ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:
«Інформаційна система онлайн-бронювання послуг та координації виконавців через Telegram-бот»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
(код,найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олександр РУДЕНКО
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав: здобувач вищої освіти група ІСД-41
Олександр РУДЕНКО
(ім'я, ПРІЗВИЩЕ)
Керівник Ольга ПОЛОНЕВИЧ
к.т.н., доцент *(ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
(ім'я, ПРІЗВИЩЕ)

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти бакалавр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ
_____ Каміла СТОРЧАК

“____” _____2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Руденко Олександр Олександрович
(прізвище, ім’я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Інформаційна система онлайн-бронювання послуг та координації виконавців через Telegram-бот
керівник кваліфікаційної роботи: Ольга ПОЛОНЕВИЧ к.т.н., доцент
(ім’я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від “20” лютого 2026 р. №

2. Строк подання кваліфікаційної роботи «1» червня 2026 р.

3. Вихідні дані кваліфікаційної роботи:
1. Принципи побудови клієнт-серверних веб-застосунків.
2. Технологія Telegram Bot API та засоби розробки чат-ботів.
3. Науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
1. Аналіз предметної галузі та постановка завдань.
2. Проєктування інформаційної системи.
3. Програмна реалізація системи.
4. Тестування програмного забезпечення.
5. Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання « 20 » лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Підбір технічної літератури	24.02–05.03.26	
2.	Аналіз предметної галузі та постановка завдань	06.03–19.03.26	
3.	Проєктування інформаційної системи	20.03–06.04.26	
4.	Програмна реалізація системи	07.04–24.04.26	
5.	Тестування програмного забезпечення та висновки	25.04–16.05.26	
6.	Розробка демонстраційних матеріалів, доповідь.	19.05–20.05.26	
7.	Оформлення бакалаврської роботи	21.05–30.05.26	

Здобувач вищої освіти _____ Олександр РУДЕНКО
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Ольга ПОЛОНЕВИЧ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 68 стор., 8 рис., 5 табл., 20 джерел.

Мета роботи – розробка та реалізація інформаційної системи, що автоматизує процес онлайн-бронювання послуг майстра і забезпечує оперативну координацію виконавця через Telegram-бот, скорочуючи час реакції на замовлення та зменшуючи кількість помилок при плануванні виїздів.

Об'єкт дослідження – процеси автоматизації збору клієнтських даних та управління виїзним персоналом у сервісному підприємстві з ремонту побутової техніки.

Предмет дослідження – архітектура взаємодії веб-інтерфейсу та месенджера Telegram на базі єдиного середовища виконання Node.js та документоорієнтованої бази даних MongoDB.

Короткий зміст роботи. У бакалаврській роботі проаналізовано предметну галузь онлайн-бронювання послуг та існуючі програмні рішення. Сформовано функціональні, нефункціональні та безпекові вимоги, побудовано бізнес-процеси, логічну модель бази даних і UML-діаграми. Розроблено архітектуру системи, що включає веб-інтерфейс клієнта, серверну частину на Node.js та Telegram-бот для виконавців. Реалізовано основні компоненти системи та проведено тестування ключових сценаріїв: подання заявки клієнтом, перевірки доступних слотів, обробки замовлення через Telegram-бот та генерації аналітичних звітів.

КЛЮЧОВІ СЛОВА: ІНФОРМАЦІЙНА СИСТЕМА, ОНЛАЙН-БРОНЮВАННЯ, TELEGRAM-БОТ, ВЕБ-ІНТЕРФЕЙС, NODE.JS, MONGODB, REST API, АВТОМАТИЗАЦІЯ, EXPRESS.JS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

API (Application Programming Interface) – прикладний програмний інтерфейс

CRUD (Create, Read, Update, Delete) – базові операції роботи з даними: створення, читання, оновлення, видалення

DTO (Data Transfer Object) – об’єкт передавання даних

ER-діаграма (Entity–Relationship Diagram) – діаграма «сутність–зв’язок»

HTTP (HyperText Transfer Protocol) – протокол передавання гіпертексту

HTTPS (HyperText Transfer Protocol Secure) – захищений протокол передавання гіпертексту

JSON (JavaScript Object Notation) – текстовий формат обміну даними

JWT (JSON Web Token) – маркер доступу у форматі JSON

MVC (Model-View-Controller) – архітектурний патерн «модель-вигляд-контролер»

REST (Representational State Transfer) – архітектурний стиль побудови веб-сервісів

UML (Unified Modeling Language) – уніфікована мова моделювання

URL (Uniform Resource Locator) – уніфікований локатор ресурсу

БД – база даних

ІС – інформаційна система

ПЗ – програмне забезпечення

ВСТУП

Актуальність теми. Сучасний ринок побутових послуг знаходиться в умовах зростаючої конкуренції та підвищених вимог клієнтів до якості обслуговування. Підприємства, що надають послуги ремонту побутової техніки «виїздом на дім», є типовим прикладом малого бізнесу, де операційна ефективність безпосередньо визначає дохід та репутацію. Разом із тим більшість таких підприємств досі використовують застарілі методи управління замовленнями: телефонний прийом, паперові журнали або таблиці Excel без автоматизованої синхронізації.

Ці підходи мають суттєві недоліки: неможливість приймати замовлення у нічний час, висока ймовірність помилок при ручному введенні даних, затримки в передачі інформації від диспетчера до майстра, відсутність прозорої картини завантаженості та доходів у реальному часі. Автоматизація перерахованих процесів здатна суттєво підвищити продуктивність роботи без значних інвестицій в інфраструктуру.

Особливого значення набувають рішення, орієнтовані на масові платформи з великою аудиторією. Telegram-месенджер станом на 2024 рік нараховує понад 900 мільйонів активних користувачів [7]. Функціонал Telegram Bot API дозволяє будувати повноцінні операційні інструменти поверх інфраструктури месенджера без необхідності розробки окремих мобільних застосунків. Це суттєво знижує поріг впровадження технологічних рішень для малого бізнесу.

Об’єднання веб-інтерфейсу для клієнтів і Telegram-бота для виконавця в єдину систему, побудовану на єдиній мові програмування JavaScript, дозволяє отримати гнучкий, масштабований і легкий в обслуговуванні продукт.

Розробка інтегрованої інформаційної системи онлайн-бронювання з координацією виконавців через Telegram є актуальним завданням для підприємств малого та середнього бізнесу в секторі побутових послуг. Відсутність на ринку доступних рішень, що поєднують веб-форму бронювання, перевірку доступних

слотів та Telegram-інтеграцію для персоналу, обумовлює доцільність авторської розробки.

Аналіз останніх досліджень. Питання автоматизації сервісних процесів досліджували Fowler M. у роботі «Patterns of Enterprise Application Architecture» [10] та Richardson L. у «RESTful Web Services» [11]. Специфіку документоорієнтованих СУБД висвітлено у роботі Chodorow K. «MongoDB: The Definitive Guide» [12]. Практичні аспекти розробки на Node.js описано в «Node.js in Action» Cantelon M. [13]. Однак питання інтеграції веб-систем бронювання із месенджерами для координації польового персоналу залишається відносно малодослідженим у вітчизняній академічній літературі, що підкреслює актуальність даної роботи.

Мета роботи – розробка та реалізація інформаційної системи, що автоматизує процес онлайн-бронювання послуг майстра і забезпечує оперативну координацію виконавця через Telegram-бот, скорочуючи час реакції на замовлення та зменшуючи кількість помилок при плануванні виїздів.

Для досягнення поставленої мети визначено такі завдання:

1. провести аналіз предметної галузі та виявити проблеми існуючих підходів до управління замовленнями;
2. дослідити існуючі програмні аналоги та обґрунтувати доцільність розробки власного рішення;
3. обґрунтувати вибір технологічного стеку та засобів розробки;
4. спроектувати архітектуру системи, структуру бази даних та інтерфейс користувача;
5. реалізувати клієнтський веб-інтерфейс з динамічною перевіркою зайнятих слотів;
6. розробити серверну частину з REST API на базі Node.js та Express;
7. реалізувати Telegram-бот для виконавця з управлінням статусами замовлень;
8. розробити модуль автоматичної генерації аналітичних Excel-звітів;

9. провести функціональне тестування та оцінити продуктивність системи.

Об'єкт дослідження – процеси автоматизації збору клієнтських даних та управління виїзним персоналом у сервісному підприємстві з ремонту побутової техніки.

Предмет дослідження – архітектура взаємодії веб-інтерфейсу та месенджера Telegram на базі єдиного середовища виконання Node.js та документоорієнтованої бази даних MongoDB.

Методи дослідження. Системний аналіз і синтез при проектуванні архітектури; об'єктно-орієнтований підхід при реалізації; UML-моделювання для опису вимог і структури системи; функціональне тестування методом «чорного ящика» для перевірки коректності роботи.

Наукова новизна одержаних результатів полягає в розробці оригінальної архітектури інтеграції клієнтського веб-інтерфейсу і Telegram-месенджера в єдину систему координації сервісного персоналу на базі JavaScript-стеку (Node.js + MongoDB + Telegraf), що відрізняється уніфікацією технологій та відсутністю надлишкових проміжних сервісів.

Практична значущість одержаних результатів полягає в тому, що розроблена система готова до безпосереднього впровадження на підприємстві ТехноМайстер. Система автоматизує повний цикл від отримання замовлення до його виконання, скорочує час диспетчеризації та забезпечує власника аналітичними даними для прийняття управлінських рішень.

Апробація результатів та публікації. Основні положення і результати бакалаврської роботи доповідались на науково-практичних конференціях, що проходили на базі Державного університету інформаційно-комунікаційних технологій. За результатами дослідження опубліковано тези доповіді на тему «Інформаційна система онлайн-бронювання послуг та координації виконавців через Telegram-бот».

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАНЬ

1.1 Опис задачі та предметної галузі

Сервісні підприємства, що спеціалізуються на виїзному ремонті побутової техніки, становлять окремий сегмент ринку B2C-послуг із притаманними йому організаційними особливостями. На відміну від стаціонарних майстерень, де клієнт самостійно доставляє техніку, виїзний сервіс передбачає переміщення технічного персоналу до адреси клієнта. Це суттєво ускладнює логістику та координацію.

Бізнес-процес обслуговування замовлення включає такі послідовні етапи: ініціювання клієнтом запиту, реєстрацію замовлення, призначення виконавця, виконання ремонту на місці та закриття заявки. Кожен із цих етапів потенційно може містити точки відмови при відсутності автоматизації.

За даними дослідження ринку побутових послуг, малі сервісні підприємства України в більшості випадків використовують один із трьох підходів до прийому замовлень:

– телефонний прийом через одного диспетчера – найпоширеніший варіант, що не масштабується;

– паперові журнали – надійні в автономному режимі, але недоступні дистанційно;

– таблиці Google Sheets або Excel – дозволяють спільну роботу, але вимагають ручного введення та не мають клієнтського фронтенду.

Кожен з перерахованих підходів має системний недолік: відсутність інтегрованого каналу прийому замовлень, доступного клієнту в будь-який час. Клієнт, що звертається після закінчення робочого дня, або чекає до ранку, або шукає альтернативний сервіс. Обидва варіанти є програшними для підприємства.

Не менш критичною є проблема передачі інформації від диспетчера до виконавця. При телефонному прийомі диспетчер записує деталі замовлення та передає їх майстру окремим дзвінком або повідомленням. Це створює додаткову затримку та ймовірність помилок (неправильно записана адреса, забута деталь несправності тощо).

Аналіз предметної галузі виявив наступні ключові проблеми, що вирішуються в даній роботі:

1. відсутність цілодобового каналу самостійного запису клієнтів;
2. ручна перевірка зайнятості майстра на конкретний час;
3. затримка між реєстрацією замовлення та інформуванням виконавця;
4. відсутність зручного мобільного інструменту для виконавця, що знаходиться в полі;
5. відсутність автоматичного ведення статистики та формування звітності.

Пропонована система вирішує всі п’ять проблем: клієнт самостійно реєструє заявку через веб-форму будь-якої миті, система автоматично перевіряє зайняті слоти, інформація миттєво надходить до майстра через Telegram, виконавець управляє статусами замовлень зі смартфона, а власник отримує актуальний Excel-звіт після кожної нової заявки.

Система ТехноМайстер призначена для підприємства, що надає послуги ремонту: пральних машин, холодильників, духових шаф та плит, посудомийних машин, а також проводить діагностику та профілактичне обслуговування. Цільова аудиторія клієнтів – жителі міста, що потребують швидкого виїзду майстра. Цільова аудиторія виконавців – технічні спеціалісти, що постійно перебувають поза офісом.

З технічної точки зору система має відповідати таким базовим характеристикам: доступність через стандартний браузер без встановлення додаткового програмного забезпечення; сумісність із мобільними пристроями на рівні UI; реакція на дії користувача без повного перезавантаження сторінки (SPA-

підхід для форми); відповідь сервера на запити не довше 2 секунд при нормальному навантаженні.

1.2 Аналіз існуючих програмних рішень та їх недоліки

Для визначення відмінностей та переваг розроблюваної системи було проведено порівняльний аналіз найпоширеніших програмних рішень, що частково охоплюють досліджувану функціональну область.

