Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Центр післядипломної освіти
(повна назва)

Кафедра програмної інженерії
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система обліку замовлень з 3D-друку

(тема)

Виконав:
студент 4 курсу, групи ПЗПІп-22-1

Пивовар Є. В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник к.т.н., доц. кафедри ПІ Русакова Н. Є.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Центр післядипломної освіти
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

**ЗАВДАННЯ**
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Пивовару Євгену Валентиновичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система обліку замовлень з 3D-друку
Затверджена наказом по університету від 17.06. 2024р. № 588 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 19.09.2024
3. Вихідні дані до роботи Розробити веб-додаток для власників 3D-принтерів, який повинен давати можливість користувачеві створювати, переглядати, редагувати та видаляти замовлення та дані про клієнтів; додаток повинен бути написаний за допомогою мови програмування JavaScript, платформи NodeJS та бібліотеки React з використанням бази даних MySQL.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, інструкція користувача, тестування розробленого програмного забезпечення, висновки, додатки.

# КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної галузі | 20.05.2024 | *виконано* |
| 2 | Створення специфікації ПЗ | 25.05.2024 | *виконано* |
| 3 | Проектування ПЗ | 05.06.2024 | *виконано* |
| 4 | Розробка ПЗ | 04.07.2024 | *виконано* |
| 5 | Тестування ПЗ | 07.07.2024 | *виконано* |
| 6 | Оформлення пояснювальної записки | 10.07.2024 | *виконано* |
| 7 | Підготовка презентації та доповіді | 13.07.2024 | *виконано* |
| 8 | Попередній захист | 18.07.2024 | *виконано* |
| 9 | Нормоконтроль, рецензування | 19.07.2024 | *виконано* |
| 10 | Здача роботи у електронний архів | 19.07.2024 | *виконано* |
| 11 | Допуск до захисту у зав. кафедри | 20.07.2024 | *виконано* |

Дата видачі завдання 6 травня 2024р.

Студент (ка)_____ Пивовар Є. В.
(підпис)

Керівник роботи _____ доц. кафедри ПІ Русакова Н. Є.
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра складається з 5 розділів, містить 26 рисунки, 3 таблиці, 10 джерел та 3 додатка. Загальна кількість сторінок - 82.

БІБЛІОТЕКА REACTJS, ВЕБ-ДОДАТОК, МОВА ПРОГРАМУВАННЯ JAVASCRIPT, РЕЛЯЦІЙНА БАЗА ДАНИХ MYSQL.

Об’єктом розробки є веб-додаток «Print Orders».

Метаю роботи є розробка програмної системи обліку замовлень з 3D-друку у вигляді веб-додатку. Додаток має бути простим у використанні та не вимагати від користувача якихось особливих технічних знань.

Методи розробки базуються на застосуванні середовища WebStorm для розробки програмного забезпечення, MySQL Workbench для створення та управління базою даних, та використанні інших додаткових засобів.

Результатом роботи є веб-додаток «Print Orders», який дозволяє користувачеві створювати, переглядати, редагувати та видаляти замовлення, також система забезпечує можливість зберігати та обробляти інформацію про клієнтів, у тому числі їх контактні дані та історію замовлень.

The explanatory note to the bachelor's thesis consists of 5 chapters, contains 26 figures, 3 tables, 10 sources and 3 appendices. The total number of pages is 82.

LIBRARY: ReactJS, WEB APPLICATION, PROGRAMMING LANGUAGE: JavaScript, RELATIONAL DATABASE: MySQL.

The object of development is the "Print Orders" web application.

The purpose of the work is to develop a software system for accounting for 3D printing orders in the form of a web application. The application should be easy to use and not require any special technical knowledge from the user.

Development methods are based on the application of the WebStorm environment for software development, MySQL Workbench for database creation and management, and the use of other additional tools.

The result of the work is the Print Orders web application, which allows the user to create, view, edit and delete orders, and the system provides the ability to store and process customer information, including their contact details and order history.

Я, Пивовар Євген Валентинович, студент гр. ПЗПІп-22-1, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система обліку замовлень з 3D-друку», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

AJAX – Asynchronous JavaScript and XML

API – Application Programming Interface

CPU – Central Processing Unit

CRM – Customer Relationship Management

CSS – Cascading Style Sheets

DOM – Document Object Model

HTML – Hyper Text Markup Language

HTTP – Hypertext Transfer Protocol

JS – JavaScript

JWT – JSON Web Token

PDF – Portable Document Format

QA – Quality Assurance

SPA – Single-Page Applications

SQL – Structured Query Language

UML – Unified Modeling Language

UI – User Interface

БД – база даних

ПЗ – програмне забезпечення

ПС – програмна система

СУБД – система управління базами даних

# ВСТУП

За останні роки виробництво та використання 3D-принтерів значно зросло, привертаючи увагу широкого кола користувачів. Цей технологічний прорив став ключовим фактором у виробництві різноманітних виробів та прототипів з метою індивідуалізації та вдосконалення продукції. Завдяки можливостям 3D-друку, компанії та індивідуальні підприємці можуть виготовляти складні деталі та вироби, що раніше були недоступні або економічно невигідні для виробництва.

Власники 3D-принтерів, що займаються виготовленням різних виробів під замовлення, стикаються з необхідністю у впорядкуванні замовлень, управлінні клієнтською базою та контролі якості продукції. Відсутність належного інструментарію для управління цими процесами може призвести до хаосу, помилок у замовленнях, втрати клієнтів та зниження якості продукції.

