МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

КОРНІЛЕНКО ОЛЕКСАНДР СЕРГІЙОВИЧ

Допускається до захисту:
завідувач кафедри
інформаційних технологій,
к. т. н., доцент
_____Т.В. Нескородєва
«_____» _____ 2021 р.

АВТОМАТИЗОВАНА СИСТЕМА ДЛЯ КОЛЕКТИВНОГО ЗАМОВЛЕННЯ ТАКСІ

Спеціальність 122 Комп'ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:

Штовба С.Д., професор кафедри
інформаційних технологій,
д.т.н, професор.

Оцінка: _____/_____/_____
(бали за шкалою ЄКТС / за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2021

Корніленко О.С. Розробка автоматизованої системи колективного замовлення таксі. Спеціальність 122 «Комп'ютерні науки», освітня програма «Сучасні інформаційні технології та програмування». Донецький національний університет імені Василя Стуса, Вінниця, 2021.

У кваліфікаційній роботі проаналізовано предметну область та порівняно поширені системи таксі в Україні. Розроблено автоматизовану систему з веб-інтерфейсом для замовлення персонального та колективного таксі. Система має такі переваги: пасажири за одну поїдку платять менше, водії отримують більше. Використання колективного таксі призводить до зменшення заторів та шкідливих викидів в атмосферу.

Ключові слова: автоматизована система, Java, Cuba platform, MySql, Rest API, колективне замовлення таксі.

61 с., 3 табл., 20 рис., 40 джерел.

Kornilenko O. Automated system for collective taxi ordering. Specialty 122 "Computer Science", educational program "Modern Information Technologies and Programming". Vasyl' Stus Donetsk National University, Vinnytsia, 2021.

The qualification work analyzes the subject area and relatively common taxi systems in Ukraine. An automated system with a web interface for ordering personal and collective taxis has been developed. The system has the following advantages: passengers pay less for one trip, drivers get more. The use of a collective taxi reduces congestion and harmful emissions into the atmosphere.

Keywords: automated system, Java, Cuba platform, MySql, Rest API, collective taxi order.

61 Pages, 3 Tables, 20 Fig., 40 Ref.

ЗМІСТ

Вступ

Актуальність роботи. На сьогоднішній день послуги таксі користуються величезною популярністю серед всіх категорій населення. За допомогою даного сервісу можна прибути в місце призначення швидко та з комфортом. В часи пандемії місце в громадському транспорті обмежене, та не завжди є можливість ним скористуватись, тому таксі стає все популярнішим засобом пересування. Таксі – це невід'ємний атрибут великих і малих міст. Особливо зручно користуватись послугами таксі, коли потрібно взяти з собою досить громіздкий багаж, наприклад коли потрібно дістатись до вокзалу чи аеропорту, проїхати з таким багажем в громадському транспорті виявиться досить проблематичним. Використовуючи попереднє замовлення таксі. Можна бути впевненим в тому, що машина прибуде в строго призначений час, що виключить можливість запізнення на рейс.

Проте одним з мінусів таксі є його дороговизна, досить велика кількість людей не можуть дозволити собі таксі послуги. Одним з варіантів вирішення даної проблеми це послуги колективного таксі. Тобто користувачі можуть заздалегідь замовити дану послугу, і якщо декілька людей планують їхати в одне місце(при чому пасажири можуть знаходитись в різних куточках міста), водій таксі заїде за всіма пасажирами та доставить до місця призначення. Перевагами такої послуги є те, що за одну поїздку пасажир платить менше, оскільки їде не сам, а з декількома іншими пасажирами. Водій в свою чергу отримує плату з кожного пасажира, завдяки чому за одну поїздку отримує більше. Також, якщо більшість людей будуть користуватись саме колективним таксі, то кількість автомобілів на дорозі зменшиться, що призведе до зменшення заторів та викидів шкідливих речовин в атмосферу. В Україні немає такої послуги, тому є необхідність створення такої системи.

Мета роботи. Розробити автоматизовану систему, яка забезпечить можливість онлайнового замовлення колективних поїздок з різних локацій з мінімізацією загальних витрат часу на одну поїздку.

Для досягнення мети вирішуються такі завдання:

- аналіз предметної області;
- аналіз поширених програмних продукті виклику таксі;
- розробка архітектури автоматизованої системи колективного замовлення таксі та вибір інструментарію для її реалізації;
- розробка алгоритмів та програмна реалізація автоматизованої системи колективного замовлення таксі;
- розробка документація та тестування автоматизованої системи колективного замовлення таксі.

Об’єкт дослідження. Автоматизація колективного замовлення таксі.

Предмет дослідження. Алгоритми, програми та бази даних даних для реалізації колективного замовлення таксі.

Методи дослідження: системний аналіз, дослідження операцій, дискретна оптимізація, метаевристичні технології, програмування.

Практичне значення результатів. Розроблена автоматизована система, що є інформаційною основою створення нового сервісу з надання послуг таксі, який за рахунок колективного замовлення дозволяє: 1) зменшити ціну поїздки для одного пасажира; 2) збільшити кількість клієнтів таксі; 3) зменшити завантаженість транспортної мережі міста та покращити екологічний рівень.

Структура роботи. Бакалаврська включає в себе 61 сторінку, 20 рисунків, 2 таблиці і список літератури з 40 джерел.

Розділ 1.
Аналіз предметної області

1.1 Аналіз та характеристика об’єкту дослідження

Базуючись на [5] наведемо відомості про таксі. Таксі - легковий автомобіль, обладнаний розпізнавальним ліхтарем оранжевого кольору, який встановлюється на даху автомобіля, діючим таксометром, сигнальним ліхтарем із зеленим та червоним світлом, розташованим у верхньому правому кутку лобового скла, і який має нанесені композиції з квадратів, розташованих у шаховому порядку на дверцятах автомобіля з лівого та правого боків, призначений для надання послуг з перевезення пасажирів та їхнього багажу в індивідуальному порядку.

Базуючись на [5] наведемо відомості про таксометр. Таксометр - прилад, призначений для інформування пасажирів про вартість поїздки та реєстрації параметрів роботи автомобіля-таксі, визначених законодавством.

Перевезення пасажирів на таксі – це перевезення пасажирів на таксі в межах України, часто в межах одного населеного пункту. Послуги з перевезення на таксі надаються громадянам у порядку черги на стоянках таксі та на шляху прямування, а також на замовлення (звичайне або термінове, усне, письмове чи за телефоном) [5].

У сучасному світі сфера таксі є досить популярною та актуальною. Громадський транспорт не завжди є найкращим вибором при поїздці у пункт призначення, а також не завжди задовольняє потреби окремих жителів міста. Власний транспортний засіб є не у кожної людини, а також потребує великих грошових вкладень та догляду за автомобілем. Тому безліч людей користуються послугами таксі щодня, тому що це зручно та комфортно. Данну послугу можна замовити за допомогою мобільного застосування, веб-застосування або за телефоном.

Зазвичай при замовленні таксі за телефоном, користувач зв’язується з диспетчером, дану процедуру можна описати таким алгоритмом:

1. Людина, яка хоче скористатись даною послугою дзвонить по номеру.
2. Диспетчер call-центру відповідає на дзвінок та отримує інформацію, щодо місця посадки та висадки.
3. Замовлення формується на розсилається усім водіям на зміні.
4. Вільний водій приймає замовлення та сповіщає про це диспетчера, який в свою чергу повідомляє клієнта sms-повідомленням, або дзвінком.
5. Водій виконує замовлення, після чого доповідає диспетчеру.

Проте останнім часом замовленням таксі користується все менше людей, тому що це є не досить зручним через такі проблеми:

- у користувача може не бути потрібної суми на рахунку, щоб здійснити дзвінок;
- проблеми зі мобільним зв’язком, у зв’язку з чим диспетчер може почути не правильну інформацію, щодо адреси посадки чи висадки;
- в деяких людей є вади слуху, тому вони фізично не можуть замовити послугу;
- іноді лінії зв’язку перевантажені та потрібно досить довго очікувати відповідь диспетчера.

Більш зручний спосіб скористатись послугою таксі – це мобільний додаток. Проте це також буває не досить зручно, тому що мобільні програми займають місце на пристрої, що є досить критичним для деяких користувачів. Також вони потребують постійного оновлення та зазвичай «прив’язують» аккаунт до пристрою.

Веб-застосування є досить зручним у використанні, адже виключає мінуси виклику таксі, які наведені вище. Метою розробки є автоматизована саме система, тому для даною роботи підійде алгоритм виклику таксі, який виключає роботу диспетчерів, а саме:

- Користувач, який хоче скористатись послугою, заходить на сайт та на карті, або в спеціально визначених місцях, вказує адресу посадки та висадки, при цьому система автоматично має рахувати приблизну вартість поїздки;
- Водії відслідковують всі замовлення на спеціальній сторінці, можуть вибрати серед декількох замовлень найбільш вигідне на їх погляд та прийняти його;
- Водій виконує поїздку, на спеціальній сторінці відмічає замовлення як виконане та отримує оплату.

Таким чином потрібно створити веб-додаток, який:

- виключає роботу диспетчерів;
- не потребує встановлення окремого додатку;
- може використовуватись з любого пристрою;
- має бути зручним та простим у використанні;

1.2 Огляд і порівняння поширених систем таксі

Сфера таксі є досить розвинутою в Україні, є досить багато компаній, які конкурують між собою та пропонують скористатись даною послугою онлайн, серед них:

Uber – компанія, яка одна з перших дозволила замовляти таксі онлайн. Застосунок доступний у більш ніж 67 країнах світу, втому числі і в Україні. Данна компанію пропонує замовлення таксі за допомогою мобільного додатку, за телефоном або через веб-додаток. [1].

Рисунок 1.1 - Логотип Uber

З плюсів можна виділити:

- зручний мобільний додаток;
- можливість відслідковувати автомобіль до зазначеного місця;
- розповсюдженість;
- можливість послуги оплати онлайн.

З мінусів можна виділити:

- Досить важко замовити послугу через сайт;
- Сервіс не провіряє водіїв та не відповідає за них;
- Немає можливості викликати таксі колективно.

Uklon – служба замовлення таксі за вимогою за допомогою інтернету. Заснована у 2010 році та фактично була одною з перших популярних служб на ринку України. Також є можливість замовити приватне авто. Один з переваг Uklon є замовлення таксі без участі диспетчера, за допомогою мобільно та веб-додатку. Uklon не є повноцінною службою таксі — це агрегатор замовлень, який використовується для розподілу дзвінків між водієм таксі та службою [2].

Рисунок 1.2 - Логоти Uklon

Bolt – служба таксі, яка вперше з'явилась в Естонії, надає послуги замовлення таксі, електричних скутерів, мотоциклів та приватних автомобілів за допомогою однойменного мобільного додатку [3]. Можна встановити два окремих додатка: для водіїв та для пасажирів. Замовити таксі можна лише через мобільний додаток.

Рисунок 1.3 - Логотип Bolt

Taxi 838 – Українська служба замовлення таксі, досить довгий час замовити послугу можна було лише за телефоном та лише в 2019 році сервіс створив свій власний мобільний додаток. Має високий попит у користувачів за рахунок дешевих послуг та швидкості реагування на замовлення [4].

Рисунок 1.4 - Логотип TAXI838

Існує також багато інших популятрних сервісів, які дозволяють замовити послуги таксі різнипи способами, проте в Україні немає сервісу, який надасть замовити таксі колективно. Також досить часто в таких сервісах після того як користувач залишив заявку, диспетчер або водій телефеноує для уточнення даних, що іноді буває не досить зручно.

Потрібно наголосити, що веб-сервіс буде сторений повністю автоматизованим, тобто буде виключати участь диспетчера, а ткож надавати можливість замовити таксі колекитвно.

Розглянувши вище перелічені системи можна зробити наступні висновки: на данний момент існує досить багато аналогів розробки, в кожної є свої переваги та недоліки, проте жоден з сервісів не надає можливість викликати таксі колективно. Також більшість онлайн-сервісів є досить незручними у використанні та лише імітують автоматизовану систему. Більшість людей користуюсь мобільними додатками, проте є частина людей яким це незручно, що доводить доцільнісь та актуальність розробки даного веб-додатку.

1.3 Постановка задачі

Ретельно проаналізувавши предметну область та оглянувши аналоги які є на ригку, можна сформувати завдання, які необхідно виконати, та вигоди до системи, яка розробляється.

Потрібно створити веб-додаток, інтерфейс якого буде простим у застосуванні та зрозумілим необізнаному користувачу. При чому має забезпечувати реєстраці та авторицзацію користувачів до акаунуту з любого пристрою.

Система має передбачати 2 типи користувачів: водій та клієнт, кожен тип буде виконувати різні задачі. Клієт має оформляти заявку на послугу, водій в свою чергу повинен відповідати на запит клієнта та виконувати замовлення.

Також додаток повинен реалізовувати освноні завдання:

- Клієнт повинен мати можливість швидко і зручно замовити таксі без участі диспетчнра, а водій приймати замовлення клієтнта без участі диспетчера та звершувати його.

Варто зазначити що важливою складовою проекту є можливість замовлення таксі колективно, тобто якщо декільком користувачам потрібно дістатись до певного місця, вони можуть заздалегіть замовити послугу колективного таксі на певний час(при чому користувачі можуть знаходитись у різних точках міста), кількість пасажирів в даному випадку буде визначатись кількістю сидячих місць в автомобілі, система автоматично буде знаходити такий набір маршрутів, щоб заїхати за кожним пасажиром та доставити їх до місця призначення, при цьому мінімізувати відстань, яку проїжджає автомобіль за всю поїздку для економії палива.

Також веб-застосування повинне надавати доступ як зараєєстрованим так і незареєстрованим користувачам, просте в цьому випадку буде відображатись мінімальна іфнормація, а саме місце посадки, місце висадки та номер телефону.

Система повина надавати такі можливості:

- реєстрація та авторизація користувачів, а саме клієнтів та водіїв; При реєстрації водіїв потрібно зазначати персональні данні і данні про автомобіль на якому будуть здійснюватись перевезення.
- можливість редагування персональної інформації на сайні в особистому кабінеті користувача;
- клієнт повинен мати можливість бачити профіть водія, який буде виконувати замовлення;
- водій повинен мати можливість бачити профіть клієнта, який буде замовляти послугу;
- зручний та адаптивний веб-інтерфейс;
- можливість зазначати місце посадки на висдки на карті;
- зручну вкладку для водіїв, де вони зможуть вибирати та приймати замовлення;
- можливість замовлення як персонального так і колективного таксі;
- ціна за маршрут повинна показуватися з одразу як встановлена початкова та кінцева точка маршруту;

Веб-додаток повинен мати зручну та інтуїтивну навігацію, виважений стиль та єдину кольорову гаму та не повинен вимагати постійної підтримки зі сторонои програміста.

Отже зважаючи на вимоги та проведений аналіз області, варто також розробити такі інформаційні сторінки:

- «Про компанію» - де буде міститись основна інформація про веб-додаток, його переваги та короткий опис;
- «Допомога» - де можна буде задати будь-яке питання фахівцю, щодо користування додатком, а також буде реалізована підтримка користувачів;
- «Контакти» - де буде міститись контактна інформація, дані про компанію, адреса, телефон, електронна адреса.
- зручне відкно аавторизаці на сайті;

- сторінка з правилами для клієнтів та інших відвідуваів сайту;
- сторінка з правилами сервісу для користувачів які хочуть стати водіями, де будуть вимоги, щодо перевезень, технічних характеристик автомобіля(дозволені марки, роки виготовлення та ін.);
- «Особистий кабінет» - тільки для авторизовних користувачів, де буде відображатись персональна інформація, а також потрібно реалізувати можливість змінни цих даних.

Висновок за розділом 1

Отже послуг колективного таксі, не реалізовано в популярних службах таксі в Україні. Багато служб мають незручний інтерфейс та незадовільняють певних користувачів. Варно зазначити, що актуальність теми тільки зросла через пандемію короно вірусу, адже громадський транспорт може перевозити лише обмежену кількість пасажирів. Значною перевагою колективного таксі є значно нижча вартість відносно персонального таксі, що дозволить користуватись частіше користуватись данними послугами.

