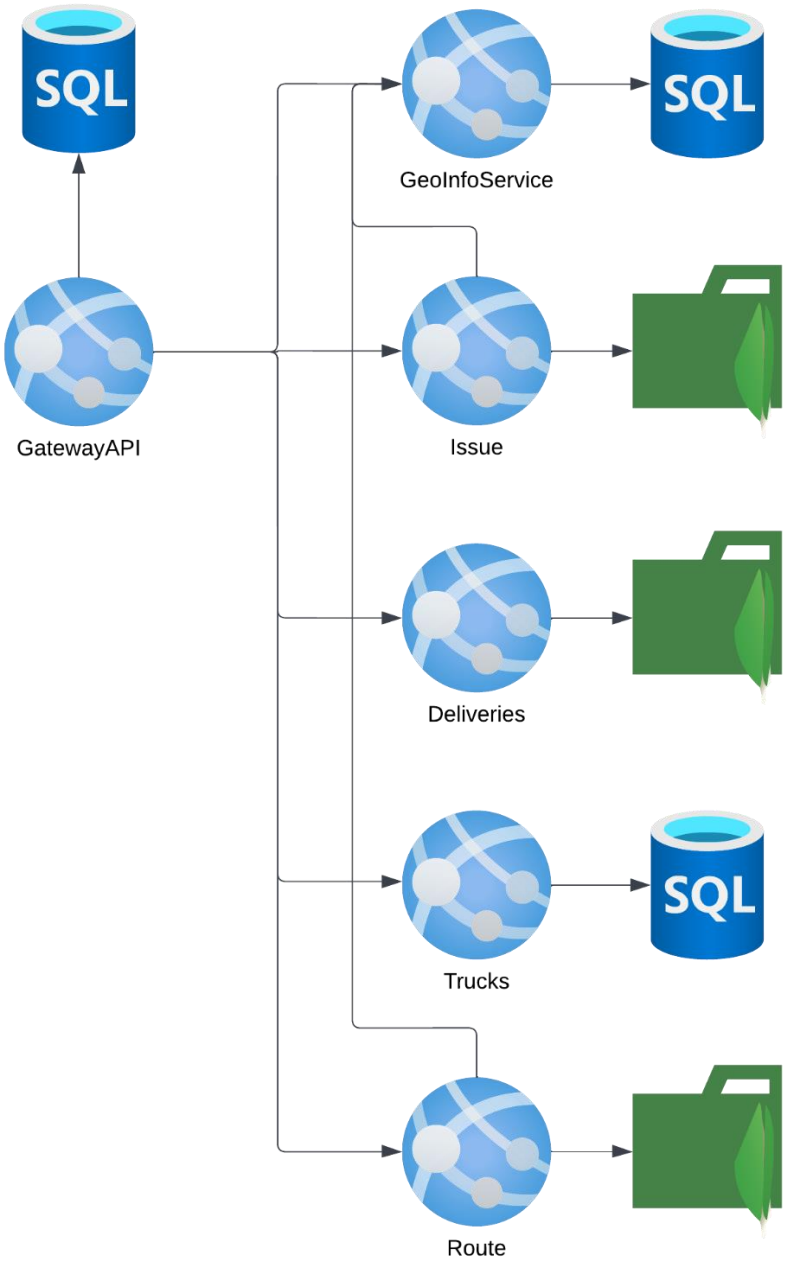

Рисунок 3.8. Діаграма розгортання існуючої системи

З усього існуючого функціоналу системи нас найбільше цікавить сервіс вантажівок. Додатково деякий функціонал модуля може бути залежним від функціоналу сервісу проблем та геоінформаціного сервісу. Крім того, для реалізації нового модуля необхідно розробити додатковий веб та мобільні застосунки, що також розширять функціонал системи.

При аналізі існуючого сервісу вантажівок маємо діаграму сутностей яка наразі присутня в системі було виявлено що частина функціоналу вже була реалізована до того і стоїть задача розширити цей сервіс замість написання нового. Для прикладу наведу ER діаграму вже існуючого сервісу.

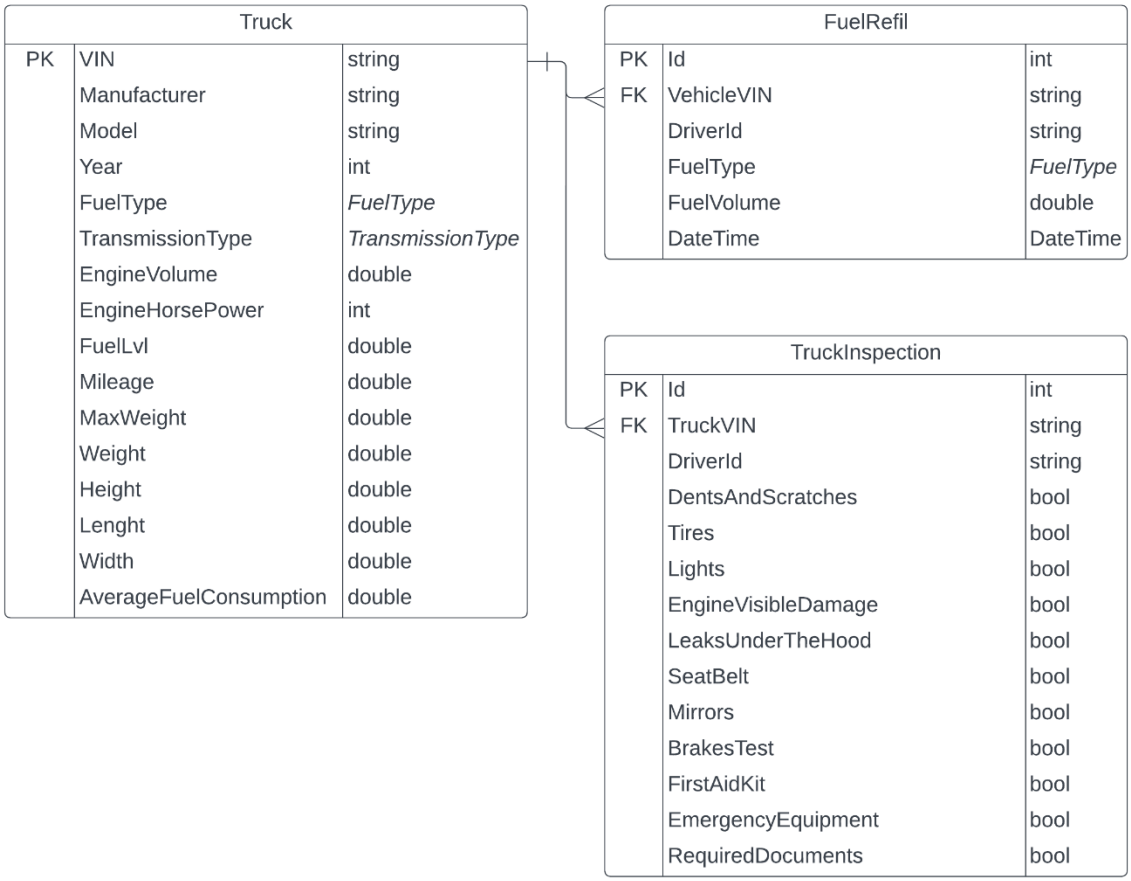

Рисунок 3.9. Діаграма бази даних старої реалізації модуля.

У порівнянні з запланованою реалізацією додається 12 таблиць і додатковий функціонал у вигляді більш детального контролю за запчастинами, дігностикою та ремонтами. Крім того, нова структура таблиць додає економічного контролю за цими даними: людиногодини механіків, ціна запчастин, дані для тренування моделі прогнозу LCC.

3.2 Інтеграція в процеси логістичної компанії

Серед процесів логістичної компанії можна виділити ті, на які буде впливати новостворений модуль можна виділити процеси майже по всьому проекту.

Під час планування маршрутів та призначення потрібного транспортного засобу на маршрут логіст буде бачити поточний стан транспорту і планувати відповідно до орієнтовних термінів завершення робіт з ремонту чи діагностики транспорту. Крім того, буде змога відсіювати транспорт який все ще буде на ремонті на момент відправлення Це дозволить зменшити час на прийняття рішень для логіста за рахунок відсутності необхідності зв’язуватись в асинхронному або синхронному форматі з механіком чи водієм і дозволить обробити заявки швидше.

Під час виконання замовлення модуль обслуговування автотранспорту дозволить логісту визначати точне місцерозташування вантажівки і бачити основні її показники, збільшивши можливості для контролю за статусом поїздки. Крім того, така система дозволяє клієнту розуміти де приблизно знаходиться його вантаж і в якому він статусі.

Якщо щось йде не так, то модуль обробки позаштатних ситуацій може автоматично створювати заявки механікам на ремонт чи діагностику відповідно до ситуації що виникла. Наприклад, водій залишив скаргу на те що в дорозі перестав працювати кондиціонер що впливає на комфорт водія. Менеджер обробляє цю заявку (якщо заявка не оброблена автоматично) та виставляє тег несправності системи у вантажівки. По прибуттю в депо механік буде вже мати сформовану заявку на ремонт системи кондиціонування. Іншим прикладом може бути ДТП яке унеможливлює подальшу доставку вантажу цією ж вантажівкою. На місце звідки йде GPS сигнал від вантажівки можна направити евакуатор та нову вантажівку щоб продовжити виконання замовлення. При цьому нову вантажівку можна обрати із тих що зараз знаходяться найближче до місця ДТП. Таким чином підвищується

відмовостійкість системи доставок і клієнт має менше ризиків які пов’язані з доставкою його вантажів.