Для вирішення цих проблем і покращення ефективності необхідний веб-додаток, який надає користувачам зручний інструмент для управління замовленнями, клієнтами та іншими важливими аспектами. Такий додаток має спростити процеси взаємодії з клієнтами, полегшити контроль за виконанням замовлень та забезпечити ефективне управління ресурсами. Веб-додаток також дозволить власникам 3D-принтерів легко відстежувати стан своїх принтерів, планувати обслуговування та забезпечувати безперебійну роботу обладнання.

Метою кваліфікаційної роботи є створення програмної системи, тобто веб - додатку, який має набір функцій, що включають створення, видалення та редагування замовлень, управління клієнтською базою даних та базою обладнання та виробів, а також адміністрування користувачів.

Створення такого продукту відповідає потребам ринку та сприятиме підвищенню конкурентоспроможності власників 3D-принтерів, а також забезпечить їм зручний інструмент для досягнення своїх цілей. Додаток буде розроблено з урахуванням простоти та інтуїтивності користувацького інтерфейсу, що забезпечить його зручність для користувачів різного рівня підготовки.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

## 1.1 Опис предметної області

Останніми роками виробництво та використання 3D-принтерів значно зросли, привертаючи увагу широкого кола користувачів. Цей технологічний прорив став ключовим фактором у виготовленні різноманітних виробів та прототипів, сприяючи індивідуалізації та вдосконаленню продукції. Технологія 3D-друку відкрила нові можливості для малого та середнього бізнесу, дозволяючи створювати унікальні та складні деталі за відносно короткий час і за прийнятну ціну.

Власники 3D-принтерів, які займаються виготовленням виробів на замовлення, стикаються з необхідністю впорядкування замовлень, управління клієнтською базою та контролю якості продукції. Основні проблеми, з якими вони зіштовхуються, включають складність у відстеженні численних замовлень, управлінні даними клієнтів та ефективному використанні обладнання.

3D-друкарі, які є власниками одного 3D-принтера, можливо декількох, або невеликого бізнесу, що спеціалізується на виготовленні виробів за допомогою 3d - принтерів, для яких кожне замовлення важливе, шукають зручний спосіб керувати ними, тримаючи під контролем дані клієнтів та принтерів. Їм потрібен додаток, який пропонує конкретний спектр функцій, що включає створення, видалення та редагування замовлень, управління клієнтською базою даних та базою обладнання та матеріалів, а також адміністрування користувачів. Додаток повинен максимально спростити ваш робочий процес, забезпечуючи ефективне та зручне управління всіма аспектами вашої діяльності. Це дозволить зменшити адміністративні витрати, підвищити продуктивність та покращити якість обслуговування клієнтів. Впровадження такого інструменту сприятиме підвищенню конкурентоспроможності вашого бізнесу на ринку 3D-друку, допомагаючи вам залишатися на крок попереду конкурентів та задовольняти потреби найвимогливіших клієнтів.

## 1.2 Аналіз існуючих аналогів

Наразі існує достатньо CRM-систем, які вирішують питання автоматизації ведення обліку замовлень та іншими питаннями, пов’язаними з взаємодією з клієнтами, отриманням статистичних даних для аналізу та прийняття рішень.

Creatio – це інтегрована платформа для автоматизації бізнес-процесів (див. рис. 1.1). Вона дозволяє компаніям керувати процесами взаємодії з клієнтами, автоматизувати різні бізнес-процеси та підвищувати ефективність роботи. Її основна мета - забезпечити комплексний підхід до управління продажами, маркетингом, обслуговуванням клієнтів і внутрішніми бізнес-процесами. Система Creatio була розроблена компанією Terrasoft і відома своєю гнучкістю та можливістю легкої адаптації під потреби різних бізнесів.

