

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 «Веб-технології»)
на тему:

«Вебсистема аукціонів авто»

Виконав студент групи ВТ-22-2
ОСТАПЕНКО Владислав Олександрович

Керівник роботи:
ДУНЄВ Сергій Сергійович

Рецензент:
ЄФРЕМОВ Юрій Миколайович

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: **ОСТАПЕНКО Владислав Олександрович**

Керівник роботи: **ДУНЄВ Сергій Сергійович**

Тема роботи: **«Вебсистема аукціонів авто»**,

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: **використання веборієнтованих технологій для розробки системи аукціонів авто.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Дунєв С. С.	02.03.2026р.	02.03.2026р.
2	Дунєв С. С.	06.04.2026р.	06.04.2026р.
3	Дунєв С. С.	04.05.2026р.	04.05.2026р.

РЕФЕРАТ

Метою кваліфікаційної роботи є розробка вебсистеми аукціонів авто, яка забезпечує автоматизацію процесів купівлі-продажу транспортних засобів через інтернет. У роботі проведено аналіз існуючих онлайн-платформ для продажу автомобілів, визначено функціональні та нефункціональні вимоги до системи, спроектовано архітектуру програмного продукту на основі клієнт-серверного підходу та реалізовано основні модулі вебзастосунку.

У межах роботи реалізовано реєстрацію та автентифікацію користувачів, створення аукціонів, подання ставок, автоматичне завершення торгів, визначення переможця, двосторонній чат між учасниками, особисті кабінети покупця і продавця, а також генерацію інвойсів. Для реалізації використано сучасний технологічний стек: Next.js, TypeScript і Tailwind CSS для клієнтської частини, NestJS, TypeORM, JWT і Socket.IO для серверної частини, PostgreSQL для збереження даних, Docker і Docker Compose для розгортання програмного комплексу.

Розроблена система протестована на коректність виконання основних функціональних сценаріїв, стабільність роботи та базові механізми безпеки. Результатом роботи є готовий до локального розгортання програмний продукт, який може бути використаний приватними користувачами або невеликими автосалонами для організації онлайн-аукціонів авто. Пояснювальна записка містить 73 сторінок, 21 ілюстрацію, 9 таблиць, 30 джерел.

Ключові слова: АУКЦІОН, ВЕБСИСТЕМА, NEXT.JS, NESTJS, POSTGRESQL, SOCKET.IO.

					ІПЗ.КР.Б - 121 - 26 - ПЗ					
Змн.	Арк.	№ докум.	Підпис	Дата						
Розроб.		Остапенко В. О.			Веб система аукціонів авто Пояснювальна записка			Літ.	Арк.	Аркушів
Керівник.		Дунєв С. С.							3	73
Рецензент		Єфремов Ю.М.						Житомирська політехніка, група ВТ-22-2		
Зав. каф.		Вакалюк Т.А								

ABSTRACT

The purpose of this bachelor's thesis is to develop a web-based car auction system that automates the process of buying and selling vehicles online. The thesis includes an analysis of existing online car sales platforms, identification of functional and non-functional requirements, design of a client-server architecture, and implementation of the main modules of the web application.

The developed system includes user registration and authentication, auction creation, bidding, automatic auction closing, winner determination, bidirectional messaging between participants, personal dashboards for buyers and sellers, and invoice generation. The system is implemented using a modern technology stack: Next.js, TypeScript, and Tailwind CSS for the client side, NestJS, TypeORM, JWT, and Socket.IO for the server side, PostgreSQL for data storage, and Docker with Docker Compose for deployment.

The application has been tested for correct execution of the main functional scenarios, stability, and basic security mechanisms. The result is a ready-to-deploy software product that can be used by private users or small auto dealerships to organize online car auctions. The explanatory note consists of 73 pages, 21 illustrations, 9 tables, and 30 references.

Keywords: AUCTION, WEB SYSTEM, NEXT.JS, NESTJS, POSTGRESQL, SOCKET.IO.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ	9
1.1 Аналіз аналогів програмного продукту	9
1.2. Постановка задачі та аналіз аналогів програмного продукту	15
1.3. Вибір архітектури вебсистеми	19
1.4. Вибір інструментальних засобів та вимоги до апаратного забезпечення	25
Висновки до першого розділу	33
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ	35
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	35
2.2. Розробка бази даних системи	40
2.3. Проєктування та реалізація алгоритмів роботи системи	44
2.4. Реалізація додатку веб-орієнтованої системи аукціонів авто	48
Висновки до другого розділу	53
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ	55
3.1. Порядок встановлення та налаштування параметрів системи через Docker	55
3.2. Структура інтерфейсу вебсистеми аукціонів авто	59
3.3. Тестування роботи програмного комплексу	64
Висновки до третього розділу	69
ВИСНОВКИ	71
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	73
ДОДАТКИ	76

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API – Application Programming Interface – інтерфейс програмування додатків

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

CSS – Cascading Style Sheets - мова опису стилів вебсторінок

DB – Database - база даних

DTO – Data Transfer Object - об'єкт передачі даних

HTML – HyperText Markup Language - мова розмітки гіпертексту

HTTP – HyperText Transfer Protocol - протокол передачі гіпертексту

JWT – JSON Web Token - стандарт безпечного обміну даними

ORM – Object-Relational Mapping - об'єктно-реляційне відображення

REST – Representational State Transfer - архітектурний стиль взаємодії систем

SQL – Structured Query Language - мова запитів до реляційних баз даних

UI – User Interface - інтерфейс користувача

UX – User Experience - досвід користувача

UML – Unified Modeling Language - уніфікована мова моделювання

URL – Uniform Resource Locator - уніфікований покажчик ресурсу

UUID – Universally Unique Identifier – універсальний унікальний ідентифікатор

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Сучасний ринок продажу транспортних засобів активно переходить у цифровий простір. Традиційні методи, такі як особисті зустрічі або оголошення на спеціалізованих майданчиках, поступово замінюються інтернет-платформами, що забезпечують швидший доступ до інформації про автомобілі та спрощують взаємодію між учасниками угоди. Водночас наявні рішення, зокрема Copart, AutoBidMaster та eBay Motors [1–3], часто орієнтовані на комерційних учасників ринку, автосалони, дилерів або професійних перекупників. Це створює потребу у доступній вебсистемі, яка дає змогу приватним користувачам створювати аукціони, брати участь у торгах, отримувати інформацію про перемогу, спілкуватися з іншою стороною угоди та завантажувати інвойс.

Актуальність обраної теми полягає у тому, що автоматизація процесу купівлі-продажу автомобілів через аукціонну модель дає змогу підвищити прозорість ціноутворення, зменшити ризики шахрайства, забезпечити зручну комунікацію між покупцем і продавцем, а також автоматизувати частину документообігу шляхом формування інвойсів. Крім того, веборієнтований формат системи робить процес купівлі-продажу доступним для широкого кола користувачів без необхідності залучення посередників або використання складних комерційних платформ.

Метою даної кваліфікаційної роботи є розробка повноцінної вебсистеми аукціонів авто, яка реалізує повний життєвий цикл аукціону: від створення та подання ставок до автоматичного завершення, визначення переможця, двостороннього спілкування та генерації офіційного інвойсу.

Для досягнення поставленої мети у роботі передбачено виконання комплексу взаємопов'язаних завдань. Насамперед необхідно провести аналіз предметної області та існуючих аналогів, визначити функціональні й нефункціональні вимоги до системи, спроектувати архітектуру програмного продукту та структуру бази даних. Наступним етапом є реалізація основних

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

модулів системи, зокрема реєстрації та автентифікації користувачів, управління аукціонами, подання ставок, двостороннього чату, особистих кабінетів покупця і продавця та генерації інвойсів. Завершальним етапом є перевірка працездатності системи, забезпечення базових механізмів безпеки, валідації даних і підготовка технічної документації.

Об'єктом дослідження є процес купівлі-продажу автомобілів через онлайн-аукціони.

Предметом дослідження – веб-додаток, що реалізує цей процес на основі сучасних технологій розробки.

У роботі використано методи системного аналізу, об'єктно-орієнтованого проектування, UML-моделювання та тестування програмного забезпечення. Інформаційною базою роботи стали матеріали щодо веборієнтованих систем, онлайн-аукціонів, клієнт-серверної архітектури, UML-моделювання, а також офіційна документація обраних технологій.

Практичне значення роботи полягає у створенні готового до впровадження програмного продукту, який може бути використаний як приватними особами, так і невеликими автосалонами для організації власних онлайн-аукціонів. Система забезпечує повну автоматизацію бізнес-процесів, зменшує людський фактор та надає зручний, інтуїтивно зрозумілий інтерфейс для всіх учасників. Крім того, архітектура системи дозволяє легко розширювати функціонал у майбутньому – наприклад, шляхом інтеграції платіжних систем (LiqPay, Stripe), додавання мобільного додатку або адміністративної панелі.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз аналогів програмного продукту

Сучасний ринок платформ для онлайн-продажу автомобілів представлений як універсальними торговельними майданчиками, так і спеціалізованими аукціонними системами. Для визначення стандартного функціоналу та обґрунтування власної розробки було проаналізовано три найпоширеніші платформи: Copart, AutoBidMaster та eBay Motors.

Copart є однією з найбільш відомих платформ онлайн-продажу автомобілів і має велику базу транспортних засобів, що охоплює мільйони лотів на рік. Платформа підтримує повноцінну аукціонну модель з можливістю участі як наживо, так і в режимі попередньої ставки, і використовується в багатьох країнах світу [1].

Головна сторінка Copart (рис. 1.1.) демонструє орієнтацію платформи на масштаб та професійну аудиторію: великий пошуковий блок з фільтрами за маркою, моделлю, локацією та статусом пошкодження, а також блоки для реєстрації дилерів і страхових компаній.

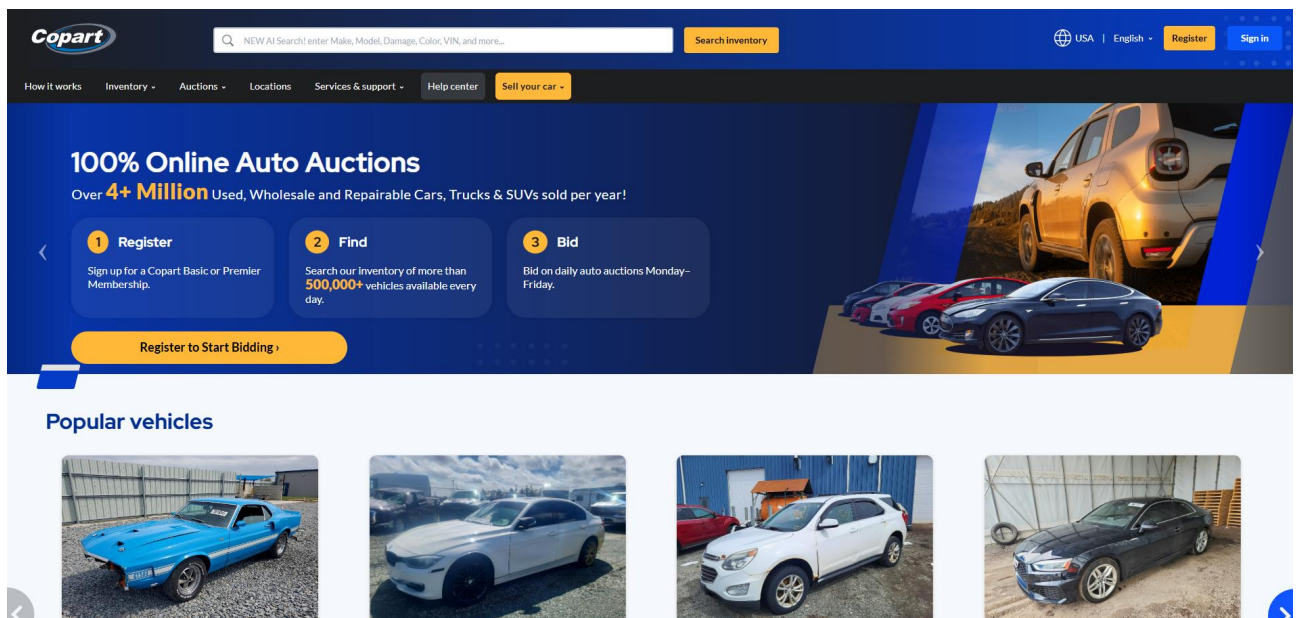


Рис. 1.1. Головна сторінка платформи Copart

Платформа здебільшого орієнтована на комерційних учасників ринку – дилерів, страхові компанії та професійних перекупників, які купують

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

пошкоджені або конфісковані транспортні засоби для подальшого перепродажу чи розбирання на запчастини. На рисунку 1.2. зображено сторінку конкретного лота з детальним описом пошкоджень, історією аукціону та формою подання ставки, що вимагає попередньо підтвердженого облікового запису з депозитом.

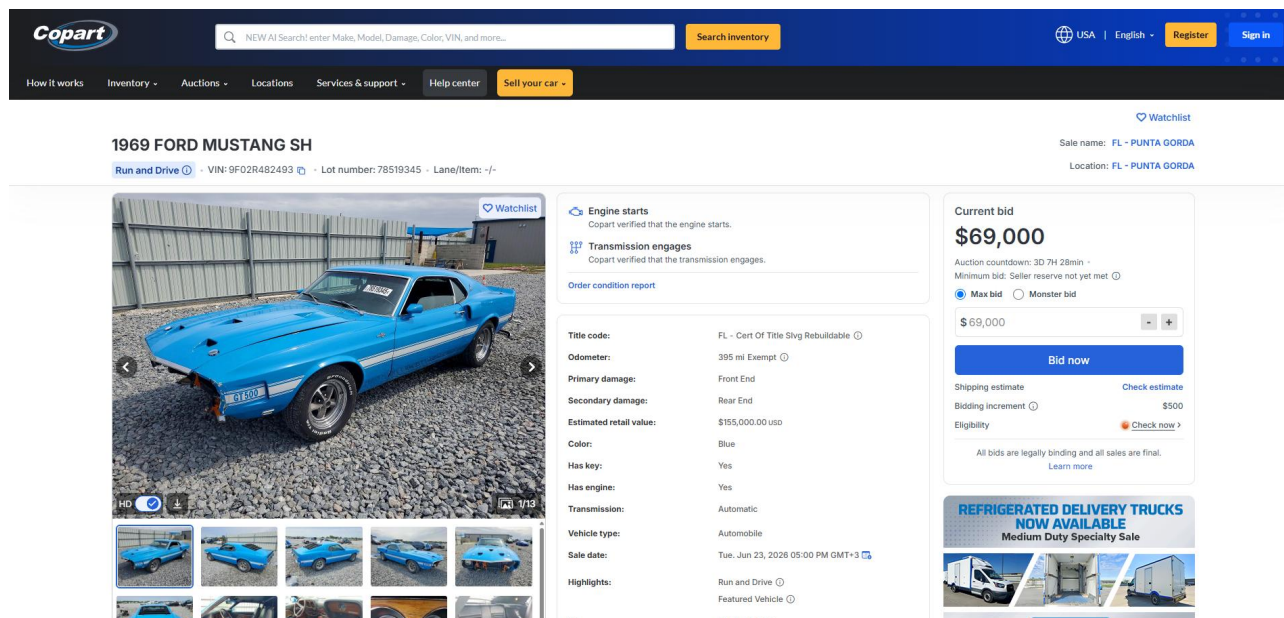


Рис. 1.2. Сторінка лота з формою подання ставки на Copart

Для звичайного користувача така платформа має високий поріг входження: складні правила участі, обов'язкові комісії за реєстрацію та участь у торгах, а також вимоги до підтвердження статусу ліцензованого покупця в багатьох регіонах. Це робить Copart незручним рішенням для людини, яка хоче просто продати або купити один автомобіль без посередників.

Попри ці обмеження, Copart має й суттєві переваги, які варто врахувати. Платформа забезпечує високий рівень прозорості завдяки детальним фотозвітам та описам технічного стану кожного лота, а також надає доступ до автомобілів за цінами, нижчими за ринкові, через продаж пошкоджених або страхових випадків. Велика база активних користувачів формує конкурентне середовище торгів, що сприяє об'єктивному ціноутворенню. Крім того, наявність мобільного застосунку та можливості участі в торгах віддалено робить платформу зручною для покупців з будь-якої точки світу.

AutoBidMaster спеціалізується на онлайн-аукціонах автомобілів і позиціонується як доступніша альтернатива для звичайних користувачів

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

порівняно з Copart [2]. Платформа надає можливість брати участь у торгах без прямої дилерської ліцензії, що знижує бар'єр входження для звичайного покупця.

Головна сторінка AutoBidMaster (рис. 1.3.) орієнтована на спрощений процес реєстрації та участі в торгах: коротка форма реєстрації, перелік переваг для приватних покупців та посилання на популярні категорії лотів.

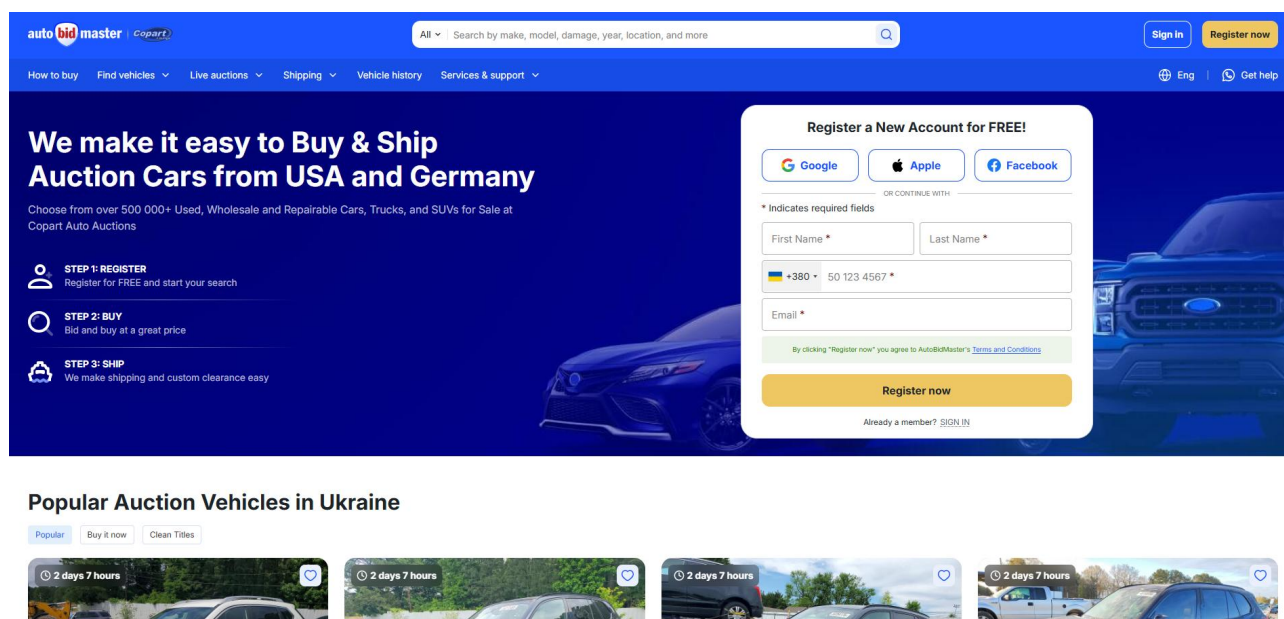


Рис. 1.3. Головна сторінка платформи AutoBidMaster

Функціональність платформи переважно зосереджена на участі в уже сформованих аукціонах, що проводяться партнерськими майданчиками. На рисунку 1.4. показано список активних лотів з фільтрацією за станом автомобіля, ціною та локацією.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

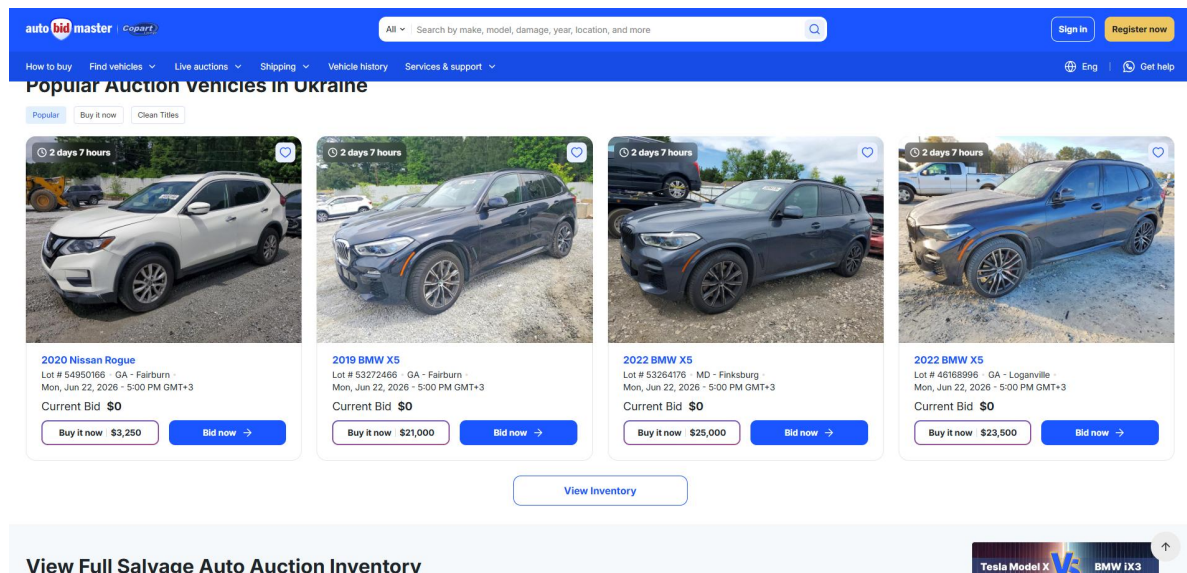


Рис. 1.4. Список активних лотів на AutoBidMaster

Водночас для задачі створення власної простої системи з особистими кабінетами, інтегрованим чатом між учасниками угоди та автоматичною генерацією інвойсів функціоналу AutoBidMaster недостатньо – платформа не передбачає прямої комунікації між продавцем і покупцем після завершення торгів, а документообіг обмежується стандартними формами без можливості кастомізації.

eBay Motors є відомим майданчиком для продажу автомобілів, запчастин та супутніх товарів у складі глобальної торговельної платформи eBay [3]. На відміну від Copart та AutoBidMaster, eBay Motors підтримує продаж транспортних засобів безпосередньо приватними особами без посередників.

