Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система керування вантажними перевезеннями
Back-end клієнтської частини
(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-6

Мацак С. О.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник ст. викл. кафедри ПІ Саманцов О. О.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З. В. Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Мацаку Станіславу Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система керування вантажними перевезеннями. Back-end клієнтської частини.

Затверджена наказом по університету від 20.05. 2024р. № 471 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 10.06.2024

3. Вихідні дані до роботи Розробити програмну систему керування вантажними перевезеннями, а саме такі елементи програми: серверна частина клієнтського застосунку

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення , висновки, додатки

_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	08.04.2024	*виконано*
2	Створення специфікації ПЗ	15.04.2024	*виконано*
3	Проектування ПЗ	25.04.2024	*виконано*
4	Розробка ПЗ	19.05.2024	*виконано*
5	Тестування ПЗ	25.05.2024	*виконано*
6	Оформлення пояснювальної записки	30.05.2024	*виконано*
7	Підготовка презентації та доповіді	03.05.2024	*виконано*
8	Попередній захист	26.05.2024	*виконано*
9	Нормоконтроль, рецензування	05.06.2024	*виконано*
10	Здача роботи у електронний архів	10.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	10.06.2024	*виконано*

Дата видачі завдання 08 травня 2024р.

Студент  Мацак С. О.
(підпис)

Керівник роботи _____ ст.викл. кафедри ПІ Саманцов О.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 108 стор., 18 рис., 12 джерел, 3 додатки.

ЛОГІСТИКА, NODE JS, ДОСТАВКА, JAVA SPRING, ВАНТАЖ, ТРАНСПОРТУВАННЯ, МІКРОСЕРВІСИ.

Об’єкт розробки – backend-частина програмної системи для керування вантажними перевезеннями.

Мета розробки – створення системи вантажних перевезень як посередника між водіями та клієнтами.

Метод рішення – середовище розробки JetBrains IDEA, JetBrains WebStorm, мови програмування JavaScript, Java, фреймворки Spring, Node.JS Express.

У результаті розробки створено клієнтську серверну частину програмної системи, що складається з мобільного застосунку, клієнтської та адміністраторської серверних частин та адміністраторського веб-застосунку.

LOGISTICS, NODE JS, DELIVERY, JAVA SPRING, CARGO, TRANSPORTATION, MICROSERVICES.

The object of development is the backend part of a software system for cargo transportation management.

The purpose of development is to create a freight transportation system as an intermediary between drivers and customers.

Solution method - JetBrains IDEA development environment, JetBrains WebStorm, JavaScript, Java, and other programming languages, Java, Spring, Node.JS Express frameworks.

As a result of the development, a client server part of the software system was created, consisting of a mobile application, client and administrative server parts, and an administrative web application.

Я, Мацак Станіслав Олегович, студент групи ПЗПІ-20-6, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система керування вантажними перевезеннями. Back-end клієнтської частини», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений із діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Мета кваліфікаційної роботи – проєктування архітектури для програмної системи для керуванням вантажними перевезеннями та програмна реалізація серверної частини цієї системи.

Було розроблено клієнтську back-end частину системи для керування вантажними перевезеннями. Основна мета серверної частини системи - забезпечити ефективну взаємодію між усіма учасниками процесу доставки. API клієнтської частини дозволяє замовникам вказувати параметри вантажу, такі як розміри, тип вантажу, а також час і місце доставки. Водночас, водії мають змогу вказувати свою готовність до прийняття замовлень, визначаючи радіус пошуку актуальних замовлень або плануючи маршрути для майбутніх поїздок.

Система використовує алгоритми для вибору найоптимальніших варіантів доставки, базуючись на критеріях швидкості, вартості та доступності водіїв у конкретний момент часу. Для цього, зокрема, існує вибір між терміновою доставкою або доставкою, запланованою на кілька тижнів чи місяців наперед. Крім того, система надає можливість вибору послуги з завантаження та розвантаження вантажу.

Для підтримки високої продуктивності та масштабованості, система побудована на архітектурі мікросервісів з використанням Node.js та Java Spring для окремих сервісів. Основними сервісами є сервіси для оброблення замовлення та взаємодії з водієм. Також створено сервіс для аутентифікації та авторизації користувачів. Зв’язок із сервісами відбувається за допомогою легкого API, що лише викликає сервіси та повертає кінцевий результат користувачу (мобільному застосунку). Використання бази даних MongoDB дозволяє забезпечити високу доступність та гнучкість управління даними.

Результатом роботи є серверна частина системи вантажних перевезень, що легко розгортається та дає змогу створити ефективний мобільний клієнт.

.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної галузі

Розвиток Інтернету наприкінці XIX ст. – на початку XXI ст. призвів до появи нової форми надання послуг. Електронна комерція стала рушієм розвитку великої кількості дотичних до неї галузей та пов’язаних із ними спеціалістів, що, зокрема, стосується і логістики.

Транспортна логістика – це система з організації доставки, а саме з переміщення будь-яких матеріальних предметів, речовин тощо з однієї точки в іншу за оптимальним маршрутом. Транспорт як провідна галузь економіки забезпечує функціонування і розвиток усіх галузей господарського комплексу країни, виступає фундаментальною основою їх взаємодії у процесі економічного розвитку [1].

Сфера логістики, зокрема завдяки електронній комерції, значно розвинулась та стала доступнішою для пересічних громадян як у світі, так і в Україні. Упродовж 2010-2019 років зафіксовано зростання розміру ринку логістичних послуг в Україні у понад 3,8 рази (див. рис. 1.1) [2].

Рисунок 1.1 – Обсяг реалізованих логістичних послуг в Україні за 2010-2019 рр. (за даними Держстату)

Велику частину послуг складають саме поштові компанії та кур'єрська діяльність. Зазвичай взаємодія користувача із поштовими компаніями виглядає наступним чином:

– відправник створює замовлення через застосунок, або ж оформлює замовлення на відправлення безпосередньо у вантажному відділенні;
– відправник передає посилку кур'єру або працівнику відділення;
– відправник отримує чек та може відслідковувати статус замовлення у мобільному додатку, за допомогою PUSH-сповіщень чи SMS-повідомлень;
– отримувач отримує вантаж, відправник отримує інформацію про завершення доставки.

Натомість життєвий цикл замовлення є значно складнішим, а величезна кількість процесів – прихована від клієнтів. Безпосереднє перевезення ж є чи не найменш простою та незначною з погляду витрачених ресурсів частиною. Саме простота взаємодії приваблює користувачів, і вони готові платити більші гроші за більш зрозумілий та якісний сервіс, який, до того ж, працює швидко.

Логістичні компанії створюють конкуренцію, пропонуючи максимальну якість, швидкість доставки, різні тарифні плани, спеціальні умови для бізнесу, послуги кур'єрів «від дверей до дверей», зручні методи оплати та близькість пунктів відправлення/видачі пакунків до клієнтів тощо. Певні компанії, зокрема Нова Пошта, впроваджують інноваційні методи сортування посилок [3], що дозволяє без участі людини за лічені хвилини завантажити автомобілі з різними точками призначення.

Однак іноді необхідно перевезти велику кількість вантажу, до того ж об'ємного. Наприклад, є потреба змінити місце проживання та перевезти свої речі з однієї квартири на іншу. Або ж необхідно перевезти усе обладнання зі студії звукозапису на концерт, чи доставити все для ремонту будинку з міського центру до віддаленого селища. Зазвичай люди шукають знайомих з вантажними автомобілями, орендують авто за контактами, знайденими в соціальних мережах чи сайтах оголошень та наймають вантажників. Такий процес не є відкритим, зручним та швидким. Окрім того, необхідно власноруч переконуватись у доброчесності перевізників та вантажників,

покладатись на усні домовленості, знаходити вигідні пропозиції з різних куточків мережі Інтернет. Натомість вже давно існують компанії, що пропонують послуги «мувінгу».

Мувінг (від англійського «moving» – «переїзд») — організація процесу перевезення і зберігання речей під час переїзду (квартирного, офісного, будинкового, дачного тощо), власне надання клієнту на платній основі послуг переїзду [4].

Мувінгові компанії дозволяють зекономити кошти на відправленні великої кількості речей, а також пропонують ряд послуг, які поштові перевізники зазвичай не надають, зокрема:

- обережне пакування речей у різний матеріал, залежно від його крихкості та цінності;
- самостійне завантаження та розвантаження речей;
- орієнтованість на збереження майна, використання спеціальних матеріалів та інструментів для транспортування та завантаження;
- гнучкий графік перевезень, орієнтований на клієнта;
- перевезення коштовних чи великогабаритних вантажів.

Популяризації мувінгових компаній, зокрема, посприяв розвиток інформаційних технологій: поява високошвидкісного інтернету, зменшення вартості електроніки і (найголовніше) популяризація та розвиток онлайн спільнот по інтересам, які розробили свою систему модерації та правила гри, збільшивши рівень довіри між користувачами [5].

1.2 Виявлення та вирішення проблем

Зазвичай послуги мувінгових компаній є недешевими. Однак існує чимало сценаріїв, коли потрібен не увесь список послуг, що вони надають. Клієнт завжди намагається знайти максимально дешевий варіант послуги за прийнятного для нього рівня якості.

Собівартість набору послуг мувінгових компаній важко знизити з ряду причин. Окрім того, мувінгові компанії зазвичай є прив’язаними локально до певної місцевості – великого міста чи агломерації, і навіть за наявності мережі з кількох

відділень, компанії відмовляються виконувати замовлення у віддалених куточках (сільській місцевості, важкодоступній місцевості, в прилеглих до зони бойових дій територіях тощо), або ж значно підвищують цінник за надані послуги.

Для того, щоб вирішити обидві проблеми, необхідно переглянути економічну модель та знайти кращий відповідник для зазначених умов. Однією з розвинених моделей є економіка спільної участі.

Економіка спільної участі – це економічна модель, в якій люди можуть позичати або орендувати активи, що належать іншим. Модель економіки спільного участі, використовується переважно, коли ціна активу відносно висока, а актив використовують не весь час повною мірою [6]. Ця система приймає різні форми, часто використовуючи інформаційні технології для розширення можливостей людей, корпорацій, некомерційних організацій та уряду, надаючи інформацію, яка дозволяє розподілити, спільно або повторно використати надлишкові потужності в товарах і послугах. Загальною передумовою є те, що, коли інформація про надлишкові товари є загальною, то це розширює важливість цих товарів від бізнесу і фізичних осіб до суспільства [7].

Відповідно можна значно підвищити ефективність використання ресурсів, що є надлишковими, активів, що використовуються не в повній мірі. У випадку мувінгових компаній, що займаються вантажними перевезеннями, їх автомобіль може повертатися до офісу порожнім. Хоча цілком імовірно, що за маршрутом його повернення, чи з невеликим відхиленням, він міг би виконати ще одне чи кілька замовлень.

У цьому й полягає значна перевага систем, що ґрунтуються на основі принципів економіки спільної участі. Якщо створити систему, де водій може вказати, коли він буде рухатись за певним маршрутом, і яку частину свого ресурсу (вантажного відділення автомобіля) він готовий використати для виконання замовлення, можна вирішити відразу декілька проблем:

- ефективне використання транспорту, з мінімізацією порожніх поїздок, відповідно зменшенням витрат на пальне та інші експлуатаційні витрати;

– підвищення доходів для водіїв – за рахунок виконання замовлень протягом тих відрізків поїздок, де перевезення вантажу не планувалось взагалі;

– широке географічне охоплення – дорожня система, зокрема в Україні, є дуже розвиненою, тож є чималі шанси, що маршрут запланованого перевезення вантажу збігається з маршрутом когось із водіїв, або ж знаходиться поруч, із невеликим відхиленням. Це стосується як сільської чи гірської місцевості, де в будь-якому випадку проживають люди і мають автомобільне сполучення, так і територій, прилеглих до зони бойових дій. Зокрема, волонтери та військові на грузових машинах чи пікапах мають змогу допомогти тим, хто цього потребує, і перевезти речі замовника на тилові території, оскільки часто повертаються з порожніми автівками;

– зменшення навантаження на дорожню систему у великих містах;

– зменшення вартості перевезення вантажу, особливо за найбільш популярними маршрутами, що однозначно приваблює чимало клієнтів.

Візьмемо до прикладу сценарій використання служб таксі, що хоча й орієнтуються на перевезення, однак мають в якості основної цінності не товар, пакунок чи вантаж, а людину – клієнта, що бажає перемістити між певними точками. У 2009 році компанія Uber змінила уявлення про служби таксі - надавши своїм клієнтам, першокласний, надійний та зручний транспорт за доступною ціною. Особливість платформи – поєднання водіїв з особистими авто та пасажирів, що взаємодіють між собою через мобільні додатки. Компанія не мала власного автопарку, а лише надавала послуги з пошуку водіїв та клієнтів для обох сторін.

Отже, проблеми, що стосуються безпеки, надійності та адекватного ціноутворення, що наявні при транспортуванні великих об’ємних перевезень, в службах таксі були максимально усунуті завдяки появі компаній-посередників між водіями та клієнтами, що надають послуги з пошуку замовлень та піклуються про належний рівень наданих послуг та ринкові ціни на послуги – в залежності від поточного попиту та маршруту поїздки.

Створення сервісу-посередника між клієнтами, що бажають транспортувати свої речі та водіями, що готові виконати такі замовлення, вирішує певний перелік проблем.

Для клієнта вирішуються такі проблеми:

– не потрібно власноруч шукати різні компанії, що займаються транспортування речей, читати оголошення в соціальних мережах, слідкувати за доступністю водіїв та підлаштовуватись під їх графік;

– зникає необхідність контактувати (за телефоном чи в письмовому вигляді) з компаніями з перевезення, водіями чи вантажниками;

– зникає недовіра до надійності водія чи перевізників (наприклад, без сервісу клієнт ніяк не застрахований від того, що водій просто відмовиться від перевезення у найвідповідальніший момент; у разі виникнення проблем сервіс зобов'язаний замінити водія) та щодо їх намірів (сервіс ідентифікує водіїв, що співпрацюють з компанією, натомість власноруч перевірити документи водія вдається не завжди);

– складність вибору зручного і для клієнта, і для водія часу;

Натомість водій позбувається таких проблем:

– ненадійність клієнта, зокрема вчасної та повної оплати послуг від нього;

– необхідність ведення діалогів у месенджерах чи за телефоном, зокрема в дорозі;

– вирішення проблем проводиться за участю третьої сторони, що зважає на правила та обов'язки, яких дотримуються усі сторони під час виконання послуг;

– можливість відмінити заплановані поїздки за форс-мажорних обставин, без втрати репутації від клієнтів;

– висока вартість поїздки пустого вантажного автомобіля.

1.3 Аналіз конкурентів

Конкурентами системи, що створюється, можна вважати як транспортні компанії, що здійснюють вантажні перевезення, оскільки їх основна задача збігається з нашою – транспортувати набір предметів з одінєї точки світу до іншої, так і служби таксі, що працюють за бізнес-моделлю Uber, оскільки вони мають схожу концепцію взаємодії клієнта та системи.

1.3.1 Нова Пошта

Нова пошта є українською компанією з експрес-доставки, що першочергово орієнтована на доставку для фізичних осіб. Має широку мережу відділень, поштоматів по всій країні.

Веб-сайт компанії - https://novaposhta.ua.

Переваги:

- зручний та приємний інтерфейс (див. рис. 1.2);
- розрахунок вартості відправлення безпосередньо в застосунку;
- велика кількість відділень та поштоматів;
- наявність страхування відправлення.

Недоліки:

- після заповнення всієї інформації про відправлення (що знаходиться на 1 екрані), при спробі змінити місце відправлення чи отримання посилки, вся інформація скидається і її доводиться вносити знову;
- складність оформлення великих перевезень;
- достатньо висока ціна для оформлення об’ємних чи важких перевезень;
- відсутність можливості замовити послугу завантаження/розвантаження (кур’єр може допомогти, якщо матиме можливість та бажання, однак безпосередньо послуга відсутня).

Отже, Нова Пошта теоретично може виконати завдання з транспортування всього набору речей з однієї квартири на іншу, однак вона все ж лишається

поштовою компанією, через що не має ряду необхідних послуг. Окрім того, має відповідне ціноутворення.

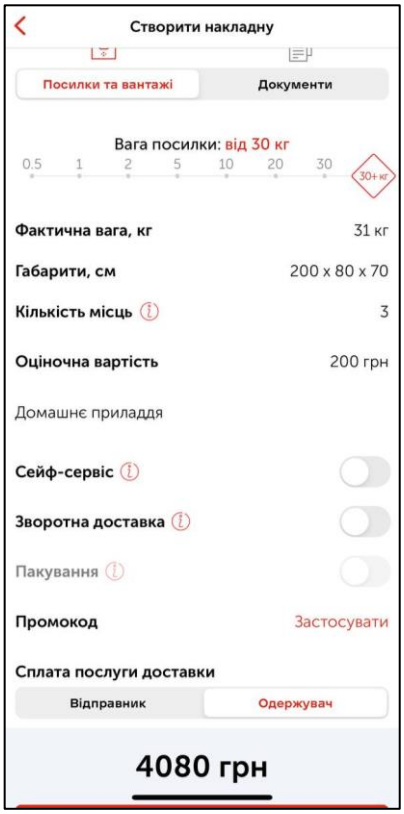


Рисунок 1.2 – Інтерфейс створення замовлення через мобільний додаток «Нова Пошта»

1.3.2 Мураха

Мураха – засновник індустрії мувінгу в Україні, пропонує комплексні послуги з переїзду дому та бізнесу.

Веб-сайт компанії: https://www.muraha.ua.

Переваги:

– компанія є мувінговою, тобто близької за ідеологією до системи, що створюється;

– наявні послуги з пакування та розпакування речей, складання та розбирання меблів, прибирання до заселення чи після виїзду;

– можливість сплати за послуги онлайн.

Недоліки:

– відсутність розрахунку безпосередньо в системі – клієнту все рівно потрібно зв’язуватись з оператором для визначення ціни переїзду;

– відсутність мобільного додатку. Компанія Мураха має лише веб-застосунок (див. рис. 1.3);

– служба підтримки комунікує з клієнтом лише за електронною поштою;

– працює локально в кількох містах.

Отже, Мураха є компанією, близькою за концепцією до системи, що розробляється. Однак послуги надаються безпосередньо компанією та її водіями. Немає можливості виконувати перевезення, не будучи співробітником компанії, неможливо замовити перевезення, проживаючи в місті, де не розташовано офісу компанії. Відповідно, ціна є вищою за ту, що могла би бути за умови використання економіки спільної участі. До того ж, ціну перевезення взагалі не можна дізнатись безпосередньо в системі, а також компанія не має мобільного застосунку.

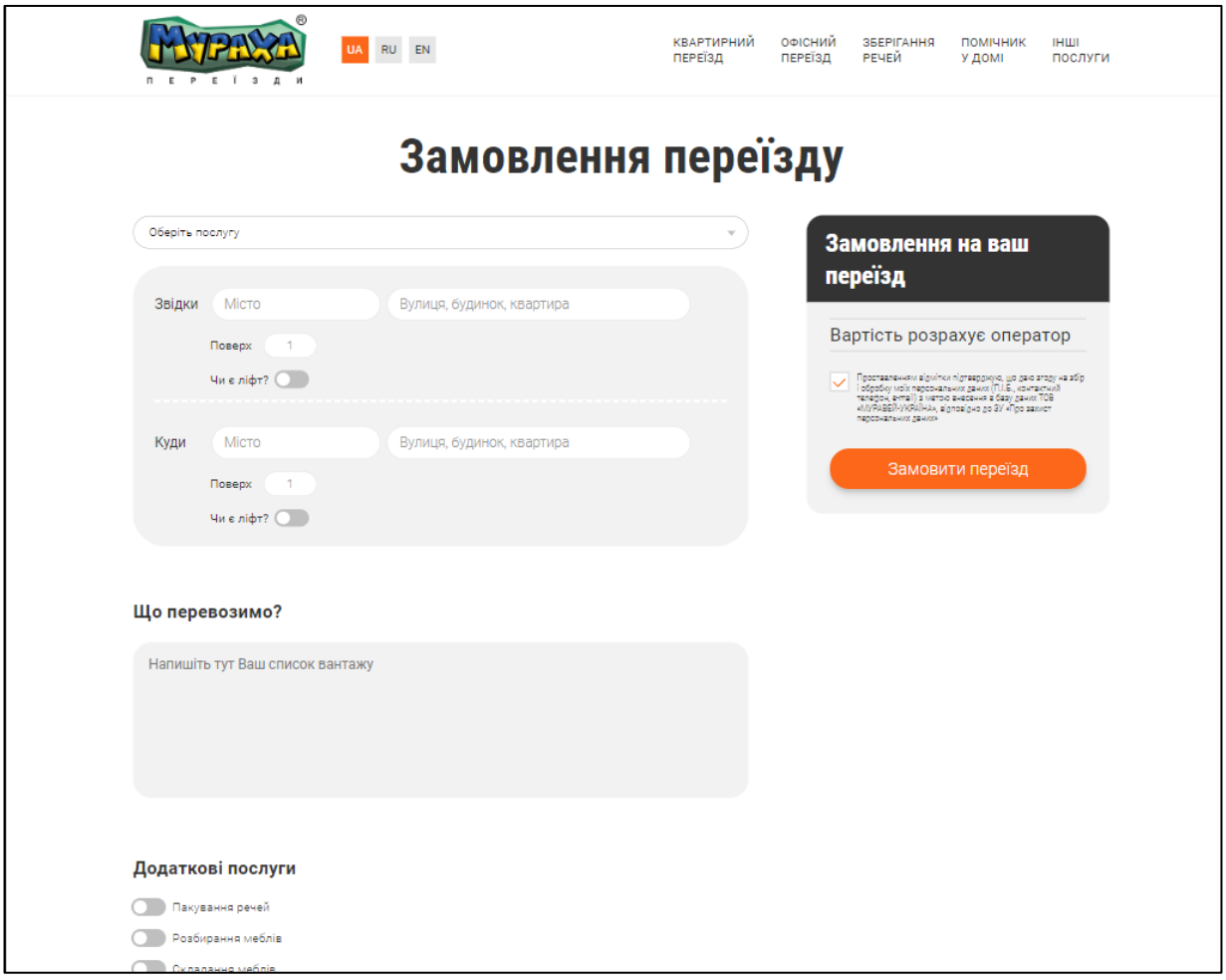


Рисунок 1.3 – Веб-інтерфейс замовлення переїзду в компанії «Мураха»

1.3.3 Uklon

Uklon – український райдхейлінг сервіс, що працює у 27 містах України та об’єднує водіїв і пасажирів.

Веб-сайт компанії: https://uklon.com.ua

Переваги:

– швидка доставка, яку можна замовити відразу і впродовж години вже завершити перевезення;

– використання принципів економіки спільної участі;

– зручний мобільний застосунок (див. рис. 1.4);

– система рейтингів водія та користувача-замовника;

– служба підтримки безпосередньо в застосунку;

– динамічне визначення вартості в залежності від зайнятості автомобілів та ситуації на дорогах.

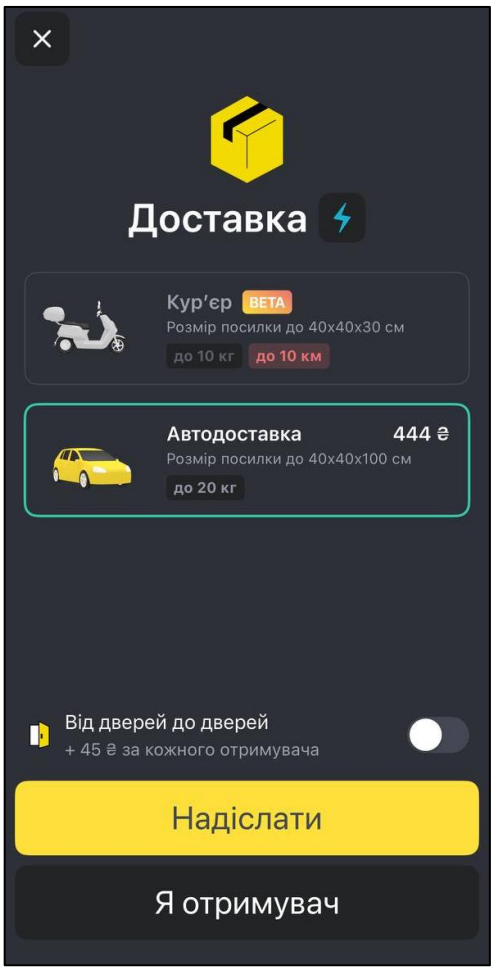


Рисунок 1.4 – Інтерфейс мобільного застосунку «Uklon»

Недоліки:

- значні обмеження в розмірах вантажу для перевезення;
- географічна обмеженість (послуги надають у великих містах, однак маленькі міста та села не можуть скористатись ними, або ж вартість помітно зростає).

Отже, Uklon використовує концепцію економіки спільної часті, завдяки чому завжди є висока доступність автомобілів та водіїв. Однак цей застосунок все ж позиціонується як надавач послуг таксі, а не вантажних перевезень, тож можливість надіслати посилку тут – лише опція. Яка, до того ж, працює нормально в межах міста, однак не здатна задовольнити потребу в переміщенні об’ємного вантажу на велику відстань.

