

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
(повне найменування факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти (освітній ступінь))

на тему РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ ОРЕНДИ
ТРАНСПОРТНИХ ЗАСОБІВ

Виконав: студент 4 курсу, групи КНТ-520
спеціальності _____

123 «Комп'ютерна інженерія»

(код і найменування спеціальності)

Освітня програма (спеціалізація) _____

«Комп'ютерна інженерія»

ВИНИЧЕНКО Д. В.

(ПРИЗВИЩЕ та ініціали)

Керівник ХОХЛОВ М.М.

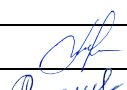
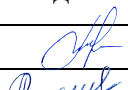
(ПРИЗВИЩЕ та ініціали)

Рецензент КОЗИНА Г.Л.

(ПРИЗВИЩЕ та ініціали)

Сторінки сайту системи оренди транспортних засобів.

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ХОХЛОВ М. М., ст., викл		
нормоконтроль	ПОЛЬСЬКА О.В., ст. викл.		

7. Дата видачі завдання 22.02.2024 р.

КАЛЕНДАРНИЙ ПЛАН

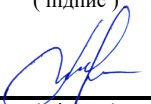
№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	1 Примітка
1	Аналіз наявних технологій для реалізації системи	08.03.2024 р.	
2	Розробка структури системи	12.03.2024 р.	
3	Розробка алгоритму роботи системи	18.03.2024 р.	
4	Розробка бази даних системи	01.04.2024 р.	
5	Розробка клієнтської частини системи	10.04.2024 р.	
6	Розробка серверної частини системи	01.05.2024 р.	
7	Оформлення пояснювальної записки	13.05.2024 р.	
8	Оформлення графічного матеріалу	18.05.2024 р.	

Студент (ка)


 (підпис)
Данііл ВИНІЧЕНКО

(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)


 (підпис)
Микола ХОХЛОВ

(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 60 с., 4 табл., 32 рис., 15 лістингів., 15 джерел.

АРХІТЕКТУРА, JAVASCRIPT, СКБД, MYSQL, HEIDISQL, ER-МОДЕЛЬ,
БАЗА ДАНИХ, КОМП'ЮТЕРНА СИСТЕМА, СЕРВЕР, КЛІЄНТ

Об'єкт розробки – комп'ютерна система оренди транспорту.

Мета проєкту – розробка комп'ютерної системи оренди транспорту для автоматизації роботи бізнесу та обліку транспорту та оренди.

В ході розробки було проаналізовано технічне завдання. Проведено проєктування комп'ютерної системи, в результаті якого було сформовано базу даних (БД), визначено алгоритм роботи сайту, та побудовано його макет. Для розробки бази даних було обрано систему керування базами даних (СКБД) MySQL та допоміжне програмне забезпечення HeidiSQL. Клієнтську частину розроблено на мові JavaScript. Серверну частину розроблено на мові JavaScript з використанням технології Node.js та фреймворку Express.js.

В результаті розроблена комп'ютерна система має можливість створення замовлення на оренду. База даних містить дані про автомобілі та дані про заявки на оренду, додатково заявки містять дані про клієнтів. Клієнтська частина має зручний та зрозумілий інтерфейс з своїм дизайном.

ЗМІСТ

Вступ.....	7
1 Визначення технологій розробки	9
1.1 Аналіз аналогічних комп'ютерних систем	9
1.2 Вибір архітектури системи.....	11
1.2.1 Клієнт-серверна архітектура.....	11
1.2.2 Монолітна архітектура.....	13
1.2.3 Serverless архітектура.....	14
1.2.4 Мікросервісна архітектура	15
1.3 Вибір технологій для клієнтської та серверної частини	16
1.3.1 Мова програмування PHP	19
1.3.2 Мова програмування Python	20
1.3.3 Мова програмування Ruby	21
1.3.4 Мова програмування Java.....	22
1.3.5 Мова програмування JavaScript.....	23
1.4 Вибір системи керування базами даних	24
1.4.1 Система керування базами даних MySQL.....	24
1.4.2 Система керування базами даних Oracle Database	25
1.4.3 Система керування базами даних PostgreSQL	26
1.4.4 Система керування базами даних Microsoft Access.....	26
1.4.5 Допоміжне програмне забезпечення HeidiSQL	27
1.5 Висновки з аналізу технологій.....	27
2 Моделювання предметної області	29
2.1 Загальні вимоги до комп'ютерної системи	29
2.2 Функціональні можливості користувачів	30
2.3 Проєктування бази даних системи	32
2.3.1 Концептуальне проєктування бази даних.....	32
2.3.2 Логічне проєктування бази даних	34

2.4 Діаграми послідовностей.....	37
3 Програмна розробка системи.....	39
3.1 Розробка клієнтської частини	39
3.1.1 Розробка графічного інтерфейсу користувача	39
3.1.2 Розробка функціональної частини клієнтського інтерфейсу	42
3.2 Розробка серверної частини	45
4 Тестування розробленої комп'ютерної системи.....	52
4.1 Перевірка адаптивності	52
4.2 Перевірка функціоналу відправки замовлень	54
Висновки	55
Перелік джерел посилання	56
Додаток А Основний програмний код сторінки з формою	58

ВСТУП

Інтернет насичений великою кількістю вебсайтів, які містять різноманітну інформацію та можуть задовольнити потреби різних клієнтів. Велика кількість вебсистем кожен день оброблює багато запитів на придбання товарів та послуг. Власники бізнесів зацікавлені у збільшенні обсягів продажу, впливу та рекламуванню своїх продуктів. Задовольнити ці потреби допомагає створення своїх власних сайтів.

Одним з розповсюджених бізнесів є оренда транспортних засобів, це сфера діяльності, в якій компанії здають в оренду транспорт іншим компаніям або приватним особам за певну плату. Оренда транспортних засобів може бути вигідною для компаній та їх клієнтів, які потребують транспорт. Оренда може бути на декілька годин, днів, тижнів або на довший термін, залежно від потреб клієнта.

Для успішного ведення бізнесу оренди транспортних засобів важливо мати ефективну систему бронювання та обліку автомобілів, яке може забезпечити власна вебсистема бізнесу. Бізнес оренди транспортних засобів може бути досить прибутковим, особливо якщо підприємство вдало розміщується в місці з великим попитом на транспортні послуги, але власний вебсайт дозволяє не бути залежним від місця та надавати послуги по всій країні.

У сучасному світі необхідно мати сайт для ведення успішного бізнесу. Придбання офісу біля відвідуваного місця більше не є головним у залученні клієнтів, створений вебсайт дозволяє вийти бізнесу на глобальний ринок. Вебсистема може працювати цілодобово, що дозволить клієнтам користуватись послугами бізнесу у будь-який зручний момент часу. Важливим є також економія коштів на рекламі та комунікації з клієнтами. Вебсистеми можуть обробляти, зберігати та сортувати запити клієнтів для подальшого їх зручного використання.

Велика та постійно зростаюча індустрія розробки вебсистем задовольняє зростаючий попит бізнесів. Найважливішими у цій індустрії є такі спеціалісти:

- Full Stack Developer, це розробник який може створити повністю функціональну та готову вебсистему з нуля, він має знання для розробки клієнтської та серверної частини сайту;

- Back-end Developer, розробник який створює серверну частину вебсайту, реалізує внутрішню логіку роботи системи, забезпечуючи його взаємодію з базами даних та зовнішніми сервісами.

- Front-end Developer, це розробник інтерфейсу взаємодії між клієнтом та вебсистемою, від нього залежить якість передачі інформації від сервера до клієнта.

Виходячи з основних професій індустрії зрозуміло, що складовими вебсистеми є серверна, яка відповідає за обробку запитів, та клієнтська частина, яка відповідає за відправку та отримання цих запитів.

Розробка клієнтської частини включає обрання мови програмування, фреймворку, створення дизайну майбутнього сайту та обрання інших необхідних технологій для працездатності інтерфейсу користувача.

Проектування серверної частини включає обрання серверу, універсальної мови програмування та системи управління базою даних.

Отже зрозуміло як поверхнево влаштована вебсистема, хто її розроблює, кому вона потрібна та навіщо. В дипломній роботі буде створено комп'ютерну систему для оренди транспортних засобів включно з її серверною та клієнтською частинами.

1 ВИЗНАЧЕННЯ ТЕХНОЛОГІЙ РОЗРОБКИ

Головним та невід’ємним етапом створення власної вебсистеми є Визначення технологій розробки, треба зрозуміти з чим ми маємо справу, яку інформацію та функціонал треба розмістити на вебсайті. Необхідно обрати архітектуру системи, технології для клієнтської та серверної частини, базу даних. Для цього потрібно аналізувати наявні технології. Вибір правильних технологій дозволить зручно працювати з вебсистемою та легко її масштабувати у подальшому.

1.1 Аналіз аналогічних комп’ютерних систем

Аналіз аналогічних комп’ютерних систем включає в себе дослідження, порівняння та оцінку різних систем з метою виявлення їхніх сильних та слабких сторін, визначення доцільності розробки своєї комп’ютерної системи. Для аналізу аналогічних систем було обрано RENTAL.UA, Їду.сам та Payless.

Перша, RENTAL.UA, система надає увесь необхідний функціонал, має зручний інтерфейс, який зображено на рисунку 1.1.

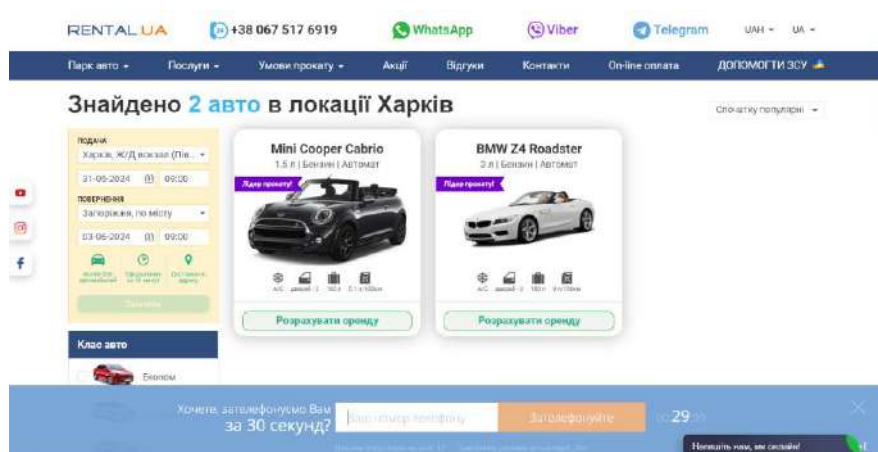


Рисунок 1.1 – Інтерфейс системи RENTAL.UA

Ця система має перевантажений інтерфейс непотрібними спливаючими вікнами та повідомленнями. Також немає можливості прорахунку ціни оренди на обраний період часу, система оплати сумнівна.

Вебсистема оренди їду.сам, яка зображена на рисунку 1.2, має увесь необхідний функціонал та розроблена для користувачів, які здають авто та орендують.

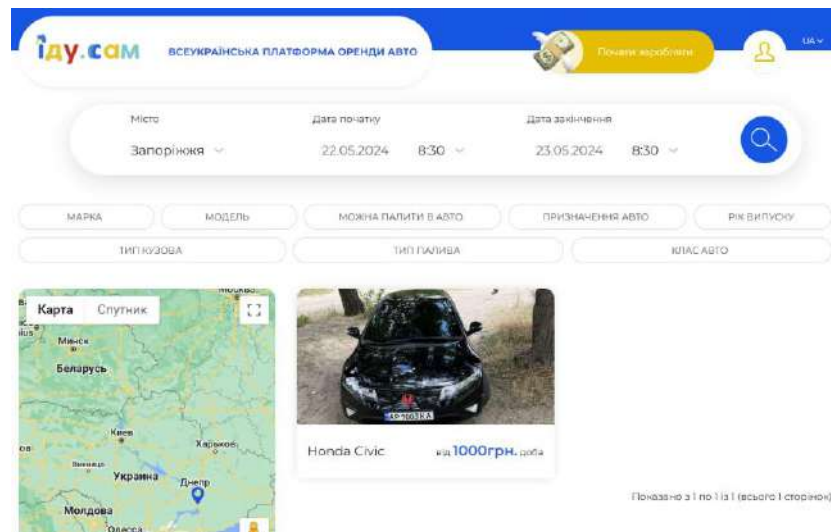


Рисунок 1.2 – Інтерфейс вебсистеми їду.сам

Ця система має не дуже привабливий інтерфейс, через злиття білих елементів на білому фоні, та не доречну інтеграцію Google Maps, яка не має функціональний взаємодій з вебсистемою.

Система Payless має зручний інтерфейс, довге оформлення замовлення та немає реєстрації. Інтерфейс вебсистеми Payless зображено на рисунку 1.3.

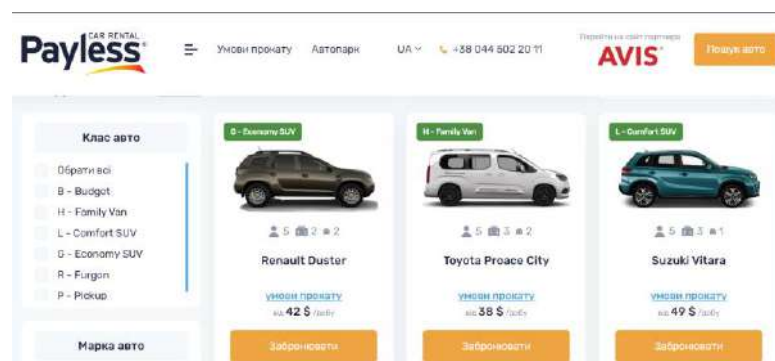


Рисунок 1.3 – Інтерфейс системи Payless

Payless включає в себе багато додаткових опцій у оформленні замовлення та інтуїтивно-зрозумілий інтерфейс.

Аналіз існуючих систем дозволив знайти сильні сторони досліджуваних систем, які можна запозичити, та слабкі місця, які необхідно уникнути. Дослідження подібних систем зазначило, що готових та безкоштовних рішень не існує, тобто у відкритому доступі немає жодної комп'ютерної системи оренди транспорту. Розробка особистої системи доцільна через відсутність можливості придбання готової системи та наявність індивідуальних вимог проекту.

1.2 Вибір архітектури системи

Архітектура системи являє собою розроблену структуру з визначенням взаємодії компонентів інтерфейсу з внутрішніми процесами. Основною задачею архітектури системи є створення структури з полегшенням розробки та підтримки. Від обраної архітектури залежить надійність, швидкість роботи внутрішніх процесів системи, складність рефакторингу коду та подальше масштабування системи.

Через зростаючі потреби ринку створюються нові, вдосконалені, архітектури систем. Під час проектування вебсистеми актуальні такі архітектури:

- клієнт-серверна архітектура;
- монолітна архітектура;
- serverless архітектура;
- мікросервісна архітектура.

1.2.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура побудована на основі клієнта та сервера. Клієнт являє собою пристрій користувача, який підправляє запити до сервера та отримує інформацію або виконує якісь дії. Сервером є пристрій, який має увесь необхідний функціонал для обробки клієнтських запитів, може зберігати дані у

базах даних, та може виконувати увесь необхідний функціонал. Модель даної архітектури полягає у відправці запитів від клієнта до сервера, де сервер оброблює дані та надсилає їх назад до клієнта. Клієнт-серверна архітектура зображена на рисунку 1.4. Сервер може оброблювати одночасно декілька запитів клієнтів, запити можуть вставати в чергу та виконуватись послідовно. У таких чергах запитам можуть надаватись пріоритети, для більш зручної їх обробки[1].