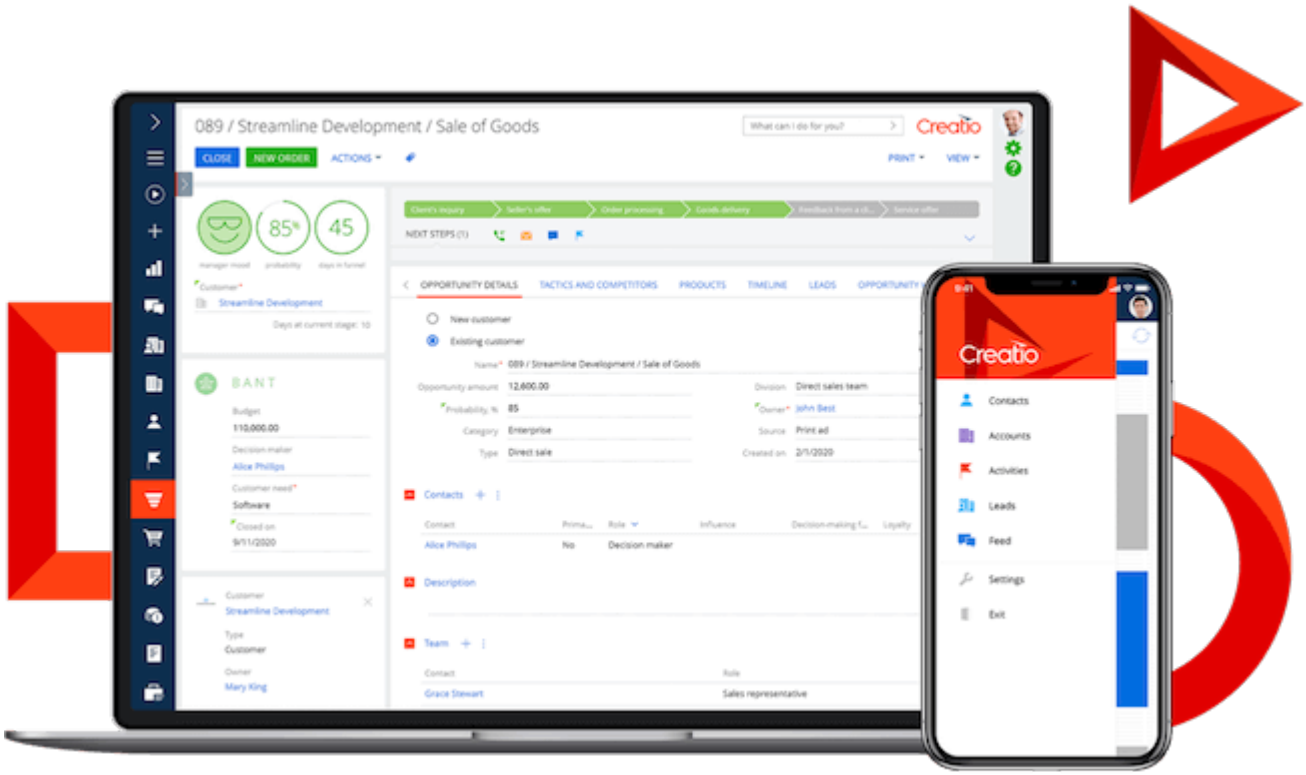



Рисунок 1.1 – CRM-система Creatio (за даними [1])

Однією з ключових функцій Creatio є управління продажами. Система дозволяє компаніям автоматизувати процеси від залучення потенційних клієнтів до укладення угод і управління контрактами. Завдяки цьому, менеджери з продажу

можуть зосередитися на стратегічних завданнях, тоді як рутинні операції виконуються автоматично.

Також, система допомагає планувати, проводити та аналізувати маркетингові кампанії. Вона дозволяє сегментувати клієнтську базу, створювати персоналізовані пропозиції та відстежувати ефективність маркетингових заходів. Це забезпечує більш цілеспрямований підхід до маркетингу та підвищує його ефективність.

Creatio надає потужні інструменти для обслуговування клієнтів (див. рис. 1.2). Система дозволяє швидко обробляти запити клієнтів, автоматизувати сервісні процеси та забезпечувати високий рівень обслуговування через різні канали комунікації. Це сприяє підвищенню задоволеності клієнтів і їх лояльності.

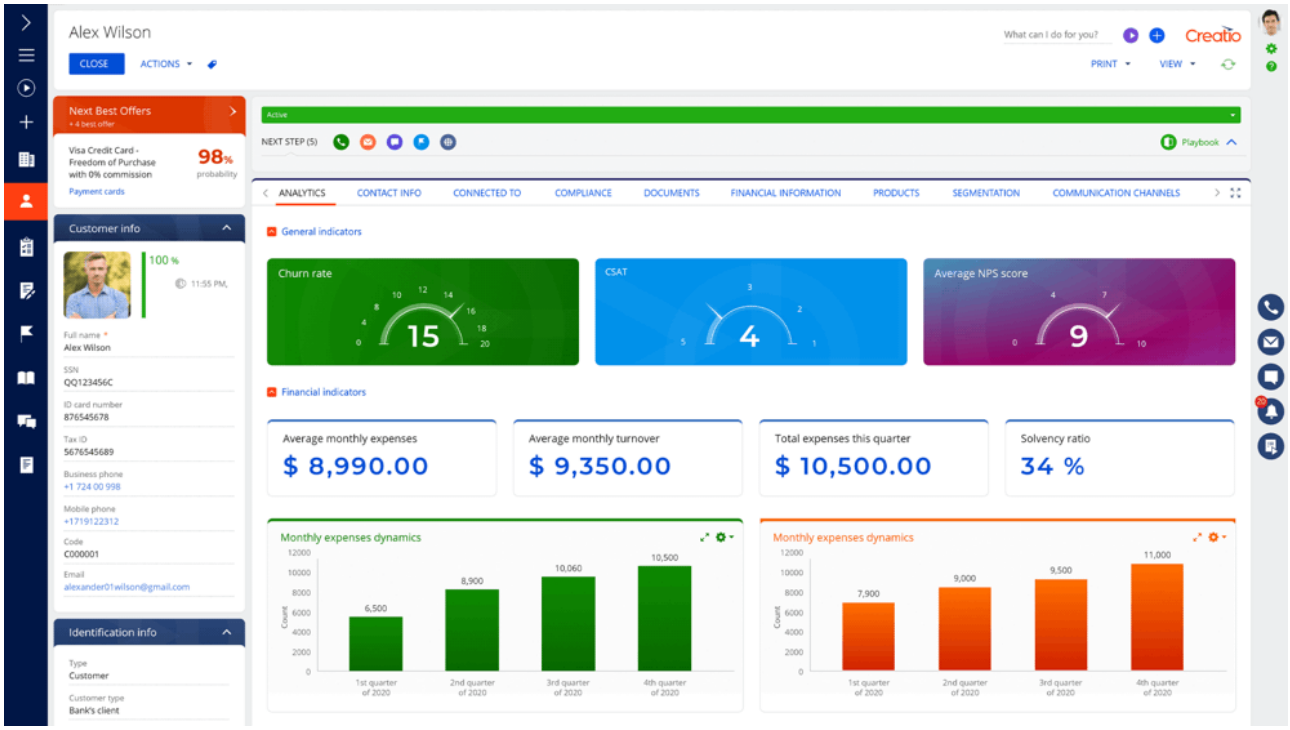



Рисунок 1.2 – Вкладка клієнта в Creatio (за даними [1])

Крім того, Creatio включає функції для автоматизації бізнес-процесів. Система дозволяє візуально моделювати та оптимізувати внутрішні процеси, контролювати виконання завдань і розподіл ресурсів. Це забезпечує підвищення продуктивності та зниження витрат (див. рис. 1.3).