1.3.4 Висновок з аналізу конкурентів

Проаналізувавши конкурентів можна дійти до висновку, що об’єднавши переваг кожної з вище описаних платформ можна створити нову систему, що зможе надати користувачам новий досвід. Така система зможе надати такі послуги, якими незручно користуватись у застосунках чи веб-версіях наявних на сьогодні конкурентів.

1.4 Постановка задачі

Під час виконання роботи має бути спроектована та реалізована клієнтська backend-частина програмної системи для керування вантажними перевезеннями. Її архітектура має бути спроєктованою згідно з принципами економіки спільної участі.

Користувач-замовник повинен мати можливість оформлювати замовлення на перевезення вантажу довільного розміру відразу ж, або на заплановану дату та час.

Система повинна визначати вартість такого перевезення та після проведення оплати пропонувати водіям, чиї автомобілі відповідають необхідним критеріям, виконати це замовлення.

Система повинна вирішити проблеми наявних на сьогодні застосунків для перевезення вантажів, серед яких основними є:

– географічна обмеженість зони перевезення;

– висока вартість;

– відсутність можливості водієві запропонувати ресурс свого автомобіля для одиничного перевезення чи нерегулярних поїздок (за аналогією до платформи «Bla-Bla-Car»);

– відсутність розуміння вартості послуг безпосередньо із системи, без додаткового зв'язку із операторами компаній.

Система повинна мати інтеграцію з одним з популярних картографічних сервісів та кілька способів аутентифікації.

Система повинна реалізовувати такий функціонал:

а) функціонал клієнта-замовника:

1) реєстрація в системі за допомогою електронної пошти або Google Sign In;

2) створення замовлення на визначений час та у визначене місце, з набором інформації, переданої від користувача;

3) відслідковування поточного статусу замовлення;

4) оплата за допомогою Google Pay;

5) звернення до служби підтримки;

б) функціонал клієнта-водія:

1) реєстрація в якості водія за допомогою наявного акаунту замовника із заповненням додаткової інформації та верифікацією;

2) внесення даних про заплановані майбутні поїздки, впродовж яких водій може виконати замовлення;

3) надсилання запропонованих перевезень водієві;

4) звернення до служби підтримки;

5) можливість онлайн-підбору замовлень неподалік у визначеному радіусі;

в) додатковий функціонал:

1) зберігання логів та метрик для подальшого використання адміністраторами;

2) надання сервісу для реєстрації модераторів/адміністраторів та керування ними.

Система має забезпечувати синхронне та асинхронне оброблення замовлень водіями. Для клієнтів, що потребують якомога швидшого перевезення вантажу, підбір відбувається за водіями, що очікують замовлення в режимі реального часу, або заздалегідь запланували маршрут, що проходить через вказані замовником локації. Для клієнтів, що вказують точну дату та час відправлення, підбір водіїв починається відразу, і закінчується після того, як один з водіїв прийме замовлення.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Під час виконання кваліфікаційної роботи треба спроектувати та реалізувати клієнтську backend-частину програмної системи для керування вантажними перевезеннями. Ця система повинна відтворювати такий функціонал:

– реєстрація в системі за допомогою електронної пошти. Користувач повинен придумати валідний пароль, що містить мінімум 8 символів, серед яких є цифри та букви;

– реєстрація в системі за допомогою Google Sign In. Не потребує додаткового введення паролю. Серверна частина повинна валідувати токен аутентифікації та переконатись, що його надіслано не зловмисником;

– реєстрація модератора. Потребує прав адміністратора;

– створення замовлення. Клієнт вказує звідки та куди буде здійснено перевезення вантажу, його об’єм та вагу, опис; вказує чи потрібна допомога вантажника (за додаткову плату); обирає дату та час відправлення, або коли відправлення має бути завершено (можливий варіант якомога швидшого виконання замовлення, тобто пошук в режимі реального часу);

– клієнт повинен мати можливість відкладеного замовлення – тобто забронювати перевезення за кілька днів, тижнів чи навіть за місяць до очікуваної дати поїздки, а також можливість синхронного пошуку водія-перевізника в режимі реального часу;

– оплата замовлення. Клієнт може обрати, чи буде він сплачувати готівкою, чи відразу в додатку за допомогою Google Pay. Серверна частина повинна валідувати токен та перевірити інформацію про платіж, перш ніж перевести статус замовлення на наступний етап;

– перегляд замовлення – замовник може бачити інформацію про водія, його автомобіль, рейтинг, час прибуття та на якому етапі замовлення наразі знаходиться;

– звернення до служби підтримки – водій або замовник можуть звернутись до служби підтримки, для того щоб вирішити конфліктну ситуацію чи

проблему, що сталась у ході виконання замовлення. Процес ведення чату модерується адміністраторською частиною системи;

– клієнт може стати водієм. Для цього необхідно надати документи, що підтверджують право на власність автомобілем, технічний паспорт автомобіля, його номер, копію водійського посвідчення;

– водій може змінювати об’єм доступного для перевезення вантажу в залежності від поточної завантаженості вантажного відділення автомобіля;

– водій може вказувати, за якими маршрутами він планує пересуватись у майбутньому, для того щоб отримати замовлення, маршрут якого що збігається з маршрутом водія або проходить з невеликим відхиленням у певних точках;

– водій може вмикати онлайн-підбір, для того щоб шукати замовлення для виконання у заданому в налаштуваннях радіусі;

– зберігати інформацію про запити до сервера – кількість вдалих запитів, кількість помилок сервера (HTTP status 5xx), кількість помилок користувача (HTTP status 4xx), зокрема неавторизованих запитів;

– зберігати інформацію про ряд подій в системі, зокрема про реєстрацію користувачів, аутентифікації користувачів, створення замовлення, кількість активних водіїв у системі, тощо;

– система повинна розраховувати коефіцієнти вартості перевезення на основі певних факторів, таких як поточна завантаженість водіїв, час доби, популярність маршруту, відстань, специфічність вантажу та застосовувати їх під час етапу розрахунку вартості перевезення;

– система повинна розраховувати найкоротшу відстань та план перевезення за умови виконання кількох замовлень одночасно чи послідовно водієм;

– необхідно створити документацію серверної частини клієнтського застосунку та зручний веб-інтерфейс для перегляду цієї документації і тестування запитів;

– потрібно створити зручний робочий простір для розробників як серверної, так і мобільної частини. Уся потрібна інфраструктура повинна

запускатись за допомогою однієї команди в терміналі. Універсальність платформи має бути забезпечена завдяки контейнерам Docker, щоб розробник клієнтської частини міг проводити розробку без додаткових проблем з боку операційної системи.

Серверна частина застосунку повинна мати інтеграцію з такими сторонніми API, як Google Auth API, Google Pay API, Google Maps API.

Повна система складається із мобільного застосунку для клієнтів та водіїв, серверної частини, що відповідає за обробку замовлень та підбір водіїв, а також адміністраторського веб-застосунку та серверного рішення для взаємодії з ним. У межах цієї кваліфікаційної роботи необхідно створити саме серверну частину, що оброблює замовлення та робить підбір водіїв, а також надає API для мобільного застосунку, яким користуються водії та клієнти-замовники.

Сервіси авторизації, підбору та обробки замовлень мають бути відокремленими.

Систему має бути розроблено з використанням мікросервісної архітектури та найкращих принципів використання хмарних технологій.

Система повинна мати високу пропускну здатність, бути масштабованою і стійкою. Всі серверні помилки повинні перехоплюватись і оброблюватись, виникнення помилки не повинно призводити до зупинки процесу виконання програми. Недоступність серверу обробки замовлень не повинна впливати на доступність сервісу авторизації і навпаки.

Розгортання повної системи має відбуватись за допомогою Docker-контейнерів, щоб максимально абстрагуватись від особливостей різних операційних систем та легко вести розробку і подальше розгортання у хмарній інфраструктурі.

API, що сервер надає мобільному застосунку має бути доступним за протколом HTTP/S, та дотримуватись принципів REST API.

Також необхідно дотримуватись основних вимог до безпеки:

– паролі користувачів не можуть зберігатись у відкритому вигляді і потребують шифрування перед збереженням до бази даних;

– усі змінні середовища не повинні знаходитись у публічному доступі, в тому числі у репозиторіях на GitHub. Вони передаються секретно особисто кожному розробнику чи іншій сторонній людині, що повинна мати доступ до системи;

– база даних не повинна бути доступною для публічної Інтернет-мережі, для унеможливлення безпосереднього доступу до неї. Вона має розміщуватись у приватній мережі Docker/Kubernetes чи в іншій мережі, що надає хмарний провайдер;

– усі запити, окрім запитів на реєстрацію, потребують Bearer-токену в Auth заголовку HTTP-запиту. Цей токен повинен мати термін життя не більш ніж 30 хвилин, для уникнення тривалого доступу до ураженого акаунту зловмисником;

– разом із токеном користувач повинен отримувати «refresh-токен», що дозволяє не повертати користувача на екран аутентифікації після закінчення строку дії основного токену.

Основними вимогами до апаратного забезпечення, на якому буде працювати сервер є:

– процесор Intel Core i5-7300HQ або потужніший та новіший процесор, що має 4 ядра, 3.5 GHz;

– не менше 8 Гб оперативної пам’яті DDR3 чи вищого стандарту.

Необхідна наявність встановленого програмного забезпечення Docker, що дозволяє запускати контейнери з сервісами та API.

Під час розробки коду серверної частини застосунку необхідно дотримуватись основних принципів створення стійкої та масштабованої архітектури та принципів написання чистого коду, зокрема принципів DRY, SOLID, а також прагнути до однозначних виразних імен та невеликих функцій.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

Діаграма прецедентів (див. рис. 3.1) відображає основні сценарії використання системи, що створюється.

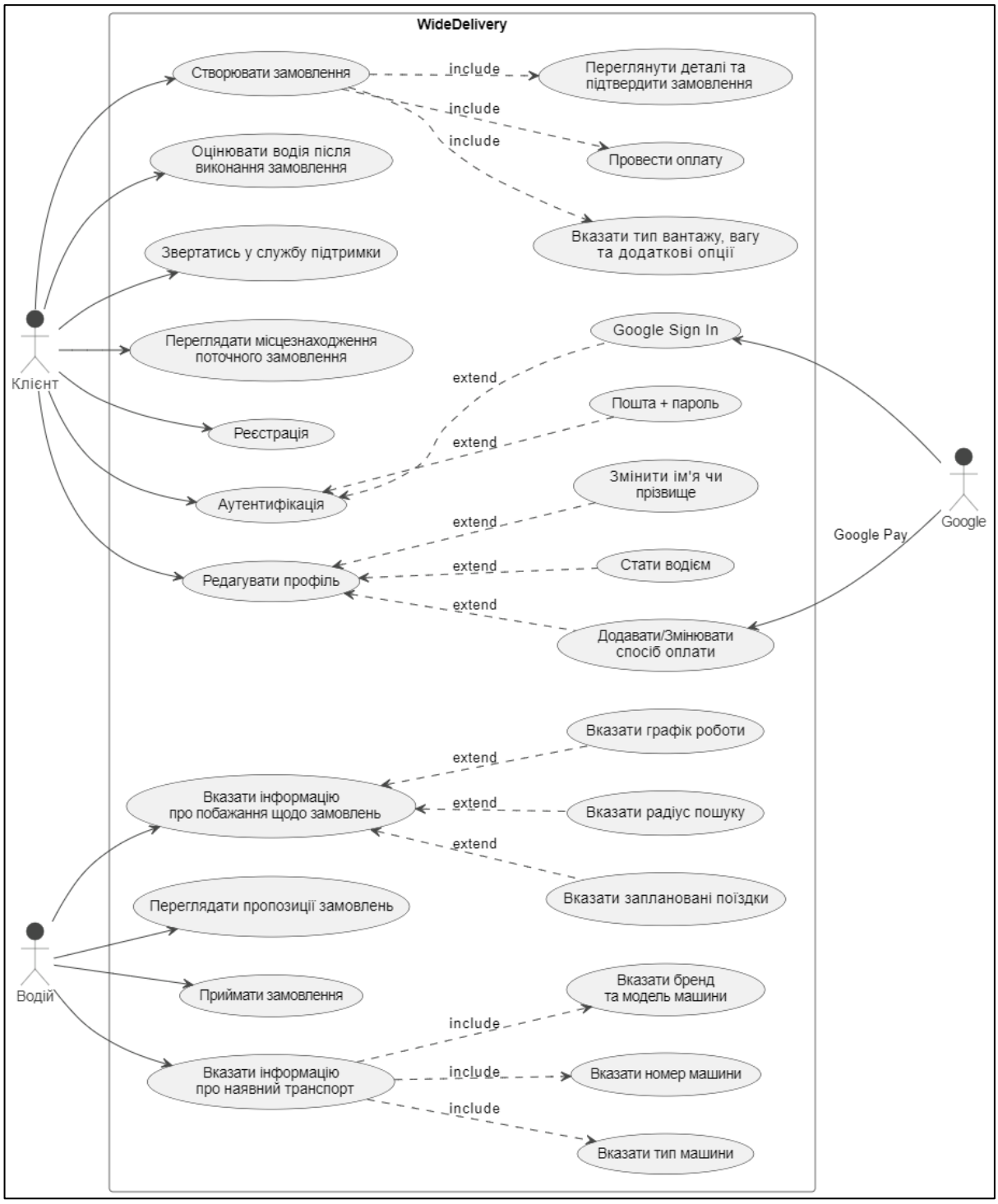


Рисунок 3.1 – Use Case UML діаграма системи вантажних перевезень

UML діаграма послідовності (див. рис. 3.2) зображує основний бізнес-процес в системі – створення та оброблення замовлення.

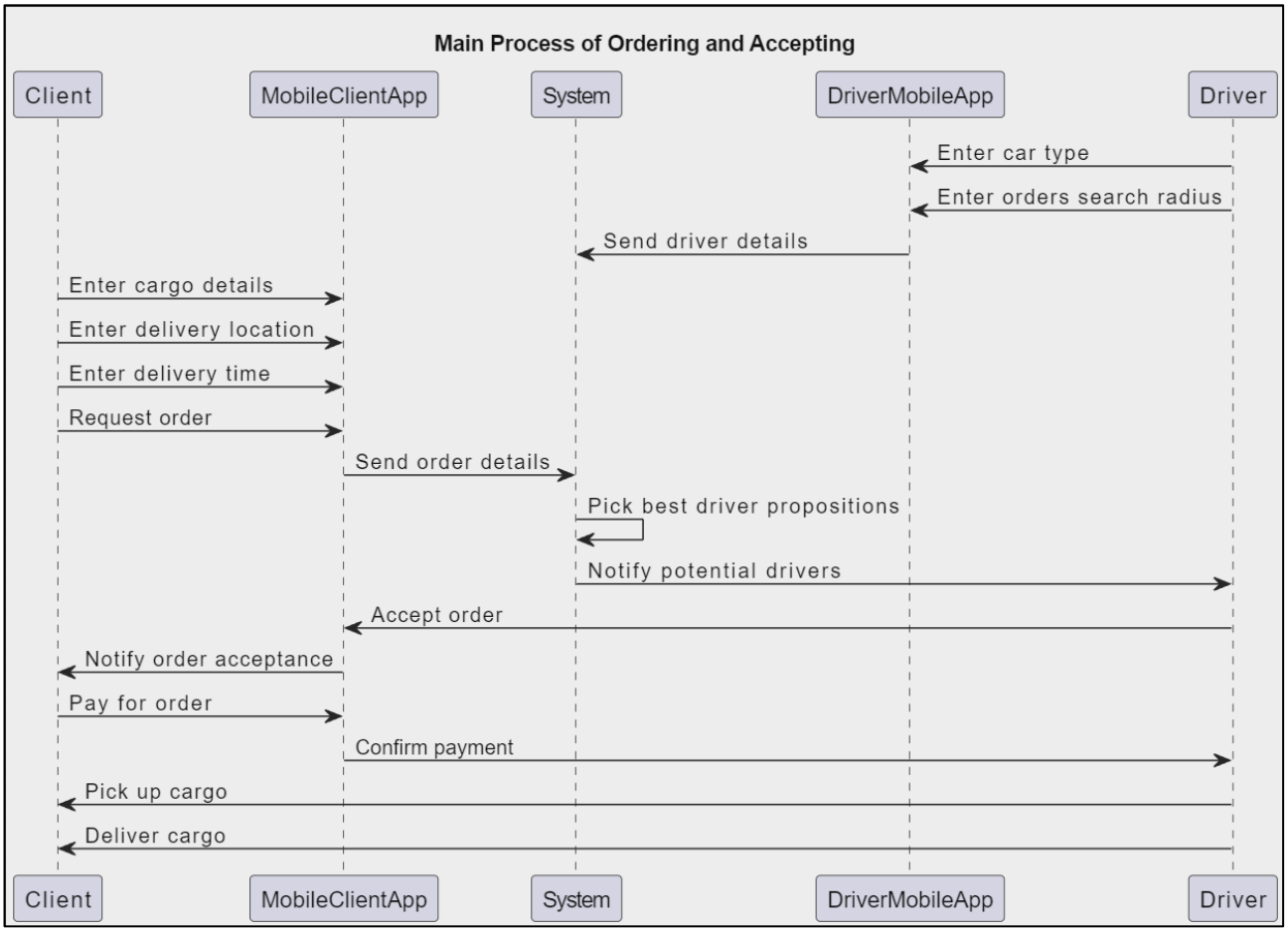


Рисунок 3.2 – UML-діаграма послідовності системи вантажних перевезень

Діаграма потоку даних (див. рис. 3.3) також зображує основний процес обробки замовлення, однак концентрується в першу чергу саме на тому, як переміщуються, зберігаються дані в системі, та в який момент потім використовуються.

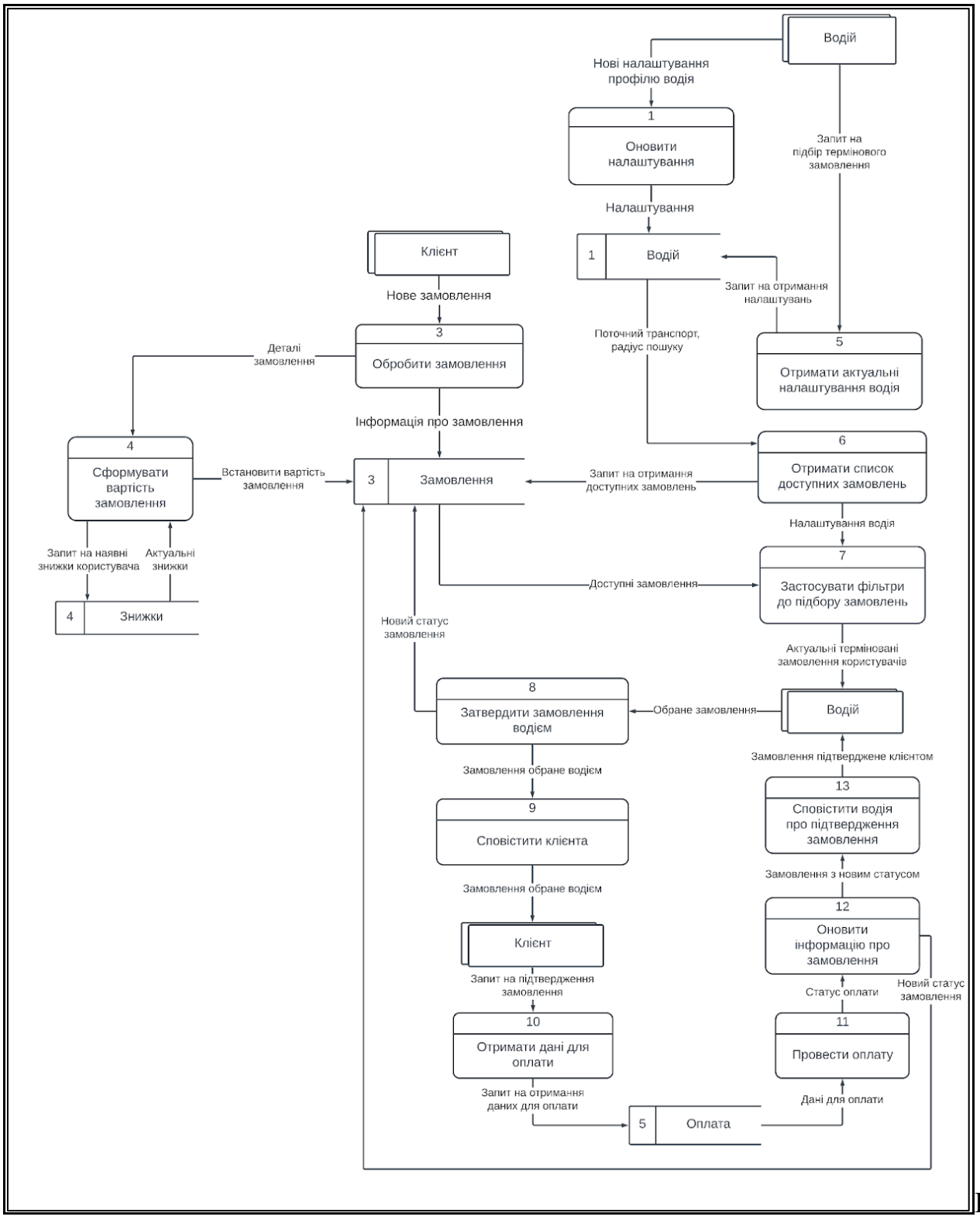


Рисунок 3.3 – UML-діаграма потоку даних

Створення UML-діаграм є важливим етапом для початку проєктування системи, оскільки дозволяє усім учасникам розробки проєкту затвердити спільне бачення багатьох нюансів.

3.2 Проєктування архітектури ПЗ

Архітектура програмної системи - це форма, яку надають цій системі ті, хто її створює. Ця форма полягає у поділі системи на компоненти, розташуванні цих компонентів і способах, якими ці компоненти взаємодіють один з одним. Мета цієї форми - полегшити розробку, розгортання, експлуатацію та обслуговування програмної системи, що міститься в ній. [6]

Основними і найбільш популярними типами архітектури програмних систем є монолітна (також називається «n-рівнева») та мікросервісна архітектури. Обидві архітектури мають свої переваги, і кінцевий вибір має складатись з урахуванням потреб конкретної системи. Неправильне рішення може призвести до використання надмірної кількості ресурсів – часу розробників, коштів на підтримання та розвиток інфраструктури чи внесення змін. Тому дуже важливо на етапі проєктування зважити можливі наслідки обраного рішення.

Ключовими перевагами монолітної архітектури є:

- швидкість розробки початкового продукту та його розгортання;
- легкість тестування;
- легкість роботи із даними – відсутність розподілених транзакцій, наявність централізованої бази даних чи кількох баз даних, доступ до яких все рівно залишається консистентним;
- економічність, у випадку створення невеликої системи.

В свою чергу мікросервісна архітектура має такі переваги:

- гнучкість у розробці – можна обрати різні технології та мови програмування для різних задач в межах однієї системи;
- масштабованість – можна збільшувати вертикально чи горизонтально кожен із мікросервісів, залежно від навантаження на них; розподіляти їх географічно, ближче до клієнта;
- відмовостійкість - у випадку відмови одного мікросервісу, інші частини системи продовжують функціонувати;

– легкість внесення змін впродовж тривалого часу розробки, без потреби внесення змін до великої кількості інших компонентів системи;

– легкість ведення розробки кількома командами розробників, що можуть вносити зміни в межах окремих мікросервісів.

Система, що створюється, має бути стійкою до високих навантажень та легко масштабуватися. Навантаження на різні частини системи можуть сильно відрізнятися від дня до дня та від години до години. Тому було прийнято рішення обрати мікросервісну архітектуру.

При розробці мікросервісів рекомендовано починати із кількох досить об'ємних сервісів, і поступово виділяти з них менші підсервіси. Такий підхід є зручнішим та менш болючим, ніж поєднання кількох мікросервісів або ж підтримка великої кількості сервісів, що мають дуже специфічне призначення і відповідають за надто вузький спектр задач [9].

Основними бізнес-процесами в системі є створення/обробка замовлення клієнта та підбір (створення пропозицій) замовлення для водія. Саме ці бізнес-процеси є фундаментом, навколо яких варто будувати 2 основні мікросервіси. Третім мікросервісом є сервіс авторизації та аутентифікації. Виділяти цю частину в окремий мікросервіс є загальноприйнятою практикою. Таким чином ми створюємо єдину точку, через які повинні проходити всі авторизовані запити.

Оскільки сервіси не мають бути доступними публічно, вся комунікація між ними ведеться за допомогою спільного сервісу, що надає API клієнтам системи, що створюється.

Схема мікросервісної архітектури клієнтської backend-частини системи вантажних перевезень зображена на рис. 3.4.

Сервіси не взаємодіють між собою безпосередньо. API викликає необхідні мікросервіси та отримує від них результат роботи, потім за потреби викликає інший мікросервіс або ж повертає результат виконання клієнтові.

