Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка автоматизованої системи управління готелем»

Виконав: студент IV курсу
групи ЗПІ-18б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Пацалюк В. С.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Хошаба О.М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. КН Колесницький О.К.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____ О.Н. Романюк
«____» _____ 2022 р.

ВНТУ– 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
25 березня 2022 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Пацалюку Володимиру Сергійовичу

1. Тема роботи – «Розробка автоматизованої системи управління готелем»
Керівник роботи: Хошаба Олександр Мирославович , к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 24 березня 2022 р. № 66
2. Строк подання студентом роботи 13 червня 2022 р.
3. Вихідні дані до роботи: Середовище розробки – PhpStorm, Мови розробки – html, css, php, js, база даних MySQL.
4. 4. Зміст розрахунково–пояснювальної записки: вступ; обґрунтування вибору методу розробки та постановка задачі дослідження; проектування структури, методу та алгоритмів роботи програмного продукту; розробка програмних компонент для додатку; тестування програми; висновки; список використаних джерел; додатки.
5. Перелік графічного матеріалу: титульний слайд, актуальність теми, мета, об'єкт та предмет дослідження, задачі дослідження, новизна отриманих результатів, практична цінність одержаних результатів, порівняльний аналіз аналогів, блок-схема авторизації, блок-схема додавання нового постояльця, блок-схема друкування графічних і текстових звітів та тестування додатку

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Хошаба О. М., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання 25 березня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз та постановка задачі	26. 03.2022 – 02.04.2022	Вик.
2	Проектування структури, методу та алгоритмів роботи програмного продукту	03.04.2022 - 16.04.2022	Вик.
3	Вибір середовища та мови розробки	17.04.2022 - 27.04.2022	Вик.
4	Розробка модулів програми	28.04.2022 - 17.05.2022	Вик.
5	Тестування програми	18.05.2022 - 25.05.2022	Вик.
6	Написання інструкції користувача	26.05.2022 - 01.06.2022	Вик.
7	Оформлення матеріалів до захисту БДР	02.06.2022– 10.06.2022	Вик.

Студент _____ **Пацалюк В. С.**

(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ **Хошаба О. М.**

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 51 сторінок формату А4 містить 23 рисунків та 4 таблиць, список використаних джерел містить 22 найменувань.

У бакалаврській дипломній роботі проведено детальний аналіз та оцінку веб-додатків. Встановлено об’єкт, предмет, завдання та методи дослідження. Сформульовано мету дослідження – підвищення ефективності та спрощення управління готелем, шляхом використання спеціалізованого софту. Для реалізації мети було розроблено веб-додаток автоматизовану систему управління готелем.

Під час проектування веб-додатоку для управління готелем були визначені вимоги та критерії щодо розробленого продукту.

Створений web-додаток для управління готелем розроблено з використанням мови програмування PHP та фреймворку Laravel в бекенді, а шаблон сайту розроблений задопомогою HTML, CSS та JavaScript, середовища розробки PHPStorm 2021, для тестування коду використано бібліотеку PHPUnit. В результаті виконання бакалаврської дипломної роботи розроблено веб-додаток та протестовано його.

ABSTRACT

The bachelor's thesis consists of 51 A4 pages containing 25 figures and 4 tables, the list of used sources contains 22 titles.

A detailed analysis and evaluation of web applications was conducted in the bachelor's thesis. The object, subject, tasks and research methods are established. The research method is formulated - increase of efficiency and increase of management efficiency by means of special software. To achieve this goal, a web application automated by the hotel management system was developed.

The requirements and criteria for the developed product were defined during the design of the web application for hotel management.

The created web application for hotel management is developed using PHP programming languages and Laravel backend framework, site template developed using HTML, CSS and JavaScript, PHPStorm 2021 environment development, to test the code of the used PHPUnit library. As a result of the bachelor's thesis, a web application was developed and tested.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сьогодні туризм – це явище яке, повязане з різними науками такими як: історія, географія, архітектура, спорт та культура країни в яку їде турист, та сервіс який надають для туриста [1].

Хоча і в останні роки туризм переживає не найкращі часи, в основному через вссесвітню пандемію яка розпочалась в 2020 році, але з винайденням вакцини все таки потрохи люди починають подорожувати.

Туризм також залежить від готельно ресторaного бізнесу, так як більшість туристів подоружуючи зупиняються в готелях для відпочинку, або ж спеціальних туристичних програм які надають ті чи інші готелі.

Готельний та ресторанний бізнес [2] – складова туристичної сфери, яка спрямована на задоволення туристичних потреб населення у вигляді житла, харчування, транспортного й екскурсійного обслуговування та іншого сервісу.

З розвитком туризму готельний бізнес є важливою сферою обслуговування. Велика конкуренція у цій сфері надає поштовх її лідерам впроваджувати сучасні технології, такі як системи управління готелем. Система управління готелем це складна система яка складається з декількох підсистем, таких як система управління номерами, персоналом, постояльцями, адміністративна система та система обліку.

Наразі існує доволі небагато реалізацій систем управління готелем. Кожна з них має свої недоліки, найчастіше яким є відсутність інтеграції з іншими підсистемами, наприклад такими які дозволяють слідкувати за ефективністю персоналу. Через такі недоліки у адміністратора виникають проблеми з логістикою бізнес, що напряму впливає на якість обслуговування, що в свою чергу з часом призводить до відмови працівників подовжувати роботу в готелі, а ще в гіршому варіанті відмову туристів від використання використання готелю для відпочинку [3].

Тому розробка власної системи управління готелем яка прокриє більшість недоліків інших аналогічних систем є досить актуальною задачею.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою дослідження є підвищення ефективності адміністрування готельного бізнесу, що дозволить покращити та спростити роботу персоналу, та дозволить з економити час клієнтів.

Відповідно до поставленої мети в бакалаврській дипломній роботі потрібно вирішити такі **завдання**:

– проаналізувати сучасний стан розвитку технологій у готельному бізнесі;

– розробити алгоритм авторизації;

– розробити алгоритм додавання готелів та даних про них у базу даних;

– розробити алгоритм генерування текстових та графічних звітів потрібних користувачеві;

– розробити алгоритми для реалізації власної системи управління;

– розробити програмні компоненти системи управління готелем;

– провести тестування програмного коду;

– провести тестування web-додатку.

Об’єкт дослідження – процеси реалізації автоматизованої системи управління готелем.

Предмет дослідження – методи та засоби розробки автоматизованої системи управління готелем.

Методи дослідження. У бакалаврському дипломному проекті: використовувалися теорії чисел і математичної статистики для генерування фінансових звітів для статистики та друкування; комп’ютерне моделювання та засоби проектування веб-додатків для розробки програмного веб-додатку автоматизованої системи управління готелем; методи тестування для перевірки роботи створеного програмного додатку.

Новизна отриманих результатів.

1. Удосконалено алгоритм додавання нових постояльців до готелю, який на відміну від інших, дозволяє створити рахунок користувача, для зберігання коштів.

2. Удосконалено алгоритм генерування графічних та текстових звітів, що дозволяє швидко та ефективно формувати їх.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для автоматизованої системи управління готелем.

Особистий внесок здобувача. Усі наукові результати отримані автором особисто. У наукових працях, опублікованих у співавторстві, автору належать такі результати: інформаційні технології у готельному бізнесі [4].

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на всеукраїнських конференціях: Всеукраїнська науково-практична Інтернет-конференція студентів, аспірантів та молодих науковців «МОЛОДЬ В НАУЦІ: ДОСЛІДЖЕННЯ, ПРОБЛЕМИ, ПЕРСПЕКТИВИ (МН-2022)» (16-17 червня 2022 р., м. Вінниця).

Публікації. Основні результати досліджень опубліковано в 1 науковій праці у збірниках матеріалів конференцій.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз стану проблеми

Інформаційні технології зробили значний внесок у наш світ, вони допомагають автоматизувати роботу в багатьох сферах нашого життя та підвищити ефективність самої людини [5]. При збільшені кількості клієнтів, виникає проблема навантаження на працівників, що призводить до затримок в роботі, що в свою чергу впливає на якість обслуговування, тому для виправлення цих проблем існують системи інформаційних технологій які допомагають в роботі.

Системи інформаційних технологій, що використовуються в туристичній сфері мають комплексний характер, складаються, як правило, з електронних інформаційних систем авіаліній, системи проведення телеконференцій, відеосистем, комп’ютерів, інформаційних систем управління, глобальних комп'ютерні системи резервування, інтегрованих комунікаційних мереж, комп’ютерної системи бронювання, електронної пересилки грошей, телефонних мереж, системи мультимедіа, смарт-карток [6].

Впродовж останіх років туризм стає популярним, через те що теперишні технології дозволяють подорожувати, витрачаючи мінімум часу на дорогу до місця презначення, так як літаком за декілька годин можна дістатись будь-якого місця планети. Саме тому готельний бізнес стає теж популярним, і потребує сучасних систем управління готелем для ефективної роботи.

Головною перевагою інтеграцій новітніх технологій у готельній бізнес, є покращення якості обслуговування. Різні системи управління допомагають працівникам витрачати менше часу на виконання обовязків та ефективно працювати.

Комп’ютерні системи бронювання – це автоматизовані системи, що використовуються для зберігання та отримання інформації та здійснення

транзакцій, пов’язаних з готелями, авіаперельотами чи іншими видами діяльності [7].

Сучасні GDS зазвичай дозволяють користувачам бронювати готельні номери, орендувати автомобілі, авіаквитки, а також інші заходи та тури. Вони також надають доступ до залізничних та автобусних резервацій, але вони не завжди інтегровані в основну систему. Вони також використовуються для передачі даних користувачам готельної індустрії, здійснення бронювання та забезпечення того, щоб готель не був заповнений.

Глобальна система розповсюдження (GDS) об’єднує резервні запаси в одну комп’ютерну мережу. Найпоширенішим використанням GDS є туристична індустрія, в основному для авіакомпаній, готельних компаній та компаній з прокату автомобілів. Таким чином, постачальники послуг, такі як туристичні агенції та туристичні онлайн-агентства, можуть продавати квитки на один і той самий рейс, готель чи оренду автомобіля [8].

Інтегровані системи управління людськими ресурсами використовуються для автоматизації роботи персоналу будь-якої компанії. Насамперед, ці системи необхідні керівництву для отримання оперативної інформації з усіх питань, що стосуються структури компанії, персоналу, вакансій та інформації про співробітників. Швидко прийняти правильне рішення зможе лише керівник, який зможе швидко оцінити ситуацію на основі аналізу поточної інформації про ситуацію в компанії. Таким чином, важливим фактором при використанні систем управління персоналом є здатність системи нарахування заробітної плати інтегруватися з бухгалтерськими та бізнес-системами [9].

Інформаційна система управління, по суті, складається з п'яти компонентів, апаратного забезпечення, програмного забезпечення, бази даних, мережі та людей. Ці п’ять компонентів об’єднуються для виконання введення, процесу, виходу, зворотного зв’язку та керування.

Апаратне забезпечення складається з пристрою введення/виводу, процесора, операційної системи та медіа-пристроїв. Програмне забезпечення складається з різних програм і процедур.

База даних складається з даних, організованих у необхідній структурі.

Мережа складається з концентраторів, засобів зв'язку та мережевих пристроїв. Люди складаються з операторів пристроїв, адміністраторів мережі та спеціаліста з системи.

Обробка інформації складається з введення; обробка даних, зберігання даних, виведення та контроль. На етапі введення дані інструкції надходять до систем, які на етапі процесу обробляються програмними програмами та іншими запитами. На етапі виведення дані подаються у структурованому форматі та звітах.

Системи управління так звані CRM системи, дозволють користувачеві спростити роботу, надаючи можливість в одному додатку мати велику кількість функцій які дозволять спростити управління [10].

Не дивлячись на високий темп розвитку інформаційних технологій в сучасному світі систем управління готелем є не багато, а недосконаліть цих систем, вказує на те, що питання програмної реалізації системи управління готелем залишається відкритим.

Хоча і на ринку є системи управління готелем гідні уваги, власники які володіють мережею готелів, найчастіше замовляють розробку системи у IT фірмі, через те що системи які є у продажі в інтернеті перегружані функціоналом через те що розробники хотіли розширити її на інші сфери, або ж у системі недостотньо потрібного функціоналу.

Таким чином основною проблемою у готельній сфері є не достатньо розвинені інформаційні системи які допомагають у управлінні.

Отже, системи управління готелем це технології які направлені щоб підвищити ефективність роботи персоналу та автоматизувати деякі функції за які відповідав персонал.

1.2 Порівняльний аналіз аналогів

Не зважаючи на усі переваги розробки свої системи управління готелем, для початку проведемо аналіз вже існуючих реалізацій таких систем. Серед

існуючих систем найбільш схожими до нашої системи по функціонал та призначенню є:

– Clock PMS;

– Hotelinstinct;

– Lite PMS;

– HotelCloud.

Clock PMS – система управління готелем.

До переваг даної системи можна віднести такі функції друк звітів, історія взаємодії з клієнтом, база клієнтів. Основні недоліки – відсутність графічних звітів та не має можливості слідкувати за ефективністю персоналу. Встановлюється на сервер або хмару тому доступна через любий браузер. На рисунку 1.1 наведено користувацький інтерфейс додатку.

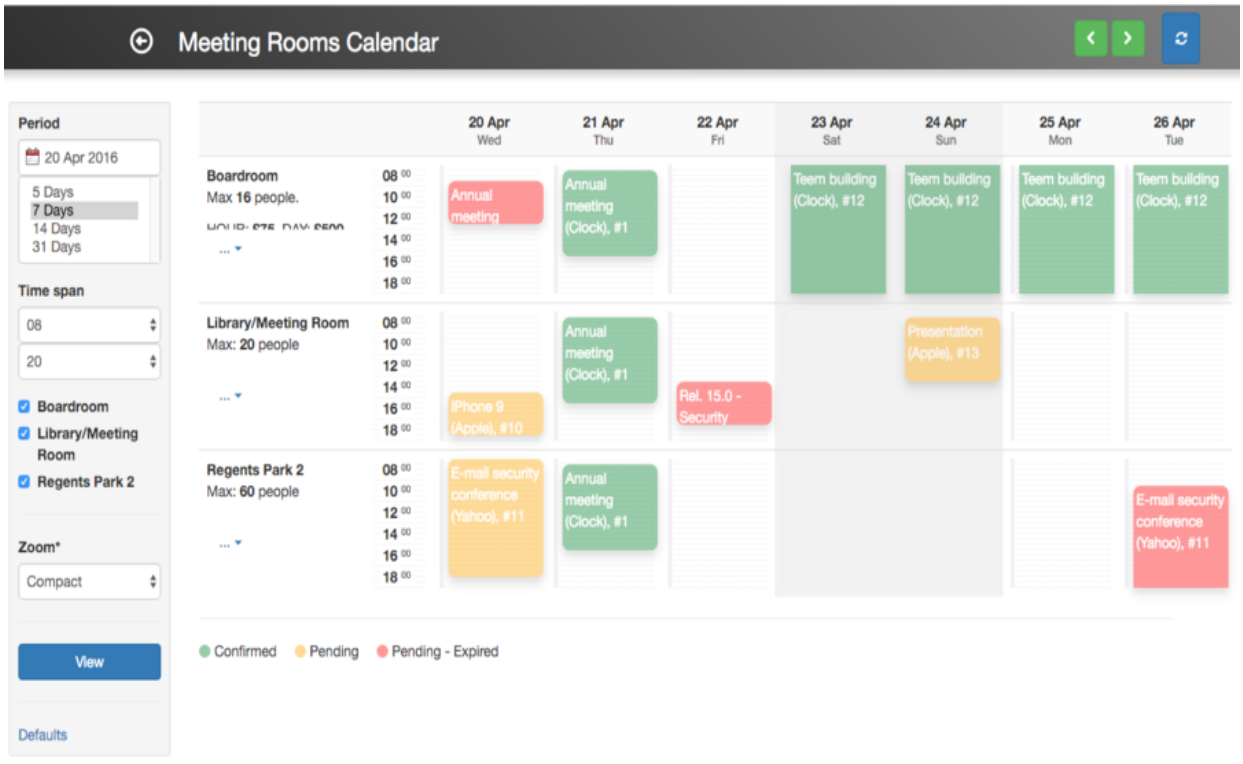


Рисунок 1.1 – Користувацький інтерфейс програми Clock PMS

Hotelinstinct – система управління готелем.

Система допомагає в управлінні готелем та дозволяє виводити інформацію про постояльців та записувати її в базу даних, виводити аналітику.

Основні недоліки – відсутність графічних звітів та не має можливості слідкувати за ефективністю персоналу. Встановлюється на сервер або хмару тому доступна через любий браузер. Користувацький інтерфейс додатку наведено на рисунку 1.2

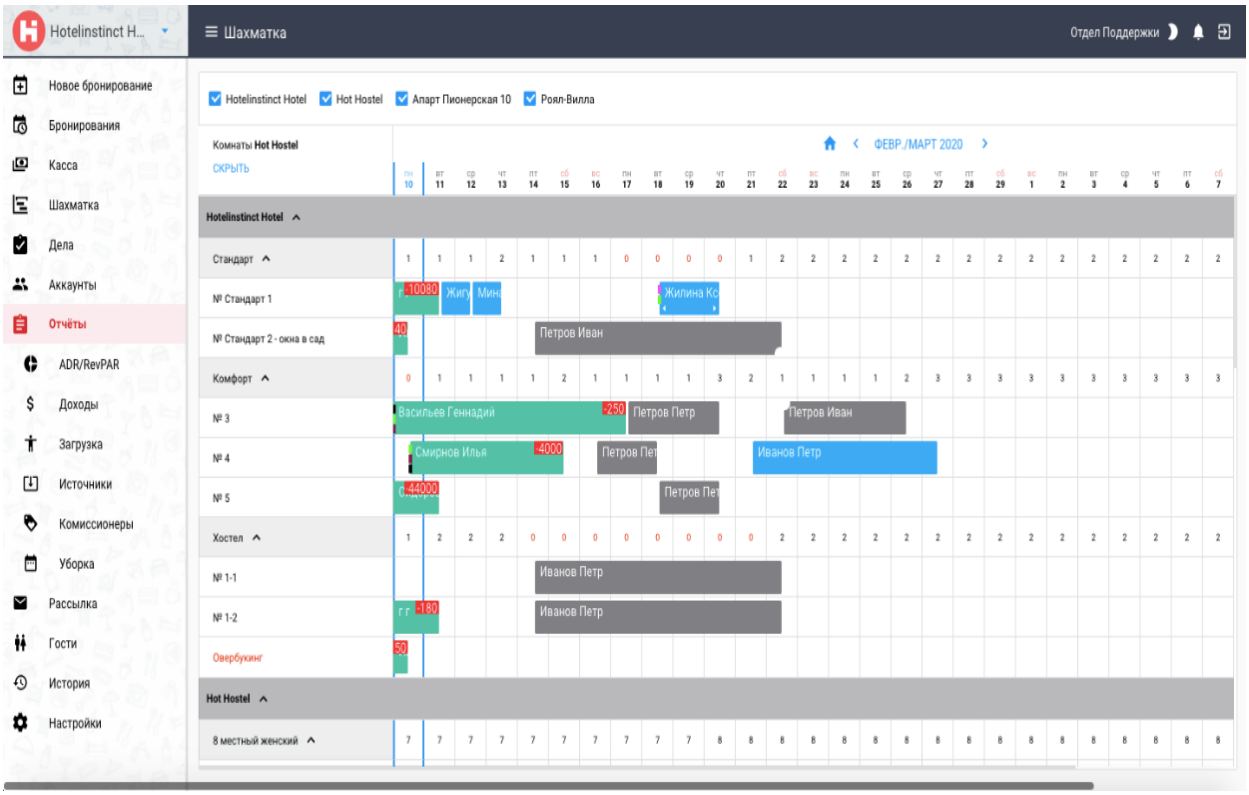


Рисунок 1.3 – Користувацький інтерфейс програми Hotelinstinct

Lite PMS – повністю безкоштовний система управління готелем. Система направлена на допомого в управлінні готелю та автоматизації різних функцій.