Головна сторінка розділу eBay Motors (рис. 1.5.) демонструє ширший асортимент, ніж спеціалізовані автоаукціони: окрім цілих автомобілів, платформа пропонує запчастини, аксесуари та мотоцикли, що відображено в навігаційній структурі сторінки.

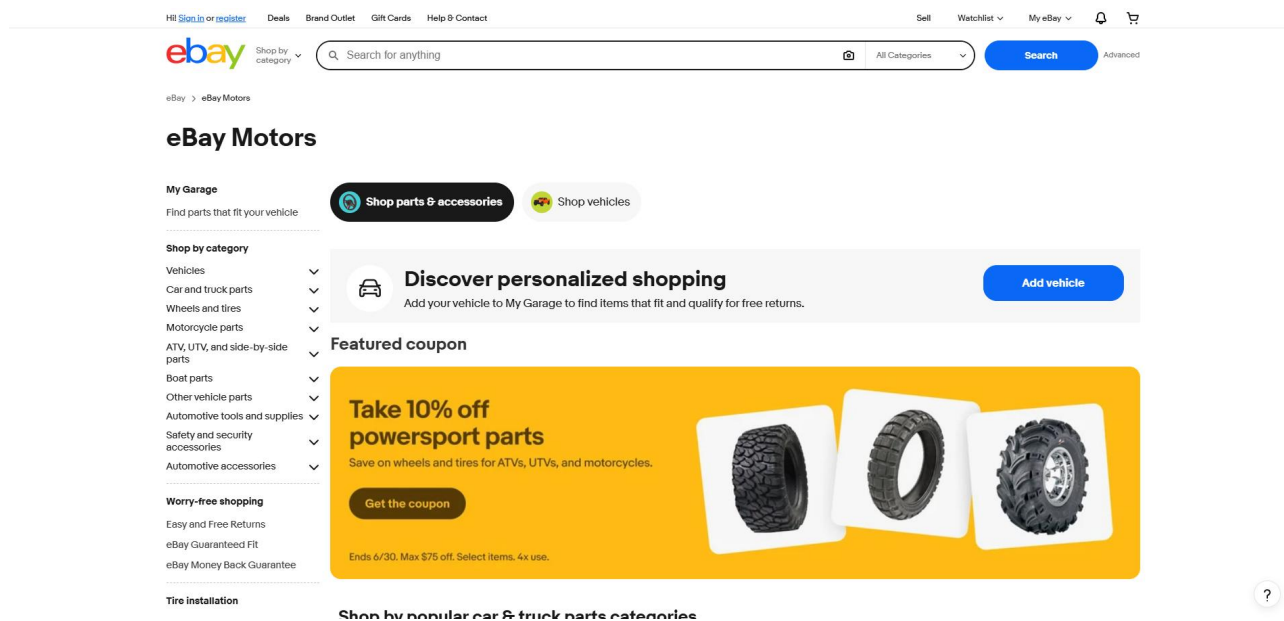


Рис. 1.5. Головна сторінка розділу eBay Motors

Аукціонна модель доступна на eBay Motors, проте не є основним сценарієм використання – більшість оголошень публікуються з фіксованою ціною або опцією «Купити зараз». На рисунку 1.6. зображено сторінку оголошення автомобіля з варіантами участі в торгах та прямої покупки одночасно.

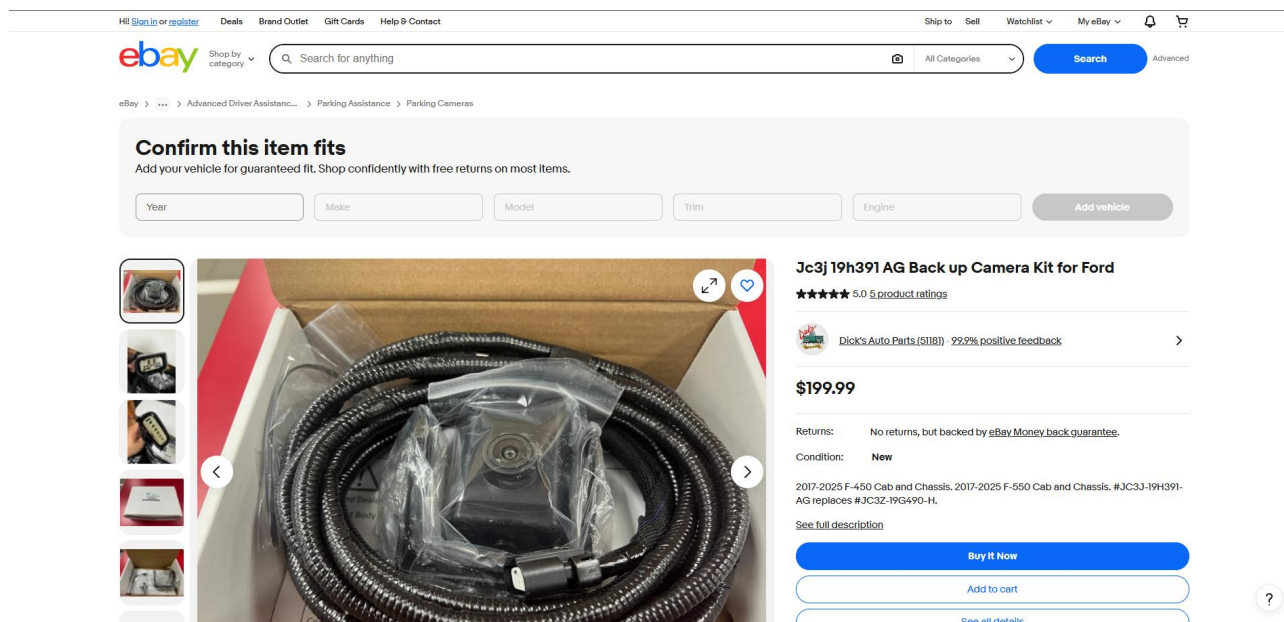


Рис. 1.6. Сторінка оголошення автомобіля на

eBay Motors eBay Motors не орієнтована саме на автоматизацію повного циклу угоди між покупцем і продавцем у межах окремої спеціалізованої системи. Платформа не пропонує вбудованого чату для узгодження деталей

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

угоди, автоматичного завершення аукціону з визначенням переможця за окремою бізнес-логікою чи генерації інвойсів – ці функції залишаються поза межами стандартного функціоналу eBay як універсального торговельного майданчика.

Водночас платформа має суттєві переваги завдяки своєму масштабу та репутації: величезна аудиторія покупців і продавців, відпрацьована система захисту покупця (eBay Buyer Protection), інтегровані способи оплати та доставки, а також можливість продажу супутніх товарів (запчастин, аксесуарів, мотоциклів) в межах однієї екосистеми. Однак саме ця універсальність є й обмеженням для задачі даної роботи: eBay Motors побудована як загальна торговельна платформа, а не як спеціалізований інструмент для проведення автоаукціонів із чітко визначеним життєвим циклом угоди.

На основі проведеного аналізу трьох платформ (Copart, AutoBidMaster та eBay Motors) складено порівняльну таблицю їхніх функціональних можливостей у зіставленні з можливостями власної системи (табл. 1.1). Порівняння здійснено за критеріями, що є визначальними для приватного користувача: доступність реєстрації, повнота аукціонної моделі, наявність автоматизованих механізмів завершення торгів та визначення переможця, вбудована комунікація між учасниками угоди, зручність особистих кабінетів, автоматична генерація супровідної документації та адаптивність інтерфейсу для мобільних пристроїв.

Таблиця 1.1.

Порівняльний аналіз функціональних можливостей існуючих платформ
онлайн-аукціонів авто

Функціонал	Copart	eBay Motors	AutoBidMaster	Пропонована система
Реєстрація приватних осіб	Недоступна	Доступна	Доступна	Доступна
Повноцінна аукціонна модель	Так	Обмежена	Так	Так
Автоматичне	Так	Ні	Ні	Так

завершення аукціонів				
Автоматичне визначення переможця	Так	Так	Так	Так
Вбудований двосторонній чат між учасниками	Ні	Ні	Ні	Так
Особисті кабінети для продавців та покупців	Так	Так	Обмежено	Так
Автоматична генерація інвойсів	Так	Ні	Ні	Так
Адаптивний інтерфейс для мобільних пристроїв	Частково	Так	Ні	Так

Порівняння аналогів показує, що наявні платформи або занадто складні для приватних користувачів, або не забезпечують повного циклу онлайн-аукціону. Розроблювана система повинна усунути ці обмеження шляхом реалізації простого інтерфейсу, підтримки приватних користувачів, автоматичного завершення аукціонів, визначення переможця, збереження історії ставок, створення замовлення, формування текстового інвойсу та забезпечення комунікації між учасниками через чат. Таким чином, постановка задачі полягає у створенні вебсистеми, яка забезпечує повний процес проведення автоаукціону від створення лота до післяаукціонної взаємодії між покупцем і продавцем.

1.2. Постановка задачі та аналіз аналогів програмного продукту

Проведений аналіз ринку онлайн-продажу автомобілів показав необхідність впровадження автоматизованої аукціонної системи для ефективної організації торгів, обробки ставок, контролю учасників та документального оформлення угод. Відсутність доступного рішення для приватних користувачів призводить до високого порогу входження на існуючі платформи, ускладнює процес купівлі-продажу та обмежує можливості прозорості комунікації між сторонами угоди.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

Вебсистема аукціонів авто повинна автоматизувати ключові бізнес-процеси онлайн-торгів. Перший з них – управління автомобілями та аукціонами. Система забезпечує додавання транспортних засобів із зазначенням марки, моделі, року випуску, VIN-коду, опису пошкоджень та фотографій, а також створення аукціону на основі доданого автомобіля з визначенням часу початку і завершення торгів.

Наступним по важливості варто виділити подання ставок, бо це є один із центральних модулів системи. Покупець може подавати ставки для активного аукціону, система перевіряє коректність суми відносно поточної максимальної ставки та зберігає історію торгів. Після завершення аукціону система автоматично визначає переможця за найбільшою ставкою.

Автоматичне завершення аукціонів реалізовано як окремий механізм, що передбачає періодичну перевірку активних лотів, час завершення яких уже минув. Для кожного такого аукціону система визначає переможця, створює замовлення зі статусом очікування оплати та змінює статус аукціону на завершений, що унеможливлює подання нових ставок.

Управління замовленнями та генерація інвойсів здійснюється через окремий модуль, де ведеться облік замовлень, створених для переможців торгів, із зазначенням суми до сплати та статусу. На основі даних замовлення формується текстовий інвойс із інформацією про автомобіль, покупця, продавця та суму до сплати.

Двосторонній чат реалізовано на основі Socket.IO для комунікації між покупцем і продавцем після завершення аукціону. Повідомлення прив'язуються до конкретного аукціону, що дозволяє зберігати історію спілкування в межах окремої угоди та відстежувати статус прочитання кожного повідомлення.

Рольова модель доступу реалізована на основі JWT-токенів і визначає права кожного користувача при вході до системи. Гість має доступ лише до перегляду активних аукціонів без можливості участі в торгах. Покупець може подавати ставки, переглядати власні перемоги, завантажувати інвойси та

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

спілкуватися з продавцем. Продавець може додавати автомобілі, створювати аукціони, переглядати власні активні та завершені лоти.

Особисті кабінети доступні авторизованим користувачам у вигляді розділів «Мої перемоги» для покупця та «Мої аукціони» для продавця, де відображається статус кожного лота, інформація про переможця та доступні дії (перегляд деталей, завантаження інвойсу або перехід до чату).

Для досягнення поставленої мети в рамках кваліфікаційної роботи необхідно вирішити наступні задачі:

- провести аналіз предметної області онлайн-аукціонів авто, визначити основні бізнес-процеси та ролі користувачів системи;
- дослідити існуючі програмні аналоги та визначити їх переваги і недоліки;
- обґрунтувати вибір клієнт-серверної архітектури та технологічного стеку;
- спроектувати структуру реляційної бази даних PostgreSQL та REST API основних сутностей;
- реалізувати серверну частину на базі NestJS із REST API та модульною архітектурою;
- реалізувати клієнтську частину на базі Next.js з TypeScript та Tailwind CSS;
- забезпечити коректну роботу бізнес-логіки: автоматичне завершення аукціонів, визначення переможця, генерацію інвойсів;

Результатом виконання поставлених задач є функціонуюча вебсистема аукціонів авто, яка може бути розгорнута в реальних умовах для приватних користувачів та невеликих автосалонів і масштабована відповідно до потреб бізнесу.

Для систематизації вимог до вебсистеми аукціонів авто складено перелік функціональних вимог із зазначенням пріоритету реалізації (табл. 1.2.). Пріоритет реалізації визначався виходячи з критичності вимоги для базового функціонування системи: вимоги з високим пріоритетом є обов'язковими для

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

першого релізу, з середнім – реалізуються в основному функціональному блоці, з низьким – є розширенням базового функціоналу.

Таблиця 1.2.

Функціональні вимоги до вебсистеми аукціонів авто

№	Функціональна вимога	Пріоритет
1	Реєстрація та авторизація користувачів з розмежуванням ролей	Високий
2	Додавання автомобілів із зазначенням характеристик та фото	Високий
3	Створення аукціону з визначенням часу завершення торгів	Високий
4	Подання ставок з перевіркою коректності суми	Високий
5	Автоматичне завершення аукціонів та визначення переможця	Високий
6	Перегляд активних аукціонів неавторизованим користувачем	Високий
7	Особисті кабінети покупця та продавця	Високий
8	Генерація текстового інвойсу для переможця торгів	Середній
9	Двосторонній чат між покупцем і продавцем	Середній
10	Історія ставок у межах конкретного аукціону	Середній
11	Індикація непрочитаних повідомлень у чаті	Низький
12	Адаптивний перегляд аукціонів на мобільних пристроях	Середній

Окрім функціональних вимог, визначено нефункціональні вимоги, що встановлюють обмеження на якісні характеристики системи (табл. 1.3.). Нефункціональні вимоги охоплюють п'ять ключових категорій: продуктивність, безпека, надійність, масштабованість та зручність використання. Дотримання цих вимог забезпечує не лише коректну роботу системи, а й належний рівень захисту облікових даних та фінансової інформації учасників торгів, що є критично важливим для практичного застосування розробленої системи.

Таблиця 1.3.

Нефункціональні вимоги до вебсистеми аукціонів авто

Категорія	Вимога
Продуктивність	Час відгуку системи не більше 2 секунд для типових сценаріїв
Безпека	Зберігання паролів у хешованому вигляді (bcrypt)
Безпека	Автентифікація на основі JWT-токенів
Безпека	Захист від SQL-ін'єкцій через параметризовані запити TypeORM
Надійність	Валідація вхідних даних засобами class-validator

Масштабованість	Контейнеризоване розгортання через Docker і Docker Compose
Зручність	Адаптивний інтерфейс для пристроїв з шириною екрана від 320 пікселів

1.3. Вибір архітектури вебсистеми

Під час проєктування вебсистеми аукціонів авто одним із ключових завдань є вибір архітектурного підходу, оскільки саме архітектура визначає принципи побудови програмного комплексу, розподіл відповідальностей між його частинами, спосіб взаємодії користувача із системою та можливості подальшого розвитку. Для цієї роботи важливо було обрати таку архітектуру, яка дозволяє реалізувати основний функціонал онлайн-аукціону, забезпечити стабільну роботу з базою даних, підтримати автентифікацію користувачів, обробку ставок, генерацію інвойсів і двосторонню комунікацію між покупцем і продавцем. У межах проєктування було розглянуто кілька архітектурних підходів, зокрема монолітну архітектуру, мікросервісну архітектуру та клієнт-серверну модель із використанням REST API [16, 17, 24].

Монолітна архітектура передбачає об'єднання всіх основних компонентів системи в одному програмному застосунку. У такому підході інтерфейс користувача, серверна логіка, модулі роботи з користувачами, автомобілями, аукціонами, ставками, замовленнями, повідомленнями та базою даних можуть розміщуватися в межах єдиного програмного комплексу. Основною перевагою монолітної архітектури є простота початкової розробки, оскільки всі частини системи знаходяться в одному середовищі, мають спільну структуру та можуть запускатися як єдиний застосунок. Такий підхід часто використовується для невеликих програмних продуктів або прототипів, де першочерговим завданням є швидка реалізація базового функціоналу.

Водночас для вебсистеми аукціонів авто монолітна архітектура має низку обмежень. У разі розширення функціональності зміни в одному модулі можуть впливати на інші частини системи, що ускладнює тестування, підтримку та подальшу модернізацію. Наприклад, зміна логіки подання ставок або автоматичного завершення аукціонів може потребувати додаткової перевірки

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

всього застосунку, якщо між модулями немає чіткого розділення відповідальностей. Крім того, монолітний підхід ускладнює масштабування окремих частин системи. Якщо найбільше навантаження припадає на модуль ставок або чат, масштабувати доведеться весь застосунок, а не тільки окремий компонент. Саме тому монолітна архітектура не є оптимальним рішенням для системи, у якій передбачено окрему клієнтську частину, серверну бізнес-логіку та базу даних.

Іншим розглянутим варіантом є мікросервісна архітектура. Вона передбачає поділ програмного комплексу на набір окремих незалежних сервісів, кожен із яких відповідає за конкретну функціональну область. Для системи автоаукціонів окремими сервісами могли б бути сервіс користувачів, сервіс автомобілів, сервіс аукціонів, сервіс ставок, сервіс замовлень, сервіс повідомлень та сервіс інвойсів. Кожен із таких сервісів міг би мати власну логіку, власний інтерфейс взаємодії та навіть окреме сховище даних. Такий підхід є доцільним для великих високонавантажених систем, де потрібно незалежно масштабувати окремі модулі, розгортати їх окремо та забезпечувати високу відмовостійкість.

Попри переваги, мікросервісна архітектура є надмірно складною для меж цієї кваліфікаційної роботи. Її використання потребувало б додаткового проектування міжсервісної взаємодії, налаштування брокерів повідомлень або API-шлюзу, окремого логування, моніторингу, обробки помилок між сервісами та складнішого процесу розгортання. Для невеликої системи, яка реалізується в межах бакалаврської роботи, така складність не є виправданою. Основною метою роботи є створення завершеного програмного продукту, який демонструє повний цикл онлайн-аукціону, а не побудова розподіленої інфраструктури корпоративного рівня. Тому мікросервісний підхід було розглянуто як перспективний варіант для майбутнього масштабування, але не як основну архітектуру поточної реалізації.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

З урахуванням функціональних вимог, строків розробки та необхідності отримати завершений працездатний програмний продукт було обрано клієнт-серверну архітектуру з використанням REST API [16, 24]. У цій моделі система поділяється на клієнтську частину, серверну частину та базу даних. Клієнтська частина відповідає за відображення інтерфейсу користувача, сторінок аукціонів, форм реєстрації та авторизації, особистих кабінетів, сторінки чату та розділу перемог покупця. Серверна частина реалізує бізнес-логіку системи, обробляє запити, перевіряє права доступу, виконує операції з базою даних, завершує аукціони, визначає переможців і формує інвойси. База даних зберігає інформацію про користувачів, автомобілі, аукціони, ставки, замовлення та повідомлення.