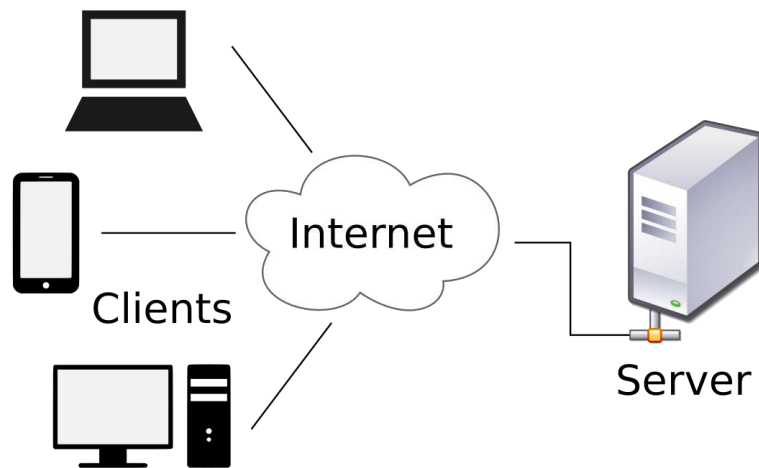


Рисунок 1.4 – Клієнт-серверна архітектура

Перевагами такої архітектури є:

- зручність розробки та підтримки. Розподілення на серверну та клієнтську частину дозволяє розроблювати їх окремо;
- продуктивність. Клієнт-серверна архітектура дозволяє розподілити обчислення між клієнтом та сервером;
- підтримка різних платформ. Клієнтська частина може бути реалізована на будь-якому пристрої та взаємодіяти з серверною частиною без обмежень системи користувача;
- масштабованість. Модульність даного архітектурного рішення дозволяє легко додавати новий функціонал до системи.

Недоліками клієнт-серверної архітектури є залежність від мережі та наявність єдиної точки відмови у вигляді серверу.

При створенні системи з використанням клієнт-серверної архітектури потрібно обрати її рівень:

- однорівнева архітектура. Така архітектура реалізує увесь функціонал та збереження даних на одному пристрої. Це дозволяє працювати в автономному режимі та безпечно зберігати усі дані на пристрої користувача;

- дворівнева архітектура. Функціонал розподіляється між клієнтом та сервером. Усі дані зберігається на сервері. Клієнтська частина отримує усі необхідні дані від серверної частини та спілкується з нею за допомогою повідомлень;

- трирівнева архітектура. Інтерфейс у клієнтській частині, функціонал та обробка даних відбувається на сервері, а усі дані зберігається на іншому сервері. Такий підхід дозволяє збільшити безпеку системи, але потребує збільшення фінансових витрат;

- n-рівнева архітектура. Схожа на трирівневу архітектуру, але різниця полягає в тому, що N-рівнева архітектура має більшу кількість серверів для обробки даних та їх зберігання. Така архітектура дозволить отримати підвищення продуктивності та стабільності.

1.2.2 Монолітна архітектура

При монолітній архітектурі весь додаток розроблюється як єдина та нероздільна система. У такої системи весь функціонал розроблений як одне єдине програмне забезпечення та виконується на одній операційній системі. Такий підхід застарілий, але досі популярний. Здебільшого підходить для програм, де важлива цілісність, простота розгортання та управління. Здебільшого монолітною архітектурою користуються у невеликих командах розробників та проєктів, де простота та швидкість розробки переважають над потребами у високій масштабованості та гнучкості. Зображення монолітної архітектури наведено на рисунку 1.5[3].

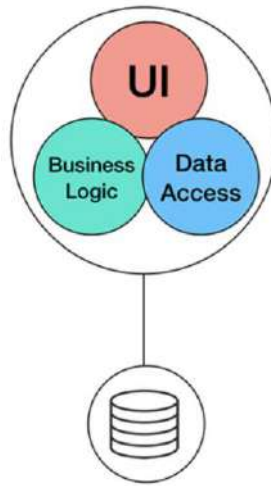


Рисунок 1.5 – Монолітна архітектура

Перевагами підходу монолітної архітектури є:

- швидкість розробки. Оскільки усі його компоненти зосереджені в одному місці, він є простішим у розробці;
- легкість тестування. Усі компоненти взаємозв'язані один з іншим, це дозволяє швидко знайти та розв'язати проблему.

Великим недоліком такої архітектури є складна масштабованість, оскільки компоненти залежать один від одного, то впровадження нового функціоналу може бути складним. Складність підтримки такої архітектури залежить від її розміру, тому вона не підходить для великих систем.

1.2.3 Serverless архітектура

Serverless архітектура являє собою модель хмарних обчислень, та передбачає використання платформи провайдера як виконання написаного коду системи. Serverless архітектура зображена на рисунку 1.6. Перевагами Serverless архітектури є:

- простота розробки. Відокремлений код Serverless архітектури, зазвичай, дуже простий;
- продуктивність. Усі обчислення виконуються на стороні провайдера;
- підтримка різних платформ. Будь-які пристрої можуть бути підключені, немає залежності від операційної системи користувача;

– низькі витрати на інфраструктуру системи. Адміністрування серверу перекладається на провайдера.

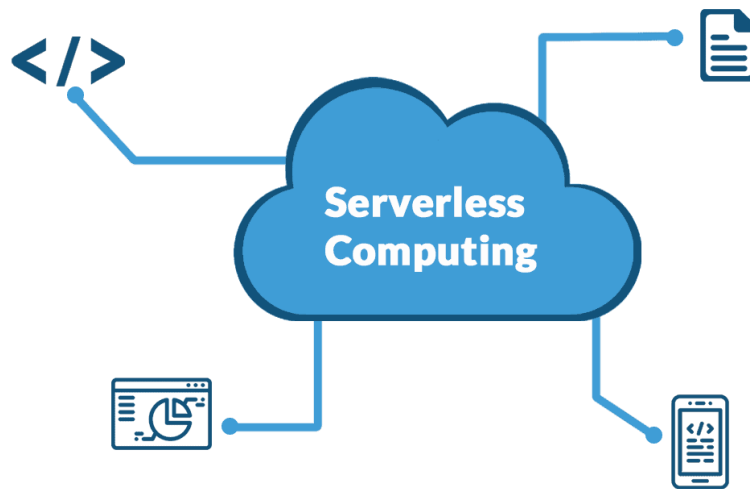


Рисунок 1.6 – Serverless архітектура

Основними недоліками є обмежене масштабування та обмежений цикл життя, додаток запускається у виділеному контейнері та вимикається через деякий час.

1.2.4 Мікросервісна архітектура

Мікросервісна архітектура являє собою систему з набору слабко-пов'язаних сервісів. Базується на основі розподілу на різні модулі та спілкування їх за допомогою повідомлень. Мікросервісна архітектура наведена на рисунку 1.4. Кожен сервіс, цієї архітектури, відповідає за свою функцію.

З плюсів даної архітектури можна виділити:

- модульність. Кожен модуль відповідає за свій функціонал, модулі можуть бути написані на різних мовах програмування;
- масштабованість. Модульність дозволяє легко додавати новий функціонал;
- надійність. Завдяки розбиттю функціоналу на слабко-пов'язані сервіси, досягається висока стійкість до відмов.

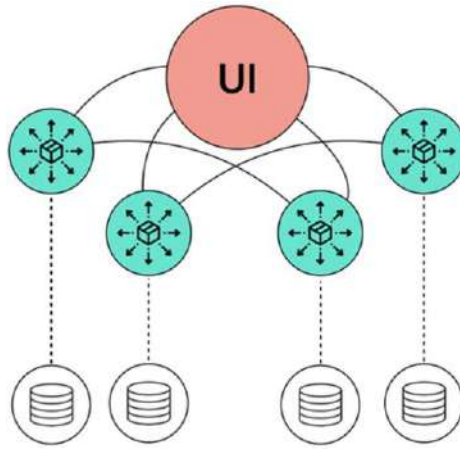


Рисунок 1.7 – Мікросервісна архітектура

Серед мінусів мікросервісної архітектури потрібно виділити її низьку швидкість та складність розробки. Якщо не повністю продумати функціонал додатку, то внесення змін у модулі системи буде складніше ніж внесення змін в інших архітектурах. Крім цього, мікросервісна архітектура може призвести до збільшення накладних витрат на комунікацію між сервісами, що може знижувати продуктивність системи. Це особливо відчутно у випадках, коли сервіси часто взаємодіють між собою.

1.3 Вибір технологій для клієнтської та серверної частини

Обрання технологій для клієнтської та серверної частини є важливим етапом від якого залежить майбутня реалізація проєкту, його функціональні можливості та продуктивність. Клієнтська частина повинна бути реалізована як вебсторінка, тобто для її завантаження потрібен будь-який пристрій з підтримкою браузерів. Серверна частина реалізується на окремому орендованому, або придбаному, серверному пристрої.

Існує три необхідні компоненти клієнтської частини системи, це мова розмітки документів, мова для опису та стилізації документів та мова для реалізації функціональної частини сайту.

Мова розмітки документів необхідна для створення структури сторінки: заголовки, абзаци, списки тощо. Цей код відповідає за те як саме браузер буде відображати контент на сторінці. Популярними мовами розмітки документів є HTML 1.1, HTML 5 та XHTML.

HTML являє собою стандартну та першу мову розмітки, яка використалась для створення та відображення вебсторінок. HTML визначає структуру та вміст вебсторінки за допомогою різних елементів та тегів. Усі версії HTML до її п'ятої версії, HTML5, називаються HTML, це пов'язано з великою різницею між ними. Останньою версією HTML є 4.01, яка вийшла у 1999 році. HTML 4.01 включає в себе підтримку мов скриптів та таблиць стилів[5].

Наступною версією HTML 4.1, яка вийшла у 2000 році, є XHTML. XHTML був розроблений для виправлення недоліків HTML шляхом введення певних функцій XML. Він не набув популярності та не сумісний з усіма браузерами.

HTML5 є новою версією HTML, яка включає в себе багато нових функцій та можливостей. Вона має покращену підтримку мультимедіа, вебзасоби, покращені форми та багато іншого.

Стилізація та опис зовнішнього вигляду необхідні для створення привабливих та читабельних вебсторінок. Завдяки мов каскадних таблиць стилів браузер розуміє, як саме відображати елементи мови розмітки документа. Під час розробки вебсистеми популярні таблиці стилів CSS, SASS та SCSS.

CSS (Cascading Style Sheets) — мова для опису та стилізації зовнішнього вигляду документа. Завдяки CSS-коду браузер розуміє, як саме відображати елементи HTML коду. Наприклад, CSS задає кольори та параметри шрифтів, визначає, як розташовуватимуться різні блоки сайту. Ще він дозволяє відтворювати документ у різних видах: для друку, виведення на екран або для читання скринрідером.

SASS (Syntactically Awesome Style Sheets) — SCSS насправді не додає нічого нового в мову CSS. Це просто новий синтаксис, який здебільшого скорочує час написання стилів. Sass — це перша специфікація Sassy CSS, її розробка розпочалася у 2006 році. Пізніше розробили альтернативний синтаксис, який

дехто вважає кращою версією мови, а також змінилося початкове розширення .sass на нове .scss.

SCSS (Syntactically Cascading Style Sheets) — є одним з найпопулярніших препроцесорів CSS. SCSS використовує синтаксис, сумісний зі звичайним CSS, з можливістю використання додаткових функцій, таких як змінні, вкладення, міксіни, умови, цикли та багато інших.

SASS та SCSS є CSS-препроцесорами — це надбудови над CSS, спеціальні скрипти, які розширюють можливості CSS та спрощують процес створення стилів, які потім вбудовуються у файли CSS. Препроцесори дозволяють використовувати змінні, умовні оператори, цикли та інші зручні функції, яких немає у звичайному CSS. На рисунку 1.5 зображено різницю синтаксисів SASS, SCSS та CSS.

SASS	SCSS	CSS
<pre> nav ul margin: 0 padding: 2px list-style: square li display: inline-block a display: block padding: 5px 10px text-decoration: underline dotted green </pre>	<pre> nav { ul { margin: 0; padding: 2px; list-style: square; } li { display: inline-block; } a { display: block; padding: 5px 10px; text-decoration: underline dotted green; } } </pre>	<pre> nav ul { margin: 0; padding: 2px; list-style: square; } nav li { display: inline-block; } nav a { display: block; padding: 5px 10px; text-decoration: underline dotted green; } </pre>

Рисунок 1.8 – Синтаксис SASS, SCSS та CSS

Для розробки клієнтської та серверної частини системи необхідно обрати мову програмування для реалізації інтерфейсу користувача та функцій серверу відповідно. Від обраної мови залежить швидкість розробки, важкість її підтримки та функціональні можливості розроблюваної системи. Популярними універсальними мовами програмування є PHP, Python, Ruby, Java та JavaScript. Розробка вебсистем полегшується за допомогою використання фреймворків для мов програмування. Фреймворк — це інструмент, який має на меті спростити розробку та надати готові рішення для розв’язання поставлених задач.

1.3.1 Мова програмування PHP

PHP являє собою популярну універсальну мову програмування, створену спеціально для веброзробок, його код може бути впроваджуватись прямо у HTML. Ця мова програмування з відкритими вихідними кодами, над розвитком якого працюють програмісти-ентузіасти з усього світу. Він має простий синтаксис, частково схожий синтаксис мов Java та C++. Сферою використання PHP мови є написання скриптів або повноцінних вебзастосунків на серверній стороні, створення сценаріїв для командного рядка та написання графічних інтерфейсів. PHP відрізняється від JavaScript тим, що PHP-скрипти виконуються на сервері та генерують HTML, який надсилається клієнту[6]. Перевагами використання PHP мови є:

- простий синтаксис. PHP швидко освоюється. PHP-код легко читається незалежно від способу використання. Підходить для написання невеликих скриптів або створення потужних додатків з використанням об'єктноорієнтованого підходу до реалізації програми;
- підтримка різних платформ. PHP сумісний з усіма популярними платформами. Написані на ньому програми успішно працюють на різному серверному програмному забезпеченні;
- масштабованість. Зростання апаратних ресурсів дозволяє досягти максимальної продуктивності програм, написаних на PHP. Підтримує розподіл вебпрограм на кілька серверів. Простий синтаксис дозволяє швидко орієнтуватися у коді та додавати нові функції без проблем;
- вбудованість у HTML-документи. До HTML-сторінки можна легко додати контент, написаний на мові PHP. PHP блоки додаються HTML-тегів не порушуючи структуру документа;
- постійний розвиток мови. Спільнота розробників постійно працює над впровадженням додаткового функціонала, що розширює можливості мови, спрощенням синтаксису та покращенням захисту від можливих атак.

Одним з недоліків PHP є можливість зміни синтаксису. Сайти, які написані на старій версії PHP дуже складно переписати на нову версію, набагато легше

написати цей сайт повністю з нуля. Таким чином майбутні модифікації або масштабування вебсистеми іншими розробниками може бути проблематичним.

Найпопулярнішими фреймворками для PHP є Laravel, Yii та CodeIgniter. Laravel – це безкоштовний фреймворк для швидкої розробки великих вебдодатків. Yii простий та високопродуктивний універсальний фреймворк з влаштованим генератором коду Gii, який дозволяє прискорити процес розробки. CodeIgniter зарекомендував себе як надійний та компактний фреймворк, який використовується для побудови повноцінних вебсистем та додатків.