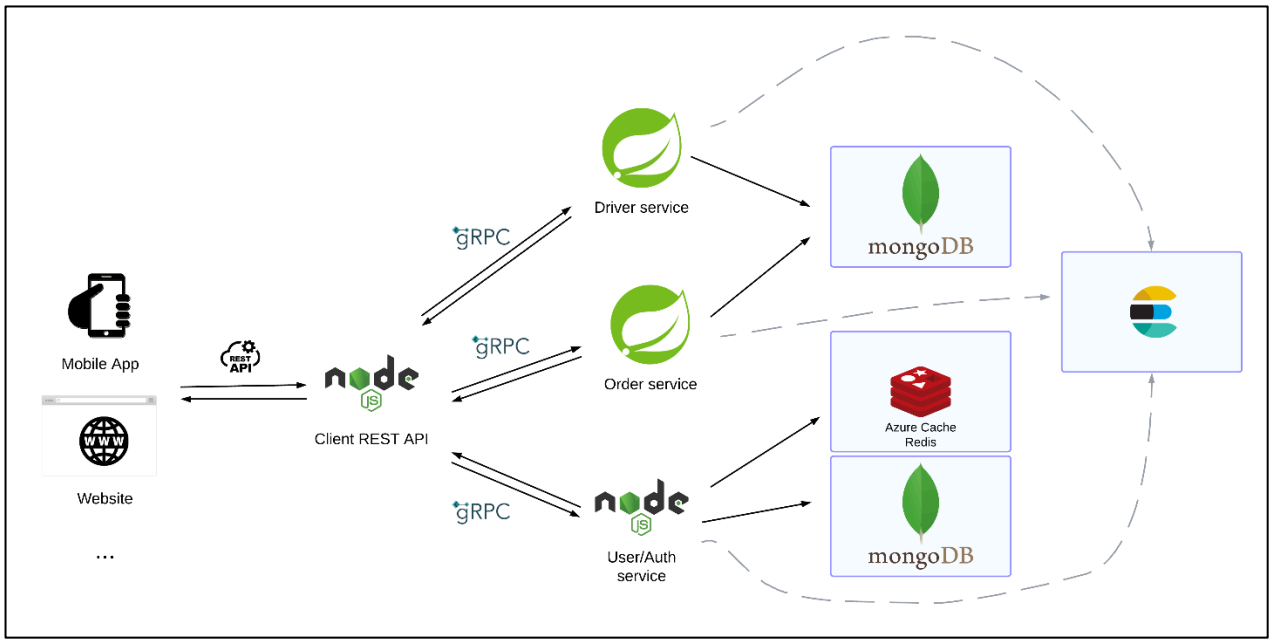


Рисунок 3.4 – Мікросервісна архітектура клієнтської backend-частини системи вантажних перевезень

Отже, існує єдина точка входу, відкрита для публічної мережі – API, побудоване за допомогою Node.JS. API взаємодіє з сервісом аутентифікації та авторизації, що також є сервісом користувачів, що теж побудований на Node.JS. Сервіси обробки замовлення та підбору замовлення для водіїв написані на Java Spring Framework.

Використання Node.JS для API та сервісів автентифікації/користувача, а також Spring для сервісів замовлення та водія - продуманий підхід для балансування продуктивності, масштабованості та обробки даних у реальному часі.

Зокрема, вимогами до сервісу авторизації є:

– мінімальні затримки під час авторизації кожного запиту;
– можливість обробки великої кількості запитів;
– висока доступність та надійність.

Вимоги до спільного API є схожими, втім запити до нього є значно об’ємнішими. Попри це, складної логіки в ньому, як і у сервісі авторизації, немає. API є лише посередником між клієнтом та сервісами, що виконують основну частину роботи. Також API повинно легко горизонтально масштабуватись. Для цього чудово

підходить Node.JS, що відомий своїми неблокуючими операціями вводу/виводу та Event-Driven архітектурою. Особливості Node.JS роблять цю технологію надзвичайно швидкою для завдань, пов'язаних з вводом/виводом. Це ідеально підходить для API та служб автентифікації, де потрібно ефективно обробляти численні одночасні запити без важких обчислень. Node.js має невелику вагу і може бути легко масштабований горизонтально на декілька серверів, що має вирішальне значення для обробки потенційно великої кількості запитів одночасно.

З іншого боку, сервіси обробки замовлень та взаємодії з водієм мають специфічні вимоги:

- надійна обробка транзакцій;
- складна обробка бізнес-логіки;
- робота з кількома потоками одночасно;
- інтеграція із зовнішніми системами;
- масштабованість для обробки великих навантажень у пікові моменти.

Типізована Java, обгорнута в фреймворк Spring, що спрощує Dependency Injection – це чудова технологія, що може задовольнити вказані потреби. Продуктивність Java вигідна для завдань, що вимагають значних ресурсів процесора, та під час обробки даних за складними алгоритмами. Багатопотоковість Java дозволяє максимально ефективно використовувати ресурси серверного часу та скорочує затримку між запитом від клієнта та поверненням результату.

Для комунікації API та сервісів використовується система віддаленого виклику процедур gRPC. Вона дозволяє за допомогою спеціальних файлів, що мають розширення .proto, визначити структуру повідомлень, що передаються між різними застосунками (наприклад, між клієнтом та сервером) та сервіси, що відповідають за обробку цих повідомлень. Після ініціалізації, клієнт може користуватись логікою, що прописана на сервері за допомогою виклику методу, ніби ця логіка присутня безпосередньо в клієнта. При цьому сервер може мати інші технології, іншу мову програмування, бути розташованим у зовсім іншому місці. gRPC є особливо корисним під час передачі потоків повідомлення (які можна передавати як від

клієнта до сервера, так і від сервера до клієнта), чого неможливо добитись із використанням HTTP/S-протоколу.

3.3 Проєктування структури зберігання даних

У розподілених системах CAP-теорема (теорема Брюера) пояснює, що коли мережі стають розподіленими, існує компроміс між узгодженістю та доступністю та можливістю партиціонування даних [10].

Оскільки архітектура серверної частини системи є мікросервісною, потрібно визначити, що є важливішим під час роботи з даними – узгодженість (consistency), доступність (availability) чи стійкість до поділу даних (partition tolerance).

Характер даних та операцій у застосунку є таким, що дані не завжди потребують суворої узгодженості в кожен моменту часу. Наприклад, інформація про доступність водія або статус доставки не обов'язково повинна бути ідеально синхронізована на всіх вузлах в будь-який час. Важливіше, щоб система залишалася доступною і швидко реагувала, навіть якщо деякі дані можуть бути трохи застарілими протягом короткого періоду часу через мережеві розділи або затримки. У багатьох випадках кінцева узгодженість є прийнятним компромісом. Кінцева узгодженість означає, що система гарантує, що всі оновлення будуть поширюватися по системі і зрештою стануть узгодженими. Це дозволяє системі обробляти операції без затримок, навіть при великому навантаженні або часткових збоях мережі, гарантуючи, що водії та клієнти можуть продовжувати взаємодіяти із застосунком без перерв.

Користувацький досвід значно більше залежить від доступності та швидкості реагування системи, ніж від ідеально узгоджених даних. Зважаючи на те, що застосунок може використовуватись у будь-якій країні, важливо також попіклуватись про географічне розподілення та реплікацію даних. Таким чином, доступність та стійкість до поділу даних є більш значущою, ніж узгодженість в нашій системі. За CAP теоремою це AP (availability and partition tolerance) рішення.

NoSQL-системи є гарним рішенням для досягнення максимальної швидкодії з можливістю розподілення даних за рахунок втрати підтримки постійної

консистентності, тобто узгодженості. Зокрема, база MongoDB підтримує зазначені характеристики, дотримуючись AP сектору за теоремою CAP (див. рис. 3.5).

Тому більшість даних в системі вантажних перевезень, що створюється, варто зберігати саме в MongoDB. MongoDB – це документо-орієнтована система керування базами даних з відкритим вихідним кодом, що не потребує опису схеми таблиць, як це відбувається у реляційних базах даних, і зберігає дані в JSON-подібному фораті. MongoDB вміє робити швидкий пошук за текстом, на відміну від переважної більшості SQL-систем. Вона легко масштабується і реплікується.

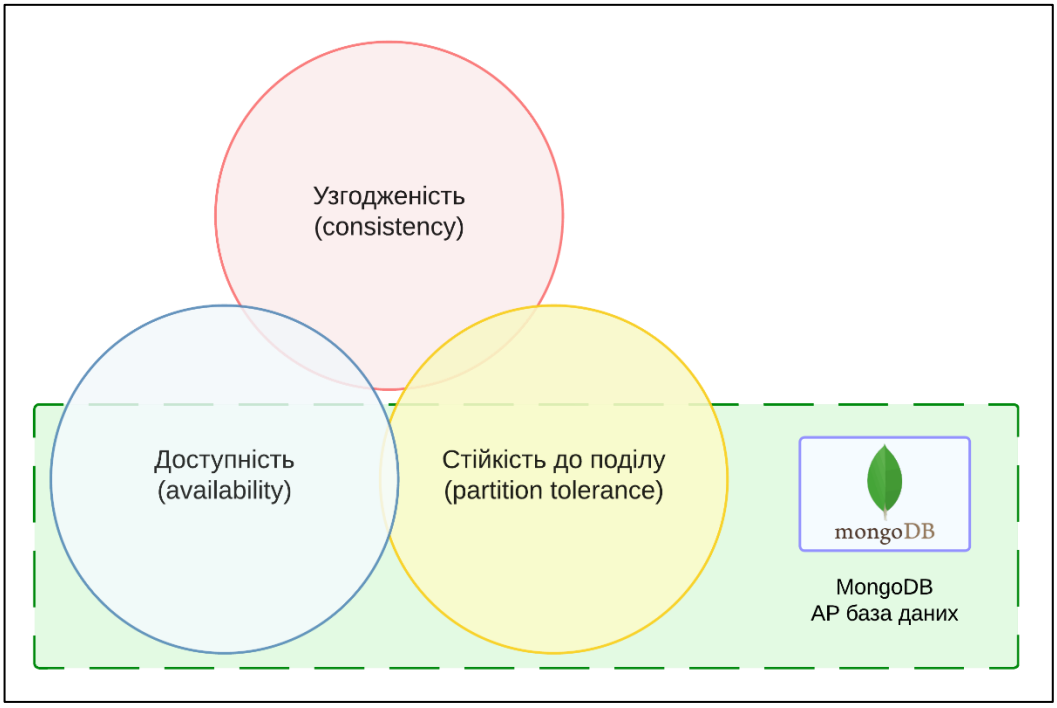


Рисунок 3.5 – Розташування MongoDB відносно CAP-теореми

Auth/User Service, як і сервіси обробки замовлень та обслуговування водіїв, зберігає інформацію про користувачів до MongoDB. Але цей сервіс також використовує Redis для зберігання сесій авторизації користувачів.

Кожен запит, що надсилає користувач, необхідно авторизувати. Для того, щоб не навантажувати сервіс авторизації постійними запитами до бази даних, сесії користувачів можна кешувати. Основна інформація про користувача зберігається до того часу, поки термін дії refresh-токена не спливе. Redis є сховищем кешу і дозволяє швидко повертати значення за поданим ключем. В нашому випадку ключ, за

яким зберігається інформація про користувача в Redis, зберігається в тілі auth-токена.

Також система зберігає дані до Elasticsearch.

За допомогою Logstash, логи та метрики з сервісів зберігаються в Elasticsearch. Потім, за допомогою Kibana ці дані можна візуалізувати, зокрема для відображення статистичної інформації. Також дані, збережені до Elasticsearch, можуть використовуватись адміністраторською серверною частиною.

3.4 Створення UI-версії документації API системи

Для зручної розробки клієнтського застосунку (мобільної чи веб-версій) необхідно надати розробнику детальний опис API серверної частини.

Специфікація OpenAPI (OAS) визначає стандарт, мовно-діагностичний інтерфейс для HTTP API, що дозволяє як людям, так і комп'ютерам відкривати і розуміти можливості сервісу без доступу до вихідного коду, документації або через перевірку мережевого трафіку [11]. Саме специфікацію OpenAPI й обрано в якості основи для побудови документації.

Нижче наведено частину файл openapi.yaml, де заповнюється документація API, що потрібна для створення мобільного клієнту:

```
openapi: 3.1.0
info:
  title: Wide Delivery Mobile Client API
  version: 0.0.1
  description: "The client API for Android app"
  contact:
    name: Stanislav Matsak, Wide Delivery client API developer

servers:
  - url: http://widedelivery.localhost/api
  - url: client-api:3000

/auth/register:
    post:
      description: Register the user.
      tags:
        - auth
      requestBody:
        required: true
        content:
```

```
      'application/json':
        schema:
          $ref: '#/components/schemas/Registration'
responses:
  '201':
    description: User registered successfully.
  '409':
    description: Registration failed. Maybe email was used before
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/HandledError'
  '500':
    description: Internal error.
```

У наведеному вище фрагменті оформлено документацію API для кінцевої точки /auth/register. Описано очікуване тіло запиту, можливо відповіді від сервера та їх коди.

Завдяки утиліті Swagger UI вище описана документація перетворюється в зручну веб-сторінку з навігацією та розділами. Вигляд UI-версії документації зображено на рисунках 3.6, 3.7.

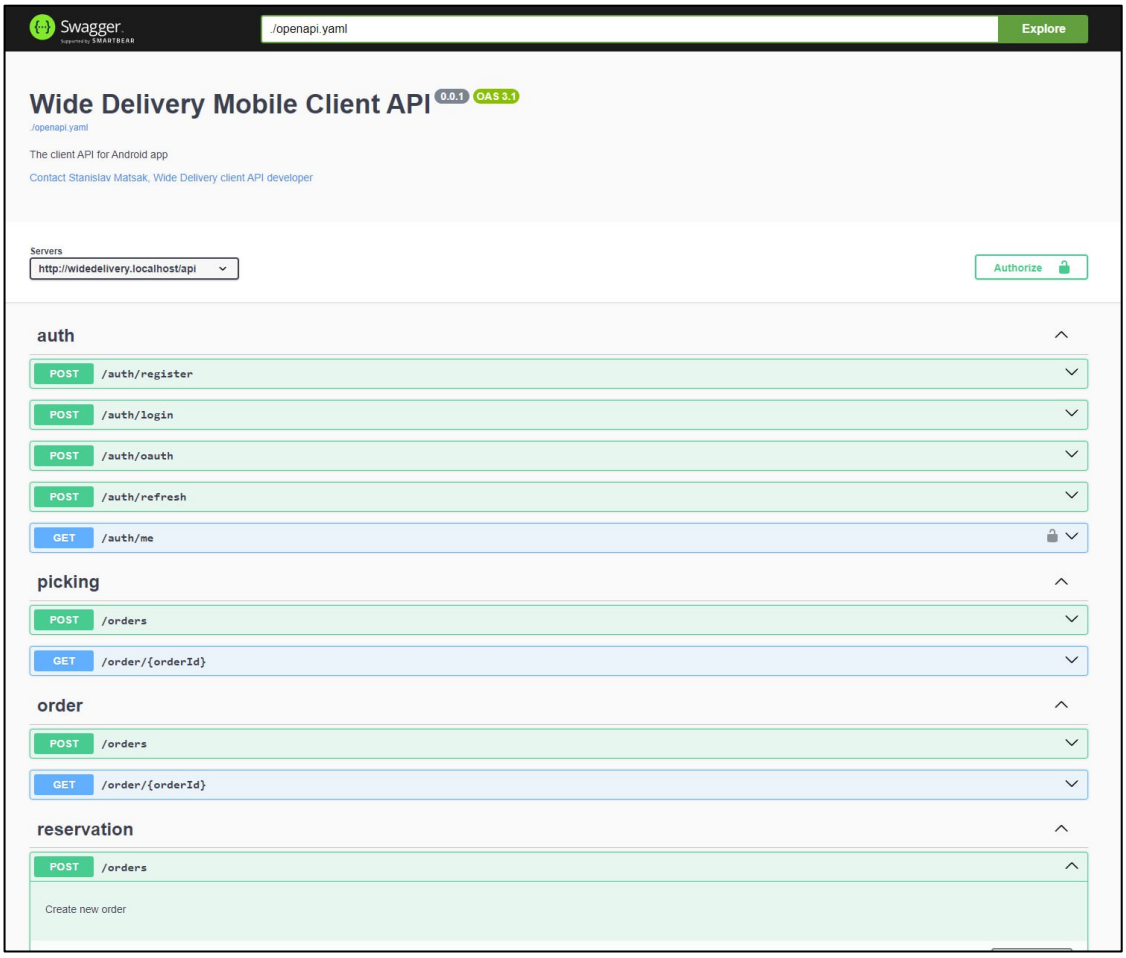


Рисунок 3.6 – Загальний вигляд UI-версії створеної API-документації

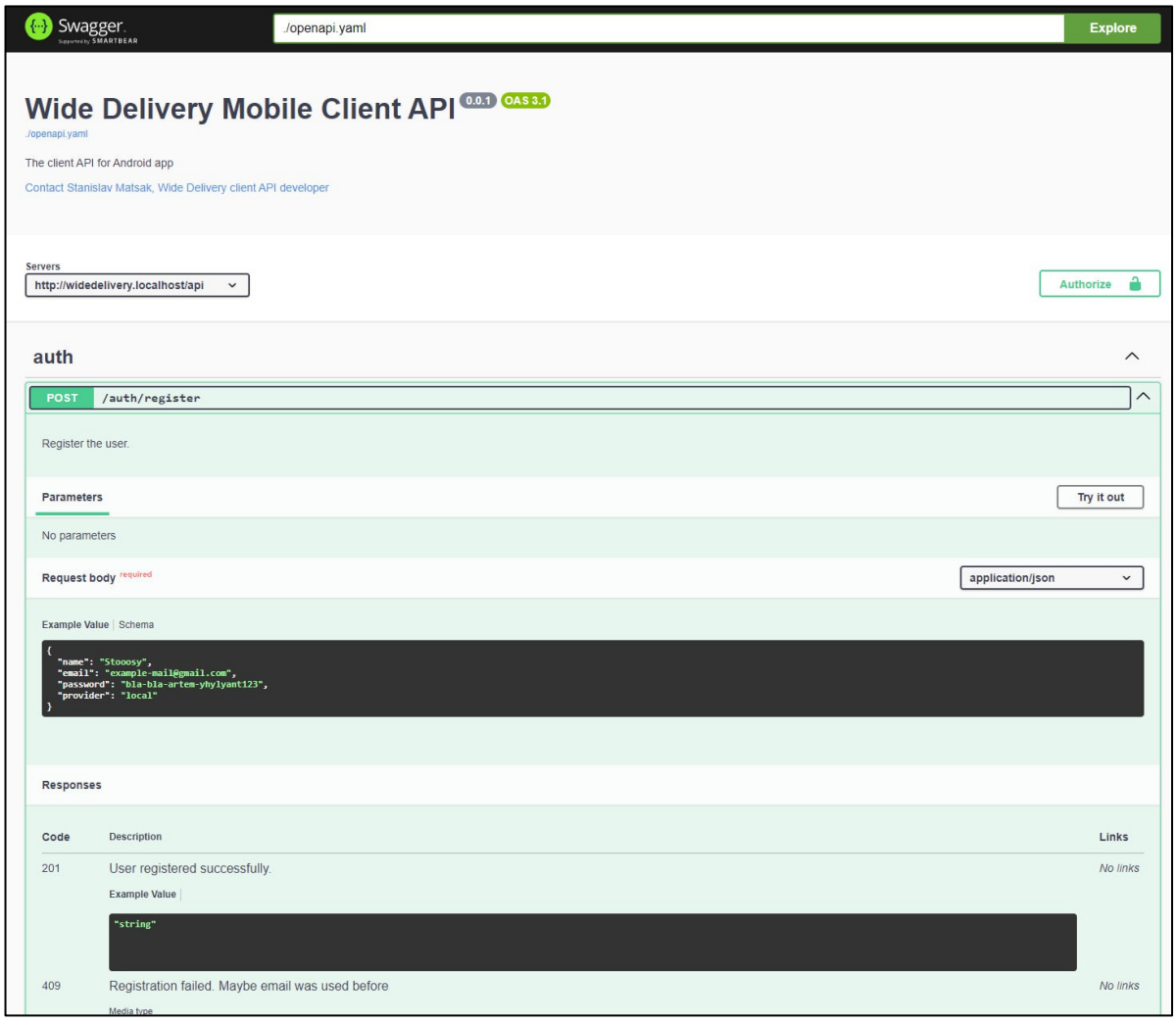


Рисунок 3.7 – Детальний опис кінцевої точки /auth/register у створеній документації

Розробнику мобільного клієнту необхідно лише відкрити цю веб-сторінку в себе на локальній машині для того, щоб мати можливість перегляду документації та надсилання запитів для перевірки кінцевих точок безпосередньо у цьому ж вікні браузеру.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

Предметна область, в межах якої розробляється система вантажних перевезень, не передбачає створення та розробки великої кількості математичних операцій зі складною логікою. Більшість необхідних рішень, зокрема для обчислення тривалості перевезення, дистанції між заданими координатами, визначення найкоротшої відстані, надає Google Maps API. Таке рішення є оптимальним, оскільки ситуації на дорогах є динамічними, і створення, зберігання та оновлення даних щодо стану доріг є надто дорогою та працемісткою задачею. Натомість провайдер, що отримує дані від водіїв безпосередньо під час руху, завжди має актуальну інформацію, попереджує про можливі ризики та пропонує найкращі маршрути в конкретний момент часу.

Потреба у створенні певних алгоритмів все ж існує. Зокрема, необхідно розробити алгоритми для визначення ціни перевезення та розробити загальний алгоритм підбору замовлення для водія, що вказав запланований маршрут поїздки.

4.1 Розробка алгоритму підбору замовлень для водія

Для отримання інформації про очікуваний час поїздки чи виконання замовлення, загальний маршрут за конкретними координатами тощо використовується Google Directions API.

Алгоритм обрахування коефіцієнту релевантності замовлення для водія подано у вигляді формули 4.1.

$$k = 1 - \frac{\Delta_o}{T_d + \Delta_{max}} \tag{4.1}$$

де $\Delta_{max} = \alpha T_d + \beta T_o$ – максимально допустимий час відхилення.

T_d - загальний час, необхідний для проїзду маршруту водія.

T_o - час, необхідний для проїзду маршруту замовлення.

Δ_o- додатковий час, який водій витратить на проїзд до початкової та з кінцевої точки маршруту замовлення.

Оскільки в результаті обчислення коефіцієнт може вийти за рамки від 0 до 1, на нього також накладаються обмеження, що описуються формулою 4.2.

$$k = \max(0, \min(1, k_0)) \tag{4.2}$$

де k_0 – значення, отримане в результаті обчислення коефіцієнта у формулі 4.1.

Ця модель дозволяє розрахувати, наскільки легко маршрут замовлення може бути інтегрований у графік водія без значних втрат часу. Коефіцієнти α та β можуть бути адаптовані для різних типів транспортних засобів або різних умов дорожнього руху, забезпечуючи гнучкість для різних логістичних вимог. В цілому, вони визначаються емпіричним шляхом. Прийнятні результати в ході аналізу результатів алгоритму утворились з коефіцієнтами $\alpha = 0.05$ для часу поїздки водія та $\beta = 0.1$ – для часу перевезення замовлення. Втім у подальшому коефіцієнти можуть змінюватись, в залежності від потреби.

Обрахування Δ_{max} є необхідним для того, щоб динамічно визначати критичну відстань, за якої замовлення буде відхилено автоматично та не буде пропонуватись водієві, оскільки для коротких подорожей недоцільно пропонувати водієві довгі замовлення, ділянки яких можуть повністю збігатись з маршрутом водія, та все ж йому доведеться значно більший час витрачати на виконання замовлення, ніж на заплановану поїздку.

Код створеного алгоритму наведено нижче:

```
public static double calculateOrderRelevanceForDriverTrip(OrderMatching-
Model order, DriverTripDto driverTrip) {
    String orderOrigin = order.getDepartureLongitude() + "," + order.getDe-
partureLatitude();
    String orderDestination = order.getDestinationLongitude() + "," + or-
der.getDestinationLatitude();
```

```
    String driverTripOrigin = driverTrip.getDepartureLongitude() + "," +
driverTrip.getDepartureLatitude();
    String driverTripDestination = driverTrip.getDestinationLongitude() +
"," + driverTrip.getDestinationLatitude();

    try {
      long driverTripTime = RouteService.calculateRouteTime(driverTri-
pOrigin, driverTripDestination, TravelMode.DRIVING);
      long orderNeededTime = RouteService.calculateRouteTime(orderOrigin,
orderDestination, TravelMode.DRIVING);
      long extraTimeToCompleteOrder = RouteService.calculateRoute-
Time(driverTripOrigin, orderOrigin, TravelMode.DRIVING) +
          RouteService.calculateRouteTime(orderDestination, driverTripDes-
tination, TravelMode.DRIVING);

      double deltaMax = 0.05 * driverTripTime + 0.1 * orderNeededTime;
      return Math.max(0, Math.min(1, 1 - extraTimeToCompleteOrder /
(driverTripTime + deltaMax)));
    } catch (Exception e) {
      e.printStackTrace();
      Thread.currentThread().interrupt();
      throw new RuntimeException("Cannot calculate time between points");
    }
  }
```

4.2 Розробка системи авторизації та аутентифікації користувача

Система авторизації в системі вантажних перевезень передбачає створення JWT-токенів під час аутентифікації користувача та подальше надсилання створеного токену доступу з кожним запитом до сервера. Цей токен доступу має коротку тривалість життя для того щоб зловмисник, у разі компрометації токену, мав можливість виконувати операції лише обмежений час. Було визначено, що 10 хвилин – оптимальна тривалість життя цього токену. У парі з ним також надсилається refresh-токен, за допомогою якого можна оновити токен доступу без повторної аутентифікації користувачем. Це дозволяє значно рідше повертати користувача до сторінки входу, що приємно відображається на взаємодії користувачів із сервісом. Тривалість життя цього токену значно вища, і досягає 7 діб.

