

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

**«Вебзастосунок для адміністрування роботи
сервісного центру з ремонту побутової техніки»**

Виконав студент групи ЗІПЗ-22-1
ШЕВЕЛЬ Володимир Святославович

Керівник роботи:
ТАЛАВЕР Олег Володимирович

Рецензент:
ГРОМСЬКИЙ Олександр Олександрович

Житомир - 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ШЕВЕЛЬ Володимир Святославович**

Керівник роботи: **ТАЛАВЕР Олег Володимирович**

Тема роботи: **«Вебзастосунок для адміністрування роботи сервісного центру з ремонту побутової техніки»**,

затверджена Наказом закладу вищої освіти від «27» лютого 2026 р., №93/с

Вихідні дані для роботи: Об'єктом дослідження є процеси автоматизації управління роботою сервісного центру з ремонту побутової техніки.

Предметом дослідження виступає вебзастосунок, що реалізує клієнт-серверну архітектуру на основі патерну MVC для автоматизації зазначених процесів.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Талавер О. В.	27.04.2026р.	27.04.2026р.
2	Талавер О. В.	27.04.2026р.	27.04.2026р.
3	Талавер О. В.	27.04.2026р.	01.06.2026р.

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра складається з трьох розділів, містить 16 таблиць, 23 рисунки, 30 джерел - загалом 84 сторінки.

Метою кваліфікаційної роботи є розробка та реалізація вебзастосунку для адміністрування роботи сервісного центру з ремонту побутової техніки, що забезпечує автоматизацію ключових бізнес-процесів підприємства з використанням сучасних технологій вебпрограмування.

Об'єктом дослідження є процеси автоматизації управління роботою сервісного центру з ремонту побутової техніки. Предметом дослідження є методи, моделі та програмні засоби розробки вебзастосунку для автоматизації ключових бізнес-процесів сервісного центру з використанням клієнт-серверної архітектури та патерну MVC.

У результаті виконання роботи розроблено вебзастосунок, що включає адміністративну панель для обліку ремонтів та замовлень запасних частин, модуль SMS-інформування клієнтів через SmsClub API, механізм OTP-верифікації видачі техніки, систему довідників із ієрархічною структурою (вид, бренд, модель), рольову модель доступу з трьома рівнями (адміністратор, менеджер, майстер), модуль друку квитанцій та публічний вебсайт сервісного центру.

Розроблений вебзастосунок може бути впроваджений у роботу сервісних центрів з ремонту побутової техніки як типове рішення з можливістю адаптації під конкретні потреби підприємства.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, СЕРВІСНИЙ ЦЕНТР, РЕМОНТ ПОБУТОВОЇ ТЕХНІКИ, PHP, MYSQL

					ІПЗ.КР.Б – 121 – 26 – ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата							
Розроб.		Шевель В. С.			Вебзастосунок для адміністрування роботи сервісного центру з ремонту побутової техніки Пояснювальна записка			Літ.	Арк.	Аркушів	
Керівник		Талавер О.В.								3	111
Рецензент		Громський О.О.						Житомирська політехніка, група ЗІПЗ-22-І			
Зав. каф.		Вакалюк Т.А									

ABSTRACT

The explanatory note of the bachelor's qualification thesis consists of three chapters, contains 16 tables, 23 figures, 30 references - a total of 84 pages.

The purpose of the qualification thesis is the development and implementation of a web application for administrating the operations of a household appliance repair service center, providing automation of key business processes using modern web programming technologies.

The object of the research is the processes of automation of service center management for household appliance repair. The subject of the research is a web application implementing client-server architecture based on the MVC pattern.

As a result of the work, a web application has been developed that includes an administrative panel for repair and spare parts order management, an SMS notification module via SmsClub API, an OTP verification mechanism for device handover, a hierarchical reference system (device type , brand, model), a role-based access control with three levels (administrator, manager, technician), a receipt printing module, and a public website for the service center.

The developed web application can be deployed in household appliance repair service centers as a standard solution with the possibility of adaptation to the specific needs of an enterprise.

KEYWORDS: WEB APPLICATION, SERVICE CENTER, HOUSEHOLD APPLIANCE REPAIR, PHP, MYSQL

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AJAX - Asynchronous JavaScript and XML - технологія асинхронної взаємодії клієнта та сервера

API - Application Programming Interface - програмний інтерфейс застосунку

CSRF - Cross-Site Request Forgery - міжсайтова підробка запитів

CSS - Cascading Style Sheets - каскадні таблиці стилів

CRUD - Create, Read, Update, Delete - базові операції з даними

HTML - HyperText Markup Language - мова розмітки гіпертексту

HTTP - HyperText Transfer Protocol - протокол передачі гіпертексту

JSON - JavaScript Object Notation - формат обміну даними

MVC - Model-View-Controller - архітектурний патерн розподілу відповідальності

OTP - One-Time Password - одноразовий пароль

PHP - PHP: Hypertext Preprocessor - серверна мова програмування

PSR - PHP Standards Recommendations - стандарти PHP-розробки

REST - Representational State Transfer - архітектурний стиль API

SaaS - Software as a Service - програмне забезпечення як послуга

SMS - Short Message Service - служба коротких повідомлень

SQL - Structured Query Language - мова структурованих запитів

СУБД - Система управління базами даних

UI - User Interface - інтерфейс користувача

UML - Unified Modeling Language - уніфікована мова моделювання

URL - Uniform Resource Locator - уніфікований визначник ресурсу

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	5
ВСТУП.	8
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ....	11
1.1 Постановка задачі.....	11
1.2 Аналіз предметної області.....	14
1.3 Аналіз існуючих рішень	17
1.4 Вибір архітектури.....	23
1.5 Вибір інструментальних засобів та технологій	26
Висновки до розділу 1	31
РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	33
2.1 Варіанти використання системи	33
2.2 Проектування бази даних системи	37
2.3 Проектування та реалізація алгоритмів роботи системи	42
2.4 Реалізація програмного комплексу	51
Висновки до розділу 2	59
РОЗДІЛ 3. ІНТЕРФЕЙС, ТЕСТУВАННЯ ТА РОЗГОРТАННЯ СИСТЕМИ..	61
3.1 Порядок встановлення та налаштування системи	61
3.2 Опис інтерфейсу користувача	64
3.3 Тестування програмного комплексу	74
Висновки до розділу 3	78
ВИСНОВКИ.....	80
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	82

ДОДАТКИ	85
Додаток А	86
Додаток Б.....	97
Додаток В	101
Додаток Г.....	105

ВСТУП

Сфера технічного обслуговування та ремонту побутової техніки є невід'ємною складовою сучасної сервісної економіки України. Зростання кількості та різноманіття побутових електроприладів у домогосподарствах, ускладнення конструкції сучасних пристроїв, а також підвищення вимог споживачів до якості та оперативності обслуговування зумовлюють необхідність впровадження ефективних інформаційних систем у діяльність сервісних підприємств. Традиційні методи ведення обліку - паперові журнали та електронні таблиці - не забезпечують належного рівня автоматизації бізнес-процесів, що негативно позначається на продуктивності роботи, якості клієнтського сервісу та конкурентоспроможності підприємства.

Існуючі комерційні рішення для автоматизації сервісних центрів, зокрема RO App, Orderry та BytePhase, функціонують за моделлю SaaS і передбачають щомісячну абонентську плату, зберігання даних на зовнішніх серверах та обмежену можливість адаптації під специфічні потреби конкретного підприємства. Для невеликого сервісного центру, що працює з персональними даними клієнтів та партнерських організацій, актуальною є розробка власного вебзастосунку, який поєднуватиме адміністративний модуль із публічним сайтом підприємства, забезпечуватиме автоматичне SMS-інформування клієнтів, верифікацію видачі пристроїв та розгортання на власному сервері без залежності від сторонніх провайдерів.

Зазначене зумовлює актуальність теми випускної кваліфікаційної роботи бакалавра та доцільність розробки спеціалізованого вебзастосунку для адміністрування роботи сервісного центру з ремонту побутової техніки.

Метою кваліфікаційної роботи є розробка та реалізація вебзастосунку для адміністрування роботи сервісного центру з ремонту побутової техніки, що забезпечує автоматизацію ключових бізнес-процесів підприємства - обліку ремонтів і замовлень запчастин, інформування бази клієнтів, верифікації видачі

техніки та формування звітності - з використанням сучасних технологій програмування.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- провести аналіз предметної галузі сервісного обслуговування побутової техніки та визначити перелік бізнес-процесів, що підлягають автоматизації;
- виконати порівняльний аналіз існуючих програмних рішень для управління сервісними центрами та обґрунтувати доцільність власної розробки;
- сформулювати функціональні та нефункціональні вимоги до розроблюваної системи;
- обґрунтувати вибір архітектурного підходу та технологічного стеку розробки;
- спроектувати структуру бази даних та алгоритми роботи основних модулів системи;
- реалізувати програмний комплекс вебзастосунку відповідно до визначених вимог;
- виконати тестування реалізованого рішення та описати порядок його розгортання і використання.

Об'єктом дослідження є процеси автоматизації управління роботою сервісного центру з ремонту побутової техніки.

Предметом дослідження є методи, моделі та програмні засоби розробки вебзастосунку для автоматизації ключових бізнес-процесів сервісного центру з використанням клієнт-серверної архітектури та патерну MVC.

Методи дослідження. У роботі використано методи системного аналізу для дослідження предметної галузі та визначення вимог, об'єктно-орієнтоване проектування для побудови архітектури системи, методи реляційного моделювання для проектування бази даних, а також методи автоматизованого тестування (unit-тестування засобами PHPUnit та статичний аналіз засобами PHPStan) для перевірки коректності роботи програмного комплексу.

Практичне значення отриманих результатів полягає у створенні повнофункціонального вебзастосунку, готового до впровадження у реальне підприємство сфери сервісного обслуговування побутової техніки. Розроблений програмний комплекс забезпечує автоматизацію обліку ремонтів та замовлень, SMS-інформування клієнтів, верифікацію видачі техніки за допомогою одноразового коду, формування та друк квитанцій, ведення довідників і статистики, а також надає публічний вебсайт підприємства. Система може бути використана як типове рішення для аналогічних сервісних центрів із мінімальною адаптацією під конкретні потреби підприємства.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Постановка задачі

Сфера технічного обслуговування та ремонту побутової техніки є невід'ємною складовою сучасної сервісної економіки. За даними галузевих досліджень, обсяг ринку сервісного обслуговування побутових електроприладів щороку зростає, що зумовлено кількома чинниками: збільшенням кількості та різноманіття побутової техніки в домогосподарствах, ускладненням конструкції сучасних приладів, що робить самостійний ремонт практично неможливим, а також зростанням усвідомлення споживачами доцільності професійного обслуговування порівняно з придбанням нового пристрою [1].

До впровадження інформаційної системи облік заявок на ремонт у сервісному центрі вівся вручну або у таблицях Microsoft Excel. Такий підхід має низку суттєвих недоліків, що критично впливають на якість роботи підприємства. По-перше, високий ризик втрати або пошкодження даних: паперовий журнал може бути загублений, а файл Excel - випадково видалений або пошкоджений без можливості відновлення. По-друге, відсутність централізованого доступу: у разі роботи кількох співробітників одночасно кожен з них оперує власною копією даних, що призводить до дублювання, конфліктів версій та неузгодженості інформації. По-третє, неможливість автоматичного сповіщення клієнтів: адміністратор змушений вручну телефонувати кожному клієнту для повідомлення про готовність ремонту, що забирає значний робочий час і збільшує ймовірність людської помилки. По-четверте, труднощі з формуванням звітності: підрахунок кількості ремонтів, середньої вартості, навантаження на майстрів потребує ручної обробки даних і є трудомістким процесом. По-п'яте, відсутність верифікації при видачі техніки: за ручного обліку неможливо гарантувати, що пристрій видано саме його власнику, що створює ризик шахрайства.

Функціональні вимоги до системи формуються на підставі виявлених бізнес-процесів сервісного центру та охоплюють чотирнадцять основних аспектів.

Система повинна забезпечувати авторизацію адміністраторів та майстрів із захистом сесій та механізмом CSRF-токенів, щоб запобігти несанкціонованому доступу до адміністративних функцій та захистити від міжсайтової підробки запитів.

Система повинна реалізовувати повний цикл обліку ремонтів, що включає прийом заявки із зазначенням клієнта, виду техніки, бренду, моделі та опису несправності, призначення відповідального майстра, фіксацію вартості роботи та запчастин, а також послідовну зміну статусів відповідно до прогресу виконання ремонту.

Система повинна забезпечувати окремий модуль обліку замовлень запасних частин та комплектуючих з аналогічною до ремонтів статусною моделлю, але із власною ієрархією довідників пристроїв. Система повинна надсилати автоматичні SMS-повідомлення клієнту при зміні статусу ремонту або замовлення через інтеграцію з українським SMS-шлюзом SmsClub, причому текст повідомлення має формуватися на основі налаштовуваних шаблонів, що зберігаються в базі даних та можуть бути змінені адміністратором без втручання у програмний код.

Система повинна реалізовувати процедуру верифікації видачі пристрою клієнту за допомогою одноразового чотиризначного числового коду (One-Time Password, OTP), що надсилається на мобільний телефон клієнта та діє протягом обмеженого часу - 10 хвилин. Такий механізм забезпечує додатковий рівень безпеки та виключає можливість видачі техніки стороннім особам. Система повинна вести централізовану базу клієнтів із можливістю пошуку за іменем або номером телефону, перегляду та редагування контактних даних, а також перегляду повної історії ремонтів та замовлень конкретного клієнта.

Система повинна підтримувати облік партнерських організацій із зберіганням назви, контактних даних, реквізитів та можливістю прив'язки ремонтів і замовлень до конкретного партнера.

Система повинна забезпечувати формування та друк квитанцій про прийом і

видачу техніки з можливістю налаштування тексту квитанцій.

Система повинна підтримувати ієрархічний довідник пристроїв з трирівневою структурою (вид техніки - бренд - модель) та можливістю додавання нових позицій безпосередньо під час оформлення ремонту без необхідності попереднього переходу до розділу довідників.

Система повинна надавати статистичну інформацію про роботу кожного майстра, відображати на головній панелі (dashboard) кількість ремонтів та замовлень у різних статусах, а також надсилати автоматичні нагадування клієнтам про необхідність забрати готову техніку за допомогою cron-задачі, що виконується за розкладом. Нарешті, система повинна містити публічний вебсайт сервісного центру із відображенням переліку послуг, партнерських організацій та контактної інформації.

Нефункціональні вимоги до системи визначають якісні параметри, яким має відповідати вебзастосунок. Нижче наведено зведену таблицю нефункціональних вимог (табл. 1.1).

Таблиця 1.1
Нефункціональні вимоги до вебзастосунку

Категорія	Вимога
Безпека	Захист від CSRF-атак, хешування паролів (bcrypt, cost 12), захищене зберігання сесій, збереження секретних ключів у змінних середовища
Продуктивність	Час відповіді на AJAX-запити не більше 3 с, коректна обробка паралельних запитів від кількох співробітників без втрати цілісності даних
Зручність	Адаптивний інтерфейс, пошук, сортування та пагінація табличних даних (DataTables), мінімум дій для виконання типових операцій
Надійність	Автоматичне резервне копіювання БД за розкладом (cron), логування помилок у файл, розмежування середовищ (local / production)
Супроводжуваність	MVC-архітектура, PSR-4 автозавантаження, сервісний шар, unit-тести (PHPUnit), статичний аналіз (PHPStan)
Розгортання	Docker-контейнеризація, міграції БД, конфігурація через .env-файл, відтворюване середовище

Наведені вимоги слугують основою для проектування архітектури системи та вибору інструментальних засобів розробки.

1.2 Аналіз предметної області

Типовий сервісний центр з ремонту побутової техніки обслуговує широкий спектр пристроїв: пральні та посудомийні машини, холодильники, мікрохвильові печі, кавомашини, телевізори, DVD-програвачі, медіацентри, фени, прасування, пирососи та інші електроприлади. Центр може працювати як із фізичними особами - кінцевими споживачами, так і з юридичними особами - партнерськими організаціями, що направляють техніку на гарантійний або платний ремонт. Наявність двох категорій клієнтів суттєво ускладнює облік, оскільки для партнерських організацій необхідно вести окрему документацію, відстежувати умови гарантійного обслуговування та формувати специфічну звітність.

Основними бізнес-процесами сервісного центру, що потребують автоматизації, є наступні. Процес прийому техніки на ремонт включає реєстрацію звернення клієнта, ідентифікацію пристрою за видом, брендом та моделлю, фіксацію опису несправності зі слів клієнта, призначення відповідального майстра та формування квитанції про прийом. Процес діагностики та ремонту передбачає визначення фактичної несправності, оцінку вартості ремонту та запасних частин, узгодження вартості з клієнтом, виконання ремонтних робіт та фіксацію результату. Процес управління статусами полягає у послідовному переведенні замовлення між станами відповідно до прогресу виконання: від прийому в роботу через етап діагностики та ремонту до стану готовності та остаточної видачі клієнту. Процес видачі техніки включає інформування клієнта про готовність, верифікацію особи отримувача, передачу пристрою та оформлення квитанції про видачу. Процес замовлення запасних частин охоплює формування заявки на потрібні деталі, відстеження статусу доставки, облік вартості та прив'язку до конкретного ремонту або клієнта. Паралельно із зазначеними основними процесами сервісний центр здійснює ведення бази клієнтів та партнерських організацій з фіксацією контактних

даних і історії звернень, управління довідниками видів техніки, брендів і моделей, формування прайс-листів послуг, збір статистики по виконаних ремонтах у розрізі майстрів для оцінки продуктивності, а також ведення публічної інформаційної сторінки з переліком послуг, партнерів та контактною інформацією.

Автоматизація управлінських процесів у сфері сервісного обслуговування є одним із ключових чинників підвищення ефективності малого та середнього бізнесу в умовах зростаючої конкуренції на ринку послуг. Сучасний клієнт очікує оперативного інформування про стан свого замовлення, прозорого ціноутворення та мінімального часу очікування - вимог, які принципово неможливо задовольнити без відповідного програмного забезпечення [1; 28].

Особливо важливим для сервісного центру є те, що вебзастосунок дозволяє кільком співробітникам - адміністратору, майстрам, менеджеру - одночасно працювати з однією базою даних без ризику конфліктів версій. Адміністратор може оформлювати нову заявку, поки майстер оновлює статус поточного ремонту з іншого робочого місця. При цьому всі зміни одразу відображаються для всіх користувачів, що забезпечує актуальність інформації в реальному часі.

Впровадження розробленого вебзастосунку дозволить сервісному центру досягти кількох стратегічних цілей. По-перше, суттєво скоротити час на оформлення заявки та видачу техніки завдяки автоматизації рутинних операцій та попередньому заповненню полів. По-друге, усунути ризик втрати або дублювання даних шляхом переходу від ручного обліку до централізованої бази даних із механізмом резервного копіювання. По-третє, підвищити рівень клієнтського сервісу завдяки автоматичним SMS-нотифікаціям, що інформують клієнта про стан його замовлення без необхідності ручного телефонування. По-четверте, забезпечити безпеку процесу видачі техніки через механізм ОТР-верифікації, що виключає можливість отримання пристрою сторонньою особою. По-п'яте, надати керівництву підприємства інструменти моніторингу та контролю роботи співробітників завдяки статистиці по майстрах та журналу змін.

Окремо слід розглянути інформаційні потоки, що циркулюють у діяльності типового сервісного центру. При прийомі пристрою формується первинний документ - квитанція, що фіксує контактні дані клієнта, опис пристрою та заявлену несправність. Далі інформація про ремонт передається майстру, який оновлює статус виконання та вносить дані про використані запасні частини та вартість робіт. Після завершення ремонту інформація повертається до адміністратора для підготовки до видачі. Паралельно клієнт має бути проінформований про зміну статусу свого замовлення. Таким чином, інформаційний потік має циклічний характер та включає щонайменше чотирьох учасників: клієнта, адміністратора, майстра та керівника підприємства.

Важливим аспектом предметної області є необхідність ведення ієрархічних довідників пристроїв. Сервісний центр працює з різноманітною побутовою технікою, яку доцільно класифікувати за трирівневою ієрархією: вид пристрою (телевізор, пральна машина, мікрохвильова піч тощо), бренд виробника та конкретна модель. Така класифікація дозволяє формувати статистику ремонтів за типами пристроїв, виявляти найбільш поширені несправності та оптимізувати закупівлю запасних частин [28]. Окремо ведеться аналогічна класифікація для замовлень запасних частин, оскільки номенклатура комплектуючих може відрізнятися від номенклатури пристроїв, що приймаються на ремонт.

Додатковою особливістю досліджуваного підприємства є робота з партнерськими організаціями - юридичними особами, які направляють техніку на гарантійний або платний ремонт. У цьому випадку замовником виступає не фізична особа, а підприємство-партнер, що зумовлює специфічні вимоги до обліку: необхідно фіксувати як контактні дані партнерської організації, так і дані кінцевого користувача пристрою. Це впливає на алгоритм визначення адресата SMS-повідомлень та процедуру верифікації при видачі пристрою.

Ще одним суттєвим аспектом є життєвий цикл ремонту, що визначає послідовність станів замовлення від моменту прийому до видачі клієнту. Типовий

життєвий цикл включає стани: «В роботі» (прийнято та передано майстру), «Завершено» (ремонт виконано, очікує видачі) та «Видано» (пристрій повернено клієнту). Перехід між станами супроводжується бізнес-правилами: при переході у стан «Завершено» фіксується фінальна вартість та автоматично надсилається SMS-повідомлення клієнту; при переході у стан «Видано» виконується процедура OTP-верифікації особи отримувача. Окрім основних станів, система має відстежувати прострочені ремонти - замовлення, що перебувають у стані «В роботі» довше визначеного терміну, та готові замовлення, що очікують видачі понад встановлений час зберігання.

1.3 Аналіз існуючих рішень

На сучасному ринку програмного забезпечення для сервісних центрів існує низка готових рішень різного рівня складності та вартості, серед яких RepairShopr [27], RO App, Orderry, BytePhase та інші [26]. Для обґрунтування доцільності власної розробки необхідно провести детальний порівняльний аналіз найбільш поширених систем. Для цього було обрано три рішення, що найбільш повно відповідають профілю досліджуваного підприємства: RO App (раніше RemOnline), Orderry та BytePhase.

RO App (раніше RemOnline) є хмарною платформою українського походження, спеціально розробленою для автоматизації сервісних центрів, ремонтних майстерень та невеликих виробничих підприємств. Платформа пропонує широкий набір функцій: облік замовлень і ремонтів від прийому до видачі, ведення бази клієнтів із повною історією взаємодій, управління складом запасних частин із відстеженням залишків, фінансовий облік з розрахунком собівартості та прибутковості кожного замовлення, аналітика та звітність у вигляді графіків і таблиць. Система доступна як вебзастосунок, а також має мобільні додатки для Android та iOS, що дозволяє майстрам та адміністраторам працювати з будь-якого пристрою [26].

