**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**Факультет економіки, підприємництва та інформаційних технологій**

**Кафедра інформаційних технологій**

**Допущено до захисту**
"_____" _____ 2026 р.
Зав. кафедри, к.т.н., доцент
_____ **Ліщук Р.І.**

# КВАЛІФІКАЦІЙНА РОБОТА
# БАКАЛАВРА

## РОЗРОБКА ВЕБДОДАТКА ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСІВ УПРАВЛІННЯ ГОТЕЛЬНИМ КОМПЛЕКСОМ

Виконав – студент 4 курсу, групи 41-кн
освітньої програми "Комп'ютерні науки"
спеціальності 122 Комп'ютерні науки
**Білий Олексій Володимирович**

Науковий керівник – д.т.н., професор
**Кучерук Володимир Юрійович**

Умань – 2026

**ЗМІСТ**

Перелік умовних позначень, скорочень і термінів**Помилка! Закладку не визначено.**

РОЗДІЛ 1. ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ ..... 5

1.1 Аналіз стану проблеми ..... 5

1.2 Порівняльний аналіз аналогів..... 7

1.3 Аналіз методів розв’язання поставленої задачі..... 12

РОЗДІЛ 2. ПРОЕКТУВАННЯ СТРУКТУРИ, МЕТОДУ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ..... 16

2.1 Вибір архітектури web-додатку..... 16

2.2 Проектування структури графічного інтерфейсу web-додатка ..... 21

2.3 Проектування методів та основних алгоритмів web-додатку ..... 24

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТ ДЛЯ WEB-ДОДАТКУ ..... 28

3.1 Варіантний аналіз і обґрунтування вибору мови програмування ... 28

3.2 Вибір середовища та розробка бази даних..... 30

3.3. Тестування програми..... 44

ВИСНОВКИ ..... 63

СПИСОК ЛІТЕРАТУРИ ..... 64

ДОДАТКИ ..... **Помилка! Закладку не визначено.**

Додаток А. Лістинг програми..... **Помилка! Закладку не визначено.**

## ВСТУП

**Обґрунтування вибору теми.** Сучасна індустрія туризму є складним міждисциплінарним феноменом, що тісно переплітається з історією, географією, архітектурою, спортом та автентичною культурою дестинацій, а також безпосередньо залежить від якості сервісу [1]. Попри суттєві кризові явища останніх років, зумовлені наслідками глобальної пандемії, галузь демонструє поступову стабілізацію та відновлення інтересу населення до подорожей. Невіддільним фактором функціонування цієї сфери є готельно-ресторанне господарство, оскільки більшість мандрівників потребують тимчасового житла або орієнтуються на спеціалізовані готельні програми відпочинку. Готельний та ресторанний бізнес виступає базовою складовою туристичного сектору, що забезпечує комплексне задоволення потреб рекреантів у проживанні, харчуванні, транспортному та іншому обслуговуванні [2]. У період інтенсивного розвитку туризму дана сфера стає дедалі вагомішим сегментом ринку послуг. Жорстка конкуренція змушує лідерів галузі активно впроваджувати інноваційні технології, серед яких ключова роль належить системам управління готелем. Таке програмне забезпечення є багаторівневим комплексом, який об’єднує модулі контролю номерного фонду, обліку постояльців, адміністрування, фінансового аналізу та менеджменту персоналу. Проте на сьогодні ринок пропонує обмежену кількість подібних рішень, а наявні аналоги мають суттєві недоліки, найпоширенішим серед яких є відсутність безшовної інтеграції між окремими підсистемами, зокрема інструментами моніторингу ефективності працівників. Наслідком таких технічних недоліків стають логістичні збої та ускладнення операційного менеджменту для адміністрації, що безпосередньо погіршує рівень сервісу. Це, у свою чергу, спричиняє демотивацію персоналу аж до звільнення, а також призводить до втрати лояльності клієнтів і репутаційних збитків підприємства [3]. З огляду на це, проєктування та розробка авторської системи управління готелем, яка б усувала функціональні недоліки існуючих продуктів, є вкрай актуальним науково-практичним завданням.

**Мета та завдання дослідження.** Метою роботи є оптимізація процесів

адміністрування у сфері гостинності задля підвищення ефективності роботи персоналу та мінімізації часових витрат клієнтів під час обслуговування. Для досягнення цієї мети необхідно здійснити комплексний аналіз поточного стану автоматизації готельного бізнесу, спроєктувати безпечний алгоритм автентифікації користувачів, а також створити механізм динамічного внесення й оновлення відомостей про заклади розміщення у базі даних. Крім того, передбачено розробку алгоритмічної бази для формування текстової та графічної звітності, проєктування загальної архітектури оригінальної системи управління та безпосереднє програмування її вебкомпонентів із подальшим всебічним тестуванням написаного коду та готового застосунку загалом. Об'єктом дослідження виступають процеси проєктування, розробки та впровадження автоматизованих систем керування готельним бізнесом. Предметом дослідження є сукупність методів, технологічних засобів та інструментів програмування, що використовуються для створення таких комплексів. Методологічну основу роботи становлять положення теорії чисел та математичної статистики, які застосовано для формування аналітичних і фінансових звітів. Також у процесі дослідження використано методи комп'ютерного моделювання та сучасні інструменти вебпроєктування для безпосереднього створення програмного продукту, а для перевірки його працездатності, надійності та відповідності вимогам задіяно сучасні методики тестування програмного забезпечення.

Наукова новизна дослідження полягає в оптимізації алгоритму реєстрації та поселення нових клієнтів, який, на відміну від наявних аналогів, передбачає автоматичне створення персонального віртуального рахунку користувача для керування його коштами. Крім того, зазнав удосконалення алгоритм формування аналітичної звітності, що забезпечує підвищення швидкості та ефективності генерації графічних і текстових звітів для менеджменту. Практична значущість роботи полягає в тому, що обґрунтовані теоретичні положення та запропоновані алгоритмічні рішення були безпосередньо втілені у вигляді функціональних програмних засобів, які утворили готову до використання автоматизовану систему управління готельним бізнесом.

# РОЗДІЛ 1. ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

## 1.1 Аналіз стану проблеми

Вплив інформаційних технологій на сучасне суспільство є фундаментальним, адже автоматизація процесів у різних сферах життєдіяльності суттєво підвищує продуктивність людської праці [5]. Масштабування клієнтської бази неминуче спричиняє критичне навантаження на персонал, що призводить до затримок в операційній діяльності та знижує рівень сервісу, проте впровадження спеціалізованих інформаційних комплексів дозволяє ефективно нівелювати ці ризики. У туристичному секторі такі технології мають глибоко інтегрований характер, об’єднуючи в єдину інфраструктуру електронні системи авіакомпаній, засоби телеконференцій, управлінські інформаційні платформи, глобальні мережі резервування та бронювання, системи мультимедіа, інтегровані комунікації, цифрові інструменти електронних переказів і смарткартки [6]. Стрімке зростання популярності індустрії туризму в останні роки зумовлене також розвитком транспортних технологій, які дозволяють за кілька годин дістатися будь-якої точки планети та мінімізувати час у дорозі. Ця тенденція безпосередньо стимулює розвиток і затребуваність готельного бізнесу, успішне функціонування якого наразі повністю залежить від використання сучасних високотехнологічних систем управління.

Ключовою перевагою інтеграції інноваційних технологій у сферу гостинності є суттєве підвищення якості клієнтського сервісу, оскільки автоматизовані системи управління дозволяють персоналу оптимізувати виконання щоденних обов’язків та значно підвищити продуктивність праці. У цьому контексті комп’ютерні системи бронювання виступають ефективним інструментом для збереження, оперативного пошуку інформації та здійснення різноманітних транзакцій, пов’язаних із розміщенням у готелях, авіаперевезеннями та іншими видами туристичної діяльності [7]. Сучасні глобальні дистриб’юторські системи надають користувачам комплексні

можливості для резервування готельних номерів, прокату транспортних засобів, купівлі авіаквитків, а також організації турів, хоча інтеграція залізничних та автобусних перевезень у загальну структуру не завжди є повною. Крім того, такі платформи забезпечують безперебійний обмін даними в готельній індустрії та допомагають уникати критичного переповнення номерного фонду. Об’єднуючи доступні ресурси різних постачальників послуг в єдину комп’ютерну мережу, глобальні системи розповсюдження обслуговують переважно авіакомпанії, готельні мережі та агенції з прокату авто, що дозволяє традиційним та онлайн-туроператорам здійснювати паралельні продажі з єдиної бази даних [8]. Водночас для внутрішньої оптимізації підприємств використовуються інтегровані системи управління людськими ресурсами, які автоматизують кадрове діловодство та забезпечують менеджмент оперативними відомостями щодо структури компанії, вакансій і продуктивності працівників. Наявність такої актуальної аналітики є критично важливою для ухвалення своєчасних та обґрунтованих управлінських рішень. Саме тому вагомим критерієм ефективності сучасних кадрових платформ є здатність модулів нарахування заробітної плати безперешкодно інтегруватися з бухгалтерськими та загальними бізнес-системами підприємства [9].

Архітектура будь-якої інформаційної системи управління базується на взаємодії п'яти фундаментальних елементів, до яких належать апаратне й програмне забезпечення, бази даних, мережева інфраструктура та безпосередньо людські ресурси. Ці компоненти синхронно функціонують для забезпечення циклу введення, обробки, зберігання, виведення та контролю даних, підкріплених зворотним зв'язком. При цьому технічна складова охоплює пристрої введення й виведення інформації, процесори та операційні системи, тоді як програмний складник формується з різноманітних прикладних застосунків і процедур. Структуровані відомості накопичуються у базах даних, а обмін ними координується за допомогою концентраторів та інших мережевих пристроїв під наглядом операторів, системних фахівців і мережевих адміністраторів. Сам процес обробки передбачає надходження первинних

інструкцій до системи, їх програмну трансформацію та подальше представлення результатів у вигляді чітких звітів і структурованих масивів інформації. Схожий принцип діє і в системах управління взаємовідносинами з клієнтами, відомих як CRM-платформи, які консолідують широке коло функцій в одному цифровому просторі задля спрощення менеджменту [10]. Попри стрімку глобальну цифровізацію, ринок спеціалізованого програмного забезпечення для готельної індустрії все ще демонструє брак досконалих рішень, що робить питання розробки ефективних систем управління готелем надзвичайно актуальним. Наявні комерційні продукти часто не задовольняють специфічні потреби великих готельних мереж, оскільки розробники масового софту намагаються зробити його універсальним для різних сфер, через що інтерфейси стають перевантаженими зайвими опціями, або ж навпаки, у них відсутні критично важливі для конкретного бізнесу інструменти. Саме через цей функціональний дисбаланс власники готелів часто віддають перевагу створенню індивідуального програмного забезпечення під замовлення в IT-компаніях. Отже, ключовою проблемою галузі залишається дефіцит адаптованих та якісних управлінських систем, покликаних автоматизувати рутинні операції та суттєво підвищити продуктивність праці персоналу.

### 1.2 Порівняльний аналіз аналогів

Перш ніж перейти до обґрунтування переваг створення власної системи управління готелем, доцільно здійснити детальний порівняльний аналіз уже функціонуючих програмних рішень, що представлені на ринку. Серед сучасних розробок найбільш близькими за своїм функціональним наповненням та цільовим призначенням до проєкту існуючих систем є такі платформи, як Clock PMS, Hotelinstinct, Lite PMS та HotelCloud. Оцінюючи систему Clock PMS, варто відзначити її вагомі практичні переваги, серед яких виділяється можливість оперативного друку різноманітної звітності, ведення детальної історії взаємодії з кожним відвідувачем, а також наявність структурованої клієнтської бази. Водночас цей програмний продукт має і суттєві обмеження, які негативно

впливають на якість адміністрування. Головними недоліками є повна відсутність інструментів для генерації наочних графічних звітів та неможливість здійснювати моніторинг і оцінку ефективності роботи обслуговуючого персоналу. З технічної точки зору ця платформа розгортається на віддаленому сервері або у хмарному сховищі, завдяки чому користувачі можуть отримати безперешкодний доступ до робочого простору за допомогою будь-якого сучасного веббраузера. Візуальне оформлення та особливості організації робочого простору користувацького інтерфейсу згаданого додатка детально відображено на рис. 1.1.




Рисунок 1.1 – Користувацький інтерфейс програми Clock PMS

Розглядаючи систему управління готелем Hotelinstinct, варто зазначити, що вона розроблена для оптимізації адміністрування та автоматизації рутинних процесів у закладах розміщення. Цей програмний продукт забезпечує ефективне збирання, тривале збереження у базі даних та оперативне виведення детальних відомостей про постояльців, а також дозволяє формувати загальні аналітичні показники діяльності підприємства. Попри корисний функціонал, платформа має низку критичних обмежень, аналогічних до попереднього зразка, серед яких

найбільш відчутними є відсутність інструментів для візуалізації даних у вигляді графічних звітів та повний брак модулів для контролю й оцінювання продуктивності роботи готельного персоналу. Вебзастосунок орієнтований на хмарне розгортання або інсталяцію на віддаленому сервері, що гарантує гнучкість використання та доступ до системи з будь-якого комп'ютера чи мобільного пристрою через стандартний інтернет-браузер. Специфіку побудови та стилістику користувацького інтерфейсу цієї програми наочно продемонстровано на рис. 1.2.

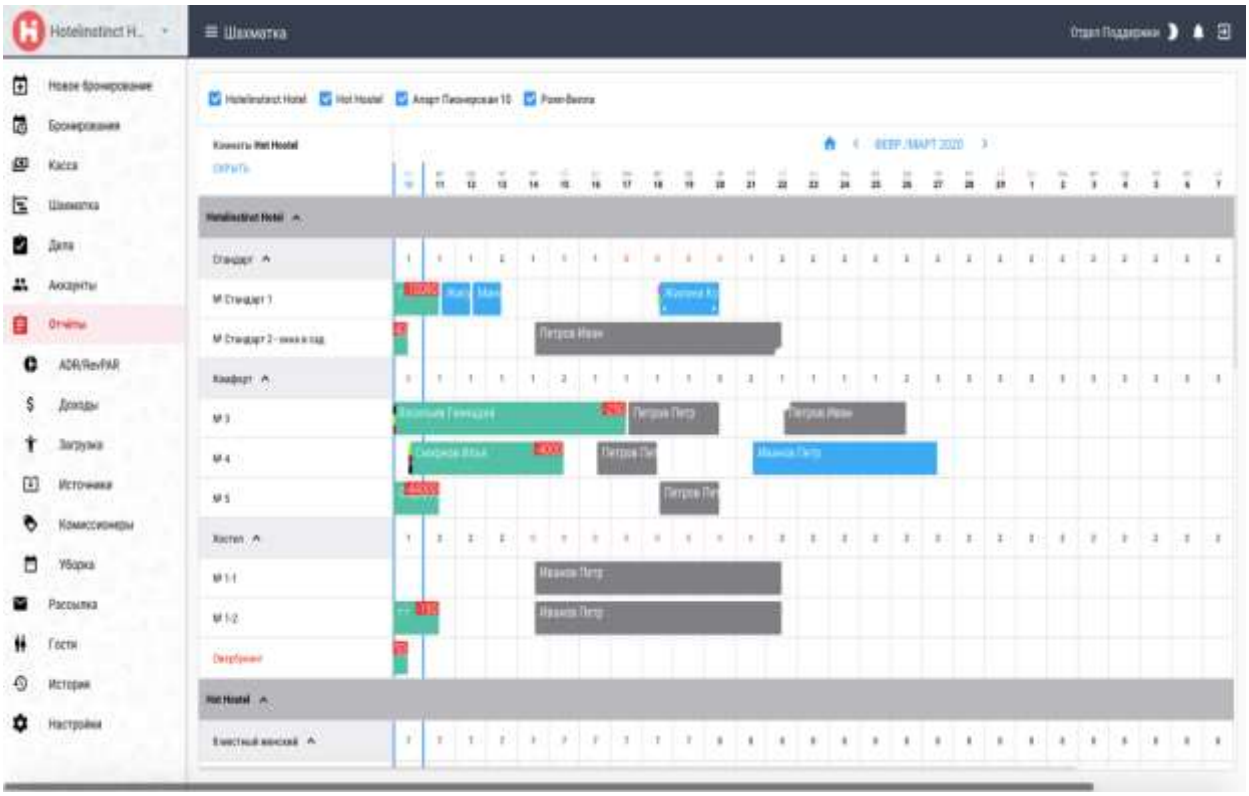



Рисунок 1.2 – Користувацький інтерфейс програми Hotelinstinct

Безкоштовна цифрова платформа Lite PMS розроблена спеціально для оптимізації внутрішнього менеджменту та комплексної автоматизації різноманітних операційних процесів у готельній сфері. Ключові переваги цього програмного рішення зосереджені навколо ефективного збирання персональних відомостей про клієнтів з їх подальшою фіксацією в електронній базі даних, а також у можливості формування базових аналітичних показників діяльності закладу. Водночас інструментарій системи містить суттєві функціональні

прогалини, оскільки в ньому повністю відсутні модулі для генерації наочних графічних звітів та немає жодних технічних засобів для моніторингу працездатності й оцінки продуктивності обслуговуючого персоналу. Продукт орієнтований на хмарне розгортання або розміщення на віддалених серверних потужностях, що забезпечує користувачам безперешкодний доступ до системи з будь-якого сучасного веббраузера. Загальна стилістика, структура та особливості побудови робочого простору користувацького інтерфейсу цього додатка детально відображені на рис. 1.3.