Після завершення рейсу механік проводить діагностику, і сповіщає про її хід в систему. Таким чином дані про транспорт оновлюються та менеджер з обслуговування автопарку може виявляти тренди пов’язані з певним транспортним засобом, водієм, маркою або моделлю транспортного засобу. Таким чином покращується якість рішень що приймає менеджер.

3.3 Реалізація обліку транспортних засобів та їх технічного стану

Почнімо з розробки серверної частини застосунку. Для обліку транспортних засобів будуть використовуватись такі таблиці з бази даних: Trucks, MileageChecks, FuelChecks, Maintenances. В цих таблицях зберігається основна інформація: властивості самого транспортного засобу, дані про пробіг та рівень палива, огляди та ремонти. На основі цих даних можна створити опис апі яка буде виконувати основні задачі. Наведу як приклад уривок документації до апі що розроблялась для цих задач.

```
GET /trucks
Опис: Отримання списку всіх транспортних засобів.
Вихідні дані:
json
[
  {
    "id": 1,
    "registrationNumber": "AA1234BB",
    "model": "Ford Transit",
    "year": 2020,
    "status": "active"
  },
  {
    "id": 2,
    "registrationNumber": "AB5678CD",
    "model": "Toyota Corolla",
    "year": 2018,
    "status": "inactive"
  }
]
POST /trucks
Опис: Додавання нового транспортного засобу.
Вхідні дані:
json
{
```

```
  "registrationNumber": "AA1234BB",
  "model": "Ford Transit",
  "year": 2020,
  "fuelType": "diesel",
  "owner": "Company X"
}
Вихідні дані:
json
{
  "id": 1,
  "message": "Truck added successfully."
}
GET /trucks/{id}
Опис: Отримання деталей конкретного транспортного засобу.
Вихідні дані:
json
{
  "id": 1,
  "registrationNumber": "AA1234BB",
  "model": "Ford Transit",
  "year": 2020,
  "fuelType": "diesel",
  "owner": "Company X",
  "technicalState": "good",
  "lastServiceDate": "2024-11-15"
}
PUT /trucks/{id}
Опис: Оновлення інформації про транспортний засіб.
Вхідні дані:
json
{
  "model": "Ford Transit Custom",
  "fuelType": "electric",
  "status": "active"
}
Вихідні дані:
json
{
  "message": "Truck updated successfully."
}
DELETE /trucks/{id}
Опис: Видалення транспортного засобу.
Вихідні дані:
json
{
  "message": "Truck deleted successfully."
}
GET /maintenance
Опис: Отримання списку запланованих технічних обслуговувань.
Вихідні дані:
json
[
  {
```

```
    "id": 101,
    "vehicleId": 1,
    "scheduledDate": "2024-12-01",
    "description": "Oil change",
    "status": "pending"
  },
  {
    "id": 102,
    "vehicleId": 2,
    "scheduledDate": "2024-12-05",
    "description": "Brake inspection",
    "status": "completed"
  }
]
```

POST /maintenance
Опис: Додавання нового запису про технічне обслуговування.
Вхідні дані:
json

```
{
  "vehicleId": 1,
  "scheduledDate": "2024-12-01",
  "description": "Oil change"
}
```

Вихідні дані:
json

```
{
  "id": 101,
  "message": "Maintenance scheduled successfully."
}
```

PUT /maintenance/{id}
Опис: Оновлення статусу технічного обслуговування.
Вхідні дані:
json

```
{
  "status": "completed"
}
```

Вихідні дані:
json

```
{
  "message": "Maintenance status updated."
}
```

GET /monitoring/vehicles/{id}
Опис: Отримання поточного технічного стану транспортного засобу.
Вихідні дані:
json

```
{
  "id": 1,
  "engineState": "good",
  "brakesState": "average",
  "oilLevel": "low",
  "tiresPressure": "normal",
  "alerts": [
    "Oil level is low. Please refill."
```

```
  ]
}
POST /monitoring/alerts
Опис: Створення запису про сповіщення для транспортного засобу.
Вхідні дані:
json
{
  "vehicleId": 1,
  "alertType": "critical",
  "message": "Brake system requires immediate inspection."
}
Вихідні дані:
json
{
  "message": "Alert added successfully."
}
```

3.4 Реалізація прогнозування вартості обслуговування транспорту та його LCC

Для реалізації прогнозування вартості обслуговування вартості та LCC було реалізовано 2 нейронні мережі: одна прогнозує подальші витрати на обслуговування. Друга на основі даних з першої моделі та поточних даних про виручку та доходи від використання вантажівки прогнозує оптимальний час для її заміни.

Для моделі що прогнозує вартість обслуговування було обрано метод градієнтного бустінгу на основі таких факторів: марка та модель вантажівки, пробіг (км), вік транспортного засобу (роки), тип та об’єм двигуна, стан шин (береться середній стан шин для вантажівки), кількість попередніх ремонтів, серйозність поломок, частка незапланованих ремонтів від загальної, кількість замінених запчастин, витрати на пальне за останній рік (л/100 км), середнє навантаження (т/місяць), кліматичні умови (температура, вологість, складність доріг), витрати на запасні частини за останній рік, вартість робіт у сервісних центрах, середній досвід водіїв (роки) за останній рік, стиль водіння (агресивний/помірний).

Для другої моделі використовується регресія на основі Random Forest алгоритму. Для цього використовуються такі фактори: витрати на обслуговування (з першої моделі), виручка від вантажівки за останній рік,

амортизація вантажівки, капітальні витрати на придбання нової вантажівки, зниження продуктивності через старіння, час простою, економія на пальному (л/100 км), економія на ремонтах (витрати на обслуговування – сервісні планові витрати), вартість продажу поточної вантажівки.

Отримані моделі формують дані що потім можуть бути збережені в документну базу даних в вигляді JSON документів. Пізніше, коли необхідно відобразити ці дані вони забираються і з них формують графіки та аналітика. Нижче наведено приклад такого JSON документа.


```
{
    "ownershipYears": 10,
    "currency": "USD",
    "costComponents": {
        "purchasePrice": 150000,
        "fuelCost": {
            "yearlyCosts": [
20000,21000,22000,23000,24000,25000,26000,27000,28000,29000
            ],
            "total": 245000
        },
        "maintenanceCost": {
            "yearlyCosts": [
3000,3200,3500,4000,4500,5000,5500,6000,6500,7000
            ],
            "total": 45500
        },
        "insuranceCost": {
            "yearlyCosts": [
5000,5100,5200,5300,5400,5500,5600,5700,5800,5900
            ],
            "total": 54600
        },
        "depreciation": {
            "yearlyValues": [
-20000,-18000,-16000,-15000,-14000,-13000,-12000,-11000,-10000,-
9000
            ],
            "total": -140000
        },
        "resaleValue": {
            "estimated": 10000
        }
    },
    "lifetimeCostSummary": {
        "totalCost": 405100,
        "netCostAfterResale": 395100
    },
    "yearlySummary": [
```

```
        {
            "year": 1,
            "totalYearlyCost": 43000,
            "cumulativeCost": 43000
        },
        {
            "year": 2,
            "totalYearlyCost": 44300,
            "cumulativeCost": 87300
        },
        {
            "year": 3,
            "totalYearlyCost": 45700,
            "cumulativeCost": 133000
        },
        {
            "year": 4,
            "totalYearlyCost": 47300,
            "cumulativeCost": 180300
        },
        {
            "year": 5,
            "totalYearlyCost": 48900,
            "cumulativeCost": 229200
        },
        {
            "year": 6,
            "totalYearlyCost": 50500,
            "cumulativeCost": 279700
        },
        {
            "year": 7,
            "totalYearlyCost": 52100,
            "cumulativeCost": 331800
        },
        {
            "year": 8,
            "totalYearlyCost": 53800,
            "cumulativeCost": 385600
        },
        {
            "year": 9,
            "totalYearlyCost": 55500,
            "cumulativeCost": 441100
        },
        {
            "year": 10,
            "totalYearlyCost": 57200,
            "cumulativeCost": 498300
        }
    ]
}
```

3.5 Реалізація користувацького інтерфейсу

З іншої сторони модуля стоїть користувацький інтерфейс. Вимоги до користувацького інтерфейсу включають в себе простоту, єдиний витриманий стиль, інформативність. Інтерфейс вимагає повної відповідності процесам що відбуваються у системі, щоб правильним чином описати їх. В процесах в межах модуля беруть участь 3 типи користувачі: водій, механік та менеджер. Для кожного з них необхідно розробити свій план та схему інтерфейсу. При цьому можна використовувати спільний стиль та оформлення що допоможе зменшити час на розробку. Перелічимо список основних екранів для кожного з користувачів.

Для менеджера обов’язковою є панель управління на якій відображається необхідні для роботи статистичні дані, список вантажівок разом з їх економічними показниками та поточним статусом. Також відображається список запитів на придбання запчастин та ремонт авто. Далі серед важливих екранів є окремі, більш деталізовані екрани економічних показників, включно з прогнозами вартості та оптимальним часом заміни транспортного засобу.