Головними перевагами системи є те що вона дозволяє виводити інформацію про постояльців та записувати її в базу даних, виводити аналітику. До основних недоліків можна віднести відсутність графічних звітів та не має можливості слідкувати за ефективністю персоналу. Встановлюється на сервер або хмару тому доступна через любий браузер.

На рисунку 1.4 наведено користувацький інтерфейс додатку.

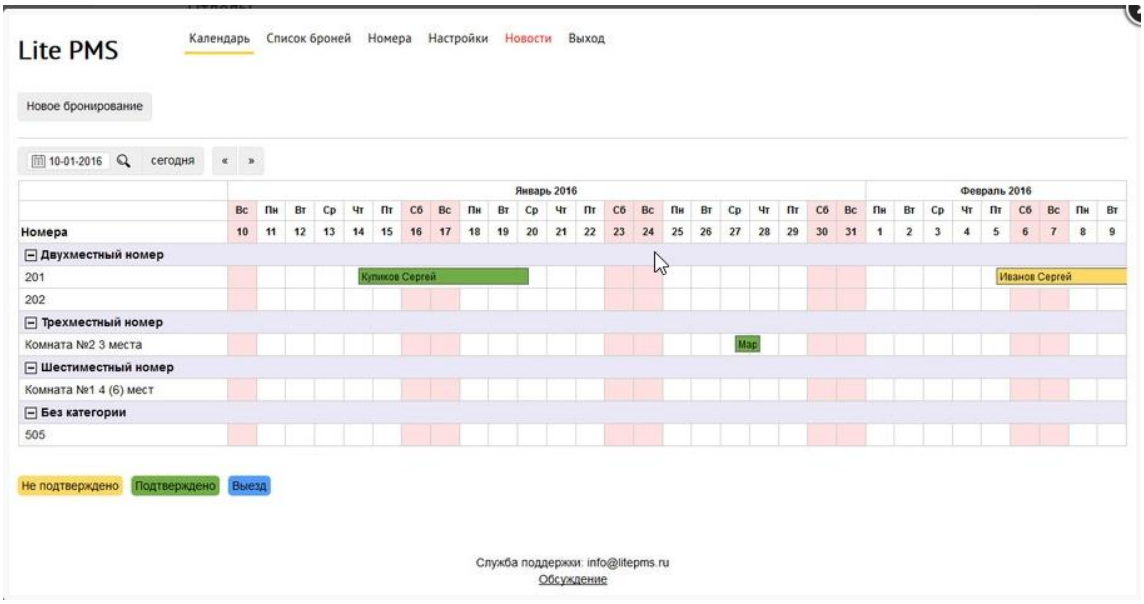


Рисунок 1.4 – Користувацький інтерфейс програми Lite PMS

HotelCloud – система управління готелем. Система допомагає в управлінні готелем та дозволяє виводити інформацію про постояльців та записувати її в базу даних, виводити аналітику. Основні недоліки – відсутність можливості слідкувати за ефективністю персоналу. Встановлюється на сервер або хмару тому доступна через любий браузер.

Користувацький інтерфейс додатку наведено на рисунку 1.5.

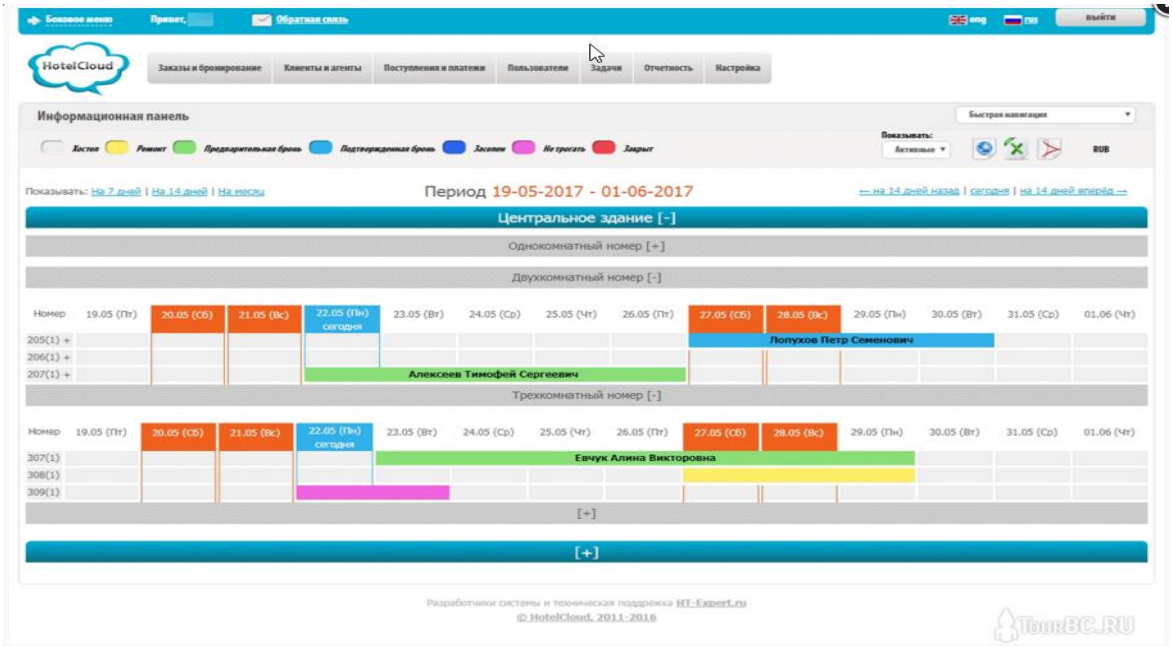


Рисунок 1.5 – Користувацький інтерфейс програми HotelCloud

Після аналізу усіх аналогів визначено їхні переваги та недоліки та проведено порівняння із розроблюваним Telegram-ботом під назвою «Expense Control». Результат порівняння зведено в таблицю 1.1. Для позначення наявності критерію використовується “+”, а для відсутності використовується “-”, а для того часткової наявності використовується “+-”..

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	-	-	+	-	+
Бронюванн через інтернет	+	+	-	-	-
Кросплатформність	+	+	+	+	+
Підсумковий результат	2	2	2	1	4

В результаті порівняння існуючих аналогів було зроблено висновок, що розробка системи управління готелем є доцільною. В результаті розробки отримаємо продукт, який покриває недоліки існуючих аналогів.

1.3 Аналіз методів розв’язання поставленої задачі

Існує декілька варіантів розробки системи це програмний додаток, який буде встановлюватися на персональний комп’ютер, або ж web-додаток який можна встановити на сервер або хмару і мати доступ з любого пристрою через браузер.

Більш ефективним способом розв’язку поставленої задачі є саме розробка web-додатку. Тому було вирішено розробляти саме його.

Веб-додаток – це програмне забезпечення або програма, яку можна відкрити за допомогою будь-якого браузера. Зовнішній інтерфейс веб програми розробляється за допомогою таких мов програмування: HTML, CSS, Javascript, які підтримуються на будь-якому браузері (Opera, Chrome, Mozilla). У той час як для написання серверної частини (Back-end) може використовуватися будь-яка інша мова програмування або фреймворк, Python, PhP, Ruby, Java.

Основні переваги веб-застосунків:

- веб додатки можуть застосовуватися на будь-якій операційній системі (Linux, Mac, Windows), оскільки всі вони підтримують сучасні браузери;
- у зв’язку з тим, що у веб-додатку використовується той самий код порівняно з desktop додатками їх набагато легше підтримувати;
- додаток простіше програмувати оскільки він не включає багато роботи з елементами ПК(ядро,процесор, відеокарта);
- на відміну від мобільних додатків, для веб-додатків не потрібно схвалення жодних платформ, щоб випустити свою програму;
- веб додатки це більш економний варіант для будь-якого підприємства

оскільки веб-програми не вимагають підписки або покупки ліцензій, а можуть використовуватися як SaaS-сервіс, що значно дешевше.

Для зручної розробки додаток можна поділити на декілька модулів, розробка інтерфейсу системи, розробка основного функціоналу (такого як додавання готелів, додавання номерів, додавання постояльців, та обробка цих даних), також розробка функції виводу графічних звітів.

Для веб-додатків, існує спеціальний інтервейс так званий веб-інтерфейс.

Веб-інтерфейс – це сукупність засобів, за допомогою яких користувач взаємодіє з веб-сайтом або будь-якою іншою програмою через браузер. Веб-інтерфейси набули широкого поширення у зв'язку зі зростанням популярності всесвітньої павутини і відповідно - повсюдного розповсюдження веб-браузерів. Поширення веб-інтерфейсу пов'язане зі зростанням популярності Інтернету та повсюдного використання веб-браузерів. Щоб сторінка могла відповідати вимогам веб-інтерфейсу, вона повинна мати однаковий зовнішній вигляд з однаковим функціональним під час роботи в різних браузерах.

Класичний метод створення веб-інтерфейсів – використання коду HTML із CSS та JavaScript'a. Але різна реалізація HTML, CSS, DOM тощо у браузерах може викликати проблеми розробки веб-додатків. Роботі інтерфейсу також можуть заважати можливість користувача настроювати такі параметри браузера, як шрифт, колір, підтримка сценаріїв та ін.

Інтерфейс системи повинен бути зручним та зрозумілим щоб навіть людина без досвіду могла з ним працювати, тобто повинен бути дружелюбним. Дружелюбний інтерфейс відображає, наскільки зручно користуватися веб-ресурсом. Головним показником дружелюбного інтерфейсу є його ергономічність, чим менше дій на сайті виконує людина, тим він зручніший.

Основую веб-додатків являється бекенд частина.

Бекенд - це програмно-апаратна частина сервісу. Це набір засобів, за допомогою яких відбувається реалізація логіки веб-сайту. Це те, що приховано від наших очей, тобто відбувається поза комп'ютером та браузером.

Як тільки користува введе запит на сторінці пошукової системи та натисніть клавішу Введення, frontend закінчиться і почнеться backend. Ваш запит відправиться на сервер Яндекса або Google, тобто за місцем розташування алгоритмів пошуку. Саме там і відбувається вся магія. Але на моніторі з'являються дані, які користувач запитує, - це відбувається повернення в frontend.

Також можна сказати, що backend це процес об'єднання користувача з сервером.

Для розробки бекенд частини буде використовуватися мова програмування PHP. Це одна з найпопулярніших мов програмування для веб-розробки. Також буде використаний фреймворк Laravel. Це фреймворк для розробки веб-додатків, що реалізує шаблон модель–представлення–контролер. Даний фреймворк містить велику кількість готових архітектурних рішень, які легко можна впровадити розроблячи власний веб-додаток.

В якості бази даних проекту використаємо MysQL. Це одна з найпопулярніших СУБД, що розроблена Oracle. Вона поширюється по ліцензії. Вона має широкий функціонал якого більше чим достатньо для розробки веб-додатка.

Основний функціонал повинен відповідати усім вимогам. У системі повинна бути можливість додавати нові готелі, номери, постояльців та інших даних в базу даних. Повинний бути модуль обліку грошових коштів, повинна бути можливість генерування звітів як текстових, так і графічних.

1.4 Постановка задач розробки web-додатку система управління готелем

Після аналізу питання розробки web-додатку система управління готелем, було визначено наступні завдання, які необхідно виконати для розробки програмного продукту:

– розробити алгоритм авторизації;

– розробити алгоритм додавання готелів, номерів, постояльців та інших даних в базу даних;

– розробити алгоритм роботи каси;

– розробити алгоритм генерування текстових та графічних звітів у системі;

– розробити програмні компоненти системи;

– провести тестування програмного продукту.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У розділі було розглянуто стан питання існуючих систем управління готелем на сьогоднішній день. Було проведено аналіз існуючих аналогів та проведено їх порівняння між собою та власною системою. У результаті доведено доцільність розробки власної системи. Також було проведено аналіз існуючих підходів до вирішення поставленої задачі. Було встановлено основні завдання, які необхідно виконати для розробки власної системи.

2 ПРОЕКТУВАННЯ СТРУКТУРИ, МЕТОДУ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Вибір архітектури web-додатку

Для початку визначитися, що саме маємо на увазі під поняттям «веб-додаток».

Клієнт, сервер та база даних.

Веб-додаток — це додаток клієнт-сервер, в якому браузер являється клієнтом, а веб-сервер (загалом кажучи) є сервером. Основна частина додатку зазвичай знаходиться на стороні веб-сервера, який відповідає за обробку отриманих запитів відповідно до бізнес-логіки продукту і формує відповідь, яку надсилає користувачеві. Далі до роботи приступає браузер, який вже перетворює отриману від сервера відповідь у зрозумілий звичайному користувачеві графічний інтерфейс.

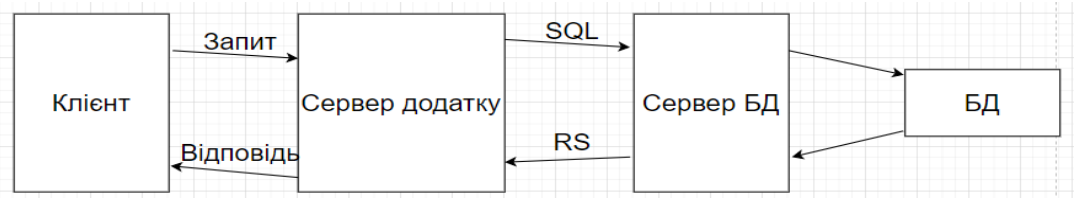


Рисунок 2.1 – Клієнт-серверна архітектура веб-додатку

1. Клієнт

Зазвичай клієнтом є браузер, але є винятки (коли один веб-сервер (ВС1) запитує інший (ВС2), роль клієнта виконує веб-сервер ВС1). У класичній ситуації (коли клієнт виступає в ролі браузера) браузер повинен обробити відповідь, отриману від веб-сервера, щоб користувач міг побачити графічний інтерфейс програми у вікні браузера, який містить інформацію, створену за допомогою HTML, CSS та Впроваджено технологію JS). Саме ці технології «унікально інструктують» браузер, як «малювати» те, що він отримує натомість.

2. Сервер

Веб-сервер — це сервер, який приймає HTTP-запити від клієнтів і видає на них відповіді HTTP. Щоб уникнути можливої плутанини, веб-сервер означає як програмне забезпечення, яке виконує функції веб-сервера, так і комп'ютер, на якому безпосередньо це програмне забезпечення. Найпоширенішими типами програмного забезпечення для веб-серверів є Apache, IIS та NGINX. Додаток для тестування працює на веб-сервері і може бути реалізований за допомогою широкого спектру мов програмування: PHP, Python, Ruby, Java, Perl тощо.

3. База даних

У класичній теорії говориться про дві «сторони» веб-додатка, але якщо уважніше поглянути на весь процес роботи програми, то можна побачити, що інша «сторона» невидима, але досить активно задіяна в Інтернеті. алгоритм. додаток - основні дані. Насправді він не є частиною веб-сервера, але більшість додатків просто не може виконувати всі покладені на них функції без нього, оскільки база даних зберігає всю динамічну інформацію програми (облікові дані, дані користувача тощо).

База даних - це досить широкий термін, який використовується не тільки в області інформаційних технологій. У контексті моєї роботи це інформаційна модель, яка дозволяє впорядковано зберігати дані про об'єкт або групу об'єктів, які мають набір властивостей, які можна класифікувати. Бази даних функціонують під керуванням так званих систем управління базами даних (далі – СУБД). Найпопулярнішими СУБД є MySQL, MS SQL Server, PostgreSQL, Oracle (усі клієнт-сервер).

Існують також вбудовані СУБД і файлові сервери. Для загальної розробки згадаю лише одну популярну вбудовану СУБД – SQLite, що використовується в деяких браузерах, API Android, Skype та інших відомих програмах. Взаємодія з перерахованими СУБД здійснюється на основі спеціальної структурованої мови запитів - SQL.

Структура сайту - це логічне побудова всіх сторінок сайту, категорій та під-категорій. Це логічна схема, відповідно до якої всі сторінки та розділи веб-

додатку розташовані відносно один одного та принцип, за яким вони взаємопов'язані між собою.

Існує 3 основних архітектурних шаблони для розробки додатків, це Model-View-Controller (далі MVC), Model-View-Presenter (далі MVP), Model-View- ViewModel (далі MVVM).

MVC - шаблон (паттерн) програмування, що розділяє архітектуру додатка на три модулі: модель (Model), представлення (View), контролер (Controller). Він дозволяє змінювати кожен компонент незалежно один від одного для простої розробки та підтримки веб-програм.

- Модель (Model). Це основна логіка програми. Відповідає за дані, методи роботи з ними та структуру програми. Модель реагує на команди з контролера та видає інформацію та/або змінює свій стан. Вона передає дані у виставу.
- Подання (View). Завданням компонента є візуалізація інформації, яку він отримує від моделі. View відображає дані на рівні інтерфейсу користувача. Наприклад, у вигляді таблиці чи списку. Подання визначає зовнішній вигляд програми та способи взаємодії з ним.
- Контролер (Controller). Він забезпечує взаємодію із системою: обробляє дії користувача, перевіряє отриману інформацію та передає її моделі. Контролер визначає, як програма буде реагувати на дії користувача.

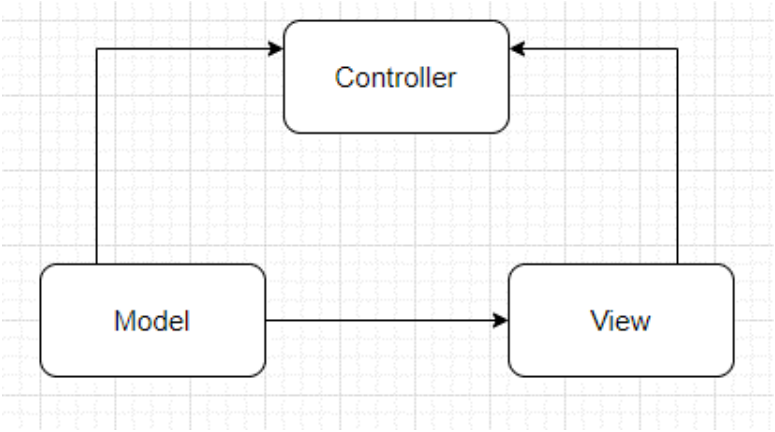


Рисунок 2.2 – структура MVC

Шаблон MVP долає проблеми MVC і забезпечує простий спосіб структурувати коди проекту. Причина широкого визнання MVP полягає в тому, що він забезпечує модульність, можливість тестування та більш чисту та зручну базу коду. Він складається з наступних трьох компонентів:

- Модель (model)**:** шар для зберігання даних. Він відповідає за обробку логіки домену (реальних бізнес-правил) і зв'язок з базою даних і мережевими рівнями.
- Перегляд (view)**:** шар інтерфейсу користувача (Інтерфейс користувача). Він забезпечує візуалізацію даних і відстежує дії користувача, щоб сповістити доповідача.
- Доповідач (presenter)**:** отримує дані з моделі та застосовує логіку інтерфейсу користувача, щоб вирішити, що відображати. Він керує станом представлення та виконує дії відповідно до сповіщень, які користувач вводить із представлення даних.