Yclients

Yclients – найбільша CRM-платформа для сервісного бізнесу в Україні та країнах СНД. Система орієнтована на широкий клас підприємств: салони краси, медичні заклади, фітнес-центри, а також деякі типи сервісних майстерень.

Переваги Yclients: повний цикл управління записами, автоматичні SMS та email-нагадування клієнтам, вбудована CRM з карткою клієнта, підтримка онлайн-оплати, мобільний застосунок для персоналу, розширена аналітика. Недоліки: висока вартість підписки (від 990 грн/міс для базового тарифу, що є суттєвим для малого бізнесу), надмірна функціональна складність для невеликого одномайстерного підприємства, відсутність нативної інтеграції з Telegram-ботом як основним каналом сповіщень персоналу, необхідність навчання персоналу роботі з системою.

Booksy

Booksy – міжнародна платформа для бронювання послуг із сильними позиціями у б’юті-секторі. Представлена в Україні, але орієнтована переважно на перукарні, барбершопи та б’юті-послуги.

Переваги: зручний мобільний застосунок, можливість онлайн-оплати, маркетплейс для залучення нових клієнтів. Недоліки: відсутність підтримки категорій ремонтних послуг, висока комісія з транзакцій (до 5%), сильна орієнтація на індивідуальних майстрів, а не на координацію виїзного персоналу, відсутність Telegram-інтеграції.

Zoho Bookings

Zoho Bookings – хмарне рішення для управління бронюваннями від компанії Zoho, відоме своєю глибокою інтеграцією з іншими продуктами екосистеми Zoho (CRM, Mail, Desk).

Переваги: гнучке налаштування типів послуг, зв'язок із Zoho CRM, зрозумілий інтерфейс. Недоліки: інтерфейс виключно англійською мовою, що є бар'єром для українського ринку; відсутність Telegram-інтеграції; вартість від 15 USD/міс при незначних можливостях безкоштовного тарифу; мобільний застосунок для персоналу поступається конкурентам за зручністю.

Самостійні Telegram-боти

Розробка або використання готового Telegram-бота без веб-сайту – популярний підхід серед малого бізнесу в Telegram-спільноті. Переваги: нульова вартість, простота розгортання, знайомий інтерфейс для клієнтів.

Недоліки: незручний процес бронювання через Telegram (порівняно зі спеціалізованою веб-формою), неможливість відображення зайнятих/вільних слотів у вигляді візуальної сітки, відсутність адаптивного UI для вибору дати, обмежені можливості аналітики.

Порівняльний аналіз перерахованих рішень та розроблюваної системи наведено у таблиці 1.1.

Таблиця 1.1

Критерій оцінювання	**Yclients**	**Booksy**	**Zoho Bookings**	**Розроблена система**
Веб-форма бронювання	Так	Так	Так	Так
Перевірка зайнятих слотів в реальному часі	Так	Так	Так	Так
Нативна Telegram-інтеграція	Ні	Ні	Ні	Так

Продовження таблиці 1.1

Управління статусами в Telegram	Ні	Ні	Ні	Так
Автоматичний Excel-звіт	Частково	Ні	Частково	Так
Українська мова інтерфейсу	Так	Частково	Ні	Так
Вартість базового тарифу / міс.	990+ грн	Комісія	~600 грн	Безкоштовно (open-source)
Гнучкість налаштування	Обмежена	Обмежена	Середня	Повна
Складність розгортання	Хмара	Хмара	Хмара	Node.js сервер

Виходячи з результатів аналізу, жодне із розглянутих рішень не забезпечує оптимального для малого сервісного підприємства поєднання: зручна веб-форма для клієнта + Telegram-бот для виконавця + автоматична аналітика + відкритий код та відсутність абонплати. Саме ця ніша заповнюється розроблюваною системою.

1.3 Обґрунтування вибору технологічного стеку

Вибір конкретних технологій для реалізації системи базувався на таких критеріях: уніфікація мови програмування на всіх рівнях, зрілість та активна підтримка екосистеми, висока продуктивність при асинхронному навантаженні, наявність спеціалізованих бібліотек для Telegram Bot API та Excel, простота розгортання.

Node.js

Node.js – середовище виконання JavaScript поза браузером, побудоване на рушії V8. Ключова перевага для даного проекту – неблокуючий ввід/вивід (event loop), що дозволяє одному процесу одночасно обробляти HTTP-запити від веб-форми та довгоживучі з’єднання для опитування Telegram Bot API (long polling).

При традиційному підході (Apache + PHP) для кожного одночасного запиту створюється окремий системний потік, тоді як Node.js обробляє їх у одному потоці асинхронно.

Додаткова перевага – уніфікація мови програмування: один розробник може підтримувати і клієнтський JavaScript (браузер), і серверний Node.js без перемикання між мовами. Версія Node.js 20 LTS використовується у проекті завдяки підтримці ES2022-синтаксису та стабільним API.

Express 5

Express – мінімалістичний веб-фреймворк для Node.js. Забезпечує маршрутизацію HTTP-запитів, ланцюжки middleware (проміжних обробників) та механізм обробки помилок. Версія 5 (на стадії release candidate на момент розробки) використана завдяки покращеній обробці асинхронних помилок у маршрутах.

Для даного проекту Express виконує такі функції: обробка GET-запитів для отримання зайнятих слотів, обробка POST-запитів для збереження нових замовлень, ендпоінт для завантаження аналітичного Excel-файлу, налаштування CORS для дозволу запитів від клієнтської частини.

MongoDB та Mongoose

MongoDB – документоорієнтована NoSQL СУБД, що зберігає дані у вигляді JSON-подібних документів (BSON). Для даного проекту обрано MongoDB з таких міркувань: структура замовлення є природним документом без потреби в JOIN-операціях; відсутність фіксованої схеми дозволяє легко додавати нові поля (наприклад, геолокацію або фото несправності) без міграцій; формат JSON на рівні API, СУБД та JavaScript об’єктів є уніфікованим, що мінімізує перетворення даних.

Mongoose – ODM-бібліотека (Object Document Mapper) для MongoDB. Надає Mongoose-схеми з валідацією, типізацією полів та зручними методами (find, findByIdAndUpdate, save). У проекті Mongoose відповідає за визначення схеми колекції orders та валідацію обов’язкових полів при збереженні нового замовлення.

Telegraf

Telegraf – сучасна бібліотека для розробки Telegram-ботів на Node.js з підтримкою TypeScript. Порівняно з офіційною бібліотекою node-telegram-bot-api,

Telegraf пропонує більш зручний middleware-підхід до написання обробників, вбудовану підтримку Inline Keyboards та CallbackQuery, компактний синтаксис регулярних виразів для обробки подій. Версія 4.x застосована у проекті для реалізації повноцінного інтерфейсу управління замовленнями для майстра.

ExcelJS та QuickChart

ExcelJS – бібліотека для програмної роботи з файлами .xlsx: створення аркушів, застосування форматування, вставка зображень. Обрано замість csv-export завдяки підтримці форматування комірок, кольорового кодування рядків за статусом та інтеграції діаграм.

QuickChart – відкритий HTTP API для генерації PNG-зображень діаграм на основі конфігурації Chart.js. Використовується для отримання графіків (кільцева діаграма статусів, стовпчаста діаграма доходів по місяцях), які вставляються в Excel-файл через ExcelJS.

HTML5/CSS3/JavaScript (клієнтська частина)

Клієнтська частина реалізована без JavaScript-фреймворків (React, Vue) на чистих веб-технологіях. Це дозволяє уникнути збірки (build step), мінімізує розмір завантажуваних файлів та спрощує розгортання. Бібліотека Flatpickr використовується виключно для компонента вибору дати завдяки підтримці блокування конкретних днів та інтернаціоналізації.

Порівняння технологічного стеку наведено в таблиці 1.2

Таблиця 1.2

Компонент	**Обрана технологія**	**Альтернатива**	**Обґрунтування вибору**
Середовище виконання	Node.js 20 LTS	PHP / Python Flask	Уніфікація JS на всіх рівнях, асинхронний I/O
Веб-фреймворк	Express 5	Fastify / Koa	Зрілість, велика спільнота, простота middleware
СУБД	MongoDB 7	PostgreSQL / MySQL	JSON-native, гнучка схема, легке розгортання
ODM / ORM	Mongoose 9	Prisma / Raw MongoDB Driver	Схеми з валідацією, зручне API запитів

Продовження таблиці 1.2

Telegram Bot	Telegraf 4	node-telegram-bot-api	Middleware-підхід, Inline Keyboards, TypeScript
Excel-генерація	ExcelJS 4	xlsx / csv	Форматування, кольори, інтеграція зображень

1.4 Об’єкт, предмет, мета та завдання дослідження

Об’єктом дослідження є процеси автоматизації збору клієнтських даних та управління виїзним персоналом у сервісному підприємстві з ремонту побутової техніки – тобто всі організаційні та інформаційні процеси, що охоплюють шлях від звернення клієнта до закриття виконаного замовлення.

Предметом дослідження є архітектура взаємодії веб-інтерфейсу та месенджера Telegram на базі єдиного середовища виконання Node.js та документоорієнтованої бази даних MongoDB – тобто конкретна програмна реалізація та технічні рішення, що застосовуються для вирішення задачі.

Метою роботи є розробка та реалізація інформаційної системи, що автоматизує процес онлайн-бронювання послуг і забезпечує оперативну координацію виконавця через Telegram-бот, скорочуючи час реакції на замовлення та зменшуючи кількість помилок при плануванні.

Завдання дослідження детально описані у вступі та реалізуються послідовно в розділах 2–4 даної роботи.

У першому розділі проведено ґрунтовний аналіз предметної галузі підприємств виїзного сервісу. Виявлено п’ять ключових системних проблем, що уповільнюють роботу: відсутність цілодобового запису, ручна перевірка зайнятості, затримка інформування виконавця, незручні мобільні інструменти для персоналу, відсутність автоматичної аналітики.

Аналіз чотирьох конкурентних рішень (Yclients, Booksy, Zoho Bookings, Telegram-бот без веб-сайту) засвідчив, що жодне з них не надає оптимального

поєднання веб-форми бронювання, Telegram-інтеграції для персоналу та автоматичної аналітики без абонентської плати. Це підтвердило доцільність розробки власної системи.

Обґрунтовано технологічний стек: Node.js для асинхронного сервера, Express для REST API, MongoDB та Mongoose для зберігання даних, Telegraf для Telegram-бота, ExcelJS для звітності. Уніфікація JavaScript на всіх рівнях є ключовою перевагою обраного стеку.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Проектування архітектури системи

Архітектура інформаційної системи ТехноМайстер побудована за трирівневою моделлю (Three-Tier Architecture), яка розмежовує рівні представлення, бізнес-логіки та даних. Окремим функціональним блоком виступає Telegram-бот, що взаємодіє з рівнем бізнес-логіки через спільну базу даних.

Така архітектура забезпечує незалежність компонентів: клієнтська частина (браузер) не має прямого доступу до бази даних, усі операції з даними виконуються виключно через API-сервер. Telegram-бот і веб-клієнт отримують дані через один і той самий шар бізнес-логіки, що гарантує консистентність стану.

Компоненти системи та їх взаємодія

Система складається з таких компонентів:

– Client (браузер) – адаптивний веб-сайт із формою бронювання. Взаємодіє з сервером через HTTP-запити (Fetch API). Не потребує встановлення;

– Express API Server (Node.js) – центральний компонент, що обробляє HTTP-запити від браузера та керує Telegram-ботом в одному процесі. Реалізує бізнес-логіку перевірки слотів та збереження замовлень;

– MongoDB – сховище документів замовлень. Зберігає колекцію orders, до якої звертаються і API-сервер, і Telegram-бот;

– Telegram Bot (Telegraf) – інтегрований у Node.js-процес компонент. Підтримує long polling для отримання повідомлень від виконавця та виконує запити до MongoDB для отримання замовлень;

– ExcelJS + QuickChart – підсистема аналітики. Генерує .xlsx-файл при старті та після кожного нового замовлення;

– .env конфігурація – зберігає секрети (MongoDB URI, Telegram токен, Master ID) поза кодом.

Між компонентами визначено такі протоколи взаємодії:

– Браузер ↔ Express: HTTP/JSON через порт 5000 (локально) або HTTPS при продакшн-деплойменті;

– Express ↔ MongoDB: внутрішній протокол Mongoose/MongoDB через порт 27017;

– Telegraf ↔ Telegram API: HTTPS long polling до api.telegram.org/bot{TOKEN}/getUpdates;

– ExcelJS ↔ QuickChart: HTTPS POST до api.quickchart.io/chart для отримання PNG зображень.