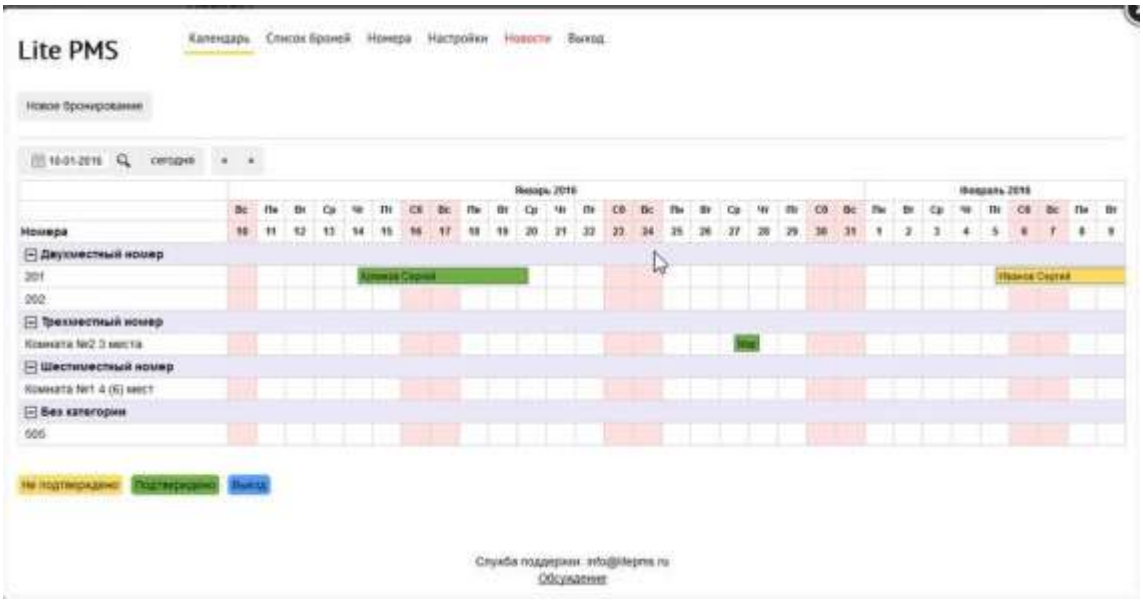



Рисунок 1.3 – Користувацький інтерфейс програми Lite PMS

Автоматизована система HotelCloud розроблена для оптимізації адміністрування готельного бізнесу та координації ключових внутрішніх процесів. Головними функціональними можливостями цієї платформи є збір і систематизація відомостей про клієнтів із подальшим їх надійним збереженням у базі даних, а також генерація загальних аналітичних звітів щодо діяльності закладу розміщення. Водночас вагомим недоліком програмного продукту є повна відсутність інструментів для контролю, обліку та оцінювання продуктивності роботи обслуговуючого персоналу. Цей вебзастосунок орієнтований на розгортання в хмарній інфраструктурі або на віддаленому сервері, завдяки чому користувачі отримують гнучкий і безперешкодний доступ

до системи за допомогою будь-якого сучасного інтернет-браузера. Специфіку побудови робочого простору та особливості користувацького інтерфейсу зазначеної програми наочно продемонстровано на рис. 1.4.

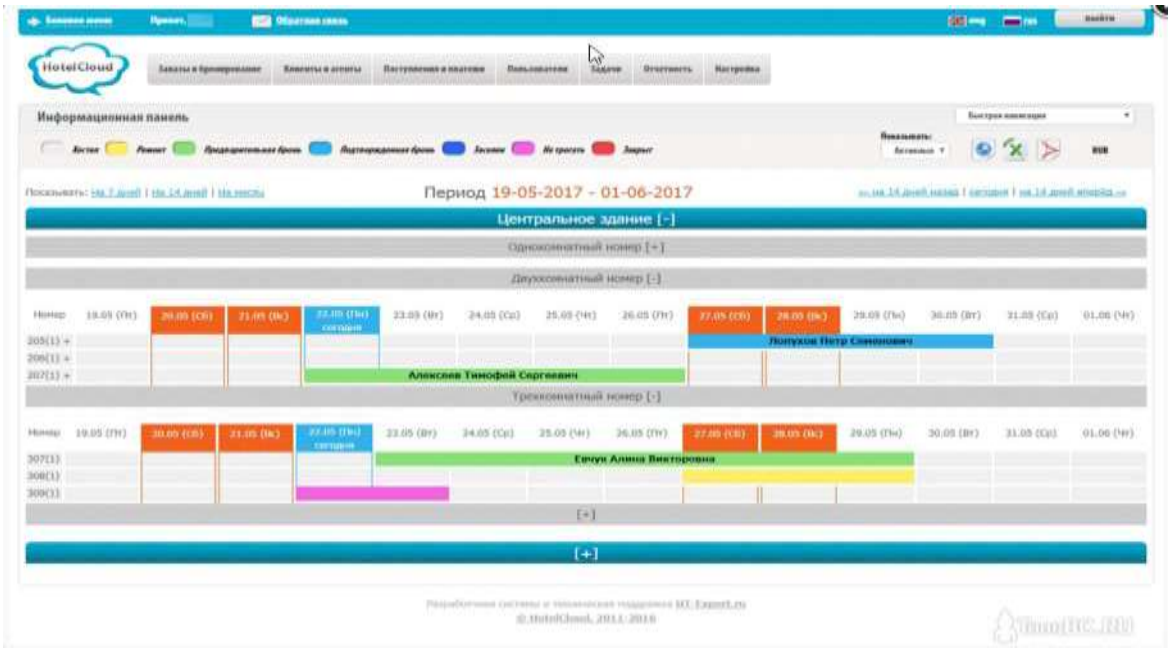



Рисунок 1.4 – Користувацький інтерфейс програми HotelCloud

Комплексний аналіз наявних аналогів дозволив чітко визначити їхні ключові переваги та функціональні обмеження, що стало основою для подальшого порівняння цих продуктів із розроблюваним Телеграм-ботом Expense Control. Усі результати проведеного аналітичного зіставлення були систематизовані та детально відображені в табл. 1.1. Для забезпечення наочності оцінювання технічних можливостей кожної платформи було застосовано спеціальні символьні позначення, де знак плюс вказує на повну відповідність системи певному критерію, знак мінус відображає абсолютну відсутність необхідної функції, а комбінований знак плюс-мінус свідчить про часткову реалізацію відповідного інструменту в архітектурі програмного забезпечення.

Проведене порівняльне дослідження наявних ринкових аналогів дозволило обґрунтувати високу доцільність проєктування та створення оригінальної системи управління готелем. Кінцевим результатом розробки стане конкурентоспроможний програмний продукт, унікальна архітектура якого дозволить повністю нівелювати функціональні недоліки та технічні обмеження

існуючих платформ.

Таблиця 1.1 – Порівняльні характеристики систем

| Критерій | Clock PMS | Hotelinstinct | Lite PMS | HotelCloud | Власна система |
|---|---|---|---|---|---|
| Можливість слідкувати за ефективністю персоналу | - | - | - | - | + |
| Графічні звіти | - | - | - | - | - |
| Безкоштовність повного функціоналу | - | - | + | - | + |
| Бронювання через інтернет | + | + | - | - | - |
| Кросплатформність | + | + | + | + | + |
| Підсумковий результат | 2 | 2 | 2 | 1 | 4 |

### 1.3 Аналіз методів розв’язання поставленої задачі

Під час проєктування системи управління розглядалося кілька можливих архітектурних рішень, серед яких створення класичного програмного додатка для встановлення на персональний комп’ютер або розробка гнучкого вебзастосунку з можливістю розгортання на сервері чи в хмарному просторі для забезпечення віддаленого доступу з будь-якого пристрою. Найбільш ефективним та перспективним способом розв’язання поставленої задачі визначено саме створення вебдодатка, що й зумовило остаточний вибір цього напрямку розробки. Таке програмне забезпечення функціонує безпосередньо через інтернет-браузер, що нівелює потребу у складній локальній інсталяції. При цьому клієнтська частина або зовнішній інтерфейс програми створюється за допомогою базових технологій вебпрограмування, до яких належать мови розмітки та стилізації HTML і CSS, а також мова сценаріїв JavaScript, які повноцінно підтримуються всіма сучасними браузерами, зокрема Chrome, Opera та Mozilla. Водночас для реалізації серверної логіки та обробки даних на бекенді може бути задіяна практично будь-яка інша високорівнева мова програмування або спеціалізований фреймворк, наприклад Python, PHP, Ruby чи Java.

Ключові переваги використання вебзастосунків полягають у їхній універсальності, адже вони здатні стабільно функціонувати на будь-якій операційній системі, включаючи Linux, Mac та Windows, завдяки повній сумісності з усіма сучасними браузерами. Крім того, наявність єдиної кодової бази суттєво спрощує подальше адміністрування, оновлення та технічну підтримку порівняно зі складними десктопними програмами. Процес безпосереднього програмування таких платформ також є менш трудомістким, оскільки розробнику не потрібно налаштовувати низькорівневу взаємодію з апаратними компонентами комп'ютера, такими як процесор, графічна карта чи операційне ядро. Важливою організаційною перевагою перед мобільними додатками є повна відсутність потреби у проходженні тривалих перевірок та отриманні офіційних схвалень від сторонніх цифрових платформ і маркетплейсів для релізу свого продукту. З економічної точки зору вебрішення виступають найбільш вигідним варіантом для підприємств, оскільки вони позбавляють бізнес необхідності придбання коштовних локальних ліцензій або оформлення індивідуальних підписок, дозволяючи використовувати програмне забезпечення як доступну та гнучку хмарну послугу за моделлю SaaS.

Задля оптимізації процесу проєктування та розробки архітектуру створюваного застосунку доцільно розподілити на кілька функціональних модулів, що охоплюють проєктування користувацького інтерфейсу, реалізацію базової бізнес-логіки для внесення й обробки відомостей про готелі, номери та постояльців, а також створення окремого інструменту для генерації наочної графічної звітності. Взаємодія користувача із системою здійснюється через спеціалізований вебінтерфейс, який являє собою сукупність візуальних та інтерактивних засобів, доступних через будь-який інтернет-браузер. Стрімка еволюція та повсюдне впровадження таких інтерфейсів безпосередньо пов'язані з глобальним розвитком всесвітньої мережі та масовим використанням вебпереглядачів як основного інструменту роботи з інформацією. При цьому ключовою вимогою до сучасного вебінтерфейсу є забезпечення кросбраузерності, що гарантує абсолютно ідентичне відображення графічних

елементів та стабільне функціонування всіх закладених можливостей незалежно від обраного користувачем програмного забезпечення для доступу в інтернет.

Традиційний підхід до розробки вебінтерфейсів базується на комплексному використанні HTML розмітки, стилів CSS та сценаріїв JavaScript. Проте специфіка інтерпретації цих технологій різними вебпереглядачами, а також відмінності у реалізації об'єктної моделі документа здатні створювати певні труднощі в процесі програмування. Додатковим дестабілізуючим фактором для коректного відображення елементів можуть стати індивідуальні налаштування браузера користувачем, зокрема зміна шрифтів, колірної палітри чи примусове вимкнення підтримки скриптів. Саме тому інтерфейс створюваної системи має бути максимально інтуїтивним та ергономічним, щоб забезпечити комфортну роботу навіть для осіб без попереднього досвіду. Головним критерієм якості такого дизайну є мінімізація кількості дій, які людина повинна виконати для досягнення бажаного результату. Водночас основою будь-якого вебзастосунку виступає його серверна частина, відома як бекенд. Вона являє собою програмно-апаратний комплекс, який відповідає за реалізацію всієї внутрішньої логіки сервісу та обробку даних. Ця діяльність повністю прихована від кінцевого користувача, оскільки всі процеси відбуваються безпосередньо на віддаленому сервері, а не в браузері комп'ютера. Межа між клієнтською та серверною частинами чітко простежується під час відправлення будь-якого запиту, коли після натискання клавіші введення інформація передається на сервери пошукових або спеціалізованих систем, де запускаються алгоритми обробки. Щойно опрацьовані дані повертаються на монітор користувача у зрозумілому візуальному форматі, процес знову переходить на сторону клієнтського інтерфейсу. Таким чином, серверна частина виконує роль сполучної ланки, яка забезпечує безперебійну взаємодію користувача з обчислювальними ресурсами сервера.

Для реалізації серверної частини проєкту обрано мову програмування PHP, яка є одним із найпопулярніших та перевірених часом інструментів у сфері вебдизайну та розробки. Процес програмування здійснюватиметься на базі

сучасного фреймворку Laravel, який функціонує за класичним архітектурним шаблоном MVC. Вибір цього фреймворку обумовлений наявністю великої кількості готових інженерних рішень та вбудованих інструментів, що дозволяє суттєво прискорити побудову надійної структури майбутнього вебзастосунку. Як систему управління базами даних для нагромадження та збереження інформації вирішено задіяти MySQL від компанії Oracle. Ця поширювана за ліцензією реляційна СУБД займає лідируючі позиції на ринку завдяки своїй стабільності та широким функціональним можливостям, які повністю покривають усі технічні потреби створюваного продукту. При цьому ключові компоненти системи мають бездоганно виконувати закладені операційні завдання. Архітектура програми передбачатиме обов'язкову наявність інструментів для динамічного внесення до бази даних нових об'єктів розміщення, наявного номерного фонду, відомостей про клієнтів та інших супутніх параметрів. Окреме місце в структурі посяде спеціалізований фінансовий модуль для ведення точного обліку грошових коштів, який тісно взаємодіятиме з блоком аналітики, відповідальним за оперативну генерацію деталізованих текстових документів та наочних графічних звітів.

# РОЗДІЛ 2. ПРОЕКТУВАННЯ СТРУКТУРИ, МЕТОДУ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

## 2.1 Вибір архітектури web-додатку

Для початку необхідно чітко визначити сутність поняття веб-додаток, яка базується на трикомпонентній структурі, що охоплює такі елементи як клієнт, сервер та база даних. За своєю архітектурною моделлю веб-додаток являє собою систему типу клієнт-сервер, де в ролі безпосереднього клієнта виступає інтернет-браузер, а функції серверної сторони виконує веб-сервер. Ключовий масив обчислювальних процесів та алгоритмів традиційно зосереджений саме на стороні веб-сервера, який бере на себе відповідальність за прийом, аналіз та всебічну обробку надхідних запитів відповідно до закладеної бізнес-логіки програмного продукту, після чого генерує фінальну відповідь для відправки назад. На заключному етапі взаємодії до процесу долучається браузер, головне завдання якого полягає у трансформації отриманих від сервера технічних даних та їх візуалізації у вигляді зрозумілого, інтерактивного та зручного для кінцевого користувача графічного інтерфейсу.

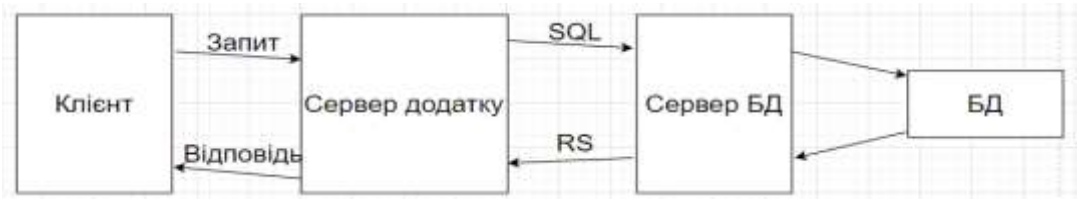



Рисунок 2.1 – Клієнт-серверна архітектура веб-додатку

В межах класичної архітектури першим ключовим елементом є клієнт, роль якого зазвичай виконує інтернет-браузер, хоча існують специфічні сценарії, коли один веб-сервер ВС1 надсилає запит до іншого сервера ВС2, і тоді саме сервер ВС1 тимчасово перебирає на себе клієнтські функції. У стандартній ситуації браузер забезпечує обробку отриманих від сервера технічних даних, трансформуючи їх у повноцінний графічний інтерфейс користувача. Цей процес візуалізації відбувається завдяки чітким інструкціям, закладеним за допомогою

технологій HTML, CSS та JS, які безпосередньо вказують браузеру, як саме необхідно відображати та структурувати отриманий контент. Другим обов'язковим складником є веб-сервер, що відповідає за прийом вхідних HTTP-запитів від клієнтів та генерацію відповідних HTTP-відповідей. Це поняття традиційно об'єднує в собі як спеціалізоване програмне забезпечення, так і безпосередньо фізичний або віртуальний комп’ютер, на якому воно функціонує, причому найпопулярнішими програмними рішеннями в цій категорії є Apache, IIS та NGINX. Клієнт-серверний додаток працює безпосередньо в середовищі веб-сервера і може бути написаний на базі різних мов програмування, серед яких найчастіше використовуються PHP, Python, Ruby, Java та Perl. Третім фундаментальним компонентом, який хоч і залишається невидимим під час поверхневого розгляду класичної двосторонньої моделі, є база даних. Вона функціонує як автономний елемент поза безпосередньою структурою веб-сервера, проте є критично важливою для повноцінної роботи більшості сучасних систем, оскільки саме в ній нагромаджується та систематизується вся динамічна інформація застосунку, включаючи облікові записи користувачів, персональні профілі та інші супутні робочі відомості.

Поняття бази даних є надзвичайно широким терміном, сфера застосування якого виходить далеко за межі суто інформаційних технологій. У межах цього дослідження база даних розглядається як спеціалізована інформаційна модель, призначена для впорядкованого та систематизованого збереження відомостей про конкретний об’єкт або взаємопов'язану групу об’єктів, що володіють набором класифікованих властивостей. Функціонування таких масивів інформації забезпечується завдяки спеціальному програмному забезпеченню, відомому як системи управління базами даних, серед яких найвищий рівень популярності у клієнт-серверному сегменті мають такі платформи як MySQL, MS SQL Server, PostgreSQL та Oracle. Поряд із ними існують вбудовані СУБД та рішення на базі файлових серверів, де як приклад широко відомого вбудованого інструменту варто згадати SQLite, що активно застосовується в архітектурі сучасних інтернет-браузерів, Android API, програмі Skype та багатьох інших

поширених програмних продуктах. Взаємодія користувача або додатка з усіма згаданими системами керування реалізується за допомогою єдиної універсальної структурованої мови запитів SQL. Водночас для повноцінного функціонування всього вебзастосунку критично важливою є структура сайту, яка визначає логічну побудову абсолютно всіх наявних вебсторінок, базових категорій та внутрішніх підкатегорій. Ця логічна схема чітко регламентує взаємне розташування розділів відносно один одного, а також визначає базові принципи та технічні зв'язки, за якими вони взаємодіють у загальній системі.