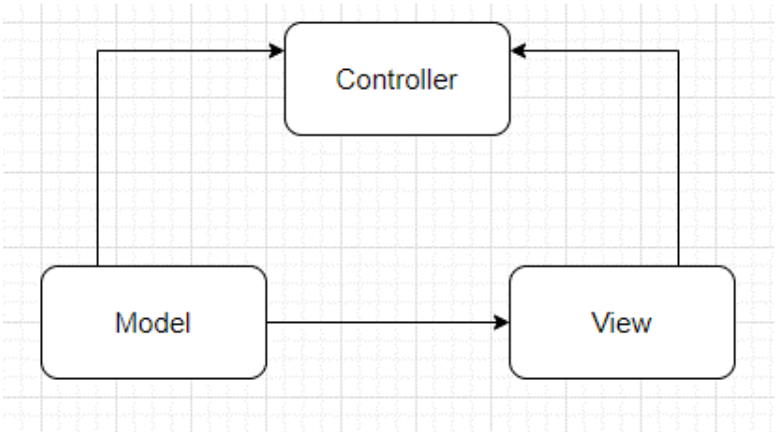


Рисунок 2.3 – структура MVP

MVVM є одним із архітектурних шаблонів, який покращує розділення проблем, він дозволяє відокремити логіку інтерфейсу користувача від бізнес-логіки (або серверної) логіки. Його мета (з іншими шаблонами MVC) полягає в досягненні наступного принципу «Зберігати код інтерфейсу користувача простим і вільним від логіки програми, щоб полегшити керування ним» .

MVVM в основному має такі шари:

– Модель: представляє дані та бізнес-логіку програми. Однією з рекомендованих стратегій реалізації цього шару є надання його даних за допомогою спостережуваних, щоб повністю відокремити їх від ViewModel або будь-якого іншого спостерігача/споживача (це буде проілюстровано в нашому прикладі програми MVVM нижче).
– ViewModel: взаємодіє з моделлю, а також готує спостережувані елементи, які можна спостерігати за допомогою View. ViewModel може додатково надавати гачки для представлення, щоб передавати події в модель.

  Однією з важливих стратегій реалізації цього рівня є відокремлення його від View, тобто ViewModel не повинна знати про представлення, з яким взаємодіє.
– Перегляд: нарешті, роль перегляду в цьому шаблоні полягає в тому, щоб спостерігати (або підписатися на) спостережувану ViewModel, щоб отримати дані, щоб відповідно оновити елементи інтерфейсу користувача.

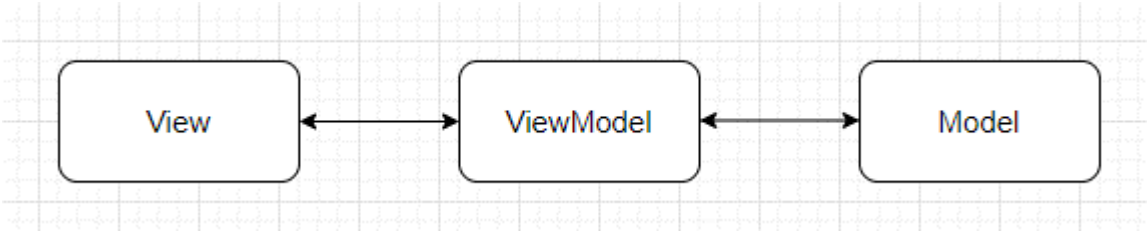


Рисунок 2.4 – структура MVMM

Найчастіше для web-додатків вибирають MVC архітектуру, але для веб її змінюють. В MVC для веб модель не може на пряму спілкуватися з виглядом, вона це робить через контролер.

Тому для розробки автоматизованої системи управління готелем було обрано архітектуру MVC, так як вона являється найкращим варіантом для розробки web-додатків.

2.2 Проектування структури графічного інтерфейсу web-додатка

Загальна структура усіх вікон буде складатися з меню та контенту який буде знаходитись посередині.

У шапці додатку буде знаходитися логотип, деякий текст та кнопка з випадаючим блоком з даними про адміністратора та кнопками для виходу. Зліва екрану буде знаходитись меню, а у центрі буде основний контент який буде змінюватися від переходу на іншу сторінку.

На рисунках 2.1, 2.2 та 2.3 зображені з проектовані структурні схеми інтерфейсів вікон web-додатку.

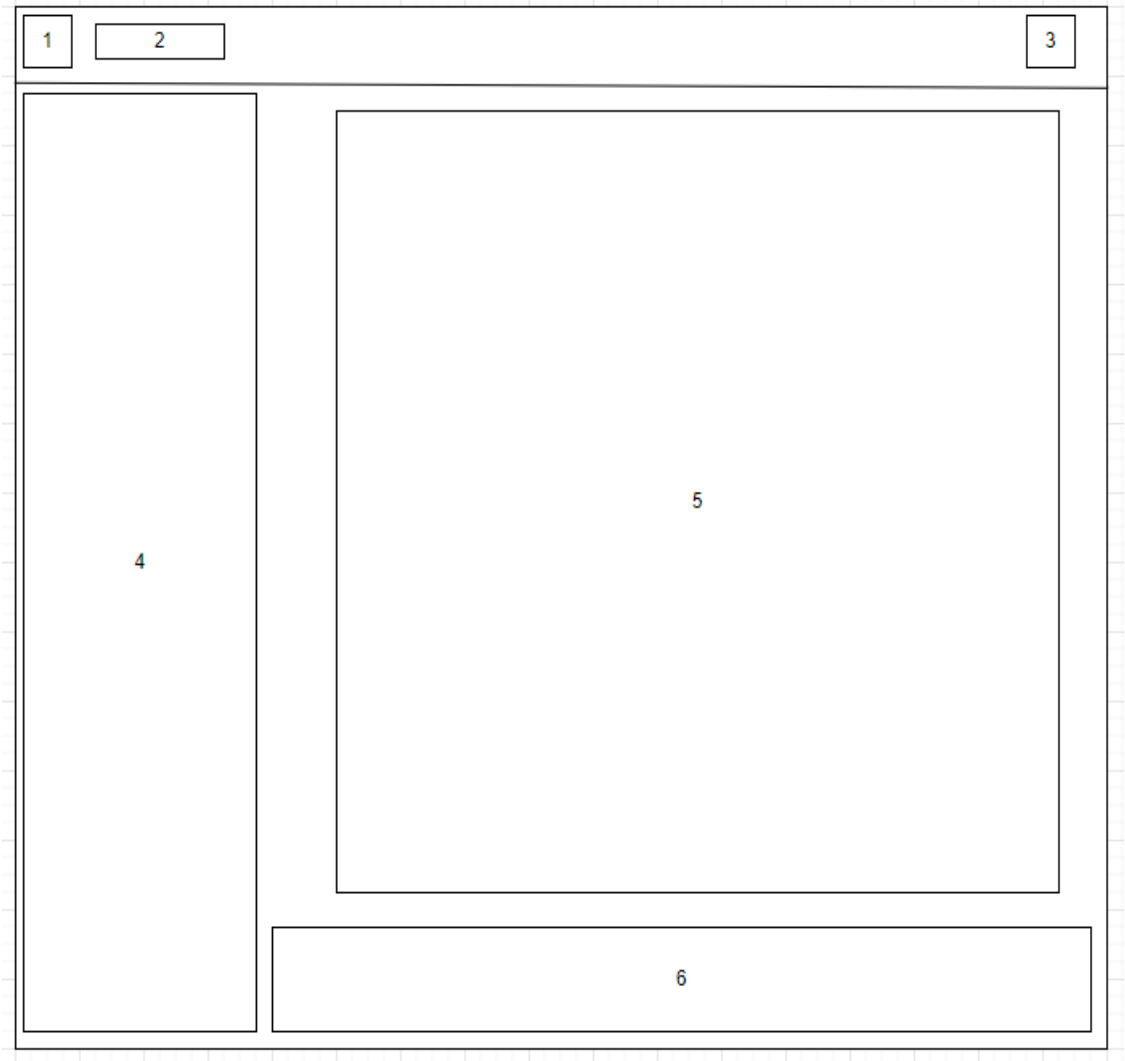


Рисунок 2.5 – Графічна схема головного вікна боту

Основні елементи інтерфейсу головного вікна боту «Expense Control»:

1. Логотип.

2. Текст.

3. Кнопка «Адміністратора».

4. Меню.

5. Контент.

6. Футер.

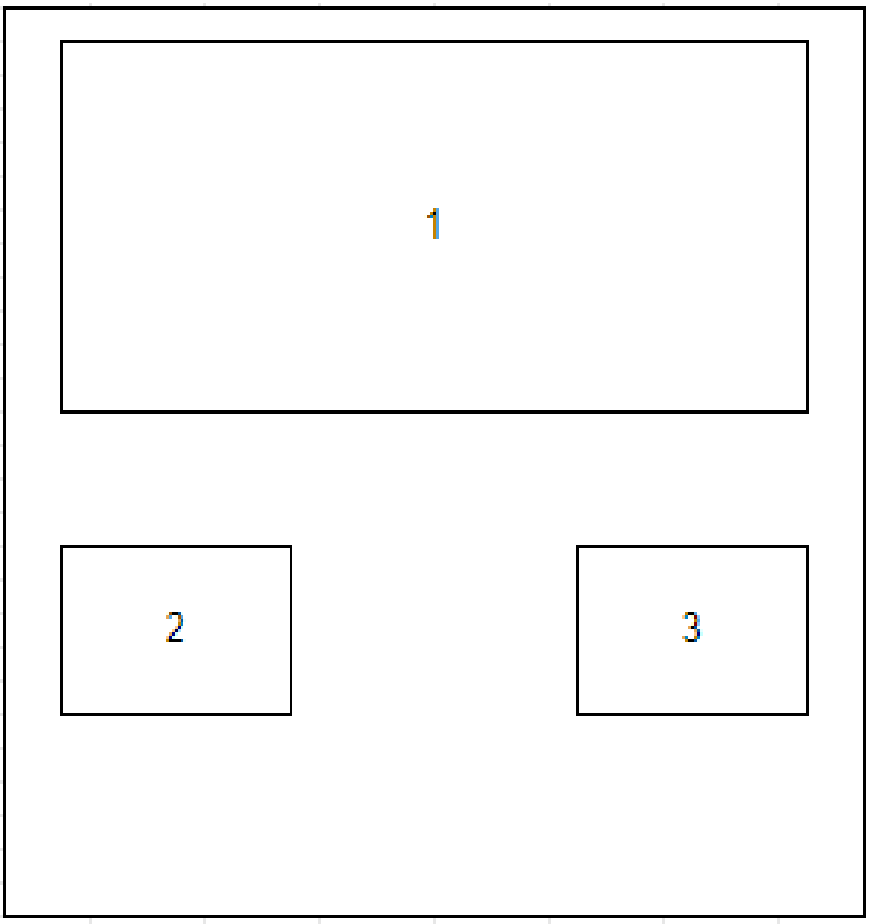


Рисунок 2.6 – Графічна схема модального вікна адміністратора

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 пралодя

5. Кнопка «Ввійти»

6. Футер

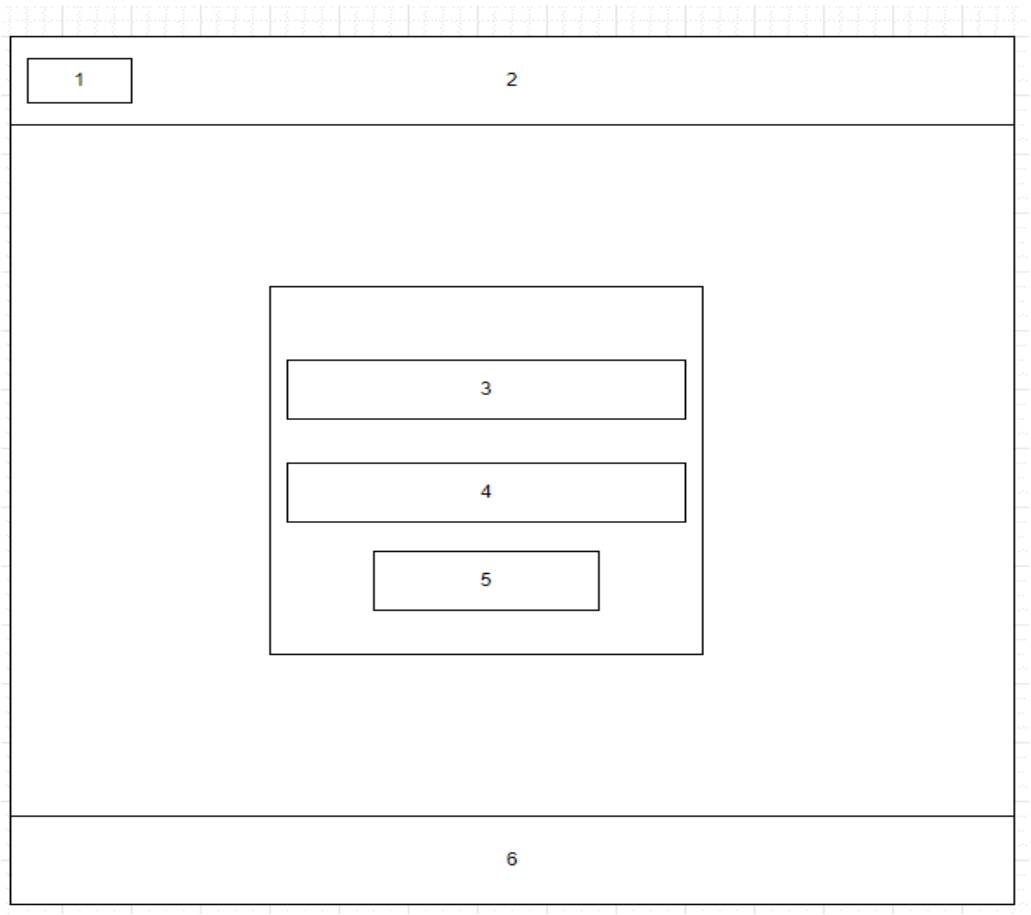


Рисунок 2.7 – Графічна схема модального вікна адміністратора

Проектування інтерфейсу є важливим етапом у процесі розробки web-додатка, оскільки зручність та зрозумілість інтерфейсу являється одним з основних факторів для вибору клієнтом.

2.3 Проектування методів та основних алгоритмів web-додатку

Алгоритм додавання нових постояльців в систему складається із наступних кроків (рисунок 2.4).:

Крок 1. Початок.

Крок 2. Отримати від адміністратора дані із форми.

Крок 3. Перевірити на коректність введені дані. Якщо дані коректні – перейти до кроку 4, якщо ні – до кроку 7.

Крок 4. Записати дані в база даних.

Крок 5. Вивести повідомлення “Постоялець доданий”.

Крок 6. Вивести повідомлення “Дані не коректні”.

Крок 7. Кінець.

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

Одним з важливих акспетів є звіти тому з проектуємо алгоритм генерування текстових та графічних звітів(рисунок 2.5).

Алгоритм генерування текстових та графічних звітів про витрати складається із наступних кроків:

Крок 1. Початок.

Крок 2. Отримати від користувача тип звіту.

Крок 3. Вибрати із бази даних потрібні дані.

Крок 4. Проаналізувати дані з бд.

Крок 5. Вивести кнопки «Звіт графічно» та «Звіт текстом», якщо нажата перша – перейти до кроку 6, якщо друга – перейти до кроку 8.

Крок 6. Побудувати діаграми.

Крок 7. Вивести на екран звіт в графічні формі.

Крок 8. Вивести на екран звіт текстом.

Крок 9 Кінець.

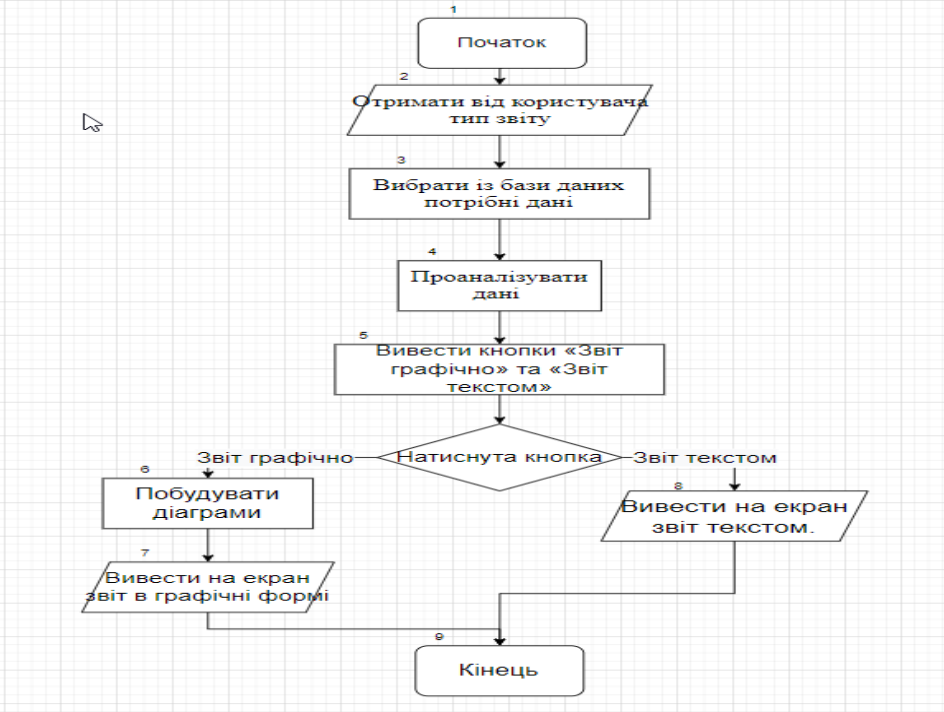


Рисунок 2.9 – Блок-схема алгоритму генерування текстових та графічних звітів

Один із основних алгоритмів у системі, є алгортм авторизації він складається з наступних кроків(рисунок 2.6).

Крок 1. Початок.

Крок 2. Ввід данних у поля авторизації.

Крок 3. Перевірити на коректність введені дані. Якщо дані коректні – перейти до кроку 4, якщо ні – до кроку 7.

Крок 4. Перевірити чи користувач із такими даними є у базі даних. Якщо є – перейти до кроку 5, якщо ні – до кроку 7.

Крок 5. Зберигти дані користувача в сесію.

Крок 6. Виконати редирект на головну сторінку системи.

Крок 7. Вивести повідомлення “Дані не коректні”.

Крок 8. Кінець.