Розділ 2
Аналітична частина

2.1 Технології для реалізації системи

2.1.1 Поняття фреймворку

Базуючись на [6] наведемо відомості про Фреймворк. Фреймворк (Framework, каркас, платформа, структура, інфраструктура) – програмна платформа, що визначає структуру програмної системи або інфраструктура програмних рішень, що полегшує розробку складних систем. У програмуванні програмна платформа – це абстракція, де програмне забезпечення, яке забезпечує загальні функції, може вибірково модифікуватись за допомогою додаткового написання коду. Фреймворк – це універсальне середовище програмування, яке забезпечує певні функції, як частина більшої програмної платформи для полегшення розробки програмних продуктів і рішень. Фреймворк може мати у своєму складі підтримуючі програми, компілятори, бібліотеки коду, набори інтерфейсів та інтерфейси прикладного програмування(API), які поєднують в собі різні компоненти, які забезпечують розробку програмного проекту або системи.

На відмінну від бібліотек фреймворки відрізняються за такими основними рисами:

- інверсія контролю – у фреймворку потік загальної програми управління викликаний не користувачем, а безпосередньо фреймворком порівняно з бібліотеками або користувальницькими програмами;
- розширюваність – шляхом вибіркового охоплення користувачі можуть розшити структуру або програмісти для надання певних функцій можуть додавати власні коди користувачів;
- немодифікований код фреймворку – як правило, розширення, реалізовані користувачем не повинні прийматись для модифікації коду.

Тобто користувачі фреймворку можуть його розширювати, не змінюючи його коду; він спрощує процес розробки складних високотехнологічних проектів. Для виконання таких задач фреймворк надає такі можливості:

- мовна сумісність – програми, які створені різними мовами програмування, можуть використовувати сегменти коду один в одного. Більше того програми будуть «розуміти» один одного і активно взаємодіяти, якщо вони написані на різних мовах програмування.
- сумісність з операційною системою – користувачі можуть працювати в будь-якій з існуючих операційних систем, яка в них встановлена;
- призначений інтерфейс є універсальним для кожного користувача;
- стійкість коду – платформа реалізує ряд дуже вадливих механізмів, які активно гарантують безпеку додатків;
- код багаторазово використовується та сумісний;
- кросплатформність – завдяки фреймворку можна створювати програми, які працюють на різних програмних платформах.[6].

Кожен фреймворк призначений для вирішення певних завдань. Очевидно, що кожен фреймворк має свої особливості використання, які слід знати та розуміти перед початком розробки будь-якого проекту.

2.1.2 Різновиди фреймворків

Великій кількості фреймворків посприяв швидкий розвито технології створення веб-сайтів. Відповідно від використовуваної мови їх можна розділити на кілька категорій.

Популярні python-фреймворки:

Django – один з найпопулярніших фреймворків на мові програмування Phyton. Особливістю даного фреймворку є принцип «don`t repeat yourself(не повторюйся)». Додатки на Django побудовані однією або кількома

програмами, які рекомендується робити віддаленими і підключеними. Це одна з суттєвих відмінностей архітектури від усіх інших. Крім того URL-адреса у Django задається не автоматично зі структури контролера, а чітко налаштовуються за допомогою регулярних виразів.

Django має такі можливості:

- ORM, API з підтримкою транзакцій для доступу до бази даних;
- вбудований з наявними перекладами великою кількістю мов інтерфейс адміністратора;
- реалізований на основі регулярних виразів URL-диспетчер;
- система кешування;
- розширювана шаблонна система з тегами та успадкуванням;
- авторизація та автентифікація з можливістю підключення зовнішніх модулей: LDAP, OpenID та деякі інші;
- для побудови додаткових програм обробки запитів реалізована система фільтрів, як наприклад кешування та стиснення URL – адреси та підтримка анонімних сеансів;
- для роботи з формами створена бібліотека яка надає такі функції успадкування, побудова форми на основі існуючої моделі бази даних;
- доступ через адміністративний додаток до автоматичної документації по тегам шаблонів і моделей даних [7].

Flask – мікрофреймворк, що використовує набір інструментів Werkzeug (бібліотека службових програм WSGI Python) та шаблони Jinja2 для створення веб-додатків з використання мови програмування Python. Фреймворку Flask не потрібні спеціальні інструменти та бібліотеки, тому його називають мікрофреймворком. Він не має налаштувань бази даних, форм перевірки та будь-яких інших існуючих компонентів сторонніх бібліотек, що забезпечують загальні функції. Однак, у Flask реалізована підтримка розширень, які можуть додавати функціональність, ніби це а було реалізовано в самому фреймворку. Для різних технологій аутентифікації і деяких взаємопов’язаних інструментів

для загальної структури, об’єктно-реляційних моделей, валідації форм реалізовані спеціальні розширення. Основна програма оновлюється набагато рідше ніж розширення.

Flask має такі можливості:

- містить відладчик;
- реалізований сервер розробки;
- вбудована для модульного тестування підткска;
- На стороні клієнтка сеансів реалізована підтримка куків;
- заснований на Unicode;

Twisted – це орієнтований мережевий фреймворк, який поширюється за ліцензією MIT(безкоштовне програмне забезпечення), реалізований на мові програмування Python.

Twisted – це фреймворк для розробки онлайн-додатків, призначений для повного розділення між логічними протоколами та фізичними транспортними рівнями. Як правило орієнтуючись на семантику підключення на основі потоку, таку як HTTP або POP3. Безпосередньо перед тим, як інформація передається в екземпляр логічного протоколу відбувається зв’язок між логічним протоколом на транспортним рівнем. Фреймворк спрямований на об’єднання. Через всі протоколи є доступ до всіх функціональних можливостей [9].

Tornado – фреймворк, який не блокується під час запитів та легко розширюється. Tornado є ідеальним для тривалих запитів, веб-сокетів та додатків, яким потрібне довготривале з’єднання з великою кількістю користувачів, оскільки реалізована можливість масштабуватись до сотень тисяч відкритих з’єднань [10].

Pyramid – фреймворк для Phyton код якого є відкритим. Одною з особливостей даного фреймворку є легкість та простота використання для програмістів при створенні веб-додатків.

Основними прицпаци Pyramid при розробці є:

- простота – можливість отримати результат навіть при невеликому розумінні Pyramid, тобто потрібно мати розуміння основних концепцій та немає необхідності використовувати конкретних методик для створення додатку;
- мінімалізм – для створення додатку фреймворк вирішує лише базові проблеми;
- документація – слідуючи з концепції мінімалізму, відсутня документація кожного аспекту Pyramid;
- швидкість – будь-які шаблони або прості генерації відповідей відбувається швидко, тому що фреймворк створений для забезпечення значно швидкого виконання загальних завдань;
- надійність – фреймворк протестований на розроблений консервативно;
- відкритість – Pyramid має відкритий вихідний код та поширюється під безкоштовною ліцензією [11] .

Популярні java-фреймворки:

Spring MVC – фреймворк, головною цілю якого є підтримка архітектури Spring MVC. Для розробки веб-додатків Spring надає готові компоненти, які можна використовувати. В Spring MVC в якості команди або об’єкта може використовуватись будь-який об’єкт. У Spring реалізована дуже гнучке та зручне зв’язування даних та не потрібно використовувати інтерфейси на базові класи які є специфічними [12].

Java Server Faces (JSF) – фреймворк реалізований на мові програмування Java, який служить для розробки веб-додатків. Головною метою є полегшення розробки користувацького інтерфейсу та інших додатків. JSF базується на використанні компонентів на відміну від більшості MVC фреймворків [13].

Google Web Toolkit (GWT) – дозволяє користувачам розробляти Ajax-додатки, є безкоштовним Java фреймворком. За допомогою Google Web Toolkit у користувачів є можливість здійснювати свій вибір засобів Java

розробки для побудови та налагодження Ajax-додатків. Компілятор фреймворку переводить Java додаток в окремі JavaScript файли одразу після завершення розробки додатку. Використовуючи Java коментарі реалізована можливість додавання Java код в JavaScript [13].

CUBA – це фреймворк з відкритим кодом, мета якого - уніфікація розробки корпоративних додатків. Фреймворк поєднує в собі перевірені рішення для побудови інформаційних систем рівня підприємства і високоефективні інструменти розробки, все це значно прискорює процес створення додатків [14].

Фреймворки від Microsoft: ASP.NET (ASP.NET MVC) і Net Framework [15].

ASP.NET – фреймворк створений компанією Майкрософт для створення веб-додатків, має відкритий код. Популярний серед великої кількості користувачів, які створюють додатки та служби для будь-яких пристроїв та операційних систем. APS.NET дозволяє писати код сторінки на мовах C#, C, C++ та іншких, оскільки базується на багатомовних функціях .NET [15].

.NET – платформа, розроблена фірмою Майкрософт, яка використовується для розробки звичайних програм та веб-додатків. Однією з основних ідей є сумісність написаних на різних мовах служб. В .NET реалізований поділ на дві частини: середовище виконання та інструментарій розробки [16].

Популярні JS-фреймворки:

React – популярна декларативна JavaScript бібліотека, створена компанією Facebook для побудування інтерфейсу, є ефективною та гнучкою, що дозволяє легко створювати інтерактивні користувальницькі інтерфейси. Фреймворк не лише підтримує побудову об’єктно-орієнтованих додатків, а й активно заохочує це. Додатки, які створені за допомогою React є довговічними, оскільки розробники фреймворку досить ретельно поставились до розробки функції зворотної сумісності. За останні декілька років

обізнаність користувачів про фреймворк значно зросла. У застосунках React обробляє лише користувацький інтерфейс, що відповідає шаблону модель-вид-контролер(MVC). Завдяки цьому фреймворк з легкістю може використовуватись у поєднанні з іншими JavaScript бібліотеками [17].

Vue.js - це прогресивний фреймворк для створення користувальницьких інтерфейсів, створений Еваном Ю. та іншими 234 ентузіастами і отримав понад 121 000 зірок на GitHub. Він включає доступну кореневу бібліотеку, яка головним чином вирішує проблему рівня презентації, та екосистему додаткових бібліотек, що дозволяє створювати складні та величезні односторінкові додатки (Single-Page Applications). Вимова Vue.js точно така ж, як і слово view, і вона має 4000 зірок GitHub біллше, ніж React. Vue вдається зробити маркетингову прірву: про це чув майже кожен розробник. Можна вважати, що це пов'язано з величезними зусиллями, які Еван та його команда докладали з 2017 року, беручи участь у різних зустрічах та конференціях та організовуючи власні зустрічі. Однак серед розробників все ще існує розрив у знаннях. Щоб подолати цей розрив, необхідно створити більше навчальних матеріалів щодо використання Vue.js у 2021 році [19].

Angular - фреймворк від компанії Google, який отримав майже 44 тисячі зірок на GitHub. Являє собою платформу, яка спрощує збірку додатків в інтернет. У Angular поєднуються декларативні шаблони, впровадження залежності, двостороннє зв'язування даних і кращі практики вирішення проблем розробки. Ця платформа дозволяє збирати додатки для веб, мобільних пристроїв і настільних ПК. У ній пропонується самий зручний і зрозумілий для початківців інтерфейс командного рядка (CLI) і навіть консоль (Console) - клієнт з графічним інтерфейсом. [18]

Redux – бібліотека JavaScript, яка призначена для розробки інтерфейсу користувача, має відкритий код. Бібліотека була створена у 2015 Деном Абрамовим і Ендою Кларком. Redux зберігає дерево об’єктів стану усієї програми в одному сховищі. Завдяки цьому принципі налагодження на перевірку програми здійснювати значно легше. Це також дозволяє

підтримувати стан програми під час процесу розробки, щоб пришвидшити цикл розробки. Redux з легкістю можна використовувати разом з React або іншими бібліотеками JavaScript. [20]

2.1.3 Фреймворк Cuba

У фреймворку Cuba реалізовано досить багато функцій, завдяки яким з ним досить легко та зручно працювати, а саме:

- Платформа CUBA призначена для розробки корпоративних рішень, що характеризуються складною моделлю даних, десятками або сотнями екранів, великою кількістю бізнес-логіки, а також вимогами до контролю прав доступу.
- Широкий набір готової функціональності, розвинені засоби генерації коду, візуальний дизайнер інтерфейсів, а також підтримка hot deploy радикально скорочують час і вартість розробки рішення.
- Платформа повністю побудована на стеку відкритих Java технологій, що дозволяє використовувати напрацювання найбільшої екосистеми вільного ПЗ в світі, а також контролювати вихідний код всього стека. Відкрита архітектура дозволяє перевизначати поведінку більшості механізмів платформи, забезпечуючи високу гнучкість. Розробники можуть використовувати популярні Java IDE і мають повний доступ до вихідного коду.
- Додатки на платформі легко вбудовуються в IT-інфраструктуру завдяки підтримці основних баз даних і серверів додатків, а також можливості роботи в хмарі. Платформа дозволяє забезпечити відмовостійкість і масштабованість рішень, а універсальний REST API надає кошти інтеграції з зовнішніми системами.
- Скорочення (або виключення) витрат замовника на придбання ліцензій у зв'язку із застосуванням вільно розповсюджуваних технологій.

- Крос-платформенність: можливість роботи під будь операційною системою і практично з будь-якою системою управління базами даних (СУБД), в тому числі вільно поширюваними.
- Застосування ефективних засобів мережевої безпеки і розмежування прав доступу користувачів.
- Автоматичний аудит дій користувачів, включаючи історію змін даних.
- Простота інтеграції з зовнішніми додатками.
- Простота і швидкість розробки бізнес-додатків, впровадження та оновлення; можливість поновлення без зупинки системи (для критичних систем, що працюють в цілодобовому режимі).
- Можливість роботи в системі, створеній на платформі, з будь-якої точки світу, де є Інтернет.
- Відкритий програмний код: можливість подальшого розширення додатків власними силами замовника.
- Архітектура додатків на платформі CUBA [14].

2.1.4 Rest API

Базуючись на [26] наведемо відомості про REST. Дизайн REST або RESTful API (Репрезентативна передача стану) призначений для використання переваг існуючих протоколів. Хоча REST можна використовувати майже за будь-яким протоколом, він зазвичай використовує переваги HTTP, коли використовується для веб-API. Це означає, що розробникам не потрібно встановлювати бібліотеки або додаткове програмне забезпечення, щоб скористатися дизайном REST API. Дизайн REST API був описаний Роєм Філдінгом у 2000 році. Rest є досить гнучким. Оскільки дані не прив'язані до методів та ресурсів. За допомогою правильної реалізації гіпермедіа у REST реалізована можливість обробляти кілька типів викликів, повертати різні формати даних і навіть структурно змінюватись [26].

Керівні принципи написання RESTful-інтерфейсів:

- Client-Server. Відокремлюючи призначений для користувача інтерфейс від сховища даних, ми покращуємо переносимість призначеного для користувача інтерфейсу на інші платформи і покращуємо масштабованість серверних компонент зарахунок їх спрощення.
- Stateless (без стану). Кожен запит від клієнта до сервера повинен містити в собі всю необхідну інформацію і не може покладатися на який-небудь стан, що зберігається на стороні сервера. Таким чином, інформація про поточну сесії повинна повністю зберігатися у клієнта.
- Cacheable (кешувального). Це обмеження вимагає, щоб для даних у відповіді на запит явно було зазначено - можна їх кешувати чи ні. Якщо відповідь підтримує кешування, то клієнт має право повторно використовувати дані в наступних еквівалентних запитів без звернення на сервер.
- Uniform interface (однаковість інтерфейсу). Якщо застосувати до систем інженерний принцип спільності / однаковості, то архітектура всього програми стане простіше, а взаємодія стане прозорішою і зрозумілішою. Для виконання цього принципу необхідно дотримуватися кількох архітектурних обмежень. REST накладає на інтерфейс чотири обмеження: 1) ідентичність ресурсів; 2) маніпуляція над ресурсами через подання; 3) вичерпні, зрозумілі людині повідомлення; 4) гіпермедіа (hypermedia) як движок для стану програми (HATEOAS) - посилання на інші ресурси всередині програми.
- Layered system (багаторівнева система). Многоуровневость досягається зарахунок обмеження поведінки компонентів таким чином, що компоненти "не бачать" інші компоненти, крім розташованих на найближчих рівнях, з якими вони взаємодіють.
- Code on demand (код в міру необхідності, необятельно). REST дозволяє нарощувати функціональність клієнтської програми у міру необхідності за допомогою скачування і виконання коду у вигляді аплетів або

скриптів. Це спрощує клієнтські програми, зменшуючи кількість заздалегідь написаних можливостей [26].