У сфері розробки програмного забезпечення виділяють три фундаментальні архітектурні шаблони, до яких належать Model-View-Controller або MVC, Model-View-Presenter або MVP, а також Model-View-ViewModel або MVVM. Серед них детального розгляду заслуговує патерн MVC, який передбачає чіткий розподіл структури застосунку на три автономні компоненти, що дозволяє модифікувати кожен елемент незалежно від інших, суттєво спрощуючи процес створення та подальшого технічного супроводу вебпрограм. Перший складник, відомий як Model або модель, зосереджує в собі ключову бізнес-логіку, структуру програми та методи опрацювання інформації. Вона динамічно реагує на керівні команди, що надходять від контролера, змінює свій внутрішній стан або генерує необхідні відомості для їх подальшої передачі на наступний рівень. Другий компонент, який називається View або представлення, відповідає за безпосередню візуалізацію отриманих від моделі даних на рівні користувацького інтерфейсу, трансформуючи їх у наочну графічну форму, та визначає загальний зовнішній вигляд системи і доступні способи взаємодії з нею. Третій елемент, визначений як Controller або контролер, виконує функцію сполучної ланки, яка забезпечує повноцінну взаємодію з користувачем шляхом перехоплення його дій, верифікації отриманих вхідних даних та їх наступного делегування моделі, що фактично регламентує загальний характер реагування програми на будь-які зовнішні запити.

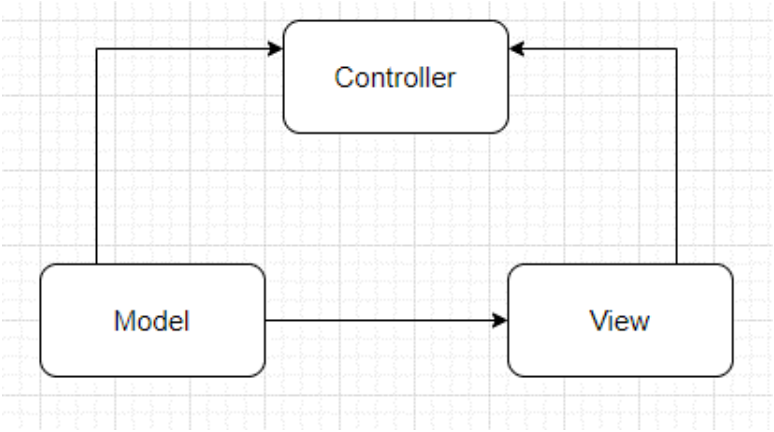



Рисунок 2.2 – структура MVC

Архітектурний шаблон MVP покликаний ефективно усувати системні недоліки класичної моделі MVC, пропонуючи простіший і логічніший підхід до структурування коду програмного проєкту. Високий рівень популярності та широке визнання цього патерна обумовлені тим, що його впровадження гарантує відмінну модульність системи, високу якість проведення автоматизованого тестування, а також формування значно чистішої та легшої в обслуговуванні кодової бази. Конструкція шаблону тримається на взаємодії трьох автономних рівнів, де першим складником виступає model або модель, яка функціонує як ізольований шар для збереження даних і бере на себе повну обробку доменної логіки, реальних бізнес-правил, а також координацію зв'язку з базою даних та мережевими рівнями. Другим елементом є view або перегляд, який представляє шар безпосереднього інтерфейсу користувача та відповідає за візуалізацію отриманої інформації, паралельно відстежуючи будь-які зовнішні дії та активності відвідувача сайту з метою оперативного сповіщення наступного компонента. Цим третім критично важливим складником виступає presenter або доповідач, який акумулює відомості від моделі та застосовує до них специфічну логіку інтерфейсу користувача для ухвалення рішення щодо фінального відображення контенту. Він безпосередньо керує поточним станом перегляду та здійснює необхідні системні дії у відповідь на кожне нове сповіщення, яке надходить від користувача через візуальний інтерфейс даних.

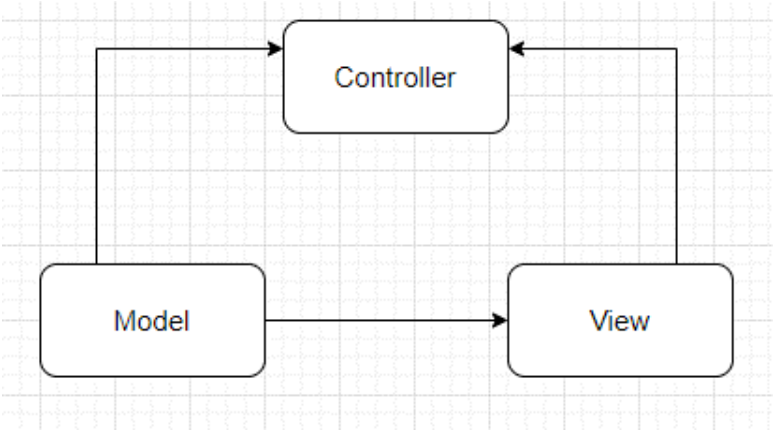



Рисунок 2.3 – Структура MVP

Архітектурний шаблон MVVM виступає ефективним інструментом для покращення розподілу системних обов'язків, оскільки він дозволяє повністю відокремити логіку користувацького інтерфейсу від базової бізнес- або серверної логіки програми. Головна мета цього підходу, яка є спільною і для інших патернів сімейства MVC, полягає в реалізації фундаментального принципу, згідно з яким код візуального інтерфейсу має залишатися максимально простим та очищеним від внутрішніх алгоритмів, що суттєво полегшує подальше адміністрування та масштабування проєкту. Структура цього шаблону тримається на взаємодії трьох автономних шарів, де першим компонентом є модель, яка безпосередньо уособлює всі дані та правила бізнес-логіки додатка. Для досягнення максимальної ізоляції цього рівня від інших споживачів інформації рекомендується організовувати передачу його даних за допомогою спеціальних спостережуваних об'єктів, що дозволяє повністю відмежувати модель від наступного архітектурного елемента. Цим другим складником виступає ViewModel, яка безпосередньо взаємодіє з моделлю та готує інформаційні масиви у вигляді елементів, доступних для моніторингу з боку інтерфейсу. Крім того, цей проміжний шар може надавати специфічні програмні гачки для трансляції зовнішніх подій назад у модель. Однією з найважливіших стратегічних вимог при побудові ViewModel є її абсолютна автономність від візуальної частини, тобто цей компонент не повинен володіти жодними відомостями про конкретне представлення, з яким він взаємодіє. На заключному

етапі архітектурного ланцюжка функціонує перегляд, роль якого в межах описуваного патерна зводиться до безперервного спостереження або підписки на оновлення від ViewModel, що дозволяє динамічно отримувати свіжі дані та оперативно коригувати відповідно до них поточний стан елементів користувацького інтерфейсу.

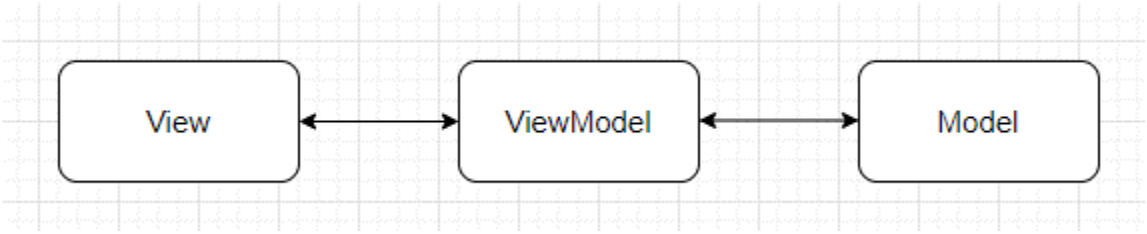



Рисунок 2.4 – структура MVMM

Для вебзастосунків традиційно найчастіше обирають саме архітектурну модель MVC, проте в умовах сучасного інтернет-середовища цей шаблон зазнає певних модифікацій. Специфіка адаптації полягає в тому, що в мережевому просторі компонент модель позбавлений можливості безпосередньо взаємодіяти із шаром вигляду, тому будь-яка комунікація та передача даних між ними здійснюється виключно за посередництва контролера. Враховуючи ці особливості, для проєктування та розробки автоматизованої системи управління готелем було обрано саме модифіковану архітектуру MVC, оскільки такий підхід виступає найбільш оптимальним та ефективним рішенням для побудови сучасних динамічних вебдодатків.

## 2.2 Проектування структури графічного інтерфейсу web-додатка

Базовий візуальний каркас абсолютно всіх екранних форм проєктується за єдиним принципом і включає в себе навігаційне меню та центральну область для відомостей. У верхній частині або шапці застосунку передбачено постійне розміщення фірмового логотипа, супутнього текстового супроводу, а також спеціального інтерактивного елемента, який при натисканні розгортає блок із персональними даними поточного адміністратора та кнопками для завершення сесії користувача. Ліва сторона робочого простору виділяється під головне меню

системи, тоді як уся центральна зона відводиться під динамічний основний контент, який автоматично оновлюється та змінює свій вміст під час переходу між різними розділами сайту. Графічне втілення та детальне розташування елементів у межах описаного простору наочно демонструють перша, друга та третя структурні схеми інтерфейсів вікон веб-додатка, наведені у другому розділі.

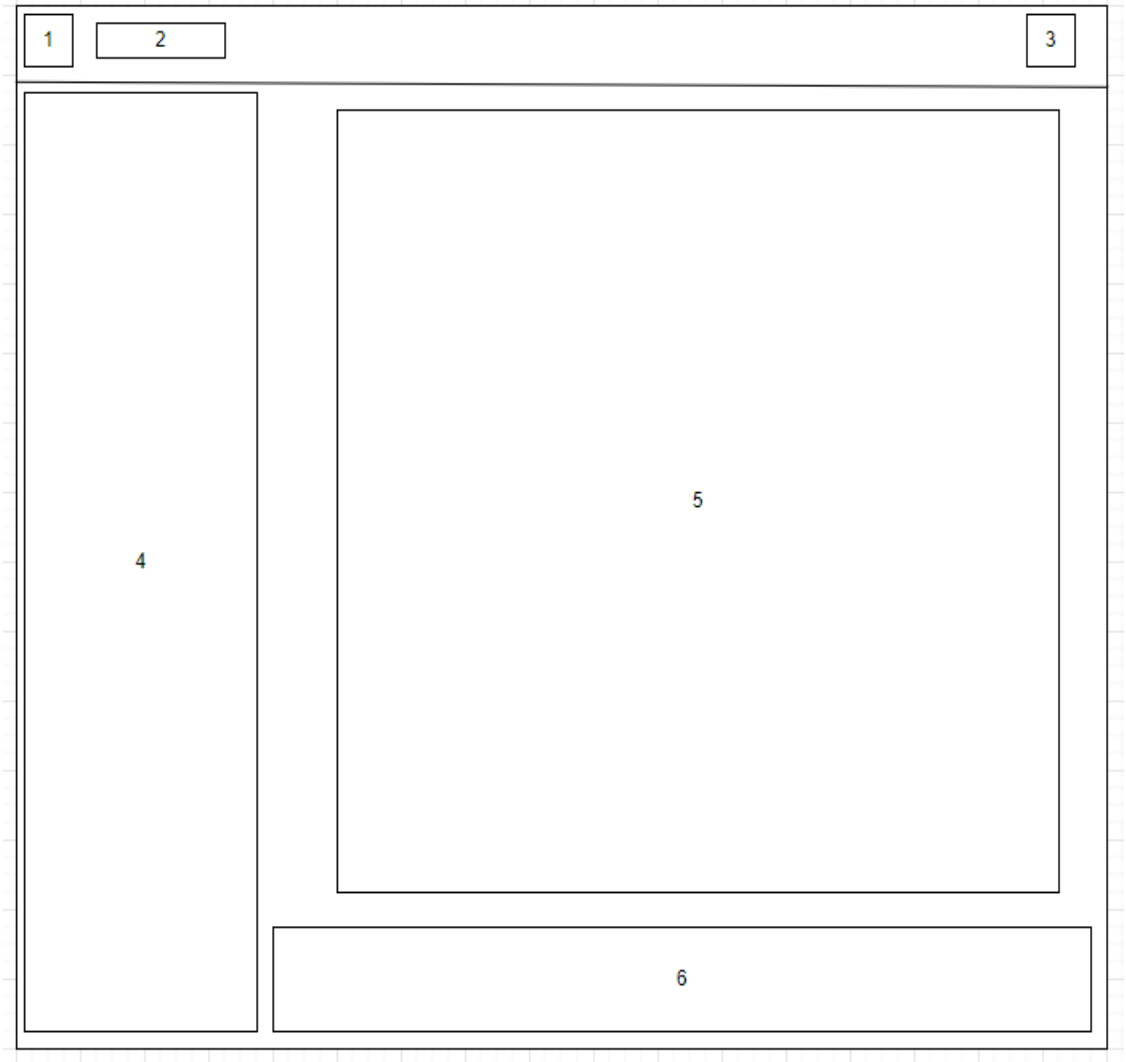



Рисунок 2.5 – Графічна схема головного вікна боту

Основні елементи інтерфейсу головного вікна боту «Expense Control»: 1. Логотип; 2. Текст; 3. Кнопка «Адміністратора»; 4. Меню; 5. Контент; 6. Футер.

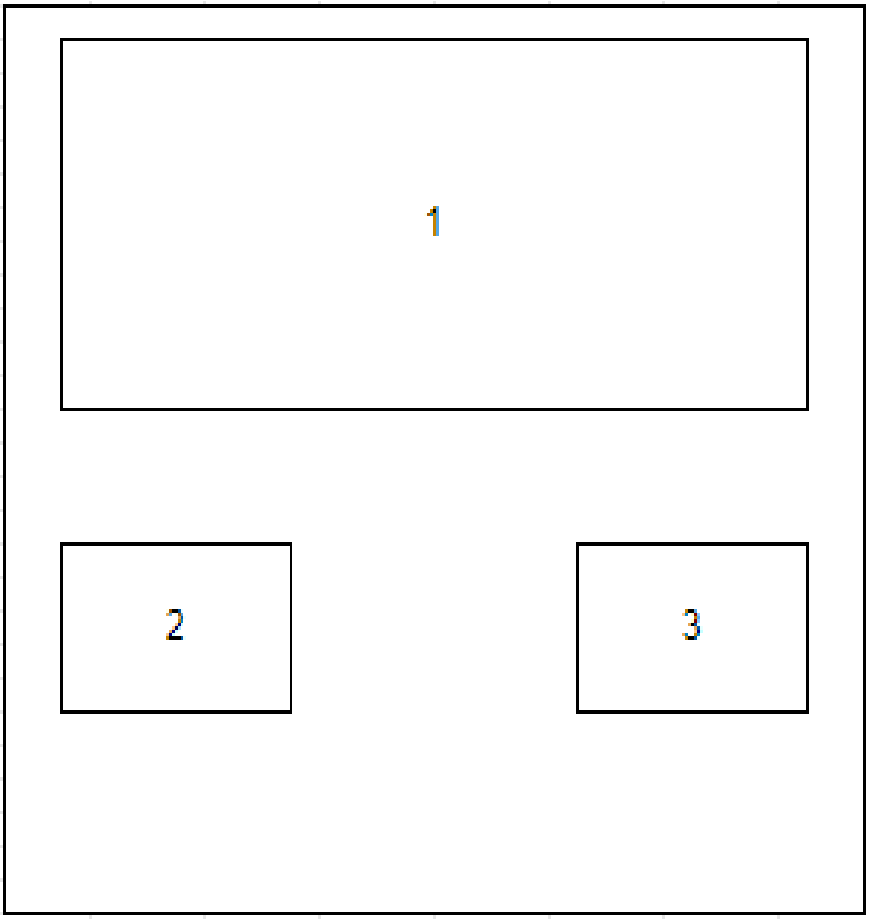



Рисунок 2.6 – Графічна схема модального вікна адміністратора

Основні елементи інтерфейсу вікна інформації про бота: 1. Зображення і ім’я адміністратора; 2. Кнопка «Профіль»; 3. Кнопка «Вихід».

Основні елементи екрану авторизації: 1. Логотип; 2. Текст; 3. Поле для в ведення логіну; 4. Поле для введення пароля; 5. Кнопка «Ввійти»; 6. Футер

Створення зручного та інтуїтивно зрозумілого візуального простору виступає критично важливим етапом у загальному циклі проєктування веб-додатка. Якість та ергономічність користувацького інтерфейсу безпосередньо впливають на загальне сприйняття продукту, оскільки саме комфорт під час взаємодії із системою та легкість освоєння її функцій є ключовими критеріями, якими керується клієнт при виборі кінцевого програмного рішення для свого бізнесу.