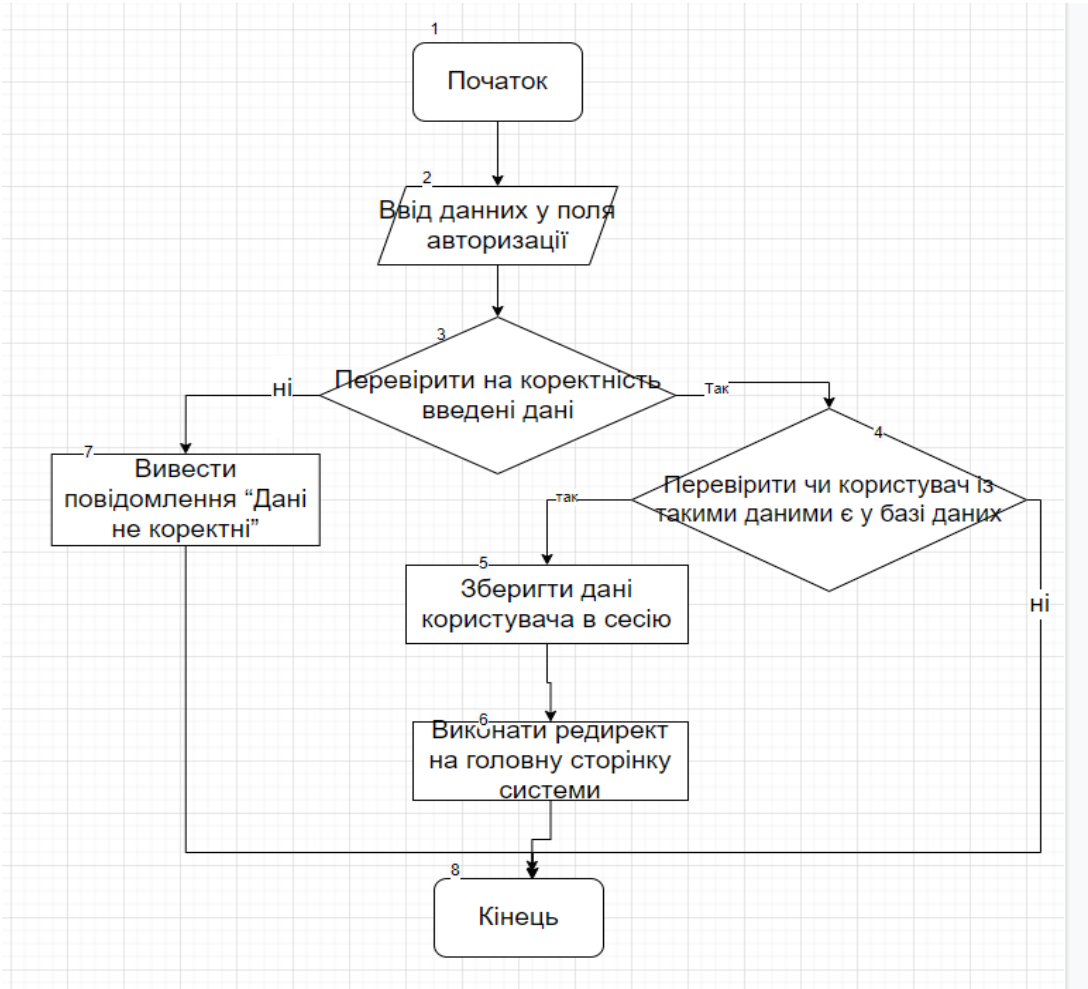


Рисунок 2.10 – Блок-схема авторизації

2.4 Висновки

У другому розділі було розроблено схему інтерфейсу який буде зручним у використанні та зрозумілим для користувача. Було розроблено алгоритм додавання постояльців в систему та побудовано блок-схему алгоритму. Також було розроблено алгоритм генерування текстових та графічних звітів та побудовано блок-схему.

3 РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТ ДЛЯ WEB-ДОДАТКУ

3.1 Варіантний аналіз і обґрунтування вибору мови програмування

Розглянемо найбільш популярні мови програмування, які використовуються для розробки сайтів та web-додатків, а саме PHP, JavaScript та Python.

PHP – скриптова мова програмування, за допомогою якої можна створювати додатки та веб-сайти, причому як прості одно сторінкові лендінги, так і величезні важко-навантажуванні системи, які відвідують сотні тисяч користувачів що дня. PHP являючись однією із перших мов програмування саме для веб-розробки пройшла довгий шлях практично із самого початку зародження інтернету, саме через це залишається однією із найпопулярніших та найбільш затребуваних мов програмування у веб-розробці [11].

До особливостей мови програмування PHP відносять наступні:

– інтеграція з базами даних. PHP підтримує безліч баз даних, таких як Oracle, MySQL та інші;

– простота використання;

– швидкість. Він швидше, ніж інші мови скриптів;

– популярність. Наявність великого різноманіття джерел інформації про PHP.

– відкритий вихідний код.

JavaScript (JS) — динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки [12]:

– популярність. Наявність великого різноманіття джерел інформації про JavaScript;

– простота використання;

– велика кількість фреймворків.

Python – інтерпретована об'єктно-орієнтована мова програмування високого рівня із строгою динамічною типізацією. Дана мова програмування в першу чергу орієнтована на підвищення продуктивності розробника за рахунок простоти свого коду та зрозумілості документації. Це мова, яка застосовується для вирішення широкого спектру завдань. Найчастіше Python застосовують в роботі з великими даними і при розробці сайтів і мобільних ігор [13].

Python, як і будь-яка інша мова програмування, має свої особливості:

– кросплатформність. Python – це інтерпретована мова, його інтерпретатори існують для багатьох платформ, тому з його запуском на будь-якій ОС не повинно виникнути проблем;

– поширеність. Python доступна величезна кількість сервісів, середовищ розробки і фреймворків;

– популярність. Наявність великого різноманіття джерел інформації про Python.

Результати порівняння розглянутих мов програмування за обраними критеріями наведено в таблиці 3.1. Для позначення наявності критерію використовується “+”, а для відсутності використовується “-”, а для того часткової наявності використовується “+-”.

Таблиця 3.1 – Порівняння мов програмування

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

3.2 Вибір середовища розробки та бази даних

Не менш важливим, ніж питання вибору мови програмування, є питання вибору інтегрованого середовища розробки. Інтегрованим середовищем розробки (IDE) – це комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм [14]. Більшість сучасних середовищ розробки мають можливість авто-доповнення коду. Розглянемо найбільш популярні інтегровані середовища розробки для мови програмування PHP, а саме PhpStorm, Eclipse та Visual Studio Code.

PhpStorm являє собою інтегроване середовище розробки для PHP, HTML і JavaScript з можливостями аналізу коду на льоту, запобігання помилок у сирцевому коді і автоматизованими засобами рефакторинга для PHP і JavaScript. Авто-доповнення коду в PhpStorm підтримує специфікацію PHP, включаючи генератори, співпрограми, простори імен, замикання, типажі і синтаксис коротких масивів. Присутній повноцінний SQL-редактор з можливістю редагування отриманих результатів запитів [15].

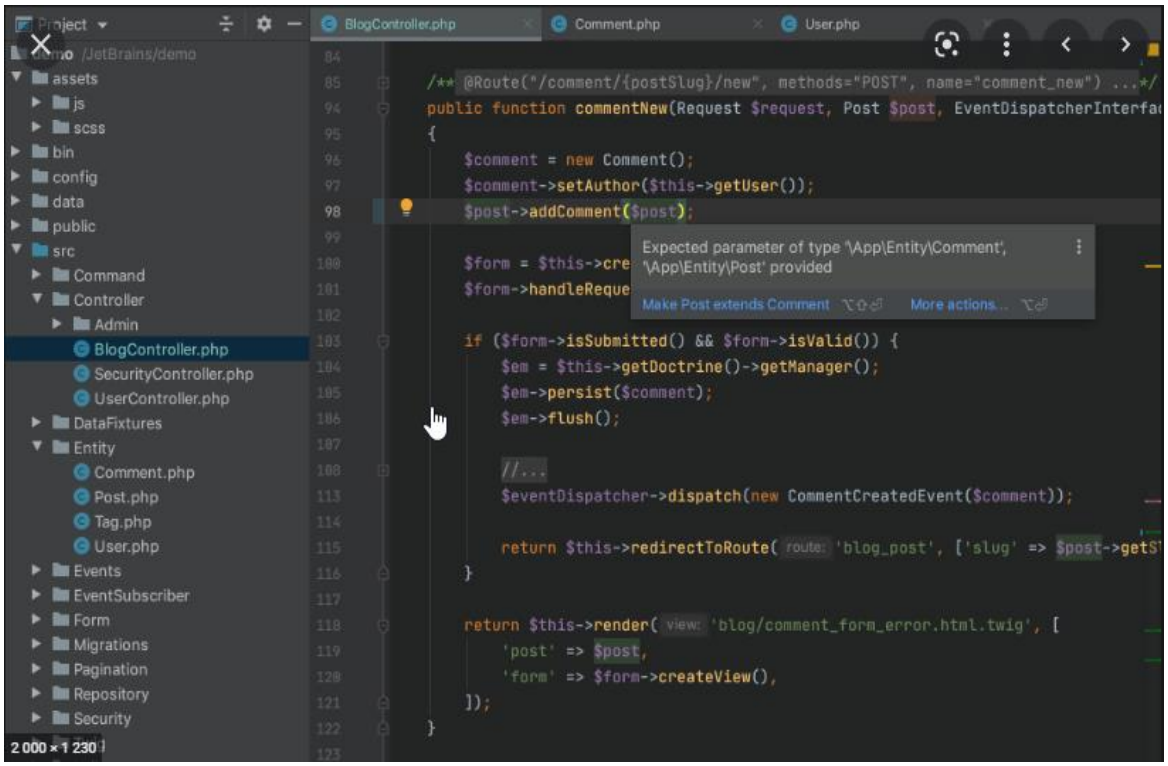


Рисунок 3.1 – Інтерфейс програмного додатку «PhpStorm»

Eclipse – вільне модульне інтегроване середовище розробки програмного забезпечення. Розроблене компанією Eclipse Foundation і включає проекти, такі як платформа Eclipse, набір інструментів для програмістів на мові Java, системи контролю версій, конструктори GUI тощо. Написаний в основному на Java, може бути використаний для розробки застосунків на Java і, за допомогою різних плагінів, на інших мовах програмування, включаючи PHP [16].

Переваги: Підсвічування коду. Підказки при помилках. Підказки при наборі коду. Автоматичне форматування коду.

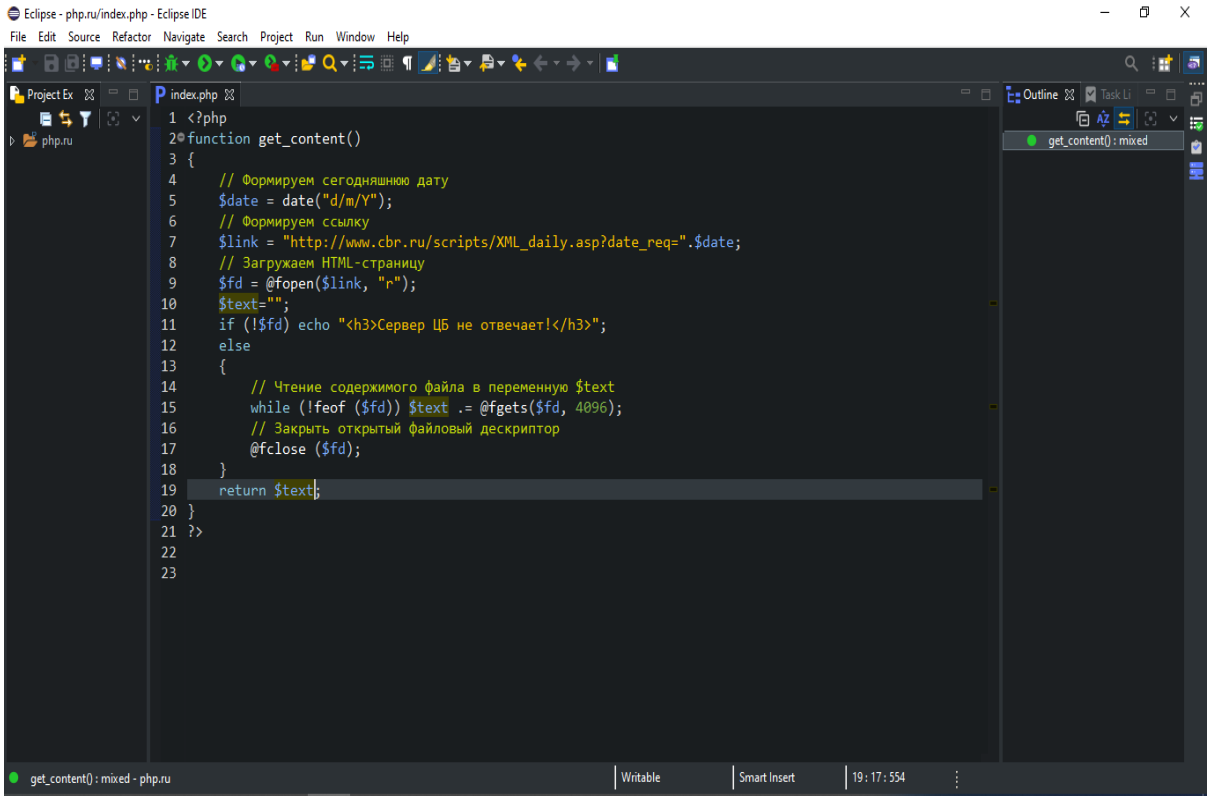


Рисунок 3.2 – Інтерфейс програмного додатку «Eclipse»

Visual Studio Code – це один із найпопулярніших редакторів коду, розроблений корпорацією Microsoft. Він розповсюджується у безкоштовному доступі та підтримується всіма актуальними операційними системами: Windows, Linux та macOS. VS Code є звичайним текстовим редактором з можливістю підключення різних плагінів, що дає можливість працювати з різними мовами програмування для розробки будь-якого IT-продукту [17].

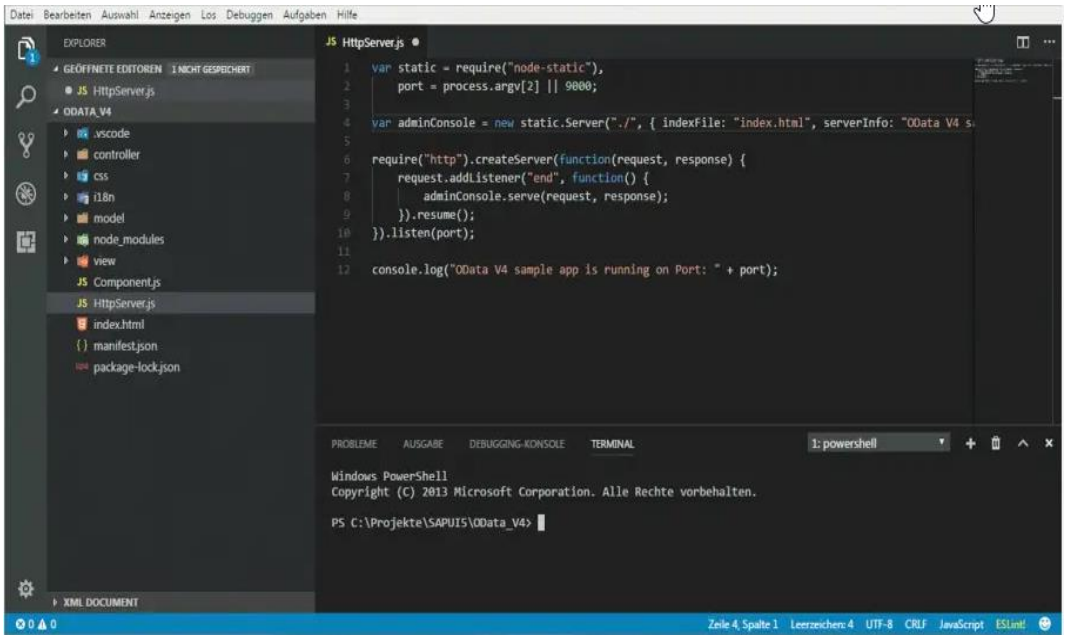


Рисунок 3.3 – Інтерфейс програмного додатку «Eclipse»

Результати порівняння розглянутих інтегрованих середовищ розробки за обраними критеріями наведено в таблиці 3.2. Для позначення наявності критерію використовується “+”, а для відсутності використовується “-”, а для того часткової наявності використовується “+-”..

Таблиця 3.2 – Порівняння інтегрованих середовищ розробки

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

Після порівняння середовищ розробки для мови програмування PHP, наведеного у таблиці 3.2 можна зробити висновок, що саме PhpStorm виявився найкращи варіантом серед усіх запропонованих.

Для зберігання даних системи, потрібно визначитися з базою даних, тому виконаємо порівняння найпопулярніших з них, а саме PostgreSQL, SQLite та MySQL.

PostgreSQL — це розширена реляційна база даних корпоративного класу з відкритим кодом, яка підтримує запити як SQL (реляційні), так і JSON (нереляційні). Це високо-стабільна система керування базами даних, що підтримується більш ніж 20-річним розвитком спільноти, що сприяло її високому рівню стійкості, цілісності та коректності. PostgreSQL використовується як основне сховище даних або сховище даних для багатьох веб-, мобільних, геопросторових та аналітичних додатків. Проте її надто широкий функціонал та складність у порівнянні із MySQL та SQLite роблять її не найкращим вибором [18].

SQLite — це швидка і легка одно-файлова СУБД, що вбудовується, на мові C, яка не має сервера і дозволяє зберігати всю базу локально на одному пристрої. Для роботи SQLite не потрібні сторонні бібліотеки або служби. Поняття "вбудований" означає, що СУБД не використовує парадигму клієнт-сервер. Двигун SQLite — це не окремий процес, з яким взаємодіє програма, а бібліотека. Програма компонується з нею, і двигун служить складовою програми. Як протокол обміну використовуються виклики функцій (API) бібліотеки SQLite. SQLite здебільшого використовують для не великих проектів, тому для нашої системи воне не підходить [19].

MySQL — це система управління реляційною базою даних (RDBMS), розроблена Oracle, яка базується на мові структурованих запитів (SQL) [20]. MySQL є однією з найбільш пізнаваних технологій в сучасній екосистемі великих даних. Його часто називають найпопулярнішою базою даних і в даний час користуються широким та ефективним використанням незалежно від галузі.

Очевидно, що будь-хто, хто займається корпоративними даними або загальними IT, повинен принаймні прагнути до базового знайомства з MySQL. За допомогою MySQL навіть ті, хто вперше в реляційних системах, можуть відразу створити швидкі, потужні та безпечні системи зберігання даних. Програмний синтаксис та інтерфейси MySQL також є ідеальними шлюзами у широкий світ інших популярних мов запитів і сховищ структурованих даних. Через її простоту, безпеку MySQL являється найкращим вибором для розробки системи управління отелем.

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 важконавантажених проектів	+	-	+
Підсумковий результат	2	2.5	4

Згідно таблиці 3.3, можна дійти до висновку що MySQL є найкращим варіантом для розробки web-додатку система управління готелем.

Отже, на основі проведеного аналізу засобів розробки було визначено, що мова програмування Python, середовище розробки Visual Studio Code та СУБД SQLite є найкращим вибором для реалізації розроблюваного Telegram-боту.

3.3 Програмна реалізація web-додатку

Для реалізації web-додатку було реалізовано велику кількість алгоритмів. Основними серед них є наступні:

– алгоритм авторизації користувача в систему;

– алгоритм додавання готелів в систему;

– алгоритм додавання постояльців в систему;

– алгоритм визначення суми яка є на рахунку системи;

– алгоритм визначення суми яка є на рахунку у постояльців;

– алгоритм визначення заборгованості у постояльців;

– алгоритм генерування текстових звітів про витрати постояльця за обраний ним період;

– алгоритм генерування графічних звітів про витрати постояльця за обраний ним період;

Для початку роботи у системі адміністратор в першу чергу повинен авторизуватись. На сторінці входу адміністратор повинен ввести дані для входу, після чого відбудиться валідація даних. Тоді якщо дані валідні відбувається запит до бази даних і перевіряється чи користувач з такими даними являється адміністратором, якщо так то виконується вхід і дані користувача записуються в сесію.

```
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 збереженні',
            ]);
        }
```

Для виходу з системи виконується інший алгоритм, який видаляє користувача з сесії.

```
public function destroy(Request $request)
```

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

Далі один з основних алгоритмів web-додатку це алгоритм, який дозволяє додавати нові готелі у систему. Для початку адміністратор на сторінці додавання готелю повинен ввести усі дані які вимагає система у форму і відправляє їх на сервер.

```
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!');
  }
```

Спочатку виконується валідація, якщо дані не валідні то на сторінці виводиться повідомлення про це. Якщо дані валідні то, спочатку збурігаються фото на сервері, за збереження фото відповідає алгоритм збереження фото.

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

Після чого виконується функція для збереження готелю яка при позитивному результаті повертає ID нового готелю.

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

Цей ID потрібен для збереження даних про поверхи та секції готелю, він передається у функцію збереження поверхів і секцій.