2.1.5 Google maps API

Базуючись на [27] наведемо відомості про Google Maps API. Google Maps API - це абсолютно безкоштовна та програмована картографічна послуга, яку надає Google. Ви можете легко розмістити його на своєму веб-сайті та регулярно позначати своє місцезнаходження, щоб клієнти могли легко визначити ваше місцезнаходження.

Сама послуга представлена у вигляді набору протоколів, за допомогою яких програмісти та веб-розробники можуть створювати різні додатки легко та швидко. До недавнього часу API були тісно пов'язані з операційними системамии. Але за останні кілька років ця тенденція розвинулася настільки швидко, що API стали незамінним інструментом в Інтернеті.

Сервіс має свої відмінні переваги, які будуть цікаві не тільки програмістам, але і звичайним користувачам:

- Цей сервіс має свої унікальні переваги, зацікавлять не лише програмістів, а й звичайних користувачів: Простий. Послуга дуже проста у використанні, користувачам потрібно лише вказати своє місцезнаходження та вказати дані, які відображатимуться на карті. У вказаному місці з’явиться позначка, а вся поточна інформація про об’єкт відображатиметься після клацання;
- Хороша візуалізація. Відвідувачі веб-сайту відразу зрозуміють як користуватись додатком;
- Доступні функції. Послуга може бути розроблена відповідно до загальної теми веб-сайту, ще більше розширюючи його можливості задовольняти потреби відвідувачів.

2.1.6 MySql:

Баруючись на [28] наведемо відомості про MySQL. MySQL - це система керування різними базами даних(реляційними), розроблена «ТсХ» для

збільшення швидкості обробки об'ємних баз даних. Була створена як альтернатива комерційним системам із відкритим кодом (СКБД). MySQL була дуже подібною до mSQL спершу, але згодом вона продовжувала розширюватися, і нині MySQL є однією з переважаючих систем управління базами даних. В основному застосовується для складання динамічних веб-сторінок, так як вона має хорошу підтримку різних мов програмування, що є головною перевагою.

Основні переваги MySQL включають:

- Масштабованість. MySQL має можливість підтримки роботи великих баз даних, що було підтверджено в Yahoo!, Google, HP та Associated Press. Відповідно до документації, що додається до MySQL, окремі бази даних, що користуються MySQL AB (розробник MySQL), зберігають до 50 мільйонів записів.
- Переносимість. MySQL працює на різноманітних платформах, включаючи Unix, Linux, Windows, OS / 2, Solaris, Mac OC.
- Підключення. MySQL має мережеву будову. Кілька користувачів можуть одночасно дістати доступ до MySQL з будь-якого місця в Інтернеті. MySQL має безліч інтерфейсів додатків (API), які дають можливістьпідключатися до MySQL із додатків, написаних на C, C ++, Perl, PHP, Java, Python та інших мовах.
- Захист. MySQL має систему контролю доступу до даних, яка надає шифрування даних під час передачі.
- Робоча швидкість.
- Простота в експлуатації. MySQL дуже простий в установці та впровадженні, а також простий в управлінні.
- Відкритий код.

2.1.7 Мова програмування Java

Базуючись на [29] наведемо відомості про мову програмування Java. Java - це об’єктно-орієнтована мова програмування, яка була випущена у 1995 році

«Sun Microsystems». Вона є основним компонентом платформи Java. З 2009 р. компанія Oracle придбала компанію Sun Microsystems у минулому році. У формальній реалізації програма Java компілюється в байт-код. Віртуальною машиною він інтерпретується для певної платформи під час виконання. Oracle надає компілятор Java та віртуальну машину Java, які відповідають специфікації Java Community Process під загальною публічною ліцензією GNU. Мова перебрала багато синтаксису з C та C ++. Особливо базується на об'єктній моделі C ++, але вона була змінена. Виключена можливість виникнення певних конфліктів, спричинених помилками програміста, та спрощує процес розробки об’єктно-орієнтованих програм. Багато операцій, які програміст повинен виконати на C / C ++, делегуються віртуальній машині. Java в основному розроблена як незалежна від платформи мова, тому вона має менше низькорівневих способів для праці з апаратним забезпеченням. У порівнянні з C ++, це скорочує темп роботи доповнень. За необхідності Java надає можливість викликати підпрограми, написані іншими мовами програмування, і може перетворювати програми на Java у байт-коди, що працюють на віртуальній машині Java(програмі JVM, яка обробляє байт-коди), а інструкції передають обладнанню як інтерпретатор. Але на відміну від тексту, байт-код може бути оброблений швидше. Перевага такого методу виконання програм-у абсолютній незалежності байт-коду від ОС та обладнання, що дозволяє запускати програми Java на будь-якому гаджеті, що підтримує віртуальні машини. Ще однією значущою рисою технології Java є гнучка система безпеки, оскільки виконання програми цілком контролюється віртуальною машиною. Будь-яка операція, що перевищує встановлені повноваження програми (наприклад, намагання отримати шлях до даних без дозволу або підключення до іншого комп'ютера), негайно призведе до її переривання. Це дозволяє користувачам завантажувати програми, написані на Java, із невідомих джерел на свої комп’ютери (або інші пристрої, такі як мобільні телефони), не турбуючись про зараження вірусом та втрату цінної інформації [29]

2.2 Розробка архітектури додатку

Дякуючи швидкому процвітанню Інтернету та концентрації більшості інформації в базі даних на сервері, клієнт-серверна архітектура стала популярною.

Архітектуру клієнт-сервер визначають як концепцію інформаційної мережі, більша частка ресурсів в якій зібрана на серверах, що обслуговують користувачів.

Архітектура окреслює такі види компонентів:

- Група серверів, що наділяють інформацією чи іншими сервісами програми, що мають до них прохід;
- Група користувачів,що користуються послугами, що представляються сервером;
- Мережа для забезпечення взаємозв'язку між користувачами та серверами.

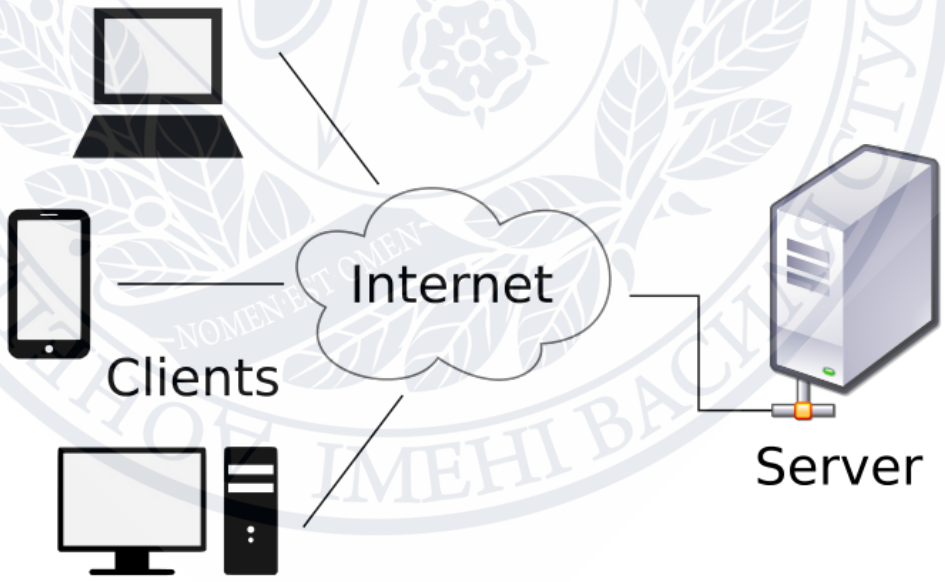


Рисунок 2.1 - Клієнт - серверна архітектура

Правило взаємозв'зку між користувачем і сервером звуть протоколом обмінювання (протокол взаємодії)

Взаємодія клієнт-серверної моделі в основному окреслюється подільністю відповідальності між користувачем і сервером.

Логічно кажучи, ми можемо розділити операції на 3 ступені:

- Ступінь подання даних, фактично користувальницький інтерфейс, дбає про зображення даних користувачеві та введення команд управління від нього;
- Ступінь прикладний, що здійснює головну логіку програми і виконує тут необхідну обробку інформації;
- Ступінь керування даними, що гарантує зберігання та доступ до даних [30].

2.3 Аналіз проектних рішень

Це завдання можна розглядати як задачу комівояжера (комівояжер — бродячий торговець), яка полягає в тому, щоб знайти найкращий маршрут через задані точки по одному разу, проте без повернення в початкову точку. Вхідні дані:

- кількість місць;
- декартові координати кожного міста;

Вихідні дані:

- довжина найкращого шляху;
- оптимальний шлях.

Оптимальний шлях – це кортеж міст, а довжина шляху між містами є найменшою серед усіх шляхів, які, як вважають відповідають умовам задачі.

Для зручності завдання моделі задачі слід скористатися теорією графів, тобто потрібно відшукати гамільтонів цикл або гамільтонів маршрут в графі.

2.3.1 Алгоритм Флойда-Воршелла

Базуючись на [31] наведемо відомості про алгоритм Флойда-Воршелла. Алгоритм Флойда-Воршелла застосовується щоб розв’язати завдання про найменший шлях в зваженому графі з додатними та від’ємними вагами ребер (проте без від’ємних циклів). У стандартному виконанні алгоритм надає довжину (загальну вагу) найменшого шляху серед усіх пар вершин, проте не дасть інформації щодо самого шляху. Різноманітні варіанти цього алгоритму застосовуються і для пошуку транзитивних замикань у відносинах чи

(зважаючи на метод Шульце) для пошуку найбільшого шляху серед усіх пар вершин у зваженому графі. Алгоритм Флойда-Воршеля це варіант динамічного програмування. Був опублікований у 1962 році в своєму звичному вигляді Робертом Флойдом. Однак це майже те саме, що і алгоритм який був представлений Бернардом Роєм у 1959 р. та Стівеном Воршеллом у 1962 р. Крім того є схожість з алгоритмом Кліні (опублікований у 1956 р.) де йдеться про детерміновані кінцеві автомати перетворені в регулярні вирази.

Пітер Інгерман у перший раз запропонував нинішнє визначення алгоритму, як 3-ьох вкладених циклів у 1962 році. Цей алгоритм також носить назву алгоритм Флойда, алгоритм Роя-Воршелла або Роя-Флойда або ж алгоритм WFI. Спосіб Воршеля зіставляє усі припустимі шляхи між кожнісінькою парою вершин у графі. Здійснюється за $\Theta(|V|^3)$ порівнянь. Зважаючи на те, що у графі може знаходитись до $\Omega(|V|^2)$ребер, це дуже примітивно, і кожна комбінація буде перевірена. З цією метою він робить це способом покращення оцінки по найменшому шляху між двома вершинами, поки оцінка не стала найкращою.

2.3.2 Алгоритм Дейкстри

Базуючись на [32] наведемо відомості про алгоритм Дейкстри. Алгоритм Дейкстри будує найкоротший шлях від вихідної вершини графа до інших (у випадку їхньої наявності). При робочому процесі алгоритму завжди відмічаються досліджені вершини графа. Крім того, вершини, позначені останніми (у цей момент), ближчі до вихідних вершин, ніж усі немарковані вершини, але далі, ніж усі позначені вершини. Спершу представлена вихідна вершина; певно, що наступна верх відмічатиметься як найбільш близька і сусідня з вихідною вершиною. Позначте деякі вершини на певному кроці. Відомий найкоротший шлях від вихідної вершини до спостережуваної вершини.

Коротко, алгоритм можна записати наступним чином:

Крок 1. Ініціалізуємо всі вершини, значення яких ∞. Ініціалізуйте початкову вершину зі значенням 0.

Крок 2. Позначте початкову вершину як пройдену, а всі інші вершини як не пройдену.

Крок 3. Виберіть початкову вершину.

Крок 4. Обчисліть відстань усіх сусідніх вузлів обраної вершини та використовуйте вартість ребра для підсумовування відстані.

Крок 5. Якщо розрахована відстань до вузла менше поточного значення, оновіть відстань.

Крок 6. Виберіть вершину з найменшою відстанню і зверніть увагу на цю вершину.

Крок 7. Повторюйте кроки 4 - 7, поки не залишите непомічені вузли. [32]

2.3.3 Метод гілок і меж

Базуючись на [33] наведемо відомості про спосіб гілок та меж.Спосіб гілок та меж заснований на ідеї розподілу набору дозволених вирішень на підмножини по порядку. Всюди у цьому методі необхідно перевіряти частини підрозділу, щоб визначити, чи вміщує підмножина найкраще вирішення; огляд виконується шляхом вичеслення найнижчого балу функції цілі на підмножині. У випадку коли оцінка внизу більше рекорду - найвдаліший з встановлених розв'язань, то підмножина буде відхилена. Якщо є можливість віднайти найліпше рішення, то можна відмовитись розглядати підмножину. Коли роль функції цілі менше, ніж рекорд, то трапляється його зміна. В кінці виконання алгоритму рекорд це і є результат праці.

У випадку коли всі частинки розділу можна відкинути, запишіть найкраще рішення. Інакше, виберіть найперспективніше (можливо найнижче значення нижчого балом) із підмножини, воно і підлягає розділенню. Нові підмножини будуть перевірені ще раз.

2.3.3.1 Алгоритм Літтла-Мурті-Суїні-Керролла

Базуючись на [34] наведемо відомості про Алгоритм Літтла. Алгоритм Літтла - це окремий випадок методу гілок та меж, тобто в гіршому випадку його складність дорівнює складності повного перебору. Його теорія описується наступним чином: усі гамільтонові цикли в графі існують у множині S. На кожному кроці в S шукається ребро (i, j), виключення якого з маршруту максимально збільшить оцінку знизу. Далі відбувається розбиття множини на два непересічних S1 і S2. S1 - всі цикли, що містять ребро (i, j) і не містять (j, i). S2 - всі цикли, що не містять (i, j). Далі обчисліть нижню оцінку для довжини шляху кожного набору, і якщо вона перевищує вже знайдену довжину рішення, то відкиньте; інакше обробляйте S1 і S2 таким же чином, як S.

Алгоритм Літтл-Мурті-Свіні-Керролла може бути виражений як:

1. Знайдімо мінімальний елемент у кожнісінькому ряді матриці цінності і позбавимо з усіх елементів рядка. Для стовпця, який не містить нулів, зробимо це саме.Одержимо матрицю вартості, кожнісінький рядок і стовпець містить принаймні 1 нульову частину.
2. Для всіх нульових частин матриці сij ми обчислюємо показник Гі j, який становить суму мінімального елементу і рядка (не включаючи елемента Сі, j = 0) та малого елемента j стовпець. Вибираємо найбільший ГК,l = max {Гі,j} показник з ГК. Гамільтонів контур вводиться відповідною дугою (k, l).
3. Видаліть k-й рядок і l стовпець, а значення елементів Cl, k змініть на необмеженість (через те, що дуга (k, l) міститься в контурі, поворотний шлях від l до k недійсний).
4. Повторюйте алгоритм на кроці 1, доки стан матриці не стане рівним 2.
5. Потім в поточний орієнтований граф вносимо 2 відсутні дуги, які визначаються матрицею порядка 2. Отримуємо гамільтонів контур.

