НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"____" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

Програмне забезпечення інформаційної системи прокату автомобілів

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
К.т.н., доцент _____ Вайганг Г.О.

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ Вайганг Г.О.
науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Жидченко В.М.
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"____" _____ 2025 р.

ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи студенту

Жидченку В'ячеславу Миколайовичу

Спеціальність 121 «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: Програмне забезпечення інформаційної системи прокату автомобілів

затверджена наказом ректора НУБіП України № 2249 "С" від 16.12.2024

Термін подання завершеної роботи на кафедру 2025.06.04
(рік, місяць, число)

Вихідні дані до роботи: опис програмного забезпечення

Перелік питань що розглядаються:
1. Системний аналіз предметної області інформаційної системи
2. Проєктування інформаційного та програмного забезпечення
3. Розробка інформаційного та програмного забезпечення
4. Рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання «_____»_____2025 р.
Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ Вайганг Г.О.
науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Жидченко В.М.
(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

GUI (Graphical User Interface) – графічний інтерфейс користувача.

UI (User Interface) – інтерфейс користувача, що охоплює всі засоби взаємодії з програмним забезпеченням.

JSON (JavaScript Object Notation) – формат зберігання структурованих даних у вигляді тексту.

PyQt6 – бібліотека для створення графічного інтерфейсу користувача у Python на основі Qt6.

MVC (Model–View–Controller) – архітектурний шаблон для побудови програмного забезпечення з поділом на модель, подання та контролер.

ER-діаграма (Entity-Relationship Diagram) – діаграма сутностей і зв’язків, яка моделює логічну структуру бази даних.

UML (Unified Modeling Language) – уніфікована мова моделювання програмного забезпечення.

PDF (Portable Document Format) – формат електронного документа, призначений для збереження форматування та друку.

UUID (Universally Unique Identifier) – універсальний унікальний ідентифікатор, що використовується для ідентифікації об’єктів.

CRUD (Create, Read, Update, Delete) – базовий набір операцій для управління даними.

SQL (Structured Query Language) – мова структурованих запитів, яка використовується для керування реляційними базами даних.

API (Application Programming Interface) – інтерфейс прикладного програмування, що дозволяє взаємодію між програмними компонентами.

ReportLab – Python-бібліотека для генерації PDF-документів.

Flask – мікрофреймворк для створення веб-додатків на Python.

UX (User Experience) – досвід користувача при взаємодії з системою.

ВСТУП

У сучасних умовах цифровізації транспортної інфраструктури та зростання попиту на мобільні послуги оренди значного поширення набувають сервіси з прокату автомобілів, що функціонують як у фізичному, так і в онлайн-середовищі. Важливою передумовою ефективного функціонування таких сервісів є впровадження спеціалізованих інформаційних систем, здатних забезпечити автоматизовану обробку замовлень, управління автопарком, ведення обліку клієнтів, контроль наявності транспортних засобів і підтримку процесу укладання договорів. Програмне забезпечення для подібних систем повинне відповідати сучасним вимогам до масштабованості, інтерактивності, доступності та безпеки обробки персональних даних.

Із урахуванням багатокомпонентної структури бізнес-процесів у сфері оренди автомобілів, особливої уваги потребують аспекти взаємодії між підсистемами зберігання, обробки, аналізу й візуалізації даних. Застосування інформаційних технологій у цій галузі дозволяє забезпечити не лише оперативне обслуговування клієнтів, а й гнучке управління логістичними потоками, моніторинг стану автопарку, оптимізацію фінансових розрахунків, а також інтеграцію з платіжними шлюзами та системами ідентифікації користувачів. Саме тому виникає потреба у розробці спеціалізованого інструменту, що реалізує повний життєвий цикл прокату – від реєстрації користувача до завершення договору оренди й повернення транспортного засобу.

Метою дипломної роботи є розробка програмного забезпечення інформаційної системи прокату автомобілів, яка забезпечує автоматизацію ключових процесів взаємодії між клієнтом, адміністратором сервісу та системою обліку.

Така система має включати модулі реєстрації користувачів, керування транспортними засобами, бронювання, підписання договорів, обліку платежів та генерації звітної інформації. Окрему увагу приділено організації бази даних, розробці інтерфейсу користувача та моделюванню внутрішньої логіки

програмного засобу з урахуванням сучасних стандартів побудови програмних систем.

У межах поставленої мети передбачається виконання таких **завдань**:

– здійснити системний аналіз предметної області з урахуванням особливостей організації процесу прокату автомобілів;

– визначити функціональні та нефункціональні вимоги до інформаційної системи прокату;

– побудувати концептуальну модель бази даних та структуру основних сутностей системи;

– розробити архітектуру програмного забезпечення із поділом на логічні модулі;

– реалізувати графічний інтерфейс користувача з урахуванням принципів доступності, зручності та адаптивності;

– створити програмні модулі для управління клієнтами, транспортними засобами, бронюваннями та платежами;

– забезпечити функціональність генерації звітності щодо стану замовлень і використання автопарку;

– провести тестування системи в умовах моделювання реального робочого середовища;

– сформувати рекомендації щодо впровадження та масштабування системи в комерційних умовах.

Об’єктом дослідження виступає процес автоматизації бізнес-операцій у сфері оренди автомобілів шляхом застосування інформаційних технологій.

Предметом дослідження є засоби, методи та алгоритми проєктування, реалізації і тестування інформаційної системи прокату з урахуванням архітектурних і функціональних вимог до її компонентів.

Наукова новизна дослідження полягає у створенні універсального, масштабованого і функціонально завершеного програмного рішення, адаптованого до потреб малого та середнього бізнесу в сфері автомобільного прокату. Розроблена система базується на локальному серверному рішенні, що

підтримує автономну роботу, не потребує сторонніх підключень до зовнішніх баз і дозволяє забезпечити контроль даних у межах підприємства. Інтеграція в систему інструментів керування угодами, фінансовою інформацією, а також користувацький інтерфейс із адаптивною верстkою створюють підґрунтя для її ефективного використання в реальному середовищі експлуатації.

Структура дипломної роботи охоплює чотири основні розділи. У першому розділі подано аналіз предметної області, досліджено поточний стан автоматизації процесів у сфері прокату, а також виявлено основні проблеми, що потребують вирішення. У другому розділі розроблено концептуальну модель інформаційної системи, наведено архітектурне рішення, а також сформульовано вимоги до її компонентів. Третій розділ присвячено реалізації програмного забезпечення, опису бази даних, інтерфейсу користувача та логіки взаємодії. Четвертий розділ містить результати тестування, приклади використання системи у моделюваному середовищі, а також рекомендації щодо її впровадження та подальшої модернізації.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сфера прокату автомобілів належить до динамічних галузей сервісної економіки, в якій інтенсивно застосовуються сучасні інформаційні технології для оптимізації взаємодії між клієнтом і компанією-орендодавцем. Високий рівень конкуренції, потреба в швидкому обслуговуванні, мінімізація людського фактору в обробці заявок і формування звітності зумовлюють актуальність використання автоматизованих систем управління процесом оренди транспортних засобів. У межах таких систем особливу роль відіграє графічний інтерфейс користувача, адже саме він забезпечує комунікацію між замовником послуг, працівником сервісу та внутрішніми модулями програмного забезпечення.

Графічний інтерфейс інформаційної системи прокату автомобілів повинен бути інтуїтивно зрозумілим, адаптивним до різних типів пристроїв і платформ, забезпечувати ефективну навігацію між розділами та швидкий доступ до ключових функцій. Водночас інтерфейс має відповідати вимогам інформаційної безпеки, зокрема при роботі з персональними даними, договорами, історією платежів, а також виконувати роль посередника між користувачем і логікою обробки даних. На відміну від суто інформаційних порталів, у подібній системі інтерфейс реалізує повноцінну бізнес-логіку, включаючи механізми реєстрації, авторизації, керування сесіями, бронювання автомобілів, обліку штрафів, перегляду технічного стану та завершення операцій оренди.

В умовах кросплатформного використання програмного забезпечення важливим є врахування особливостей реалізації інтерфейсів на мобільних і десктопних пристроях. У мобільному середовищі пріоритет надається компактності, сенсорному керуванню та адаптивному розміщенню елементів, тоді як десктопні версії орієнтовані на розширену функціональність, гнучку навігацію, підтримку клавіатурного вводу та масштабовані таблиці даних.

Відповідно, побудова єдиного GUI, придатного для різних платформ, потребує застосування універсальних патернів та адаптивного дизайну, який враховує обмеження екрана, тип взаємодії та особливості користувацької поведінки. У табл. 1.1 представлено узагальнене порівняння характеристик мобільних та десктопних інтерфейсів у контексті реалізації інформаційної системи прокату автомобілів.

Таблиця 1.1

Порівняльна характеристика мобільних та десктопних інтерфейсів інформаційної системи прокату

Параметр	Мобільні інтерфейси	Десктопні інтерфейси
Тип взаємодії	Сенсорне введення	Клавіатура, миша
Навігація	Вкладки, свайпи, адаптивні меню	Панелі інструментів, меню
Доступність інформації	Обмежена через розмір дисплея	Повна видимість розділів і таблиць
Адаптивність	Обов’язкова	Опціональна
Частота оновлень	Часті оновлення UI з оновленнями ОС	Менш регулярні зміни
Пріоритет	Швидкий доступ до базових функцій	Контроль, деталізація, багатовіконність

Як видно з порівняння, розробка графічного інтерфейсу програмного забезпечення для сфери прокату вимагає чіткого поділу функціоналу відповідно до платформи з урахуванням гайдлайнів, таких як Material Design для Android або Fluent UI для Windows. Такі інтерфейси мають забезпечувати легку масштабованість, інтеграцію з системою бронювання, платіжним модулем та базою даних, яка містить відомості про транспортні засоби, клієнтів, договори, платежі, графіки доступності та технічне обслуговування. Крім цього, інтерфейс повинен враховувати регламентовані вимоги до обробки персональних даних відповідно до законодавства [1].

Предметна область дипломної роботи охоплює концепції, методи та засоби, що використовуються при побудові графічних інтерфейсів у програмних системах прокату автомобілів, із фокусом на адаптивність, кросплатформність і

функціональну завершеність. На основі вивчених відмінностей між типами інтерфейсів буде сформовано критерії оцінки зручності, відповідності стандартам і ефективності GUI, що застосовуються при створенні програмного засобу аналізу інтерфейсів прокатних систем.

1.2 Аналіз вимог до програмної системи

Аналіз вимог до інформаційної системи є ключовим етапом розробки програмного забезпечення, що забезпечує відповідність очікуванням користувачів, відповідність нормативним стандартам і доцільність архітектурних рішень. Вимоги поділяються на функціональні, що описують поведінку системи, та нефункціональні, які регламентують обмеження, якість і технічні умови експлуатації. Врахування обох категорій дозволяє створити повноцінну, надійну та придатну до використання в реальному середовищі систему. Основні функціональні вимоги до системи прокату автомобілів узагальнено в табл. 1.2.

Таблиця 1.2

Функціональні вимоги до системи прокату автомобілів

№	Вимога	Призначення
1	Реєстрація та авторизація користувача	Ідентифікація клієнтів та захист персональних даних
2	Пошук і фільтрація автомобілів за параметрами	Забезпечення зручного вибору транспортного засобу
3	Бронювання автомобіля	Створення замовлення з фіксацією дати, часу та локації
4	Облік транспортних засобів	Ведення реєстру автопарку з технічними характеристиками
5	Генерація договору оренди та платіжного документа	Формалізація прокатної угоди та інтеграція з платіжною системою
6	Перегляд історії замовлень та статусу поточних бронювань	Надання користувачу доступу до особистої інформації

Функціональні вимоги формують основу логіки застосунку й визначають набір необхідних модулів та сценаріїв взаємодії. У свою чергу, нефункціональні

вимоги описують характеристики, що забезпечують надійність, ефективність та відповідність сучасним технологічним стандартам, включаючи масштабованість, безпеку, інтерфейсні вимоги та обмеження на обробку даних. Систематизований перелік таких вимог наведено в табл. 1.3.

Таблиця 1.3

Нефункціональні вимоги до програмного забезпечення

№	Категорія	Вимога
1	Продуктивність	Система повинна обробляти до 100 одночасних запитів без зниження швидкодії
2	Доступність	Інтерфейс має коректно відображатися на пристроях із різним розширенням екрана
3	Безпека	Усі персональні дані мають зберігатися у зашифрованому вигляді
4	Надійність	Після критичних помилок система повинна автоматично відновлювати збережені сесії
5	Масштабованість	Архітектура має підтримувати розширення без повної перебудови коду
6	Юзабіліті	Інтерфейс має бути інтуїтивно зрозумілим для користувачів без технічної підготовки

Загальна сукупність вимог також формується на основі бізнес-цілей, очікуваної моделі використання та реальних сценаріїв експлуатації. Для системи прокату автомобілів це означає підтримку не лише базових операцій, а й додаткових функцій, що підвищують конкурентоспроможність сервісу, таких як система сповіщень, інтеграція з API платіжних шлюзів або гнучке управління тарифами залежно від попиту. Водночас для забезпечення логіки доступу до функцій системи відповідно до ролі користувача доцільно використовувати ролеву модель взаємодії, яка представлена в табл. 1.4.

Таблиця 1.4

Розподіл функціональності за ролями користувачів

Роль	Доступні дії
Гість	Перегляд автопарку, реєстрація, ознайомлення з умовами
Клієнт	Бронювання, перегляд статусу, оплата, перегляд історії
Менеджер	Облік транспорту, підтвердження заявок, генерація договорів
Адміністратор	Керування користувачами, зміна тарифів, формування звітності

Формалізація вимог на ранньому етапі проєктування є необхідною умовою створення структурованої та стійкої інформаційної системи. Ретельно сформульовані функціональні, нефункціональні та ролеві вимоги дозволяють мінімізувати ризики у процесі реалізації, забезпечити цільову поведінку системи та спростити подальше тестування й обслуговування програмного забезпечення. Отримані результати цього етапу стануть основою для побудови логічної моделі системи та реалізації архітектурної структури відповідно до заданих обмежень і технічних умов.

