Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

за спеціальністю 123 «Комп'ютерна інженерія»

Тема наукової роботи: ІНФОРМАЦІЙНА СИСТЕМА КЕРУВАННЯ СЛУЖБОЮ ТАКСІ

Виконала _____ І. О. Поспєлова

Керівник роботи _____ І. О. Музика

Нормоконтроль _____ Д. І. Кузнєцов

Завідувач кафедри _____ А. І. Купін

Кривий Ріг
2023

Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Ступінь вищої освіти магістр
Спеціальність 123 «Комп'ютерна інженерія»

**ЗАТВЕРДЖУЮ**
Завідувач кафедри, голова циклової комісії

_____ А. І. Купін

"____" _____ 20__ року

# З А В Д А Н Н Я
## НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

_____
(прізвище, ім'я, по батькові)

1. Тема роботи _____
_____
_____

керівник роботи _____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "___"_____ 20__року №__

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи _____
_____
_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_____
_____
_____
_____
_____
_____

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
_____
_____
_____
_____

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання_____

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів магістерської роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**Студент** _____ _____
(підпис) (прізвище та ініціали)

**Керівник роботи** _____ _____
(підпис) (прізвище та ініціали)

# РЕФЕРАТ

Пояснювальна записка: 90 сторінок, 26 рисунків, 4 таблиці, 2 додатки, 26 використаних джерел.

Об’єкт дослідження – інформаційна система керування службою таксі.

Робота складається з чотирьох розділів.

Перший розділ присвячено аналізу існуючих служб таксі за допомогою евристичних методів. Зроблено огляд сучасних, існуючих підходів аналізу конкурентів та їх пропозицій. Обґрунтовано необхідність розробки інформаційної системи керування службою таксі. Висвітлено питання актуальності використання служб таксі серед повсякденних потреб користувачів.

Другий розділ розкриває питання алгоритму розвя’зання поставленої задачі Проведено моделювання, розроблено математичну модель процесу передачі даних через клієнт-серверну систему.

У третьому розділі виконано описання інформаційного забезпечення та розроблено бази даних програми інформаційної системи керування службою таксі.

Четвертий розділ полягав у безпосередній розробці програмного забезпечення, представленні зображень головних форм, створенню класів логіки додатків та розробці методів роботи програми.

Ключові слова: КОМП’ЮТЕРНА МЕРЕЖА, МАРШРУТИЗАТОР, СЕРВЕР, ПРОГРАМА, ІНФОРМАЦІЙНА СИСТЕМА.

Master’s work: 90 pages, 26 figures, 4 tables, 2 appendices, 26 used sources.

The object of the study is the taxi service management information system.

The work consists of four sections.

The first chapter is devoted to the analysis of existing taxi services using heuristic methods. An overview of modern, existing approaches to the analysis of competitors and their offers was made. The need to develop an information system for taxi service management is substantiated. The question of the relevance of using taxi services among the daily needs of users is highlighted.

The second section reveals the issue of the algorithm for solving the given problem. Simulation was carried out, a mathematical model of the process of data transmission through the client-server system was developed.

In the third chapter, the description of the information support was performed and the databases of the taxi service management information system program were developed.

The fourth section consisted in the direct development of the software, the presentation of images of the main forms, the creation of classes of application logic and the development of methods of operation of the program.

Keywords: COMPUTER NETWORK, ROUTER, SERVER, PROGRAM, INFORMATION SYSTEM.

**ЗМІСТ**

# ВСТУП

Сьогодні в усьому світі існує розвинена сфера послуг, яка виступає драйвером економічного розвитку багатьох країн світу. На цьогоддення існує багато видів послуг, однією з них є сфера надання послуг з перевезення пасажирів, яка включає перевезення людей на таксі. Зараз цей вид послуг є першим у списку з найбільш затребуваних у всіх країнах з розвиненою інфраструктурою. Таксисти є абсолютно в будь-якому досить великому місті, де є відстань між важливими об'єктами інфраструктури. Вони завжди використовували технології, орієнтовані на клієнтів, щоб спілкуватися з клієнтами, і ці технології постійно розвиваються.

Наразі кожен має при собі телефон для спілкування, отримання інформації, розваг, замовлення послуг тощо. Виклик таксі – лише одна з таких послуг. На ринку існує велика кількість різноманітного програмного забезпечення (Uber, Uklon, Bolt, Shark та інші сервіси) для виклику таксі. Деякі структури дзвінків все ще базуються на старій перевіреній системі з диспетчерською службою, так, дзвінок по телефону. Хоча ця система в певній формі склалася. Диспетчерські служби таксі існують дуже давно, і за час їх існування змінилися принципи передачі замовлень таксистам, але незмінною залишилася їх суть: вони приймають замовлення на транспортні послуги і передають їх безпосередньо таксистам.


Рисунок 1 – Таксі

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | КНУ.РМ.123.23.02.В | | |
| Змн. | Арк. | № документа | Підпис | Дата | | | |
| Розробив | | Поспєлова | | | ВСТУП | Літера | Аркуш | Аркушів |
| Перевірив | | Музика | | | | | | |
| | | | | | | ЗКІ-22м | | |
| Н.контроль | | Кузнєцов | | | | | | |
| Затвердив | | Купін | | | | | | |

Перша ітерація таких сервісів була спрямована на передачу замовлень по раціях, диспетчери отримували замовлення та передавали їх таксистам, таксист, який був ближче до замовлення, повідомляв диспетчерам, що він вільний і знаходиться близько до клієнта, тому замовлення було заброньовано для нього.

Згодом з'явилися мобільні телефони і робота таких служб дещо спростилася, наприклад, інформацію про замовлення можна було надсилати таксистам за допомогою SMS, що досить добре прискорювало роботу послуг.

Сьогодні такі системи побудовані на системах зв'язку «клієнт-сервер»: є сервер диспетчерської, є клієнти, диспетчери, які на основі замовлень клієнтів створюють замовлення у своїх клієнтських програмах, які в свою чергу передають їх на сервер, який передає інформацію про порядок в ефірі клієнтської програми на пристроях Android. Далі таксисти вирішують прийняти своє замовлення в залежності від того, доцільно воно чи ні, а саме замовлення буде в ефірі, поки диспетчери його звідти не заберуть або не візьмуть на виконання.

**Метою кваліфікаційної роботи є** аналіз поточного програмного забезпечення на ринку та розробка системи виклику таксі, яка в свою чергу буде практично кращою за те, що є на сьогодні. Тобто розроблене програмне забезпечення має виключити з існуючої системи диспетчерську службу, що є морально застарілим методом вирішення транспортної проблеми, а також є додатковим тягарем, який здорожує утримання служби таксі.

Перше: необхідно досліджувати можливе зменшення витрат на диспетчерську службу, її утримання: витрати на електроенергію, зношеність обладнання та зарплату диспетчерів, яка є найбільшою графою в цьому списку.

Друге: кожна диспетчерська має власний сервер для обміну даними між диспетчерами, які формують замовлення, і таксистами, у яких на смартфонах є спеціальне програмне забезпечення, яке дозволяє брати замовлення за плату від суми самого замовлення, що є основним способом дохід за послугу таксі.

Третє: необхідно проаналізувати методи, які дозволять максимально просто і швидко розгорнути таку систему для максимально швидкого виходу в робочий режим.

**Основним методом дослідження є** евристичний аналіз програмного забезпечення, наявного на ринку України. Програми, побудовані на системі диспетчерських служб, складають більшість у цій програмній ніші, але на ринку є й такі, які спрямовані на пряме замовлення транспортних послуг.

Аналіз подібних програм дозволить зрозуміти основу основного ринку надання послуг такого роду та методи вдосконалення, які можливо буде використати при розробці дипломного проекту програмного забезпечення. А ще вдосконалити свої знання у вивченні ринку надання послуг, що в свою чергу покращить уміння аналізувати, що необхідно для розробки програмного забезпечення будь-якого призначення.

Щоб краще зрозуміти мету завдання, необхідно зрозуміти, що в даному

випадку є предметом дослідження, а що об'єктом.

**Об’єктом дослідження є** процес управління транспортними послугами з використанням інформаційної системи.

**Предметом дослідження є** евристичний аналіз програмного забезпечення, представленого на ринку, яке дозволить зрозуміти принципи роботи, на яких базуються диспетчерські служби таксі та подібні сервіси в цілому.

Основним джерелом інформації для вивчення програмного забезпечення для таксистів є Google Market, де воно знаходиться у вільному доступі. А також на сайтах розробників програмного забезпечення для диспетчерських служб, які описують можливості свого продукту та принципи роботи.

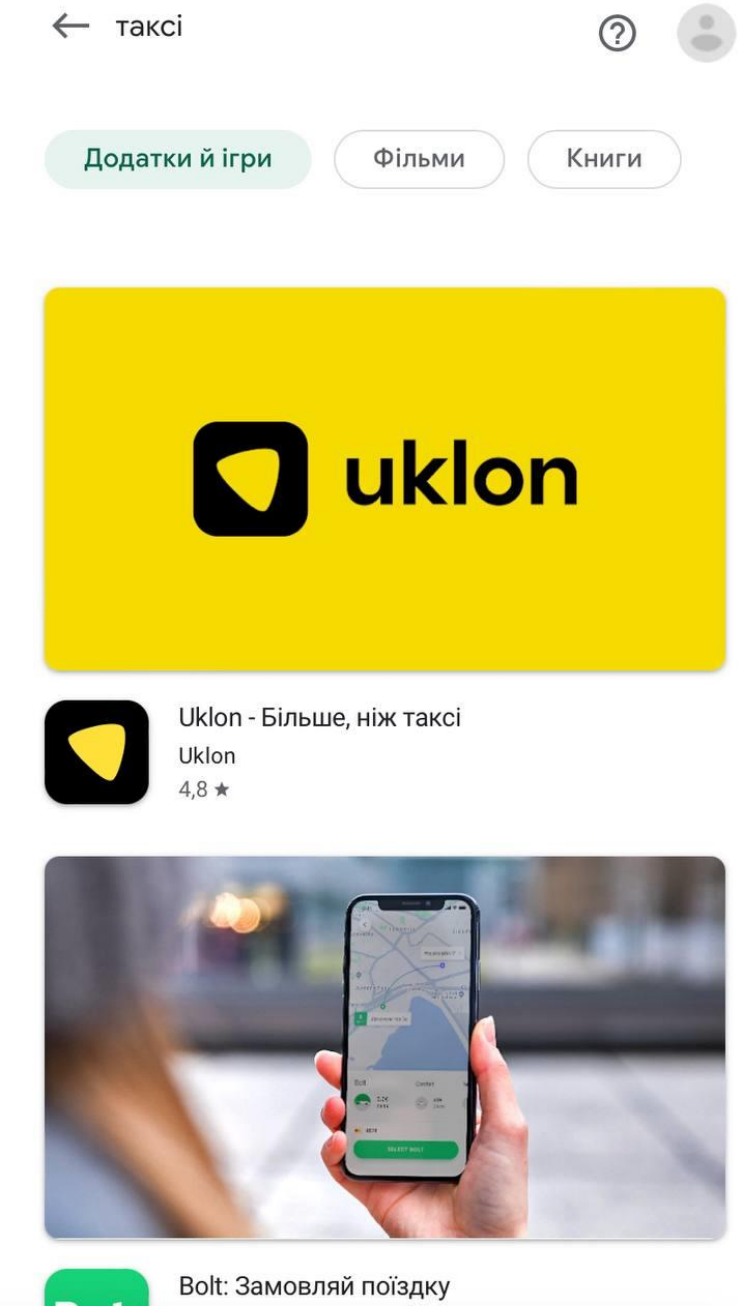


Рисунок 2 – Приклад джерела вивчення

# РОЗДІЛ 1
# АНАЛІЗ ІНФОРМАЦІЙНИХ СИСТЕМ СЛУЖБ ТАКСІ

## 1.1. Огляд структури існуючих аналогів

За час проходження обов'язкової переддипломної практики мені вдалося потрапити в одну із диспетчерських служб таксі при підприємстві в місті Кривий Ріг, а саме диспетчерська службового таксі на ПАТ «АрселорМіттал Кривий Ріг».

На місці роботи будо досить багато техніки:

- Сервер (на ньому була розміщена серверна частина диспетчерської та сервер самої бази даних).
- Робочі станції (використовувались диспетчерами для прийому викликів від клієнтів, створення замовлень та зв'язку з водіями для їх координації).
- Маршутизатор (служив для підключення до інтернету та маршутизації запросів клієнтських додатків таксистів від зовнішньої IP-адреси до серверу).
- GSM-шлюзи (використовували протоколи IP-телефонії для прийому дзвінків та переадресації на вільних диспетчерів).

На сервері було розташовано програму від компанії «EVOS» під назвою «Таксі навігатор». Дана програма використовувалась для зв'язку між диспетчерами і таксистами, вона одночасно підтримувала клієнт-серверний зв'язок між двома особами, й також використовувалась для зв'язку між клієнтами та базою даних.

Дана серверна програма є досить потужним і надійним рішенням для подібних цілей, хороший контроль за базою даних, наявність інструментів для покарання недобросовісних таксистів, потужний функціонал для створення різного роду звітів, цей програмний продукт існує вже досить довгий час, але як і усі програми має свої недоліки.

Наприклад карта розташована в правій частині вікна (рисунок 1.1) являється додатковим платним доповненням до серверної програми. Її функціонал закінчується на тому що вона позволяє виділяти на ній зони, наприклад мікрорайон, по яким працюють таксисти, та йде отримання замовлень. Хоча для такого роду дій достатньо підключити до програми за допомогою API Google Maps, який буде більш вигідний як з точки зору функціоналу, так із технічної точки зору, так як ця сервіс-карта постійно отримує обнови.

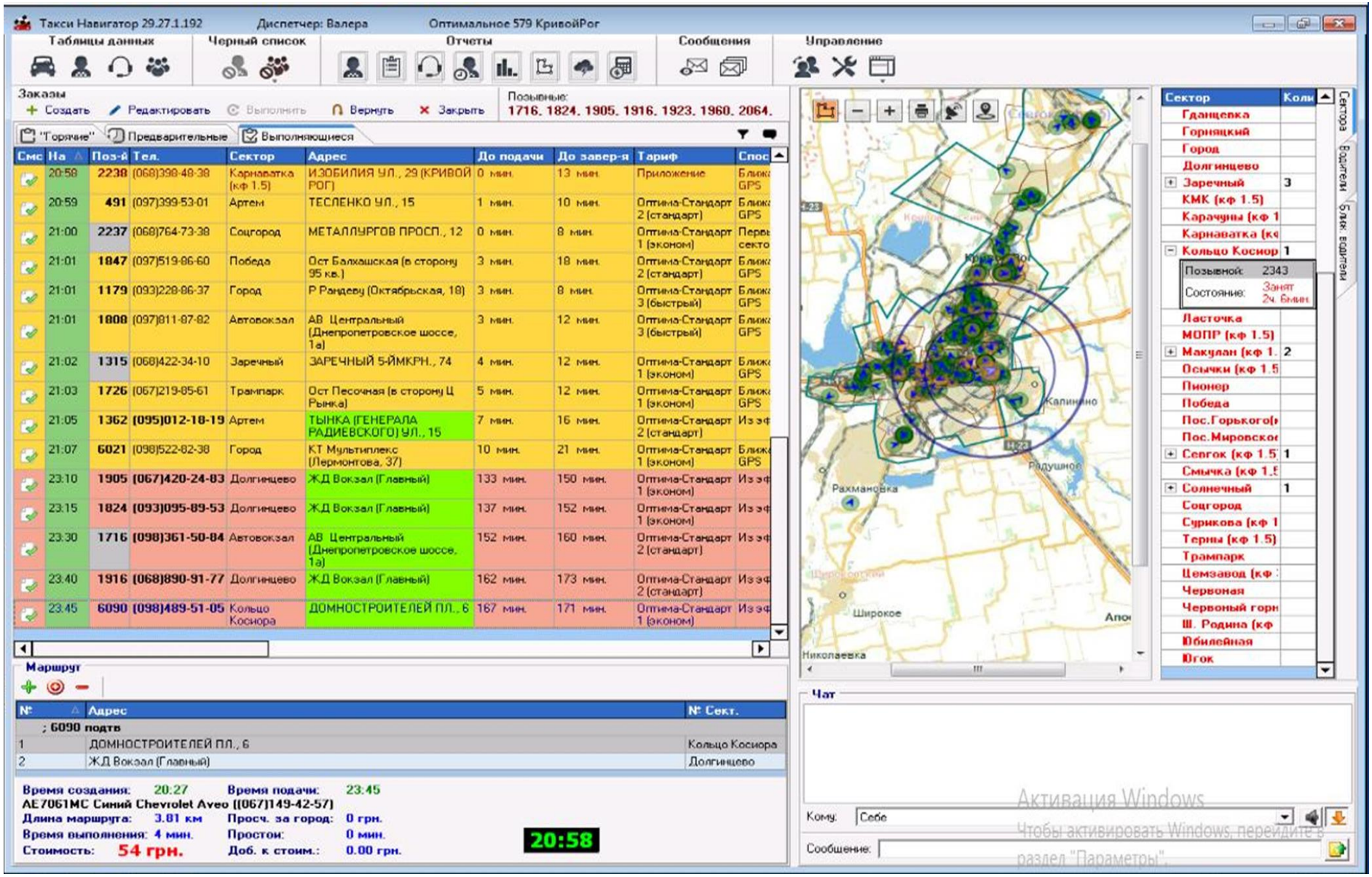


Рисунок 1.1 - Головне вікно серверної програми диспетчерської служби

За певний час перебування в диспетчерській було помічено деяку закономірність, а саме: серверний комп’ютер не досить потужний (Intel quad core 4-ри ядра, 8 Gb оперативної пам’яті), в той час, коли немає обміну повідомленнями сервера з клієнтами центральний процесор використовувався цією програмою постійно майже на чверть, що дає досить просту відповідь на внутрішню архітектуру сервера, цей сервер побудовано на багато-потоковій архітектурі з використанням безкінечного циклу «While» для слухача сереверу.

Проблема зі слухачем серевера не єдина, програма сама по собі дуже сильно перенавантажена різноманітними налаштуваннями, які в сьогоденні просто непотрібні. Прикладом можна привести використання налаштувань для рацій, які вже не використовуються більше 15 років. Також системи оплати та GPRS трекінгу сильно перевантажені налаштуваннями, які по суті непотрібні.

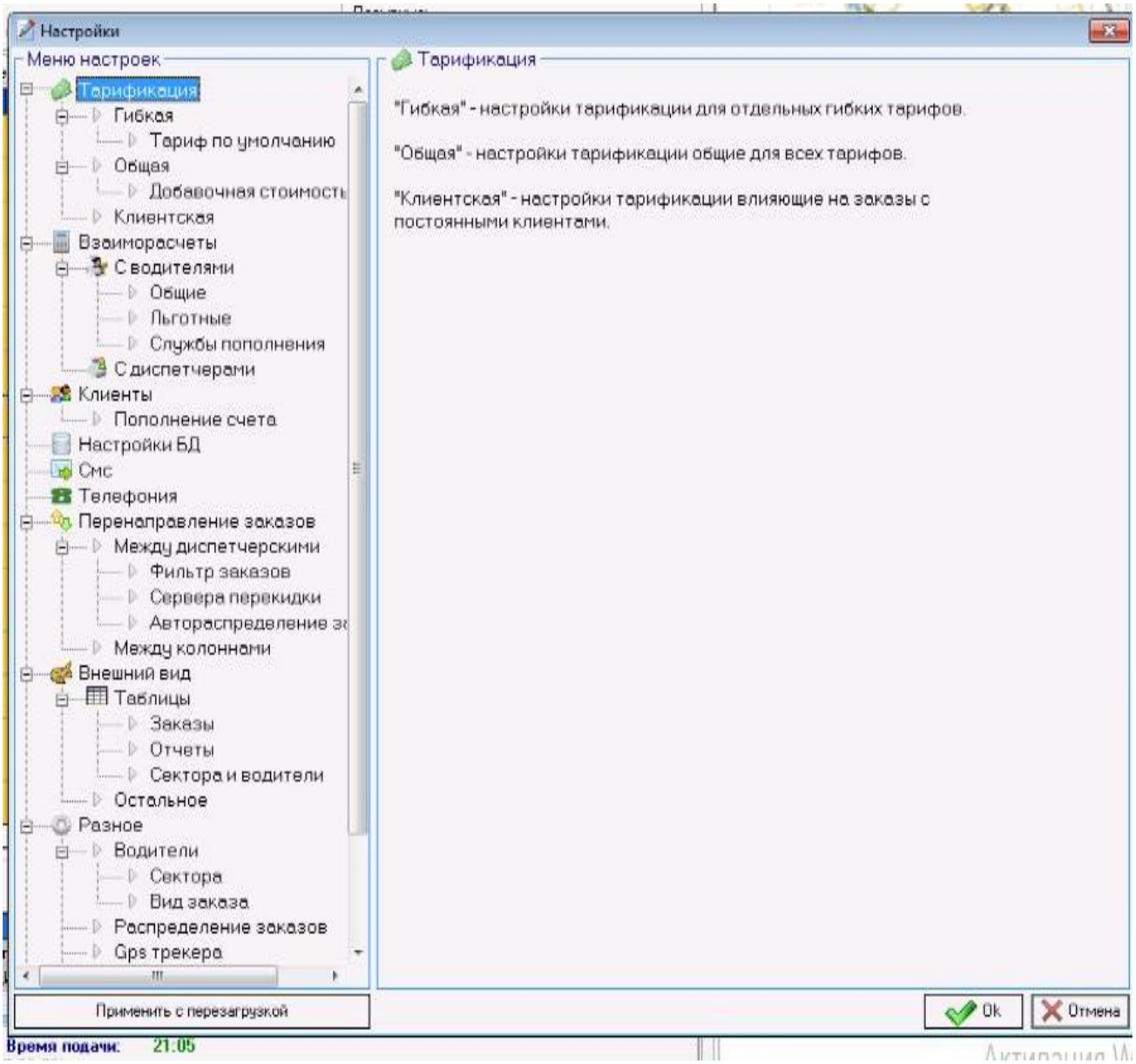


Рисунок 1.2 - Перелік налаштувань серверної програми

На сьогоднішній день відкривається досить багато молодих фірм які починають займатися подібним видом діяльності, якщо узагальнити, диспетчерська служба займається продажем інформації. Вони продають інформацію про те, де їх чекає клієнт, якому потрібні послуги транспортування. І кожній такій службі потрібне своє програмне забезпечення, але, якщо фірма молода і не має в своєму розпорядженні достатньо фінансів для оплати використання вище наведеного прикладу (який є досить дорогим), то вони почнуть шукати більш дешевий аналог програмного забезпечення для себе, але на ринку України подібного практично немає, що наводить на думку про перспективи створення даного продукту через відсутність завеликої конкуренції.

Плюсом є те, що розробка програмного рішення буде базуватись на клієнт-серверній моделі побудованій на асинхронній архітектурі з використанням гнучкої мови програмування C# 5.0, в якій закладена асинхронна модель з використанням ключових слів (async await), які заміняють старі методи «коллбеків» і роблять створення серверу в декілька разів простіше, ніж старі методи.