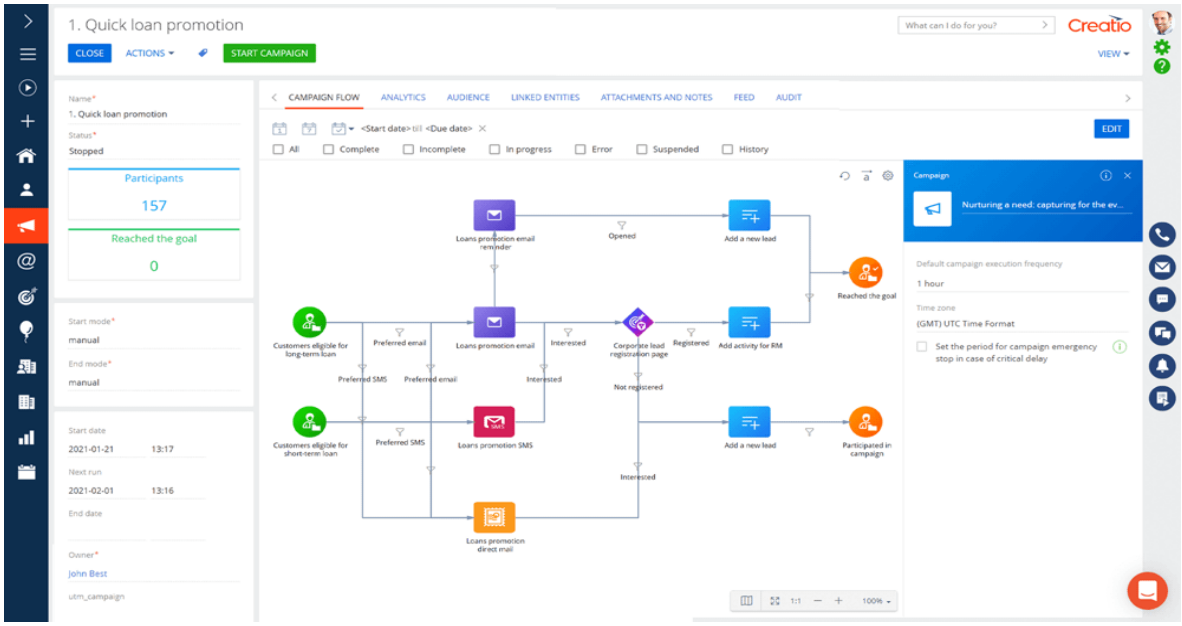



Рисунок 1.3 – Вкладка моделі процесу в Creatio (за даними [1])

Завдяки своїй гнучкості та можливостям інтеграції з іншими системами, Creatio стає незамінним інструментом для компаній, які прагнуть оптимізувати свої бізнес-процеси та покращити взаємодію з клієнтами.

Плюси системи:

– гнучкість та масштабованість: система легко адаптується під потреби різних бізнесів і може бути масштабована в залежності від зростання компанії;
– інтеграція: підтримка інтеграції з іншими системами та додатками, що дозволяє створювати єдине інформаційне середовище;
– user-friendly інтерфейс: інтуїтивно зрозумілий інтерфейс, що полегшує освоєння системи та підвищує ефективність роботи;
– широкий функціонал: об'єднання CRM та BPM функціоналу в одній платформі, що дозволяє комплексно управляти бізнесом.

Мінуси системи:

– ціна: висока вартість ліцензії та впровадження може бути недоступною для малих підприємств;
– складність налаштування: незважаючи на гнучкість, налаштування та адаптація системи можуть вимагати значних зусиль і часу;

– залежність від інтернету: хмарна версія системи потребує стабільного інтернет-з'єднання, що може бути проблемою в регіонах з поганою інфраструктурою.

Creatio є потужним інструментом для автоматизації бізнес-процесів та управління взаємодією з клієнтами. Її гнучкість, багатий функціонал та можливість інтеграції з іншими системами роблять її привабливим вибором для середніх та великих компаній, які прагнуть підвищити ефективність своєї роботи. Однак, висока вартість та складність налаштування можуть бути стримуючими факторами для малих підприємств.

Salesforce Sales Cloud – це хмарна CRM-платформа, розроблена компанією Salesforce, яка призначена для автоматизації та оптимізації процесів продажів (див. рис. 1.4).