На екрані вантажівок відображаються всі вантажівки що прикріплені до цього менеджера, їх стан, остання діагностика, економічні показники, історію місцезнаходження, динаміку використання палива, данні з бортового комп’ютера та систем контролю відкривання дверей на причепі (за їх наявності).

На екрані запчастин можна побачити запити на отримання запчастин від механіків, а також ознайомитись з поточною кількістю тих чи інших запчастин на складі, також можна додати закупку та її статус.

В процесі подальшого використання цей та інші екрани можуть бути розширені додатковими показниками, відповідно до індивідуальних потреб менеджера.

В особистому кабінеті механіка (Рисунок 3.10) основними екранами є головна, список вантажівок, деталей та запитів на ремонт. Головна сторінка містить коротку інформацію про самого механіка, а також вибірку найціннішої інформації з кожного з інших екранів. Наприклад, найближчі планові обслуговування, поточна взята в роботу вантажівка, а також список непрочитаних запитів. На екрані вантажівок можна побачити поточний стан усіх вантажівок які прикріплені до депо механіка, їх найближче планове обслуговування, чи буде це заміна деталей чи діагностика.

Рисунок 3.10. Особистий кабінет механіка

При натисканні на конкретний рядок в списку відкривається детальна інформація про вантажіку – історію ремонтів та діагностик, рейси та список запланованих обслуговувань який можна розширити або змінити. Заплановані обслуговування можна прив'язати на певну дату у випадку якщо це, наприклад, зміна гуми на зимову, або на певний пробіг від поточного, наприклад, якщо мова йде про заміну мастила чи фільтрів, які змінюються раз

в 5000 кілометрів пробігу. Тут же можна додати нові види обслуговування і прив’язати їх до календаря чи пробігу.

На екрані запчастин є можливість отримати список наявних запчастин а також конкретний номер полиці на складі де вона знаходиться. Це дозволяє швидко отримати потрібну деталь і не витрачати зайвий час на її пошук. Проте така система вимагає дисципліни від механіків, які будуть класти деталі там де вони повинні бути. На цьому ж екрані можна ознайомитись з запитами на отримання деталей, їх статус та останні оновлення по складу.

Рисунок 3.11. Особистий кабінет водія

Для водія (Рисунок 3.11) на головній сторінці відображається інформація про самого водія, інформація про поточну його вантажівку, її статус. Також відображається список останніх рейсів водія. Хоч рейси не є частиною розробленого модуля, проте щоб створити для водія єдину точку входу і не змушувати перемикатись на окремий інтерфейс для рейсів, їх винесено сюди. Так само можна знайти і інформацію про наступний рейс: дату, контрольні точки, час в дорозі, загальну протяжність маршруту.

На екрані вантажівок водій може знайти детальну інформацію про поточну вантажівку. На відміну від схожих екранів у механіка та менеджера, у водія немає доступу до всіх вантажівок які є в компанії, лише до поточної. Там можна знайти інформацію про наступні обслуговування, історію ремонтів та діагностик, рівні витрат палива та інші метрики з бортового комп'ютера вантажівки (за наявності цих даних в системі).

На екрані запитів можна знайти історію запитів від водія до механіків чи менеджера щодо вантажівок на яких працює водій. Інформація включає в собі статус заявок, можливість переглянути її деталі та листування щодо неї.

3.6 Реалізація системи підтримки життєздатності мікросервіса

Для забезпечення життєздатності мікросервісів було проведено модернізацію їх кодової бази. Було додано систему телеметрії яка дозволяє чітко відслідковувати різноманітні метрики, які можна розділити на такі типи:

1. Лічильник – тип метрики створений для нескінченно збільшуваних величин, таких як кількість запитів, помилок або виконаних завдань
2. Вимірювач – це просте цілочислене значення яке може змінюватись як на збільшення, так і на зменшення. Може використовуватись для таких величин як об'єм оперативної пам'яті що зараз використовується, кількість одночасних запитів
3. Гістограми – накопичувальні метрики які можуть вимірювати час на виконання запитів, їх розміри, тощо. Відрізняється від лічильника тим що фіксує час або якусь іншу величину між двома викликами гістограми.

Ці 3 типи метрик дозволяють описувати роботоздатність та ефективність мікросервісів, так як дозволить оптимізувати витрати на використання інфраструктури та навантаження на певний мікросервіс. Крім того реалізована система моніторингу того, чи запущений наразі мікросервіс, що реалізується доволі просто: кожний мікросервіс тепер містить спеціальний ендпоінт який завжди повертає 200ОК, а у випадку якщо цього не стається, то його контейнер автоматично перезапускається.

Загалом збираються дані про такі метрики: кількість запитів, час відповіді, розподіл HTTP-кодів статусу, RPS, кількість виконаних SQL-запитів, середній час виконання SQL-запитів, відсоток невдалих запитів до БД, кількість відкритих підключень до БД, використання пам’яті, процесорне навантаження, кількість активних потоків, кількість зовнішніх HTTP-запитів, час виконання зовнішніх запитів, кількість помилок зовнішніх запитів, результати кешу (hit/miss), кількість повідомлень у черзі, час очікування повідомлень у черзі, кількість помилок обробки повідомлень.

Рисунок 3.12. Система моніторингу Grafana

Усі ці метрики та системи підтримки життєздатності передаються в систему моніторингу Grafana, яка дозволяє переглядати всі ці метрики під час роботи системи та виявляти проблеми які можуть бути виправлені або покращені. Це може бути зменшення часу відповіді на певних запитах, або виявлення довгої відповіді від бази даних на вибірках певного типу. Також якщо певний сервіс використовує малу частку з виділених йому ресурсів – виділені ресурси можна зменшити, щоб зменшити вартість утримання системи. Так само якщо сервіс витрачає багато ресурсів, або має помилки пов’язані з їх нехваткою, то можна відреагувати виділивши більше оперативної пам’яті або ядер процесора.

РОЗДІЛ 4. АНАЛІЗ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ РОЗРОБКИ

4.1 Оцінка зменшення витрат на ремонти і простої

Під час роботи логістичної компанії так чи інакше виникає необхідність обслуговувати транспортні засоби, це нормальний процес. Якщо ми кажемо про ремонти, то значно дешевше ремонтувати щось на початковому етапі його зношування, а не тоді коли проблема зачепила і сусідні модулі. Наприклад, коли зношується стійка амортизатора, то при несвоєчасній її заміні вона може потягнути за собою і інші деталі підвіски, зробивши ремонт багатократно дорожчим ніж початковий. Щоб таких ситуацій було менше необхідно проводити як і планові діагностики стану транспортного засобу, так і при виникненні нарікань на роботу перевіряти певні модулі. Несвоєрідний стукіт, потріскування, відмінна від звичайної робота може бути симптомом внутрішніх проблем.

Простої це ще одна причина непрямих втрат грошей підприємством. Комунікація між людьми може затягуватись в силу різних причин та банальних помилок. Через це можуть виникати ситуації коли вантажівка простоює надто довго і втрачаються гроші які могли би бути зароблені якби вона працювала. В простій відноситься також і час на ремонти. Якщо ремонт швидкий, то час який раніше витрачався на ремонт тепер може витрачатись на рейс, що збільшує дохідність.

Ці 2 проблеми можуть бути частково вирішені за рахунок такого функціоналу системи обслуговування автотранспорту:

- Автоматизоване планування ТО: модуль аналізує пробіг, час використання та інші параметри кожного транспортного засобу, автоматично формуючи графік обслуговування. Це допомагає уникнути надмірного зносу вузлів і механізмів, зменшуючи ймовірність несподіваних поломок.

- Попередження критичних несправностей: завчасне виявлення зношених деталей через діагностику або інтервальну заміну витратних матеріалів (наприклад, гальмівних колодок або ременів ГРМ) значно знижує ризик дорогих аварійних ремонтів.
- Інтеграція телематики: за допомогою датчиків модуль може відстежувати критичні показники в реальному часі, такі як температура двигуна, рівень масла чи стан шин. Це дає змогу оперативно реагувати на відхилення від норми.
- Рання діагностика несправностей: дані з телематичних пристроїв або системи OBD-II (діагностичний інтерфейс автомобіля) дозволяють виявити дрібні несправності до того, як вони стануть серйозними.
- Управління запасами запчастин: модуль відстежує потребу в запчастинах і замовляє їх завчасно. Це скорочує час очікування необхідних деталей, що може статися під час аварійного ремонту.
- Мінімізація "ефекту доміно": несвоєчасна заміна або ремонт однієї деталі може спричинити пошкодження інших систем. Наприклад, невчасна заміна ременя ГРМ може пошкодити клапани або навіть двигун.
- Зниження витрат на евакуацію: завдяки своєчасному ТО ризик раптових поломок на дорозі, що вимагають евакуатора, зменшується.
- Швидка діагностика: система допомагає проводити діагностику швидше завдяки історії обслуговування та показникам стану машини. Це дозволяє скоротити час простою транспорту в автосервісі.
- Альтернативне планування: завдяки модулю диспетчери можуть оперативно планувати заміну несправного автомобіля на справний для виконання доставки.

Раннє виявлення та профілактика можуть знижувати витрати на ремонт до 20%. За рахунок скорочення простоїв збільшується продуктивність автопарку. Зменшення накладних витрат дозволить компанії зберегти маржу

на кожну з поїздок навіть знизивши тарифи, що дозволяє перемагати в конкурентній боротьбі.