Оскільки програма передбачає базовий функціонал, це вирішує проблему з перевантаженням налаштувань і заодно робить її більш дешевою та простішою у користуванні за аналоги, що апріорі робить її більш конкурентоздатною на ринку програмного забезпечення.

Розробка покращеної системи для виклику таксі вимагає докладного аналізу наявного програмного забезпечення на ринку.

Евристичні методи — це евристика, або спосіб вирішення проблеми, який ґрунтується на досвіді та правилах високої ймовірності, але не забезпечує точного оптимального рішення. Однак вони будуть корисними для аналізу існуючого програмного забезпечення та розробки нової системи виклику таксі. Розглянемо евристичні методи на прикладі порівняння трьох додатків таксі: Uber, Uklon і Bolt. Ось деякі евристики, які будуть дуже корисними для цього аналізу та розробки вдосконаленої системи:

**1. Аналіз функціональності:**

По-перше, ретельно вивчаємо функціональні можливості доступних на ринку країни додатків для виклику таксі. Створіть список ключових функцій, які потрібні користувачам у такій програмі, включаючи бронювання таксі, визначення місцезнаходження, підтримку онлайн-платежів, відстеження таксі тощо.

Оцінюємо, наскільки існуючі програми відповідають цим функціональним вимогам. Визначаємо прогалини та слабкі сторони нашої системи, які можна покращити.

**2. Вивчення користувацького досвіду:**

Проаналізуймо відгуки користувачів і оцінки існуючих програм. Дізнаймося, на які проблеми чи слабкі сторони вони зазвичай звертають увагу, а також на які переваги вони прагнуть побачити.

Звертаймо увагу на запити та скарги користувачів. Дізнаймося, які аспекти взаємодії з користувачем не задовольняють користувачів і які можна покращити.

**3. Аналіз конкурентів:**

Досліджуємо існуючі системи виклику таксі, які вже успішно працюють на ринку. Уважно вивчаємо їх функціонал, інтерфейс, інноваційні рішення та особливості.

Розставляємо пріоритети для ключових переваг, які ви хочете отримати від нашої системи виклику таксі. Наприклад, це міг бути покращений користувальницький інтерфейс, надійність і чуйність, інноваційні функції тощо.

**4. Інновації та технології:**

Розгляньмо можливість використання новітніх технологій для покращення нашої системи. Наприклад, можливість використовувати штучний інтелект для прогнозування попиту, оптимізації маршрутів і покращення рекомендацій для користувачів.

Дізнаймося про інновації в безпеці даних і конфіденційності користувачів. Розгляньмо можливість інтеграції інших сервісів, наприклад спільного використання автомобілів, об’єднавши їх на одній платформі.

Після проведення цього аналізу ми матимемо чітке уявлення про те, як створити вдосконалену систему виклику таксі. Ми зможемо зосередитися на покращенні функціональності, користувацького досвіду та використання новітніх технологій для створення конкурентоспроможної системи, яка відповідає потребам користувачів.

За всіма описаними вище евристичними методами ми розберемо найвідоміші сервіси виклику таксі в Україні Uber, Uklon та Bolt.

Рисунок 1.3 – Логотипи сервісів виклику таксі

**1. Збір та аналіз відгуків користувачів:**

Uber:

Перегляд відгуків і рейтингів водіїв і користувачів на платформі Uber може надати важливу інформацію про задоволеність користувачів. Позитивні й негативні відгуки свідчать про якість обслуговування[27].

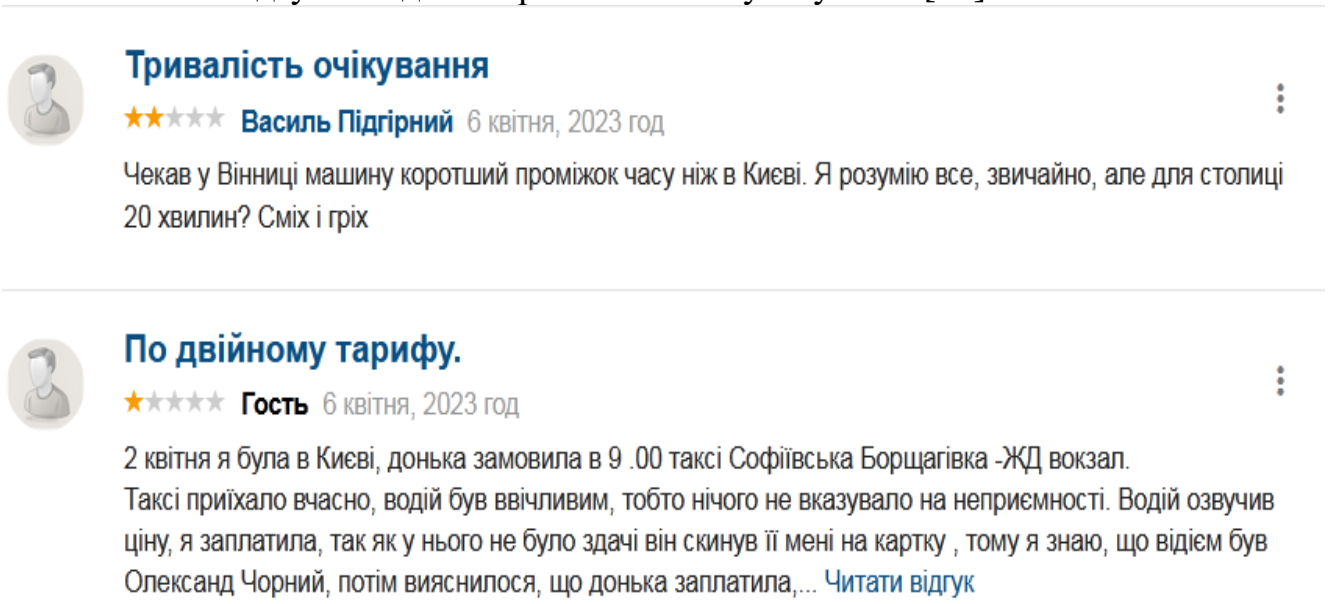


Рисунок 1.4 – Відгуки Uber

Uklon:

Те ж саме стосується і Uklon, де за оцінками водіїв і коментарями користувачів можна визначити якість послуги[27].

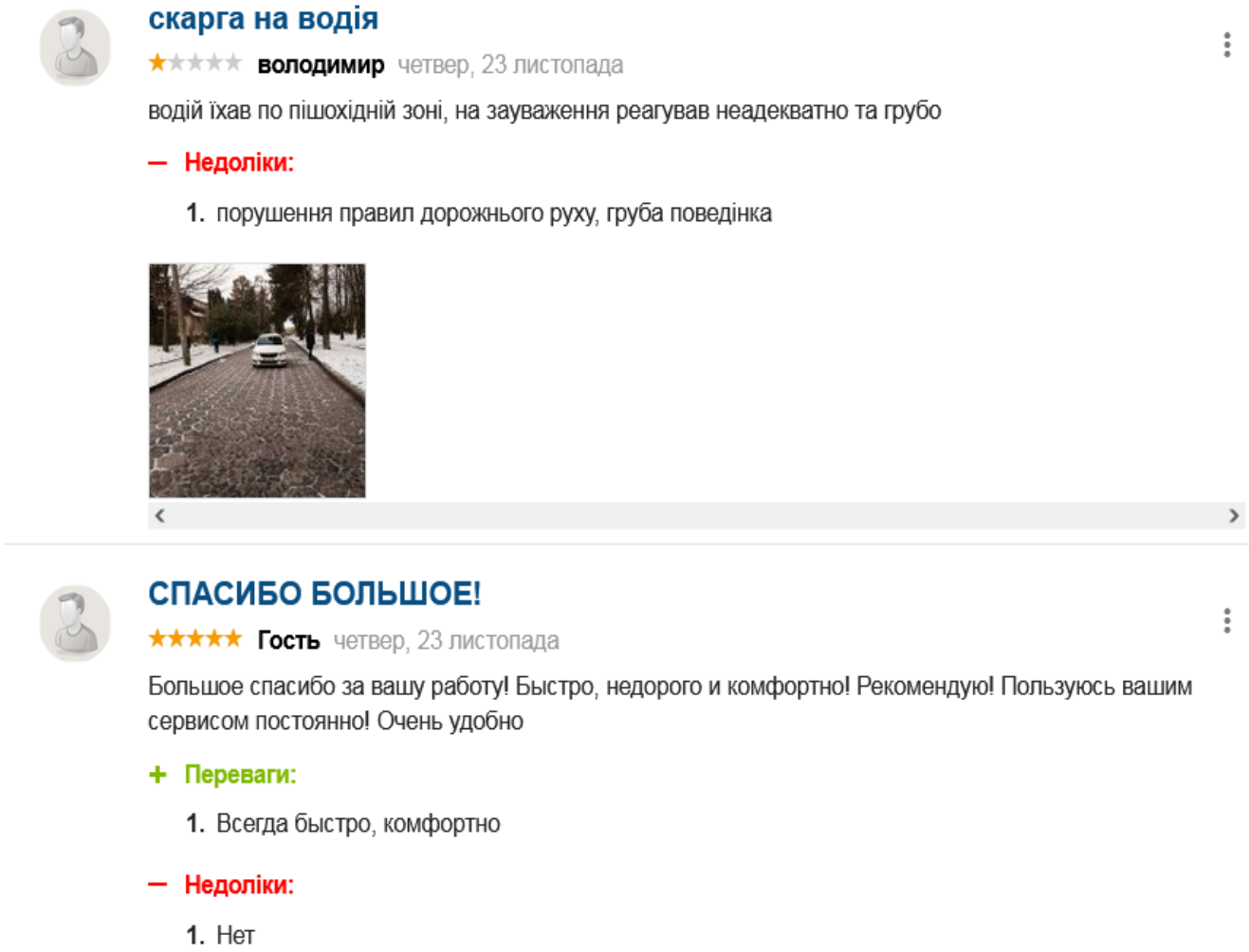


Рисунок 1.5 – Відгуки Uklon

Bolt:

Збір відгуків і оцінок користувачів на Bolt допомагає оцінити якість обслуговування та водіїв[27].


Рисунок 1.6 – Відгуки Bolt

Евристика: Аналіз великої кількості позитивних та незадоволених відгуків і всіх поставлених оцінок може свідчити про популярність і високу якість обслуговування на цій платформі.

**2. Аналіз функціональності та інтерфейсу:**

Uber:

Зручний інтерфейс додатку Uber, можливість вибору різноманітних видів послуг і висока автоматизація процесу замовлення роблять його популярним[28].

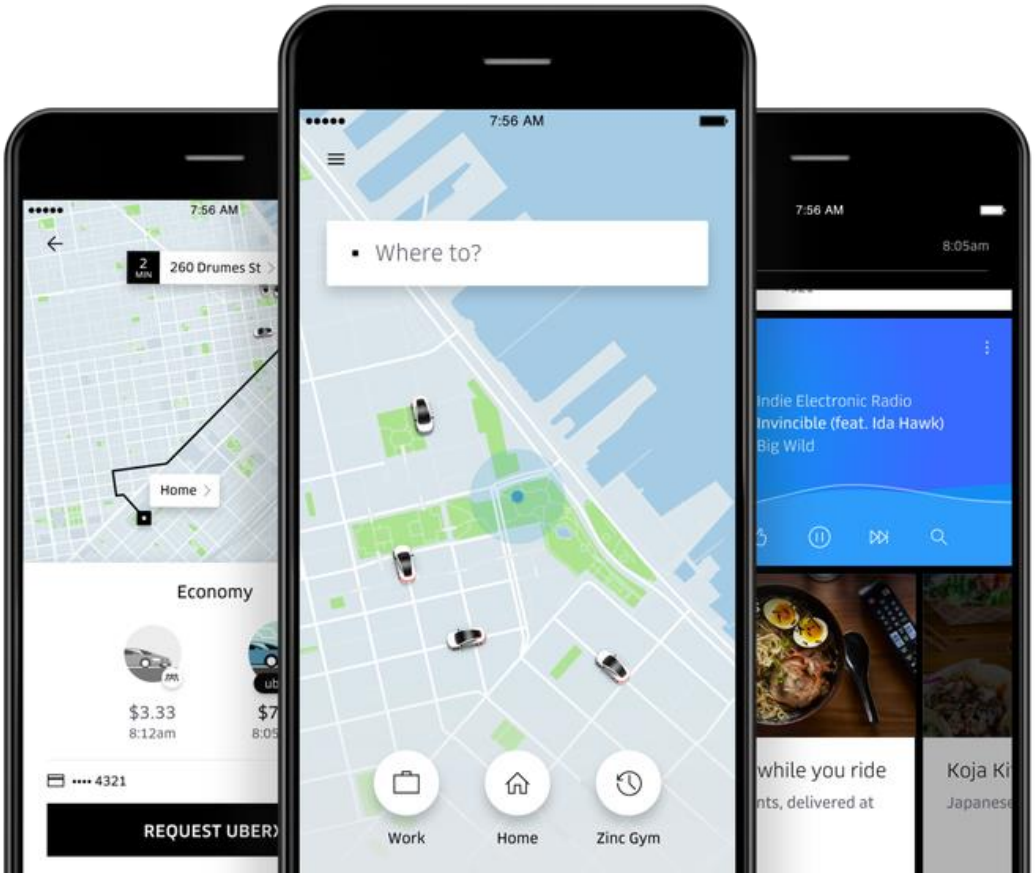


Рисунок 1.7 – Інтерфейс Uber

Uklon:

Uklon також пропонує зручний інтерфейс і можливість оплати онлайн. Вони також дозволяють окремим водіям реєструватися[29].

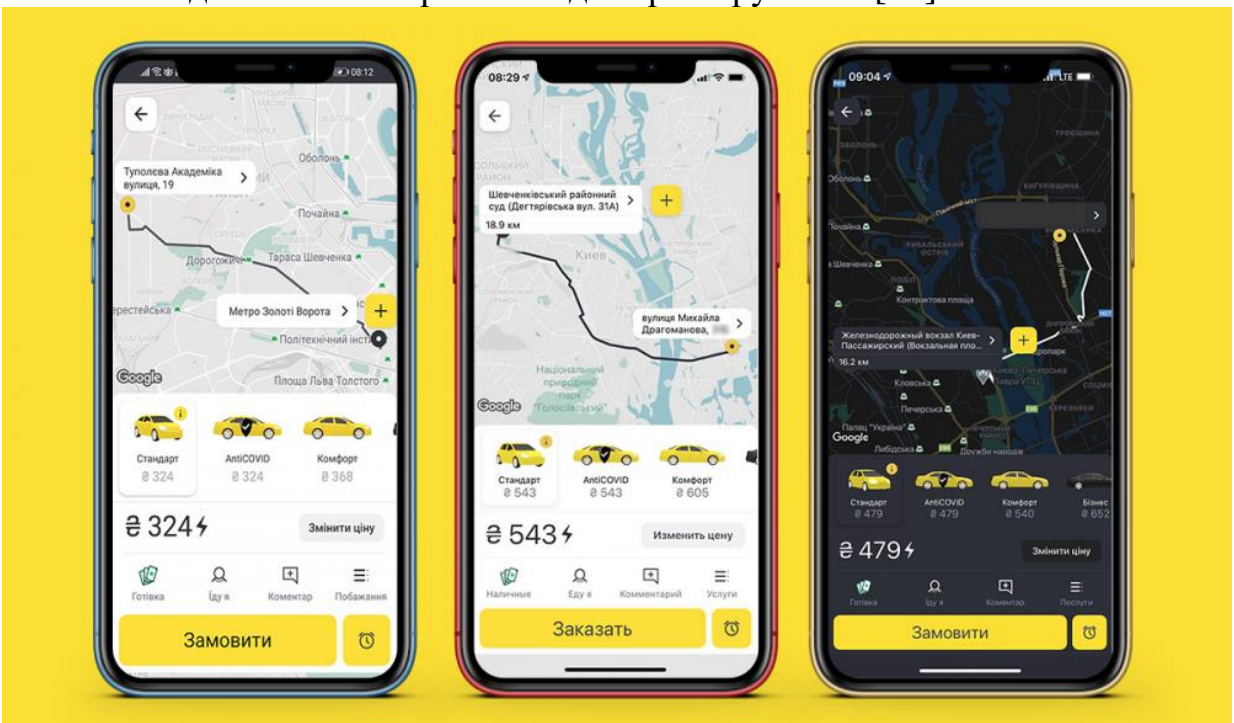


Рисунок 1.8 – Інтерфейс Uklon

Bolt:

Bolt відомий своєю швидкістю і простотою використання програми для замовлення послуг[30].

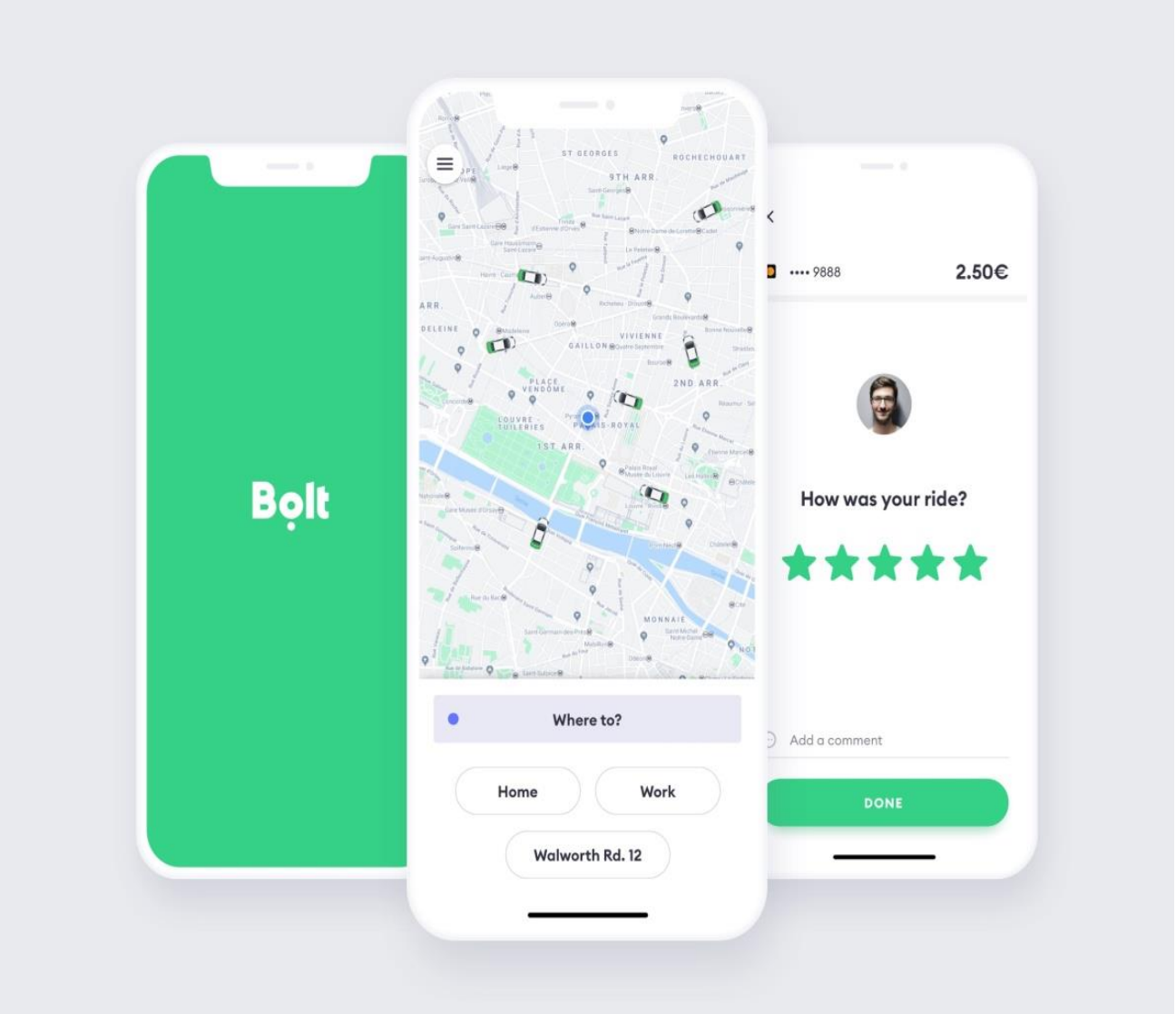


Рисунок 1.9 – Інтерфейс Bolt

Евристика: інтуїтивно зрозумілий і простий інтерфейс, а також швидкість обробки замовлень можуть бути ключовими факторами успіху нової програми.

**3. Аналіз цін і акцій:**

Uber:

Uber пропонує різноманітні акції, знижки та програму лояльності, що робить їхній сервіс привабливим для користувачів[28].


Рисунок 1.9 – Акції Uber

Uklon:

Uklon також проводить акції та знижки для користувачів[29].


Рисунок 1.10 – Акції Uklon

Болт:

Компанія Bolt виділяється конкурентними цінами на послуги.

Евристика: Аналіз цін і дій конкурентів на основі конкурентоспроможності та привабливості цін для власного застосування[30].

## Тарифи служби таксі Болт у 2023 році

(**Мінімальні ціни у 2023 році**: до трьох кілометрів, низька завантаженність доріг, нема збільшеного спросу)

| **Автомобіль/послуга** | **Ціни від** |
|---|---|
| **Bolt**<br>Швидкі, зручні, якісні поїздки | 72.00 UAH |
| **Economy**<br>Найдоступніший спосіб пересуватися по місту | 68.00 UAH |
| **XL**<br>6-місні автомобілі для великих груп або додаткового багажу | 137.00 UAH |
| **Доставка**<br>Швидка доставка | 77.00 UAH |
| **Pets**<br>Для поїздок с домашніми улюбленцями | 77.00 UAH |
| **Comfort+**<br>Поїздки на авто преміум-класу | 82.00 UAH |
| **Робочі поїздки**<br>Поїздки на роботу | 98.00 UAH |

Рисунок 1.11 – Ціни Bolt

**4. Аналіз маркетингових стратегій:**

Uber:

Uber знає про активну рекламу та спонсорські угоди з державними та приватними компаніями[28].


Рисунок 1.12 – Угода Uber

Uklon:

Uklon активно працює на місцевому ринку та підтримує заходи для користувачів[29].