1.3.2 Мова програмування Python

Python є високорівневою мовою програмування, друга за популярністю у світі мова програмування. Областями використання є розробка вебзастосунків, програмного забезпечення, машинного навчання. Простота Python дозволяє швидко її вивчити. Найпростіший синтаксис Python дозволяє швидко та легко писати будь-які додатки та адаптувати високорівневу логіку. Ця мова підтримує велику кількість платформ та підходить для розробки будь-яких систем[7]. Перевагами використання Python є:

- дуже простий та логічний синтаксис. Python відомий своєю простотою та читабельністю. Синтаксис мови дуже схожий на англійську мову, завдяки такому синтаксису вихідний код зрозуміє будь-який розробник;
- масштабованість. Python дає можливість легко розширювати складні застосунки. Простий синтаксис дозволяє швидко орієнтуватися у коді та додавати нові функції без проблем;
- швидкість розробки. Розробка на мові Python випадків проходить швидше, ніж на інших мовах програмування, це зумовлене простотою синтаксису та великою кількістю документації;
- мультиплатформність. Інтерпретована мова програмування Python дозволяє запускати програми на будь-якій платформі. Адаптувати код під окремі платформи не потрібно.

Python є високорівневою мовою програмування, який написаний на мові C++, тому він повільніше працює ніж Java. Це особливо помітно в обчислювальних задачах або додатках, де швидкодія має велике значення.

Серед фреймворків для Python виділяються Django, Pyramid та Flask. Django дозволяє створити простий та чистий код, а також володіє підтримкою усіх популярних систем управління базами даних. Фреймворк Pyramid зарекомендував себе як зручний інструмент для створення додатків будь-якого розміру та краще підходить для роботи з API. Flask має влаштований сервер розробки та підходить для швидкого створення вебсистем.

1.3.3 Мова програмування Ruby

Високорівнева мова програмування Ruby, як і Python, написана на мові C++ та відома простотою синтаксису. Вона є одною з найпростіших мов програмування. Мова зосереджується на зручності та мінімізації трудових витрат[8].

Ваговими перевагами Ruby є:

- швидкість розробки. Висока швидкість розробки досягається завдяки простому синтаксису, а також великому вибору готових рішень для різних типів завдань. Ruby має велику колекцію бібліотек готових рішень під назвою RubyGems;
- підтримка різних платформ. Універсальна мова програмування Ruby підтримується багатьма платформами.

Головними недоліками Ruby виступає повільна робота, порівняно з іншими мовами програмування, та виконання коду потребує більше обчислювальної потужності. Мала підтримка зі сторони великих корпорацій призвела до малої кількості документації, у порівнянні з іншими відомими мовами.

Серед фреймворків для Ruby цікавим є тільки Ruby on Rails, який має велику кількість функцій та зарекомендував себе як зручний фреймворк для створення вебзастосунків.

1.3.4 Мова програмування Java

Універсальна мова програмування Java являє собою інструмент, який дозволяє запускати код відразу для усіх платформ без необхідності перекомпіляції. Жорстка типізація, автоматичне керування пам'яттю та об'єктноорієнтованість дозволило стати Java одною з найпоширеніших мов програмування. Java володіє зрозумілим синтаксисом та великою кількістю бібліотек[9].

Перевагами використання мови Java є:

- швидкість роботи. Завдяки своїй віртуальній машині, Java має хорошу продуктивність та підходить для великих обчислювальних задач;
- багатопоточність. Підтримка багатопоточності дозволяє досягнути високої продуктивності та опрацьовувати велику кількість даних. Одночасне виконання декількох задач облегшує ефективне використання ресурсів;
- масштабованість. Java підтримує розробку невеликих та великих програмних систем. Завдяки наявності багатопоточності, дозволяє масштабувати систему зі зростанням навантаження;
- мультиплатформність. Код, написаний на Java, може запустити будь-яка платформа, яка підтримує віртуальну машину Java;
- велика бібліотека. Наявність стандартної бібліотеки, яка містить велику кількість класів та методів для розв'язання різних задач, дозволяє розробникам швидше розроблювати системи.

Основним недоліком мови Java є велике, порівняно з іншими мовами програмування, споживання пам'яті. Таке споживання пов'язано з автоматичною роботою з пам'яттю.

Існує велика кількість різноманітних фреймворків для Java, для веброзробки популярні фреймворки Spring Framework, PrimeFaces та Blade. Spring Framework є одним з найпопулярніших фреймворків, який дозволяє застосовувати залежності та підходить для створення корпоративних додатків. PrimeFaces характеризується як легкий фреймворк для розробки користувацьких інтерфейсів. Фреймворк Blade володіє підтримкою модульності та великою кількістю вебкомпонентів Java.

1.3.5 Мова програмування JavaScript

JavaScript являє собою прототипну мову з динамічною типізацією. Мова розроблена для створення програмних сценаріїв, які виконують певні задані дії. Завдяки постійному розвитку, мова JavaScript може вирішувати велику кількість різних задач. В основному її використовують для створення вебзастосунків, завдяки чому на сайтах додається велика кількість інтерактивності. JavaScript створена з подібним синтаксисом як у Java, але структурно являє собою іншу мову[10].

Перевагами використання мови JavaScript є:

- мультиплатформність. JavaScript підтримує велику кількість платформ, які підтримують браузер;
- вбудована підтримка Document Object Model. Це дозволяє динамічно взаємодіяти з об'єктами вебсторінки, оброблення подій та внесення змін;
- велика кількість бібліотек. Завдяки постійному розвитку, у JavaScript є велика кількість готових рішень та фреймворків для полегшення розробки систем;
- сумісність. Легке інтегрування у технології HTML і CSS, а також підтримка асинхронності та динамічної типізації.

Головними недоліками JavaScript є менша продуктивність у порівнянні з іншими мовами програмування та у великих проєктах динамічна типізація може створити труднощі у підтримці.

JavaScript має велику кількість фреймворків, найпопулярнішими з яких є React.js, Vue.js, Angular та Express.js. React.js дозволяє створювати компоненти інтерфейсу для мобільних та десктопних пристроїв. Його використовують популярні компанії такі як Facebook, Instagram, WhatsApp. Vue.js використовується для створення користувацького інтерфейсу та поширений за його просту інтеграцію в JavaScript проєкти. Фреймворк Angular постійно розвивається та має зручну інтеграцію з HTML та CSS. Express – це вебфреймворк для програм Node.js, що надає набір функцій для мобільних та вебдодатків. Програма Node.js дозволяє розробникам створювати сервери, вебдодатки, інструменти командного рядка та сценарії.

1.4 Вибір системи керування базами даних

Для ефективного керування базою даних використовують системи керування базами даних (СКБД). СКБД – це програмна система, яку використовують для зберігання, отримання та виконання запитів до даних. СКБД являє собою інтерфейс між користувачем і БД, дозволяючи користувачам створювати, читати, оновлювати та видаляти дані. СКБД дозволяє оптимізувати організацію БД за допомогою нормалізації, яка розбиває велику таблицю на менші таблиці, якщо будь-який з її атрибутів має надлишкові значення. СКБД класифікують за моделями даних, існують: реляційні, розподілені, ієрархічні, об'єктно-орієнтовані та мережеві.

СКБД також забезпечують різні рівні доступу, дозволяючи адміністраторам встановлювати права доступу для різних користувачів і груп. Це допомагає захистити дані від несанкціонованого доступу та маніпуляцій. Крім того, сучасні СКБД підтримують резервне копіювання та відновлення даних, що забезпечує захист даних від різних проблем.

Серед великої кількості зручних СКБД найпоширенішими є MySQL, Oracle Database, PostgreSQL та Microsoft Access.

1.4.1 Система керування базами даних MySQL

MySQL являє собою найпопулярнішу СКБД, після Oracle Database, з відкритим кодом та великою кількістю функцій, розроблених користувачами. MySQL має підтримку усіх основних платформ, включаючи Red Hat, Fedora, Ubuntu, Debian, Solaris, Microsoft Windows та Apple MacOS. Інтерфейс цієї СКБД зображено на рисунку 1.9. Реалізована підтримка великої кількості мов програмування (C, C++, C#, PHP, Java, Ruby, Python та Perl). Існує підтримка декількох процесорів та багатопоточності. З мінусів цієї СКБД особливо виділяється невелика надійність та мала кількість оновлень[12].

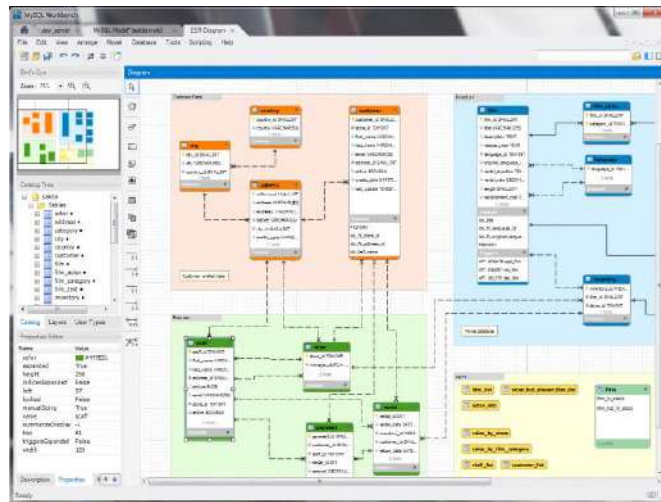


Рисунок 1.9 – Інтерфейс СКБД MySQL

1.4.2 Система керування базами даних Oracle Database

Oracle Database є найбільш вживаною СКБД та має дві свої основні версії, такі як Oracle Database 19c та Oracle Database 21c. Oracle Database 19c являє собою надійну та стабільну версію. Oracle Database 21c містить у собі велику кількість нововведень, але програє у надійності та стабільності своїй старшій версії. Oracle Database підтримується персональними комп'ютерами та мобільними пристроями. Безкоштовна версія підтримує один процесор та 1гб оперативної пам'яті, тоді як платна версія дозволяє використовувати до 11гб оперативної пам'яті. Велика надійність досягається ізольованими транзакціями[13]. Графічний користувацький інтерфейс СКБД Oracle Database наведено на рисунку 1.10.

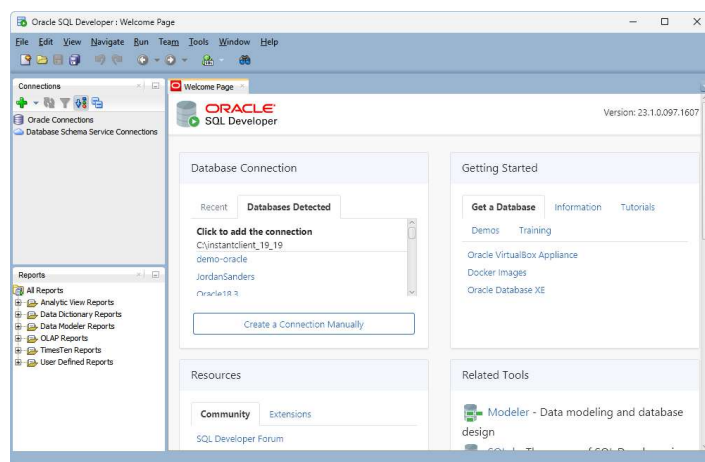


Рисунок 1.10 – Інтерфейс СКБД Oracle Database

1.4.3 Система керування базами даних PostgreSQL

PostgreSQL умовно-безкоштовна СКБД, отримати її можливо безкоштовно, але за модифікацію та підтримку потрібно платити. Ця СКБД є об'єктно-реляційною, може працювати з БД будь-якого розміру та великою кількістю запитів. PostgreSQL має підтримку C-сумісних модулів та XML, JSON і NoSQL-баз. Данна СКБД може працювати тільки на таких платформах: Linux, MacOS та Windows. Надійність PostgreSQL досягається складними та надійними механізмами транзакцій та реплікацій. На рисунку 1.11 зображено інтерфейс PostgreSQL.

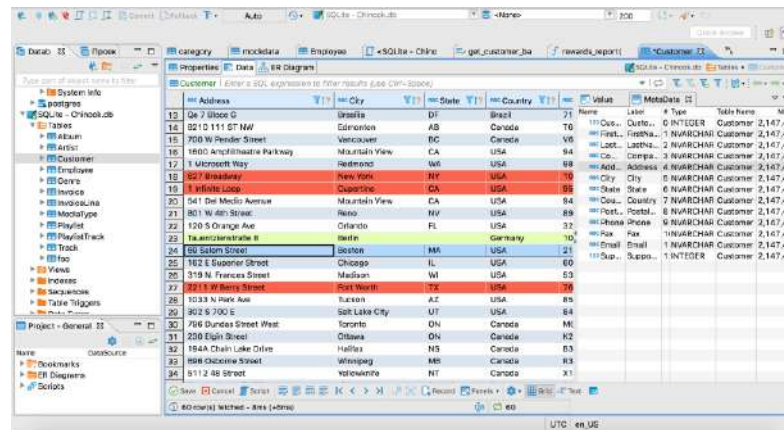


Рисунок 1.11 – Інтерфейс СКБД PostgreSQL

1.4.4 Система керування базами даних Microsoft Access

Microsoft Access являє собою найвідомішу СКБД, яка поєднує реляційну систему баз даних Jet від Microsoft із засобами розробки програмного забезпечення та графічним інтерфейсом користувача. Основною та єдиною платформою цієї СКБД є Windows. MS Access дозволяє писати спеціальний код за допомогою мови програмування Visual Basic for Applications, це дозволяє створювати власні функції та автоматизувати складні завдання. Ця СКБД має зручну інтеграцію з програмами пакету Microsoft Office, такими як Excel і Word. Мінусами СКБД Microsoft Access можна виділити тільки використання однієї платформи Windows[14].

1.4.5 Допоміжне програмне забезпечення HeidiSQL

Додатково до СКБД існує безкоштовне програмне забезпечення HeidiSQL, яке просте у використанні та дозволяє переглядати та редагувати дані та структури з комп'ютерів, на яких працює одна із систем керування базами даних MariaDB, MySQL, Microsoft SQL, PostgreSQL та SQLite[15]. Інтерфейс HeidiSQL зображено на рисунку 1.12.

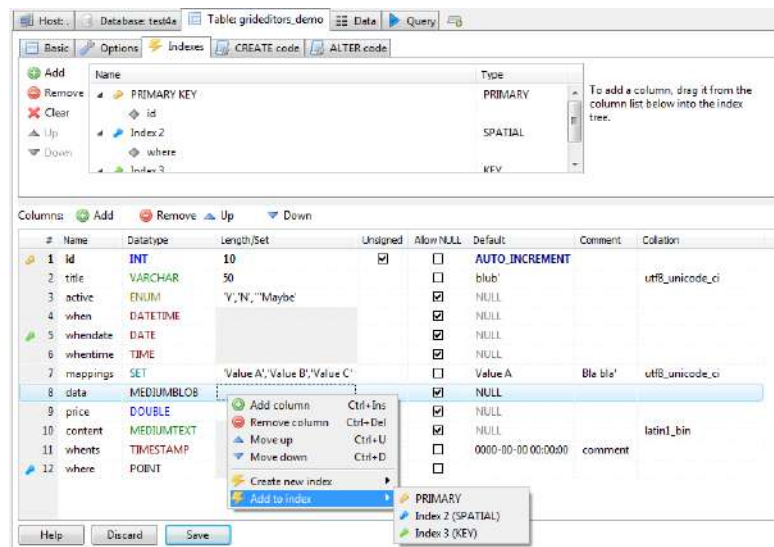


Рисунок 1.12 – Інтерфейс HeidiSQL

HeidiSQL має зручний інтерфейс, який дозволяє виконувати велику кількість різноманітних операцій з СКБД, та можливість імпортування та експортування БД. З мінусів можна виділити доступність тільки на платформі Windows.