1.3 Моделювання предметної області

Моделювання предметної області є необхідним етапом формалізації вимог до інформаційної системи, що дозволяє наочно представити основні функції, учасників процесу, а також взаємодію між підсистемами в межах обраної архітектури. Для системи прокату автомобілів модель предметної області формувалась із використанням стандартних нотацій UML, що дозволяють зафіксувати логіку взаємодії користувачів із системою, послідовність операцій, ключові бізнес-процеси та структуру реалізації. Такий підхід сприяє однозначному тлумаченню функціональних сценаріїв під час реалізації ПЗ.

На першому етапі було побудовано діаграму варіантів використання (use case diagram), що відображає основні ролі користувачів і відповідні їм функції системи. У моделі виділено три основних актори: клієнт, менеджер та адміністратор. Клієнт має доступ до функцій реєстрації, авторизації, перегляду автомобілів, створення замовлення та ініціації повернення транспортного засобу. Менеджер, у свою чергу, здійснює обробку замовлень та підтвердження повернення автомобіля. Адміністратор виконує управління автопарком і формує звітність за результатами роботи системи. Діаграма варіантів використання подана на рис. 1.1.

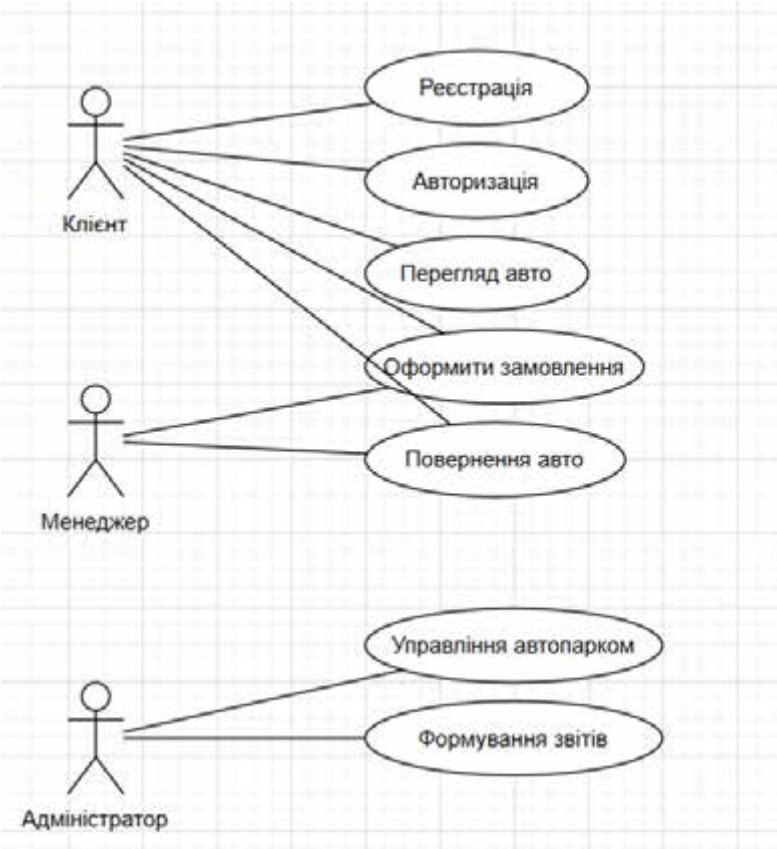

Рис. 1.1 Діаграма варіантів використання інформаційної системи прокату автомобілів

З метою деталізації динаміки взаємодії між об’єктами системи розроблено діаграму послідовності (sequence diagram), що ілюструє повний цикл операції бронювання автомобіля. У ній послідовно відображено дії користувача з моменту відкриття програми, авторизації, пошуку авто, введення деталей бронювання, перевірки доступності автомобіля, створення бронювання, формування договору оренди та відображення результатів. Комунікація між учасниками процесу реалізована через компоненти графічного інтерфейсу, контролера, модуля бронювання, бази даних і генератора договору. Такий сценарій забезпечує цілісне уявлення про структуру викликів і залежностей у системі. Діаграма наведена на рис. 1.2.

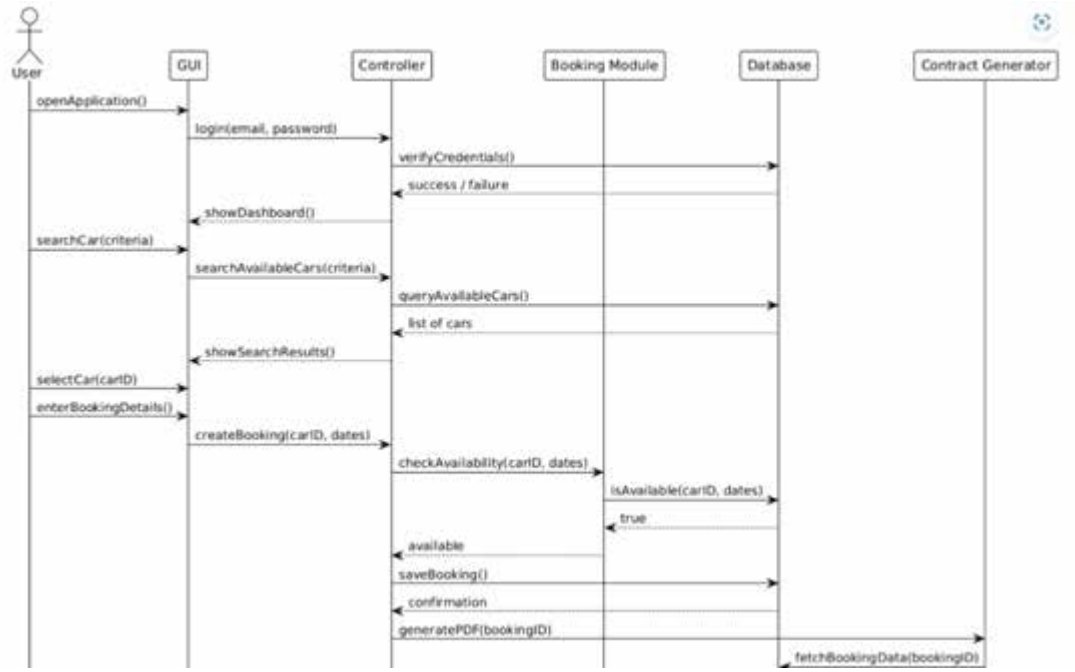

Рис. 1.2 Діаграма послідовності взаємодії користувача з системою бронювання

Окремий бізнес-процес у сфері прокату – це повернення автомобіля, що передбачає перевірку ідентифікатора замовлення, технічного стану транспортного засобу, розрахунок остаточної вартості та підтвердження завершення договору. Для моделювання цього процесу використано діаграму активності (activity diagram), що дозволяє наочно представити логіку виконання дій із відповідною послідовністю. У результаті моделювання було виділено чотири ключові етапи: введення коду замовлення, перевірка стану автомобіля, розрахунок вартості оренди, підтвердження завершення. Завершенням процесу є фіксація повернення автомобіля в базі даних. Вказана логіка представлена на рис. 1.3.

Таким чином, побудовані UML-діаграми дозволяють формалізувати модель поведінки системи, уточнити вимоги до її функціональних компонентів і закласти основу для архітектурного проєктування. Візуальні моделі дають змогу узгодити очікування користувача із внутрішньою логікою ПЗ та забезпечити відповідність між специфікацією вимог і фактичною реалізацією системи.

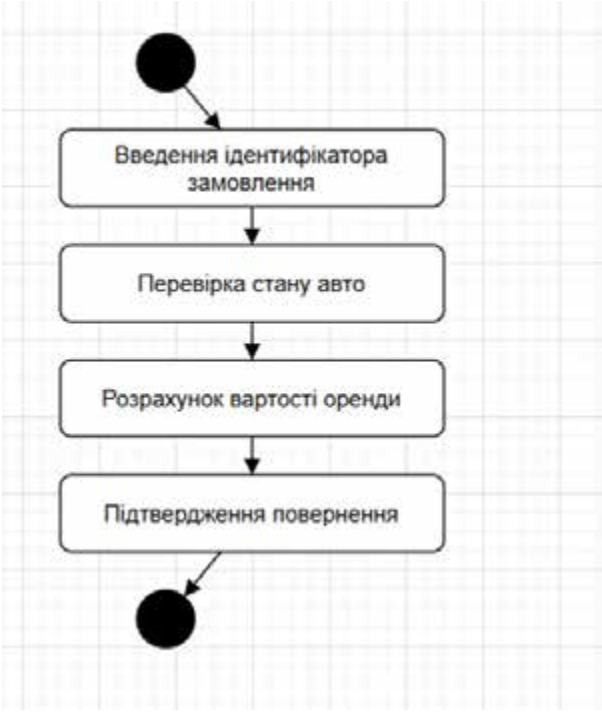

Рис. 1.3 іаграма активності процесу повернення автомобіля після прокату

1.4 Огляд інформаційних джерел та існуючих рішень

У процесі дослідження предметної області програмного забезпечення для прокату автомобілів особливу увагу приділено вивченню існуючих рішень, що вже реалізовані у вигляді повнофункціональних вебплатформ і мобільних застосунків. Більшість таких сервісів поєднує компоненти пошуку автомобіля за параметрами, бронювання, онлайн-оплати, керування угодами та підтримки клієнтів. Аналіз функціональності, архітектури та інтерфейсної реалізації сучасних платформ дозволяє виокремити ключові підходи до побудови інформаційних систем даного типу та визначити найефективніші практики

Більшість сучасних систем реалізують адаптивний UI згідно принципів Material Design [24], що забезпечує доступність на різних платформах. На практиці це реалізовано у сервісах, таких як RentalCars [26], Hertz [27], Sixt [28] та Localrent [29]. Платформа RentalCars позиціонується як агрегатор, що дозволяє порівнювати пропозиції від численних локальних і глобальних компаній, забезпечуючи доступ до фільтрів за категорією автомобіля, умовами

оренди, типом трансмісії, політикою пального та умовами скасування. Інтерфейс системи реалізовано з орієнтацією на зручність взаємодії користувача з великою кількістю опцій у межах єдиного вікна перегляду (рис. 1.4). Система підтримує адаптивний дизайн та інтеграцію з email- і push-сповіщеннями.

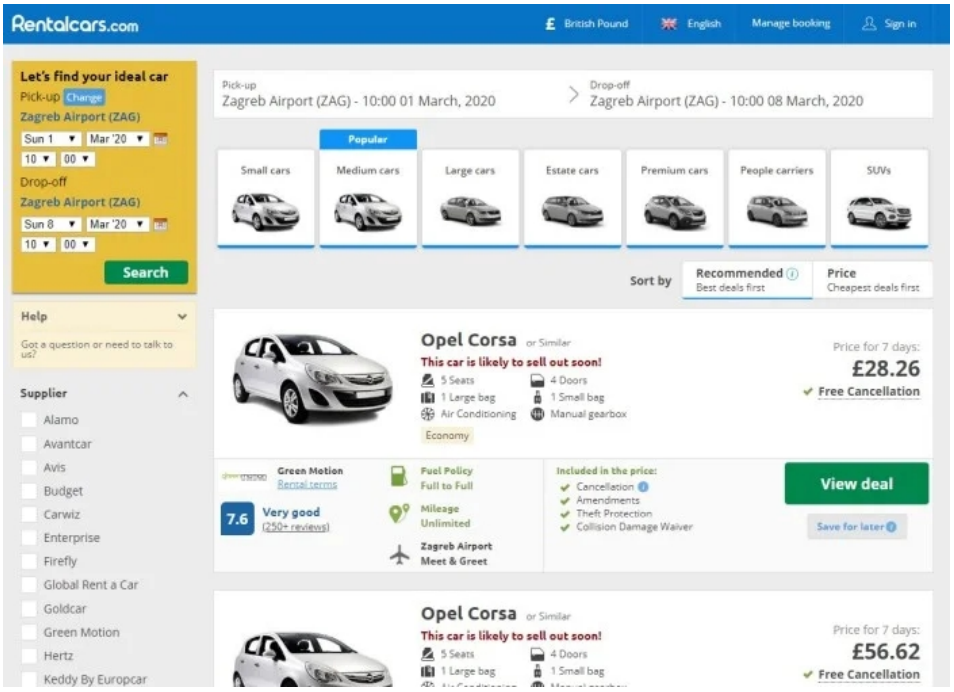

Рис. 1.4 RentalCars інтерфейс

Сервіс Hertz (рис. 1.5), як представник класичного бізнес-моделі автопрокату, орієнтований на корпоративних клієнтів і включає розширену підтримку бонусних програм, GPS-моніторинг транспорту та аналітичні звіти щодо завантаженості автопарку.

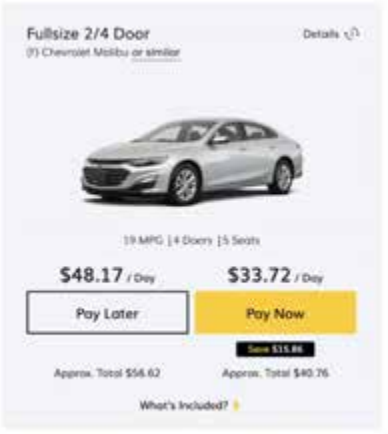

Рис. 1.5 Інтерфейс сервісу Hertz

Його функціональні компоненти включають CRM-модуль, персоніфіковану історію бронювань і диференційовані тарифи для постійних клієнтів. Платформа здебільшого працює в десктопному середовищі, але також має адаптовану мобільну версію для оперативного доступу.