На Рис. 1.1 наведено вигляд сторінки додатку CRM RO App, що містить

основну інформацію по компанії

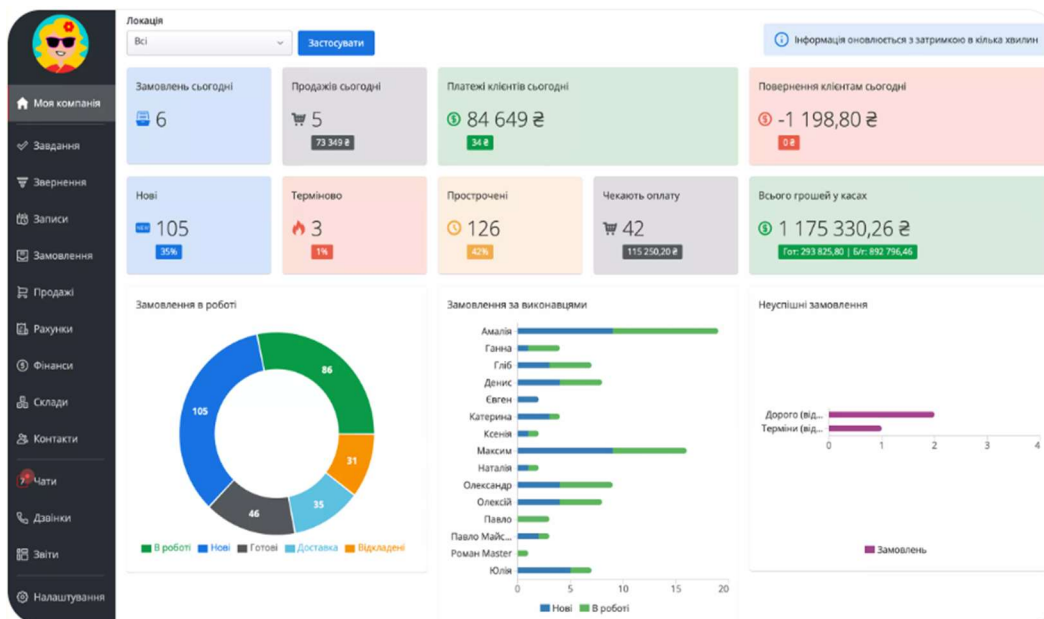


Рисунок 1.1 - Дашборд CRM RO App

Серед ключових переваг RO App варто зазначити наявність понад 200 готових інтеграцій із зовнішніми сервісами - месенджерами (Telegram, Viber), системами оплати, бухгалтерськими програмами, а також IP-телефонією. Платформа підтримує налаштування канбан-дошки для візуалізації процесу ремонту, двофакторну автентифікацію для захисту облікових записів, друк етикеток та документів. Інтерфейс системи повністю локалізований українською мовою.

Водночас RO App має ряд суттєвих обмежень для невеликого сервісного центру. Насамперед, це модель ціноутворення за підпискою: вартість використання залежить від обраного тарифного плану, кількості підключених співробітників та обсягу використаних інтеграцій. Для підприємства з 3-5 працівниками щомісячна абонентська плата є відчутною статтею витрат, що перевищує вартість утримання власного серверного рішення. Крім того, всі дані зберігаються виключно на хмарних серверах провайдера, що обмежує контроль підприємства над власною інформацією та створює залежність від стабільності роботи та цінової політики третьої сторони.

Orderry є міжнародною хмарною платформою для управління сервісними

центрами, що позиціонується як комплексне рішення для ремонтних майстерень будь-якого профілю - від ремонту електроніки та побутової техніки до автосервісів та виїзних бригад. Система об'єднує в одному інтерфейсі модулі управління ремонтними заявками, складом, CRM, розкладом робіт, нарахуванням заробітної плати та аналітикою. Orderry має мобільні додатки для техніків, що дозволяють обробляти заявки, оновлювати статуси та приймати оплату безпосередньо на виїзді [26].

На Рис. 1.2, де наведено вигляд сторінки додатку Orderry, що містить основну інформацію по компанії, можна помітити, що інтерфейс аналогічний з RO App.

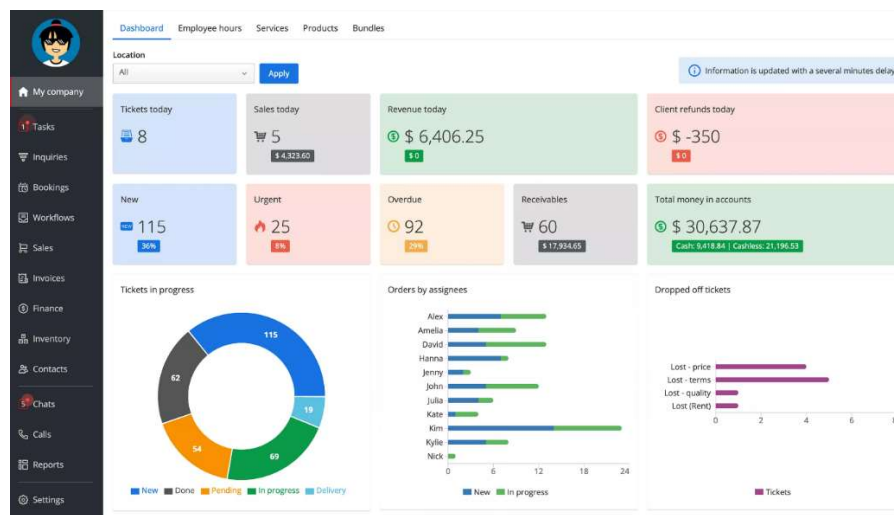


Рисунок 1.2 - Дашборд додатку Orderry

Серед функціональних переваг Orderry слід виділити автоматичне визначення пристрою за IMEI-номером, що прискорює процес прийому техніки, підтримку штрихкодів для роботи зі складом, інтеграцію з месенджерами (WhatsApp, Facebook Messenger, Instagram Direct, Telegram), можливість збору відгуків та автоматизації Google-рецензій, а також розширену систему звітності з понад 20 типами звітів. Платформа підтримує роботу з кількома локаціями одночасно, що робить її привабливою для мережевих сервісних центрів.

Основними обмеженнями Orderry для досліджуваного підприємства є платна модель використання з тарифами, що зростають залежно від кількості користувачів та активованих функцій, а також орієнтованість на англomовний ринок - хоча

платформа підтримує кілька мов, основна документація та служба підтримки орієнтовані на англомовних користувачів. Крім того, надмірна кількість функцій, орієнтованих на великі мережі (нарахування зарплати, управління виїзними бригадами, інтеграція з QuickBooks), є зайвою для невеликого сервісного центру та ускладнює інтерфейс.

BytePhase є хмарною CRM-системою, спеціалізованою на сервісних центрах з ремонту електроніки та побутової техніки. Платформа надає кожному клієнту власний піддомен, що забезпечує певний рівень індивідуалізації. На Рис. 1.3 наведено вигляд сторінки BytePhase, де описані основні особливості системи.

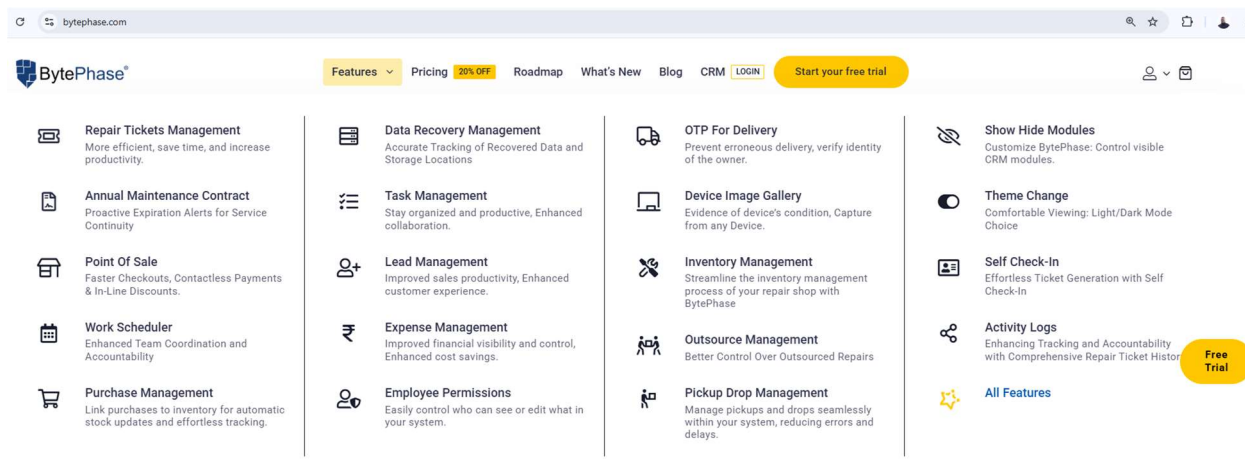


Рисунок 1.3 - Особливості BytePhase

Серед ключових можливостей BytePhase - відстеження заявок у реальному часі із детальним журналом змін, управління складом запасних частин із прив'язкою до конкретних ремонтів, OTP-верифікація при видачі пристроїв клієнту, комунікація через WhatsApp та SMS, базова аналітика продуктивності та виставлення рахунків. Система орієнтована переважно на міжнародний ринок і на даний момент використовується підприємствами у понад 32 країнах [26].

Суттєвими обмеженнями BytePhase для застосування у вітчизняних умовах є англомовний інтерфейс без локалізації українською мовою, платна підписка від 20 доларів США на місяць із обмеженням кількості користувачів та функцій у базових тарифах, а також відсутність специфічних для українського ринку інтеграцій - зокрема, з вітчизняними SMS-шлюзами та месенджерами. Крім того, як і у випадку

з іншими хмарними рішеннями, дані зберігаються на зовнішніх серверах, що обмежує контроль підприємства над інформацією.

Для наочного порівняння ключових характеристик розглянутих систем та вебзастосунку власної розробки нижче наведено зведену порівняльну таблицю.

Таблиця 1.2

Порівняльний аналіз програмних продуктів для управління сервісним центром

Критерій	RO App	Orderry	BytePhase	Власна розробка
Мова інтерфейсу	Українська	Англ. + інші	Англійська	Українська
Модель розгортання	Хмара (SaaS)	Хмара (SaaS)	Хмара (SaaS)	Власний сервер
Вартість	Від 9 €/міс.	Від \$39/міс.	Від \$20/міс.	Безкоштовно
Облік ремонтів	Так	Так	Так	Так
Облік замовлень	Так	Так	Так	Так
SMS-нотифікації	Так	Так	Так	Так
OTP-верифікація	Ні	Ні	Так	Так
Облік партнерів	Частково	Частково	Ні	Так
Публічний сайт	Ні	Ні	Ні	Так
SMS-шаблони	Так	Так	Обмежено	Так
Відкритий код	Ні	Ні	Ні	Так
Docker	Ні	Ні	Ні	Так
Мобільний додаток	Так	Так	Ні	Ні
Управління складом	Так	Так	Так	Ні
Резервне копіювання	Так (хмара)	Так (хмара)	Так (хмара)	Так (cron)

Як свідчать результати порівняльного аналізу, представлені в (табл. 1.2), кожне з розглянутих комерційних рішень має свої переваги, але водночас жодне не задовольняє повною мірою специфічних потреб досліджуваного сервісного центру.

По-перше, усі три аналоги функціонують виключно за моделлю SaaS (Software as a Service), тобто дані підприємства зберігаються на серверах стороннього провайдера. Для невеликого сервісного центру, що працює з персональними даними клієнтів - іменами та номерами телефонів - це створює

додаткові ризики з точки зору конфіденційності та відповідності вимогам Закону України «Про захист персональних даних» [2]. Власне розгортання на серверах підприємства повністю усуває цю залежність.

По-друге, комерційні системи передбачають щомісячну абонентську плату, розмір якої зростає з кількістю підключених співробітників і активованих модулів. Для сервісного центру з 3-5 працівниками сумарні річні витрати на підписку можуть перевищити вартість оренди виділеного серверу, на якому буде розгорнуто власне рішення. Власна розробка не потребує ліцензійних платежів і може експлуатуватися необмежено довго.

По-третє, жодна з розглянутих систем не поєднує в собі адміністративний модуль управління ремонтами з публічним вебсайтом підприємства. Комерційні рішення пропонують виключно внутрішню панель управління, тоді як для представлення сервісного центру в мережі Інтернет підприємство змушене окремо замовляти або розробляти вебсайт. Розроблюваний вебзастосунок об'єднує обидві складові в одному програмному комплексі.

По-четверте, специфічна для досліджуваного підприємства потреба у веденні обліку партнерських організацій - юридичних осіб, що направляють техніку на гарантійний ремонт, - реалізована лише частково у RO App та Orderry (у вигляді загальних полів для контрагентів) і повністю відсутня у BytePhase. Власна розробка дозволяє реалізувати цю функцію з урахуванням конкретних бізнес-правил підприємства.

Разом з тим, слід зазначити, що комерційні рішення мають певні переваги, яких позбавлена власна розробка: наявність мобільних додатків (RO App, Orderry), розвинений модуль управління складом, інтеграція з бухгалтерськими системами та можливість масштабування на мережу локацій. Ці функції, однак, не є критичними для поточних потреб досліджуваного підприємства і можуть бути реалізовані у наступних версіях системи.

1.4 Вибір архітектури

При виборі форми реалізації програмного продукту - настільний додаток, мобільний застосунок або вебзастосунок - було проведено аналіз переваг та недоліків кожного підходу. Настільний (десктопний) додаток потребує окремого встановлення на кожне робоче місце, що ускладнює оновлення та підтримку, а також обмежує доступ до системи конкретним комп'ютером. Мобільний застосунок, хоча й забезпечує мобільність, є витратним у розробці через необхідність підтримки кількох платформ (Android, iOS) та обмежений у можливостях роботи з великими масивами табличних даних. Вебзастосунок є оптимальним компромісом: він не потребує встановлення на клієнтські пристрої, забезпечує централізоване зберігання даних на сервері, дозволяє працювати з будь-якого пристрою з веббраузером - комп'ютера, планшета чи смартфона - та має низьку вартість розгортання і підтримки [4].

Перед початком розробки вебзастосунку необхідно визначити архітектурний підхід, який забезпечить чітке розмежування відповідальності між компонентами системи, спрощуватиме підтримку коду та дозволить масштабувати функціонал без значного рефакторингу. У межах цього проєкту обрано архітектурний патерн MVC (Model-View-Controller) - один із найпоширеніших підходів до побудови вебзастосунків, що передбачає розподіл логіки на три незалежні рівні [11].

Рівень Model (Модель) відповідає за роботу з даними: виконання запитів до бази даних, обробку бізнес-логіки та валідацію вхідних даних. Кожна сутність предметної галузі матиме відповідний клас моделі, що інкапсулює операції створення, читання, оновлення та видалення записів у базі даних. Рівень View (Представлення) відповідає за формування та відображення HTML-сторінок, що передаються браузеру користувача. Шаблони представлень реалізуються з мінімальною логікою - лише вивід даних, підготовлених контролером. Рівень Controller (Контролер) є посередником між моделлю та представленням: приймає HTTP-запити від користувача, звертається до відповідних методів моделі, формує

дані та передає їх у шаблон [12].

На Рис. 1.4 представлено загальну схему архітектурного патерну MVC, що ілюструє взаємодію між компонентами системи та потік обробки запитів від користувача.

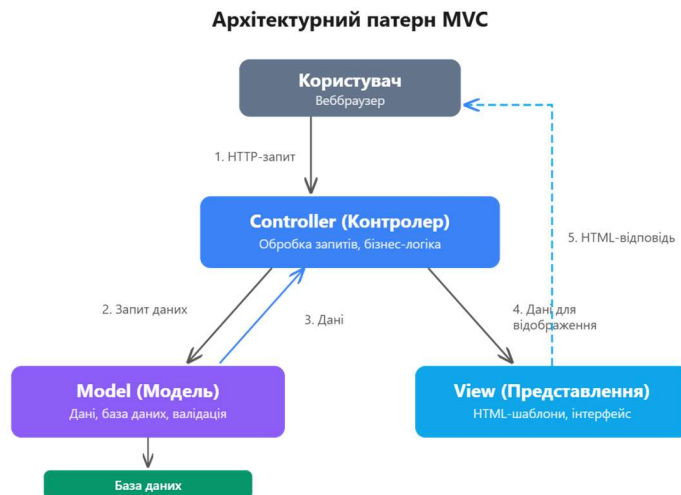


Рисунок. 1.4 - Загальна схема архітектурного патерну MVC

Як видно зі схеми, користувач надсилає HTTP-запит через веббраузер (крок 1), який обробляється контролером. Контролер звертається до моделі для отримання необхідних даних (кроки 2-3), після чого передає їх у представлення для формування HTML-відповіді (кроки 4-5). Така організація забезпечує чітке розмежування обов'язків: модель не залежить від способу відображення даних, представлення не містить бізнес-логіки, а контролер координує їх взаємодію.

На додаток до класичного MVC у системі планується реалізувати сервісний шар (Service Layer) - окремі класи, що інкапсулюватимуть складну бізнес-логіку, яка не належить ані до моделі, ані до контролера. Сервісний шар забезпечуватиме взаємодію із зовнішніми сервісами, реалізацію логіки зміни статусів замовлень із відправкою повідомлень, а також вирішення допоміжних задач, що потребують координації кількох моделей. Такий підхід забезпечує принцип єдиної відповідальності (Single Responsibility Principle) та суттєво спрощує unit-тестування окремих компонентів системи [12; 13].

Перед вибором архітектурного патерну було розглянуто альтернативні підходи до організації програмного коду вебзастосунку. Патерн MVVM (Model-View-ViewModel), широко застосовуваний у фреймворках для побудови односторінкових застосунків (SPA), передбачає реактивне двостороннє зв'язування даних між рівнем відображення та рівнем бізнес-логіки. Для серверного PHP-застосунку з традиційною генерацією HTML-сторінок на стороні сервера цей патерн є надлишковим, оскільки відсутня потреба у реактивному оновленні інтерфейсу без перезавантаження сторінки. Патерн MVP (Model-View-Presenter) подібний до MVC, але передбачає, що Presenter повністю контролює View та обробляє всі події користувача, що ускладнює архітектуру без суттєвих переваг для вебзастосунку із серверними HTML-шаблонами [13].

Мікросервісна архітектура, за якої кожен функціональний модуль реалізується як окремий незалежний сервіс зі власною базою даних та API, є виправданою для великих розподілених систем із командами з десятків розробників. Однак для застосунку, що обслуговує одне підприємство із трьома-п'ятьма одночасними користувачами, мікросервісний підхід створює невиправдану операційну складність: необхідність оркестрації контейнерів, міжсервісної комунікації, розподілених транзакцій та окремого моніторингу кожного сервісу [12].

Таким чином, патерн MVC з розширеним сервісним шаром є оптимальним вибором для даного проєкту, оскільки забезпечує достатній рівень розділення відповідальності, простоту розуміння та підтримки коду, а також можливість поступового масштабування системи без необхідності перебудови архітектури. Сервісний шар доповнює класичний MVC саме в тих місцях, де бізнес-логіка виходить за межі відповідальності окремої моделі або контролера, зберігаючи при цьому загальну простоту архітектури.

Взаємодію між клієнтською та серверною частинами планується реалізувати за допомогою двох механізмів. Для стандартних операцій (завантаження сторінок, форми створення та редагування) використовуватиметься класична модель запит-

відповідь HTTP. Для операцій, що потребують оновлення частини сторінки без її повного перезавантаження - зокрема, динамічне завантаження залежних списків, верифікація одноразових кодів, оновлення статусів - застосовуватиметься технологія AJAX із поверненням даних у форматі JSON [15].

Для зберігання та обробки даних у системі планується використовувати систему управління реляційними базами даних.

1.5 Вибір інструментальних засобів та технологій

Для реалізації серверної частини вебзастосунку обрано мову програмування PHP версії 8.2, що є однією з найпоширеніших мов для розробки серверних вебзастосунків у світі. Вибір мови програмування PHP зумовлений кількома ключовими чинниками [4; 5].

По-перше, PHP має надзвичайно широкую екосистему бібліотек, інструментів та документації, що суттєво прискорює розробку. По-друге, PHP нативно підтримує роботу з HTTP-запитами, сесіями, куками та формами - усім тим, що є базовою потребою будь-якого вебзастосунку. По-третє, PHP 8.x привніс низку сучасних можливостей: строга типізація, іменовані аргументи, перелічення (enums), readonly-властивості, що наближає мову до рівня зрілих об'єктно-орієнтованих мов. По-четверте, PHP легко розгортається на будь-якому хостингу або сервері, що важливо для невеликого підприємства.

У вебзастосунку буде використано сучасні можливості PHP 8.2: простори імен (namespaces) за стандартом PSR-4 для організації класів, автозавантаження класів через Composer, строга типізація аргументів та значень, що повертаються методами, а також використання вбудованих функцій для роботи з рядками, масивами та датами. Для виконання PHP-коду на сервері забезпечить веб-сервер Apache 2.4 з підключеним модулем mod_rewrite, який дозволяє реалізувати чисті URL-адреси (SEO-friendly URLs) шляхом перенаправлення всіх запитів через єдину точку входу [6].

Продовження табл. 1.3

Паралельний доступ	Так	Так	Обмежено
Підтримка PHP (нативна)	MySQLi / PDO	PDO	PDO / SQLite3
Зовнішні ключі	Так (InnoDB)	Так	Обмежено
Підтримка Docker	Офіційний образ	Офіційний образ	Вбудована
Придатність для вебзастосунків	Висока	Висока	Низька

Клієнтську частину вебзастосунку планується реалізувати з використанням стандартних технологій веброзробки: HTML5, CSS3 та JavaScript. Для прискорення розробки адаптивного інтерфейсу обрано CSS-фреймворк Bootstrap, що надає готову сітку макету, компоненти форм, таблиць, кнопок, модальних вікон та повідомлень. Bootstrap забезпечує коректне відображення інтерфейсу на екранах різного розміру - від настільних комп'ютерів до планшетів і смартфонів - без написання додаткового CSS-коду [23].

Для роботи з табличними даними в адміністративній панелі планується використати бібліотеку DataTables - jQuery-плагін, що перетворює звичайні HTML-таблиці на інтерактивні компоненти з вбудованим пошуком, сортуванням за будь-яким стовпцем, пагінацією та можливістю вибору кількості записів на сторінці. Оскільки основний масив інформації в системі - ремонти, замовлення, клієнти - представлятиметься у вигляді великих таблиць, використання DataTables є обґрунтованим рішенням, що суттєво підвищить зручність роботи з системою [22].

Для реалізації динамічної взаємодії між клієнтом і сервером планується використати бібліотеку jQuery та механізм AJAX. Зокрема, AJAX-запити застосовуватимуться для динамічного завантаження залежних списків (наприклад, перелік брендів залежно від обраної категорії пристрою, перелік моделей залежно від обраного бренду), а також для перевірки одноразових кодів верифікації без перезавантаження сторінки. Такий підхід мінімізує час відповіді інтерфейсу та покращує користувацький досвід [23].

Для реалізації функції автоматичного SMS-інформування клієнтів обрано українську службу розсилки повідомлень SmsClub. Вибір даного провайдера зумовлений наявністю зручного REST API, підтримкою кирилиці та налаштування імені відправника, а також орієнтованістю на український ринок [24; 25]. Взаємодію з API SmsClub планується реалізувати у вигляді окремого сервісу, що інкапсулюватиме логіку формування HTTP-запитів, обробки відповідей та логування відправлених повідомлень. Токен доступу до API зберігатиметься у змінних середовища, що відповідає вимогам безпеки щодо зберігання секретних ключів [20].