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

Для того щоб додати нового постояльця потрібно щоб в готелі були номера, тому також було розроблено алгоритм додавання номерів. Після того як на сторінці додавання номерів адміністратор введе усі дані, відбувається валідація даних, якщо дані не валідні на сторінці виводиться про це повідомлення, тоді якщо дані валідні виконується збереження номера.

```
        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!');  }
```

Після того як, усі дані добавлені, можна посиляти постояльця в номер. Для того щоб додати постояльця на сторінці у формі додавання постояльця, адміністратор повинен заповнити усі поля та підтвердити відправлення форми. Якщо усі дані валідні, то виконується додавання постояльця, якщо ні то виводиться повідомлення про це.

```
                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!');  }
```

Для правильної роботи готелю, потрібно вести облік грошових коштів що приходять і виходять з системи. Тому для початку продемонструю як саме можуть приходити кошти на рахунок системи та рахунок постояльця, для цього розроблений алгоритм приходу коштів. При оплаті послуг зразу у готелі, адміністратор повинен занести в систему приходної відомості. Для цього на спеціальній сторінці адміністратор вводить суму та вибирає постояльця якщо це прихідна відомость на рахунок постояльця, після чого відправляє форму і на сервері зберігається прихід в системі.

```
public function store(Request $request)
{
$data = $request->all();
$modelTransaction = new AccountTransaction();

$modelTransaction->store($data);
return redirect()->back()->withSuccess('Успішно!');
}.
```

А відхід із каси виконується як квитанція з рахунку постояльця. Для початку адміністратор заповнює дані, такі як сума, дані постояльця, послуга за яку виписується квитанція.

```
public function store(Request $request)
    {
        $modelInvoice = new Invoice();

        $data = $request->all();

        $invoice = $modelInvoice->store($data);

        return redirect()->back()->withSuccess('Успішно!');
    }
```

Алгоритм визначення усієї суми у касі. Спочатку із бази даних в дві зміних витягуються прихід і відхід грошових коштів.

```
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;    }
```

Після цього, відбувається перебір постояльців у циклі і для кожного постояльця виконується функція getBalance(), яка повертає баланс на рахунку постояльця.

```
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 виконання функції getBalance(), якщо баланс на рахунку позитивний то додаємо його до змінної $total яка являється усією сумою яка є на рахунках у постояльців.

Алгоритм який визначає суму заборгованості на рахунках постояльців. Спочатку із бази даних вибираються постояльці.

```
          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;
}
```

Далі відбувається перебір постояльців у циклі і для кожного постояльця виконується функція getBalance(), яка повертає баланс на рахунку постояльця. Після виконання функції getBalance(), якщо баланс на рахунку негативний то додаємо його до змінної $total яка являється усією сумою заборгованості яка є нарахунках у постояльців.

Також було розроблено декілька алгоритмів які на стороні клієнта додають на сторінку потрібні елементи, без перезагрузки сторінки. При натисканні кнопки з ID add-tariff-service, спочатку виконується очищення місця куди буде доданий елемент від інших елементів, після чого виконується ajax запит, який повертає потрібні дані після чого вони проставляються у потрібні місця html коду, який свою чергу додається на сторінку.

```
$(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");
                        }
                    }) })
                }          });    });
 });
```

3.4 Висновки

У даному розділі було проведено аналіз мов програмування PHP, JavaScript та Python, середовищ розробки PhpStorm, Eclipse та Visual Studio Code і баз даних PostgreSQL, SQLite та MySQL. В результаті даного аналізу було вирішено що розроблятися web-додаток буде на мові PHP в середовищі розробки PhpStorm а зберігатися дані будуть у базі даних MySQL. Крім того, було проведено опис основних алгоритмів web-додатку.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Аналіз методів тестування програмного забезпечення

Тестування програмного забезпечення (software testing) – це процес аналізу чи експлуатації програмного забезпечення з метою виявлення дефектів. Як правило, даний етап циклу розробки програмного забезпечення є найтривалішим та найбільш ресурсозатратним. Виділяють два основних види тестування: статичне та динамічне [21].

Статичне тестування — це набір методів тестування програмного забезпечення для перевірки коду без його виконання. Він включає статичні огляди та статичний аналіз, які є формою статичного тестування.

Статичні огляди – метод тестування який включає пошук помилок або розуміння документації чи коду, що перевіряється. Типи помилок, виявлені статичними оглядами, включають недержавні стандарти кодування, відсутні вимоги та поганий дизайн.

Статичний аналіз використовує інструменти для пошуку вразливостей програмного забезпечення, схильних до помилок методів кодування, залежностей, недійсної перевірки даних, логічних помилок тощо.

Через складність сучасних мов програмування статичний аналіз зазвичай виконується за допомогою інструментів, тоді як статичні огляди зазвичай виконуються вручну.

Динамічне тестування — це метод тестування програмного забезпечення, за допомогою якого досліджується поведінка програмного забезпечення під час виконання. На відміну від статичного тестування, динамічне тестування компілює та запускає код, що переглядається. Це передбачає надання програмному забезпеченню вхідних даних та перевірку його результатів. Динамічне тестування має на меті перевірити, чи працює програмне забезпечення належним чином.

Тестування методом білого ящика – цей метод заснований на тому, що розробник тесту має доступ до коду програм і може писати код, який пов’язаний з бібліотеками ПЗ, що тестується. Тестування білого ящика – це поглиблення під внутрішній устрій системи, за межі її зовнішніх інтерфейсів.

Тестування методом чорного ящика – цей метод заснований на тому, що тестувальник має доступ до ПЗ тільки через ті ж інтерфейси, що і замовник або користувач, або зовнішні інтерфейси, що дозволяють іншому комп’ютеру або іншому процесу підключитися до системи для тестування. Тест-дизайн, заснований на техніці чорного ящика – процедура вибору тест-кейсів на основі аналізу функціональної або нефункціональної специфікації компонента або системи без знання її внутрішнього устрою.

Тестування зручності використання – це метод тестування, спрямований на встановлення ступеня зручності використання, навченості, зрозумілості та привабливості для користувачів розроблювального продукту в контексті заданих умов .

Функціональне тестування – передбачає аналіз функціональних характеристик додатка та перевірку на невідповідності між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і бізнес-вимоги. Функціональне тестування фактично імітує фактичне використання системи .

У результаті порівняння методик тестування, для тестування інтерфейсу і функціоналу web-додатку було обрано метод «чорного ящика», через те, що проаналізувавши вимоги злегкістю зможемо визначити тест кейси для перевірки різної функціональності.

4.2 Особливості тестування веб-додатків

Розглянемо особливості тестування веб-додатків.

При тестувані верстки потрібно звернути увагу на такі пункти:

Візуальна частина:

- неправильне відображення блоків, що становлять інтерфейс, не стикування кольорової гами;
- тестування локалізованих версій (переклад сайту);
- відповідність макету (шари у PhotoShop);
- при зменшенні/збільшенні масштабів (75-150%) без візуальних недоліків;
- підсвічування полів із помилками;
- перевірка у дозволах (+прокручування);

Перевірити можна так: FirefoxMenu -> Інструменти -> Веб-розробник -> Адаптивний дизайн або Resolution Test Plugin у Chrome.

Доступність та відсутність JS помилок:

- чи натискаються клікабельні елементи (внутрішні/зовнішні посилання, посилання на електронну пошту, кнопки, іконки);
- при наведенні на клікабельні змінюється курсор, інакше – ні;
- підказки на незрозумілих клікабельних елементах;
- при відключенні зображень повинні бути підписи невеликим сірим кольором (Web Developer -> Images -> Replace Image With Alt Attributes);
- працездатність при вимкненому JS. Критичні функції повинні бути доступні без JS (в Web Developer -> Disable -> Disable JS -> All JS)

Коректна робота, надійна верстка:

- перевірка роботи з даними (введення великої та малої кількості тексту у форму; блоки з контентом змінюються місцями (Firebug (HTML -> Edit)));
- перевірка роботи стилів (введення тексту із заголовками, з абзацом та без, з картинками).

Так як в класичній ситуації клієнт є браузером, тому тестування кросбраузерності є доволі актуальним.

Кросбраузерність – це властивість сайту однаково відображатися та функціонувати у відповідності до поставленого завдання в усіх браузерах.

Кросбраузерне тестування – це тип нефункціонального тестування, який дає змогу перевірити, чи працює ваш веб-сайт належним чином, коли клієнт отримуєте доступ за допомогою [21].:

Різні комбінації браузера та ОС , наприклад, у популярних браузерах, таких як Firefox, Chrome, Edge, Safari, у будь-якій із популярних операційних систем, як-от Windows, macOS, iOS та Android.

Різні пристрої , наприклад, користувачі можуть переглядати ваш веб-сайт і взаємодіяти з ним на популярних пристроях — смартфонах, планшетах, настільних комп'ютерах, ноутбуках тощо.

Допоміжні інструменти, тобто веб-сайт сумісний з допоміжними технологіями, такими як програми зчитування з екрана, для людей з різними можливостями.

Йдеться про доставку випусків, які максимально залежать від веб-переглядача, що є ключем до забезпечення однорідної роботи користувачів у різноманітному, постійно зростаючому діапазоні браузерів/пристроїв.

Розглянемо які саме функції аналізуються під час кросбраузерного тестування.

Класифікуємо функції, які будуть проходити тестування, таким чином:

Основна функціональність: для забезпечення роботи основних функцій у більшості комбінацій браузера та ОС. Наприклад, розробник може тестувати, щоб перевірити, що:

- Усі діалогові вікна та меню працюють належним чином;
- Усі поля форми приймають введення після їх правильної перевірки;
- Веб-сайт правильно обробляє файли cookie першої сторони (і такі функції, як персоналізація, які залежать від них);
- Безперебійне сенсорне введення для мобільних телефонів або планшетів.

Дизайн: це гарантує, що зовнішній вигляд веб-сайту — шрифти, зображення та макет — відповідає специфікаціям, наданим командою дизайнерів.

Доступність: враховує відповідність Правилам доступності веб-вмісту (WCAG), щоб дозволити користувачам з різними можливостями отримати доступ до веб-сайту.

Відповідність: перевіряє, що дизайн плавний і підходить для різних розмірів/орієнтацій екрана.

Однією з важливих складових інтернет-додатків є форми для заповнення, взаємодію з якими користувач здійснює за допомогою того ж таки уважно розглянутого клієнта. Проте, ці форми дуже часто є джерелом дефектів, які, влаштовуючись у «продакшені», можуть завдати великих фінансових і репутаційних збитків компанії.

Щоб не пропустити дефекти форм на продакшин розглянемо кілька простих кроків:

1. Ретельно перевіряємо обов'язковість заповнення полів та наявність відповідного маркування у них.

2. Заповнивши та надіславши форму, переконуємось у тому, що з даними відбувається саме те, що заплановано. Якщо дані мають бути внесені до бази даних, перевіряємо, чи коректно завершився процес (зрештою, про це можна попросити розробника, якщо не вистачає своїх знань SQL або прав доступу до БД).

3. Використовуємо чит-листи для тестування форм.

4. Перевіряємо, чи виводяться зрозумілі користувачеві інформаційні повідомлення необхідність заповнення порожніх полів після спроби надіслати форму.

5. Звертаємо увагу на реалізацію екранування символів у полях форм, що є потенційним джерелом вразливостей для програми та користувачів. Екранування має здійснюватися лише на рівні клієнта, а й сервера, відключити який у клієнті досить просто (наприклад, з допомогою спеціальних плагінів, які

знімають всі можливі обмеження у кілька кліків, як-от Web Developer Toolbar – Forms).

6. Переконуємося, що після заповнення форми користувачеві надходить підтверджуючий лист із зазначенням відповідного відправника, а саме тіло листа відповідає вимогам (у тому числі і на працездатність посилань).

7. Використовуємо спеціальні допоміжні інструменти для тестування форм (наприклад, Web Developer Toolbar).

Також важливим аспектом тестування веб-додатків є тестування сервера.

Тестування частини веб-додатка, розміщеної на веб-сервері, можна здійснити без обходу графічного інтерфейсу (клієнта), але це вимагає певного рівня технічних знань і навичок, а також використання додаткових інструментів. Розглянемо веб-сервер з точки зору навантажувального тестування та встановлення.

Встановлення на веб-сервер Отже, перед початком тестів необхідно встановити (інсталювати) веб-додаток на веб-сервер. Отже, перед початком тестів необхідно встановити (інсталювати) веб-додаток на веб-сервер. Фактично це схоже на тестування настільних додатків, але є різниця в нюансах, які необхідно враховувати і тестувати, особливо якщо мова йде про програмне забезпечення, що розповсюджується для локальної установки на веб-серверах користувачів. .

- Для встановлення більшості веб-додатків потрібні спеціальні знання з адміністрування операційної системи. Спробуйте встановити програму на кількох веб-серверах. В ідеалі це будуть найпопулярніші технології у ваших користувачів, список яких потребуватиме попереднього дослідження.
- Встановлюючи веб-додаток, перевіряємо, чи програма дійсно встановлена в каталозі, у вказаній базі даних, використовує вибраний префікс та дотримується інших пунктів конфігурації.
- Переконайтеся, що програму можна встановити як з локального хосту, так і віддалено.

- Якщо процес встановлення є інтерактивним і вибір кожного користувача на певному етапі впливає на наступні кроки, вам доведеться пройти всі гілки, оскільки встановлення є першим кроком користувача у використанні вашої програми.
- Не забувайте про негативні тести: перевірки за умов недостатніх ресурсів (дискового простору, оперативної пам'яті) або недостатніх привілеїв, переривання процесу встановлення під час його активної фази (наприклад, відправлення сервера на перезавантаження).

Тестування навантаження імітує роботу з програмою кількох користувачів. Навантажувальний тест моделює роботу з додатком кількох користувачів. Цей тип тестування проводиться за допомогою спеціальних інструментів (наприклад jMeter), головне призначення яких — визначення профілів навантаження та штучне створення для них навантаження, що розкриває граничні можливості програми (або від сервера) у умови роботи з ним або рядом користувачів.

Отримана інформація ретельно аналізується з подальшим виявленням «тугих шийок» і кінцевих програмних і апаратних можливостей, які потім використовуються для забезпечення стабільності веб-сервера та запущеної на ньому програми.

Ось приклад із практики великого комерційного продукту, який давно працює з різними видами договорів. Одного разу вийшов ще один довгоочікуваний тип контракту, а наступного дня система повністю перестала працювати, а служба підтримки була завалена великою кількістю дзвінків. Виною тому була помилка тестування: команда перевіряла одночасну роботу з десятками тисяч контрактів, але ніхто не міг передбачити, що на практиці це будуть сотні тисяч або навіть мільйони контрактів. Тестування навантаження дозволяє виявити потенційні проблеми такого характеру на етапі тестування.

Ще одним компонентом вже згаданих веб-додатків є база даних, в якій додаток зберігає всю необхідну інформацію. Щоб база даних служила гідним сховищем інформації для вашої програми, при тестуванні слід звернути увагу на наступні основні моменти:

- Інформація, що вводиться через інтерфейс, повинна зберігатися в базі даних у незміненому (оригінальному) вигляді.
- Інформація, що зберігається в базі даних, повинна відображатися однаково в кожній частині програми (якщо бізнес-логіка програми не вимагає іншого).
- Назви таблиць і структура бази даних повинні відповідати проектній документації.
- Повинні переконатися, що запити не обробляються занадто довго і що кількість підключень достатня. Моніторинг працездатності бази даних є одним з ключових моментів тесту.
- Слід зазначити, що видалення запису в базі даних не завжди супроводжується повним видаленням сутності. Іноді використовується так зване «псевдо-видалення» і потрібно перевірити, чи правильно це зроблено.
- Рекомендується або заповнити порожню базу даних на тестовому стенді випадково згенерованими даними, або взяти дамп із виробництва та «заповнити» його базою після обфускації даних. Іноді кваліфікація тестувальника (або інші причини) не дозволяє виконати цей процес самостійно: в цьому випадку рекомендується звернутися за допомогою до фахівців інфраструктури або розробників.

Взаємодія відбувається за допомогою повідомлень (запитів і відповідей): відповідь сервера має надходити від клієнта на надісланий запит. Класичний запит/відповідь складається з 3 компонентів:

- стартова лінія;
- назва;
- текст повідомлення.

Працюючи з відповідями, тестувальнику слід спочатку звернути увагу на методи та коди статусу в стартовому рядку.

Найпопулярнішими методами є GET, HEAD і POST:

- Метод GET. Використовується для запиту вмісту, розміщеного на сервері (наприклад, GET /path/resource?param1=value1¶m2=value2 HTTP/1.1).
- Метод ГОЛОВКА. По суті, це нічим не відрізняється від наведеного вище методу, проте відповідь сервера на такий запит позбавлена «тіла», а практичне застосування зосереджено на легкому використанні, щоб отримати мінімум інформації про сервер/продукт або охоплення статусу.
- Метод POST. Цей метод використовується для передачі даних на сервер, але його база «прихована» в тілі, що відрізняє його від GET. Наприклад, на момент написання цієї статті весь текст розміщується в тілі запиту POST; Після обробки сервером стаття додається на сайт.
- Є й інші методи: PUT, DELETE, CONNECT, TRACE, PATCH тощо.

Те, що компоненти веб-додатка взаємодіють один з одним через HTTP, є гарною новиною для тестувальника, оскільки це дозволяє не тільки стежити за спілкуванням, поглибити логіку роботи та порівняти її з технічними вимогами, але й безпосередньо впливати на програму. доклавши власну руку до надісланих запитів.

Типовими програмами, які можна використовувати для створення запитів, є Fiddler або Postman. За допомогою Fiddler тестувальник може легко відстежувати всі клієнтські запити та відповіді, переглядати їх деталі, а також вносити власні зміни та надсилати модифіковані запити на сервер, щоб оцінити поведінку системи в цьому випадку.

В запитах повинно враховуватись:

- Використання правильного методу запиту. Якщо при тестувані користувач надсилає повідомлення, а на сервер надсилається лише запит GET, то тут явно щось не так.
- Здається, що натискання тієї ж функціональної клавіші генерує той самий запит. Але є прецеденти, коли натискання кнопки призводило до

ситуації, коли JavaScript переписав запит на сусідню кнопку і змінив її призначення.

– Потрібно щоб тестувальник розумів запити які надсилає. Вони досить прості для розуміння, особливо якщо мова йде про GET - вони логічно зрозумілі навіть для звичайного користувача. Синтаксичний аналіз запитів – це можливість виявити приховану помилку набагато швидше, ніж шукати її в інтерфейсі користувача.

4.3 Тестування розробленого програмного продукту

Для того щоб розпочати тестування методом «чорного ящика» потрібно розробити так звані тест-кейси. Тому для тестування web-додатку система управління готелем були розроблені настіпні тест-кейси:

Тест-кейс №1 – Авторизація в систему

1. Перейти на сторінку входу.
2. Ввести дані у поля форми і натистнути кнопку «Ввійти».
3. Перейти на головну сторінку системи.

Очікуваним результатом даного тест-кейсу є перехід на головну сторінку системи.

Для авторизації в системі введемо дані для входу(рисунок 4.1).

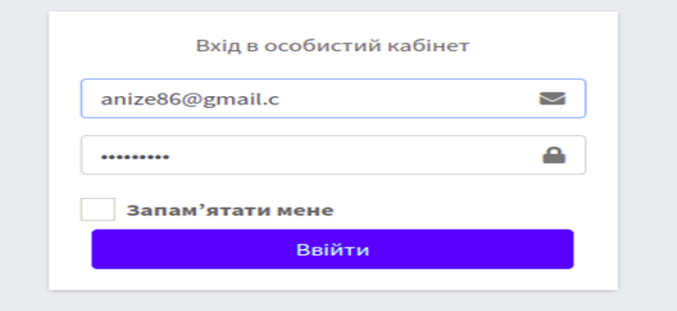


Рисунок 4.1 – Дані для авторизації

Після натиснення кнопки «Ввійти», на сервері відбувається перевірка і якщо усі дані введено вірно, нас повинно перекинути на головну сторінку системи.

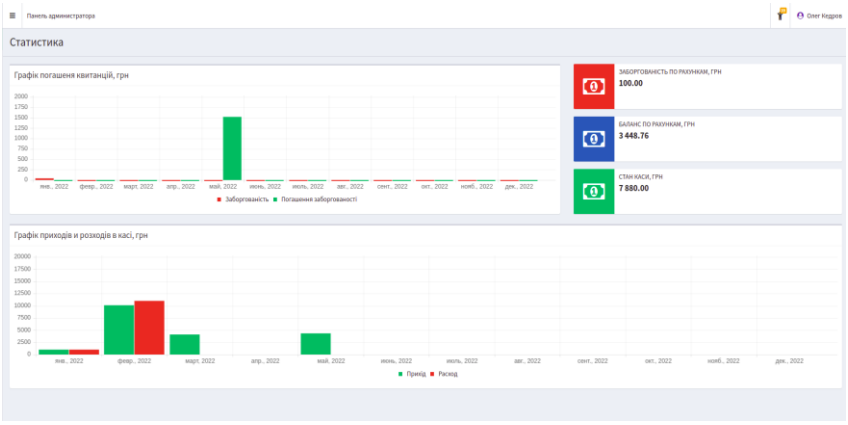


Рисунок 4.3 – Результат виконання тест-кейсу №1

Фактичний результат результат відповідає очікуваному, тому тест-кейс №1 успішно пройдено.

Тест-кейс №2 – Додавання нового готелю:

1. Перейти на сторінку додавання нового готелю.
2. Ввести усі дані Готель 1 і вул.Келицька 98.
3. Натиснути кнопку «Зберегти».
4. Отримати відповідь «Успішно».

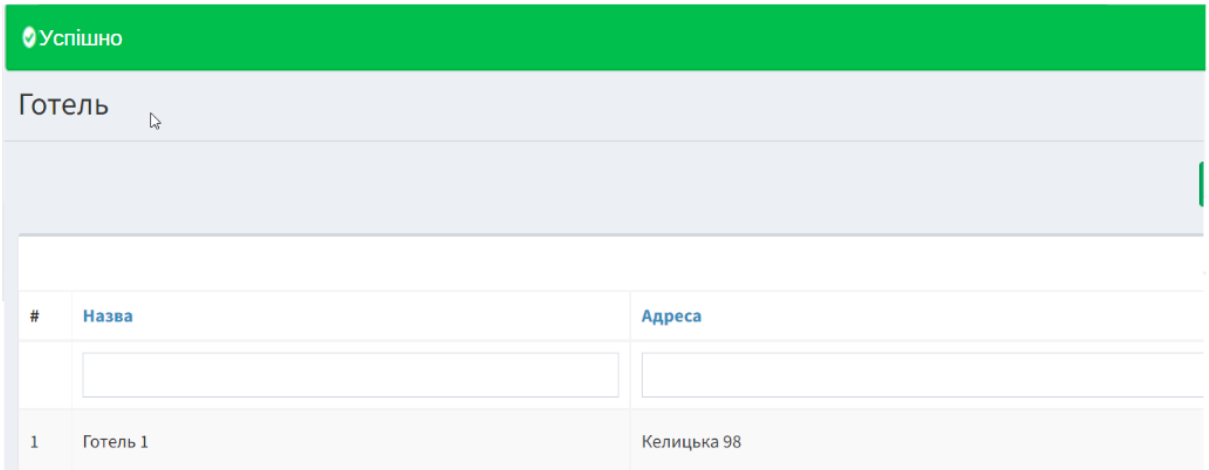


Рисунок 4.2 – Результат виконання тест-кейсу №2

В результаті виконання тест-кейсу система видала повідомлення «Успішно» і в таблиці можна побачити готель із нашими введеними даними. Тобто, фактичний результат відповідає очікуваному, тому тест-кейс №2 успішно пройдено.

Тест-кейс №3 – Додавання нового номеру:

1. Перейти на сторінку додавання нового номеру.
2. Ввести дані номеру.
3. Натиснути кнопку «Зберегти».
4. Отримати відповідь «Успішно».

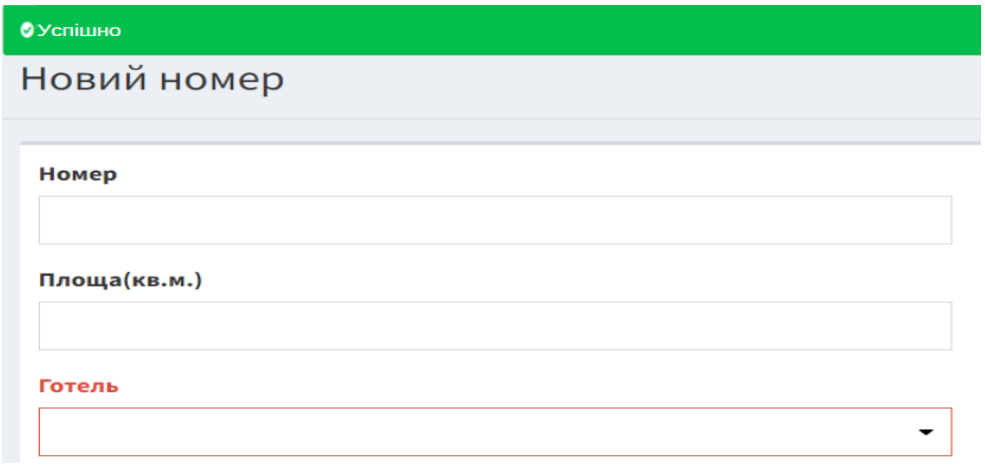


Рисунок 4.3 – Результат виконання тест-кейсу №3

В результаті виконання тест-кейсу система видала повідомлення «Успішно». Тобто, фактичний результат відповідає очікованому, тому тест-кейс №3 успішно пройдено.

Тест-кейс №4 – Додавання нового постояльця:

1. Перейти на сторінку додавання нового постояльця.
2. Ввести дані постояльця.
3. Натиснути кнопку «Зберегти».
4. Отримати відповідь «Успішно».

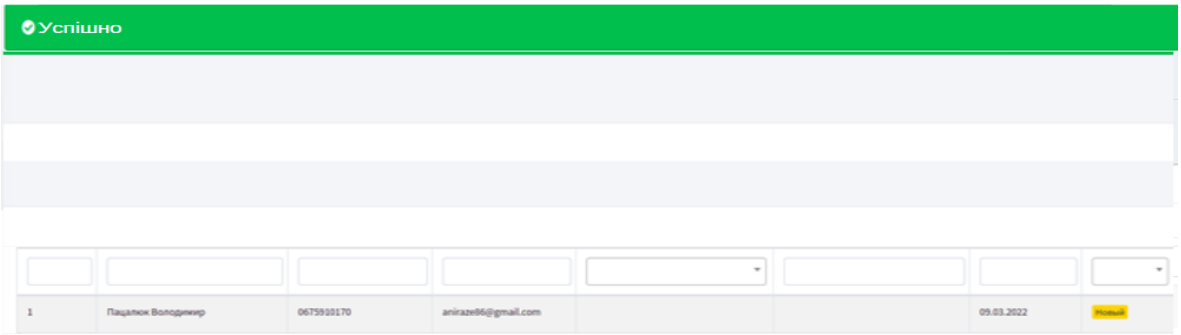


Рисунок 4.4 – Результат виконання тест-кейсу №4

Фактичний результат виконання даного тест-кейсу повністю відповідає очікуваному, отже тест-кейс №4 успішно пройдено.

Тест-кейс №5 – Поповнення рахунку системи:

1. Перейти на сторінку додавання приходу.
2. Ввести суму.
3. Натиснути кнопку «Зберегти».
4. Отримати відповідь «Успішно».

Рисунок 4.5 – Результат виконання тест-кейсу №5

Після поповнення рахунку стан каси повинен збільшитися, на рисунку 4.6 продемонстровано початковий стан та стан після виконання тест-кейсу.

Рисунок 4.6 – Наглядний приклад виконання тест-кейсу №5

Тест-кейс №6 – Поповнення рахунку системи:

2. Перейти на сторінку додавання приходу для рахунку постояльця.
3. Ввести суму.
4. Натиснути кнопку «Зберегти».
5. Отримати відповідь «Успішно».

Рисунок 4.7 – Результат виконання тест-кейсу №6

Фактичний результат виконання даного тест-кейсу повністю відповідає очікуваному, отже тест-кейс №6 успішно пройдено.

При проведенні тестування програмного додатку було отримано повну відповідність фактичних результатів очікуваним. Помилок чи багів при роботі програми не виявлено, додаток працював в точності так, як було прогнозовано відповідно до описаного функціоналу. У результаті доведено повну працездатність програмного додатку та його відповідність поставленому технічному завданню.

4.4 Висновки

В цьому розділі було описано методи тестування веб-додатків та обрано метод для проведення тестування автоматизованої системи управління готелем було обрано метод «чорної скриньки», який передбачає тестування функціоналу додатку використовуючи тест-кейси. Результат тестування програмного продукту довів його працездатність.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено автоматизовану систему управління готелем. Розроблений додаток призначений для підвищення ефективності персоналу шляхом автоматизування основних завдань адміністрації готелем.

Було проведено аналіз існуючих систем управління готелем. В результаті аналізу аналогів було висвітлено їхні основні недоліки. Проведено їхнє порівняння із власним програмним продуктом, в результаті було визначено доцільність розробки нового програмного додатку для управління готелем. Виконано постановку задач розробки програмного продукту.

Перед написанням програмного системи було з проектовано графічний інтерфейс та блок-схеми алгоритмів обчислення прогнозованої суми витрат за категоріями, сортування витрат за категоріями та їх збереження в базі даних і генерування текстових та графічних звітів про витрати. Розроблено основні модулі програми.

Було проведено варіантний аналіз та обґрунтування вибору програмних засобів при розробці програми, обрано мову програмування PHP, середовище розробки PhpStorm та СУБД MySql. В процесі розробки було описано програмну реалізацію основних модулів програми.

Було розглянуто основні види та методи тестування, обрано метод «чорного ящика». Проведено тестування за тест-кейсів, яке довело повну працездатність програмного додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Туризм [Електронний ресурс] – Режим доступу до ресурсу: https://www.tourism.gov.ua/.

2. Лілія Н. Готельно-ресторанний бізнес / Н. Лілія, Н. Наталія., 2019. – 344 с.

3. Автоматизовані системи управління готелями [Електронний ресурс] – Режим доступу до ресурсу: https://tourlib.net/statti_ukr/gudzovata.htm.

4. Пацалюк В. С. ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ У ГОТЕЛЬНОМУ БІЗНЕСІ [Електронний ресурс] / Володимир Сергійович Пацалюк // МОЛОДЬ В НАУЦІ: ДОСЛІДЖЕННЯ, ПРОБЛЕМИ, ПЕРСПЕКТИВИ (МН-2022). – 2022. – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/view/16274.

5. ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В СУЧАСНОМУ СВІТІ [Електронний ресурс] – Режим доступу до ресурсу: 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.

6. Інформаційна система [Електронний ресурс] – Режим доступу до ресурсу: https://www.britannica.com/topic/information-system.

7. Що таке комп’ютерні системи бронювання [Електронний ресурс] – Режим доступу до ресурсу: https://www.igi-global.com/dictionary/computer-reservation-systems-crs/41576..

8. Глобальна система розподілу [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%93%D0%BB%D0%BE%D0%B1%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%80%D0%BE%D0%B7%D0%BF%D0%BE%D0%B4%D1%96%D0%BB%D1%83_(GDS).

9. Інтегровані системи управління людськими ресурсами [Електронний ресурс] – Режим доступу до ресурсу: https://kbp.aero/docs/Nastanova.pdf.

10. Paul G. CRM at the Speed of Light, Fourth Edition: Social CRM 2.0 Strategies, Tools, and Techniques for Engaging Your Customers 4th Edition / Greenberg Paul., 2009. – 688 p.

11. Kevin T. Programming PHP: Creating Dynamic Web Pages / T. Kevin, M. Peter., 2020. – 544 p.

12. Jon D. JavaScript and jQuery: Interactive Front-End Web Development / Duckett Jon., 2014. – 640 p.

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

14. Інтегрованим середовищем розробки [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%BE%D0%B2%D0%B8%D1%89%D0%B5_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8.

15. PhphStorm IDE [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/phpstorm/.

16. Eclipse IDE [Електронний ресурс] – Режим доступу до ресурсу: https://www.eclipse.org/.

17. Visual Studio Code IDE [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/.

18. Regina O. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database / Obe Regina., 2017. – 312 p.

19. Jay A K. Using SQLite: Small. Fast. Reliable. Choose Any Three / Kreibich Jay A., 2010. – 530 p.

20. Joel M. Murach's MySQL / Murach Joel., 2019. – 628 p.

21. David Amos Foundations of Software Testing ISTQB Certification, 4th edition,/ Amos D., 2019. – 530 p.

22. Mohan G. Full Stack Testing / Gayathri Mohan., 2022. – 658 p.

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф.
_____ О. Н. Романюк
"__" _____ 2022 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка автоматизованої системи управління готелем»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доцент О.М.Хошаба
"__" _____ 2022 р.
Виконав:
_____ студент гр. 3ПІ-18б В. С. Пацалюк
"__" _____ 2022 р.

Вінниця – 2022 року.

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка автоматизованої системи управління готелем».

Галузь застосування – готельний бізнес.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №66 від «24» березня 2022 р. ректора по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою бакалаврської роботи є підвищення ефективності роботи персоналу шляхом автоматизації основних процесів при роботі готелю.

Призначення розробки – розробка та програмна реалізація автоматизованої системи управління готелем.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР.

1. Kevin T. Programming PHP: Creating Dynamic Web Pages / T. Kevin, M. Peter., 2020. – 544 с.

2. Автоматизовані системи управління готелями [Електронний ресурс] – Режим доступу до ресурсу: https://tourlib.net/statti_ukr/gudzovata.htm..

3. Романюк О.Н. Організація баз даних і знань / О.Н. Романюк, Т.О. Савчук // Навчальний посібник. – Вінниця: «УНІВЕРСУМ-Вінниця», 2003. – 123 с. – ISBN 966-641-081-8.

5. Технічні вимоги

Модель розробки – водоспадна; архітектурний шаблон проєктування – MVC; система управління базами даних – MySQL; мова запитів – SQL; вхідні

дані – база даних із даними готелю, номерів, постояльців; вихідні дані – звіт про постояльців; середовище розробки – PhpStorm; мова програмування – Php.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БДР;
2. Технічне завдання;
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

Додаток Б. Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка автоматизованої системи управління готелем»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Хошаба О. М.

Оригінальність	91,7
Схожість	8,3

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Пацалюк В. С.

Керівник роботи _____ Хошаба О. М.

Додаток В Лістинг програми

PersonalAccountController.php