4.2 Оцінка оптимізації використання автопарку

Для оптимізації використання автопарку треба дотримуватись певних правил: обслуговування транспортних засобів має виконуватись в той час коли він не треба для перевезень.

Для ведення обліку ефективності використання транспорту необхідно моніторити стан та продуктивність транспорту. Чим менший проміжок між даними та їх обробкою, тим ефективнішим є менеджмент і контроль за транспортним засобом.

Маючи на руках інформацію про стан тих чи інших транспортних засобів можна керувати призначеннями їх на ті чи інші маршрути. Таким чином можна автотранспорт з найменшим ризиком поломки призначати найдовші або найскладніші маршрути мінімізуючи таким чином ризики.

Підсумовуючи все, можна виділити найважливіший функціонал нового модуля який допоможе в оптимізації використання автопарку:

- Прогнозування потреб у технічному обслуговуванні (ТО): модуль аналізує дані про пробіг, час роботи, навантаження на транспортні засоби та історію ТО, щоб визначити оптимальні періоди для технічного обслуговування. Це дозволяє уникнути ситуацій, коли транспорт стає несправним у розпал роботи.
- Синхронізація з графіком перевезень: ТО планується у ті періоди, коли транспортний засіб не потрібен для виконання замовлень. Наприклад, планування обслуговування на час із найменшим завантаженням компанії.
- Збір даних у реальному часі: модуль інтегрується з телематичними системами та датчиками на транспорті, які надають інформацію про стан двигуна, витрати пального, знос шин, рівень масла тощо. Це дозволяє приймати швидкі рішення щодо технічного стану транспорту.

- Розподіл навантаження: аналізуючи стан кожного транспортного засобу, система може запропонувати оптимальний розподіл навантаження між машинами, щоб уникнути перевантаження окремих одиниць і зменшити ризик поломок.
- Інтеграція з маршрутними модулями: модуль може працювати разом із системами управління маршрутами, враховуючи стан транспорту. Так, при виборі транспорту що буде призначено на маршрут логіст отримає дані і про його технічний стан. Таким чином забезпечується підтримка рішення логіста за рахунок надання йому інформації про транспортний засіб. Пізніше це можна виділити в окрему рекомендаційну систему яка б підбирала найбільш підходящий транспорт автоматично.
- Прозорий облік ресурсів: модуль створює централізовану базу даних, яка відображає стан, місцезнаходження та продуктивність кожного транспортного засобу. Це дозволяє диспетчерам швидше приймати обґрунтовані рішення.
- Зменшення перевитрат пального: регулярне обслуговування та моніторинг технічного стану дозволяють підтримувати транспорт у оптимальному робочому стані, зменшуючи споживання пального.
- Виявлення слабких місць: звіти про технічний стан і використання кожного транспортного засобу дозволяють виявляти транспорт із високими витратами на ремонт або низькою ефективністю. Це допомагає ухвалювати рішення про заміну чи модернізацію.

Зменшення простоїв за рахунок кращої організації ТО та діагностики дозволяє ще сильніше збільшити прибутки які отримує той самий транспорт. Також модуль дозволить раціоналізувати використання коштів на пальне, запчастини та зарплати працівників.

4.3 Оцінка впливу мінімізації ризиків штрафів та компенсацій

Коли ми кажемо про мінімізацію ризиків штрафів та компенсацій мається на увазі простота виконання регуляторних вимог, зниження ризиків

аварій, тощо. Крім того, будь яка несправність в дорозі створює затримку в доставці вантажа клієнта. Звичайно, час на форс-мажори закладається в час доставки, але все це має свої ліміти і не подобається замовнику. Крім того, коли водій приїжджає зарано через закладання запасного часу він простоює в очікуванні розвантаження або завантаження. Крім того враховуючи специфіку вантажного автотранспорту на нього можуть накладатись додаткові сертифікації та перевірки які проходить лише справний транспорт.

Узагальнивши що треба для мінімізації ризиків штрафів та компенсацій можна виділити такий список функціонал який допоможе в цьому:

- Автоматизація контролю за регламентами: модуль зберігає інформацію про обов'язкові технічні огляди, екологічні сертифікати (наприклад, рівень викидів CO_2), страховки, дозволи на перевезення тощо. Система автоматично нагадує про терміни виконання цих процедур. Як результат, зменшується ризик пропуску важливих дат, що може призвести до штрафів або заборони використання транспорту.
- Сертифікація відповідності: регулярне обслуговування гарантує, що транспортні засоби відповідають нормам безпеки, екологічним стандартам та технічним вимогам, установленим законодавством. Якщо вантажівка вчасно проходить перевірку рівня викидів, це запобігає штрафам у зонах екологічного контролю.
- Своєчасне технічне обслуговування: модуль забезпечує регулярну діагностику та профілактичний ремонт. Наприклад, завчасна заміна гальмівних колодок чи діагностика рульової системи знижує ймовірність аварій через технічні несправності. Відповідне зниження кількості аварій може заощадити кошти на компенсації збитків іншим учасникам аварії.
- Відстеження порушень водіїв: за допомогою телематичних систем модуль може контролювати швидкість руху, різкі гальмування або перевищення тривалості робочого часу водія. Це дозволяє уникнути штрафів за порушення правил дорожнього руху або норм робочого часу.

- Ефективність використання пального: справний транспорт споживає менше пального, що зменшує викиди CO_2 і ризик штрафів за перевищення екологічних норм.
- Уникнення затримок доставки: завдяки плановому ТО і прогнозуванню потреб в ремонті транспортні засоби знаходяться в належному стані, що дозволяє дотримуватися графіків доставки. Затримки часто призводять до штрафних санкцій з боку клієнтів або втрати контрактів.
- Звіти та документація: модуль автоматично генерує звіти про виконання ТО, стан транспорту та дотримання нормативів. У разі перевірок або спорів із клієнтами це забезпечує наявність доказової бази для захисту інтересів компанії. Як приклад: Якщо клієнт скаржиться на пошкодження вантажу, дані про технічний стан і маршрут можуть підтвердити, що транспорт функціонував належним чином.

Як наслідок впровадження основними покращеннями буде: зменшення кількості штрафів за недотримання норм, правил, графіків технічного обслуговування. Скорочення розмірів компенсацій за рахунок уникнення аварій затримок чи пошкодження вантажу. Репутація компанії буде кращою якщо не буде помічено порушення нормативів чи аварій.

4.4 Оцінка впливу підвищення прозорості витрат

Під час неефективного управління автопарком можуть спостерігатись випадки коли деякі витрати не були враховані або виникли більшими ніж очікувалось. Крім того, відсутність систематизації даних про витрати позбавляє фінансистів можливості вести ефективну аналітику витрат. Прозорість витрат також дозволить виявляти викиди у витратах пов’язані з певним водієм, механіком чи транспортним засобом. Окрім того, будь хто з тих хто має доступ до вантажівки може займатись крадіжками чи шахрайством, що теж можна виявляти маючи детальну статистику.

За допомогою модуля можна оптимізувати витрати використовуючи такий функціонал та особливості програми:

- Централізоване збирання даних: модуль автоматично фіксує витрати, пов’язані з ремонтом, технічним обслуговуванням, закупівлею запчастин, паливом, страховкою та податками. Дані збираються в одній системі, що дозволяє уникнути плутанини чи втрати інформації.
- Класифікація витрат: система розподіляє витрати за категоріями (ремонт, ТО, паливо, страхування тощо) та прив’язує їх до конкретного транспортного засобу, водія чи механіка.
- Контроль витрат на паливо: інтеграція із системами GPS і телематики дозволяє в режимі реального часу відстежувати витрати на пальне залежно від маршрутів, швидкості, часу простоїв і манери водіння.
- Оперативне реагування на зростання витрат: якщо модуль фіксує аномальне підвищення витрат, наприклад, через перевитрату пального чи незапланований ремонт, система повідомляє диспетчера чи керівника.
- Управління закупівлями запчастин: система аналізує історію поломок і ТО, прогнозує потребу в запчастинах, дозволяючи купувати їх оптом або заздалегідь, що знижує витрати та простої.
- Прогноз витрат: модуль використовує історичні дані для прогнозування майбутніх витрат на ТО, ремонти чи паливо. Це допомагає планувати бюджет автопарку. Самі по собі витрати – це буденність будь-якого бізнесу. Страшніше коли бізнес стикається з незапланованими витратами.
- Аналіз ефективності інвестицій: дані про витрати порівнюються з доходами, які генерує кожен транспортний засіб, дозволяючи оцінити рентабельність його експлуатації.
- Зв’язок із бухгалтерією: витрати на транспорт інтегруються в загальну фінансову систему компанії. Це спрощує складання фінансових звітів, облік амортизації та аналіз загальних витрат.

- Контроль перевищення бюджетів: модуль відстежує, щоб витрати на транспорт не перевищували затверджений бюджет. У разі перевищення система надсилає попередження.
- Аналіз відхилень: модуль може виявляти відхилення у витратах, які не відповідають звичайним тенденціям. Наприклад, різке зростання витрат на пальне може вказувати на крадіжку або неправильне списання.
- Прозорість звітності: автоматизована система виключає ручні помилки або маніпуляції, які можуть виникати при введенні даних вручну.
- Візуалізація даних: модуль надає інтерактивні графіки, діаграми та звіти, які показують динаміку витрат у розрізі часу, категорій та транспортних засобів. Менеджер може краще сприйняти певні дані якщо вони подані правильним чином, за рахунок вибору правильного набору графіків, тощо.