Для забезпечення відтворюваності середовища розробки та спрощення розгортання на продуктивному сервері обрано технологію контейнеризації Docker разом із інструментом Docker Compose [17; 18]. Контейнеризація вирішує одну з ключових проблем веброзробки - залежність від конкретної конфігурації операційної системи та встановленого програмного забезпечення: завдяки Docker система однаково запускається на будь-якому комп'ютері або сервері, де встановлений Docker Engine, незалежно від операційної системи.

Забезпечення якості програмного коду є невід'ємною складовою сучасного процесу розробки програмного забезпечення. У проєкті планується використати два інструменти контролю якості: PHPUnit для автоматизованого тестування та PHPStan для статичного аналізу коду [14; 15].

PHPUnit є стандартом галузі для unit-тестування PHP-застосунків. У межах проєкту тестами планується покрити насамперед сервісний шар системи - компоненти, що реалізовуватимуть найбільш критичну бізнес-логіку: взаємодію із зовнішнім SMS-шлюзом, логіку визначення клієнта, алгоритми зміни статусів замовлень та генерації одноразових кодів верифікації. Автоматизоване тестування дозволить перевіряти коректність роботи цих компонентів після кожної зміни коду, що суттєво знижує ризик регресій [15; 16].

PHPStan є інструментом статичного аналізу коду, що виявляє потенційні

помилки ще до запуску програми: звернення до неіснуючих методів або властивостей, невідповідність типів, недосяжний код та інші проблеми. Статичний аналіз виконується без запуску коду і доповнює тестування, охоплюючи ті аспекти якості, які важко перевірити тестами. Запуск обох інструментів планується автоматизувати через скрипти менеджера залежностей Composer, що дозволить виконувати повну перевірку якості коду однією командою перед кожним комітом [14].

Таблиця 1.4

Зведена характеристика технологічного стеку вебзастосунку

Категорія	Технологія/Версія	Призначення
Серверна мова	PHP 8.2	Серверна логіка, MVC, сервіси
Веб-сервер	Apache 2.4	Обробка HTTP-запитів, mod_rewrite
СУБД	MySQL 8.0	Зберігання та обробка даних
Управління залежностями	Composer 2	Автозавантаження класів, бібліотеки
CSS-фреймворк	Bootstrap	Адаптивний інтерфейс, компоненти UI
Таблиці даних	DataTables	Пошук, сортування, пагінація таблиць
AJAX / JS	jQuery	Динамічна взаємодія клієнт-сервер
SMS-шлюз	SmsClub API	Нотифікації клієнтів, OTP-верифікація
Контейнеризація	Docker + Docker Compose	Розгортання, ізоляція середовища
Unit-тестування	PHPUnit 10.5	Автоматизоване тестування сервісів
Статичний аналіз	PHPStan 1.10	Виявлення помилок типізації
Архітектурний патерн	MVC + Service Layer	Розподіл відповідальності компонентів

Наведений у (табл. 1,4) технологічний стек є збалансованим поєднанням перевірених часом інструментів (PHP, MySQL, Apache) та сучасних практик розробки (Docker-контейнеризація, автоматичне тестування, статичний аналіз). Кожен з обраних компонентів має відкритий вихідний код або доступний безкоштовно, що виключає необхідність придбання комерційних ліцензій для розгортання системи на підприємстві.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи виконано аналіз предметної області сервісного обслуговування побутової техніки, досліджено існуючі програмні рішення, обґрунтовано вибір архітектурного підходу та визначено технологічний стек розробки вебзастосунку.

У підрозділі 1.1 сформульовано постановку задачі, сформовано чотирнадцять функціональних вимог до системи (авторизація, облік ремонтів і замовлень, SMS-нотифікації, OTP-верифікація видачі, ведення бази клієнтів і партнерів, довідники пристроїв, друк квитанцій, статистика, публічний сайт) та шість груп нефункціональних вимог (безпека, продуктивність, зручність, надійність, супроводжуваність, розгортання).

У підрозділі 1.2 проведено аналіз предметної області: досліджено типову організацію роботи сервісного центру, ідентифіковано дев'ять ключових бізнес-процесів, що потребують автоматизації (прийом техніки, діагностика та ремонт, управління статусами, видача, замовлення запчастин, ведення бази клієнтів і партнерів, управління довідниками, формування звітності, публічний сайт), та обґрунтовано необхідність впровадження спеціалізованого програмного забезпечення замість ручного обліку.

У підрозділі 1.3 виконано порівняльний аналіз трьох існуючих комерційних рішень - RO App, Orderry та BytePhase за шістнадцятьма критеріями. Аналіз показав, що жодна з існуючих систем не задовольняє одночасно всіх специфічних потреб досліджуваного підприємства: розгортання на власному сервері без абонентської плати, поєднання адміністративного модуля з публічним вебсайтом, обліку партнерських організацій та OTP-верифікації видачі, що обґрунтовує доцільність власної розробки.

У підрозділі 1.4 обґрунтовано вибір архітектурного патерну MVC (Model-View-Controller) з розширеним сервісним шаром як основу організації програмного коду, а також вибір форми реалізації у вигляді вебзастосунку якщо порівнювати

настільним та мобільним додатком.

У підрозділі 1.5 визначено та обґрунтовано технологічний стек розробки: мова програмування PHP 8.2, СУБД MySQL 8.0, веб-сервер Apache 2.4, CSS-фреймворк Bootstrap, бібліотека DataTables, SMS-шлюз SmsClub API, Docker для контейнеризації, PHPUnit для автоматизованого тестування та PHPStan для статичного аналізу коду. Усі обрані компоненти мають відкритий вихідний код або доступні безкоштовно, що виключає необхідність придбання комерційних ліцензій.

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання системи

На основі аналізу предметної галузі та визначених у першому розділі вимог виконано специфікацію варіантів використання (use cases) вебзастосунку. Для формалізації взаємодії користувачів із системою було ідентифіковано акторів, визначено функціональні та нефункціональні вимоги від кожного типу користувачів, а також побудовано діаграму варіантів використання у нотації UML 2.0 [29; 30].

У системі виділено три основні категорії акторів, кожна з яких має чітко визначений набір повноважень та обмежень. Адміністратор є привілейованим користувачем, що має повний доступ до всіх функцій системи, включаючи управління обліковими записами інших користувачів, налаштування параметрів системи, SMS-шаблонів, вмісту публічного сайту та довідників пристроїв. Менеджер має доступ до основних функцій управління ремонтами, замовленнями, управління персоналом, клієнтами та довідниками, але не може змінювати налаштування сайту та редагувати облікові записи адміністраторів. Майстер є обмеженим користувачем, який бачить лише призначені йому ремонти та може змінювати їх статус, але не має доступу до замовлень, довідників, налаштувань та управління персоналом. Відвідувач публічного сайту може переглядати інформацію про сервісний центр, його послуги та контакти, але не має доступу до адміністративної панелі.

На основі визначених акторів було специфіковано сімнадцять варіантів використання системи, що відображають типові сценарії взаємодії користувачів із програмним комплексом. Варіанти використання охоплюють повний цикл обслуговування клієнта: від авторизації в системі та реєстрації ремонту до верифікації видачі та перегляду статистики. У (табл. 2.1) наведено перелік основних варіантів використання із зазначенням акторів та пріоритетів.

Таблиця 2.1
Варіанти використання системи

ID	Варіант використання	Актори	Пріоритет
UC-1	Авторизація в адміністративній панелі	Адміністратор, Менеджер, Майстер	Високий
UC-2	Реєстрація нового ремонту	Адміністратор, Менеджер	Високий
UC-3	Редагування даних ремонту	Адміністратор, Менеджер	Високий
UC-4	Зміна статусу ремонту на «Завершено»	Адміністратор, Менеджер, Майстер	Високий
UC-5	Верифікація видачі пристрою (OTP)	Адміністратор, Менеджер	Високий
UC-6	Реєстрація замовлення запчастин	Адміністратор, Менеджер	Високий
C-7	Редагування та зміна статусу замовлення	Адміністратор, Менеджер	Високий
UC-8	Перегляд призначених ремонтів	Майстер	Високий
UC-9	Управління клієнтами (CRUD)	Адміністратор, Менеджер	Середній
UC-10	Управління партнерами (CRUD)	Адміністратор, Менеджер	Середній
UC-11	Управління довідниками (пристрої, бренди, моделі)	Адміністратор, Менеджер	Середній
UC-12	Друк квитанцій прийому та видачі	Адміністратор, Менеджер	Середній
UC-13	Перегляд статистики та dashboard	Адміністратор, Менеджер	Середній
UC-14	Управління персоналом	Адміністратор, Менеджер*	Середній
UC-15	Налаштування SMS-шаблонів та текстів квитанцій	Адміністратор, Менеджер	Низький
UC-16	Налаштування публічного сайту	Адміністратор	Низький
UC-17	Перегляд публічного сайту	Відвідувач	Середній

*Примітка: * - менеджер може редагувати облікові записи менеджерів та майстрів, але не адміністраторів. Менеджер також не може деактивувати або видалити власний обліковий запис.*

На Рис. 2.1 наведено діаграму варіантів використання системи в нотації UML, що візуалізує взаємодію акторів із функціональними модулями вебзастосунку.

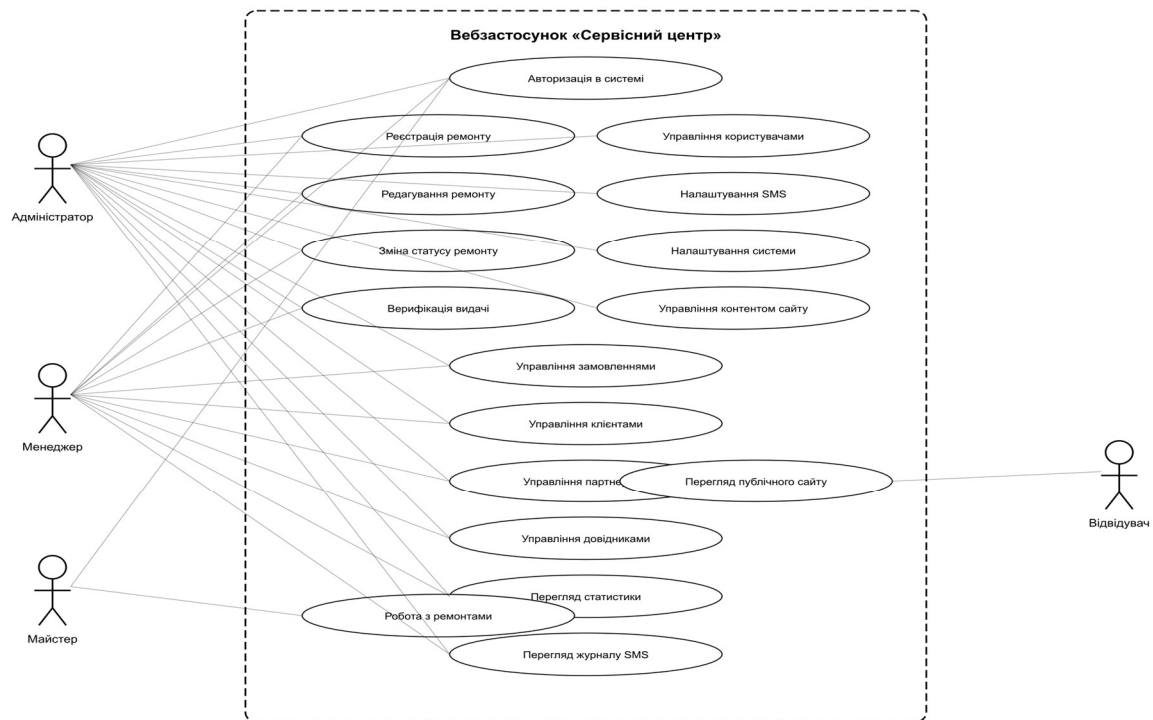


Рисунок 2.1 - Діаграма варіантів використання системи (UML Use Case Diagram)

Як видно з діаграми, адміністратор має доступ до всіх варіантів використання, менеджер - до більшості функцій крім системних налаштувань, майстер - лише до роботи з призначеними ремонтами, а відвідувач взаємодіє виключно з публічним сайтом.

Розглянемо детальніше найбільш критичні варіанти використання. Варіант UC-2 «Реєстрація нового ремонту» є центральним бізнес-процесом системи. Адміністратор або менеджер заповнює форму, що включає вибір або створення клієнта (з можливістю прив'язки до партнерської організації), вибір категорії пристрою з ієрархічного довідника (вид - бренд - модель), при цьому якщо потрібний бренд або модель відсутні у довіднику, їх можна створити безпосередньо

з форми ремонту без переходу до окремого інтерфейсу. Далі фіксуються опис несправності, серійний номер, дата придбання, комплектація, вартісні показники та призначається відповідальний майстер. Після збереження система автоматично присвоює ремонту статус «В роботі» [29].

Варіант UC-5 «Верифікація видачі пристрою» реалізує двоетапний процес безпечної видачі. На першому етапі адміністратор або менеджер ініціює процес видачі, після чого система генерує одноразовий чотиризначний числовий код, зберігає його у таблиці `verification_codes` та надсилає код на мобільний телефон клієнта через `SmsClub API`. На другому етапі клієнт називає отриманий код, користувач вводить його у форму верифікації, система перевіряє введений код із збереженням та, у разі відповідності і за умови, що код не прострочений (10 хвилин), переводить ремонт у статус «Видано» із фіксацією дати та часу видачі [30].

У (табл. 2.2) наведено детальний опис варіанту використання UC-2 «Реєстрація нового ремонту», що є центральним бізнес-процесом системи.

Таблиця 2.2
Опис варіанту використання UC-2

Ідентифікатор	UC-2
Назва	Реєстрація нового ремонту
Актори	Адміністратор, Менеджер
Передумова	Користувач авторизований в системі та має роль Адміністратор або Менеджер
Основний потік	1. Користувач відкриває форму створення ремонту 2. Обирає або створює клієнта (за номером телефону) 3. За потреби обирає партнерську організацію 4. Обирає категорію пристрою з ієрархічного довідника (вид → бренд → модель) 5. Вводить опис несправності, серійний номер, комплектацію 6. Вказує вартісні показники та призначає майстра 7. Натискає кнопку «Зберегти»
Альтернативний потік	4а. Якщо бренд/модель відсутні у довіднику, користувач вводить нову назву (прапорець <code>create_new_device/brand/model</code>). Система перевіряє дублікати через <code>TRIM/LOWER</code> і створює запис лише за відсутності
Постумова	Ремонт створений у БД зі статусом «В роботі» (<code>status_id = 1</code>)
Виняткові ситуації	Невалідний CSRF-токен → повернення на форму з повідомленням про помилку

У (табл. 2.3) наведено детальний опис варіанту використання UC-5.

Таблиця 2.3
Опис варіанту використання UC-5

Ідентифікатор	UC-5
Назва	Верифікація видачі пристрою
Актори	Адміністратор, Менеджер
Передумова	Ремонт має статус «Готово» (status_id = 2)
Основний потік	1. Користувач натискає «Видати» на сторінці ремонту 2. Система генерує 4-значний OTP-код (rand 1000-9999) 3. Код зберігається у таблиці verification_codes (термін дії 10 хв) 4. Попередні невикористані коди деактивуються 5. SMS з кодом надсилається клієнту через SmsClub API 6. Клієнт повідомляє код, користувач вводить його у форму 7. Система перевіряє код, термін дії та прапорець is_used
Альтернативний потік	7а. Код невірний або прострочений → повідомлення про помилку, можливість повторної генерації
Постумова	Ремонт переведений у статус «Видано» (status_id = 3), зафіксовано date_issue
Виняткові ситуації	SMS-шлюз недоступний → код генерується, але SMS не доставлене; код можна продиктувати клієнту по телефону

Нефункціональні вимоги до системи включають шість категорій: вимоги до продуктивності (час відгуку на дії користувача не більше 2 секунд), вимоги до безпеки (захист від OWASP Top Ten 2021, хешування паролів, CSRF-захист), вимоги до надійності (автоматичне резервне копіювання, ідемпотентні міграції), вимоги до зручності використання (адаптивний інтерфейс, інтуїтивна навігація), вимоги до розгортання (контейнеризація через Docker, незалежність від ОС сервера) та вимоги до підтримки (документована структура коду, автоматизоване тестування) [3; 20].

2.2 Проєктування бази даних системи

Реляційна база даних є центральним компонентом системи зберігання даних вебзастосунку. На основі аналізу предметної галузі та визначених варіантів використання було спроєктовано структуру бази даних, що складається з 21 таблиць (включаючи таблиці, створені міграціями). Структура бази даних спроєктована з дотриманням третьої нормальної форми (3НФ), що забезпечує мінімізацію надмірності даних та підтримку цілісності через зовнішні ключі [10].

Центральними таблицями системи є repairs (ремонти) та orders (замовлення запчастин), навколо яких побудовано решту структури. Таблиця repairs зберігає повну інформацію про кожен ремонт: зовнішні ключі на клієнта (client_id), партнера (partner_id), пристрій (device_id), бренд (brand_id), модель (model_id), статус (status_id), відповідального майстра (master_id) та адміністратора, що створив запис (user_id), а також текстові поля для опису несправності, серійного номера, комплектації, нотаток про ремонт та числові поля для вартісних показників. Таблиця orders має аналогічну структуру, але із власними довідниками пристроїв (order_devices, order_brands, order_models), що дозволяє вести незалежні каталоги для ремонтів та замовлень запчастин.

Довідникові таблиці забезпечують нормалізацію даних та зручність управління класифікаторами. Таблиця clients зберігає персональні дані клієнтів: ім'я, номер телефону, адресу та додаткову інформацію, з унікальним обмеженням на номер телефону для запобігання дублювання записів. Таблиця partners містить інформацію про юридичних осіб - партнерські організації, що направляють техніку на гарантійний ремонт. Таблиця users зберігає облікові записи працівників із полями email, хешованого пароля, ролі (1 - адміністратор, 2 - менеджер, 3 - майстер) та статусу активності.

У (табл. 2.4) наведено опис основних таблиць бази даних із зазначенням їх призначення та ключових полів.

Таблиця 2.4
Основні таблиці бази даних

Таблиця БД	Призначення	Ключові поля
repairs	Зберігання інформації про ремонти	id, client_id, device_id, brand_id, model_id, status_id, master_id, defect, repair_cost
orders	Замовлення запчастин	id, client_id, order_device_id, order_brand_id, status_id
clients	Персональні дані клієнтів	id, name, phone, address, partner_id

Продовження табл. 2.4

partners	Партнерські організації	id, name, phone, address, email
users	Облікові записи працівників	id, email, password, name, role, is_active
devices	Категорії пристроїв (ремонтів)	id, title
brands	Бренди пристроїв (ремонтів)	id, name, device_id
models	Моделі пристроїв (ремонтів)	id, name, brand_id
order_devices	Категорії пристроїв (замовлення)	id, title
order_brands	Бренди пристроїв (замовлення)	id, name, order_device_id
order_models	Моделі пристроїв (замовлення)	id, name, order_brand_id
statuses	Статуси ремонтів та замовлень	id, title (В роботі, Готово, Видано)
verification_codes	ОТР-коди верифікації видачі	id, repair_id, code, expires_at, is_used
sms_logs	Журнал надісланих SMS	id, phone, message, status, repair_id, type, sms_id
settings	Конфігурація системи (ключ-значення)	id, key_name, value
services	Послуги на публічному сайті	id, title, description
prices	Прайс-лист послуг	id, service_id, title, price
site_info	Інформація на публічному сайті	id, key_name, value

Ієрархічна структура довідників пристроїв реалізована через зовнішні ключі: таблиця `brands` посилається на `devices` через поле `device_id`, а таблиця `models` посилається на `brands` через поле `brand_id`. Аналогічна ієрархія існує для замовлень: `order_brands` - `order_devices`, `order_models` - `order_brands`. Це дозволяє при виборі категорії пристрою у формі динамічно завантажувати лише відповідні бренди, а при виборі бренду - лише відповідні моделі, що реалізовано через AJAX-запити [22].

Таблиця `statuses` є спільною для ремонтів та замовлень і містить три записи: «В роботі» (`id = 1`), «Готово» (`id = 2`), «Видано» (`id = 3`). Статусна модель реалізована як послідовний перехід: при створенні запису автоматично встановлюється статус «В роботі», адміністратор або менеджер може перевести його в «Готово» (що тригерить SMS-повідомлення клієнту), а потім у «Видано» (що потребує ОТР-верифікації для ремонтів). Зворотній перехід статусів не передбачений бізнес-логікою системи.

Таблиця `verification_codes` зберігає коди верифікації, що генеруються при видачі пристроїв. Кожен запис містить зовнішній ключ на ремонт (`repair_id`), значення коду (`code`), час закінчення дії (`expires_at`) та прапорець використання (`is_used`). Код дійсний протягом 10 хвилин, після чого автоматично вважається простроченим.

Таблиця `settings` реалізує механізм «ключ-значення» для зберігання конфігураційних параметрів системи: текстів публічного сайту, SMS-шаблонів, порогу днів для нагадування та інших налаштувань, що можуть змінюватися адміністратором без модифікації програмного коду.

Для управління еволюцією схеми бази даних реалізовано систему міграцій. Кожна міграція представляє собою SQL-файл у директорії `db/migrations`, що містить інструкції `CREATE TABLE`, `ALTER TABLE` або `INSERT`. Міграції виконуються в алфавітному порядку та є ідемпотентними (використовують `IF NOT EXISTS` та `ON DUPLICATE KEY UPDATE`), що дозволяє безпечно повторювати їх. На даний момент система містить 12 файлів міграцій [10].

У (табл. 2.5) наведено детальну структуру центральної таблиці `repairs` із зазначенням типів даних, обов'язковості та призначення кожного поля.

Таблиця 2.5
Структура таблиці `repairs`

Поле	Тип	NULL	Призначення
<code>id</code>	INT	Ні	Первинний ключ (AUTO_INCREMENT)
<code>device_id</code>	INT	Ні	Зовнішній ключ на таблицю <code>devices</code>
<code>brand_id</code>	INT	Ні	Зовнішній ключ на таблицю <code>brands</code>
<code>model_id</code>	INT	Ні	Зовнішній ключ на таблицю <code>models</code>
<code>status_id</code>	INT	Ні	Зовнішній ключ на таблицю <code>statuses</code>
<code>user_id</code>	INT	Ні	Хто створив запис (FK → <code>users</code>)
<code>client_id</code>	INT	Так	Клієнт (FK → <code>clients</code>)
<code>partner_id</code>	INT	Так	Партнер (FK → <code>partners</code>)
<code>master_id</code>	INT	Так	Призначений майстер (FK → <code>users</code>)
<code>create_date</code>	DATETIME	Ні	Дата та час створення ремонту
<code>finish_date</code>	DATETIME	Так	Дата та час завершення ремонту
<code>date_issue</code>	DATETIME	Так	Дата та час видачі клієнту

Продовження табл. 2.5

defect	TEXT	Так	Опис несправності
serial_number	VARCHAR(150)	Так	Серійний номер пристрою
date_purchase	DATE	Так	Дата придбання пристрою
equipment	TEXT	Так	Комплектація при прийомі
repair_cost	DOUBLE	Ні	Попередня вартість ремонту
max_repair_cost	DOUBLE	Так	Максимальна вартість
finish_repair_cost	DOUBLE	Так	Фактична вартість ремонту
detail_cost	DOUBLE	Так	Вартість запчастин
diagnostic_cost	DOUBLE	Так	Вартість діагностики
category_repair_id	INT	Ні	Категорія ремонту
note	TEXT	Так	Внутрішні нотатки
about_repair	TEXT	Ні	Опис виконаних робіт
is_ready	INT	Ні	Прапорець готовності (0/1)

Як видно з (табл. 2.5), таблиця repairs містить 24 поля, що охоплюють повний життєвий цикл ремонту: від прийому (create_date, defect, equipment) через процес виконання (master_id, repair_cost, diagnostic_cost) до завершення (finish_date, finish_repair_cost, about_repair) та видачі (date_issue, is_ready). Поля partner_id, finish_date, date_issue та вартісні показники допускають значення NULL, оскільки заповнюються на різних етапах життєвого циклу ремонту.