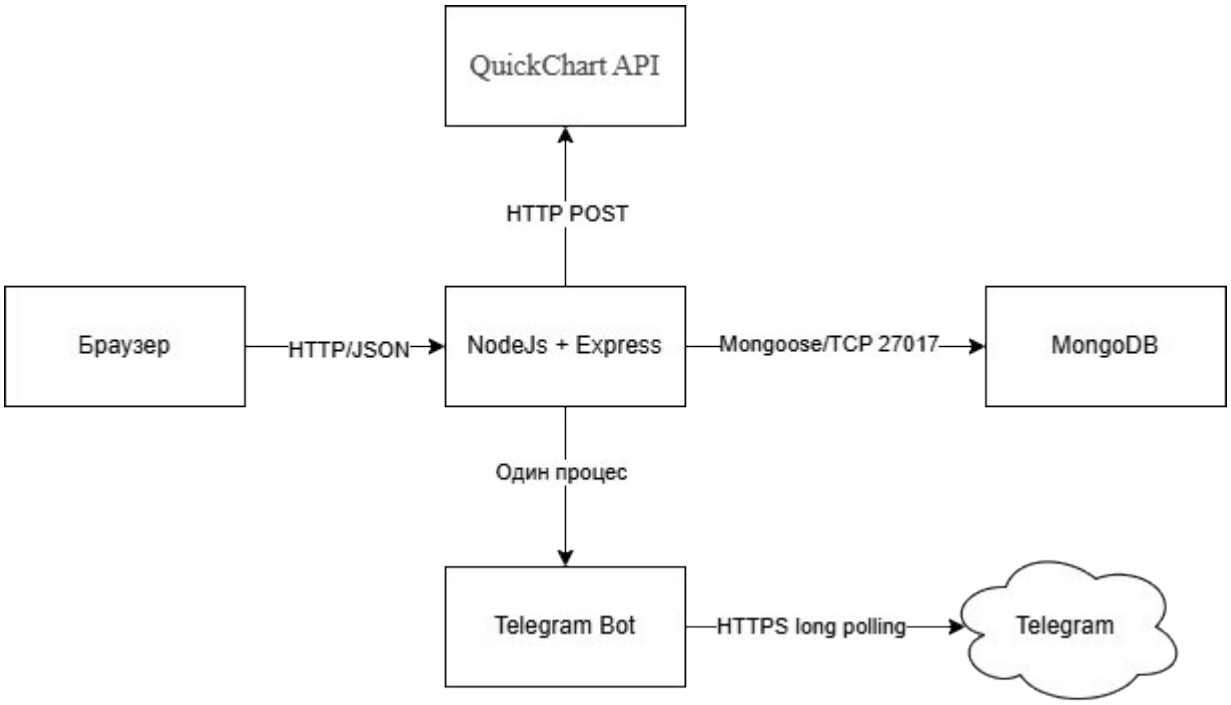


Рис. 2.1. Структурна схема архітектури інформаційної системи ТехноМайстер

Потік даних при бронюванні

Розглянемо детальний потік даних при типовому сценарії – клієнт здійснює бронювання:

1. Клієнт відкриває сторінку, обирає послугу та дату.

2. Браузер надсилає GET /api/busy-slots?date=ДД.ММ.РРРР.
3. bookingController знаходить замовлення на цю дату у MongoDB та повертає масив часів.
4. Браузер відображає слоти, позначаючи зайняті як неактивні.
5. Клієнт обирає вільний слот, заповнює контактні дані та натискає «Підтвердити».
6. Браузер надсилає POST /api/book із JSON-тілом замовлення.
7. Mongoose валідує дані за схемою OrderSchema та зберігає документ у MongoDB.
8. scheduleGeneration() ініціює відкладену (2c) регенерацію Excel-файлу.
9. Клієнт бачить повідомлення про успішне бронювання.
10. Паралельно виконавець через Telegram може в будь-який момент запросити список замовлень, не порушуючи основний потік.

2.2 Моделювання функціональних та нефункціональних вимог

Функціональні вимоги описують поведінку системи з точки зору двох основних акторів: Клієнта (відвідувач сайту) та Майстра (технічний виконавець). Для кожного актора визначено набір варіантів використання (Use Cases).

Варіанти використання для Клієнта

ВП-1: Перегляд послуг. Клієнт відвідує сайт та переглядає перелік доступних послуг із орієнтовними цінами. Передумови: відсутні. Постумови: клієнт обізнаний про послуги.

ВП-2: Вибір послуги. Клієнт обирає тип послуги зі списку. Передумови: сторінка завантажена. Постумови: поле послуги заповнено, прогрес-індикатор переходить до кроку 2.

ВП-3: Вибір дати. Клієнт відкриває календар Flatpickr та обирає робочий день поточного місяця. Передумови: обрано послугу. Постумови: система виконує запит на отримання зайнятих слотів для обраної дати.

ВП-4: Перегляд вільних слотів. Система відображає сітку часових слотів із позначенням зайнятих. Передумови: обрано дату, отримано відповідь API. Постумови: клієнт бачить актуальну картину зайнятості.

ВП-5: Вибір часу. Клієнт натискає на вільний слот. Передумови: слот активний (вільний). Постумови: час зафіксовано у стані форми, кнопка підтвердження стає доступнішою.

ВП-6: Введення контактних даних. Клієнт вводить ім’я, телефон, адресу та необов’язковий коментар. Передумови: обрано послугу, дату та час. Постумови: поля заповнено, кнопка «Підтвердити» стає активною при повністю заповненій формі.

ВП-7: Відправка форми. Клієнт натискає «Підтвердити запис». Передумови: всі обов’язкові поля заповнено. Постумови: замовлення збережено у MongoDB, клієнт бачить підтвердження.

Варіанти використання для Майстра

ВП-8: Авторизація у боті. Майстер надсилає /start боту. Передумови: chatId майстра зареєстрований у змінній MASTER_ID. Постумови: з’являється Reply Keyboard із двома кнопками. Альтернативний сценарій: якщо chatId не відповідає MASTER_ID – бот ігнорує повідомлення.

ВП-9: Перегляд замовлень на сьогодні. Майстер натискає «📆 Сьогоднішні записи». Передумови: авторизований. Постумови: відображаються картки активних замовлень на поточну дату (статуси new та in_progress), відсортовані за часом.

ВП-10: Перегляд всіх активних замовлень. Майстер натискає «📊 Всі записи». Передумови: авторизований. Постумови: відображаються всі замовлення зі статусами new та in_progress.

ВП-11: Зміна статусу замовлення. Майстер натискає Inline-кнопку «⏳ В процесі», «✅ Виконано» або «❌ Скасовано» у картці замовлення. Передумови: картка замовлення відображена. Постумови: статус у MongoDB оновлено, картка редагується з новим статусом, Excel-звіт оновлюється з затримкою 2с.

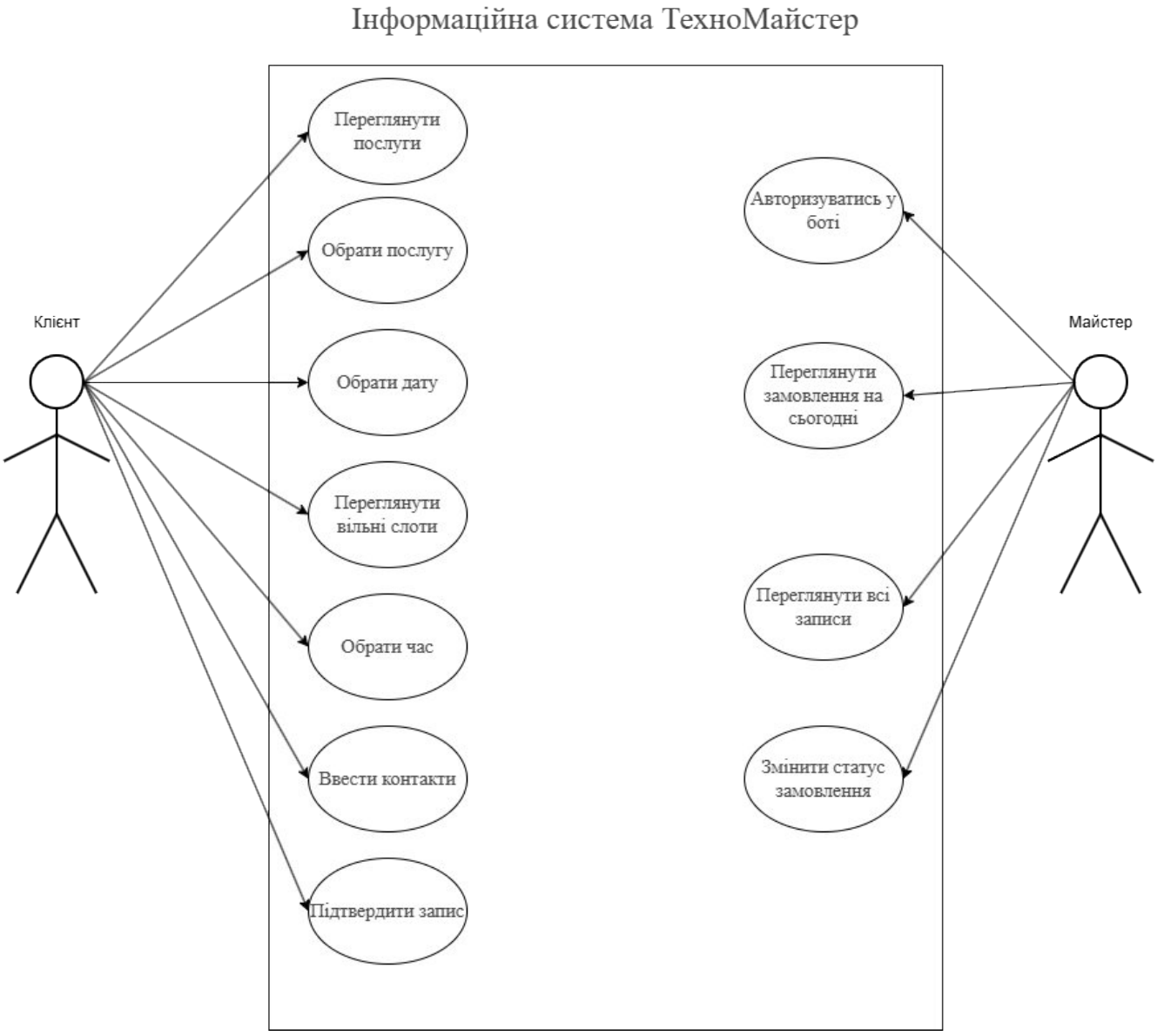


Рис. 2.2. Діаграма варіантів використання системи

Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи та обмеження:

– Продуктивність: середній час відповіді API не перевищує 200 мс для GET-запитів та 500 мс для POST-запитів при навантаженні до 20 одночасних користувачів;

– Адаптивність UI: веб-інтерфейс коректно відображається на пристроях із шириною екрана від 320px до 2560px (mobile-first підхід);

– Безпека: доступ до функцій Telegram-бота обмежено перевіркою chatId; вхідні дані форми валідуються на стороні сервера через Mongoose-схему;

– Надійність: збереження замовлень гарантується MongoDB; у разі помилки генерації Excel-файлу сервер не зупиняється (помилка логується, основний процес продовжується);

– Супроводжуваність: код розбито на незалежні модулі (controllers, models, routes, services); конфігурація зберігається в .env та не дублюється у коді.

2.3 Проектування структури бази даних

База даних системи реалізована на MongoDB та містить одну основну колекцію orders. Документоорієнтована модель MongoDB дозволяє зберігати всі атрибути замовлення в єдиному документі, що усуває потребу у JOIN-операціях та забезпечує атомарність при збереженні.

Схема документа Order

Схема документа визначена через Mongoose OrderSchema у файлі backend/models/order.js. Кожен документ колекції orders відповідає одному замовленню та містить наступні поля:

Таблиця 2.1

Поле	**Тип (Mongoose)**	**Обов' язкове**	**Опис і призначення**

Продовження таблиці 2.1

_id	ObjectId (авто)	Авто	Унікальний ідентифікатор документа, генерується MongoDB автоматично
name	String	Так	Повне ім’я клієнта, вводиться у формі бронювання
phone	String	Так	Номер телефону клієнта, відображається клікабельним посиланням у Telegram
address	String	Так	Адреса виклику майстра: вулиця, будинок, квартира
service	String	Так	Тип обраної послуги зі списку (напр. «Ремонт пральної машини»)
date	String	Так	Дата запису у форматі ДД.ММ.РРРР. String (а не Date) для спрощення порівнянь без часового поясу
time	String	Так	Час запису у форматі ГГ:ХХ (напр. «14:00»). Зберігається як рядок для точного порівняння
comment	String	Ні	Необов’язковий коментар клієнта (опис несправності, побажання)
status	String (enum)	Так (default: ’new’)	Поточний статус: new / in_progress / completed / canceled. Керується майстром через Telegram-бот
createdAt	Date (авто)	Авто	Дата та час створення запису. Використовується для сортування в аналітиці

Принципи роботи зі статусами

Поле status є ключовим для координації між клієнтом і майстром. Поле price зберігає фактичну вартість, введену майстром після виконання (для ремонтів) або встановлену автоматично (для фіксованих послуг). Поле status проходить такий життєвий цикл:

– new – початковий статус, встановлюється автоматично при POST /api/book;

– in_progress – майстер виїхав до клієнта або розпочав роботу;

– completed – робота виконана, замовлення закрите;

– canceled – замовлення скасоване (клієнт або майстер).