1.5 Висновки з аналізу технологій

Переглянуто актуальні архітектури систем, аналізовано принципи їх застосування та переваги з недоліками. Для системи оренди транспортних засобів доцільно використати клієнт-серверну архітектуру. Така архітектура дозволить

швидше розробити систему, мати продуктивну, безпечну та надійну систему. У подальшому, масштабування такої системи не буде складним. Підтримка різних платформ дозволить розробити клієнтську частину під пристрої з різними операційними системами та розміром екрану. Розроблювана система для бізнесу оренди транспортних засобів повинна не сильно навантажувати клієнтський пристрій та бути доступним через мережу, тому клієнт-серверна архітектура ідеально підходить до розроблюваного проєкту. Розроблювана система буде мати однорівневу архітектуру. Система для оренди транспортних засобів не потребує великої продуктивності, тому створення окремих серверів тільки збільшить витрати.

Аналізом мов розмітки є висновок, що всі три мови є лише варіаціями одного і того ж стандарту HTML, але включають різні синтаксичні правила і функції. XHTML і HTML4 вже застаріли та HTML5, через його широке використання та розширену функціональність, є вдалим вибором для розробки вебсторінок. Мовою текстової розмітки обрано HTML5.

За стилізацію та опис зовнішнього вигляду вебсторінок обрано каскадну таблицю стилів CSS. Використання препроцесорів дозволяє пришвидшити розробку, але проєктована система не буде мати багато вебсторінок, тому використання препроцесорів буде зайвим.

Згідно з вимогами проєктованої системи, для створення клієнтської та серверної частини системи обрано мову програмування JavaScript. Використання одної мови програмування для серверної та клієнтської частини прискорить та полегшить розробку системи. Для серверної частини системи обрано програму Node.js та фреймворк Express.js, що дозволить запустити код не в середовищі браузера та полегшить роботу з даними.

Серед зазначених СКБД було обрано MySQL та клієнт доступу до СКБД HeidiSQL. Такий вибір надасть зручний інтерфейс. Це дозволить легко створювати таблиці, форми, запити та звіти, не вимагаючи великих технічних знань.

2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

Важливим етапом розробки будь-якої комп'ютерної системи є моделювання предметної області. Воно включає складення загальних вимог, функціональних можливостей користувачів, проєктування бази даних, опис таблиць бази даних та діаграм послідовностей. Даний етап розробки системи дозволить чітко визначити структуру і функціональні можливості системи та допоможе уникнути проблем при розробці системи.

2.1 Загальні вимоги до комп'ютерної системи

Завданням розробляємої вебсистеми є її рекламування, надання необхідної інформації та обробка замовлень її послуг. Користувачами системи є люди, які мають необхідність у отриманні тимчасового транспортного засобу.

Загальними вимогами до комп'ютерної системи оренди транспортних засобів є:

- зручний та інтуїтивно-зрозумілий інтерфейс сторінок з динамічним вмістом, сторінки мають бути виконані у одному зрозумілому оформленні;
- мати інформаційну сторінку, сторінку відправки замовлення та сторінку особистого кабінету користувача;
- проста та швидка система авторизації за допомогою акаунту Google. Завданням авторизації на сайті є прив'язка відправлених замовлень до окремого акаунту та відображення поточних замовлень;
- можливість створення та відправки замовлення. Створення замовлення повинно включати обрання автомобіля, дати початку оренди, кількість днів оренди та номер телефону для зв'язку;

- надання поточної інформації про стан замовлення, прив'язаних до окремого акаунту Google користувача;
- зберігання у базі даних усіх замовлень та необхідних для роботи даних.

2.2 Функціональні можливості користувачів

Функціональні можливості користувачів вебсистеми включають в себе набір функцій, які дозволяють взаємодіяти з системою та отримувати необхідну користувачам інформацію. Розроблювана система має декілька ролей, для яких необхідно визначити їх окремі функціональні можливості.

Розроблювана комп'ютерна система має такі типи користувачів:

- гість;
- клієнт;
- адміністратор.

Гостем є неавторизований користувач, який має найменші функціональні можливості, які зображено на рисунку 2.1.

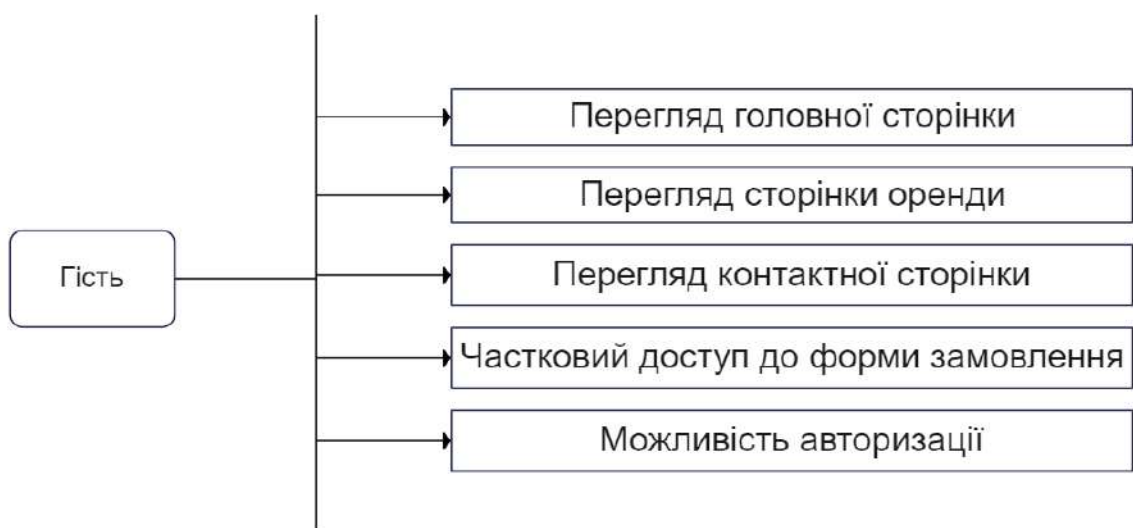


Рисунок 2.1 – Функціонал користувача з роллю «Гість»

Клієнт являє собою авторизованого користувача, який має функціональні можливості для створення та перегляду своїх замовлень, які наведено на рисунку 2.2.



Рисунок 2.2 – Функціонал користувача з роллю «Гість»

Адміністратором є користувач, який має можливість прийняття і редагування заявок та доступ до усієї бази даних. Функціональні можливості адміністратора наведено на рисунку 2.3.

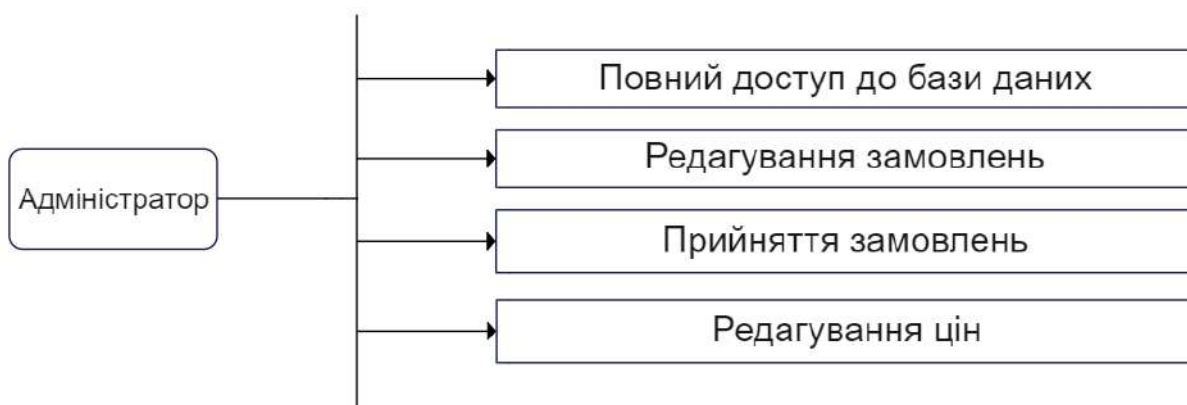


Рисунок 2.3 – Функціонал користувача з роллю «Адміністратор»

2.3 Проєктування бази даних системи

Проєктування бази даних складається з двох етапів, концептуального та логічного. Концептуальне проєктування включає визначення сутностей, схеми бази даних, зв'язків між сутностями та проєктування ER-моделі бази даних. Логічне проєктування включає перетворення концептуальної моделі в логічну модель з урахуванням особливостей обраної СКБД. Процес проєктування бази даних є ітераційним та може вимагати повернення до попередніх етапів для вдосконалення моделі. Він забезпечує створення ефективної, масштабованої та зручної в управлінні бази даних.

2.3.1 Концептуальне проєктування бази даних

Виходячи з потреб системи, можна виділити наступні сутності: «car», «body», «fuel», «rent». Схема бази даних має вигляд:

- «car» (car_id, car_name, fuel_id, class, body_id, number_of_seats, car_quantity, car_price, car_img);
- «fuel» (fuel_id, fuel_name);
- «body» (body_id, body_name);
- «rent» (rent_id, day_of_rent, car_id, client_name, client_email, client_number, number_days_rent, car_return, paid).

Наступним етапом складання концептуальної моделі бази даних є визначення зв'язків між сутностями. Правильне встановлення зв'язків забезпечує цілісність даних та адекватне відображення взаємодій між об'єктами системи. Можна визначити наступні взаємозв'язки між сутностями:

- «car» → «fuel» (1:Б) – Кожне авто споживає свій окремий вид палива, тип палива може повторюватись (рис. 2.4);

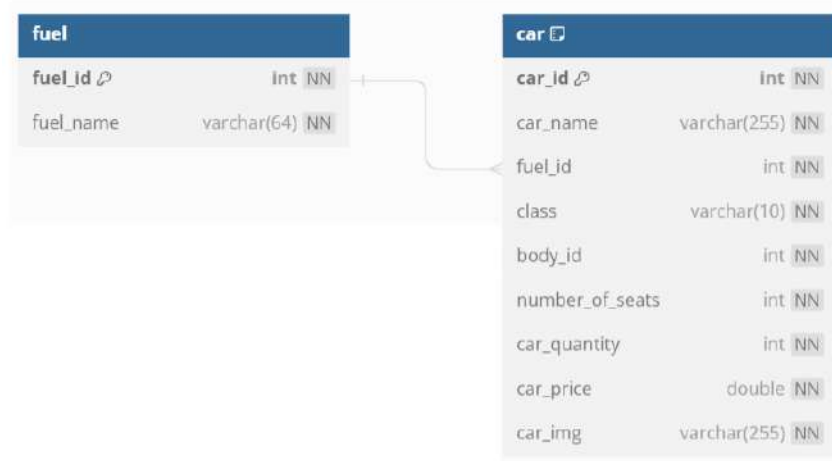


Рисунок 2.4 – Відношення «car» → «fuel» (1:Б)

- «car» → «body» (1:Б) – Транспорт розділяють за типами кузова, марка може повторюватись (рис. 2.5);

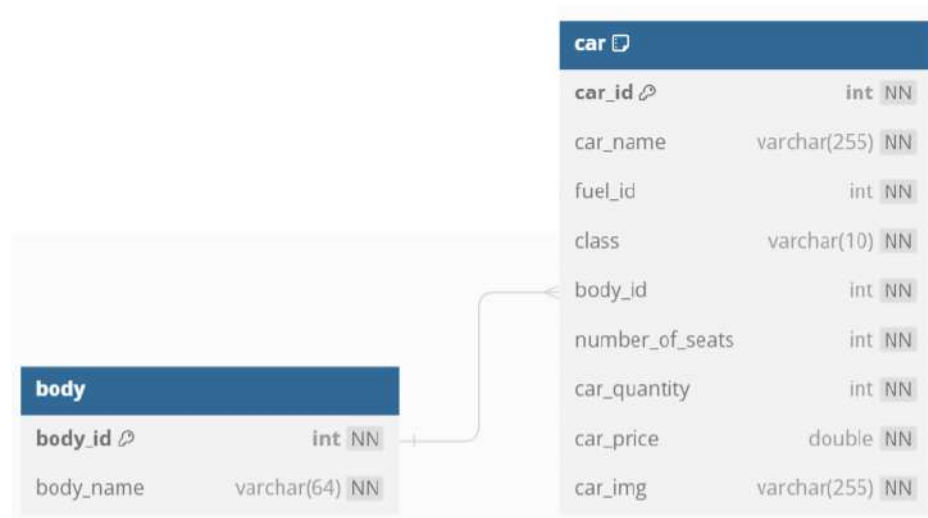


Рисунок 2.5 – Відношення «car» → «body» (1:Б)

- «car» → «rent» (1:Б) – Оренда автомобілів, зазвичай, має велику кількість однакових автомобілів, через це різні замовлення можуть містити в собі однакові ідентифікаційні номери цих автомобілів, або клієнти можуть замовити один й той-же автомобіль (рис. 2.6);

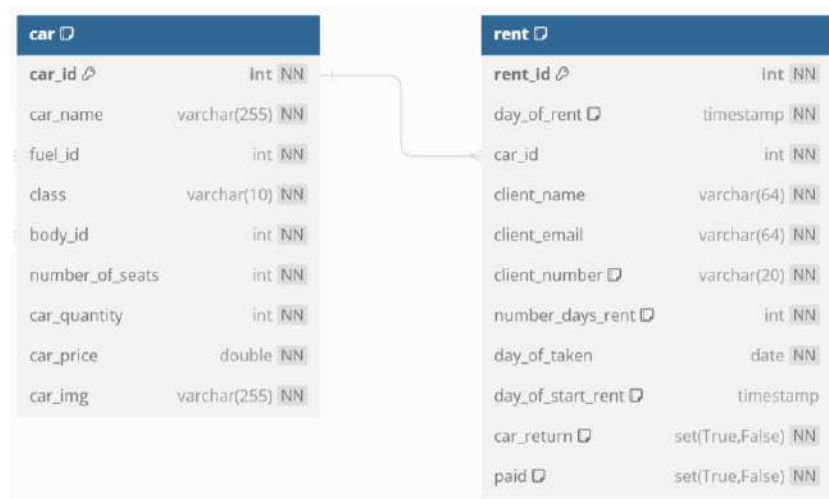


Рисунок 2.6 – Відношення «car» → «rent» (1:Б)

Проаналізувавши всі зв'язки між відношеннями бази даних, що проектується, отримуємо ER-модель, яку наведено на рисунку 2.7.

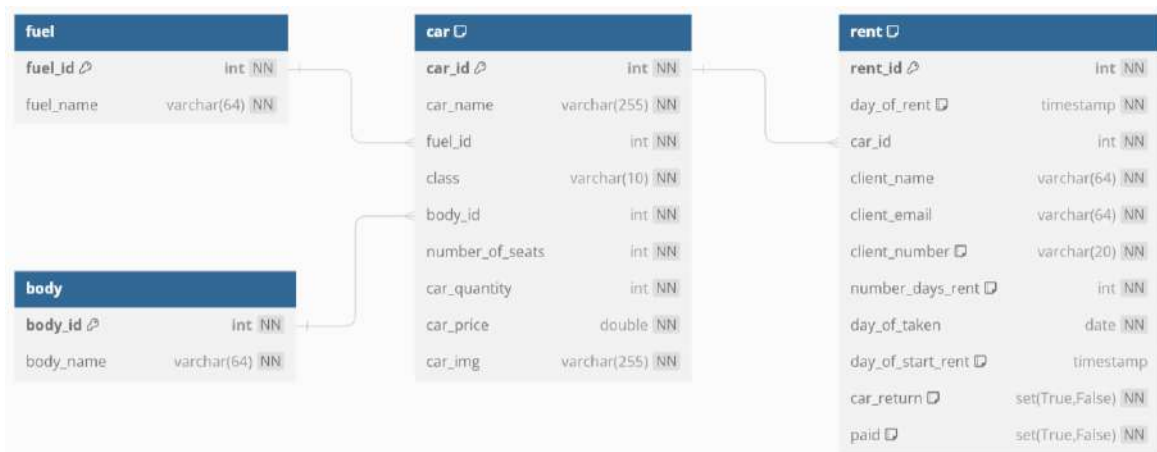


Рисунок 2.7 – ER-модель бази даних системи оренду транспорту

2.3.2 Логічне проєктування бази даних

В процесі проєктування логічної моделі бази даних визначається кількість і склад таблиць для зберігання вихідної інформації. Кожна таблиця представляється назвою, переліком атрибутів і первинним ключем. Визначаються зв'язку між таблицями, а також обмеження цілісності даних. Отримана логічна модель перевіряється на відповідність правилам нормалізації.