2.3.4 Алгроритм Беллмана-Форда

Посилаючись на [35] наведемо відомості про алгоритм Беллмана-Форда. Алгоритм Беллмана-Форда - це алгоритм для обчислення найкоротшого шляху з вихідної вершини до других вершин у зваженому графіку спрямованості. Звичайно, для того ж завдання це повільніше від алгоритму Дейкстри, проте більш загальний, так як може обробляти графи з негативними вагами на деяких ребрах. Як правило, алгоритм називається в честь 2 дослідників - Річарда Беллмана та Лестера Форда, які представили алгоритм відповідно в 1958 і 1956 роках. Однак Едвард Форест Мур теж показав цей алгоритм у 1957 році, тому його деколи іменують алгоритмом Беллмана-Форда-Мура. Як згадувалося раніше, алгоритм Беллмана-Форда є доречним для обробки графів з гранями негативної ваги. Проте, якщо граф має "від'ємні цикли", тобто цикли, де сума ваг ребер становить від'ємному значенню, тоді для даного графа немає дерева найменшого шляху (всякий шлях цього типу може бути покращений іншим проходом, який утворює негативний цикл).

У даному разі алгоритм Беллмана-Форда може виявляти цикли негативної довжини та повідомляти про їх існування, але він не може дати правильної відповіді, це означає відшукати найменший шлях якщо негативний цикл можна досягти зверху джерела.

Давайте запишемо алгоритм Беллмана-Форда детальніше. Подивимось на кілька спрямованих графів із зваженими ребрами, які не містять циклів від’ємної довжини. І уявімо, вам потрібно знайти найменший шлях від вибраної вершини до всіх інших вершин графа:

1. Перш ніж запускати алгоритм, усі вершини графа є такими, що не працюють, а ребра - не переглядаються. Під час виконання алгоритму кожній вершині присвоюється число d_x, яке відповідне довжині найменшого шляху від вершини e до вершини x, що містить лише пройдені вершини. На першій стадії ми заклали $d_a = 0$ і $d_x = \infty$ для всіх x, відмінних від a. Крім того, на даному етапі, вершині a присвоюється

мітка «пройдена» і покладається $y = a$ (y – остання з пройдених вершин).

2. Для кожної вершини графа G перелічіть значення d_x таким чином:$d_x = \min\{d_x d_y + m_{yx}\}$ (1) (де m_{yx} - вага ребра (y, x)). Якщо $d_x = \infty$ для всіх нездоланних вершин x, то алгоритм Беллмана-Форда повинен бути завершений (у вихідному графі немає шляху від вершини до нездоланної вершини). В іншому випадку мітка "пройдено" повинна бути призначена вершині x з найменшим значенням d_x. Крім того, ребра, що ведуть до вершин, обраних на цьому етапі, є модифікованими (оскільки принаймні дуга досягається відповідно до виразу (1)). Після розміщення $y = x$ ітераційний процес триває. Зазначимо, що якщо для деякої опрацьованої вершини x відбувається поменшення розміру d_x , то з цієї вершини і інцидентного їй переглянутого ребра відповідні мітки усуваються.
3. Процес алгоритму Беллмана-Форда закінчиться лише тоді, коли всі вершини графа G пройдуть і коли після чергового виконання кроку номер два жодне з чисел d_x не змінило свого значення.

2.3.5 Алгроритм Джонсона

Базуючись на [36] наведемо відомості про алгоритм Джонсона. Алгоритм Джонсона дає змогу відшукати найкоротшу путь серед усіх пар вершин у зваженому спрямованому графіку. Цей алгоритм спрацьовує, коли графа містить ребра з позитивними або негативними вагами та немає циклів з від’ємними вагами. Графік G = (V, E) має вагову функцію : E → R. Коли вага ребра на графіку невід’ємна, ви можете відтворити алгоритм Дейкстри один раз для всіх вершин, щоб відшукати найкоротший путь між всіма парами вершин. Якщо графа містить ребра з негативними значеннями ваг, проте не має 23 від’ємних циклів, можна обрахувати іншу множину ребер. Новий набір, що містить ваги ребер, повинен слідувати таким вимогам:

- для всіх ребер (v, v) нова вага (u, v)> 0;

- для усіх пар вершин u, v ∈ V шлях P є найменшим шляхом від u до v вершини з застосуванням вагової функції тоді і тільки тоді, коли P — теж є найменшим шляхом з u до v вершини з ваговою функцією.

2.3.6 Алгроритм Лі

Посилаючись на [37] наведемо відомості про алгоритм Лі. Алгоритм Лі - спосіб, який дає змогу відшукати найкоротший шлях у графі із ребрами які мають одиничну довжину. Цей алгоритм закладений на алгоритмі пошуку. Він використовується для пошуку мінімального шляху у графі, як правило в цілому шукає тільки довжину. На двовимірній карті (матриці) що створюється з «прохідних» та «непрохідних» комірок, позначають початкову та кінцеву комірки. Призначення алгоритму - прокласти мінімальний шлях з початкової комірки до кінцевої, безумовно, якщо є можливість.З самого початку хвиля розповсюджується у всіх напрямках, і кожна клітина, крізь яку проходила хвиля, зазначаються «пройденою». Зі свого боку, хвилі не вдається пройти крізь клітини, позначені як "пройдена" чи "непрохідна". Хвиля перебуває в русі, доки не добереться до кінця або доки не залишиться «пустих» клітин. Якщо хвиля проклала шлях через усі вільні та доступні комірки, проте не дойшла до кінця, то шлях з початку до кінця не може бути прокладений. В кінцевому рахунку, шлях розміщується в протилежному напрямку (з кінця до початку) і тримається у масиві. Хвильовий алгоритм вважається одним з головних методів автоматизованого трасування друкованих плат. Він містить масу різних модифікацій, які можуть поліпшити рішення з урахуванням витрат пам'яті та стрімкості. Подібним чином, одним із типових використань хвильового алгоритму є відстеження найкоротшої відстані в стратегічних комп’ютерних іграх на карті.

Алгоритм складається з декількох етапів, включаючи такі: стадія підготовки, стадія розповсюдження хвилі та стадія будування шляху.На стадіїпідготовки комірки, що існують на карті, аналізуються для визначення зайнятих. Інші комірки позначають як вільні, а їх вага дорівнює "0". В

лічильнику кроків К записуємо 1. Етап поширення хвилі полягає у пошуку кінцевої точки шляхом перебирання ближніх комірок і присвоєння їм належних ваг.

Першим ділом контролюються всі комірки, які є суміжними з вихідною. Якщо вага комірки становить «0», то їй дають роль з лічильника кроків. Комірки які мають інші вагу(були заповнені на колишніх етапах) та комірки, позначені як «зайняті» зостаються без відмінностей). При тому якщо одна із комірок є кінцевою точкою, алгоритм йде на подальшу стадію-стадію будування шляху. Інакше лічильник кроків прибавляє одиницю і алгоритм, який окреслений для стартової точки, повторюється для кожної точки, яка має вагу К-1. У випадку незнаходження фінішної точки і якщо для точок які мають вагу К-1 немає доступних комірок, то можна дойти до висновку, що шляху немає. На стадії будування шляху(згортання хвилі) треба стартувати з останньої точки. Для неї вибирається комірка з її околу, вага комірки буде становити К. Потім для цієї ж комірки шукаємо суміжна, вага якої дорівнює К-1. Отож так продовжується доти, поки не буде виявлено комірку з вагою один, в околі якої розташована стартова точка(вага дорівнює 0). У такий спосіб отримуємо готовий шлях, який теж відносить до множини шляхів мінімальної довжини.

2.3.7 Алгроритм A*

Базуючись на [38] наведемо відомості про Алгоритм А*. Алгоритм пошуку А-відноситься до евристичного алгоритму пошуку. Застосовується для знаходження найменшого шляху серед двох вершин графа з додатними вагами ребер. Описано в 1968 році Пітером Хартом, Нільсом Нільссоном і Бертрамом Рафаелем.

Алгоритм застосовує допоміжні функції (евристику) для керівництва напрямком пошуку і зменшення його тривалості. Алгоритм є повним в тому плані, що він завжди може знайти оптимальне рішення (якщо воно є)

Алгоритм A * спочатку відвідує вершини найкоротшого можливого шляху до цілі. Для того, щоб ідентифікувати такі верхівки, усякій знайденій верхівці присвоюється роль, рівне довжині найменшого шляху від стартової верхівки до останньої, яка проходить через вибрану вершину. Спочатку виберіть верхівку з мінімальним значенням.

Функція вершини окреслюється наступним чином:

$$f(x) = g(x) + h(x) \quad (2.1)$$

де:

g (x) - це функція, значення якої рівне ціні шляху від стартової верхівки до x;

h (x) - евристична функція, що використовується для оцінки вартості шляху від верху до кінця.

Евристика, яка використовується, не повинна переоцінювати вартість шляху. Зразком оцінки є пряма лінія: колективний шлях не може бути меншим за пряму.

Алгоритм поділяє верхівки на 3 категорії:

• Невідомі: цих вершин не знайдено. Їхній шлях досі не зрозумілий. Спочатку алгоритму усі вершини, крім початкових, відносяться до невідомого класу.

• Відомі (OpenList): Відомий (імовірно, не оптимальний) шлях цих верхівок. Усі знайдені вершини зберігаються у списку разом із їх значеннями. З цього переліку спочатку виберіть найбільш перспективну вершину. Певна реалізація цього переліку має великий вплив на швидкість алгоритму, як правило, у формі черги пріоритетів (наприклад, бінарної купи). Від початку алгоритму лише початкові верхівки відносяться до відомих вершин

• Абсолютно досліджені вершини (ClosedList): відомий найкоротший шлях до цих верхів.

Певно досліджені вершини добавляються до так званого закритого переліку, щоб запобігти багатократному дослідженню вже досліджених

верхів. Перелік вершин, цілковито вивчених на старті алгоритму стає порожнім.

Кожнісінька відома або повністю досліджена вершина містить показник на попередню вершину. Завдяки цьому покажчику ви можете перейти звідси до вихідної точки.

У випадку коли вершина x повністю досліджена (або відкрита, розслаблена), сусідні верхівки добавляються до переліку вже отриманих вершин, а сама верхівка добавляється до цілком вивченого переліку. Показник на колишню верхівки встановлений на x. Сусідні верхівки , які вже є у цілком вивченому переліку вершин, не будуть додані до відомого переліку вершин, а зворотний вказівник не зміниться. Якщо знайдений шлях менший за знайдений шлях, то сусідні верхівки у переліку знайдених верхівок лише новішають (значення та показник на попередню верхівку).

Коли фінішна вершина попадає в повністю вивчений перелік верхівок , алгоритм припиняється. Знайдений шлях відтворюється покажчиком на минулу вершину. У випадку коли перелік вже знайдених вершин спустошиться, то рішення проблеми не існує, і алгоритм зупиняє пошук.

Відтворений за зворотніми показниками, відомий шлях стартує від останньої верхівки до першої. Для того, щоб негайно отримати шлях у правильному напрямку, від стартової верхівки до останньої вершини, за умови задачі необхідно переставити початок і кінець. Якщо ви шукаєте шлях, починаючи з останньої верхівки, відтворений перелік розпочинатиметься з стартової верхівки та йтиме до останньої.

2.3.8 Алгоритм мурашиної колонії

Базуючись на [39] наведемо відомості про мурашиний алгоритм. Мурашиний алгоритм (ACO) - ефективний поліноміальний алгоритм для пошуку наближених вирішень задач комівояжера та вирішення подібних задач знаходження маршрутів на графі. Суть цього методу: аналіз та використання моделі дій мурах для знаходження дороги від групи до джерела живлення, що є евристичною оптимізацією. Перша ідея цього алгоритму була проголошена доктором Марко Доріго у 1992 р., маючи на меті знайти оптимальний шлях на графі. У дійсному світі мурахи (спершу) безладно гуляють і приходять назад до своїх колоній у пошуках їжі, прокладаючи шлях феромонами. Коли інші мурахи знайдуть такий слід, вони, скоріш за все, будуть слідувати за ними. Якщо вони нарешті знайдуть джерело живлення, вони не будуть відстежувати ланцюг, а зміцнять його. З часом феромонні сліди починають випаровуватися, зменшуючи привабливість.

Чим довше потрібно старатись для досягнення цілі та повернення, тим більше феромонних слідів буде випаровуватися. Для порівняння, в короткостроковій перспективі проходження буде швидшим, і як результат, щільність феромону все ще залишається високою. Випаровування феромонів також має властивість уникати необхідності в локальних оптимальних рішеннях. Якщо феромон не випаровується, найкращою буде путь, обрана першим.Таким чином, дослідження просторового рішення будуть обмеженими. Тому, коли одна мураха помічає (наприклад, невелику відстань) шлях від групи до джерела гостинців, інша мураха, імовірніше, піде цією дорогою, і позитивні відгуки зрештою змусять усіх мурах вибрати найкоротший шлях..

2.3.9 Порівняння алгоритмів

Складність алгоритму – це кількісна характеристика, що відображує споживані алгоритмом ресурси під час свого виконання. Складність

алгоритмів зазвичай оцінюють за часом виконання або по використовуваній пам'яті.

Таблиця 2.1 – Порівняння алгоритмів пошуку оптимального шляху

Назва алоритму	Складність	Використання пам’яті()	Недоліки
Дейкстри	$O(V * V + E) = O(V^2)$	Використовується великий осяг пам’яті	Працює тільки для графів без ребер від'ємної довжини
Беллмана - Форда	$O(VE)$	Використовується великий осяг пам’яті	Може використовуватись лише для графів з невеликою кількістю вершин. Зміни в топології мережі не відображаються швидко.
Літтла-Мурті-Суїні-Керролла	$O(n^3)$	Використовується великий осяг пам’яті	Головний недолік полягає у відсутності ознаки оптимальності і, як наслідок, в необхідності повністю вирішувати задачі лінійного програмування, асоційовані з кожною вершиною багатогранника допустимих рішень.

Продовження таблиці 2.1

Флойда - Воршелла	$O(n^3)$	Використовується великий осяг пам’яті	Має обчилювальну складність, його застосування вимагатиме значних витрат з боку ресурсів обчислювальної машини
A*	O(|n| * log|n|)	Менше використання пам’яті зумовлене кількістю вершин, що обробляються	Залежність алгоритму від евристики, обмеженість ресурсів пам’яті, тимчасова складність
Лі	O(n)	Використовується великий осяг пам’яті	Мала швидкодія і великий обсяг оперативної пам'яті ЕОМ
Алгоритм мурашиної колонії	$O(NC*n^2*m)$	Використовується великий осяг пам’яті	Ефективність його роботи залежить від значень керуючих параметрів, і потреба їх підбору.

Продовження таблиці 2.1

Джонсона	O(n*E*log(n))	Використовується великий осяг пам’яті	Необхідність знати оцінку складності задачі. Якщо встановлена межа глибини буде занадто великою, це призведе до зайвих затрат ресурсів на пошук, якщо занадто малою — ціль не буде досягнута.

Висновок за розділом 2

У даному розділі було представлено та порівняно алгоритми пошуку найкоротшого шляху таких як: алгоритм Дейкстри, алгоритм Флойда-Воршелла, Алгоритм Літтла-Мкрті-Суїні-Керрола, алгоритм Беллмана-Форда, алгоритм Джонсона, алгоритм Лі, алгоритм А*, алгоритм мурашиної колонії. У кожного алгоритму є свої переваги та недоліки, одним з найважливіших показників це використання ресурсів, які будуть використовуватись. Для реалізації буде використовуватись алгоритм мурашиної колонії, оскільки завдяки ньому можна побудувати оптимальний шлях через декілька точок досить швидко.

Також було представленні технології для створення додатку. У ході роботи буде використано мову програмування Java, так як вона має значні переваги, так як: кросплатформеність, простота та швидкодія. Основним фреймворком є Cuba, так як тут є такі переваги: можливість швидкої та простої взаємодії з базою даних, відкритий код, часте оновлення, доступ до великої кількості бібліотек.