Клієнт-серверна архітектура є зручною для цієї системи, оскільки вона забезпечує чітке розділення відповідальностей. Клієнтська частина не звертається безпосередньо до бази даних, а надсилає HTTP-запити до серверної частини. Сервер, у свою чергу, виконує перевірку вхідних даних, контролює авторизацію користувача, обробляє бізнес-логіку та взаємодіє з PostgreSQL через TypeORM. Такий підхід підвищує безпеку, оскільки користувач не має прямого доступу до структури бази даних або внутрішніх механізмів роботи системи. Крім того, це спрощує подальший супровід проєкту: зміни в інтерфейсі можна виконувати незалежно від логіки бази даних, а серверну частину можна розширювати без повної перебудови клієнтського рівня.

REST API використовується як основний механізм взаємодії між клієнтською та серверною частинами. Через REST-запити виконуються реєстрація та авторизація користувачів, отримання списку аукціонів, перегляд конкретного лота, створення автомобіля та аукціону, подання ставки, перегляд перемог покупця, завантаження інвойсу та робота з особистими кабінетами. Такий підхід є зрозумілим, добре підтримується сучасними вебфреймворками та дозволяє чітко структурувати маршрути API. Крім того, REST API спрощує подальшу інтеграцію з іншими клієнтами. У майбутньому до цієї серверної

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

частини можна підключити мобільний застосунок або адміністративну панель без повного переписування backend-логіки.

Окрему роль в архітектурі системи відіграє WebSocket-комунікація через Socket.IO [13, 17, 25]. REST API є зручним для більшості стандартних операцій, однак не всі функції системи можуть ефективно працювати лише через звичайні HTTP-запити. Наприклад, чат між покупцем і продавцем потребує швидкого обміну повідомленнями, а оновлення інформації про ставки бажано виконувати без постійного ручного оновлення сторінки. Саме тому для сценаріїв, пов'язаних із подіями в реальному часі, до архітектури додано WebSocket-рівень. Він дозволяє серверу передавати оновлення клієнту одразу після виникнення події, що покращує зручність використання системи та робить взаємодію користувачів більш динамічною.

Архітектура системи також враховує розподіл користувачів за ролями. У системі передбачено покупця та продавця, і для кожної ролі доступні різні сценарії роботи. Покупець може переглядати аукціони, подавати ставки, переглядати власні перемоги, завантажувати інвойс і спілкуватися з продавцем. Продавець може додавати автомобілі, створювати аукціони, переглядати власні активні та завершені лоти, бачити переможця та вести комунікацію з покупцем. Серверна частина виконує перевірку прав доступу та не дозволяє користувачу виконувати дії, які не відповідають його ролі. Це важливо для забезпечення логічної цілісності системи та захисту від некоректних операцій.

У межах обраної архітектури клієнтська частина реалізується на Next.js, серверна частина – на NestJS, а база даних – на PostgreSQL. Такий технологічний поділ відповідає клієнт-серверній моделі та дозволяє організувати систему як набір взаємопов'язаних, але логічно розділених компонентів. Клієнтська частина формує запити та відображає результати користувачу, серверна частина обробляє ці запити та реалізує бізнес-правила, а база даних забезпечує постійне збереження інформації. Для розгортання всі

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

компоненти можуть бути запуснені в окремих контейнерах за допомогою Docker і Docker Compose, що також відповідає обраній архітектурній моделі.

Порівняння трьох підходів показало, що монолітна архітектура є простою для початкової розробки, але обмежує гнучкість системи в майбутньому. Мікросервісна архітектура забезпечує високий рівень масштабованості, однак є надмірно складною для реалізації в межах кваліфікаційної роботи. Клієнт-серверна архітектура з REST API є компромісним і найбільш доцільним варіантом, оскільки вона забезпечує достатню гнучкість, зрозумілу структуру, розділення відповідальностей і можливість подальшого розширення. У майбутньому система може бути доповнена адміністративною панеллю, модулем онлайн-оплати, мобільним застосунком, аналітичним модулем або окремими сервісами для повідомлень і статистики. На основі проведеного аналізу складено порівняльну таблицю трьох архітектурних підходів за ключовими критеріями (табл. 1.4).

Таблиця 1.4.

Порівняння архітектурних підходів

Критерій	Монолітна	Мікросервісна	Клієнт-серверна
Складність розробки	Низька	Висока	Середня
Розділення відповідальностей	Відсутнє	Повне	Часткове
Незалежне масштабування	Неможливе	Так	Обмежене
Складність розгортання	Низька	Висока	Середня
Швидкість реалізації	Висока	Низька	Середня
Відповідність обсягу роботи	Надлишково проста	Надлишково складна	Оптимальна

Архітектура системи представлена на рисунку 1.7. Діаграма ілюструє розподіл відповідальностей між клієнтською частиною, серверною логікою та сховищем даних, а також показує точки взаємодії компонентів через REST API та WebSocket-з'єднання. Така візуалізація дозволяє наочно простежити шлях обробки запиту від інтерфейсу користувача до бази даних і у зворотному напрямку.

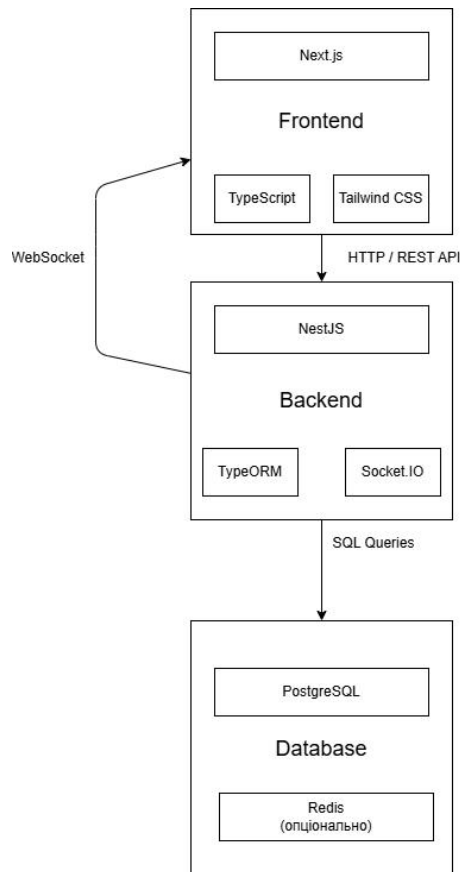


Рис. 1.7. Діаграма компонентів вебсистеми аукціонів авто

Як показано на рисунку 1.7., система складається з клієнтської частини, серверної частини та бази даних. Клієнтська частина взаємодіє з користувачем і надсилає запити до backend API. Серверна частина обробляє ці запити, виконує автентифікацію, перевіряє права доступу, реалізує бізнес-логіку аукціонів і взаємодіє з PostgreSQL. Для обміну повідомленнями та подій у реальному часі використовується Socket.IO. Така структура забезпечує повний цикл роботи вебсистеми аукціонів авто та відповідає поставленим функціональним вимогам.

Взаємодію клієнтської та серверної частин через два паралельні канали зв'язку (REST API для стандартних операцій та WebSocket для подій реального часу) зображено на рисунку 1.8. Такий гібридний підхід дозволяє використовувати кожен протокол відповідно до його сильних сторін: REST забезпечує прості та передбачувані запит-відповідь операції з кешуванням на боці клієнта, тоді як WebSocket підтримує постійне з'єднання, необхідне для миттєвого обміну повідомленнями без затримок на повторні HTTP-запити. Розділення каналів зв'язку також спрощує масштабування системи в

майбутньому, оскільки REST API та WebSocket-сервіс можуть бути винесені в окремі компоненти за потреби. Крім того, така архітектура зменшує навантаження на сервер у звичайному режимі роботи, оскільки WebSocket-з'єднання активується лише для користувачів, які перебувають у чаті, тоді як решта операцій обробляється через короткоживучі REST-запити.

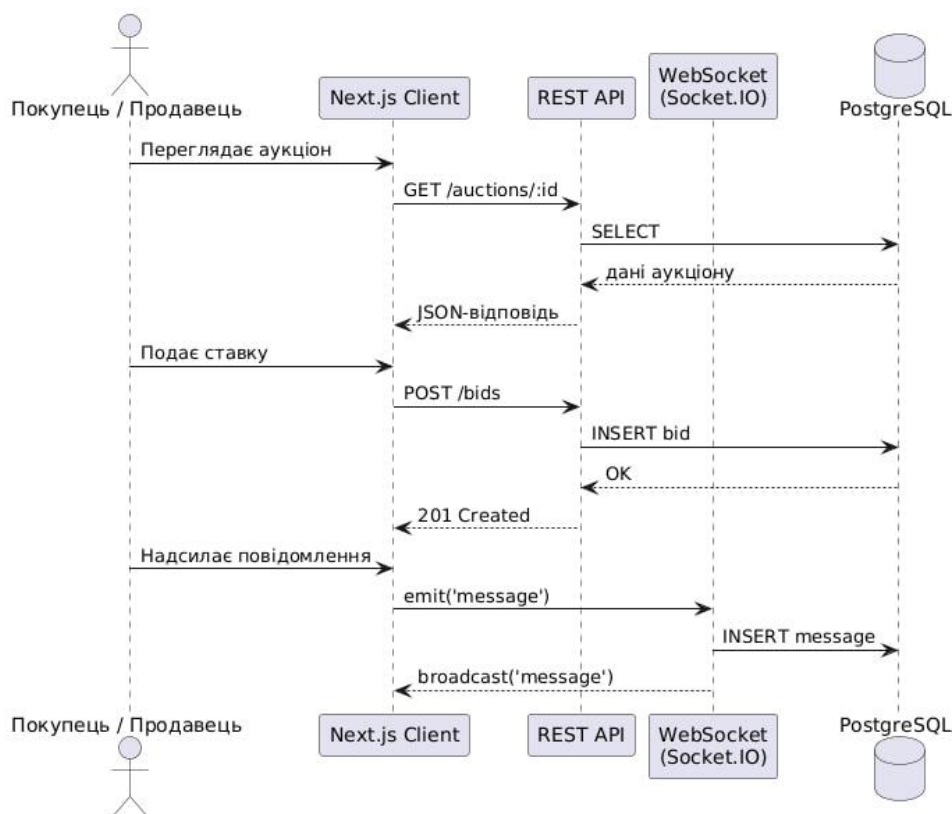


Рис. 1.8. Схема взаємодії через REST API та WebSocket

Як показано на рисунку 1.8., REST API обробляє дискретні операції (перегляд, подання ставки), що завершуються одним запитом-відповіддю, тоді як WebSocket підтримує постійне з'єднання для миттєвої доставки повідомлень чату без необхідності опитування сервера.

1.4. Вибір інструментальних засобів та вимоги до апаратного забезпечення

Вибір інструментальних засобів є важливим етапом розробки вебсистеми аукціонів авто, оскільки від обраного технологічного стеку залежить швидкість реалізації, стабільність роботи, зручність подальшої підтримки та можливість розширення програмного продукту. Для цієї системи було обрано набір

сучасних технологій, які забезпечують розділення клієнтської та серверної частин, роботу з реляційною базою даних, підтримку автентифікації, обмін повідомленнями в реальному часі та контейнеризоване розгортання. До основних інструментів належать Next.js, TypeScript, Tailwind CSS, Axios, NestJS, TypeORM, PostgreSQL, JWT, Socket.IO, Docker і Docker Compose [4–15, 17, 19, 22–25].

Клієнтську частину системи реалізовано з використанням Next.js. Цей фреймворк побудований на основі React і надає зручні механізми для створення сторінок, маршрутизації, компонентної структури та роботи з клієнтською логікою [4, 5]. Для вебсистеми аукціонів авто Next.js є доцільним вибором, оскільки інтерфейс складається з кількох взаємопов'язаних сторінок: головної сторінки зі списком аукціонів, сторінки конкретного лота, сторінки реєстрації та входу, особистого кабінету продавця, особистого кабінету покупця, сторінки перемог та чату. Завдяки маршрутизації Next.js можна логічно організувати ці частини та забезпечити зручний перехід між ними.

Важливою перевагою Next.js є можливість створення структурованого вебінтерфейсу з повторним використанням компонентів. У системі аукціонів це особливо корисно, оскільки окремі елементи інтерфейсу повторюються на різних сторінках. Наприклад, картки автомобілів, кнопки переходу до аукціону, форми введення даних, елементи навігації та блоки відображення статусів можуть бути реалізовані як окремі компоненти. Це зменшує дублювання коду, спрощує підтримку інтерфейсу та дає змогу швидше вносити зміни в дизайн або функціональність системи.

Для підвищення надійності коду використовується TypeScript [6]. На відміну від звичайного JavaScript, TypeScript підтримує статичну типізацію, що дозволяє виявляти частину помилок ще на етапі розробки. Для вебсистеми, яка оперує користувачами, автомобілями, аукціонами, ставками, замовленнями та повідомленнями, типізація є важливою перевагою. Вона дозволяє чітко описати структуру даних, які передаються між клієнтською та серверною частинами, і

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						26
Змн.	Арк.	№ докум.	Підпис	Дата		

зменшує ризик некоректного використання об'єктів у програмному коді. Наприклад, дані аукціону можуть містити ідентифікатор, автомобіль, продавця, поточну ставку, час завершення та статус. Завдяки TypeScript ці поля можна формалізувати та контролювати під час розробки.

Оформлення клієнтської частини виконано за допомогою Tailwind CSS [7]. Цей інструмент дозволяє створювати адаптивний інтерфейс за допомогою набору утилітарних класів. Для системи аукціонів авто важливо, щоб сторінки коректно відображалися на різних пристроях, оскільки користувач може переглядати лоти, робити ставки або відповідати на повідомлення як із персонального комп'ютера, так і зі смартфона. Tailwind CSS дає змогу швидко налаштовувати відступи, розміри, сітки, кольори, адаптивність і поведінку елементів без створення великої кількості окремих CSS-файлів. Це спрощує розробку інтерфейсу та дозволяє підтримувати єдиний стиль сторінок.

Для обміну даними між клієнтською частиною та backend API використовується Axios [8, 24]. Він застосовується для виконання HTTP-запитів до серверної частини під час реєстрації, авторизації, отримання списку аукціонів, завантаження інформації про конкретний лот, створення ставки, перегляду перемог і завантаження інвойсу. Використання Axios спрощує роботу з API, оскільки дозволяє централізовано налаштовувати базову адресу сервера, заголовки авторизації, обробку відповідей і помилок. Для цієї системи це важливо, оскільки значна частина функцій доступна лише авторизованим користувачам і потребує передачі JWT-токена разом із запитом.

Серверну частину системи реалізовано на основі NestJS [9, 10]. Цей фреймворк побудований для створення масштабованих серверних застосунків на Node.js і добре підходить для модульної організації коду. У вебсистемі аукціонів авто серверна частина містить окремі модулі для користувачів, автентифікації, автомобілів, аукціонів, ставок, замовлень і повідомлень. Кожен модуль може мати власні контролери, сервіси та сутності. Контролери відповідають за приймання HTTP-запитів, сервіси реалізують бізнес-логіку, а

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

сутності описують структуру даних. Такий підхід робить код зрозумілим, логічно розділеним і зручним для супроводу.

Однією з важливих особливостей NestJS є підтримка dependency injection. Завдяки цьому сервіси можуть отримувати необхідні залежності через конструктор, що спрощує зв'язок між модулями та робить код більш структурованим. Наприклад, сервіс аукціонів може використовувати репозиторії аукціонів, ставок і замовлень, а сервіс повідомлень може працювати з репозиторієм повідомлень та даними користувачів. Така організація дозволяє розділити логіку на окремі частини та уникнути надмірного дублювання коду.

Для роботи з базою даних використовується TypeORM [11]. Він виконує роль об'єктно-реляційного відображення та дозволяє працювати з таблицями бази даних через TypeScript-класи. Це зручно для системи, у якій основними сутностями є User, Vehicle, Auction, Bid, Order та Message. Кожна сутність описує відповідну таблицю бази даних, її поля та зв'язки з іншими сутностями. Завдяки TypeORM можна виконувати створення, оновлення, пошук і видалення записів через репозиторії, не формуючи SQL-запити вручну. Це зменшує ризик помилок і спрощує взаємодію серверної частини з базою даних. Перед остаточним вибором TypeORM розглядалися альтернативні рішення для роботи з базою даних у екосистемі Node.js – Prisma та Sequelize (табл. 1.5.). Обидва ORM мають широку спільноту та підтримку TypeScript, проте поступаються TypeORM за глибиною інтеграції з декораторною архітектурою NestJS, що ускладнило б узгодження сутностей із рештою застосунку.

Таблиця 1.5.

Порівняння ORM-рішень для Node.js

Критерій	TypeORM	Prisma	Sequelize
TypeScript-декоратори	Так	Ні (схема .prisma)	Опціонально
Автогенерація типів	Часткова	Повна	Відсутня
Підтримка міграцій	Так	Так	Так
Поріг входження	Середній	Низький	Середній

Інтеграція з NestJS	Нативна	Через адаптер	Через адаптер
Активність спільноти	Висока	Дуже висока	Середня

TypeORM обрано завдяки нативній інтеграції з NestJS через декоратори та органічному поєднанню з модульною архітектурою серверної частини, що дозволяє описувати сутності безпосередньо як TypeScript-класи без додаткового шару конфігурації.

Як систему керування базами даних обрано PostgreSQL [12]. Вона є доцільною для цієї роботи, оскільки дані системи мають чітко виражену реляційну структуру. Користувачі пов'язані з автомобілями, аукціонами, ставками, замовленнями та повідомленнями. Один продавець може створювати багато аукціонів, один аукціон може містити багато ставок, а після завершення торгів створюється замовлення для переможця. PostgreSQL підтримує зовнішні ключі, індекси, транзакції, UUID і механізми забезпечення цілісності даних. Це дозволяє надійно зберігати інформацію та підтримувати логічні зв'язки між основними об'єктами системи.

Автентифікація користувачів реалізується з використанням JWT-токенів [19, 26, 30]. Після успішного входу користувача сервер формує токен, у якому міститься інформація про ідентифікатор користувача, електронну пошту та роль. Надалі цей токен передається клієнтською частиною в заголовках запитів до захищених маршрутів. Такий підхід дозволяє серверу визначати користувача без збереження сесії на backend-стороні. Для системи аукціонів це важливо, оскільки доступ до створення аукціонів, подання ставок, перегляду перемог, завантаження інвойсів і роботи з чатом має бути доступний лише авторизованим користувачам.

Для реалізації функцій у реальному часі використовується Socket.IO [13, 23, 25]. У межах системи він застосовується насамперед для чату між покупцем і продавцем, а також може використовуватися для оновлення подій, пов'язаних зі ставками. На відміну від звичайних HTTP-запитів, WebSocket-комунікація дозволяє підтримувати постійне з'єднання між клієнтом і сервером. Це означає, що сервер може передавати нові повідомлення користувачу одразу після їх

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

створення. Для післяаукціонної комунікації та швидкого обміну інформацією між учасниками угоди така можливість є важливою.

Автоматичне завершення аукціонів реалізується за допомогою ScheduleModule у NestJS [22]. Цей механізм дозволяє запускати періодичні задачі за визначеним розкладом. У системі аукціонів планувальник регулярно перевіряє, чи є активні аукціони, час завершення яких уже минув. Якщо такі аукціони знайдено, серверна частина визначає найвищу ставку, створює замовлення для переможця та змінює статус аукціону на завершений. Завдяки цьому процес завершення торгів не потребує ручного втручання адміністратора або продавця.

Для розгортання програмного комплексу використано Docker і Docker Compose [14, 15]. Docker дозволяє запускати окремі частини системи в ізольованих контейнерах, а Docker Compose забезпечує одночасний запуск клієнтської частини, серверної частини та бази даних за допомогою одного конфігураційного файлу. Це спрощує підготовку середовища для розробки, тестування та демонстрації системи. Користувачу не потрібно окремо встановлювати всі залежності, налаштовувати PostgreSQL і вручну запускати кожен компонент. Достатньо виконати запуск через Docker Compose, після чого всі основні сервіси працюватимуть у межах єдиного середовища.

Контейнеризація також підвищує передбачуваність роботи системи. Якщо програмний комплекс запускається на іншому комп'ютері, Docker дозволяє зменшити кількість проблем, пов'язаних із різними версіями Node.js, PostgreSQL або локальних пакетів. Це особливо важливо під час перевірки кваліфікаційної роботи, оскільки система має бути відтворюваною та готовою до демонстрації. Крім того, використання Docker створює основу для подальшого розгортання системи у виробничому середовищі.