```
<?php
    namespace App\Http\Controllers;
    use App\Models\Hotel;
    use App\Models\PersonalAccounts;
    use Illuminate\Http\Request;
    class PersonalAccountController extends Controller
    {
        public function index()
        {
            $modelHotel = new Hotel();
            $modelAccount = new PersonalAccounts();
            $Hotels = $modelHotel->getHotel();
            $accounts = $modelAccount->getAccounts();
           return view('admin.personalAccount.main', ['Hotels' => $Hotels, 'accounts' => $accounts]);
        }
        public function create()
        {
            $modelHotel = new Hotel();
            $Hotels = $modelHotel->getHotel();
            return view('admin.personalAccount.create', ['Hotels' => $Hotels]);
        }
        public function store(Request $request)
        {
            $request->validate([
                'account.number' => 'required',
            ], [
                'account.number.required' => 'Введіть номер'
            ]);
            $modelAccount = new PersonalAccounts();
            $ data = $ request-> all ();

            $modelAccount->store($data);
            return redirect()->back()->withSuccess('Успішно!');
        }
```

```
    public function show(PersonalAccounts $personalAccount)
    {
        return view('admin.personalAccount.show', ['accounts' => $personalAccount]);
    }

    public function edit(PersonalAccounts $personalAccount)
    {
        $modelHotel = new Hotel();
        $Hotels = $modelHotel->getHotel();
        return view('admin.personalAccount.create', ['Hotels' => $Hotels, 'personalAccount' => $personalAccount]);
    }
    public function update(Request $request, PersonalAccounts $personalAccount)
    {
        $request->validate([
            'account.number' => 'required|unique:personal_accounts,number,'.$personalAccount->id,
        ], [
            'account.number.required' => 'Введіть номер'
        ]);
        $ data = $ request-> all ();
        $personalAccount->getRelationValue('Number');

        $personalAccount->store($data);
        return redirect()->back()->withSuccess('Успішно!');

    }
    public function delete(int $id)
    {
        $modelAccount = new PersonalAccounts();
        $account = $modelAccount->getAccountsIds($id);
        $account->delete();
        return redirect()->back()->withSuccess('Успішно!');
    }
    public function getAccount(string $number)
    {
        $modelAccount = new PersonalAccounts();
```

```
        $account = $modelAccount->getAccountNumber($number);
        $account['Hotel'] = $account->Number->Hotels;
        $account['section'] = $account->Number->sections;
        return json_encode($account);
    }
}
```

HotelController.php

```
<?php
namespace App\Http\Controllers;
use App\Helper\ImageSaver;
use App\Models\Hotel;
use App\Models\HotelFlour;
use App\Models\HotelSection;
use Illuminate\Http\Request;
class HotelController extends Controller
{
    public function index()
    {
        $modelHotel = new Hotel();

        $Hotels = $modelHotel->getHotel();

        return view('admin.Hotel.main', ['Hotels' => $Hotels]);
    }

    public function show(int $id)
    {
        $modelHotel = new Hotel();
        $Hotel = $modelHotel->getHotelIds($id);
        return view('admin.Hotel.show', ['Hotel' => $Hotel]);
    }

    public function edit(int $id)
    {
        $modelHotel = new Hotel();
        $Hotel = $modelHotel->getHotelIds($id);
        return view('admin.Hotel.edit', ['Hotel' => $Hotel]);
```

```
}
public function create()
{
    return view('admin.Hotel.create');
}

public function save(Request $request)
{
    $request->validate([
        'HotelImage.*' => 'mimes:jpeg,jpg,png,svg|max:5000',
        'nameFlour.*' => 'required',
        'nameSection.*' => 'required',

    ]);
    $modelHotel = new Hotel();
    $modelHotelSection = новий HotelSection();
    $modelHotelFlour = новий HotelFlour();
    $imageSaved = новий 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!');
}
public function update(Request $request, $id)
{
    $request->validate([
```

```
        'HotelImage.*' => 'mimes:jpeg,jpg,png,svg|max:5000',
        'nameFlour.*' => 'required',
        'nameSection.*' => 'required',

    ]);
    $modelHotel = new Hotel();
    $modelHotelSection = новий HotelSection();
    $modelHotelFlour = новий HotelFlour();
    $imageSaved = новий ImageSaver();
    $Hotel = $modelHotel->getHotelIds($id);
    $ data = $ request-> all ();
    if ($request->file('HotelImage') !== null) {
        $file = $request->file('HotelImage');
        foreach ($file as $key => $item) {
            if ($data['pathHotelImage'] !== null) {
                $imageSaved->remove($data['pathHotelImage'][$key]);
            }
            $path[$key] = $imageSaved->upl($item, 'Hotel');
        }
        $data['HotelImages'] = $path
    }

    $HotelId = $modelHotel->updates($data, $Hotel);
    if (isset($data['nameSection'])) {
        $sectionId = $modelHotelSection->updates($data, $id);
    }
    if (isset($data['nameFlour'])) {
        $flourId = $modelHotelFlour->updates($data,$id);
    }
    return redirect()->route('adminHotelIndex')->withSuccess('Успішно!');
}
public function removeHotel($id)
{
    $modelHotel = new Hotel();
    $imageSaved = новий ImageSaver();
    $Hotel = $modelHotel->getHotelIds($id);
    $i = 1;
```