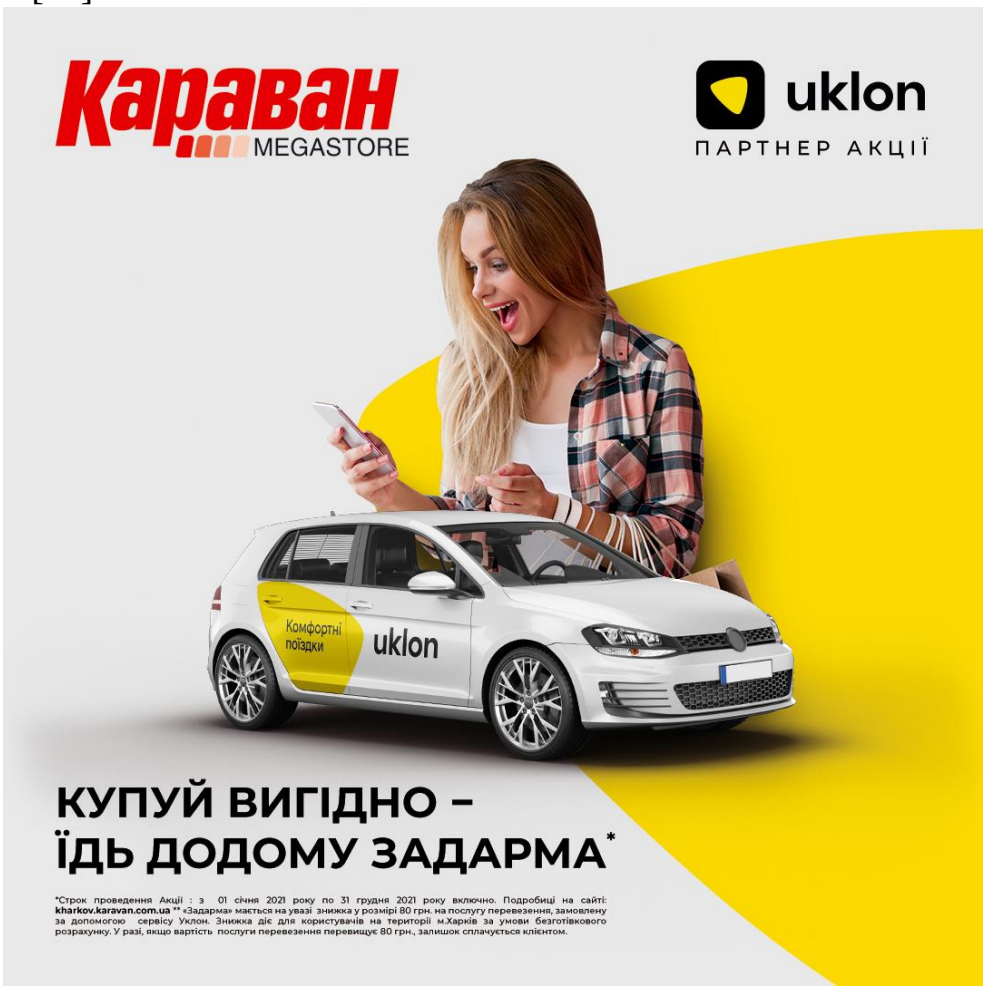


Рисунок 1.13 – Угода Uklon

Болт:

Bolt також використовує маркетингові кампанії для залучення нових користувачів[30].

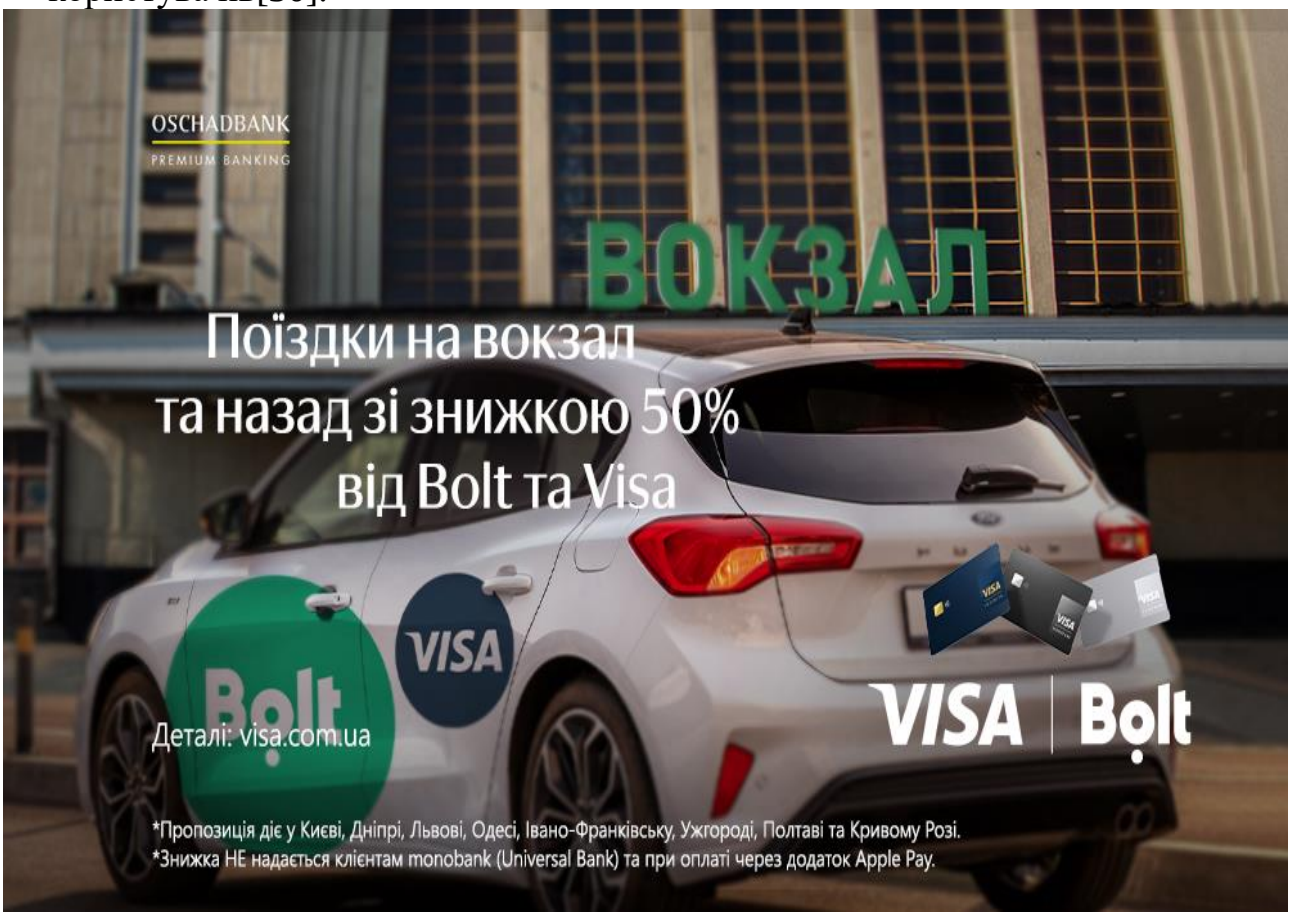


Рисунок 1.14 – Угода Bolt

Евристика: Аналіз маркетингових стратегій конкурентів на основі ефективних методів залучення та утримання користувачів.

На основі евристичного аналізу трьох провідних служб таксі - Uber, Uklon і Bolt можна зробити наступні перечислені висновки:

1. Важливість зручного інтерфейсу: усі три служби надають користувачам зручний інтерфейс для замовлення послуг. Важливу роль у популярності додатків таксі відіграє простота та швидкість обслуговування.

2. Конкурентоспроможні ціни та акції: акції та знижки, а також конкурентоспроможні ціни можуть бути рішеннями для залучення нових користувачів і утримання існуючих.

3. Оцінки та відгуки користувачів: Позитивні оцінки та відгуки користувачів свідчать про високу якість обслуговування та довіру до платформи.

4. Маркетингові стратегії: Ефективні маркетингові стратегії та рекламні кампанії допомагають залучити нових користувачів.

Загальний висновок: для успіху, окрім виклику таксі, потрібне підключення до якості послуг, зручного інтерфейсу, конкурентних цін та ефективних маркетингових стратегій. Розробники нової системи виклику

таксі повинні пам’ятати про ці ключові чинники та упускати їх із уваги у час розробки та виведення свого продукту на ринок.

## 1.2. Характеристика задачі.

Характеристика задач полягає у описанні призначення задач, підстав для їх рішення, предмету задач, а також вироблення перших вимог до розробляємої програми, що стосуються її інтерфейсу, вхідної та вихідної інформації, звітування та інше.

Для того, щоб правильно сформулювати постановку задачі, необхідно, насамперед, вивчити сутність предметної області, на обробку елементів якої спрямована головна мета даної дипломної роботи.

Завданням даної дипломної роботи є створення програмного забезпечення для базового функціонування служби таксі які виключає із свого функціоналу диспетчерську службу, а саме[9]:

- створення серверного додатку;
  1. проектування візуального інтерфейсу;
  2. створення БД яку буде використовувати сервер для роботи;
  3. Створення таблиць в БД;
  4. реалізація функцій для роботи сервера;
- створення архітектури бази даних;
  1. вибір серверу БД;
  2. проектування БД;
  3. налаштування підключення до серверу;
- створення точки входу та обробників інформації андроїд додатку для таксистів;
  1. Описання моделі поведінки такого мобільного додатку;
  2. Створення під-класу, класу сервера для можливості підключення таких додатків.
- наповнення БД для тестування даного ПЗ;
- тестування створеної програмної системи.

## 1.3. Вхідна інформація.

До вхідної інформації відносять дані, що необхідні для розв'язання аналітичних задач. Вхідна первинна інформація є найбільш детальною і становить основу для наступної логічної та арифметичної обробки даних.

В моєму проекті вхідна інформація представлена в двох видах і акумулюється в базі даних як необхідна для роботі та архівна.

Перший тип даних це реєстраційна дані таксистів, які необхідні для їх авторизації, та подальшої роботі з програмою, адже від неї залежить чи зможе таксист взагалі авторизуватись в програмі. Плюс в цих даних закладена інформація яка допоможе розпізнати автомобіль замовнику (номер авто, тип та колір). Так само інформація про таксиста буде зв’язана з інформацією про замовлення для перегляду статистичних даних.

Другий тип даних, це безпосередньо саме замовлення, при створенні

його замовником, в БД створюється запис для роботи з ним. Вносяться дані про маршрут, а потім це замовлення виноситься сервером в ефір таксистів які його можуть взяти на виконання. Після виконання замовлення, таксист закриває його і після цього воно знаходиться в БД як архівне.

## 1.4. Вихідна інформація.

Вихідною інформацією програми є результат її роботи. В моєму випадку це архівні та статистичні дані по роботі одного чи групи таксистів, загальні статистичні дані, та передача інформації від сервера до клієнта при підтвердженні таксистом замовлення.

При перегляді інформації про таксиста виводяться всі його виконані замовлення за весь період роботі, чи за деякий визначений період. Ця інформація є найбільш необхідною тому що за допомогою неї можна блокувати таксистів які по декілька місяців не працюють з програмою, чи розвідників які збирають інформацію для конкурентів.

Данні по замовленням представляють зальну картину роботи сервісу, по ній можна знайти все що необхідно. Загальну кількість замовлень які були виконані таксистами, загальну кількість замовлень які були прийняті але таксисти від них відмовились. Вести фінансовий аналіз ринку, аналізувати загальну кількість замовлень по місту де вони були створені, таким образом визначати вигідно працювати в місті чи ні.

Ще одною вихідною інформацією є дані таксиста які після прийняття замовлення передаються на сторону замовника. Ця інформація вже є в БД і при передачі вже є вихідною. Ці дані використовуються для розпізнання клієнтом авто таксиста, адже людина чисто фізично не може знати який автомобіль приїхав за ним

## Висновки до розділу 1

У цьому розділі була поставлена задача, сформована характеристика задачі, описано вхідні та вихідні дані програми.

Мною було описано аналог існуючої системи служби таксі сформованої на диспетчерській системі, описано суть завдання по створенню системи яка б виключила із свого циклу диспетчерську службу. Також було сформована характеристика та завдання дипломного проекту для розробки ПЗ. Описані її вхідні та вихідні дані.

Сьогодні використання ПЗ для замовлення різного роду послуг не рідкіть, вони використовуються як на міцевому рівні, так і на рівні держав, і розробка програми подібного роду є доволі актуальною задачею.

# РОЗДІЛ 2
# РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

## 2.1. Розробка методів (моделей).

Комп'ютерна програма (англ. Computer program) — набір послідовних інструкцій у вигляді слів, цифр, кодів, схем, символів чи в будь-якому іншому вигляді, виражених у формі, придатній для зчитування та виконання обчислювальною машиною (комп'ютером), які приводять його у дію для досягнення певної мети або результату (це поняття охоплює як операційну систему, так і прикладну програму, виражені у сирцевому або об'єктному кодах)[25].

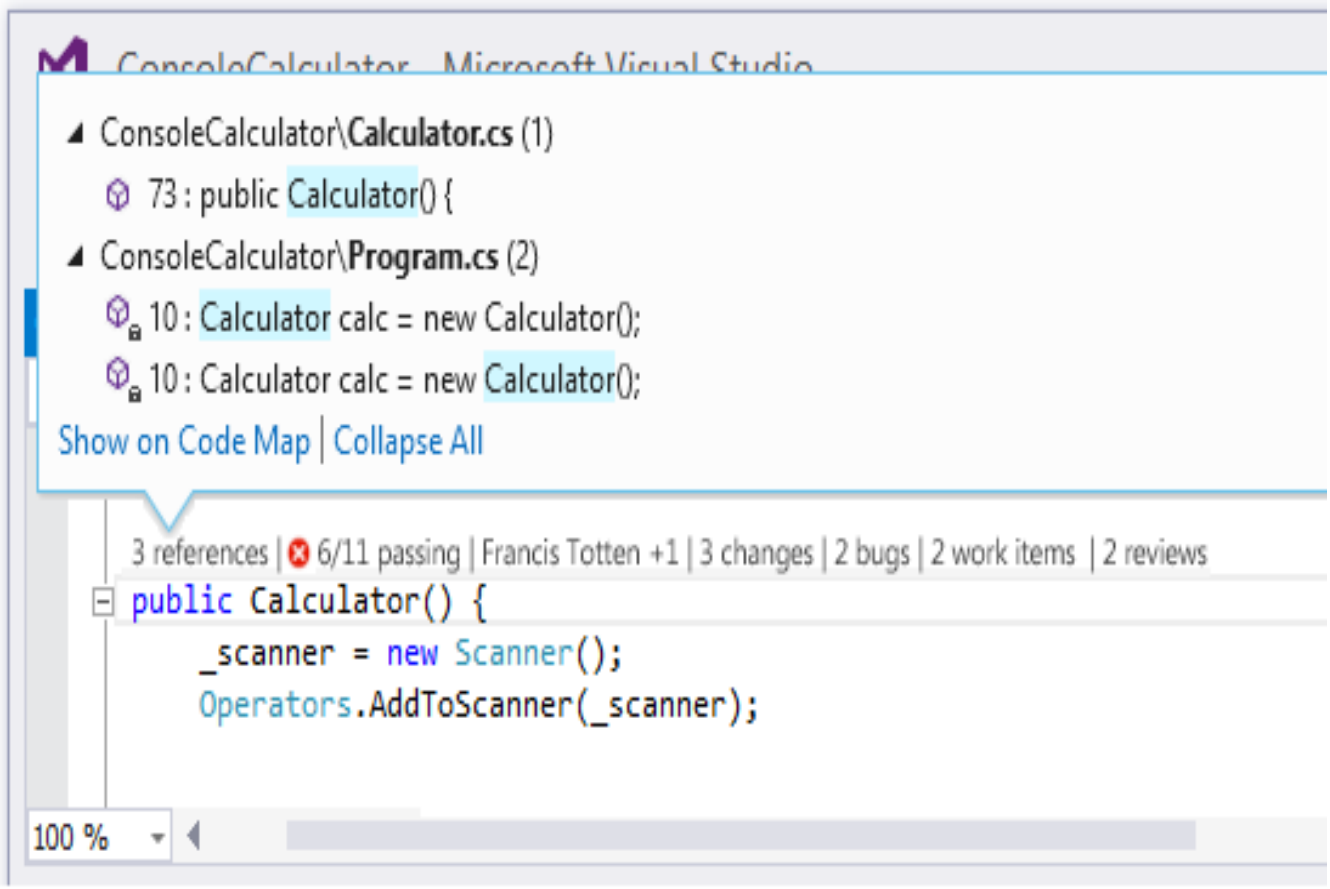


Рисунок 2.1 – Комп'ютерна програма

По-іншому комп'ютерна програма визначається як набір комп'ютерних команд, що складають запис алгоритму на одній з мов програмування.

Програма може бути написана в текстовому форматі мовою програмування, відображатися графічно за допомогою блок-схем і передаватися на комп'ютер у вигляді електричних сигналів.

Якщо комп'ютерна програма не представлена у вигляді послідовності машинного коду в системі команд процесора обчислювальної системи, її необхідно спочатку перетворити в такий код за допомогою компілятора або запустити програму за допомогою програмного інтерпретатора.

Обробка інформації-це цілий комплекс операцій (збір, введення, запис, перетворення, зчитування, зберігання, знищення, Реєстрація), що виконуються з використанням технічних і програмних засобів, включаючи обмін по каналах передачі даних.

Згідно ДСТУ 2938-94[5]:

Обробка інформації-це систематичне виконання операцій з інформацією.

Система обробки інформації-це система, що складається з набору технічних і програмних засобів, а також робочого персоналу, що забезпечує обробку інформації.

Під час розробки програми її модульна структура може бути сформована різними способами і використовуватися для визначення порядку програмування та налагодження модулів, зазначених у цій структурі.

Отже, ми можемо говорити про різні способи розробки структури програми. Зазвичай у літературі обговорюються два методи: метод розробки знизу вгору та метод розробки зверху вниз.

Метод розробки знизу вгору полягає в наступному[9]. По-перше, модульна структура програми побудована у вигляді дерева. Потім програмні модулі програмуються в порядку, починаючи з модуля найнижчого рівня (лист дерева модульної структури програми), так що для кожного попереднього програмованого модуля всі модулі, до яких можна отримати доступ, вже запрограмовані. Після того, як всі модулі програми запрограмовані, вони, як правило, тестуються по черзі в тому ж порядку (за зростанням), в якому відбувалося програмування, і кожен модуль представлений в програмуванні через безпосередньо запрограмовані підлеглі модулі, а в тесті використовуються вже налагоджені модулі. Це буде. Однак сучасні технології не рекомендують таку процедуру розробки програми.

По-перше, програмування модуля не вимагає наявності тексту модуля, який він використовує, і для його тестування можна замінити використовувані модулі їх імітаторами (плагінами).

По-друге, кожна програма в якійсь мірі підпорядкована внутрішньому, але глобальному розумінню своїх модулів (принципів реалізації, припущень, структур даних і т.д.). Це визначає її концептуальну цілісність і формується в процесі її розробки. При розробці вище ця глобальна інформація для модулів нижчого рівня ще не повністю зрозуміла, тому, якщо під час програмування інших модулів вносяться значні вдосконалення в цю глобальну інформацію (наприклад, зміни в глобальній структурі даних), часто виникає необхідність їх перепрограмування.

По-третє, під час тестування знизу вгору, для кожного модуля (крім основного) готується необхідний стан інформаційного середовища тестованого модуля, і необхідний доступ це призводить до великого обсягу "налагоджувального" програмування, і в той же час не гарантує, що модуль був точно протестований в умовах виконання виробничої програми.

Метод розробки зверху вниз полягає в наступному[9]. Як і в попередньому методі, модульна структура програми спочатку будується у вигляді дерева. Потім програмний модуль запускається з модуля найвищого рівня (основного), і лише якщо модуль, який отримує доступ до нього, вже запрограмований, після того, як запрограмовані всі інші програмні модулі, вони по черзі тестуються та налагоджуються в тому ж порядку (зверху вниз). Це гарна ідея. У цьому випадку спочатку тестується основний модуль програми, що представляє всю тестову програму, і, отже, знаходиться в "природному" стані інформаційного середовища, в якій ця програма починає виконуватися. У той же час модуль, до якого може отримати доступ основний модуль, замінюється імітатором (так званим плагіном). Кожен модуль симулятора представлений дуже простим фрагментом програми, який в основному інформує вас про те, що Ви звертаєтеся до модельованого модулю, виконує обробку значень вхідних параметрів, необхідних для коректної роботи програми (іноді з їх роздруківкою) і, при необхідності, виводить попередньо збережений доступний результат. Після завершення тестування і налагодження основного і наступних модулів, якщо в симуляторі в даний момент представлений один з модулів, перейдіть до тестування.1


Рисунок 2.2 – Спадаюча розробка

Для цього імітатор обраного для тестування модуля замінюється самим цим модулем, крім того, модуль, обраний для тестування одночасно, кожен такий модуль тестується в умовах "природного" стану інформаційного середовища, що виникає перед зверненням до цього модуля під час виконання тестової програми. Те

Таким чином, великий обсяг "налагоджувального" програмування в висхідних тестах замінюється програмуванням досить простого симулятора модулів, що використовуються в програмі. Крім того, симулятор можна використовувати для підігравання процесу вибору тесту, задаючи бажаний результат, що видається симулятором.

При такому порядку розробки програми вся необхідна Глобальна інформація генерується вчасно.Тобто це дуже неприємна причина прорахунку, коли програмний модуль усувається. Деякі недоліки розробки зверху вниз, що призводять до певних труднощів у її застосуванні, - це необхідність абстрагуватися від основних функцій використовуваної мови програмування та придумувати абстрактні операції, які необхідно реалізувати за допомогою модулів, призначених програмі пізніше. Однак здатність до такої Абстракції, здається, є необхідною умовою для розробки великих програмних засобів, тому її потрібно розвивати.

Етапи створення мат моделі[24].

При математичному моделюванні вивчається не сам фактичний фізичний процес, а модель, необхідна для збереження основних особливостей розглянутого процесу, яка настільки проста, що одночасно вивчається математичними методами.

Створення математичних моделей фізичних явищ можна розділити на етапи. Вибираються базові значення (кілька базових величин), які характеризують процес. При математичному моделюванні поширення тепла такими значеннями є температури точок середовища, зазвичай є функцією просторових координат і часу.

На другому кроці ми виводимо визначальну формулу для фундаментальних величин, що характеризують процес. При вивченні поширення тепла таким рівнянням є рівняння теплопровідності. Для виведення цього рівняння використовується закон збереження тепла в будь-якому обсязі нерівномірно нагрітої середовища.

Диференціальні рівняння, отримані на другому етапі, мають багато рішень. Отже, недостатньо описати конкретний процес. Таким чином, на 3-му етапі побудови математичної моделі з набору рішень певних рівнянь виводяться так звані чіткі умови, які дозволяють ідентифікувати єдине рішення, що характеризує даний модельований процес. Нагадаємо, що у випадку рівнянь математичної фізики такими додатковими умовами є граничні умови та початкові умови.

Таким чином, математична модель процесу, що вивчається з використанням рівнянь математичної фізики, складається з основної кількості диференціальних рівнянь, що характеризують процес, і додаткових умов, що дозволяють отримати єдине рішення цього рівняння – рішення, що описує даний конкретний фізичний процес.