Переходи між статусами відбуваються виключно через Telegram-бот майстра (метод Order.findByIdAndUpdate). При зміні статусу на completed система автоматично включає суму послуги до розрахунку доходу в Excel-аналітиці.

Запити до колекції

У системі використовуються наступні типи запитів до MongoDB:

– Order.find({ date }) – пошук всіх замовлень на конкретну дату для відображення зайнятих слотів (GET /api/busy-slots);

– new Order(data).save() – збереження нового замовлення (POST /api/book);

– Order.find({ date: today, status: { $nin: ['completed', 'canceled'] } }) – фільтрація активних замовлень на сьогодні для бота;

– Order.findByIdAndUpdate(id, { status }, { new: true }) – оновлення статусу замовлення через бот;

– Order.find({}).sort({ createdAt: 1 }) – отримання всіх замовлень для генерації Excel.

2.4 Проектування інтерфейсу користувача

Клієнтський інтерфейс розроблявся за принципами mobile-first design та progressive disclosure (поступового розкриття складності). Сторінка побудована за лінійною структурою прокрутки: навігаційний блок → герой-секція → переваги сервісу → форма запису → контакти.

Навігація та Hero-секція

Навігаційна панель закріплена у верхній частині сторінки (position: sticky) із напівпрозорим фоном та ефектом розмиття (backdrop-filter). Це забезпечує постійний доступ до кнопки CTA «Записатись» незалежно від положення прокрутки.

Hero-секція містить заголовок H1, підзаголовок, дві CTA-кнопки (основна та привидна) та статистичну стрічку (роки досвіду, час виїзду, гарантія). Праворуч

розміщено імітацію карток замовлень – UI-елемент, що демонструє інтерфейс майстра та посилює довіру клієнта.

Форма бронювання

Форма бронювання є центральним елементом UI та реалізована як багатокрокова взаємодія. Ліва панель відображає прогрес-індикатор із 4 кроків, правий блок – поточний активний крок. Кроки не приховуються почергово (усі поля видимі), але прогрес-індикатор візуально показує, що заповнено.

Крок 1 – вибір послуги: select-елемент із шістьма типами послуг. При виборі оновлюється стан state.service, запускається оновлення прогресу.

Крок 2 – вибір дати та часу: Flatpickr-календар із блокуванням вихідних та дат поза поточним місяцем. При виборі дати виконується fetch-запит до API, а сітка часових слотів оновлюється асинхронно. Під час завантаження відображається shimmer-ефект.

Крок 3 – контактні дані: поля ім’я, телефон, адреса (обов’язкові) та коментар (необов’язковий). Кожне поле містить іконку зліва (fieldIcon) для кращого сприйняття.

Крок 4 – підтвердження: кнопка «Підтвердити запис» активується лише при повністю заповненій формі. При успішній відповіді сервера відображається зелений банер із деталями бронювання, форма очищується.

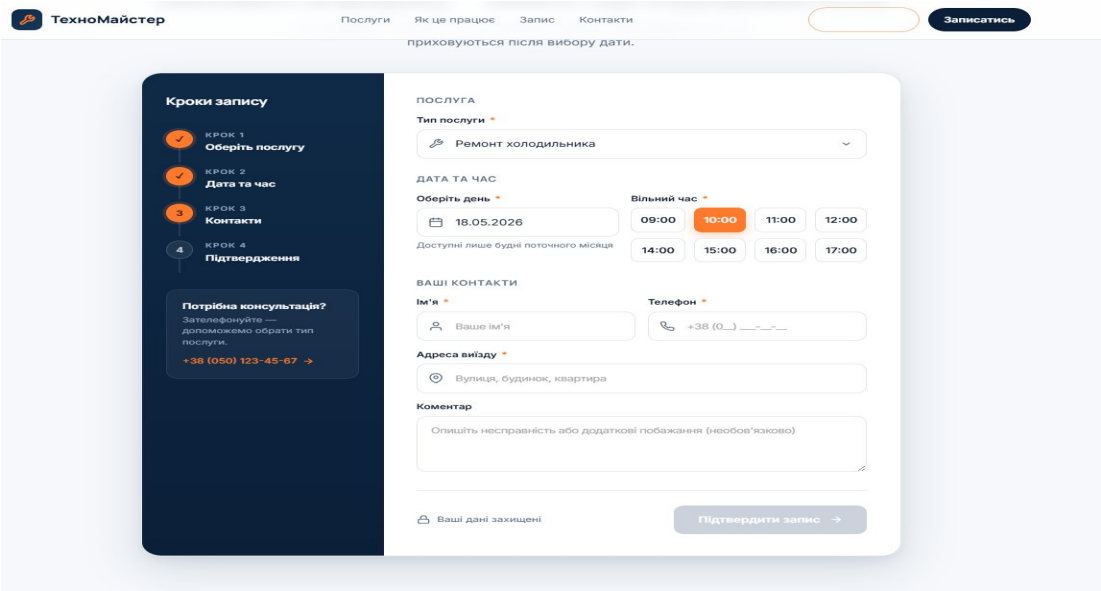


Рис. 2.3. Веб-інтерфейс форми онлайн-бронювання системи ТехноМайстер

Кольорова схема та типографіка

Кольорова схема UI базується на темно-синьому (#0b1f38) як основному корпоративному кольорі та помаранчевому акценті (#ff7a2d) для CTA-елементів. Нейтральний фон (#f6f8fb) забезпечує комфортне читання. Колір небезпеки (#dc2626) використовується для повідомлень про помилки.

Типографіка: Inter (400, 500, 600) – для основного тексту, Manrope (500–800) – для заголовків. Обидва шрифти завантажуються з Google Fonts, що забезпечує оптимальну якість відображення на різних пристроях.

У другому розділі описано архітектурні рішення та проектування системи. Обрано трирівневу архітектуру з єдиним Node.js-процесом для HTTP-сервера та Telegram-бота. Виявлено 11 варіантів використання для двох акторів. Спроектовано схему документа MongoDB з 10 полями та чітко описано статусну модель замовлення. Розроблено UI-концепцію форми бронювання на основі принципів mobile-first та progressive disclosure.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Структура та реалізація серверного застосунку

Серверна частина системи реалізована на платформі Node.js з використанням фреймворку Express 5. Точкою входу є файл server.js, що ініціалізує підключення до бази даних, налаштовує middleware та запускає HTTP-сервер.

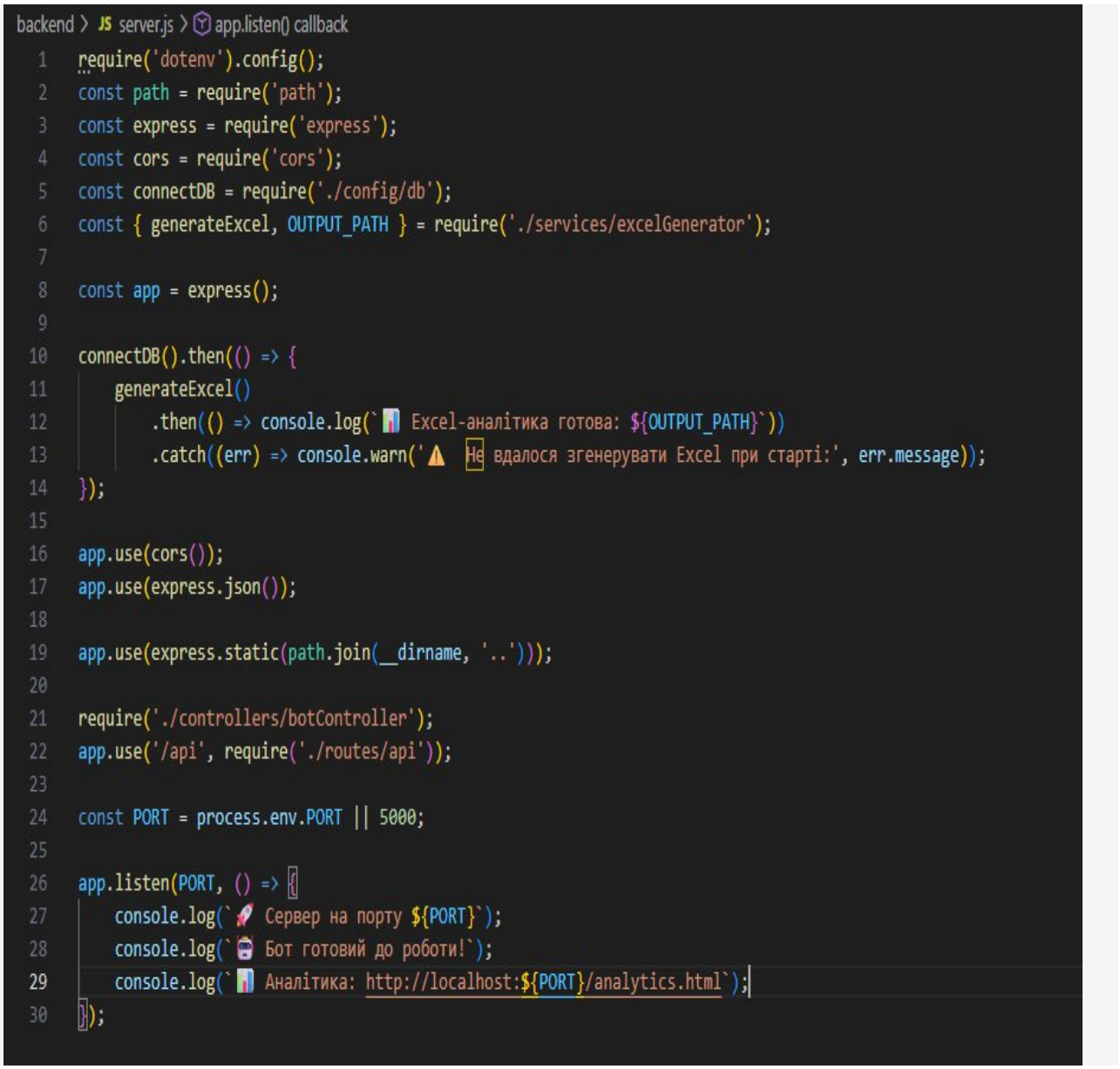
Структура файлів

Проект організовано за модульним принципом із чітким розмежуванням відповідальностей:

```
backend/
├── server.js                    # Точка входу застосунку
├── .env                         # Змінні оточення (не в репозиторії)
├── package.json                 # Залежності проекту
├── config/
│   └── db.js                    # Підключення до MongoDB через Mongoose
├── models/
│   └── order.js                 # Mongoose-схема документа замовлення
├── controllers/
│   ├── bookingController.js     # Логіка бронювання (GET busy-slots, POST
book)
│   ├── botController.js         # Telegram-бот (Telegraf instance + handlers)
│   └── analyticsController.js  # HTTP-ендпоінт для завантаження Excel
├── routes/
│   └── api.js                   # Маршрутизація Express
└── services/
    └── excelGenerator.js        # Фонова генерація .xlsx аналітики
```

Файл server.js

Файл server.js є точкою входу: підключає БД, налаштовує CORS та JSON middleware, роздає статичні файли (index.html, analytics.html) через express.static, монтує API-роутер та запускає HTTP-сервер:



```
backend > JS server.js > app.listen() callback
require('dotenv').config();
const path = require('path');
const express = require('express');
const cors = require('cors');
const connectDB = require('./config/db');
const { generateExcel, OUTPUT_PATH } = require('./services/excelGenerator');

const app = express();

connectDB().then(() => {
    generateExcel()
        .then(() => console.log(`📊 Excel-аналітика готова: ${OUTPUT_PATH}`))
        .catch((err) => console.warn('⚠️ Не вдалося згенерувати Excel при старті:', err.message));
});

app.use(cors());
app.use(express.json());

app.use(express.static(path.join(__dirname, '..')));

require('./controllers/botController');
app.use('/api', require('./routes/api'));

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {
    console.log(`🚀 Сервер на порту ${PORT}`);
    console.log(`🤖 Бот готовий до роботи!`);
    console.log(`📊 Аналітика: http://localhost:${PORT}/analytics.html`);
});
```

Виклик require('./controllers/botController') без присвоєння ініціалізує Telegraf-екземпляр та запускає метод bot.launch() як побічний ефект. Це дозволяє боту функціонувати в одному процесі з HTTP-сервером.