База даних системи оренди транспорту складається з чотирьох таблиць:

- “car” (табл 2.1);
- “fuel” (табл 2.2);
- “body” (табл 2.3);
- “rent” (табл 2.4);

Таблиця 2.1 – Склад відношення «car»

Атрибут	Тип даних	Допустимі значення	Значення за замовчанням	Обов'яз -ковість	Примітки
car_id	INT	Натуральні числа	AUTO_INCREMENT	Так	Первинний ключ
car_name	VARCHAR	Латиниця та кирилиця	-	Так	
fuel_id	INT	Натуральні числа	-	Так	Зовнішній ключ
class	VARCHAR	Латиниця та кирилиця	-	Так	
body_id	INT	Натуральні числа	-	Так	Зовнішній ключ
number_of_seats	INT	Натуральні числа	-	Так	
car_quantity	INT	Натуральні числа	-	Так	
car_price	DOUBLE	Дійсні числа	-	Так	
car_img	VARCHAR	Латиниця та кирилиця	-	Так	

Таблиця 2.2 – Склад відношення «fuel»

Атрибут	Тип даних	Допустимі значення	Значення за замовчанням	Обов'яз -ковість	Примітки
fuel_id	INT	Натуральні числа	AUTO_INCREMENT	Так	Первинний ключ
fuel_name	VARCHAR	Латиниця та кирилиця	-	Так	

Таблиця 2.3 – Склад відношення «body»

Атрибут	Тип даних	Допустимі значення	Значення за замовчанням	Обов'яз -ковість	Примітки
body_id	INT	Натуральні числа	AUTO_INCREMENT	Так	Первинний ключ
body_name	VARCHAR	Латиниця та кирилиця	-	Так	

Таблиця 2.4 – Склад відношення «rent»

Атрибут	Тип даних	Допустимі значення	Значення за замовчанням	Обов'яз-ковість	Примітки
rent_id	INT	Натуральні числа	-	Так	Первинний ключ
day_of_rent	TIMESTAMP	Повний формат дати	-	Так	
car_id	INT	Натуральні числа	-	Так	Зовнішній ключ
client_name	VARCHAR	Латиниця та кирилиця	-	Так	
client_email	VARCHAR	Латиниця та кирилиця	-	Так	
client_number	VARCHAR	Латиниця та кирилиця	-	Так	
number_days_rent	INT	Натуральні числа	-	Так	
day_of_taken	DATE	Короткий формат дати	-	Так	
day_of_start_rent	TIMESTAMP	Повний формат дати	-	Ні	
car_return	SET	Логічний тип	-	Так	
paid	SET	Логічний тип	-	Так	

Створено таблиці «car», «fuel», «body», «rent» за допомогою програмного забезпечення HeidiSQL в СКБД MySQL. Отриману логічну модель бази даних перевірено на відповідність правилам третьої нормальної форми, яка включає відповідність правилам другої та першої нормальної форми, а саме:

- всі неключові атрибути залежать тільки від первинного ключа;
- всі неключові атрибути повністю залежать від первинного ключа;
- всі стовпці таблиці містять лише атомарні значення;
- кожний стовпець містить значення одного типу;
- кожна таблиця має первинний ключ.

2.4 Діаграми послідовностей

Діаграми послідовностей використовуються для моделювання поведінки системи через взаємодію між об'єктами з плином часу. Вони відображають взаємодію об'єктів між собою за допомогою обміну повідомленнями для виконання певного функціоналу.

Створено діаграму послідовності для життєвого циклу функціоналу створення замовлення. Розроблену діаграму зображено на рисунку 2.8

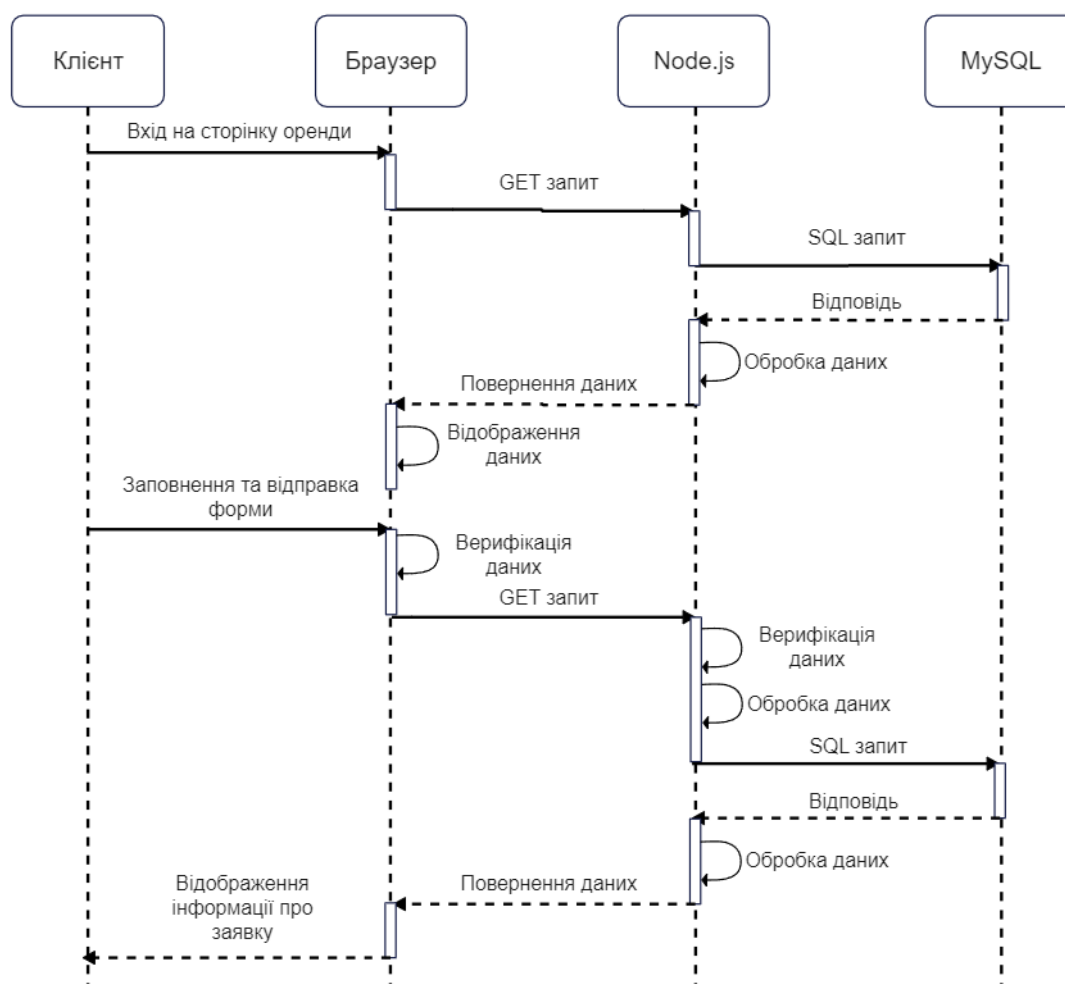


Рисунок 2.8 – Діаграма послідовностей створення замовлення

Після відкриття клієнтом сторінки оренди, клієнтська частина системи відправляє GET запит до серверної частини, яка відправляє SQL запит до бази

даних на отримання необхідної інформації, та отримує відповідь. Далі клієнтська частина системи формує з отриманих даних код сторінки. Після заповнення клієнтом форми та натискання кнопки її відправлення, клієнтська частина системи верифікує дані та, якщо вони відповідають вимогам, відправляє їх до серверної частини GET запитом. Серверна частина верифікує дані, оброблює їх та відправляє SQL запит до бази даних для запису заявки. Після отримання відповіді про успішний запис, серверна частина оброблює дані та відсилає їх до клієнтської частини, де формується попап сторінки, який відображає усю необхідну користувачу інформацію про відправлене замовлення.

Другою діаграмою, яка зображена на рисунку 2.9, є діаграма послідовності для життєвого циклу функціоналу відображення статусів клієнтських замовлень.

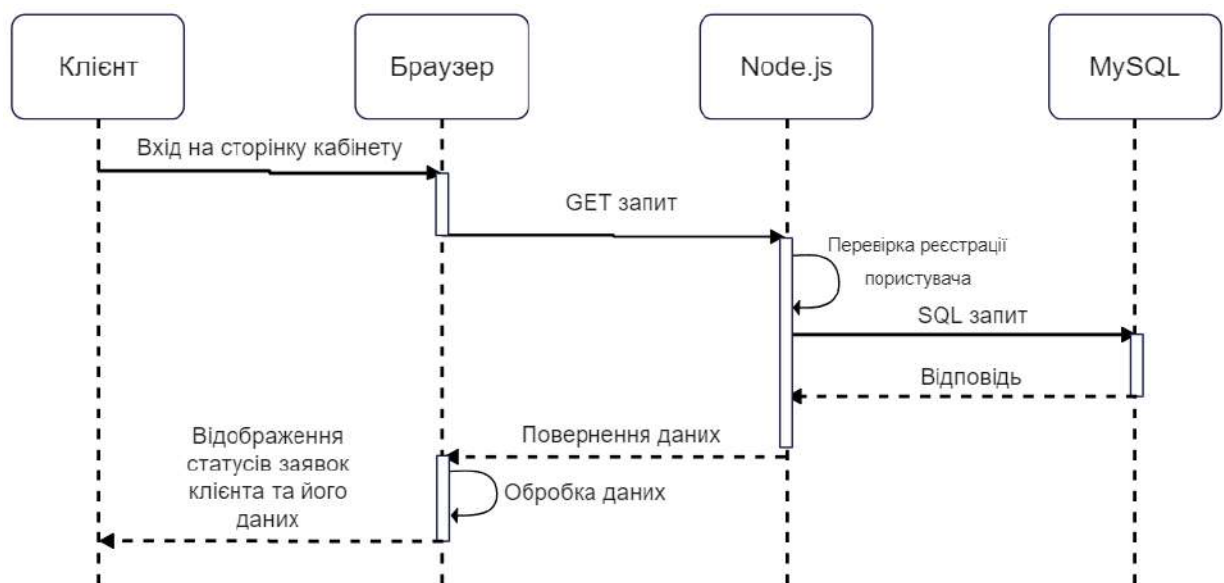


Рисунок 2.9 – Діаграма послідовностей відображення інформації на сторінці особистого кабінету клієнта

Після переходу клієнтом до сторінки особистого кабінету формується запит від клієнтської частини до серверної на отримання необхідних даних. Серверна частина перевіряє чи авторизований клієнт та формує SQL запит на отримання усіх даних про замовлення цього клієнта. Після отримання відповіді сервером, він відправляє необхідні дані до клієнтської частини, де ці дані оброблюються та формується зміст сторінки.

3 ПРОГРАМНА РОЗРОБКА СИСТЕМИ

3.1 Розробка клієнтської частини

У багатьох випадках розробку вебсистеми розпочинають з клієнтської частини. Спочатку створюється макет, який відображає структуру та дизайн сайту. Після цього на основі макету створюється сам сайт. Клієнтська частина визначає, як саме будуть представлені дані та які функції буде використовувати сервер. Іншими словами, від того, що є на клієнтській частині, залежить, які функції потрібно створити у серверній частині системи. Таким чином, клієнтська частина задає вимоги до серверної частини, визначаючи, які запити та обробка даних потрібна для повноцінної роботи системи.

3.1.1 Розробка графічного інтерфейсу користувача

Розробка інтерфейсу користувача починається з макету вебсайту, який повинен відповідати загальноприйнятим стандартам, привертати увагу та бути зручним для користувачів. Розроблений дизайн інтерфейсу системи, за допомогою онлайн-сервісу для розробки інтерфейсів та дизайнів – “Figma”, зображено на рисунку 3.1.

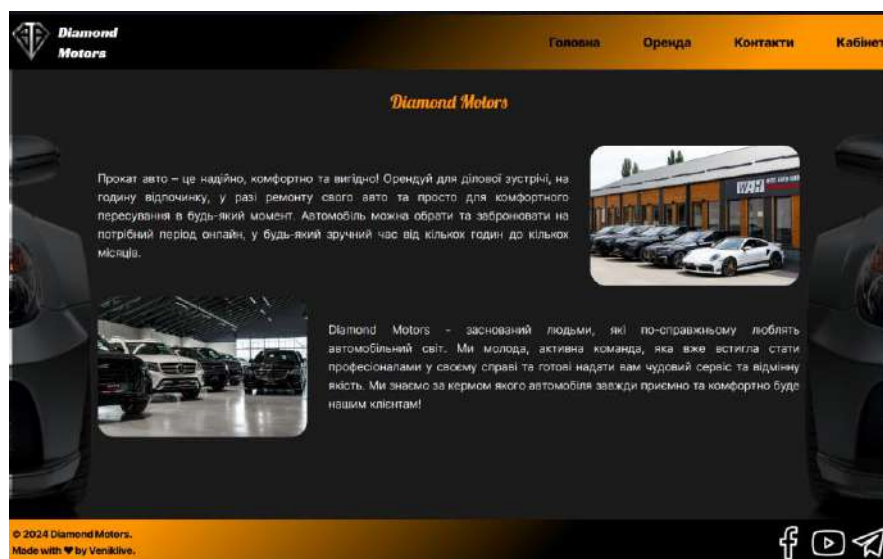


Рисунок 3.1 – Дизайн інформаційної сторінки сайту

Згідно технічного завдання, необхідно розробити сторінку з контактами, яка дозволить користувачам зв'язатися з компанією для швидкого вирішення питань або отримання консультації. Дизайн контактної сторінки зображено на рисунку 3.2.

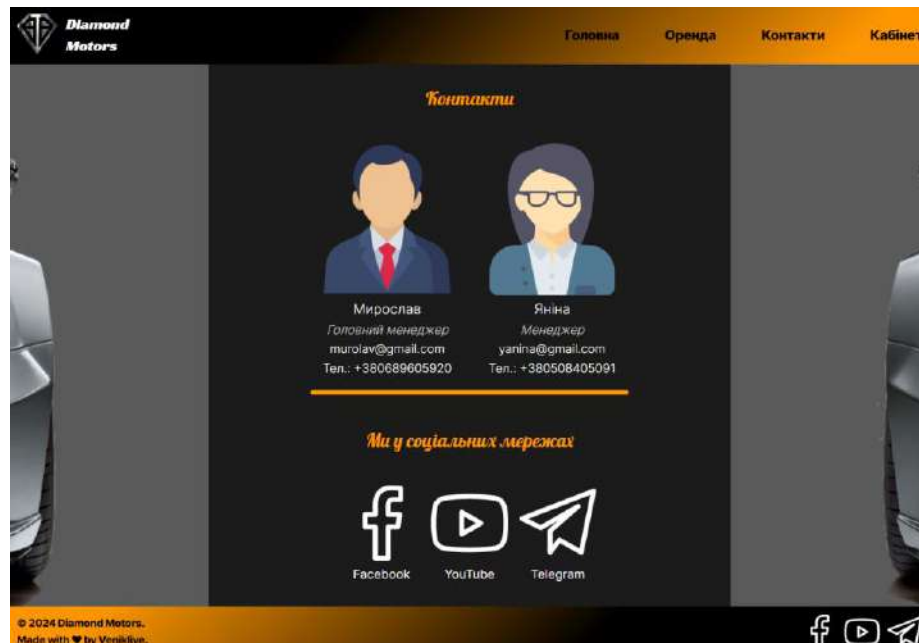


Рисунок 3.2 – Дизайн контактної сторінки сайту

Найважливішою сторінкою сайту є сторінка з формою оренди. Форма оренди повинна містити в собі:

- слайдер з автомобілями;
- календар для обрання дня початку оренди;
- лічильник кількості днів оренди;
- блок для виведення загальної ціни оренди;
- поле для вводу контактного номеру телефона;
- кнопка відправки форми.