Доступ до неї корисний не тільки для внутрішніх користувачів - їх власних співробітників, але і для великої кількості зовнішніх користувачів, які так чи інакше пов'язані з цією організацією. Наприклад, біржі зазвичай також пов'язані з банківськими системами, компаніями та страховими компаніями. Банки зацікавлені в тому, як складаються курси купівлі та продажу, а також у швидкій реєстрації покупок і продажів через обмінний банк. Так само компаніям завжди важливо знати, як надходять кошти на їхні чекові рахунки, відвантажуються товари та здійснюються платежі. Очевидно, що ці проблеми можна вирішити шляхом створення та використання локальних та глобальних мереж передачі даних та надання користувачам належного доступу до бази даних.

Зазвичай такі проблеми вирішуються за допомогою

телекомунікаційних технологій і клієнт / сервер. Давайте зупинимося на цьому питанні більш детально про доступ користувачів до віддалених баз даних. Зокрема, технологія SQL забезпечує віддалений доступ до бази даних.

В основі бази даних лежить програма, що складається з сервера баз даних, джерела даних та мережевого програмного забезпечення для підключення клієнтів до мережі.Як ми визначили раніше сьогодні, сервери баз даних, такі як Oracle, Informix, Sybase та Interbase, є загальними та ефективними. Сервер баз даних створюється на робочому місці адміністратора бази даних, і клієнт отримує відповідний доступ до таблиць відповідно до посадових обов'язків та статусу.

Інтерфейсна частина-це програмне забезпечення, яке використовується на робочому місці користувача, тобто конкретний компонент системи автоматизації, призначений для вирішення проблеми цього користувача. Цей штучний інтелект або програмний пакет може бути розроблений розробниками різних мов програмування, таких як C++, Pascal, Delphi, Visual C++ і т. д.

Розглянемо процес взаємодії між додатком і інтерфейсною частиною. Прикладні частини розміщуються на сервері разом з даними бази даних. Користувачами прикладної частини бази даних є адміністратори баз даних, програмісти, розробники автоматизованих систем, аналітики та системні адміністратори. Інтерфейсна частина розміщується на комп'ютері кінцевого користувача, тобто операторів введення даних, бухгалтерів, операціоністів, фінансистів, економістів і т. д.

База даних може бути локальною, коли користувач підключається безпосередньо, і може бути підключена віддалено, якщо вона підключена з великої відстані. Підключення до віддаленої бази даних виконується з використанням мережевої підтримки і відповідного протоколу передачі даних.Математична модель диспетчерської служби таксі може бути складною і включати різні складові для оптимізації процесів вибору водіїв, маршрутизації поїздок та вирішення завдань пріоритету в реальному часі. Давайте розглянемо загальну математичну модель такої диспетчерської системи з коротким описом і поясненням кожної частини[4].

**1. Модель клієнтів і їх замовлень:**

Клієнти (C): В даній моделі кожен клієнт є окремим об'єктом. Кожен клієнт може вимагати таксі в певний час та місце.

**2. Модель водіїв і їх розташування:**

Водії (D): Кожен водій також є окремим об'єктом. Вони можуть перебувати в певних точках міста, а їх доступність для поїздок може змінюватися з часом.

**3. Модель замовлень та завдань:**

Замовлення (O): Кожне замовлення має власні характеристики, такі як час створення та місце призначення.

**4. Функція вибору водія:**

Функція вибору (Dispatch Function): Ця функція визначає, якому водію буде надано певне замовлення. Вона може враховувати різні фактори, такі як відстань між водієм і місцем призначення, час очікування для клієнта та інші обмеження.

**5. Функція маршрутизації:**

Функція маршрутизації (Routing Function): Ця функція визначає найкращий маршрут для виконання замовлення. Вона може враховувати рух транспорту, дорожні умови та інші чинники для забезпечення найшвидшого та найефективнішого маршруту.

**6. Параметри і обмеження:**

Параметри (P) і обмеження (Constraints): Модель може включати різні параметри та обмеження, такі як максимальна відстань між клієнтом і водієм для прийняття замовлення, максимальний час очікування клієнта тощо.

**7. Функціональні цілі:**

Цільова функція (Objective Function): Модель може мати цільову функцію, яка максимізує покриття замовлень, мінімізує час очікування клієнтів або мінімізує вартість поїздок.

Математична модель диспетчерської служби таксі може бути реалізована у вигляді оптимізаційної задачі, де основною метою є найкраще призначення водіїв для замовлень та оптимізація маршрутів для максимізації задоволення клієнтів та водіїв, а також оптимізація вартості і часу поїздок.

Ця математична модель допомагає диспетчерській службі приймати рішення в режимі реального часу, щоб оптимізувати використання ресурсів та забезпечити ефективний сервіс для клієнтів і водіїв. Модель може бути покращена та розширена, враховуючи конкретні особливості ринку і вимоги користувачів.

Після пояснень і опису всіх необхідних пунктів розглянемо формули, необхідні для нашої математичної моделі.

**1. Параметри моделі:**

C - Множина клієнтів. Кожен клієнт характеризується інформацією про місце, звідки він хоче замовити таксі, і час, коли він це хоче зробити.

D - Множина водіїв. Кожен водій також має свої характеристики, наприклад, розташування в певний момент часу.

O - Множина замовлень. Кожне замовлення має свою власну інформацію про місце призначення та час створення.

x_{ij} - Бінарна змінна, яка дорівнює 1, якщо водій і обслуговує замовлення j, та 0 в іншому випадку.

t_{ij} - Час, який відводиться для водія і на обслуговування замовлення j.

d_{ij} - Відстань між водієм і та місцем призначення замовлення j.

**2. Цільова функція:**

Основною метою є мінімізація загального часу або витрат для задоволення всіх замовлень. Для цього ми можемо сформулювати цільову функцію так:

Мінімізувати: $\min Z = i \in D + j \in O^{xij*tij}$

Де:

Z - загальний час обслуговування всіх замовлень

i - водій

j - замовлення

**3. Обмеження:**

Кожне замовлення має бути обслуговано одним водієм:

$i \in D^{xij} = 1$ для всіх $j \in O$.

Кожен водій може обслуговувати лише одне замовлення:

$j \in O^{xij} \leq 1$ для всіх $i \in O$ .

Водії повинні бути доступними для замовлень:

$j \in O^{xij} \leq 1$ для всіх $I \in D$.

Обмеження на час і відстань можуть бути додані, щоб врахувати певні умови та обмеження.

**4. Оптимізація:**

Ця математична модель може бути вирішена за допомогою методів оптимізації, таких як цілочисельне програмування (integer programming), лінійне програмування (linear programming), а також евристичні методи оптимізації, які враховують складність задачі та обмеження в реальному часі.

Для розв'язання цієї математичної моделі використовуються методи оптимізації, такі як цілочисельне програмування або лінійне програмування. Мета полягає в знаходженні оптимальних значень змінних x_{ij} та мінімізації цільової функції Z.

**5. Додаткові фактори:**

В реальних системах додаткові фактори, такі як трафік, популярність водіїв, попередні призначення та попередження про нові замовлення, також можуть враховуватися в моделі для поліпшення точності та продуктивності.

Ця математична модель дозволяє диспетчерській службі таксі оптимізувати розподіл водіїв на замовлення та маршрутизацію, забезпечуючи ефективність та якість обслуговування клієнтів. Важливо враховувати, що реальні системи можуть включати додаткові функції та обмеження для врахування різних умов та вимог.

**2.2. Розробка алгоритму вирішення задачі.**

Основою розробки програмного забезпечення є алгоритми. Алгоритм-це ясний і точний наказ Виконавцю виконати набір дій, спрямованих на досягнення заданої мети або рішення заданої задачі[10].

Основні характеристики алгоритму.

Виразність. Для того щоб виконавець досяг поставлених перед ним цілей за допомогою даного алгоритму, він повинен вміти слідувати всім його інструкціям.

Визначеність (ясність). Проте зрозумілий алгоритм не повинен містити інструкцій, і його зміст може сприйматися розпливчасто. Крім того, така ситуація неприйнятна в алгоритмі, якщо після виконання чергового замовлення алгоритму Виконавець не розуміє, що потрібно зробити на наступному кроці.

Дискретність. Як згадувалося вище, алгоритм встановлює повну послідовність дій, які необхідно виконати для вирішення завдання. У той же час, для виконання цих дій вони розділені на прості кроки в певному порядку. Тільки виконавши дію попереднього порядку, ви зможете виконати дію наступного порядку. Таке розбиття алгоритму на окремі базові дії (команди), які легко виконуються даним виконавцем, називається дискретизацією.

Масовий характер. Дуже важливо, що скомпільований алгоритм забезпечує рішення, а не 1 задачу, але він може вирішувати широкий спектр задач такого типу класу.

Ефективність. Взагалі кажучи, ясно, що виконання алгоритму має завершуватися отриманням кінцевого результату. Тобто ситуації, які в деяких випадках можуть призвести до так званих "циклів", повинні бути виключені при написанні алгоритму.

Ефективність-кожен крок алгоритму повинен бути виконаний точно протягом кінцевого періоду часу.

Розробка програмного забезпечення здійснюється методами проектування "зверху вниз" - спочатку завдання визначаються в загальних рисах, потім поступово поліпшуються цілі, алгоритми, структури даних, обмеження і т.д. беручи до уваги все більш дрібні деталі завдання.

Метод покрокового поліпшення є одним з основних методів розробки програм.1 назвемо це низхідним програмуванням.

Оскільки будь-яка програма являє собою алгоритмічне рішення поставленого завдання, необхідно дотримуватися декількох принципів: по-перше, це сприяє створенню зрозумілих програм, по-друге, полегшує налагодження непрацюючих програм і, по-третє, зменшує загальну кількість логічних помилок. 1. Багаторівневе структурування є однією з найбільш важливих заходів в цьому напрямку. Розрізняють наступні рівні структурування[10]:

- синтаксичний;
- процедурний;
- модульний;
- об’єктно-компонентний.

Синтаксичне структурування полягає у дотриманні деяких простих принципів, що визначають сучасну культуру написання програм:

- основним програмним блоком є тіло самої програми;
- програмний блок записується з використанням відповідного відступу від початку рядка;
- блоки можуть бути вкладені;
- команди на одному і тому ж рівні вкладеності записуються з однаковим відступом від початку рядка;
- команди на одному рівні вкладеності і близькі за призначенням іноді зручно записувати в одному рядку;

Рекомендується, щоб кожен програмний блок був якомога меншим. Якщо розмір програмного блоку перевищує 20 рядків, або він повторюється кілька разів в різних місцях програми, краще розміщувати такі блоки у вигляді підпрограм (процедур або функцій). Може бути дуже корисно розбити всю програму на окремі блоки, оформлені у вигляді підпрограм. Наприклад, введення даних, обчислення допоміжних значень, побудова графіка, відображення результатів на екрані і т. д.

Процедурне структурування значно підвищує зрозумілість програми і дозволяє виконувати завдання в окремих, легкодоступних частинах.

Модульне структурування використовується при створенні великих програм. Коли в програмі використовується велика кількість різних процедур і функцій, всі процедури діляться на окремі групи, проектуються, компілюються і зберігаються у вигляді окремих файлів модулів. Часто вся бібліотека програмного забезпечення складається з таких модулів. Щоб використовувати процедури та функції з певного модуля у вашій основній

програмі, ви повинні позначити список усіх модулів, які використовують підпрограми, у наступному рядку після заголовка програми після ключового слова uses[10].

Якщо компілятор виявить незнайомий ідентифікатор у програмі, який не відображається в розділі опису поточної програми, послідовний пошук цього імені видасть помилку, що вказує на те, що цей ідентифікатор невідомий, якщо він не знайдений там у рядку uses. Оскільки модуль зберігається в скомпільованому форматі, при компіляції основної програми необхідно створити в модулі розділ, що містить необхідні програми.

Рівень об'єктного компонента включає в себе використання спеціальних спеціалізованих ідеологій і методологій для написання програм. Це забезпечує досить високий рівень підготовки фахівців, що використовують цю техніку. У той же час вся програма складається з безлічі різних об'єктів. Сама програма також є об'єктом. Кожен об'єкт складається з набору властивостей і методів. Властивості визначають кількісні та якісні характеристики об'єктів, а методи визначають дії, які ці об'єкти можуть виконувати. Всі об'єкти працюють в єдиному програмному середовищі, яка завжди очікує будь-якої події. Такими подіями можуть бути натискання клавіш, клацання миші в певних областях екрана або покращення положення курсору миші в певних областях екрана. Потім спеціальний програмний менеджер перенаправляє цю подію на потрібний об'єкт для обробки і викликає відповідний метод, для цих цілей, використовуючи об'єктно-орієнтовану методологію, можна написати величезний програмний пакет. Майже все новітнє програмне забезпечення (ОС Windows, Word, Excel, Access і т.д.) написано з використанням цієї методики.

Головним алгоритмом програми дипломного проекту є обмін даними між сервером і його клієнтами, але є досить багато більш менших алгоритмів які є не менш важливими чим головний.

Задача головного алгоритму програми є наглядно продемонструвати роботу серверу та послідовність потоку даних які передаються на тому чи іншому блоці виконання алгоритму програми. Головним завданням програми є створення нового замовлення та передача його в ефір таксистам, після цього необхідно дождатися коли таксист прийме замовлення на виконання, в свою чергу серверу необхідно передати клієнту дані про таксиста який взяв замовлення останнього на виконання, а після виконання самого замовлення необхідно щоб таксист завершив замовлення через програму, а сервер передав інформацію на додаток замовлення щоб логіка додатка могла очистити дані про поточне замовлення.


Рисунок 2.1 - Головний алгоритм роботи серверу

Блок-схема зображена вище не є одною із необхідних, присутня схема авторизації, реєстрації, створення нового замовлення, відказ від замовлення. Вони всі є вторинними, але з цього переліку можна винести реєстрацію як особливу функцію. Реєстрація є точкою входу в роботу з програмою (Рисунок 2.2). На цьому етапі вносяться дані про користувача, його автомобіль та номера документів.

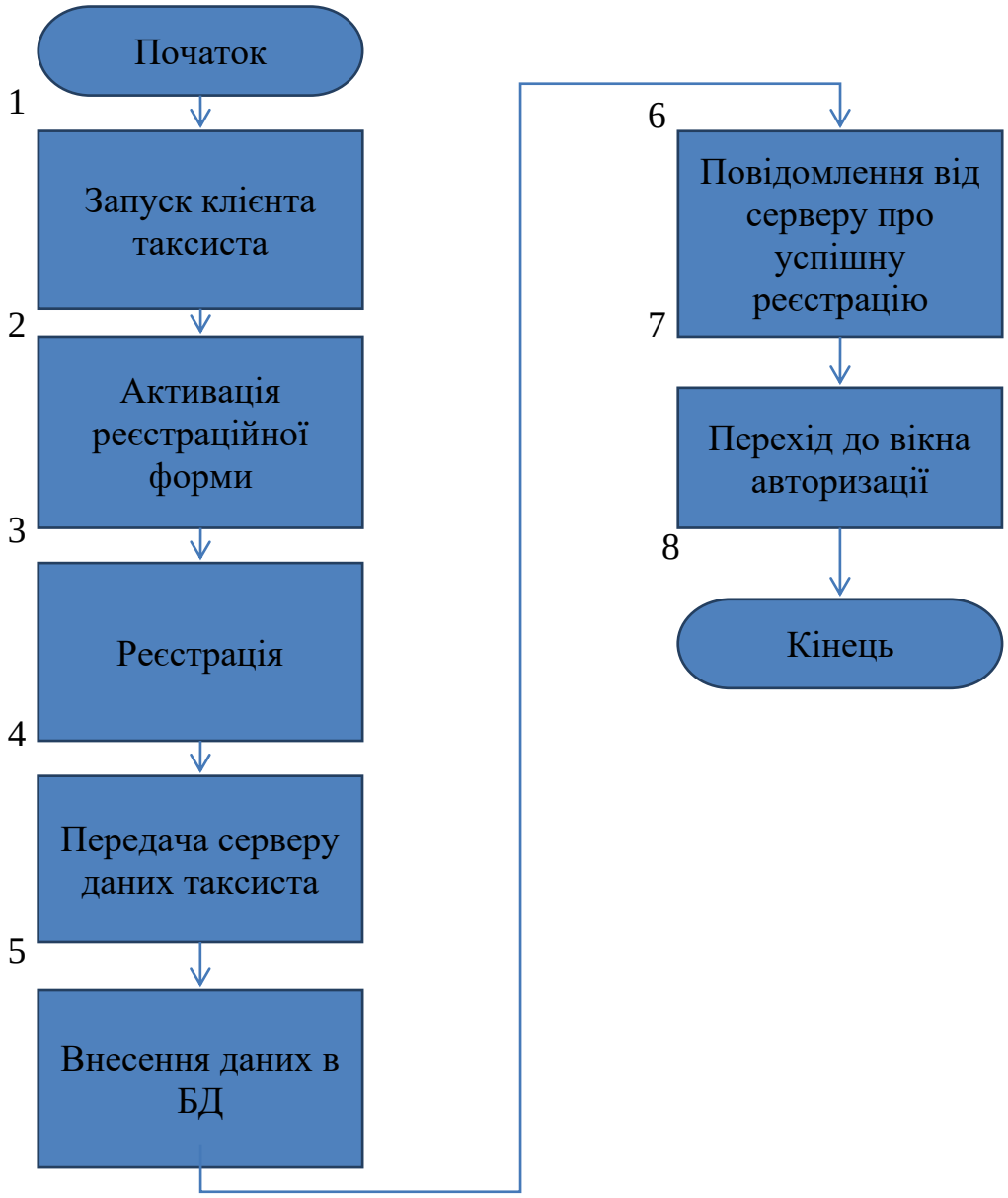


Рисунок 2.2 - Алгоритм авторизації на клієнті таксиста

Алгоритм приведений вище показує роботу клієнта таксиста за сервером, тут можна знайти і мережеву роботу, і роботу з базою даних. При роботі з мережею по протоколу TCP передається операційний код реєстрації по якому сервер робить висновок що дані які прийшли від клієнта е реєстраційними і їх потрібно занести в БД та передати на клієнт повідомлення про успішну авторизацію.

## Висновки до розділу 2

Задачею цього розділу було описання методів створення алгоритмів та розробка алгоритму для вирішення задачі передбаченою темою кваліфікаційної роботи.

Розробка алгоритму є самою базовою і абсолютно необхідною роботою перед розробкою ПЗ.

Написання зрозумілих і складних алгоритмів є цілим мистецтвом, для цього необхідно мати знання в доволі обширному колі наук: математиці, аналізі, логіці, та інших науках, які зачіпає задача, що потребує алгоритмічного вирішення.

# РОЗДІЛ 3

# ОРГАНІЗАЦІЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1. Загальна характеристика інформаційного забезпечення.

Загальна характеристика інформаційного забезпечення

Інформаційне забезпечення – це сукупність документів, нормативно-правової бази і реалізованих рішень для автоматичної обробки економічної інформації або обсягу, розташування і форми композиції інформації, що циркулює в інформаційних системах[14].

Основними принципами створення інформаційної підтримки є цілісність, надійність, контроль, захист від несанкціонованого доступу, однаковість і гнучкість, стандартизація та уніфікація, адаптивність і мінімізація помилок введення/виведення інформації.

Розвиток інформаційного забезпечення є одним з найважливіших елементів розвитку інформаційних систем, і необхідно забезпечити, щоб::

1. Уніфікація і зберігання інформації, необхідної для вирішення завдань; уніфікація інформаційних масивів для всіх завдань інформаційних систем;
2. Єдиний введення інформації та її багатоцільове використання;
3. Різні способи доступу до даних;
4. Низька вартість зберігання, використання та модифікації даних.

Інформаційна підтримка складається з наступних частин:

- Методологія та навчальні матеріали;
- Система класифікації та кодування шин;
- Інформаційна база.

Один з ключових компонентів інформаційної підтримки, інформаційна база (ІБ) - це набір впорядкованої інформації, використовуваної в процесі функціонування ІБ, який складається з внутрішньої машини і зовнішньої машини (external machine)[15].

Внутрішня машинна інформаційна база-це сукупність даних на машинних носіях, яка є частиною системи інформаційної безпеки і використовується ІС.

Внутрішня машинна інформаційна безпека цієї інформаційної системи організована у вигляді спеціального масиву даних, умовно розділеного на оперативну і нормативно-довідкову інформацію, Вся інформація тут знаходиться на рідкокристалічному дисплеї, але архів може бути створений в будь-який момент[14].

Зовнішня машинна (позамашинна) інформаційна база-це сукупність повідомлень, сигналів і документів, призначених для безпосереднього сприйняття людиною без використання комп'ютерних технологій захисту інформації. Вона складається з вхідних, вихідних і нормативних довідкових повідомлень[14].

При розробці вбудованої інформаційної бази даних особлива увага була приділена широкому використанню зберігаються в ній даних і простоті, з якою використання даних може бути легко змінено. У цьому проекті була запропонована файлова організація внутрішньої машинної інформаційної системи для забезпечення гнучкого використання даних та їх ефективного використання. Особливістю пропонованої інформаційної бази є наявність нормативних довідкових даних.

Масив нормативної та довідкової інформації може бути використаний для реалізації Введення даних для контролю наявності інформації, внесеної до файлів інституту.

Організація інформаційної підтримки даної ІС пов'язана з розробкою систем класифікації та кодування, організацією збору і передачі інформації, розробкою зовнішніх і внутрішніх машинних інформаційних баз і формуванням вихідних даних.

У разі розроблення ІЗ використовуються такі нормативні документи: ДСТУ 20912-75, ДСТУ 6.01.1—87, ДСТУ 19.2—75, РД 30—34.698—90, постанови Державного комітету статистики України.

Також інформаційне забезпечення відповідає за зберігання статистичних даних по роботі програми, які в свою чергу необхідні для планування роботи всього сервісу.

## 3.2. Побудова системи класифікації та кодування.

Використання систем класифікації та кодування є необхідним елементом автоматизованої обробки даних.

Класифікація-це один із засобів вивчення властивостей досліджуваних об'єктів шляхом їх упорядкування та систематизації.1

Сукупність об'єктів дослідження ділиться на підмножини відповідно до значень певних характеристик і ознак, а результати, отримані при їх використанні, називаються системою класифікації, і ця система називається класифікаційною системою[6].

Класифікація є одним з найважливіших етапів проектування інформаційного забезпечення автоматизованих систем, а також забезпечує основу для аналізу та моделювання інформаційних потоків.1

Щоб класифікувати об'єкти будь-якої природи, необхідно визначити набір функцій класифікації, які визначають основу розподілу об'єктів.

Класифікація інформації повинна відповідати наступним основним вимогам[6]:

1) забезпечення повноти охоплення об’єктів множини, що вивчається;
2) не перетинання груп об’єктів, що виділяються;
3) можливість включення нових груп об’єктів;
4) лаконічність, чіткість і зрозумілість класифікаційних ознак;
5) незмінність прийнятої класифікаційної ознаки на всіх рівнях класифікації.

Основні системи класифікації інформації включають ієрархію, грані та змішування. Ієрархічна система класифікації-це система, яка ділить набір об'єктів на підмножини, які виконуються послідовно відповідно до заданих характеристик.

Основний набір об'єктів спочатку ділиться на підмножини, сформовані на одній і тій же основі, що дозволяє отримати різні значення. Потім кожне результуюче підмножина ділиться на групи відповідно до значень наступних атрибутів: потім ці групи діляться на підгрупи відповідно до наступних критеріїв і т. д.