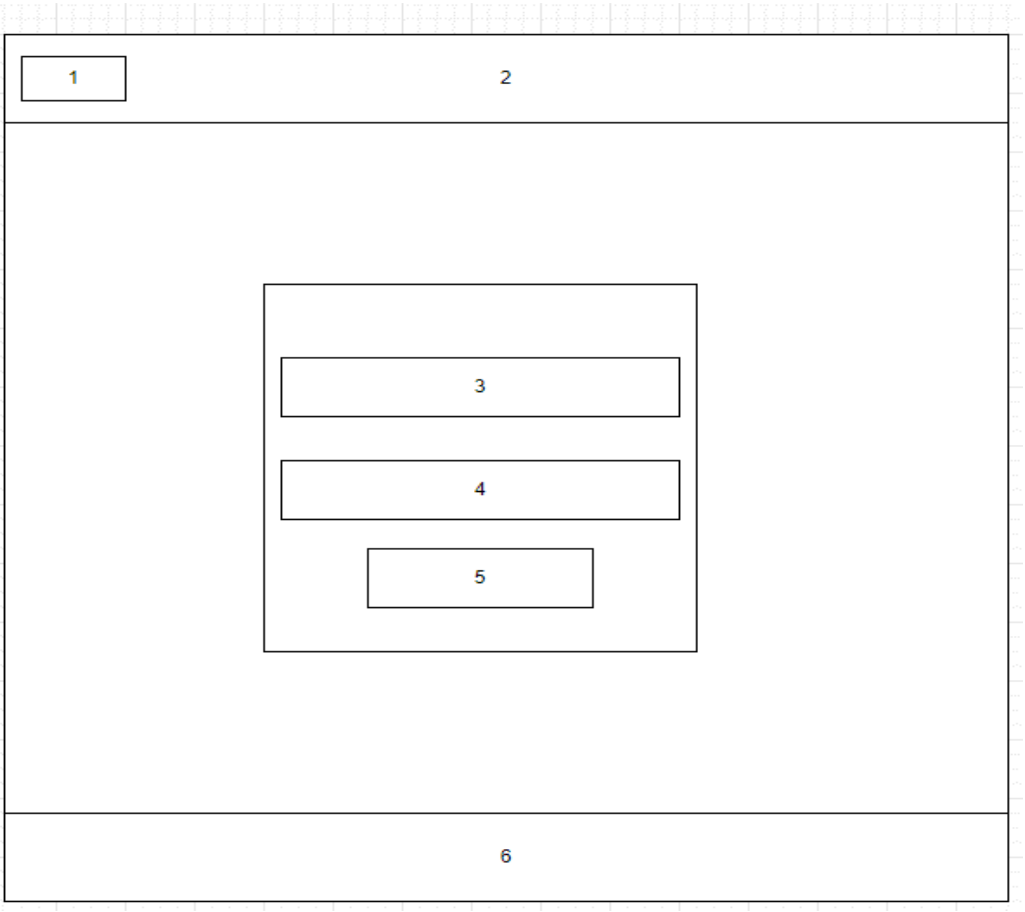



Рисунок 2.7 – Графічна схема модального вікна адміністратора

### 2.3 Проектування методів та основних алгоритмів web-додатку

Процес інтеграції нових клієнтів у базу даних системи підпорядкований чітко визначеній послідовності дій, графічне втілення якої відображено на рис. 2.4. На початковому етапі алгоритму здійснюється збір первинної інформації, яку адміністратор закладу вносить у відповідні поля електронної форми застосунку. Після відправки цих відомостей система запускає автоматичний цикл верифікації для перевірки повноти та коректності заповнення всіх обов'язкових реквізитів. У разі успішного проходження цієї перевірки та підтвердження валідності даних програмний комплекс виконує операцію запису отриманого масиву інформації безпосередньо в реляційну базу даних, після чого сигналізує про успіх процедури шляхом виведення на екран сповіщення про успішне додавання постояльця. Якщо ж на етапі перевірки виявляються критичні помилки, невідповідності або порожні поля, система миттєво відхиляє запит, перенаправляє процес до фінальної стадії та демонструє адміністратору

попередження про некоректність введених параметрів, що знаменує собою завершення роботи цього операційного алгоритму.




Рисунок 2.8 – Блок-схема алгоритму додавання нових постояльців в систему

Аналітичний блок є одним із найважливіших аспектів функціонування системи, тому логіка роботи цього модуля базується на спеціально спроєктованому алгоритмі генерації текстових та графічних звітів про витрати, візуальну схему якого наочно представлено на п'ятому рис. 2.9. На початковому етапі виконання процедури програма очікує дій від користувача та приймає від нього вхідні параметри щодо бажаного типу аналітики. Отримавши ці критерії, система надсилає відповідний запит до сховища інформації, вибирає з бази даних необхідні масиви відомостей та здійснює їх комплексний внутрішній аналіз і сортування. Після завершення обробки даних на екрані формуються інтерактивні елементи керування у вигляді кнопок для вибору графічного або текстового формату відображення. Якщо користувач ініціює створення першого варіанта, застосунок запускає внутрішні графічні інструменти для побудови відповідних

діаграм і графіків, після чого демонструє на моніторі готовий звіт у наочній візуальній формі. У разі вибору текстового варіанта система оминає процес візуалізації та відразу виводить на екран упорядковану текстову та цифрову звітність, що в обох випадках приводить роботу цього операційного циклу до свого логічного завершення.

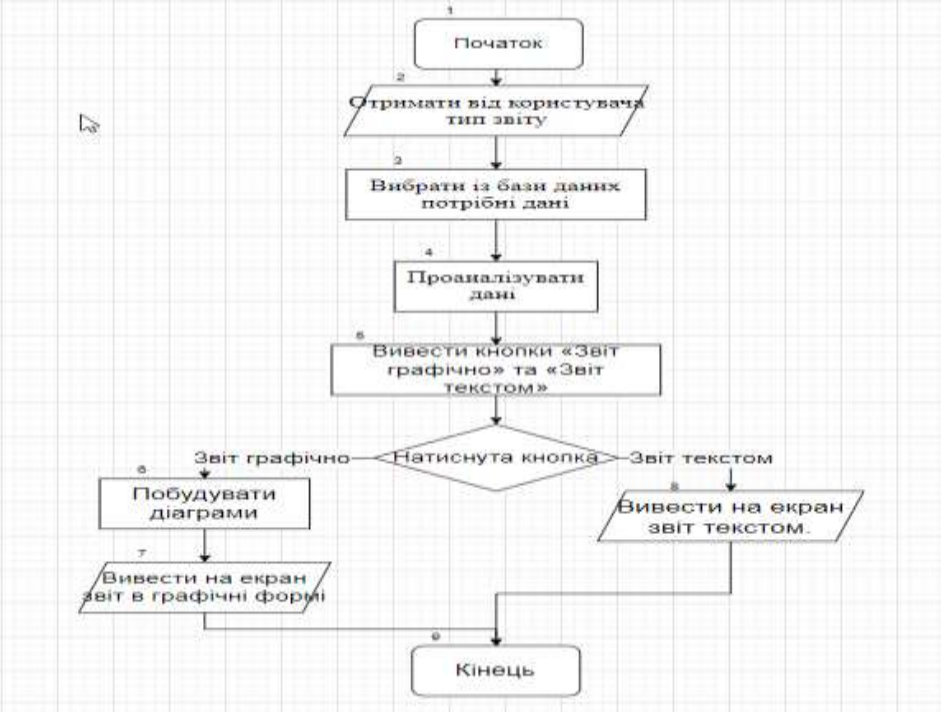



Рисунок 2.9 – Блок-схема алгоритму генерування текстових та графічних звітів

Фундаментальним механізмом забезпечення безпеки та розмежування прав доступу в системі виступає алгоритм авторизації користувачів, детальну схему послідовності дій якого відображено на рис. 2.10. На початковому етапі цієї процедури програма очікує від користувача заповнення відповідних інтерактивних полів автентифікації. Після надсилання форми застосунок першочергово здійснює перевірку введеної інформації на відповідність базовим правилам валідації. Якщо дані успішно проходять цей первинний фільтр, процес переходить до наступного етапу верифікації, де система надсилає запит до сховища інформації, щоб перевірити наявність облікового запису з такими параметрами безпосередньо в базі даних. За умови успішного знаходження відповідного профайлу програма виконує збереження ідентифікаційних

відомостей користувача в поточну сесію для підтримки його активного статусу, після чого автоматично здійснює перенаправлення та редирект на головну робочу сторінку системи. Водночас, якщо на етапі первинної валідації або під час пошуку в базі даних виявляються помилки чи невідповідності, алгоритм миттєво відхиляє запит, демонструє на екрані попередження про некоректність введених даних, після чого робота цього захищеного циклу завершується.

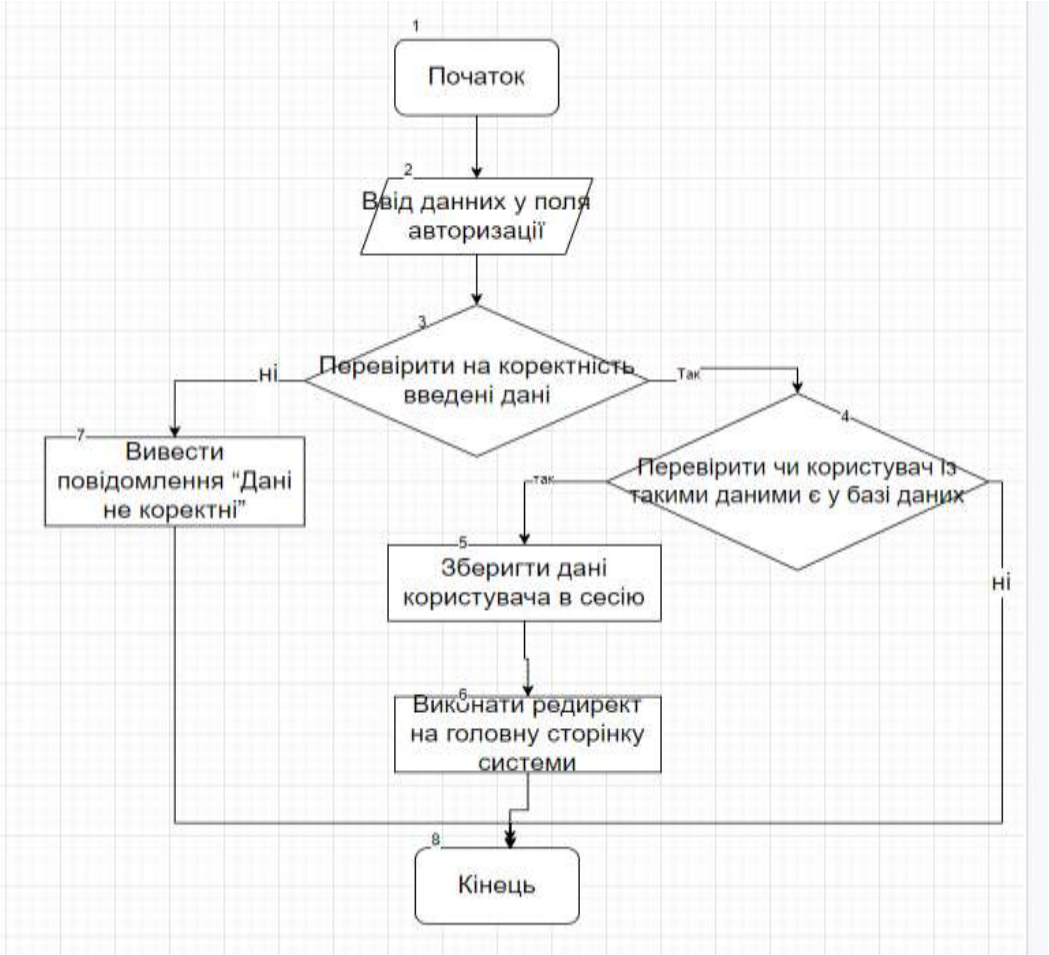



Рисунок 2.10 – Блок-схема авторизації

# РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТ ДЛЯ WEB-ДОДАТКУ

## 3.1 Варіантний аналіз і обґрунтування вибору мови програмування

У сучасному вебдизайні та індустрії створення цифрових продуктів провідні позиції посідають кілька найбільш затребуваних мов програмування, серед яких особливе місце належить PHP, JavaScript та Python. Перша з них, PHP, являє собою потужну скриптову мову, яка пройшла тривалий шлях еволюції практично від самого початку зародження інтернету, що дозволило їй стати фундаментальним інструментом як для проектування найпростіших односторінкових сайтів, так і для розгортання складних високонавантажених систем із колосальним щоденним трафіком. До ключових переваг цього інструменту традиційно відносять глибоку інтеграцію з різноманітними популярними системами управління базами даних, включаючи рішення від Oracle та MySQL, високу швидкість виконання сценаріїв порівняно з багатьма іншими інтерпретованими мовами, загальну легкість освоєння, відкритий вихідний код, а також величезну глобальну спільноту розробників, яка сформувала базу навчальних матеріалів. Паралельно з цим невід'ємною частиною інтернет-технологій є JavaScript або JS, що функціонує як динамічна об'єктно-орієнтована мова прототипного типу, яка базується на загальноприйнятому стандарті ECMAScript. Цей інструмент переважно орієнтований на створення інтерактивних сценаріїв безпосередньо на стороні клієнта, забезпечуючи повноцінну асинхронну комунікацію з сервером, гнучке управління браузерними процесами, а також миттєву зміну візуальної структури та стилістичного оформлення вебсторінок у відповідь на дії користувача. Популярність JavaScript підкріплюється простотою його практичного впровадження, колосальним обсягом доступної документації та наявністю величезної кількості спеціалізованих фреймворків, які дозволяють суттєво оптимізувати процес написання коду для сучасних інтерфейсів.

Мова програмування Python уособлює собою високорівневий

інтерпретований інструмент об'єктно-орієнтованого спрямування, що функціонує на засадах строгої динамічної типізації. Ключова філософія цього інструменту зосереджена на максимізації продуктивності праці програміста завдяки лаконічності й прозорості синтаксису, а також наявності вичерпної та доступної документації. Завдяки своїй універсальності ця технологія успішно залучається для розв'язання колосального спектра завдань, проте найважливішими сферами її практичного розгортання традиційно залишаються аналітика великих даних, побудова вебсайтів та індустрія мобільних ігор. Серед специфічних архітектурних переваг Python особливе місце посідає його абсолютна кросплатформність, адже відповідні інтерпретатори адаптовані під більшість сучасних операційних систем, що повністю нівелює проблеми із запуском коду в різних програмних середовищах. Крім того, високий рівень поширення забезпечив появу величезної кількості супутніх сервісів, спеціалізованих середовищ розробки та готових фреймворків, а надзвичайна глобальна популярність мови гарантує розробникам легкий доступ до різноманітних інформаційних джерел та навчальних матеріалів. Для комплексного зіставлення технологічних можливостей усіх проаналізованих інструментів за визначеними технічними параметрами сформовано порівняльну табл. 3.1, де повноцінне втілення конкретного критерію позначається позитивним індикатором, повна відсутність маркується негативним символом, а часткова реалізація або наявність певних обмежень відображається за допомогою комбінованого знака.

Таблиця 3.1 – Порівняння мов програмування

| Критерій | PHP | JavaScript | Python |
|---|---|---|---|
| Об’єктно-орієнтованість | + | + | + |
| Простота синтаксису | + | +- | + |
| Обширність та доступність документації | + | + | + |
| Направленість мови в WEB | + | + | +- |
| Легкість роботи із базами даних | + | +- | + |
| Різноманітність фреймворків, які полегшить розробку | + | + | + |
| Підсумковий результат | 6 | 5 | 5 |

### 3.2 Вибір середовища та розробка бази даних

Поряд із визначенням оптимальної мови програмування критично важливим етапом виступає підбір відповідного інтегрованого середовища розробки. Таке середовище, відоме під абревіатурою IDE, являє собою універсальний програмний комплекс, спеціально створений для підвищення ефективності створення цифрових продуктів. Традиційно його внутрішня структура об'єднує в собі високофункціональний редактор вихідного коду, спеціалізовані інструменти для автоматизації процесів збирання проєкту та вбудовані засоби для пошуку й усунення помилок у програмі. При цьому переважна більшість сучасних інструментів цього класу оснащена інтелектуальними системами автоматичного доповнення тексту, що суттєво прискорює написання інструкцій. У контексті створення вебдодатків на базі PHP найбільшу популярність та поширення серед фахівців здобули такі платформи як PhpStorm, Eclipse та Visual Studio Code. Особливе місце в цьому переліку посідає PhpStorm, який функціонує як потужне спеціалізоване середовище для роботи з PHP, HTML та JavaScript. Його архітектурною перевагою є здатність здійснювати глибокий аналіз написаного коду безпосередньо в процесі його введення, що дозволяє розробнику вчасно запобігати появі синтаксичних та логічних помилок, а також використовувати автоматизовані інструменти для проведення безпечного рефакторингу. Інтелектуальний помічник автодоповнення в цьому середовищі бездоганно адаптований під усі сучасні специфікації мови, включаючи повноцінну підтримку генераторів, співпрограм, просторів імен, замикань, типажів та скороченого синтаксису масивів. Додатковим вагомим бонусом для інженера є наявність вбудованого повнофункціонального редактора SQL, який дозволяє не лише надсилати запити до сховищ даних, а й безпосередньо коригувати отримані результати в інтерактивному режимі.