У слайдері з автомобілями необхідно розмістити назву транспортного засобу, тип палива, клас, тип кузова, кількість місць, кількість вільних автомобілів та ціну за день оренди. Зображення форми оренди наведено на рисунку 3.3.

Mercedes-Benz W223
 Тип палива: Дизель
 Клас: S
 Тип кузова: Седан
 Кількість місць: 5
 Кількість авто: 9
 Ціна за день: 2000

Оберіть день початку оренди:
 06.06.2024

Оберіть кількість днів оренди:
 5

Вартість оренди:
 10000 грн

Введіть номер телефону:
 380XXXXXXXX

ВІДПРАВИТИ ЗАЯВУ

Рисунок 3.3 – Дизайн форми

Коли клієнт відправить форму йому відкриється попап, у якому буде відображатись інформація про успішну відправлену заявку або помилка заповнення форми. Створений попап успішної відправки замовлення зображено на рисунку 3.4. Попап з помилкою заповнення форми наведено у рисунку 3.5.

Ви відправили замовлення

Ваше ім'я: Venik
 Ваша пошта: venikvivivi@gmail.com
 Ваш номер: 380956234646
 Орендований транспорт: Alfa Romeo 159
 Кількість днів оренди: 12
 Вартість оренди: 18000 грн

Рисунок 3.4 – Вспливаюче вікно успішної відправки форми

Ваше замовлення не відправлено

Перевірте правильність введених даних. Можливо замовляемого автомобіля немає у наявності або не обрана початкова дата оренди.

Рисунок 3.5 – Вспливаюче вікно провальної відправки форми

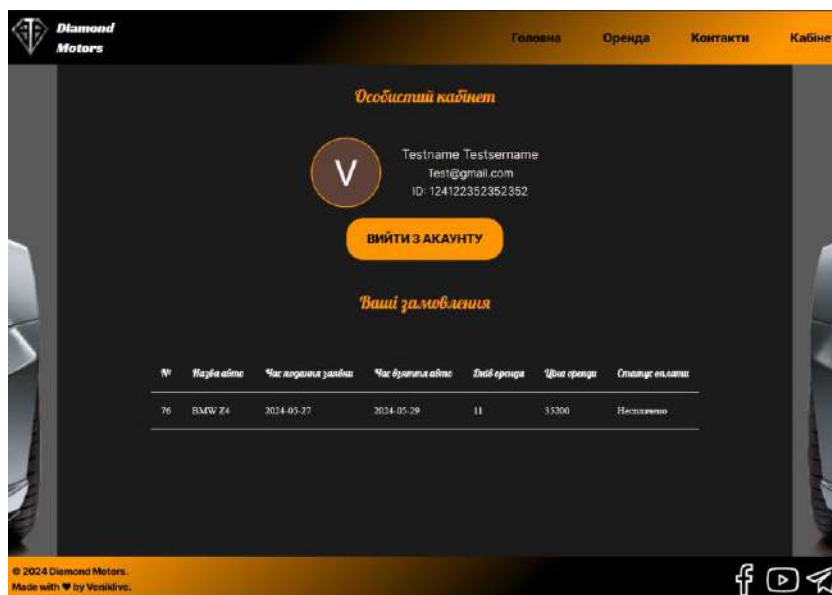


Рисунок 3.6 – Дизайн особистого кабінету

Особистий кабінет містить усю необхідну інформацію про користувача, кнопку виходу з акаунту та інформацію про замовлення клієнта

3.1.2 Розробка функціональної частини клієнтського інтерфейсу

Найважливішою функцією є динамічне створення вмісту сторінки оренди, а саме виведення інформації про доступні авто, створення форми та функціонал відправки форми до серверної частини. Код який відправляє запит на отримання інформації про авто та динамічно створює слайди у слайдері на сторінці наведено у лістингу 3.1.

Лістинг 3.1 – Відправка GET запиту та створення слайдів

```
new Promise(async (resolve) => {
  let response = await
fetch(`http://localhost:5015/ShowDataBase`)
  let data = await response.json();
  var length = data.length;
  const search_row = await document.querySelector('.swiper-
wrapper');
  //Створення слайдів з автомобілями
  for (const item of data) {
    search_row.innerHTML +=
<div class="swiper-slide" style="background: center / cover no-
repeat url('${item.car_img}');">
<div class="hide">${item.car_id}</div>
<div class="swiper-name">${item.car_name}</div>
```

```

    <div class="swiper-fuel">Тип палива:
<span>${item.fuel_name}</span></div>
    <div class="swiper-class">Клас:
<span>${item.class}</span></div>
    <div class="swiper-body">Тип кузова:
<span>${item.body_name}</span></div>
    <div class="swiper-seats">Кількість місць:
<span>${item.number_of_seats}</span></div>
    <div class="swiper-quantity">Кількість авто: <span
class="quantityCar">${item.car_quantity}</span></div>
    <div class="swiper-price">Ціна за день: <span
class="price">${item.car_price}</span></div>
</div>`;
length = length - 1;};

```

Після обрання автомобіля користувачу необхідно обрати кількість днів оренди, для цього у формі маєтся лічильник у вигляді двох кнопок та тексту з числом. Реалізація кнопок зображено у лістингу 3.2. Клієнтська частина шукає дві кнопки з класами «rent-form__img minus» та «rent-form__img plus», вішає на них обробник подій, після якого додається чи віднімається кількість днів оренди та викликаються функції оновлення ціни та оновлення даних о кількості вільних авто.

Лістинг 3.2 – Реалізація кнопок збільшення та зменшення кількості днів оренди.

```

const buttonPlus = document.querySelector(".rent-
form__img.plus");
const buttonMinus = document.querySelector(".rent-
form__img.minus");
buttonPlus.onclick = function(){
    const quantityDays = document.querySelector(".rent-
form__counter");
    if
(parseInt(quantityDays.innerHTML)<30){quantityDays.innerHTML=parseIn
t(quantityDays.innerHTML)+1;
        updatePrice();
        updateCarQuantity();}};
buttonMinus.onclick = function(){
    const quantityDays = document.querySelector(".rent
form__counter");
    if (parseInt(quantityDays.innerHTML)>1){
        quantityDays.innerHTML=parseInt(quantityDays.innerHTML
)-1;
        updatePrice();
        updateCarQuantity();}};

```

При завантаженні форми та її зміні необхідно оновлювати інформацію щодо ціни та кількості вільних автомобілів. Код реалізації лічильника ціни оренди наведено у лістингу 3.3, в якому спочатку знаходиться блок слайдера з класом «active» та його дочірні блоки, які містять ціну та кількість днів, після чого ціна рахується та створює рядок у контейнері з класом «rent-form__display».

Лістинг 3.3 – Реалізація функції зміни ціни оренди.

```
function updatePrice(){
  const quantityDays = document.querySelector(".rent-
form__counter");
  const getPrice = document.querySelector(".swiper-slide.swiper-
slide-active").querySelector(".price");
  const setPrice = document.querySelector(".rent-form__display");
  setPrice.innerHTML=parseInt(getPrice.innerHTML)*parseInt(qu
antityDays.innerHTML)+" грн";}
```

Клієнт не може орендувати вже зайняті автомобілі, тому є необхідність створення функції, яка зможе надати поточну кількість вільних автомобілів у обраному для оренди проміжку часу. Для цього створено функцію, код якої зображено у лістингу 3.4. Вона бере дані з контейнерів сторінки, які містять дату початку оренди та кількість днів оренди, після чого відправляє GET запит до серверної частини системи на отримання кількості вільних авто у цьому проміжку часу. Після отримання необхідних даних, замінює кількість автомобілів на всіх слайдах.

Лістинг 3.4 – Реалізація функції зміни кількості автомобілів.

```
async function updateCarQuantity(){
  const swipers = document.querySelectorAll(".swiper-slide");
  const checkDate = document.querySelector(".rent-
form__calendar").value;
  const numberDays = document.querySelector(".rent-
form__counter").textContent;
  let response = await
fetch(`http://localhost:5015/CheckCarQuantity?checkDate=${checkDate}
&numberDays=${numberDays}`);
  let carData = await response.json();
  swipers.forEach(element => {
    const carID = element.querySelector(".hide").textContent;
    const carQuantity = element.querySelector(".swiper-quantity");
```

```

const car = carData.find(obj => obj.car_id == carID);
if (car) {
  carQuantity.querySelector('.quantityCar').innerHTML =
car.available_cars;}
});}

```

Після розробки необхідних функцій отримання поточної інформації про авто, створено код, який додає до сторінки форму, яка залежить від статусу користувача. Якщо користувач системи є гостем, тоді йому відображають слайдер з автомобілями, калькулятор цін з обранням кількості днів оренди та кнопку реєстрації. Коли користувачем є клієнт, тобто авторизований користувач, відображається вся форма включно з обранням дня початку оренди та полем для вводу номера телефону. Код реалізації форми оренди наведено у лістингу А.1.

3.2 Розробка серверної частини

Розробивши клієнтську частину системи перейдено до розробки серверної. Згідно технічного завдання, для розробки серверної частини обрано мову JS та асинхронне середовище виконання обрано Node.js. У серверній частині необхідно реалізувати підключення до бази даних та динамічно обчислювати запити клієнтів до серверу.

Від серверної частини потрібні такі функції:

- підключення до бази даних;
- обробка запитів на отримання інформації про увесь транспорт з його характеристиками;
- авторизація за допомогою Google сервісу;
- відправка вільних автомобілів за вказаним проміжком часу;
- обробка запитів на додання заявок до БД;
- обробка запитів на отримання поточної інформації про стан замовлень.

Для підключення бази даних на сервері необхідно вказати порт мережі та зв'язати його з БД. Портом обрано число “5015”. Для підключення БД до серверу вказано назву бази даних “rent_car” та пароль для її доступу “root”. Код підключення до БД наведено у лістингу 3.5.

Лістинг 3.5 – JS код підключення до БД

```
const app = express();
const port = 5015;
var connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'root',
  database: 'rent_car'});
```

Обробку запитів на отримання інформації про транспорт реалізовано за допомогою фреймворка Express.js. У коді написано підключення до БД, відправка запиту на мові SQL до БД та отримування необхідної інформації у вигляді відповіді. Увесь код для обробки запиту наведено у лістингу 3.6.

Лістинг 3.6 – код обробки запиту на отримання даних про авто

```
app.use(
  '/ShowDataBase',
  async(req, res)=>{
    (await connection).query(`SELECT
      c.car_id,
      c.car_name,
      c.class,
      c.number_of_seats,
      c.car_quantity,
      c.car_price,
      c.car_img,
      f.fuel_name,
      b.body_name
    FROM car c
      JOIN body b ON b.body_id = c.body_id
      JOIN fuel f ON f.fuel_id = c.fuel_id`)
    .then(response=>{
      res.send(response[0]);}))});
```

Авторизація за допомогою Google сервісів розроблена з використанням стратегій «passport-google-oauth2» та бібліотеки «Passport». При ініціалізації

passport стратегією Google вказується дані клієнта та адрес зворотного виклику для повернення результатів запиту. Код ініціалізації наведено у лістингу 3.7.

Лістинг 3.7 – ініціалізація passport

```
passport.use(new GoogleStrategy({
  clientID: GOOGLE_CLIENT_ID,
  clientSecret: GOOGLE_CLIENT_SECRET,
  callbackURL: "http://localhost:5015/auth/google/callback",
  passReqToCallback: true, },
```

Для роботи авторизації необхідно створити ідентифікатор клієнта OAuth та профілю сервера електронної пошти. OAuth (Open Authorization) — це відкритий стандарт для авторизації, що дозволяє стороннім додаткам отримувати обмежений доступ до ресурсів користувача без передачі паролів. Налаштування Google OAuth2.0, для аутентифікації користувачів у Node.js за допомогою бібліотеки Passport, наведено у лістингу 3.8.

Лістинг 3.8 – Налаштування Google OAuth2.0

```
const passport = require('passport');
const GoogleStrategy = require('passport-google-
oauth2').Strategy;
const GOOGLE_CLIENT_ID = "436472644930-
bd3ulhjmppl08g3lmjbtu410m0hkg53m.apps.googleusercontent.com";
const GOOGLE_CLIENT_SECRET = "GOCSPX-
DjsDM1oJdpctF1MEkXbh4q0NYTi8";
passport.use(new GoogleStrategy({
  clientID: GOOGLE_CLIENT_ID,
  clientSecret: GOOGLE_CLIENT_SECRET,
  callbackURL: "http://localhost:5015/auth/google/callback",
  passReqToCallback: true, },
```

Далі необхідно налаштувати авторизацію на сервері через проміжне програмне забезпечення express необхідно передати сесію за налаштуваннями, ініціалізацію passport та налаштовану сесію passport. Функція `isLoggedIn()`, у випадку неуспішної авторизації, повертає статус «401». Функція авторизації наведена у лістингу 3.9.

Лістинг 3.9 – Код функції авторизації

```
app.use(session({
  secret: 'cats',
  resave: false,
  saveUninitialized: true
}));
app.use(passport.initialize());
app.use(passport.session());
function isLoggedIn(req, res, next) {
  res.setHeader('Access-Control-Allow-Credentials', true);
  req.user ? next() : res.sendStatus(401);
}
```

Після авторизації, користувач перенаправляється на сторінку особистого кабінету, або при провальній авторизації його відправляють на сторінку повторної авторизації. Код функції перенаправлення наведено у лістингу 3.10.

Лістинг 3.10 – Код функції перенаправлення користувача

```
app.get('/auth/google',
passport.authenticate('google', { scope: [ 'email', 'profile'
]}));
app.get( '/auth/google/callback',
  passport.authenticate('google', {
    successRedirect: 'http://localhost:5500/cabinet.html',
    failureRedirect: '/auth/google/failure'}));
```

Задля отримання інформації про користувача створено функцію, яка повертає усю необхідну інформацію у вигляді JSON рядка, код функції наведено у лістингу 3.11.

Лістинг 3.11 – Код отримання даних від авторизованого користувача

```
app.get('/protected', isLoggedIn, (req, res) => {
  res.setHeader('Access-Control-Allow-Credentials', true);
  let info = {
    picture: req.user.picture,
    id: req.user.id,
    email: req.user.email,
    name: req.user.given_name,
    surname: req.user.family_name,};
  let json = JSON.stringify(info);
  res.send(json);
});
```

Останнім кроком створення повноцінної системи авторизації є надання користувачу можливості виходу з акаунту. Для цього знищується поточна сесія користувача, тим самим деавторизуючи його. Після виходу з акаунту, будь-яка подальша взаємодія з системою вимагатиме повторного входу.

Код виходу з акаунту наведено у лістингу 3.12. Він встановлює заголовок `Access-Control-Allow-Credentials`, щоб дозволити передачу облікових даних в крос-доменних запитах, викликає метод `logout`, щоб завершити сесію, а потім знищує сесію користувача.