Між обраними групами об'єктів встановлюється певна ієрархія. У цьому випадку кожна підмножина належить лише 1 верхньому набору. Ієрархічна система класифікації характеризується глибиною, тобто кількістю підрозділів у первинному наборі, або, що еквівалентно, кількістю заданих класифікаційних ознак.

Ієрархічна система класифікації характеризується простотою, чіткістю, логічною композицією та хорошою адаптивністю до ручної обробки. Недоліками таких систем є жорсткість структури через фіксованих функцій і порядку їх розміщення, складність включення нових функцій, необхідність у великому резервному обсязі.

Перевагою системи класифікації фасетів є гнучкість її структури, можливість виключення старих фасетів, включаючи нові.

До недоліків такої системи можна віднести відсутність традицій в разі ручної обробки даних і складність її використання, а також недостатнє використання виробничих потужностей через те, що багато можливих комбінації фасетів не мають практичного застосування. Зміни в системі класифікації включають використання обох вищевказаних систем.

В ієрархічній системі класифікації елементи не повинні перекриватися на жодному рівні підмножини. Сума елементів усіх наборів на кожному рівні дорівнює кількості елементів у первинному (початковому) наборі. Для фасетної класифікації це не потрібно, і сума елементів у підмножині може бути більшою, ніж кількість елементів у вихідному наборі.

У фасетній системі важливо, щоб ознаки не відтворювалися.

Обраний метод класифікації повинен відповідати наступним вимогам[6]:

1) бути достатньо містким і повним;
2) характеризуватися достатньою економічно обґрунтованою глибиною;
3) кількість ознак має бути виправданою;
4) забезпечувати розв’язання всіх комплексів задач;
5) характеризуватися лаконічністю, гнучкістю та якістю класифікаційних ознак.

При проектуванні інформаційної системи можуть використовуватися різні системи кодування, включаючи порядкове, послідовно-порядкове, послідовне і повторюване кодування.

Інструмент порядкового кодування призначений для формування знака з натуральних чисел і їх призначень. Це найбільш повний і простий інструмент. Використовується для чітких позицій.

Безперервне усереднення по порядку відноситься до формування кодів з числа натуральних рядів і фіксації окремих серій або діапазонів цих чисел відповідно до об'єктом класифікації з однаковими характеристиками, і їх присвоєння використовується для 2-значних позицій.

Послідовний засіб-це формування класифікаційного групуючого коду або об'єкта класифікації з використанням коду і присвоєння йому послідовно розташованих підгруп, отриманих за допомогою інструменту ієрархічної класифікації.

Паралельний інструмент використовує інструмент класифікації фасетів та незалежний код групування, отриманий під час його призначення, для створення класифікаційного коду групування або об'єкта класифікації.

При формуванні системи класифікації та кодування об'єктів використовуються різні комбінації методів класифікації та кодування, вибір яких залежить від призначення класифікатора, деталей розв'язуваних завдань і вибору комп'ютерної технології.

До нашого коду пред'являються наступні вимоги:

1. забезпечення розв’язання всіх задач системи за їх мінімальної довжини кодів;
2. єдність кодів на всіх рівнях;
3. структура коду повинна забезпечити групування інформації в необхідних розмірах;
4. коди можуть бути як внутрішньо-машинні, так і зовнішні.

Внутрішньо-машинні коди використовуються обчислювальною системою, а зовнішні, крім цього, й користувачем.

### 3.3. Структура баз даних та інформаційних масивів.

База даних (DB) - це впорядкований набір логічно взаємопов'язаних даних, що використовуються спільно і призначені для задоволення інформаційних потреб користувача. Технічно кажучи, вона також включає систему управління базами даних[15].

Основне завдання бази даних-забезпечити зберігання значного обсягу інформації та надати доступ користувачам або прикладним програмам. Таким чином, база даних складається з 2 частин: збереженої інформації та системи управління нею. Для забезпечення ефективного доступу записи даних організовані у вигляді послідовності фактів

Система управління базами даних (СУБД) - це набір програмних і мовних засобів, необхідних для створення бази даних, підтримки її в актуальному стані і організації пошуку потрібної Вам інформації.Основні можливості СУБД[15]

1. Поповнення, розширення та відновлення БД;
2. Висока надійність зберігання інформації;
3. Засоби захисту інформації в СУБД;
4. Виведення повної й достовірної інформації на запити користувача.

База даних базується на моделі даних - фіксованій системі концепцій та правил для представлення структури даних, стану та динаміки проблемних областей у базі даних. Ієрархічні, мережеві та реляційні моделі даних послідовно використовувалися в різні епохи. Сьогодні все більшого поширення набуває об'єктно-орієнтований підхід до конфігурації баз даних ГІС.

Типи структур баз даних[15]:

Ієрархічна модель даних

Часто об'єкти знаходяться у відносинах, званих ієрархіями: часткові відносини, відносини видів, відносини підпорядкування. Об'єкти в ієрархічному взаємозв'язку утворюють дерево "орієнтований граф" тільки з 1 вершиною, яка не підпорядкована іншій вершині (ця вершина називається коренем дерева).

Концептуальна схема ієрархічної моделі являє собою набір типів записів, пов'язаних з 1 або більше деревами типом посилання. Всі типи зв'язків в цій моделі відносяться до типу "від 1 до декількох" і зображені стрілками.

Мережева модель даних

У мережевій моделі даних поняття головного і підлеглого об'єктів дещо розширені. Будь-який об'єкт може бути як провідним, так і підлеглим (в мережевій моделі головний об'єкт визначається терміном "власник набору", і один і той же об'єкт може виконувати як роль власника, так і роль члена набору одночасно. Це означає, що кожен об'єкт може брати участь у будь-якій кількості взаємозв'язків.

Подібно ієрархічним моделям, мережеві моделі також можуть бути представлені у вигляді орієнтованих графів. Однак у цьому випадку граф може містити цикли.

Ця структура набагато гнучкіша та виразніша, ніж попередня, і підходить для моделювання більш широкого класу завдань. У цій моделі вершини-це сутності, а ребра, які їх з'єднують, - це відносини між ними.

Реляційна модель даних.

У реляційній моделі даних об'єкти та зв'язки між ними представлені за допомогою таблиць. Зв'язки також представлені у вигляді об'єктів. Кожна таблиця представляє 1 об'єкт і складається з рядків і стовпців. Таблиця повинна мати первинний ключ (ключовий елемент) — поле або комбінацію полів, які ідентифікують кожен рядок в таблиці єдиним способом.

Назва "реляційний" пов'язано з тим фактом, що кожен запис в таблиці даних містить інформацію, що відноситься до певного об'єкту. Крім того, різні типи взаємопов'язаних (тобто знаходяться в певних відносинах) даних в моделі можна розглядати як єдине ціле.

Ця таблиця має наступні властивості:

Кожен елемент таблиці представляє 1 елемент даних;

Немає повторюваних груп;

Це означає, що елементи стовпця мають однакову природу;

Стовпцю присвоюється унікальне ім'я;

У таблиці немає двох однакових рядків.

Порядок рядків і стовпців у таблиці довільний, і такий тип таблиці називається відношенням. У сучасній практиці термін " запис "використовується для рядків, а термін" поле " - для стовпців.

Основні принципи створення БД:

1. Висока швидкодія (малий час відгуку на запит).
2. Простота оновлення даних.
3. Незалежність даних.
4. Спільне використання даних багатьма користувачами.
5. Безпека даних - захист даних від навмисного чи ненавмисного порушення секретності, спотворення або руйнування.
6. Стандартизація побудови та експлуатації БД (фактично СУБД).
7. Адекватність відображення даних відповідної предметної області.

Вимоги до інформаційної підтримки наступні:

Інформаційна підтримка повинна бути достатньою для виконання всіх функцій автоматизованого ІР.

Для кодування інформації, яка використовується лише цим ІР, необхідно застосувати класифікатор, який має клієнт ІР.

Ви повинні використовувати Класифікатор цього рівня для кодування вихідної інформації, що використовується на найвищому рівні ІР, якщо не вказано інше.

Інформаційна підтримка ІС повинна поєднуватися з інформаційною підтримкою систем, які взаємодіють з ІС з точки зору змісту, систем кодування, методів адресації, форматів даних і форматів відображення інформації, отриманої та опублікованої інформаційною системою.

Формат документа, створеного інформаційною системою, повинен відповідати вимогам стандарту USD або нормативно-технічної документації замовника ІР.

Формат документа і відеокадрів, які вводяться або змінюються через ІР-термінал, повинен відповідати відповідним технічним характеристикам терміналу.

Формат подання вихідної інформації ІР повинен бути узгоджений з замовником (користувачем) системи.

Терміни і скорочення, використовувані в початковому повідомленні, повинні бути загальноприйнятими в даній предметній області і узгодженими з замовниками системи.

ІР повинен забезпечувати необхідні заходи для контролю та оновлення даних в інформаційному масиві ІР, оновлювати масив після збою технічних засобів ІР і контролювати ідентифікатор однойменної інформації в базі даних.

Microsoft SQL від Постачальника .NET Framework для SQL Server. Ця база даних є локальною для серверної програми і розташована безпосередньо в каталозі програми.

Microsoft SQL Server-це система управління базами даних, розроблена корпорацією Microsoft. Сервер даних виконує свою основну функцію зберігання та надання даних у відповідь на запити інших програм, працюючи як на одному сервері, так і в мережі[4].

Мова, що використовується для запитів, - Transact-SQL, написана спільно Microsoft та Sybase. Transact-SQL-це реалізація стандарту ANSI / ISO для мови структурованих запитів SQL з розширеннями. Він використовується як для баз даних малого та середнього розміру, так і для баз даних великого підприємства. Протягом багатьох років він успішно конкурував з іншими системами управління базами даних.

Microsoft SQL Server використовує версію SQL під назвою Transact-SQL (скорочено T-SQL) як мову запитів, яка є реалізацією SQL-92 (стандарт ISO для SQL) з багатьма розширеннями. T-SQL дозволяє використовувати додатковий синтаксис збережених процедур і забезпечує підтримку транзакцій (взаємодія бази даних з додатком управління). Microsoft SQL Server і Sybase ASE взаємодіють з мережею, використовуючи протокол

прикладного рівня, який називається tabular data stream (TDS, протокол передачі табличних даних).

Microsoft SQL Server також підтримує Open Database Connectivity (ODBC) — інтерфейс для взаємодії з додатками і СУБД. Версія SQL Server 2005 надає можливість підключатися до користувачів через служби веб-сервера з використанням протоколу soap.

Це дозволяє клієнтським програмам, що не належать до Windows, підключатися між платформами до SQL Server. Microsoft також випустила сертифікований драйвер JDBC, який дозволяє програмам на базі Java (таким як BEA та IBM Websphere) підключатися до Microsoft SQL Server 2000 та 2005.

SQL Server підтримує дзеркальне відображення бази даних та кластеризацію. Кластер SQL server - це однаково настроюваний набір серверів, і ця схема допомагає розподіляти навантаження між кількома серверами. Усі сервери мають однакове віртуальне ім'я, і дані розподіляються за IP-адресою комп'ютера кластера під час робочого циклу. Крім того, в разі збою одного з серверів кластера доступна автоматична передача навантаження на інший сервер.

SQL Server підтримує надмірне дублювання даних у наступних 3 сценаріях[3]:

1. Знімок: Виконується «знімок» бази даних, який сервер відправляє одержувачам.
2. Історія змін: Всі зміни бази даних безперервно передаються користувачам.
3. Синхронізація з іншими серверами: Бази даних декількох серверів синхронізуються між собою. Зміни усіх баз даних відбуваються незалежно на кожному сервері, а під час синхронізації відбувається звірка даних. Дублювання такого типу передбачає можливість вирішення протиріч між базами даних.

SQL Server 2005 має вбудовану підтримку .NET Framework. Завдяки цьому, процедури бази даних, що зберігаються, можуть бути написані на будь-якій мові платформи .NET з використанням повного набору бібліотек, доступних для .NET Framework. На відміну від інших процесів, .NET Framework виділяє додаткову пам'ять і будує засоби керування SQL Server, не використовуючи вбудовані засоби Windows. Це підвищує продуктивність порівняно із загальними алгоритмами Windows, оскільки алгоритми розподілу ресурсів спеціально налагоджені для використання у структурах SQL Server.

В подібному випадку використання локальної бази є декілька плюсів та мінусів такого підходу до архітектури БД. По-перше швидкість роботи буде швидше за рахунок відсутності виділеного серверу обробки запитів до Баз Даних. По-друге працювати з локальною БД більш зручніше і простіше ніж з сервером, наприклад відсутність складного адміністрування доступу до БД, по скільки єдиним користувачем БД є сервер дипломного проекту.

Але є і мінуси, наприклад з плином часу файл БД буде заповнюватись і займати багато простору на диску, а «відрізати» частину старих даних буде надто складно в порівнянні з MySQL Server.

Структура БД в проекті представлена в виді декількох таблиць. Перша таблиця представляє собою набір даних інформації про таксистів таблиця 3.1. Друга безпосередньо займається архівуванням даних про замовлення які вже виконані на момент запиту до таблиці. Третя таблиця є робочою, туди заносяться дані про активні замовлення, ті що на даний момент в ефірі або ті що виконуються таксистами.

Таблиця 3.1 – **Таблиця Car_Drivers**

| **Назва поля** | **Тип даних** | **Призначення** |
|---|---|---|
| Id | int | Унікальний ідентифікатор |
| Phone_Num | nchar(13) | Номер телефону таксиста, це поле використовується як логін для входу в Android додаток |
| Password | nchar(255) | Пароль для входу |
| Car_Num | nchar(10) | Державний номер авто таксиста, необхідний для передачі замовнику для ідентифікації авто |
| Car_Color | nchar(10) | Колір авто |
| Tech_Car_Num | nchar(50) | Тех. Номер автомобіля |
| Car_Type | nchar(50) | Марка автомобіля |

Дана таблиця є головною для додатка таксистів, оскільки містить їх дані. Вона розроблена безпосередньо за допомогою редактора баз даних Visual Studio (рисунок 3.1).

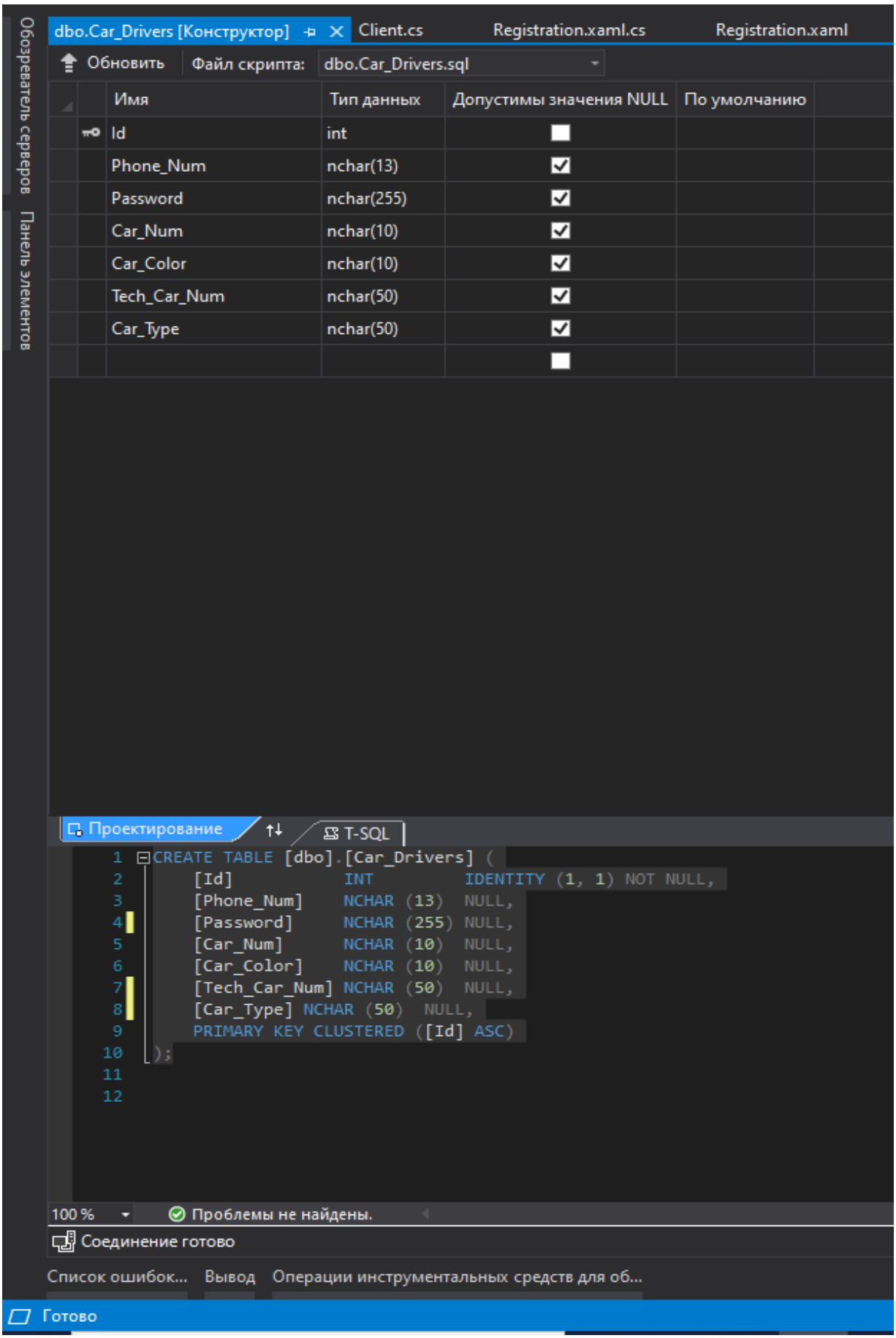


Рис. 3.1 - Редактор таблиці Car_Drivers

Наступна таблиця служить для зберігання даних про замовлення, за допомогою неї можливо робити аналіз ринку та отримувати дані про картину роботи сервісу в цілому оскільки зберігання такої інформації є стратегічно необхідно.

Таблиця 3.2 – **Таблиця Orders_Table**

| **Назва поля** | **Тип даних** | **Призначення** |
|---|---|---|
| Id | int | Унікальний ідентифікатор |
| ID_Car_Driver | nchar(13) | Ідентифікатор таксиста |
| Point_Departude | nchar(150) | Пункт відправлення |
| Arrival_Point | nchar(150) | Пункт призначення |
| Order_Create_Time | datetime | Дата та час створення замовлення |
| Order_Accept_Time | datetime | Дата та час прийняття замовлення |
| IsAccept | nchar(5) | Поле вказуюче чи було прийняте замовлення |
| IsEnd | nchar(5) | Поле вказуюче чи було завершене замовлення |

Описання деяких полів таблиці 3.2:

1. Поле ID_Car_Driver – використовується для прикріплення запису таксиста до запису замовлення коли той приймає замовлення на виконання, що дозволяє продивлятися інфонографіку по таксисту.
2. Поле Order_Create_Time – виступає для зберігання часу та дати створення замовлення, необхідне для пошуку невиконаних замовлень при перегляді статистичної інформації.
3. Поле Order_Accept_Time – виступає датою та часом прийняття замовлення, воно необхідне для.
4. Поле IsAccept – визначає чи було прийняте замовлення на виконання, що дозволяє зробити спеціальну мітку для замовлення при пошуку невиконаних замовлень. Заповнюється при прийнятті таксистом замовлення на виконання разом з полем ID_Car_Driver.
5. Поле IsEnd – мітка яка дозволяє з’ясувати, чи було замовлення виконане таксистом, чи ні.

Зберігання статистики є одним із основних положень успішного розвитку розитку бізнесу в світі, і дана таблиця реалізує його. Зображення редактора таблиці розташовано на (Рис. 3.2)

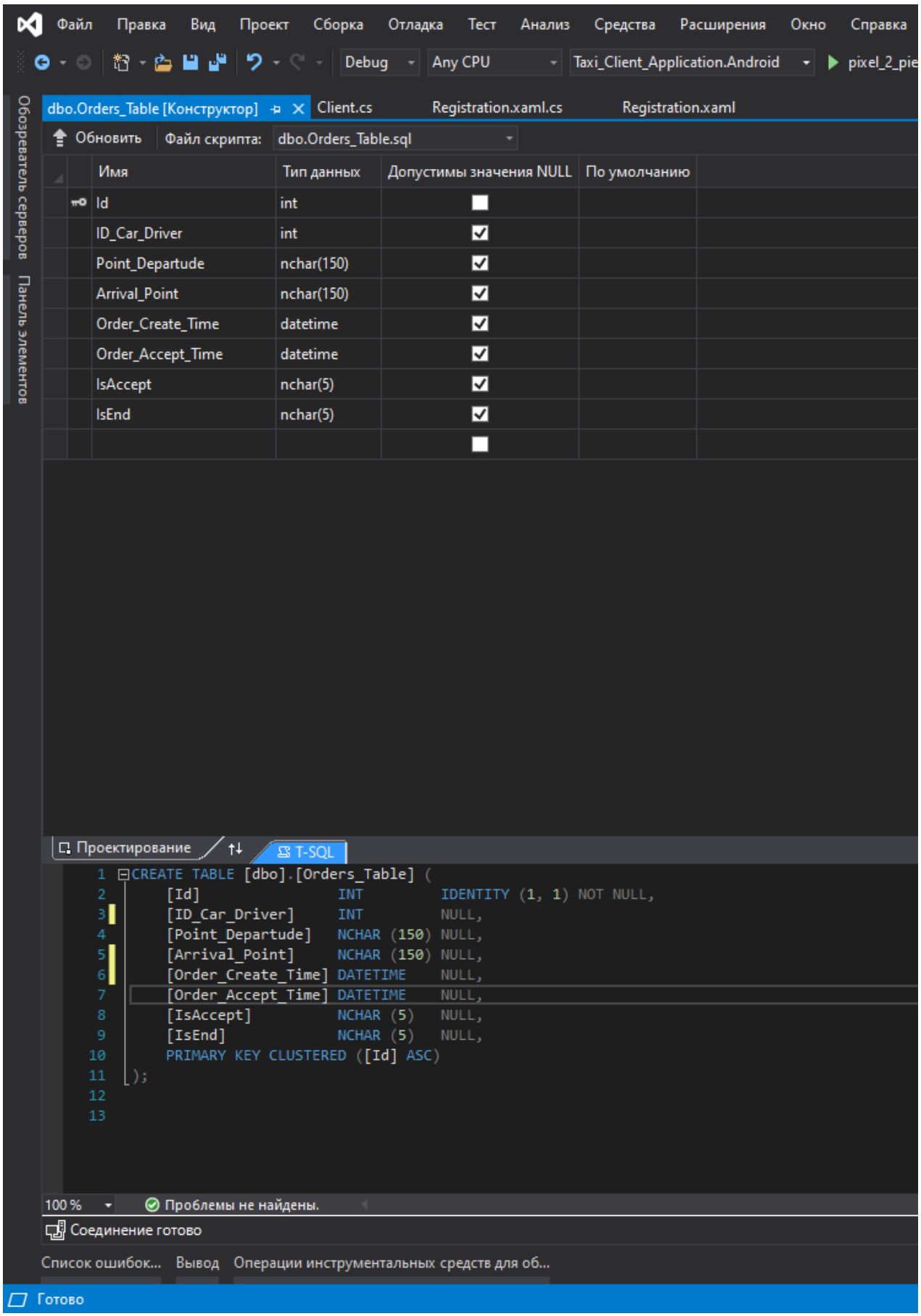


Рис. 3.2 - Редактор таблиці Orders_Table

Останньою таблицею в програмі є Active_Orders_Table таблиця 3.3. Вона є робочею таблицею оскільки використовується для тимчасового зберігання активних замовлень, ті що були недавно створені чи ті що в даний момент виконуються. Вона була створена мною оскільки необхідно забезпечити швидку роботу з записами активних замовлень, оскільки при використанні таблиці з архівними замовлення швидкість роботи може дуже сильно знизитись по причині надзвичайно великої кількості записів.