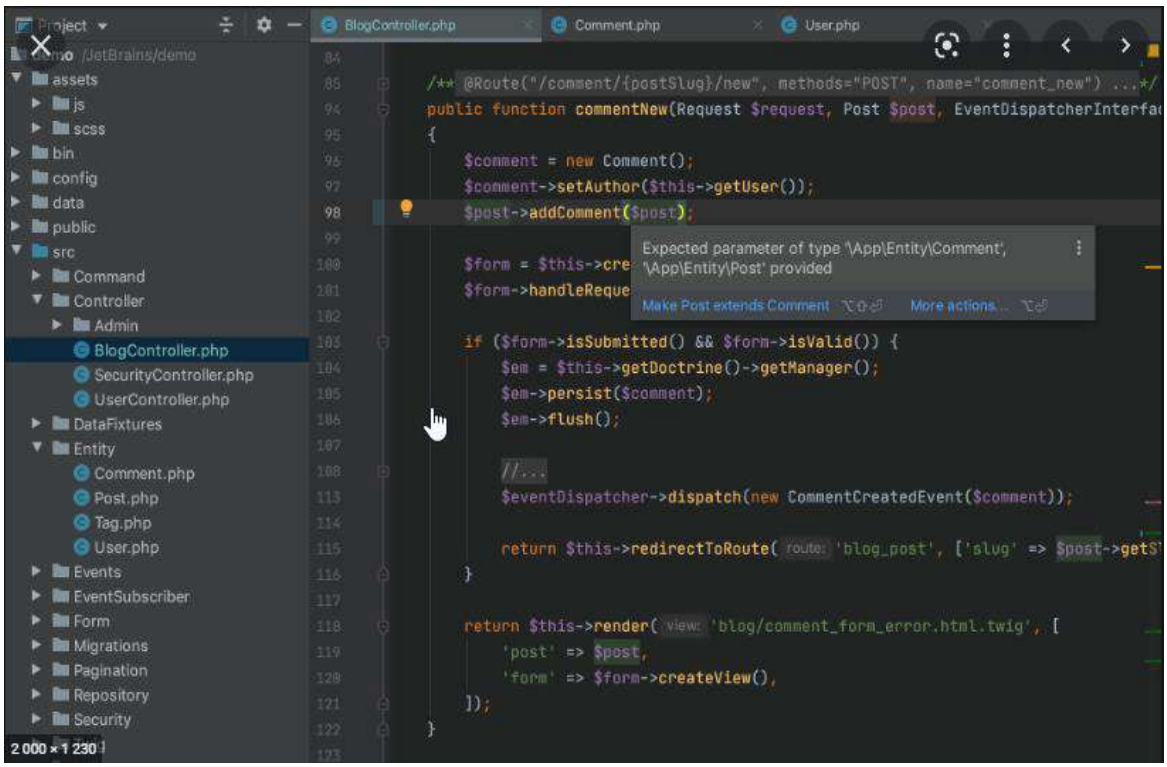



Рисунок 3.1 – Інтерфейс програмного додатку «PhpStorm»

Середовище Eclipse представляє собою безкоштовну модульну платформу для комплексної розробки програмного забезпечення, підтримкою та розвитком якої займається організація Eclipse Foundation. Цей інструментарій об'єднує в собі різноманітні підпроєкти, включаючи базову системну архітектуру, розширений набір засобів для розробників на мові Java, інтегровані модулі взаємодії з системами контролю версій, а також графічні конструктори для створення користувацьких інтерфейсів. Оскільки саме середовище створене переважно на Java, воно орієнтоване на роботу з цією мовою програмування, проте завдяки гнучкій архітектурі та підключенню численних сторонніх плагінів його можна успішно адаптувати для написання коду іншими мовами, серед яких і PHP. До суттєвих позитивних характеристик цієї платформи відносять наявність інтелектуального підсвічування синтаксису, функцію миттєвого інформування розробника про виявлені помилки, динамічні підказки безпосередньо під час введення текстових команд, а також вбудований інструмент для автоматичного вирівнювання та структурування вихідного коду за визначеними стандартами.

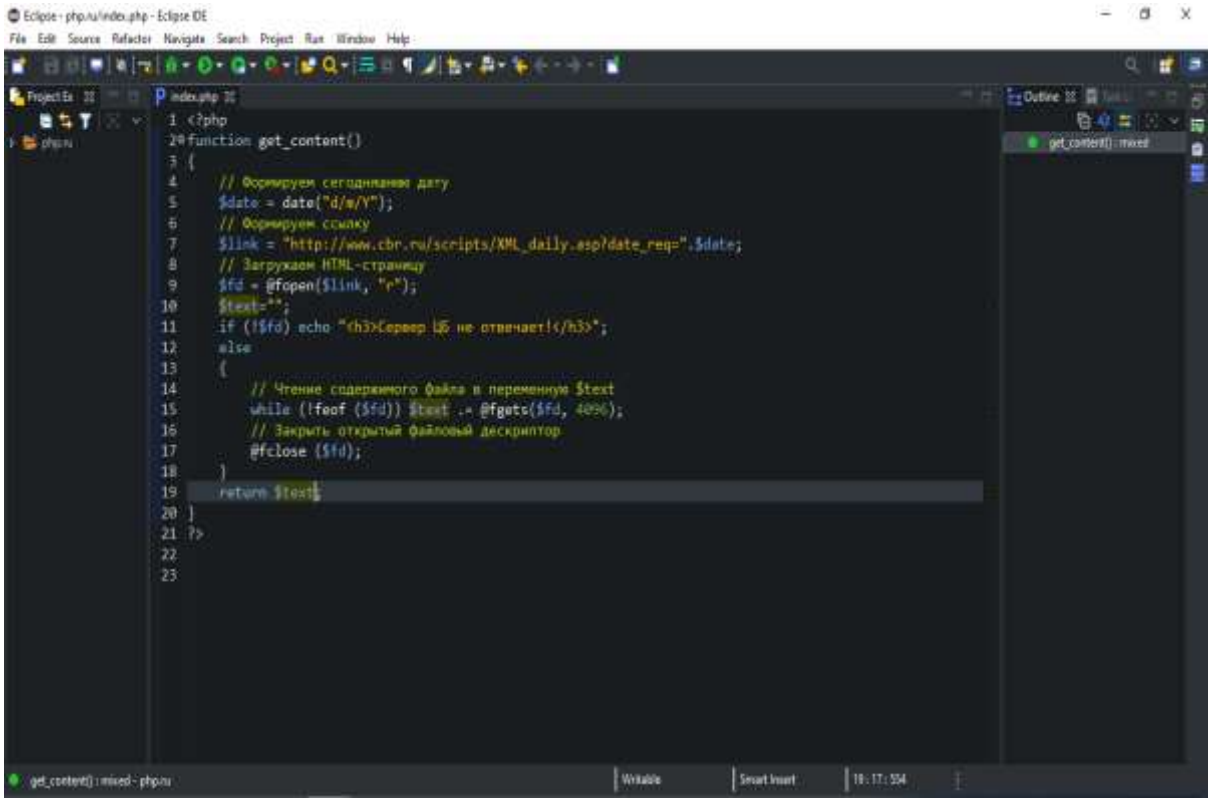



Рисунок 3.2 – Інтерфейс програмного додатку «Eclipse»

Програмний продукт Visual Studio Code є одним із найбільш затребуваних та масово використовуваних редакторів вихідного коду, створеним фахівцями корпорації Microsoft. Цей інструментарій поширюється на повністю безоплатній основі та демонструє бездоганну сумісність з усіма сучасними операційними системами, включаючи Windows, Linux та macOS. За своєю базовою концепцією VS Code функціонує як легковажний текстовий редактор, проте завдяки потужній системі інтеграції різноманітних розширень та плагінів він здатний трансформуватися у повноцінне робоче середовище для взаємодії з будь-якими мовами програмування, що дозволяє успішно застосовувати його для створення різнопланових цифрових продуктів та інноваційних технологічних рішень.

Узагальнені підсумки аналізу та зіставлення технічних можливостей провідних інтегрованих середовищ розробки з урахуванням визначених порівняльних параметрів відображено у табл. 3.2.

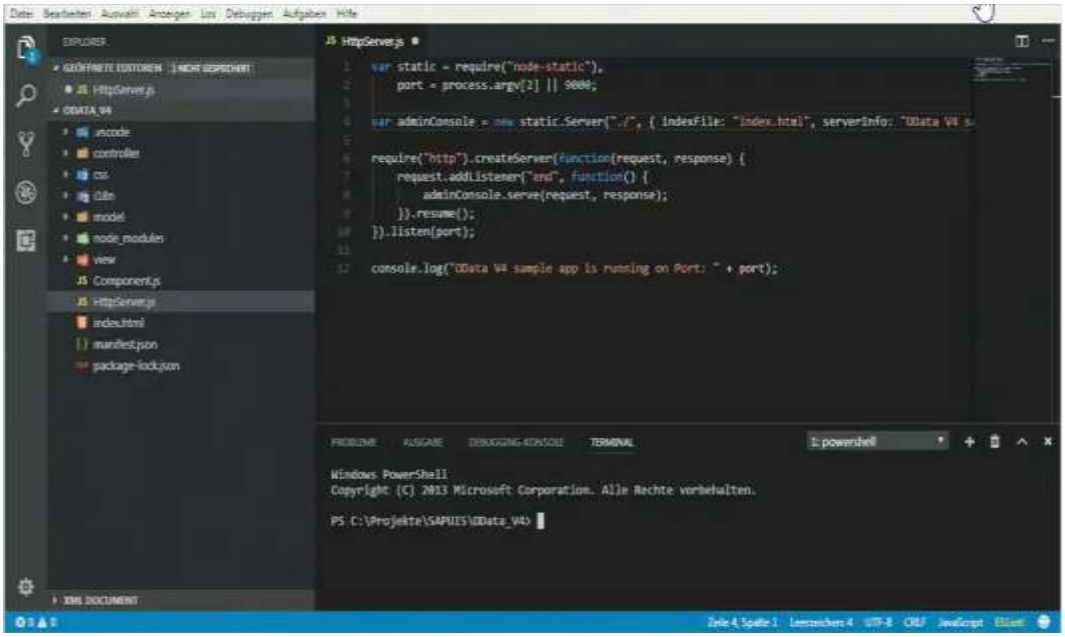


Рисунок 3.3 – Інтерфейс програмного додатку «Eclipse»

З метою максимальної наочності та структурування інформації автори застосували єдину систему графічних маркерів, де позитивний символ вказує на повноцінну присутність функціональної характеристики, негативний знак свідчить про її абсолютну відсутність у базовому програмному рішенні, а комбінований оператор сигналізує про обмежену або часткову реалізацію відповідного критерію в межах платформи.

Таблиця 3.2 – Порівняння інтегрованих середовищ розробки

| Критерій | PhpStorm | Eclipse | Visual Code |
|---|---|---|---|
| Безкоштовність | +- | + | + |
| Кросплатформність | + | + | + |
| Швидкість роботи | + | +- | + |
| Направленість на Web-розробку | + | + | +- |
| Магазин розширень | + | - | + |
| Автодоповнення коду | + | +- | +- |
| Підсумковий результат | 5,5 | 3,5 | 5 |

Проведений комплексний аналіз можливостей різноманітних середовищ розробки для мови програмування PHP, детально відображений у табл. 3.2, дозволяє зробити однозначний висновок, що саме платформа PhpStorm

продемонструвала суттєву перевагу та виявилася найбільш оптимальним вибором з-поміж усіх розглянутих інструментів. Наступним критично важливим кроком для забезпечення надійного збереження всієї системної інформації є підбір архітектури бази даних, для чого доцільно здійснити порівняльне дослідження таких популярних рішень як PostgreSQL, SQLite та MySQL. Перша з них, PostgreSQL, представляє собою просунуту реляційну систему корпоративного класу з відкритим вихідним кодом, яка відрізняється унікальною здатністю ефективно опрацьовувати як традиційні реляційні SQL-запити, так і нереляційні структури даних у форматі JSON. Ця платформа відома своєю винятковою стабільністю, що базується на більш ніж двадцятирічній історії безперервного розвитку глобальної спільноти, яка заклала в її основу найвищі стандарти стійкості, інформаційної цілісності та точності обчислень. Завдяки цим властивостям вона часто виступає головним ядром для зберігання відомостей у масштабних мобільних, аналітичних, геопросторових та мережевих застосунках, проте надмірно розгалужена внутрішня архітектура та високий рівень складності налаштування у зіставленні з конкурентами роблять її не зовсім раціональним вибором для поточних завдань. На противагу їй, SQLite позиціонується як швидка та максимально полегшена однофайлова вбудована система керування базами даних, розроблена на мові C, яка повністю позбавлена серверної складової та дозволяє локально утримувати весь інформаційний масив безпосередньо на пристрої кінцевого користувача без залучення додаткових служб чи сторонніх бібліотек. Специфіка її вбудованого статусу полягає у відмові від класичної парадигми взаємодії клієнта з сервером, оскільки ядро цієї СУБД функціонує не як ізольований операційний процес, а як інтегрована програмна бібліотека, з якою безпосередньо компонується сам застосунок, здійснюючи обмін даними через внутрішні виклики функцій API. Попри всі свої технологічні переваги, SQLite орієнтована переважно на обслуговування локальних програм або компактних стартапів, що робить її використання неефективним та неприйнятним для розгортання повноцінної великої системи.

Програмний продукт MySQL являє собою систему управління

реляційними базами даних від корпорації Oracle, функціонування якої повністю базується на використанні стандартизованої мови структурованих запитів SQL. Ця технологія посідає одне з ключових і найбільш упізнаваних місць у сучасній глобальній екосистемі великих даних, заслужено утримуючи статус найпопулярнішої платформи для структурування інформації, яка знаходить успішне та масове застосування практично в усіх сферах сучасного бізнесу та виробництва. З огляду на це, базове знайомство з принципами роботи MySQL є практично обов'язковим стандартом для будь-якого фахівця, чия діяльність пов'язана з корпоративними інформаційними масивами або загальною сферою інформаційних технологій. Завдяки продуманій архітектурі цього інструменту навіть розробники-початківці, які раніше не мали практичного досвіду взаємодії з реляційними структурами, отримують можливість оперативно проєктувати швидкі, високопродуктивні та захищені сховища відомостей. Окрім того, специфічний програмний синтаксис та інтуїтивно зрозумілі робочі інтерфейси MySQL виступають своєрідним універсальним містком, який суттєво полегшує подальше вивчення інших поширених мов запитів і складніших систем збереження структурованих даних. Саме поєднання виняткової простоти розгортання, високої швидкодії та надійних механізмів безпеки робить MySQL найбільш виправданим та оптимальним вибором для проектування та успішної реалізації бази даних автоматизованої системи управління готелем.

На основі аналітичних даних, представлених у табл. 3.3, стає очевидним, що система управління реляційними базами даних MySQL виступає найбільш оптимальним та ефективним рішенням для проєктування та підтримки серверної частини вебзастосунку для адміністрування готельного бізнесу. Водночас проведений комплексний аналіз різноманітних технологічних інструментів дозволив чітко визначити інший стек для супутнього цифрового продукту, а саме мову програмування Python, інтегроване середовище Visual Studio Code та полегшену СУБД SQLite, які в сукупності є найкращим технологічним вибором для успішної реалізації та розгортання запланованого Telegram-бота.

Таблиця 3.3 – Порівняння систем управління базами даних Для позначення наявності критерію використовується “+”, а для відсутності використовується “-”, а для того часткової наявності використовується “+-”

| Критерій | PostgreSQL | SQLite | MySQL |
|---|---|---|---|
| Потужність та функціонал | + | +- | + |
| Швидкість читання даних | - | + | + |
| Легкість використання | - | + | + |
| Доцільність використання для розробки важких проектів | + | - | + |
| Підсумковий результат | 2 | 2.5 | 4 |

У процесі безпосереднього створення основного вебдодатка було спроєктовано та впроваджено значний масив взаємопов'язаних програмних алгоритмів, що забезпечують його повноцінну життєдіяльність. Серед цього розгалуженого комплексу ключову роль відіграють процедури безпечної автентифікації та авторизації користувачів у системі, механізми інтеграції та первинного внесення нових готельних об'єктів, а також алгоритми реєстрації та додавання постояльців до загальної бази. Крім того, функціональне ядро системи включає важливі математично-аналітичні модулі, відповідальні за точне обчислення поточного фінансового балансу на рахунку готельного комплексу, визначення обсягу грошових коштів на рахунках клієнтів, а також автоматичне виявлення та калькуляцію поточної заборгованості мешканців. Загальну функціональність платформи гармонійно доповнюють спеціалізовані підсистеми генерації звітності, які дозволяють користувачам формувати деталізовані звіти про фінансові витрати постояльця за будь-який обраний часовий інтервал як у класичному текстовому та цифровому форматі, так і у вигляді наочних графічних діаграм.