Розділ 3

Програмна реалізація автоматизованої системи колективного замовлення таксі

3.1 Специфікація системи

З огляду на завдання, визначені в розділі 1.3, дослідивши сферу замовлень таксі та розробки веб-додатків можна сформувати чіткіші вимоги до програми та даних:

У базу даних потрібно зберігати таку інформацію:

- Особисті дані клієнта: ім’я, прізвище, вік, пошта, фотографії.
- Особисті дані водія: ім’я, прізвище, вік, пошта, телефон, марка автомобіля, номер посвідчення водія, виробник та марка автомобіля, номер транспортного засобу, фотографії.
- Інформацію про замовлення: дані клієнта, дані водія, адресу посадки, адресу висадки, тип оплати(готівкова, безготівкова), ціна, дата та час подачі заявки, а також дата та час завершення поїздки.
- Перелік допустимих виробників та марок транспортних засобів у конкретній службі по замовленню таксі.

До інформації, яку необхідно зберігати, висуваються наступні обмеження:

- Вік водія повинен бути в певних межах.
- Вік клієнта повинен бути більше ніж 15 років.
- Такі дані як номер телефону, логін мають бути унікальні.
- Номер автомобіля повинен бути унікальним.
- Номер посвідчення водія повинен бути унікальним.
- Водій може мати лише одне активне замовлення.
- Клієнт може мати лише одне активне замовлення.
- Варіанти оплати повинні обтратись зі списку.

- Автомобіль(марка та виробник) повинен належати до переліку дозволених автомобілей.
- Номер телефону має відповідати формату номеру України.

Для правильної роботи система повинна мати такі можливості:

- Швидка та зручна реєстрація для клієнтів.
- Швидка та зручна реєстрація для водіїв, де повинні вказуватись та перевірятись обов'язкові дані(інформація про автомобіль, інформація про посвідчення водія).
- Можливість входу в особистий кабінет, де має бути реалізована можливість зміни паролю, особистої інформації, фотографії
- Попередній розрахунок можливої вартості замовлення.
- «Прийняти поїздку» - водії на спеціальній сторінці можуть прийняти замовлення з відображенням відповідної інформації.
- «Скасувати поїздку» - клієнт може скасувати замовлення, якщо воно не було прийняте жодним водієм.
- «Завершити поїздку» - водії при виконанні замовлення вказують, що замовлення виконане.

Архітектура системи є клієнт-серверною, де на клієнтській частині має бути реалізована можливість навігації по сайту а також виконання різноманітних функцій за допомогою різних кнопок та форм. Серверна частина повинна реалізовувати роботу з базою даних та обробляти запити з клієнтської частини.

Сайт повинен працювати в окремому вікні браузера та коректно відображатись на всіх можливих типах пристроїв. Також додаток має підтримуватись всіма сучасними браузерами.

Веб-додаток має відповідати таким умовам надійності:

- Доступ до бази даних має розподілятись відповідно до ролей користувачів сайту.
- Доступ до замовлень є лише у водіїв. Після того як водій прийняв замовлення, воно повинне відображатись лише у цього водія.

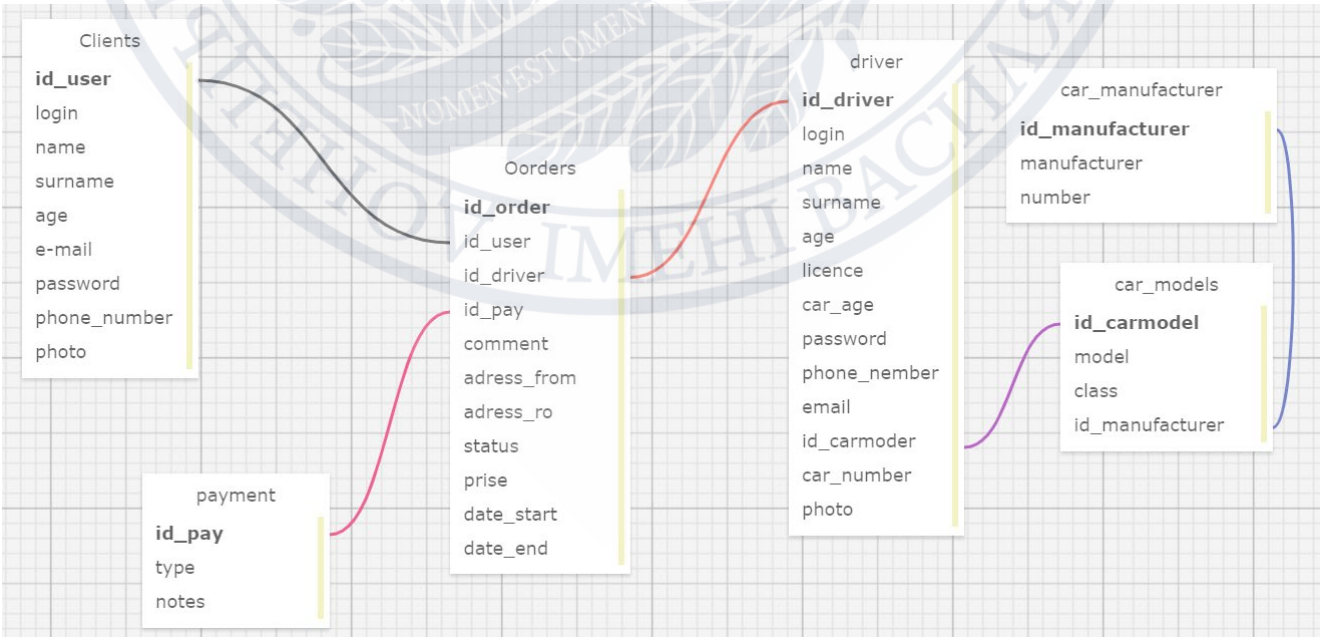
- Робити замовлення може лише зареєстрований користувач.
- Водій матиме доступ до сторінки з замовленнями, лише після реєстрації та повного заповнення потрібної інформації.
- Якщо користувач спробує перейти на сторінки до яких в нього немає доступу, система має виконувати переадресацію даного користувача на сторінку з помилкою.

3.2 Схема бази даних

Оскільки сайт спрямований на людей, які хочуть замовити таксі та на водіїв, які виконують замовлення користувачів база даних включатиме наступні таблиці:

- Orders – замовлення;
- Clients – клієнти;
- Payment – оплата;
- Driver – водії;
- Car_models – марка автомобіля;
- Car_manufacturer – виробник автомобіля.

База даних містить всі необхідні поля там має схему даних представлену на рис. 3.1.



- Рисунок 3.1 - Схема бази даних

Таблиці, що зображені на схемі, мають наступний опис:

Таблиця 3.1 – Таблиці бази даних

Назва поля	Тип даних
ORDERS	
id_order	Int
id_user	Int
id_driver	Int
id_pay	Int
coment	VarChar
address_from	VarChar
address_to	VarChar
status	Int
prise	Int
date_start	datetime
date_end	datetime
DRIVERS	
name	VarChar
surname	VarChar
age	Int
licence	VarChar
car_age	Int
login	VarChar
password	VarChar
car_number	VarChar
id_carmodel	Int
phone_number	VarChar
email	VarChar
id_carmodel	Int
PAYMENT	

Продовження таблиці 3.1

id_pay	Int
type	VarChar
notes	text
CLIENTS	
id_user	Int
name	VarChar
surname	VarChar
age	Int
e-mail	VarChar
login	VarChar
password	VarChar
phone_number	VarChar
photo	longtext
CAR_ MANUFACTURER	
id_manufacturer	Int
manufacturer	VarChar
number	VarChar
CAR_MODELS	
id_carmodel	Int
model	VarChar
class	Integer
id_manufacturer	VarChar

Таблиця ORDERS містить в собі дані про замовлення, має ключі з унікальними ідентифікаторами водіїв, клієнтів та оплатою. Також містить інформацію про адресу посадки, адресу висадки, статус поїздки, ціну поїздки та час початку та завершення замовлення.

В таблиці DRIVERS міститься інформація про водіїв, а саме: ім’я, прізвище, вік, дані про посвідчення водія, рік випуску автомобіля, номер телефону, номер автомобіля, фотографію та зовнішній ключ з унікальним ідентифікатором марки автомобіля.

В таблиці CLIENTS міститься інформація про клієнтів, а саме: ім’я, прізвище, вік, email, номер телефону фотографія.

В таблиці CAR_MODELS міститься інформація про дозволені марки автомобілей та зовнішній ключ з унікальним ідентифікатором виробника.

В таблиці CAR_MANUFACTURER міститься інформація про дозволених виробників автомобіля.

В таблиці PAYMENT міститься інформація про оплату.

3.3 Тестування алгоритму мурашиної колонії

У ході роботи була виконана програмна реалізація мурашиного на мові програмування Java. Для підвищення ефективності роботи провелись тести, мета яких було знайти параметри алгоритму, що гарантують найменшу довжину маршруту при мінімально витраченому часу на його пошук. Тестування було проведено з різною кількістю вузлів та чисельністю популяції(вид таблицю 3.2)

Таблиця 3.2 – Результати тестування алгоритму

Кількість вузлів	Чисельність популяції	Час проходження маршруту (у хвилинах)	Час обрахунків(у мілі секундах)
5	10	246.82	525936
5	100	240.60	594801
5	10	240.60	190672
5	50	240.60	280735
5	100	240.60	320717
5	10	240.60	418961
5	50	240.60	455337

Продовження таблиці 3.2

5	100	240.60	172333
9	10	347.33	979128
9	50	347.80	1019110
9	100	345.37	1040919
9	10	328.97	1069626
9	50	319.52	1105175
9	100	320.48	1133307
9	10	355.80	1160534
9	50	350.50	1209408

Приведені вище результати показують, що при невеликій кількості вузлів кількість популяції в меншій мірі впливають безпосередньо на довжину маршруту, але помітно збільшують час обрахунків.

3.4 Архітектура системи

У роботі використано концепцію MVC(model-view-controller) та було отримано модулі та взаємодії між ними, які зображено на рисунку 3.2.

Кожен модуль відповідає за свої функції, а саме:

- USERS – користувачі;
- Presentation Layer – користувацький інтерфейс;
- Business Layer та Calling Services – відповідають за бізнес логіку додатку;
- Data Layer – об’єктне відображення бази даних;
- Data Sources – база даних;
- Services – зовнішні сервіси.

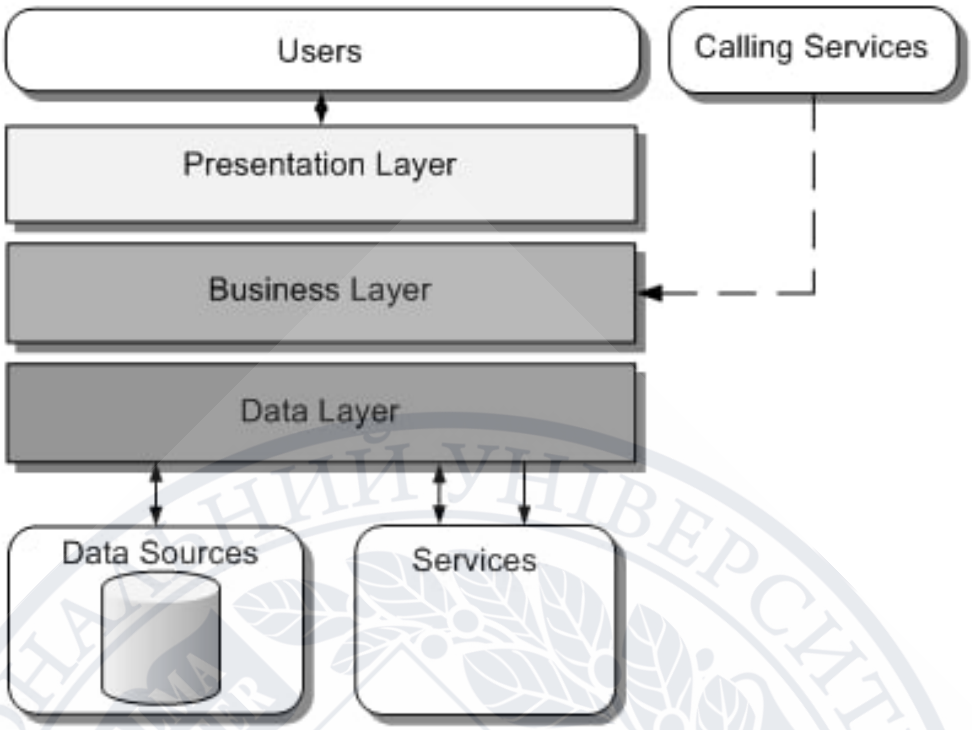


Рисунок 3.2 – Архітектура додатку

3.5 Інтерфейс програми

При вході на сайт перше, що бачить користувач це сторінка авторизації(див рисунок 3.3). У системі реалізовано розподілення користувачів на ролі: адміністратор, модератор, водій та клієнт. Адміністратор має найбільше прав доступу, в нього є можливість адмініструвати сайт, бути водієм та бути клієнтом тому доцільно розглядати функції системи ввійшовши як адміністратор.

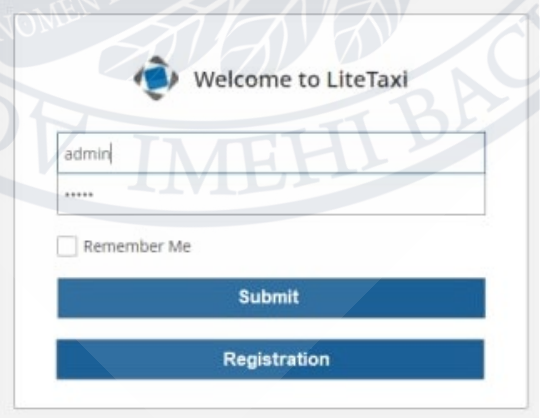


Рисунок 4.3 - Сторінка авторизації користувача

Після авторизації доступне меню в якому є 5 пунктів(див рисунок 3.4)

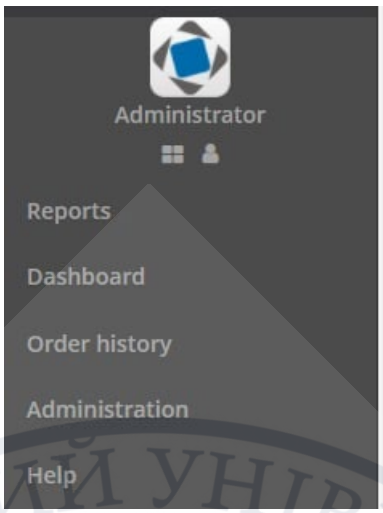


Рисунок 3.4 - Меню користувача

Пункт Reports доступний лише для адміністратора та здійснює генерацію звітів про поїздки.

Пункт Dashboard доступний теж лише для адміністратора та здійснює візуальну генерацію звітів про поїздки та показує поточний стан інфраструктури таксі.

Пункт Order history показує історію замовлень.

В пункті Administration відбувається адміністративне керування сайтом, можна здійснювати керування користувачами(реєстрація та видалення), керувати ролями користувачів та бачити стан системи.

Пункт Help містить інструкцію для користувачів.

Розглянемо детальніше пункт меню Order(див рисунок 3.5).

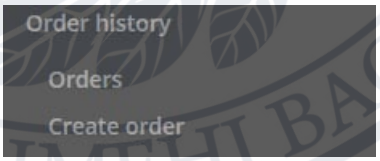


Рисунок 3.5 - Пункт меню Order

Тут є два підпункти: у підпункті Orders можна переглянути історію всіх замовлень, у підпункті Create order можна розмістити нове замовлення.

Натиснувши на пункт Create order з’являється спливаюче вікно, де можна замовити таксі(див рисунок 3.6).

Рисунок 3.6 - Спливаюче вікно з замовленням таксі

У полі Start address вказується місце посадки. У полі End address вказується місце висадки. При чому адресу можна вказати як вручну, так і вказати місце на карті.

Якщо вибрати пункт ‘’Is collective order’’ то замовлення таксі буде колективним. Після заповнення всіх необхідних полів потрібно натиснути на кнопку ‘’OK’’, після чого буде спливаюче вікно, яке повідомляє, що замовлення формується(див рисунок 3.7).

Рисунок 3.7 - Вікно в якому повідомляється, що замовлення формується.

Після того як замовлення сформується користувачеві спливе інформація, де будуть вказані дані про поїздку.