На Рис. 2.2 наведено фізичну модель бази даних (ER-діаграму), що відображає структуру таблиць, типи даних полів та зв'язки між сутностями. Як видно з діаграми, центральними сутностями є repairs та orders, які пов'язані із довідниковими таблицями через зовнішні ключі. Зв'язки типу «один-до-багатьох» забезпечують референтну цілісність: видалення запису з довідникової таблиці неможливе, якщо на нього посилаються записи в таблицях ремонтів або замовлень. Ієрархічна структура devices - brands - models та її аналог для замовлень утворюють деревоподібну класифікацію пристроїв. Таблиці sms_logs та settings забезпечують підсистему SMS-нотифікацій, зберігаючи історію відправлених повідомлень та шаблони текстів для кожного статусу ремонту.

Вебзастосунок для адміністрування сервісного центру – База даних

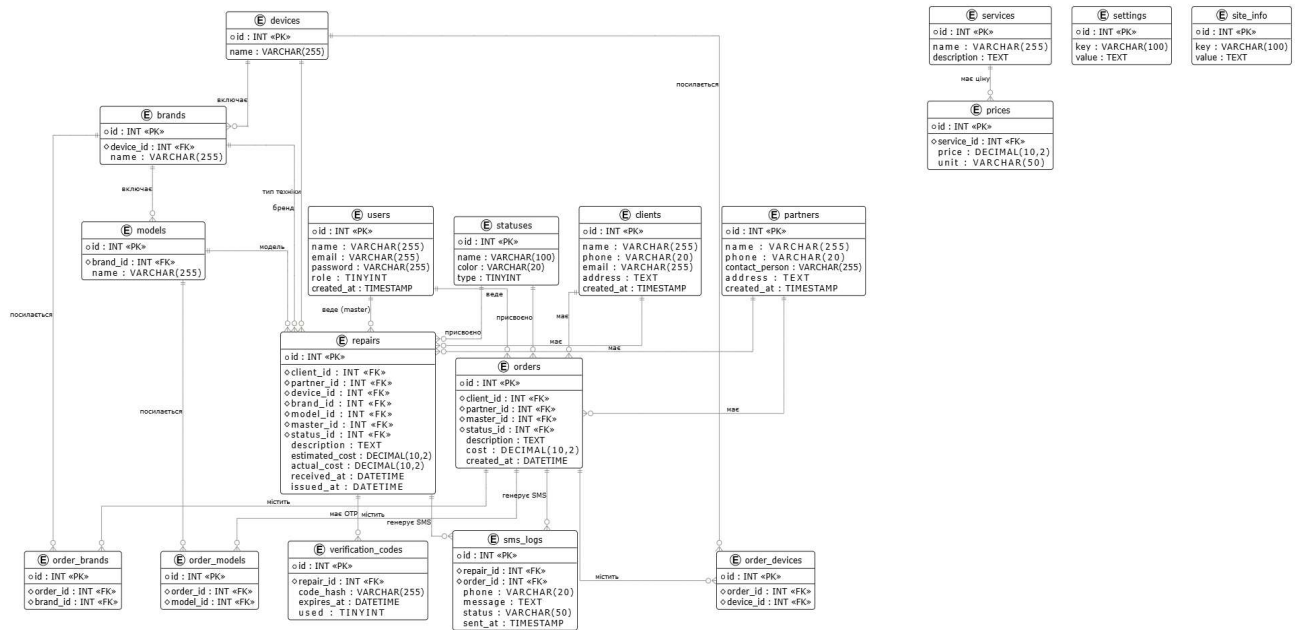


Рисунок 2.2 - Фізична модель (ER-діаграма) бази даних системи

2.3 Проектування та реалізація алгоритмів роботи системи

Архітектура вебзастосунку побудована на патерні MVC (Model-View-Controller) з розширеним сервісним шаром (Service Layer), що інкапсулює складну бізнес-логіку, яка виходить за межі відповідальності контролерів або моделей. На Рис. 2.3 наведено узагальнену архітектуру системи, що демонструє взаємодію компонентів.

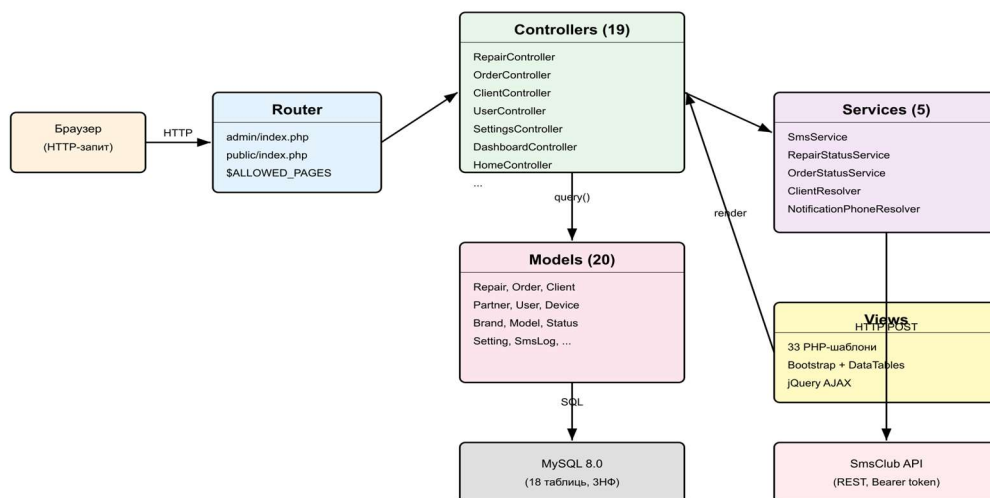


Рисунок 2.3 - Архітектура системи за патерном MVC з сервісним шаром

Маршрутизація запитів здійснюється через єдину точку входу: index.php для публічного сайту та admin/index.php для адміністративної панелі. Точка входу аналізує GET-параметр page, перевіряє його наявність у білому списку дозволених сторінок та делегує обробку відповідному PHP-файлу. Такий підхід забезпечує централізований контроль доступу та захист від несанкціонованого звернення до внутрішніх скриптів [12]. У лістингу 2.1 наведено фрагмент маршрутизації адміністративної панелі.

```
$ALLOWED_PAGES = [
    'dashboard', 'repairs', 'add-repair', 'edit-repair',
    'orders', 'clients', 'partners', 'users',
    'devices', 'models', 'brands', 'settings', ...
];

$page = (string)($_GET['page'] ?? 'dashboard');

if (!in_array($page, $ALLOWED_PAGES, true)) {
    http_response_code(404);
    header('Location: index.php?page=dashboard');
    exit;
}

$file = __DIR__ . '/' . $page . '.php';
require_once $file;
```

Лістинг 2.1 - Маршрутизація запитів в admin/index.php

Рольова модель доступу реалізована безпосередньо в маршрутизаторі: масиви \$ADMIN_ONLY_PAGES та \$SUPERADMIN_ONLY_PAGES визначають сторінки, доступні лише адміністраторам та менеджерам або лише адміністраторам відповідно. При спробі майстра відкрити адміністративну сторінку він перенаправляється на список своїх ремонтів [12]. Алгоритм створення ремонту є одним із найбільш складних у системі та ілюструє взаємодію кількох компонентів

архітектури. При надходженні POST-запиту RepairController виконує послідовність кроків, зображену на Рис. 2.4.

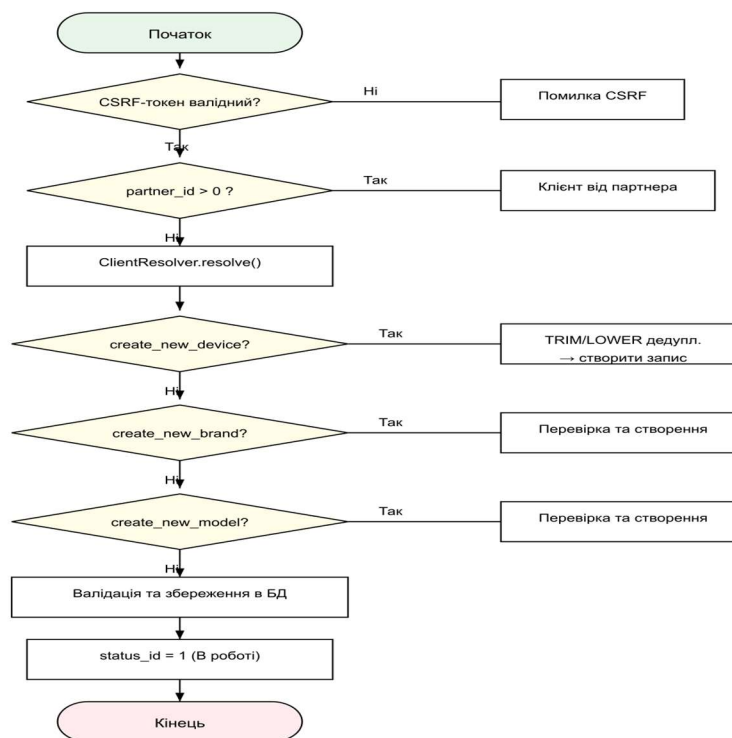


Рисунок 2.4 - Блок-схема алгоритму створення нового ремонту

Спочатку перевіряється CSRF-токен для захисту від міжсайтової підробки запитів. Далі клас ClientResolver визначає клієнта: якщо вказано партнера ($partner_id > 0$), клієнт створюється або знаходиться в контексті партнерської організації; якщо партнер не вказаний, виконується пошук за номером телефону або створення нового клієнта. Після цього обробляється ієрархія пристроїв: якщо користувач ввів нову назву пристрою, бренду або моделі (встановлені прапорці `create_new_device`, `create_new_brand`, `create_new_model`), система перевіряє чи не існує вже запис із такою назвою у базі даних (порівняння без урахування регістру через SQL-функції `TRIM` та `LOWER`), і лише у разі відсутності створює новий запис. Нарешті, формується масив даних ремонту, виконується валідація та створюється запис у базі зі статусом «В роботі» [12].

У лістингу 2.2 наведено першу частину методу `create()` класу `RepairController`, що демонструє механізм валідації CSRF-токена та визначення клієнта.

```

public function create() {
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        if (!Security::validateToken(
            $_POST['csrf_token'] ?? '')) {
            $this->redirectCreateError('csrf');
        }
        $partnerId = (int)($_POST['partner_id'] ?? 0);
        $clientResult = $this->clientResolver
            ->resolve($_POST, $partnerId);
        if (!empty($clientResult['error'])) {
            $this->redirectCreateError(
                $clientResult['error']);
        }
        $clientId = $clientResult['client_id'] ?? null;
    }
}

```

Лістинг 2.2 - Валідація та визначення клієнта у методі create()

На першому етапі виконується перевірка CSRF-токена для захисту від міжсайтової підробки запитів. Далі клас ClientResolver визначає клієнта: якщо вказано ідентифікатор партнера, клієнт створюється або знаходиться в контексті партнерської організації. У лістингу 2.3 наведено другу частину методу, що демонструє обробку ієрархії пристроїв із можливістю створення нових записів у довідниках.

```

$deviceId = $_POST['device_id'] ?? '';
$title = $_POST['device_title'] ?? '';
if (isset($_POST['create_new_device'])
    && trim($title) !== '') {
    $existing = $this->deviceModel
        ->getByName(trim($title));
}

```

```

        if ($existing && !empty($existing['id'])) {
            $deviceId = $existing['id'];
        } else {
            $deviceId = $this->deviceModel
                ->createAndReturnId(
                    ['title' => trim($deviceTitle)]
                );
        }
    }
    // Аналогічна логіка для brand та model...
}
}

```

Лістинг 2.3 - Обробка ієрархії пристроїв у методі create()

Алгоритм зміни статусу ремонту реалізований у класі RepairStatusService, який ізолює логіку нотифікацій від контролера. При зміні статусу контролер викликає метод sendStatusSms(), передаючи ідентифікатор ремонту, попередній та новий статуси, а також дані ремонту. Сервіс перевіряє: чи дійсно відбулася зміна статусу (якщо ні - SMS не надсилається), чи є новий статус таким, що потребує нотифікації (лише COMPLETED та ISSUED тригерять SMS). Далі клас NotificationPhoneResolver визначає номер телефону для відправки: якщо ремонт прив'язаний до партнера - використовується телефон клієнта, а якщо клієнта немає - телефон партнера. Для статусу COMPLETED формується повідомлення з шаблону, що включає назву пристрою, фінальну вартість (сума finish_repair_cost та detail_cost) та кількість днів зберігання. Для статусу ISSUED формується коротке повідомлення про видачу [24].

Алгоритм ОТР-верифікації видачі пристрою є ключовим елементом системи безпеки та реалізує двоетапну перевірку особи отримувача. Послідовність кроків алгоритму наведено на Рис. 2.5.

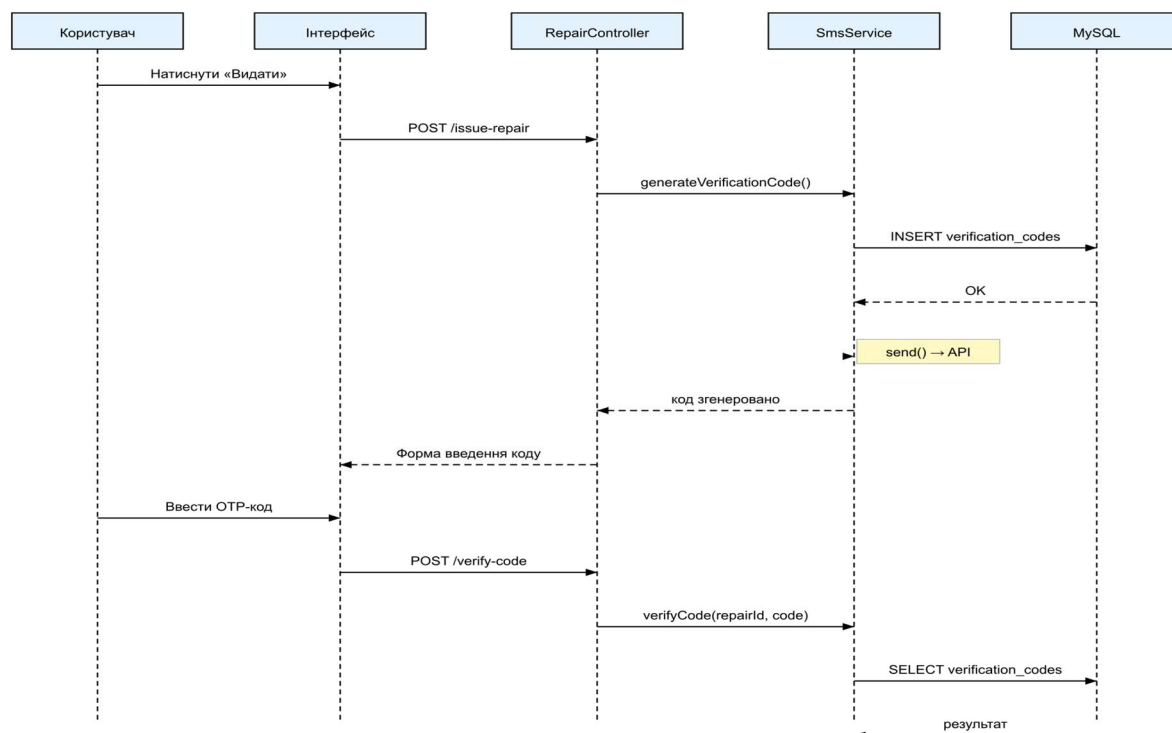


Рисунок 2.5 - Діаграма послідовності процесу OTP-верифікації видачі пристрою

При ініціації видачі система генерує випадковий чотиризначний код за допомогою функції `random_int(1000, 9999)`, зберігає його у таблиці `verification_codes` разом із `repair_id` та часом закінчення дії (поточний час + 10 хвилин). Попередні невикористані коди для цього ремонту деактивуються. Одночасно оригінальний код надсилається клієнту через `SmsService::send()`. При введенні коду користувачем система завантажує запис із бази, перевіряє відповідність коду, валідує час дії та прапорець використання, і у разі успіху переводить ремонт у статус «Видано» із фіксацією дати та часу видачі. У лістингу 2.4 наведено метод генерації одноразового коду верифікації.

```

public function generateVerificationCode($repairId) {
    $code = rand(1000, 9999);
    $expiresAt = date('Y-m-d H:i:s',
        strtotime('+10 minutes'));
    $this->db->query(
        "UPDATE verification_codes SET is_used = 1
        WHERE repair_id = ? AND is_used = 0",

```

```

        [$repairId]
    );
    $this->db->query(
        "INSERT INTO verification_codes
        (repair_id, code, expires_at)
        VALUES (?, ?, ?)",
        [$repairId, $code, $expiresAt]
    );
    return $code;
}

```

Лістинг 2.4 - Генерація OTP-коду в класі SmsService

Метод генерує випадковий чотиризначний числовий код за допомогою функції `rand(1000, 9999)`, обчислює час закінчення дії (поточний час плюс 10 хвилин) та зберігає запис у таблиці `verification_codes`. Перед створенням нового коду всі попередні невикористані коди для цього ремонту деактивуються (`is_used = 1`), що гарантує наявність лише одного активного коду в кожний момент часу. У лістингу 2.5 наведено метод верифікації введеного коду.

```

public function verifyCode($repairId, $code) {
    $sql = "SELECT id FROM verification_codes
    WHERE repair_id = ? AND code = ?
    AND is_used = 0
    AND expires_at > NOW()
    ORDER BY id DESC LIMIT 1";
    $result = $this->db->query(
        $sql, [$repairId, $code]);
    if ($result->num_rows > 0) {
        $row = $result->fetch_assoc();
        $this->db->query(
            "UPDATE verification_codes
            SET is_used = 1 WHERE id = ?",
            [$row['id']]
        );
    }
}

```

```

        );
        return true;
    }
    return false;
}

```

Лістинг 2.5 - Верифікація OTP-коду у класі SmsService

Метод виконує SQL-запит із чотирма умовами фільтрації: відповідність ремонту (repair_id), збіг коду, прапорець невикористаності (is_used = 0) та перевірка терміну дії (expires_at > NOW()). При успішній валідації код позначається як використаний, що унеможливлює його повторне застосування [20; 21].

Інтеграція із зовнішнім SMS-шлюзом реалізована у класі SmsService, який інкапсулює всю логіку взаємодії з API сервісу SmsClub. Клас надає метод send(), що приймає номер телефону та текст повідомлення, формує HTTP POST-запит до кінцевої точки API із заголовком авторизації у форматі Bearer-токена та тілом запиту у форматі JSON, обробляє відповідь сервера та повертає результат операції. Токен доступу до API зберігається у змінних середовища (.env файл) і ніколи не потрапляє до репозиторію вихідного коду, що відповідає вимогам безпеки щодо зберігання секретних ключів [20; 24]. У лістингу 2.6 наведено першу частину методу send() класу SmsService, що демонструє підготовку HTTP-запиту до API SmsClub.

```

public function send($phone, $message,
    $repairId = null, $type = 'manual',
    $orderId = null) {
    $cleanPhone = self::normalizeUaPhone($phone);
    $token = defined('SMS_API_TOKEN')
        ? SMS_API_TOKEN : '';
    $sender = defined('SMS_SENDER_NAME')
        ? SMS_SENDER_NAME : 'REMONT';
    $url = 'https://im.smsclub.mobi/sms/send';

```

```

$data = [
    'phone'    => [$cleanPhone],
    'message'  => $message,
    'src_addr' => $sender ];

```

Лістинг 2.6 - Підготовка запиту до SMS API у методі send()

Метод починається з нормалізації номера телефону до формату 380XXXXXXXXX за допомогою статичного методу `normalizeUaPhone()`, який видаляє всі нецифрові символи та додає код країни за потреби. Токен авторизації та ім'я відправника зчитуються з констант, визначених у файлі конфігурації на основі змінних середовища [24; 25]. У лістингу 2.7 наведено виконання HTTP-запиту та обробку відповіді API.

```

$ch = curl_init($url);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($data));
curl_setopt($ch, CURLOPT_HTTPHEADER, [
    'Content-Type: application/json',
    'Authorization: Bearer ' . $token
]);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($ch);
$httpCode = curl_getinfo($ch,
    CURLINFO_HTTP_CODE);
curl_close($ch);
$this->log($phone, $message, $status,
    $repairId, $type, $smsId, $orderId);
return $status === 'sent';
}

```

Лістинг 2.7 - Виконання HTTP-запиту та логування у методі send()

SMS-повідомлення надсилаються автоматично у двох випадках. По-перше, при зміні статусу ремонту або замовлення - відповідний текст формується з

налаштовуваного шаблону, збереженого в таблиці налаштувань системи, з підстановкою номера замовлення. По-друге, окремий cron-сценарій виконує щоденну розсилку нагадувань клієнтам, чия техніка перебуває у статусі «Готово» вже понад визначений адміністратором час. Крім того, при переведенні ремонту у статус «Видано» система генерує одноразовий чотиризначний числовий код, надсилає його клієнту та зберігає хеш-значення у базі даних. Код дійсний протягом 10 хвилин, після чого автоматично скидається - це унеможлиблює зловживання та підвищує безпеку процесу видачі [20].

Метод `buildMessage()` дозволяє формувати текст повідомлення із шаблону, збереженого в таблиці `settings`: шаблон містить плейсхолдери (`{device}`, `{final_cost}`, `{days_to_pickup}` тощо), які замінюються реальними значеннями перед відправкою. Якщо шаблон не знайдено в базі, використовується значення за замовчуванням, передане другим параметром. Усі відправлені повідомлення логуються в таблицю `sms_logs` із зазначенням номера, тексту, ідентифікатора ремонту або замовлення, статусу відправки та часу [24].

Автоматичне нагадування про необхідність забрати готову техніку реалізовано у вигляді PHP-скрипту, що запускається через cron щоденно. Скрипт вибирає з бази всі ремонти зі статусом `COMPLETED`, дата завершення яких перевищує встановлений в налаштуваннях поріг (за замовчуванням - 3 доби), та для кожного з них формує і надсилає SMS-нагадування через `SmsService`. Це дозволяє мінімізувати кількість пристроїв, що зберігаються у сервісному центрі після завершення ремонту, та покращити клієнтський досвід.

2.4 Реалізація програмного комплексу

Програмний комплекс вебзастосунку складається з двох основних частин: адміністративної панелі та публічного сайту. Адміністративна панель містить 33 PHP-сторінки у директорії `admin/`, кожна з яких відповідає за окрему функціональну область: список ремонтів (`repairs.php`), форма створення ремонту (`add-repair.php`), форма редагування (`edit-repair.php`), форма видачі з верифікацією (`issue-repair.php`),

аналогічні сторінки для замовлень, клієнтів, партнерів, довідників, налаштувань та статистики. Публічний сайт реалізований єдиним контролером HomeController та шаблоном home.php, що формує landing page із секціями: героїчний блок, послуги, про нас, партнери, контакти та футер.