На основі проведеного аналізу складено зведену таблицю обраного технологічного стеку (табл. 1.6.), що узагальнює всі розглянуті вище інструменти за категоріями застосування – клієнтська частина, серверна

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						30
Змн.	Арк.	№ докум.	Підпис	Дата		

частина, база даних, обмін повідомленнями в реальному часі та розгортання. Таке узагальнення дозволяє наочно простежити, як окремі технологічні рішення поєднуються в єдину архітектуру програмного комплексу та яку конкретну задачу вирішує кожна з них.

Таблиця 1.6.

Технологічний стек вебсистеми аукціонів авто

Категорія	Технологія	Призначення
Клієнтська частина	Next.js	Фреймворк для побудови інтерфейсу
Клієнтська частина	TypeScript	Статична типізація
Клієнтська частина	Tailwind CSS	Адаптивна стилізація
Клієнтська частина	Axios	HTTP-клієнт
Серверна частина	NestJS	Серверний фреймворк
Серверна частина	TypeORM	Робота з базою даних
Серверна частина	JWT	Автентифікація
Серверна частина	ScheduleModule	Періодичні задачі
База даних	PostgreSQL	Реляційна СКБД
Реальний час	Socket.IO	WebSocket-комунікація
Розгортання	Docker / Docker Compose	Контейнеризація

Узагальнену схему технологічного стеку за рівнями архітектури зображено на рисунку 1.9. Така візуалізація доповнює табличне представлення, наочно демонструючи напрямки взаємодії між рівнями та протоколи, що їх з'єднують, що полегшує розуміння того, як окремі технології утворюють цілісну архітектуру програмного комплексу.

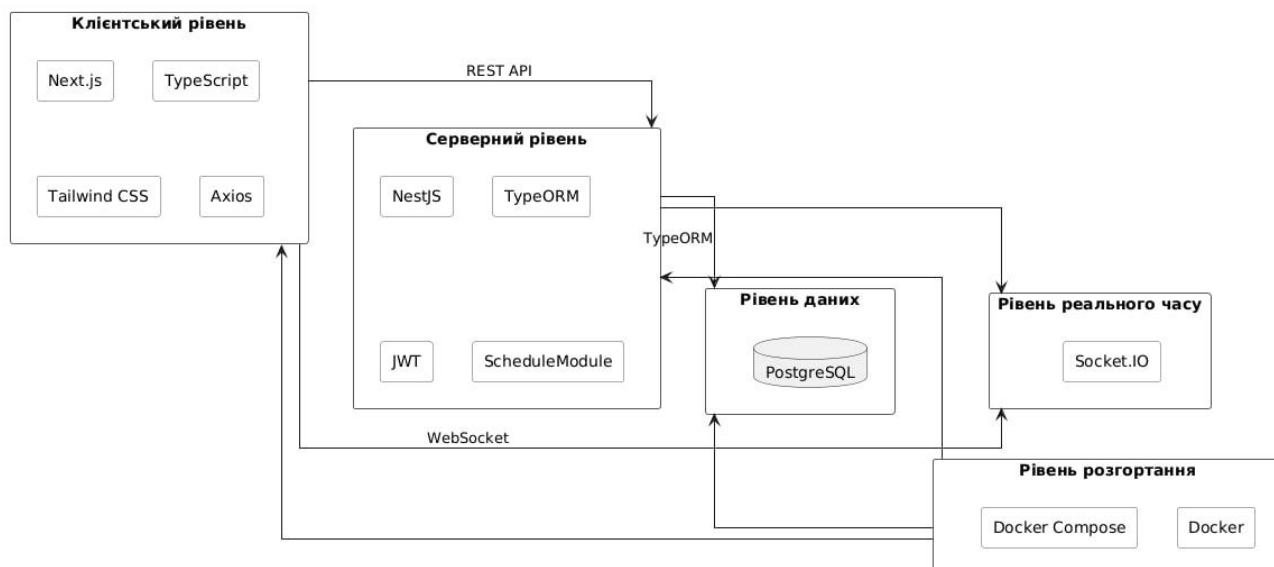


Рис. 1.9. Схема технологічного стеку вебсистеми аукціонів авто

До апаратного забезпечення для локального запуску системи не висуваються надмірно високі вимоги. Для демонстрації та тестування достатньо персонального комп'ютера з процесором рівня Intel Core i3 або аналогічним, 4 ГБ оперативної пам'яті, 2 ГБ вільного місця на диску та операційною системою Windows, Linux або macOS. Такі характеристики дозволяють запустити клієнтську частину, серверну частину та PostgreSQL у локальному середовищі. Для комфортнішої роботи, особливо під час запуску через Docker, бажано мати більший обсяг оперативної пам'яті, оскільки одночасна робота кількох контейнерів може створювати додаткове навантаження на систему.

Для виробничого середовища доцільно використовувати сервер із двома або більше ядрами процесора, щонайменше 4 ГБ оперативної пам'яті та окремим сховищем для бази даних. У разі збільшення кількості користувачів систему можна масштабувати поступово. Спочатку доцільно оптимізувати запити до бази даних, додати індекси до полів, які часто використовуються під час пошуку або фільтрації, а також винести PostgreSQL в окремий керований сервіс. У подальшому можна додати кешування, балансування навантаження або розділення окремих модулів на самостійні сервіси. Таким чином, обраний

технологічний стек не лише забезпечує реалізацію поточних вимог, а й залишає можливості для подальшого розвитку вебсистеми аукціонів авто.

Висновки до першого розділу

У першому розділі проведено аналіз предметної області онлайн-аукціонів авто та визначено основні проблеми, які виникають під час купівлі-продажу транспортних засобів через інтернет. Встановлено, що значна частина існуючих платформ орієнтована переважно на дилерів, автосалони або професійних учасників ринку, що створює додаткові труднощі для приватних користувачів.

У межах аналізу аналогів розглянуто три провідні платформи – Copart, AutoBidMaster та eBay Motors. Порівняння їх функціональних можливостей показало, що жодна з розглянутих систем повністю не закриває потребу у простій системі для приватних користувачів із вбудованим чатом, автоматизованим завершенням аукціонів і генерацією документального підтвердження угоди.

На основі проведеного аналізу сформульовано постановку задачі: визначено перелік бізнес-процесів, що підлягають автоматизації, описано рольову модель доступу та задачі, які необхідно реалізувати в подальших розділах роботи. Складено переліки функціональних і нефункціональних вимог до системи з визначенням пріоритету реалізації кожної з них.

Обґрунтовано вибір архітектурного підходу. Розглянуто монолітну, мікросервісну та клієнт-серверну архітектури, за результатами порівняння обрано клієнт-серверну модель з використанням REST API, оскільки вона забезпечує чітке розділення відповідальностей між frontend і backend-частинами та є достатньою для реалізації заявленого функціоналу.

Окремо обґрунтовано вибір інструментальних засобів розробки. Для клієнтської частини обрано Next.js, TypeScript, Tailwind CSS та Axios, для серверної частини – NestJS, TypeORM, JWT, Socket.IO та PostgreSQL. Проведено порівняльний аналіз ORM-рішень, за результатами якого підтверджено доцільність TypeORM завдяки нативній інтеграції з NestJS. Для

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дата		

розгортання системи використано Docker і Docker Compose, визначено вимоги до апаратного забезпечення для локального та виробничого середовищ.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

На етапі проєктування було проведено аналіз вимог до програмного продукту з урахуванням ролей користувачів, специфіки предметної області та обраної архітектури. У результаті було визначено функціональні й нефункціональні вимоги, побудовано UML-діаграму варіантів використання та описано об'єктну модель системи [18].

Функціональні вимоги визначають основні можливості системи, які мають бути реалізовані для задоволення потреб користувачів. У системі передбачено три основні ролі: гість, покупець і продавець. Гість є неавторизованим користувачем, який може переглядати список активних аукціонів, відкривати сторінку окремого лота, ознайомлюватися з характеристиками автомобіля, поточною ставкою та часом до завершення торгів. Для отримання доступу до розширених функцій гість може створити обліковий запис і виконати вхід у систему.

Продавець у системі відповідає за створення лотів та управління власними аукціонами. Він може додавати автомобілі із зазначенням марки, моделі, року випуску, VIN-коду, опису пошкоджень і фотографій. На основі доданого автомобіля продавець створює аукціон, визначає час його завершення та реквізити для оплати. У власному кабінеті продавець переглядає активні й завершені аукціони, бачить інформацію про переможця завершених торгів і може зв'язатися з покупцем через чат.

Покупець після авторизації отримує можливість брати участь в активних аукціонах шляхом подання ставок. Система повинна перевіряти, що ставка подається для активного аукціону та є коректною за значенням. Крім того, покупець може перейти до вбудованого чату для спілкування з продавцем після завершення торгів.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

Нефункціональні вимоги визначають якість роботи системи та її взаємодію з користувачем. Вебдодаток має забезпечувати швидку відповідь на дії користувача, коректне завантаження сторінок, стабільну обробку ставок і підтримку одночасної роботи кількох користувачів. Інтерфейс повинен бути адаптивним для мобільних пристроїв із шириною екрана від 320 пікселів. Окрема увага приділяється консистентності даних під час одночасного подання кількох ставок різними покупцями – система має коректно визначати останню найвищу ставку без втрати чи дублювання записів.

Окрему увагу приділено безпеці: паролі користувачів зберігаються у хешованому вигляді з використанням bcrypt [26, 27], автентифікація реалізована через JWT-токени [19, 30], вхідні дані проходять валідацію [21], а запити до бази даних виконуються через TypeORM із використанням параметризованих запитів [11, 28]. Інтерфейс системи є україномовним і підтримує коректне відображення кирилиці завдяки використанню UTF-8.

На основі визначених функціональних вимог та ролей користувачів сформовано перелік варіантів використання – типових сценаріїв взаємодії, які система повинна підтримувати для кожної категорії учасників. Кожен варіант використання описує конкретну дію користувача та очікувану реакцію системи, що дозволяє перейти від словесного опису вимог до формалізованої моделі поведінки програмного продукту.

Для формалізації взаємодії користувачів із системою побудовано UML-діаграму варіантів використання, яка представлена на рисунку 2.1. Діаграма відображає основні ролі користувачів і функції, доступні кожній із них. Гість взаємодіє з відкритою частиною системи, покупець виконує дії, пов'язані з участю в торгах, переглядом перемог, завантаженням інвойсу та спілкуванням із продавцем, а продавець відповідає за створення автомобілів, аукціонів і післяаукціонну комунікацію з покупцем. Діаграма демонструє, що базові функції (перегляд аукціонів) доступні гостям, тоді як розширені можливості (ставка, чат, інвойс) доступні лише авторизованим користувачам.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						36
Змн.	Арк.	№ докум.	Підпис	Дата		

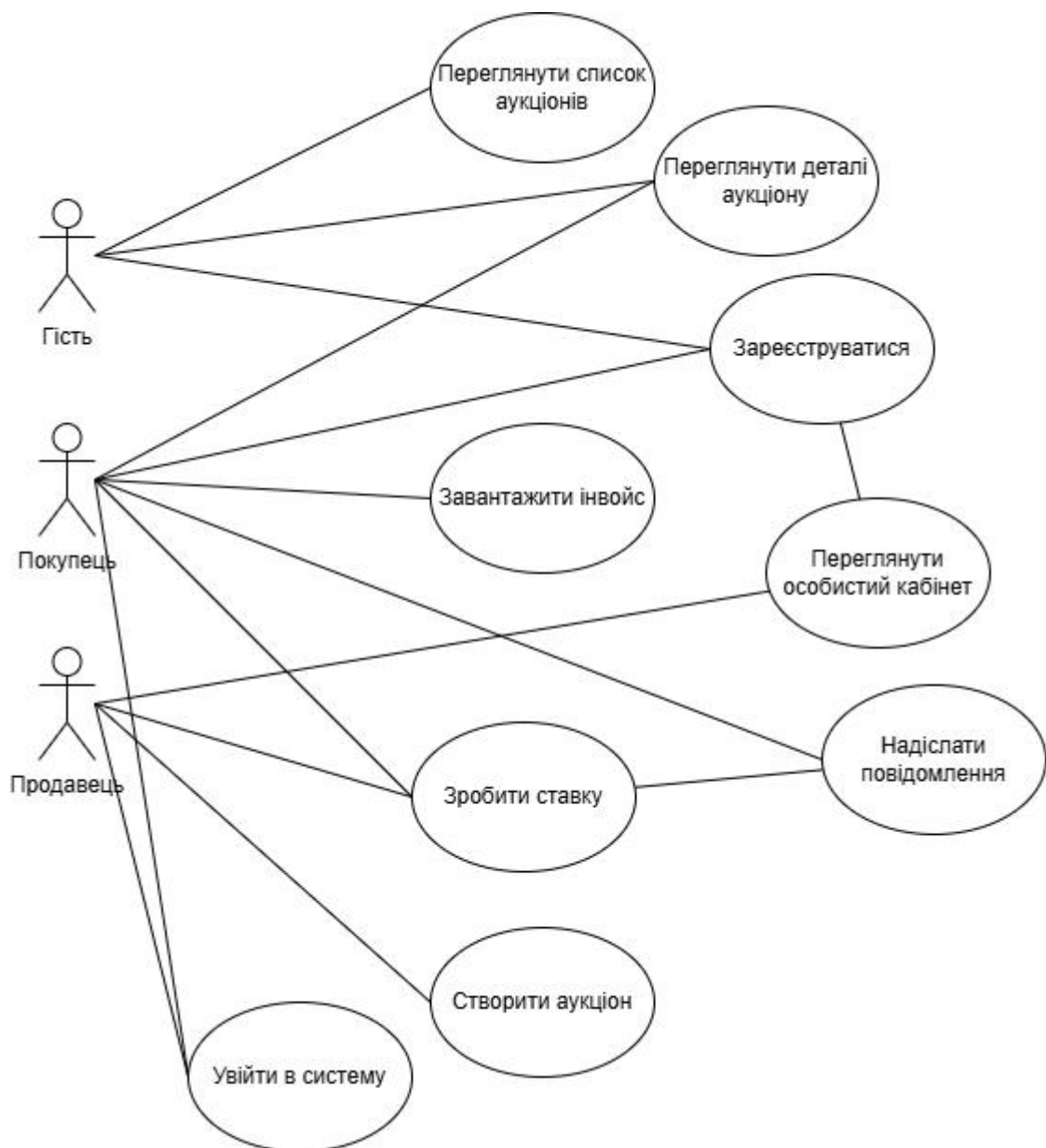


Рис. 2.1. Діаграма варіантів використання

Як показано на рисунку 2.1., базові функції системи доступні неавторизованому користувачу, тоді як створення аукціонів, подання ставок, перегляд особистих кабінетів, завантаження інвойсів і надсилання повідомлень потребують авторизації. Такий розподіл дозволяє відокремити відкриту частину вебсистеми від функцій, які потребують перевірки користувача та його ролі.

Для деталізації взаємодії користувачів із системою наведено опис ключових сценаріїв використання у табличній формі (табл. 2.1.–2.3.). На відміну від діаграми, яка демонструє лише перелік доступних дій, табличний опис розкриває послідовність кроків кожного сценарію, умови його виконання

та можливі альтернативні варіанти розвитку подій, що особливо важливо для процесів зі складною бізнес-логікою – подання ставки, автоматичного завершення торгів та обміну повідомленнями. Кожен сценарій описано за єдиною структурою: назва, актор, передумови виконання, основний потік дій, альтернативний потік у разі відхилення від штатного сценарію та постумови, що фіксують результат успішного виконання. Така уніфікована форма опису полегшує подальше використання сценаріїв як основи для написання тестових випадків на етапі перевірки працездатності системи.

Таблиця 2.1.

Опис сценарію «Подання ставки»

Елемент	Опис
Назва	Подання ставки
Актор	Покупець
Передумови	Покупець авторизований; аукціон активний
Основний потік	1. Покупець відкриває сторінку аукціону. 2. Вводить суму ставки. 3. Система перевіряє, що сума перевищує поточну максимальну ставку. 4. Ставка зберігається в базі даних. 5. Поточна ціна аукціону оновлюється.
Альтернативний потік	3а. Якщо сума менша або дорівнює поточній максимальній — система відхиляє ставку з повідомленням про помилку.
Постумови	Ставка відображається в історії торгів аукціону.

Таблиця 2.2.

Опис сценарію «Автоматичне завершення аукціону»

Елемент	Опис
Назва	Автоматичне завершення аукціону
Актор	Система (планувальник)
Передумови	Час завершення аукціону минув; статус ще не позначено завершеним
Основний потік	1. Планувальник виявляє прострочений аукціон. 2. Система отримує всі ставки аукціону. 3. Визначається ставка з найбільшою сумою. 4. Створюється замовлення для переможця. 5. Статус аукціону змінюється на завершений.
Альтернативний потік	2а. Якщо ставок немає — аукціон позначається завершеним без створення замовлення.
Постумови	Переможець отримує доступ до розділу «Мої перемоги».

Опис сценарію «Надсилання повідомлення в чаті»

Елемент	Опис
Назва	Надсилання повідомлення
Актор	Покупець, Продавець
Передумови	Аукціон завершено; користувач авторизований
Основний потік	1. Користувач відкриває чат аукціону. 2. Вводить текст повідомлення. 3. Повідомлення передається через Socket.IO. 4. Повідомлення зберігається в базі даних. 5. Отримувач бачить повідомлення в реальному часі.
Постумови	Повідомлення додається до історії чату аукціону.

Об'єктна модель системи побудована на основі сутностей User, Vehicle, Auction, Bid, Order та Message (рис. 2.2.). Сутність User описує користувача системи, його облікові дані та роль. Vehicle зберігає характеристики автомобіля, який продавець додає до системи. Auction описує аукціон, його часові межі, статус завершення, продавця та пов'язаний автомобіль. Bid фіксує ставку покупця, її суму, час створення та належність до конкретного аукціону. Order створюється після завершення торгів і використовується для збереження інформації про переможця, суму до сплати та статус замовлення. Message забезпечує збереження повідомлень між покупцем і продавцем у межах конкретного аукціону.

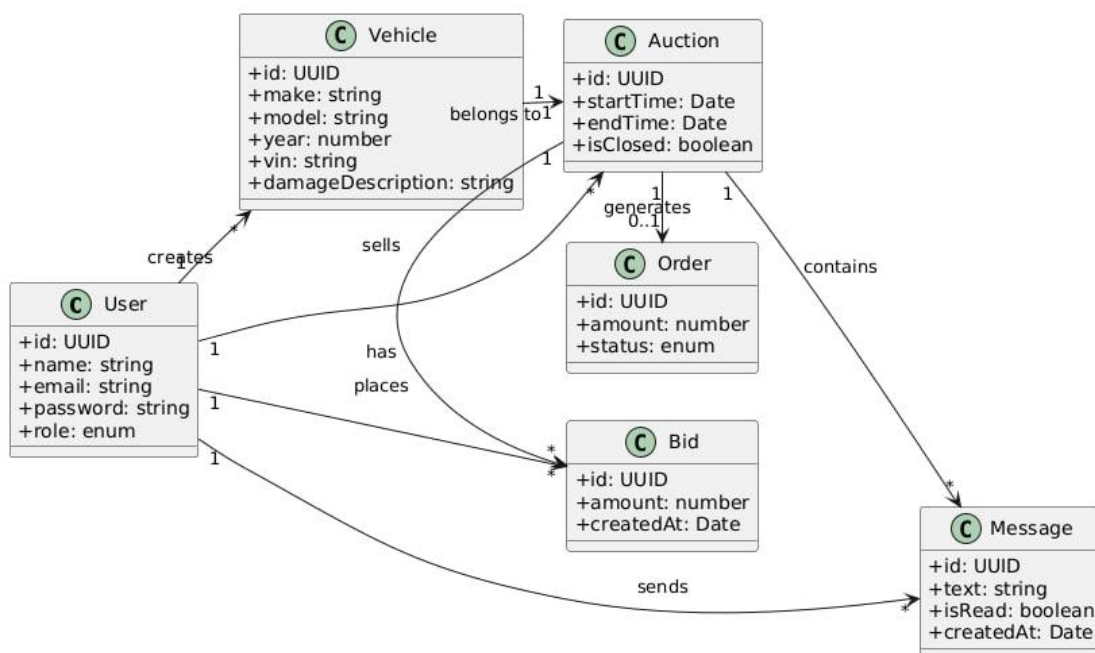


Рис. 2.2. Діаграма класів об'єктної моделі системи

Усі сутності реалізовані як класи TypeScript із використанням декораторів TypeORM [6, 11]. Зв'язки між сутностями відображають бізнес-логіку системи: один продавець може мати багато автомобілів і аукціонів, один аукціон може містити багато ставок, після завершення аукціону створюється замовлення для покупця-переможця, а повідомлення прив'язуються до конкретного аукціону, відправника та отримувача. Така об'єктна структура забезпечує логічну цілісність даних і створює основу для реалізації функціональних модулів системи.