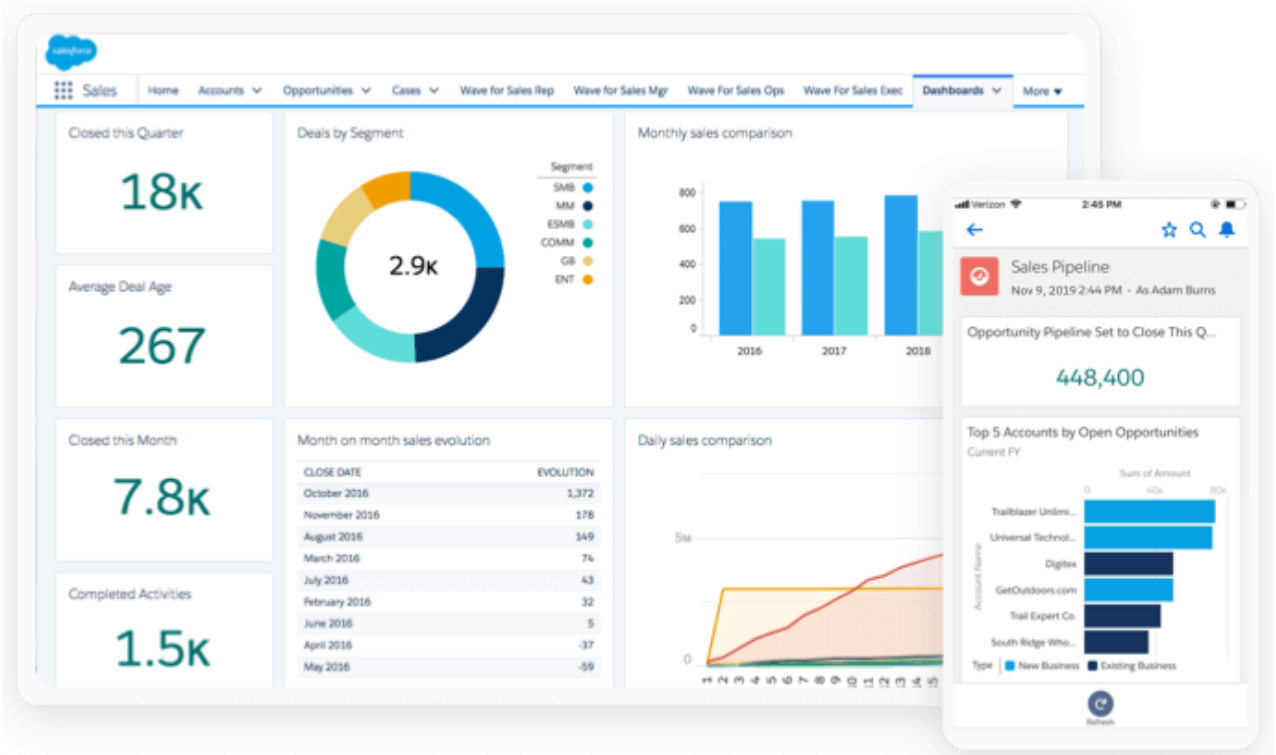



Рисунок 1.4 – CRM-система Salesforce Sales Cloud (за даними [2])

Система забезпечує компаніям ефективні інструменти для управління взаємодією з клієнтами, відстеження продажів та аналітики, що дозволяє підвищити продуктивність і результати роботи відділу продажів.

Salesforce Sales Cloud призначена для того, щоб допомогти компаніям ефективно керувати процесами продажів та взаємодією з клієнтами. Це інструмент, який надає відділу продажів можливість автоматизувати рутинні завдання, відстежувати кожен етап угоди та забезпечувати високий рівень обслуговування клієнтів (див. рис. 1.5).