Як результат за рахунок впровадження системи план-максимум виглядає так: Кожна витрата зрозуміла, обґрунтована й прив’язана до конкретного транспортного засобу чи водія. Відповідно, стає набагато простіше оптимізувати бюджет за рахунок прийняття вчасних обґрунтованих даними рішень які призведуть до скорочення витрат і підвищення рентабельності. Прозорість витрат може стати вагомою причиною для інвесторів вкласти гроші в компанію.

4.5 Оцінка впливу від покращення конкурентоспроможності

Для того щоб компанія могла існувати необхідно не лише скорочувати витрати, а і збільшувати доходи. Зробити це можна за рахунок покращення процесів у власній компанії що допоможе перемогти у конкурентній боротьбі. Найважливішими фактором у конкурентній боротьбі в такій сфері є ціна, надійність та репутація компанії. Ось як впровадження системи управління обслуговуванням автопарком може вплинути на ці 3 фактори:

- Ефективне управління витратами: завдяки автоматизації обліку витрат на технічне обслуговування, паливо, ремонт і амортизацію компанія

значно знижує накладні витрати. Це дозволяє запропонувати клієнтам вигідніші тарифи, зберігаючи прибутковість.

- Уникнення простоїв і зривів доставки: прогнозування поломок і своєчасне планування технічного обслуговування мінімізують ризики раптових відмов техніки. Це дозволяє дотримуватися термінів доставки, що є критично важливим для клієнтів і відповідно компанія яка не зриває строки має перевагу у очах клієнтів.
- Швидка реакція на запити клієнтів: завдяки прозорості даних компанія може оперативно надавати клієнтам інформацію про статус доставки, маршрут і технічний стан транспортного засобу.
- Мінімізація скарг і претензій: надійний автопарк і контроль стану транспорту зменшують ризик втрат чи пошкодження вантажу, що підвищує довіру клієнтів.
- Репутація сучасної компанії: використання автоматизованих систем демонструє клієнтам і партнерам готовність компанії до інновацій, що створює позитивний імідж на відміну від компаній що використовують застаріле програмне забезпечення
- Підтримка сталого розвитку: екологічність автопарку, оптимізація маршрутів і зменшення викидів CO_2 завдяки підтримці техніки в належному стані відповідають сучасним вимогам до корпоративної відповідальності. Це стає особливо актуально в часи коли репутація компанії може дуже легко монетизуватись за рахунок поширення через сарафанне радіо, що є найефективнішим методом реклами.
- Дані для стратегічного управління: система надає керівництву доступ до детальної аналітики витрат, ефективності автопарку, показників надійності та рентабельності. Це дозволяє швидко реагувати на ринкові зміни й оптимізувати бізнес-процеси.
- Гнучкість у плануванні: завдяки прогнозуванню потреб в обслуговуванні компанія може краще адаптуватися до змін попиту чи умов роботи.

- Рекомендації та репутація: задоволені клієнти та партнери стають джерелом рекомендацій, що підвищує привабливість компанії на ринку.
- Залучення більше кваліфікованих та надійних працівників: за рахунок хорошої репутації компанії вона може залучати найкращих робітників не збільшуючи їх зарплатню. Це досягається прозорими вимогами до роботи, умовами взаємодії та відсутністю бюрократії на робочому місці

У результаті система впливає на виграш конкурентної боротьби таким чином: компанія може підтримувати конкурентоспроможні ціни і при цьому залишатись прибутковою. Компанія сприймається як надійний партнер маючи нульову або мінімальну кількість зриву строків доставок, пошкодження вантажів, інших інцидентів. Клієнти що мають позитивний досвід на основі попередніх двох аргументів з більшою вірогідністю будуть радити того ж перевізника своїм партнерам. Велика кількість можливостей для аналітики дозволяє швидше за конкурентів реагувати на зміни на ринку. Наприклад, за даними компанія певна модель вантажівок має кращі показники по сукупним витратам ніж інші, відповідно чим раніше наша компанія це помітить, тим швидше може бути модернізований автопарк. Зменшення цін на обслуговування автопарку може дозволити компанії навіть робити демпінг цін заради підриву позицій конкурентів, таким чином захоплюючи ринки.

У підсумку впровадження такої системи може дати стійку перевагу на ринку, допомагаючи залучати нових клієнтів і зберігати довгострокові відносини з існуючими партнерами.

4.6 Оцінка вартості виконання робіт по розробці та розгортанню системи

Після того як було обґрунтовано необхідність інтеграції нового модуля в програмне забезпечення, настала пора поговорити про вартість такого

маневру для бізнесу. Так як модуль закладається як розширення вже існуючої системи яка розроблялась не так давно, тому буде вважатись що

компанія замовляє розширення функціоналу у тієї ж IT компанії що і раніше займалась розробкою програмного забезпечення логістичної компанії

Таблиця 3.1. Погодинна вартість команди що бере участь в розробці.

Посада	Рівень	Вартість від $/год	Вартість до $/год
Product owner / project manager	Middle/Senior	20	25
Business analyst	Middle	11	13
Business analyst	Junior	4	6
Backend developer	Senior	25	40
Backend developer	String Junior	5,5	7
Backend developer	Trainee/Junior	3,5	3,5
Frontend developer	Middle	11	13
Frontend developer	Trainee/Junior	3,5	3,5
Android developer	Middle	11	13
IOS developer	Middle	11	13
ML Engineer	Middle	12	14
QA Automotive	Middle	9	11
QA Automotive	Trainee/Junior	3,5	3,5
QA Manual	Middle	5,5	5,5
QA Manual	Trainee/Junior	3,5	3,5
QA Manual	Trainee/Junior	3,5	3,5

Усі розрахунки вартості будуть вираховуватись на основі даних станом на 25.11.2024. Для розробки подібного продукту необхідно залучити доволі велику команду: Product owner/project manager, 2 business analyst, 3 backend і 2 frontend розробників. Якщо мова йде і про розробку мобільного застосунку, то необхідно залучити ще 4 мобільних розробників: 2 android чи IOS розробники, в залежності від того яку техніку буде видано співробітникам. Якщо ж ніякої техніки співробітникам видаватись не буде, то треба наймати команду для

розробки застосунку для обох платформ, і від цього варіанту я і буду відштовхуватись. Для забезпечення роботи моделей штучного інтелекту також треба найняти одного спеціаліста який буде цим займатись. Для забезпечення необхідних рівнів якості програмного забезпечення також необхідно залучити команду QA інженерів яка складається з 5 людей: 2 на автоматизоване тестування і 3 на ручне. Більш детальні дані про склад команди та її вартість наведені в таблиці 3.1:

Сумарно протягом для аналізу вимог необхідно 2392 години які будуть розподілені наступним чином: 2016 годин на двох бізнес аналітиків та 376 годин для Product owner. Таким чином маємо 6 місяців роботи для бізнес-аналітиків.

На розробку бекенд частини необхідно витратити 3528 годин розподілених між 3 розробниками: кожному по 1176 годин, або 7 місяців роботи. Додатково виділимо ще 120 годин часу Product owner на консультування та роз’яснення вимог розробникам.

На розробку користувацького веб застосунку необхідно 3528 годин. Цей час розподіляється між двома фронтенд розробниками. Додатково виділимо ще 120 годин часу Product owner на консультування та роз’яснення вимог розробникам.

На розробку користувацького android застосунку необхідно 1176 годин. Додатково виділимо ще 120 годин часу Product owner на консультування та роз’яснення вимог розробникам.

На розробку користувацького IOS застосунку необхідно 1176 годин. Додатково виділимо ще 120 годин часу Product owner на консультування та роз’яснення вимог розробникам.

На розробку моделей та їх тренування ML розробнику необхідно буде 1084 години та близько 20 годин Product owner на роз’яснення задач.

На тестування необхідно виділити 5880 годин. Вони рівномірно розподіляються між всіма тестувальниками. Крім того, 180 годин витратить на це Product owner.

На інфраструктурні витрати: 2 полігони: для розробників та для тестування необхідно буде витрачати по 100$ на кожний в місяць сумарно.

Загальний час на виконання проекту складає 7 місяців з яких місяць виділяється на бета версію під час якої продукт вже запущено в реліз, але регулярно випускаються правки. Кінцева вартість робіт вказано в таблиці нижче:

Таблиця 3.2. Загальна вартість етапів розробки застосунку

Найменування	Вартість мінімальна, $	Вартість максимальна, $
Аналіз вимог та формування ТЗ	22 640	28 552
Розробка серверної частини	42 384	62 388
Розробка клієнтського застосунку (веб)	18 552	21 804
Розробка клієнтського застосунку (android)	15 336	18 288
Розробка клієнтського застосунку (IOS)	15 336	18 288
Розробка моделей та штучного інтелекту	12 496	14 612
Тестування	33 000	36 252
Ліцензії ПО для розробників	400	400
Полігони для тестування та розробки	700	700
Всього	160 844	201 284

В таблиці не враховані витрати на обладнання робочих місць, пошук співробітників, тощо які лежать на компанії яка розробляє продукт. Власноруч логістична компанія навряд буде займатись подібними розробками через специфіку розробки додатків такого розміру.