2.2. Розробка бази даних системи

База даних є центральним компонентом вебсистеми аукціонів авто, оскільки саме вона забезпечує збереження, цілісність і доступність інформації про користувачів, транспортні засоби, аукціони, ставки, замовлення та повідомлення. Для реалізації обрано реляційну систему керування базами даних PostgreSQL, яка підходить для роботи зі структурованими даними та підтримує зв'язки між сутностями, транзакції, індекси й унікальні ідентифікатори. Така модель є доцільною для системи аукціонів, оскільки між основними об'єктами існують чіткі залежності: користувач створює автомобілі та аукціони, покупець робить ставки, після завершення торгів створюється замовлення, а повідомлення прив'язуються до конкретної угоди.

Фізична структура бази даних представлена у вигляді ER-діаграми на рисунку 2.3. Вона відображає основні таблиці системи, їхні поля, первинні та зовнішні ключі, а також взаємозв'язки між сутностями. У межах системи використовуються таблиці users, vehicles, auctions, bids, orders та messages. Разом вони формують основу для реалізації основних бізнес-процесів: реєстрації користувачів, створення автомобілів, проведення аукціонів, подання ставок, визначення переможця, формування замовлення та обміну повідомленнями між покупцем і продавцем.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

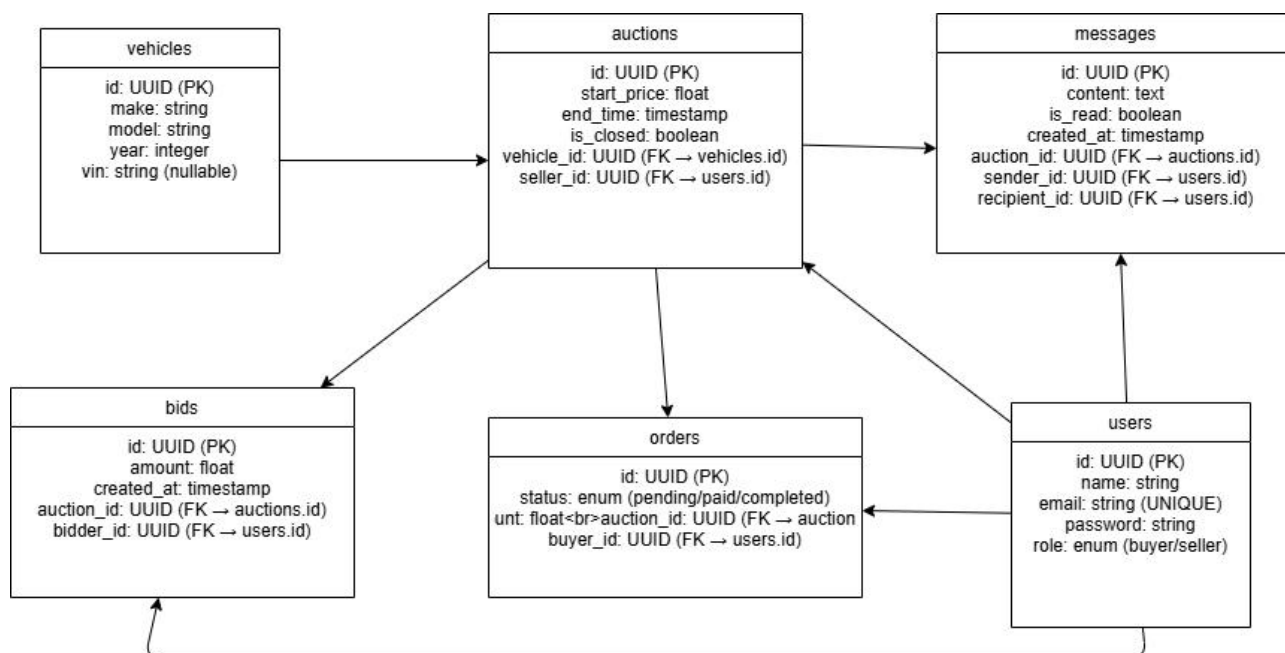


Рис. 2.3. Діаграма бази даних

Як показано на рисунку 2.3., структура бази даних складається з шести основних сутностей, які відповідають ключовим об'єктам предметної області. Центральними таблицями є users, vehicles та auctions, оскільки саме вони визначають користувачів системи, транспортні засоби та створені аукціони. Таблиці bids, orders та messages доповнюють цю структуру й забезпечують роботу ставок, післяаукціонних замовлень та комунікації між покупцем і продавцем.

Таблиця users призначена для зберігання даних користувачів системи. У ній міститься унікальний ідентифікатор користувача, ім'я, електронна пошта, хеш пароля та роль у системі. Роль визначає доступні сценарії роботи користувача: покупець може брати участь в аукціонах і робити ставки, а продавець може додавати автомобілі та створювати аукціони. Поле електронної пошти має бути унікальним, оскільки воно використовується під час автентифікації та не може повторюватися для різних облікових записів. Пароль не зберігається у відкритому вигляді, а записується до бази у вигляді хешу, що підвищує рівень захисту облікових даних.

Таблиця vehicles використовується для збереження інформації про автомобілі, які продавці додають до системи. Вона містить унікальний

ідентифікатор транспортного засобу, марку, модель, рік випуску, VIN-код, опис пошкоджень, додатковий опис і посилання на зображення. Ці дані використовуються під час створення аукціону та відображаються на сторінці конкретного лота. Автомобіль належить продавцю, тому таблиця vehicles логічно пов'язана з таблицею users. Один продавець може додати багато автомобілів, однак кожен конкретний активний аукціон пов'язується з одним транспортним засобом.

Таблиця auctions відповідає за збереження даних про аукціони. У ній фіксується унікальний ідентифікатор аукціону, час початку, час завершення, статус завершення, платіжні реквізити продавця, зв'язок із автомобілем і зв'язок із продавцем. Поле часу завершення використовується планувальником для автоматичної перевірки активних аукціонів. Якщо час завершення вже минув, а статус аукціону ще не позначено як завершений, серверна частина обробляє такий запис, визначає переможця за найбільшою ставкою та створює замовлення.

Таблиця bids зберігає ставки покупців. Кожна ставка має власний ідентифікатор, суму, час створення, зв'язок із конкретним аукціоном і зв'язок із користувачем, який зробив ставку. Один аукціон може містити багато ставок, оскільки в торгах можуть брати участь різні покупці. Під час завершення аукціону саме дані з таблиці bids використовуються для визначення переможця. Система аналізує всі ставки конкретного аукціону, знаходить найбільшу суму та створює замовлення для користувача, який зробив цю ставку.

Таблиця orders використовується для збереження інформації про замовлення, яке створюється після завершення аукціону. У ній зберігається ідентифікатор замовлення, сума до сплати, статус, зв'язок з аукціоном і зв'язок із покупцем. Статус замовлення дає змогу відстежувати стан післяаукціонної взаємодії між покупцем і продавцем. На основі даних із таблиці orders формується інвойс, який містить інформацію про замовлення, автомобіль, покупця та суму до сплати.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця messages забезпечує збереження повідомлень між покупцем і продавцем. Вона містить ідентифікатор повідомлення, текст, дату створення, статус прочитання, ідентифікатор аукціону, відправника та отримувача. Прив'язка повідомлень до аукціону дає змогу зберігати історію листування в межах конкретної угоди. Завдяки полю is_read система може визначати кількість непрочитаних повідомлень і відображати відповідну індикацію в інтерфейсі користувача.

Зв'язки між таблицями реалізовані переважно за принципом «один-до-багатьох». Один користувач у ролі продавця може мати багато автомобілів і багато аукціонів, а користувач у ролі покупця може робити багато ставок у різних аукціонах. Один аукціон пов'язаний з одним автомобілем, але може містити багато ставок і повідомлень. Після завершення торгів для аукціону створюється одне замовлення, яке фіксує переможця та суму до сплати. Така структура відповідає логіці предметної області й дозволяє коректно відобразити процес онлайн-аукціону на рівні бази даних.

Для забезпечення цілісності даних у системі використовуються зовнішні ключі між основними таблицями. Вони дають змогу підтримувати коректні зв'язки між користувачами, автомобілями, аукціонами, ставками, замовленнями та повідомленнями. Запити до бази даних виконуються через TypeORM-репозиторії, що дозволяє працювати з даними на рівні TypeScript-сутностей і зменшує потребу у ручному написанні SQL-запитів. Такий підхід спрощує супровід програмного коду та знижує ризик помилок під час роботи з базою даних.

Таким чином, розроблена структура бази даних забезпечує збереження всіх даних, необхідних для роботи вебсистеми аукціонів авто. Вона підтримує логічні зв'язки між основними сутностями, дозволяє реалізувати автоматичне завершення аукціонів, визначення переможця, створення замовлень, генерацію інвойсів і роботу чату між учасниками угоди.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

2.3. Проектування та реалізація алгоритмів роботи системи

Реалізація бізнес-логіки вебсистеми аукціонів авто базується на наборі алгоритмів, які описують основні сценарії роботи користувачів і серверної частини. До ключових процесів належать автоматичне завершення аукціонів, визначення переможця, робота двостороннього чату, генерація інвойсу та подання ставок. Для формалізації цих процесів використано UML-діаграми послідовності, які дають змогу показати порядок взаємодії між frontend-частиною, backend-сервісами та базою даних [18].

Одним із головних алгоритмів системи є автоматичне завершення аукціонів. Його призначення полягає в тому, щоб після завершення часу торгів система самостійно перевірила активні аукціони, визначила переможця та створила відповідне замовлення. Такий підхід виключає необхідність ручного втручання адміністратора або продавця й забезпечує однакові правила завершення торгів для всіх користувачів.

Автоматична перевірка реалізується за допомогою планувальника завдань NestJS ScheduleModule [22]. Планувальник періодично запускає перевірку активних аукціонів і звертається до сервісу аукціонів. Сервіс отримує з бази даних записи, у яких час завершення вже минув, а статус аукціону ще не позначений як завершений. Для кожного такого аукціону система завантажує пов'язані ставки, визначає найбільшу суму, встановлює переможця та створює замовлення зі статусом очікування оплати. Після цього статус аукціону оновлюється, що унеможливорює подання нових ставок.

Повний лістинг реалізації планувальника автоматичного завершення аукціонів наведено в додатку В. Код сервісу, який виконує пошук завершених аукціонів, визначає переможця та створює замовлення, наведено в додатку Г.

UML-діаграма послідовності автоматичного завершення аукціону представлена на рисунку 2.4.

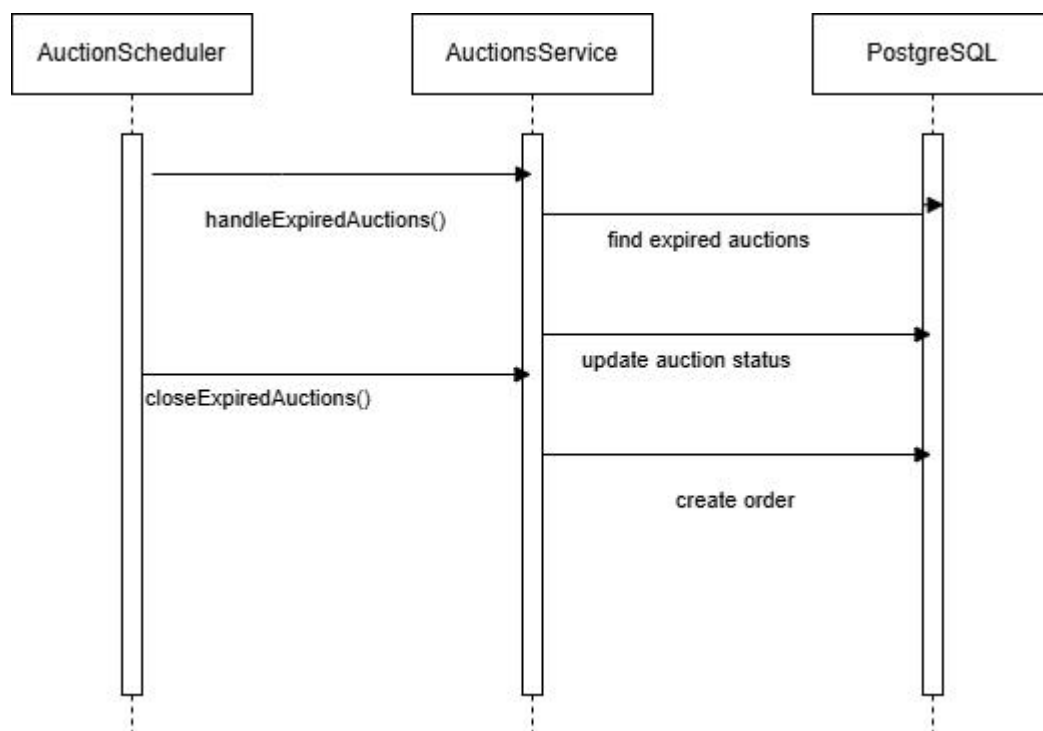


Рис. 2.4. Діаграма послідовності взаємодії компонентів при завершенні аукціону

Як показано на рисунку 2.4., процес починається з виклику планувальника AuctionScheduler. Далі запит передається до AuctionsService, який звертається до PostgreSQL для отримання активних аукціонів із завершеним часом. Після аналізу ставок сервіс визначає переможця, створює замовлення та оновлює статус аукціону. Така послідовність забезпечує автоматизоване завершення торгів і збереження результатів у базі даних.

Наступним важливим алгоритмом є робота двостороннього чату між покупцем і продавцем. Чат використовується після завершення аукціону для уточнення деталей угоди, оплати або передачі автомобіля. Для реалізації обміну повідомленнями використовується Socket.IO, що забезпечує двосторонню комунікацію між клієнтом і сервером у реальному часі [13, 17, 23, 25].

Під час надсилання повідомлення frontend передає на backend ідентифікатор аукціону, відправника, отримувача та текст повідомлення. Backend зберігає повідомлення в таблиці messages, після чого воно може бути відображене отримувачу. Якщо користувач перебуває в чаті, повідомлення

надходить у реальному часі. Якщо користувач офлайн, повідомлення зберігається в базі даних і стає доступним після наступного відкриття чату.

Під час завантаження історії чату система виконує запит до таблиці messages за ідентифікатором аукціону та користувача. Повідомлення повертаються у хронологічному порядку, що дозволяє відтворити повну історію спілкування. Непрочитані повідомлення після відкриття чату позначаються як прочитані, завдяки чому система може коректно відображати кількість нових повідомлень у навігаційній панелі.

Повний лістинг реалізації сервісу повідомлень наведено в додатку Е.

UML-діаграма послідовності роботи чату представлена на рисунку 2.5.

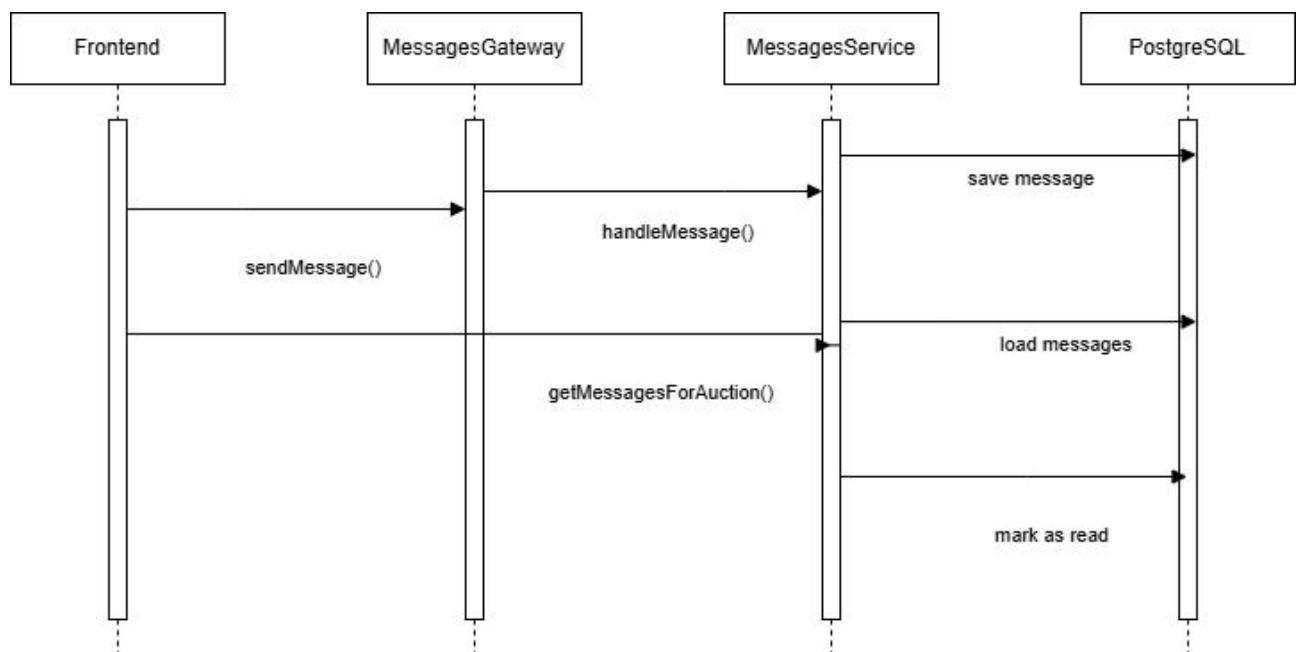


Рис. 2.5. Діаграма послідовності роботи чату

На рисунку 2.5. показано взаємодію між frontend-частиною, сервісом повідомлень і базою даних. Користувач надсилає повідомлення через інтерфейс, backend обробляє запит, зберігає повідомлення в PostgreSQL і повертає оновлену історію листування. Такий алгоритм забезпечує збереження повідомлень, підтримку історії чату та коректну роботу індикатора непрочитаних повідомлень.

Окремим алгоритмом системи є генерація інвойсу. Інвойс формується після завершення аукціону та створення замовлення. Його призначення полягає

в тому, щоб надати покупцю текстовий документ із даними про замовлення, автомобіль, покупця, продавця та суму до сплати. У поточній реалізації інвойс створюється у форматі TXT з підтримкою кодування UTF-8, що забезпечує коректне відображення українського тексту.

Під час запиту на завантаження інвойсу frontend звертається до OrdersController. Контролер передає ідентифікатор замовлення до OrdersService, який отримує з бази даних пов'язані дані про замовлення, аукціон, автомобіль і покупця. На основі цих даних формується текстовий шаблон інвойсу, після чого він перетворюється в буфер UTF-8 і повертається користувачу як файл для завантаження [9, 24].

Повний лістинг реалізації генерації інвойсу наведено в додатку Ж.

UML-діаграма послідовності генерації інвойсу представлена на рисунку 2.6.

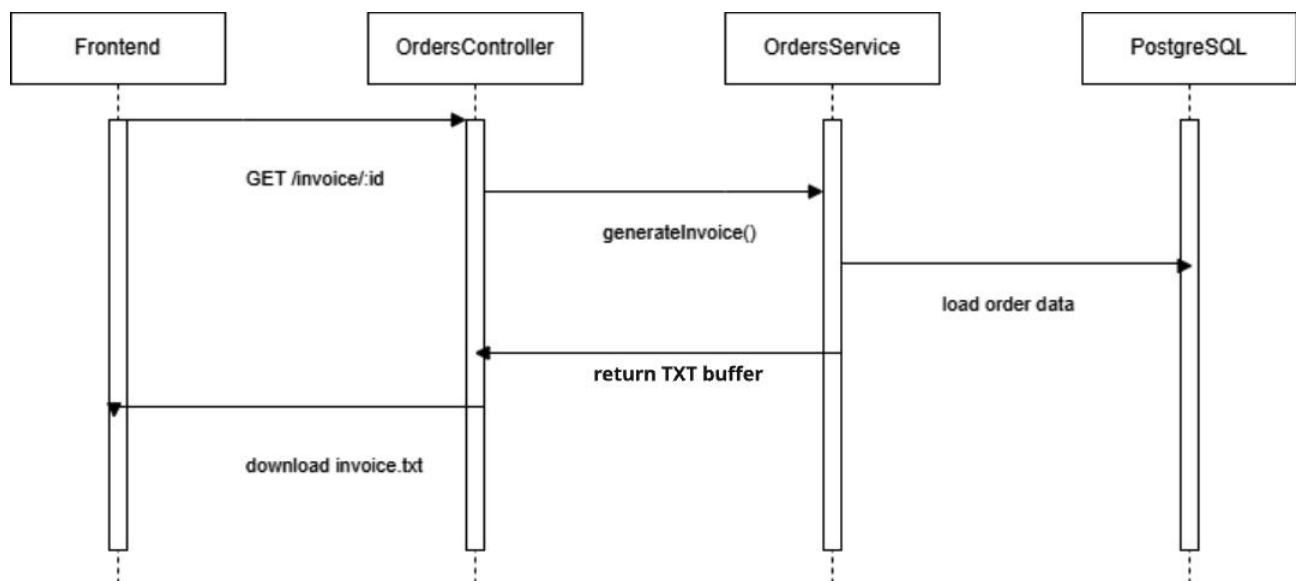


Рис. 2.6. Діаграма послідовності генерації інвойсу

Як показано на рисунку 2.6., процес генерації інвойсу починається із запиту користувача на frontend-частині. Далі запит обробляється контролером замовлень, після чого сервіс замовлень отримує необхідні дані з PostgreSQL і формує текстовий файл. Важливо, що результатом роботи сервісу є саме TXT buffer, а не PDF buffer, оскільки інвойс у системі генерується у текстовому форматі.