Таблиця 3.2 – **Таблиця Active_Orders_Table**

| **Назва поля** | **Тип даних** | **Призначення** |
|---|---|---|
| Id | int | Унікальний ідентифікатор |
| ID_Car_Driver | nchar(13) | Ідентифікатор таксиста |
| Point_Departude | nchar(150) | Пункт відправлення |
| Arrival_Point | nchar(150) | Пункт призначення |
| Order_Create_Time | datetime | Дата та час створення замовлення |
| Order_Accept_Time | datetime | Дата та час прийняття замовлення |
| IsAccept | nchar(5) | Поле вказуюче чи було прийняте замовлення |
| IsEnd | nchar(5) | Поле вказуюче чи було завершене замовлення |

Дана таблиця повністю схожа на архівну, але заради швидкості роботи серверної програми вона необхідна.

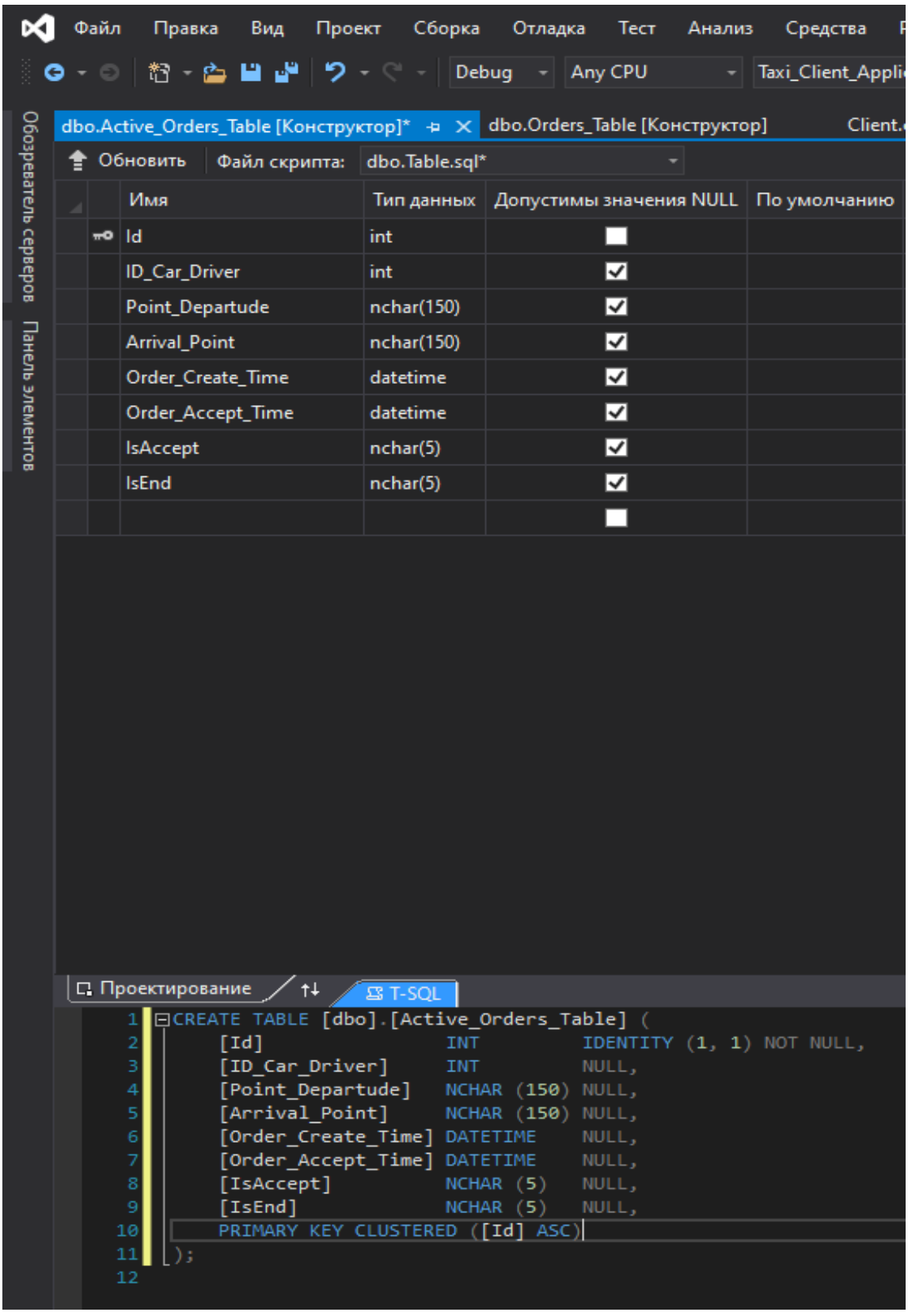


Рис. 3.2. Редактор таблиці Active_Orders_Table

## Висновки до розділу 3

Завдання 3-го розділу передбачає описання інформаційного забезпечення та розробку бази даних програми яка розробляється згідно з завданням до кваліфікаційної роботи.

Раніше, в часи перших комп'ютерів, нічого подібного БД не було, дані в ЕОМ вносились руками, задавалися алгоритмічні дії, а після проведення обчислення виводилися на перфо-стрічку.

З часом техніка розвивалася і постало питання про зберігання даних безпосередньо на комп'ютері, тоді і виникли перші бази даних, структурі їх зберігання и перші патерни  для роботи зними.

Сьогодні ніодна програма не працює без звязку з БД. У них можуть зберігатися терабайти даних які життєво необхідні для роботи як малого бізнесу, так і для великих корпорацій. У сфері науки ці терабайти даних декілька заменшена цифра, в БД НАСА наприклад зберігаються петабайти даних про вивчення вього, починаючи від таблички множення і закінчуючи астрофізикою і далекими галактиками.

# РОЗДІЛ 4
# РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗАДАЧІ

## 4.1. Опис головного модулю програми (головного вікна).

Програма для дипломної роботі за темою розробка програмного забезпечення для замовлення такси розробляється за допомогою технології Windows Presentation Foundation для розробки додатків під операційну систему Windows та Xamarin Forms під Android.

Основний проект має назву «Diplom» він в свою чергу має три під проекти серверний додаток з тою же назвою що й проект, додаток під платформу Android для таксистів «Taxi_Client_Application» та додаток для замовників «Custormer_Client_Application».

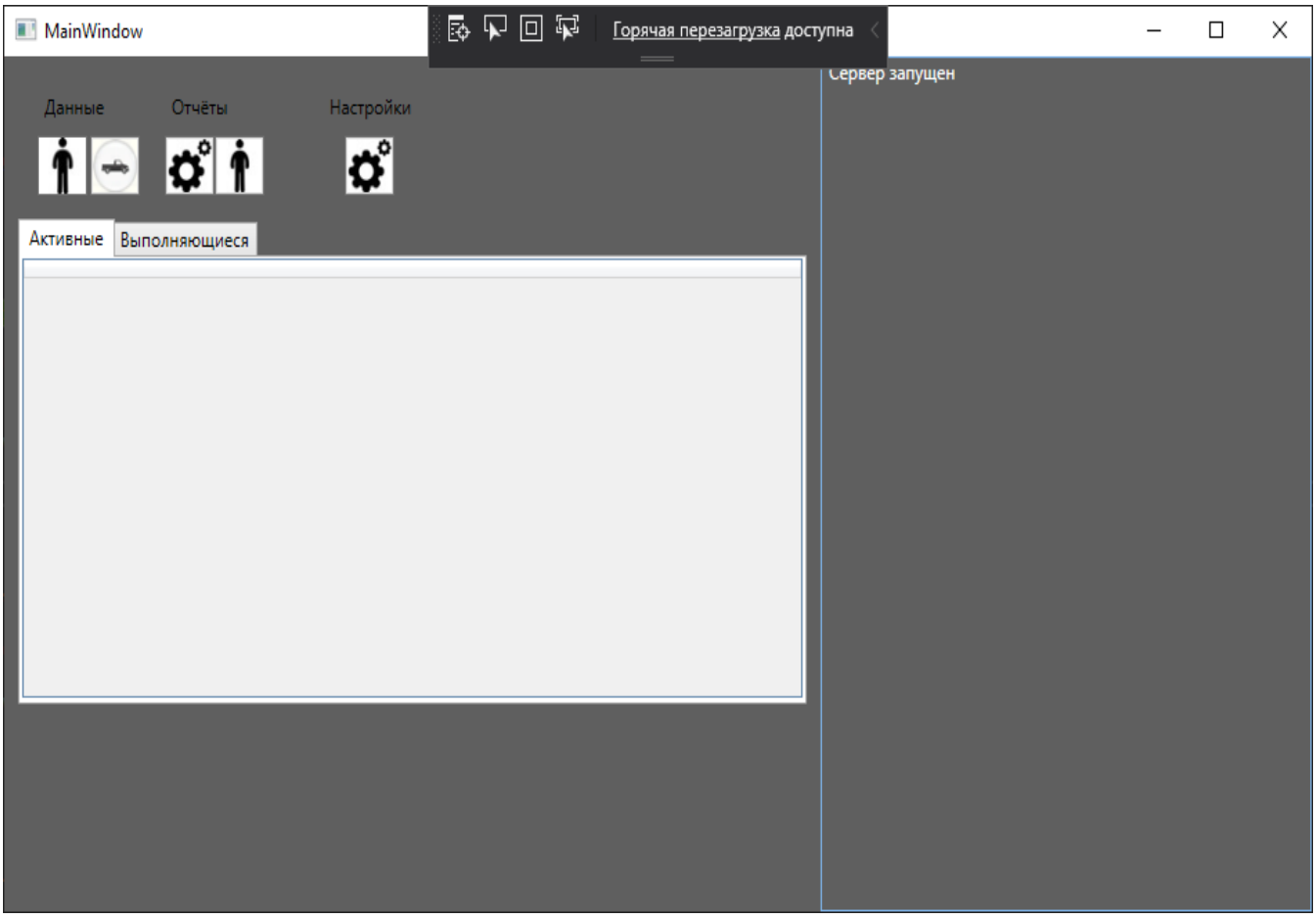


Рисунок - 4.1. Головне вікно серверу

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | КНУ.РМ.123.23.02.РІ | | |
| Змн. | Арк. | № документа | Підпис | Дата | | | |
| Розробив | | Поспєлова | | | РОЗДІЛ IV | Літера | Аркуш | Аркушів |
| Перевірив | | Музика | | | | | | |
| | | | | | | ЗКІ-22м | | |
| Н.контроль | | Кузнєцов | | | | | | |
| Затвердив | | Купін | | | | | | |

Головним являється серверний додаток (Рисунок 4.1), він складається:

1. MainWindow – головне вікно програми, при його запуску автоматично стартує і сервер.
2. Server.cs – клас слухача сервера, призначений для сканування вхідних запитів на підключення клієнтів.
3. Client.cs – клас роботи з клієнтами, містить всю логіку роботи для вхідних запитів та відправки відповідей для клієнтів підключених до серверу.
4. ReportWindow – вікно для відображення запитів до БД, з ціллю продивитись її дані.

Програмні продукти розробляються з використанням засобів розробки мови програмування Microsoft Visual C#.

Історія основних гілок мов програмування, що призвели до появи C#, сягає 60-х років, а саме епохи появи мови B. останній є типовим представником ранніх імперативних мов програмування. Мова B була винайдена в 1963 році творчою командою розробників, а головним творцем мови був К.Томпсон з Массачусетського технологічного інституту (Ken Thompson, MIT).Вважається, що це Томпсон. Основною метою розробки мови була реалізація операційної системи UNIX. Існуюча мова PL / I, що використовувалася в мейнфреймах IBM на той час, була досить громіздкою і не підходила до поставленої задачі, ніж нове оригінальне рішення вчених і практиків.

Наступним кроком в" алфавіті " мов програмування, що призвів до мови C#, стала мова C, винайдена на основі мови B в 1972 році. Автором нової мови програмування був К.К., який працював в Інституті AT&T (AT&T Bell Telephone Laboratories).Томпсон і Д.Річі (Dennis Ritchie) є на Facebook. Версія мови с розширила B для явного використання типів, структур та багатьох нових операцій. Подальша розробка мови відбувалася в тій же організації. І знову приблизно через 10 років, у 1984 році Б. Б'ярне Страуструп (Bell Labs) виступив з проектом для мови C++, розширенням ООП для мови C, яке вводить концепцію класів як об'єктів даних.

Назва C++ в новій мові-R. запропоновано Маскітті (Rics Mascitti, Bell Labs).

Нарешті, вже в 2000 році, тобто більше 15 років потому, Microsoft випустила в світ нове покоління C++ під назвою C#."Мова заснована на суворій компонентній архітектурі та реалізує передові механізми захисту коду.

Мова програмування C# являє собою поєднання кращих рис багатьох своїх попередників. На додаток до вищезгаданих гілок мови BC-C++, в наш

час необхідно вказати ще кілька важливих мов програмування: Java і Visual Basic.

Об'єктна модель com components (основний стандарт Microsoft для проектування компонентів та реалізації програмного забезпечення) та модель Java Beans, базовий стандарт для компонентів Sun Microsystems природно, мова програмування C# успадковує багато функцій від попередника мови visual Basic, створеної Microsoft[2].

Мова програмування C# базується на суворій компонентній архітектурі та реалізує вдосконалені механізми захисту коду.

Давайте перерахуємо найбільш характерні риси подібності між мовами програмування C# і Java.

Перш за все, обидві мови відносяться до категорії об'єктно-орієнтованих і припускають унікальність імітації. Інші важливі особливості, що зближують мови програмування C # та Java, включають механізми інтерфейсу, обробку винятків, процеси або "потоки". Збір сміття та простори імен реалізовані однаково в цих двох мовах. Обидві мови програмування характеризуються суворою типізацією і динамічним завантаженням коду при виконанні програм.

Від свого прямого попередника, мови програмування C++, мова C # успадковує" перевантажені " оператори, небезпечні арифметичні операції з плаваючою комою і багато інших синтаксичні особливості, але, незважаючи на те, що багато конструктивні синтаксичні механізми і особливості реалізації успадковані від попередників (C++, Visual Basic, Java), судячи з мови програмування c #, ця нова мова поки недоступний. Функціональність нової мови програмування не обмежується сумою функцій її історичних попередників.

Деякі з принципово важливих рішень, реалізованих Microsoft на мові програмування C#, включають[1]:

1. компонентно-орієнтований підхід до програмування (який характерний і для ідеології Microsoft. NET в цілому);
2. властивості як засіб інкапсуляції даних (характерно також в цілому для ООП);
3. обробка подій (маються розширення, в тому числі в частині обробки виключень, зокрема, оператор try);
4. обробка подій (маються розширення, в тому числі в частині обробки виключень, зокрема, оператор try);
5. делегати (delegate - розвиток покажчика на функцію в мові C#);
6. індексатори (indexer - оператори індексу для звернення до елементів класу-контейнера);
7. перевантажені оператори (розвиток ООП);
8. оператор foreach (обробка всіх елементів класів-колекцій, аналог

Visual Basic);

9. механізми boxing і unboxing для перетворення типів;

10. атрибути (засіб оперування метаданими в COM-моделі);

11. прямокутні масиви (набір елементів з доступом за номером індексу і однаковою кількістю стовпців і рядк До числа принципово важливих рішень, які реалізовані корпорацією Microsoft у мові програмування C #, можна віднести наступні:

12. компонентно-орієнтований підхід до програмування (який характерний і для ідеології Microsoft. NET в цілому);

ADO .NET (ActiveX Data Objects .NET) є набором класів, що реалізують програмні інтерфейси для полегшення підключення до баз даних програми незалежно від особливостей реалізації конкретної системи управління базами даних і від структури самої бази даних, а також незалежно від місця розташування цієї самої бази - зокрема, в розподіленому середовищі (клієнт-серверний додаток) на стороні сервера[22].

ADO .NET широко використовується спільно з технологією web-програмування з використанням об'єктів ASP.NET для доступу до розташованих на сервері баз даних з боку клієнта.

Особливість викладу матеріалу цього розділу полягає в наступному.

Рішення навіть найпростішої завдання, пов'язаної з даними, передбачає використання безлічі різноманітних об'єктів - представників класів ADO .NET, які знаходяться між собою в досить складних взаєминах. Через ці відносини строго послідовний опис елементів ADO .NET є досить проблематичним. З якого б елемента не починався опис, завжди передбачається попереднє уявлення про безліч інших елементів.

З цієї причини часто згадка і навіть приклади використання деяких класів передують їх докладного опису.

Робота з БД на рівні додатки .NET - це робота[20]:

1. з великою кількістю оголошень класів, які містять оголошення успадкованих методів і властивостей, призначених для вирішення завдання добування інформації з бази даних;

2. з безліччю об'єктів-представників класів, які забезпечують роботу з базами даних;

3. з безліччю значень і властивостей конкретних об'єктів, що відбивають специфіку структури конкретної бази даних.

Функціональні особливості цієї складної системи взаємодіючих класів забезпечують однакову роботу з базами даних незалежно від системи управління базою і її реалізації.

Звичайно ж, при написанні програми, яка взаємодіє з базами даних, програміст все може зробити своїми руками. Правда, в силу складності завдання (багато всяких деталей доведеться виточувати), часу на розробку такого додатка може знадобитися досить багато.

Для програміста - розробника програми принциповою стає інформація про логічнуорганізацію (структуру таблиць, відносини і обмеженнях) даної конкретної бази даних - тобто про те, як цю базу бачить додаток.

Діяльність програміста - розробника додатків для роботи з базами даних в основі своїй нічим не відрізняється від того, що було раніше. Ті ж оголошення класів і інтерфейсів.

А тому бажано:

1. розуміти принципи організації і взаємодії класів, які забезпечують роботу з базою даних;

2. представляти структуру, призначення та принципи роботи відповідних об'єктів;

3. знати, для чого і як застосовувати різні деталі при організації взаємодії з базами даних;

4. вміти створювати компоненти ADO.NET заданої конфігурації з використанням допоміжних засобів (чарівників), що надаються в рамках Visual Studio .NET.

В частини ADO.NET, що підключається, є наступні основні класи:

Connection. Цей клас, що дозволяє встановлювати підключення до джерела даних. (OleDbConnection, SqlConnection, OracleConnection)

Transaction. Об'єкт транзакцій (OleDbTransaction, SqlTransaction, OracleTransaction. В ADO.NET є простір імен System.Transaction)

DataAdapter. Це своєрідний шлюз між автономними і підключеними аспектами ADO.NET. Він встановлює підключення, і якщо підключення вже встановлено, містить достатньо інформації, щоб сприймати дані автономних об'єктів і взаємодіяти з базою даних. (DataAdapter - SqlDataAdapter, OracleDataAdapter)[20]

Command. Це клас представляє виконувану команду в базовому джерелі даних.

Parameter. Об'єкт параметр команди.

DataReader. Це еквівалент конвеєрного курсора з можливістю тільки читання даних в прямому напрямку.

Об'єкти DataSet дозволяють викликає збірці (на зразок веб -сторінки або програми, що виконується на настільному комп'ютері) працювати з вмістом DataSet, змінювати його, не вимагаючи підключення до джерела даних, і відправляти назад блоки змінених даних для обробки за допомогою відповідного адаптера даних. Але , мабуть, саме фундаментальна відмінність між класичною ADO і ADO.NET полягає в тому, що ADO.NET є керованою кодової бібліотекою, і , значить, підкоряється тим же правилам , що і будь-яка керована бібліотека. Типи , складові ADO.NET , використовують протокол управління пам'яттю CLR , належать до тієї ж системи типів (класи, інтерфейси, перерахування, структури і делегати ) , і доступ до них можливий за допомогою будь-якої мови .NET. Класи ADO.NET знаходяться в збірці System.Data.dll.

Інтегрованим середовищем розробки я вибрав продукт корпорації Microsoft - IDE Microsoft Visual Studio Community 2019. Це потужне середовище розробки яке дозволяє створювати програми на різних мовах програмування та під різні платформи.

Microsoft Visual Studio-це лінійка продуктів Microsoft, яка включає

інтегроване середовище розробки програмного забезпечення та безліч інших інструментів. За допомогою цих продуктів ви можете: Windows, Windows Mobile, Windows CE,. NET Framework, Xbox, Windows Phone.NET ви можете розробляти як консольні програми, так і програми з графічними інтерфейсами, включаючи ті, що підтримують технологію Windows Forms, як у власному, так і в керованому коді для Compact Framework, усіх платформ, що підтримуються Silverlight.

Visual Studio включає редактор вихідного коду, що підтримує технологію IntelliSense і дозволяє легко виконувати рефакторинг вашого коду. Вбудований налагоджувач функціонує або як налагоджувач вихідного коду, або як налагоджувач машинного рівня. Інші вбудовані інструменти включають редактор форм, веб-редактор, конструктор класів та Data Visual Studio, що полегшує створення графічного інтерфейсу для вашої програми, додавання підтримки систем контролю версій вихідного коду (наприклад, Subversion та Visual SourceSafe), додавання нових наборів інструментів (наприклад, перетягування), а також додавання нових наборів інструментів (наприклад, перетягування)[19]. Створюйте та розгортайте сторонні програми (плагіни), які розширюють функціональність майже на кожному рівні, наприклад, додаючи інші аспекти процесу розробки програмного забезпечення (наприклад, редагування основною мовою програмування або візуальний дизайн коду) або додаючи клієнт Team Explorer для роботи з Team Foundation Server. Ви можете підключити його до комп'ютера.Адроїд додатки розроблялись за допомогою мови програмування Xamarin основаній на мові написання серверу. Серед додатків головним є додаток для таксистів.

Структура додатка «Taxi_Client_Application»:

1. MainPage – головна навігаційна панель додатка, через неї можливо перейти до будь-якої сторінки в додатку.
2. Login – стартова сторінка додатка, має перехід до сторінки реєстрації.
3. Orders – сторінка де відображені всі замовлення які є в ефірі.
4. Registration – сторінка реєстрації
5. CurrentOrder – сторінка замовлення перебуваючого в виконанні.
6. Client.cs – клас з логікою роботи з сервером служби.