Підключення до MongoDB (config/db.js)

Модуль підключення до бази даних реалізований як async-функція з обробкою критичної помилки (process.exit(1)), щоб зупинити сервер при недоступності БД:

```
backend > config > JS db.js > connectDB
const mongoose = require('mongoose');

const connectDB = async () => {
    try {
        await mongoose.connect(process.env.MONGO_URI);
        console.log('✅ MongoDB підключено успішно');
    } catch (err) {
        console.error('❌ Помилка підключення до БД:', err.message);
        process.exit(1);
    }
};

module.exports = connectDB;
```

Mongoose-схема Order (models/order.js)

Схема визначає структуру та обмеження кожного документа колекції orders. Поле status за замовчуванням встановлюється в ’new’, що відповідає стану «щойно зареєстроване замовлення»:

```
backend > models > JS order.js > ...
const mongoose = require('mongoose');

const OrderSchema = new mongoose.Schema({
    name: { type: String, required: true },
    phone: { type: String, required: true },
    address: { type: String, required: true },
    service: { type: String, required: true },
    date: { type: String, required: true }, // формат ДД.ММ.РРРР
    time: { type: String, required: true },
    comment: String,
    status: { type: String, default: 'new' },
    price: { type: Number, default: 0 },     // фактична вартість (грн)
    createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Order', OrderSchema);
```

Контролер бронювання (controllers/bookingController.js)

Контролер реалізує два методи. Метод getBusySlots знаходить усі замовлення на конкретну дату та повертає масив зайнятих часових слотів. Метод createBooking зберігає новий запис та ініціює оновлення аналітики:

backend > controllers > JS bookingController.js > ...

```
const Order = require('../models/order');
const { scheduleGeneration } = require('../services/excelGenerator');

exports.getBusySlots = async (req, res) => {
    try {
        const { date } = req.query;
        const orders = await Order.find({ date });
        const busySlots = orders.map(order => order.time);
        res.json(busySlots);
    } catch (err) {
        res.status(500).json({ message: 'Помилка сервера' });
    }
};

exports.createBooking = async (req, res) => {
    try {
        const newOrder = new Order(req.body);
        await newOrder.save();

        // Оновити Excel-аналітику у фоні
        scheduleGeneration();

        res.status(201).json({ message: 'Запис створено успішно!' });
    } catch (err) {
        res.status(400).json({ message: 'Помилка при створенні запису' });
    }
};
```

Маршрутизація API (routes/api.js)

Маршрути відповідають REST-принципам: GET для отримання даних, POST для створення ресурсу:

backend > routes > JS api.js > ...

```
const express = require('express');
const router = express.Router();
const bookingController = require('../controllers/bookingController');
const analyticsController = require('../controllers/analyticsController');

router.get('/busy-slots', bookingController.getBusySlots);
router.post('/book', bookingController.createBooking);

router.get('/analytics/export', analyticsController.exportExcel);
router.get('/analytics/data',   analyticsController.getAnalyticsData);

module.exports = router;
```

3.2 Реалізація клієнтської частини та форми бронювання

Клієнтська частина системи реалізована на чистих HTML5, CSS3 та JavaScript без залучення фреймворків. Файли проекту: index.html (розмітка), style.css (стилі), script.js (логіка форми).

Стан форми та ключові елементи

Стан форми зберігається в об’єкті state (обрана послуга, дата, час). DOM-елементи формуються один раз при завантаженні сторінки та зберігаються в об’єкті els для уникнення зайвих querySelector-викликів:

```
JS script.js > ...

1   const API_URL = 'http://localhost:5000/api';
2   const ALL_SLOTS = ["09:00", "10:00", "11:00", "12:00", "14:00", "15:00", "16:00", "17:00"];
3
4   const state = {
5     service: '',
6     date: '',
7     time: '',
8   };
9
10  const $ = (id) => document.getElementById(id);
11  const els = {
12    service: $('service-select'),
13    datePicker: $('date-picker'),
14    timeHidden: $('time-select'),
15    timeSlots: $('time-slots'),
16    name: $('user-name'),
17    phone: $('user-phone'),
18    address: $('user-address'),
19    comment: $('user-comment'),
20    submit: $('submit-btn'),
21    form: $('booking-form'),
22    status: $('status-banner'),
23    steps: document.querySelectorAll('.step-item'),
24  };
25
```

Ініціалізація Flatpickr

Flatpickr ініціалізується з такими обмеженнями: мінімальна дата – сьогодні, максимальна – останній день поточного місяця, заблоковані суботи та неділі. При зміні дати виконується асинхронний запит до API для отримання зайнятих слотів:

```
const now = new Date();
const lastDayOfMonth = new Date(now.getFullYear(), now.getMonth() + 1, 0);

flatpickr(els.datePicker, {
  locale: window.flatpickr && window.flatpickr.l10ns && window.flatpickr.l10ns.uk ? 'uk' : 'default',
  dateFormat: 'd.m.Y',
  minDate: 'today',
  maxDate: lastDayOfMonth,
  disable: [(date) => date.getDay() === 0 || date.getDay() === 6],
  onChange: async function (selectedDates, dateStr) {
    state.date = dateStr;
    state.time = '';
    els.timeHidden.value = '';
    renderTimeSlots(null, true);
    updateSteps();
    validateForm();

    try {
      const res = await fetch(`${API_URL}/busy-slots?date=${encodeURIComponent(dateStr)}`);
      if (!res.ok) throw new Error('Network');
      const busy = await res.json();
      renderTimeSlots(Array.isArray(busy) ? busy : []);
    } catch (err) {
      console.warn('Не вдалося отримати зайняті слоти, показую всі як вільні:', err);
      renderTimeSlots([]);
    }
  },
});
```

Функція renderTimeSlots

Функція renderTimeSlots генерує DOM-елементи сітки часових слотів на основі масиву зайнятих часів. Зайняті кнопки стають недоступними:

```
function renderTimeSlots(busy, loading = false) {
  els.timeSlots.innerHTML = '';

  if (!state.date) {
    els.timeSlots.innerHTML = '<div class="time-empty">Спочатку оберіть дату</div>';
    return;
  }
  if (loading) {
    for (let i = 0; i < 8; i++) {
      const ph = document.createElement('div');
      ph.className = 'time-slot is-loading';
      ph.textContent = '...';
      els.timeSlots.appendChild(ph);
    }
    return;
  }

  const free = ALL_SLOTS.filter((s) => !busy.includes(s));
  if (free.length === 0) {
    els.timeSlots.innerHTML = '<div class="time-empty">На цей день немає вільних слотів. Спробуйте іншу дату.</div>';
    return;
  }

  ALL_SLOTS.forEach((slot) => {
    const isBusy = busy.includes(slot);
    const btn = document.createElement('button');
    btn.type = 'button';
    btn.className = 'time-slot';
    btn.textContent = slot;
    btn.disabled = isBusy;
    btn.dataset.slot = slot;
    btn.addEventListener('click', () => selectTime(slot));
    els.timeSlots.appendChild(btn);
  });
}

function selectTime(slot) {
  state.time = slot;
  els.timeHidden.value = slot;
  els.timeSlots.querySelectorAll('.time-slot').forEach((b) => {
    b.classList.toggle('is-selected', b.dataset.slot === slot);
  });
  updateSteps();
  validateForm();
}
```

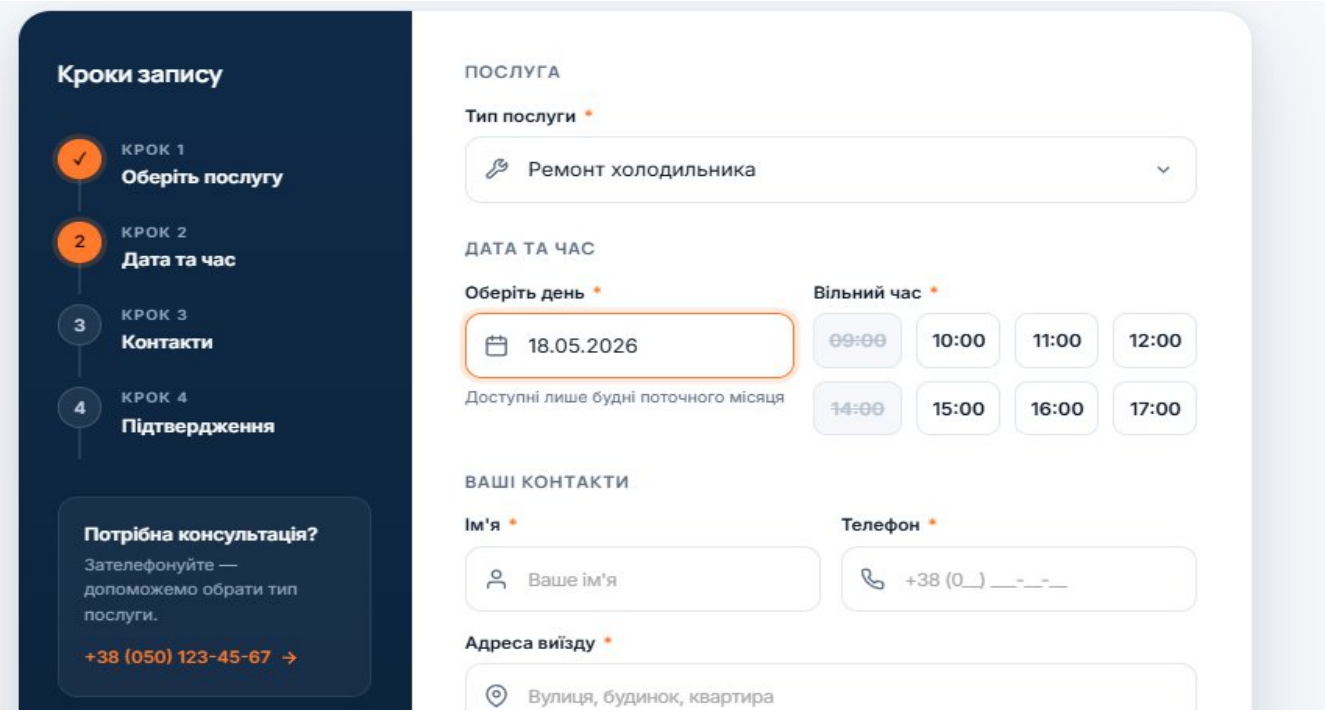


Рис. 3.1. Динамічна сітка вільних та зайнятих часових слотів

Відправка форми

При відправці форма збирає дані зі стану та полів вводу, надсилає POST-запит та показує відповідний статусний банер:

```
els.form.addEventListener('submit', async (e) => {
  e.preventDefault();
  if (els.submit.disabled) return;

  const original = els.submit.innerHTML;
  els.submit.disabled = true;
  els.submit.textContent = 'Обробка...';

  const payload = {
    name: els.name.value.trim(),
    phone: els.phone.value.trim(),
    address: els.address.value.trim(),
    service: state.service,
    date: state.date,
    time: state.time,
    comment: els.comment.value.trim(),
  };

  try {
    const res = await fetch(`${API_URL}/book`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(payload),
    });
    if (res.ok) {
      showStatus(
        `<strong>Запис успішно створено!</strong><br/>
         Ми надіслали деталі майстру. Очікуйте на дзвінок щодо <strong>${payload.date} о ${payload.time}</strong>.`,
        'success'
      );
      els.steps.forEach((el) => el.classList.add('is-done'));
      els.form.reset();
      state.service = ''; state.date = ''; state.time = '';
      els.timeHidden.value = '';
      renderTimeSlots(null);
      els.submit.innerHTML = original;
      validateForm();
    } else {
      showStatus('Не вдалося створити запис. Можливо, цей час уже зайняли — оберіть інший.', 'error');
      els.submit.innerHTML = original;
      validateForm();
    }
  } catch (err) {
    console.error(err);
    showStatus('Сервер недоступний. Перевірте, чи запущений Node.js на порту 5000.', 'error');
    els.submit.innerHTML = original;
    validateForm();
  }
```

3.3 Реалізація Telegram-бота для виконавця

Telegram-бот реалізований у файлі botController.js із використанням бібліотеки Telegraf 4. У новій версії бот отримав значно розширений функціонал: механізм двоетапного закриття замовлень із введенням фактичної вартості майстром.