Алгоритм подання ставки забезпечує коректну участь покупців в активних аукціонах. Перед збереженням ставки система перевіряє, чи існує аукціон, чи він не завершений і чи сума ставки є коректною. Після успішної перевірки створюється новий запис у таблиці bids, який містить суму ставки, ідентифікатор аукціону та покупця. Після збереження ставки frontend отримує оновлену інформацію про поточну ціну аукціону [5, 8, 24].

Повний лістинг реалізації подання ставки наведено в додатку И.

Таким чином, ключові алгоритми системи охоплюють повний цикл роботи онлайн-аукціону. Автоматичне завершення торгів забезпечує визначення переможця без ручного втручання, чат підтримує післяаукціонну комунікацію між учасниками, генерація інвойсу формує документальне підтвердження угоди, а алгоритм подання ставки забезпечує коректне проведення торгів. Формалізація цих процесів за допомогою UML-діаграм і винесення повних лістингів коду в додатки роблять опис системи більш структурованим і зручним для сприйняття.

2.4. Реалізація додатку веб-орієнтованої системи аукціонів авто

Реалізація програмного комплексу є ключовим етапом кваліфікаційної роботи, оскільки саме на цьому етапі теоретичні положення, архітектурні рішення та розроблені алгоритми перетворюються на працездатний програмний продукт. Систему реалізовано з дотриманням принципів модульності, безпеки, масштабованості та зручності подальшого супроводу. Програмний комплекс складається з клієнтської частини, що забезпечує взаємодію користувача з інтерфейсом, та серверної частини, яка відповідає за бізнес-логіку, обробку запитів, автентифікацію, роботу з аукціонами, ставками, повідомленнями, замовленнями та базою даних.

Система побудована за клієнт-серверною архітектурою з чітким розділенням відповідальностей між frontend і backend-рівнями. Клієнтська частина реалізована з використанням Next.js 14, що забезпечує створення

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						48
Змн.	Арк.	№ докум.	Підпис	Дата		

вебінтерфейсу на основі React, підтримку маршрутизації та ефективну побудову сторінок [4, 5]. Для підвищення надійності коду використано TypeScript, який забезпечує статичну типізацію та допомагає зменшити кількість помилок під час розробки [6]. Оформлення інтерфейсу виконано за допомогою Tailwind CSS, що дозволяє створювати адаптивні компоненти та забезпечувати коректне відображення сторінок на різних пристроях [7]. Для обміну даними між frontend-частиною та backend API використовується Axios, який виконує HTTP-запити до серверної частини [8, 24]. Функції, пов'язані з оновленням даних у реальному часі, зокрема чат і події ставок, реалізуються за допомогою Socket.IO Client [13, 17, 25].

Серверна частина системи реалізована на основі NestJS 10, який забезпечує модульну структуру застосунку, використання контролерів, сервісів і механізму dependency injection [9, 10]. Для взаємодії з реляційною базою даних застосовано TypeORM, що дозволяє описувати таблиці у вигляді TypeScript-сутностей і виконувати операції з даними через репозиторії [11]. Як систему керування базами даних використано PostgreSQL 15, оскільки вона підтримує реляційні зв'язки, транзакції, UUID та роботу зі структурованими даними [12]. Автентифікація користувачів реалізована за допомогою JWT-токенів, що дозволяє захищати маршрути та передавати інформацію про користувача між клієнтом і сервером [19, 26, 30]. Для автоматичного завершення аукціонів використовується ScheduleModule, який дає змогу запускати періодичні задачі за розкладом [22]. Для двосторонньої комунікації в реальному часі застосовано Socket.IO, що використовується під час роботи чату та оновлення ставок [13, 23, 25].

Такий технологічний стек дозволяє реалізувати всі основні функції вебсистеми аукціонів авто, зокрема реєстрацію та автентифікацію користувачів, створення автомобілів і аукціонів, подання ставок, автоматичне завершення торгів, визначення переможця, формування замовлення, генерацію інвойсу та обмін повідомленнями між покупцем і продавцем. Поєднання Next.js і NestJS

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						49
Змн.	Арк.	№ докум.	Підпис	Дата		

забезпечує чіткий поділ між інтерфейсом користувача та серверною логікою, а використання PostgreSQL і TypeORM дозволяє зберігати структуровані дані та підтримувати зв'язки між основними сутностями системи.

Автентифікація є першим етапом взаємодії користувача із системою. Вона реалізована за допомогою JWT-токенів, які формуються після успішного входу користувача або реєстрації. Під час авторизації система спочатку шукає користувача за електронною поштою. Якщо користувача не знайдено або введений пароль не відповідає збереженому хешу, сервер повертає помилку авторизації. Якщо перевірка проходить успішно, backend формує JWT-токен, у якому містяться ідентифікатор користувача, електронна пошта та роль. Повний лістинг реалізації методу автентифікації наведено в додатку А.

Доступ до захищених маршрутів системи контролюється за допомогою JwtAuthGuard. Це означає, що особисті кабінети, перегляд власних перемог, завантаження інвойсів і робота з повідомленнями доступні лише авторизованим користувачам. Під час кожного запиту frontend передає JWT-токен у заголовок авторизації, а backend перевіряє його коректність і визначає користувача. Якщо токен відсутній або недійсний, доступ до маршруту блокується. Приклад реалізації захисту маршруту наведено в додатку Б.

Центральним механізмом системи є автоматичне завершення аукціонів. Воно реалізоване за допомогою планувальника завдань, який періодично перевіряє наявність активних аукціонів із завершеним часом торгів. Якщо система знаходить такий аукціон, вона передає його на обробку сервісу аукціонів. Планувальник працює автоматично, без участі користувача або адміністратора, що забезпечує однакові умови завершення торгів для всіх учасників. Повний лістинг планувальника автоматичного завершення аукціонів наведено в додатку В.

Сервіс аукціонів виконує основну бізнес-логіку завершення торгів. Спочатку він отримує з бази даних аукціони, у яких час завершення вже минув, але статус ще не позначено як завершений. Далі для кожного такого аукціону

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

перевіряється наявність ставок. Якщо ставок немає, аукціон пропускається. Якщо ставки існують, система визначає найбільшу ставку, встановлює користувача, який її зробив, як переможця та створює для нього замовлення зі статусом очікування оплати. Після цього аукціон позначається як завершений, що блокує можливість подання нових ставок. Повний лістинг відповідного сервісного методу наведено в додатку Г.

Двосторонній чат реалізовано для забезпечення комунікації між покупцем і продавцем після завершення аукціону. WebSocket-шлюз приймає повідомлення від клієнта, передає його до сервісу повідомлень для збереження в базі даних і надсилає оновлення учасникам відповідного чату. Користувач підключається до кімнати чату за ідентифікатором аукціону, що дозволяє ізолювати повідомлення конкретної угоди від інших чатів. Повний лістинг WebSocket-шлюзу повідомлень наведено в додатку Д.

Сервіс повідомлень відповідає за збереження та завантаження історії спілкування. Під час відкриття чату система отримує всі повідомлення, пов'язані з конкретним аукціоном і поточним користувачем. Повідомлення впорядковуються за часом створення, що дозволяє відобразити історію у правильній хронологічній послідовності. Якщо серед отриманих повідомлень є непрочитані повідомлення, адресовані поточному користувачу, система позначає їх як прочитані. Це забезпечує коректну роботу індикатора непрочитаних повідомлень у навігаційній панелі. Повний лістинг реалізації сервісу повідомлень наведено в додатку Е.

Генерація інвойсу реалізована як окремий механізм у модулі замовлень. Інвойс формується після завершення аукціону та створення замовлення для переможця. Сервіс замовлень отримує з бази даних інформацію про замовлення, аукціон, автомобіль і покупця. Якщо замовлення не знайдено, система повертає помилку. Якщо дані отримано успішно, формується текстовий документ, який містить номер замовлення, дату, статус, ім'я покупця, характеристики

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

автомобіля, VIN і суму до сплати. Повний лістинг генерації текстового інвойсу наведено в додатку Ж.

Передача інвойсу користувачу виконується через контролер замовлень. Контролер отримує ідентифікатор замовлення, викликає сервіс генерації інвойсу та повертає сформований файл як вкладення. У відповіді встановлюється тип вмісту text/plain із кодуванням UTF-8, а також назва файлу у форматі invoice_<ідентифікатор замовлення>.txt. Такий підхід дозволяє завантажувати інвойс у будь-якому сучасному браузері без додаткових бібліотек або конвертації у PDF. Повний лістинг контролера генерації інвойсу наведено в додатку З.

Під час реалізації програмного комплексу окрему увагу приділено забезпеченню безпеки та коректної обробки даних. Усі DTO-класи проходять валідацію за допомогою class-validator, що дає змогу перевіряти вхідні дані ще до виконання бізнес-логіки [21]. Запити до бази даних виконуються через TypeORM із використанням параметризованих запитів, що знижує ризик SQL-ін'єкцій [11, 28]. Паролі користувачів не зберігаються у відкритому вигляді, а хешуються за допомогою bcrypt, що підвищує рівень захисту облікових записів [27].

Особисті кабінети користувачів реалізовані відповідно до ролей у системі. Продавець має доступ до розділу «Мої аукціони», де відображаються його активні та завершені аукціони, інформація про переможця завершених торгів і можливість перейти до спілкування з покупцем. Покупець використовує розділ «Мої перемоги», у якому відображаються виграні аукціони, статуси замовлень, можливість завантаження інвойсу та перехід до чату з продавцем.

Інтерфейс системи розроблено з урахуванням адаптивності та зручності використання на різних пристроях. Для стилізації застосовано Tailwind CSS, що дозволяє створювати адаптивні компоненти та забезпечувати коректне відображення сторінок як на персональних комп'ютерах, так і на мобільних

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дата		

пристроях [7]. Повні лістинги ключових компонентів програмного комплексу наведено у додатках.

Висновки до другого розділу

У другому розділі було виконано проектування та реалізацію основних компонентів веборієнтованої системи аукціонів авто. На основі визначених функціональних і нефункціональних вимог описано ролі користувачів, побудовано UML-діаграму варіантів використання та сформовано об'єктну модель системи. Визначені сутності User, Vehicle, Auction, Bid, Order та Message відображають основні об'єкти предметної області та забезпечують логічну основу для реалізації програмного комплексу.

Також було розроблено структуру бази даних у PostgreSQL. Вона забезпечує збереження даних про користувачів, автомобілі, аукціони, ставки, замовлення та повідомлення. Зв'язки між таблицями дозволяють підтримувати цілісність даних і реалізовувати основні бізнес-процеси системи. Використання TypeORM забезпечує зручну роботу з базою даних на рівні TypeScript-сутностей і репозиторіїв.

У розділі було спроектовано основні алгоритми роботи системи. Автоматичне завершення аукціонів реалізується через планувальник завдань NestJS, переможець визначається на основі найбільшої ставки, чат забезпечує обмін повідомленнями між покупцем і продавцем, а генерація інвойсу формує текстовий документ із даними замовлення. Ці процеси формалізовано за допомогою UML-діаграм послідовності, що спрощує розуміння логіки роботи системи.

Програмний комплекс реалізовано з використанням сучасного технологічного стеку. Клієнтська частина побудована на Next.js, TypeScript і Tailwind CSS, а серверна частина реалізована на NestJS із використанням TypeORM, PostgreSQL, JWT та Socket.IO. Такий підхід забезпечує чітке

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						53
Змн.	Арк.	№ докум.	Підпис	Дата		

розділення відповідальностей між frontend і backend-рівнями, а також створює основу для подальшого розвитку системи.

Окрему увагу приділено безпеці програмного комплексу. Паролі користувачів зберігаються у хешованому вигляді, запити до бази даних виконуються через TypeORM із використанням параметризованих запитів, а доступ до захищених маршрутів контролюється за допомогою JWT-автентифікації. Таким чином, завдання другого розділу виконано, а реалізована система має модульну структуру, відповідає визначеним вимогам і готова до тестування.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ

3.1. Порядок встановлення та налаштування параметрів системи через Docker

Вебсистема аукціонів авто реалізована як програмний комплекс, що складається з кількох взаємопов'язаних компонентів. До його складу входять frontend-частина, backend API та база даних PostgreSQL. Для забезпечення стабільного запуску цих компонентів використовується контейнеризація за допомогою Docker і Docker Compose [14, 15]. Такий підхід дозволяє запускати кожен компонент системи в окремому ізолюваному контейнері, але водночас забезпечувати їхню взаємодію в межах єдиної Docker-мережі.

Використання Docker є доцільним для цієї системи, оскільки програмний комплекс має декілька залежностей і потребує узгодженої роботи серверної частини, клієнтської частини та бази даних. Без контейнеризації користувачу потрібно було б окремо встановлювати Node.js, PostgreSQL, налаштовувати змінні середовища, перевіряти версії пакетів і вручну запускати кожен компонент. Docker Compose спрощує цей процес, оскільки всі параметри запуску описуються в одному конфігураційному файлі, після чого система може бути запущена однією командою.

Для локального розгортання системи необхідно мати встановлений Docker Engine версії не нижче 20.10 та Docker Compose версії не нижче 2.0. Система може бути запущена на Linux, macOS або Windows 10/11 з підтримкою WSL2. Наявність WSL2 для Windows є важливою, оскільки вона забезпечує коректну роботу Linux-контейнерів і дозволяє уникнути проблем із файловою системою, мережею та продуктивністю контейнерів.

Першим етапом розгортання є отримання вихідного коду програмного комплексу з репозиторію. Після клонування репозиторію користувач переходить до кореневої директорії проєкту, у якій розташований файл `docker-compose.yml`, а також окремі директорії frontend і backend-частин. Саме з

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

кореневої директорії виконується запуск усієї системи, оскільки Docker Compose використовує конфігурацію, розташовану на цьому рівні.

Перед запуском системи необхідно налаштувати змінні середовища. Для цього в корені проєкту створюється файл `.env`, у якому задаються параметри підключення до бази даних, секретний ключ для JWT-автентифікації та адреса backend API для клієнтської частини. До параметрів бази даних належать хост, порт, ім'я користувача, пароль і назва бази даних. Для локального запуску як хост бази даних використовується назва Docker-сервісу `db`, оскільки backend звертається до PostgreSQL не через `localhost`, а через внутрішню мережу Docker Compose.

Окремим важливим параметром є `JWT_SECRET`, який використовується серверною частиною для підпису та перевірки JWT-токенів. У локальному середовищі може застосовуватися тестове значення, однак для виробничого середовища необхідно використовувати складний секретний ключ довжиною не менше 32 символів. Це важливо для запобігання підробці токенів і несанкціонованому доступу до захищених маршрутів системи.

Змінна `NEXT_PUBLIC_API_URL` використовується frontend-частиною для визначення адреси backend API. У локальному середовищі backend доступний за адресою `http://localhost:3001`, тому саме це значення використовується для надсилання HTTP-запитів із клієнтської частини. Після запуску системи користувач відкриває frontend у браузері за адресою `http://localhost:3000`, а всі запити до серверної частини автоматично спрямовуються на backend API.

Запуск програмного комплексу виконується командою `docker-compose up --build`. Параметр `--build` означає, що перед запуском Docker Compose виконає збірку образів для frontend і backend-частин на основі відповідних Dockerfile. Під час першого запуску цей процес може зайняти певний час, оскільки Docker завантажує базові образи, встановлює залежності Node.js і

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						56
Змн.	Арк.	№ докум.	Підпис	Дата		

створює робочі контейнери. Після завершення збірки всі сервіси запускаються автоматично.

Перелік основних змінних середовища, необхідних для запуску системи, узагальнено в таблиці 3.1.

Таблиця 3.1.

Змінні середовища вебсистеми аукціонів авто

Сервіс	Змінна	Призначення
backend	DB_HOST	Хост бази даних (назва Docker-сервісу db)
backend	DB_PORT	Порт PostgreSQL
backend	DB_USERNAME	Ім'я користувача бази даних
backend	DB_PASSWORD	Пароль користувача бази даних
backend	DB_NAME	Назва бази даних
backend	JWT_SECRET	Секретний ключ підпису JWT-токенів

У конфігурації Docker Compose база даних PostgreSQL запускається як окремий сервіс. Для неї задаються назва бази даних, ім'я користувача, пароль, відкритий порт і volume для постійного збереження даних. Використання volume `postgres\_data` дозволяє не втрачати дані після перезапуску контейнерів, оскільки інформація зберігається поза життєвим циклом конкретного контейнера. Це важливо для стабільної роботи системи під час тестування та повторного запуску. Загальну схему взаємодії контейнерів у середовищі Docker Compose зображено на рисунку 3.1.

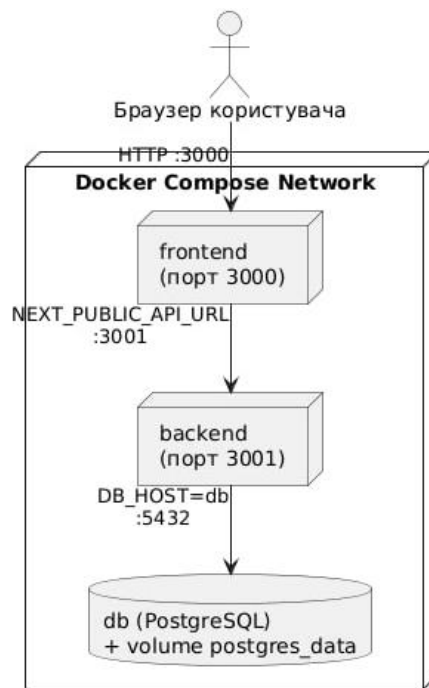


Рис. 3.1. Схема контейнерів Docker Compose

Backend-сервіс збирається з директорії `backend` і відкриває порт 3001. Він отримує необхідні параметри через змінні середовища, зокрема адресу бази даних і секретний ключ JWT. У конфігурації також визначено залежність backend від сервісу бази даних. Це означає, що Docker Compose спочатку запускає PostgreSQL, а вже потім запускає серверну частину. Такий порядок запуску зменшує ризик помилок підключення до бази даних на старті системи.

Frontend-сервіс збирається з директорії `frontend` і відкриває порт 3000. Він відповідає за відображення інтерфейсу користувача, сторінок аукціонів, форм реєстрації та входу, особистих кабінетів, сторінки конкретного аукціону, чату та розділу перемог покупця. Після запуску frontend-контейнер взаємодіє з backend через налаштовану адресу API. Завдяки цьому користувач працює з системою через браузер, а вся бізнес-логіка обробляється серверною частиною.

Після успішного запуску системи у терміналі відображаються журнали роботи контейнерів. За ними можна перевірити, чи успішно запущено базу даних, backend і frontend. Якщо запуск виконано без помилок, backend API доступний за адресою `http://localhost:3001`, а frontend-інтерфейс – за адресою `http://localhost:3000`. Для зупинки системи можна використати команду

`docker-compose down`, яка завершує роботу контейнерів. Якщо потрібно повністю видалити також збережені дані бази, додатково видаляється відповідний Docker volume.

Конфігурація `docker-compose.yml` забезпечує автоматизований запуск усіх основних сервісів у правильному порядку. Вона описує образ PostgreSQL, правила збірки frontend і backend, відкриті порти, змінні середовища, залежності між контейнерами та volume для збереження даних. Повний лістинг конфігураційного файлу `docker-compose.yml` наведено в додатку К, а приклад структури файлу `.env` - у додатку Л.

Для розгортання системи у виробничому середовищі необхідно врахувати додаткові вимоги безпеки. Зокрема, потрібно замінити тестові паролі бази даних, задати надійний `JWT\_SECRET`, обмежити доступ до службових портів, налаштувати HTTPS-з'єднання та використовувати окремі облікові записи для доступу до бази даних. Також доцільно запускати PostgreSQL в окремому керованому середовищі або на виділеному сервері, якщо очікується збільшення навантаження.