Процедура початку роботи з програмним комплексом передбачає обов'язкове проходження первинної автентифікації користувача, який має права адміністратора системи. На спеціалізованій сторінці входу співробітник вносить

свої персональні ідентифікаційні відомості у відповідні інтерактивні поля, після чого система автоматично запускає внутрішній процес перевірки та валідації отриманих даних. За умови успішного підтвердження коректності та заповнення всіх необхідних реквізитів додаток формує та надсилає запит до сховища інформації з метою верифікації прав доступу, перевіряючи, чи дійсно обліковий запис із такими параметрами має адміністративний статус. У разі позитивної відповіді від бази даних система надає повний доступ до робочого інтерфейсу, одночасно виконуючи операцію збереження службових даних користувача в поточну сесію для підтримки його авторизованого стану протягом усього періоду роботи.

```
public function store(Request $request)
{
$credentials = $request->validate([
'email' => ['required', 'email'],
'password' => ['required'],
]);
if (Auth::guard('admin')->attempt($credentials)) {
$request->session()->regenerate();
return redirect()->route('adminMainPage');
}
return back()->withErrors([
'email' => 'Помилка при збережені',
]);
}
```

Процедура завершення робочого сеансу та виходу з вебдодатка базується на окремому програмному алгоритмі, який миттєво очищує поточні ідентифікаційні дані та повністю видаляє активного користувача з поточної сесії системи.

```
public function destroy(Request $request)
{
Auth::guard('admin')->logout();
$request->session()->invalidate();
$request->session()->regenerateToken();
return redirect('/');
```

Наступним фундаментальним механізмом у структурі вебдодатка виступає програмний алгоритм, який забезпечує можливість інтеграції та внесення нових готельних об'єктів до загальної бази даних системи. Для ініціації цього процесу відповідальний адміністратор переходить на спеціалізовану робочу сторінку, де послідовно заповнює всі встановлені електронні поля інтерактивної форми, після

чого здійснює безпосередню відправку введеного масиву відомостей на сервер для подальшої обробки та збереження.

```
public function save(Request $request)
{
$request->validate([
'hotelImage.*' => 'mimes:jpeg,jpg,png,svg|max:5000',
'nameFlour.*' => 'required',
'nameSection.*' => 'required'
]);
$modelHotel = new Hotel ();
$modelHotelSection = new HotelSection();
$modelHotelFlour = new HotelFlour();
$imageSaved = new ImageSaver();
$data = $request->all();
if ($request->file('hotelImage') !== null) {
$file = $request->file('hotelImage');
foreach ($file as $key => $item) {
$path[$key] = $imageSaved->upl($item, 'hotel');
}
$data['hotelImages'] = $path; }
$hotelId =  $modelHotel->create($data);
$sectionId =  $modelHotelSection->create($data, $hotelId);
$flourId =  $modelHotelFlour->create($data,$hotelId);
return redirect()->route('adminHotelIndex')->withSuccess('Успішно!');
}
```

Первинним етапом обробки надісланої інформації є обов'язкова перевірка її коректності, за результатами якої, у разі виявлення будь-яких помилок чи невідповідностей, система миттєво генерує та відображає на екрані відповідне сповіщення про неуспішну валідацію. Якщо ж усі внесені відомості повністю відповідають встановленим правилам і критеріям, додаток переходить до наступної фази, де першочергово здійснюється завантаження та фіксація графічних файлів у файловій системі сервера. Цей процес безпечного розміщення зображень контролюється та забезпечується ізольованим спеціалізованим алгоритмом збереження фото.

```
public function upl($file, $dir){
$path = $file->store($dir, 'public');
return $path;
}
```

Після завершення процедури завантаження графічних матеріалів система активує спеціалізовану функцію для фіксації готельного об'єкта, яка здійснює запис усієї супутньої інформації у сховище і за умови успішного виконання цієї операції автоматично генерує та повертає унікальний числовий ідентифікатор ID щойно створеного готелю.

```
$hotelId =  $modelHotel->create($data);
```

Отриманий унікальний числовий ідентифікатор ID виступає обов'язковим технічним параметром для подальшої систематизації та структурування архітектури будівлі, оскільки він безпосередньо транслюється у внутрішню функцію фіксації даних про поверхи та секції готельного комплексу для їх точного прив'язування до конкретного об'єкта.

```
$sectionId =  $modelHotelSection->create($data, $hotelId);
$flourId =  $modelHotelFlour->create($data,$hotelId);
```

Процедура інтеграції нового клієнта до готельної бази вимагає обов'язкової наявності вільних кімнат у житловому фонді, у зв'язку з чим у програмному комплексі було окремо передбачено та реалізовано алгоритм створення і додавання номерів. Після того як уповноважений адміністратор заповнює всі необхідні текстові та числові поля на відповідній вебсторінці, система автоматично запускає процес внутрішньої перевірки інформації. Якщо під час валідації виявляються будь-які помилки, некоректні значення або пропущені обов'язкові параметри, інтерфейс додатка миттєво відображає для користувача текстове сповіщення про неуспішний результат. У протилежному випадку, коли весь масив внесених даних повністю відповідає встановленим технічним вимогам і критеріям валідності, система безперешкодно виконує фінальну операцію збереження сформованого номера у сховищі даних.

```
public function save(Request $request)
{
$request->validate([
'numberNumber' => 'required',
'hotele' => 'required',
], [
'numberNumber.required' => ‘Введіть номер’,
'hotel.required' => 'Виберіть готель
]);
$modelApartament = new Number();
$modelAccount = new PersonalAccounts();
$data = $request->all();
$apartmantId = $modelApartament->creates($data);
if ($data['numberPersonalAccount'] !== null) {
$modelAccount->updateApartament($apartmantId, $data);
}
return redirect()->route('adminNumberCreate')->withSuccess(Успішно!'); }
```

За умови успішного внесення всієї необхідної інформації про житловий фонд стає доступною процедура безпосереднього оформлення та заселення

клієнта в обрану кімнату. Для реалізації цього завдання уповноважений адміністратор відкриває спеціалізовану вебсторінку, де послідовно вносить анкетні відомості у кожне поле електронної форми реєстрації постояльця, після чого ініціює надсилання цих даних на сервер. Наступним обов'язковим кроком програмний комплекс здійснює автоматичну перевірку надісланого масиву інформації, і якщо всі параметри повністю відповідають встановленим правилам валідності, додаток успішно фіксує та зберігає дані нового мешканця у системі, тоді як у разі виявлення помилок чи некоректних значень робочий інтерфейс відразу сповіщає користувача про це за допомогою відповідного інформаційного повідомлення.

```
public function save(Request $request)
{
$request->validate([
'email' => 'required|email|unique:users',
'password' => 'required',
'password2' => 'required|same:password'
],
['email.required' => 'Потрібно заповнити поле«Email (логін)».!',
'email.unique' => 'Email (логін) має бути унікальним.!',
'email.email' => 'Email (логін) повинен мати @.!',
'password.required' => 'Потрібно заповнити поле «Пароль».',
'password2.required' => 'Потрібно заповнити поле «Повторіть пароль».',
'password2.same' => 'Паролі не збігаються',
]
);
$modelUser = new User();
$imageSave = new ImageSaver();
$data = $request->all();
if ($request->file('image') !== null) {

$file = $request->file('image');
if ($data['imageId'] !== null) {
$imageSave->remove($data['imageId']);
}
$data['pathImage'] = $imageSave->upl($file, 'user');
}
$user = $modelUser->creates($data);
return redirect()->route('userIndex')->withSuccess(Успішно!'); }
```

З метою забезпечення стабільного та коректного функціонування готельного комплексу виникає критична необхідність у веденні точного і безперервного обліку всіх фінансових потоків, які надходять до системи або вилучаються з неї. Механізм фіксації грошових надходжень на баланс готелю, а також на індивідуальні рахунки клієнтів базується на спеціально розробленому алгоритмі приходу коштів. У ситуації, коли оплата різноманітних послуг

здійснюється безпосередньо на рецепції готелю, уповноважений адміністратор зобов'язаний невідкладно відобразити цю операцію у внутрішній прибутковій відомості. Для успішного проведення платежу співробітник використовує спеціалізовану вебсторінку, де вносить точну суму транзакції та, у разі потреби зарахування грошей на баланс конкретного мешканця, обирає відповідне ім'я постояльця з бази даних, після чого підтверджує надсилання електронної форми, що ініціює автоматичне збереження та фіксацію інформації про прихід коштів на стороні сервера.

```
public function store(Request $request)
{
$data = $request->all();
$modelTransaction = new AccountTransaction();
$modelTransaction->store($data);
return redirect()->back()->withSuccess('Успішно!');
}
```

Процес фіксації видаткових фінансових операцій та списання коштів із каси здійснюється шляхом формування спеціалізованої квитанції, яка безпосередньо прив'язується до індивідуального рахунку конкретного постояльця. На початковому етапі цієї процедури уповноважений адміністратор вносить у систему базові параметри транзакції, де послідовно вказує точну суму списання, ідентифікаційні відомості клієнта, а також конкретне найменування наданої послуги, яка є підставою для виписки цього розрахункового документа.

```
public function store(Request $request)
{
$modelInvoice = new Invoice();
$data = $request->all();
$invoice = $modelInvoice->store($data);
return redirect()->back()->withSuccess('Успішно!');
}
```

Програмний механізм, відповідальний за точний розрахунок загального фінансового залишку безпосередньо в касі готелю, базується на послідовному аналізі грошових потоків. На початковому етапі виконання цього алгоритму система здійснює пряме звернення до сховища інформації, звідки в автоматичному режимі вилучає повні масиви даних щодо всіх прибуткових операцій та видаткових транзакцій, після чого отримані сумарні показники

фіксуються у двох окремих службових змінних для подальшого обчислення різниці.

```
public function getCashboxIn()
{
$ins = $this->where([['type', '=', 'in'], ['status', '=', 10]])->sum('amount');
return $ins;
}
public function getCashboxOut()
{
$outs = $this->where([['type', '=', 'out'], ['status', '=', 10]])-
>whereNull('account_id')->sum('amount');
return $outs;
}
```

Після цього, визначається загальна сума яка є у касі.

```
public function getCashboxBalance()
{
$balance = $this->getCashboxIn() - $this->getCashboxOut();
return $balance;
}
```

Алгоритм визначення суми на рахунку у постояльців. Спочатку із бази даних вибираються постояльці.

```
public function getBalanceTotal()
{
$accountsQuery = $this->select('personal_accounts.*')->join('account_transactions',
'personal_accounts.id', '=', 'account_transactions.account_id')

->whereNotNull('account_transactions.account_id')->whereNull('invoice_id')-
>
groupBy('account_transactions.account_id')->get();
$total = 0;
foreach ($accountsQuery as $account){
$balance = $account->getBalance();
if ($balance > 0) {
$total += $balance;
} }
return $total; }
```

На наступному етапі програмний алгоритм запускає послідовний ітераційний перебір усього списку зареєстрованих мешканців за допомогою циклу, в межах якого для кожного окремого клієнта викликається спеціалізована функція getBalance(), що в автоматичному режимі обчислює та повертає поточний фінансовий баланс його персонального рахунку.

```
public function getBalance()
{
$ins = $this->accountTransactions()->where([['type', '=', 'in'], ['status', '=', 10]])-
>sum('amount');
$outs = $this->accountTransactions()->where([['type', '=', 'out'], ['status', '=', 10]])-
>sum('amount');
$balance = $ins - $outs;
return $balance;
}
```

Після завершення роботи функції getBalance() система аналізує отримане значення, і у випадку, якщо фінансовий залишок на рахунку є позитивним, цей показник автоматично підсумовується та додається до службової змінної $total, яка акумулює в собі сукупний обсяг грошових коштів, наявних на балансах усіх клієнтів. Своєю чергою програмний алгоритм, призначений для точного обчислення сумарного обсягу заборгованості серед мешканців готелю, на першому етапі свого виконання здійснює звернення до бази даних для формування та вилучення повного списку зареєстрованих постояльців.

```
public function getBalanceDebtTotal()
{
$accountsQuery =  $this->select('personal_accounts.*')->join('Number', 'Number.id',
'=', 'personal_accounts.Number_id')->with('Number')

->whereIn('Number.Hotels_id', [1, 3])->get()
$total = 0;
foreach ($accountsQuery as $account){
$balance = $account->getBalance();
if ($balance < 0) {
$total += $balance;
}
}
return $total;
}
```

На наступному етапі програмний алгоритм запускає послідовний ітераційний перебір усього списку зареєстрованих мешканців за допомогою циклу, в межах якого для кожного окремого клієнта викликається спеціалізована функція getBalance(), що в автоматичному режимі обчислює та повертає поточний фінансовий баланс його персонального рахунку. Після завершення роботи функції getBalance() система аналізує отримане значення, і у випадку, якщо фінансовий залишок є негативним, цей від'ємний показник підсумовується та додається до службової змінної $total, яка акумулює в собі сукупний обсяг

заборгованості серед усіх постояльців готелю. Окрім серверної логіки, архітектура застосунку включає декілька спеціалізованих клієнтських алгоритмів, які забезпечують динамічне додавання необхідних інтерфейсних елементів на вебсторінку без її повного перезавантаження. Зокрема, у момент взаємодії користувача з елементом керування, що має ідентифікатор ID add-tariff-service, програма першочергово здійснює повне очищення цільової області від застарілих компонентів, після чого ініціює асинхронний ajax запит до сервера. Отриманий у результаті цієї операції масив актуальних даних автоматично розподіляється по відповідних позиціях шаблону html коду, який у фінальній фазі безперешкодно інтегрується безпосередньо в активне вікно браузера.

```
$(document).on('click', '#add-tariff-service', function (e) {
var rows = $(this).parents('.invoiceService');
$('.invoiceService').remove();
console.log($('#tariff_id').val());
varCount++;
$.ajax({
url: '/tariff/get-tariff-id/' + $('#tariff_id').val(),
type: "GET",
headers: {

'X-CSRF-Token': '{{csrf_token()}}'
},
dataType: 'json',

success: function (json) {
json['tariffServices'].forEach(function (tariffService, key) {
console.log(tariffService);
$('#invoiceService').append($node);

$('.invoiceServiceId option[value=' + tariffService["service_id"] +
']').attr("selected", "selected");
json['services'].forEach(function (service, key) {
if (service["id"] == tariffService["service_id"]) {
$('.unit option[value=' + service["service_unit_id"] + ']').attr("selected",
"selected");
}
}) })
} }); });
});
```

### 3.3. Тестування програми

Службовий процес перевірки працездатності цифрових продуктів, відомий

у професійному середовищі як software testing, відображає собою комплексний аналіз або безпосередню експлуатацію програмного забезпечення, спрямовану на своєчасне виявлення різноманітних прихованих дефектів і помилок. Зазвичай саме ця фаза загального життєвого циклу створення програмних рішень вимагає найбільшої кількості часу та залучення значних інтелектуальних і технічних ресурсів. У сучасній інженерії виділяють два фундаментальні напрямки реалізації цього процесу, якими є статичне та динамічне тестування. Перший напрямок, а саме статичне тестування, об'єднує в собі групу методологій, що дозволяють детально вивчити внутрішню структуру застосунку та перевірити синтаксис і логіку коду без фактичного запуску самої програми в операційному середовищі. Цей підхід традиційно поділяється на дві ключові форми, представлені статичними оглядами та інструментальним статичним аналізом. Метод статичних оглядів орієнтований переважно на ручний пошук архітектурних розбіжностей, а також на глибоке вивчення супутньої технічної документації чи вихідних текстів програм. Серед найбільш поширених проблем, які вдається виявити завдяки таким оглядам, можна назвати суттєві відхилення від затверджених стандартів кодування, відсутність або неповне відображення критичних вимог до продукту, а також загальні вади в дизайні системи. Своєю чергою, статичний аналіз покладається на використання спеціалізованих автоматизованих засобів для сканування інфраструктури на предмет наявності потенційних уразливостей, схильних до регулярних збоїв методів написання коду, некоректних внутрішніх залежностей, помилок у валідації вхідних даних та різноманітних логічних прорахунків. З огляду на високий рівень складності та розгалуженості архітектури сучасних мов програмування, процедури статичного аналізу практично завжди довіряють автоматизованому інструментарію, тоді як класичні статичні огляди традиційно здійснюються фахівцями у ручному режимі.

Другий фундаментальний напрямок, відомий як динамічне тестування, являє собою методологію дослідження програмного забезпечення, за якої детально аналізується безпосередня поведінка системи під час її фактичного

виконання. Головна відмінність цього підходу від статичних методів полягає в обов'язковій компіляції та запуску аналізованого вихідного коду, що передбачає подачу в систему певних вхідних параметрів і подальшу верифікацію отриманих результатів з метою підтвердження повної працездатності продукту. У межах цього напрямку існує кілька ключових концепцій, серед яких особливе місце посідає тестування методом білого ящика. Цей підхід базується на глибокому зануренні у внутрішню архітектуру та логіку застосунку за межі його зовнішніх інтерфейсів, оскільки фахівець має повний доступ до вихідного коду і може створювати тестові скрипти, безпосередньо пов'язані з внутрішніми бібліотеками програмного забезпечення. Повною протилежністю є тестування методом чорного ящика, за якого взаємодія із системою здійснюється виключно через загальнодоступні інтерфейси користувача або зовнішні канали зв'язку, що дозволяють іншим процесам чи комп'ютерам підключатися до платформи для її перевірки. Процедура проектування тест-кейсів за такою технологією спирається суто на всебічний аналіз функціональних або нефункціональних специфікацій компонента, залишаючи внутрішній устрій системи повністю ізольованим від уваги тестувальника. Наостанок, важливим аспектом оцінки цифрового продукту є тестування зручності використання, яке фокусується на визначенні ступеня ергономічності, простоти освоєння, зрозумілості та загальної естетичної привабливості інтерфейсу для кінцевих користувачів у межах чітко заданих умов експлуатації.

Процес функціонального тестування передбачає детальний аналіз усіх операційних характеристик програмного забезпечення та виявлення можливих суперечностей між фактичною поведінкою впроваджених механізмів і очікуваними результатами, задекларованими у технічній специфікації та бізнес-вимогах. За своєю суттю такий підхід повністю імітує реальну повсякденну експлуатацію системи кінцевим користувачем. Порівняльний аналіз існуючих методологій перевірки показав, що для всебічного контролю інтерфейсу та функціональних можливостей створюваного вебдодатка найбільш доцільно застосовувати метод чорного ящика, оскільки чітко сформульовані вимоги

дозволяють без зайвих труднощів спроектувати ефективні тест кейси для верифікації кожного окремого елемента системи. При дослідженні специфіки тестування інтерфейсної верстки особливу увагу необхідно приділяти візуальній складовій цифрового продукту. Це включає виявлення будь-яких порушень у відображенні структурних блоків інтерфейсу, усунення невідповідностей обраної кольорової гами та ретельну перевірку локалізованих версій вебсайту на предмет коректності перекладу. Крім того, важливим етапом є контроль точної відповідності готового інтерфейсу вихідному графічному макету, створеному в програмах на кшталт PhotoShop. Також обов'язково аналізується стабільність відображення елементів при зміні масштабів сторінки в діапазоні від сімдесяти п'яти до ста п'ятдесяти відсотків, оцінюється правильність інтерактивного підсвічування полів у разі виникнення помилок введення та перевіряється поведінка сайту на екранах із різною роздільною здатністю з урахуванням коректності роботи смуг прокручування. Для практичної реалізації таких перевірок без візуальних дефектів фахівці використовують спеціалізований інструментарій браузерів, наприклад, вбудовану функцію адаптивного дизайну, доступну через меню інструментів веб-розробника у браузері Firefox, або сторонні розширення на зразок Resolution Test Plugin в середовищі Chrome.