Окрему увагу привертає рішення Sixt, яке поєднує сучасний візуальний стиль із гнучкою логікою підбору та керування автомобілями. Головною особливістю цього сервісу є географічна масштабованість: клієнт може здійснити бронювання транспортного засобу в одній країні та повернути його в іншій, що потребує складної внутрішньої логіки обробки замовлень, розрахунків і логістики. Інтерфейс Sixt (рис. 1.6) реалізовано на базі елементів із швидким доступом до форми пошуку, перегляду популярних категорій і персоніфікованих пропозицій, що свідчить про високу ступінь оптимізації користувацького досвіду.

Рис. 1.6 Інтерфейс сервісу Sixt

Ще одним прикладом є Localrent, платформа, орієнтована на локальні автопрокати в туристичних регіонах. Вона дозволяє напряму взаємодіяти з невеликими компаніями без участі посередників. Такий підхід дозволяє реалізовувати мінімалістичний та швидкий інтерфейс, що фокусується на простому бронюванні та базових функціях взаємодії з клієнтом. У цьому випадку особливу увагу приділено простоті UI, що забезпечує мінімальний час взаємодії

з системою й ефективне завантаження на мобільних пристроях навіть за умов обмеженого інтернет-з’єднанн, головна сторінка сервісу представлена на рис.1.7.

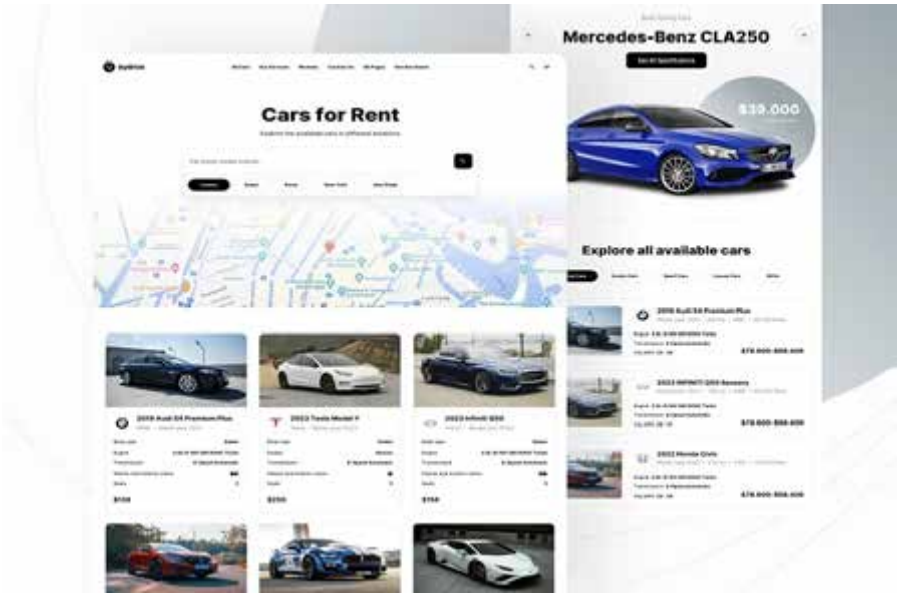

Рис. 1.7 Головна сторінка сервісу Locarent

Варто відзначити, що для всіх зазначених платформ характерна наявність власних підходів до побудови бази даних, інтерфейсних структур і бізнес-логіки. У більшості випадків архітектура системи будується за принципом поділу на фронтенд, бекенд та інтеграційний рівень, що дозволяє масштабувати функціонал без порушення цілісності системи. Аналізуючи функціональність, логіку побудови GUI та архітектурні рішення зазначених систем, можна виокремити перелік найважливіших характеристик, які буде враховано під час проєктування власного програмного забезпечення, а також сформулювати вимоги до зручності, прозорості й доступності інтерфейсу на різних платформах.

Для більш повного та об’єктивного аналізу функціональних і інтерфейсних можливостей існуючих рішень доцільно перейти до розгляду порівняльної таблиці, яка узагальнює ключові характеристики платформ оренди автомобілів. Такий підхід дозволяє систематизувати дані, виявити спільні риси та відмінності між сервісами, а також сформувати підґрунтя для подальшого формулювання вимог до розроблюваного програмного забезпечення. У таблиці 1.5 враховано такі аспекти, як структура інтерфейсу, наявність адаптивного дизайну, підтримка

платіжних модулів, механізми пошуку та фільтрації, а також загальна зручність користування з погляду кінцевого споживача.

Таблиця 1.5

Порівняльна характеристика платформ прокату автомобілів

Характеристика	RentalCars	Hertz	Sixt	Localrent
Тип рішення	Агрегатор	Компанія	Компанія	Платформа-посередник
Орієнтація	Масовий клієнт	Корпоративний	Туризм / Бізнес	Туризм / Локальний
Адаптивність інтерфейсу	Висока	Середня	Висока	Висока
Мобільна версія / застосунок	Є	Є	Є	Легка веб-версія
Пошук та фільтри	Розширені	Стандартні	Розширені	Базові
Підтримка багатомовності	Є	Є	Є	Частково
Онлайн-оплата	Є	Є	Є	Залежить від партнера
Звітність / історія замовлень	Обмежена	Детальна	Детальна	Обмежена
Гнучкість налаштувань угоди	Середня	Висока	Висока	Середня
Географічне покриття	Глобальне	Глобальне	Глобальне	Локальне

Як видно з таблиці, кожна з платформ реалізує власну стратегію побудови інформаційної системи з урахуванням бізнес-моделі, цільової аудиторії та технічних можливостей. RentalCars робить акцент на масштабності та зручності вибору з-поміж десятків компаній, тоді як Hertz і Sixt реалізують більш спеціалізовані й керовані середовища. У свою чергу, Localrent вирізняється простотою та доступністю, що є критично важливим для туристичного сегмента. Виявлені характеристики й відмінності будуть враховані при формуванні архітектури й функціонального складу програмного забезпечення, розробка якого здійснюється в межах даного дослідження.

1.5 Постановка завдання

На основі проведеного аналізу предметної області, порівняння існуючих рішень і сервісів, а також результатів формалізації вимог до системи, сформульовано технічне завдання на створення інформаційної системи для автоматизації процесу прокату автомобілів. Метою системи є забезпечення зручного, безпечного та ефективного середовища для взаємодії між користувачами та прокатним сервісом, з реалізацією повного життєвого циклу оренди – від пошуку автомобіля до завершення договору й обробки звітної інформації.

Інформаційна система повинна функціонувати локально або на внутрішньому сервері організації без потреби в зовнішніх API чи хмарних платформах. Це забезпечує підвищений рівень автономності, захист персональних даних і контроль над роботою системи. Основні вхідні параметри для функціонування системи охоплюють відомості про клієнтів, транспортні засоби, параметри оренди, календарні обмеження, тарифні плани, платіжні реквізити та конфігурацію угоди. Структуру вхідних даних наведено в табл. 1.6.

Таблиця 1.6

Основні вхідні параметри системи прокату

Категорія	Параметри
Дані користувача	Ім’я, email, телефон, паспортні дані, статус облікового запису
Інформація про авто	Назва, марка, модель, категорія, реєстраційний номер, статус доступності
Деталі замовлення	Дата початку і завершення, місце отримання та повернення, вартість
Тарифний план	Щоденна вартість, страховка, заставна сума, знижки
Платіжна інформація	Тип оплати, дата, статус, сума, платіжна сесія
Календар доступності	Графік бронювання автомобіля, тимчасові обмеження

На основі цих параметрів система повинна виконувати автоматизовану обробку запиту користувача, перевірку доступності авто, формування прокатної угоди, розрахунок вартості з урахуванням тарифів і знижок, а також збереження результатів у внутрішній базі даних. Ключовим аспектом є надання користувачу зворотного зв’язку у вигляді звітності, статусу замовлення, підтверджень і повідомлень. Очікувані результати функціонування системи узагальнено в табл. 1.7.

Таблиця 1.7

Очікувані результати роботи системи

Компонент результату	Опис
Підтвердження замовлення	Генерація унікального коду замовлення та фіксація в БД
Договір оренди	Структурований PDF-документ з деталями клієнта, авто та умовами
Фінансовий звіт	Інформація про оплату, залишок, страхову суму, дату завершення
Статистика користування	Перелік завершених бронювань, рейтинг популярності авто
Повідомлення користувача	Сповіщення про статус, нагадування, попередження
Експорт результатів	Вивантаження угод у PDF, CSV, HTML

Архітектура інформаційної системи повинна передбачати модульну реалізацію з чітким поділом на функціональні компоненти: введення даних, логіка обробки замовлень, управління автопарком, генерація договорів, аналітика, формування звітності та інтерфейс користувача. Уся система реалізовується як десктопний застосунок на Python із графічним інтерфейсом на базі PyQt6. Вибір даної технології дозволяє створити локальну, стабільну, кросплатформну систему з високою швидкодією. Основні компоненти системи наведено в табл. 1.8.

Таблиця 1.8

Програмні компоненти системи прокату автомобілів

Компонент	Призначення
Модуль автентифікації	Реєстрація, вхід, перевірка прав доступу
Управління автопарком	Додавання, редагування, архівування автомобілів
Обробка замовлень	Пошук, перевірка наявності, бронювання, оновлення статусу
Генератор договорів	Формування друкованої угоди з даними замовлення
Платіжний модуль	Облік оплат, фіксація транзакцій, генерація квитанцій
Аналітичний модуль	Побудова статистичних звітів, історія операцій
Графічний інтерфейс (GUI)	Інтерактивне управління процесами з боку клієнта і менеджера

З технічної точки зору система реалізовується на основі архітектурної моделі MVC (Model–View–Controller), що дозволяє відокремити логіку бізнес-операцій, візуальне представлення та зберігання даних. Такий підхід забезпечує спрощене тестування окремих компонентів, гнучкість масштабування та підтримку життєвого циклу застосунку.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Формування логічної моделі даних є важливим етапом проєктування інформаційної системи, оскільки вона визначає основні сутності предметної області, зв’язки між ними та ключові атрибути. У межах системи прокату автомобілів модель побудована на основі аналізу бізнес-процесів та вимог до зберігання даних про клієнтів, автомобілі, бронювання, платежі, персонал та технічні огляди. Основна мета логічної моделі – забезпечити цілісність даних, уникнення надлишковості та підтримку нормалізованої структури. Як результат моделювання, розроблено ER-діаграму (Entity–Relationship), яка представлена на рис. 2.1.

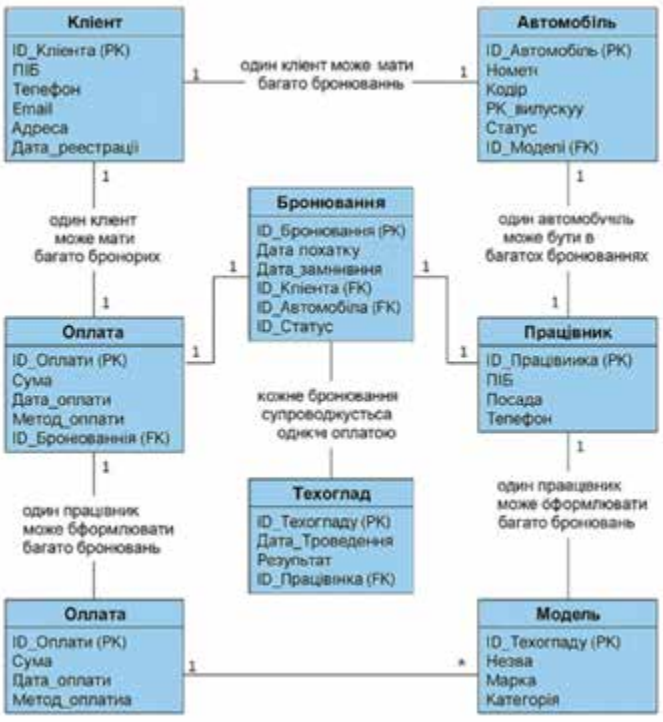

Рис. 2.1 Логічна модель даних системи прокату автомобілів (ER-діаграма)

На основі наведеної ER-моделі система охоплює вісім основних сутностей: Клієнт, Автомобіль, Модель, Бронювання, Оплата, Працівник, Техогляд та

Статус. Зв’язки між ними реалізовано як "один-до-багатьох" (1:N), що відповідає типовим сценаріям: один клієнт може створювати кілька бронювань, кожне бронювання стосується одного авто, одна оплата прив’язана до одного замовлення, а один працівник може здійснювати багато техоглядів. Для кожної сутності визначено первинні ключі, типові атрибути, а також зовнішні ключі, що забезпечують референційну цілісність. Структурований опис логічної моделі подано в табл. 2.1.

Таблиця 2.1

Сутності, атрибути та зв’язки логічної моделі даних

Сутність	Основні атрибути	Зв’язки
Клієнт	ID_Клієнта (PK), ПІБ, Телефон, Email, Адреса, Дата_реєстрації	1:N → Бронювання
Автомобіль	ID_Автомобіля (PK), Номер, Кодір, Рік_випуску, Статус, ID_Моделі (FK)	1:N → Бронювання
Модель	ID_Моделі (PK), Назва, Марка, Категорія	1:N → Автомобіль
Бронювання	ID_Бронювання (PK), Дата_початку, Дата_завершення, ID_Клієнта (FK), ID_Автомобіля (FK), ID_Статус	1:1 → Оплата, 1:N → Техогляд
Оплата	ID_Оплати (PK), Сума, Дата_оплати, Метод_оплати, ID_Бронювання (FK)	1:1 ← Бронювання
Працівник	ID_Працівника (PK), ПІБ, Посада, Телефон	1:N → Техогляд
Техогляд	ID_Техогляду (PK), Дата_проведення, Результат, ID_Працівника (FK), ID_Бронювання (FK)	N:1 ← Працівник, N:1 ← Бронювання
Статус	ID_Статус (PK), Назва_статусу	1:N → Бронювання

Логічна модель забезпечує уніфіковану схему зберігання даних і покриває всі основні компоненти системи: облік клієнтів, реєстрацію транспортних засобів, фіксацію бронювань і оплат, призначення працівників до техоглядів та ведення статусів угод. Така структура повністю відповідає принципам третьої нормальної форми (3NF), що гарантує мінімізацію дублювання даних та покращує узгодженість під час виконання запитів.