Таким чином, використання Docker і Docker Compose дозволяє зробити процес встановлення та налаштування вебсистеми аукціонів авто простим, повторюваним і контрольованим. Контейнеризація забезпечує однакову поведінку системи на різних комп'ютерах, спрощує тестування та створює основу для подальшого розгортання програмного комплексу в реальному середовищі експлуатації.

3.2. Структура інтерфейсу вебсистеми аукціонів авто

Інтерфейс вебсистеми аукціонів авто розроблено з урахуванням сучасних принципів побудови вебінтерфейсів, зокрема простоти, інтуїтивності, адаптивності та послідовності взаємодії користувача із системою [4, 5, 7]. Основна задача інтерфейсу полягає в тому, щоб забезпечити швидкий доступ до перегляду аукціонів, детальної інформації про автомобіль, подання ставок,

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

перегляду результатів торгів, завантаження інвойсу та післяаукціонної комунікації між учасниками угоди. Структура інтерфейсу побудована відповідно до ролей користувачів і основних сценаріїв роботи з програмним продуктом. Усі сторінки системи мають підтримувати коректне відображення на різних пристроях, що забезпечується використанням адаптивної верстки та можливостей Tailwind CSS [7].

Головна сторінка є початковою точкою взаємодії користувача із системою. На ній відображається список доступних аукціонів у вигляді карток автомобілів. Кожна картка містить основну інформацію про транспортний засіб, зокрема марку, модель, рік випуску, фотографію, поточну ставку, статус аукціону та час до завершення торгів. Такий формат подання даних дає змогу користувачу швидко оцінити доступні пропозиції та перейти до детального перегляду конкретного лота. Незареєстрований користувач може переглядати аукціони та відкривати сторінку окремого автомобіля, однак для подання ставки йому необхідно пройти авторизацію або створити обліковий запис. Це дозволяє відокремити відкриту частину системи від функцій, які потребують ідентифікації користувача.

Як показано на рисунку 3.2., головна сторінка виконує роль навігаційного центру системи. Вона дозволяє користувачу переглянути перелік актуальних аукціонів і швидко перейти до сторінки зацікавленого автомобіля. Візуальне подання лотів у вигляді карток робить інтерфейс зрозумілим і зручним для використання, оскільки основна інформація про автомобіль подається одразу без необхідності відкривати кожен аукціон окремо.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

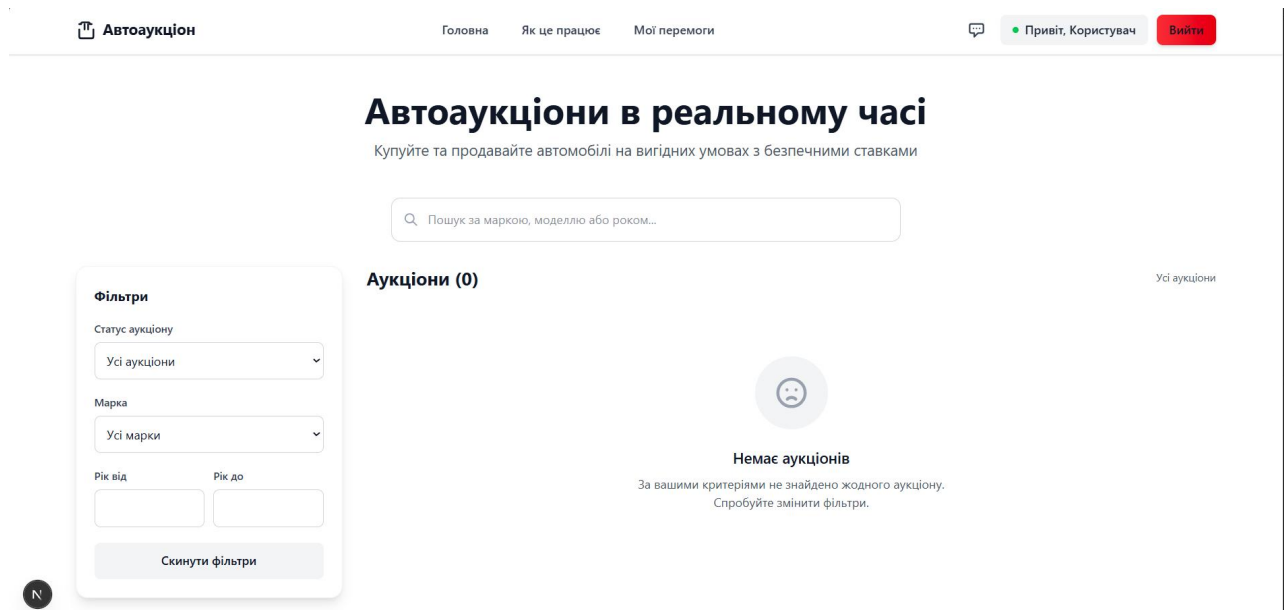


Рис. 3.2. Головна сторінка зі списком аукціонів

Сторінка конкретного аукціону призначена для детального перегляду інформації про транспортний засіб і участі в торгах. На цій сторінці користувач бачить фотографії автомобіля, його технічні характеристики, VIN-код, опис пошкоджень, поточну найвищу ставку, історію ставок і час до завершення аукціону. Для авторизованого покупця доступна форма подання ставки. Після введення суми система перевіряє коректність даних і передає запит на серверну частину через HTTP-запит до backend API, що відповідає загальній клієнт-серверній логіці роботи застосунку [8, 24]. Якщо ставка відповідає умовам аукціону, вона зберігається в базі даних і відображається в історії ставок. Після завершення аукціону сторінка змінює свій стан: користувач бачить, що торги завершені, а переможець отримує доступ до подальших дій, пов'язаних із замовленням та інвойсом.

На рисунку 3.3. представлено сторінку конкретного аукціону, яка поєднує інформаційний і функціональний блоки. Інформаційна частина дає змогу користувачу ознайомитися з характеристиками автомобіля, а функціональна частина забезпечує можливість участі в торгах. Такий підхід дозволяє покупцю приймати рішення на основі повної інформації про лот, не переходячи між кількома різними сторінками.

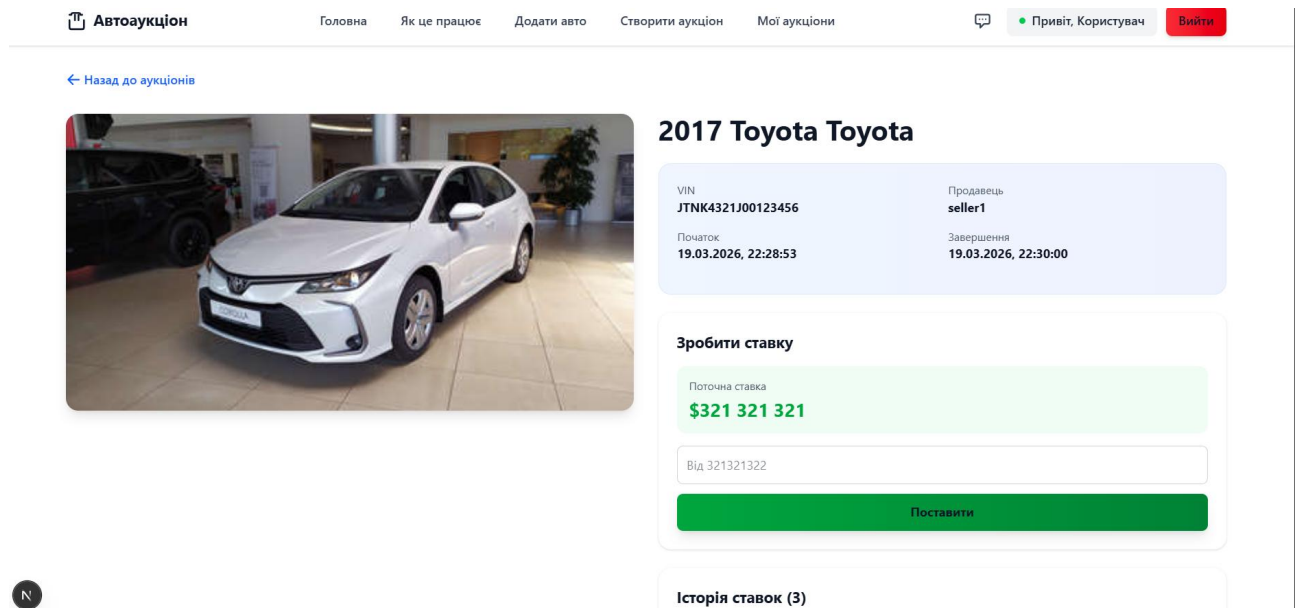


Рис. 3.3. Сторінка конкретного аукціону

Особистий кабінет продавця призначений для управління власними аукціонами. У цьому розділі продавець може переглядати створені ним лоти, бачити їхній поточний статус і контролювати результати торгів. Активні аукціони відображаються окремо від завершених, що спрощує навігацію та дозволяє швидко визначити, які лоти ще перебувають у процесі торгів, а які вже завершені. Для завершених аукціонів система відображає інформацію про переможця, що дає продавцю змогу перейти до післяаукціонної взаємодії з покупцем. Наявність кнопки переходу до чату дозволяє швидко розпочати спілкування щодо оплати, передачі автомобіля або інших деталей угоди.

Як показано на рисунку 3.4., особистий кабінет продавця виконує роль робочого простору для користувача, який створює аукціони. Він дозволяє не лише переглядати власні лоти, а й контролювати їхній стан після завершення торгів. Це важливо для практичного використання системи, оскільки продавець повинен мати швидкий доступ до інформації про переможця та можливість зв'язатися з ним без використання зовнішніх каналів комунікації.

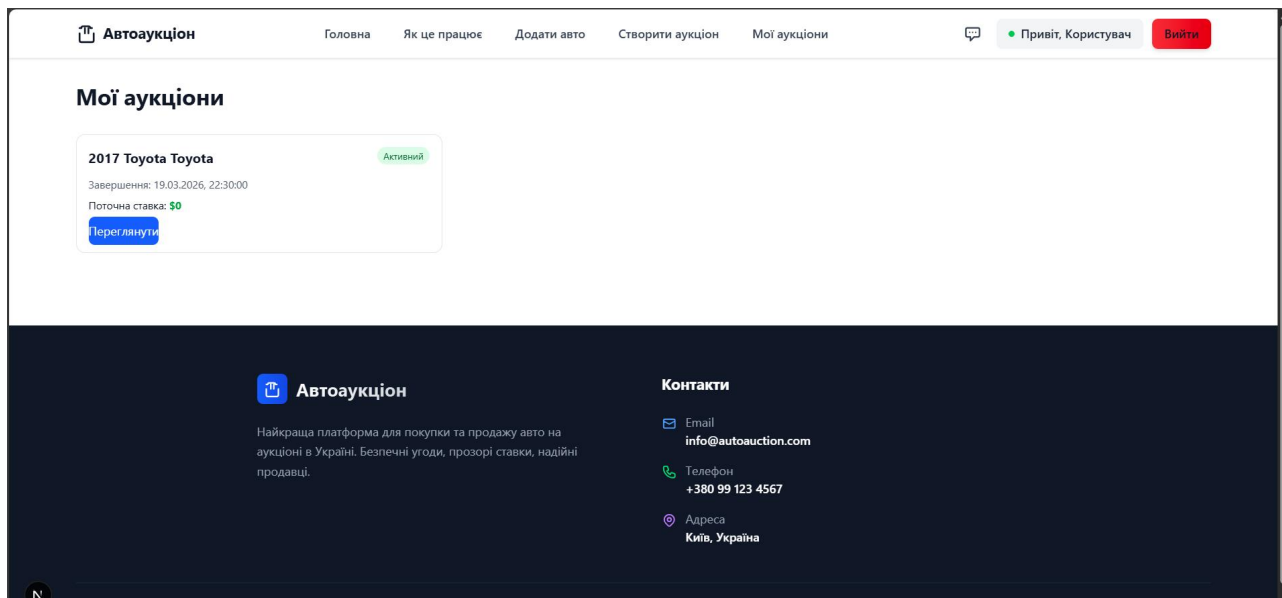


Рис. 3.4. Особистий кабінет продавця

Особистий кабінет покупця реалізовано через розділ «Мої перемоги». У цьому розділі відображаються аукціони, у яких користувач став переможцем. Для кожного виграного аукціону система показує дані про автомобіль, суму переможної ставки, статус замовлення та доступні дії. Покупець може переглянути інформацію про виграну угоду, завантажити інвойс у текстовому форматі, змінити статус оплати або перейти до чату з продавцем. Така організація інтерфейсу дає змогу покупцю швидко знайти потрібну інформацію після завершення торгів і виконати необхідні дії в межах одного розділу.

Окремим важливим елементом інтерфейсу є чат між покупцем і продавцем. Він використовується після завершення аукціону для уточнення деталей угоди, оплати, передачі автомобіля або інших організаційних питань. Повідомлення прив'язуються до конкретного аукціону, тому історія спілкування зберігається в межах окремої угоди. Це зручно для користувачів, оскільки вони можуть бачити контекст спілкування та не змішувати повідомлення з різних аукціонів. Обмін повідомленнями реалізується з використанням Socket.IO, що забезпечує двосторонню комунікацію між клієнтською і серверною частинами в режимі реального часу [13, 23, 25]. Непрочитані повідомлення відображаються через індикатор у навігаційній

панелі, що дозволяє користувачу швидко помітити нову активність і перейти до відповідного чату.

Інтерфейс системи також містить форми реєстрації та авторизації, які забезпечують доступ до персоналізованих функцій. Під час реєстрації користувач вводить ім'я, електронну пошту, пароль і обирає роль у системі. Вибір ролі впливає на доступний функціонал: покупець отримує можливість робити ставки та переглядати власні перемоги, а продавець може додавати автомобілі й створювати аукціони. Під час авторизації користувач вводить електронну пошту та пароль, після чого система перевіряє дані й надає доступ до захищених розділів.

Загальна структура інтерфейсу побудована таким чином, щоб основні дії були логічно пов'язані між собою. Користувач спочатку переглядає список аукціонів, потім відкриває сторінку конкретного лота, після авторизації бере участь у торгах, а після завершення аукціону переходить до особистого кабінету, інвойсу або чату. Така послідовність відповідає реальному процесу онлайн-аукціону та робить систему зрозумілою навіть для користувачів, які раніше не працювали з подібними платформами.

Адаптивність інтерфейсу забезпечує можливість використання системи на різних пристроях. Сторінки мають коректно відображатися як на персональних комп'ютерах, так і на мобільних телефонах. Це важливо для вебсистеми аукціонів, оскільки користувач може переглядати лоти, робити ставки або відповідати на повідомлення не лише з робочого комп'ютера, а й зі смартфона. У результаті інтерфейс забезпечує повний цикл взаємодії користувача із системою: від перегляду аукціону до завершення угоди та комунікації між учасниками.

3.3. Тестування роботи програмного комплексу

Тестування програмного комплексу проводилося з метою перевірки відповідності реалізованої вебсистеми функціональним і нефункціональним

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

вимогам, визначеним на етапі аналізу та проєктування. Основна увага приділялася перевірці тих сценаріїв, які є ключовими для роботи онлайн-аукціону авто: реєстрації та авторизації користувачів, створенню аукціону, поданню ставки, автоматичному завершенню торгів, генерації інвойсу та надсиланню повідомлень між покупцем і продавцем. Тестування виконувалося з позиції кінцевого користувача за методом чорного ящика, тобто перевірялася поведінка системи під час виконання типових дій в інтерфейсі без прямого аналізу внутрішньої реалізації програмного коду [16].

Першим було перевірено сценарій реєстрації нового користувача. Користувач відкривав форму реєстрації, вводив ім'я, електронну пошту, пароль і обирав роль у системі. Очікуваним результатом було створення нового облікового запису, збереження даних користувача в базі даних і надання доступу до функцій відповідно до обраної ролі. Під час тестування підтверджено, що система коректно обробляє введені дані, не допускає реєстрацію з некоректною електронною поштою та після успішного створення облікового запису дозволяє користувачу перейти до подальшої роботи із системою.

Як показано на рисунку 3.5., форма реєстрації містить основні поля, необхідні для створення облікового запису. Вона є важливою частиною системи, оскільки саме після реєстрації користувач отримує доступ до ролей покупця або продавця. У межах тестування було підтверджено, що після успішного введення даних користувач може продовжити роботу з системою без повторного ручного налаштування облікового запису.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						65
Змн.	Арк.	№ докум.	Підпис	Дата		

Автоаукціон

ГоловнаЯк це працює

УвійтиРеєстрація

Реєстрація

Ваша роль:

☐ Покупець
☒ Продавець

Зареєструватися

Вже є акаунт? Увійдіть

Автоаукціон

Контакти

Рис. 3.5. Форма реєстрації та успішне повідомлення

Наступним етапом було перевірено авторизацію користувача. Для цього використовувалися як коректні, так і некоректні облікові дані. У разі правильного введення електронної пошти та пароля система повинна була надати доступ до захищених розділів. У разі введення неправильного пароля або неіснуючої електронної пошти доступ мав бути заблокований. Результати перевірки показали, що система коректно обробляє обидва сценарії: авторизований користувач отримує доступ до особистого кабінету, а неавторизований не може перейти до захищених функцій.

Окремо було перевірено сценарій створення аукціону продавцем. Продавець спочатку додавав автомобіль із зазначенням основних характеристик, після чого створював аукціон і визначав час завершення торгів. Очікуваним результатом було збереження даних про автомобіль та аукціон у базі даних, а також поява нового лота у списку доступних аукціонів. Під час тестування підтверджено, що система коректно прив'язує створений аукціон до продавця та відповідного автомобіля, а новий лот відображається на головній сторінці.

На рисунку 3.6. показано форму створення аукціону, яка використовується продавцем для запуску торгів. У процесі тестування було

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						66
Змн.	Арк.	№ докум.	Підпис	Дата		

перевірено, що введені дані передаються на серверну частину, зберігаються в базі даних і після цього використовуються для відображення аукціону в інтерфейсі користувача. Це підтверджує коректність взаємодії між клієнтською частиною, серверною логікою та базою даних.

Рис. 3.6. Форма створення аукціону

Сценарій подання ставки перевірявся з боку авторизованого покупця. Користувач відкривав сторінку активного аукціону, вводив суму ставки та надсилав її через форму. Очікуваним результатом було збереження ставки в базі даних і оновлення інформації про поточну ціну аукціону. Під час тестування було підтверджено, що система обробляє коректні значення ставки, не дозволяє подавати ставки для завершених аукціонів і оновлює інформацію на сторінці після успішної операції. Це є важливим для забезпечення прозорості торгів і коректного визначення переможця.

Важливим етапом тестування була перевірка автоматичного завершення аукціону. Для цього створювався аукціон із коротким часом завершення, після чого система мала самостійно перевірити його статус, визначити найвищу ставку, закрити торги та створити замовлення для переможця. Очікуваним результатом було оновлення статусу аукціону та поява відповідного запису в розділі «Мої перемоги» у покупця. Під час тестування підтверджено, що

механізм автоматичного завершення працює без ручного втручання, а переможець визначається на основі найбільшої ставки.

Також було перевірено роботу механізму генерації інвойсу. Після завершення аукціону покупець, який став переможцем, переходив до розділу «Мої перемоги» та виконував завантаження інвойсу. Очікуваним результатом було отримання текстового файлу з даними про замовлення, автомобіль, покупця та суму до сплати. Тестування підтвердило, що файл успішно завантажується у браузері, має формат TXT, підтримує кодування UTF-8 і коректно відображає український текст. Завдяки цьому інвойс може використовуватися як підтвердження результату завершення аукціону.

Сценарій надсилання повідомлення перевірявся після завершення аукціону. Покупець і продавець переходили до чату, пов'язаного з конкретним аукціоном, після чого один із користувачів надсилав повідомлення. Очікуваним результатом було збереження повідомлення в базі даних і його відображення в історії листування. У процесі тестування підтверджено, що повідомлення зберігаються, відображаються у правильній хронологічній послідовності та залишаються доступними після повторного відкриття сторінки чату.

Перевірка безпеки охоплювала доступ до захищених сторінок без авторизації, спроби виконання запитів без JWT-токена та введення некоректних даних у форми. Система не дозволяє неавторизованому користувачу отримати доступ до особистих кабінетів, створення аукціонів, перегляду власних перемог і завантаження інвойсів. Запити до бази даних виконуються через TypeORM із використанням параметризованих запитів, що зменшує ризик SQL-ін'єкцій [11, 28]. Клієнтська частина реалізована на основі Next.js, який екранує вивід і знижує ризики некоректної інтерпретації користувацького вмісту в інтерфейсі [5, 29]. Доступ до захищених ресурсів контролюється JWT-автентифікацією, тому запити без коректного токена не отримують доступу до приватних даних користувача [19, 20, 30].