Кінцева ціна для замовника буде більшою враховуючи амортизацію, дохід компанії-виконавця та інше, тому можна сміливо додати, припустимо, +50% від зарплат, тому маємо проміжок між 241 266 та 301 928 USD.

4.7 Загальний підрахунок економічної ефективності

Для підрахунку загальної економічної ефективності будемо використовувати приблизні дані, адже у нас немає доступу до конкретних показників компанії. Крім того, варто враховувати час на навчання користування програмою персоналом що може зайняти до року, і, відповідно, протягом цього часу ефективність від впровадження програми буде поступово зростати. Перші об’єктивні результати від використання програми будуть помітні через 9-15 місяців після впровадження. Максимальна ефективність буде досягнута на 18-24 місяці, тобто через 2 роки. Для повного впровадження програми на всіх рівнях і інтеграції в бізнес-процеси необхідно приблизно 3-4 роки.

Ефект від скорочення операційних витрат може бути такий:

- Витрати на ремонт та обслуговування зменшиться на близько 10-25% за рахунок попереджувального обслуговування та планових ремонтів.
- Витрати на паливо можуть зменшитись на показники близько 5-15% за рахунок моніторингу та кращого технічного стану вантажівок.
- На простоях витрати можуть зменшитись приблизно на 10-20%.

Відповідно лише на скороченні витрат на цих категоріях, відповідно до часток цих категорій в загальній розбивці витрат, прибуток виросте на 10-20% за рахунок скорочення операційних витрат.

Вплив на покращення клієнтоорієнтованості та відповідного збільшення кількості нових клієнтів та можливості просити більше за рахунок надійності та прогнозованості перевізника оцінюється в 5-15% за рахунок збільшення обсягів замовлень.

В перший рік після впровадження вплив буде мінімальним, від навіть незначної втрати ефективності у 3% до збільшення ефективності на 3%, що не є надто значними цифрами. Після того почнеться період росту ефективності використання самої програми, і, відповідно її впливу на процеси компанії та прибутки. Довгостроковий ефект оцінюється в 20-40% прибутку що буде поступово нарощуватись впродовж перших 3 років. Цей показник залежить від

масштабу автопарку, поточного рівня конкурентоспроможності та ефективності впровадження системи.

У висновку що стосується росту прибутків маємо: приріст у середньостроковій перспективі (1-2 роки) може досягати 15-25%, в довгостроковій (3+ років) цей показник виросте до 30-40%.

Враховуючи попередні витрати в районі 241 266 - 301 928 USD, можемо розрахувати який поточний дохід має бути у компанії для забезпечення окупності в найближчі 10 років. Сума приросту прибутку за 10 років має дорівнювати вартості впровадження:

$$5\% \cdot P + 15\% \cdot P + 7 \cdot 30\% \cdot P = \text{вартість впровадження},$$

де P — початковий прибуток компанії.

Розрахуємо сумарний приріст прибутку за 5 років:

- 1-й рік: 0.05P
- 2-й рік: 0.15P
- 3-8й роки: 7*0.30P=2.10P
- Сумарний приріст: 0.05P+0.15P+2.10P=2.60P

Приріст дорівнює вартості впровадження, тому:

2.60P=вартість впровадження

P=вартість впровадження/2.6:

- Для мінімальної вартості 241 266: 92 794
- Для максимальної вартості 301 928: 116 126

Висновок:

Поточний прибуток компанії повинен бути в межах **92 794 – 116 126** USD для окупності впровадження нового програмного забезпечення за 10 років.

Так як мова йде про повну окупність лише з приросту, то базовий прибуток у компанії звісно ж лишається, але на кінець 3го року він становитиме 120 632 - 150 964 USD.

ВИСНОВКИ

У кваліфікаційній роботі було проведено дослідження потреб логістичних компаній в управлінні автопарком.

У дослідницькій частині було проведено аналіз програм-конкурентів, їх підходів до вирішення подібних задач. Також досліджено можливі вимоги до майбутнього програмного забезпечення, виявлено його схожості з системами підтримки прийняття рішень.

Розроблено концепцію модуля обслуговування автопарку, його основний функціонал та задачі які цей функціонал вирішує. Серед них окремо виділю відстеження графіка технічного обслуговування транспортних засобів, автоматизація нагадувань про необхідність обслуговування, формування звітів про витрати на технічне обслуговування, інтеграція з існуючими інформаційними системами логістичної компанії.

Розроблено програмне забезпечення модуля на основі використання поточної реалізації логістичної компанії. Реалізація дозволяє використовувати ті самі механізми що вже використовувались раніше.

За результатами тестування в лабораторних умовах модуль дозволяє вирішувати старі рутинні задачі набагато швидше, оптимізувати витрати та скоротити час простою автотранспорту.

Розроблений модуль обслуговування автопарку відповідає вимогам сучасних логістичних компаній та демонструє високу ефективність. Використання такого програмного забезпечення сприяє підвищенню конкурентоспроможності компанії завдяки зменшенню операційних витрат та покращенню управління ресурсами.

Подальші дослідження можуть бути спрямовані на розширення функціоналу модуля, зокрема на інтеграцію систем моніторингу водіїв, аналіз екологічності транспортних засобів та прогнозування витрат на основі великих даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A complete guide to fleet management software development: process, features & costs (2024) [Електронний ресурс] – Режим доступу до ресурсу: https://volpis.com/blog/guide-to-fleet-management-software-development/.
2. Bedard S. What is fleet management software? [Електронний ресурс] / Sofie Bedard. – 2018. – Режим доступу до ресурсу: https://cetaris.com/fleet-maintenance-blog/what-is-fleet-management-software.
3. What is Fleet Management Software? [Електронний ресурс] – Режим доступу до ресурсу: https://www.teletracnavman.com/fleet-management-software/resources/what-is-fleet-management-software.
4. Fuel Management System Development: Features, Costs and Trends [Електронний ресурс] – Режим доступу до ресурсу: https://amconsoft.com/fuel-management-system-development-on-the-road-to-fuel-economy/
5. Torre C. .NET Microservices: Architecture for containerized .NET Applications / C. Torre, B. Wagner, M. Rousos. – Redmond, Washington 98052-6399: Microsoft Corporation, 2022. – 333 с. – (Microsoft Developer Division, .NET and Visual Studio product teams).

ДОДАТОК А – КОД ЗАСТОСУНКУ

GatewayApi/AuthenticationService

```
using System.Security.Claims;
using Microsoft.AspNetCore.Identity;
using SolarLogistics.Contracts.Constants;
using SolarLogistics.GatewayAPI.Services.Authentication.Models;
using
SolarLogistics.GatewayAPI.Services.Authentication.Models.Enum;
using SolarLogistics.GatewayAPI.Services.Token;

namespace SolarLogistics.GatewayAPI.Services.Authentication;

//TODO: Reset password logic, Email confirmation?
public class AuthenticationService : IAuthenticationService
{
private ILogger<AuthenticationService> _logger;
private ITokenService _tokenService;
private UserManager<IdentityUser> _userManager;
private SignInManager<IdentityUser> _signInManager;

public AuthenticationService(
UserManager<IdentityUser> userManager,
SignInManager<IdentityUser> signInManager,
ILogger<AuthenticationService> logger,
ITokenService tokenService)
{
_userManager = userManager;
_signInManager = signInManager;
_logger = logger;
_tokenService = tokenService;
}

public async Task<RegistrationResult>
RegisterAsync(RegistrationModel model)
{
var user = await _userManager.FindByEmailAsync(model.Email);
if (user != null)
{
return new RegistrationResult()
{
User = user,
Result = RegistrationResultEnum.UserAlreadyExist
};
}

user = new IdentityUser()
{
Email = model.Email,
UserName = model.Email,
SecurityStamp = Guid.NewGuid().ToString()
```

```
};

var result = await _userManager.CreateAsync(user,
model.Password);
if (!result.Succeeded)
{
LogIdentityErrors(result.Errors);

return new RegistrationResult()
{
User = null,
Result = RegistrationResultEnum.InternalError
};
}

result = await _userManager.AddToRolesAsync(user, model.Roles);
if (!result.Succeeded)
{
LogIdentityErrors(result.Errors);

return new RegistrationResult()
{
User = null,
Result = RegistrationResultEnum.InternalError
};
}

var claims = new []
{
new Claim(ClaimTypes.Email, model.Email),
new Claim(ClaimTypes.Name, model.FirstName),
new Claim(ClaimTypes.Surname, model.LastName),
new Claim(ClaimTypes.Sid, user.Id),
new Claim(ClaimTypes.MobilePhone, model.MobilePhone ??
string.Empty),
new Claim(ClaimTypes.DateOfBirth, model.DateOfBirth.ToString()
?? string.Empty),
new Claim(AdditionalClaims.Telegram, model.Telegram ??
string.Empty),
new Claim(AdditionalClaims.Skype, model.Skype ?? string.Empty),
new Claim(AdditionalClaims.Badge, model.Badge ?? string.Empty),
new Claim(AdditionalClaims.Category, model.Category ??
string.Empty)
}.Concat(model.Roles.Select(role =>
new Claim(ClaimTypes.Role, role))
).ToList();

await _userManager.AddClaimsAsync(user, claims);

return new RegistrationResult
{
User = user,
Result = RegistrationResultEnum.Success
```

```
};
}


public async Task<LoginResult> LoginAsync(LoginModel model)
{
var user = await _userManager.FindByEmailAsync(model.Email);
if (user is null)
{
return new LoginResult()
{
Result = LoginResultEnum.WrongEmail
};
}

var sighInResult = await _signInManager.PasswordSignInAsync(
user: user,
password: model.Password,
isPersistent: false,
lockoutOnFailure: false);

if (!sighInResult.Succeeded)
{
return new LoginResult()
{
Result = LoginResultEnum.WrongPassword
};
}

var roles = await _userManager.GetRolesAsync(user);
return new LoginResult()
{
Result = LoginResultEnum.Success,
User = user,
Roles = roles
};
}

private void LogIdentityErrors(IEnumerable<IdentityError>
errors)
{
foreach (var error in errors)
{
_logger.LogError(error.Description);
}
}
}
```