Отримана ER-модель є основою для побудови фізичної структури бази даних, а також для створення модулів імпорту, пошуку, аналітики і звітності в

системі, що розробляється. У наступних підпунктах буде уточнено зв’язки між класами, компоненти програмної реалізації, а також структуру бази у форматі SQL-схеми.

2.2 Діаграма класів і кооперації

Для забезпечення об’єктно-орієнтованого підходу до проєктування програмного забезпечення побудовано діаграму класів, яка відображає основні логічні сутності системи, їх атрибути, методи та зв’язки. Вона слугує основою для формування внутрішньої структури коду та визначає функціональні залежності між об’єктами. Модель дозволяє формалізувати об’єкти предметної області, які будуть реалізовані у програмному середовищі, включаючи користувача, автомобіль, замовлення, оплату та керуючі компоненти системи. Структура діаграми класів наведена на рис. 2.2.

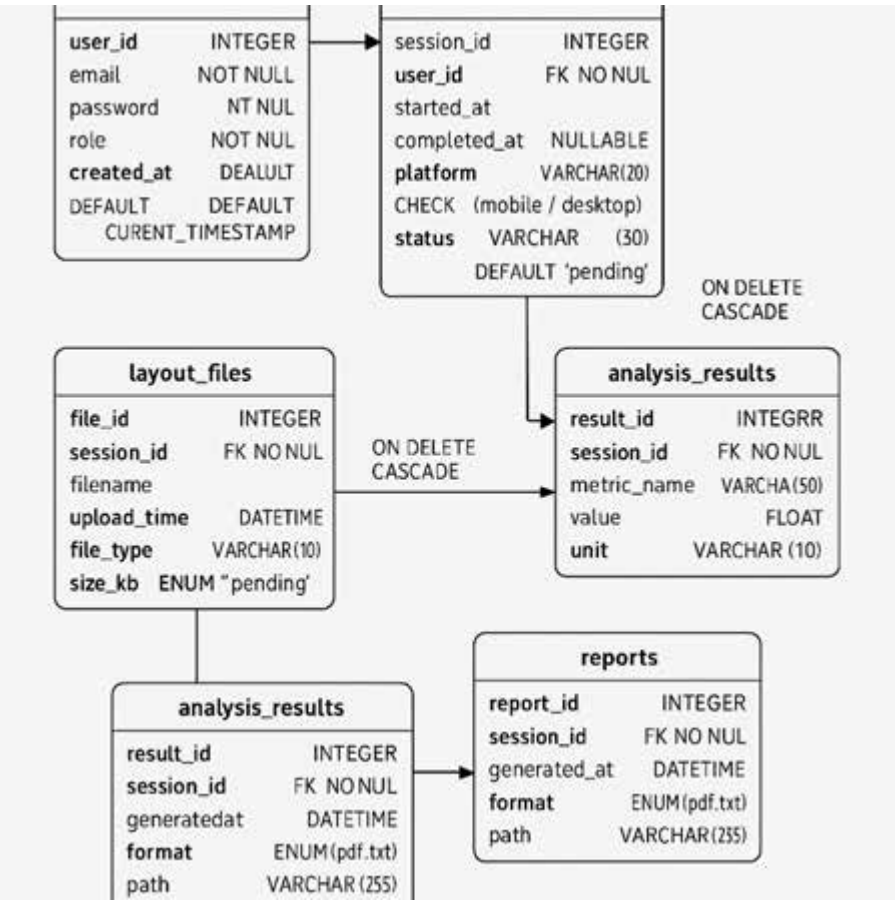

Рис. 2.2 UML-діаграма класів системи прокату автомобілів

Діаграма класів охоплює основні об’єкти з атрибутами, відповідними до логічної моделі даних, включаючи User, Car, Reservation, Payment, Controller, Form. Клас User містить атрибути для зберігання персональної інформації та методи взаємодії із системою, зокрема аутентифікацію. Клас Reservation реалізує логіку створення та управління замовленням, тоді як клас Payment відповідає за обробку фінансових операцій. Клас Car містить технічні та ідентифікаційні характеристики автомобіля. Контролер ReservationManager координує взаємодію між компонентами та реалізує бізнес-логіку процесу бронювання. Кожен клас описано відповідними методами, що ілюструють поведінкові аспекти системи.

Для моделювання комунікації між об’єктами, що беруть участь у виконанні сценарію, побудовано діаграму кооперації (діаграму взаємодії об'єктів), яка фіксує обмін повідомленнями між інстанціями класів під час обробки запиту на бронювання автомобіля. У цій моделі акцент зроблено на реальних об’єктах: клієнт, форма введення, контролер, база даних і модуль оплати. Уся послідовність починається з вибору автомобіля клієнтом, перевірки доступності, створення замовлення, ініціації оплати та підтвердження результату, що завершуються виведенням повідомлення з підтвердженням. Взаємозв’язки реалізовано через нумеровані повідомлення у межах єдиного сценарію. Відповідна діаграма представлена на рис. 2.3.

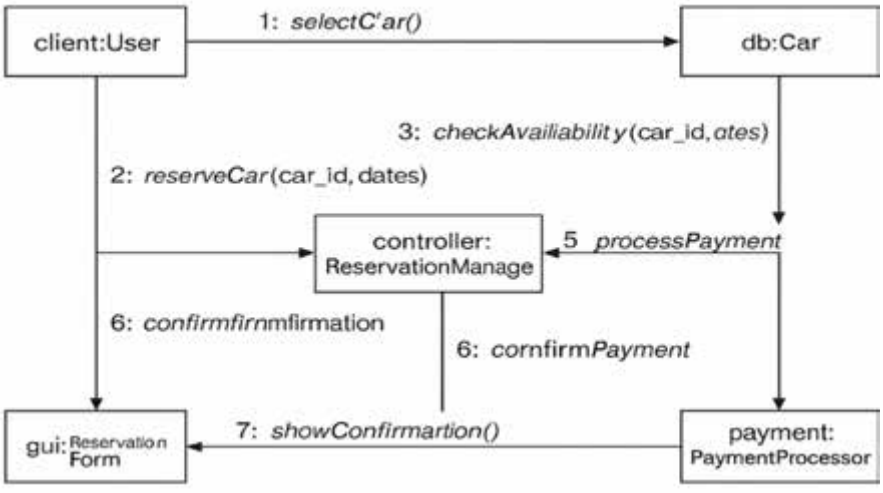

Рис. 2.3 UML-діаграма кооперації для сценарію бронювання авто

Як видно з діаграми, процес бронювання передбачає взаємодію щонайменше п’яти об’єктів. Користувач client:User ініціює вибір авто (selectCar()), після чого через контролер передається запит на резервування (reserveCar()). Контролер controller:ReservationManager викликає перевірку доступності у базі (checkAvailability()), а після цього ініціює оплату через об’єкт payment:PaymentProcessor. Завершенням процесу є підтвердження оплати та відображення повідомлення (showConfirmation()). Важливо, що всі зв’язки вказано з урахуванням напрямку взаємодії, що забезпечує точне розуміння залежностей між компонентами системи.

Загалом, побудовані діаграми класів і кооперації дозволяють створити чітке уявлення про логіку обробки основного бізнес-процесу системи. Вони забезпечують формальне підґрунтя для реалізації внутрішньої структури програмного забезпечення та слугуватимуть основою для реалізації модульного середовища у рамках архітектурного рішення.

2.3 Діаграма пакетів

У рамках модульного проєктування програмного забезпечення інформаційної системи прокату автомобілів було створено діаграму пакетів, що дозволяє формалізувати розподіл функціональності за логічними компонентами. Такий підхід забезпечує структурну декомпозицію системи, зручність масштабування та підтримки, а також сприяє відокремленню відповідальностей між модулями. Кожен пакет відповідає за певний аспект роботи системи та взаємодіє з іншими згідно визначених залежностей. Структура діаграми пакетів представлена на рис. 2.4.

Усього виділено сім логічних пакетів: ui, controller, analyzer, reporting, platform, utils та database (останній може бути реалізований у вигляді зовнішнього модуля або інтегрованого шару).

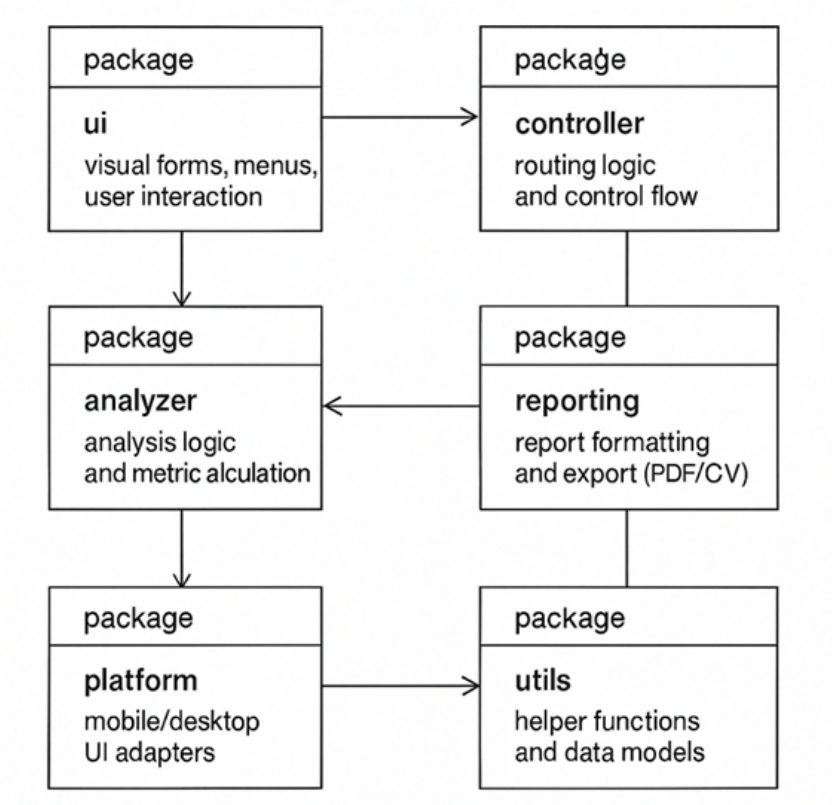

Рис. 2.4 Діаграма пакетів програмного забезпечення системи прокату автомобілів

Пакет ui відповідає за графічні форми, меню та взаємодію з користувачем. Він напряму пов’язаний із controller, який реалізує логіку маршрутизації, обробку подій та керування даними. Пакет controller, у свою чергу, координує взаємодію з пакетом analyzer, що реалізує логіку перевірки параметрів бронювання, доступності авто, а також моделі розрахунку тривалості та вартості оренди.

Пакет reporting отримує оброблені дані з analyzer і відповідає за форматування результатів, створення звітів, підтверджень та договорів у форматах PDF або CSV. Пакет platform забезпечує підтримку адаптерів для десктопних і мобільних середовищ, дозволяючи відображати інтерфейс на різних пристроях із врахуванням специфіки платформи. Пакет utils містить допоміжні функції, утиліти та структури даних, які використовуються іншими модулями системи для реалізації повторюваних операцій.

Залежності між пакетами вказані односторонніми стрілками, що демонструють напрям імпорту або використання. Наприклад, ui залежить від controller, а analyzer – від utils і platform, що забезпечує гнучкість обчислень і логіки для різних типів клієнтських середовищ. Така структура підтримує принципи слабкого зв’язку та високої когезії, що є бажаними для довготривалих проєктів з потенціалом розширення.

У сукупності, діаграма пакетів дозволяє сформувати уявлення про логічну архітектуру застосунку та забезпечити контроль над межами модулів при реалізації та інтеграції компонентів системи. Вона слугує основою для наступного етапу – побудови компонентної та розгортальної архітектури, які деталізують фізичну реалізацію програмних частин.

2.4 Діаграма компонентів

У рамках структурного проєктування програмного забезпечення системи прокату автомобілів розроблено діаграму компонентів, яка ілюструє взаємозв’язки між основними функціональними модулями. Дана діаграма дозволяє наочно представити архітектуру застосунку, визначити залежності між компонентами, а також логічно розмежувати обов’язки частин системи за рівнями: клієнтський інтерфейс, бізнес-логіка, доступ до даних та зовнішні бібліотеки. Такий підхід забезпечує модульність, спрощує тестування, масштабування та супровід програмного продукту. Структуру архітектурної моделі наведено на рис. 2.5.

Діаграма компонентів охоплює чотири рівні архітектури: клієнтський рівень (Client Application), бізнес-логіка (Business Logic), рівень доступу до даних (Data Layer) та зовнішні бібліотеки (External Libraries). На рівні клієнта функціонує GUI_Module та форма бронювання ReservationForm, які відповідають за збір вхідних даних, побудову екранів і взаємодію з користувачем. Вони передають керування основному контролеру ReservationController, що виступає ядром логіки системи та координує обробку запитів.