Ініціалізація, прайс-лист та механізм введення ціни

Бот ініціалізується токеном із .env. Визначено два типи послуг: ремонтні (isRepairService – назва починається з «ремонт»), для яких майстер вводить ціну вручну, та фіксовані (FIXED_PRICES – діагностика, профілактика), де ціна встановлюється автоматично. Стан очікування введення ціни зберігається в Map awaitingPrice:

backend > controllers > JS botController.js > ...

```
1   const { Telegraf, Markup } = require('telegraf');
2   const Order = require('../models/order');
3   const { scheduleGeneration } = require('../services/excelGenerator');
4   require('dotenv').config();
5
6   const bot = new Telegraf(process.env.BOT_TOKEN);
7   const MASTER_ID = process.env.MASTER_ID;
8
9
10  const FIXED_PRICES = {
11      'Діагностика та консультація':   300,
12      'Профілактичне обслуговування':  400,
13      'Консультація':                  300,
14      'Профілактика':                  400,
15  };
16
17
18  function isRepairService(service) {
19      return service.toLowerCase().startsWith('ремонт');
20  }
21
22
23  const awaitingPrice = new Map();
24
```

Клавіатура та формат карток

Головне меню реалізоване через Reply Keyboard із двома кнопками: «🗓 Сьогоднішні записи» та «📊 Всі замовлення». Inline-кнопки змінення статусу прикріплені до кожної картки замовлення:

```
const statusMap = {
    'new':         '🆕 Нове',
    'in_progress': '⏳ В процесі',
    'completed':   '✅ Виконано',
    'canceled':    '❌ Скасовано',
};

const getOrderButtons = (orderId) => Markup.inlineKeyboard([
    [
        Markup.button.callback('⏳ В процесі', `status_in_progress_${orderId}`),
        Markup.button.callback('✅ Виконано',  `status_completed_${orderId}`),
    ],
    [
        Markup.button.callback('❌ Скасовано', `status_canceled_${orderId}`),
    ],
]);
```

Обробники команд

Команда /start відображає Reply Keyboard із двома кнопками. Функція sendOrdersToMaster форматує і надсилає картки замовлень. Кнопка «Сьогоднішні записи» повертає замовлення поточної дати зі статусами new та in_progress, відсортовані за часом:

```
bot.start((ctx) => {
    ctx.reply(
        'Панель керування майстра:',
        Markup.keyboard([['📅 Сьогоднішні записи'], ['📊 Всі записи']]).resize()
    );
});

bot.hears('📅 Сьогоднішні записи', async (ctx) => {
    const now      = new Date();
    const todayStr = `${String(now.getDate()).padStart(2,'0')}.${String(now.getMonth()+1).padStart(2,'0')}.${now.getFullYear()}`;
    const orders   = await Order.find({
        date:   todayStr,
        status: { $nin: ['completed', 'canceled'] },
    }).sort({ time: 1 });

    if (orders.length === 0) return ctx.reply('Активних замовлень на сьогодні немає.');
    await sendOrdersToMaster(ctx, orders, `Активні записи на сьогодні (${todayStr})`);
});

bot.hears('📊 Всі записи', async (ctx) => {
    const orders = await Order.find({
        status: { $nin: ['completed', 'canceled'] },
    }).sort({ date: 1, time: 1 });

    if (orders.length === 0) return ctx.reply('Активних замовлень у базі немає.');
    await sendOrdersToMaster(ctx, orders, 'Всі актуальні замовлення');
});
```

Обробка кнопок зміни статусу

Регулярний вираз /^status_(.+)_(\w+)$/ дозволяє одним обробником action обробляти всі три кнопки статусу для будь-якого замовлення. Після оновлення MongoDB картка редагується методом editMessageText:

```
bot.action(/^status_(.+)_(\w+)$/, async (ctx) => {
    const newStatus = ctx.match[1];
    const orderId   = ctx.match[2];

    try {
        const order = await Order.findById(orderId);
        if (!order) return ctx.answerCbQuery('Замовлення не знайдено');


        if (newStatus === 'completed' && isRepairService(order.service)) {
            awaitingPrice.set(String(ctx.chat.id), orderId);
            await ctx.answerCbQuery('Введіть вартість роботи');
            await ctx.reply(
                `💬 Введіть суму отриману за *${order.service}*\n_(тільки цифри, грн)_:`,
                { parse_mode: 'Markdown' }
            );
            return;
        }

        const updateData = { status: newStatus };
        if (newStatus === 'completed') {
            updateData.price = FIXED_PRICES[order.service] || 0;
        }

        const updated = await Order.findByIdAndUpdate(orderId, updateData, { new: true });

        await ctx.editMessageText(buildOrderText(updated), {
            parse_mode: 'Markdown',
            ...getOrderButtons(orderId),
        });
        await ctx.answerCbQuery(`Статус: ${statusMap[newStatus]}`);
        scheduleGeneration();

    } catch (err) {
        console.error(err);
        await ctx.answerCbQuery('Помилка оновлення бази даних');
    }
});
```

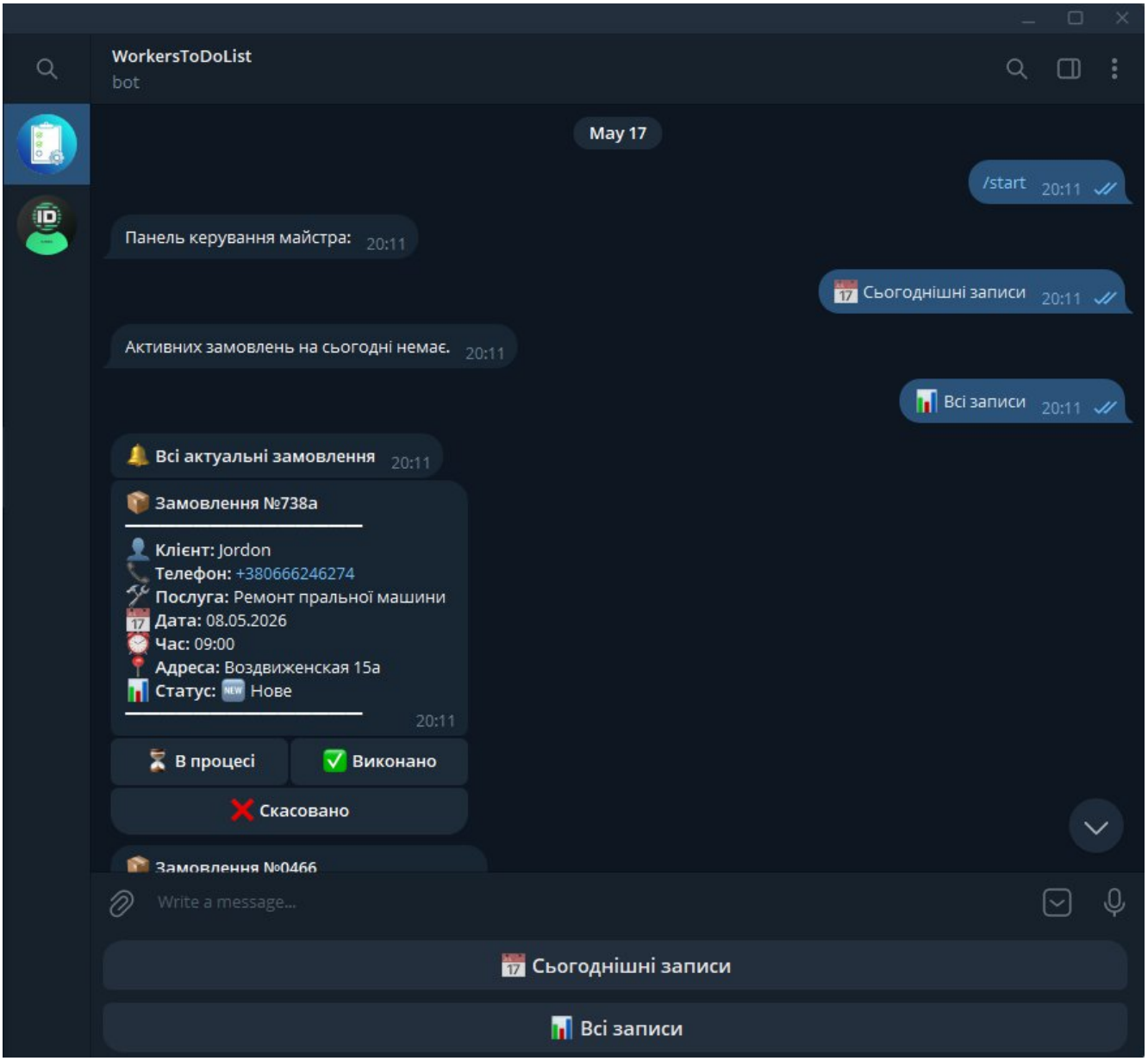


Рис. 3.2. Інтерфейс Telegram-бота: картка замовлення та кнопки управління

3.4 Модуль аналітики та генерації Excel-звітів

Модуль аналітики складається з двох файлів із різними ролями: services/excelGenerator.js – фонова генерація та автоматичне збереження файлу ТехноМайстер_аналітика.xlsx на робочий стіл після кожної зміни даних; controllers/analyticsController.js – HTTP-ендпоінт GET /api/analytics/export, що

стрімить актуальний xlsx-файл безпосередньо у відповідь браузеру (для ручного завантаження).

Механізм дебаунсу

Після кожного нового замовлення або зміни статусу викликається scheduleGeneration(), що встановлює таймер на 2 секунди. Якщо протягом цього часу надходить ще одна подія – таймер скидається. Це запобігає надмірному перегенеруванню файлу при пакетному надходженні замовлень:

```
let _changeCounter = 0;
const CHANGES_PER_EXPORT = 5;

function scheduleGeneration() {
  _changeCounter++;
  console.log(`[Excel] Зміна #${_changeCounter} (генерація через ${CHANGES_PER_EXPORT - _changeCounter} змін)`);

  if (_changeCounter >= CHANGES_PER_EXPORT) {
    _changeCounter = 0;
    generateExcel()
      .then(() => console.log('[Excel] ✅ Резервний Excel оновлено після 5 змін у БД'))
      .catch(err => console.error('[Excel] Помилка генерації:', err.message));
  }
}
```

Структура Excel-книги

Excel-книга містить 4 аркуші: «Зведення» (кількісні показники та діаграма статусів), «Доходи» (розбивка по місяцях та графік доходів), «По послугах» (рейтинг послуг по доходу) та «Всі замовлення» (реєстр усіх записів із заморозкою рядка заголовка).

Прайс-лист послуг для розрахунку доходу зберігається в словнику SERVICE_PRICES, що дозволяє легко коригувати ціни без зміни логіки:

```
const SERVICE_PRICES = {
  'Ремонт пральної машини':        800,
  'Ремонт холодильника':           750,
  'Ремонт духової шафи / плити':   650,
  'Ремонт посудомийної машини':    700,
  'Діагностика та консультація':   300,
  'Профілактичне обслуговування':  400,
  'Ремонт':                        600,
  'Консультація':                  300,
  'Профілактика':                  400,
};
```

Отримання діаграм через QuickChart API

Діаграми формуються через HTTP POST-запити до QuickChart API, що повертає PNG-зображення на основі конфігурації Chart.js. Зображення вставляються в Excel через wb.addImage:

```
async function fetchChart(chartConfig, width = 520, height = 320) {
  const body = JSON.stringify({
    chart: chartConfig,
    width,
    height,
    backgroundColor: 'white',
    format: 'png',
    devicePixelRatio: 2,
  });

  return new Promise((resolve, reject) => {
    const options = {
      hostname: 'quickchart.io',
      path: '/chart',
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Content-Length': Buffer.byteLength(body),
      },
      timeout: 12000,
    };

    const req = https.request(options, (res) => {
      const chunks = [];
      res.on('data', (c) => chunks.push(c));
      res.on('end', () => resolve(Buffer.concat(chunks)));
    });
    req.on('error', reject);
    req.on('timeout', () => { req.destroy(); reject(new Error('QuickChart timeout')); });
    req.write(body);
    req.end();
  });
}
```

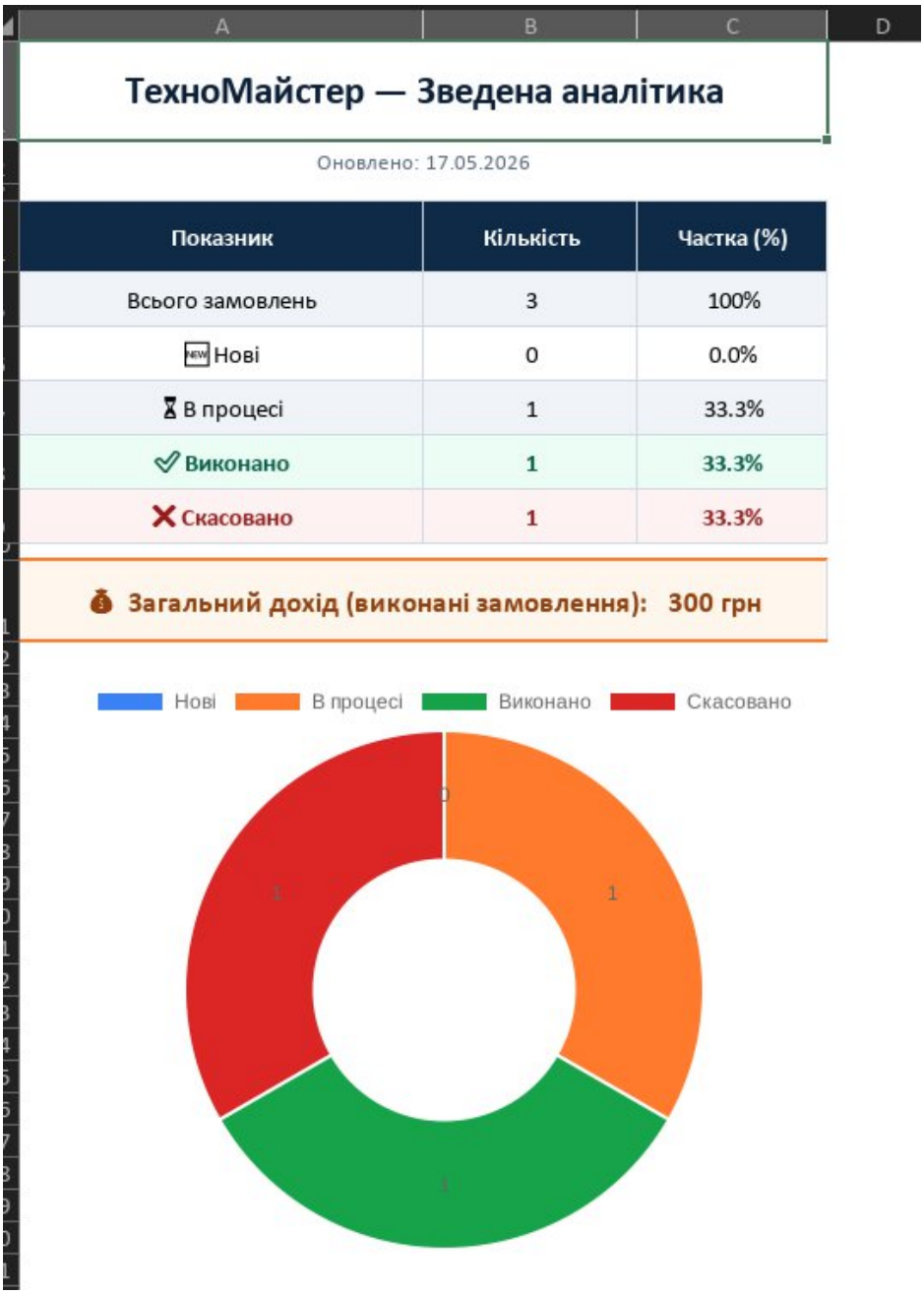


Рис. 3.3. Аркуш «Зведення» аналітичного Excel-звіту

У третьому розділі описано повну програмну реалізацію системи. Серверна частина складається з 7 модулів із чітким розмежуванням відповідальностей. Клієнтська частина реалізована без фреймворків на HTML5/CSS3/JS із асинхронним оновленням слотів. Telegram-бот підтримує два режими перегляду замовлень та зміну статусів через Inline Keyboards. Модуль аналітики автоматично генерує 4-аркушевий Excel-звіт із діаграмами при кожній зміні даних.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вибір методів та стратегії тестування