Файлова структура проекту організована за принципом розділення відповідальностей. Директорія app/ містить ядро застосунку: Core/ (базові класи Database, Security, Autoloader), Controllers/ (19 контролерів), Models/ (20 моделей), Services/ (5 сервісних класів) та Views/ (шаблони). Директорія admin/ містить сторінки адміністративної панелі, статичні ресурси (CSS, JS) та AJAX-скрипти. Директорія config/ зберігає конфігурацію застосунку, а db/ - SQL-файли ініціалізації та міграцій. Директорія public/ містить файли публічного сайту, tests/ - автоматизовані тести, а bin/ - утилітні скрипти (cron-завдання, резервне копіювання).

Контролерний шар системи включає 19 класів, організованих у просторі імен App\Controllers. Кожен контролер відповідає за конкретну сутність або функціональну область: RepairController (ремонти), OrderController (замовлення), ClientController (клієнти), PartnerController (партнери), DeviceController, BrandController, ModelController (довідники ремонтів), OrderDeviceController, OrderBrandController, OrderModelController (довідники замовлень), UserController (користувачі), PriceController (прайс-лист), SettingsController (налаштування), DashboardController (головна панель), ReceiptController (квитанції), SmsController (журнал SMS), HomeController (публічний сайт), ServiceController (послуги на сайті), SiteInfoController (інформація про сайт). Базовий клас BaseController надає спільний функціонал: перевірку авторизації та валідацію CSRF-токена [12].

У лістингу 2.8 наведено метод перевірки авторизації базового контролера.

```
class BaseController {
    protected function checkAuth() {
        if (session_status() === PHP_SESSION_NONE) {
            session_start();
        }
    }
}
```

```

    }
    if (!isset($_SESSION['admin_id'])) {
        header('Location: index.php');
        exit;
    }
}
}

```

Лістинг 2.8 - Метод перевірки авторизації у BaseController

Метод `checkAuth()` перевіряє наявність ідентифікатора адміністратора у сесії та перенаправляє неавторизованого користувача на сторінку входу. Аналогічний метод `validateCsrf()` реалізує захист від CSRF-атак: при обробці POST-запитів перевіряється токен, переданий через приховане поле форми або заголовок X-CSRF-TOKEN. Для AJAX-запитів метод повертає JSON-відповідь з кодом 403, для звичайних запитів - текстове повідомлення про помилку. Обидва методи викликаються у конструкторах дочірніх контролерів, що гарантує перевірку безпеки для кожного запиту [20; 21].

Модельний шар містить 20 класів у просторі імен `App\Models`. Кожна модель інкапсулює роботу з відповідною таблицею бази даних через клас `Database`. Базовий функціонал моделей включає методи `getAll()` для отримання всіх записів з можливістю фільтрації, `getById()` для отримання одного запису за ідентифікатором, `create()` для створення нового запису, `update()` для оновлення існуючого та `delete()` для видалення. Модель `Repair` додатково реалізує методи `getStatus()` для фільтрації за статусом, `getByMaster()` для фільтрації за майстром, `getCount()` для підрахунку загальної кількості та `getCountsByStatus()` для підрахунку записів у кожному статусі, що використовується для формування `dashboard` [8].

Клас `Database` реалізує патерн обгортки (`Wrapper`) над розширенням `MySQLi` та надає уніфікований інтерфейс для виконання запитів. Метод `select()` повертає масив рядків, `selectOne()` повертає один рядок або `null`, `execute()` виконує

INSERT/UPDATE/DELETE та повертає кількість зачеплених рядків. Усі методи використовують підготовлені запити (prepared statements) з параметризованими значеннями, що унеможливорює SQL-ін'єкції. Клас також підтримує транзакції через методи beginTransaction(), commit() та rollback(), а також автоматичне визначення хосту бази даних для коректної роботи як у локальному середовищі, так і в Docker-контейнері [9; 20].

У лістингу 2.9 наведено публічні методи select() та execute() класу Database, що надають уніфікований інтерфейс для виконання запитів.

```
public function select(string $sql,
    array $params = []): array {
    $result = $this->query($sql, $params);
    $rows = [];
    if ($result instanceof \mysqli_result) {
        while ($row = $result->fetch_assoc()) {
            $rows[] = $row;
        }
    }
    return $rows; }

public function execute(string $sql,
    array $params = []): int {
    $result = $this->query($sql, $params);
    if (is_int($result)) {
        return $result;
    }
    return 0; }
```

Лістинг 2.9 - Методи select() та execute() класу Database

Метод select() виконує SELECT-запит та повертає масив асоціативних масивів, де кожен елемент відповідає одному рядку результату. Метод execute() призначений для запитів INSERT, UPDATE та DELETE і повертає кількість зачеплених рядків. Обидва методи делегують виконання приватному методу query(), що інкапсулює логіку prepared statements. У лістингу 2.10 наведено реалізацію цього методу.

```

public function query($sql, $params = []) {
    $stmt = $this->conn->prepare($sql);
    if (!empty($params)) {
        $types = '';
        foreach ($params as $param) {
            if (is_int($param))      $types .= 'i';
            elseif (is_float($param)) $types .= 'd';
            elseif ($param === null)  $types .= 's';
            else                      $types .= 's';
        }
        $stmt->bind_param($types, ...$params);
    }
    $stmt->execute();
    if ($stmt->field_count > 0) {
        return $stmt->get_result();
    }
    return $stmt->affected_rows;
}

```

Лістинг 2.10 - Метод query() з prepared statements

Сервісний шар системи включає п'ять класів у просторі імен App\Services, кожен з яких інкапсулює окрему область бізнес-логіки. SmsService відповідає за відправку SMS через SmsClub API, формування повідомлень із шаблонів, генерацію та верифікацію OTP-кодів, а також перевірку статусів доставки. RepairStatusService та OrderStatusService реалізують логіку реакції на зміну статусів ремонтів та замовлень відповідно, включаючи визначення номера телефону для нотифікації, формування тексту повідомлення та виклик SmsService. ClientResolver інкапсулює логіку визначення або створення клієнта при оформленні ремонту чи замовлення: обробляє сценарії з партнером та без, перевіряє існування клієнта за номером телефону та створює нового при необхідності. NotificationPhoneResolver визначає номер телефону для відправки SMS-повідомлення з урахуванням контексту: для

ремонту з партнером - телефон клієнта, для ремонту без партнера - телефон клієнта, при відсутності клієнта - телефон партнера [12].

У (табл. 2.6) наведено детальний опис класів сервісного шару із зазначенням їх основних методів та відповідальності.

Таблиця 2.6
Класи сервісного шару системи

Клас	Основні методи	Відповідальність
SmsService	send(), buildMessage(), generateVerificationCode(), verifyCode(), checkStatuses(), getBalance()	Інтеграція з SMS-шлюзом SmsClub API, генерація та перевірка OTP-кодів
RepairStatusService	sendStatusSms(), handleCompleted(), handleIssued()	Реакція на зміну статусу ремонту: визначення телефону, формування SMS
OrderStatusService	sendStatusSms(), handleCompleted(), handleIssued()	Реакція на зміну статусу замовлення: аналогічна логіка для замовлень
ClientResolver	resolve(), findByPhone(), createNew()	Визначення або створення клієнта при оформленні ремонту/замовлення
NotificationPhoneResolver	resolve()	Визначення номера телефону для SMS з урахуванням партнера/клієнта

У (табл. 2.7) наведено основні методи класу Database, що забезпечує уніфікований інтерфейс доступу до бази даних.

Таблиця 2.7
Методи класу Database

Метод	Параметри	Повертає	Опис
select()	\$sql, \$params	array	Виконує SELECT, повертає масив рядків
selectOne()	\$sql, \$params	array null	Повертає перший рядок або null
execute()	\$sql, \$params	int	Виконує INSERT/UPDATE/DELETE, повертає affected_rows
query()	\$sql, \$params	mixed	Базовий метод із prepared statements
insert()	\$table, \$data	bool	Вставка рядка в таблицю

Продовження табл. 2.7

update()	\$table, \$where	\$data, bool	Оновлення рядків за умовою
delete()	\$table, \$where	bool	Видалення рядків за умовою
beginTransaction()	-	bool	Початок транзакції
commit()	-	bool	Фіксація транзакції
rollback()	-	bool	Відкат транзакції
getLastInsertId()	-	int	ID останнього вставленого рядка

На Рис. 2.6 наведено діаграму основних компонентів системи в нотатії UML, що відображає структуру основних класів, їх атрибути, методи та зв'язки між ними.



Рисунок 2.6 - Діаграма класів системи (UML Class Diagram)

Інтерфейс адміністративної панелі побудований на основі CSS-фреймворку Bootstrap з використанням компонентів: навігаційна панель (navbar), бічне меню (sidebar), картки (cards) для відображення статистики на dashboard, модальні вікна (modals) для підтвердження дій, форми (forms) для введення даних. Табличні дані відображаються за допомогою бібліотеки DataTables, що забезпечує вбудований пошук, сортування за будь-яким стовпцем, пагінацію та адаптивне відображення на мобільних пристроях [22; 23].

Динамічна взаємодія реалізована через AJAX-запити на основі бібліотеки jQuery. При виборі категорії пристрою у формі ремонту JavaScript-обробник надсилає AJAX-запит до серверного скрипту, який повертає JSON-масив відповідних брендів. Після вибору бренду аналогічний запит завантажує список моделей. Такий каскадний підхід мінімізує обсяг даних, що передаються клієнту, та забезпечує швидкий відгук інтерфейсу. У лістингу 2.11 наведено серверний обробник AJAX-запиту для завантаження брендів за обраним пристроєм.

```
$brandModel = new Brand();
$device_id = $_GET['device_id'] ?? '';
if ($device_id) {
    $brands = $brandModel->getBrandsByDevice($device_id);
    echo json_encode([
        'success' => true,
        'data'     => $brands,
    ]);
} else {
    echo json_encode([
        'success' => true,
        'data'     => [],
    ]);
}
```

Лістинг 2.11 - AJAX-обробник завантаження брендів (get-brands.php)

Аналогічно для верифікації OTP-коду: введений код надсилається AJAX-запитом, сервер перевіряє його та повертає JSON-відповідь, на основі якої JavaScript динамічно оновлює інтерфейс сторінки [15; 23].

Система безпеки вебзастосунку реалізована на кількох рівнях. На рівні авторизації - паролі зберігаються в хешованому вигляді (bcrypt, cost factor 12), сесії захищені від фіксації та перехоплення через налаштування cookie (httponly, samesite, secure). На рівні захисту від CSRF - кожна форма містить прихований токен, що генерується класом Security та перевіряється при обробці POST-запитів,

у тому числі для AJAX-запитів через заголовок X-CSRF-TOKEN. На рівні захисту від SQL-ін'єкцій - усі запити до бази виконуються через prepared statements з автоматичним визначенням типів параметрів. На рівні захисту секретів - API-токени, паролі бази даних та інші конфіденційні дані зберігаються у файлі .env який виключено з репозиторію через .gitignore [20].

На Рис. 2.7 наведено діаграму компонентів системи, що демонструє взаємозв'язки між основними модулями вебзастосунку та зовнішніми сервісами.

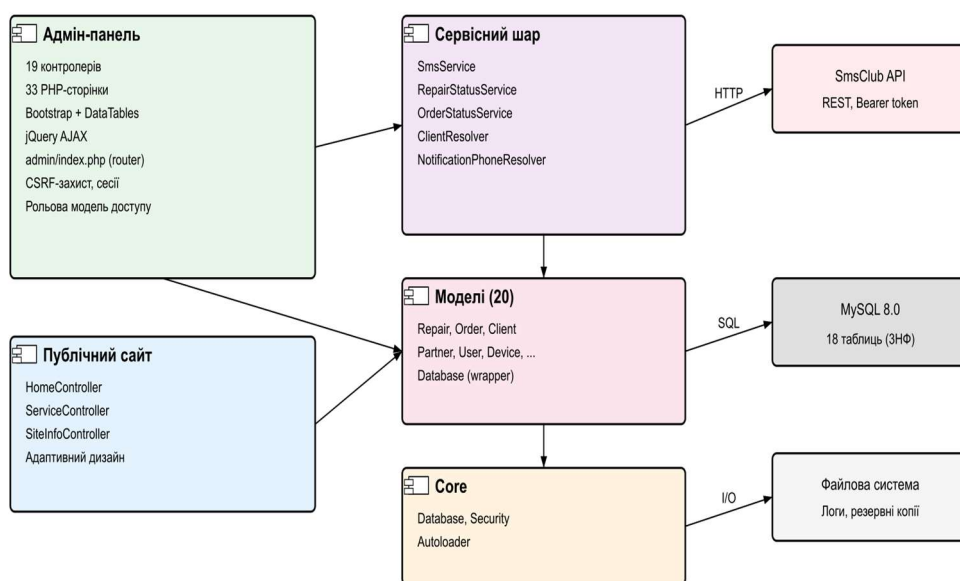


Рисунок 2.7 - Діаграма компонентів системи (UML Component Diagram)

Як видно з діаграми, система складається з трьох основних компонентів: адміністративна панель (19 контролерів, 33 сторінки), публічний сайт (HomeController, ServiceController, SiteInfoController) та сервісний шар (5 сервісних класів). Зовнішніми залежностями є база даних MySQL, SMS-шлюз SmsClub API та файлова система для зберігання логів та резервних копій.

Висновки до розділу 2

У другому розділі виконано проектування та описано реалізацію програмного забезпечення вебзастосунку для адміністрування сервісного центру з ремонту побутової техніки.

У підрозділі 2.1 визначено чотирьох акторів системи (адміністратор, менеджер, майстер, відвідувач) та специфіковано варіанти використання, що охоплюють повний цикл обслуговування клієнта: від авторизації в системі та реєстрації ремонту до верифікації видачі пристрою та перегляду статистики. Побудовано діаграму варіантів використання у нотації UML 2.0, наведено детальні описи двох найбільш критичних варіантів - UC-2 «Реєстрація нового ремонту» та UC-5 «Верифікація видачі пристрою (ОТР)».

У підрозділі 2.2 спроектовано реляційну базу з дотриманням третьої нормальної форми (3НФ). Центральними сутностями є таблиці repairs та orders, навколо яких побудовано решту структури через зовнішні ключі. Реалізовано ієрархічну структуру довідників пристроїв (devices - brands - models), систему для керованого оновлення схеми бази даних та наведено ER-діаграму фізичної моделі.

У підрозділі 2.3 описано архітектуру системи за патерном MVC з розширеним сервісним шаром та наведено алгоритми роботи ключових модулів: створення ремонту з динамічним довідником пристроїв через клас ClientResolver та перевірку дублікатів через SQL-функції TRIM та LOWER, зміну статусів із автоматичними SMS-нотифікаціями через класи RepairStatusService та NotificationPhoneResolver, ОТР-верифікацію видачі з генерацією чотиризначного коду та обмеженням дії у десять хвилин, а також автоматичне нагадування клієнтам через cron-задачу.

У підрозділі 2.4 представлено реалізацію програмного комплексу: дев'ятнадцять контролерів у просторі імен App\Controllers, двадцять моделей, п'ять сервісних класів (SmsService, RepairStatusService, OrderStatusService, ClientResolver, NotificationPhoneResolver), клас Database із prepared statements для захисту від SQL-ін'єкцій та систему безпеки на кількох рівнях (bcrypt-хешування паролів, CSRF-токени, захищені сесії, зберігання секретів у .env). Наведено діаграму класів та діаграму компонентів системи, лістинги ключових фрагментів програмного коду, а також описано інтерфейсні рішення на основі Bootstrap та DataTables із механізмом динамічної взаємодії через AJAX.

РОЗДІЛ 3. ІНТЕРФЕЙС, ТЕСТУВАННЯ ТА РОЗГОРТАННЯ СИСТЕМИ

3.1 Порядок встановлення та налаштування системи

Розроблений вебзастосунок розгортається з використанням технології контейнеризації Docker, що забезпечує відтворюваність середовища та незалежність від конфігурації операційної системи сервера. Для встановлення та запуску системи необхідні наступні програмні засоби: Docker Engine версії 20.10 або новіше та Docker Compose версії 2.0 або новіше. На операційних системах Windows та macOS достатньо встановити Docker Desktop, який включає обидва компоненти [17; 18]. Інфраструктура проєкту описана у файлі `docker-compose.yml` і показано у (табл. 3.1).

Таблиця 3.1
Склад Docker-інфраструктури вебзастосунку

Сервіс	Образ	Порт	Призначення
app	php:8.2-apache (custom)	8080:80	PHP-застосунок + Apache
db	mysql:8.0	3306:3306	База даних MySQL
phpmyadmin	phpmyadmin/phpmyadmin	8082:80	Адміністрування БД

Перший сервіс - app - відповідає за PHP-застосунок, побудований на основі офіційного образу `php:8.2-apache`. Під час збирання образу автоматично встановлюються розширення `mysqli` та `pdo_mysql`, підключається модуль `Apache mod_rewrite` та копіюється `Composer`. Другий сервіс - db - запускає `MySQL 8.0` з автоматичною ініціалізацією бази даних зі скрипту `SQL` при першому запуску. Третій сервіс - `phpmyadmin` - надає графічний вебінтерфейс для адміністрування бази даних у середовищі розробки. Усі сервіси об'єднані у спільну мережу Docker і взаємодіють між собою за іменами сервісів, що спрощує конфігурацію [18; 19].

Процедура розгортання системи складається з кількох послідовних кроків. Спочатку необхідно клонувати репозиторій проєкту з віддаленого сховища командою `git clone`. Далі створюється файл конфігурації середовища `.env` шляхом

копіювання шаблону `cp .env.example .env`, у якому задаються параметри підключення до бази даних, облікові дані SMS-шлюзу SmsClub та режим роботи системи (local або production). Після цього виконується команда `docker-compose up -d --build`, яка збирає Docker-образи трьох сервісів (app на базі `php:8.2-apache`, db на базі `mysql:8.0` та `phpmyadmin`) і запускає контейнери у фоновому режимі. Під час першого запуску автоматично виконується скрипт початкового налаштування `demo-setup.php`, який застосовує міграції бази даних та створює обліковий запис адміністратора. Для зупинки системи використовується команда `docker-compose down`, а для зупинки з видаленням даних бази даних - `docker-compose down -v`. Процедuru розгортання зведено у (табл 3.2)

Таблиця 3.2
Команди розгортання системи

Крок	Команда	Призначення
1	<code>git clone <url></code>	Клонування репозиторію
2	<code>cp .env.example .env</code>	Створення файлу конфігурації
3	<code>docker-compose up -d --build</code>	Збирання образів та запуск контейнерів
5	<code>docker-compose down</code>	Зупинка контейнерів
6	<code>docker-compose down -v</code>	Зупинка з видаленням даних БД

На першому кроці необхідно клонувати репозиторій проєкту з системи контролю версій Git на цільовий сервер або робочу станцію. На другому кроці необхідно створити файл конфігурації середовища `.env` на основі шаблону `.env.example`, заповнивши параметри підключення до бази даних (`DB_HOST`, `DB_USER`, `DB_PASSWORD`, `DB_NAME`), параметри SMS-шлюзу (`SMS_API_TOKEN`, `SMS_SENDER_NAME`) та загальні параметри застосунку (`APP_ENV`, `APP_URL`). Файл `.env` виключено з репозиторію через `.gitignore`, що забезпечує безпеку конфіденційних даних [20].

На третьому кроці виконується команда `docker-compose up -d --build`, яка ініціює збирання Docker-образу PHP-застосунку відповідно до інструкцій у `Dockerfile`, завантаження офіційних образів MySQL 8.0 та phpMyAdmin, створення

контейнерів та їх запуск у фоновому режимі. Під час першого запуску MySQL автоматично виконує файл ініціалізації `collect7_tandem.sql`, що створює структуру бази даних та заповнює довідники початковими даними. Docker Compose забезпечує правильну послідовність запуску: контейнер застосунку чекає на готовність MySQL завдяки механізму `healthcheck` [18; 19].

На четвертому кроці виконується скрипт початкового налаштування `demo-setup.php`, який запускає всі міграції бази даних із директорії `db/migrations`, створює обліковий запис адміністратора, заповнює таблицю `settings` текстовими налаштуваннями для публічного сайту та оновлює контактну інформацію підприємства в таблиці `site_info`. Цей скрипт є ідемпотентним - його можна запускати повторно без ризику дублювання даних.

Після успішного розгортання система доступна за адресою `http://localhost:8080/` (публічний сайт), `http://localhost:8080/admin/` (адміністративна панель) та `http://localhost:8082/` (phpMyAdmin для адміністрування бази даних). Порти можуть бути змінені у файлі `docker-compose.yml` відповідно до потреб підприємства.

Система підтримує два режими роботи, що визначаються змінною середовища `APP_ENV`. У режимі `local` увімкнено відображення помилок PHP на сторінці, що спрощує діагностику під час розробки. У режимі `production` помилки логуються у файл, а користувачу відображається загальне повідомлення про помилку, що відповідає вимогам безпеки.

Резервне копіювання бази даних здійснюється автоматично за допомогою `bash`-скрипту `backup_db.sh`, який створює дамп бази даних у форматі SQL з позначкою дати та часу і зберігає його у директорії `db/backups`. Скрипт рекомендується додати до планувальника `cron` для щоденного виконання. Відновлення з резервної копії виконується імпортом SQL-файлу через phpMyAdmin або командний рядок `mysql` [10].

Система міграцій бази даних дозволяє керовано вносити зміни до структури

БД. Кожна міграція представляє собою SQL-файл у директорії db/migrations із іменем, що включає дату створення та короткий опис змін. Міграції виконуються в алфавітному порядку при запуску demo-setup.php та є ідемпотентними - використовують конструкції IF NOT EXISTS та ON DUPLICATE KEY UPDATE, що дозволяє безпечно повторювати їх. На даний момент система містить 12 файлів міграцій, що охоплюють створення таблиць settings, verification_codes, dashboard_links, receipt_texts, додавання полів для публічного сайту, зовнішніх ключів та індексів.

3.2 Опис інтерфейсу користувача

Інтерфейс вебзастосунку складається з двох частин: публічного сайту сервісного центру та адміністративної панелі управління. Обидві частини побудовані на основі CSS-фреймворку Bootstrap, що забезпечує адаптивне відображення на пристроях з різною роздільною здатністю екрана [23].

Публічний сайт являє собою односторінковий лендінг, що містить інформацію про сервісний центр та надає потенційним клієнтам контактні дані підприємства. Усі текстові елементи публічного сайту зберігаються в таблиці settings бази даних та можуть бути змінені адміністратором через інтерфейс управління без необхідності редагування програмного коду. Сторінка складається з шести основних секцій, кожна з яких виконує окрему комунікативну функцію.

Героїчний блок (hero section) розташований у верхній частині сторінки та є першим візуальним елементом, який бачить відвідувач. Він містить назву сервісного центру, короткий слоган, а також дві кнопки швидкої навігації: «Наші послуги» (прокрутка до відповідної секції) та «Зв'язатися з нами» (прокрутка до секції контактів). Фонове зображення та текст заголовка забезпечують впізнаваність бренду та формують перше враження про підприємство. Навігаційна панель дозволяє швидко переходити між секціями сторінки Рис. 3.1.