Інформація про користувача після успішної аутентифікації зберігається в розподіленому сховищі типу “ключ-значення” Redis. Оскільки користувач надсилає JWT-токен з кожним запитом, існує необхідність перевірки його прав доступу та отримання об’єкту користувача для більшості запитів. Для того, щоб збільшити пропускну здібність серверу та усунути потребу в додатковому запиті до бази даних

для кожного HTTP-запиту, основні дані користувача зберігаються в сесії Redis. Сховище Redis є надшвидким у випадку, коли ключ має примітивний тип даних та не є складним об'єктом. Тому під час створення JWT-токену генерується ідентифікатор, за яким у Redis зберігається інформація про користувача. Потім цей ідентифікатор зберігається в тілі JWT-токену. Це додає ще один рівень безпеки, оскільки зловмисник, що зміг отримати JWT-токен не буде знати будь-якої інформації про користувача (його e-mail, ім'я, ідентифікатор в базі даних тощо), як це зазвичай відбувається при зберіганні такої інформації безпосередньо в тілі токену.

Метод для створення токенів доступу та відновлення токену доступу наведено в прикладі коду нижче:

```
export const signTokens = async (user: User) => {

  const randomUUID = uuidv4();
  // 1. Create Session
  await redisClient.set(randomUUID, JSON.stringify(user), {
    EX: customConfig.redisCacheExpiresIn * 60,
  });

  // 2. Create Access and Refresh tokens
  const access_token = signJwt({ sub: randomUUID },
'accessTokenPrivateKey', {
    expiresIn: `${customConfig.accessTokenExpiresIn}m`,
  });
  const refresh_token = signJwt({ sub: randomUUID },
'refreshTokenPrivateKey', {
    expiresIn: `${customConfig.refreshTokenExpiresIn}m`,
  });
  return { access_token, refresh_token };
};

export const signJwt = (
  payload: Object,
  key: 'accessTokenPrivateKey' | 'refreshTokenPrivateKey',
  options: SignOptions = {}
) => {
  const privateKey = Buffer.from(customConfig[key],
'base64').toString('ascii');
  return jwt.sign(payload, privateKey, {
    ...(options && options),
    algorithm: 'RS256',
  });
};
```

У сховищі Redis зберігаються ідентифікатори в якості ключа та об’єкти з даними користувачів у якості значення (див. рис. 4.1).

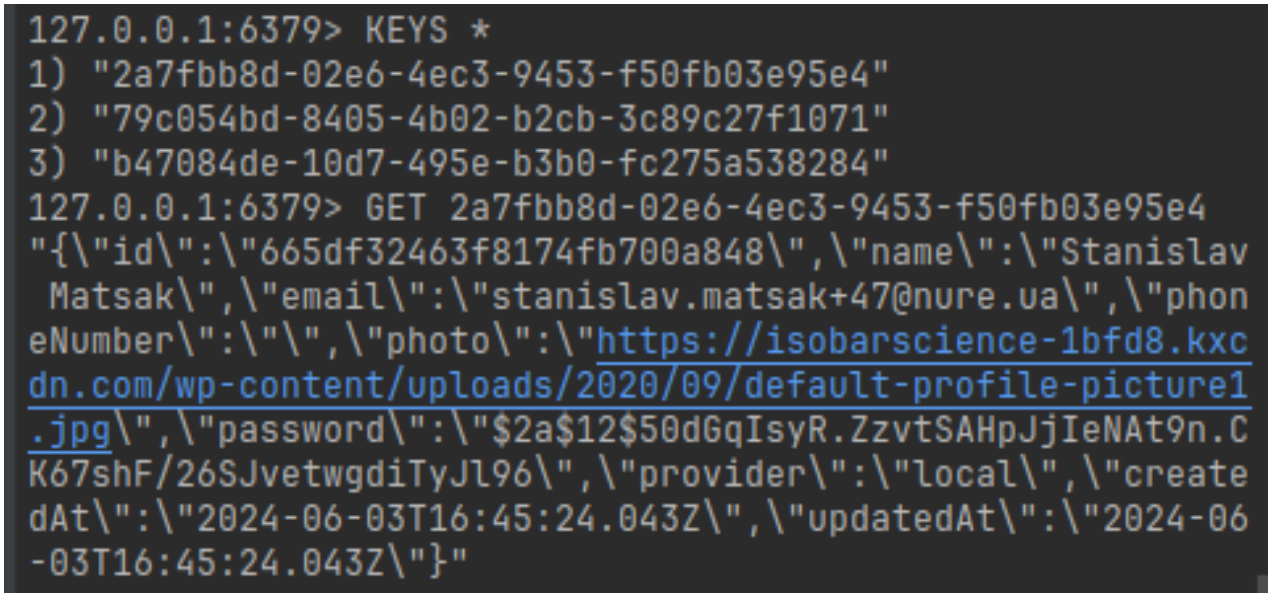


Рисунок 4.1 – Вигляд ідентифікаторів та об’єктів з даними користувача в сховищі Redis

Згодом сховище видаляє кешовані дані та за потреби знову отримує об’єкт з даними користувача із основної бази даних.

4.3 Розробка системи логування

Об’ємна система, що містить багато компонентів (і такою можна вважати мікросервісну архітектуру системи вантажних перевезень) потребує постійного моніторингу стану її компонентів, слідкування за метриками в режимі реального часу та їх збір з метою аналізу тенденцій за статистичними показниками. Для того, щоб адміністратори та модератори мали змогу слідкувати за потрібною їм інформацією, необхідно впровадити систему логування та зберігання логів. Для цього було використано стек технологій ELK (Elasticsearch, Logstash, Kibana).

Спочатку в сервісі необхідно налаштувати логер – компонент, що збирає інформацію під час виконання програми та передає її до консолі, файлу тощо. У сервісі client-api, що приймає запити користувача та комунікує з сервісами, що

безпосередньо зберігають, оброблюють та повертають дані в потрібному для API вигляді, конфігурація логера виглядає як у прикладі коду, наведеному нижче:

```
import winston, {Logger} from "winston";
import ecsFormat from "@elastic/ecs-winston-format";
import customConfig from "../config/default";

const logFilesPath = customConfig.logPath;

const httpReqLogger: Logger = winston.createLogger({
  level: 'debug',
  format: ecsFormat({ convertReqRes: true }),
  transports: [
    new winston.transports.File({
      filename: logFilesPath,
      level: 'debug'
    })
  ]
});

export default httpReqLogger;
```

Таким чином, усі зібрані логи записують в файл, що зберігається в контейнері Docker.

Logstash є конвеєром обробки даних, він лише отримує певні дані з input-джерел, оброблює їх за допомогою фільтрів та передає на вихід у певне сховище чи інший обробник. Таким обробником є Elasticsearch – розподілену потужну аналітичну пошукову систему. В якості джерела даних використовується Filebeat, оскільки логи зберігаються у файли і саме ці дані необхідно передати в конвеєр для обробки. Конфігурація конвеєру має назву пайплайн та оголошується у файлі logstash.conf в контейнері, де працює Logstash. Вигляд пайплайну обробки логів для системи вантажних перевезень у вигляді коду подано нижче:

```
input {
  beats {
    port => "5044"
  }
}
filter {
  json {
    source => "message"
    target => "parsed_json"
    remove_field => ["message"]
```

```
  }
  mutate {
   rename => { "[parsed_json][@timestamp]" => "@timestamp" }
   rename => { "[parsed_json][client]" => "client" }
   rename => { "[parsed_json][ecs.version]" => "ecs.version" }
   rename => { "[parsed_json][http]" => "http" }
   rename => { "[parsed_json][log.level]" => "log.level" }
   rename => { "[parsed_json][url]" => "url" }
   rename => { "[parsed_json][user_agent]" => "user_agent" }
   rename => { "[parsed_json][message]" => "message" }
   remove_field => ["parsed_json"]
  }
  geoip {
    source => "[client][address]"
     target => "[ip_decoded]"
  }
}
output {
  stdout { codec => rubydebug }
   elasticsearch {
    hosts => ["elastic-host"]
    index => "filebeat-logs-%{+YYYY.MM.dd}"
    api_key => "private-api-key"
   }
}
```

Серед фільтрів, окрім розбивання логу на окремі поля, також використовується geoip-фільтр. Він дозволяє за поданою в логах ip-адресою знайти фізичне місце (країну, місто) та зберігає цю інформацію в лозі. Цю інформацію, як і набір інших полів, що зберігаються в логах, можна візуалізувати за допомогою Kibana (див. рис. 4.2).

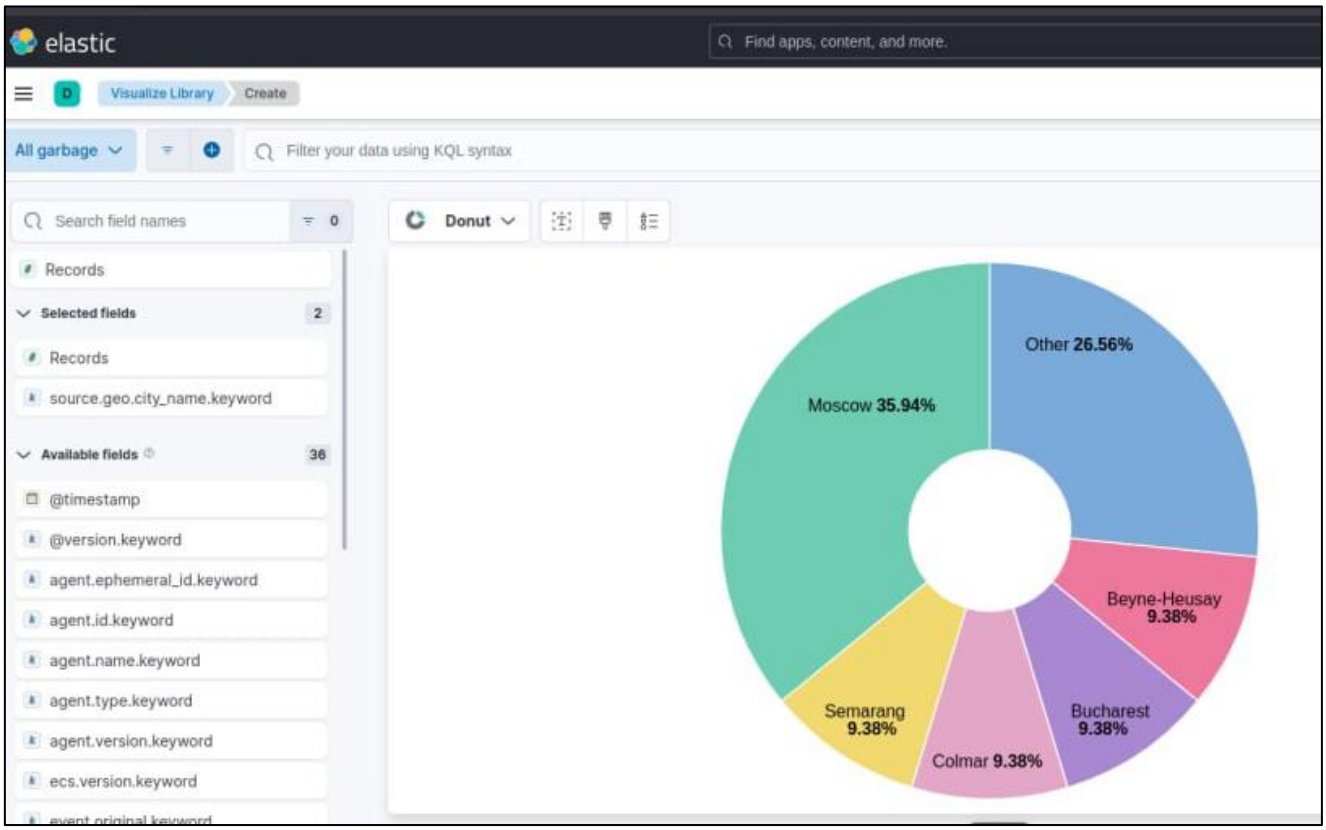


Рисунок 4.2 – Візуалізація зібраної за допомогою логів інформації щодо географічного місцезнаходження клієнта

У подальшому ця інформація може бути використана для аналізу статистичних даних та відображення закономірностей поведінки користувачів в залежності від певних параметрів.

4.3 Взаємодія мікросервісів з центральним API

Головний сервіс-контролер API комунікує з іншими сервісами за допомогою протоколу protobuf. Створюється текстовий файл з описом повідомлень, що будуть передаватись та сервісами, що оброблюватимуть повідомлення, у форматі .proto. Після компіляції та запуску серверу gRPC, можна використовувати обробники, що написані навіть іншою мовою програмування, просто викликаючи їх, ніби вони існують безпосередньо в сервісі API.

Вміст файлу з розширенням .proto, що описує сервіс обробки замовлень, наведено нижче.

```
syntax = "proto3";
option java_multiple_files = true;
option java_package = "com.widedelivery.order.service";
option java_outer_classname = "OrderProto";
package com.widedelivery.order.service;
import "com.widedelivery.order/rpc_create_order.proto";
import "com.widedelivery.order/rpc_get_matched_orders.proto";
import "com.widedelivery.order/rpc_add_driver_to_matching.proto";
import "com.widedelivery.order/rpc_remove_driver_from_matching.proto";
import "com.widedelivery.order/rpc_link_driver_with_order.proto";
import "com.widedelivery.order/order.proto";

service OrderService {
  rpc CreateOrder(com.widedelivery.order.proto.CreateOrderInput) returns
(com.widedelivery.order.proto.CreateOrderResponse) {}
  rpc GetOrder(GetOrderInput) returns
(com.widedelivery.order.proto.OrderResponse) {}
  rpc GetOrders(GetOrdersRequest) returns (GetOrdersResponse) {}
  rpc SearchOrders(SearchOrdersRequest) returns (SearchOrdersResponse) {}
  rpc
AddDriverForMatching(com.widedelivery.order.proto.AddDriverToMatchingInput)
returns (com.widedelivery.order.proto.GenericResponse) {}
  rpc
RemoveDriverFromMatching(com.widedelivery.order.proto.RemoveDriverFromMatch
ingInput) returns (com.widedelivery.order.proto.GenericResponse) {}
  rpc GetMatchedOrders(com.widedelivery.order.proto.GetMatchedOrdersInput)
returns (stream com.widedelivery.order.proto.GetMatchedOrdersOutput) {}
  rpc
LinkDriverWithOrder(com.widedelivery.order.proto.LinkDriverWithOrderInput)
returns (com.widedelivery.order.proto.OrderResponse) {}
}

message GetOrderInput {
  string order_id = 1;
}
message GetOrdersRequest {
  int32 page_number = 1;
  int32 page_size = 2;
}

message GetOrdersResponse {
  repeated proto.Order orders = 1;
  int32 total_pages = 2;
  int32 current_page = 3;
}

message SearchOrdersRequest {
  map<string, string> search_params = 1;
  int32 page_number = 2;
  int32 page_size = 3;
}

message SearchOrdersResponse {
  repeated proto.Order orders = 1;
  int32 total_pages = 2;
  int32 current_page = 3;
}
```

У головному API-контролері, що працює на Node.JS сервіс grpc, написаний на Java Spring, викликається як звичайна функція:

```
public static getOrderById(orderId: string): Promise<OrderDto> {
        return new Promise((resolve, reject) => {
            orderService.getOrder({
                order_id: orderId
            }, (err: any, result: any) => {
                if (err) {
                    console.error(err);
                    throw new Error(`Cannot get order by id ${orderId}`);
                } else {
                    console.log(result);
                    resolve(OrderDto.parseFromGrpcResponse(result.order));
                }
            })
        });
 }
```

Сервіс обробки замовлення, в свою чергу, оброблює цей запит та повертає реузльтат – замовлення, за допомогою протоколу protobuf:

```
@Override
public void getOrder(GetOrderInput request, StreamObserver<OrderResponse>
responseObserver) {
        String id = request.getOrderId();

        OrderModel order = orderService.getOrder(id);

        responseObserver.onNext(OrderResponse
                .newBuilder()
                .setOrder(OrderMapper.toGrpcModel(order))
                .build());

        responseObserver.onCompleted();
 }
```

Таким чином головне API виступає в ролі шлюзу, приймаючи HTTP/S запити від користувачів мобільного застосунку, надсилаючи та збираючи дані з інших сервісів за допомогою протоколу protobuf із використанням системи виклику віддалених процедур gRPC, та повертаючи результат, обгорнений у зручну оболонку назад на пристрій користувача, де мобільний застосунок відображає ці дані за допомогою приємного графічного інтерфейсу.

4.4 Розробка сервісу сповіщень користувачів

Одним із завдань API клієнтської частини системи вантажних перевезень є також розсилання сповіщень про реєстрацію та зміни статусу замовлення клієнту чи водієві. Реалізовано це за допомогою розсилки електронних листів, що працює з головного API-сервісу, написаного на Node.js. Для кожного з електронних листів створюється свій шаблон, з єдиним оформленням, стилями та даними, що потрібно передати для заповнення шаблону. Для цього використано бібліотеку Pug. Генерація готового html-коду, що вбудовується в електронний лист відбувається завдяки внесенню вхідних даних до шаблону листа. Згенерований лист в більшості основних клієнтів електронної пошти підтримує основний функціонал html-розмітки та css-стилів. Отриманий лист візуально приємний та інформативний (див. рис. 4.3).

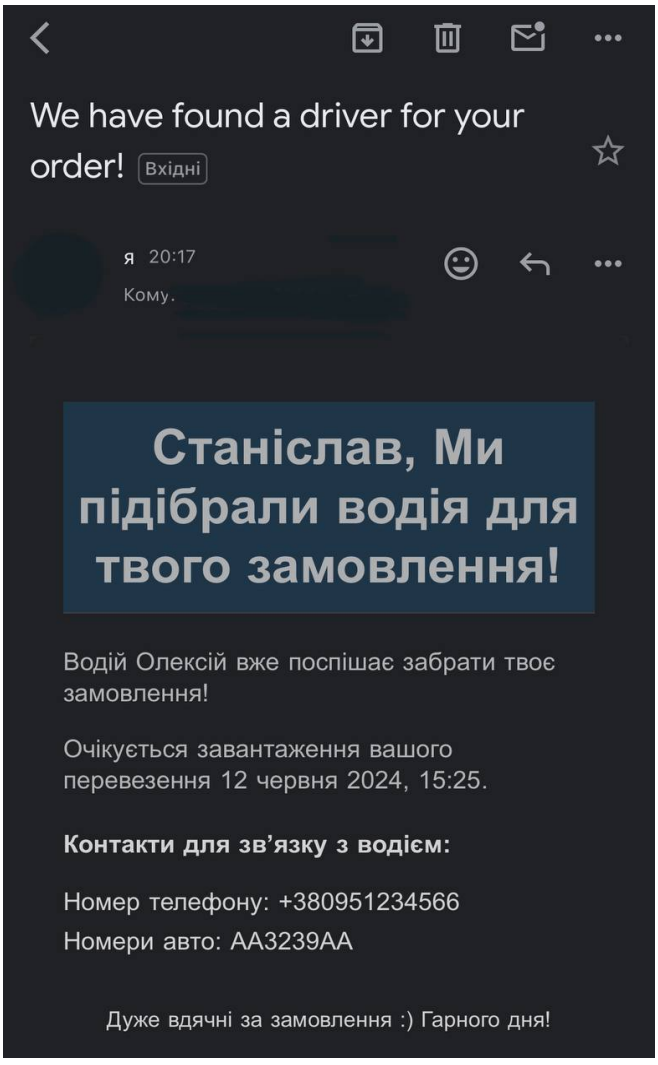


Рисунок 4.3 - Вигляд електронного листа, що надсилає сервіс сповіщень, отриманого за допомогою мобільного застосунку Gmail

Розмітку відповідного шаблону для повідомлення про успішний підбір водія до створеного користувачем замовлення подано нижче у вигляді вмісту файлу з розширенням .pug.

```
doctype html
html
    head
        title= title
    body(style='background-color: #f9f9f9; font-family: Arial, sans-serif;
color: #333;')
        .container(style='max-width: 600px; margin: auto; padding: 20px;
background-color: #fff; border-radius: 8px; box-shadow: 0 2px 4px rgba(0,
0, 0, 0.1);')
            h1(style='background-color: #cce5ff; color: #5d5d5d; border-
bottom: 1px solid #ddd; padding-bottom: 10px; padding-top: 10px; text-
align: center;')
                | #{user_name}, Ми підібрали водія для твого замовлення!
            p(style='color: #555;')
                | Водій #{driver_name} вже поспішає забрати твоє замов-
лення!
            p(style='color: #555;')
                | Очікується завантаження вашого перевезення #{order_date}.
            .contact-info(style='margin-top: 20px;')
                p(style='margin-bottom: 5px; font-weight: bold;')
                    | Контакти для зв'язку з водієм:
                ul(style='list-style-type: none; padding: 0;')
                    li(style='margin-bottom: 5px;')
                        | Номер телефону: #{driver_phone}
                    li(style='margin-bottom: 5px;')
                        | Номери авто: #{truck_plate}
            p(style='margin-top: 20px; font-size: 14px; text-align: center;
background-color: #ffffff; padding: 10px; border-radius: 4px; box-shadow: 0
1px 3px rgba(0, 0, 0, 0.1);')
                | Дуже вдячні за замовлення :) Гарного дня!
```

Сервіс сповіщень є зручним способом проінформувати користувача про важливі зміни в статусах його замовлень або ж про можливі знижки, оновлення, спеціальні пропозиції тощо.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмного забезпечення відіграє критичну роль у розробці якісних додатків, забезпечуючи виявлення та виправлення помилок перед релізом продукту. Воно допомагає підтримувати стабільність системи, забезпечує відповідність до вимог користувача, а також знижує вартість подальшої підтримки шляхом мінімізації кількості несправностей. Тестування також забезпечує високий рівень користувацького досвіду, що є надзвичайно важливим у конкурентному середовищі мобільних та веб-додатків.

Для забезпечення всебічної перевірки програмного забезпечення розробленого додатку було використано кілька методів тестування, основними з яких є юніт-тестування (перевірка окремих модулів на наявність помилок в алгоритмах і логіці обробки даних) та інтеграційне тестування (перевірка взаємодії між різними модулями і сервісами).

Основними сервісами, що потребують максимальної уваги до тестування є сервіси авторизації – для забезпечення необхідного рівня безпеки застосунку, та сервіс обробки замовлень. Клієнтське API є шлюзом та не містить додаткової логіки, сервіс взаємодії з водієм теж містить відносно малий об’єм логічних операцій.

5.1 Модульне тестування

Модульне тестування (Unit testing) – метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. У процедурному програмуванні модулем вважають окрему функцію або процедуру. В об’єктно-орієнтованому програмуванні – метод [12].

Сервіс обробки замовлень містить створений власноруч алгоритм, що має бути добре протестованим для надійної роботи системи - алгоритм підбору замовлень.

Значення отриманого коефіцієнту має бути рівним 1 для ідентичних маршрутів, дорівнювати 0 для абсолютно різних маршрутів, та не виходити за діапазон від 0 до 1 для будь-яких інших обчислень.

Для дотримання цих 3 умов створено 3 модульні тести, що перевіряють виконання алгоритмом цих вимог. Сервіс, що викликає Google Directions API було загорнути в «мок» – обгортку навколо модуля, що дозволяє підміняти функціональність під час виконання коду. Це зроблено з метою заощадження коштів, що не будуть витрачатись, оскільки сервіс не використовує справжнє API, а також з метою дотримання концепції модульних тестів, що перевіряють правильність роботи конкретного модуля системи, і не торкаються перевірки функціональності інших модулів, роблячи припущення, що всі вони працюють правильно.

Таким чином, створено три основні тести для цього сервісу.