GatewayApi/Program.cs

```
using System.Security.Claims;
using System.Text;
using Microsoft.AspNetCore.Authentication.JwtBearer;
```

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi.Models;
using SolarLogistics.GatewayAPI.Database;
using SolarLogistics.GatewayAPI.Services.Authentication;
using SolarLogistics.GatewayAPI.Services.Token;

var builder = WebApplication.CreateBuilder(args);
var configuration = builder.Configuration;
// Add services to the container.

builder.Services.AddDbContext<ApplicationDatabase>(
    options =>
options.UseSqlServer(configuration.GetConnectionString("Azure"))
);

builder.Services.AddIdentity<IdentityUser, IdentityRole>(options
=>
    {
        options.ClaimsIdentity.UserIdClaimType = ClaimTypes.Sid;
        options.ClaimsIdentity.EmailClaimType =
ClaimTypes.Email;
        options.ClaimsIdentity.RoleClaimType = ClaimTypes.Role;
        options.User.RequireUniqueEmail = true;
        options.Password.RequireDigit = true;
        options.Password.RequireLowercase = true;
        options.Password.RequireNonAlphanumeric = false;
        options.Password.RequireUppercase = true;
        options.Password.RequiredLength = 8;
    })
    .AddEntityFrameworkStores<ApplicationDatabase>();

builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme =
JwtBearerDefaults.AuthenticationScheme;
    options.DefaultScheme =
JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(o =>
{
    o.TokenValidationParameters = new TokenValidationParameters
    {
        ValidIssuer = configuration["Jwt:Issuer"] ?? throw new
NullReferenceException("Issuer not found"),
        ValidAudience = configuration["Jwt:Audience"] ?? throw
new NullReferenceException("Audience not found"),
        IssuerSigningKey = new SymmetricSecurityKey
            (Encoding.UTF8.GetBytes(configuration["Jwt:Key"] ??
throw new NullReferenceException("Key not found"))),
        ValidateIssuer = true,
```

```
        ValidateAudience = true,
        ValidateLifetime = false,
        ValidateIssuerSigningKey = true
    };
});

builder.Services.AddAuthorization();

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddTransient<ITokenService, TokenService>();
builder.Services.AddScoped<IAuthenticationService,
AuthenticationService>();
builder.Services.AddTransient<HttpClient>();

builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo()
    {
        Version = "v1",
        Contact = new OpenApiContact()
        {
            Name = "Bohdan Zaika"
        }
    });
    c.ResolveConflictingActions(apiDescriptor =>
apiDescriptor.First());
    c.AddSecurityDefinition("Bearer", new
OpenApiSecurityScheme()
    {
        Name = "Authorization",
        Type = SecuritySchemeType.ApiKey,
        Scheme = "Bearer",
        BearerFormat = "JWT",
        In = ParameterLocation.Header,
        Description = "JWT Authorization header using the Bearer
scheme."
    });
    c.AddSecurityRequirement(new OpenApiSecurityRequirement()
    {
        {
            new OpenApiSecurityScheme()
            {
                Reference = new OpenApiReference()
                {
                    Type = ReferenceType.SecurityScheme,
                    Id = "Bearer"
                }
```

```
            },
            new List<string>()
        }
    });
});

var app = builder.Build();

app.UseSwagger();
app.UseSwaggerUI();

app.UseHttpsRedirection();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();
Seed.Initialize(app.Services);
app.Run();
```

DeliveriesService/MongoDbDeliveriesService

```
using MongoDB.Bson;
using MongoDB.Driver;
using SolarLogistics.DeliveriesService.Domain;
using SolarLogistics.DeliveriesService.Domain.Client;

namespace SolarLogistics.DeliveriesService.Services;

class MongoDbDeliveriesService : IDeliveriesService
{
    private readonly IDbClient _dbClient;

    public MongoDbDeliveriesService(IDbClient dbClient)
    {
        _dbClient = dbClient;
    }

    public IMongoCollection<Delivery> Collection =>
_dbClient.DeliveriesCollection;

    public async Task<List<Delivery>> GetAllAsync()
    {
        return (await Collection.FindAsync(_ => true)).ToList();
    }

    public async Task<List<Delivery>> GetAsync(Func<Delivery,
bool> predicate)
    {
        return (await Collection.FindAsync(x =>
predicate(x))).ToList();
    }
```

```
    public async Task<Delivery> AddAsync(Delivery delivery)
    {
        Validate(delivery);

        delivery.Status = DeliveryStatus.Added;

        await Collection.InsertOneAsync(delivery);
        return delivery;
    }

    public async Task<Delivery> UpdateAsync(Delivery delivery)
    {
        var filter = Builders<Delivery>.Filter
            .Eq(x => x.Id, delivery.Id);
        Validate(delivery);

        var replacingResult = await
Collection.ReplaceOneAsync(filter, delivery);
        if (!replacingResult.IsAcknowledged)
        {
            throw new ArgumentException($"Delivery with Id =
{delivery.Id} not exist");
        }

        return delivery;
    }

    public async Task<bool> DeleteAsync(string deliveryId)
    {
        var result = await Collection.DeleteOneAsync(x => x.Id
== deliveryId);
        return result.IsAcknowledged;
    }

    private static void Validate(Delivery delivery)
    {
        if (delivery.ArrivalPeriod == null &&
delivery.LoadingPeriod == null)
        {
            throw new ArgumentException("Delivery must have at
least one of arriving of loading periods");
        }

        if (delivery.ArrivalPeriod != null)
        {
            if (delivery.ArrivalPeriod.Start == null &&
delivery.ArrivalPeriod.End == null)
            {
                throw new ArgumentException(
                    $"{nameof(delivery.ArrivalPeriod)} must have
least at one of start or end times",
                    nameof(delivery.ArrivalPeriod));
            }
```

```
        }
        if (delivery.LoadingPeriod != null)
        {
            if (delivery.LoadingPeriod.Start == null &&
delivery.LoadingPeriod.End == null)
            {
                throw new ArgumentException(
                    $"{nameof(delivery.LoadingPeriod)} must have
at least one of start or end times",
                    nameof(delivery.LoadingPeriod));
            }
        }

        if (delivery.ClientId == null)
        {
            throw new
NullReferenceException($"{nameof(delivery.ClientId)} is null");
        }

        if (delivery.CargoTypes.Length == 0)
        {
            throw new ArgumentException($"Delivery must have at
least one cargo type", nameof(delivery.CargoTypes));
        }
    }
}
```

Contracts/DeliveriesService/Http/Responses/DeliveryResponse

```
using SolarLogistics.Contracts.DeliveriesService.Enum;
using
SolarLogistics.Contracts.GeoInformationService.Http.Responses;
using SolarLogistics.Contracts.Utility;

namespace
SolarLogistics.Contracts.DeliveriesService.Http.Responses;

public record DeliveryResponse
{
    public string Id { get; set; }
    public DirectionResponse Direction { get; set; }
    public string ClientId { get; set; }
    public DeliveryType Type { get; set; }
    public string Description { get; set; }
    public string? PaymentId { get; set; }
    public TimePeriod LoadingPeriod { get; set; }
    public TimePeriod ArrivalPeriod { get; set; }
    public DeliveryStatus Status { get; set; }
    public double? CargoCost { get; set; }
    public double? CargoWeight { get; set; }
    public double? CargoLenght { get; set; }
    public double? CargoWidth { get; set; }
    public double? CargoHeight { get; set; }
```

```
    public string CargoType { get; set; }
    public string TrailerType { get; set; }
    public string[] CargoCategories { get; set; }
}
```

GeoinfoService/MatrixService

```
using System.Net;
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;
using
SolarLogistics.Contracts.GeoInformationService.Http.Requests;
using
SolarLogistics.Contracts.GeoInformationService.Http.Responses;
using SolarLogistics.Errors;
using SolarLogistics.GeoInformationService.AddressService;
using SolarLogistics.GeoInformationService.Domain;
using
SolarLogistics.GeoInformationService.MatrixService.Models.Reques
t;
using
SolarLogistics.GeoInformationService.MatrixService.Models.Respon
se;

namespace SolarLogistics.GeoInformationService.MatrixService;

public class MatrixService : IMatrixService
{
    private readonly IAddressService _addressService;
    private readonly HttpClient _client;
    private readonly ILogger<MatrixService> _logger;
    private string _googleApiKey;

    public MatrixService(IAddressService addressService,
HttpClient client, IConfiguration configuration,
ILogger<MatrixService> logger)
    {
        _addressService = addressService;
        _client = client;
        _logger = logger;
        _googleApiKey = configuration["Google:Key"] ?? throw new
InvalidOperationException();;
    }

    public async Task<IEnumerable<MatrixUnitResponse>>
CreateMatrix(MatrixRequest request)
    {
        List<MatrixUnitResponse> results = new
List<MatrixUnitResponse>();
        List<Address> origins = new();
        foreach (var id in request.OriginAddressIds)
        {
```