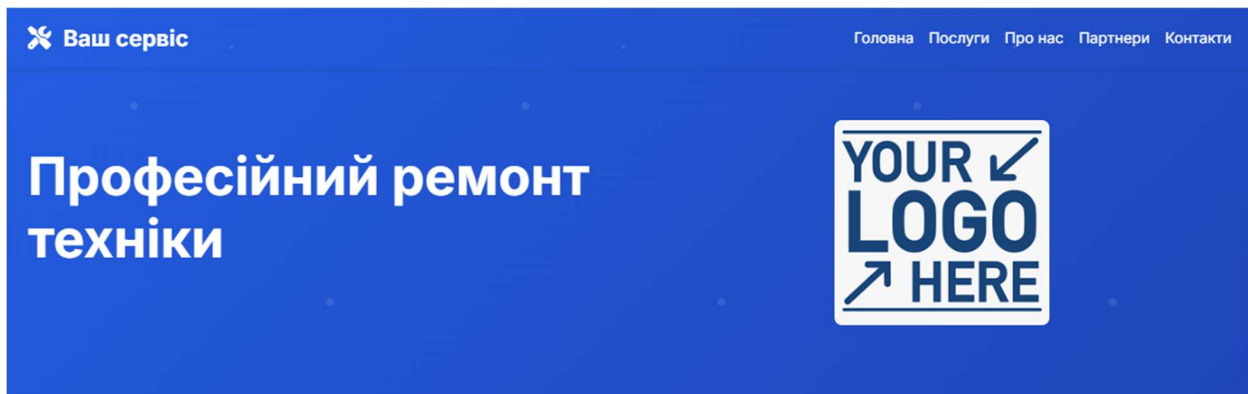


Рисунок 3.1 - Героїчний блок публічного сайту

Секція «Про нас» містить текстовий опис сервісного центру, його досвід та переваги. Блок включає чотири ключові характеристики підприємства (досвідчені майстри, оригінальні запчастини, гарантія на роботу, швидкий ремонт) та статистичні показники: кількість відремонтованих пристроїв, відсоток задоволених клієнтів, років досвіду роботи та режим підтримки. Текстовий вміст та статистика зберігаються у таблиці `site_info` та можуть бути оновлені адміністратором Рис. 3.2.

Про сервіс Service

Ми - команда професіоналів з багаторічним досвідом у ремонті техніки. Наша мета - надавати якісні послуги за доступними цінами.

- ✓ Гарантія на роботи
- ✓ Оригінальні запчастини
- ✓ Швидкий ремонт
- ✓ Безкоштовна діагностика

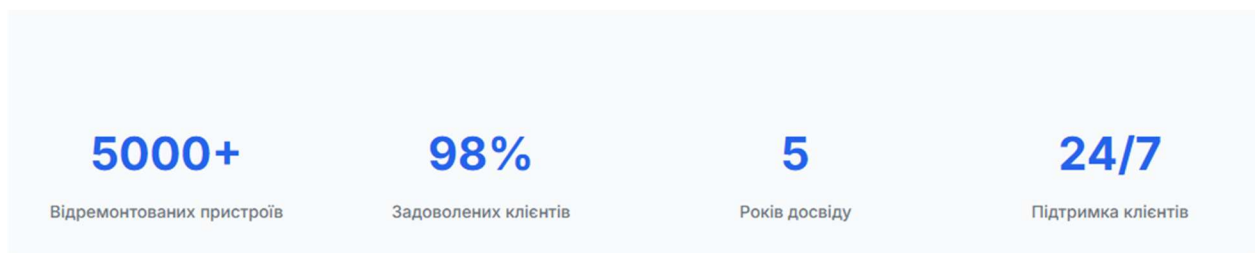


Рисунок 3.2 - Секція «Про нас» публічного сайту

Секція послуг відображає перелік видів ремонту, які надає сервісний центр. Кожна послуга представлена у вигляді картки з іконкою, назвою та коротким описом. Дані про послуги зберігаються в таблиці services бази даних і можуть бути додані, відредаговані або видалені через адміністративну панель. Якщо список послуг порожній, відображається відповідне повідомлення. Секція дозволяє швидко зрозуміти спеціалізацію сервісного центру Рис. 3.3.

Наші послуги

Ми надаємо повний спектр послуг з ремонту та обслуговування техніки

Послуги незабаром з'являться

Рисунок 3.3 - Секція послуг публічного сайту

Секція партнерів відображає перелік організацій, з якими співпрацює сервісний центр у сфері гарантійного обслуговування. Кожен партнер представлений карткою з назвою та контактними даними. Дані завантажуються з таблиці partners бази даних, що забезпечує автоматичне оновлення інформації при додаванні нових партнерів через адміністративну панель Рис. 3.4.

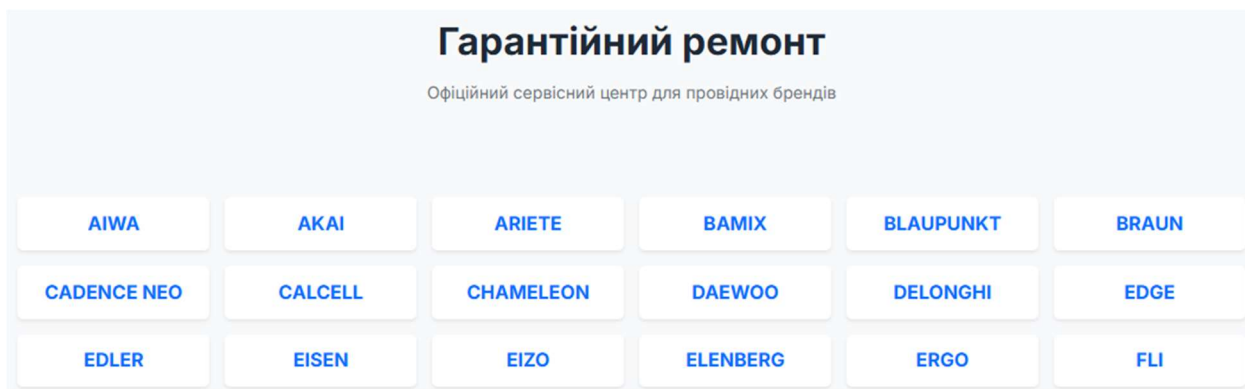


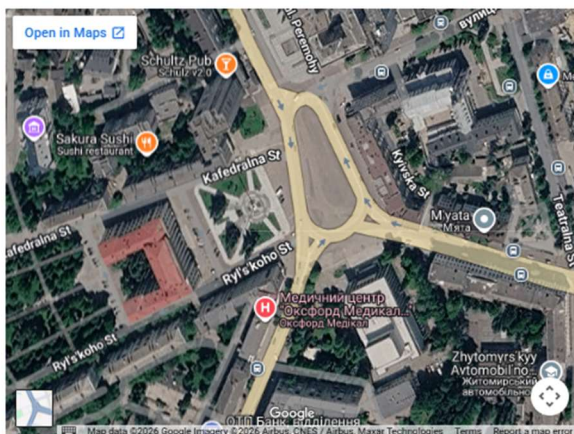
Рисунок 3.4 - Секція партнерів публічного сайту

Секція контактів надає відвідувачу всю необхідну інформацію для зв'язку із сервісним центром: адресу, номер телефону, електронну пошту та графік роботи.

Поруч із контактними даними вбудована карта Google Maps, що відображає місцезнаходження підприємства та дозволяє побудувати маршрут. URL карти зберігається у полі `map_url` таблиці `site_info` і може бути змінений адміністратором для відображення актуальної геолокації Рис. 3.5.

Зв'яжіться з нами

Ми завжди готові допомогти вам



Контактна інформація

-  **Адреса**
м. Місто, вул. Вулиця 8
-  **Телефон**
+38 (077) 777-77-77
-  **Email**
info@exsamle.com
-  **Графік роботи**
Пн-Пт: 9:00 - 18:00 (без перерви)
Сб: 09:00 - 14:00 (без перерви)
Нд: Вихідний

Рисунок 3.5 - Секція контактів із картою Google Maps

Футер (нижній колонтитул) сайту містить чотири колонки з додатковою навігацією: послуги, компанія, підтримка та контакти. Кожна колонка має заголовок та чотири посилання, тексти яких зберігаються у таблиці `settings`. У нижній частині футера розміщено текст копірайту. Футер забезпечує доступ до додаткової інформації та формує завершений вигляд.

Адміністративна панель доступна за адресою `/admin/` та захищена системою авторизації. При першому зверненні відображається форма входу, що приймає email користувача та пароль. Після успішної авторизації система перевіряє роль користувача (адміністратор, менеджер або майстер) та формує інтерфейс відповідно до рівня доступу Рис. 3.6.

Рисунок 3.6 - Форма авторизації адміністративної панелі

У системі реалізовано рольову модель доступу з трьома рівнями. Адміністратор має повний доступ до всіх функцій системи, включаючи налаштування публічного сайту, SMS-шаблонів та управління обліковими записами. Менеджер має доступ до основних функцій управління ремонтами, замовленнями, клієнтами та довідниками, але не може змінювати налаштування сайту та редагувати облікові записи адміністраторів. Майстер має обмежений доступ: він бачить лише призначені йому ремонти та може змінювати їх статус, але не має доступу до замовлень, довідників, налаштувань та управління персоналом. У підсумку маємо (табл. 3.3)

Таблиця 3.3
Розподіл доступу за ролями користувачів

Функція	Адміністратор	Менеджер	Майстер
Dashboard (головна панель)	Так	Так	Ні
Статистика по майстрах	Так	Так	Ні
Управління ремонтами	Всі	Всі	Тільки свої
Створення ремонту	Так	Так	Ні

Продовження табл. 3.3

Видача ремонту (ОТР)	Так	Так	Ні
Замовлення запчастин	Так	Так	Ні
Довідники (пристрої, бренди, моделі)	Так	Так	Ні
Клієнти та партнери	Так	Так	Ні
Управління персоналом	Так	Так*	Ні
Налаштування сайту	Так	Ні	Ні
SMS-шаблони, тексти квитанцій	Так	Так	Ні

*Примітка: * - менеджер може редагувати облікові записи менеджерів та майстрів, але не адміністраторів. Також менеджер не може деактивувати або видалити власний обліковий запис.*

Головна панель (Dashboard) відображається після авторизації та містить зведену інформацію про поточний стан системи: загальну кількість ремонтів, кількість ремонтів у роботі, кількість прострочених ремонтів (у роботі більше 14 днів), кількість завершених ремонтів, аналогічні показники для замовлень запчастин, а також баланс SMS-повідомлень. Під картками статистики відображається таблиця прострочених ремонтів із зазначенням клієнта, пристрою та кількості днів прострочення Рис. 3.7.

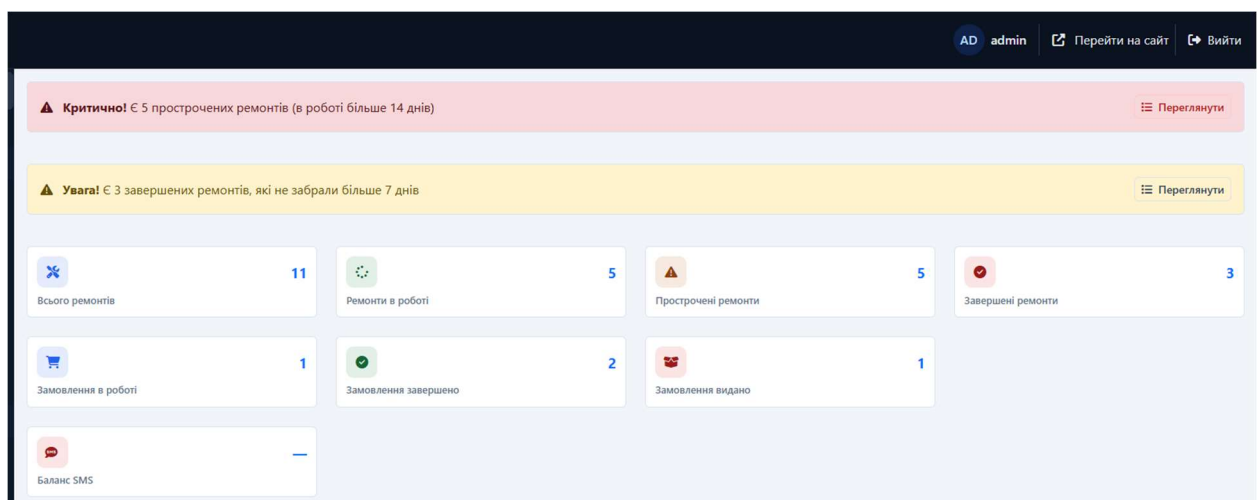


Рисунок 3.7 - Dashboard адміністративної панелі

Сторінка управління ремонтами є центральним елементом адміністративної панелі. У верхній частині розташовані фільтри за статусом (Всі, В роботі, Завершені, Видано, Прострочені) та фільтр за майстром, що дозволяє швидко знайти потрібні ремонти. Нижче відображається інтерактивна таблиця DataTables із колонками: ID, Інформація (пристрій + клієнт), Статус, Дата створення, Дата завершення, Дії. Таблиця підтримує пошук за будь-яким полем, сортування за будь-яким стовпцем та пагінацію. Для кожного ремонту доступні кнопки дій: перегляд деталей, друк квитанції, редагування, робота з ремонтом та швидка видача Рис. 3.8.

Управління ремонтами Всього: 11 [+ Додати ремонт](#)

Фільтр по статусу: Всі 11 В роботі 5 Завершені 3 Видано 3 Прострочені 5

Фільтр по майстру: Всі 11 master1 4 Холодильники 2 Телевізори 1 Ткаченко 2 Іваненко 2

Показати 25 записів Пошук:

ID	Інформація	Статус	Дата створення	Дата завершення	Дії
108	Машинка для підстригання волосся VALERA 1 VL-SWISS-5400 Савченко Юлія м. Житомир, бул. Старий, 7 0681234567	Видано	10.04.2026	Завершено: 18.04.2026 Видано: 20.04.2026	🔍 🖨 ✎ 🗑
107	Блендер заглибний BRAUN asdf4321 BR-MQ5035-789 LG м. Житомир, пр. Миру, 40 0637208545	Видано	15.04.2026	Завершено: 20.04.2026 Видано: 22.04.2026	🔍 🖨 ✎ 🗑
106	Телевізор AIWA asdf4321 VF-VF391-12345 SAMSUNG м. Житомир, пр. Миру, 40 0976755522	Завершено	28.04.2026	Завершено: 14.05.2026	🔍 🖨 ✎ 🗑 ✅
105	Телевізор ERGO asdf4321 ERGO-43DF6500 Мельник Дмитро м. Житомир, вул. Небесної Сотні, 12 0971234567	Завершено	03.05.2026	Завершено: 16.05.2026	🔍 🖨 ✎ 🗑 ✅
104	Пилосос THOMAS sw14ff TH-AQUA-788562 Бондаренко Ірина м. Житомир, вул. Грушевського, 8 0661234567	Завершено	01.05.2026	Завершено: 15.05.2026	🔍 🖨 ✎ 🗑 ✅
103	Ваги KENWOOD asdf4321 KW-FPM250 LG м. Житомир, пр. Миру, 40 0637208545	В роботі	14.05.2026 39 дн.	-	🔍 🖨 ✎ 🗑
102	Кавомашина DELONGHI asdf4321 DL-ECAM-22110 Коваленко Наталія м. Житомир, вул. Перемоги, 10, кв. 5 0931112233	В роботі	12.05.2026 32 дн.	-	🔍 🖨 ✎ 🗑
101	Телевізор AIWA asdf4321 TV-AIWA-2024-001 Петренко Олександр м. Житомир, вул. Київська, 25 0671234567	В роботі	10.05.2026 34 дн.	-	🔍 🖨 ✎ 🗑
3	Пилосос CASTLE sw14ff asdfgh7543 SAMSUNG м. Житомир, пр. Миру, 40 0976755522	В роботі	17.04.2026 57 дн.	-	🔍 🖨 ✎ 🗑
2	Телефон APPLE Iphone 17 asd123gre567 LG м. Житомир, пр. Миру, 40 0637208545	В роботі	17.04.2026 57 дн.	-	🔍 🖨 ✎ 🗑
1	Телевізор AIWA asdf4321 q1w2e3r4t5y6u7i8 LG м. Житомир, пр. Миру, 40 0637208545	Видано	16.04.2026	Завершено: 17.04.2026 Видано: 12.06.2026	🔍 🖨 ✎ 🗑

Показано записи з 1 по 11 із 11 записів Перша Попередня 1 Наступна Остання

Рисунок 3.8 - Сторінка управління ремонтами

При натисканні кнопки «Перегляд» відкривається модальне вікно з повною інформацією про ремонт: статус, пристрій, бренд, модель, серійний номер, категорія, дати створення та завершення, контактні дані клієнта, опис несправності, комплектація та примітки. У нижній частині модального вікна розташовані кнопки друку квитанції, редагування, повторного ремонту та переходу до роботи з ремонтом Рис. 3.9.

Деталі ремонту #108

Статус:	Видано
Пристрій:	Машинка для підстригання волосся
Бренд:	VALERA
Модель:	1
Серійний номер:	VL-SWISS-5400
Категорія:	Платний
Дата створення:	10.04.2026
Номер для SMS:	0681234567
Клієнт:	Савченко Юлія
Телефон клієнта:	0681234567
Адреса клієнта:	м. Житомир, бул. Старий, 7
Дефект:	Перегрівається, вимикається через 2 хв
Комплектність:	Насадка-дифузор
Майстер:	master1
Ремонт виконано:	Так
Про ремонт:	Замінено термодатчик
Фінальна вартість роботи, грн:	1300
Вартість деталей, грн.:	400

Друквати квитанцію
Редагування ремонту
Повторний ремонт
Робота з ремонтом

1024-001
В роботі
10.05.2026
34 дн

Рисунок 3.9 - Модальне вікно деталей ремонту

Модальне вікно «Робота з ремонтом» дозволяє змінювати статус ремонту, вказувати фінальну вартість роботи та деталей, описувати виконаний ремонт. При зміні статусу на «Завершено» система автоматично формує SMS-повідомлення з налаштованого шаблону та надсилає його клієнту. При зміні на «Видано» ініціюється процедура OTP-верифікації: система генерує одноразовий код, надсилає його на телефон клієнта, а адміністратор вводить названий клієнтом код у форму підтвердження Рис. 3.10.

Робота з ремонтом #108

Статус:
Видано

Пристрій:
Машинка для
підстригання волосся

Бренд:
VALERA

Модель:
1

Майстер:
master1

Категорія:
Платний

Серійний №:
VL-SWISS-5400

Клієнт:
Савченко Юлія | 0681234567

Створено:
10.04.2026

Завершено:
18.04.2026

SMS на:
0681234567

Статус

<

В роботі

Завершено

Видано

☒  Відправляти SMS при зміні статусу

Фінальна вартість роботи, грн

1300

Вартість деталей, грн

400

Про ремонт

Замінено термодатчик

✓ Ремонт виконано

 Надіслати SMS

Закрити

Редагування

Зберегти

Рисунок 3.10 - Модальне вікно роботи з ремонтом

Форма створення нового ремонту складається з кількох логічних блоків. Блок клієнта дозволяє обрати існуючого клієнта із випадającego списку з пошуком або створити нового. При наявності партнера - клієнт прив'язується до партнерської організації. Блок пристрою реалізує ієрархічний вибір: при обранні категорії пристрою (наприклад, «Телевізор») через AJAX-запит динамічно завантажується список відповідних брендів, а при обранні бренду - список моделей. Якщо потрібний бренд або модель відсутні у довіднику, їх можна створити безпосередньо з форми ремонту за допомогою відповідних чекбоксів та текстових полів. Блок опису містить поля для серійного номера, опису несправності, комплектації, дати придбання та примітки. Блок вартості включає попередню вартість ремонту, вартість деталей, максимальну узгоджену вартість та вартість діагностики Рис. 3.11.

Додати новий ремонт

Пристрій Почніть вводити назву пристрою...	Бренд Почніть вводити назву бренду...	Модель Почніть вводити назву моделі...
Серійний номер Введіть серійний номер	Категорія ремонту Оберіть категорію	Дата продажу ДД.ММ.РРРР
Дефект Опишіть дефект	Комплектність Опис комплектації	Примітки Додаткова інформація
Вартість роботи 0.00	Максимальна вартість ремонту 0.00	Ціна діагностики 0.00

Майстер
☒ master1 ☐ Холодильники ☐ Телевізори ☐ Ткаченко ☐ Іваненко

Клієнт
☒ Клієнт ☐ Партнер

0501234567 ПІБ клієнта

Адреса клієнта (необов'язково)

Додаткова інформація (необов'язково)

Скасувати Додати

Рисунок 3.11 - Форма створення нового ремонту

Сторінка управління клієнтами відображає таблицю з переліком усіх клієнтів сервісного центру. Для кожного клієнта зазначено ім'я, номер телефону, адресу та додаткову інформацію. Доступні операції: додавання нового клієнта, редагування даних існуючого, перегляд історії ремонтів та замовлень. Пошук по таблиці здійснюється за іменем або номером телефону Рис 3.12.

Управління клієнтами Всього: 12 + Додати клієнта

Показати 10 записів Пошук:

ID	Ім'я	Телефон	Адреса	Інформація	Дії
108	Савченко Юлія	0681234567	м. Житомир, бул. Старий, 7		
107	Іваненко Максим	0991234567	м. Житомир, вул. Шевченка, 50	Гарантійний ремонт	
106	Ткаченко Оксана	0631234567	м. Житомир, вул. Лесі Українки, 33		
105	Мельник Дмитро	0971234567	м. Житомир, вул. Небесної Сотні, 12	VIP клієнт	
104	Бондаренко Ірина	0661234567	м. Житомир, вул. Грушевського, 8		
103	Шевченко Андрій	0501234567	м. Житомир, пр. Миру, 40	Корпоративний клієнт	
102	Коваленко Наталія	0931112233	м. Житомир, вул. Перемоги, 10, кв. 5		
101	Петренко Олександр	0671234567	м. Житомир, вул. Київська, 25	Постійний клієнт	
10	SAMSUNG	0976755522	м. Житомир, пр. Миру, 40	test	
9	LG	0637208545	м. Житомир, пр. Миру, 40	dfsd111	

Показано записи з 1 по 10 із 12 записів

Перша Попередня 1 2 Наступна Остання

Рисунок 3.12 - Сторінка управління клієнтами

Сторінка керування партнерами має аналогічну структуру та дозволяє

вести облік юридичних осіб, що направляють техніку на гарантійний або платний ремонт. Для кожного партнера зберігаються назва, контактний телефон, адреса та додаткова інформація.

Сторінка управління персоналом відображає список усіх зареєстрованих користувачів системи з їх ролями (Адміністратор, Менеджер, Майстер) та статусами (Active, Inactive). Адміністратор може додавати нових користувачів, редагувати існуючих та деактивувати облікові записи. Система забороняє деактивацію та видалення власного облікового запису, а менеджер не може редагувати облікові записи адміністраторів Рис. 3.13.