Перший тест перевіряє отримання значення 1 при передачі в якості аргументів ідентичних маршрутів замовлення та водія:

```
@Test
  public void
testCalculateOrderRelevanceForDriverTripReturnHighestCoefficientForTheSameR
outes() throws Exception {
    //mock
    RouteService routeService = Mockito.mock(RouteService.class);
    GeoLocationUtils.setRouteService(routeService);

    OrderMatchingModel order = new OrderMatchingModel();
    order.setDepartureLatitude("46.3912047");
    order.setDepartureLongitude("22.2593465");
    order.setDestinationLatitude("51.9068063");
    order.setDestinationLongitude("38.0494896");
    DriverTripDto driverTrip = new DriverTripDto();
    driverTrip.setDepartureLatitude("46.3912047");
    driverTrip.setDepartureLongitude("22.2593465");
    driverTrip.setDestinationLatitude("51.9068063");
    driverTrip.setDestinationLongitude("38.0494896");

    when(routeService.calculateRouteTime("46.3912047,22.2593465",
"51.9068063,38.0494896", TravelMode.DRIVING)).thenReturn(1000L);

    // when
    double coeff =
GeoLocationUtils.calculateOrderRelevanceForDriverTrip(order, driverTrip);

    // then
    assertEquals(1, coeff); }
```

Другий тест перевіряє отримання 0 при передачі в якості аргументів кардинально різних маршрутів, що віддалені одне від одного та потребують значного додаткового часу для виконання від водія:

```
@Test
  public void
testCalculateOrderRelevanceForDriverTripReturnZeroForVeryDifferentRoutes()
throws Exception {
    // mock
    RouteService routeService = Mockito.mock(RouteService.class);
    GeoLocationUtils.setRouteService(routeService);

    OrderMatchingModel order = new OrderMatchingModel();
    order.setDepartureLatitude("21.3912047");
    order.setDepartureLongitude("10.2593465");
    order.setDestinationLatitude("23.1122334");
    order.setDestinationLongitude("12.0123456");

    DriverTripDto driverTrip = new DriverTripDto();
    driverTrip.setDepartureLatitude("46.3912047");
    driverTrip.setDepartureLongitude("22.2593465");
    driverTrip.setDestinationLatitude("51.9068063");
    driverTrip.setDestinationLongitude("38.0494896");

    when(routeService.calculateRouteTime("46.3912047,22.2593465",
"51.9068063,38.0494896", TravelMode.DRIVING)).thenReturn(2000L);
    when(routeService.calculateRouteTime("21.3912047,10.2593465",
"23.1122334,12.0123456", TravelMode.DRIVING)).thenReturn(5000L);

    when(routeService.calculateRouteTime("46.3912047,22.2593465",
"21.3912047,10.2593465", TravelMode.DRIVING)).thenReturn(6500L);
    when(routeService.calculateRouteTime("23.1122334,12.0123456",
"51.9068063,38.0494896", TravelMode.DRIVING)).thenReturn(90000L);

    // when
    double coeff =
GeoLocationUtils.calculateOrderRelevanceForDriverTrip(order, driverTrip);

    // then
    assertEquals(0, coeff);
  }
```

Третій тест перевіряє отримання будь-яких значень від 0 до 1 впродовж 100000 ітерацій виклику функції обрахування коефіцієнту релевантності замовлення для водія з випадково обраними координатами початків та кінців маршрутів:

```
@Test
  public void
testCalculateOrderRelevanceForDriverTripNeverMoreThanOneOrLessThanZero()
throws Exception {
    // mock
    RouteService routeService = Mockito.mock(RouteService.class);
    GeoLocationUtils.setRouteService(routeService);
    GeoLocationUtils.setMIN_LATITUDE(-180);
    GeoLocationUtils.setMAX_LATITUDE(180);
    GeoLocationUtils.setMIN_LONGITUDE(-90);
    GeoLocationUtils.setMAX_LONGITUDE(90);

    when(routeService.calculateRouteTime(anyString(), anyString(),
eq(TravelMode.DRIVING))).thenReturn(new Random().nextLong());
    for (int i = 0; i < 100_000; i++) {
      OrderMatchingModel order = new OrderMatchingModel();
      order.setDepartureLatitude(String.format("%.7f",
GeoLocationUtils.generateRandomGeoLocationPoint().getLatitude()));
      order.setDepartureLongitude(String.format("%.7f",
GeoLocationUtils.generateRandomGeoLocationPoint().getLongitude()));
      order.setDestinationLatitude(String.format("%.7f",
GeoLocationUtils.generateRandomGeoLocationPoint().getLatitude()));
      order.setDestinationLongitude(String.format("%.7f",
GeoLocationUtils.generateRandomGeoLocationPoint().getLongitude()));

      DriverTripDto driverTrip = new DriverTripDto();
      driverTrip.setDepartureLatitude(String.format("%.7f",
GeoLocationUtils.generateRandomGeoLocationPoint().getLatitude()));
      driverTrip.setDepartureLongitude(String.format("%.7f",
GeoLocationUtils.generateRandomGeoLocationPoint().getLongitude()));
      driverTrip.setDestinationLatitude(String.format("%.7f",
GeoLocationUtils.generateRandomGeoLocationPoint().getLatitude()));
      driverTrip.setDestinationLongitude(String.format("%.7f",
GeoLocationUtils.generateRandomGeoLocationPoint().getLatitude()));

      // when
      double coeff =
GeoLocationUtils.calculateOrderRelevanceForDriverTrip(order, driverTrip);

      // then
      assertTrue(coeff >= 0 && coeff <= 1);

      // log to check correctenss of input points
      if (i % 1000 == 0) {
        System.out.println("coeff = " + coeff + " order from: " +
order.getDepartureLatitude() + "," +
            order.getDepartureLongitude() + " to: " +
order.getDestinationLatitude() + ","
            + order.getDestinationLongitude());
      }
    }
  }
```

Таким чином основну логіку серверної частини було покрито модульними тестами.

5.2 Інтеграційне тестування

Для тестування клієнтського API системи вантажних перевезень було створено інтеграційні тести, що перевіряють можливість виклику кінцевих точок та перевіряють отримані результати. В процесі створення також було «замоковано» виклики gRPC, тож клієнтське API перевіряє лише правильність роботи власних модулів. Як і прийнято під час створення тестів, перевіряються позитивні та негативні сценарії.

Код інтеграційного тесту успішного сценарію реєстрації користувача наведено нижче.

```
it('should register a user successfully', async () => {
    const userData: SignUpUserInput = {
      name: 'Test User',
      email: 'test@example.com',
      password: 'password123',
      phoneNumber: '+1234567890',
      provider: 'LOCAL'
    };

    const expectedResponse: SignUpUserResponse = {
      user: { id: '1', name: 'Test User', email: 'test@example.com', role:
'USER', phoneNumber: '+1234567890', provider: 'LOCAL' }
    };

    signUpUserStub.callsArgWith(1, null, expectedResponse);

    // Act
    const response = await request(app as Application)
      .post('/auth/register')
      .send(userData);

    // Assert
    expect(response.status).toBe(201);

    expect(response.body).toHaveProperty('id');
    expect(response.body.id).toEqual(expect.any(String));

    expect(response.body).toHaveProperty('email', userData.email);
    expect(response.body).toHaveProperty('role', 'USER');

    expect(response.body).not.toHaveProperty('password');

    expect(signUpUserStub.calledOnce).toBe(true);
  });
```

Такий тест перевіряє отриманий статус виконання некоректного запиту, перевіряє наявність додаткових полів, що пояснюють причину виникнення помилки, а також перевіряє що сервіс gRPC було викликано лише один раз.

Система готова до використання і оновлення в коді можна публікувати тоді, коли результати виконання тестів є успішними (див. рис. 5.1).

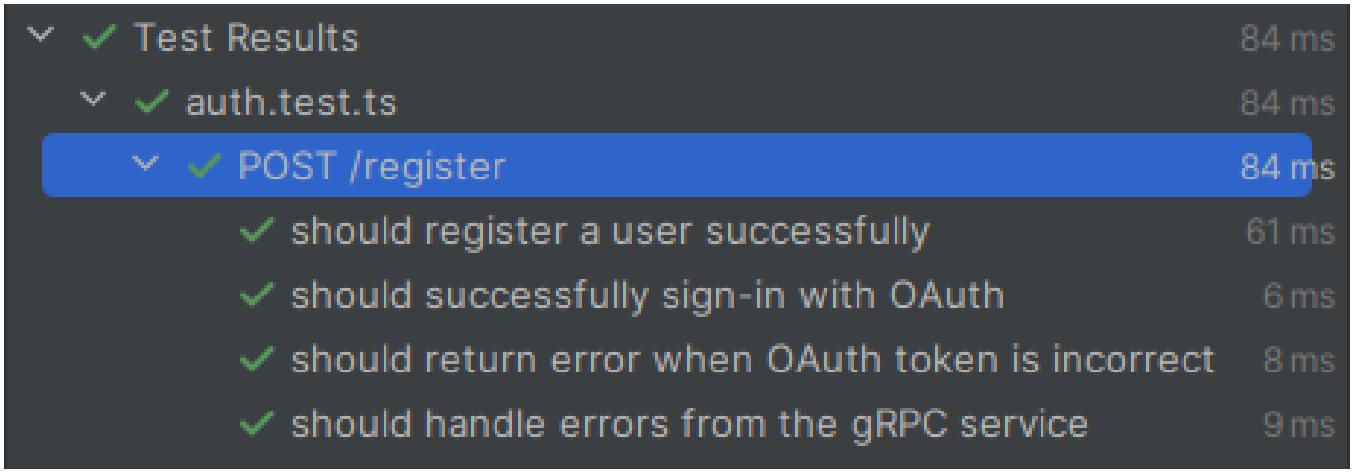


Рисунок 5.1 – Успішне виконання інтеграційних тестів клієнтського API для кінцевої точки реєстрації користувача

Для мокування gRPC-сервісу використано функції, що виконуються перед та після кожного тесту.

```
describe('POST /register', () => {

  let signUpUserStub: sinon.SinonStub;

  beforeEach(() => {
    // Create a stub for the gRPC service call
    signUpUserStub = sinon.stub(authService, 'signUpUser');
  });


  afterEach(() => {
    // Restore the original function after each test
    signUpUserStub.restore();
  });
   …
  }
```

Інтеграційні тести дозволили автоматизовано перевіряти очікувані відповіді чи помилки від серверу.

5.3 Мануальне тестування кінцевих точок API

Модульне та інтеграційне тестування є важливими етапами під час розробки, дозволяють швидше та безпечніше вносити зміни до коду та перевіряти правильність роботи системи загалом. Втім, під час розробки зручно використовувати також платформи для мануального виклику кінцевих точок та перевірки правильності отриманих результатів. Це також дозволяє розробникам клієнтських застосунків швидше познайомитись з API, впевнитись у його коректній поведінці чи протестувати різноманітні сценарії. Тому, для мануального тестування кінцевих точок було створено колекцію в Postman – утиліті для тестування API.

За допомогою створеної колекції можна перевірити що кінцева точка повертає очікуваний результат у відповідь на виклик її з коректними параметрами запиту (див. рис. 5.2).

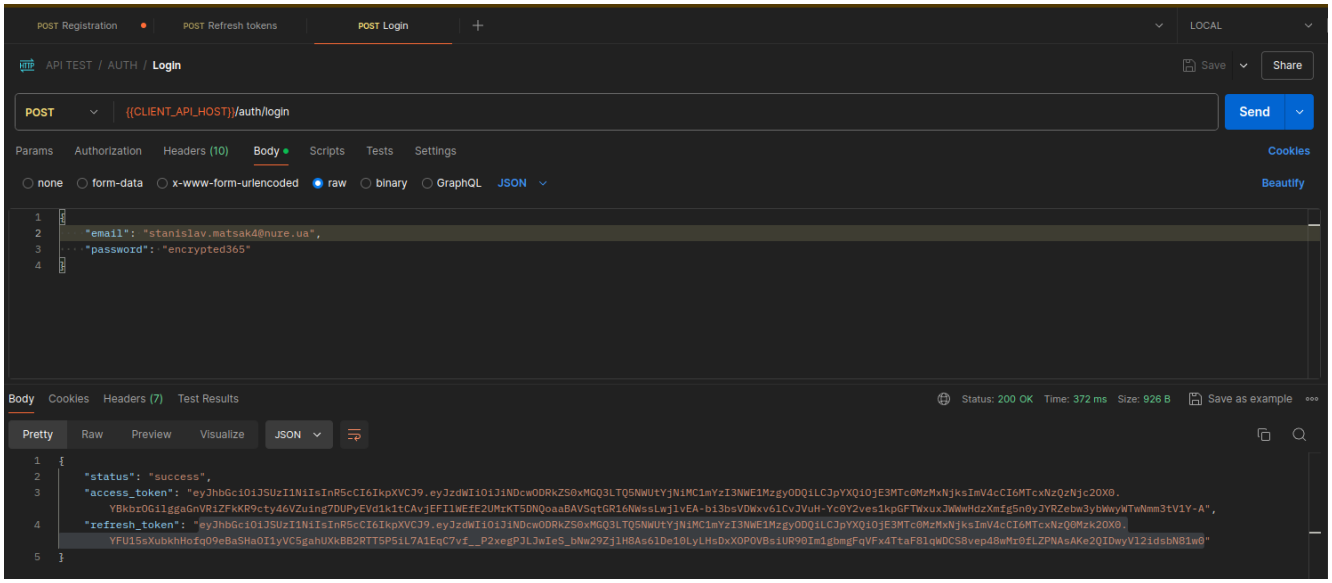


Рисунок 5.2 – Результат виклику кінцевої точки API входу в застосунок із коректними полями тіла запиту

Також можна перевірити, як поводиться система, коли її кінцеві точки намагаються викликати без авторизації чи з протермінованим токеном доступу (див. рис. 5.3).

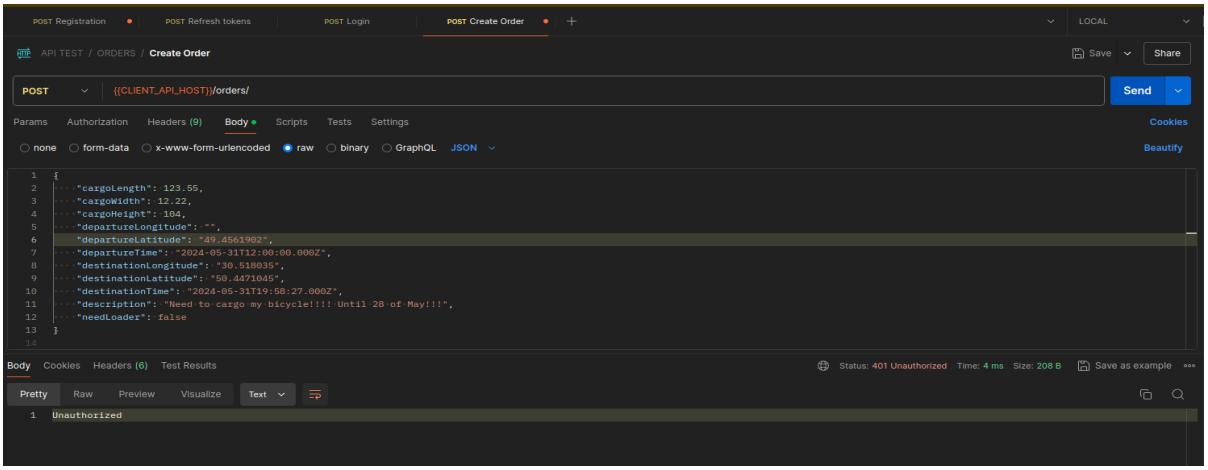


Рисунок 5.3 – Повернення помилки через відсутність токену доступу в заголовках запиту

Отже, можна ретельніше перевірити статуси помилок та тіла відповідей, що надсилає сервер у відповідь на помилкові запити користувача чи помилки роботи сервера. Це корисно як для розробників чи тестувальників серверної частини, так і для розробників мобільного клієнта.

Відсутність ідентифікаторів користувачів чи іншої інформації, що може бути використана зловмисниками для спроб атак на систему чи безпосередньо користувачів є важливою складовою безпеки серверного рішення. Оскільки необхідна інформація для авторизації користувача вже міститься в JWT-токені, можна прибрати звичні для API ідентифікатори на кшталт /users/:id для отримання інформації про себе (яку, наприклад, можна переглянути в меню налаштувань). Такі кінцеві точки теж легко можна протестувати в Postman завдяки можливості вказувати різні типи токенів, що передаються разом із запитом (див. рис. 5.4.).

Postman також дозволяє протестувати gRPC-сервіси, створити кілька прикладів виклику кінцевої точки з різними аргументами (параметрами, полями тіла запиту, заголовками тощо), що особливо зручно для розробників, що в подальшому використовують створене API. Створення зручних інструментів для взаємодії з сервером є однією з задач спеціаліста, що розроблює серверне рішення, оскільки якість продукту значно покращується, його стає легше тестувати та знаходити помилки.

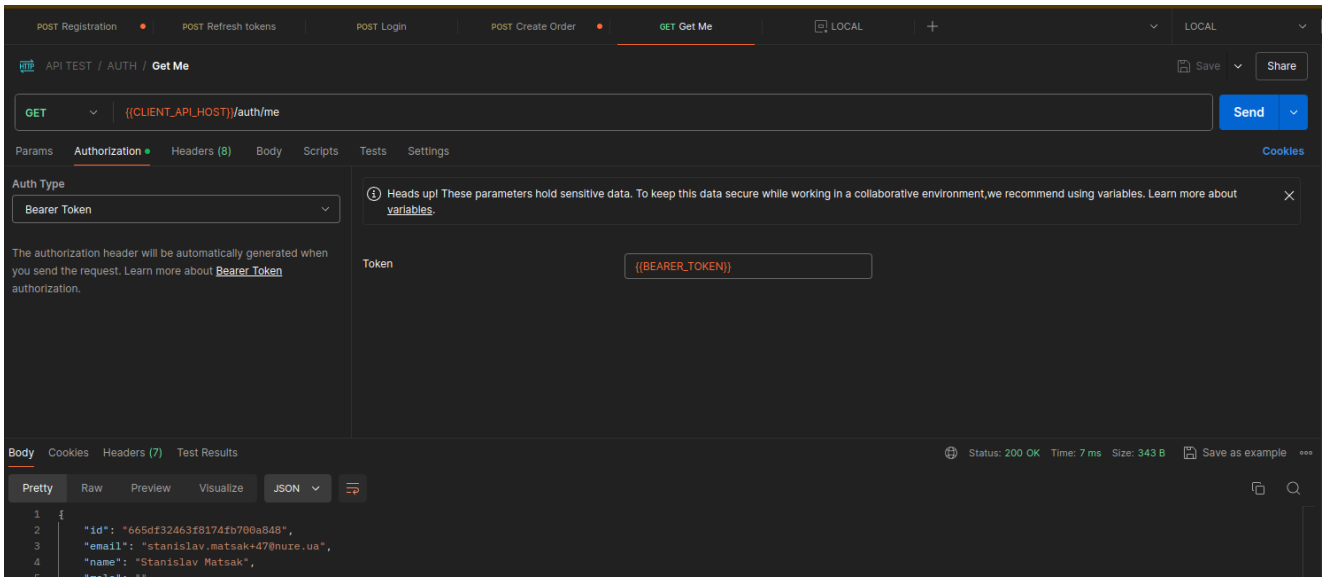


Рисунок 5.4 – Тестування кінцевої точки для отримання інформації про користувача, що виконує запит (ідентифікується за токеном, отриманим після аутентифікації в системі)

Отже, в ході створення серверної частини програмного забезпечення для системи вантажних перевезень, для основних сервісів створено модульні тести; для клієнтського API створено інтеграційні тести; створено колекцію в Postman, що дозволяє вручну протестувати позитивні та негативні сценарії на створених кінцевих точках.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено серверну частину клієнтської частини системи для керування вантажними перевезеннями. Початковий аналіз предметної області вантажних перевезень і доставки вантажів дозволив чітко визначити ключові потреби та завдання, які мала вирішити система. В результаті цього аналізу були окреслені основні функції застосунку, створено UML діаграми, спроєктовано основну серверну частину та засоби роботи із даними.

Функціонал системи включає можливість користувачів реєструватися, вносити зміни у свої персональні дані, створювати замовлення на перевезення, отримувати актуальну інформацію про перевезення, а також ефективно вирішувати можливі конфліктні ситуації під час доставки за допомогою служби підтримки.

Завершений продукт демонструє, що розроблене програмне забезпечення відповідає сучасним вимогам ринку вантажних перевезень. Застосунок забезпечує надійний і ефективний інструмент для керування логістикою, що сприяє оптимізації часу і витрат, пов'язаних із організацією незручних об’ємних вантажних перевезень.

Серверну частину системи вантажних перевезень було покрито модульними та інтеграційними тестами, а так додатково протестовано мануально за допомогою інструменту тестування API Postman.

Серверна частина надає API-документацію у зручному для розробників вигляді. Це дозволяє швидко працювати із кінцевими точками, тобто інтерфейсом самої серверної частини.

Також серверна частина зберігає інформацію про запити та основні метрики, що дозволяє ефективно слідкувати за можливими збоями в системі або проблемними місцями з адміністраторської частини системи.

В результаті роботи вдалось досягнути цілей, поставлених на початку проєктування та розробки системи керування вантажними перевезеннями.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Логістика: навч. посіб. / О. М. Тридід, Г. М. Азаренкова, С. В. Мішина, І.І. Борисенко. – К. : Знання, 2008. – 566 с.

2. «Статистичне оцінювання національного ринку логістичних послуг» Гринчак Н. А. Державна служба статистики України. Національна академія статистики, обліку та аудиту. – К, 2021. – 23 с.

3. Нова Пошта, Київський інноваційний термінал. [Електронний ресурс] – URL: https://novaposhta.ua/kyivskiy_innovatsiyniy_terminal (дата звернення: 15.05.2024).

4. Дописувачі Вікіпедії, «Мувінг» Українська Вікіпедія, [Електронний ресурс] – URL: https://uk.wikipedia.org/wiki/Мувінг (дата звернення: 15.05.2024).

5. «Аналіз особливостей функціонування і розвитку економіки спільної участі на прикладі бізнес-моделі компанії Uber» - Дзюба О.М., Матвієнко К.М., Національний транспортний університет, Київ, Україна, Науковий журнал. Випуск 5, 2017. – 9 с.

6. Sharing Economy [Електронний ресурс]. – URL: http://www.investopedia.com/terms/s/sharing-economy.asp (дата звернення: 15.05.2024).

7. Why the Collaborative Economy Is Changing Everything by Jacob Morgan [Електронний ресурс]. – URL: http://www.forbes.com/sites/jacobmorgan/2014/10/16/why-the-collaborative-economy-ischanging-everything/#406e4afe28a1 (дата звернення: 15.05.2024).

8. R. C. Martin – «Clean Architecture», ISBN-13: 978-0-13-449416-6. ISBN-10: 0-13-449416-4., 2018. – 432 с.

9. R. Mitra, I. Nadareishvili – «Microservices: Up and Running: A Step-by-Step Guide to Building a Microservices Architecture», ISBN-10: 1492075450. ISBN-13: 978-1492075455., 2020. – 316 с.

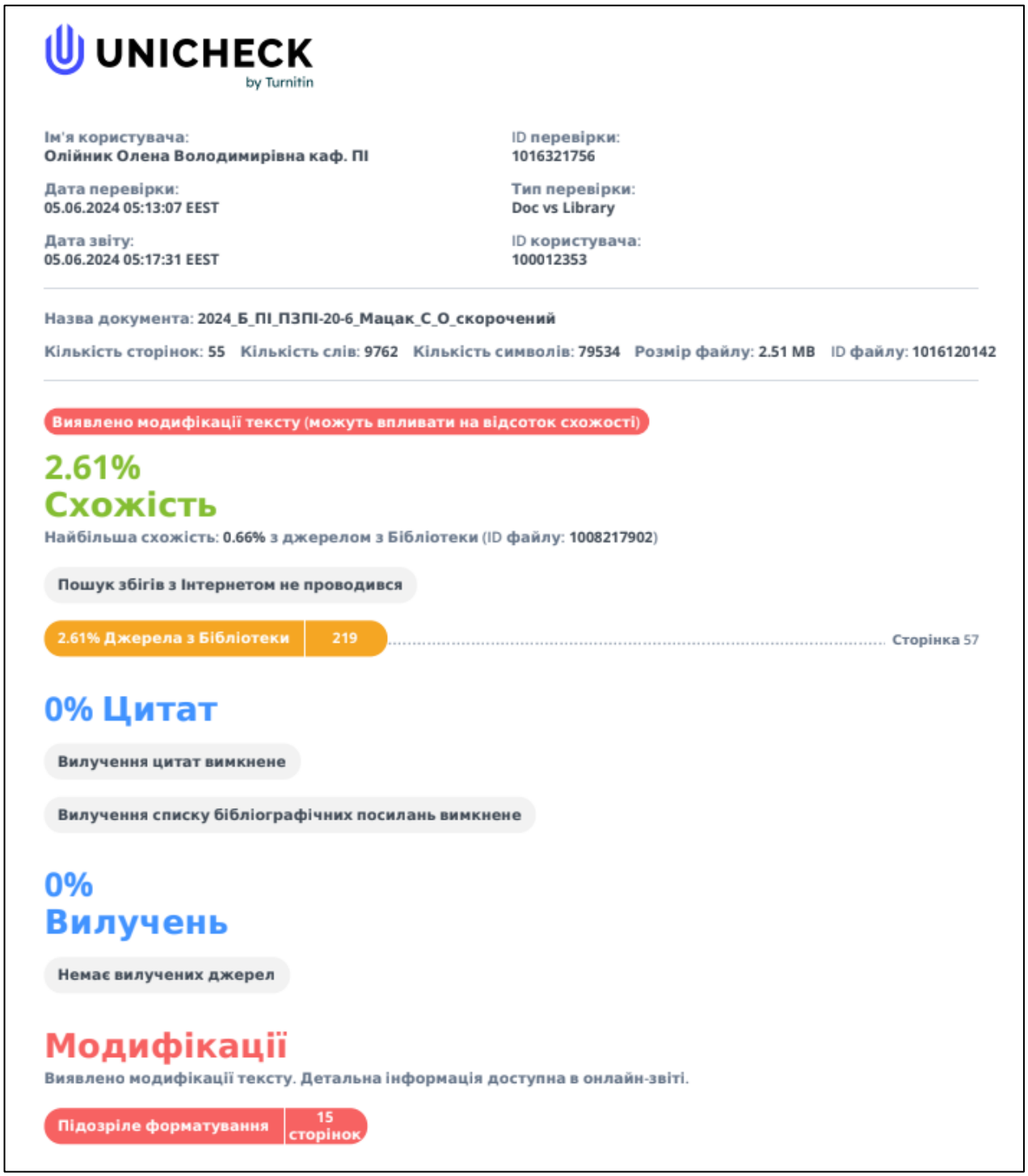
10. E. A. Lee - «Quantifying and Generalizing the CAP Theorem» 2021. – 51 с.