У межах аналізу технічного стану вебресурсу критично важливим аспектом є детальна перевірка доступності елементів та повної відсутності JS помилок у процесі взаємодії з інтерфейсом. Під час цього етапу тестувальник верифікує коректність спрацьовування всіх інтерактивних компонентів сайту, включаючи внутрішні та зовнішні посилання, адреси електронної пошти, кнопки та графічні іконки. Обов'язковою умовою якісної реалізації інтерфейсу є автоматична зміна вигляду курсору миші при його наведенні на будь-які клікабельні зони, тоді як для статичних об'єктів така поведінка має бути відсутня. Складні або нетипові інтерактивні елементи мають забезпечуватися зрозумілими текстовими підказками для полегшення сприйняття інформації користувачем. Крім того, у разі навмисного вимкнення відображення графіки у браузері, замість картинок на екрані повинні з'являтися їхні текстові описи, виконані невеликим

шрифтом сірого кольору, що можна легко проконтролювати за допомогою спеціалізованого розширення Web Developer через інструмент заміни зображень на їхні Alt відомості. Не менш суттєвим фактором стабільності платформи виступає її працездатність в умовах повністю вимкненого JS, де всі життєво важливі та критичні функції сайту повинні залишатися доступними для користувача, що перевіряється деактивацією всіх скриптів у згаданому плагіні Web Developer. Наступний комплекс перевірок спрямований на оцінку надійності верстки та коректності обробки текстового наповнення, де фахівці аналізують поведінку інтерфейсу при введенні екстремально великих або мінімальних обсягів інформації в електронні форми, а також досліджують стійкість розташування контентних блоків при їхній штучній зміні місцями через інструменти редагування HTML коду. Також здійснюється всебічний контроль роботи стилів під час відображення текстових масивів, що містять заголовки різного рівня, структуровані абзаци, суцільний текст без поділу на блоки чи інтегровані графічні зображення. Оскільки в традиційних умовах взаємодія користувача із системою відбувається безпосередньо через вебпереглядач, особливої актуальності набуває перевірка сумісності з різними програмами для серфінгу. Кросбраузерність визначає фундаментальну здатність сайту зберігати ідентичний візуальний вигляд, структуру та повноцінний набір функцій відповідно до початкових технічних завдань незалежно від конкретного браузера, у якому його відкриває кінцевий споживач.

Кросбраузерне тестування відображає собою специфічний різновид нефункціонального контролю якості, спрямований на всебічну перевірку коректності та стабільності роботи вебресурсу в умовах, коли кінцеві користувачі отримують до нього доступ через найрізноманітніші програмно-апаратні платформи. Цей процес передбачає аналіз поведінки сайту в умовах використання великої кількості варіативних комбінацій сучасних вебпереглядачів та операційних систем, охоплюючи взаємодію додатка з такими поширеними браузерами, як Firefox, Chrome, Edge і Safari, розгорнутими в середовищі актуальних операційних систем Windows, macOS, iOS та Android.

Окрім програмного забезпечення, об'єктом дослідження стає сумісність із різнотипними фізичними пристроями, оскільки відвідувачі мають можливість вивчати контент та взаємодіяти з інтерфейсом сайту за допомогою смартфонів, планшетів, настільних комп'ютерів або портативних ноутбуків. Важливим розширенням цього підходу є верифікація доступності цифрового продукту для спеціалізованих допоміжних інструментів, що гарантує безперешкодне сприйняття інформації людьми з інвалідністю за допомогою програм зчитування з екрана. Головна філософія та мета впровадження такого тестування полягає в ретельній підготовці та доставці програмних релізів, які повністю нівелюють технічні розбіжності між різними середовищами виконання, що є критично важливим фактором для забезпечення абсолютно однорідного, передбачуваного та якісного досвіду взаємодії користувача з платформою в умовах постійного розширення глобального ринку браузерів і мобільних пристроїв.

Під час проведення кросбраузерного тестування детальному аналізу піддається кілька фундаментальних категорій функцій програмного продукту, першою з яких є основна функціональність, що відповідає за стабільну роботу базових можливостей системи у переважній більшості комбінацій операційних систем та вебпереглядачів. У межах цього напрямку інженер перевіряє коректність функціонування всіх інтерактивних діалогових вікон і навігаційних меню, оцінює здатність полів електронних форм безпомилково приймати та обробляти введену інформацію після проходження етапу валідації, а також контролює правильність роботи з файлами cookie першої сторони, від яких безпосередньо залежать механізми персоналізації інтерфейсу. Додатково тут аналізується точність і безперебійність реагування елементів сайту на сенсорне введення при використанні різноманітних мобільних телефонів або планшетів. Наступна категорія охоплює загальний дизайн вебресурсу і покликана гарантувати, що фактичне візуальне відображення текстових шрифтів, графічних зображень та загального макета сторінок стовідсотково відповідає технічним специфікаціям, затвердженим дизайнерською командою. Не менш важливим аспектом є доступність платформи, яка передбачає обов'язкову перевірку сайту

на повну відповідність міжнародним правилам доступності вебвмісту WCAG, що дозволяє людям із різними фізичними можливостями безперешкодно взаємодіяти з контентом. Фінальна категорія, представлена відповідністю, фокусується на верифікації гнучкості та адаптивності інтерфейсу, підтверджуючи, що розроблений дизайн є плавним і автоматично підлаштовується під будь-які розміри або просторові орієнтації екрана пристрою.

Невіддільним елементом сучасних вебресурсів виступають різноманітні форми для заповнення, взаємодію з якими відвідувачі здійснюють безпосередньо через інтерфейс браузера. Проте саме ці компоненти дуже часто стають основним джерелом виникнення програмних помилок, які в разі потрапляння на фінальний етап продакшені здатні спричинити серйозні фінансові втрати та завдати відчутної шкоди репутації компанії.

Для ефективного запобігання появі дефектів у робочому середовищі розроблено комплекс послідовних кроків, де першочергово здійснюється ретельна перевірка обов'язковості заповнення ключових полів і наявності біля них відповідних візуальних маркерів. Після внесення інформації та відправки форми тестувальник обов'язково переконується, що надіслані відомості обробляються строго за запланованим сценарієм. У ситуаціях, коли дані мають фіксуватися в базі, виконується верифікація коректності завершення цієї операції всередині сховища, для чого за потреби залучають розробника у випадку обмежених прав доступу чи недостатнього рівня володіння мовою SQL.

Значне підвищення якості перевірок досягається завдяки регулярному використанню спеціалізованих чит-листи для тестування форм. Окремий акцент під час аналізу робиться на оцінці зрозумілості та інформативності системних сповіщень, які з'являються на екрані при спробі відправити незаповнену форму і вказують користувачеві на необхідність виправлення помилок. Крім того, пильна увага приділяється технічній реалізації екранування символів у текстових полях, оскільки відсутність належної фільтрації створює критичні вразливості для безпеки додатка та конфіденційності клієнтів. Такий захист повинен

функціонувати не лише на рівні інтерфейсу користувача, а й в обов'язковому порядку дублюватися на стороні сервера, оскільки клієнтські обмеження можна легко обійти або повністю відключити всього за кілька кліків за допомогою спеціальних інструментів на кшталт плагіна Web Developer Toolbar – Forms.

У межах перевірки зворотного зв'язку тестувальник обов'язково контролює, щоб після успішного завершення операції на адресу користувача надходив офіційний підтверджуючий лист із коректно вказаними даними відправника, а сам зміст цього повідомлення повністю відповідав встановленим технічним вимогам та містив виключно працездатні посилання. Фінальною складовою цього процесу є активне залучення додаткового допоміжного інструментарію, яскравим представником якого є розширення Web Developer Toolbar, що дозволяє суттєво оптимізувати та прискорити всебічний аналіз форм.

Паралельно з оцінкою клієнтської частини критично важливим етапом перевірки інтернет-ресурсів є комплексне тестування сервера. Дослідження логіки веб-додатка, розгорнутого безпосередньо на стороні веб-сервер, можна ефективно реалізувати навіть без задіяння графічного інтерфейсу клієнта, проте такий підхід вимагає від фахівця високого рівня профільних технічних знань, специфічних навичок та активного застосування спеціалізованого інструментарію. У цьому контексті роботу з веб-сервер доцільно детально проаналізувати крізь призму процедур інсталяції системи, а також з погляду подальшого навантажувального тестування.

Безпосередній процес встановлення програмного забезпечення на веб-сервер виступає обов'язковою передумовою для початку будь-яких подальших перевірок. За своєю технічною суттю ця процедура має багато спільного з тестуванням класичних настільних додатків, проте вона характеризується низкою специфічних нюансів, які необхідно обов'язково враховувати та ретельно контролювати, що стає особливо актуальним у разі роботи з програмними продуктами, призначеними для локального розгортання на власних серверних потужностях кінцевих користувачів.

Оскільки успішна інсталяція більшості сучасних веб-додатків неможлива

без глибокого розуміння системного адміністрування операційної системи, інженеру з якості важливо виконати тестове розгортання продукту в умовах декількох різних серверних конфігурацій. Найбільш оптимальним рішенням тут є вибір технологій, які мають найвищу популярність серед цільової аудиторії, для визначення яких заздалегідь проводиться окреме аналітичне дослідження.

Під час проведення інсталяційних робіт здійснюється обов'язкова верифікація того, що всі файли програми коректно розподіляються по цільових каталогах, інформаційні структури створюються саме у визначеній базі даних, а сама система чітко застосовує обраний розробниками префікс та суворо дотримується решти параметрів заданої конфігурації. Також інженер повинен переконатися, що архітектура застосунку дозволяє безперешкодно виконувати процедуру встановлення як локально безпосередньо з хоста, так і з використанням каналів віддаленого доступу.

Якщо розробниками передбачено інтерактивний сценарій інсталяції, де кожен окремий вибір користувача на поточному етапі безпосередньо коригує або змінює зміст усіх наступних кроків, тестувальнику доведеться повністю пройти та детально задокументувати абсолютно всі існуючі логічні гілки цього процесу, адже етап встановлення є першим практичним досвідом взаємодії клієнта з готовою програмою.

Окреме місце в цій стратегії посідає проектування негативних тестів, у межах яких штучно моделюються умови критичного дефіциту апаратних ресурсів, включаючи жорсткі обмеження доступного дискового простору чи обсягу оперативної пам'яті. У цих же сценаріях відтворюються ситуації з хронічною нестачею системних привілеїв облікового запису або виконується примусове агресивне переривання інсталяційного процесу в момент його найбільш активної фази, наприклад, шляхом раптового відправлення сервера на повне перезавантаження.

Процедура, відома як тестування навантаження, орієнтована на штучне моделювання умов одночасної та інтенсивної взаємодії великої кількості кінцевих споживачів із програмним продуктом. Цей специфічний різновид

контролю якості реалізується із залученням спеціалізованого програмного інструментарію, яскравим прикладом якого є утиліта jMeter, де ключовим завданням виступає точне проектування профільних моделей поведінки та генерація критичних обсягів трафіку для виявлення граничних експлуатаційних можливостей веб-додатка або безпосередньо самого сервера в режимі масового підключення.

Уся накопичена під час таких випробувань інформація піддається детальному експертному аналізу, що дозволяє локалізувати так звані тугі шийки, а також чітко окреслити кінцеві апаратні та програмні ліміти системи, які згодом беруться за основу при оптимізації інфраструктури задля забезпечення максимальної стабільності веб-сервера та розгорнутого на ньому цифрового рішення.

Реальним прикладом із практичної діяльності може слугувати ситуація навколо масштабного комерційного продукту, архітектура якого тривалий час успішно опрацьовувала різноманітні формати електронних угод. Після впровадження чергового оновлення з новим типом контрактів уся система наступного дня зазнала повного краху, що спровокувало критичне перевантаження служби технічної підтримки через масові скарги клієнтів. Головною причиною цього інциденту став серйозний прорахунок на етапі контролю якості, оскільки інженери обмежилися перевіркою стабільності платформи в межах кількох десятків тисяч документів, не врахувавши реальний сценарій життєдіяльності системи, де на практиці утворилися сотні тисяч або навіть мільйони паралельних запитів. Саме своєчасне та правильно організоване тестування навантаження покликане виявляти подібні приховані архітектурні загрози ще на стадії попередньої перевірки до моменту офіційного релізу.

Фундаментальним складником інфраструктури сучасних інтернет-ресурсів виступає база даних, яка забезпечує надійне довгострокове збереження всієї операційної інформації застосунку. Для того щоб сховище функціонувало максимально ефективно та безпечно, у процесі його перевірки інженерам з якості необхідно сфокусуватися на кількох критично важливих аспектів.

Першочергово контролюється автентичність інформації, що означає, що будь-які відомості, внесені користувачем через графічний інтерфейс, мають фіксуватися в таблицях у своєму вихідному оригінальному вигляді без жодних несанкціонованих спотворень чи автоматичних змін. Паралельно з цим перевіряється консистентність даних, яка гарантує, що збережені масиви інформації відображаються абсолютно ідентично в усіх пов'язаних модулях і розділах програми, якщо тільки специфічна бізнес-логіка продукту прямо не передбачає іншого формату виведення.

Окремим етапом аналізу є верифікація внутрішньої архітектури сховища, де назви всіх створених таблиць, типи полів та загальна структура бази даних проходять ретельну звірку на повну відповідність затвердженій проектній документації. Разом із цим вагома увага приділяється продуктивності системи, де фахівці здійснюють безперервний моніторинг працездатності, оцінюють швидкість обробки складних запитів та перевіряють наявність достатнього ліміту паралельних підключень, оскільки стабільність цього вузла є одним із ключових факторів успіху всього процесу тестування.

Під час дослідження операцій видалення інформації слід враховувати, що очищення запису в таблиці далеко не завжди означає фізичне усунення сутності з диска, оскільки в багатьох архітектурах застосовується механізм псевдо-видалення, правильність логічного налаштування якого також потребує обов'язкової інструментальної перевірки.

Наостанок, для створення адекватних умов тестування настійно рекомендується заповнювати чисту базу даних на тестовому стенді великим масивом випадково згенерованої інформації або використовувати реальний технічний дамп, імпортований безпосередньо з робочого сервера. У разі використання реальних відомостей обов'язковим попереднім кроком є процедура обфускації даних для захисту конфіденційності користувачів. Якщо наявна кваліфікація тестувальника або встановлені політики безпеки не дозволяють реалізувати цей підготовчий процес самостійно, до вирішення завдання в обов'язковому порядку залучаються розробники або профільні фахівці з

управління інфраструктурою.

Мережевий обмін даними всередині системи базується на безперервному надсиланні структурованих повідомлень, де кожна офіційна відповідь сервера формується виключно як реакція на відповідний вхідний запит, надісланий з боку клієнта. У традиційній архітектурі будь-який подібний інформаційний пакет обов'язково об'єднує в собі три фундаментальні складники, представлені стартовою лінією, набором заголовків та безпосереднім текстом самого повідомлення.

Під час детального аналізу мережевого трафіку інженеру з контролю якості першочергово необхідно сфокусуватися на вивченні специфічних методів та числових кодів статусу, які містяться безпосередньо у стартовому рядку транзакції. Серед усього розмаїття мережевих команд найбільш поширеними та затребуваними є GET, HEAD і POST.

Перший із них, а саме метод GET, застосовується у випадках, коли клієнтському додатку потрібно затребувати та отримати певний інформаційний вміст, що зберігається на віддаленому сервері. Наступний метод HEAD за своєю технічною логікою практично повністю ідентичний попередньому, проте ключова відмінність полягає в тому, що відповідь сервера на такий запит повністю позбавлена тіла, а його головна практична цінність зосереджена на швидкому та економічному отриманні базових метаданих про стан ресурсу чи поточний статус системи. Своєю чергою, метод POST розроблений для безпечного та об'ємного передавання інформації на сторону сервера, причому весь масив даних інкапсулюється безпосередньо в тілі повідомлення, що суттєво відрізняє його від механізмів GET. Наочним прикладом роботи цього методу є публікація статті, коли весь текстовий матеріал вміщується всередину запиту POST і після успішного опрацювання серверною логікою з'являється на вебресурсі. Окрім згаданої трійки, у сучасних вебтехнологіях активно використовуються й інші спеціалізовані команди, серед яких варто виділити PUT, DELETE, CONNECT, TRACE та PATCH.

Той факт, що абсолютно всі внутрішні та зовнішні компоненти сучасного

застосунку взаємодіють між собою на базі єдиного протоколу HTTP, відкриває перед тестувальником широкі можливості. Це дозволяє не лише здійснювати повний моніторинг інформаційного обміну та глибше вивчати операційну логіку продукту для її подальшого порівняння з технічними вимогами, а й надає інструменти для прямого моделювання поведінки програми шляхом ручного коригування та самостійного проектування надісланих запитів.

Для практичного створення, перехоплення та ручного керування мережевими пакетами зазвичай використовуються такі спеціалізовані програмні інструменти, як Fiddler або Postman. Завдяки функціональним можливостям платформи Fiddler інженер з контролю якості отримує зручний механізм для безперервного відстеження абсолютно всіх вхідних і вихідних транзакцій між клієнтом і сервером, що дозволяє детально аналізувати їхні внутрішні параметри, оперативно вносити корективи у вихідні структури та повторно надсилати модифіковані запити з метою всебічної оцінки поведінки системи в нестандартних умовах.

У процесі такого інструментального аналізу особливу увагу необхідно приділяти кільком критичним технічним чинникам, серед яких першочерговим є контроль застосування правильного методу передачі даних. Якщо під час відправки користувачем розлогого текстового повідомлення система замість очікуваної об'ємної команди генерує виключно базовий запит GET, це свідчить про наявність явного архітектурного дефекту.

Крім того, за звичайних умов взаємодія з однією і тією ж інтерактивною клавішею має викликати ідентичну реакцію системи, проте в інженерній практиці трапляються серйозні прецеденти, коли внаслідок збоїв у логіці клієнтських скриптів JavaScript помилково переписував структуру запиту, дублюючи або підміняючи призначення сусіднього елемента керування.