```
            var address = (await
_addressService.GetAddressById(id)).Address ?? throw new
NullReferenceException();
            origins.Add(address);
        }
        List<Address> destinations = new();
        foreach (var id in request.DestinationAddressIds)
        {
            var address = (await
_addressService.GetAddressById(id)).Address ?? throw new
NullReferenceException();
            destinations.Add(address);
        }

        RouteMatrixRequest routesApiRequest = new()
        {
            Destinations = destinations.Select(dest =>
                new Destinations(dest.GoogleLongitude.Value,
dest.GoogleLatitude.Value)).ToArray(),
            Origins = origins.Select(orig =>
                new Origins(orig.GoogleLongitude.Value,
orig.GoogleLatitude.Value)).ToArray(),
            TravelMode = "DRIVE",
            RoutingPreference = "TRAFFIC_UNAWARE"
        };

        var googleResponse = await
SendToRoutesApi(routesApiRequest);

        foreach (var unit in googleResponse)
        {
            results.Add(new MatrixUnitResponse(
                origins[unit.OriginIndex].Id,
                destinations[unit.DestinationIndex].Id,
                unit.DistanceMeters / 1000,
                new TimeSpan(0, 0,
Convert.ToInt32(unit.Duration[..^1])),
                null));
        }

        return results;
    }

    private async Task<IEnumerable<RouteMatrixResponseElement>>
SendToRoutesApi(RouteMatrixRequest requestBody)
    {
        using var request = new HttpRequestMessage();
        var content = new
StringContent(JsonConvert.SerializeObject(requestBody, new
JsonSerializerSettings()
        {
            ContractResolver = new
CamelCasePropertyNamesContractResolver(),
```

```
            Formatting = Formatting.Indented
        }));
        content.Headers.ContentType =
System.Net.Http.Headers.MediaTypeHeaderValue.Parse("application/
json");

        request.Content = content;
        request.Method = new HttpMethod("POST");

request.Headers.Accept.Add(System.Net.Http.Headers.MediaTypeWith
QualityHeaderValue.Parse("application/json"));
        request.Headers.Add("X-Goog-FieldMask",
"originIndex,destinationIndex,duration,distanceMeters,status,con
dition");
        request.Headers.Add("X-Goog-Api-Key", _googleApiKey);

        request.RequestUri = new
Uri("https://routes.googleapis.com/distanceMatrix/v2:computeRout
eMatrix",
            UriKind.Absolute);
        var response = await _client.SendAsync(request,
HttpCompletionOption.ResponseHeadersRead);
        _logger.LogInformation(response.ToString());
        if (response.StatusCode == HttpStatusCode.OK)
        {
            var objectResponse = await
response.Content.ReadAsStringAsync();

            return
Newtonsoft.Json.JsonConvert.DeserializeObject<IEnumerable<RouteM
atrixResponseElement>>(
                objectResponse) ?? throw new
InvalidOperationException();
        }

        throw new InvalidOperationException();
    }
}
```

GeoinfoService/GoogleService

```
using SolarLogistics.GeoInformationService.Domain;
using SolarLogistics.GeoInformationService.GoogleApiNSwag;
using SolarLogistics.GeoInformationService.GoogleService.Models;

namespace SolarLogistics.GeoInformationService.GoogleService;

class GoogleService : IGoogleService
{
    private readonly GoogleMapsApiClient _googleHttpClient;
    private readonly string _googleMapsRouteUiAddress;
    private readonly string _googleMapsPlacesUiAddress;
    private readonly ILogger<GoogleService> _logger;
```

```
    public GoogleService(GoogleMapsApiClient googleHttpClient,
IConfiguration configuration, ILogger<GoogleService> logger)
    {
        _googleHttpClient = googleHttpClient;
        _logger = logger;
        _googleMapsRouteUiAddress =
configuration["Google:RoutesBaseUrl"] ?? throw new
InvalidOperationException();
        _googleMapsPlacesUiAddress =
configuration["Google:PlacesBaseUrl"] ?? throw new
InvalidOperationException();
    }

    public async Task<DirectionDto> GetDirectionAsync(Address
destinationAddress, Address originAddress)
    {
        if (destinationAddress.GooglePlainString == null &&
originAddress.GooglePlainString == null)
        {
            destinationAddress = await
GetGeocodedAddressAsync(destinationAddress);
            originAddress = await
GetGeocodedAddressAsync(originAddress);
        }

        return await
GetDirectionAsync(destinationAddress.GooglePlainString,
originAddress.GooglePlainString)
            with
            {
                Destination = destinationAddress,
                Origin = originAddress
            };
    }

    public async Task<DirectionDto> GetDirectionAsync(string
destinationAddress, string originAddress)
    {
        var response = await _googleHttpClient.DirectionsAsync(
            destination: destinationAddress,
            origin: originAddress,
            language: Language.Uk);
        var route = response.Routes.FirstOrDefault() ??
                    throw new InvalidOperationException("Google
maps route wasn`t found");
        var leg = route.Legs.Single();
        _logger.LogInformation(leg.Duration.Text);
        _logger.LogInformation(leg.Duration.Value.ToString());
        var result = new DirectionDto
        {
            Distance = leg.Distance.Value,
```

```
            Duration = new TimeSpan(0, 0, 0,
(int)leg.Duration.Value),
            GoogleMapsUiUrl = _googleMapsRouteUiAddress +
                              leg.StartAddress.Replace(" ",
"%20") + "/" +
                              leg.EndAddress.Replace(" ", "%20")
+ "/",
            Destination = await GetGeocodedAddressAsync(new
Address()
            {
                GoogleLatitude = leg.EndLocation.Lat,
                GoogleLongitude = leg.EndLocation.Lng,
                GooglePlainString = leg.EndAddress
            }),
            Origin = await GetGeocodedAddressAsync(new Address()
            {
                GoogleLatitude = leg.StartLocation.Lat,
                GoogleLongitude = leg.StartLocation.Lng,
                GooglePlainString = leg.EndAddress
            })
        };

        return result;
    }

    public async Task<Address> GetGeocodedAddressAsync(Address
address)
    {
        GeocodingResponse response;
        var fullAddress = string.Join(" ",
            address.Building,
            address.Street,
            address.Sublocality,
            address.City,
            address.State,
            address.Country,
            address.PostalCode);
        if (!string.IsNullOrWhiteSpace(fullAddress))
        {
            response = await _googleHttpClient.GeocodeAsync(
                address: fullAddress,
                language: Language.Uk);
        }
        else
        {
            if
(!string.IsNullOrWhiteSpace(address.GooglePlainString))
            {
                response = await _googleHttpClient.GeocodeAsync(
                    address: address.GooglePlainString,
                    language: Language.Uk);
            }
            else
```

```
            {
                response = await _googleHttpClient.GeocodeAsync(
                    latlng:
$"{address.GoogleLatitude},{address.GoogleLongitude}",
                    language: Language.Uk);
            }
        }

        var result = response.Results.First();
        address.Building = result.AddressComponents
                                 .FirstOrDefault(x =>
x.Types.Contains(Types.StreetNumber))
                                 ?.LongName
                             ?? address.Building;

        address.Street = result.AddressComponents
                               .FirstOrDefault(x =>
x.Types.Contains(Types.Route))
                               ?.LongName
                           ?? address.Street;

        address.Sublocality = result.AddressComponents
                              .FirstOrDefault(
                                  x =>

x.Types.Contains(Types.Sublocality) || // City part
(Шевченківський район міста Києва)

x.Types.Contains(Types.AdministrativeAreaLevel2)) // State part
(Криворізький район Дніпропетровської області)
                             ?.LongName
                         ?? address.Sublocality;

        address.City = result.AddressComponents
                             .FirstOrDefault(x =>
x.Types.Contains(Types.Locality))
                             ?.LongName
                         ?? address.City;

        address.State = result.AddressComponents
                              .FirstOrDefault(x =>
x.Types.Contains(Types.AdministrativeAreaLevel1))
                              ?.LongName
                          ?? address.State;

        address.Country = result.AddressComponents
                              .FirstOrDefault(x =>
x.Types.Contains(Types.Country))
                              ?.LongName
                          ?? address.Country;

        address.PostalCode = result.AddressComponents
```

```
                                .FirstOrDefault(x =>
x.Types.Contains(Types.PostalCode))
                                ?.LongName
                            ?? address.PostalCode;

        address.GoogleMapsLink =
$"{_googleMapsPlacesUiAddress}{result.FormattedAddress.Replace("
", "%20")}/";

        address.GooglePlainString = result.FormattedAddress;
        address.GoogleLongitude = result.Geometry.Location.Lng;
        address.GoogleLatitude = result.Geometry.Location.Lat;

        return address;
    }


}
```