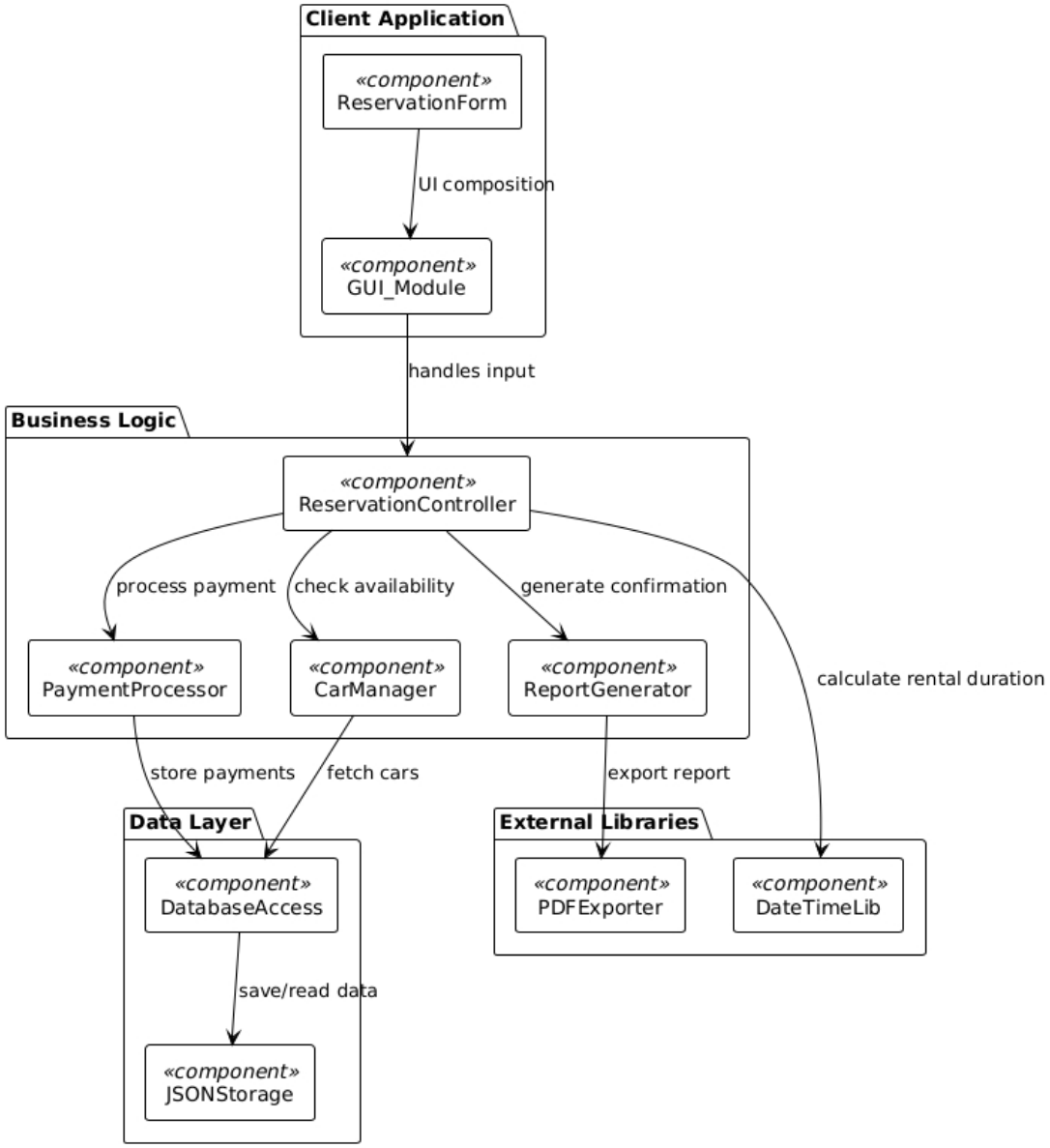

Рис. 2.5 Діаграма компонентів програмного забезпечення прокату автомобілів

Контролер взаємодіє з трьома ключовими підкомпонентами: CarManager (перевірка доступності авто), PaymentProcessor (обробка платежів) та ReportGenerator (формування договорів і підтверджень). Для доступу до даних всі компоненти логіки використовують DatabaseAccess, який, у свою чергу, взаємодіє з фізичним сховищем JSON-структур у модулі JSONStorage. Зовнішні залежності представлені модулями PDFExporter для створення звітів та DateTimeLib для обчислення дати початку та завершення оренди. Опис кожного компонента подано у табл. 2.2.

Таблиця 2.2

Опис компонентів системи прокату автомобілів

Компонент	Призначення
GUI_Module	Головний графічний інтерфейс для взаємодії з користувачем
ReservationForm	Форма для введення даних про бронювання автомобіля
ReservationController	Координує запити, перевіряє доступність авто, ініціює оплату, формує звіти
CarManager	Визначає доступні авто за критеріями дати і категорії
PaymentProcessor	Опрацьовує платіжні запити та зберігає інформацію про транзакції
ReportGenerator	Формує підтвердження та договір оренди у форматі PDF
DatabaseAccess	Інтерфейс доступу до структурованих даних у локальному сховищі
JSONStorage	Зберігає дані про клієнтів, авто, замовлення у JSON-форматі
PDFExporter	Генерує документи на основі отриманих параметрів замовлення
DateTimeLib	Визначає тривалість оренди, формат дати, перевірку коректності періодів

Запропонована модульна структура сприяє реалізації принципів слабкого зв’язку (loose coupling) і високої когезії (high cohesion), оскільки кожен компонент реалізує вузькоспеціалізовану функцію й взаємодіє із зовнішніми частинами виключно через визначені інтерфейси. Такий підхід дозволяє ізольовано тестувати й оновлювати окремі модулі, не впливаючи на решту частин системи.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Для зберігання, доступу та обробки структурованих даних у розроблюваній системі прокату автомобілів обрано файлову модель інформаційної бази на основі формату JSON, яка функціонує локально і не потребує підключення до зовнішнього серверного середовища. Такий підхід повністю відповідає вимогам до автономності програмного забезпечення, забезпечує конфіденційність персональних та транзакційних даних, і гарантує стабільну роботу у випадках, коли доступ до мережі відсутній або обмежений. На відміну від класичних клієнт-серверних моделей, файлово-орієнтоване сховище не вимагає налаштування серверів СУБД та дозволяє запустити систему без додаткових зовнішніх залежностей.

Основу зберігання становить модуль JsonStorage, який інкапсулює логіку взаємодії з файловою структурою. Він забезпечує збереження інформації про клієнтів, автомобілі, бронювання, оплату та звіти у вигляді окремих JSON-документів. Кожен запис містить повну інформацію про відповідну сутність: у випадку з бронюванням – це ID клієнта, автомобіля, дата початку, дата завершення, статус, сума та платіжний метод; у випадку з клієнтом – його ПІБ, контактні дані, історія оренд. Такий формат дозволяє легко серіалізувати/десеріалізувати об’єкти з боку бізнес-логіки системи та взаємодіяти з даними без потреби в SQL-запитах.

Обґрунтування вибору саме JSON-моделі подано в табл. 3.1, де наведено порівняння основних параметрів у контексті потреб даного проєкту.

Таблиця 3.1

Порівняння моделей збереження даних у системі

Критерій	JSON-файлова модель	Реляційна СУБД (SQL)
Підтримка структури	Ієрархічна, вкладені об’єкти	Таблична, нормалізована
Простота розгортання	Висока, не потребує окремого сервера	Потребує налаштування сервера та БД
Продуктивність у малих проєктах	Висока для локального застосунку	Надлишкова для нескладних сценаріїв
Підтримка автономності	Повна, можлива офлайн-робота	Обмежена, потрібна клієнт-серверна модель
Масштабованість	Обмежена, потребує зовнішніх механізмів	Висока, підтримка кластеризації та реплікації
Гнучкість структури	Висока, необов’язкові поля	Жорстка схема, обов’язкова типізація
Швидкість доступу	Висока при прямому доступі до файлів	Швидка при належному індексуванні

Як показує таблиця, для настільного застосунку з локальним зберіганням даних, відсутністю критичних навантажень та вимогою до офлайн-роботи обрана модель є доцільною. Вона дозволяє зберігати кожне бронювання або клієнтську сесію у вигляді окремого файлу, який може бути легко перенесений, скопійований чи відновлений у разі необхідності. Структура документів є ієрархічною, що забезпечує вбудовану підтримку вкладених об’єктів – наприклад, бронювання із вкладеною інформацією про оплату чи стан техогляду.

Уся логіка взаємодії із даними інкапсулюється в окремому класі JsonStorage, що реалізує CRUD-операції, фільтрацію та пошук. У разі масштабування системи або переходу до хмарної інфраструктури дана структура без змін може бути перенесена у документно-орієнтовану СУБД, таку як MongoDB, що дозволить зберегти сумісність та мінімізувати витрати на міграцію. Таким чином, вибір JSON-файлової моделі повністю узгоджується з архітектурною стратегією та вимогами до цільового середовища використання.

3.2 Розробка інформаційної бази

Інформаційна база системи виступає ключовим елементом зберігання, обробки та відтворення даних, що забезпечують функціонування сервісу прокату автомобілів. У відповідності до архітектурної концепції, викладеної у попередніх підпунктах, було обрано підхід до збереження даних у форматі JSON-документів, які логічно структуруються за типами об’єктів: користувачі, автомобілі, замовлення, оплати та сесії. Це дозволяє досягнути гнучкої, автономної та незалежної від серверного середовища моделі, що повністю узгоджується з критеріями застосування системи у локальному режимі. На рис.3.1 представлена фізична модель даних

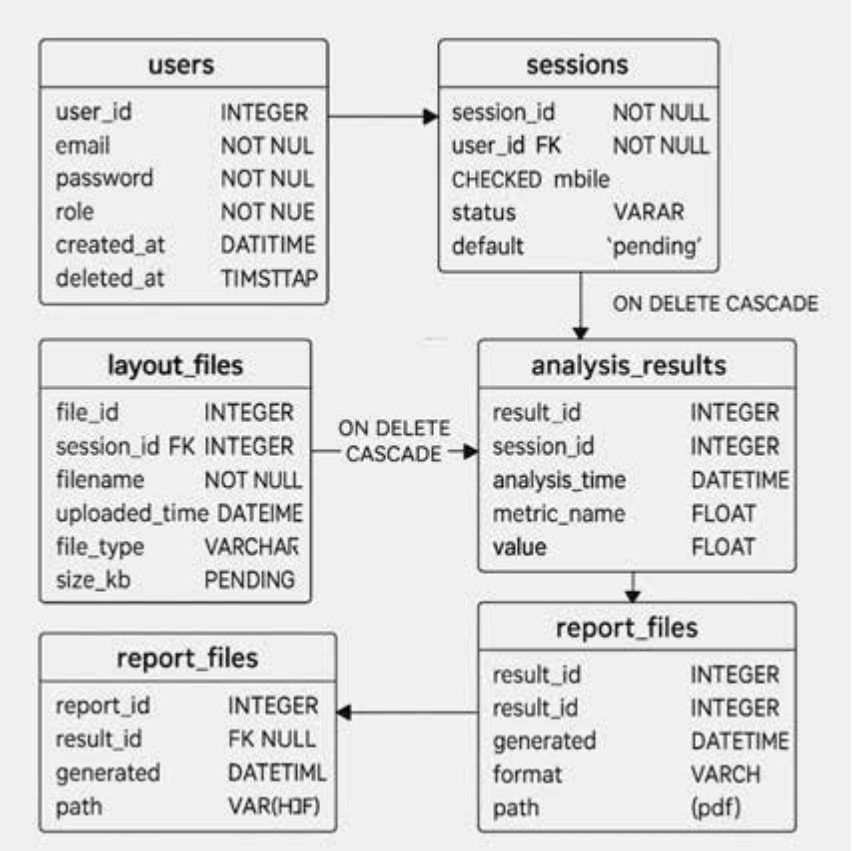

Рис. 3.1 Фізична модель даних

Фізична структура інформаційної бази формалізована у вигляді базової схеми, в якій представлено взаємозв’язки між логічними сутностями: users, sessions, layout_files, analysis_results, report_files. Всі таблиці реалізовані як

окремі JSON-файли, що зберігаються у локальному каталозі data/ і обробляються через централізовану систему доступу на основі функцій читання та запису. Згідно моделі, кожен користувач може створити кілька сесій, які пов’язуються з UI-файлами, що завантажуються, та результатами їх аналізу. На основі сесійних результатів формуються підсумкові звіти, які також мають відповідні метадані: час створення, формат та шлях збереження.

```
def load_json(file):
    with open(os.path.join(DATA_PATH, file), "r", encoding="utf-8") as f:
        return json.load(f)

def save_json(file, data):
    with open(os.path.join(DATA_PATH, file), "w", encoding="utf-8") as f:
        json.dump(data, f, indent=2, ensure_ascii=False)

@app.route("/api/cars", methods=["GET"])
def get_cars():
    cars = load_json("cars.json")
    return jsonify(cars)

@app.route("/api/register", methods=["POST"])
def register_user():
    users = load_json("users.json")
    new_user = request.json
    new_user["id"] = str(uuid.uuid4())
    users.append(new_user)
    save_json("users.json", users)
    return jsonify({"status": "registered", "user_id": new_user["id"]})

@app.route("/api/book", methods=["POST"])
def create_booking():
    bookings = load_json("bookings.json")
    new_booking = request.json
    new_booking["id"] = str(uuid.uuid4())
    new_booking["timestamp"] = datetime.utcnow().isoformat()
    bookings.append(new_booking)
    save_json("bookings.json", bookings)
    return jsonify({"status": "booking_confirmed", "booking_id":
new_booking["id"]})

@app.route("/api/pay", methods=["POST"])
def process_payment():
    payments = load_json("payments.json")
    new_payment = request.json
    new_payment["id"] = str(uuid.uuid4())
    new_payment["date"] = datetime.utcnow().isoformat()
    payments.append(new_payment)
    save_json("payments.json", payments)
    return jsonify({"status": "payment_success", "payment_id":
new_payment["id"]})

@app.route("/api/bookings/<user_id>", methods=["GET"])
def get_user_bookings(user_id):
    bookings = load_json("bookings.json")
    user_bookings = [b for b in bookings if b.get("user_id") == user_id]
```

Рис. 3.2 Структура логічної інформаційної бази системи у форматі JSON

Ключова взаємодія між даними реалізується через атрибути user_id, session_id, result_id, які виконують функцію зовнішніх ключів у межах відповідних об’єктів. Наприклад, об’єкт layout_files містить посилання на сесію через session_id, а об’єкт analysis_results посилається на ті ж самі сесії як джерело метрик. Така логіка збереження дозволяє реалізувати механізм каскадного видалення при очищенні історії або пов’язаних об’єктів. Кожна структура має мінімальний, але достатній набір атрибутів для забезпечення як обов’язкових, так і опціональних сценаріїв використання.