```
        foreach ($Hotel as $key => $item) {
            $imageSaved->remove($Hotel['image' . $i]);
            $i++;
        }
        $Hotel->delete();
        return redirect()->route('adminHotelIndex')->withSuccess('Успішно!');
    }
    public function removeHotelFlour($id)
    {
        $modelHotelFlour = новий HotelFlour();
        $flour = $modelHotelFlour->getFlourIds($id);
        $flour->delete();
    }
    public function removeSectionFlour($id)
    {
        $modelHotelSection = новий HotelSection();
        $section = $modelHotelSection->getSectionIds($id);
        $section->delete();
    }
    public function HotelId(int $id)
    {
        $modelHotel = new Hotel();
        $Hotel = $modelHotel->getHotelIds($id)
        return json_encode($Hotel);
    }
    public function sectionId(int $id)
    {
        $modelHotelSection = новий HotelSection();
        $section = $modelHotelSection->getSectionIds($id);
        return json_encode($section);
    }
}
```

NumberController.php

```
<?php
namespace App\Http\Controllers;
use App\Models\Number;
use App\Models\Hotel;
```

```
use App\Models\PersonalAccounts;
use App\Models\Tariff;
use App\Models\User;
use Illuminate\Http\Request;
class NumberController extends Controller
{
    public function create()
    {
        $modelTariff = новий Tariff();
        $modelAccounts = новий PersonalAccounts();
        $modelHotel = new Hotel();
        $modelUser = New User();



        $Hotel = $modelHotel->getHotel();
        $tariffs = $modelTariff->getTariff();
        $accounts = $modelAccounts->getAccounts();
        $users = $modelUser->getUsers();
        return view('admin.Number.create', ['Hotels' => $Hotel, 'tariffs' => $tariffs,
                                            'accounts' => $accounts, 'users' => $users]);
    }

    public function getHotel(Request $request): object
    {
        $modelHotel = new Hotel();
        $ data = $ request-> all ();
        $Hotel = $modelHotel->getHotelIds($data['name']);
        return response()->json($Hotel);
    }
    public function save(Request $request)
    {
        $request->validate([
            'numberNumber' => 'required',
            'Hotel' => 'required',
        ], [
                'numberNumber.required' => 'Введіть номер',
                'Hotel.required' => 'Виберіть будинок'
```

```
        ]);
    $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!');
}
public function index()
{
    $modelNumber = новий Number();
    $apartments = $modelNumber->getNumber();
    return view('admin.Number.main', ['apartments' => $apartments]);
}
public function edit($id)
{
    $modelNumber = новий Number();
    $modelTariff = новий Tariff();
    $modelAccounts = новий PersonalAccounts();
    $modelHotel = new Hotel();
    $modelUser = New User();
    $Hotel = $modelHotel->getHotel();
    $tariffs = $modelTariff->getTariff();
    $accounts = $modelAccounts->getAccounts();
    $users = $modelUser->getUsers();
    $apartments = $modelNumber->getNumberIds($id);
    return view('admin.Number.edit', ['apartments' => $apartments, 'Hotels' => $Hotel, 'tariffs' => $tariffs,
        'accounts' => $accounts, 'users' => $users]);
}
public function updates(Request $request, $id)
{
    $request->validate([
        'numberNumber' => 'required',
        'Hotel' => 'required',
```

```
'numberPersonalAccount' => 'unique:personal_accounts,number,'.$request->numberPersonalAccountId,

        ]);
        $modelApartament = new Number();
        $ data = $ request-> all ();
        $Number = $modelApartament->getNumberIds($id);
        $modelApartament->updates($data, $Number);
        return redirect()->route('adminNumberIndex')->withSuccess('Успішно!');
    }
    public function show($id)
    {
        $modelNumber = новий Number();
        $Number = $modelNumber->getNumberIds($id);
        return view('admin.Number.show', ['Number' => $Number]);
    }
    public function remove($id)
    {
        $modelNumber = новий Number();
        $Number = $modelNumber->getNumberIds($id);

        $Number->delete();
        return redirect()->route('adminNumberIndex')->withSuccess('Успішно!');
    }
    public function getApartments(int $id)
    {
        $modelApartments = new Number();
        $apartments = $modelApartments->getNumberIds($id);
        return json_encode($apartments);
    }
}
```

AccountTransactionController.php

```
<?php
namespace App\Http\Controllers;

use App\Models\AccountTransaction;
use App\Models\PersonalAccounts;
use App\Models\User;
```

```
use Illuminate\Http\Request;
class AccountTransactionController extends Controller
{


  public function create()
    {
      $modelAccount = new PersonalAccounts();
      $modelUser = New User();
      $modelTransaction = новий AccountTransaction();
      $accounts = $modelAccount->getAccounts();
      $users = $modelUser->getUsers();
      $transactionUid = $modelTransaction->getLastUid() + 1
      return view('admin.transaction.create', ['accounts' => $accounts, 'users' => $users, 'transactionUid' =>
$transactionUid]);
    }
    public function store(Request $request)
    {
      $ data = $ request-> all ();
      $modelTransaction = новий AccountTransaction();
      $modelTransaction->store($data);
      return redirect()->back()->withSuccess('Успішно!');
    }
    public function getUser($id)
    {
      $modelUser = New User();
      $user = $modelUser->getUserIds($id);
      $user->Number->accounts;
      dd($user);
      return json_encode($user);
    }
}
```

InvoiceController.php

```
<?php
namespace App\Http\Controllers;
use App\Models\AccountTransaction;
use App\Models\Hotel;
use App\Models\Invoice;
use App\Models\InvoiceService;
```

```
use App\Models\PersonalAccounts;
use App\Models\Service;
use App\Models\ServiceUnit;
use App\Models\Tariff;
use App\Models\TariffService;
use Carbon\Carbon;
use Illuminate\Http\Request;
class InvoiceController extends Controller
{
    public function index()
    {
        $modelInvoice = New Invoice();
        $modelTransaction = новий AccountTransaction();
        $modelAccount= new PersonalAccounts();
        $invoices = $modelInvoice->getInvoices();
        $data['debt'] = $modelAccount->getBalanceDebtTotal();
        $data['total'] = $modelAccount->getBalanceTotal();
        $data['cashBox'] = $modelTransaction->getCashboxBalance();
        foreach ($invoices as $invoice){
            $invoice['dateMonth'] = Carbon::createFromFormat('d-m-Y', $invoice["uid_date"])->locale('ru')->isoFormat('MMMM YYYY');
            foreach ($invoice->invoiceService як $invoiceService){
                $invoice['priceTotal'] += $invoiceService['price'];
            }

        return view('admin.invoice.main', ['invoices' => $invoices, 'data' => $data])
    }
    public function create(Request $request)
    {
        $modelInvoice = New Invoice();
        $modelHotel = new Hotel();
        $modelTariff = новий Tariff();
        $modelService = new Service();
        $modelServiceUnit = новий ServiceUnit();
        $Hotels = $modelHotel->getHotel();
        $tariffs = $modelTariff->getTariff();
        $services = $modelService->getData();
        $units = $modelServiceUnit->getData();
```

```
    $invoiceUid = $modelInvoice->getLastUid();
    $invoiceId = $request->get('invoice_id');
    $invoice = $modelInvoice->getInvoiceIds($invoiceId);
   if (isset($invoice)) {
      foreach ($invoice->invoiceService as $service) {
         $invoice['price'] += $service['price'];
      }
   }
    return view('admin.invoice.create', ['Hotels' => $Hotels, 'tariffs' => $tariffs, 'services' => $services,
       'units' => $units, 'invoiceUid' => $invoiceUid, 'invoice' => $invoice]);
}
public function store(Request $request)
{
    $modelInvoice = New Invoice();
    $ data = $ request-> all ();
    $invoice = $modelInvoice->store($data);
    return redirect()->back()->withSuccess('Успішно!');
}
public function show(Invoice $invoice)
{
    foreach ($invoice->invoiceService as $service) {
       $invoice['price'] += $service['price'];
    }
    return view('admin.invoice.show', ['invoice' => $invoice]);
}
public function edit(Invoice $invoice)
{
    $modelHotel = new Hotel();
    $modelTariff = новий Tariff();
    $modelService = new Service();
    $modelServiceUnit = новий ServiceUnit();
    $Hotels = $modelHotel->getHotel();
    $tariffs = $modelTariff->getTariff();
    $services = $modelService->getData();
    $units = $modelServiceUnit->getData();
    foreach ($invoice->invoiceService as $service) {
       $invoice['price'] += $service['price'];
```

```
        }
        return view('admin.invoice.edit', ['Hotels' => $Hotels, 'tariffs' => $tariffs, 'services' => $services,
            'units' => $units, 'invoice' => $invoice]);
    }
    public function update(Request $request, Invoice $invoice)
    {
        $ data = $ request-> all ();
        $invoiceId = $invoice->updates($data, $invoice);
        return redirect()->back()->withSuccess('Успішно!');
    }
    public function removeInvoice(int $id)
    {
        $modelInvoice = New Invoice();
        $invoice = $modelInvoice->getInvoiceIds($id);
        $invoice->delete();
        return redirect()->back()->withSuccess('Успішно!');
    }
    public function invoiceAjaxDelete(Request $request)
    {
       $ data = $ request-> all ();
       $modelInvoice = New Invoice();
       $id = $modelInvoice->removeInvoiceMany($data);
        return json_encode($id);
    }
    public function removeInvoiceService(int $id)
    {
        $modelInvoiceService = новий InvoiceService();
        $invoice = $modelInvoiceService->removeInvoiceService($id);
        return redirect()->back()->withSuccess('Успішно!');
    }
}
```

TariffController.php

```
<?php
namespace App\Http\Controllers;
use App\Models\Service;
use App\Models\ServiceUnit;
use App\Models\Tariff;
```

```
use App\Models\TariffService;
use Illuminate\Http\Request;
class TariffController extends Controller
{
    public function index()
    {
        $modelTariff = новий Tariff();
        $data = $modelTariff->getTariff();
        return view('admin.tariff.main', ['data' => $data]);
    }
    public function create(Request $request)
    {
        $modelServices = new Service();
        $modelTariff = новий Tariff();
        $tariffId = $request->get('tariff_id');
        $tariff = $modelTariff->getTariffIds($tariffId);
        $data = $modelServices->getData();
        return view('admin.tariff.create', ['data' => $data, 'tariff' => $tariff]);
    }
public function store(Request $request)
    {
        $modelTariff = новий Tariff();
        $modelService = новий TariffService();
        $ data = $ request-> all ();
        $tariffId = $modelTariff->create($data);
        $serviceId = $modelService->store($data, $tariffId);
        return redirect()->back()->withSuccess('Успішно!');
    }
    public function show(Tariff $tariff)
    {
        $tariff->getRelationValue('tariffService');
        return view('admin.tariff.show', ['tariff' => $tariff]);
    }
    public function edit(Tariff $tariff)
    {
        $modelServices = new Service();
        $tariff->getRelationValue('tariffService');
```

```
        $data = $modelServices->getData();
        return view('admin.tariff.edit', ['data' => $data, 'tariff' => $tariff]);
    }
    public function update(Request $request, Tariff $tariff)
    {
        $modelService = новий TariffService();
        $ data = $ request-> all ();
        $tariff->updates($tariff, $data);
        $serviceId = $modelService->store($data, $tariff['id']);
        return redirect()->back()->withSuccess('Успішно!');
    }
    public function destroy(Tariff $tariff)
    {
        $tariff->delete();
        return redirect()->back()->withSuccess('Успішно!');
    }
    public function removeTariffServices($id)
    {
        $tariffService = новий TariffService();
        $tariffService->deleteService($id);
        return redirect()->back()->withSuccess('Успішно видалено');
    }
    public function getTariffId(int $id)
    {
        $modelTariff = новий Tariff();
        $modelService = new Service();
        $modelUnit = new ServiceUnit();
        $tariff = $modelTariff->getTariffIds($id);
        $data['tariffServices'] = $tariff->tariffService;
        $data['services'] = $modelService->getData();
        $data['serviceUnit'] = $modelUnit->getData();
        return json_encode($data);
    }
}
```

Hotel.php

```
<?php
namespace App\Models;
```

```
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
class Hotel extends Model
{
    use HasFactory;
    public $timestamps = false;
    public function flours()
    {
        return $this->hasMany(HotelFlour::class,'Hotels_id');
    }
    public function sections()
    {
        return $this->hasMany(HotelSection::class,'Hotels_id');
    }
    public function getHotel()
    {
        return $this->with('flours', 'sections')->get();
    }
    public function getHotelIds(int $id)
    {
        return $this->with('flours', 'sections')->find($id);
    }
    public function create(array $data)
    {
        $this->Hotel_name = $data['name'];
        $this->Hotel_address = $data['addressHotel'];
        $this->image1 = isset($data['HotelImages'][0]) ? $ data['HotelImages'][0] : null;
        $this->image2 = isset($data['HotelImages'][1]) ? $ data['HotelImages'][1] : null;
        $this->image3 = isset($data['HotelImages'][2]) ? $ data['HotelImages'][2] : null;
        $this->image4 = isset($data['HotelImages'][3]) ? $ data['HotelImages'][3] : null;
        $this->image5 = isset($data['HotelImages'][4]) ? $ data['HotelImages'][4] : null;
        $this->save();
     return $this->id;
    }
    public function updates(array $data, Hotel $Hotel)
    {
        $Hotel->Hotel_name = $data['name'];
```

```
            $Hotel->Hotel_address = $data['addressHotel'];
            $Hotel->image1 = isset($data['HotelImages'][0]) ? $data['HotelImages'][0] : $data['pathHotelImage'][0];
            $Hotel->image2 = isset($data['HotelImages'][1]) ? $data['HotelImages'][1] : $data['pathHotelImage'][1];
            $Hotel->image3 = isset($data['HotelImages'][2]) ? $data['HotelImages'][2] : $data['pathHotelImage'][2];
            $Hotel->image4 = isset($data['HotelImages'][3]) ? $data['HotelImages'][3] : $data['pathHotelImage'][3];
            $Hotel->image5 = isset($data['HotelImages'][4]) ? $data['HotelImages'][4] : $data['pathHotelImage'][4];
        $Hotel->update();
            return $Hotel->id;
        }
    }
```

Number.php

```
<?php

namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
class Number extends Model
{
    use HasFactory;
    public $timestamps = false;

    public function flours()
    {
        return $this->belongsTo(HouseFlour::class,'house_flour_id');
    }
    public function tariffs()
    {
        return $this->belongsTo(Tariff::class,'tariff_id');
    }

    public function users()
    {
        return $this->belongsTo(User::class,'user_id');
```

```
}
public function sections()
{
   return $this->belongsTo(HouseSection::class,'house_sections_id');
}
public function accounts()
{
   return $this->hasOne(PersonalAccounts::class,'Number_id');
}
public function houses()
{
   return $this->belongsTo(House::class,'houses_id');
}
public function getNumber()
{
   return $this->with('flours', 'sections', 'tariffs', 'houses', 'users', 'accounts')->get();
}
public function getNumberIds(int $id)
{
   return $this->with('flours', 'sections', 'tariffs', 'houses', 'users', 'accounts')->find($id);
}

public function creates(array $data)
{
   $this->number = $data['numberNumber'];
   $this->square = $data['squareNumber'];
   $this->houses_id = $data['house'];
   $this->house_sections_id = $data['section'];
   $this->house_flour_id = $data['flours'];
   $this->user_id = $data['owner'];
   $this->tariff_id = $data['tariff'];

   $Number = $this->save();

   return $this->id;
}
public function updates(array $data, $Number)
```

```
    {
        $Number->number = $data['numberNumber'];
        $Number->square = $data['squareNumber'];
        $Number->houses_id = $data['house'];
        $Number->house_sections_id = $data['section'];
        $Number->house_flour_id = $data['flours'];
        $Number->user_id = $data['owner'];
        $Number->tariff_id = $data['tariff'];

        $apartament = $Number->update();