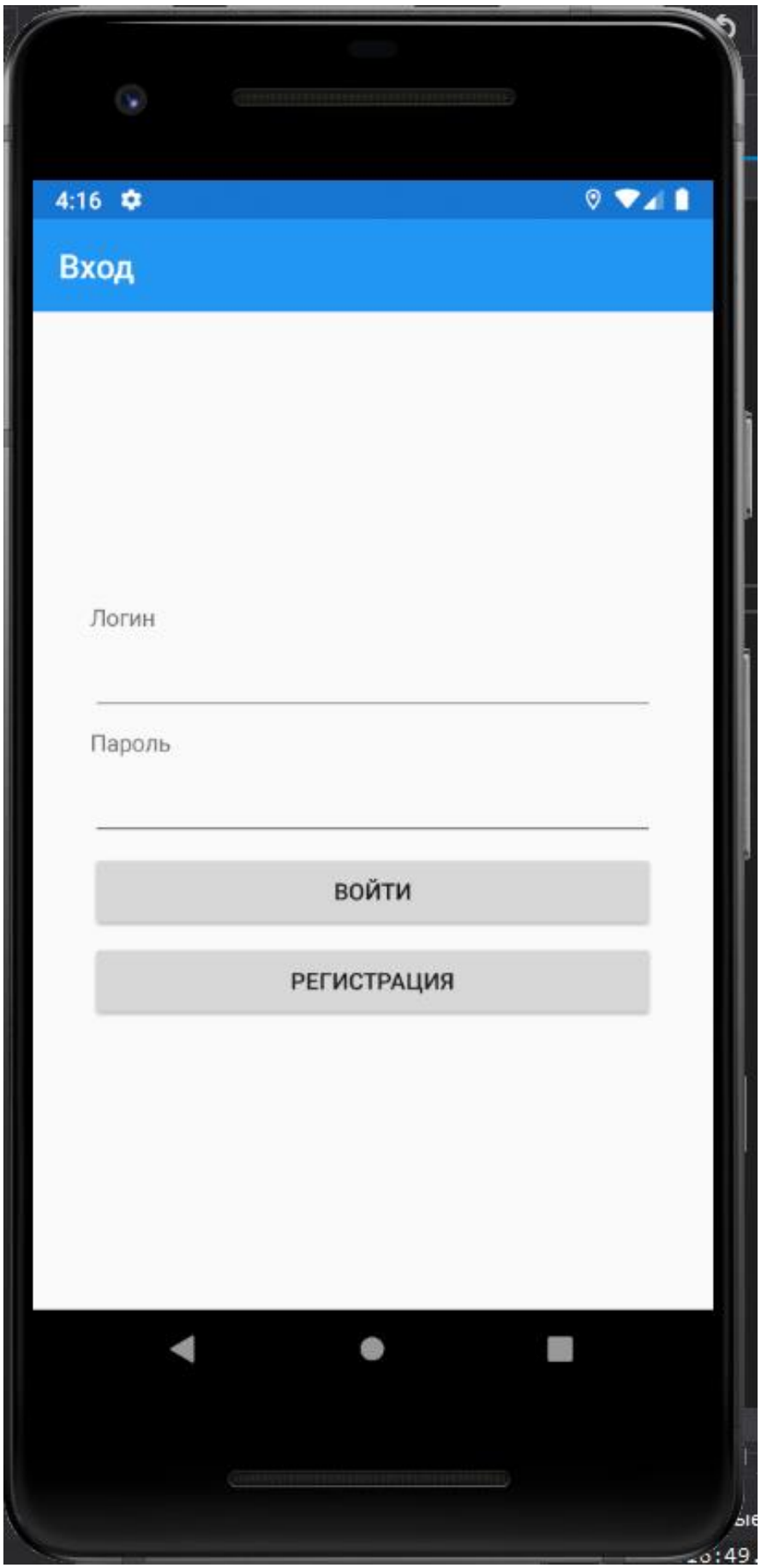


Рисунок 4.2 – Вікно авторизації додатку для таксистів

На рисунку 4.2 зображене вікно авторизації додатку для таксистів запущене на емуляторі андроїд телефону. В момент запуску додатка відбувається підключення до серверу, при авторизації відбувається перехід до сторінки з замовленнями в ефірі. Звідти теж вже доступне меню навігації додатка рис. 4.3.


Рис. 4.3 - Вікно авторизації додатку для таксистів

Xamarin - це платформа з відкритим вихідним кодом, призначена для побудови сучасних продуктивних додатків для iOS, Android і Windows з .NET. Платформа Xamarin є рівень абстракції, який забезпечує управління взаємодією між загальним кодом і кодом базової платформи. Xamarin виконується в керованому середовищі, яка реалізує такі можливості, як виділення пам'яті та збирання сміття[26].

Завдяки Xamarin в середньому 90% коду програми може використовуватися без змін на різних платформах. За допомогою цього шаблону розробник може написати всю бізнес-логіку на одній мові (або використовувати існуючий код додатка), але при цьому отримати характеристики продуктивності, оформлення і поведінку, характерні для кожної відповідної платформи.

Додатки Xamarin можна писати на ПК або Mac і компілювати у власні пакети додатків, наприклад в файли з розширенням .apk для Android або .ipa для iOS.

Платформа Xamarin створена на базі Mono, версії .NET Framework з відкритим вихідним кодом, заснованої на стандартах .NET ECMA. Mono існує майже так само довго, як і .NET Framework, і працює на більшості платформ, включаючи Linux, Unix, FreeBSD та macOS. Середовище виконання Mono автоматично обробляє такі завдання, як виділення пам'яті, прибирання сміття та забезпечення взаємодії з базовими платформами.

Xamarin поєднує в собі всі можливості існуючих платформ і ряд власних, включаючи наступні:

Повна прив'язка для базових пакетів SDK. Xamarin містить прив'язки практично для всіх базових пакетів SDK в iOS і Android. Крім того, ці прив'язки є строго типізований, що означає, що вони зручні в навігації і використанні, а також дозволяють здійснювати якісну перевірку типів під час компіляції і розробки. Строго типізовані прив'язки, які зменшують кількість помилок часу виконання і підвищити якість додатків.

Взаємодія Objective-C, Java, C і C ++. Xamarin дозволяє безпосередньо перейти бібліотеки Objective-C, Java, C і C ++ для більш ефективного використання різноманітного стороннього коду. Ця можливість дозволяє використовувати існуючі бібліотеки iOS і Android, написані на Objective-C, Java або C / C ++. Крім того, Xamarin пропонує проекти прив'язки для прив'язки власних бібліотек Objective-C і Java за допомогою декларативного синтаксису[26].

Сучасні конструкції мови. Додатки Xamarin написані сучасною мовою C #, який характеризується значними поліпшеннями в порівнянні з Objective-C і Java. Сюди входять динамічні функції мови, функціональні конструкції, наприклад лямбда-вирази, LINQ, функції паралельного програмування, універсальні шаблони і т. Д.

Надійна бібліотека базових класів (BCL). Додатки Xamarin використовують бібліотеку BCL .NET, велику колекцію класів зі всеосяжними і спрощеними можливостями, включаючи підтримку XML, баз даних, сериализации, операцій введення-виведення, рядків, мережевих

функцій і т. д. Існуючий код C # можна скомпілювати для використання в додатках, забезпечуючи доступ до тисяч бібліотек, які містять додаткові функції, що виходять за рамки BCL.

Сучасна інтегрована середовище розробки (IDE). Xamarin використовує сучасне середовище Visual Studio, в якій реалізовані такі можливості, як автозавершення коду, більш досконала система управління проектами і рішеннями, вичерпна бібліотека шаблонів проектів, інтегрована система управління версіями і багато іншого.

Підтримка кроссплатформенних мобільних додатків. Xamarin пропонує вдосконалену кроссплатформенную підтримку для трьох основних платформ - iOS, Android і Windows. Обсяг загального коду в створених додатках може досягати 90%, а бібліотека Xamarin.Essentials пропонує універсальний API-інтерфейс для доступу до загальних ресурсів на всіх трьох платформах. Це дозволяє значно скоротити витрати на розробку і час випуску продуктів на ринок для розробників, що створюють мобільні додатки.

Додатки Xamarin.Android компілюються з мови C # в проміжний мова (IL), який під час запуску програми зазнає Just-in-Time-компіляцію (JIT) в машинну збірку. Додатки Xamarin.Android працюють в середовищі виконання Mono паралельно з віртуальною машиною середовища виконання Android (ART). Xamarin надає прив'язки .NET до просторів імен Android. * I Java. *. Середовище виконання Mono звертається до цих просторів імен з використанням керованих викликаються оболонок (MCW) і надає середовищі виконання ART викликаються програми-оболонки Android (ACW), завдяки чому обидві середовища можуть викликати код один одного.

Xamarin.Forms - це платформа для користувача інтерфейсу з відкритим вихідним кодом. За допомогою Xamarin.Forms розробники можуть створювати додатки для iOS, Android і Windows на основі загальної бази коду. Xamarin.Forms дозволяє розробникам створювати призначені для користувача інтерфейси в XAML за допомогою коду програмної частини в C #. Ці інтерфейси на кожній платформі готуються до перегляду як власні елементи управління. Нижче наведені деякі приклади функцій, що надаються Xamarin.Forms:

Мова призначеного для користувача інтерфейсу XAML

1. прив'язка даних
2. жести
3. вироблений ефект
4. завдання стилю

### 4.2. Опис розроблених класів та діалогів.

В дипломному проекті із реалізованих класів для описання можна відвести декілька. Це класи клієнтів додатків та клас слухача серверу.

Клас «Server.cs» є лише в єдиному екземплярі, розташований на сервері. Його пряме завдання прослуховування IP-адреси и виділеного порта як точку входу в програму.

Слухач серверу реалізовано за асинхронною структурою, що дозволяє йому виконувати свої дії при цьому не займаючи головний потік інтерфейсу програми[7].

```
    private async void Run()
    {
        TcpListener.Start();

        parentWin.hookMessage("Сервер запущен" +
Environment.NewLine);

        await waitConnect();
    }

    private async Task waitConnect()
    {
        var tcpClient = await TcpListener.AcceptTcpClientAsync();
        clientsList.Add(tcpClient);

        parentWin.hookMessage("Пользователь подключён" +
Environment.NewLine);

        await processClientTearOff(tcpClient);


        await waitConnect();
    }
```

Код зображений вище відповідає за запуск та прослуховування порту 11000 на предмет спроби клієнта підключитись до серверу. Якщо клієнт намагається підключитись потік слухача реагує на це та запускає метод підключення клієнта до серверу

Клас «Client.cs» є як на сервері, так і на андроїд додатках, але вони в собі мають різну логіку взаємодії, логіка сервера базується на передачі відповіді клієнтам, оскільки в свою чергу клієнті посилають запроси на сервер і чекають відповіді для подальшої роботі.

Архітектура даної технології побудована за принципом, що слухач ловить запит клієнта і створює на сервері об’єкт цього клієнта я навігаційну мережеву точку.

В об'єктно-орієнтованому програмуванні клас - це спеціальна структура, яка використовується для групування пов'язаних змінних та функцій. У той же час, згідно з термінологією ООП, глобальна змінна класу (змінна-член) називається полем даних (також властивістю або атрибутом), а функція-член називається методом класу[8].

Створений та ініціалізований екземпляр класу називається об'єктом класу. На основі одного класу ви можете створити безліч об'єктів, стан

(значення полів) яких відрізняються один від одного.

На основі класу ви можете створювати підкласи, успадковуючі властивості та поведінку батьківського класу. Таким чином, ви можете створити цілу ієрархію класів. Різні мови реалізують механізми успадкування дещо по-різному. Існує множинне успадкування та одиночне успадкування. Множина-це коли підкласи створюються на основі якогось прямого батьківського елемента (як у мові програмування C++). Одиночне успадкування - це коли клас може мати 1 прямого батьківського елемента (Мова програмування Java). Суперкласи можуть мати власні суперкласи, а підкласи також можуть бути суперкласами певного класу.

Поведінка об'єкта реалізується за допомогою методів. Майже вся робота з об'єктами виконується за допомогою методів. Ви можете змінити стан об'єкта або надати доступ до даних, інкапсульованих в об'єкті. Існує кілька типів методів, які мають деякі відмінності в різних мовах програмування. Методам і полям даних можуть бути надані різні дозволи, що визначають доступ з різних частин програмного коду. Права доступу і типи методів задаються модифікаторами при написанні методу. Методи, які створюють та ініціалізують екземпляри класу, називаються конструкторами КЛАСІВ. Метод, який знищує об'єкт, називається деструктором класу.

У програмуванні існує поняття програмного інтерфейсу.Це означає список можливих обчислень, які може виконувати та чи інша частина програми.1Це включає опис того, які аргументи і в якому порядку вони повинні передаватися алгоритму з цього списку, і в якій формі вони повертаються. Інтерфейс абстрактного типу даних був винайдений для формалізованого опису таких списків. Сам алгоритм, тобто фактичний програмний код, який виконує всі ці обчислення, не задається інтерфейсом, він програмується індивідуально і називається реалізацією інтерфейсу.

Програмні інтерфейси та класи можуть бути розширені шляхом успадкування, що є одним із ключових засобів повторного використання готового коду в ооп.1 успадкований клас або інтерфейс включатиме все, що вказано для всіх його батьківських класів (мови програмування та prats, наприклад, ви можете створити власну версію текстового рядка, успадкувавши клас "текстовий рядок" від існуючого класу "текстовий рядок", але програміст не зможе використовувати його). Це пов'язано з тим, що екземпляри обох класів сумісні між програмними інтерфейсами[11].

Однією з проблем структурного програмування, з якою бореться ООП, є проблема підтримки правильних значень програмних змінних. У багатьох випадках різні програмні змінні зберігають логічно пов'язані значення, і програміст несе відповідальність за підтримку цієї логічної узгодженості.Це означає, що зв'язок не підтримується автоматично. Прикладом, згідно з правилами відділу кадрів, є пара прапорів: "звільнений" і "очікуйте премію в кінці року", якщо особа не звільнена і не очікує премію одночасно, або не звільнена і не очікує премію в кінці року. бонус в той же час, або звільнений і не очікує отримання бонусу в той же час. Тобто будь-яка частина програми, в якій встановлено прапорець "звільнений", завжди повинна знімати прапорець

"очікування бонусів в кінці року".

Хороший спосіб вирішити цю проблему-встановити прапорець "released", який недоступний для змін у всіх розділах програми, за винятком тих, які спеціально вказані в цьому спеціально зазначеному розділі, коли все написано правильно один раз і всі інші хочуть встановити або зняти прапорець"Released". Ви повинні звертатися до цього розділу в будь-який час.

В об'єктно-орієнтованій програмі прапорець "звільнений" оголошується як закритий член класу, і для його читання записується відповідний відкритий метод. Правило, що визначає можливість або неможливість прямої зміни змінної, називається правилом завдання в області доступу. Слова "private" і "public"в даному випадку є так званими "модифікаторами доступу". У деяких мовах це називається "модифікатором", оскільки воно використовується для зміни раніше встановлених прав під час успадкування класу. Тобто, кожен розділ коду має базове правило, згідно з яким, в залежності від класу, до якого він належить, певний елемент (member) цього класу та інших класів, включаючи змінні, методи, функції, константи і т.д.: ніщо в одному класі не може бачити закритий елемент іншого класу. Що стосується інших, більш складних правил, то існують інші модифікатори доступу і правила взаємодії з класами на різних мовах[17].

Майже всі члени класу можуть встановлювати модифікатори доступу (за винятком статичних конструкторів та іншого). Більшість об'єктно-орієнтованих мов програмування підтримують наступні модифікатори доступу:

private - доступ до члена дозволений тільки з методів класу, в якому цей член визначений. Спадкоємці класу більше не матимуть доступу до цього члена. ПРИВАТНЕ успадкування забороняє дочірнім класам отримувати доступ до всіх членів батьківського класу (включаючи загальнодоступних членів (c++))[13];

захищений - доступ до елемента дозволений з методів класу, в якому цей елемент визначено, і з методів його КЛАСІВ-наступників. Успадкування за типом protected створює всі відкриті члени батьківського класу.Члени захищеного класу-наступника (C++)[13];

Відкритий доступ до елемента дозволений з будь-якого коду. Успадкування за типом public не змінює модифікатор батьківського класу (c++);.

### 4.3. Опис створених функцій.

Методи в об'єктно - орієнтованому програмуванні-це підпрограми (методи класу), які використовуються виключно в класі (метод класу) або об'єкті (метод екземпляра).

Існують прості методи та статичні методи (методи класу:

1. Простий метод може отримати доступ до даних об'єкта (конкретного екземпляра цього класу).

2. Статичні методи не мають доступу до даних об'єкта, і вам не потрібно створювати екземпляр (цього класу), щоб використовувати їх.

Першим методом можна описати метод реєстрації нового таксиста в системі. Завдання просте необхідно занести дані таксиста в форму на сторінкі реєстрації рис 4.4 та натиснути клавішу зареєструватися.

Як видно з коду методу при натисненні на клавішу із текстових полів формується запит на внесення нового запису в БД, потім викликається метод класу «Client.cs» для відправлення запиту на сервер де message самме повідомлення з запитом, а цифра «2» це операційний код запиту по якому сервер розуміє що даний запит направленй на реєстрацію нового таксиста і сервер починає виконувати дії закладені логікою програмного коду, після чого відправляє відповідь про вдалість операції.

```
private void btRegistr_Clicked(object sender, EventArgs e)
        {
            string message = @"INSERT INTO [Car_Drivers] ([Phone_Num],
[Password],
            [Car_Num], [Car_Color], [Tech_Car_Num])" +
                " VALUES ("+ txtPhoneNum.Text + "," +
txtPassword.Text + "," +
                    txtCarNum.Text + 
                "," + txtCarColor.Text + "," + txtTechNum.Text + ")"
+ @"\/";

            client.sendMessage("2", message);
        }
```

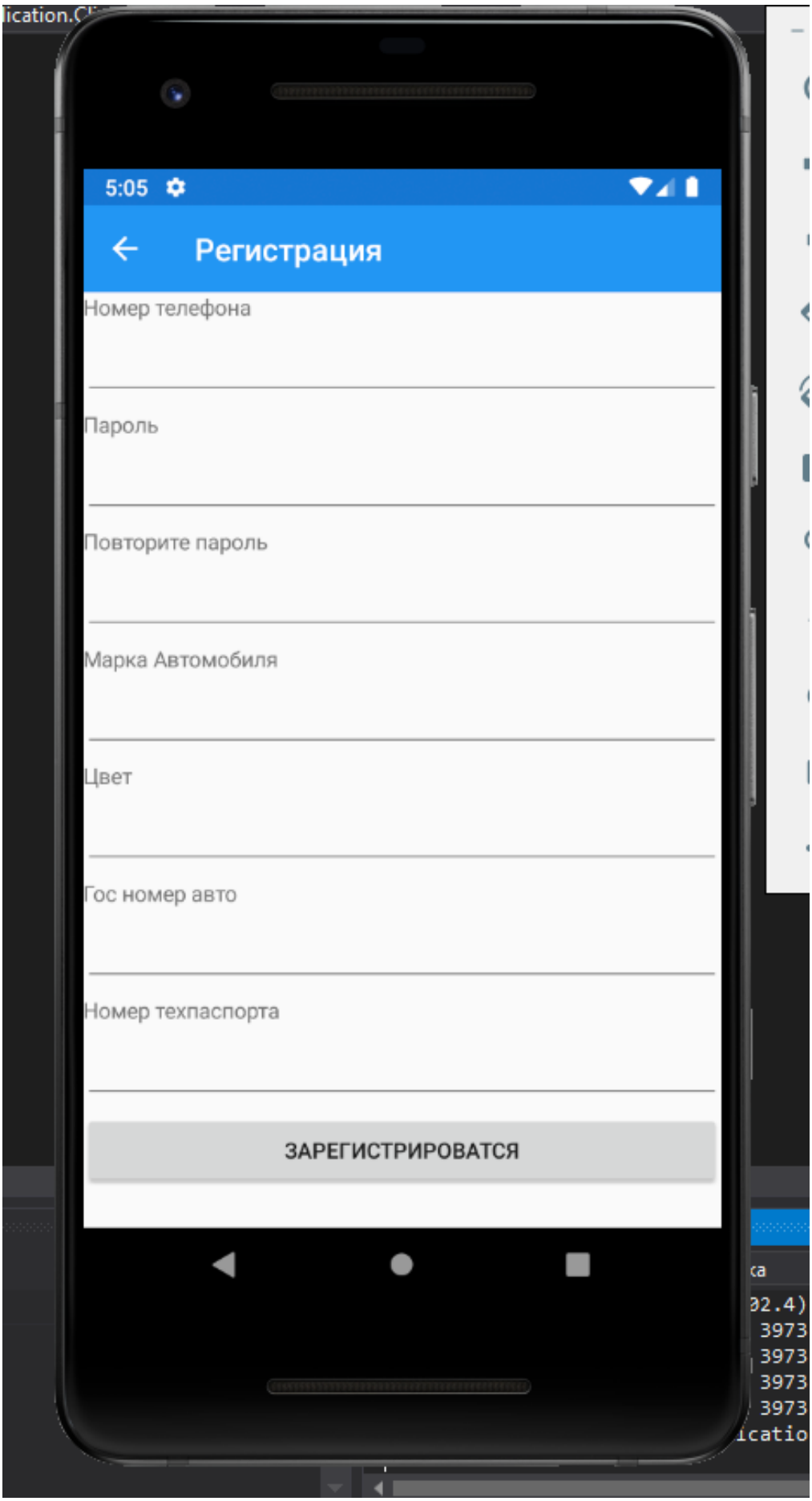


Рис. 4.4 - Реєстрації в додатку для таксистів

Наступний метод це запуск об’єкту клієнта, таксиста або замовника. Тут не має різниці адже вони обидва підключаються до серверу який знаходить в одному місці.