Уся обробка інформації реалізована в межах модуля обробки даних, побудованого за принципами REST-архітектури. Відповідні маршрути описані у прикладі коду, який демонструє функціонал реєстрації користувача, перегляду списку автомобілів, створення замовлення, обробки оплати та отримання історії бронювань. Кожна операція виконується через HTTP-запит, який транслюється у виклики методів зчитування або запису JSON-файлів. Усі об’єкти мають унікальні ідентифікатори, які генеруються на основі UUID, що забезпечує стабільність записів та їхню унікальність.

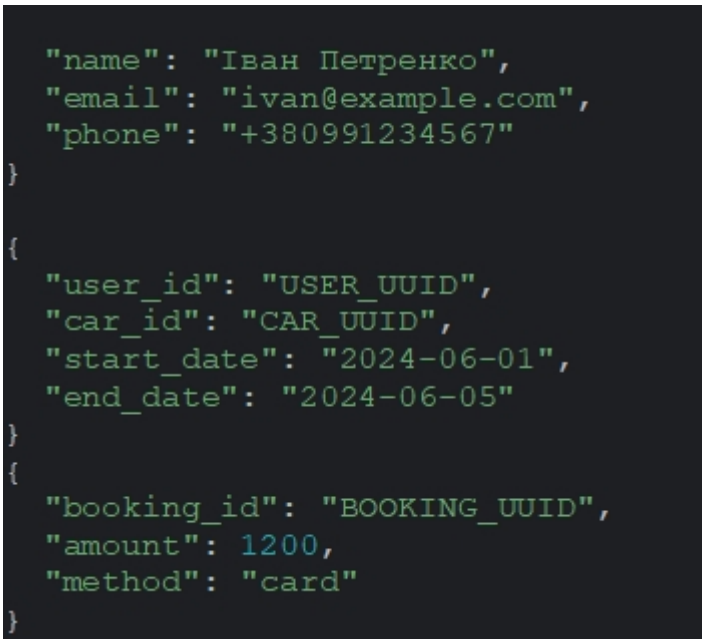

Рис. 3.3 Фрагмент реалізації обробки даних через REST-маршрути у Flask (JSON-інфраструктура)

Узагальнена структура сутностей, які формують основу інформаційної бази, подана у табл. 3.2. Кожен тип сутності реалізує власну модель представлення, яку інтерпретує бізнес-логіка системи.

Таблиця 3.2

Сутності та структури даних у JSON-інформаційній базі

Назва сутності	Ключові атрибути	Призначення
`users`	user_id, name, email, phone, created_at	Облік зареєстрованих користувачів
`cars`	car_id, brand, model, year, category, available	Опис автопарку, доступність до оренди
`bookings`	booking_id, user_id, car_id, start_date, end_date, status	Збереження замовлень користувача
`payments`	payment_id, booking_id, amount, method, date	Обробка фінансових транзакцій
`sessions`	session_id, user_id, platform, status, started_at, completed_at	Журнал активності та сесій роботи з інтерфейсами
`layout_files`	file_id, session_id, filename, uploaded_time, file_type	Збереження UI-макетів, завантажених користувачем
`analysis_results`	result_id, session_id, metric_name, value, analysis_time	Фіксація аналітичних результатів роботи з макетами
`report_files`	report_id, result_id, format, path, generated	Формування підсумкових звітів на основі аналізу

Завдяки такій структурі інформаційна база забезпечує централізоване зберігання усіх суттєвих даних, підтримує логічну цілісність системи та дозволяє розширення без порушення поточної архітектури. Подальша інтеграція з інтерфейсними або звітними модулями відбувається шляхом читання й обробки відповідних файлів, що зберігаються у підкаталогах із чіткою структурою. Такий підхід дозволяє підтримувати стабільну роботу системи навіть у середовищах з обмеженими ресурсами або вимогами до автономності.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Розробка прикладного програмного забезпечення для системи прокату автомобілів потребує ретельно обґрунтованого вибору технологічного стеку, який би забезпечував ефективну реалізацію як функціональних, так і нефункціональних вимог. Основними критеріями вибору інструментів виступали кросплатформність, гнучкість, підтримка інтерактивного графічного інтерфейсу, зручна робота з локальними даними у форматі JSON, можливість генерації звітів, а також доступність екосистеми для розширення. З урахуванням зазначених факторів було обрано мову програмування Python, бібліотеку PyQt6 для побудови GUI, стандартний модуль json для роботи з даними, фреймворк Flask для REST-маршрутів і бібліотеку reportlab для створення PDF-документів.

Вибрані інструменти забезпечують узгоджену архітектуру, зручність розробки, гнучке управління даними та надають широкі можливості для реалізації складної логіки обробки замовлень, фільтрації, пошуку та генерації підтверджувальних документів. Обґрунтований опис застосованого інструментарію подано в табл. 3.3.

Таблиця 3.3

Вибір інструментальних засобів для розробки програмного забезпечення

Інструмент / Технологія	Призначення	Обґрунтування використання
1	2	3
Python 3.x	Основна мова програмування	Висока читабельність, велика кількість бібліотек, кросплатформність
PyQt6	Побудова графічного інтерфейсу користувача (GUI)	Сучасний вигляд, гнучкий UI-дизайн, інтеграція з Python
JSON (модуль `json`)	Формат збереження даних та інтерфейс взаємодії з файловою базою	Простота, незалежність від СУБД, придатність для структурованих даних
Flask	Реалізація внутрішніх REST-маршрутів	Легка вага, швидке налаштування API, зручна інтеграція

Продовження таблиці 3.3

1	2	3
uuid, datetime	Генерація унікальних ідентифікаторів, мітки часу	Унікальність записів, точне фіксування дій користувача
reportlab / fpdf	Генерація договорів, чеків, звітів у форматі PDF	Можливість створювати структуровані документи без зовнішніх залежностей
os / pathlib	Робота з локальною файловою системою	Створення папок, перевірка наявності, управління шляхами до файлів

Застосування зазначеного інструментарію дозволяє реалізувати архітектурну модель, що базується на чіткому поділі логіки, представлення й обробки даних, з повним контролем над інтерфейсами взаємодії та відображенням даних. Завдяки використанню PyQt6, система має гнучкий і зручний інтерфейс, що дозволяє адаптацію під різні роздільності екрану. Обробка файлів JSON забезпечує автономність і збереження інформації у форматі, зручному для резервування та експорту. Flask забезпечує побудову внутрішнього API між модулями з можливістю масштабування у майбутньому. Генерація PDF-документів реалізована з урахуванням потреб юридичного підтвердження угод, що має ключове значення у системі прокату.

Обраний інструментарій дозволяє реалізувати всі функціональні вимоги до системи з високим рівнем гнучкості, підтримки локальної обробки та розширення функціоналу без необхідності зміни архітектури. У наступному підпункті буде подано опис структури компонентів системи та алгоритмічних рішень, які реалізуються у відповідних модулях.

3.4 Архітектура програмного забезпечення

Архітектура програмного забезпечення системи прокату автомобілів реалізована з урахуванням трирівневої моделі, що забезпечує поділ обов’язків між різними функціональними шарами. Такий підхід дозволяє досягти високої когезії всередині модулів і слабкого зв’язку між ними, що позитивно впливає на

масштабованість, підтримку та гнучкість при розширенні системи. Загальна архітектура застосунку подана на рис. 3.4.

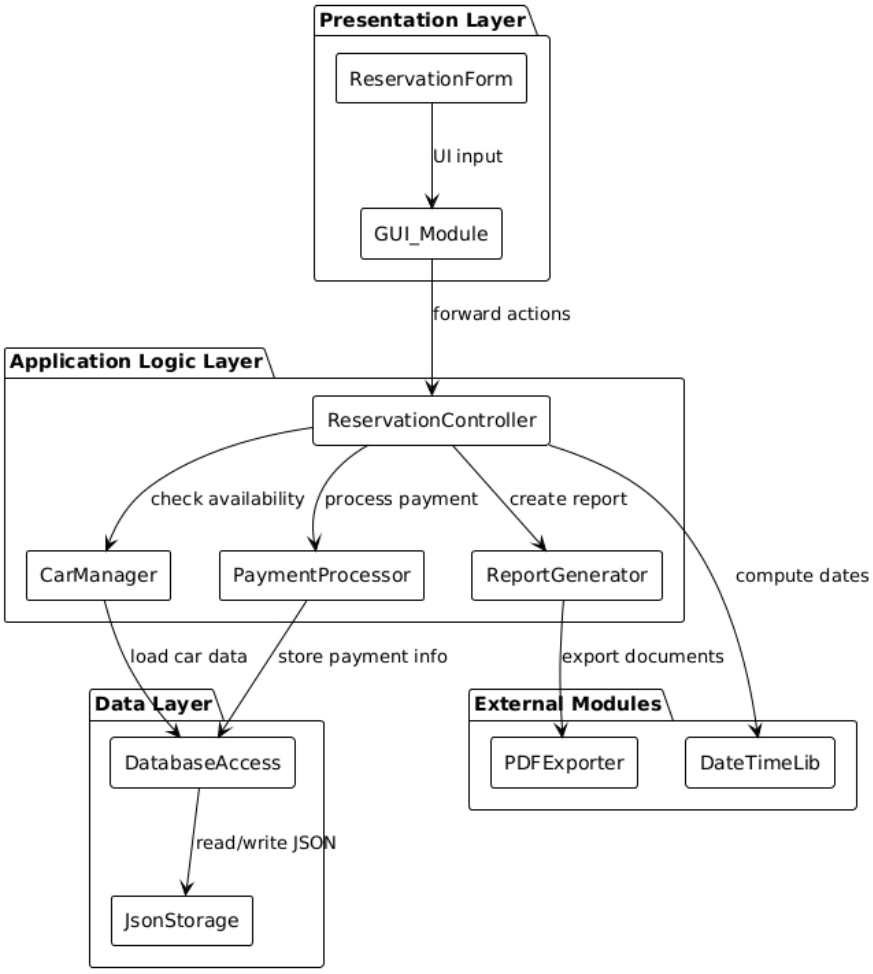

Рис. 3.4 Архітектура програмного забезпечення системи прокату автомобілів

На рівні Presentation Layer реалізовано модулі ReservationForm та GUI_Module, які відповідають за збирання вхідних даних, відображення стану системи та формування інтерфейсної логіки. Цей рівень є єдиною точкою взаємодії користувача із системою, а всі дії з GUI маршрутизуються до логіки через контролер.

Application Logic Layer виконує основну логіку обробки: ReservationController отримує події з інтерфейсу, ініціює перевірку наявності автомобіля через CarManager, обробку оплати через PaymentProcessor, а також формує звітні документи за допомогою ReportGenerator. Усі дії контролюються

централізовано, що дозволяє інкапсулювати бізнес-правила й гарантувати коректність обробки даних.

На рівні Data Layer реалізовано модулі DatabaseAccess та JsonStorage, які відповідають за читання, запис і фільтрацію даних у локальному файловому сховищі. Уся інформація зберігається у форматі JSON, що дозволяє зберігати дані у вигляді структурованих документів. Взаємодія з цим рівнем реалізована через єдиний інтерфейс, який забезпечує гнучке управління сутностями без прямої залежності бізнес-логіки від файлової структури.

На допоміжному рівні External Modules функціонують компоненти PDFExporter і DateTimeLib, які забезпечують створення PDF-документів і обробку часових параметрів. Таке винесення несуттєвих, але важливих для зручності компонентів, дозволяє дотримуватись принципів розділення відповідальностей.

Функціональна структура програмного забезпечення узагальнена в табл. 3.4, де представлено основні компоненти, їхній рівень інтеграції та призначення у межах загальної архітектури.

Таблиця 3.4

Основні компоненти програмної архітектури системи

Компонент	Рівень	Призначення
1	2	3
GUI_Module	Presentation Layer	Відображення вікон, обробка взаємодії з користувачем
ReservationForm	Presentation Layer	Збір параметрів замовлення (дата, авто, період)
ReservationController	Application Logic Layer	Централізоване управління логікою системи
CarManager	Application Logic Layer	Перевірка наявності та характеристик автомобіля
PaymentProcessor	Application Logic Layer	Опрацювання платіжних транзакцій
ReportGenerator	Application Logic Layer	Формування договору оренди та підтверджень у форматі PDF
DatabaseAccess	Data Layer	Абстракція над файловим доступом до JSON-бази

Продовження таблиці 3.4

1	2	3
JsonStorage	Data Layer	Реальне збереження й зчитування структурованих даних
PDFExporter	External Modules	Експорт звітних документів (чеків, договорів)
DateTimeLib	External Modules	Обчислення термінів оренди, перевірка дат

Представлена архітектура повністю відповідає вимогам до настільного програмного забезпечення з локальним зберіганням даних, забезпечує ізольованість логіки та підтримує можливість масштабування до мережевої версії системи без необхідності повного перепроєктування. Надалі архітектурна модель слугуватиме базою для реалізації алгоритмічної частини та впровадження модулів тестування.

3.5 Алгоритмізація та програмування програмних модулів

Процес реалізації функціональності програмного забезпечення розпочинається з формалізації алгоритмів, які відповідають за основні бізнес-процеси системи. З огляду на структуру системи прокату автомобілів, ключовими напрямами алгоритмізації стали: обробка замовлень, розрахунок вартості оренди, перевірка доступності автомобілів, генерація звітів, а також фіксація фінансових операцій.