Саме тому для успішного виявлення подібних аномалій тестувальник повинен чітко розуміти технічний зміст і структуру кожної надісланої мережевої команди. Формат таких повідомлень є досить прозорим для освоєння, а структура запитів типу GET взагалі виглядає логічно зрозумілою навіть для

пересічного користувача. Своєчасний і глибокий синтаксичний аналіз мережевих запитів відкриває можливість локалізувати та усунути приховані програмні помилки набагато швидше, ніж за умови тривалого і неефективного пошуку їхніх візуальних проявів безпосередньо через графічний інтерфейс програми.

Перед початком безпосередньої перевірки створеного цифрового продукту за методологією чорного ящика виникає технічна необхідність у попередньому проектуванні та формуванні спеціалізованих тест-кейси. З метою всебічного контролю працебності вебдодатка для управління готельним комплексом було розроблено серію таких перевірочних сценаріїв, де під першим номером виступає тест-кейс, призначений для верифікації процедури авторизації користувача в системі. Цей алгоритм передбачає послідовне виконання кількох кроків, що починаються з переходу на цільову сторінку входу, після чого здійснюється безпосереднє введення автентифікаційних відомостей у відповідні поля електронної форми та активація керуючої кнопки входу. Головною метою та очікуваним результатом успішного завершення всіх етапів цього тест-кейсу є безперешкодний автоматичний перехід користувача на головну сторінку системи, для реалізації чого на практиці у вікні інтерфейсу прописуються необхідні вхідні дані.

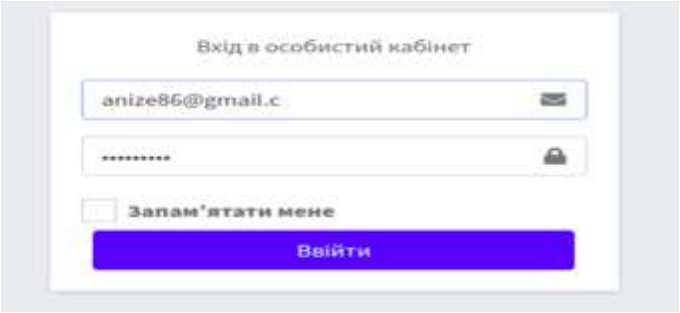



Рисунок 3.4 – Дані для авторизації

У момент активації елемента керування Ввійти клієнтський застосунок ініціює передачу відомостей на сервер, де запускається процедура верифікації автентифікаційних даних, у результаті успішного проходження якої за умови абсолютно коректного заповнення всіх полів форми система виконує

автоматичне перенаправлення користувача на головну сторінку вебдодатка.

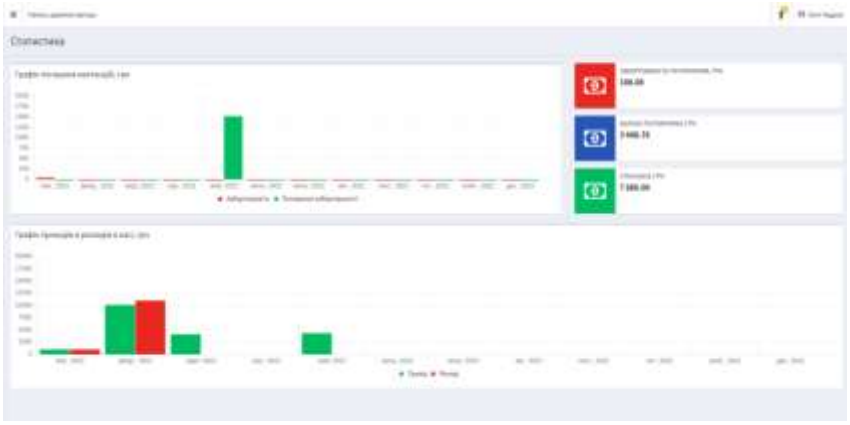


Рисунок 3.5 – Результат виконання тест-кейсу №1

Проведене випробування показало повну відповідність фактичного результату заздалегідь задекларованим очікуванням, що дозволяє офіційно вважати тест-кейс під першим номером успішно виконаним. Наступним етапом перевірки став тест-кейс для верифікації функції створення та інтеграції нової готельної одиниці в загальну базу даних сайту. Цей перевірочний сценарій передбачає першочергове відвідування спеціалізованої сторінки додавання нового об'єкта, де у відповідні текстові поля форми вносяться конкретні демонстраційні відомості, а саме назва Готель 1 та адреса вул. Келицька 98. Після завершення етапу заповнення інформаційних блоків здійснюється безпосередня активація керуючої кнопки Зберегти, а фінальною умовою успішного проходження цього алгоритму є генерація та виведення на екран офіційного системного сповіщення про позитивне завершення операції у вигляді текстової відповіді Успішно.

За результатами практичного виконання перевірочного сценарію вебдодаток згенерував офіційне сповіщення Успішно, а в загальній структурі бази даних та інтерактивній таблиці відобразився створений об'єкт із точно вказаними нами параметрами. Оскільки отриманий фактичний результат повністю тотожний заздалегідь визначеному очікуваному показнику, тест-кейс під другим номером визнається успішно пройденим. Наступним кроком контролю якості виступає тест-кейс, розроблений для детальної перевірки

процедури інтеграції нової житлової одиниці в систему. Цей алгоритм дій вимагає від користувача першочергового переходу на цільову сторінку додавання нового номеру, де у відповідні інтерактивні форми вноситься весь комплекс необхідних технічних і текстових характеристик об'єкта. Після заповнення інформаційного масиву здійснюється активація керуючого елемента Зберегти, а фінальним критерієм коректного завершення операції є автоматичне отримання від сервера позитивної системної відповіді Успішно.

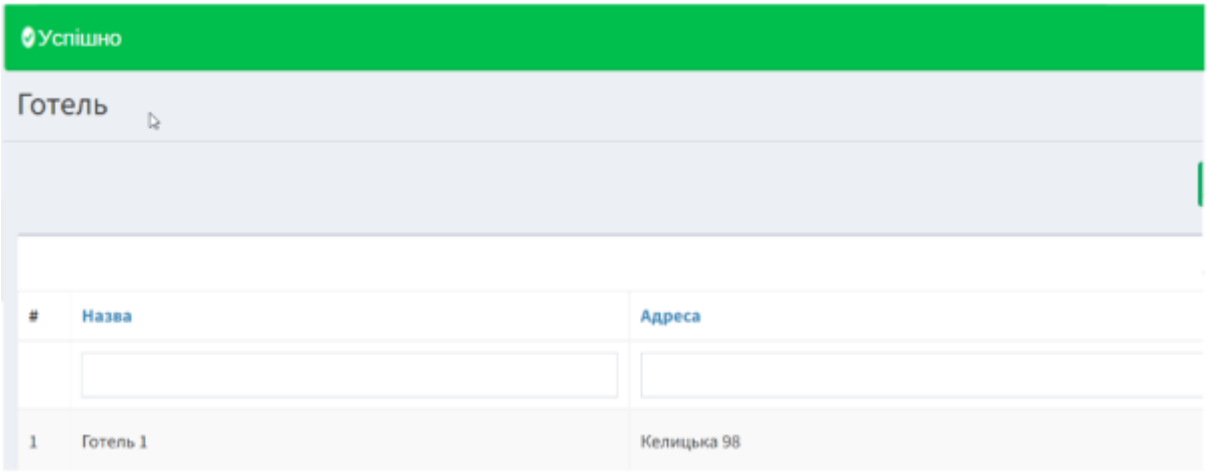



Рисунок 3.6 – Результат виконання тест-кейсу №2




Рисунок 3.7 – Результат виконання тест-кейсу №3

За результатами практичного завершення перевірочного процесу вебдодаток згенерував офіційне сповіщення Успішно, що беззаперечно підтверджує повну відповідність отриманого фактичного результату заздалегідь

задекларованим очікуванням, тому тест-кейс під третім номером вважається успішно виконаним. Наступним етапом комплексного контролю якості став тест-кейс, розроблений для детальної верифікації процедури реєстрації та внесення відомостей про нового клієнта готелю. Цей алгоритм дій передбачає першочерговий перехід користувача на спеціалізовану сторінку додавання нового постояльця, де у відповідні текстові поля та електронні форми вноситься весь масив необхідних персональних і технічних даних. Після успішного заповнення інформаційних блоків здійснюється безпосередня активація інтерактивної кнопки Зберегти, а фінальним критерієм коректного відпрацювання серверної логіки є автоматичне отримання від системи позитивної відповіді Успішно.

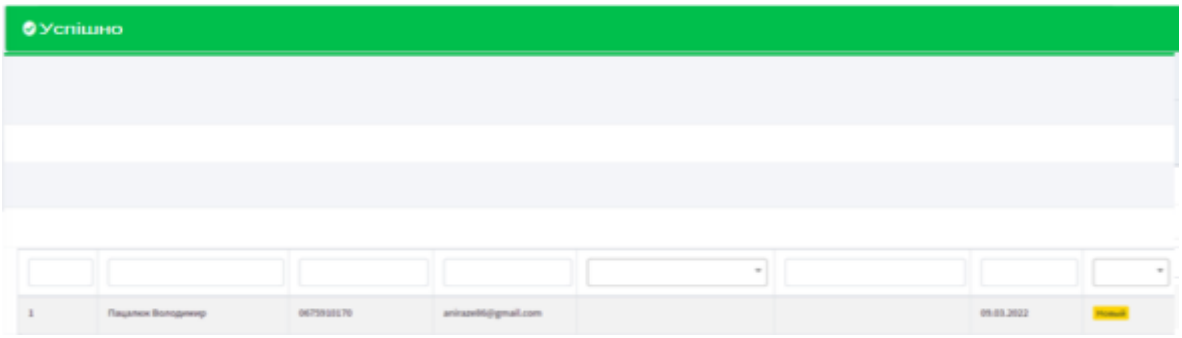



Рисунок 3.8 – Результат виконання тест-кейсу №4

Практична реалізація цього перевірочного сценарію продемонструвала абсолютну тотожність отриманого фактичного результату заздалегідь визначеним очікуванням, що дозволяє офіційно зафіксувати успішне проходження тест-кейс під четвертим номером. Наступним кроком у межах контролю якості програмного продукту виступає тест-кейс, призначений для всебічної верифікації фінансового модуля, а саме процедури поповнення балансу всередині системи. Даний алгоритм дій вимагає від користувача першочергового переходу на цільову сторінку реєстрації приходу грошових коштів, де у відповідне інтерактивне поле форми вноситься необхідна сума цифрових одиниць. Після коректного заповнення фінансового показника здійснюється безпосереднє натискання на керуючу кнопку Зберегти, а фінальним критерієм успішного опрацювання транзакції на стороні сервера є виведення на екран

позитивного системного повідомлення у вигляді відповіді Успішно.




Рисунок 3.9 – Результат виконання тест-кейсу №5

Безпосереднім технічним наслідком операції з поповнення балансу є автоматичне збільшення загального фінансового показника в касі додатка, де графічні матеріали під номером чотири шість наочно демонструють вихідні параметри системи та її оновлений стан після завершення повноцінного циклу перевірки, що слугує чітким прикладом успішного відпрацювання тест-кейс під п'ятим номером. Наступним етапом контролю якості фінансової логіки виступає тест-кейс, розроблений для верифікації процедури внесення грошових коштів безпосередньо на персональний рахунок конкретного клієнта. Цей алгоритм дій передбачає обов'язковий перехід тестувальника на спеціалізовану сторінку реєстрації приходу для фінансового балансу постояльця, де в активне поле форми вводиться необхідна сума цифрових одиниць. Після успішного заповнення грошового показника виконується безпосереднє натискання на інтерактивну кнопку Зберегти, а головним критерієм безпомилкового завершення транзакції та коректного оновлення записів є отримання від системи офіційної позитивної відповіді Успішно.




Рисунок 3.10 – Результат виконання тест-кейсу №6

Практична реалізація цього перевірочного сценарію продемонструвала абсолютну тотожність отриманого фактичного результату заздалегідь визначеним очікуванням, що дозволяє офіційно зафіксувати успішне проходження тест-кейс під шостим номером. Загалом у процесі комплексного контролю якості розробленого програмного продукту було зафіксовано стовідсоткову відповідність усіх фактичних показників запланованим очікуваним результатам. Під час активної експлуатації та перевірки різних модулів додатка не було виявлено жодних системних помилок, збоїв чи критичних багів, а сама програма функціонувала стабільно та в точній відповідності до логіки, задекларованої в описі її технічних можливостей. Підсумкові результати проведених випробувань беззаперечно доводять повну працездатність створеного цифрового рішення та його абсолютну відповідність усім вимогам початкового технічного завдання.

## ВИСНОВКИ

У межах виконання кваліфікаційної роботи бакалавра було успішно реалізовано сучасну автоматизовану систему управління готелем, головне призначення якої полягає в суттєвому підвищенні загальної ефективності роботи обслуговуючого персоналу за допомогою переведення ключових повсякденних завдань готельної адміністрації в цифровий формат. Проектуванню передував глибокий аналітичний огляд наявних на ринку систем-аналогів, у ході якого вдалося чітко висвітлити їхні головні функціональні недоліки. Наступне порівняння цих рішень із власним цифровим продуктом повністю підтвердило доцільність та економічну обґрунтованість створення принципово нового вебдодатка, що дозволило детально сформулювати та виконати фінальну постановку завдань розробки.

Безпосередньому написанню програмного коду передував етап візуального та логічного моделювання, під час якого спроектували ергономічний графічний інтерфейс та побудували блок-схеми алгоритмів для обчислення очікуваного обсягу фінансових витрат за окремими категоріями, їхнього подальшого сортування, надійного збереження в базі даних, а також автоматичного генерування звітів у текстовому та графічному форматах. На основі цих моделей було створено всі ключові модулі системи. У процесі підготовки технічного стеку було проведено варіантний аналіз та надано обґрунтування вибору інструментальних засобів, у результаті чого базовою мовою програмування було затверджено PHP, основним середовищем розробки обрано PhpStorm, а для керування інформаційними масивами залучено СУБД MySql. Під час виробництва цифрового продукту було задокументовано та описано архітектурну реалізацію кожного функціонального вузла програми.

Фінальним етапом проекту став аналіз сучасних методологій контролю якості, де серед усього різноманіття підходів розробники надали перевагу методу чорного ящика. Подальша верифікація системи за допомогою спроектованих тест-кейсів наочно продемонструвала повну відсутність дефектів, підтвердивши абсолютну працездатність та високу стабільність створеного додатка.

**СПИСОК ЛІТЕРАТУРИ**

1. Автоматизовані системи управління готелями : вебсайт. URL: https://tourlib.net/statti_ukr/gudzovata.htm (дата звернення: 11.06.2026).

2. Глобальна система розподілу / Вікіпедія : вільна енцикл. URL: https://uk.wikipedia.org/wiki/Глобальна_система_розподілу_(GDS (дата звернення: 11.06.2026).

3. Інтегровані системи управління людськими ресурсами : вебсайт. URL: https://kbp.aero/docs/Nastanova.pdf (дата звернення: 11.06.2026).

4. Інтегроване середовище розробки / Вікіпедія : вільна енцикл. URL: https://uk.wikipedia.org/wiki/Інтегроване_середовище_розробки (дата звернення: 11.06.2026).

5. Інформаційна система / Britannica : website. URL: https://www.britannica.com/topic/information-system (дата звернення: 11.06.2026).

6. Інформаційні технології в сучасному світі : вебсайт. URL: http://sophus.at.ua/publ/2013_12_19_20_kampodilsk/sekcija_7_2013_12_19_20/informacijni_tekhnologiji_v_suchasnomu_sviti/49-1-0-863 (дата звернення: 11.06.2026).

7. Лілія Н. Готельно-ресторанний бізнес. - 2019. - 344 с.

8. Пацалюк В. С. Інформаційні технології у готельному бізнесі. *Молодь в науці: дослідження, проблеми, перспективи (МН-2022)* : матеріали конференції. 2022. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/view/16274 (дата звернення: 18.05.2022).

9. Туризм : офіційний вебсайт. URL: https://www.tourism.gov.ua/ (дата звернення: 11.06.2026).

10. Що таке комп’ютерні системи бронювання / IGI Global : website. URL: https://www.igi-global.com/dictionary/computer-reservation-systems-crs/41576 (дата звернення: 11.06.2026).

11. Amos D., Bader D. Python Basics: A Practical Introduction to Python 3. Real Python, Inc., 2021. 635 p. ISBN 978-1775093329.

12. Amos D. Foundations of Software Testing ISTQB Certification. 4th ed. 2019. 530 p.

13. Duckett J. JavaScript and jQuery: Interactive Front-End Web Development. - 2014. - 640 p.

14. Eclipse IDE : website. URL: https://www.eclipse.org/ (дата звернення: 11.06.2026).

15. Greenberg P. CRM at the Speed of Light: Social CRM 2.0 Strategies, Tools, and Techniques for Engaging Your Customers. - 4th ed. - 2009. - 688 p.

16. Kreibich J. A. Using SQLite: Small. Fast. Reliable. Choose Any Three. - 2010. - 530 p.

17. Mohan G. Full Stack Testing. - 2022. - 658 p.

18. Murach J. Murach's MySQL. - 2019. - 628 p.

19. Obe R. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. - 2017. - 312 p.

20. PhpStorm IDE : website. URL: https://www.jetbrains.com/phpstorm/ (дата звернення: 11.06.2026).

21. Kevin T., Peter M. Programming PHP: Creating Dynamic Web Pages. - 2020. - 544 p.

22. Visual Studio Code IDE : website. URL: https://code.visualstudio.com/ (дата звернення: 11.06.2026).