ID	Ім'я	Email	Телефон	Права	Статус	Дії
21	eyieadmin	v.shevel@gmail.com		Адміністратор	Active	✎ ✖
18	admin	admin@example.com	0501234568	Адміністратор	Active	✎ ✖
17	Іваненко	ivanenko@service.ua	0678888888	Майстер	Active	✎ ✖
16	Ткаченко	tkachenko@service.ua	0977777777	Майстер	Active	✎ ✖
15	Телевізори	televizory@service.ua	0679874563	Майстер	Active	✎ ✖
14	Холодильники	freese@service.ua	0674123720	Майстер	Active	✎ ✖
13	master1	master1@service.ua	0987777777	Майстер	Active	✎ ✖
11	manager	manager@service.ua	0987777777	Менеджер	Active	✎ ✖

Показано записи з 1 по 8 із 8 записів

Рисунок 3.13 - Сторінка управління персоналом

Друк квитанцій реалізовано у вигляді окремої сторінки, що відкривається у новому вікні браузера з оптимізованим для друку CSS-стилем. Квитанція про прийом містить номер ремонту, дату прийому, дані клієнта, опис пристрою та несправності, комплектацію та підписи. Квитанція про видачу додатково містить дату видачі, опис виконаного ремонту та вартість. Тексти квитанцій налаштовуються через інтерфейс адміністратора.

3.3 Тестування програмного комплексу

Тестування розробленого вебзастосунку здійснюється на кількох рівнях: автоматизоване unit-тестування окремих компонентів, статичний аналіз коду та функціональне тестування інтерфейсу. Комплексний підхід до тестування дозволяє

виявити різні категорії помилок та забезпечити високу якість програмного продукту [14; 15].

Автоматизоване unit-тестування реалізовано за допомогою фреймворку PHPUnit версії 10.5, що є стандартом галузі для тестування PHP-застосунків. Тестами покрито сервісний шар системи - компоненти, що реалізують найбільш критичну бізнес-логіку та взаємодіють із зовнішніми сервісами. Тести розташовані в директорії tests/Unit/Services і організовані відповідно до структури тестованих класів [15].

Клас SmsServiceTest містить тести для сервісу відправки SMS-повідомлень через SmsClub API. Тестуються наступні сценарії: успішна відправка повідомлення з коректними параметрами, обробка помилки при відсутності API-токена, обробка помилки при некоректному номері телефону, формування повідомлення з шаблону із підстановкою змінних, а також коректна обробка відповіді API з кодом помилки. Для ізоляції тестів від зовнішнього SMS-шлюзу використовуються mock-об'єкти, що імітують HTTP-відповіді API [16].

Клас ClientResolverTest перевіряє логіку визначення або створення клієнта при оформленні ремонту. Тестуються сценарії: створення нового клієнта при відсутності в базі, знаходження існуючого клієнта за номером телефону, створення клієнта з прив'язкою до партнерської організації, обробка випадку коли ні клієнт, ні партнер не вказані.

Клас RepairStatusServiceTest та OrderStatusServiceTest тестують логіку реакції системи на зміну статусів. Перевіряються сценарії: відправка SMS при переході у статус «Завершено», відправка SMS при переході у статус «Видано», відсутність відправки при переході в «В роботі», відсутність відправки коли статус не змінився, коректне визначення номера телефону для нотифікації при різних комбінаціях клієнт/партнер. Кожен тест використовує mock-об'єкти для ізоляції сервісного шару від бази даних та зовнішнього SMS-шлюзу, що забезпечує швидке та детерміноване виконання тестів. У (табл. 3.4) наведено результати unit тестування сервісів.

Результати unit-тестування

Тестовий клас	Тестів	Пройшло	Помилوک	Результат
SmsServiceTest	5	5	0	Passed
ClientResolverTest	4	4	0	Passed
RepairStatusServiceTest	5	5	0	Passed
OrderStatusServiceTest	4	4	0	Passed
Загалом	18	18	0	Passed

Статичний аналіз коду виконується за допомогою інструменту PHPStan версії 1.10, що аналізує PHP-код без його запуску та виявляє потенційні помилки: звернення до неіснуючих методів або властивостей, невідповідність типів параметрів, використання невизначених змінних, недосяжний код та інші проблеми. PHPStan налаштований на рівень 5 (з максимальних 9), що забезпечує оптимальний баланс між суворістю перевірок та кількістю хибнопозитивних спрацьовувань. Конфігурація інструменту зберігається у файлі `phpstan.neon` у кореневій директорії проєкту [14].

Запуск обох інструментів автоматизований через скрипти Composer: команда `composer test` виконує всі unit-тести PHPUnit, а команда `composer lint` запускає статичний аналіз PHPStan. Це дозволяє розробнику виконати повну перевірку якості коду однією командою перед кожним комітом.

Функціональне тестування інтерфейсу виконувалось вручну відповідно до визначених у розділі 2 варіантів використання. Для кожного варіанту використання було визначено тестові сценарії, що охоплюють основний потік, альтернативні потоки та граничні випадки. У (табл 3.5) наведено результати функціонального тестування основних сценаріїв.

Результати функціонального тестування

ID	Тестовий сценарій	Очікуваний результат	Результат
1	Авторизація з коректними даними	Перехід на Dashboard	Passed

Продовження табл. 3.5

2	Авторизація з некоректним паролем	Повідомлення про помилку	Passed
3	Створення ремонту з новим клієнтом	Ремонт створено, статус «В роботі»	Passed
4	Створення ремонту з новим брендом	Бренд створено, ремонт збережено	Passed
5	Зміна статусу на «Завершено»	SMS надіслано клієнту	Passed
6	ОТР-верифікація з правильним кодом	Ремонт видано, дата зафіксована	Passed
7	ОТР-верифікація з неправильним кодом	Повідомлення про помилку	Passed
8	ОТР-верифікація з простроченим кодом	Код відхилено (>10 хв)	Passed
9	Друк квитанції прийому	Квитанція відкрита для друку	Passed
10	Пошук клієнта за телефоном	Знайдено відповідний запис	Passed
11	Доступ майстра до налаштувань	Redirect на сторінку ремонтів	Passed
12	Спроба деактивації власного акаунту	Алерт з заборобою	Passed

Тестування безпеки включало перевірку захисту від основних вразливостей відповідно до стандарту OWASP Top Ten 2021. Перевірено захист від SQL-ін'єкцій шляхом введення спеціальних символів та SQL-конструкцій у поля форм - усі запити виконуються через prepared statements, що унеможлиблює ін'єкцію. Перевірено захист від CSRF-атак - кожна форма містить унікальний токен, що валідується на серверній стороні, а запити без коректного токена відхиляються. Перевірено зберігання паролів - у базі даних зберігаються лише bcrypt-хеші з cost factor 12, оригінальні паролі ніде не зберігаються. Перевірено захист API-ключів - токен SmsClub зберігається у файлі .env, який виключено з репозиторію через .gitignore [20; 21].

За результатами комплексного тестування всі 18 unit-тестів пройдені

успішно, статичний аналіз PHPStan не виявив критичних помилок, всі 12 функціональних тестових сценаріїв підтвердили коректність роботи системи, а перевірка безпеки не виявила вразливостей із переліку OWASP Top Ten 2021. Це дозволяє зробити висновок про готовність розробленого програмного комплексу до впровадження у виробниче середовище.

Висновки до розділу 3

У третьому розділі описано порядок розгортання системи, наведено детальний опис інтерфейсу користувача та представлено результати комплексного тестування програмного комплексу.

У підрозділі 3.1 описано процедуру розгортання системи з використанням Docker та Docker Compose, що включає три сервіси (app на базі php:8.2-apache, db на базі mysql:8.0, phpmyadmin). Наведено покрокову інструкцію розгортання: клонування репозиторію, створення файлу конфігурації .env, збирання образів та запуск контейнерів, виконання ідемпотентного скрипту початкового налаштування demo-setup.php для міграцій та створення облікового запису адміністратора. Описано два режими роботи (local та production), механізм автоматичного резервного копіювання бази даних та систему міграцій із дванадцяти файлів.

У підрозділі 3.2 наведено детальний опис інтерфейсу публічного сайту (шість секцій: героїчний блок, послуги, про нас, партнери, контакти, футер) та адміністративної панелі (тридцять три сторінки). Описано рольову модель доступу з трьома рівнями (адміністратор, менеджер, майстер) із таблицею розподілу прав. Представлено ключові сторінки системи: Dashboard зі зведеною статистикою, сторінку управління ремонтами з фільтрами та інтерактивною таблицею DataTables, модальні вікна деталей та роботи з ремонтом, форму створення ремонту з ієрархічним вибором пристрою через AJAX, сторінки управління клієнтами та персоналом, а також модуль друку квитанцій.

У підрозділі 3.3 представлено результати комплексного тестування на

трьох рівнях. Автоматизоване unit-тестування за допомогою PHPUnit 10.5 охопило сервісний шар системи: вісімнадцять тестів у чотирьох тестових класах (SmsServiceTest, ClientResolverTest, RepairStatusServiceTest, OrderStatusServiceTest) - усі пройдені успішно. Статичний аналіз PHPStan на рівні 5 не виявив критичних помилок. Функціональне тестування за дванадцятьма сценаріями (авторизація, створення ремонту, зміна статусу, OTP-верифікація, друк квитанцій, рольовий доступ) підтвердило коректність роботи системи. Перевірка безпеки за стандартом OWASP Top Ten 2021 не виявила вразливостей. Результати тестування підтверджують готовність системи до впровадження..

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-практичну задачу розробки вебзастосунку для адміністрування роботи сервісного центру з ремонту побутової техніки. Розроблений програмний комплекс забезпечує автоматизацію ключових бізнес-процесів підприємства та відповідає усім визначеним функціональним і нефункціональним вимогам. У ході виконання роботи отримано наступні результати.

У першому розділі проведено аналіз предметної галузі сервісного обслуговування побутової техніки та визначено дев'ять ключових бізнес-процесів, що потребують автоматизації. Виконано порівняльний аналіз трьох існуючих комерційних рішень (RO App, Orderry, BytePhase) за шістнадцятьма критеріями, який показав, що жодна з існуючих систем не задовольняє одночасно всіх специфічних вимог досліджуваного підприємства: розгортання на власному сервері без абонентської плати, поєднання адміністративного модуля з публічним вебсайтом, обліку партнерських організацій та OTP-верифікації видачі. Обґрунтовано вибір архітектурного патерну MVC з розширеним сервісним шаром та визначено технологічний стек: PHP 8.2, MySQL 8.0, Apache 2.4, Bootstrap, DataTables, SmsClub API, Docker, PHPUnit, PHPStan.

У другому розділі специфіковано сімнадцять варіантів використання для чотирьох акторів системи та побудовано діаграму варіантів використання у нотації UML 2.0. Спроектовано реляційну базу даних із вісімнадцяти таблиць з дотриманням третьої нормальної форми та системою з дванадцятьма міграціями. Описано алгоритми роботи ключових модулів: створення ремонту з динамічним довідником пристроїв, зміна статусів із автоматичними SMS-нотифікаціями, OTP-верифікація видачі з генерацією чотиризначного коду та обмеженням дії у десять хвилин. Реалізовано програмний комплекс, що включає дев'ятнадцять контролерів, двадцять моделей, п'ять сервісів, тридцять три адміністративні сторінки та публічний сайт, із системою безпеки на кількох рівнях.

У третьому розділі описано процедуру розгортання системи через Docker та Docker Compose із трьома сервісами, що забезпечує відтворюваність середовища на будь-якому сервері. Наведено детальний опис інтерфейсу адміністративної панелі та публічного сайту з рольовою моделлю доступу для трьох рівнів (адміністратор, менеджер, майстер). Виконано комплексне тестування: вісімнадцять unit-тестів PHPUnit (усі пройдені успішно), статичний аналіз PHPStan на рівні 5 (без критичних помилок), дванадцять функціональних тестових сценаріїв (усі підтверджено) та перевірка безпеки за стандартом OWASP Top Ten 2021 (вразливостей не виявлено).

Практичне значення отриманих результатів полягає у створенні повнофункціонального вебзастосунку, готового до впровадження у реальне підприємство сфери сервісного обслуговування побутової техніки. Розроблений програмний комплекс може бути використаний як типове рішення для аналогічних сервісних центрів із мінімальною адаптацією.

Напрямами подальшого вдосконалення системи є розробка мобільного додатку для майстрів із можливістю оновлення статусів ремонтів з мобільного пристрою, інтеграція модуля управління складом запасних частин із відстеженням залишків та автоматичним формуванням заявок на поповнення, реалізація фінансового модуля для обліку доходів та витрат, а також впровадження аналітичного дашборду із графіками та трендами для прийняття управлінських рішень.

Напрямами подальшого вдосконалення системи є розробка мобільного додатку для майстрів із можливістю оновлення статусів ремонтів з мобільного пристрою, інтеграція модуля управління складом запасних частин із відстеженням залишків та автоматичним формуванням заявок на поповнення, реалізація фінансового модуля для обліку доходів та витрат, а також впровадження аналітичного дашборду із графіками та трендами для прийняття управлінських рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про інформацію» від 02.10.1992 № 2657-XII. Дата оновлення: 03.07.2020. URL: <https://zakon.rada.gov.ua/laws/show/2657-12> (дата звернення: 10.04.2026).

2. Закон України «Про захист персональних даних» від 01.06.2010 № 2297-VI. Дата оновлення: 16.11.2020. URL: <https://zakon.rada.gov.ua/laws/show/2297-17> (дата звернення: 10.04.2026).

3. ДСТУ ISO/IEC/IEEE 15288:2016. Інженерія систем і програмного забезпечення. Процеси життєвого циклу систем. Київ, 2016. 84 с.

4. Pitt C. Pro PHP 8 MVC: Model View Controller Architecture-Driven Application Development. 2nd ed. Apress, 2021. 388 p.

5. Lockhart J. Modern PHP: New Features and Good Practices. O'Reilly Media, 2015. 268 p.

6. PHP: Hypertext Preprocessor. Official Documentation. PHP 8.1. URL: <https://www.php.net/manual/en/index.php> (дата звернення: 15.04.2026).

7. Composer: Dependency Manager for PHP. Official Documentation. URL: <https://getcomposer.org/doc/> (дата звернення: 15.04.2026).

8. Smirnova S., Tezuysal A. MySQL Cookbook: Solutions for Database Developers and Administrators. 4th ed. O'Reilly Media, 2022. 938 p.

9. Tezuysal A., Ahmed I. Database Design and Modeling with PostgreSQL and MySQL. Packt Publishing, 2024. 222 p.

10. MySQL 8.0 Reference Manual. Oracle Corporation. URL: <https://dev.mysql.com/doc/refman/8.0/en/> (дата звернення: 12.04.2026).

11. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. 395 p.

12. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.

13. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley

Professional, 2002. 533 p.

14. PHPUnit Manual. Sebastian Bergmann. Version 10. URL: <https://docs.phpunit.de/en/10.0/> (дата звернення: 18.04.2026).

15. PHPStan: PHP Static Analysis Tool. Official Documentation. URL: <https://phpstan.org/user-guide/getting-started> (дата звернення: 18.04.2026).

16. Meszaros G. xUnit Test Patterns: Refactoring Test Code. Addison-Wesley, 2007. 883 p.

17. Schenker G. N. The Ultimate Docker Container Book. 3rd ed. Packt Publishing, 2023. 626 p.

18. Docker Documentation. Docker Inc. URL: <https://docs.docker.com/> (дата звернення: 20.04.2026).

19. Docker Compose Reference. Docker Inc. URL: <https://docs.docker.com/compose/> (дата звернення: 20.04.2026).

20. OWASP Top Ten 2021. Open Web Application Security Project. URL: <https://owasp.org/www-project-top-ten/> (дата звернення: 22.04.2026).

21. Stuttard D., Pinto M. The Web Application Hacker's Handbook. 2nd ed. Wiley, 2011. 912 p.

22. DataTables - Table plug-in for jQuery. SpryMedia Ltd. URL: <https://datatables.net/manual/> (дата звернення: 14.04.2026).

23. Flanagan D. JavaScript: The Definitive Guide. 7th ed. O'Reilly Media, 2020. 706 p.

24. SmsClub API Documentation. SmsClub Ukraine. URL: <https://im.smsclub.mobi/documentation> (дата звернення: 16.04.2026).

25. Richardson L., Amundsen M. Ruby S. RESTful Web APIs: Services for a Changing World. O'Reilly Media, 2013. 408 p.

26. 10 Best Repair Shop Management Software for Small Businesses. CRMJIO Blog. 2026. URL: <https://crmjio.com/blog/10-best-repair-shop-management-software-for-small-businesses-2026-guide> (дата звернення: 01.05.2026).

27. RepairShopr - Computer Repair Shop Software. URL: <https://www.repairshopr.com/> (дата звернення: 01.05.2026).

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

29. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd ed. Prentice Hall, 2004. 736 p.

30. UML 2.5 Specification. Object Management Group. URL: <https://www.omg.org/spec/UML/2.5.1/> (дата звернення: 05.04.2026).

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 26 – ІІЗ	
						85

Лістинг класу SmsService

Клас SmsService реалізує взаємодію з SMS-шлюзом SmsClub API для відправки повідомлень клієнтам, формування повідомлень із налаштовуваних шаблонів, генерацію та перевірку OTP-кодів для верифікації видачі, а також перевірку статусів відправлених повідомлень та балансу.

```
<?php

namespace App\Services;

use App\Core\Database;
use App\Models\Setting;

class SmsService {
    private $db;
    private $settingModel;

    public function __construct() {
        $this->db = new Database();
        try {
            $this->settingModel = new Setting();
        } catch (\Throwable $e) {
            $this->settingModel = null;
        }
    }

    public static function defaultTemplates(): array {
        return [
            'sms_template_repair_completed' => [
                'label' => 'Ремонт: готово до видачі',
                'description' => 'Плейсхолдери: {repair_id}, {device}, {final_cost}, {days_to_pickup}',
                'template' => '{device} готовый (-а) до видачі. Вартість {final_cost} грн. Зберігання {days_to_pickup} діб',
            ],
            'sms_template_repair_issued' => [
                'label' => 'Ремонт: видано',
```

```

        'description'    =>    'Плейсхолдери:    {repair_id},
{device}',
        'template' => 'Ваш пристрій {device} видано. Дякуємо,
що обрали нас!'
    ],
    'sms_template_repair_reminder' => [
        'label' => 'Ремонт: нагадування забрати',
        'description' => 'Плейсхолдери: {repair_id}, {device},
{days_to_pickup}',
        'template' => 'Ваш пристрій (ремонт #{repair_id})
готовий вже більше {days_to_pickup} днів. Будь ласка, заберіть його.'
    ],
    'sms_template_order_completed' => [
        'label' => 'Замовлення: готово до видачі',
        'description' => 'Плейсхолдери: {order_id}, {device}',
        'template' => 'Замовлення #{order_id} ({device})
прибуло та готове до видачі.'
    ],
    'sms_template_order_issued' => [
        'label' => 'Замовлення: видано',
        'description' => 'Плейсхолдери: {order_id}, {device}',
        'template' => 'Замовлення #{order_id} ({device})
видано. Дякуємо, що обрали нас!'
    ],
    'sms_template_verification_code' => [
        'label' => 'Код підтвердження видачі',
        'description' => 'Плейсхолдери: {repair_id}, {code}',
        'template' => 'Ваш код підтвердження: {code}'
    ],
];
}

/**
 * Send an SMS message.
 * Currently mocks the sending process and logs to DB.
 *
 * @param string $phone
 * @param string $message
 * @param int|null $repairId

```

```

    * @param string $type ('manual', 'automated', 'auth_code')
    * @return bool
    */
/**
 * Нормалізує український номер телефону до формату 380XXXXXXXXX.
 *
 * '+38(050) 111-22-33' → '380501112233' (12 цифр)
 * '0501112233' → '380501112233' (10 цифр з 0)
 * '501112233' → '380501112233' (9 цифр без коду
оператора)
 * 'whatever' → 'whatever_digits' (все що інше - лише
видаляємо нецифри)
 */
public static function normalizeUaPhone(string $phone): string
{
    $digits = preg_replace('/\D/', '', $phone) ?? '';
    if (strlen($digits) === 10 && str_starts_with($digits, '0')) {
        return '38' . $digits;
    }
    if (strlen($digits) === 9) {
        return '380' . $digits;
    }
    return $digits;
}

public function send($phone, $message, $repairId = null, $type =
'manual', $orderId = null) {
    $cleanPhone = self::normalizeUaPhone($phone);

    $token = defined('SMS_API_TOKEN') ? SMS_API_TOKEN : '';
    $sender = defined('SMS_SENDER_NAME') ? SMS_SENDER_NAME :
'Tandem';

    // If token is missing or default, just log and return false
(or true to not break flow in dev)
    if (empty($token) || $token === 'YOUR_TOKEN_HERE' || $token ===
'YOUR_SMS_API_TOKEN') {
        $this->log($phone, "[MOCK] $message", 'mock_sent',
$repairId, $type, null, $orderId);
        return true;
    }
}

```

```

$url = 'https://im.smsclub.mobi/sms/send';
$data = [
    'phone' => [$cleanPhone],
    'message' => $message,
    'src_addr' => $sender
];

$ch = curl_init($url);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($data));
curl_setopt($ch, CURLOPT_HTTPHEADER, [
    'Content-Type: application/json',
    'Authorization: Bearer ' . $token
]);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$response = curl_exec($ch);
$httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
$curlError = curl_error($ch);
curl_close($ch);

$status = 'failed';
$smsId = null;

if ($httpCode >= 200 && $httpCode < 300 && !$curlError) {
    $responseData = json_decode($response, true);

    // Validate if success_request exists
    if (isset($responseData['success_request']['info'])) {
        $status = 'sent';
        // The API returns: {"success_request": {"info":
{"106": "380989361131"}}}
        // So Key = SMS_ID, Value = PHONE
        foreach ($responseData['success_request']['info'] as
$id => $ph) {
            // Check if this ID corresponds to our phone number
            // API might return phone without +, so compare
carefully

            if ($ph == $cleanPhone || $ph == $phone) {

```

```

        $smsId = $id;
        break;
    }
}

// Fallback: if only one ID returned and we sent one
SMS, assume it's ours
    if (! $smsId &&
count($responseData['success_request']['info']) == 1) {
        $smsId =
array_key_first($responseData['success_request']['info']);
    }

} else {
    // Maybe partial success or error
    error_log("SmsService API logic warning: " .
$response);
    // If http code is 200 but structure is weird, we
still mark as sent to not block flow?
    // No, if no success info, let's keep it as failed or
'unknown'
    if(isset($responseData['success_request'])) {
        $status = 'sent'; // Partial success maybe?
    }
}
} else {
    error_log("SmsService API Error: Code=$httpCode
Err=$curlError Resp=$response");
}

$this->log($phone, $message, $status, $repairId, $type, $smsId,
$orderId);

return $status === 'sent';
}

public function buildMessage(string $templateKey, string
$defaultTemplate, array $variables = []): string {
    $template = $defaultTemplate;

    if ($this->settingModel instanceof Setting) {

```

```

try {
    $saved = $this->settingModel->get($templateKey, null);
    if ($saved !== null && trim((string)$saved) !== '') {
        $template = (string)$saved;
    }
} catch (\Throwable $e) {
    $template = $defaultTemplate;
}

return $this->renderTemplate($template, $variables);
}

private function renderTemplate(string $template, array
$variables): string {
    $replace = [];
    foreach ($variables as $key => $value) {
        $replace['{' . $key . '}'] = (string)$value;
    }
    return strtr($template, $replace);
}

private function log($phone, $message, $status, $repairId, $type,
$smsId = null, $orderId = null) {
    try {
        $sql = "INSERT INTO sms_logs (phone, message, status,
repair_id, order_id, type, sms_id) VALUES (?, ?, ?, ?, ?, ?, ?)";
        $this->db->query($sql, [$phone, $message, $status,
$repairId, $orderId, $type, $smsId]);
    } catch (\Exception $e) {
        // Backward compatibility for old schema without order_id.
        try {
            $sqlOld = "INSERT INTO sms_logs (phone, message,
status, repair_id, type, sms_id) VALUES (?, ?, ?, ?, ?, ?)";
            $this->db->query($sqlOld, [$phone, $message, $status,
$repairId, $type, $smsId]);
        } catch (\Exception $e2) {
            // Silently fail or log to file, but don't crash the
app
            error_log("SmsService Error: Failed to log SMS. " .
$e2->getMessage());
        }
    }
}