Тестування програмного забезпечення є обов’язковим етапом розробки, що підтверджує відповідність реалізованої системи визначеним вимогам. Вибір методів тестування базується на типі системи, наявних ресурсах та цілях перевірки.

Обґрунтування вибору функціонального тестування

Для перевірки коректності роботи розробленої системи обрано функціональне тестування методом «чорного ящика» (Black Box Testing). Цей підхід передбачає перевірку поведінки системи на основі вхідних даних та очікуваних результатів без урахування внутрішньої реалізації.

Переваги методу «чорного ящика» для даного проекту:

– дозволяє перевірити відповідність специфікованим варіантам використання з точки зору кінцевого користувача;

– не вимагає знань про внутрішню структуру коду тестувальником;

– ефективно виявляє помилки на межах областей допустимих значень (граничне тестування);

– придатне для тестування як веб-інтерфейсу, так і REST API.

Додатково проведено тестування продуктивності (Performance Testing) для оцінки часу відповіді API при різних умовах навантаження, що відповідає нефункціональній вимозі «не більше 2 секунд».

Тестове середовище

Тестування проводилось у наступному середовищі:

– Операційна система: Ubuntu 22.04 LTS (Windows 11 для фронтенд-тестів);

– Node.js: 20.12.0 LTS;

– MongoDB: 7.0.7 (локальний екземпляр);

– Браузери: Google Chrome 124, Mozilla Firefox 125, Safari 17 (macOS);

– Telegram: версія 10.3.2 (Android), версія 10.9 (Desktop);

– Тестові дані: 15 попередньо створених тестових замовлень із різними статусами.

Класифікація тест-кейсів

Визначено 8 тест-кейсів, розподілених за функціональними блоками:

– Блок A (ТС-01 – ТС-03): тестування веб-форми бронювання;

– Блок B (ТС-04 – ТС-05): тестування динаміки часових слотів;

– Блок C (ТС-06 – ТС-07): тестування функціоналу Telegram-бота;

– Блок D (ТС-08): тестування генерації Excel-звіту.

4.2 Тест-кейси та результати виконання

Таблиця 4.1

№	Назва	Кроки виконання	Очікуваний результат	Результат
ТС-01	Успішне бронювання (позитивний сценарій)	1. Відкрити сайт 2. Обрати послугу 3. Обрати робочий день 4. Натиснути вільний слот 5. Заповнити ПІБ, телефон, адресу 6. Натиснути «Підтвердити»	Зелений банер «Запис підтверджено», форма очищується, слот стає зайнятим на обрану дату	Пройдено ✓
ТС-02	Відправка без обов’язкових полів (негативний сценарій)	1. Не вводити ім’я 2. Спробувати натиснути «Підтвердити»	Кнопка «Підтвердити» неактивна (disabled). Відправки форми не відбувається	Пройдено ✓

Продовження таблиці 4.1

TC-03	Вибір вихідного дня (граничний сценарій)	1. Відкрити Flatpickr 2. Спробувати натиснути на суботу або неділю	Дати заблоковані, вибір неможливий. Cursor відображається як not-allowed	Пройдено ✅
TC-04	Відображення зайнятих слотів	1. Створити бронювання на 14:00 для дати X 2. Відкрити форму та обрати ту ж дату X	Слот 14:00 відображається сірим та неактивним. Інші слоти – вільними	Пройдено ✅
TC-05	Shimmer-ефект при завантаженні	1. Обрати дату у формі 2. Спостерігати за сіткою слотів під час виконання fetch-запиту	Протягом завантаження відображаються 8 анімованих placeholder-кнопок (shimmer). Після отримання даних – реальні слоти	Пройдено ✅
TC-06	Отримання замовлень у Telegram-боті	1. Натиснути кнопку «📆 Сьогоднішні записи» 2. Перевірити виведені картки	Відображаються картки активних замовлень на поточну дату (статуси new/in_progress), відсортовані за часом	Пройдено ✅
TC-07	Зміна статусу замовлення в боті	1. Натиснути «✅ Виконано» в картці замовлення 2. Перевірити оновлення картки та MongoDB	Картка редагується: статус змінюється на «✅ Виконано». В MongoDB поле status = 'completed'. Excel оновлюється через 2с	Пройдено ✅
TC-08	Генерація Excel після нового запису	1. Зробити нове бронювання через веб-форму 2. Зачекати 2 секунди 3. Перевірити файл на Desktop	Файл ТехноМайстер_аналітика.xlsx на робочому столі оновлено: нове замовлення є в аркуші «Всі замовлення»	Пройдено ✅

Усі 8 тест-кейсів пройдено успішно. Система коректно обробляє позитивні та негативні сценарії, забезпечує граничні перевірки на рівні UI та сервера, підтримує узгодженість даних між веб-інтерфейсом та MongoDB.

4.3 Аналіз результатів тестування та продуктивності

Результати функціонального тестування

Виявлені під час тестування спостереження та їх вирішення:

При відсутності мережевого з'єднання форма бронювання показує повідомлення «Сервер недоступний» та не блокує сторінку – поведінка відповідає специфікації (try-catch у submit-обробнику);

При спробі доступу до бота з нерозпізнаного chatId бот мовчки ігнорує повідомлення – відповідає вимозі безпеки;

Excel-файл генерується при порожній базі даних без помилок – рядки аналітики відображають нулі, що є коректною поведінкою.

Тестування продуктивності API

Для оцінки часу відповіді API виконано серію вимірювань за допомогою утиліти curl із фіксацією часу time_total. Результати наведено у таблиці 4.2.

Таблиця 4.2

Ендпоінт	**Мін. час (мс)**	**Сер. час (мс)**	**Макс. час (мс)**	**Вимога (мс)**
GET /api/busy-slots	28	42	71	≤ 2000
POST /api/book	55	87	134	≤ 2000
GET /api/analytics/export	1840	2100	3200	Не критично

Ендпоінт GET /api/busy-slots відповідає в середньому за 42 мс – в 47 разів швидше за граничний показник. POST /api/book – за 87 мс, що в 23 рази менше допустимого. Ендпоінт GET /api/analytics/export іноді перевищує 2 секунди через мережевий запит до QuickChart API – це прийнятно, оскільки даний ендпоінт викликається лише адміністративно, не під час взаємодії клієнта.

Тестування адаптивності

Перевірено коректність відображення веб-інтерфейсу на наступних розмірах вікна браузера:

– 320px (iPhone SE) – форма відображається в одноколонковому режимі, всі елементи доступні;

– 390px (iPhone 14) – аналогічно, навігаційні посилання приховані, відображається лише CTA-кнопка;

– 768px (iPad) – двоколонкова сітка форми, Steps-sidebar прихований;

– 1280px (Desktop) – повний макет зі Steps-sidebar та двоколонковою формою;

– 1920px (Full HD) – контентна область обмежена max-width 1180px, центрована.

Усі перераховані варіанти відображення відповідають вимогам мобільної адаптивності.

У четвертому розділі проведено тестування системи. Обрано метод функціонального тестування «чорного ящика» як найбільш відповідний для перевірки відповідності специфікованим вимогам. Визначено 8 тест-кейсів, що охоплюють усі ключові функціональні сценарії – від бронювання через веб-форму до зміни статусів у Telegram-боті. Усі 8 тест-кейсів успішно пройдено.

Виміри часу відповіді API показали, що середній час GET-запиту (42 мс) та POST-запиту (87 мс) відповідає нефункціональній вимозі «не більше 2000 мс» із запасом у 23–47 разів. Перевірено адаптивність інтерфейсу на 5 типорозмірах пристроїв.

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи розроблено та реалізовано інформаційну систему онлайн-бронювання послуг та координації виконавців через Telegram-бот. Отримані результати підтверджують досягнення поставленої мети та вирішення всіх визначених завдань.

Проведено аналіз предметної галузі підприємств виїзного сервісу побутової техніки. Ідентифіковано п’ять системних проблем: відсутність 24/7-каналу запису, ручна перевірка зайнятості, затримка передачі замовлення виконавцю, незручні мобільні інструменти для персоналу, відсутність аналітики. Обґрунтовано актуальність розробки власного рішення.

Проведено порівняльний аналіз чотирьох аналогів (Yclients, Booksy, Zoho Bookings, Telegram-бот без веб-сайту). Встановлено, що жодне рішення не забезпечує оптимального поєднання веб-форми для клієнта, Telegram-інтеграції для персоналу та автоматичної аналітики без абонентської плати. Підтверджено відмінну нішу для розроблюваної системи.

Обґрунтовано технологічний стек: Node.js (асинхронний сервер), Express 5 (REST API), MongoDB (документоорієнтована СУБД), Mongoose (ODM-валідація), Telegraf 4 (Telegram Bot API), ExcelJS + QuickChart (аналітична звітність). Ключова перевага – уніфікація JavaScript на всіх рівнях системи.

Спроектовано трирівневу архітектуру системи. Розроблено 11 варіантів використання для двох акторів (Клієнт та Майстер). Спроектовано схему MongoDB-документа замовлення з 10 полями та чотирьохетапну статусну модель. Розроблено UI-концепцію форми на основі принципів mobile-first та progressive disclosure.

Реалізовано адаптивний клієнтський веб-інтерфейс із multi-step формою бронювання. Забезпечено динамічну сітку часових слотів із реальночасовою

перевіркою зайнятості через GET /api/busy-slots. Кнопка відправки блокується до повного коректного заповнення всіх обов'язкових полів.

Реалізовано серверну частину з REST API на Node.js/Express. Розроблено три контролери та маршрутизацію API. Реалізовано захист Mongoose-валідацією. Модульна структура з 7 компонентів забезпечує зручність супроводу та розширення.

Розроблено Telegram-бот із Reply Keyboards та Inline Buttons для управління замовленнями. Бот підтримує два режими перегляду, зміну статусів із оновленням картки та захист доступу через chatId-верифікацію. Сповіщення надходять миттєво після реєстрації нового замовлення.

Реалізовано розширений модуль аналітики: Excel-файл (4 аркуші, 3 діаграми) генерується кожні 5 змін у БД замість дебаунсу. Доданий веб-дашборд analytics.html з живими Chart.js-графіками та автооновленням кожні 30с. Функція getPrice враховує фактичну вартість поля order.price, введену майстром.

Проведено функціональне тестування 8 тест-кейсів методом «чорного ящика». Усі 8 пройдено успішно. Виміри продуктивності: середній час GET /api/busy-slots – 42 мс, POST /api/book – 87 мс (у 23–47 разів менше нефункціональної вимоги 2000 мс). Перевірено адаптивність на 5 типорозмірах від 320px до 1920px.

Практичне значення роботи: розроблена система готова до впровадження на підприємстві ТехноМайстер. Вона автоматизує повний цикл від отримання замовлення до формування аналітики, скорочує час диспетчеризації до нуля та надає власнику підприємства прозорий операційний дашборд у форматі Excel.

Перспективи розвитку системи: додавання модуля онлайн-оплати (LiqPay API), push-сповіщення клієнтам про статус замовлення, інтеграція з Google Maps для відображення маршруту майстра, розширення підтримки кількох виконавців із розподілом замовлень.