11. OpenAPI Specification [Електронний ресурс] – URL: https://swagger.io/specification/ (дата звернення 15.05.2024)

12. Дописувачі Вікіпедії, «Модульне тестування» Українська Вікіпедія, [Електронний ресурс] – URL: https://uk.wikipedia.org/wiki/Модульне_тестування (дата звернення: 01.06.2024).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016321756

Дата перевірки:
05.06.2024 05:13:07 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
05.06.2024 05:17:31 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_ПЗПІ-20-6_Мацак_С_О_скорочений

Кількість сторінок: 55 Кількість слів: 9762 Кількість символів: 79534 Розмір файлу: 2.51 MB ID файлу: 1016120142

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

2.61%
Схожість

Найбільша схожість: 0.66% з джерелом з Бібліотеки (ID файлу: 1008217902)

Пошук збігів з Інтернетом не проводився

2.61% Джерела з Бібліотеки 219 ..... Сторінка 57

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

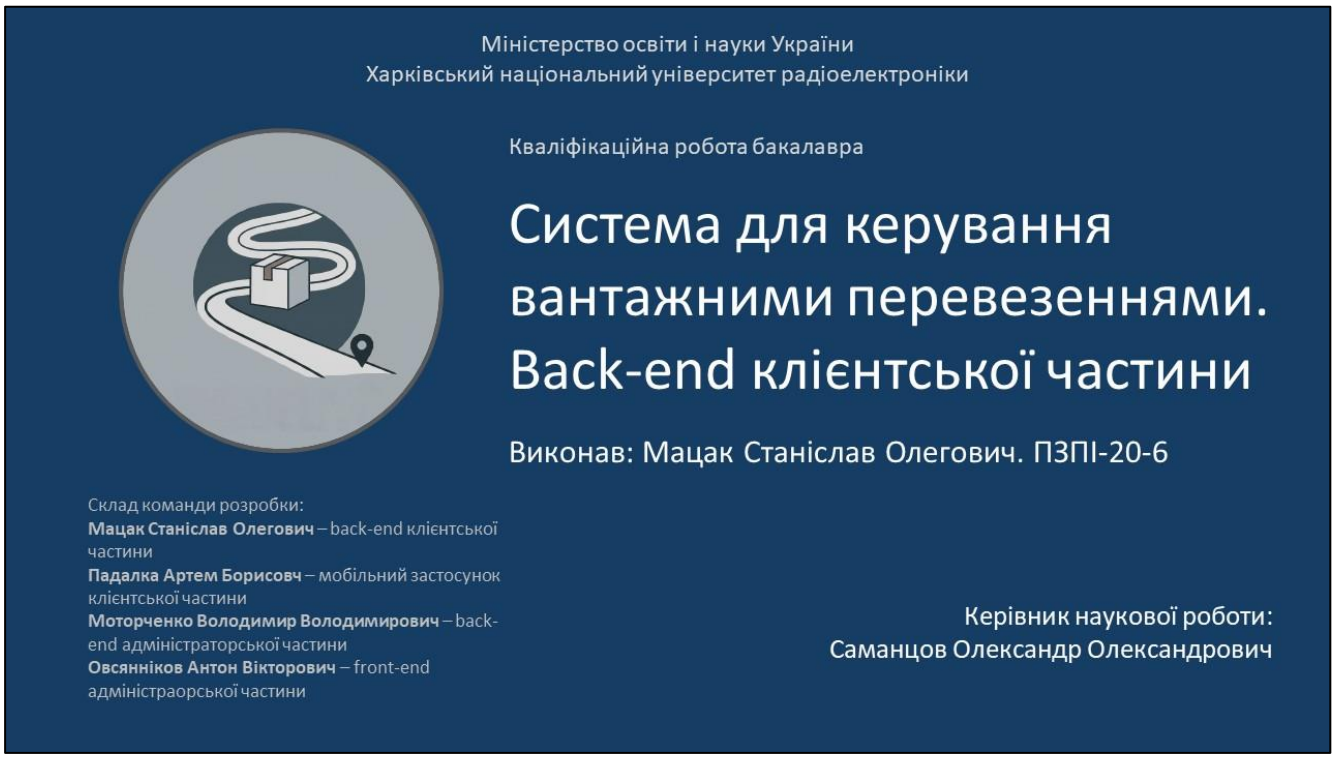
Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

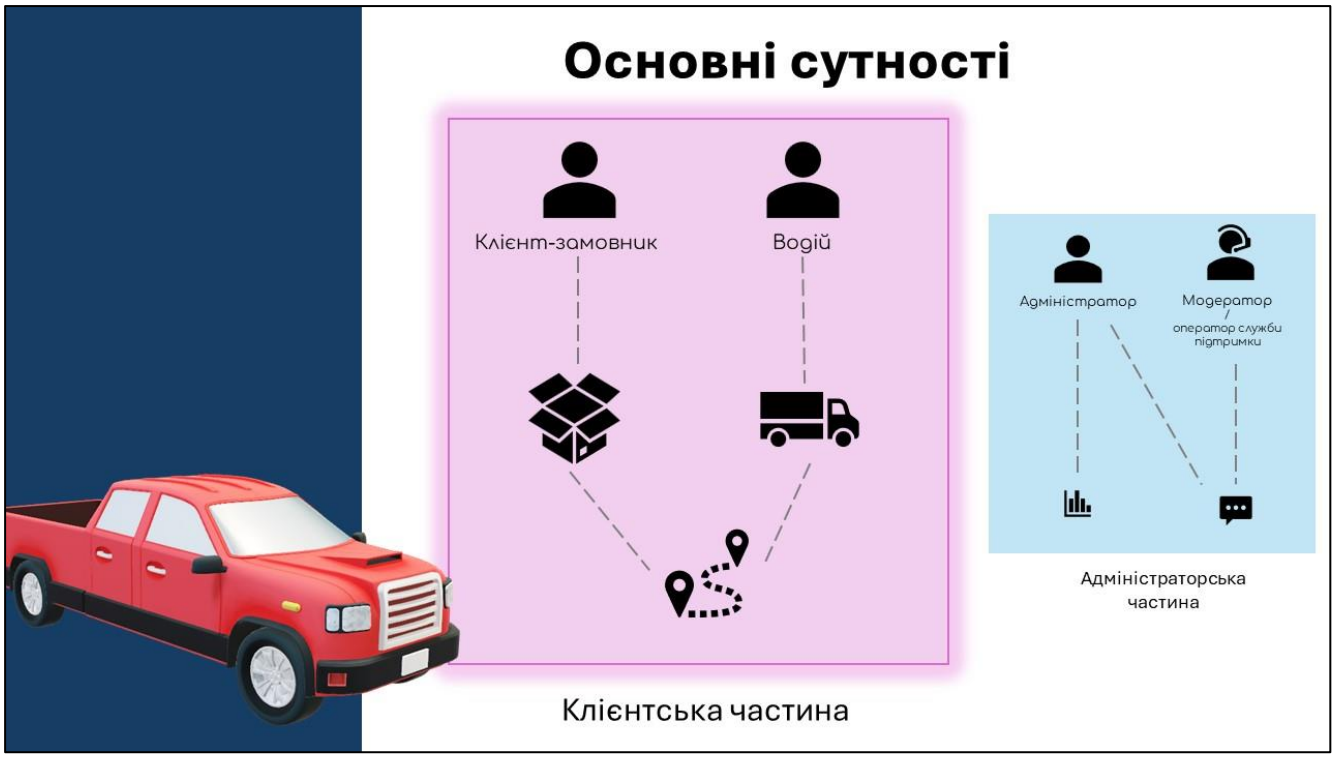
Підозріле форматування 15 сторінок

ДОДАТОК Б

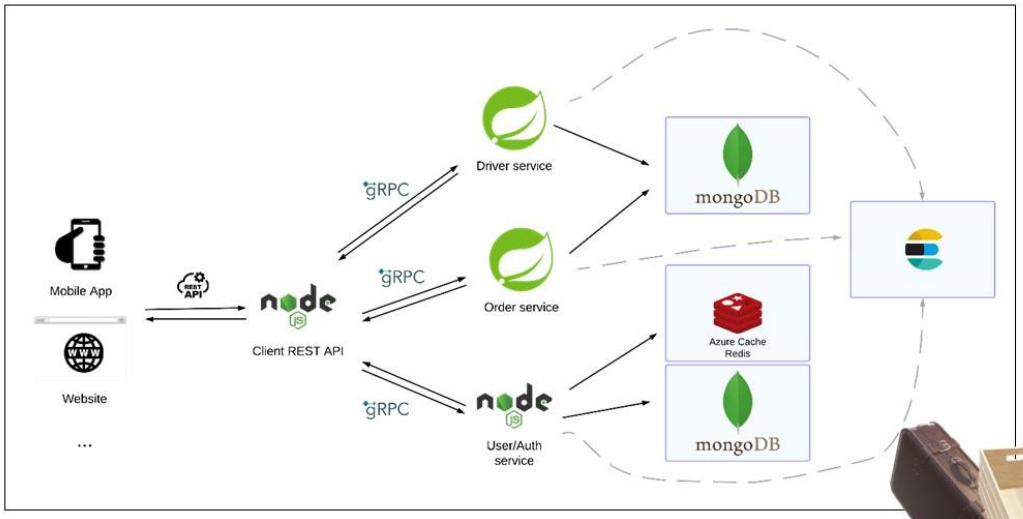
Слайди презентації




Опис системи
Унікальність
Поєднання принципів економіки спільної участі та наявних бізнес-моделей вантажних перевезень
Актуальність
Створення додаткових логістичних можливостей для віддалених регіонів, районів поблизу ЛБЗ
Зручність
Швидке замовлення за допомогою мобільного застосунку, з можливістю пошуку в режимі реального часу або запланованої поїздки


Основні сутності
Клієнт-замовник
Водій
Адміністратор
Модератор
оператор служби підтримки
Адміністраторська частина
Клієнтська частина

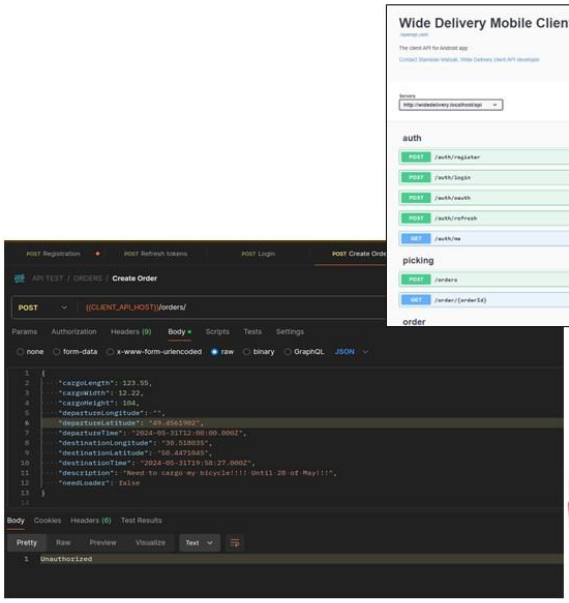
Архітектура

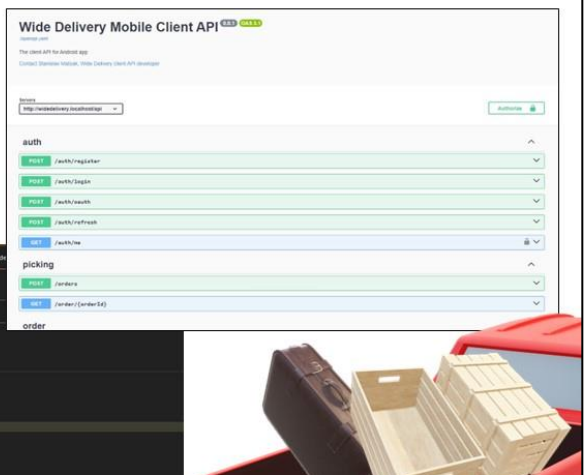


- Мікросервісний тип архітектури
- Комунікація через єдиний API-шлюз
- Окремі бази даних для кожного сервісу

Протоколи

- HTTP/S API
- gRPC (protobuf)
- WebSockets





HTTP/S API

gRPC (protobuf)

WebSockets

Протоколи

```
service OrderService {
  rpc CreateOrder(com.widedelivery.order.proto.CreateOrderInput) returns (com.widedelivery.order.proto.CreateOrderResponse) {}
  rpc GetOrder(GetOrderInput) returns (com.widedelivery.order.proto.OrderResponse) {}
```

Використання в іншому сервісі

```
public static getOrderById(orderId: string): Promise<OrderDto> {
    return new Promise((resolve, reject) => {
        orderService.getOrder({
            order_id: orderId
        }, (err: any, result: any) => {
            if (err) {
                console.error(err);
                reject(err);
            } else {
                console.log(result);
                resolve(OrderDto.parseFromGrpcResponse(result.order));
            }
        })
    });
}
```

Оголошення структури повідомлень та сервісів

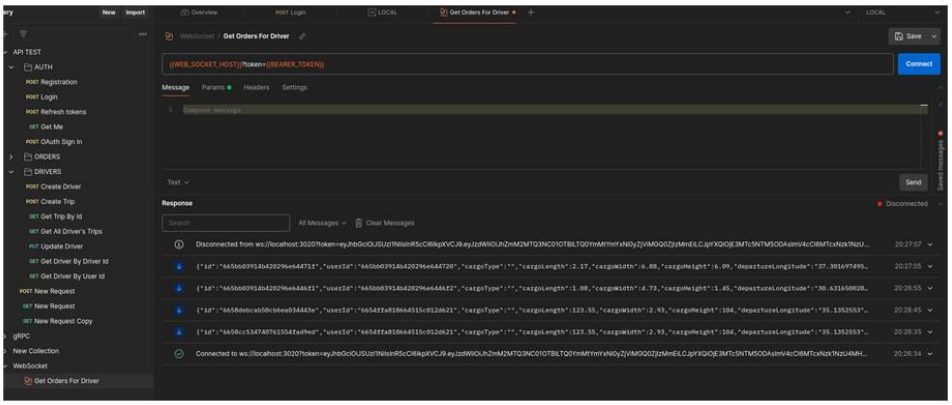
```
message Order {
    string id = 1;
    string user_id = 2;
    double cargo_length = 3;
    double cargo_width = 4;
    double cargo_height = 5;
    double cargo_weight = 6;
    string departure_longitude = 7;
    string departure_latitude = 8;
    google.protobuf.Timestamp departure_time = 9;
    string destination_longitude = 10;
    string destination_latitude = 11;
    google.protobuf.Timestamp destination_time = 12;
    string description = 13;
    bool need_loader = 14;
    string payment_method = 15;
    google.protobuf.Timestamp created_at = 16;
    google.protobuf.Timestamp updated_at = 17;
    string driver_id = 18;
    OrderStatus status = 19;
    string current_location = 20;
}
```

HTTP/S API

gRPC (protobuf)

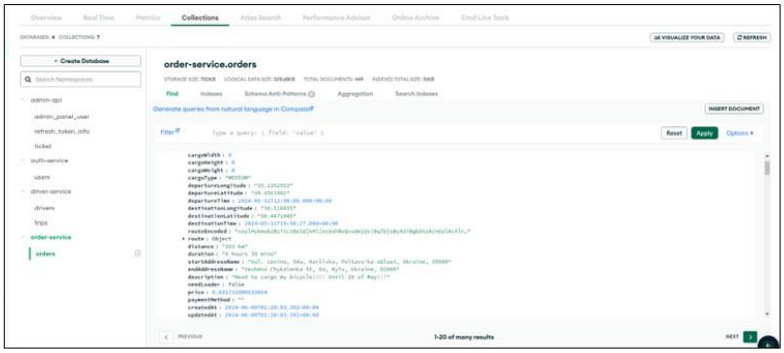
WebSockets

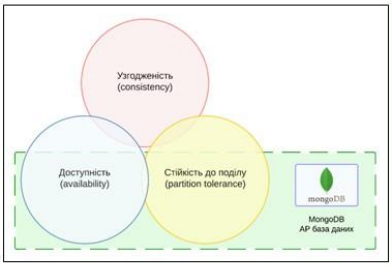
Протоколи



Бази даних та сховища зберігання

MONGO DB





- Збереження напівструктурованої інформації
- Окремий сервіс – окрема база даних
- NoSQL PA/EC (PACELC)

Бази даних та сховища зберігання

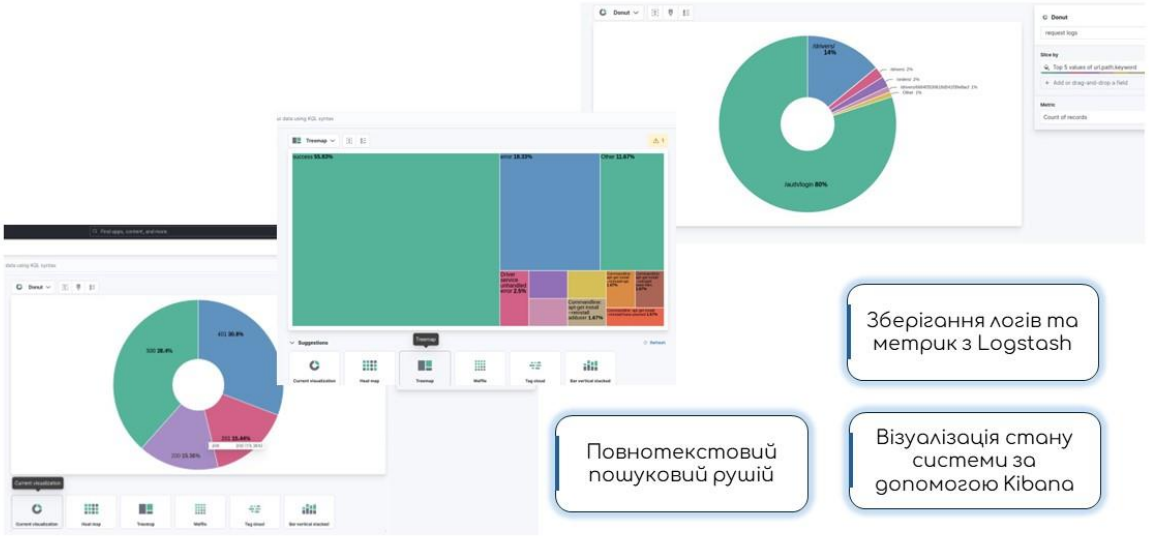
REDIS

- Розподілене сховище пар ключ-значення
- Використовується для кешування даних
- Зберігання сесій в Auth/User service

```
127.0.0.1:6379> KEYS *
1) "2a7fbb8d-02e6-4ec3-9453-f50fb03e95e4"
2) "79c054bd-8405-4b02-b2cb-3c89c27f1071"
3) "b47084de-10d7-495e-b3b0-fc275a538284"
127.0.0.1:6379> GET 2a7fbb8d-02e6-4ec3-9453-f50fb03e95e4
"{\"id\":\"665df32463f8174fb700a848\",\"name\":\"Stanislav
 Matsak\",\"email\":\"stanislav.matsak+47@nure.ua\",\"phon
eNumber\":\"\",\"photo\":\"https://isobarscience-1bfd8.kxc
dn.com/wp-content/uploads/2020/09/default-profile-picture1
.jpg\",\"password\":\"$2a$12$50dGqIsyR.ZzvtSAHpJjIeNAt9n.C
K67shF/26SJvetwgdiTyJl96\",\"provider\":\"local\",\"create
dAt\":\"2024-06-03T16:45:24.043Z\",\"updatedAt\":\"2024-06
-03T16:45:24.043Z\"}"
```

Бази даних та сховища зберігання

ELASTICSEARCH



Зберігання логів та метрик з Logstash

Повнотекстовий пошуковий рушій

Візуалізація стану системи за допомогою Kibana

Алгоритм підбору замовлень

$$k_0 = 1 - \frac{\Delta_o}{T_d + \Delta_{max}}$$

$$k = \max(0, \min(1, k_0))$$

де $\Delta_{max} = \alpha T_d + \beta T_o$ – максимально допустимий час відхилення.

T_d - загальний час, необхідний для проїзду маршруту водія.

T_o - час, необхідний для проїзду маршруту замовлення.

Δ_o - додатковий час, який водій витратить на проїзд до початкової та з кінцевої точки маршруту замовлення.

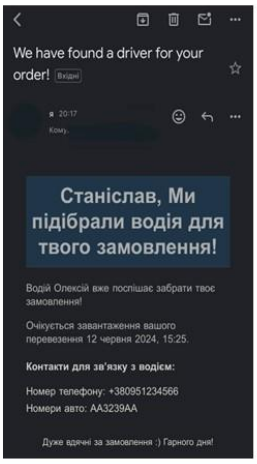
WebSocket

Використовується для тривалих клієнт-серверних з'єднань для передачі даних в режимі реального часу в обидві сторони

Сервіс надсилання запропонованих замовлень неподалік водія

Сповіщення користувача

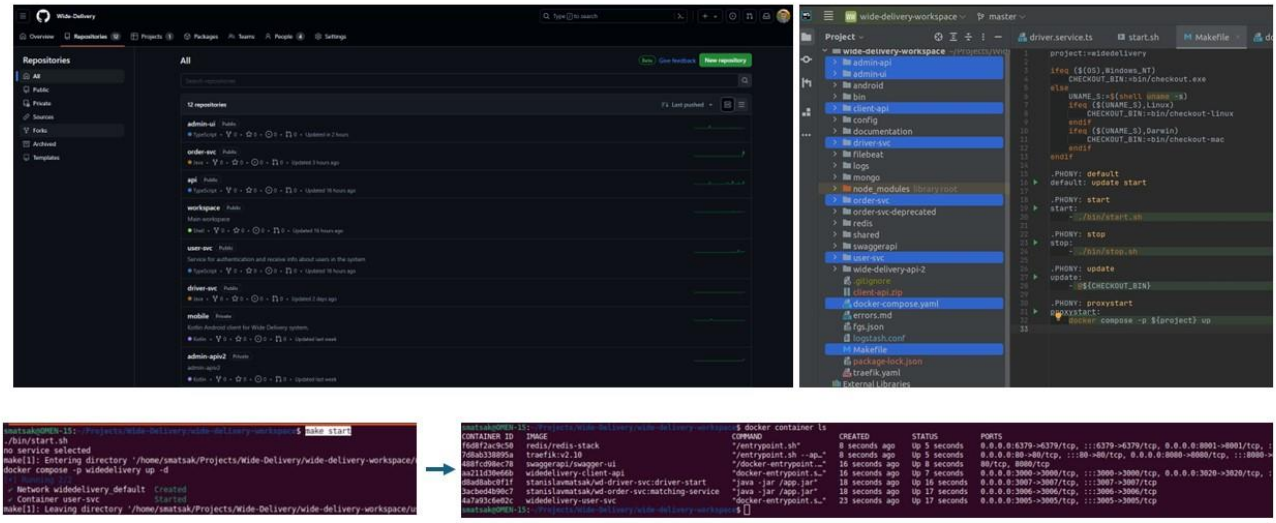
```
static async sendEmail(receiver: string, subject: string,
htmlContent: string) {
        const mailOptions = {
            from: customConfig.notificationServiceEmail,
            to: receiver,
            subject: subject,
            html: htmlContent
        };

        try {
            let info = await
this.transporter.sendMail(mailOptions);
            console.log('Message sent: %s', info.messageId);
            return info;
        } catch (error) {
            console.error('Failed to send email:', error);
            throw error;
        }
    }
```



Надсилання електронних повідомлень з спеціальними пропозиціями

Інформування про зміну статусу замовлення

Облаштування середовища розробки клієнтських застосунків



Висновки

У ході виконання кваліфікаційної роботи було розроблено Back-end системи вантажних перевезень, що ґрунтується на ідеях економіки спільної участі

Мікросервісна архітектура пов’язала сервіси авторизації, створення та обробки замовлень та сервіс взаємодії з водієм за допомогою gRPC.

Мобільний клієнтський застосунок комунікує із серверною частиною за допомогою HTTP-запитів створеного шлюзу REST API.

Мобільний клієнтський застосунок комунікує із серверною частиною за допомогою HTTP-запитів створеного шлюзу REST API.

Дякую за увагу!