```

```

    }
}

public function checkStatuses() {
    // Get all pending messages (sent but not final status)
    // Adjust statuses to exclude final ones like DELIVRD, UNDELIV,
EXPIRED
    // Also check if 'sent' matches what we insert.
    $sql = "SELECT id, sms_id, status FROM sms_logs WHERE status
IN ('sent', 'ENROUTE') AND sms_id IS NOT NULL AND sms_id != '' AND created_at
> DATE_SUB(NOW(), INTERVAL 3 DAY)";
    $rows = $this->db->fetchAll($sql);

    if (empty($rows)) {
        error_log("SmsService: No pending SMS found to check
status.");
        return 0; // Nothing to update
    }

    $smsIds = array_column($rows, 'sms_id');
    $smsMap = [];
    foreach ($rows as $row) {
        // Force string key
        $smsMap[(string)$row['sms_id']] = $row['id'];
    }

    error_log("SmsService: Checking status for IDs: " .
implode(',', $smsIds));

    // Chunk requests to avoid too large payloads (e.g. 100 at a
time)
    $chunks = array_chunk($smsIds, 100);
    $updatedCount = 0;

    $token = defined('SMS_API_TOKEN') ? SMS_API_TOKEN : '';
    if (empty($token)) return 0;

    foreach ($chunks as $chunkIds) {
        $url = 'https://im.smsclub.mobi/sms/status';
        $data = ['id_sms' => $chunkIds];
    }
}

```

```

$ch = curl_init($url);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($data));
curl_setopt($ch, CURLOPT_HTTPHEADER, [
    'Content-Type: application/json',
    'Authorization: Bearer ' . $token
]);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$response = curl_exec($ch);
$httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
curl_close($ch);

error_log("SmsService Status Response [$httpCode]: " .
$response);

if ($httpCode >= 200 && $httpCode < 300) {
    $statusData = json_decode($response, true);

    // Handle different response structures if needed
    // Structure 1: {"123": "DELIVRD", ...}
    // Structure 2: {"success_request": {"info": {"123":
"DELIVRD"}}}

    $finalStatusMap = [];
    if (isset($statusData['success_request']['info'])) {
        $finalStatusMap =
$statusData['success_request']['info'];
    } elseif (is_array($statusData) &&
!isset($statusData['success_request'])) {
        $finalStatusMap = $statusData;
    }

    if (!empty($finalStatusMap)) {
        foreach ($finalStatusMap as $sId => $newStatus) {
            // Update DB if ID exists in our map
            // Ensure key is string comparison
            $sIdStr = (string)$sId;

            if (isset($smsMap[$sIdStr])) {
                $dbId = $smsMap[$sIdStr];
            }
        }
    }
}

```

```

// Only update if status changed
// Actually, just update always to match
remote
    $this->db->query("UPDATE sms_logs SET
status = ? WHERE id = ?", [$newStatus, $dbId]);
    $updatedCount++;
}
}
}
}
}

return $updatedCount;
}

public function getHistory($repairId) {
    $sql = "SELECT * FROM sms_logs WHERE repair_id = ? ORDER BY
created_at DESC";
    return $this->db->query($sql, [$repairId]);
}

public function getOrderHistory($orderId) {
    $sql = "SELECT * FROM sms_logs WHERE order_id = ? ORDER BY
created_at DESC";
    return $this->db->query($sql, [$orderId]);
}

public function getBalance() {
    $token = defined('SMS_API_TOKEN') ? SMS_API_TOKEN : '';
    if (empty($token) || $token === 'YOUR_TOKEN_HERE' || $token ===
'YOUR_SMS_API_TOKEN') {
        return [
            'success' => false,
            'error' => 'SMS token is missing',
        ];
    }

    $url = 'https://im.smsclub.mobi/sms/balance';
    $ch = curl_init($url);
    curl_setopt($ch, CURLOPT_HTTPGET, true);
    curl_setopt($ch, CURLOPT_HTTPHEADER, [
        'Authorization: Bearer ' . $token

```

```

]);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$response = curl_exec($ch);
$httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
$curlError = curl_error($ch);
curl_close($ch);

if ($httpCode >= 200 && $httpCode < 300 && !$curlError) {
    $responseData = json_decode($response, true);
    if (isset($responseData['success_request']['info'])) {
        return [
            'success' => true,
            'data'
=> $responseData['success_request']['info'],
        ];
    }
    return [
        'success' => false,
        'error' => 'Unexpected response structure',
        'raw' => $response
    ];
}

return [
    'success' => false,
    'error' => $curlError ?: 'HTTP ' . $httpCode,
    'raw' => $response
];
}

public function generateVerificationCode($repairId) {
    $code = rand(1000, 9999);
    // Expire in 10 minutes
    $expiresAt = date('Y-m-d H:i:s', strtotime('+10 minutes'));

    // Deactivate previous unused codes for this repair
    $this->db->query("UPDATE verification_codes SET is_used = 1
WHERE repair_id = ? AND is_used = 0", [$repairId]);

    $sql = "INSERT INTO verification_codes (repair_id, code,

```

```

        expires_at) VALUES (?, ?, ?)";
        $this->db->query($sql, [$repairId, $code, $expiresAt]);

        return $code;
    }

    public function verifyCode($repairId, $code) {
        $sql = "SELECT id FROM verification_codes
                WHERE repair_id = ? AND code = ? AND is_used = 0 AND
expires_at > NOW()
                ORDER BY id DESC LIMIT 1";
        $result = $this->db->query($sql, [$repairId, $code]);

        if ($result->num_rows > 0) {
            $row = $result->fetch_assoc();
            // Mark as used
            $this->db->query("UPDATE verification_codes SET is_used =
1 WHERE id = ?", [$row['id']]);
            return true;
        }
        return false;
    }

    private function mapOrderVerificationId(int $orderId): int
    {
        return -1 * abs($orderId);    }

    public function generateOrderVerificationCode(int $orderId) {
        return $this->generateVerificationCode($this->
mapOrderVerificationId($orderId));
    }

    public function verifyOrderCode(int $orderId, string $code): bool
    {
        return $this->verifyCode($this->
mapOrderVerificationId($orderId), $code);
    }
}

```

Лістинг класу RepairStatusService

Клас RepairStatusService інкапсулює логіку реакції на зміну статусу ремонту: визначає номер телефону для нотифікації, формує повідомлення зі шаблону залежно від нового статусу та надсилає SMS через SmsService.

```
<?php

namespace App\Services;

use App\Core\Logger;
use App\Core\OrderStatus;
use App\Models\Repair;

/**
 * Інкапсулює реакцію на зміну статусу ремонту:
 * - визначає номер телефону для нотифікації;
 * - формує повідомлення зі шаблону залежно від нового статусу
 *   (completed - з фінальною вартістю та строком зберігання,
 *   issued - простіше, без вартості);
 * - надсилає SMS через SmsService.
 *
 * Раніше ця логіка дублювалась у:
 * - RepairController::edit (status change → SMS)
 * - RepairController::markAsIssued (на видачу)
 * - admin/scripts/ajax/save-issue-repair.php
 */
class RepairStatusService
{
    private SmsService $smsService;
    private NotificationPhoneResolver $phoneResolver;

    public function __construct(?SmsService $smsService = null,
        ?NotificationPhoneResolver $phoneResolver = null)
    {
        $this->smsService = $smsService ?? new SmsService();
        $this->phoneResolver = $phoneResolver ?? new
NotificationPhoneResolver();
    }
}
```

```

/**
 * Надсилає SMS-нотифікацію якщо потрібно.
 *
 * @param int    $repairId    ID ремонту
 * @param int    $oldStatusId попередній status_id
 * @param int    $newStatusId новий status_id
 * @param array  $repair      запис ремонту (для
name/phone/device_title)
 * @param array  $costData    ['finish_repair_cost' => float,
'detail_cost' => float]
 *
 *                                - для completed-шаблону рахуємо суму.
Якщо
 *                                відсутні - рахуємо 0.
 * @param array  $options    ['enable' => bool] - за замовчуванням
true.
 *
 * @return array{sent: bool, reason: ?string, status_changed: bool}
 */
public function sendStatusSms(
    int $repairId,
    int $oldStatusId,
    int $newStatusId,
    array $repair,
    array $costData = [],
    array $options = []
): array {
    $enable = $options['enable'] ?? true;
    $statusChanged = $newStatusId !== $oldStatusId;

    $result = [
        'sent' => false,
        'reason' => null,
        'status_changed' => $statusChanged,
    ];
    if (!$enable) {
        $result['reason'] = 'sms_disabled';
        return $result;
    }
    if (!$statusChanged) {
        $result['reason'] = 'status_unchanged';
        return $result;
    }
}

```

```

    }
    if (!in_array($newStatusId, [OrderStatus::COMPLETED,
OrderStatus::ISSUED], true)) {
        $result['reason'] = 'status_not_notifiable';
        return $result;
    }

    try {
        $targetPhone = $this->phoneResolver->resolve($repair);
        if ($targetPhone === '') {
            $result['reason'] = 'no_phone';
            return $result;
        }

        $deviceTitle = trim((string)($repair['device_title'] ??
'Пристрій'));

        if ($newStatusId === OrderStatus::COMPLETED) {
            $finalCost = ((float)($costData['finish_repair_cost']
?? 0))
                + ((float)($costData['detail_cost'] ?? 0));
            $message = $this->smsService->buildMessage(
                'sms_template_repair_completed',
                '{device} готовий (-а) до видачі. Вартість
{final_cost} грн. Зберігання {days_to_pickup} діб',
                [
                    'repair_id' => $repairId,
                    'device' => $deviceTitle,
                    'final_cost' => number_format($finalCost,
2, '.', ''),
                    'days_to_pickup' =>
(string)Repair::DAYS_TO_PICKUP,
                ]
            );
        } else { // ISSUED
            $message = $this->smsService->buildMessage(
                'sms_template_repair_issued',
                'Ваш пристрій {device} видано. Дякуємо, що обрали
нас!',
                [
                    'repair_id' => $repairId,

```

```

        'device'      => $deviceTitle,
    ];
    );
}

$this->smsService->send($targetPhone, $message, $repairId,
'automated');

$result['sent'] = true;
return $result;
} catch (\Throwable $e) {
    Logger::exception($e, 'RepairStatusService SMS failed', [
        'repair_id'      => $repairId,
        'old_status_id' => $oldStatusId,
        'new_status_id' => $newStatusId,
    ]);
    $result['reason'] = 'exception';
    return $result;
}
}
}

```

Лістинг класу ClientResolver

Клас ClientResolver інкапсулює логіку визначення або створення клієнта при оформленні ремонту чи замовлення з урахуванням партнерського контексту.

```
<?php

namespace App\Services;

use App\Models\Client;
use App\Models\Partner;

/**
 * Інкапсулює спільну логіку обробки даних клієнта/партнера з форм
 * замовлень та ремонтів.
 *
 * Раніше дублювалося як приватний метод `resolveClientId` у
 * OrderController і RepairController.
 *
 * Поведінка:
 * - Якщо обрано партнера ($partnerId > 0) - перевіряє, що такий є у
БД.
 * - Якщо галочка `create_new_client` і вказано ім'я або телефон -
 * шукає існуючого клієнта за телефоном/іменем; якщо не знайдено -
 * створює нового. Знайденого клієнта пов'язує з партнером (якщо є)
 * і оновлює його профіль із форми.
 * - Якщо в формі обрано існуючого клієнта (`client_id`) - пов'язує з
 * партнером (якщо є) і оновлює його профіль.
 * - Якщо є тільки партнер - повертає client_id=null.
 * - У всіх інших випадках - помилка з рекомендацією, що зробити.
 */
class ClientResolver
{
    private Client $clientModel;
    private Partner $partnerModel;

    public function __construct(?Client $clientModel = null, ?Partner
$clientModel = null)
```

```

{
    $this->clientModel = $clientModel ?? new Client();
    $this->partnerModel = $partnerModel ?? new Partner();
}

/**
 * Повертає або ['client_id' => int|null], або ['error' => string].
 *
 * @param array $post Дані форми ($_POST).
 * @param int $partnerId Обраний партнер (0, якщо не обрано).
 */
public function resolve(array $post, int $partnerId): array
{
    $clientId      = $post['client_id'] ?? '';
    $clientName     = trim((string)($post['client_name'] ?? ''));
    $clientPhone    = trim((string)($post['client_phone'] ?? ''));
    $clientAddress  = $post['client_address'] ?? '';
    // Підтримка legacy-поля extra_info зі старих форм замовлень.
    $clientInfo     = $post['client_info'] ?? ($post['extra_info']
?? '');

    $createNewClient = isset($post['create_new_client']);

    // Profile update - лише з ключів, які реально присутні у формі,
    // щоб не затирати поля, яких немає на сторінці.
    $profileUpdate = [];
    if (array_key_exists('client_name', $post))
$profileUpdate['name'] = $clientName;
    if (array_key_exists('client_phone', $post))
$profileUpdate['phone'] = $clientPhone;
    if (array_key_exists('client_address', $post))
$profileUpdate['address'] = $clientAddress;
    if (array_key_exists('client_info', $post))
$profileUpdate['client_info'] = $clientInfo;

    if ($partnerId > 0) {
        $partner = $this->partnerModel->getById($partnerId);
        if (empty($partner)) {
            return ['error' => 'Партнера не знайдено'];
        }
    }

    $hasClientPayload = !empty($clientId)

```

```

        || $createNewClient
        || $clientName !== ''
        || $clientPhone !== ''
        || trim((string)$clientAddress) !== ''
        || trim((string)$clientInfo) !== '';

        // Гілка: створення нового клієнта (але можемо знайти існуючого
по телефону/імені).
        if ($createNewClient && ($clientName !== '' || $clientPhone !==
'')) {
            $existingClients = $this->clientModel->search($clientPhone
?: $clientName);
            $found = false;

            foreach ($existingClients as $client) {
                if (!empty($clientPhone) && $client['phone'] ===
$clientPhone) {
                    $clientId = $client['id'];
                    $found = true;
                    break;
                } elseif (!empty($clientName)
&& strtolower(trim($client['name'])) ===
strtolower(trim($clientName))) {
                    $clientId = $client['id'];
                    $found = true;
                    break;
                }
            }

            if (!$found) {
                $payload = [
                    'name' => $clientName,
                    'phone' => $clientPhone,
                    'address' => $clientAddress,
                    'client_info' => $clientInfo,
                ];

                if ($partnerId > 0) {
                    $payload['partner_id'] = $partnerId;
                }
                $clientId = $this->clientModel-
>createAndReturnId($payload);

```

```

        if (!$clientId) {
            return ['error' => 'Помилка при створенні
клієнта'];
        }
    } else {
        if ($partnerId > 0) {
            $this->clientModel-
>updatePartnerId((int)$clientId, $partnerId);
        }
        if (!empty($profileUpdate)) {
            $this->clientModel->updateProfile((int)$clientId,
$profileUpdate);
        }
    }

    return ['client_id' => (int)$clientId];
}

// Гілка: вибрано існуючого клієнта зі списку.
if (!empty($clientId)) {
    if ($partnerId > 0) {
        $this->clientModel->updatePartnerId((int)$clientId,
$partnerId);
    }
    if (!empty($profileUpdate)) {
        $this->clientModel->updateProfile((int)$clientId,
$profileUpdate);
    }
    return ['client_id' => (int)$clientId];
}
// Лише партнер обраний (без клієнта) - це валідний випадок.
if ($partnerId > 0) {
    return ['client_id' => null];
}

if ($hasClientPayload) {
    return ['error' => 'Оберіть існуючого клієнта або увімкніть
створення нового клієнта'];
}
return ['error' => 'Оберіть клієнта або партнера'];
}
}

```

Лістинг класу Database

Клас Database реалізує обгортку над розширенням MySQLi з підтримкою prepared statements, транзакцій та уніфікованим інтерфейсом для SELECT, INSERT, UPDATE та DELETE операцій.

```
<?php

namespace App\Core;

class Database {
    private $host;
    private $user;
    private $password;
    private $database;
    private $conn;

    public function __construct() {
        require_once __DIR__ . '/../../config/config.php';

        $this->host = DB_HOST;
        $this->user = DB_USER;
        $this->password = DB_PASSWORD;
        $this->database = DB_NAME;

        $hostCandidates = [$this->host];
        if ($this->host === 'localhost') {
            // In Docker, localhost often points to PHP container and
            socket is missing.
            $hostCandidates[] = 'db';
            $hostCandidates[] = '127.0.0.1';
        }
        $hostCandidates=array_values(array_unique($hostCandidates));
        $lastError = '';
        foreach ($hostCandidates as $candidateHost) {
            try {
                $conn = new \mysqli($candidateHost, $this->user, $this->password, $this->database);
                if (!$conn->connect_error) {
```

```

        $this->host = $candidateHost;
        $this->conn = $conn;
        break;
    }
    $lastError = $conn->connect_error;
} catch (\mysqli_sql_exception $e) {
    $lastError = $e->getMessage();
}
}

if (!$this->conn) {
    throw new \Exception(
        "Database Connection failed for hosts [" . implode(' ', $hostCandidates) . "]: " . $lastError
    );
}

$this->conn->set_charset("utf8");
}

public function getConnection() {
    return $this->conn;
}

public function getError() {
    return $this->conn->error;
}

public function getLastInsertId() {
    return $this->conn->insert_id;
}

public function beginTransaction() {
    return $this->conn->begin_transaction();
}

public function commit() {
    return $this->conn->commit();
}

public function rollback() {
    return $this->conn->rollback();
}

```

```

public function query($sql, $params = []) {
    try {
        $stmt = $this->conn->prepare($sql);

        if ($stmt === false) {
            throw new \Exception('Failed to prepare statement: ' .
$this->conn->error . ' SQL: ' . $sql);
        }

        if (!empty($params)) {
            $types = '';
            foreach ($params as $param) {
                if (is_int($param)) {
                    $types .= 'i';
                } elseif (is_float($param)) {
                    $types .= 'd';
                } elseif ($param === null) {
                    $types .= 's'; // Treat NULL as string
                } elseif (is_string($param)) {
                    $types .= 's';
                } else {
                    $types .= 'b';
                }
            }

            $stmt->bind_param($types, ...$params);
        }
        $result = $stmt->execute();

        if ($result === false) {
            throw new \Exception('Query execution failed: ' .
$stmt->error . ' SQL: ' . $sql);
        }
        if ($stmt->field_count > 0) {
            $result = $stmt->get_result();
            if ($result === false) {
                throw new \Exception('Failed to get result set: '
. $stmt->error);
            }
            return $result;
        }
    }
}

```

```

        return $stmt->affected_rows >= 0 ? $stmt->affected_rows :
true;

    } catch (\Exception $e) {
        error_log('Database error: ' . $e->getMessage());
        throw $e;
    }
}

/**
 * Fetch a single row as an associative array.
 *
 * @deprecated Використовуйте selectOne() - той самий контракт, але
назва
        *
        *      узгоджена з SQL-термінологією.
 */
public function fetchOne($sql, $params = []) {
    return $this->selectOne($sql, $params);
}

/**
 * Fetch all rows as an array of associative arrays.
 *
 * @deprecated Використовуйте select() - той самий контракт.
 */
public function fetchAll($sql, $params = []) {
    return $this->select($sql, $params);
}

/**
 * Виконати SELECT і повернути МАСИВ рядків (порожній, якщо нічого
не знайдено).
 *
 * Гарантовано повертає `array<int, array<string, mixed>>`. Не
повертає
 * mysqli_result - викликача не треба турбувати про тип результату.
 *
 * @param string $sql      SELECT-запит з ?-плейсхолдерами.
 * @param array  $params   Значення для bind_param.
 * @return array<int, array<string, mixed>>
 */
public function select(string $sql, array $params = []): array

```

```

{
    $result = $this->query($sql, $params);
    $rows = [];
    if ($result instanceof \mysqli_result) {
        while ($row = $result->fetch_assoc()) {
            $rows[] = $row;
        }
    }
    return $rows;
}
/**
 * Виконати SELECT і повернути ПЕРШИЙ рядок або null, якщо нічого
не знайдено.
 *
 * @param string $sql    SELECT-запит з ?-плейсхолдерами.
 * @param array  $params Значення для bind_param.
 * @return array<string, mixed>|null
 */
public function selectOne(string $sql, array $params = []): ?array
{
    $result = $this->query($sql, $params);
    if ($result instanceof \mysqli_result) {
        $row = $result->fetch_assoc();
        return $row ?: null;
    }
    return null;
}

/**
 * Виконати INSERT/UPDATE/DELETE і повернути кількість зачеплених
рядків.
 *
 * Для INSERT - також можна отримати створений id через
getLastInsertId()
 * безпосередньо після виклику.
 *
 * Повертає `0`, якщо запит не зачепив жодного рядка (наприклад,
UPDATE,
 * де нові значення збігаються зі старими - це не помилка). Повертає
 * `-1` лише у випадку, якщо MySQL повернув від'ємний affected_rows
 * (підкісний edge case).

```

```

        * @param string $sql          INSERT/UPDATE/DELETE-запит з ?-
плейсхолдерами.
        * @param array $params Значення для bind_param.
        * @return int   Кількість зачеплених рядків.
        */
public function execute(string $sql, array $params = []): int
{
    $result = $this->query($sql, $params);

    // query() повертає int (affected_rows) або true (affected_rows
< 0).
    // Якщо випадково повернувся mysqli_result - це означає, що SQL
був
    // SELECT-ом, що неправильно для execute(). Повертаємо
кількість рядків.
    if (is_int($result)) {
        return $result;
    }
    if ($result === true) {
        return -1; // Особливий маркер, рідко зустрічається.
    }
    if ($result instanceof \mysqli_result) {
        // Виклик execute() для SELECT - невірне використання, але
звільняємо ресурси.
        $count = $result->num_rows;
        $result->free();
        return (int)$count;
    }
    return 0;
}

public function insert($table, $data) {
    $columns = implode(', ', array_keys($data));
    $placeholders = implode(', ', array_fill(0, count($data),
'?''));
    $sql = "INSERT INTO $table ($columns) VALUES ($placeholders)";

    $stmt = $this->conn->prepare($sql);
    $stmt->bind_param(str_repeat('s', count($data)),
...array_values($data));

    return $stmt->execute();
}

```

```

    }

    public function update($table, $data, $where, $whereParams = []) {
        $setClause = implode(' = ?, ', array_keys($data)) . ' = ?';
        $sql = "UPDATE $table SET $setClause WHERE $where";

        $stmt = $this->conn->prepare($sql);
        $params = array_merge(array_values($data), $whereParams);
        $stmt->bind_param(str_repeat('s', count($params)),
...$params);

        return $stmt->execute();
    }

    public function delete($table, $where, $params = []) {
        $sql = "DELETE FROM $table WHERE $where";
        $stmt = $this->conn->prepare($sql);

        if (!empty($params)) {
            $stmt->bind_param(str_repeat('s', count($params)),
...$params);
        }

        return $stmt->execute();
    }

    public function close() {
        $this->conn->close();
    }
}

```