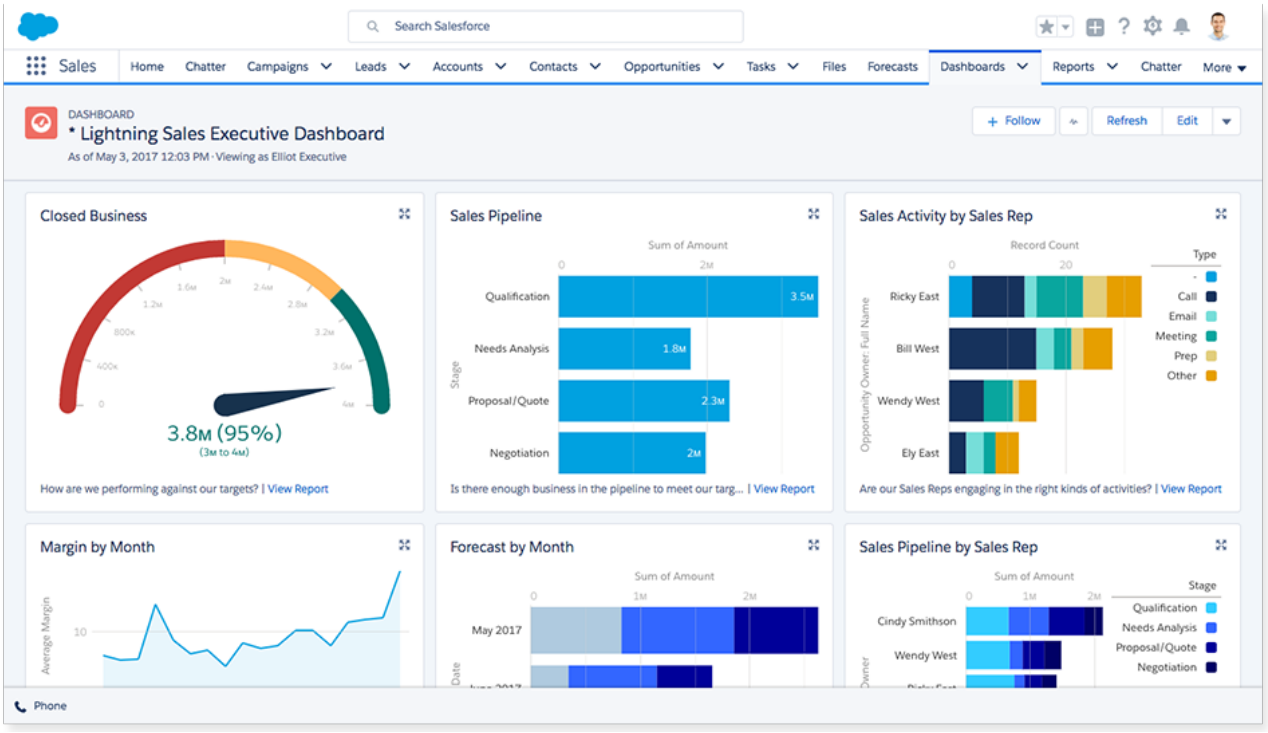



Рисунок 1.5 – Головний екран в Salesforce Sales Cloud (за даними [2])

Система дозволяє зберігати всі контакти в одному місці, відстежувати історію взаємодії з кожним клієнтом та автоматично створювати нагадування про наступні кроки.

Крім того, Sales Cloud надає інструменти для управління всіма етапами продажів, які фіксуються в системі, що допомагає бачити, де саме можуть виникнути проблеми, і вчасно реагувати на них.

Систему можна використовувати для створення детальних звітів про роботу відділу продажів, аналізу ефективності кожного менеджера та прогнозування майбутніх продажів (див. рис. 1.6). Це дозволяє приймати обґрунтовані рішення на основі реальних даних, а не інтуїції.

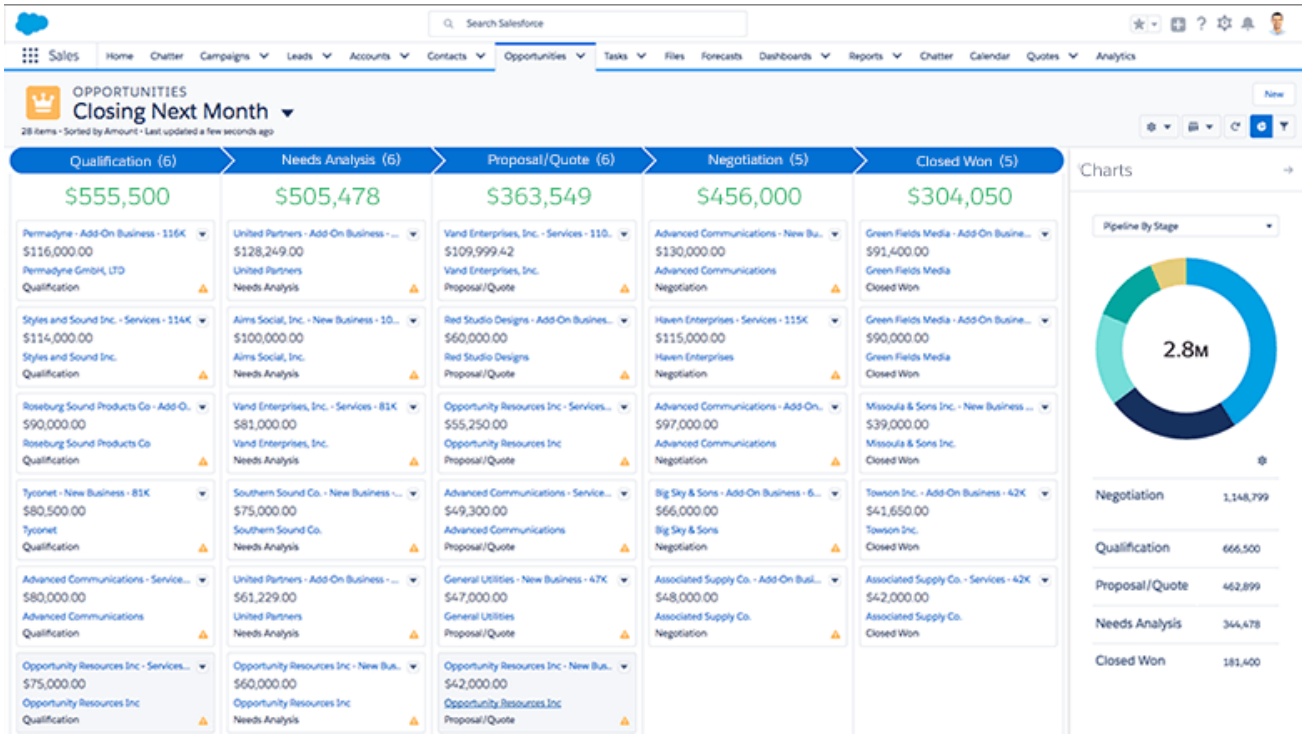



Рисунок 1.6 – Екран замовлень в Salesforce Sales Cloud (за даними [2])

Однією з важливих переваг Sales Cloud є його інтеграція з іншими продуктами Salesforce, такими як Service Cloud та Marketing Cloud. Це забезпечує безшовну взаємодію між різними відділами компанії, що в результаті покращує загальну ефективність бізнесу.

Отже, Salesforce Sales Cloud – це не просто CRM-система, а комплексне рішення для автоматизації та оптимізації продажів, яке допомагає компаніям покращувати взаємодію з клієнтами, підвищувати продуктивність команди та приймати обґрунтовані бізнес-рішення.