Лістинг 3.12 – Код виходу з акаунту

```
app.get('/logout', (req, res) => {
  res.setHeader('Access-Control-Allow-Credentials', true);
  req.logout(err => {
    if (err) {
      return next(err);
    }
    req.session.destroy(() => {
      res.send('Logout');
    });
  });
});
```

Відправка даних про вільні автомобілі за вказаним проміжком часу реалізована за допомогою SQL запиту до БД, в якому порівнюється заданий проміжок часу з проміжками часу орендованих автомобілів, результатом чого з БД до серверної частини відправляється відповідь з кількістю вільних авто, яку відправляють до клієнтської частини. Код знаходження кількості вільних автомобілів наведено у лістингу 3.13.

Лістинг 3.13 – Код знаходження кількості автомобілів за заданим проміжком часу.

```
app.use('/CheckCarQuantity',
  async(req, res)=>{
    var{checkDate, numberDays}=req.query;
    let checkDateMS = new Date(checkDate);
    checkDateMS.setDate(checkDateMS.getDate() + numberDays);
    let newDateStr = checkDateMS.toISOString().split('T')[0];
    (await connection).query(`
```

```

        SELECT car.car_id, car.car_quantity, COUNT(rent.car_id) AS
used_cars,
        car.car_quantity - COUNT(rent.car_id) AS available_cars
FROM car
LEFT JOIN rent ON car.car_id = rent.car_id
                AND '${newDateStr}' >= rent.day_of_taken
                AND '${checkDate}' <= (rent.day_of_taken + INTERVAL
rent.number_days_rent DAY)
GROUP BY car.car_id;
`)
    .then(response=>{
        res.send(response[0]);})
    }
)

```

Обробка запитів на додання заявок до бази даних реалізована за допомогою SQL-запиту від серверної частини. Цей запит вставляє нові заявки у відповідну таблицю бази даних. У коді також передбачено використання транзакцій для забезпечення цілісності даних. У разі помилки транзакція буде відкатана, щоб уникнути часткового запису даних. Код додання заявок наведено у лістингу 3.14.

Лістинг 3.14 – Реалізована обробка запитів на запис заявок.

```

app.use(
  '/AddRent',
  async(req, res)=>{
    var{car_id, client_name, client_email, client_number,
number_days_rent, day_of_taken}=req.query;
    (await connection).beginTransaction();
    try{
      var query = `INSERT INTO rent (car_id, client_name,
client_email, client_number, number_days_rent, day_of_taken)
VALUES(${car_id}, '${client_name}','${client_email}',
'${client_number}', '${number_days_rent}', '${day_of_taken}')`;
      await (await connection).query(query)
    }.catch(async (error) =>
    {
      await (await connection).rollback();
      console.log(error);
    });
    await (await connection).commit();
    catch(error){console.log(error);};
    res.status(200).json("success");
  })
)

```

Після відправки клієнтами їхніх заповнених форм, необхідно надати їм можливість перегляду статусів їхніх заявок. Реалізовано це за допомогою відправки SQL запиту до бази даних для отримання поточної інформації про стан замовлень, код наведено у лістингу 3.15. Далі від серверної до клієнтської частини надходить інформація, яка містить усі дані про замовлення від авторизованого користувача. Такий підхід дозволяє клієнтам зручно контролювати стан своїх замовлень.

Лістинг 3.15 – Запит на отримання стану замовлень клієнта

```
app.use(
  '/ShowClientsRents',
  async(req, res)=>{
    const {client_email} = req.query;
    (await connection).query(`SELECT
      r.rent_id,
      r.car_id,
      r.day_of_rent,
      r.number_days_rent,
      r.day_of_taken,
      r.car_return,
      r.paid,
      c.car_name,
      c.car_price
    FROM
      rent r
      JOIN car c ON c.car_id = r.car_id
    WHERE
      client_email = '${client_email}'
    ORDER BY rent_id;
  `)
    .then(response=>{
      res.send(response[0]);
    })
  });
```

Розробка клієнтської та серверної функціональних частин системи закінчена, далі необхідно перейти до наступного етапу розробки, тестування системи.

4 ТЕСТУВАННЯ РОЗРОБЛЕНОЇ КОМП'ЮТЕРНОЇ СИСТЕМИ

Останнім, але не менш важливим, етапом розробки є тестування системи. Воно дозволить виявити та виправити помилки розробки, забезпечуючи надійність системи. Система повинна відповідати вимогам технічного завдання та бути повністю функціональною.

4.1 Перевірка адаптивності

Спершу необхідно перевірити адаптивність вебсайту до різних розмірів дисплею. Тестування адаптивності зображено на рисунках 4.1 та 4.2. Це включає перевірку правильності відображення контенту на різних пристроях, таких як десктопи, планшети та смартфони. Необхідно перевірити, що всі елементи інтерфейсу, включаючи текст, зображення, кнопки та меню, правильно масштабуються та розташовуються в межах екранів різних розмірів.

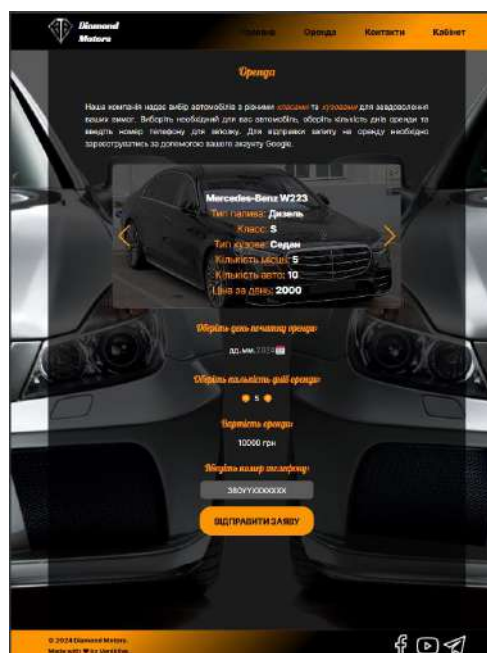


Рисунок 4.1 – Вигляд сторінки на пристроях з дисплеєм 1920 на 1080

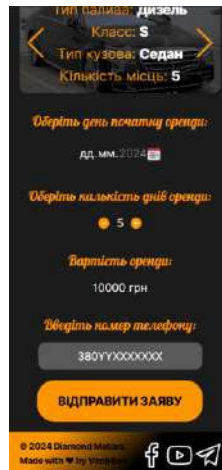


Рисунок 4.2 – Вигляд сторінки на пристроях з дисплеєм 320 на 675

Сайт відповідає усім сучасним вимогам адаптивності, правильно відображає елементи сторінок на різних екранах та зберігає свою функціональність.

4.2 Перевірка функціоналу реєстрації

Далі необхідно перевірити функціонал реєстрації користувачів. Частина робочого функціоналу реєстрації через системи Google наведена на рисунку 4.3.

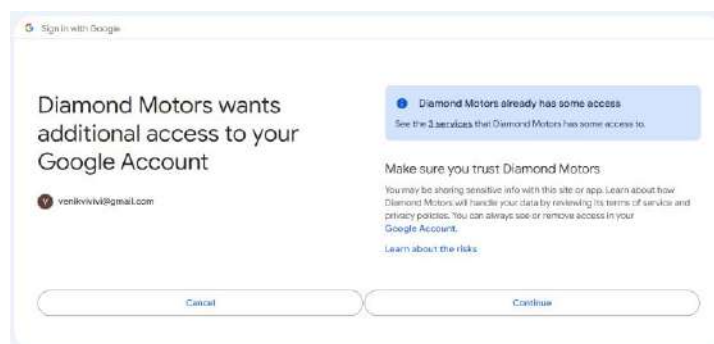


Рисунок 4.3 – Реєстрація за допомогою системи Google

Після реєстрації користувача перенаправляють на сторінку особистого кабінету, де відображається його особиста інформація. Додатково перевірено клієнтський доступ до повної форми.

4.2 Перевірка функціоналу відправки замовлень

Головним функціоналом комп'ютерної системи оренди транспорту є правильне заповнення та відправка форми. Повідомлення про успішну відправку заявки наведено на рисунку 4.4 та 4.5.

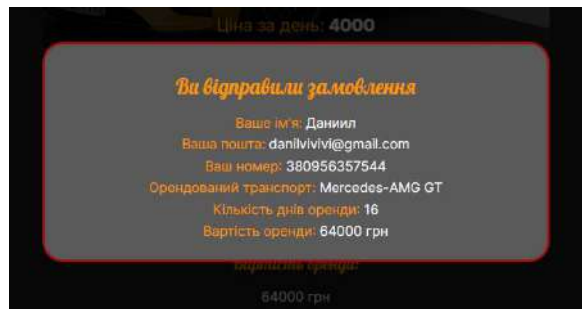


Рисунок 4.4 – Впливаюче вікно відправки форми

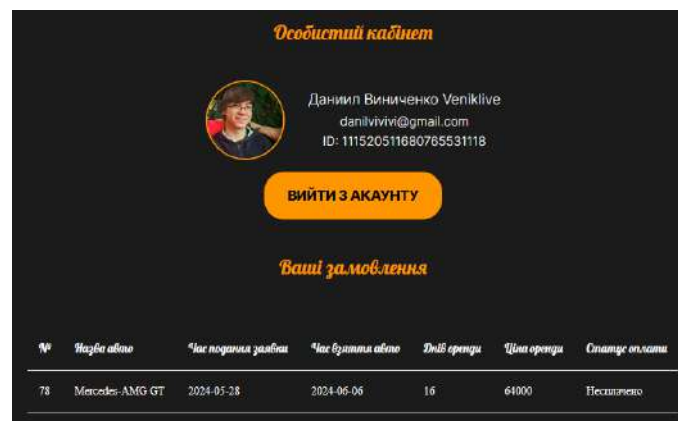


Рисунок 4.5 – Відображення стану заявки у особистому кабінеті

Функціонал системи оренди транспорту протестовано, помилки розробки виправлено. Система повністю відповідає вимогам технічного завдання. У системі клієнт може авторизуватись, відправити форму та побачити статус заявки. Адміністратор має можливість редагування вмісту бази даних, отримувати заявки від користувачів та редагувати їх. Система також оновлює дані щодо вільних автомобілів у заданому проміжку часу.

ВИСНОВКИ

Результатом роботи є розроблена комп'ютерна система оренди транспорту, яка відповідає поставленим задачам проєкту. Система забезпечує автоматизацію отримання заяв, обліку транспорту, клієнтів та оренди, спрощуючи роботу з орендою транспортних засобів.

Актуальність системи підтверджується стрімким ростом інтернет послуг, відсутністю доступних безкоштовних та платних аналогів системи, індивідуальними потребами кожного бізнесу.

Перевагами системи є її можливості легкої масштабованості для відповідності до зростаючих потреб бізнесу при його розвитку в майбутньому. Створена вебсистема має зручний та зрозумілий інтерфейс, що полегшує взаємодію користувачів з системою..

У підсумку, розроблена комп'ютерна система оренди транспорту відповідає вимогам автоматизації роботи бізнесу та обліку транспорту, клієнтів та оренди. Впровадження такої системи може покращити ефективність та надійність процесів оренди транспорту, сприяючи зручності клієнтів та її власників.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. QATestLab: Клієнт-серверна архітектура :вебсайт. URL: <https://training.qatestlab.com/blog/technical-articles/client-server-architecture/> (дата звернення: 29.04.2024);
2. ItHillel: Як обрати архітектуру для веб додатку :вебсайт. URL: <https://blog.ithillel.ua/articles/web-application-architecture> (дата звернення: 29.04.2024);
3. QALight: Монолітна архітектура :вебсайт. URL: <https://qalight-ua.translate.google.com/baza-znaniy/shho-take-monolitna-arhitektura/> (дата звернення: 29.04.2024);
4. LiquidWeb: Guide to Client-server Architecture or Model :вебсайт. URL: <https://www.liquidweb.com/blog/client-server-architecture/> (дата звернення: 29.04.2024);
5. Scandiweb: HTML vs HTML5 vs XHTML :вебсайт. URL: <https://scandiweb.com/blog/html-vs-html5-vs-xhtml-the-difference-in-a-nutshell/> (дата звернення: 29.04.2024);
6. Freehost.ua: Що таке PHP? :вебсайт. URL: <https://freehost.com.ua/faq/wiki/chto-takoe-php/> (дата звернення: 30.04.2024);
7. Freehost.ua: Що таке мова програмування Python? :вебсайт. URL: <https://freehost.com.ua/ukr/faq/wiki/chto-takoe-jazik-programmirovanija-python/> (дата звернення: 30.04.2024);
8. LemonSchool: Навіщо потрібна мова програмування Ruby? :вебсайт. URL: <https://lemon.school/blog/navishho-potribna-mova-programuvannya-ruby> (дата звернення: 30.04.2024);
9. GoIt: Що таке Java і де вона використовується :вебсайт. URL: <https://goit.global.ua/articles/shcho-take-java-i-de-vona-vykorystovuietsia/> (дата звернення: 30.04.2024);

10. GoIt: Що таке JavaScript і для чого він потрібен :вебсайт. URL: <https://goit.global/ua/articles/shcho-take-javascript-i-dlia-choho-vin-potriben/> (дата звернення: 30.04.2024);
11. HIGHLOAD: СУБД: які бувають, як вибрати :вебсайт. URL: <https://highload.today/uk/subd-yaki-buvayut-yak-vibrati/> (дата звернення: 01.05.2024);
12. Oracle: What is MySQL?: :вебсайт. URL: <https://www.oracle.com/mysql/what-is-mysql/> (дата звернення: 01.05.2024);
13. Oracle: Oracle Database Technologies :вебсайт. URL: <https://www.oracle.com/database/technologies/> (дата звернення: 01.05.2024);
14. SimpliLearn: What is Microsoft Access? :вебсайт. URL: <https://www.simplilearn.com/what-is-microsoft-access-article> (дата звернення: 01.05.2024);
15. HeidiSQL: What's this? :вебсайт. URL: <https://www.heidisql.com> (дата звернення: 01.05.2024).