Готовий відповісти на ваші
цікаві запитання

ДОДАТОК В

Специфікація програмного продукту

СПЕЦИФІКАЦІЯ ПЗ

До системи керування вантажними перевезеннями

Виконали:
Падалка Артем Борисович
Мацак Станіслав Олегович
Мотречко Володимир Володимирович
Овсянніков Антон Вікторович

Специфікація ПЗ

Вступ

1.1 Огляд продукту

Система вантажних перевезень, що ми розробляємо, є інноваційним рішенням для організації доставки вантажів в межах міста або між містами. Концепція системи базується на моделі Uber (офлайн-послуги, доступні відразу після оплати з мобільного додатку), але з деякими відмінностями, пов'язаними з особливостями вантажних перевезень.

Система повинна надати клієнтам зручний та ефективний спосіб замовлення вантажного перевезення “не відриваючись від екрану мобільного пристрою”, де клієнт мусить вказати габарити, тип вантажу та час і місце доставки. Водії, які можуть бути як представниками компаній, що займаються вантажними перевезеннями, так і звичайними користувачами, мають можливість приймати замовлення та виконувати доставку.

Продукт спрямований на модернізацію традиційних підходів до транспортної логістики, забезпечуючи зручні, швидкі та вартісно ефективні рішення для мувінгових компаній та кінцевих користувачів.

Система дозволяє користувачам замовляти перевезення вантажів через мобільний застосунок, пропонуючи опції для негайних та запланованих доставок. Водії можуть реєструватися у системі, вказуючи свої маршрути та доступність, що дозволяє системі автоматично спарювати замовлення з відповідними водіями, оптимізуючи навантаження та мінімізуючи порожні поїздки.

Ключовим нововведенням є впровадження моделі спільної участі, де користувачі можуть ділитися ресурсом своїх транспортних засобів (зокрема, вантажним простором автомобілів), для збільшення загальної ефективності і зниження витрат. Система також надає докладну інформацію про статус доставки, включаючи трекінг у реальному часі, оцінки та відгуки, що сприяє підвищенню довіри та задоволеності клієнтів.

Це програмне забезпечення має на меті не тільки підвищити ефективність логістичних операцій, але й зробити процес перевезення більш прозорим та доступним для всіх учасників ринку.

Система передбачає два варіанти доставки: “онлайн” (тобто, максимально швидко) або з запланованим часом.

Для водіїв передбачено можливість вказувати радіус пошуку, якщо вони готові приймати замовлення прямо зараз, або вказувати маршрут, за яким вони будуть їхати в майбутньому. Система автоматично підбирає замовлення, що проходять в межах радіусу або вздовж маршруту, та повідомляє клієнта про можливі варіанти доставки. Після прийняття замовлення водієм, клієнт оплачує вартість доставки та очікує, що в зазначений час водій прибуде та забере (чи допоможе завантажити) необхідний товар.

1.2 Мета

Основна мета створення програмного забезпечення - забезпечити зручну та ефективну організацію доставки вантажів для клієнтів у будь-якій точці країни; пов’язати логістично складні регіони та надати клієнтам широкий вибір способів задоволення їх потреб у перевезенні - з варіацією цін, термінів, допомоги в завантаженні/розвантаженні тощо.

Окрім того, ми хочемо максимально зменшити кількість “пустих” поїздок для вантажних автомобілів малого та середнього бізнесу - тобто максимізувати їх корисний ресурс. Для цього система буде забезпечувати можливість підбору вантажів, що мають схожий маршрут доставки, та можливість об'єднання їх в одне замовлення, а також надавати водіям можливість планувати свої маршрути з урахуванням можливих замовлень на перевезення вантажів.

Ми вбачаємо наше призначення у створенні зручного, надійного та ефективного ринку вантажних перевезень, що задовольняє потреби всіх учасників процесу, від користувачів до водіїв та компаній, що займаються перевезеннями.

1.3 Межі

Межі системи:

мобільний додаток для клієнтів;

мобільний додаток для водіїв;

веб-інтерфейс для адміністрування системи;

серверна частина для мобільних додатків (мікросервіси);

серверна частина для веб-інтерфейсу для адміністрування системи (мікросервіси);

база даних, яка зберігає інформацію про клієнтів, водіїв, вантажі, замовлення та доставки.

Межі функціоналу:

можливість замовлення вантажних перевезень клієнтами через мобільний додаток;

можливість для водіїв приймати замовлення та виконувати доставку;

можливість для адміністратора системи керувати роботою системи, відстеження замовлень та доставки, аналіз статистики та звітності;

можливість для адміністратора системи коригувати замовлення та вести діалоги з клієнтами та водіями, для вирішення проблем/питань.

Межі взаємодії з іншими системами:

інтеграція з картографічними сервісами для відображення місцезнаходження водіїв та маршрутів доставки (Google Maps);

інтеграція з платіжними системами для оплати вартості доставки через мобільний додаток (Google Pay);

інтеграція з системами збору та аналізу метрик та іншої статистики щодо роботи системи (Elasticsearch, Kibana, Logstash).

1.4 Посилання

Посилання на документи, що стосуються вимог до системи вантажних перевезень:

ДСТУ 2609-94 Вантажні автомобільні перевезення. Терміни та визначення

ДСТУ ISO 9001:2015 Системи управління якістю. Вимоги (ISO 9001:2015, IDT)

ДСТУ 3649:2010 КОЛІСНІ ТРАНСПОРТНІ ЗАСОБИ. Вимоги щодо безпечності технічного стану та методи контролювання

Посилання на документи, що стосуються вимог до програмного забезпечення:

ДСТУ ISO/IEC/IEEE 12207:2018 Інженерія систем і програмних засобів. Процеси життєвого циклу програмних засобів (ISO/IEC/IEEE 12207:2017, IDT)

ДСТУ ISO/IEC 25010:2016 Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE). Моделі якості системи та програмних засобів (ISO/IEC 25010:2011, IDT)

1.5 Означення та абревіатури

АПІ (API) - Application Programming Interface, програмний інтерфейс застосунку.

GPS (GPS) - Global Positioning System, глобальна система позиціонування.

REST (Representational State Transfer) - архітектурний стиль веб-сервісів, що використовується для взаємодії між клієнтськими та серверними додатками.

СКБД - Система керування базами даних.

WebSocket, WS - протокол, що призначений для обміну інформацією між браузером та вебсервером в режимі реального часу.

UI (User Interface) - графічний інтерфейс користувача, що використовується для взаємодії з програмним забезпеченням.

UX (User Experience) - загальний досвід користувача при взаємодії з програмним забезпеченням.

HTTPS (Hypertext Transfer Protocol Secure) - протокол передачі гіпертексту, що використовується для безпечного з'єднання між клієнтом та сервером (з’єднання мобільного застосунку із сервером встановлюється за допомогою HTTP/S).

OAuth (Open Authorization) - протокол авторизації, що використовується для надання обмеженого доступу до ресурсів користувача (авторизація через Google Sign-In працює з використанням протоколу OAuth 2.0).

Користувач - особа, що використовує мобільний застосунок (клієнт-замовник або водій).

Клієнт - особа, яка замовляє послуги з перевезення вантажів.

Водій - особа, яка виконує перевезення вантажів.

Логбук водія - електронне візуальне представлення останніх подій, що були зроблені водієм.

Замовлення - запит клієнта на перевезення вантажу.

Запланована поїздка - інформація, надана водієм, щодо його майбутньої поїздки між вказаними містами чи місцями, впродовж якої він може виконати замовлення.

Маршрут - шлях, за яким здійснюється перевезення вантажу.

Радіус пошуку - відстань, в межах якої водій готовий прийняти замовлення.

Доставка - процес перевезення вантажу від місця відправлення до місця призначення.

Вантаж - предмети, що перевозяться.

Габарити - розміри вантажу.

Тип вантажу - характеристика вантажу, що визначає особливості його перевезення (харчові продукти, меблі, будівельні матеріали тощо).

Час доставки - час, протягом якого має бути здійснено доставку вантажу.

Оцінка - оцінка клієнтом якості наданих послуг з перевезення вантажу.

Відгук - коментар клієнта про надані послуги з перевезення вантажу.

Адміністратор - особа, що здійснює управління системою.

Модератор - особа, що представляє службу підтримки, комунікує з клієнтами, водіями та компаніями і має можливість редагувати замовлення, скасовувати оплату тощо.

2 ЗАГАЛЬНИЙ ОПИС

2.1 Перспективи продукту

Ми бачимо Wide Delivery як екосистему, яка об'єднає клієнтів, водіїв та компанії, що займаються перевезеннями, надаючи їм інструменти для швидкої, надійної та ефективної доставки вантажів.

Наші плани розвитку сягають далеко за межі простого створення зручного додатку. Ми прагнемо стати лідером на ринку, задаючи нові стандарти якості, зручності та прозорості. Для досягнення цієї мети ми зосередимося на трьох ключових напрямках.

По-перше, ми будемо поступово розширювати географію обслуговування Wide Delivery. Наша мета - охопити всю територію України, зв'язавши навіть найвіддаленіші населені пункти. Ми прагнемо забезпечити доступність наших послуг для всіх, хто потребує швидкої та надійної доставки вантажів, незалежно від їх місця розташування.

По-друге, ми постійно працюємо над удосконаленням функціоналу мобільного додатку. Ми плануємо впровадити нові зручні функції, які зроблять процес замовлення та відстеження доставки ще більш інтуїтивним та ефективним. Наша команда аналізує відгуки користувачів, вивчає потреби ринку та слідкує за останніми технологічними трендами, щоб запропонувати нашим клієнтам найкращі рішення.

По-третє, ми активно працюємо над інтеграцією Wide Delivery з іншими системами. Ми розуміємо, що багато компаній вже використовують власне програмне забезпечення для управління логістикою, тому ми надамо їм можливість

інтегрувати свої системи з Wide Delivery через API. Це дозволить автоматизувати процес отримання замовлень, управління доставкою та обміну даними, спрощуючи роботу та підвищуючи ефективність бізнес-процесів.

2.2 Функції програмного забезпечення

Програмне забезпечення для управління вантажними перевезеннями включає в себе широкий спектр функцій, які забезпечують ефективність та зручність використання для клієнтів, водіїв та адміністраторів системи. Нижче наведено список кожної з основних функцій:

реєстрація та авторизація користувачів (клієнтів та водіїв);

авторизація користувачів (адміністратор, головний помічник, помічник);

введення інформації про вантаж та його характеристики;

введення інформації про місце розташування вантажу та місце призначення;

введення інформації про час доставки;

обчислення вартості доставки;

формування замовлення на доставку;

пошук водіїв, які підходять під замовлення;

приймання/відхилення замовлень водіями;

надсилання електронних листів з інформуванням про зміну статусу замовлення;

відстеження статусу замовлення;

спільна робота водіїв та клієнтів з доставкою;

ведення статистики та звітності;

адміністрування системи з веб-інтерфейсу (підтримка онлайн-чатів, редагування інформації про замовлення та скарги, тощо);

інтеграція з картографічними сервісами (Google Maps/Google Streets);

інтеграція з платіжними системами (Google Pay);

забезпечення безпеки та конфіденційності даних користувачів;

надання технічної підтримки користувачам.

2.3 Характеристики користувачів

Система передбачає наявність таких основних категорій користувачів: клієнт, водій, адміністратор, модератор.

Клієнти - фізичні та юридичні особи, які потребують послуг з доставки вантажів.

Водії - фізичні особи або транспортні компанії, що надають послуги з доставки вантажів.

Адміністратори - особи, що відповідають за управління системою, переглядом звітності, менеджмент персоналу

Модератори - особи, що мають доступ до інформації щодо клієнтів, водіїв, замовлення та можуть підтримувати комунікацію і вирішувати проблеми із взаємодією інших користувачів із системою.

Вимоги клієнтів:

простe та інтуїтивно зрозуміле користування додатком;

можливість швидкого та зручного замовлення вантажоперевезень;

можливість замовити допомогу в завантаженні вантажу чи його розвантаженні;

можливість відстежувати статус замовлення в реальному часі;

надійність та пунктуальність виконання замовлення;

гарантії безпеки та повернення коштів у разі проблем із доставкою замовлень;

зручні опції оплати;

зручні варіанти авторизації (включно з Google Sign-In);

можливість залишити відгук про виконання замовлення.

можливість звернутись до технічної підтримки, якщо є така потреба

Вимоги водіїв:

простe та інтуїтивно зрозуміле користування додатком;

можливість швидкого та зручного прийняття замовлень;

можливість планування маршрутів та вантажоперевезень, в тому числі запланованих на значний час наперед;

можливість відстежувати статус виконання замовлення;

надійну та своєчасну оплату за виконані перевезення;

можливість залишити відгук про виконання замовлення.

можливість звернутися до служби підтримки в текстовому форматі.

Вимоги адміністраторів:

доступ до інформації про клієнтів та водіїв та можливість її редагування;

можливість створювати, редагувати та видаляти модераторів та інших адміністраторів системи;

доступ до інформації про замовлення та можливість їх редагування, скасування та повернення коштів;

можливість вести чат з клієнтами та водіями для надання допомоги та розв'язання проблем;

можливість керувати скаргами та поверненнями коштів;

доступ до статистики та звітів про роботу системи;

можливість керувати доступом користувачів до системи та налаштуваннями безпеки.

Модератор має аналогічні до адміністратора вимоги щодо системи, однак він не має доступу до редагування інформації щодо персоналу, інших менеджерів та адміністраторів.

2.4 Загальні обмеження

Обмеження функціоналу

система не передбачає перевезення небезпечних вантажів, таких як вибухонебезпечні, отруйні, радіоактивні та інші речовини, що підпадають під дію чинного законодавства щодо перевезення небезпечних вантажів;

система не передбачає перевезення вантажів, що порушують чинне законодавство або права інтелектуальної власності;

система не передбачає перевезення живих істот, за винятком тварин, які перевозяться разом з власниками в спеціальних контейнерах для перевезення тварин;

система не передбачає перевезення вантажів, які перевищують максимально допустимі габарити та вагу, встановлені для перевезення на конкретному типі транспортного засобу;

максимальні допустимі габарити вантажу визначаються параметрами транспортних засобів, що використовуються водіями-перевізниками. Користувач повинен вказати точні габарити свого вантажу під час створення замовлення, щоб система могла коректно підібрати відповідний транспортний засіб. Водій повинен вказати точні габарити свого транспортного засобу, для коректного пошуку замовлення під його автомобіль;

система надає можливість користувачеві вибрати один з двох варіантів доставки: якомога швидша або запланована на певну дату в майбутньому. При виборі швидкої доставки, користувач розуміє, що вартість може бути вищою, ніж при замовленні на майбутнє. У разі вибору запланованої доставки, користувач зобов'язаний вказати точну дату і час, коли вантаж буде готовий до відправлення.

Обмеження географії:

система не розрахована на міжнародні перевезення, географія місця призначення замовлення обмежується лише територією держави, звідки замовлення створено (початково - в межах України);

система може не працювати в деяких районах, які є небезпечними для перевезення вантажів (регіон ведення бойових дій, тимчасово окуповані території тощо). Система повинна сповіщувати про неможливість перевезення вантажу ще на етапі створення замовлення.

Обмеження, що стосуються водіїв-перевізників:

водії-перевізники повинні використовувати транспортні засоби, що відповідають вимогам системи щодо габаритів і типу вантажу. Система не допускає використання транспортних засобів, що не відповідають цим вимогам, для виконання замовлень на перевезення вантажів;

водії-перевізники можуть вказувати радіус пошуку, якщо вони готові прийняти замовлення прямо зараз, або вказувати маршрут, за яким вони будуть їхати в майбутньому. Система підбирає замовлення, що проходять в межах радіусу чи вздовж маршруту (з можливими відхиленнями) і сповіщає водія-перевізника про можливі варіанти. Водій зобов'язаний вчасно повідомляти систему про зміни в своєму розкладі або маршруті;

водії-перевізники можуть надавати послуги з допомоги в завантаженні та розвантаженні вантажу, але це не є обов'язковою умовою. Якщо водій-перевізник не може надати таку послугу, він повинен чітко вказати це в своєму профілі. Ця послуга обов’язково сплачується додатково до тарифу перевезення, про що клієнта буде повідомлено відразу при створенні замовлення.

Обмеження відповідальності:

Система не несе відповідальності за вантаж під час перевезення. Відповідальність за вантаж покладається на відправника та водія-перевізника. У разі пошкодження або втрати вантажу, сторони зобов'язані самостійно вирішувати питання щодо компенсації збитків. Команда модераторів системи має обов’язок допомагати сторонам під час вирішення таких питань, однак не несе відповідальності за будь-які пошкодження;

сторони зобов'язані дотримуватися вимог щодо безпеки перевезень, встановлених чинним законодавством. У разі порушення цих вимог, сторони несуть відповідальність відповідно до закону;

система не гарантує надання послуг з перевезення вантажів у всіх випадках. У разі відсутності водіїв-перевізників, що готові виконати замовлення або за

наявності інших обставин, що перешкоджають наданню послуг, система повідомляє користувача про неможливість виконання замовлення.

2.5 Припущення і залежності

Для того, щоб Wide Delivery успішно функціонував та розвивався, ми враховуємо низку важливих факторів. По-перше, ми виходимо з припущення, що наші користувачі мають доступ до сучасних мобільних пристроїв з операційною системою Android та стабільним інтернет-з'єднанням, що дозволить їм повноцінно використовувати всі можливості мобільного додатку. Ми також очікуємо, що водії, які приєднаються до платформи, мають необхідний досвід та кваліфікацію для керування вантажними автомобілями, а також відповідально ставляться до правил дорожнього руху. Крім того, ми розраховуємо на те, що всі учасники платформи - клієнти, водії та транспортні компанії - будуть діяти відповідно до чинного законодавства України, яке регулює сферу вантажних перевезень. Ми також спираємося на стабільну та безперебійну роботу сторонніх сервісів, таких як Google Maps, Google Pay та інші платформи, інтегровані в Wide Delivery.

Не менш важливим є розуміння залежностей, які можуть впливати на роботу платформи. Очевидно, що стабільне інтернет-з'єднання є критично важливим як для клієнтів, так і для водіїв. Будь-які проблеми з інтернетом можуть призвести до збоїв у роботі Wide Delivery та ускладнити процес замовлення та відстеження доставки. Точність та доступність геолокаційних даних, що надходять з мобільних пристроїв, також є ключовим фактором для коректної роботи платформи. Неточності або відсутність геолокації можуть спричинити помилки у визначенні місця розташування клієнтів та водіїв, а також ускладнити побудову оптимальних маршрутів доставки.

Безпека та надійність інтегрованих платіжних систем також є важливою складовою успіху Wide Delivery. Від них залежить можливість клієнтів зручно та безпечно оплачувати замовлення.

Не менш важливою є довіра та репутація платформи. Ми докладаємо максимум зусиль, щоб створити Wide Delivery як надійний та безпечний сервіс, який гарантує якісне виконання замовлень, захищає інтереси всіх учасників та формує позитивний досвід взаємодії.

3 КОНКРЕТНІ ВИМОГИ

3.1 Вимоги до зовнішніх інтерфейсів

3.1.1 Інтерфейс користувача

Основна взаємодія користувача з системою Wide Delivery відбувається через мобільний додаток, який розроблено з метою забезпечення максимально зручного та інтуїтивно зрозумілого доступу до всіх функцій сервісу. Додаток, призначений як для клієнтів, так і для водіїв, має дозволяти виконувати всі необхідні дії безпосередньо в ньому, без необхідності звертатися до сторонніх сервісів.

Користувачі можуть авторизуватися в системі за допомогою облікового запису Google або ж створити обліковий запис, використовуючи свою електронну пошту. Клієнти легко створюють замовлення на перевезення, вказуючи всі необхідні параметри вантажу: його габарити, тип, адресу завантаження та доставки, бажаний час прибуття, а також мають можливість зазначити необхідність допомоги з завантаженням та розвантаженням. Водії, в свою чергу, мають доступ до детальної інформації про доступні замовлення, включаючи маршрут, тип вантажу, вартість доставки та інші важливі деталі.

Навігація в додатку має бути простою та інтуїтивно зрозумілою, дозволяючи швидко переміщуватися між різними екранами та розділами. Система забезпечує можливість відстежувати поточний статус замовлення. Користувачі також мають можливість редагувати свої профілі, змінюючи особисту інформацію та налаштування. У системі є можливість звертатися до служби підтримки у зручному месенджері Telegram, де їм зможуть надати допомогу Support’и.

Wide Delivery має включати систему оплати з чітко визначеними тарифами, що гарантує прозорість та передбачуваність вартості послуг.

Адміністратори системи Wide Delivery взаємодіють з нею через спеціалізований веб-інтерфейс, який забезпечує зручний та інтуїтивно зрозумілий доступ до всіх функцій управління сервісом. Авторизація здійснюється через ввід пошти та паролю, після чого адміністратори можуть керувати базою користувачів, включаючи створення, редагування та видалення облікових записів. Вони мають доступ до детальної інформації про всі замовлення, можуть переглядати та редагувати деталі замовлень, відстежувати статус доставки в реальному часі, а також аналізувати різні аспекти замовлень для підвищення ефективності операцій.

Крім того, веб-інтерфейс адміністратора надає можливості для аналізу статистики, що допомагає в ухваленні обґрунтованих рішень. Вони також відповідають за підтримку безпеки та конфіденційності даних користувачів. Для надання підтримки користувачам, адміністратори можуть використовувати вбудований чат або електронну пошту, що дозволяє швидко реагувати на запити та вирішувати проблеми, забезпечуючи високий рівень обслуговування.

3.1.2 Апаратні інтерфейси

Пристрої, що використовують мобільний додаток, повинні мати не менш ніж 2ГБ ОЗУ, і мати частоту процесора не нижче 1 ГГц. Мобільні пристрої повинні мати 350 Мб вільної пам’яті у постійному сховищі для встановлення додатку та його нормального функціонування.

3.1.3 Програмний інтерфейс

Мобільний додаток створюється першочергово для Android-пристроїв, отже Android версії 7.0+ є вимогою щодо ОС мобільного пристрою.

Будуть використовуватися наступні API:

Google OAuth 2.0 API;

Google Maps API;

Google Pay API.

Браузери, що запускатимуть веб-застосунок адміністраторської частини мають підтримувати стандарт ECMAScript 2015 (ES6). Отже, мінімальними необхідними версіями найпопулярніших у світі браузерів є:

Chrome 51 + або вище;

Edge 15 або вище;

Safari 10 або вище;

Mozilla Firefox 54 або вище;

Opera 38 або вище;

Internet Explorer 11 (частково).

Веб-застосунок, призначений для адміністраторів та модераторів, може бути запущений з Windows 7 або вище, Linux, MacOS 10.5 або вище.

3.1.4 Комунікаційний протокол

Wide Delivery використовує різноманітні комунікаційні протоколи для забезпечення ефективної та надійної взаємодії між різними компонентами системи.

Для спілкування мобільного додатку з сервером буде застосовано HTTP/HTTPS протокол. Це забезпечить безпечну передачу даних та захистить конфіденційну інформацію користувачів. Окрім того, буде впроваджено технологію WebSocket для підтримки швидкої передачі постійних пакетів даних із сервера на клієнт. WebSocket дозволить реалізувати функції реального часу, такі як відображення поточного місцезнаходження вантажу на карті та оновлення статусу замовлення.

Внутрішня комунікація між мікросервісами та основним API на стороні сервера буде здійснюватися за допомогою системи gRPC. gRPC - це сучасний високопродуктивний фреймворк для віддаленого виклику процедур (RPC), який забезпечує швидку та ефективну взаємодію між сервісами. Використання gRPC дозволить оптимізувати роботу серверної частини Wide Delivery та забезпечити її масштабованість.

REST буде використовуватися для надання зовнішнього API для інтеграції з іншими системами, такими як платіжні шлюзи та картографічні сервіси. REST API забезпечить стандартизований інтерфейс для взаємодії з Wide Delivery та дозволить стороннім розробникам легко інтегрувати свої сервіси з нашою платформою.

3.1.5 Обмеження пам'яті

Система Wide Delivery повинно бути оптимізованою для роботи з великими обсягами даних, зберігаючи при цьому швидку та стабільну роботу системи. Для цього необхідно застосовувати алгоритми стиснення даних, кешування, а також вміло використовувати пам'ять пристрою.

Основна інформація зберігатиметься в реляційних та NoSQL базах даних, тож додаток повинен мати мінімальну кількість інформації, що зберігається безпосередньо на мобільному девайсі.

3.1.6 Операції

Основна мета: Опис операцій, які здійснюються в рамках системи, дозволяє забезпечити зручність і ефективність процесів взаємодії користувачів з додатком.

Операції системи включають:

Створення замовлення: Користувачі можуть створювати замовлення через мобільний додаток, вказуючи деталі вантажу (габарити, тип, час доставки, місце відправлення та прибуття). Система автоматично підбирає водія на основі заданих критеріїв.

Оновлення статусу замовлення: Водії можуть оновлювати статус замовлення у режимі реального часу, що дає можливість користувачам відстежувати процес доставки.

Підтвердження доставки: Після завершення доставки водій вводить статус як завершений, а клієнт отримує повідомлення про успішну доставку з можливістю залишити відгук.

Скасування замовлення: Користувач має можливість скасувати замовлення до моменту його прийняття водієм, з інформуванням водія та поверненням коштів.