Натиснувши на підпункт меню Orders ми можемо переглянути усі замовлення які були виконані та замовлення які на даний момент у процесі(див рисунок 3.8).

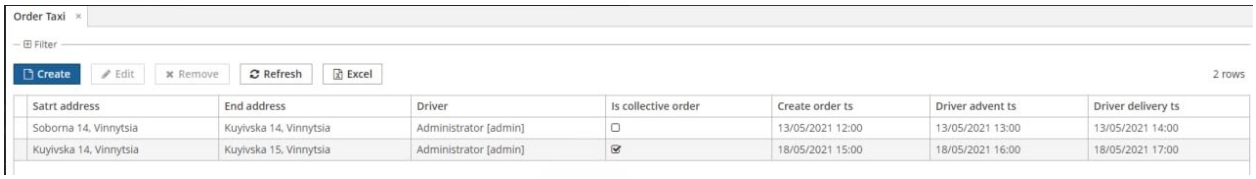


Рисунок 3.8 - Вкладка Order

У колонці Start address міститься інформаціє про місце посадки. У колонці End address міститься колонка про місце висадки. У колонці Driver міститься інформація про водія, який виконував замовлення. У колонці Is collective order міститься інформація чи було замовлення таксі колективним. У колонці Create order ts міститься інформація про дату та час створення заявки. У колонці Driver advent ts міститься інформація про дату та час прибуття водія. У колонці Driver delivery ts міститься інформація про тривалість поїздки.

Також у верхній частині містяться кнопки:

- Create – створити замовлення;
- Edit – редагувати замовлення(якщо його ще не прийняв водій);
- Refresh – оновити сторінку;
- Excel – експортувати інформацію про поїздки в таблиці Excel.

Далі детально розглянемо пункт меню Dashboard. Тут можна побачити кількість замовлень за певний проміжок часу(див рисунок 3.9).

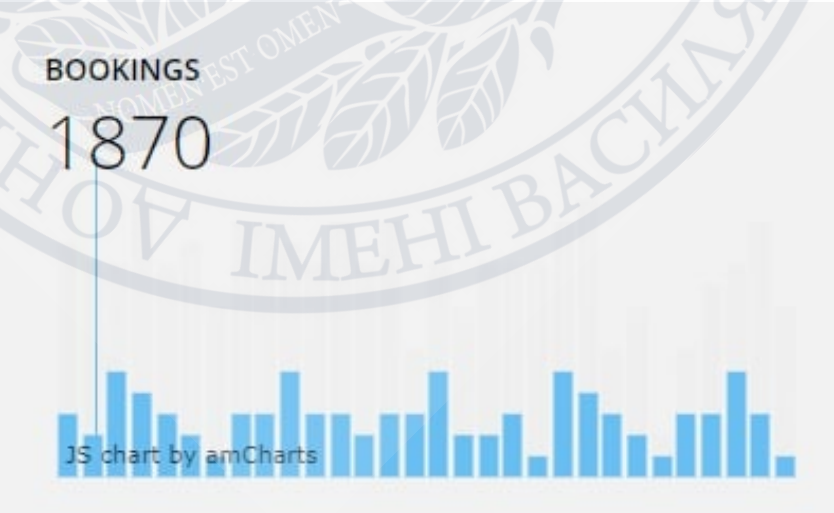


Рисунок 3.9-Кількість виконаних поїздок за останні 48 годин

Також у даному пункті меню можна побачити карту, де відображається поточне розташування усіх водіїв, які є в мережі(див рисунок 3.10).

Рисунок 3.10 - Карта з розташуванням всіх водіїв

У підпункті Car delivery можна побачити скільки водіїв виконують замовлення(зеленим кольором), скільки водіїв вільних(блакитним кольором) та скільки водіїв очікують обробки заявки(жовтим кольором)(див рисунок 3.11).

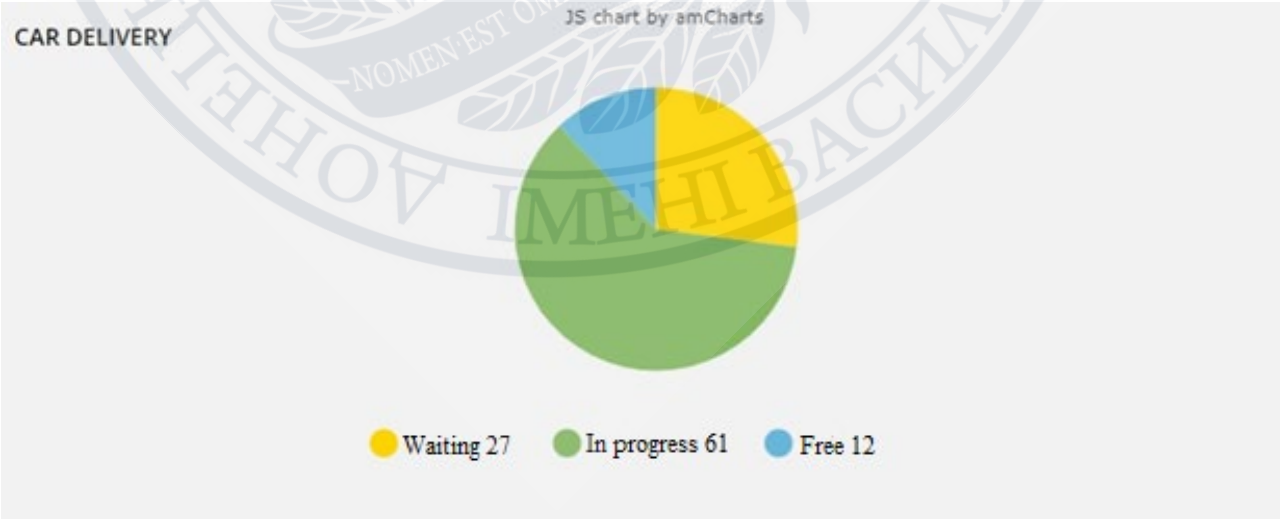


Рисунок 3.5 - Підпункт Car delivery

Розглянемо пункт меню Administrator. Тут доступне управління користувачами, а саме можна змінити данні користувача, скопіювати користувача, додати нового користувача, або ж видалити старого(див рисунок 3.12)

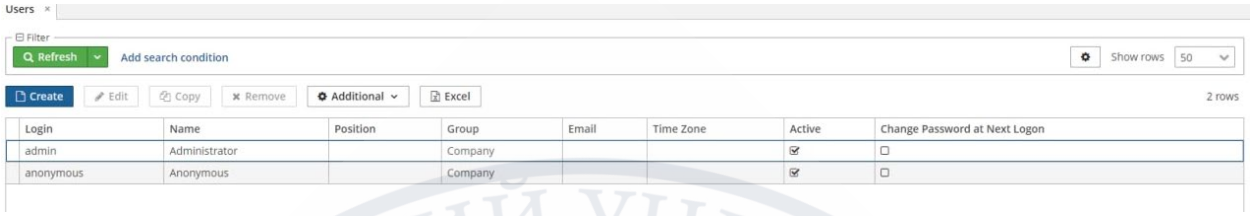


Рисунок 3.6 - Вкладка керування користувачами

.Є можливість керувати групами доступу для користувачів, можна створювати нові групи доступу та призначати їх користувачам (див рисунок 3.13).

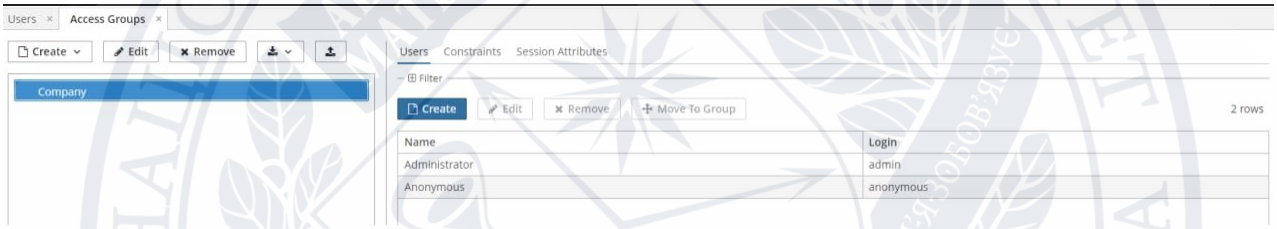


Рисунок 3.7 - Керування групами доступу

Також адміністратор може переглядати активні користувацькі сесії(див рисунок 3.14).

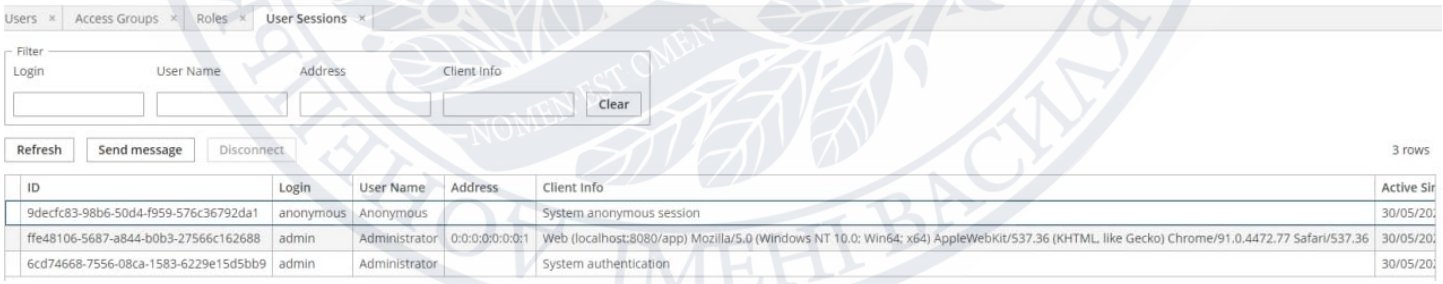


Рисунок 3.8 - Активні користувацькі сесії

У пункті logs є можливість переглядати логи серверу, тобто відображення усіх дій сервера(див рисунок 3.15).

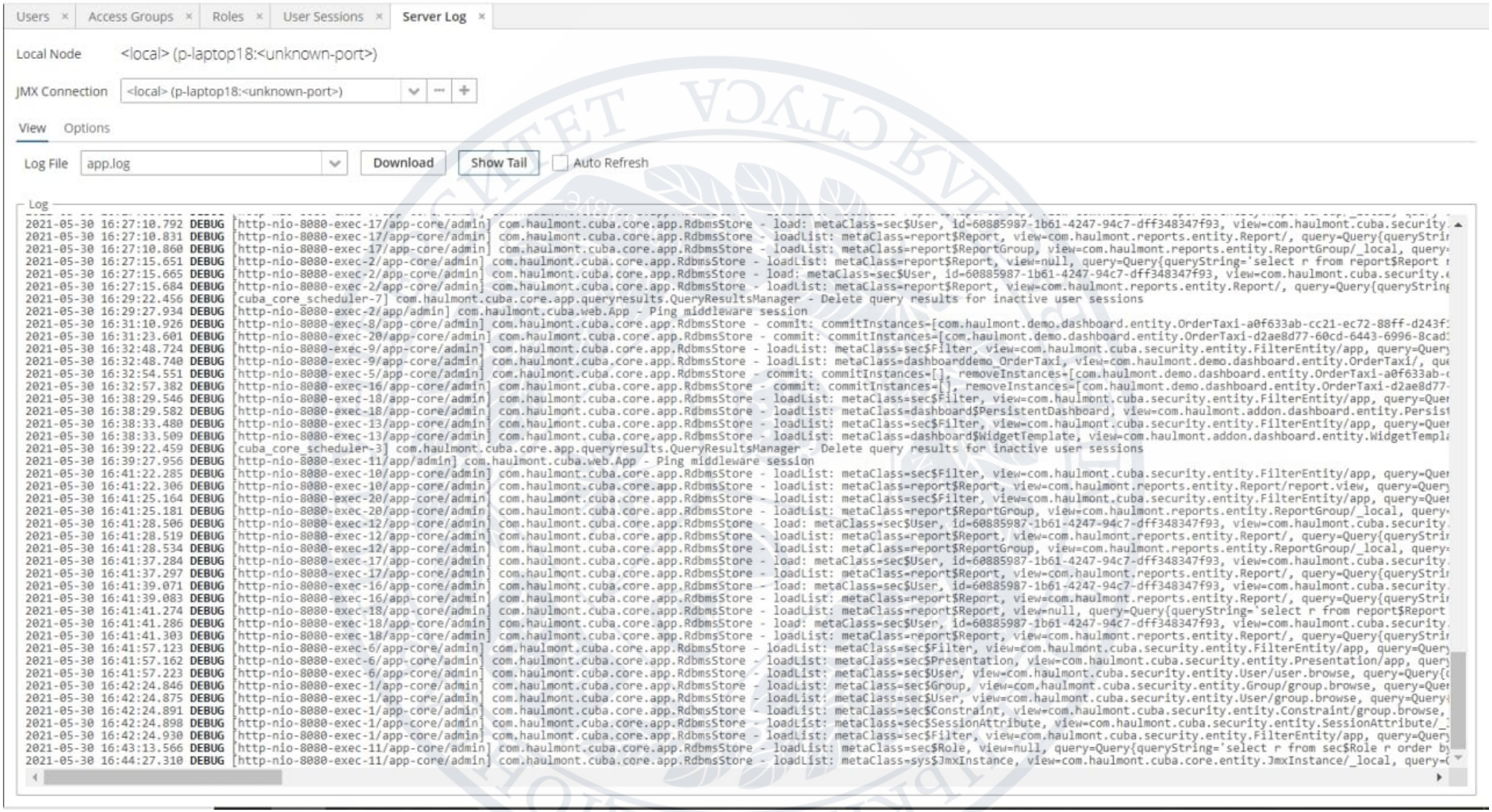


Рисунок 3.15 – Логи сервера

Висновок за розділом 3

У даному розділі було розроблено базу даних, тестування алгоритму мурашиної колонії для пошуку оптимального шляху. Розроблена архітектура додатку та взаємодія модулів програми. Виконана та продемонстрована практична реалізація веб-додатку для колективного замовлення таксі.

ВИСНОВКИ

У ході роботи було вивчено сферу замовлення таксі в Україні. Було виявлено переваги та недоліки найпопулярніших існуючих рішень, що в свою чергу дозволило розробити додаток, який зможе задовольнити велику кількість користувачів. Було вивчено можливі технології розробки, що дозволило побудувати оптимальну архітектуру додатку та логічну взаємодію окремих блоків програми. Основним фреймворком, на якому побудована серверна та клієнтська частина є Cuba. Розроблений сайт має декілька сторінок з всією важливою інформацією та функціоналом, розмежування ролей на клієнта та водія, можливість працювати з картами Google на яких будується оптимальний маршрут. У роботі був використаний мурашиний алгоритм, для знаходження оптимального шляху між декількома точками(клієнтами) на карті. Була розроблена база даних з всіма необхідними таблицями та полями для коректної роботи додатку. На сайті клієнт має можливість замовити колективне та персональне таксі, водій має можливість вибирати та приймати замовлення на окремій сторінці. Реалізована можливість розрахунку попередньої вартості поїдки. Веб-додаток працює в усіх популярних браузерах коректно та має зручний та інтуїтивно зрозумілий навіть для нового користувача інтерфейс.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Uber.[Електронний ресурс]. URL: https://www.uber.com/ua/ru/
2. Uklon.[Електронний ресурс]. URL: https://uklon.com.ua/ru/
3. Bolt.[Електронний ресурс]. URL: https://bolt.eu/uk-ua/
4. Taxi838.[Електронний ресурс]. URL: https://taxi838.ua/ua/
5. Закон України. «Про автомобільний транспорт» // Відомості Верховної Ради України(ВВР), 2001, №22, ст105.
6. Software framework [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Software_framework.
7. Обзоры web-фреймворков. [Електронний ресурс]. URL:https://praktikatech.wordpress.com/category/%D0%BE%D0%B1%D0%B7%D0%BE%D1%80%D1%8B-eb-%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA%D0%BE%D0%B2/page/2/.
8. Flask. [Електронний ресурс]. URL: https://www.fullstackpython.com/flask.html.
9. The Vision For Twisted. [Електронний ресурс]. URL: http://twistedmatrix.com/documents/current/core/howto/vision.html. – Назва з екрану.
10. Docs. Tornado Web Server. [Електронний ресурс]. URL: http://www.tornadoweb.org/en/stable/index.html.
11. Pyramid Introduction. [Електронний ресурс]. URL: http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/introduction.html.
12. Web MVC framework. URL: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html.
13. JSNI.[Електронний ресурс]. URL: http://www.gwtproject.org/doc/latest/DevGuideCodingBasicsJSNI.html.