Як видно із коду, ми запускаємо асинхронне підключення і очікуємо доки екземпляр клієнта отримає сервер і підключення буде налагоджене. Потім із клієнта отримуємо поле мережевого потоку, воно необхідне для отримання і відправки повідомлень. Після чого запускаємо асинхронний метод очікування вхідних повідомлень від серверу.

```
private async Task startClient()
        {
            await
TcpClient.ConnectAsync(IPAddress.Parse("192.168.0.89"), 11000);

            netStream = TcpClient.GetStream();

            Console.WriteLine("Connect to server");

            await ProcessAsync();

        }
```

Наступний метод служить для відправки запросів від клієнта на сервер. Якщо придивитись до формування тексту самого повідомлення можна побачити що строка clientMessage складається із операційного коду та тексту запросу, як було вже описано вище. Далі все як звичайно конвертування повідомлення в бітовий формат і відправка запросу на сервер.

```
    ·public·async·void·sendMessage(string·operationCode,·string·
message)¶
........{¶
............string·clientMessage·=·operationCode·+·@"\/"·+·message;
¶
............byte[]·buffer·=·Encoding.UTF8.GetBytes(clientMessage);¶
¶
............await·netStream.WriteAsync(buffer,·0,·buffer.Length);¶
........}¶
```

Наступні два методи призначені для отримання відповідей від серверу, ця група методів єдина для всіх додатків дипломного проекту, що в свою чергу робить простішим програмування в проекті. Методи «ProcessAsync» і «ReadFromStreamAsync» працюють послідовно. Спочатку перший метод викликає другий в якому в свою чергу є асинхронна затримка яка очікує вхідне повідомлення. Як тільки воно приходить затримка закінчується і потік продовжує свою роботу. Повертаючи методу ProcessAsync текст повідомлення де він в свою чергу конвертує бітовий масив в вихідну строку повідомлення серверу. Далі по операційному коду в операторі Switch (код вилучений через великий розмір, дивитись додатки) виконується операція яка передбачена визначеним кодом операції.

```
        async·Task<byte[]>·ReadFromStreamAsync(int·nbytes)¶
·····{¶
·····var·buf·=·new·byte[nbytes];¶
·····var·readpos·=·0;¶
¶
·····readpos·+=·await·netStream.ReadAsync(buf,·readpos,·nbytes·
-·readpos);¶
¶
·····return·buf;¶
·····}¶
¶
·····public·async·Task·ProcessAsync()¶
·····{¶
·····var·actionBuffer·=·await·ReadFromStreamAsync(4096);¶
·····var·action·=·Encoding.UTF8.GetString(actionBuffer);¶
¶
·····string[]·strArr·=·action.Split(new·string[]·{·@"\/"·},·
StringSplitOptions.None);¶
¶
·····string·operationCode·=·strArr[0];¶
·····string·operationMessage·=·strArr[1];¶
¶
·····switch·(operationCode)¶
·····{¶
¶
·····}¶
¶
·····await·ProcessAsync();¶
·····}¶
```

Останній метод, це перевірка зв’язку клієнта з сервером.

```
        public·bool·serverStatus()¶
·····{¶
·····bool·status·=·false;¶
¶
·····status·=·TcpClient.Connected;¶
¶
·····return·status;¶
·····}¶
```

Модель асинхронного програмування Task (TAP) надає абстракцію поверх асинхронного коду. Ви пишете код як послідовність тверджень, як завжди. Ви можете прочитати цей код, як ніби кожне твердження

завершується до початку наступного. Компілятор виконує ряд перетворень, тому що деякі з цих операторів можуть почати роботу і повернути завдання, яка являє собою поточну роботу[16].

Це мета цього синтаксису: включити код, який читається як послідовність операторів, але виконується в набагато більш складному порядку, заснованому на розподілі зовнішніх ресурсів і по завершенні завдань. Це аналогічно тому, як люди дають інструкції для процесів, які включають асинхронні завдання. У цій статті, ви будете використовувати приклад інструкції для виготовлення сніданку, щоб подивитися, як asyncі awaitключевие слова полегшують причини про код, який включає в себе серію асинхронних інструкцій.

## Висновки до розділу 4

Четвертий розділ дипломного проекту полягав в безпосередній розробці ПЗ, представленню зображень головних форм, створенню класів логіки додатків та розробці методів роботи програми.

Для розробки ПЗ була використана асинхронна модель мови програмування C# на ключових словах async await за допомогою класа Task. Ця модель є найбільш новою на найбільш продвинутою в даній мові програмування.

Також було налаштовано зв’язок між андроід додатками та сервером розробленим под ОС Windows, що доказує досить розвинені та уніфіковані можливості платформи .Net Framework.

# ВИСНОВКИ

У кваліфікаційній роботі було розглянуто існуючі аналоги програмного забезпечення для служби таксі. Було виявлено, що інфомаційна система, впроваджена на ПАТ «АрселорМіттал Кривий Ріг», працює за застарілими технологіями та не відповідає сучасним стандартам програмування.

У роботі була обгрунтована актуальність даної теми та освітлений повний цикл виробництва подібного програмного забезпечення, починаючи від схематичного та алгоритмічного проектування. Завершаючи розробкою програмного інтерфейса та повною реалізацією всіх необхідних для роботи функцій.

У роботі використано технології асинхронного програмування на мові C#, що забезпечує краще використання системних ресурсів при аспектах роботи подібної служби. Також використовувалась Microsoft SQL від Постачальника .NET Framework для SQL Server. Ця база даних є локальною для серверної програми і розташована безпосередньо в каталозі програми.

Структура БД в проекті представлена в виді декількох таблиць. Перша таблиця представляє собою набір даних інформації про таксистів. Друга безпосередньо займається архівуванням даних про замовлення які вже виконані на момент запиту до таблиці. Третя таблиця є робочою, туди заносяться дані про активні замовлення, ті що на даний момент в ефірі або ті що виконуються таксистами.

Все використане допоміжне програмне забезпечення таке як сервер бази даних, або бібліотеки для роботи з нею, розповсюджуються за ліцензією про безкоштовне програмне забезпечення, що означає – проблем з ліцензуванням та покупкою якихось ключів не буде.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Г. Шилдт. C#: повне керівництво
2. Head First C#, Jennifer Greene, Andrew Stellman (український переклад: Вивчаємо C#, Д. Грін, Е. Стіллмен).
3. C# 5.0 and the .NET 4.6 Framework (7th Edition), Andrew Troelsen, Philip Japikse (український переклад попереднього видавництва: Мова програмування C# 5.0 й платформа .NET 4.5, Ендрю Троєлсен).
4. Pro WPF 4.5 in C#: Windows Presentation Foundation in .NET 4.5, Matthew MacDonald (український переклад: WPF: Windows Presentation Foundation в .NET 4.5 з прикладами на C# 5.0 для професіоналів, Метью Макдональд).
5. ДСТУ 19.701-90 Єдина система програмної документації. Схеми алгоритмів, програм, даних та систем. Умовні позначення та правила виконання URL: http://internet-law.com/dstus/dstu/28346.
6. Основні напрями досліджень, що базуються на семантичному аналізі текстів URL: http://apcp.apmath.kbu.ua/ua/staff/tuzov/onapr.html/
7. Asynchronous Server Socket Example URL: https://docs.microsoft.com/en-us/dotnet/framework/network-programming/asynchronous-server-socket-example
8. Asynchronous Server Socket Example URL: https://docs.microsoft.com/en-us/dotnet/framework/network-programming/asynchronous-server-socket-example
9. Методи побудови алгоритмів URL: http://choippo.cn.sch.in.ua/navchaljni_temi/metodi_pobudovi_algoritmiv/
10. Проектування програмного забезпечення при структурному підході URL:https://studfiles.net/preview/5484923/

11. sockets client+server with await/async c# 5.0 [Електроний ресурс] / CodeProject. - URL: https://www.codeproject.com/Questions/767211/sockets-clientplusserver-with-await-async-csharp.
12. Абрамян М. Е. Visual C# на прикладах / Абрамян М. Е. – Харків. : БХВ-Харків, 2008. – 496 с.
13. Агуров П. В. C#. Розробка компонентів в MS Visual Studio 2005/2008 / Агуров П. В. – Харків. : БХВ-Харків, 2008. – 480 с.
14. Зеленков Ю. Введення в бази даних / Зеленков Ю.– Харків. : Харків, 2003.
15. Кузнецов С. Д. Основи сучасних баз даних / КузнецовС. Д. – М. : Інтернет-Університет Інформаційних Технологій; БІНОМ. Лабораторія знань. – 2007. – 484 с.
16. Нейгел К. C# 2005 для професіоналів / Нейгел К., Ивьен Б., Глинн Дж., Уотсон К. – Діалектика, 2006. – 763 с.
17. Павловская Т. А. C#. Програмування на мові високого рівня : Підручник для вузів / Павловская Т. А. – Київ. : Київ, 2007. – 432 с.
18. Петцольд Ч. Програмування для Microsoft Windows на C# / Петцольд Ч.– М. : Діалектика, 2002. – 576 с.
19. Прайс Дж. Visual C# NET. Повне керівництво / Дж. Прайс, М. Гандэрлой. – КОРОНА-прінт, 2004. – 960 с.
20. Рихтер Дж. CLR via C#. Програмування на платформі Microsoft NET Framework 4.5 на мові C# /Рихтер Дж. – Київ, 2013. – 896 с.
21. Цехнер Марio. Програмування ігор під Android / Цехнер Марio. – Київ, 2012. – 688 с.
22. Платформа ADO. NET Entity Framework URL: http://msdn.microsoft.com/ru-ru/library/bb399572.aspx.
23. Ed Burnette Hello, Android 2e; Sourcebooks, Inc. – M. :, 2009. – 250 с.

24. Послідовність рішення задачі по розробці програми URL : https://works.doklad.ua/view/wZ6ySnsOZ2U.html
25. Комп'ютерні програми, їх місце в процесі обробки інформації за допомогою комп'ютераURL: https://studopedia.su/16_180939_metodi-rozrobki-program.html
26. Документація по Xamarin URL: https://docs.microsoft.com/en-us/xamarin/
27. Сервіс відгуків «Перший незалежний чайт відгуків України» URL: https://www.otzyvua.net/uk/onlayn-zakaz-taksi.html
28. Офіційний сайт Uber URL: https://www.uber.com/ua/uk/ride/
29. Офіційний сайт Uklon URL: https://uklon.com.ua/?_gl=1%2Aquyxc7%2A_up%2AMQ..&gclid=Cj0KCQiAj_CrBhD-ARIsAIiMxT-Tr3Y0C3v0F1rzlaN5tDc0CruF80dgNSG_3ObPMO3wTt_nB9hc3hUaAq7ZEALw_wcB
30. Офіційний сайт Bolt URL: https://bolt.eu/uk-ua/

# ДОДАТОК А

Код програмного забезпечення на мові C#
«Diplom»

**MainWindow.xaml¶**

```
<Window·x:Class="Diplom.MainWindow"¶
        ·
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"¶
        ·xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"¶
        ·xmlns:d="http://schemas.microsoft.com/expression/blend/2008"¶
        ·xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"¶
        ·xmlns:local="clr-namespace:Diplom·"¶
        ·mc:Ignorable="d"¶
        ·Title="MainWindow"·Height="450"·Width="800"·
Background="#FF5F5F5F">¶
    <Grid>¶
        ·<Grid.RowDefinitions>¶
            ·<RowDefinition·Height="434*"/>¶
            ·<RowDefinition·Height="125*"/>¶
        ·</Grid.RowDefinitions>¶
        ·<Grid.ColumnDefinitions>¶
            ·<ColumnDefinition·Width="291*"/>¶
            ·<ColumnDefinition·Width="175*"·/>¶
¶
        ·</Grid.ColumnDefinitions>¶
        ·<Button·x:Name="btTaxiMan"·Tag="Таксисты"·
HorizontalAlignment="Left"·Margin="24,48,0,0"·VerticalAlignment="Top"·
Width="36"·Height="36"·RenderTransformOrigin="0.182,-0.424"·>¶
            ·<Button.Background>¶
                ·<ImageBrush·ImageSource="Image\taxi.jpg"·/>¶
            ·</Button.Background>¶
        ·</Button>¶
        ·<Button·x:Name="taxiCar"·Tag="Зарегестрированые·автомобили"·
HorizontalAlignment="Left"·Margin="62,48,0,0"·VerticalAlignment="Top"·
Width="36"·RenderTransformOrigin="4.289,2.909"·Height="36"·>¶
            ·<Image·x:Name="btTaxiCar"·Source="Image\taxiCar.jpg"·/>¶
        ·</Button>¶
        ·<Button·x:Name="setting"·Tag="Настройки"·
HorizontalAlignment="Left"·Margin="246,48,0,0"·VerticalAlignment="Top"·
Width="36"·Height="36"·RenderTransformOrigin="0.737,0.636"·>¶
            ·<Image·Name="img5"·Source="Image\setting.jpg"·/>¶
        ·</Button>¶
        ·<Button·x:Name="btReportOrders"·HorizontalAlignment="Left"·
Margin="116,48,0,0"·VerticalAlignment="Top"·Width="36"·Height="36">¶
¶
```

```
            <Image Source="Image\setting.jpg" />
        </Button>
                    <TextBox HorizontalAlignment="Left" Height="23"
   Margin="83,52,0,0" Visibility="Hidden" TextWrapping="Wrap" Text="тут
                                                                    
чат" VerticalAlignment="Top" Width="120" Grid.Column="1"
Grid.Row="1" />
        <TextBlock HorizontalAlignment="Left" Margin="285,52,0,0"
Grid.Row="1" TextWrapping="Wrap" Visibility="Hidden" Text="тут не
придумал
                  " VerticalAlignment="Top" Height="16" Width="155" />
        <TabControl x:Name="TabControl" Margin="10,98,10,10">
            <TabItem x:Name="activeCallTab" Header="Активные">
                <Grid Background="#FFE5E5E5" Margin="0,-1,0,1">
                    <DataGrid x:Name="activeCallGrid"
RenderTransformOrigin="0.5,0.5" />
                </Grid>
            </TabItem>
            <TabItem x:Name="runninsCallTab" Header="Выполняющиеся">
                <Grid Background="#FFE5E5E5">
                    <DataGrid x:Name="runningCallGrid"
RenderTransformOrigin="0.5,0.5" />
                </Grid>
            </TabItem>
        </TabControl>
                                                                    

<Button x:Name="btReportSummary" HorizontalAlignment="Left"
Margin="152,48,0,0" VerticalAlignment="Top" Width="36" Height="36"
RenderTransformOrigin="1.579,0.318" >
            <Image Source="image/taxi.jpg" />
        </Button>
        <Label Content="Данные" HorizontalAlignment="Left"
VerticalAlignment="Top" Margin="24,17,0,0" Width="110" />
        <Label Content="Отчёты" HorizontalAlignment="Left"
VerticalAlignment="Top" Margin="116,17,0,0" Width="80"
RenderTransformOrigin="0.981,0.5" />
        <Label Content="Настройки" HorizontalAlignment="Left"
VerticalAlignment="Top" Margin="230,17,0,0" Width="78" />
        <RichTextBox x:Name="txtConsole" Grid.Column="1"
Grid.RowSpan="2" IsReadOnly="True" Background="#FF5F5F5F"
Foreground="White">
            <FlowDocument>
                <Paragraph>
                    <Run Text="" />
                </Paragraph>
            </FlowDocument>
        </RichTextBox>
```

```
····</Grid>¶
</Window>¶
¶
¶
¶
MainWindow.xaml.cs¶
¶
using·System;¶
using·System.Collections.Generic;¶
using·System.Linq;¶
using·System.Text;¶
using·System.Threading.Tasks;¶
using·System.Windows;¶
using·System.Windows.Controls;¶
using·System.Windows.Data;¶
using·System.Windows.Documents;¶
using·System.Windows.Input;¶
using·System.Windows.Media;¶
using·System.Windows.Media.Imaging;¶
using·System.Windows.Navigation;¶
using·System.Windows.Shapes;¶
¶
¶
namespace·Diplom___¶
{¶
····///·<summary>¶
····///·Логика·взаимодействия·для·MainWindow.xaml¶
····///·</summary>¶
····public·partial·class·MainWindow·:·Window¶
····{¶
········public·MainWindow()¶
········{¶
············InitializeComponent();¶
············SqlConnectionLogic·sqlLogic·=·new·SqlConnectionLogic();¶
············Server·serv·=·new·Server(this,·sqlLogic);¶
········}¶
········public·void·hookMessage(string·message)¶
········{¶
     txtConsole.AppendText(message);¶
}¶
····}¶
     ·private·void·btTaxiMan_Click(object·sender,·RoutedEventArgs·e)¶
········{¶
¶
········}¶
```

```
        private void taxiCar_Click(object sender, RoutedEventArgs e)
        {

        }
        private void btReportOrders_Click(object sender,
RoutedEventArgs e)
        {

        }
}

```

**Server.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.Net.Sockets;


namespace Diplom___
{
    class Server
    {
        TcpListener TcpListener;
        List<TcpClient> clientsList = new List<TcpClient>();
        MainWindow parentWin;
        SqlConnectionLogic sqlLogic;
        public Server(MainWindow parentWin, SqlConnectionLogic
sqlLogic)
        {
            TcpListener = new TcpListener(new
IPEndPoint(IPAddress.Parse("192.168.0.89"), 11000));
            this.parentWin = parentWin;
            this.sqlLogic = sqlLogic;
            Run();
        }
        private async void Run()
        {
```

```
            TcpListener.Start();
            parentWin.hookMessage("Сервер запущен" +
Environment.NewLine);
            await waitConnect();
        }
private async Task waitConnect()
        {
            var tcpClient = await TcpListener.AcceptTcpClientAsync();
            clientsList.Add(tcpClient);
            parentWin.hookMessage("Пользователь подключён" +
Environment.NewLine);

            await processClientTearOff(tcpClient);


            await waitConnect();
        }

        async Task processClientTearOff(TcpClient Cl)
        {
            using (var client = new Client(Cl, sqlLogic, parentWin))
                await client.ProcessAsync();
        }

    }
}
```

**Client.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.Net.Sockets;

namespace Diplom___
{
    class Client : IDisposable
    {
        NetworkStream netStream;

        SqlConnectionLogic sqlLogic;
```

```
        MainWindow parentWin;

        public Client(TcpClient client, SqlConnectionLogic sqlLogic,
MainWindow parentWin)
        {
            this.sqlLogic = sqlLogic;
            this.parentWin = parentWin;
            netStream = client.GetStream();
        }
        public void Dispose()
        {
            netStream.Dispose();
        }
        async Task<byte[]> ReadFromStreamAsync(int nbytes)
        {
            var buf = new byte[nbytes];
            var readpos = 0;
   
     readpos += await netStream.ReadAsync(buf, readpos, nbytes -
readpos);
            return buf;
        }
        public async Task ProcessAsync()
        {
            var actionBuffer = await ReadFromStreamAsync(16384);
            var action = Encoding.UTF8.GetString(actionBuffer);
            string[] strArr = action.Split(new string[] { @"\/" },
StringSplitOptions.RemoveEmptyEntries);
            string operationCode = strArr[0];
            string operationMessage = strArr[1];
            switch (operationCode)
            {
                case "1":
                    log_in(operationMessage);
                    break;
                case "2":
                    Registr(operationMessage);
                    break;

```

```
            case "3":
                    New_order(operationMessage);
                    break;

                case "4":
                    Accept_order(operationMessage)
                    break;

            }

            await ProcessAsync();
        }

        public async Task sendMessage()
        {

        }

        public void log_in(string message)
        {

        }

        public void Registr(string message)
        {
            sqlLogic.sqlInsertInto(message);
        }

        public void New_order(string message)
        {

        }

        public void Accept_order(string message)
        {
```

**ДОДАТОК Б**

Код програмного забезпечення на мові C#

«Taxi_Client_Application»

**MainPage.xaml¶**

```
¶
<?xml·version="1.0"·encoding="utf-8"·?>¶
<MasterDetailPage·xmlns="http://xamarin.com/schemas/2014/forms"¶
·····xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"¶
·····xmlns:d="http://xamarin.com/schemas/2014/forms/design"¶
·····xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"¶
·····mc:Ignorable="d"¶
·····x:Class="Taxi_Client_Application.MainPage"¶
·····Title="Авторизация">¶
¶
····<MasterDetailPage.Master>¶
·····<ContentPage·Title="Меню">¶
·····<ContentPage.Content>¶
·····<StackLayout>¶
·····<Button·x:Name="btOrders"·Text="Заказы"·
Clicked="btOrders_Clicked"/>¶
·····<Button·x:Name="btSettings"·Text="Текущий·заказ"·
Clicked="btSettings_Clicked"/>¶
·····<Button·x:Name="btLogOut"·Text="Выйти·из·аккаунта"·
Clicked="btLogOut_Clicked"/>¶
·····<Button·x:Name="Exit"·Text="Выход"·
Clicked="Exit_Clicked"/>¶
·····</StackLayout>¶
·····</ContentPage.Content>¶
·····</ContentPage>¶
····</MasterDetailPage.Master>¶
¶
····<MasterDetailPage.Detail>¶
·····<ContentPage·Title="Авторизация">¶
·····<ContentPage.Content>¶
·····<Label·Text="Hello·world!"·VerticalOptions="Center"·
HorizontalOptions="Center"/>¶
·····</ContentPage.Content>¶
·····</ContentPage>¶
····</MasterDetailPage.Detail>¶
····¶
¶
</MasterDetailPage>¶
¶
```

**MainPage.xaml.cs**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using System.Net;
using System.Net.Sockets;
using Xamarin.Android;
using Xamarin.Forms.PlatformConfiguration;

namespace Taxi_Client_Application
{
    // Learn more about making custom code visible in the
Xamarin.Forms previewer
    // by visiting https://aka.ms/xamarinforms-previewer

    [DesignTimeVisible(false)]
    public partial class MainPage : MasterDetailPage
    {
        public MainPage()
        {
            InitializeComponent();
            Detail = new NavigationPage(new Orders());
        }
        private void btOrders_Clicked(object sender, EventArgs e)
        {
            Detail = new NavigationPage(new Orders());
            IsPresented = false;
        }
        private void btSettings_Clicked(object sender, EventArgs e)
        {
            Detail = new NavigationPage(new Settings());
            IsPresented = false;
        }
        private void btLogOut_Clicked(object sender, EventArgs e)
        {
            App.Current.MainPage = new NavigationPage( new Login());
            IsPresented = false;
        }
```

```
        private void Exit_Clicked(object sender, EventArgs e)
        {
            Application.Current.Quit();
        }
    }
}

```

**Registration.xaml**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             mc:Ignorable="d"
             x:Class="Taxi_Client_Application.Registration"
             Title="Регистрация">
    <ContentPage.Content>
        <StackLayout>
            <Label Text="Номер телефона" />
            <Entry x:Name="txtPhoneNum" />
            <Label Text="Пароль" />
            <Entry x:Name="txtPassword" IsPassword="True"/>
            <Label Text="Повторите пароль" />
            <Entry x:Name="txtRepeatPassword" IsPassword="True"/>
            <Label Text="Марка Автомобиля" />
<Entry x:Name="txtMarkCar" />
            <Label Text="Цвет" />
            <Entry x:Name="txtCarColor" />
            <Label Text="Гос номер авто" />
            <Entry x:Name="txtCarNum" />
            <Label Text="Номер техпаспорта" />
            <Entry x:Name="txtTechNum" />
            <Button x:Name="btRegistr" Text="Зарегистрироватся"
Clicked="btRegistr_Clicked"/>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>

```

**Registration.xaml.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace Taxi_Client_Application
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Registration : ContentPage
    {
        Client client;
        public Registration(Client client)
        {
            InitializeComponent();

            this.client = client;
        }

        private void btRegistr_Clicked(object sender, EventArgs e)
        {
            string message = @"INSERT INTO [Car_Drivers] ([Phone_Num], [Password], [Car_Num], [Car_Color] , [Tech_Car_Num])" +
                " VALUES (" + txtPhoneNum.Text + "," + txtPassword.Text + "," + txtCarNum.Text +
                "," + txtCarColor.Text + "," + txtTechNum.Text + ")" + @"\/";

            client.sendMessage("2", message);
        }
    }
}
```