        return $Number->id;
    }
}
```

Додаток Г. Графічна частина

ГРАФІЧНА ЧАСТИНА

РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ УПРАВЛІННЯ ГОТЕЛЕМ

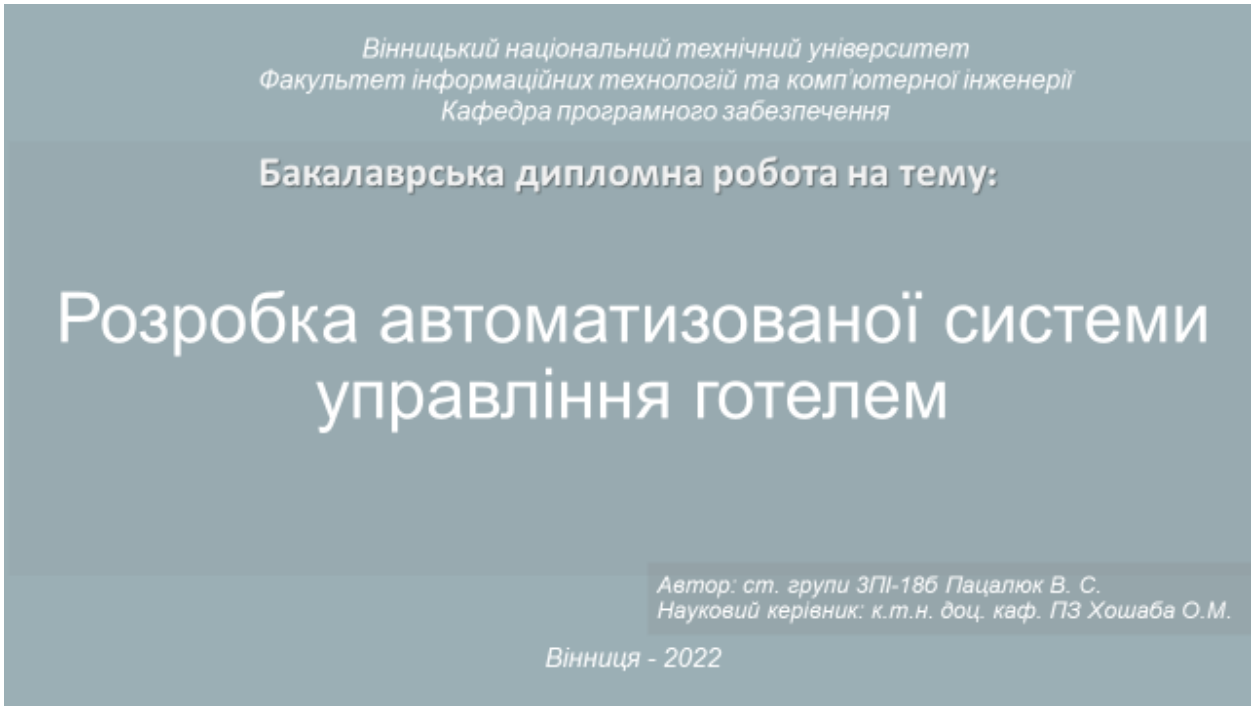
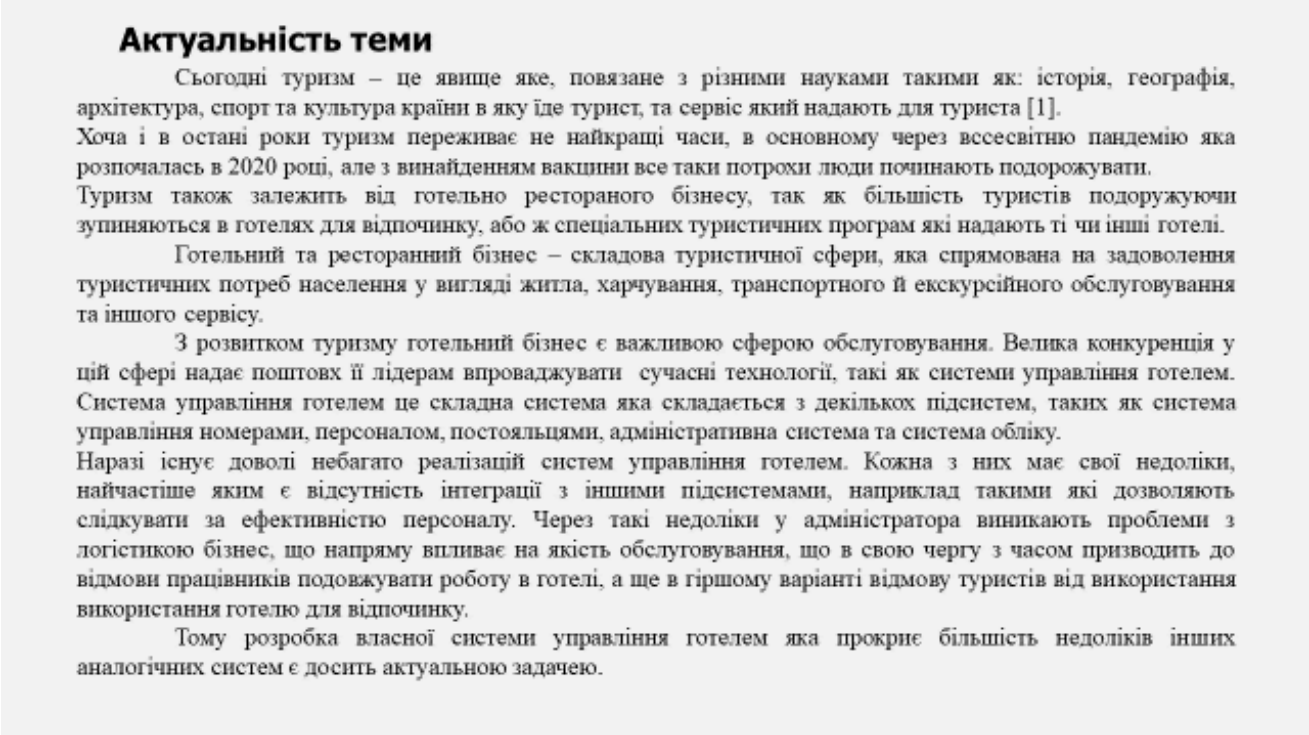


Рисунок Г.1 – Титульний слайд



Актуальність теми

Сьогодні туризм – це явище яке, повязане з різними науками такими як: історія, географія, архітектура, спорт та культура країни в яку їде турист, та сервіс який надають для туриста [1].

Хоча і в останні роки туризм переживає не найкращі часи, в основному через всесвітню пандемію яка розпочалась в 2020 році, але з винайденням вакцини все таки потрохи люди починають подорожувати.

Туризм також залежить від готельно ресторaного бізнесу, так як більшість туристів подоружуючи зупиняються в готелях для відпочинку, або ж спеціальних туристичних програм які надають ті чи інші готелі.

Готельний та ресторанний бізнес – складова туристичної сфери, яка спрямована на задоволення туристичних потреб населення у вигляді житла, харчування, транспортного й екскурсійного обслуговування та іншого сервісу.

З розвитком туризму готельний бізнес є важливою сферою обслуговування. Велика конкуренція у цій сфері надає поштовх її лідерам впроваджувати сучасні технології, такі як системи управління готелем. Система управління готелем це складна система яка складається з декількох підсистем, таких як система управління номерами, персоналом, постояльцями, адміністративна система та система обліку.

Наразі існує доволі небагато реалізацій систем управління готелем. Кожна з них має свої недоліки, найчастіше яким є відсутність інтеграції з іншими підсистемами, наприклад такими які дозволяють слідкувати за ефективністю персоналу. Через такі недоліки у адміністратора виникають проблеми з логістикою бізнес, що напряму впливає на якість обслуговування, що в свою чергу з часом призводить до відмови працівників подовжувати роботу в готелі, а ще в гіршому варіанті відмову туристів від використання використання готелю для відпочинку.

Тому розробка власної системи управління готелем яка прокриє більшість недоліків інших аналогічних систем є досить актуальною задачею.

Рисунок Г.2 – Актуальність теми

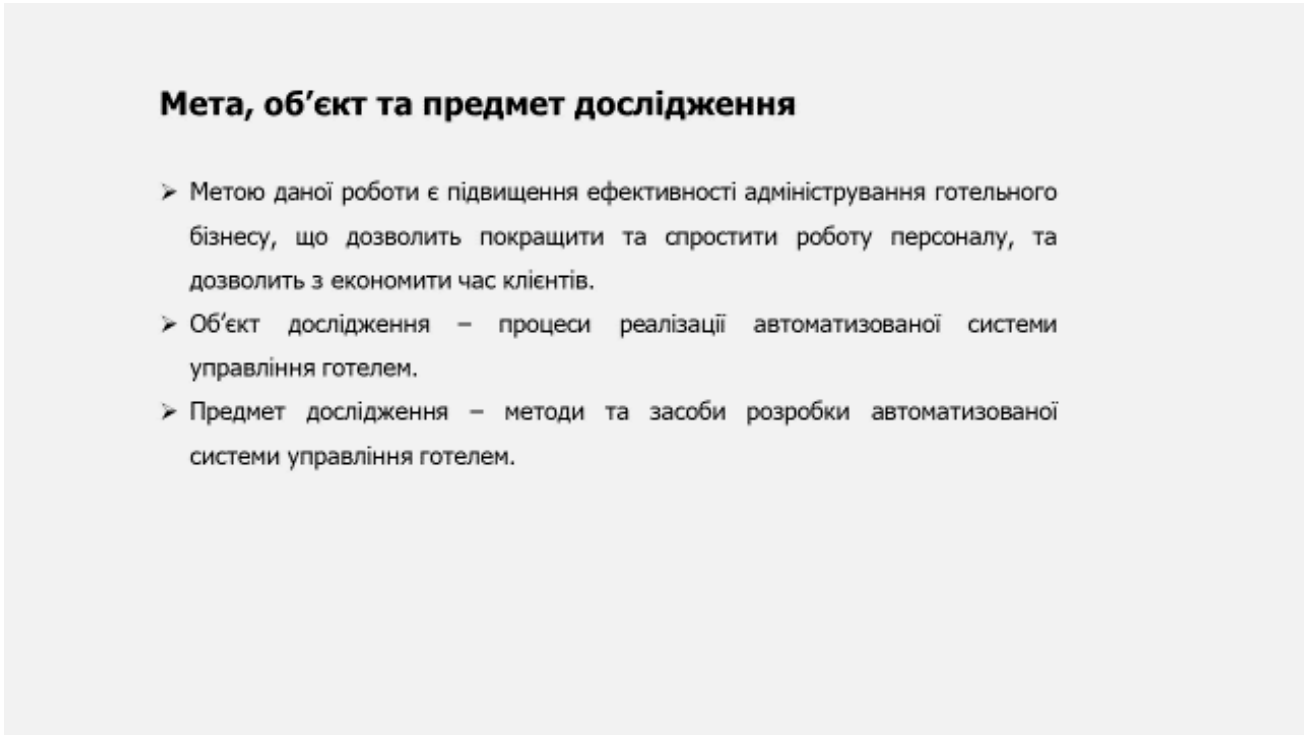


Рисунок Г.3 – Мета, об’єкт та предмет дослідження

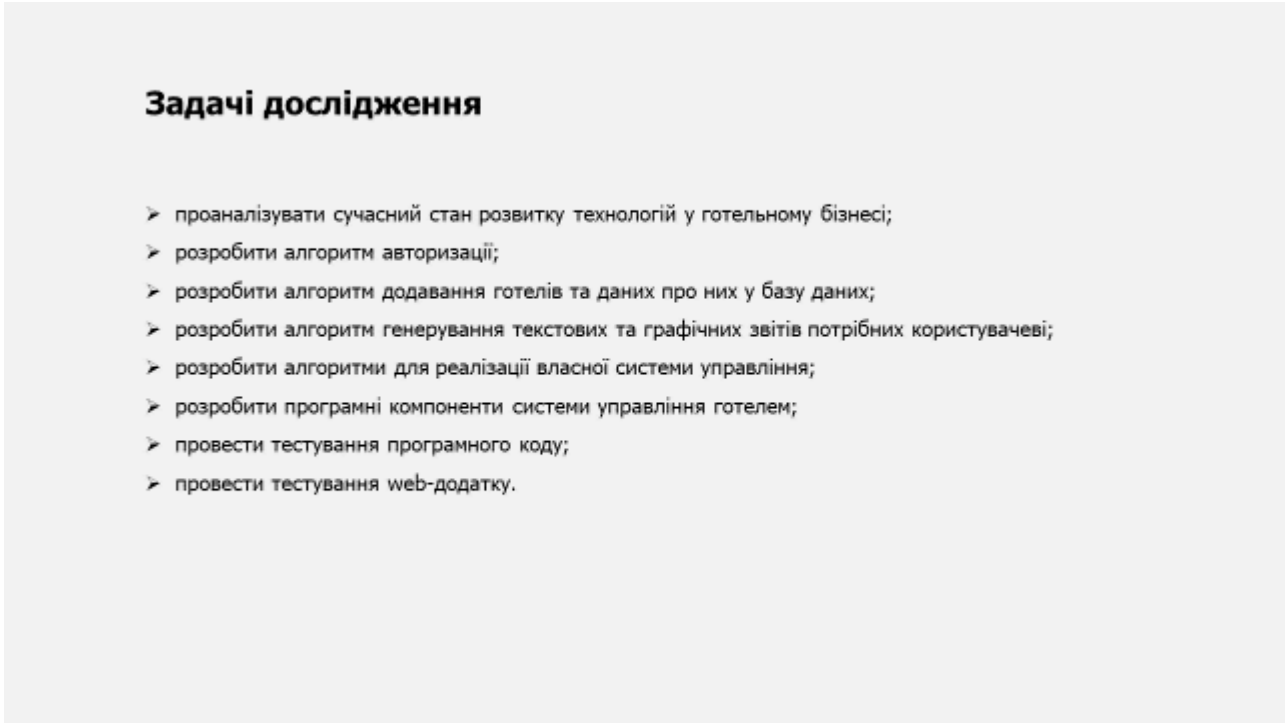


Рисунок Г.4 – Задачі дослідження

Практична цінність одержаних результатів

На основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для автоматизованої системи управління готелем

Рисунок Г.5 – Практична цінність одержаних результатів

Порівняльний аналіз аналогів

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

Рисунок Г.6 – Порівняльний аналіз аналогів

Рисунок Г.7 – Блок-схема авторизації

Рисунок Г.8 – Блок-схема алгоритму додавання нових постояльців в систему

Рисунок Г.9 – Блок-схема алгоритму генерування текстових та графічних звітів

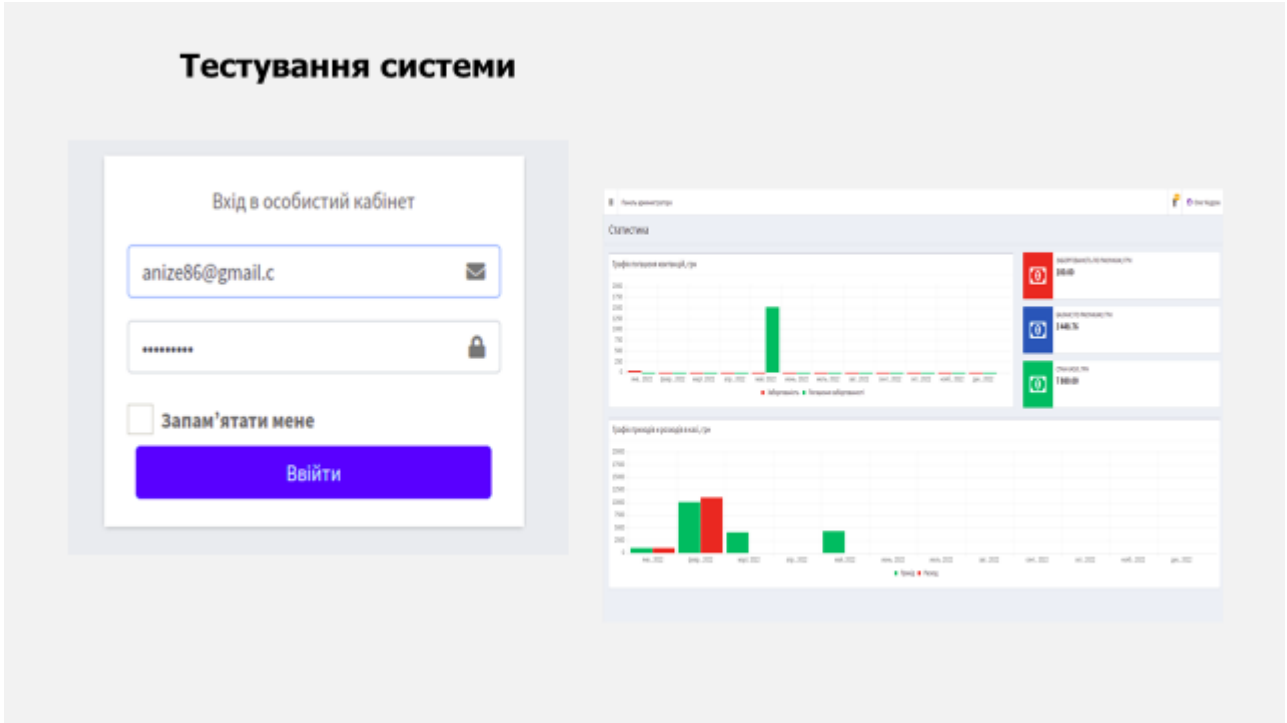


Рисунок Г.10 – Тестування програми №1

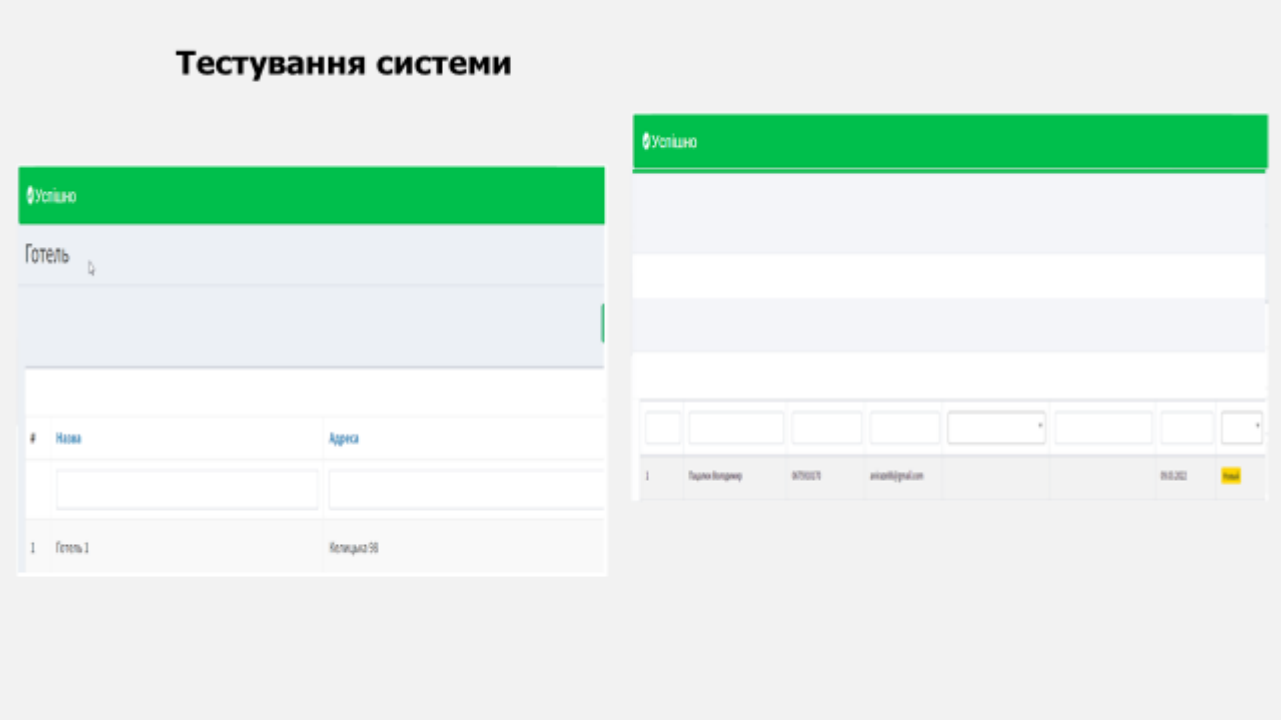


Рисунок Г.11 – Тестування програми №2

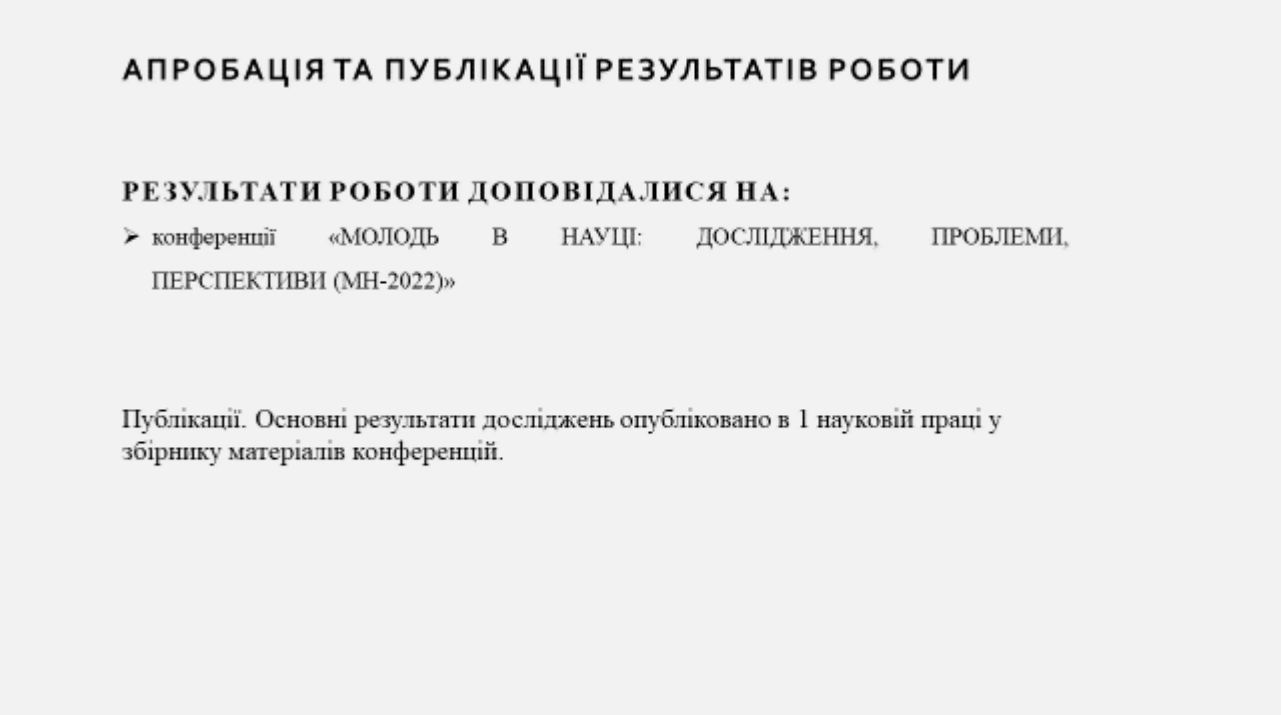


Рисунок Г.12 – Апробація та публікації результатів роботи

Рисунок Г.13 – Фінальний слайд