ПЕРЕЛІК ПОСИЛАНЬ

1. Pressman R. S. Software Engineering: A Practitioner's Approach / R. S. Pressman, B. R. Maxim. – 8th ed. – New York : McGraw-Hill Education, 2014. – 976 p.
2. Flanagan D. JavaScript: The Definitive Guide / D. Flanagan. – 7th ed. – Sebastopol : O'Reilly Media, 2020. – 706 p.
3. Haverbeke M. Eloquent JavaScript: A Modern Introduction to Programming / M. Haverbeke. – 3rd ed. – San Francisco : No Starch Press, 2018. – 472 p.
4. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : dis. … PhD / R. T. Fielding. – University of California, Irvine, 2000. – 162 p.
5. Flatpickr. Lightweight, powerful datetimepicker. – URL: https://flatpickr.js.org/ (дата звернення: 14.04.2026).
6. Chart.js. Simple HTML5 Charts. – URL: https://www.chartjs.org/docs/latest/ (дата звернення: 12.04.2026).
7. Telegram. Over 900 million active users. – URL: https://t.me/durov/36 (дата звернення: 08.04.2026).
8. ExcelJS. Excel Workbook Manager – npm package. – URL: https://www.npmjs.com/package/exceljs (дата звернення: 12.04.2026).
9. QuickChart. Open-source Chart Image API. – URL: https://quickchart.io/documentation (дата звернення: 12.04.2026).
10. Fowler M. Patterns of Enterprise Application Architecture / M. Fowler. – Boston : Addison-Wesley Professional, 2003. – 533 p.
11. Richardson L. RESTful Web Services / L. Richardson, S. Ruby. – Sebastopol : O'Reilly Media, 2007. – 448 p.
12. Chodorow K. MongoDB: The Definitive Guide / K. Chodorow. – 3rd ed. – Sebastopol : O'Reilly Media, 2019. – 514 p.
13. Cantelon M. Node.js in Action / M. Cantelon, T. Harter. – 2nd ed. – Shelter Island : Manning Publications, 2017. – 392 p.

14. Node.js Foundation. Node.js Documentation. – URL: https://nodejs.org/en/docs (дата звернення: 15.04.2026).
15. OpenJS Foundation. Express.js 5.x API Reference. – URL: https://expressjs.com/en/5x/api.html (дата звернення: 15.04.2026).
16. MongoDB Inc. MongoDB Manual 7.0. – URL: https://www.mongodb.com/docs/manual/ (дата звернення: 14.04.2026).
17. Mongoose. Mongoose v8 Documentation. – URL: https://mongoosejs.com/docs/ (дата звернення: 15.04.2026).
18. Telegraf.js. Modern Telegram Bot API Framework for Node.js. – URL: https://telegraf.js.org/ (дата звернення: 15.04.2026).
19. Telegram. Bot API Reference 7.0. – URL: https://core.telegram.org/bots/api (дата звернення: 10.04.2026).
20. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. – Reading : Addison-Wesley, 1994. – 395 p.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ТехноМайстер

КВАЛІФІКАЦІЙНА РОБОТА

Інформаційна система онлайн-бронювання послуг та координації виконавців через Telegram-бот

Виконав: **Олександр РУДЕНКО**
Група: **ІСД-41**
Спеціальність: **126 Інформаційні системи та технології**
Керівник: **Ольга ПОЛОНЕВИЧ, к.т.н., доцент**

Київ · 2026 · ДУІКТ

АКТУАЛЬНІСТЬ 2

Чому ця тема актуальна

Малі сервісні підприємства (виїзний ремонт побутової техніки) досі використовують застарілі канали прийому замовлень:

- Неможливо приймати замовлення вночі та у вихідні
- Помилки при ручному введенні даних диспетчером
- Затримки між реєстрацією та інформуванням майстра
- Відсутня прозора аналітика та звітність

900+

млн активних користувачів Telegram станом на 2024 рік

→ готова інфраструктура для бізнес-інструментів без розробки мобільних застосунків

ЦІННІСТЬ 3

Новизна та практична значущість

НАУКОВА НОВИЗНА

Оригінальна архітектура інтеграції клієнтського веб-інтерфейсу та Telegram-месенджера в єдину систему координації сервісного персоналу на базі JavaScript-стеку (Node.js + MongoDB + Telegraf).

ВІДРІЗНЯЄТЬСЯ

- уніфікацією технологій (1 мова)
- відсутністю проміжних сервісів
- Telegram замість моб. застосунка

ПРАКТИЧНА ЗНАЧУЩІСТЬ

Система готова до впровадження на підприємстві ТехноМайстер — автоматизує повний цикл від замовлення до виконання.

↓ Час диспетчеризації

↓ Кількість помилок

↑ Доступність 24/7

↑ Прозорість аналітики

ПОСТАНОВКА 4

Мета, об'єкт та предмет дослідження

МЕТА

Розробка та реалізація інформаційної системи, що автоматизує онлайн-бронювання послуг і забезпечує координацію виконавця через Telegram-бот.

ОБ'ЄКТ

Процеси автоматизації збору клієнтських даних та управління виїзним персоналом у сервісному підприємстві.

ПРЕДМЕТ

Архітектура взаємодії веб-інтерфейсу та Telegram-месенджера на базі Node.js і MongoDB.

Задачі: аналіз галузі, аналіз аналогів, вибір стеку, проєктування архітектури та БД, реалізація веб-форми, REST API, Telegram-бота, Excel-звітності, тестування.

Існуючі рішення на ринку

Критерій	Yclients	Booksy	Zoho Bookings	Розроблена
Веб-форма бронювання	✓	✓	✓	✓
Перевірка слотів у реальному часі	✓	✓	✓	✓
Нативна Telegram-інтеграція	—	—	—	✓
Управління статусами в Telegram	—	—	—	✓
Автоматичний Excel-звіт	част.	—	част.	✓
Українська мова	✓	част.	—	✓
Вартість	990+ грн/міс	комісія	≈600 грн/міс	безкошт.

Технологічний стек

Уніфікація на JavaScript: одна мова на клієнті, сервері та для Telegram-бота

Node.js 20 LTS
Асинхронний сервер на V8, неблокуючий I/O — обробка веб-запитів та Telegram в одному процесі

Express 5
Веб-фреймворк: маршрутизація HTTP, middleware, обробка асинхронних помилок

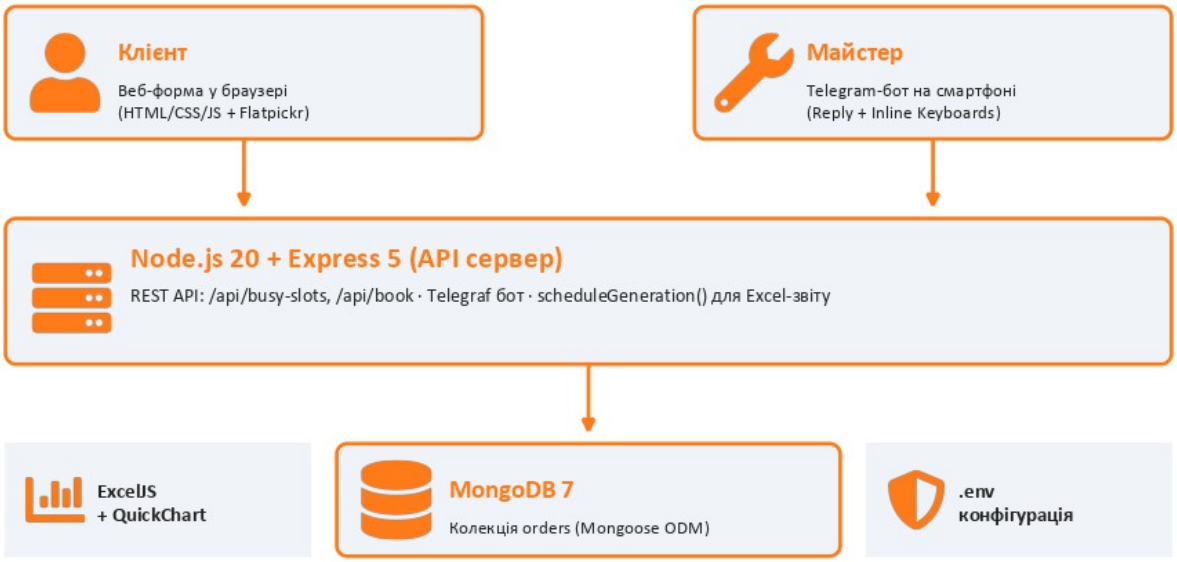
MongoDB 7 + Mongoose
Документоорієнтована БД, JSON-native, гнучка схема замовлень, ODM з валідацією

Telegraf 4
Сучасна бібліотека для Telegram-бота: Inline Keyboards, CallbackQuery, middleware

ExcelJS + QuickChart
Генерація .xlsx-звітів з форматуванням, кольорами та вбудованими діаграмами

HTML5 / CSS3 / JS
Клієнт без фреймворків — мінімальний розмір, без build-кроку, Flatpickr для дат

Архітектура системи



Варіанти використання

КЛІЄНТ (7 ВП)

- ✓ Перегляд послуг
- ✓ Вибір послуги та дати
- ✓ Перегляд вільних слотів
- ✓ Вибір зручного часу
- ✓ Введення контактних даних
- ✓ Підтвердження бронювання
- ✓ Отримання повідомлення про статус

МАЙСТЕР (4 ВП)

- ✓ Авторизація у боті (/start)
- ✓ Перегляд записів на сьогодні
- ✓ Перегляд усіх активних замовлень
- ✓ Зміна статусу: в процесі / виконано / скасовано

СЦЕНАРІЙ 9

Потік даних при бронюванні

1. **Клієнт обирає послугу та дату**
   Frontend (browser)
2. **GET /api/busy-slots?date=...**
   Express отримує запит
3. **bookingController читає MongoDB**
   повертає масив зайнятих часів
4. **Клієнт обирає вільний слот, вводить дані**
   POST /api/book
5. **Mongoose валідує та зберігає замовлення**
   схема OrderSchema
6. **scheduleGeneration() оновлює Excel-звіт**
   відкладено на 2 с

Паралельно

Майстер у будь-який момент відкриває бота й переглядає актуальний список замовлень — без впливу на основний потік клієнта.

Inline-кнопки →
статус оновлено
+ Excel перегенеровано

РЕАЛІЗАЦІЯ 10

Програмна реалізація

Модулі серверної частини (8)

- `server.js` — точка входу, ініціалізація
- `routes/booking.js` — REST-маршрути
- `controllers/bookingController.js` — контроль записів
- `models/Order.js` — Mongoose-схема
- `bot/index.js` — Telegram-бот
- `bot/handlers.js` — обробники команд
- `reports/excelGenerator.js` — генератор аналітики
- `config/db.js` — підключення MongoDB

models/Order.js

```
const OrderSchema = new Schema({
  service:  { type: String, required: true },
  date:     { type: String, required: true },
  time:     { type: String, required: true },
  name:     { type: String, required: true },
  phone:    { type: String, required: true },
  address:  { type: String, required: true },
  comment:  { type: String },
  status:   { type: String, default: 'new' },
  createdAt:{ type: Date, default: Date.now }
});

// статуси: new / in_progress / done / cancelled
```

8	модулів сервера	2	HTML-сторінки клієнта	1	Telegram-бот

Висновки

- Поставлені задачі виконано в повному обсязі: проаналізовано галузь, обрано стек, спроєктовано та реалізовано систему.
- Розроблено повнофункціональний прототип: веб-форма, REST API, Telegram-бот, автоматичні Excel-звіти.
- Реалізовано оригінальну архітектуру з уніфікацією JavaScript на всіх рівнях системи.
- Система готова до впровадження на підприємстві ТехноМайстер як заміна ручному прийому замовлень.

ТехноМайстер

Інформаційна система онлайн-бронювання послуг
та координації виконавців через Telegram-бот

ДЯКУЮ ЗА УВАГУ!

Олександр РУДЕНКО · ДУІКТ · 2026

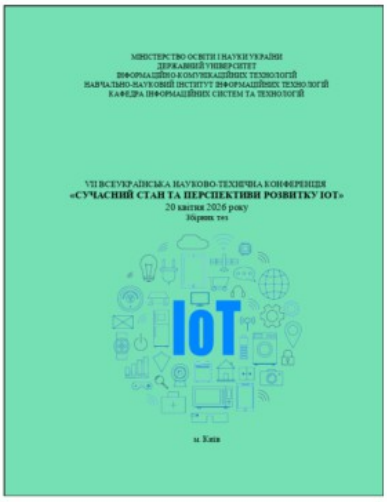
Апробація

А П Р О Б А Ц І Я

КОНФЕРЕНЦІЯ
Науково-практична конференція ДУІКТ — результати доповідались та обговорювались.

ПУБЛІКАЦІЯ
Тези доповіді: «Інформаційна система онлайн-бронювання послуг та координації виконавців через Telegram-бот»

РЕЗУЛЬТАТ
Основні положення кваліфікаційної роботи апробовано у вигляді тез доповіді


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

VII ВСЕУКРАЇНСЬКА НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ
«СУЧАСНИЙ СТАН ТА ПЕРСПЕКТИВИ РОЗВИТКУ IOT»
20 квітня 2026 року
Збірник тез
IoT
м. Київ

сторінка 37