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

Перевірка продуктивності виконувалася шляхом багаторазового відкриття основних сторінок і виконання типових дій користувача. Було перевірено завантаження головної сторінки, сторінки конкретного аукціону, особистих кабінетів, чату та інвойсу. У межах тестового середовища система забезпечила прийнятний час відгуку, а основні операції виконувалися без критичних затримок. Для узгодженості з нефункціональними вимогами в роботі зазначено, що час відгуку системи не перевищує 2 секунд у межах перевірених сценаріїв [12, 17].

Проведене тестування підтвердило коректність реалізації основних функціональних можливостей вебсистеми аукціонів авто. Реєстрація, авторизація, створення аукціону, подання ставки, автоматичне завершення торгів, генерація інвойсу та робота чату виконуються стабільно. Отримані результати свідчать, що програмний комплекс відповідає визначеним вимогам, може бути використаний для демонстрації роботи системи та є готовим до подальшого практичного вдосконалення.

Висновки до третього розділу

У третьому розділі було розглянуто порядок встановлення, налаштування та запуску вебсистеми аукціонів авто. Описано процес розгортання програмного комплексу за допомогою Docker і Docker Compose, що дозволяє запускати frontend-частину, backend API та базу даних в ізольованому середовищі. Такий підхід спрощує підготовку системи до роботи, зменшує залежність від локального середовища користувача та забезпечує стабільний запуск усіх компонентів програмного комплексу. Завдяки контейнеризації систему можна швидко розгорнути на різних операційних системах і повторно запускати без складного ручного налаштування.

Також у розділі було описано структуру інтерфейсу вебсистеми та основні сценарії взаємодії користувача із програмним продуктом. Розглянуто головну сторінку зі списком аукціонів, сторінку конкретного аукціону,

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

особистий кабінет продавця, особистий кабінет покупця та чат між учасниками угоди. Інтерфейс забезпечує доступ до функцій відповідно до ролі користувача: продавець може створювати аукціони, переглядати власні активні та завершені лоти, а покупець може робити ставки, переглядати перемоги, завантажувати інвойс і спілкуватися з продавцем. Описані елементи інтерфейсу підтверджують, що система охоплює повний цикл взаємодії учасників онлайн-аукціону.

Проведене тестування підтвердило коректну роботу основних функцій системи, зокрема реєстрації, авторизації, створення аукціону, подання ставки, автоматичного завершення торгів, генерації інвойсу та надсилання повідомлень у чаті. Окремо було перевірено роботу основних механізмів безпеки, серед яких JWT-автентифікація, валідація даних і захист запитів до бази даних. Отримані результати свідчать, що програмний комплекс відповідає визначеним функціональним і нефункціональним вимогам, працює стабільно та може бути використаний для практичної організації онлайн-аукціонів авто.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було успішно реалізовано повноцінну вебсистему аукціонів авто, що повністю відповідає поставленій меті автоматизувати процес купівлі-продажу транспортних засобів через інтернет з підтримкою приватних користувачів. Розроблений програмний продукт охоплює весь життєвий цикл аукціону: від створення та подання ставок до автоматичного завершення, визначення переможця, двостороннього спілкування та генерації інвойсу.

Були виконані всі завдання, визначені у вступі. Проведено аналіз предметної області та трьох існуючих аналогів – Copart, AutoBidMaster та eBay Motors, на основі чого обґрунтовано необхідність створення доступного рішення для приватних осіб. Розглянуто та порівняно монолітну, мікросервісну та клієнт-серверну архітектури, за результатами чого обрано клієнт-серверну модель з використанням сучасного технологічного стеку: Next.js для фронтенду, NestJS для бекенду, PostgreSQL як системи управління базою даних та Socket.IO для комунікації в реальному часі. Реалізовано ключові функціональні модулі: реєстрація та автентифікація користувачів з ролями «покупець» та «продавець», створення аукціонів із повним описом автомобіля, подання ставок у реальному часі, автоматичне завершення аукціонів за допомогою планувальника завдань, двосторонній чат між учасниками, особисті кабінети для перегляду активних та завершених аукціонів, а також генерація інвойсів у текстовому форматі з підтримкою української локалізації. Забезпечено високий рівень безпеки шляхом хешування паролів за допомогою bcrypt, використання JWT-токенів для автентифікації, захисту від SQL-ін'єкцій через параметризовані запити TypeORM та валідації всіх вхідних даних за допомогою бібліотеки class-validator. Проведено повне тестування системи, яке підтвердило коректність роботи всіх механізмів, стабільність функціонування та відповідність нефункціональним вимогам, зокрема час відгуку не перевищує 2 секунд та повна адаптивність інтерфейсу для мобільних пристроїв.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						71
Змн.	Арк.	№ докум.	Підпис	Дата		

Результати виконання роботи свідчать про відповідність розробленої системи поставленим меті та основним завданням. Спроектовано об'єктну модель та структуру реляційної бази даних PostgreSQL із шістьма основними сутностями, формалізовано ключові алгоритми роботи за допомогою UML-діаграм послідовності. Реалізовано функціональні модулі, необхідні для проведення онлайн-аукціону авто: реєстрацію та автентифікацію користувачів, створення аукціонів, подання ставок, автоматичне завершення торгів, визначення переможця, обмін повідомленнями між учасниками та генерацію інвойсів. Програмний комплекс розгортається через Docker і Docker Compose, що забезпечує відтворюваність середовища незалежно від конфігурації локального комп'ютера. Інтерфейс системи є зрозумілим для користувача, а програмний комплекс може бути використаний для практичної організації онлайн-аукціонів авто.

Окремо варто зазначити, що розроблена система має чітку модульну структуру, що спрощує її подальшу підтримку та розвиток. Поділ програмного комплексу на frontend-частину, backend API, базу даних і окремі функціональні модулі дозволяє незалежно вдосконалювати інтерфейс користувача, бізнес-логіку та механізми збереження даних. Використання Docker і Docker Compose полегшує розгортання системи в різних середовищах, а застосування сучасного технологічного стеку забезпечує достатній рівень надійності, гнучкості та масштабованості. Це дає підстави вважати розроблений програмний продукт не лише завершеним у межах кваліфікаційної роботи, а й придатним для подальшого практичного вдосконалення.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						72
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Copart. Online Car Auctions. URL: <https://www.copart.com/> (дата звернення: 08.06.2026).
2. AutoBidMaster. Online Car Auction. URL: <https://www.autobidmaster.com/en/> (дата звернення: 08.06.2026).
3. eBay Motors. Auto Parts and Vehicles. URL: https://www.ebay.com/b/Auto-Parts-and-Vehicles/6000/bn_1865334 (дата звернення: 08.06.2026).
4. React Documentation. URL: <https://react.dev/learn> (дата звернення: 08.06.2026).
5. Next.js Documentation. URL: <https://nextjs.org/docs> (дата звернення: 08.06.2026).
6. TypeScript Documentation. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 08.06.2026).
7. Tailwind CSS Documentation. URL: <https://tailwindcss.com/docs> (дата звернення: 08.06.2026).
8. Axios Documentation. URL: <https://axios-http.com/docs/intro> (дата звернення: 08.06.2026).
9. Node.js Documentation. URL: <https://nodejs.org/api/> (дата звернення: 08.06.2026).
10. NestJS Documentation. URL: <https://docs.nestjs.com/> (дата звернення: 08.06.2026).
11. TypeORM Documentation. URL: <https://typeorm.io/docs/getting-started/> (дата звернення: 08.06.2026).
12. PostgreSQL 15 Documentation. URL: <https://www.postgresql.org/docs/15/> (дата звернення: 08.06.2026).
13. Socket.IO Documentation. URL: <https://socket.io/docs/v4/> (дата звернення: 08.06.2026).
14. Docker Documentation. URL: <https://docs.docker.com/> (дата звернення: 08.06.2026).

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

- 15.Docker Compose Documentation. URL: <https://docs.docker.com/compose/> (дата звернення: 08.06.2026).
- 16.Karlsson S., Jongeling R., Čaušević A., Sundmark D. Exploring behaviours of RESTful APIs in an industrial setting. Software Quality Journal. 2024. Vol. 32. P. 1287–1324. DOI: 10.1007/s11219-024-09686-0. URL: <https://link.springer.com/article/10.1007/s11219-024-09686-0> (дата звернення: 08.06.2026).
- 17.Díaz-Gómez J. M., Quiceno-Rua E., Correa-Álvarez C. D. Protocols, Reactive Architectures, and Computing Platforms for Low-Latency, High-Concurrency Web Applications: A Systematic Literature Review. Future Internet. 2026. Vol. 18, No. 5. Article 254. DOI: 10.3390/fi18050254. URL: <https://www.mdpi.com/1999-5903/18/5/254> (дата звернення: 08.06.2026).
- 18.Visual Paradigm. What is Use Case Diagram? URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/> (дата звернення: 08.06.2026).
- 19.NestJS Authentication. URL: <https://docs.nestjs.com/security/authentication> (дата звернення: 08.06.2026).
- 20.NestJS Authorization. URL: <https://docs.nestjs.com/security/authorization> (дата звернення: 08.06.2026).
- 21.NestJS Validation. URL: <https://docs.nestjs.com/techniques/validation> (дата звернення: 08.06.2026).
- 22.NestJS Task Scheduling. URL: <https://docs.nestjs.com/techniques/task-scheduling> (дата звернення: 08.06.2026).
- 23.NestJS WebSocket Gateways. URL: <https://docs.nestjs.com/websockets/gateways> (дата звернення: 08.06.2026).
- 24.MDN Web Docs. HTTP. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP> (дата звернення: 08.06.2026).
- 25.MDN Web Docs. WebSocket API. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 08.06.2026).

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

- 26.OWASP. Authentication Cheat Sheet. URL:
https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html (дата звернення: 08.06.2026).
- 27.OWASP. Password Storage Cheat Sheet. URL:
https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 08.06.2026).
- 28.OWASP. SQL Injection Prevention Cheat Sheet. URL:
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html (дата звернення: 08.06.2026).
- 29.OWASP. Cross Site Scripting Prevention Cheat Sheet. URL:
https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html (дата звернення: 08.06.2026).
- 30.OWASP. JSON Web Token for Java Cheat Sheet. URL:
https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html (дата звернення: 08.06.2026).

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		

Повний лістинг коду AuthService.ts

/**

* Виконує автентифікацію користувача за email та паролем.

* @param email Електронна пошта користувача.

* @param password Пароль користувача у відкритому вигляді.

* @returns Об'єкт з даними автентифікованого користувача та JWT-токеном.

* @throws UnauthorizedException Якщо email не знайдено або пароль неправильний.

*/

async login(email: string, password: string) {

const user = await this.userService.findByEmail(email);

if (!user || !(await bcrypt.compare(password, user.password))) {

throw new UnauthorizedException('Invalid credentials');

}

const payload = {

email: user.email,

sub: user.id,

role: user.role,

};

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

```

return {
    user,
    token: this.jwtService.sign(payload),
};
}

```

Додаток Б

Фрагмент лістингу коду OrdersController.ts із захистом маршруту

```

@UseGuards(JwtAuthGuard)
@Get('my-wins')
findMyWins(@Req() req) {
    return this.ordersService.findMyWins(req.user.userId);
}

```

Додаток В

Повний лістинг коду AuctionScheduler.ts

```

@Injectable()
export class AuctionScheduler {
    private readonly logger = new Logger(AuctionScheduler.name);
}

```

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

```
constructor(private auctionsService: AuctionsService) {}
```

```
/**
```

```
 * Завдання, що виконується щохвилини.
```

```
 * Перевіряє наявність аукціонів, час яких уже завершився,
```

```
 * та автоматично їх закриває.
```

```
 */
```

```
@Cron(CronExpression.EVERY_MINUTE)
```

```
async handleExpiredAuctions() {
```

```
    this.logger.debug('Перевірка завершених аукціонів...');
```

```
    try {
```

```
        const closedCount = await this.auctionsService.closeExpiredAuctions();
```

```
        if (closedCount > 0) {
```

```
            this.logger.log(`Автоматично завершено ${closedCount} аукціон(ів)`);
```

```
        }
```

```
    } catch (error) {
```

```
        this.logger.error('Помилка при завершенні аукціонів:', error);
```

```
    }
```

```
}
```

```
}
```

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						79
Змн.	Арк.	№ докум.	Підпис	Дата		

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						80
Змн.	Арк.	№ докум.	Підпис	Дата		

Фрагмент лістингу коду AuctionsService.ts із реалізацією автоматичного завершення аукціонів

```

async closeExpiredAuctions(): Promise<number> {

  const expiredAuctions = await this.auctionRepository.find({

    where: {

      end_time: LessThan(new Date()),

      isClosed: false,

    },

    relations: ['bids', 'bids.bidder'],

  });

  let count = 0;

  for (const auction of expiredAuctions) {

    if (auction.bids.length === 0) {

      continue;

    }

    const winnerBid = auction.bids.reduce((max, bid) =>

      bid.amount > max.amount ? bid : max

    );

```

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						81
Змн.	Арк.	№ докум.	Підпис	Дата		

```

await this.orderRepository.save({
  auction: { id: auction.id } as Auction,
  buyer: { id: winnerBid.bidder.id } as User,
  amount: winnerBid.amount,
  status: 'pending',
});

```

```

auction.isClosed = true;

await this.auctionRepository.save(auction);

```

```

count++;
}

```

```

return count;
}

```

Додаток Д

Повний лістинг коду MessagesGateway.ts

```

@WebSocketGateway({ cors: true })

export class MessagesGateway {

  @WebSocketServer()

```

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						82
Змн.	Арк.	№ докум.	Підпис	Дата		

```
server: Server;
```

```
constructor(private messagesService: MessagesService) {}
```

```
/**
```

```
 * Обробляє вхідні повідомлення від клієнта.
```

```
 */
```

```
@SubscribeMessage('sendMessage')
```

```
async handleMessage(
```

```
  @MessageBody()
```

```
  data: {
```

```
    senderId: string;
```

```
    recipientId: string;
```

```
    auctionId: string;
```

```
    content: string;
```

```
  },
```

```
) {
```

```
  const message = await this.messagesService.sendMessage(
```

```
    data.senderId,
```

```
    data.recipientId,
```

```
    data.auctionId,
```

```
    data.content,
```

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						83
Змн.	Арк.	№ докум.	Підпис	Дата		

```

);

this.server.to(data.auctionId).emit('newMessage', message);
}

/**
 * Додає клієнта до кімнати чату.
 */
@SubscribeMessage('joinChat')
handleJoinChat(
  @ConnectedSocket() client: Socket,
  @MessageBody() auctionId: string,
) {
  client.join(auctionId);
}
}

```

Додаток Е

Фрагмент лістингу коду MessagesService.ts із завантаженням історії повідомлень

```

/**
 * Повертає історію повідомлень для конкретного аукціону

```

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						84
Змн.	Арк.	№ докум.	Підпис	Дата		

* та позначає непрочитані повідомлення як прочитані.

*/

```
async getMessagesForAuction(auctionId: string, userId: string) {
```

```
  const messages = await this.messageRepository.find({
```

```
    where: [
```

```
      {
```

```
        auctionId,
```

```
        sender: { id: userId },
```

```
      },
```

```
      {
```

```
        auctionId,
```

```
        recipient: { id: userId },
```

```
      },
```

```
    ],
```

```
    order: {
```

```
      createdAt: 'ASC',
```

```
    },
```

```
  });
```

```
  const unreadMessages = messages.filter(
```

```
    (msg) => !msg.isRead && msg.recipient.id === userId,
```

```
  );
```

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						85
Змн.	Арк.	№ докум.	Підпис	Дата		

```

if (unreadMessages.length > 0) {
  await this.messageRepository.update(
    {
      id: In(unreadMessages.map((message) => message.id)),
    },
    {
      isRead: true,
    },
  );
}

return messages;
}

```

Додаток Ж

Фрагмент лістингу коду OrdersService.ts із реалізацією генерації інвойсу

/**

* Генерує текстовий інвойс для замовлення.

* Дані для інвойсу отримуються з таблиці orders

* разом із пов'язаними даними про аукціон, автомобіль і покупця.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						86
Змн.	Арк.	№ докум.	Підпис	Дата		

*/

```
async generateInvoice(orderId: string): Promise<Buffer> {  
  const order = await this.orderRepository.findOne({  
    where: {  
      id: orderId,  
    },  
    relations: ['auction', 'auction.vehicle', 'buyer'],  
  });  
  
  if (!order) {  
    throw new BadRequestException('Замовлення не знайдено');  
  }  
  
  const invoiceContent = `  
ІНВОЙС  
Автоаукціон  
  
Номер замовлення: ${order.id}  
Дата: ${new Date().toLocaleDateString('uk-UA')}  
Статус: ${this.getStatusLabel(order.status)}  
Покупець: ${order.buyer?.name || '-'}`
```

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						87
Змн.	Арк.	№ докум.	Підпис	Дата		

```

Автомобіль: ${order.auction.vehicle.year} ${order.auction.vehicle.make}
${order.auction.vehicle.model}

VIN: ${order.auction.vehicle.vin || '-'}

Сума до сплати: $$ {order.amount.toLocaleString()}

.trim();

return Buffer.from(invoiceContent, 'utf-8');
}

```

Додаток 3

Фрагмент лістингу коду OrdersController.ts із реалізацією завантаження інвойсу

```

/**
 * Формує відповідь для завантаження інвойсу.
 * Файл повертається користувачу у форматі TXT
 * з підтримкою кодування UTF-8.
 */

@Get('invoice/:orderId')
@UseGuards(JwtAuthGuard)
async generateInvoice(
  @Param('orderId') orderId: string,
  @Res() res: Response,
) {

```

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						88
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const buffer = await this.ordersService.generateInvoice(orderId);

res.set({
  'Content-Type': 'text/plain; charset=utf-8',
  'Content-Disposition': `attachment; filename="invoice_${orderId}.txt"`,
  'Content-Length': buffer.length.toString(),
});

res.end(buffer);
}

```

Додаток И

Фрагмент лістингу коду BidsService.ts із реалізацією подання ставки

```

/**
 * Створює нову ставку для активного аукціону.
 * Перед збереженням перевіряється існування аукціону,
 * його активний статус і коректність суми ставки.
 */
async createBid(auctionId: string, bidderId: string, amount: number) {
  const auction = await this.auctionRepository.findOne({
    where: {

```

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						89
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        id: auctionId,

        isClosed: false,

    },

});

if (!auction) {

    throw new BadRequestException('Аукціон не знайдено або завершено');

}

const highestBid = await this.bidRepository.findOne({

    where: {

        auction: { id: auctionId },

    },

    order: {

        amount: 'DESC',

    },

});

if (highestBid && amount <= highestBid.amount) {

    throw new BadRequestException(

        'Ставка має бути вищою за поточну максимальну',

    );

}

```

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						90
Змн.	Арк.	№ докум.	Підпис	Дата		

```
}
```

```
const bid = this.bidRepository.create({  
  amount,  
  auction: { id: auctionId } as Auction,  
  bidder: { id: bidderId } as User,  
});
```

```
return this.bidRepository.save(bid);  
}
```

Додаток К

Повний лістинг конфігураційного файлу docker-compose.yml

```
version: '3.8'
```

```
services:
```

```
db:
```

```
  image: postgres:15
```

```
  environment:
```

```
    POSTGRES_DB: car_auction
```

```
    POSTGRES_USER: postgres
```

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						91
Змн.	Арк.	№ докум.	Підпис	Дата		

POSTGRES_PASSWORD: postgres

ports:

- "5432:5432"

volumes:

- postgres_data:/var/lib/postgresql/data

backend:

build: ./backend

ports:

- "3001:3001"

environment:

- DB_HOST=db

- DB_PORT=5432

- DB_USERNAME=postgres

- DB_PASSWORD=postgres

- DB_NAME=car_auction

- JWT_SECRET=your_strong_secret_key_here

depends_on:

- db

frontend:

build: ./frontend

					ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
						92
Змн.	Арк.	№ докум.	Підпис	Дата		

ports:

- "3000:3000"

environment:

- NEXT_PUBLIC_API_URL=http://localhost:3001

depends_on:

- backend

volumes:

postgres_data:

Додаток Л

Приклад структури конфігураційного файлу .env

Database

DB_HOST=db

DB_PORT=5432

DB_USERNAME=postgres

DB_PASSWORD=postgres

DB_NAME=car_auction

JWT

JWT_SECRET=your_strong_secret_key_here

					ІІЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						93
Змн.	Арк.	№ докум.	Підпис	Дата		

Frontend

NEXT_PUBLIC_API_URL=http://localhost:3001

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						94
Змн.	Арк.	№ докум.	Підпис	Дата		