Оплата за допомогою інтегрованих платіжних систем: Користувачі можуть оплачувати послуги через зручні та безпечні платіжні системи, інтегровані в мобільний додаток.

Зауваження: Всі операції повинні супроводжуватися детальними повідомленнями про статус операції для забезпечення прозорості та контролю за процесом доставки вантажів.

3.1.7 Функції продукту

3.1.7.1 Реєстрація, авторизація (мобільний застосунок)

Вступ

Користувач повинен мати можливість авторизуватися через Google або зареєструвати обліковий запис через пошту для подальшої авторизації в додатку під своїм обліковим записом.

Вхідні дані

Користувач завантажив застосунок, вводить електронну пошту та пароль або авторизується через обліковий запис Google та натискає кнопку підтвердження.

Обробка

За наявності акаунту, перевіряється правильність введеного паролю, інакше створюється новий акаунт. Для Google Sign In зберігати пароль в БД не потрібно.

Результати

У разі правильних надісланих даних користувач отримує токен доступу до системи та токен відновлення токену доступу. Застосунок перенаправляє його на головну сторінку мобільного додатку.

Обробка помилок

Користувач не зможе зареєструватися/увійти у разі будь-якої помилки, неправильного паролю, неіснуючого акаунту, або якщо намагається пройти аутентифікацію з Google, при створеному за допомогою пошти та паролю акаунті чи навпаки. Back-end повинен повертати HTTP-статус 500 для внутрішніх помилок сервера, 403 для неправильно введених пошти чи пароля та некоректного OAuth-токену.

3.1.7.2 Створення замовлення (мобільний застосунок)

Вступ

Користувач повинен мати можливість створити замовлення після успішної аутентифікації.

Вхідні дані

Користувач вводить інформацію про вантаж, його тип, габарити, місце призначення та час відправлення, зазначає необхідність у допомозі з завантаження чи розвантаження.

Обробка

Замовлення перевіряється на валідність, оцінюється можливість його виконання, визначається його ціна та розпочинається підбір водіїв, що можуть виконати це замовлення.

Результати

У разі коректного створення замовлення воно валідується та переходить на етап пошуку водія, що зможе виконати замовлення. Також користувач отримує інформацію про орієнтовну вартість замовлення із пропозицією щодо сплати за допомогою Google Pay.

Обробка помилок

У разі помилки створення замовлення користувач отримує інформацію про причину помилки та контактами служби підтримки, для уточнення деталей. Back-end клієнтської частини повинен повертати повідомлення, чому саме замовлення не може бути прийнято до обробки.

3.1.7.3 Оплата замовлення

Вступ

Клієнт повинен оплатити замовлення для того, щоб водій розпочав його виконання.

Вхідні дані

Замовлення успішно прийнято до обробки та клієнтський back-end визначнив вартість замовлення.

Обробка

Сторонній сервіс проводить оплату та передає результат оплати серверу. Сервер перевіряє, чи справді сторонній сервіс провів оплату, і чи запит не був виконаний зловмисником.

Результати

У разі успішної оплати розпочинається процес підбору водія.

Обробка помилок

У разі проведення оплати та списання коштів, але відсутності зміни статусу замовлення, користувач має змогу звернутись до служби підтримки. У разі помилкових відповідей від постачальника платіжних послуг, користувачу буде запропоновано ще раз оплатити замовлення.

3.1.7.4 Перегляд статусу замовлення в режимі реального часу (мобільний застосунок)

Вступ

Клієнт повинен бути проінформованим про зміну статусу замовлення та його поточне місцерозташування.

Вхідні дані

Підтверджене замовлення, що розпочало своє виконання. Водій підтвердив відправку до місця призначення. Водій передає своє місцезнаходження за допомогою увімкненого на мобільному пристрої GPS.

Обробка

Перевірка того, що GPS збігається з визначеним маршрутом перевезення та збереження поточного місцезнаходження, оцінка очікуваного часу прибуття. Надсилання сповіщення клієнтові щодо статусу замовлення за допомогою електронної пошти.

Результати

Клієнт дізнається про статус замовлення та поточне місцезнаходження вантажу.

Обробка помилок

В разі помилки отримання місцезнаходження впродовж виначеного часу модератор служби підтримки зв’язується із водієм. Також надсилається електронний лист про помилку, що сталась під час виконання замовлення.

3.1.7.5 Підбір замовлення для водія (мобільний застосунок)

Вступ

Система підбирає замовлення для водія та пропонує прийняти його.

Вхідні дані

Водій вказує свої налаштування - радіус пошуку, або запланований маршрут, можливість бути вантажником, а також поточний GPS (надає дозвіл додатку на передачу).

Обробка

Система підбирає замовлення, що відповідають вказаним у налаштуваннях критеріях, та пропонує перелік підібраних замовлень.

Результати

Водій бачить список замовлень та приймає їх, або відхиляє. В разі відхилення, замовлення більше не з’являється у списку та передається наступному водієві.

Обробка помилок

В разі помилок з отриманням замовлень водієві буде запропоновано звернутись до служби підтримки застосунку.

3.1.7.6 Звернення до служби підтримки (мобільний застосунок, система модерації)

Вступ

Користувач має можливість звернутись до служби підтримки, що працює за допомогою чат-боту Telegram.

Вхідні дані

Користувач мобільного застосунку (водій чи клієнт) потребує допомоги служби підтримки та натискає кнопку в мобільному застосунку, що розпочинає діалог в чат-боті Telegram.

Обробка

Користувач інтерфейсу адміністратора відповідає на скаргу в інтерактивному вигляді у виді чату.

Результати

Після завершення спілкування по скарзі, користувач інтерфейсу адміністратора змінює статус скарги до відповідного рішення, прийнятого обома сторонами. Клієнт також має можливість завершити розгляд питання, якщо його було вирішено.

3.1.7.7 Перегляд статистики на головній сторінці (система модерації)

Вступ

Користувач інтерфейсу адміністратора після логіну заходить на головну сторінку.

Вхідні дані

Користувач інтерфейсу адміністратора повинен бути зареєстрованим.

Обробка

Користувач інтерфейсу адміністратора перевіряється на аунтифікацію та після цього йому надається доступ до головної сторінки.

Результати

Після завершення логіну, користувач інтерфейсу адміністтратора може передивитися статистику по кількості скарг за останній місяць за кожен день, кількість замовлень за останній місяць за кожен день, та кількість скарг за статусами, кількість замовлень за статусами.

3.1.7.8 Вхід користувача до інтерфейсу адміністратора (система модерації)

Вступ

Користувач інтерфейсу адміністратора повинен увійти у свій обліковий запис.

Вхідні дані

Користувач інтерфейсу адміністратора повинен перейти на сторінку логіну та ввести пошту та пароль.

Обробка

Користувач інтерфейсу адміністратора перевіряється на аунтифікацію та після цього йому надається доступ до головної сторінки.

Результати

Після завершення логіну, користувач інтерфейсу адміністтратор має доступ до сторінок відповідно до своєї ролі у системі.

3.1.7.9 Перегляд статистики на сторінці статистики (система модерації)

Вступ

Користувач інтерфейсу адміністратора переглядає статистику.

Вхідні дані

Користувач інтерфейсу адміністратора повинен бути аутентифікований у системі та перейти на сторінку статистики.

Обробка

Користувач інтерфейсу адміністратора перевіряється на аунтифікацію та після цього йому надається доступ до сторінки статистики

Результати

Після завершення перевірки аутентифікації, користувач інтерфейсу адміністратор має доступ до сторінки статистики де може побачити статистику по скаргам та замовленням.

3.1.7.10 Зміна налаштувань замовлення (система модерації)

Вступ

Користувач інтерфейсу адміністратора за проханням клієнта змінює налаштування замовлення клієнта.

Вхідні дані

Користувач інтерфейсу адміністратора отримує прохання від клієнта та переходить на сторінку замовлення де натискає на потрібне замовлення.

Обробка

Користувач інтерфейсу адміністратора домовляється з користувачем та змінює відповідно до потреби користувача замовлення.

Результати

Після завершення, користувач інтерфейсу адміністратора бачить успішно змінене замовлення та повідомляє про це користувачу.

3.1.7.11 Видалення замовлення (система модерації)

Вступ

Користувач інтерфейсу адміністратора за певних технічних обставин видаляє замовлення.

Вхідні дані

Користувач інтерфейсу адміністратора отримує прохання від клієнта, або із-за технічних причин повинен перейти на сторінку замовлення де натискає у потрібному замовленні кнопку видалили.

Обробка

Відправляється запит на сервер, після цього потрібне замовлення знаходиться і видаляється з бази даних та повертає результат видалення.

Результат

Після завершення, користувач інтерфейсу адміністратора бачить успішне видалення замовлення та повідомляє про це користувачу.

3.1.7.12 Видалення скарг (система модерації)

Вступ

Користувач інтерфейсу адміністратора за певних технічних обставин видаляє скаргу.

Вхідні дані

Користувач інтерфейсу адміністратора отримує прохання від клієнта, або із-за технічних причин повинен перейти на сторінку скарг де натискає у потрібній скарзі кнопку видалили.

Обробка

Відправляється запит на сервер, після цього потрібна скарга знаходиться і видаляється з бази даних та повертає результат видалення.

Результат

Після завершення, користувач інтерфейсу адміністратора бачить успішне видалення скарги та повідомляє про це користувачу.

3.1.7.13 Переглядь інформації про користувача інтерфейсу адміністратора (система модерації)

Вступ

Користувач інтерфейсу адміністратора хоче подивитися інформацію про себе.

Вхідні дані

Користувач інтерфейсу адміністратора хоче подивитися інформацію про себе.

Обробка

Відправляється запит на сервер, після цього інформація за своїм профілем надсилається у відповідь.

Результат

Після завершення, користувач інтерфейсу адміністратора бачить інформацію про себе.

3.1.8 Припущення й залежності

При розробці системи вантажних перевезень ми робимо наступні припущення:

користувачі мають доступ до мобільного інтернету та сумісних пристроїв на Android/IOS для взаємодії з нашим додатком;

водії мають належні права та документи для здійснення вантажних перевезень, а також транспортні засоби, що відповідають вимогам безпеки та технічного стану;

компанії, що займаються вантажними перевезеннями, мають необхідні ліцензії та дозволи на надання послуг з перевезення вантажів;

водії зобов'язані дотримуватися правил дорожнього руху та інших нормативних вимог під час виконання замовлень на перевезення вантажів;

геолокаційні дані, що використовуються в додатку, надають точну та актуальну інформацію про місцезнаходження користувачів та водіїв, в тому числі під час виконання замовлення.

система оплати в додатку функціонує належним чином, забезпечуючи безпечну та зручну оплату послуг з перевезення вантажів;

Залежності системи:

для належної роботи системи необхідна взаємодія з зовнішніми сервісами та інтерфейсами, зокрема картографічними сервісами (Google Maps API, Google Street API), геолокаційними сервісами, системами оплати (Google Pay) тощо;

система залежить від стабільної роботи мережі Інтернет, як для користувачів, так і для водіїв. Коректність відображення актуального місцезнаходження вантажу залежить від якості підключення до мобільної мережі;

для забезпечення безпеки та якості послуг система залежить від дотримання користувачами та водіями правил дорожнього руху та інших нормативних вимог;

система залежить від рівня довіри між користувачами та водіями, а також від якості наданих послуг з перевезення вантажів;

система залежить від актуальності та повноти інформації про вантажі, маршрути та інші параметри, що вводяться користувачами та водіями;

система залежить від ефективної взаємодії між користувачами, водіями, адміністраторами системи та іншими учасниками процесу перевезення вантажів.

3.2 ФУНКЦІОНАЛЬНІ ВИМОГИ

3.2.1 FR-1 Реєстрація, авторизація

3.2.1.1 Вступ

Користувач повинен мати право авторизуватися через Google або зареєструвати обліковий запис через пошту для подальшої авторизації в додатку під своїм обліковим записом.

3.2.1.2 Вхідні дані

Користувач вводить свої дані у форму авторизації/реєстрації та натискає кнопку підтвердження. Або авторизується через обліковий запис Google

3.2.1.3 Обробка

Користувач вводить свої дані у форму авторизації/реєстрації та натискає кнопку підтвердження.

3.2.1.4 Результати

У разі правильних надісланих даних користувача перенаправляє на головну сторінку мобільного додатку, в іншому випадку отримує повідомлення про помилку.

3.2.1.5 Обробка помилок

Користувач не зможе зареєструватися/увійти у разі будь-якої помилки.

3.2.2 FR-2 Створення замовлення

3.2.2.1 Вступ

Основний процес створення замовлення зі сторони клієнта.

3.2.2.2 Вхідні дані

Користувач вводить інформацію про вантаж, його тип, габарити, місце призначення та час відправлення, зазначає необхідність у допомозі з завантаження чи розвантаження.

3.2.2.3 Обробка

Замовлення перевіряється на валідність, оцінюється можливість його виконання, визначається його ціна та розпочинається підбір водіїв, що можуть виконати це замовлення.

3.2.2.4 Результати

У разі коректного створення замовлення воно валідується та переходить на етап пошуку водія, що зможе виконати замовлення. Також користувач отримує інформацію про орієнтовну вартість замовлення.

3.2.2.5 Обробка помилок

У разі помилки створення замовлення користувач отримує інформацію про причину помилки та контактами служби підтримки, для уточнення деталей.

3.2.3 FR-3 Оплата замовлення

3.2.3.1 Вступ

Клієнт повинен оплатити замовлення для того, щоб водій розпочав його виконання.

3.2.3.2 Вхідні дані

Водій підтвердив замовлення, створене клієнтом. Клієнт отримав сповіщення про це та кінцеву варітсть перевезення.

3.2.3.3 Обробка

Сторонній сервіс проводить оплату та передає результат оплати серверу. Сервер перевіряє, чи справді сторонній сервіс провів оплату, чи запит виконаний зловмисником.

3.2.3.4 Результати

У разі успішної оплати водій сповіщується про це та може виконувати замовлення.

3.2.3.5 Обробка помилок

Користувач має змогу провести оплату кілька разів. Після чого замовлення буде відхилено або створено квиток у службу підтримки.

3.2.4 FR-4 Перегляд статусу замовлення в режимі реального часу

3.2.4.1 Вступ

Клієнт має можливість переглянути поточне місцезнаходження.

3.2.4.2 Вхідні дані

Підтверджене замовлення, що розпочало своє виконання. Водій підтвердив відправку до місця призначення. Водій передає своє місцезнаходження за допомогою увімкненого на мобільному пристрої GPS.

3.2.4.3 Обробка

Перевірка того, що GPS збігається з визначеним маршрутом перевезення та збереження поточного місцезнаходження, оцінка очікуваного часу прибуття.

3.2.4.4 Результати

Клієнт переглядає місцезнаходження вантажу та очікуваний час прибуття.

3.2.4.5 Обробка помилок

В разі помилки отримання місцезнаходження впродовж виначеного часу модератор служби підтримки зв’язується із водієм.

3.2.5 FR-5 Підбір замовлення для водія

3.2.5.1 Вступ

Система підбирає замовлення для водія та пропонує прийняти його.

3.2.5.2 Вхідні дані

Водій вказує свої налаштування - радіус пошуку, або запланований маршрут, можливість бути вантажником, а також поточний GPS (надає дозвіл додатку на передачу).

3.2.5.3 Обробка

Система підбирає замовлення, що відповідають вказаним у налаштуваннях критеріях, та пропонує перелік підібраних замовлень.

3.2.5.4 Результати

Водій бачить список замовлень та приймає їх, або відхиляє. В разі відхилення, замовлення більше не з’являється у списку та передається наступному водієві.

3.2.5.5 Обробка помилок

В разі помилок з отриманням замовлень водієві буде запропоновано звернутись до служби підтримки застосунку.

3.2.6 FR-6 Звернення до служби підтримки

3.2.6.1 Вступ

Клієнт або водій має можливість звернутись до служби підтримки.

3.2.6.2 Вхідні дані

Текст звернення до служби підтримки.

3.2.6.3 Обробка

Система отримує звернення, котре було зареєстроване через телеграм - бота, ті віддає його команді технічної підтримки.

3.2.6.4 Результати

Модератор або клієнт закриває звернення тоді, коли клієнт чи водій задоволені отриманою відповіддю та допомогою.

3.2.6.5 Обробка помилок

В разі виникнення помилок, модератор сповіщує команду розробки про проблему під час звернення.

3.2.7 FR-7 Перегляд статистики системи

3.2.7.1 Вступ

Адміністратор системи може переглядати статистику стосовно звернень та замовлень.

3.2.7.2 Вхідні дані

Відкриття сторінки статистики, з відповідним доступом

3.2.7.3 Обробка

Система перевіряє, чи має користувач достатньо прав для перегляду статистики, і якщо користувач має відповідні права, то віддає статистичні дані.

3.2.7.4 Результати

Відкриття сторінки з статистикою.

3.2.7.5 Обробка помилок

У разі виникнення помилок, користувачу буде відображене відповідне вікно, з тим причиною, чому саме він бачить цю помилку

3.2.8 FR-8 Модерація системи

3.2.8.1 Вступ

Адміністратор системи або модератор має право, у разі отримання запиту від користувача внести зміни у замовлення, виправити помилку в акаунті користувача, водія.

3.2.8.2 Вхідні дані

Відкриття сторінки з користувачами або замовленнями

3.2.83 Обробка

Система перевіряє, чи має користувач достатньо прав для перегляду відповідної сторінки, та після процесу аутентифікації пропускає користувача далі.

3.2.8.4 Результати

Відкриття сторінки з користувачами системи або замовленнями

3.2.8.5 Обробка помилок

У разі виникнення помилок, користувачу буде відображене відповідне вікно, з тим причиною, чому саме він бачить цю помилку

3.2.8 FR-8 Детальний пошук для адміністраторів системи

3.2.8.1 Вступ

Адміністратор системи або модератор має право, у разі отримання запиту скористатися детальним пошуком у системі, з усіма можливими параметрами, для того, щоб знайти відповідне замовлення, або користувача/водія.

3.2.8.2 Вхідні дані

Відкриття сторінки з користувачами або замовленнями

3.2.83 Обробка

Система перевіряє, чи має користувач достатньо прав для перегляду відповідної сторінки, та після процесу аутентифікації пропускає користувача далі.

3.2.7.4 Результати

Відкриття сторінки з користувачами системи або замовленнями

3.2.7.5 Обробка помилок

У разі виникнення помилок, користувачу буде відображене відповідне вікно, з тим причиною, чому саме він бачить цю помилку

3.3.1 Надійність

Надійність програмного забезпечення є критичною вимогою, оскільки система займається координацією та управлінням логістичними процесами, що

впливають на щоденні операції користувачів і бізнесів. Для забезпечення високого рівня надійності програмного продукту, наступні критерії повинні бути виконані:

Відновлення після збоїв: Система повинна мати можливість автоматично відновлювати роботу після непередбачених збоїв. Це включає реалізацію стратегій на кшталт автоматичного перезапуску сервісів та використання транзакційної пам'яті для забезпечення цілісності даних.

Резервне копіювання даних: Періодичне резервне копіювання даних має бути автоматизовано. Система має забезпечувати легке відновлення даних з резервних копій у випадку їх втрати або пошкодження. Дозволяється налаштувати автоматичне резервне копіювання обраної СКБД.

Висока доступність: Система повинна бути розроблена з використанням архітектурних патернів, що підтримують високу доступність, включаючи класичний поділ на кластери, балансування навантаження та відмовостійкість.

Швидке відновлення: Мінімізація часу відновлення системи після збою є обов'язковою. Система повинна мати здатність швидко відновлювати операції без значної втрати даних.

Моніторинг стану системи: Необхідно імплементувати інструменти для постійного моніторингу стану всіх компонентів системи, щоб забезпечити раннє виявлення потенційних проблем та уникнення непередбачених збоїв. Зокрема, за допомогою збору логів та метрик з Logstash до Elasticsearch.

Уніфікація середовища: для уникнення додаткових проблем під час розгортання компонентів системи необхідно створити проект, кожен елемент якого (сервіс) працює в Docker-контейнері, що працює однаково в оточенні розробника та production-оточенні за умови наявності однакових змінних середовища.

Ці вимоги до надійності забезпечують, що система може функціонувати ефективно і безперебійно, максимізуючи задоволеність користувачів і надійність бізнес-процесів.

3.3.2 Доступність

Система Wide Delivery має бути доступною для користувачів протягом більшої частини доби, враховуючи особливості ринку вантажних перевезень. Доступ до функціоналу платформи, такого як створення замовлень, пошук водіїв, відстеження доставки та спілкування з службою підтримки, буде забезпечено в проміжку часу з 6:30 до 24:00. Цей часовий діапазон обрано з урахуванням типових годин роботи більшості транспортних компаній та індивідуальних перевізників, а також потреб клієнтів, які, як правило, здійснюють замовлення на перевезення вантажів в денний та вечірній час.

Для забезпечення безпеки та конфіденційності даних, доступ до функціоналу Wide Delivery буде надано виключно авторизованим користувачам. Кожен користувач, незалежно від того, чи це клієнт, водій або представник транспортної компанії, повинен буде пройти процедуру реєстрації та створити особистий обліковий запис. Для входу в систему буде використана безпечна система авторизації, яка захистить облікові записи користувачів від несанкціонованого доступу.

3.3.4 Супроводжуваність

Основна мета: Забезпечити легкість та ефективність обслуговування, оновлення, налаштування та розширення програмного продукту протягом усього життєвого циклу.

Опис атрибутів супроводжуваності:

Модульність: Система розроблена з використанням чітко визначених модулів, що дозволяє легко замінювати або оновлювати окремі компоненти без впливу на решту системи.

Читабельність коду: Програмний код має бути написаний з дотриманням стандартів кодування та коментування, що забезпечує його легкість для розуміння та подальшого супроводження розробниками.

Документація: Повна технічна документація системи має бути надана разом з продуктом. Документація повинна включати керівництва для розробників та

користувачів, опис архітектури системи, а також процедури оновлення та вирішення проблем.

Логування та моніторинг: Система повинна включати розширені можливості логування та моніторингу, що дозволяє легко відслідковувати роботу програми та швидко виявляти та усувати помилки.

Інтернаціоналізація: Програмний продукт має підтримувати локалізацію для різних мов та регіональних налаштувань, що дозволяє адаптувати продукт для різних міжнародних ринків.

Підтримка версій: Всі оновлення програмного забезпечення мають супроводжуватись чітким веденням версій, що дозволяє користувачам легко переходити між версіями та відновлювати попередні конфігурації при необхідності.

Зауваження: Висока супроводжуваність забезпечує нижчі загальні витрати на володіння продуктом та збільшує задоволеність користувачів, що є критично важливим для довгострокового успіху програмного продукту.

3.3.5 Переносимість

Проект має бути налаштований для роботи у контейнеризованому середовищі за допомогою Docker-compose, що дозволяє забезпечити легке та швидке розгортання на різних платформах: Ubuntu, Windows, і MacOS. Розгортання повинно відбуватися за допомогою однієї команди (`make start`), що спрощує процес ініціації та управління.

Конфігурація повинна включати всі необхідні сервіси та залежності, які описуються в docker-compose.yml файлі для кожного з репозиторіїв. Усі репозиторії (окремі сервіси) мають бути поєднаними в один проект. Додатково в кожному з репозиторіїв потрібно створити Makefile, що містить команду make start, яка автоматизує процес розгортання контейнерів та залежностей.

Таким чином гарантується ідентичність середовища та легкий запуск з будь-якої платформи розробки чи розгортання.

3.3.6 Продуктивність

Система Wide Delivery повинна демонструвати високу продуктивність та забезпечувати швидку реакцію на дії користувачів, навіть за умов значного навантаження. Мобільний додаток повинен бути оптимізований для роботи на пристроях з різними характеристиками та забезпечувати плавну роботу інтерфейсу без помітних затримок. Зокрема, головний екран додатку, що містить інтерактивну карту та форми для введення даних, повинен завантажуватись менше ніж за 2 секунди. Це забезпечить позитивний користувацький досвід та дозволить клієнтам швидко розпочати процес замовлення перевезення.

Серверна частина Wide Delivery повинна бути спроможна обробляти запити від великої кількості користувачів одночасно, забезпечуючи стабільну роботу платформи навіть у періоди пікового навантаження. Система має бути спроектована з урахуванням можливості масштабування, що дозволить збільшувати її потужність паралельно зі зростанням кількості користувачів та замовлень.

3.4 Вимоги бази даних

Необхідно обрати NoSQL базу даних, що відповідає PA/EC рішенню за теоремою PACELC. Висока доступність та консистентність даних є важливими вимогами до системи. База даних має легко масштабуватись та легко піддаватись партиціонуванню. Чудовим кандидатом для такої БД є MongoDB.

Також систему необхідно забезпечити сховищем кешованих даних для швидкого доступу до потрібної інформації. Зокрема, такою інформацією є сесії користувачів, що виконали аутентифікацію та використовують мобільний застосунок, а також дані для роботи чат-боту служби підтримки в Telegram.

Логи та метрики, зокрема щодо запитів користувачів, повинні зберігатись у пошуковому сервері Elasticsearch задля можливості швидкого доступу до напівструктурованої інформації.

3.5 Інші вимоги

Кожна зміна документу має бути обговорена командою розробників і власником проекту. Після внесення цих змін - підписана кожним із них.