Алгоритмізація здійснювалась із використанням базових структур управління (розгалуження, цикли, обробка винятків), при цьому особливу увагу приділено точності обробки дат і сум, відповідності алгоритмів логіці предметної області та узгодженості даних між модулями. Загальна схема алгоритму бронювання автомобіля представлена на рис. 3.5. На етапі реалізації програмні модулі структурувалися відповідно до MVC-підходу. У модулі controller.py зосереджено логіку обробки дій користувача, перевірки введених даних, ініціації запитів до бази та виклику генерації договору.

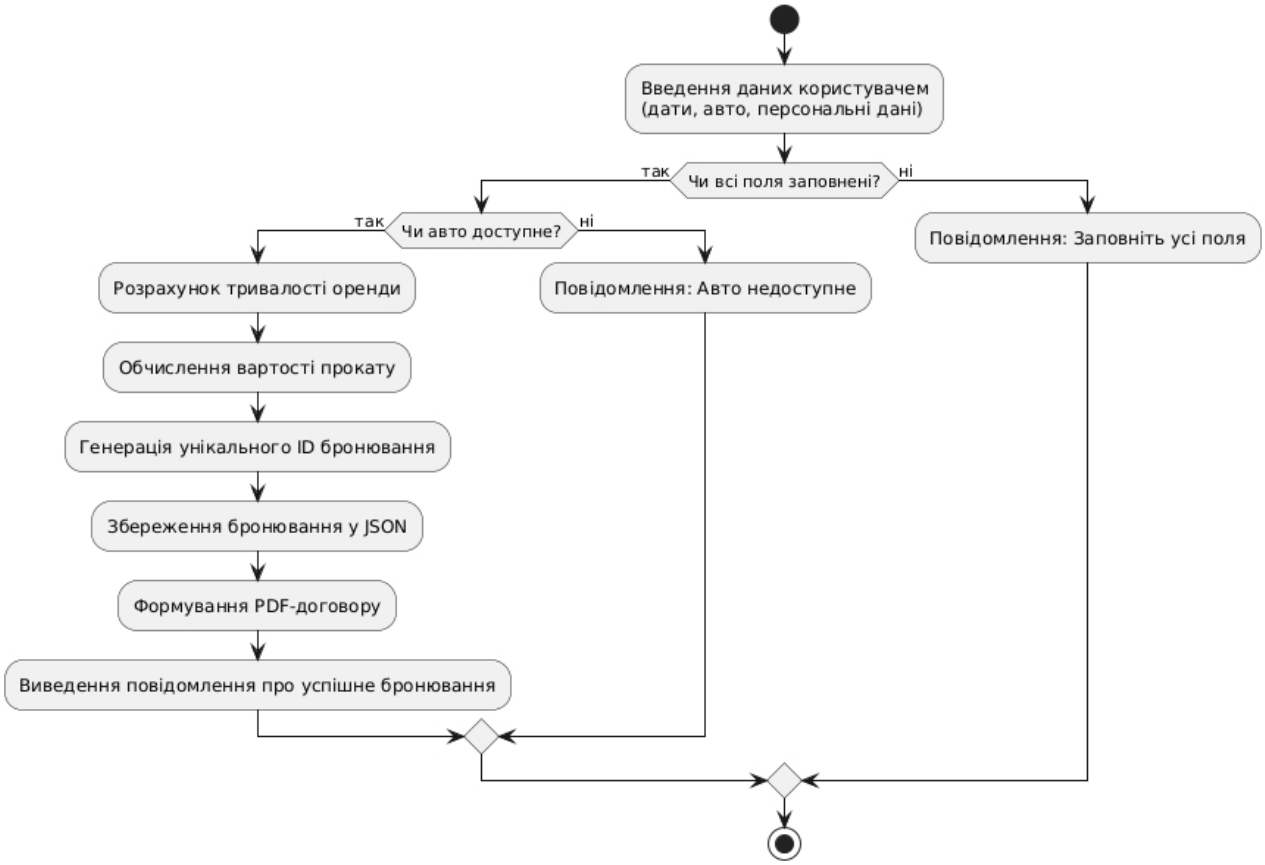

Рис. 3.5 Блок-схема алгоритму обробки бронювання

Зокрема, функція reserve_car() реалізує послідовність перевірки вхідних параметрів, викликає методи з класу JsonStorage, фіксує нове замовлення та повертає результат у вигляді підтвердження.

Важливим аспектом є реалізація обчислення вартості прокату. Для цього створено окрему функцію calculate_rental_cost(start_date, end_date, daily_rate), яка на основі вхідних дат і вартості за добу визначає тривалість оренди з точністю до одного дня, враховує граничні випадки (ідентичні дати, помилкове введення), та повертає підсумкову суму.

У модулі report_engine.py реалізовано формування PDF-документа, який містить усі деталі замовлення: дані клієнта, технічні параметри авто, строки оренди та суму. Застосовано бібліотеку ReportLab, яка дозволяє створювати документи із кастомізованими полями без потреби у сторонніх шаблонах.

Обробка файлів здійснюється у модулі json_storage.py, де функції read_data(file_name) та write_data(file_name, data) забезпечують взаємодію з JSON-документами, що зберігають інформацію про користувачів, транспортні засоби, платежі та замовлення. Структура файлів дозволяє здійснювати ефективний пошук, фільтрацію й оновлення записів.

Усі модулі зв’язано за допомогою контролера ReservationController, який виконує роль координатора між формами інтерфейсу та логікою обробки. Інтерфейс реалізовано у файлі gui.py на основі PyQt6 з використанням сигнально-слотової моделі. Це забезпечує динамічну реакцію системи на дії користувача, у тому числі валідацію введених даних, повідомлення про помилки та підтвердження успішних операцій.

На рівні взаємодії з користувачем передбачено механізми валідації, зокрема перевірка правильності формату дат, цілісності записів, запобігання повторному бронюванню недоступного авто. Результати обробки дій виводяться у вигляді повідомлень або інтерактивних форм з візуалізацією результатів.

Загалом, реалізовані програмні модулі забезпечують стабільну, логічно послідовну й автономну роботу системи. Всі алгоритми оптимізовані з урахуванням мінімального споживання ресурсів, високої читабельності коду й можливості розширення функціоналу без зміни існуючої архітектури. Такий підхід дозволяє адаптувати систему до ширшого спектра бізнес-сценаріїв і сприяє подальшому впровадженню у виробниче середовище.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

З метою підтвердження працездатності та відповідності функціональності програмного забезпечення вимогам технічного завдання було проведено тестування системи прокату автомобілів у контрольованому середовищі. Основна увага зосереджувалась на перевірці ключових функціональних модулів, зокрема створення бронювання, обробки оплати, формування звітів і візуалізації даних у графічному інтерфейсі. Тестування проводилось у локальному середовищі Windows з використанням PyQt6 як інтерфейсного рівня, JSON-файлів як джерела зберігання даних та імітованих сценаріїв взаємодії з користувачем.

Процес тестування передбачав створення нового бронювання, внесення основних параметрів (клієнт, авто, дата початку, дата завершення), обробку транзакції та підтвердження генерації звіту. Приклад успішно пройденого тестового кейсу представлено на рис. 4.1.

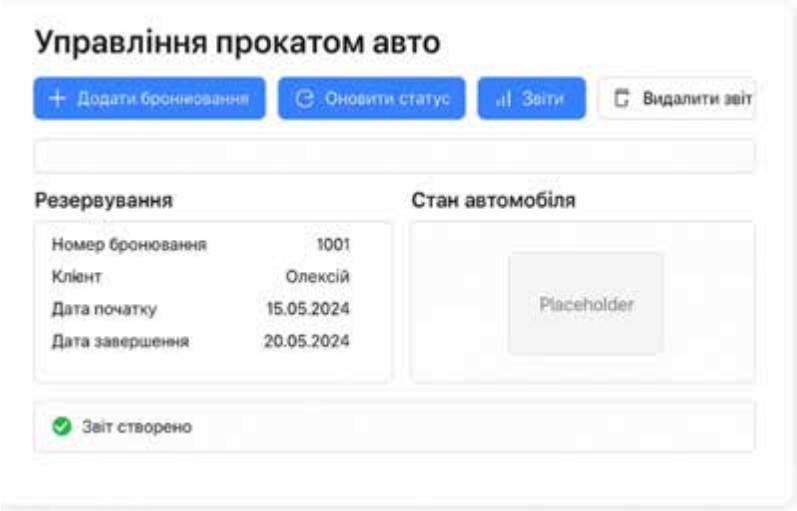

Рис. 4.1 Фрагмент графічного інтерфейсу після успішного бронювання і формування звіту

На рисунку 4.1 продемонстровано інтерфейс керування прокатом із заповненими полями, активним повідомленням про створений звіт та доступними кнопками для дій над замовленням.

Під час тестування також було перевірено правильність розрахунку загальної вартості оренди автомобіля, яка залежить від кількості днів прокату та встановленої ціни за добу. Для реалізації цієї функції використано модуль datetime, що дозволяє проводити точне обчислення різниці між датами початку та завершення замовлення. Функціональний код для цього розміщено на рис. 4.2.

```
from datetime import datetime
1 usage

def calculate_rental_cost(start_date, end_date, daily_rate):
    date_format = "%d.%m.%Y"
    start = datetime.strptime(start_date, date_format)
    end = datetime.strptime(end_date, date_format)
    days = (end - start).days + 1
    cost = days * daily_rate
    return days, cost

# Приклад використання
start = "15.05.2024"
end = "20.05.2024"
rate = 550  # грн/доба

d, total = calculate_rental_cost(start, end, rate)
print(f"Тривалість: {d} днів\nСума до сплати: {total} грн")
```

Рис. 4.2 Обчислення тривалості оренди та вартості прокату в межах клієнтської сесії

Код реалізує функцію calculate_rental_cost(), яка на основі вхідних дат та добової ставки повертає кількість днів і підсумкову суму до сплати. Усі змінні параметри вводяться у форматі dd.mm.yyyy, що відповідає стандартам локалізованої взаємодії з користувачем. Проведені тести підтвердили коректність розрахунків при різних вхідних значеннях та граничних випадках (однотипні дати, інверсія дат, високосний рік тощо).

Для верифікації функціоналу була застосована методика функціонального тестування з фокусом на цільові сценарії. Тестування проводилось вручну за тест-кейсами, що охоплюють ключові користувацькі дії. Підсумки тестування

зведено у табл. 4.1, яка містить перелік перевірених операцій, вхідні умови, очікувану поведінку системи та фактичний результат.

Таблиця 4.1

Результати тестування основного функціоналу системи

Назва тесту	Вхідні дані	Очікуваний результат	Статус
Створення бронювання	Клієнт: Олексій, Дати: 15.05–20.05.2024	Сформоване бронювання з унікальним ID	Пройдено
Генерація звіту	Замовлення з валідними параметрами	Повідомлення «Звіт створено», PDF-файл збережено	Пройдено
Оновлення статусу автомобіля	Авто після завершення бронювання	Стан оновлено, авто доступне для нового прокату	Пройдено
Видалення звіту	ID попередньо створеного звіту	Звіт видалено з файлової системи	Пройдено
Перевірка заповнення UI-даних	Поля клієнта, дати, статус	Усі значення виведено коректно, інтерфейс не зависає	Пройдено

Як видно з таблиці, усі ключові функції, які відповідають за бронювання, оновлення даних, формування документів і розрахунки, виконуються у відповідності до вимог і сценаріїв використання. Тестування підтвердило, що система перебуває у працездатному стані, виконує заявлений функціонал і може бути рекомендована для впровадження в умовах використання настільного середовища з локальною обробкою даних.

4.2 Впровадження системи

Процес впровадження системи прокату автомобілів передбачає інтеграцію програмного забезпечення у цільове середовище експлуатації з мінімальними технічними вимогами до обладнання та забезпеченням повної автономності. В основі архітектури впровадження лежить локальне середовище виконання Python-інтерпретатора на персональному комп’ютері користувача, у якому здійснюється завантаження та взаємодія з усіма модулями системи. Загальна схема розгортання програмного забезпечення подана на рис. 4.2.

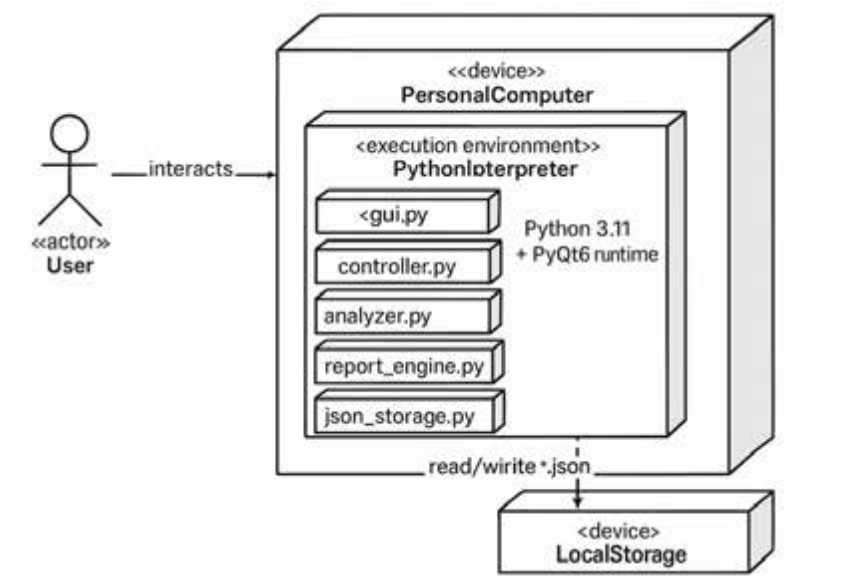

Рис. 4.3 UML-діаграма розгортання системи на локальному комп'ютері користувача

Система реалізована як десктопний застосунок з графічним інтерфейсом, побудованим за допомогою PyQt6, який виконується у середовищі Python 3.11. Всі функціональні модулі (зокрема gui.py, controller.py, analyzer.py, report_engine.py, json_storage.py) виконуються у межах єдиного процесу та взаємодіють між собою через імпортовані інтерфейси. Користувач взаємодіє із застосунком безпосередньо через інтерфейс, запускаючи його на своєму ПК у середовищі Windows, Linux або macOS. Система не потребує мережевого з'єднання або зовнішнього серверного ПЗ, що значно спрощує впровадження в умовах обмеженого технічного ресурсу або відсутності централізованої IT-інфраструктури.