ДОДАТОК А

ОСНОВНИЙ ПРОГРАМНИЙ КОД СТОРІНКИ З ФОРМОЮ

Лістинг А.1 – Код форми оренди

```

new Promise(async (resolve) => {
  let response = await fetch(`http://localhost:5015/protected`, {
    method: "GET",
    mode: "cors",
    credentials: "include"
  });
  if (response.status==401){
    const search_button = await document.querySelector('.rent-
form');
    search_button.innerHTML +=
      <button class="button open-popup Auth" name="cmdAuth"
type="submit"><span>Реєстрація</span></button>
    const buttonAuth = document.querySelector(".button.open-
popup.Auth");
    buttonAuth.onclick = async function(){
window.location.href=`http://localhost:5015/auth/google`;};
    }
    else{
      let data = await response.json();
      const search_row_calendar = await
document.querySelector('.rent-form__input_calendar');
      let yourDate = Date.now();
      let minDate=new Date(yourDate).toISOString().split('T')[0];
      let maxDate=yourDate+5184000000;
      maxDate = new Date(maxDate).toISOString().split('T')[0];
      search_row_calendar.innerHTML +=
      <div class="rent-form__title">Оберіть день початку
оренди:</div>
      <div class="rent-form__block">
      <input class="rent-form_calendar" type="date" name="calendar"
max="{maxDate}" min="{minDate}">
      </div>
      `;
      const search_row = await document.querySelector('.rent-form');
      search_row.innerHTML +=
      `
      <div class="rent-form__input_tel">
      <div class="rent-form__title">Введіть номер телефону:</div>
      <input required type="tel" name="Телефон" minlength="12"
maxlength="12" class="rent-form__tel" placeholder="380YXXXXXXX"
value="">
      </div>
      <button class="button open-popup" name="cmdPost"
type="submit"><span>Відправити заяву</span></button>
      `;
      //Перевірка правильності вводу номеру

```

```

    const phoneBlock = document.querySelector(".rent-
form__input_tel");
    phoneBlock.addEventListener("keyup", (event)=>{
        let target = event.target;
        if (target.value.length==12) {
            target.classList.add("valid");
            target.classList.remove("invalid");
            event.target.closest(".rent-
form__tel").classList.add("active");
        }else if(target.value.length>12){
            target.value = target.value.slice(0, 12);
        } else{target.classList.add("invalid");
            target.classList.remove("valid");
            event.target.closest(".rent-
form__tel").classList.remove("active");}
    });
    //Кнопка відправки замовлення
    const button = document.querySelector(".open-popup");
    button.onclick = function() {
        const phone = document.querySelector(".rent-form__tel").value;
        const id = document.querySelector(".swiper-slide.swiper-slide-
active").querySelector(".hide").innerHTML;
        const car_name = document.querySelector(".swiper-slide.swiper-
slide-active").querySelector(".swiper-name").innerHTML;
        const days = document.querySelector(".rent-
form__counter").innerHTML;
        const price = document.querySelector(".rent-
form__display").innerHTML;
        const quantityCar = document.querySelector(".swiper-
slide.swiper-slide-active").querySelector(".quantityCar").innerHTML;
        const day_of_taken = document.querySelector(".rent-
form__calendar").value;
        if (document.querySelectorAll(".selected")) {
            if (document.querySelectorAll(".active").length==1 &&
quantityCar>=1 && document.querySelector(".valid") &&
day_of_taken.length>=1) {
                const name = data.name;
                const email = data.email;
                console.log(id, name, email, phone, days);
                fetch(`http://localhost:5015/AddRent?car_id=${id}&client_name=${name}
&client_email=${email}&client_number=${phone}&number_days_rent=${da
ys}&day_of_taken=${day_of_taken}`);
                formPopup.classList.add("open");//activate Popup
                formTextPopup.classList.add("open");//activate PopupText
                const rentPopup = document.querySelector('.popup.open');
                rentPopup.innerHTML =
                `
                <div class="popup_title_text">Ви відправили
замовлення</div>
                <div class="popup_text">Ваше ім'я: <span>${name}</span></div>
                <div class="popup_text">Ваша пошта: <span>${email}</span></div>
                <div class="popup_text">Ваш номер: <span>${phone}</span></div>
                <div class="popup_text">Орендований транспорт:
<span>${car_name}</span></div>

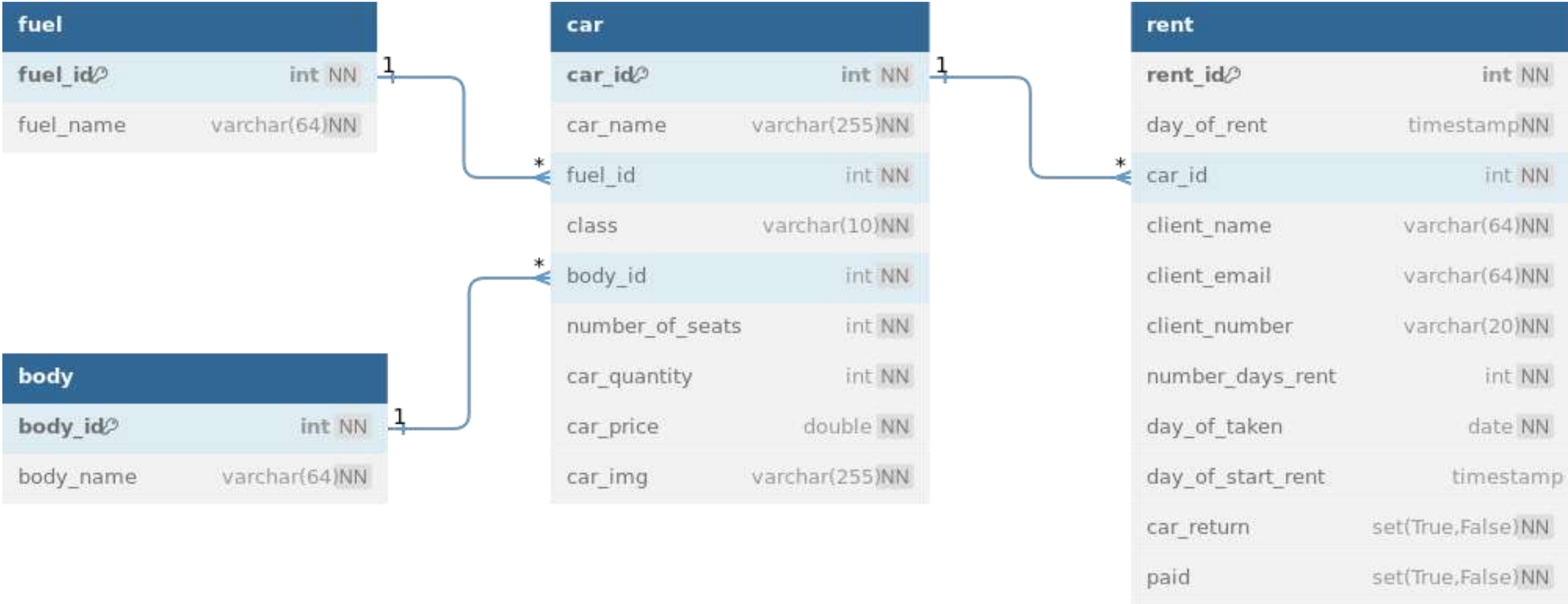
```

```

    <div class="popup_text">Кількість днів оренди:
<span>${days}</span></div>
    <div class="popup_text">Вартість оренди:
<span>${price}</span></div>`;
    }else{
        const name = data.name;
        const email = data.email;
        console.log(id, name, email, phone, days, day_of_taken);
        formPopup.classList.add("open");//activate Popup
        formTextPopup.classList.add("open");//activate PopupText
        const rentPopup = document.querySelector('.popup.open');
        rentPopup.innerHTML =
        `    <div class="popup_title_text">Ваше замовлення не
відправлено</div>
        <div class="popup_text"><span>Перевірте правильність введених
даних. Можливо замовляемого автомобіля немає у наявності або не
обрана початкова дата оренди.</span></div>`;}}
    resolve();}

```

Відношення таблиць БД системи

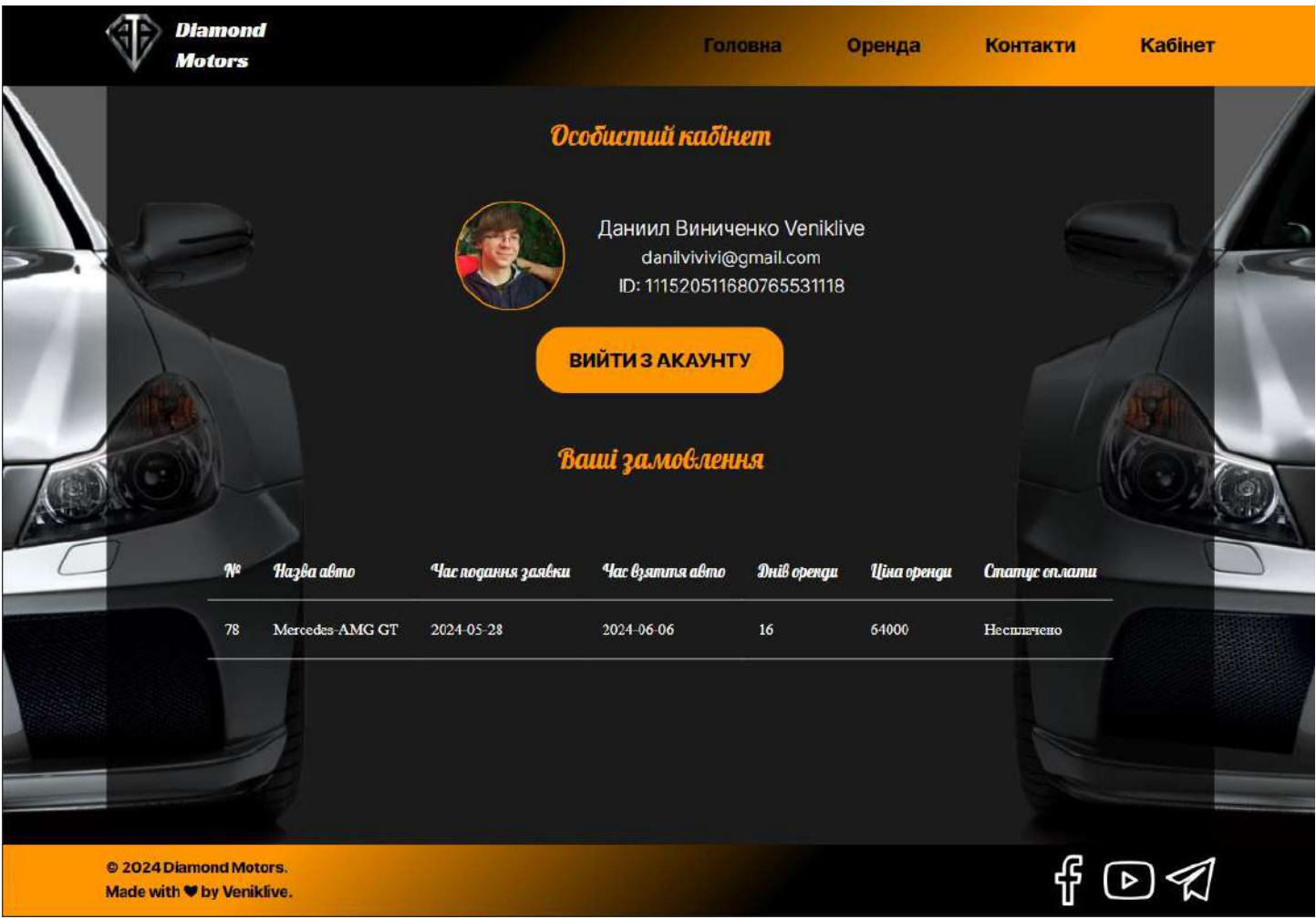


					13.02070849.00037 ПЛ1			
					Розробка комп'ютерної системи для оренди транспортних засобів	Лист.	Маса	Масшт.
Зам.	Лист.	№ докум.	Підп.	Дата				
Розроб.		Виниченко Д.В.	ВВ					
Перев.		Хохлов М.М.	ХХ					
Т.контр.					Відношення таблиць БД системи	Лист 1		Листів 1
Н.контр.		Польська О.В.	Пол			НУ «Запорізька політехніка» КНТ-520		
Затв.		Кудерметов Р.К.	Куд					

Сайт системи для оренди транспортних засобів



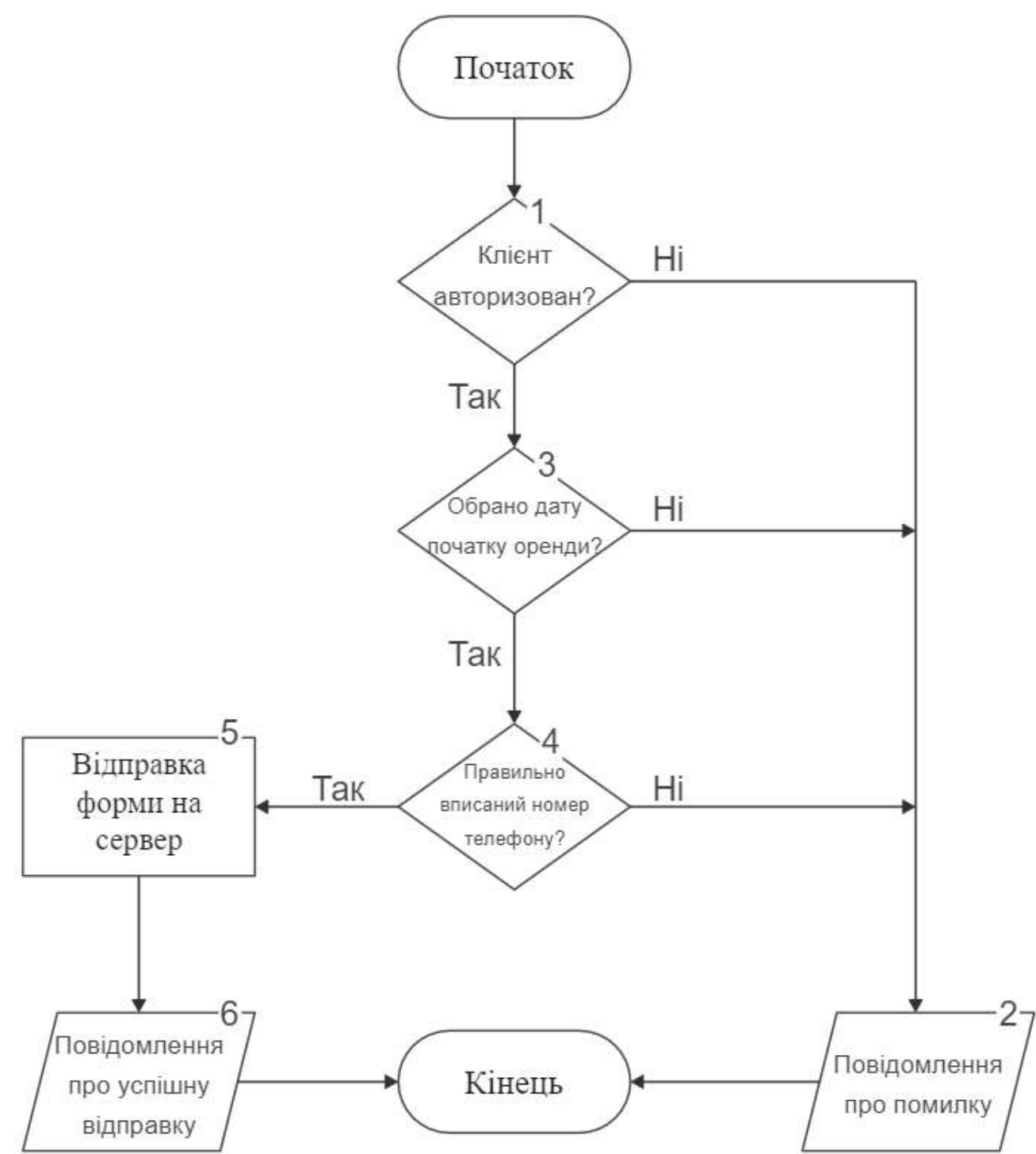
Сторінка оформлення форми



Сторінка особистого кабінету клієнта

					13.02070849.00037 ПЛ2					
					Розробка комп'ютерної системи для оренди транспортних засобів	Лит.		Маса	Масшт.	
						Лист 1		Листів 1		
						НУ «Запорізька політехніка» КНТ-520				
Зам.	Лист.	№ докум.	Підп.	Дата	Сайт системи для оренди транспортних засобів					
Розроб.		Виниченко Д.В.								
Перев.		Хохлов М.М.								
Т.контр.										
					Сайт системи для оренди транспортних засобів					
Н.контр.		Польська О.В								
Затв.		Кудерметов Р.К.								

Алгоритм відправки форми замовлення



					13.02070849.00037 ПЛЗ			
					Розробка комп'ютерної системи для оренди транспортних засобів	Лист.	Маса	Масшт.
Зам.	Лист.	№ докум.	Підп.	Дата				
Розроб.		Виниченко Д.В.	ВВ					
Перев.		Хохлов М.М.	ХХ					
Т.контр.					Алгоритм відправки форми замовлення	Лист 1		Листів 1
Н.контр.		Польська О.В.	Пол			НУ «Запорізька політехніка» КНТ-520		
Затв.		Кудерметов Р.К.	Куд					