Плюси системи:

- гнучкість та масштабованість: система може бути налаштована під конкретні потреби бізнесу та легко масштабується з ростом компанії;
- хмарна платформа: доступ до системи з будь-якого місця та пристрою, що забезпечує мобільність і гнучкість;
- розширені можливості аналітики: потужні інструменти аналітики та звітності допомагають приймати обґрунтовані бізнес-рішення;
- інтеграція з екосистемою Salesforce: легке підключення до інших продуктів Salesforce, таких як Service Cloud, Marketing Cloud та інших;

– підвищення продуктивності команди: автоматизація рутинних завдань та забезпечення інструментів для співпраці.

Мінуси системи:

– висока вартість: ліцензії Salesforce можуть бути дорогими, особливо для малих і середніх підприємств;
– складність налаштування: повне налаштування системи під потреби бізнесу може вимагати значних зусиль та часу;
– навчання персоналу: співробітники можуть потребувати часу на освоєння системи та навчання роботі з нею;
– залежність від інтернету: оскільки це хмарне рішення, для роботи системи необхідне стабільне інтернет-з'єднання.

Salesforce Sales Cloud є потужним інструментом для управління продажами та взаємодією з клієнтами. Його гнучкість, масштабованість та розширені можливості аналітики роблять його привабливим вибором для великих і середніх компаній, які прагнуть автоматизувати та оптимізувати свої процеси продажів. Проте, висока вартість та складність налаштування можуть бути значними стримуючими факторами для малих підприємств.

Keap – це хмарна CRM-платформа, розроблена для малого та середнього бізнесу. Вона надає інструменти для автоматизації маркетингу, управління продажами та клієнтськими взаєминами. Keap допомагає підприємствам залучати нових клієнтів, автоматизувати рутинні завдання та підвищувати ефективність роботи команди.

Система дозволяє автоматизувати багато рутинних завдань, звільняючи час для більш стратегічних дій. Наприклад, автоматизація маркетингових кампаній забезпечує надсилання персоналізованих електронних листів і повідомлень клієнтам на різних етапах їх взаємодії з бізнесом (див. рис. 1.7). Це означає, що клієнти отримують необхідну інформацію в потрібний момент, а компанія може зосередитися на інших важливих завданнях.

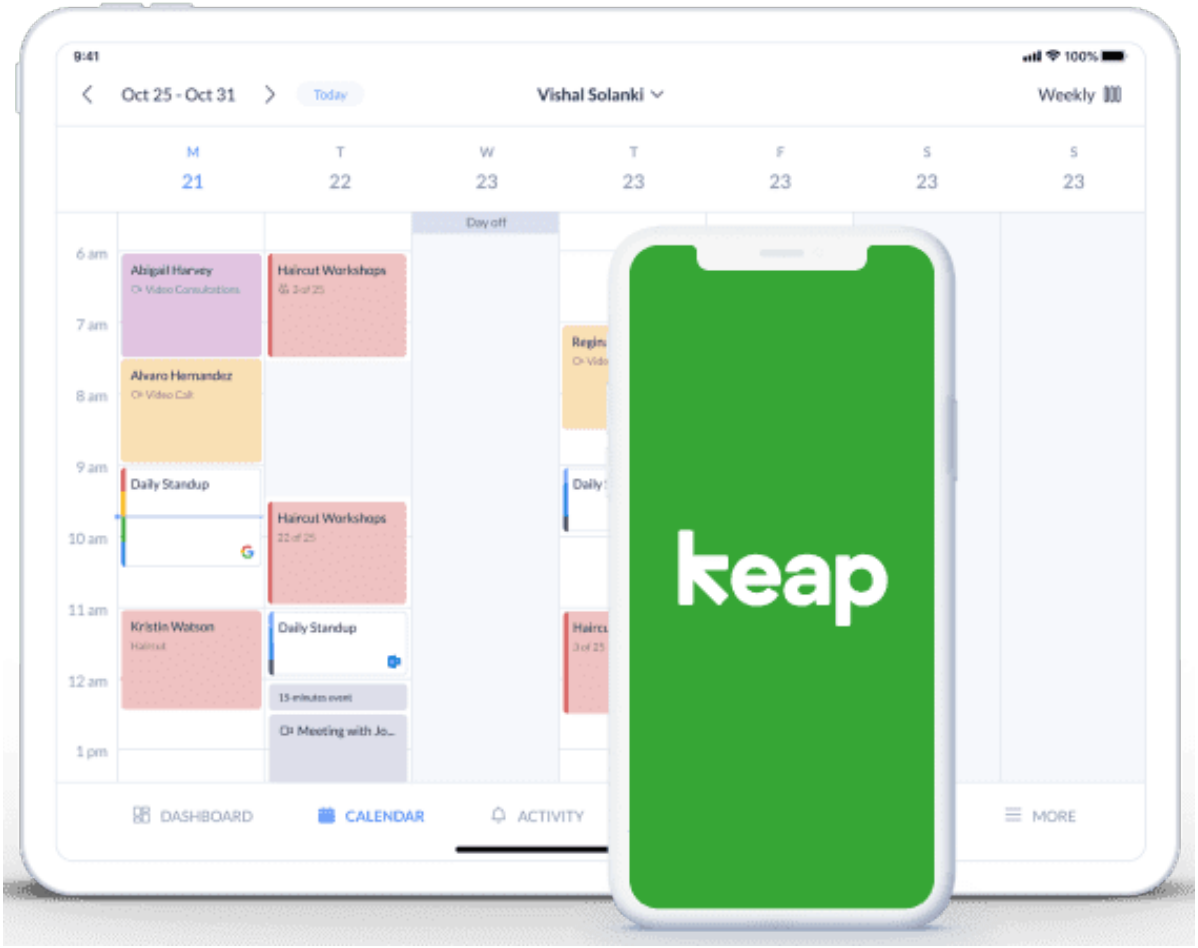



Рисунок 1.7 – CRM-система Keap (за даними [3])

Крім того, Keap допомагає ефективно управляти продажами. Всі потенційні клієнти та угоди відстежуються в єдиній системі, що дозволяє легко бачити, на якому етапі знаходиться кожна угода, які кроки потрібно зробити далі і як забезпечити успішне завершення продажу. Це дозволяє не пропустити жодної можливості та підвищує загальну продуктивність команди.