Усі дані (користувачі, автомобілі, бронювання, звіти) зберігаються у структурованих JSON-документах у файловій системі комп'ютера. Модуль json_storage.py відповідає за безпосереднє зчитування та запис даних, реалізуючи логіку доступу до файлів, які знаходяться у підкаталозі data/. Такий підхід дозволяє швидко резервувати, копіювати або переміщати дані без участі зовнішніх БД, а також підтримувати ізоляцію даних одного користувача від іншого.

Під час впровадження важливу роль відіграє узгодження системних вимог з можливостями середовища користувача. Всі необхідні параметри і конфігурації узагальнено у табл. 4.2.

Таблиця 4.2

Системні вимоги та середовище впровадження програмного забезпечення

Категорія	Вимога або конфігурація
Операційна система	Windows 10/11, Ubuntu 20.04+, macOS 12+
Середовище виконання	Python 3.11 (або вище), інтерпретатор CPython
Графічний інтерфейс	PyQt6 (бібліотека для створення UI)
Сховище даних	Файлова система, структура папок `data/*.json`
Додаткові бібліотеки	`reportlab`, `uuid`, `datetime`, `os`, `json`
Права доступу	Читання/запис до локальної директорії
Інсталяція	Локальна, шляхом запуску `install.bat` або `python main.py`
Зовнішні залежності	Відсутні (не використовуються зовнішні сервери чи API)
Можливість резервного копіювання	Повна (копіювання каталогу з JSON-файлами)

Інтеграція системи у виробниче або навчальне середовище не вимагає специфічних знань або адміністрування – достатньо виконати попереднє встановлення Python та необхідних бібліотек, після чого запускати застосунок як звичайний виконуваний скрипт. Оновлення системи відбувається шляхом заміни окремих модулів, що зберігає її розширюваність та підтримку у майбутньому.

4.3 Склад інсталяційного пакету

Для забезпечення ефективного впровадження та простоти розгортання програмного забезпечення користувачем, створено інсталяційний пакет, який містить повний набір файлів, необхідних для запуску, налаштування та роботи системи прокату автомобілів. Усі компоненти згруповані у логічні підкаталоги з урахуванням архітектури, що передбачає відокремлення інтерфейсної, логічної та файлової частин. Інсталяційний пакет адаптований до роботи у середовищі

Windows, Linux або macOS із попередньо встановленим інтерпретатором Python версії 3.11 або вище.

Фізичне розгортання інсталяційного пакету полягає у розпакуванні архіву, який містить програмні модулі (*.py), підкаталог для зберігання JSON-файлів, конфігураційні файли, а також інструкції користувача. Додатково включено файл install_requirements.sh/.bat для автоматичного встановлення всіх залежностей. Структуру інсталяційного пакету наведено у табл. 4.3.

Таблиця 4.3

Склад інсталяційного пакету інформаційної системи

Назва компонента	Тип файлу/каталогу	Призначення
`main.py`	Python-скрипт	Точка входу в систему, ініціалізація GUI та контролера
`gui.py`	Python-модуль	Побудова графічного інтерфейсу за допомогою PyQt6
`controller.py`	Python-модуль	Обробка логіки взаємодії між модулями
`car_manager.py`	Python-модуль	Перевірка наявності та характеристик автомобілів
`payment_processor.py`	Python-модуль	Модуль обробки транзакцій, розрахунок вартості
`report_engine.py`	Python-модуль	Формування звітів і експорт у форматі PDF
`json_storage.py`	Python-модуль	Збереження та зчитування даних з JSON-файлів

Продовження таблиці 4.3

`data/`	Каталог	Файлова база: `users.json`, `cars.json`, `bookings.json` тощо
`requirements.txt`	Текстовий файл	Перелік бібліотек для автоматичного встановлення
`install.bat / install.sh`	Скрипт	Автоматичне встановлення залежностей у Windows / Linux
`README.txt`	Інструкція	Короткий посібник із запуску, системні вимоги, контакти

Інсталяційний пакет не потребує спеціалізованого інсталятора або інтерфейсу налаштування – весь процес запуску полягає у встановленні необхідних залежностей із requirements.txt, після чого система може бути запущена простим викликом python main.py. Усі дані зберігаються у каталозі data/, що дозволяє зберігати ізольовану історію сесій, бронювань та звітів.

У разі розширення функціоналу або внесення оновлень, оновлення системи здійснюється шляхом заміни відповідного модулю без впливу на інші компоненти. Така структура пакету забезпечує простоту обслуговування, гнучкість розробки та мінімальні вимоги до системного середовища користувача.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було розроблено прикладне програмне забезпечення для підтримки процесу прокату автомобілів із реалізацією повноцінного графічного інтерфейсу, локальної бази даних та генерації звітної документації. Розроблена система забезпечує автоматизацію ключових дій: створення бронювань, облік клієнтів, розрахунок вартості оренди, обробку платежів та збереження даних у форматі, придатному до подальшого аналізу чи експорту. Актуальність реалізованого рішення обумовлена зростанням попиту на індивідуальні засоби керування невеликими автопарками, а також потребою в інструментах, що не залежать від постійного підключення до мережі чи зовнішніх серверів.

У першому розділі було проаналізовано предметну область, сформульовано вимоги до функціональності системи, досліджено наявні підходи до реалізації подібних інструментів, а також описано логічну модель користувацьких сценаріїв. Результатом стало формування базових принципів проєктування, включно з побудовою моделі даних та визначенням ключових об'єктів системи. У другому розділі здійснено поетапне проєктування інформаційного й програмного забезпечення, включно зі створенням ER-діаграми, UML-діаграм класів, компонентів, кооперації, пакетів і фізичної архітектури системи.

У третьому розділі розглянуто реалізацію системи на практиці: створено файлову структуру на основі JSON-сховища, розроблено компоненти, що відповідають за керування прокатом, облік клієнтів, управління бронюваннями та формування фінансової звітності. Для реалізації використано стек технологій, до складу якого увійшли Python 3.11, PyQt6, ReportLab, Flask, що забезпечило кросплатформну сумісність та зручність розгортання. Окремо створено інсталяційний пакет, адаптований для запуску на персональному комп’ютері користувача.

У четвертому розділі проведено тестування системи у реальному середовищі з перевіркою базових сценаріїв використання: створення й редагування бронювань, розрахунок вартості оренди, обробка звітів, робота з файловим сховищем. Отримані результати підтвердили стабільність функціонування, коректність логіки та готовність системи до практичного застосування. Структура інтерфейсу й організація даних відповідають вимогам зручності, простоти та надійності, необхідним у прикладному середовищі.

Таким чином, у кваліфікаційній роботі реалізовано завершений цикл розробки інформаційної системи – від постановки задачі до її технічної реалізації та апробації. Отриманий результат демонструє потенціал для адаптації у сфері автоматизації малого бізнесу, локального управління автопарками, а також як навчально-прикладна платформа для вивчення практик розробки прикладного ПЗ. Створена система є гнучкою до розширення і може бути доопрацьована для інтеграції в більш складні інформаційні сервіси або хмарні платформи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Цивільний кодекс України : Закон України від 16 січ. 2003 р. № 435-IV [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/435-15
2. Про оренду державного та комунального майна : Закон України від 3 жовт. 2019 р. № 157-IX [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/157-20
3. Про захист персональних даних : Закон України від 1 черв. 2010 р. № 2297-VI [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2297-17
4. ДСТУ ISO/IEC 9126-1:2007. Інформаційні технології. Оцінювання якості програмного забезпечення. Частина 1. Модель якості. – [Чинний від 2008-01-01].
5. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.
6. ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection – Information security management systems – Requirements.
7. MySQL 8.0 Reference Manual [Електронний ресурс]. – Oracle, 2023. – Режим доступу: https://dev.mysql.com/doc/refman/8.0/en/
8. PostgreSQL Documentation. Version 15 [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/
9. REST API Tutorial [Електронний ресурс]. – Режим доступу: https://restfulapi.net/
10. Web Content Accessibility Guidelines (WCAG) 2.1 [Електронний ресурс]. – W3C, 2018. – Режим доступу: https://www.w3.org/TR/WCAG21/
11. Nielsen J. Usability Engineering. – San Francisco : Morgan Kaufmann, 1994. – 362 p.
12. Sommerville I. Software Engineering. 10th ed. – Boston : Pearson, 2016. – 792 p.

13. Шевченко Н. П. Проектування інформаційних систем : навч. посіб. – Харків : ХНУРЕ, 2021. – 228 с.
14. Гребенюк О. М. Розробка програмного забезпечення : навч. посіб. – Київ : КНЕУ, 2020. – 312 с.
15. Тищенко В. І. Основи проектування баз даних : навч. посіб. – Львів : ЛНУ імені Івана Франка, 2019. – 198 с.
16. Шаповал М. В. Хмарні технології в ІТ-інфраструктурі підприємств. – Дніпро : НГУ, 2020. – 196 с.
17. Марченко С. І. Захист інформації в комп’ютерних системах : навч. посіб. – Київ : НАУ, 2021. – 280 с.
18. Литвин В. В. Аналіз та моделювання систем. – Львів : Видавництво Львівської політехніки, 2019. – 325 с.
19. Гребенюк В. П. Програмна інженерія. – Київ : КНУ, 2021. – 344 с.
20. Dorsey P. Oracle PL/SQL Best Practices. – Sebastopol : O’Reilly Media, 2022. – 200 p.
21. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Boston : Addison-Wesley, 1995. – 395 p.
22. Пахомов В. В. Проєктування людино-машинних інтерфейсів. – Харків : УІПА, 2020. – 144 с.
23. Microsoft. ASP.NET Core Documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core
24. Google. Material Design Guidelines [Електронний ресурс]. – Режим доступу: https://m3.material.io
25. Федоренко Л. М. Основи побудови веб-систем : навч. посіб. – Тернопіль : ТНТУ, 2022. – 198 с.
26. RentalCars.com [Електронний ресурс]. – Режим доступу: https://www.rentalcars.com/
27. Hertz Global Holdings [Електронний ресурс]. – Режим доступу: https://www.hertz.com/
28. Sixt Rent a Car [Електронний ресурс]. – Режим доступу: https://www.sixt.com/

29. Localrent [Електронний ресурс]. – Режим доступу: https://localrent.com/
30. Oracle. Java Platform Standard Edition 17 Documentation [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/en/java/
31. GitHub Docs. Introduction to GitHub Actions [Електронний ресурс]. – Режим доступу: https://docs.github.com/en/actions
32. Apache HTTP Server Documentation [Електронний ресурс]. – Режим доступу: https://httpd.apache.org/docs/
33. Mozilla Developer Network (MDN). Web APIs Documentation [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/

ДОДАТОК А

API-додаток з використанням FLASK

```
import os
import json
import uuid
from datetime import datetime
from flask import Flask, request, jsonify

app = Flask(__name__)
DATA_DIR = "rent_data"
os.makedirs(DATA_DIR, exist_ok=True)

def get_path(filename):
    return os.path.join(DATA_DIR, filename)

def load_json(filename):
    path = get_path(filename)
    if not os.path.exists(path):
        with open(path, "w", encoding="utf-8") as f:
            json.dump([], f)
    with open(path, "r", encoding="utf-8") as f:
        return json.load(f)

def save_json(filename, data):
    with open(get_path(filename), "w", encoding="utf-8") as f:
        json.dump(data, f, indent=2, ensure_ascii=False)

@app.route("/api/register", methods=["POST"])
def register_user():
    users = load_json("users.json")
    user = request.json
    user["id"] = str(uuid.uuid4())
    users.append(user)
    save_json("users.json", users)
    return jsonify({"status": "ok", "user_id": user["id"]})

@app.route("/api/cars", methods=["GET", "POST"])
def manage_cars():
    if request.method == "GET":
        return jsonify(load_json("cars.json"))
    if request.method == "POST":
        cars = load_json("cars.json")
        car = request.json
        car["id"] = str(uuid.uuid4())
        cars.append(car)
        save_json("cars.json", cars)
        return jsonify({"status": "car_added", "car_id": car["id"]})

@app.route("/api/bookings", methods=["POST"])
def create_booking():
    bookings = load_json("bookings.json")
    booking = request.json
    booking["id"] = str(uuid.uuid4())
    booking["created_at"] = datetime.utcnow().isoformat()
    bookings.append(booking)
    save_json("bookings.json", bookings)
    return jsonify({"status": "booking_confirmed", "booking_id": booking["id"]})
```

```
@app.route("/api/bookings/<user_id>", methods=["GET"])
def get_user_bookings(user_id):
    bookings = load_json("bookings.json")
    user_bookings = [b for b in bookings if b.get("user_id") == user_id]
    return jsonify(user_bookings)

@app.route("/api/payments", methods=["POST"])
def create_payment():
    payments = load_json("payments.json")
    payment = request.json
    payment["id"] = str(uuid.uuid4())
    payment["timestamp"] = datetime.utcnow().isoformat()
    payments.append(payment)
    save_json("payments.json", payments)
    return jsonify({"status": "payment_received", "payment_id": payment["id"]})

@app.route("/api/init", methods=["GET"])
def initialize_files():
    for fname in ["users.json", "cars.json", "bookings.json", "payments.json"]:
        path = get_path(fname)
        if not os.path.exists(path):
            save_json(fname, [])
    return jsonify({"status": "initialized"})

if __name__ == "__main__":
    app.run(debug=True)
```