14. Cuba.[Електронний ресурс]. URL: https://www.cuba-platform.ru/documentation/
15. Фреймворки в веб-разработке. [Електронний ресурс].URL : https://web-creator.ru/articles/about_frameworks.
16. Why .NET? [Електронний ресурс]. URL : https://www.microsoft.com/net/intro.
17. 1Banks A. React: Functional Web Development with React and Redux. - Видавництво O'Reilly Media; 1 edition, 2017 - 350 с
18. Фримен А. Angular для профессионалов. - СПб: Видавництво Пітер, 2018 - 800 с
19.. Керівництво по Vue.js [Електронний ресурс] .URL: https://metanit.com/web/vuejs/
20. Redux.[Електронний ресурс]. URL: https://redux.js.org/
21. Хаггарти Р Дискретная математика для программистов. - М.: «Техносфера», 2003.
22. Герберт Шилдт. Java 8: руководство для начинающих, 6-е изд. ООО "И.Д. Вильямс", 2015. 720с.
23. Джошуа Блох. Java. Эффективное программирование Издательство «Лори».2019.219С
24. Alex Berson. Client/Server Architectur.1992. New York 567c.
25. Хабибулин Ильдар Шаукатович. Java. Издатель БХВ-Петербург 2012. 768 с.
26. Rest архітектура.[Електронний ресурс]. URL: https://habr.com/ru/post/38730/
27. Google maps API.[Електронний ресурс]. URL: https://developers.google.com/maps/documentation?hl=ru
28. MySQL.[Електронний ресурс]. URL: https://www.mysql.com/
29. Java.[Електронний ресурс]. URL: https://www.oracle.com/ru/java/

30. Клієнт-серверна архітектура.[Електронний ресурс]. URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/
31. Алгоритм Флойда-Уоршелла.[Електронний ресурс]. URL: https://habr.com/ru/post/105825/
32. Алгоритм Дейкстры. Поиск оптимальных маршрутов на графе.[Електронний ресурс]. URL: https://habr.com/ru/post/111361/
33. Целочисленное рішення методом гілок і меж.[Електронний ресурс]. URL: https://lab-music.ru/uk/celochislennoe-reshenie-metodom-vetvei-i-granic-primer-primer-reshenii/
34. Алгоритм Литтла, Мурти, Суини и Кэрел для задачи коммивояжера.[Електронний ресурс]. URL: https://scask.ru/g_book_dpr.php?id=47
35. Алгоритм Беллмана-Форда.[Електронний ресурс]. URL: https://habr.com/ru/company/otus/blog/484382/
36. Алгоритм Джонсона.[Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B6%D0%BE%D0%BD%D1%81%D0%BE%D0%BD%D0%B0:
37. Хвильовий алгоритм.[Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/%D0%A5%D0%B2%D0%B8%D0%BB%D1%8C%D0%BE%D0%B2%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC
38. Введение в алгоритм A*.[Електронний ресурс]. URL: https://habr.com/ru/post/331192/
39. Marco Dorigo, Thomas Stützle. Ant Colony Optimization. Massachusetts Institute of Technology.2004.305c.
40. J. Park and B. Kim. The school bus routing problem: A review. European Journal of Operational Research, 2010.319c.

Додаток А

Лістинг вихідного коду програми.

Setting.gradle

```
rootProject.name = 'dashboard-demo'

def modulePrefix = 'app'

include(":${modulePrefix}-global", ":${modulePrefix}-core", ":${modulePrefix}-web")

project(":${modulePrefix}-global").projectDir = new File(settingsDir, 'modules/global')

project(":${modulePrefix}-core").projectDir = new File(settingsDir, 'modules/core')

project(":${modulePrefix}-web").projectDir = new File(settingsDir, 'modules/web')
```

gradlew.bat

```
@if "%DEBUG%" == "" @echo off
@rem ##########################################################################
@rem
@rem  Gradle startup script for Windows
@rem
@rem ##########################################################################


@rem Set local scope for the variables with windows NT shell
if "%OS%"=="Windows_NT" setlocal


set DIRNAME=%~dp0
if "%DIRNAME%" == "" set DIRNAME=.
```

```
set APP_BASE_NAME=%~n0
set APP_HOME=%DIRNAME%

@rem Add default JVM options here. You can also use JAVA_OPTS and GRADLE_OPTS to pass JVM options to this script.
set DEFAULT_JVM_OPTS=

@rem Find java.exe
if defined JAVA_HOME goto findJavaFromJavaHome

set JAVA_EXE=java.exe
%JAVA_EXE% -version >NUL 2>&1
if "%ERRORLEVEL%" == "0" goto init

echo.
echo ERROR: JAVA_HOME is not set and no 'java' command could be found in your PATH.
echo.
echo Please set the JAVA_HOME variable in your environment to match the
echo location of your Java installation.

goto fail

:findJavaFromJavaHome
set JAVA_HOME=%JAVA_HOME:"=%
set JAVA_EXE=%JAVA_HOME%/bin/java.exe

if exist "%JAVA_EXE%" goto init
```

```
echo.
echo ERROR: JAVA_HOME is set to an invalid directory: %JAVA_HOME%
echo.
echo Please set the JAVA_HOME variable in your environment to match the
echo location of your Java installation.

goto fail

:init
@rem Get command-line arguments, handling Windows variants

if not "%OS%" == "Windows_NT" goto win9xME_args

:win9xME_args
@rem Slurp the command line arguments.
set CMD_LINE_ARGS=
set _SKIP=2

:win9xME_args_slurp
if "x%~1" == "x" goto execute

set CMD_LINE_ARGS=%*

:execute
@rem Setup the command line

set CLASSPATH=%APP_HOME%\gradle\wrapper\gradle-wrapper.jar

@rem Execute Gradle
```

```
"%JAVA_EXE%" %DEFAULT_JVM_OPTS% %JAVA_OPTS%
%GRADLE_OPTS% "-Dorg.gradle.appname=%APP_BASE_NAME%" -
classpath "%CLASSPATH%" org.gradle.wrapper.GradleWrapperMain
%CMD_LINE_ARGS%


:end

@rem End local scope for the variables with windows NT shell

if "%ERRORLEVEL%"=="0" goto mainEnd


:fail

rem Set variable GRADLE_EXIT_CONSOLE if you need the _script_ return code
instead of

rem the _cmd.exe /c_ return code!

if  not "" == "%GRADLE_EXIT_CONSOLE%" exit 1

exit /b 1


:mainEnd

if "%OS%"=="Windows_NT" endlocal


:omega
```

Gradlew

```
PRG="$0"

# Need this for relative symlinks.

while [ -h "$PRG" ] ; do

   ls=`ls -ld "$PRG"`

   link=`expr "$ls" : '.*-> \(.*\)$'`

   if expr "$link" : '/.*' > /dev/null; then

      PRG="$link"

   else
```

```
        PRG=`dirname "$PRG"`"/$link"
    fi
done
SAVED="`pwd`"
cd "`dirname \"$PRG\"`/" >/dev/null
APP_HOME="`pwd -P`"
cd "$SAVED" >/dev/null

APP_NAME="Gradle"
APP_BASE_NAME=`basename "$0"`

# Add default JVM options here. You can also use JAVA_OPTS and
GRADLE_OPTS to pass JVM options to this script.
DEFAULT_JVM_OPTS=""

# Use the maximum available, or set MAX_FD != -1 to use that value.
MAX_FD="maximum"

warn () {
    echo "$*"
}

die () {
    echo
    echo "$*"
    echo
    exit 1
}
```

```
# OS specific support (must be 'true' or 'false').
cygwin=false
msys=false
darwin=false
nonstop=false
case "`uname`" in
  CYGWIN* )
    cygwin=true
    ;;
  Darwin* )
    darwin=true
    ;;
  MINGW* )
    msys=true
    ;;
  NONSTOP* )
    nonstop=true
    ;;
esac

CLASSPATH=$APP_HOME/gradle/wrapper/gradle-wrapper.jar

# Determine the Java command to use to start the JVM.
if [ -n "$JAVA_HOME" ] ; then
    if [ -x "$JAVA_HOME/jre/sh/java" ] ; then
        # IBM's JDK on AIX uses strange locations for the executables
        JAVACMD="$JAVA_HOME/jre/sh/java"
    else
        JAVACMD="$JAVA_HOME/bin/java"
```

```
    fi
    if [ ! -x "$JAVACMD" ] ; then
        die "ERROR: JAVA_HOME is set to an invalid directory: $JAVA_HOME

Please set the JAVA_HOME variable in your environment to match the
location of your Java installation."
    fi
else
    JAVACMD="java"
    which java >/dev/null 2>&1 || die "ERROR: JAVA_HOME is not set and no
'java' command could be found in your PATH.

Please set the JAVA_HOME variable in your environment to match the
location of your Java installation."
fi

# Increase the maximum file descriptors if we can.
if [ "$cygwin" = "false" -a "$darwin" = "false" -a "$nonstop" = "false" ] ; then
    MAX_FD_LIMIT=`ulimit -H -n`
    if [ $? -eq 0 ] ; then
        if [ "$MAX_FD" = "maximum" -o "$MAX_FD" = "max" ] ; then
            MAX_FD="$MAX_FD_LIMIT"
        fi
        ulimit -n $MAX_FD
        if [ $? -ne 0 ] ; then
            warn "Could not set maximum file descriptor limit: $MAX_FD"
        fi
    else
        warn "Could not query maximum file descriptor limit: $MAX_FD_LIMIT"
```

```
    fi
fi

# For Darwin, add options to specify how the application appears in the dock
if $darwin; then
    GRADLE_OPTS="$GRADLE_OPTS \"-Xdock:name=$APP_NAME\" \"-Xdock:icon=$APP_HOME/media/gradle.icns\""
fi

# For Cygwin, switch paths to Windows format before running java
if $cygwin ; then
    APP_HOME=`cygpath --path --mixed "$APP_HOME"`
    CLASSPATH=`cygpath --path --mixed "$CLASSPATH"`
    JAVACMD=`cygpath --unix "$JAVACMD"`

    # We build the pattern for arguments to be converted via cygpath
    ROOTDIRSRAW=`find -L / -maxdepth 1 -mindepth 1 -type d 2>/dev/null`
    SEP=""
    for dir in $ROOTDIRSRAW ; do
        ROOTDIRS="$ROOTDIRS$SEP$dir"
        SEP="|"
    done
    OURCYGPATTERN="(^($ROOTDIRS))"
    # Add a user-defined pattern to the cygpath arguments
    if [ "$GRADLE_CYGPATTERN" != "" ] ; then
        OURCYGPATTERN="$OURCYGPATTERN|($GRADLE_CYGPATTERN)"
    fi
    # Now convert the arguments - kludge to limit ourselves to /bin/sh
```

```
  i=0
  for arg in "$@" ; do
    CHECK=`echo "$arg"|egrep -c "$OURCYGPATTERN" -`
    CHECK2=`echo "$arg"|egrep -c "^-"`                    ### Determine if
an option

    if [ $CHECK -ne 0 ] && [ $CHECK2 -eq 0 ] ; then       ### Added a
condition
      eval `echo args$i`=`cygpath --path --ignore --mixed "$arg"`
    else
      eval `echo args$i`="\"$arg\""
    fi
    i=$((i+1))
  done
  case $i in
    (0) set -- ;;
    (1) set -- "$args0" ;;
    (2) set -- "$args0" "$args1" ;;
    (3) set -- "$args0" "$args1" "$args2" ;;
    (4) set -- "$args0" "$args1" "$args2" "$args3" ;;
    (5) set -- "$args0" "$args1" "$args2" "$args3" "$args4" ;;
    (6) set -- "$args0" "$args1" "$args2" "$args3" "$args4" "$args5" ;;
    (7) set -- "$args0" "$args1" "$args2" "$args3" "$args4" "$args5" "$args6" ;;
    (8) set -- "$args0" "$args1" "$args2" "$args3" "$args4" "$args5" "$args6"
"$args7" ;;
    (9) set -- "$args0" "$args1" "$args2" "$args3" "$args4" "$args5" "$args6"
"$args7" "$args8" ;;
  esac
fi
```

```
# Escape application args
save () {
    for i do printf %s\\n "$i" | sed "s/'/'\\\\''/g;1s/^/'/;\$s/\$/' \\\\/" ; done
    echo " "
}
APP_ARGS=$(save "$@")


# Collect all arguments for the java command, following the shell quoting and substitution rules
eval set -- $DEFAULT_JVM_OPTS $JAVA_OPTS $GRADLE_OPTS "\"-Dorg.gradle.appname=$APP_BASE_NAME\"" -classpath "\"$CLASSPATH\"" org.gradle.wrapper.GradleWrapperMain "$APP_ARGS"


# by default we should be in the correct project dir, but when run from Finder on Mac, the cwd is wrong
if [ "$(uname)" = "Darwin" ] && [ "$HOME" = "$PWD" ]; then
  cd "$(dirname "$0")"
fi


exec "$JAVACMD" "$@"
```

build.gradle

```
buildscript {
    ext.cubaVersion = '7.2.8'
    repositories {
        maven {
            url 'https://dl.bintray.com/cuba-platform/main'


        }
        jcenter()
```

```
    }
    dependencies {
        classpath "com.haulmont.gradle:cuba-plugin:$cubaVersion"
    }
}

def modulePrefix = 'app'

def globalModule = project(":${modulePrefix}-global")
def coreModule = project(":${modulePrefix}-core")
def webModule = project(":${modulePrefix}-web")

def servletApi = 'javax.servlet:javax.servlet-api:3.1.0'

apply(plugin: 'cuba')

cuba {
    artifact {
        group = 'com.haulmont.demo.dashboard'
        version = '0.1'
        isSnapshot = true
    }
    tomcat {
        dir = "$project.rootDir/deploy/tomcat"
    }
}
```

```
dependencies {
    appComponent("com.haulmont.cuba:cuba-global:$cubaVersion")
    appComponent("com.haulmont.reports:reports-global:$cubaVersion")
    appComponent("com.haulmont.charts:charts-global:$cubaVersion")
    appComponent("com.haulmont.addon.dashboard:dashboard-
global:3.2.0.BETA1")
    appComponent("com.haulmont.addon.dashboardchart:dashboardchart-
global:1.3.0.BETA1")

}

def hsql = 'org.hsqldb:hsqldb:2.4.1'

configure([globalModule, coreModule, webModule]) {
    apply(plugin: 'java')
    apply(plugin: 'maven')
    apply(plugin: 'cuba')

    dependencies {
        testCompile('junit:junit:4.12')
    }

    task sourceJar(type: Jar) {
        from file('src')
        classifier = 'sources'
    }

    artifacts {
        archives sourceJar
```

```
    }
}

configure(globalModule) {
    dependencies {
        if (!JavaVersion.current().isJava8()) {
            runtime('javax.xml.bind:jaxb-api:2.3.1')
            runtime('org.glassfish.jaxb:jaxb-runtime:2.3.1')
        }
    }
    entitiesEnhancing {
        main { enabled = true }
    }

}

configure(coreModule) {

    configurations {
        jdbc
        dbscripts
    }

    dependencies {
        compile(globalModule)
        compile('org.apache.commons:commons-dbcp2:2.5.0')
        compileOnly(servletApi)
        jdbc(hsql)
```

```
    testRuntime(hsql)

}

task cleanConf(description: 'Cleans up conf directory') {
    doLast {
        def dir = new File(cuba.tomcat.dir, "/conf/${modulePrefix}-core")
        if (dir.isDirectory()) {
            ant.delete(includeemptydirs: true) {
                fileset(dir: dir, includes: '**/*', excludes: 'local.app.properties')
            }
        }
    }
}

task deploy(dependsOn: [assemble, cleanConf], type: CubaDeployment) {
    appName = "${modulePrefix}-core"
    appJars(modulePrefix + '-global', modulePrefix + '-core')
}

task createDb(dependsOn: assembleDbScripts, description: 'Creates local
database', type: CubaDbCreation) {
    dbms = 'hsql'
    host = 'localhost:9010'
    dbName = 'dashboards'
    dbUser = 'sa'
    dbPassword = ''
}
```

```
    task updateDb(dependsOn: assembleDbScripts, description: 'Updates local
database', type: CubaDbUpdate) {
        dbms = 'hsql'
        host = 'localhost:9010'
        dbName = 'dashboards'
        dbUser = 'sa'
        dbPassword = ''
    }
}

configure(webModule) {
    configurations {
        webcontent
    }

    dependencies {
        compileOnly(servletApi)
        compile(globalModule)

    }

    task webArchive(type: Zip) {
        from file("$buildDir/web")
        from file('web')
        classifier = 'web'
    }

    artifacts {
        archives webArchive
```

```
}

task deployConf(type: Copy) {
    from file('src')
    include "com/haulmont/demo/dashboard/**"
    into "$cuba.tomcat.dir/conf/${modulePrefix}"
}

task clearMessagesCache(type: CubaClearMessagesCache) {
    appName = "${modulePrefix}"
}
deployConf.dependsOn clearMessagesCache

task cleanConf(description: 'Cleans up conf directory') {
    doLast {
        def dir = new File(cuba.tomcat.dir, "/conf/${modulePrefix}")
        if (dir.isDirectory()) {
            ant.delete(includeemptydirs: true) {
                fileset(dir: dir, includes: '**/*', excludes: 'local.app.properties')
            }
        }
    }
}

task deploy(dependsOn: [assemble, cleanConf], type: CubaDeployment) {
    appName = "${modulePrefix}"
    appJars(modulePrefix + '-global', modulePrefix + '-web')
}
task buildScssThemes(type: CubaWebScssThemeCreation)
```

```
    task deployThemes(type: CubaDeployThemeTask, dependsOn:
buildScssThemes)
    assemble.dependsOn buildScssThemes


    task themesJar(type: Jar) {
        from file('themes')
        classifier = 'themes'
    }

    artifacts {
        archives themesJar
    }
}

task undeploy(type: Delete, dependsOn: ":${modulePrefix}-web:cleanConf") {
    delete("$cuba.tomcat.dir/shared")
    delete("$cuba.tomcat.dir/webapps/${modulePrefix}-core")
    delete("$cuba.tomcat.dir/webapps/${modulePrefix}")
}

task restart(dependsOn: ['stop', ":${modulePrefix}-core:deploy",
":${modulePrefix}-web:deploy"], description: 'Redeploys applications and restarts
local Tomcat') {
    doLast {
        ant.waitfor(maxwait: 6, maxwaitunit: 'second', checkevery: 2, checkeveryunit:
'second') {
            not {
                socket(server: 'localhost', port: '8787')
            }
        }
```

```
        start.execute()
    }
}

task buildUberJar(type: CubaUberJarBuilding) {
    appProperties = ['cuba.automaticDatabaseUpdate': false]
    coreJettyEnvPath = 'modules/core/web/META-INF/jetty-env.xml'
    logbackConfigurationFile = 'etc/uber-jar-logback.xml'
    singleJar = true
}
```

App.poperties

```
##########################################################################
#                    Configuration                                       #
##########################################################################

cuba.dbmsType = hsql

cuba.springContextConfig = +com/haulmont/demo/dashboard/spring.xml

cuba.persistenceConfig = +com/haulmont/demo/dashboard/persistence.xml

cuba.metadataConfig = +com/haulmont/demo/dashboard/metadata.xml

cuba.viewsConfig = +com/haulmont/demo/dashboard/views.xml

cuba.mainMessagePack = +com.haulmont.demo.dashboard.core
```

```
cuba.keyForSecurityTokenEncryption = ds10qjUyzOhhWJ86

cuba.anonymousSessionId = 9decfc83-98b6-50d4-f959-576c36792da1

################################################################################
#                       Other                                  #
################################################################################

cuba.webContextName = app-core
cuba.availableLocales = English|en
cuba.localeSelectVisible = false
cuba.restApiUrl=http://localhost:8080/app-portal/api
cuba.webAppUrl=http://localhost:8080/app
cuba.dataSourceProvider=jndi
```

spring.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context">

    <!-- Annotation-based beans -->
    <context:component-scan base-package="com.haulmont.demo.dashboard"/>

</beans>
```

Route.java

```
import com.haulmont.chile.core.annotations.MetaClass;
import com.haulmont.chile.core.annotations.MetaProperty;
import com.haulmont.cuba.core.entity.BaseUuidEntity;

@MetaClass(name = "demo$Route")
public class Route extends BaseUuidEntity {
    private static final long serialVersionUID = 121463765559033258L;

    @MetaProperty
    protected String time;

    @MetaProperty
    protected String arrival;

    @MetaProperty
    protected String departure;

    public Route(String time, String departure, String arrival) {
        this.time = time;
        this.departure = departure;
        this.arrival = arrival;
    }

    public void setTime(String time) {
        this.time = time;
    }

    public String getTime() {
        return time;
```

```
    }

    public void setArrival(String arrival) {
        this.arrival = arrival;
    }

    public String getArrival() {
        return arrival;
    }

    public void setDeparture(String departure) {
        this.departure = departure;
    }

    public String getDeparture() {
        return departure;
    }
}
```

OrderTaxi.java

```
import com.haulmont.chile.core.annotations.NamePattern;
import com.haulmont.cuba.core.entity.StandardEntity;
import com.haulmont.cuba.security.entity.User;

import javax.persistence.*;
import java.time.LocalDateTime;

@Table(name = "DASHBOARDDEMO_ORDER_TAXI")
@Entity(name = "dashboarddemo_OrderTaxi")
```

```
@NamePattern("%s %s|satrtAddress,endAddress")
public class OrderTaxi extends StandardEntity {
    private static final long serialVersionUID = 45461799527943392O5L;

    @Column(name = "SATRT_ADDRESS", length = 2048)
    private String satrtAddress;

    @Column(name = "END_ADDRESS", length = 2048)
    private String endAddress;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "DRIVER_ID")
    private User driver;

    @Column(name = "IS_COLLECTIVE_ORDER")
    private Boolean isCollectiveOrder;

    @Column(name = "CREATE_ORDER_TS")
    private LocalDateTime createOrderTs;

    @Column(name = "DRIVER_ADVENT_TS")
    private LocalDateTime driverAdventTs;

    @Column(name = "DRIVER_DELIVERY_TS")
    private LocalDateTime driverDeliveryTs;

    public LocalDateTime getDriverDeliveryTs() {
        return driverDeliveryTs;
    }
```

```
public void setDriverDeliveryTs(LocalDateTime driverDeliveryTs) {
    this.driverDeliveryTs = driverDeliveryTs;
}

public LocalDateTime getDriverAdventTs() {
    return driverAdventTs;
}

public void setDriverAdventTs(LocalDateTime driverAdventTs) {
    this.driverAdventTs = driverAdventTs;
}

public LocalDateTime getCreateOrderTs() {
    return createOrderTs;
}

public void setCreateOrderTs(LocalDateTime createOrderTs) {
    this.createOrderTs = createOrderTs;
}

public Boolean getIsCollectiveOrder() {
    return isCollectiveOrder;
}

public void setIsCollectiveOrder(Boolean isCollectiveOrder) {
    this.isCollectiveOrder = isCollectiveOrder;
}
```

```
    public User getDriver() {
        return driver;
    }

    public void setDriver(User driver) {
        this.driver = driver;
    }

    public String getEndAddress() {
        return endAddress;
    }

    public void setEndAddress(String endAddress) {
        this.endAddress = endAddress;
    }

    public String getSatrtAddress() {
        return satrtAddress;
    }

    public void setSatrtAddress(String satrtAddress) {
        this.satrtAddress = satrtAddress;
    }
}
```

Web-app.properties

```
#################################################################################
#                       Configuration                           #
```

```
##################################################################################
```

```
cuba.springContextConfig = +com/haulmont/demo/dashboard/web-spring.xml
```

```
cuba.dispatcherSpringContextConfig = +com/haulmont/demo/dashboard/web-dispatcher-spring.xml
```

```
cuba.persistenceConfig = +com/haulmont/demo/dashboard/persistence.xml
```

```
cuba.metadataConfig = +com/haulmont/demo/dashboard/metadata.xml
```

```
cuba.viewsConfig = +com/haulmont/demo/dashboard/views.xml
```

```
cuba.windowConfig = +com/haulmont/demo/dashboard/web-screens.xml
```

```
cuba.menuConfig = +com/haulmont/demo/dashboard/web-menu.xml
```

```
cuba.permissionConfig = +com/haulmont/demo/dashboard/web-permissions.xml
```

```
cuba.mainMessagePack = +com.haulmont.demo.dashboard.web
```

```
cuba.anonymousSessionId = 9decfc83-98b6-50d4-f959-576c36792da1
```

```
cuba.creditsConfig = +
```

```
##################################################################################
#                                Other                                           #
```

```
##################################################################################
```

```
# Middleware connection
cuba.connectionUrlList = http://localhost:8080/app-core

# Set to false if the middleware works on different JVM
cuba.useLocalServiceInvocation = true

cuba.webContextName = app
cuba.webAppUrl = http://localhost:8080/app

cuba.availableLocales = English|en
cuba.localeSelectVisible = false
cuba.web.theme = hover

cuba.themeConfig = com/haulmont/cuba/havana-theme.properties
com/haulmont/cuba/halo-theme.properties com/haulmont/cuba/hover-
theme.properties

cuba.web.mainScreenId = extMainScreen
cuba.restApiUrl=http://localhost:8080/app-portal/api
cuba.web.loginDialogPoweredByLinkVisible=false
```

web-menu.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<menu-config xmlns="http://schemas.haulmont.com/cuba/menu.xsd">
    <item screen="dashboard-demo$json-screen"/>
    <menu id="application-taxi" insertBefore="administration"
caption="mainMsg://taxi.user.menu">
```

```
    <item screen="dashboarddemo_OrderTaxi.browse"
caption="mainMsg://menu_config.dashboarddemo_OrderTaxi.browse"/>
    <item screen="dashboarddemo_OrderTaxi.edit"
caption="mainMsg://menu_config.dashboarddemo_OrderTaxi.edit"/>
  </menu>
</menu-config>
```

OrderTaxiBrowser.java

```
package com.haulmont.demo.dashboard.web.screens.ordertaxi;

import com.haulmont.cuba.gui.screen.*;
import com.haulmont.demo.dashboard.entity.OrderTaxi;

@UiController("dashboarddemo_OrderTaxi.browse")
@UiDescriptor("order-taxi-browse.xml")
@LookupComponent("orderTaxisTable")
@LoadDataBeforeShow
public class OrderTaxiBrowse extends StandardLookup<OrderTaxi> {
}
```

OrderTaxiEdit.java

```
package com.haulmont.demo.dashboard.web.screens.ordertaxi;

import com.haulmont.cuba.gui.Notifications;
import com.haulmont.cuba.gui.screen.*;
import com.haulmont.demo.dashboard.entity.OrderTaxi;

import javax.inject.Inject;
```

```
@UiController("dashboarddemo_OrderTaxi.edit")
@UiDescriptor("order-taxi-edit.xml")
@EditedEntityContainer("orderTaxiDc")
@LoadDataBeforeShow
public class OrderTaxiEdit extends StandardEditor<OrderTaxi> {

    @Inject private Notifications notifications;

    @Subscribe
    public void onAfterClose(AfterCloseEvent event) {

notifications.create(Notifications.NotificationType.HUMANIZED).withCaption("
Your order has been processed. Please wait.").show();
    }
}
```

Ordertaxi-browser.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<window xmlns="http://schemas.haulmont.com/cuba/screen/window.xsd"
        xmlns:c="http://schemas.haulmont.com/cuba/screen/jpql_condition.xsd"
        caption="msg://orderTaxiBrowse.caption"
        focusComponent="orderTaxisTable"
        messagesPack="com.haulmont.demo.dashboard.web.screens.ordertaxi">
    <data readOnly="true">
        <collection id="orderTaxisDc"
                    class="com.haulmont.demo.dashboard.entity.OrderTaxi">
            <view extends="_local">
                <property name="driver" view="_minimal">
                    <property name="firstName"/>
```

```
            <property name="lastName"/>
        </property>
    </view>
    <loader id="orderTaxisDl">
        <query>
            <![CDATA[select e from dashboarddemo_OrderTaxi e]]>
        </query>
    </loader>
  </collection>
</data>
<dialogMode height="600"
        width="800"/>
<layout expand="orderTaxisTable"
    spacing="true">
  <filter id="filter"
        applyTo="orderTaxisTable"
        dataLoader="orderTaxisDl">
    <properties include=".*"/>
  </filter>
  <groupTable id="orderTaxisTable"
            width="100%"
            dataContainer="orderTaxisDc">
    <actions>
        <action id="create" type="create"/>
        <action id="edit" type="edit"/>
        <action id="remove" type="remove"/>
        <action id="refresh" type="refresh"/>
        <action id="excel" type="excel"/>
    </actions>
```

```
        <columns>
            <column id="satrtAddress"/>
            <column id="endAddress"/>
            <column id="driver"/>
            <column id="isCollectiveOrder"/>
            <column id="createOrderTs"/>
            <column id="driverAdventTs"/>
            <column id="driverDeliveryTs"/>
        </columns>
        <rowsCount/>
        <buttonsPanel id="buttonsPanel"
                      alwaysVisible="true">
            <button id="createBtn" action="orderTaxisTable.create"/>
            <button id="editBtn" action="orderTaxisTable.edit"/>
            <button id="removeBtn" action="orderTaxisTable.remove"/>
            <button id="refreshBtn" action="orderTaxisTable.refresh"/>
            <button id="excelBtn" action="orderTaxisTable.excel"/>
        </buttonsPanel>
    </groupTable>
    <hbox id="lookupActions" spacing="true" visible="false">
        <button action="lookupSelectAction"/>
        <button action="lookupCancelAction"/>
    </hbox>
  </layout>
</window>
```

Ordertaxi-edit.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<window xmlns="http://schemas.haulmont.com/cuba/screen/window.xsd"
```

```
    xmlns:chart="http://schemas.haulmont.com/charts/charts.xsd"
    caption="msg://orderTaxiEdit.caption"
    focusComponent="form"
    messagesPack="com.haulmont.demo.dashboard.web.screens.ordertaxi">
<data>
    <instance id="orderTaxiDc"
            class="com.haulmont.demo.dashboard.entity.OrderTaxi">
        <view extends="_local">
            <property name="driver" view="_minimal">
                <property name="firstName"/>
                <property name="lastName"/>
            </property>
        </view>
        <loader/>
    </instance>
</data>
<dialogMode height="AUTO"
            width="AUTO"
            modal="true"
            forceDialog="true"/>
<layout expand="editActions" spacing="true">
    <form id="form" dataContainer="orderTaxiDc">
        <column width="350px">
            <textField id="satrtAddressField" property="satrtAddress"/>
            <label value="Or select point in map" align="MIDDLE_CENTER"/>
            <vbox id="mapBox" height="100%">
                <chart:mapViewer id="map" width="100%" height="100%"/>
            </vbox>
            <textField id="endAddressField" property="endAddress"/>
```

```
            <label value="Or select point in map" align="MIDDLE_CENTER"/>
            <vbox id="mapBox2" height="100%">
               <chart:mapViewer id="map2" width="100%" height="100%"/>
            </vbox>
            <checkBox id="isCollectiveOrderField" property="isCollectiveOrder"/>
         </column>
      </form>
      <hbox id="editActions" spacing="true">
         <button action="windowCommitAndClose"/>
         <button action="windowClose"/>
      </hbox>
   </layout>
</window>
```