І, звичайно, обслуговування клієнтів також стає простішим з Keap. Система зберігає всю історію взаємодій з кожним клієнтом, що дозволяє надавати персоналізовану підтримку і швидко вирішувати будь-які питання (див. рис. 1.8). Це підвищує задоволеність клієнтів і зміцнює їхню лояльність до компанії.

Загалом, Keap є комплексним рішенням для малого і середнього бізнесу, яке об'єднує CRM, автоматизацію маркетингу та управління продажами в одній платформі. Це допомагає підприємствам працювати ефективніше, знижувати витрати часу на рутинні завдання і зосереджуватися на розвитку свого бізнесу.

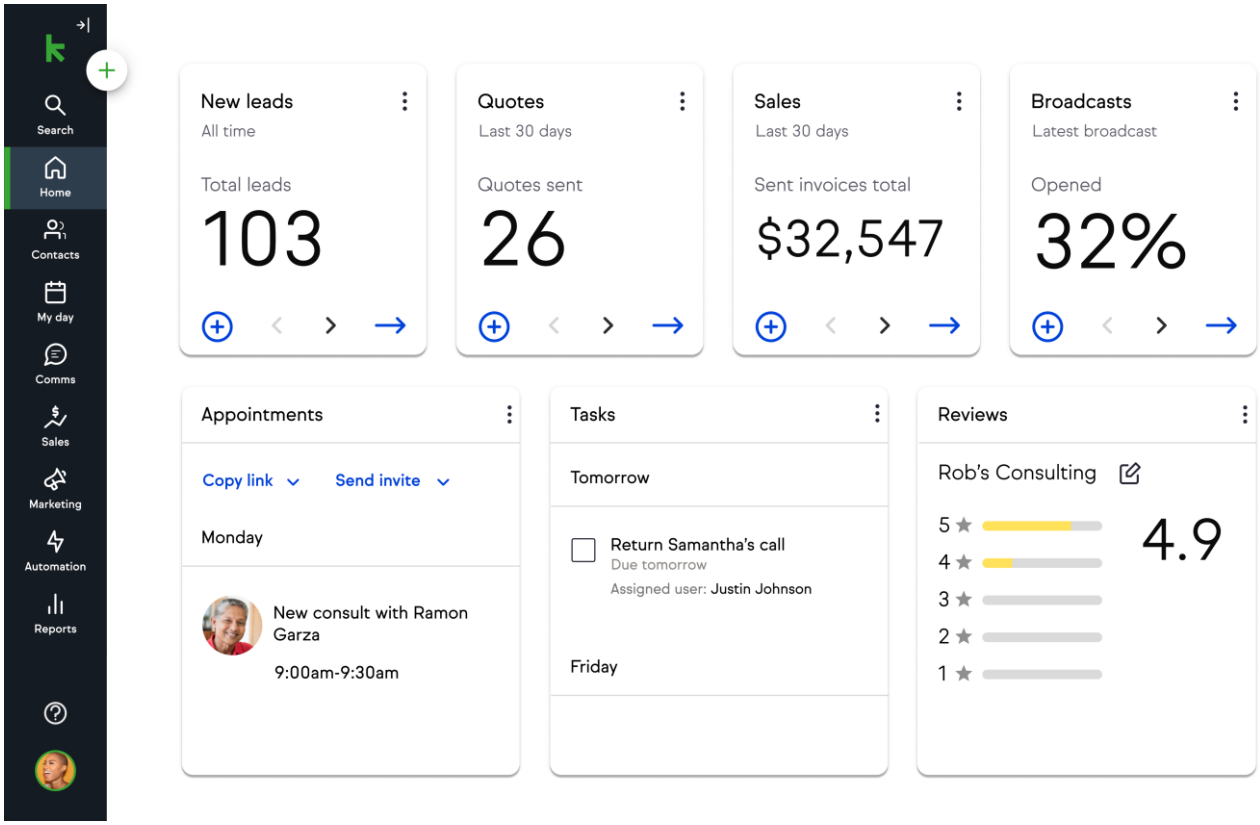



Рисунок 1.8 – Головний екран в Keap (за даними [3])

Плюси системи:

– простота використання: інтуїтивно зрозумілий інтерфейс, що полегшує освоєння системи;
– автоматизація завдань: зниження кількості ручної роботи завдяки автоматизації маркетингових та продажних процесів;
– комплексність рішення: об'єднання CRM, маркетингу та продажів в одній платформі;
– інтеграція: можливість інтеграції з іншими інструментами та додатками, що дозволяє створити єдине інформаційне середовище;
– мобільний доступ: наявність мобільного додатка для управління бізнесом на ходу.

Мінуси системи:

– вартість: для деяких малих підприємств вартість підписки може бути високою;

- потреба в навчанні: хоча інтерфейс простий, для повноцінного використання функціоналу може знадобитися деякий час на навчання;
- залежність від інтернету: для роботи системи необхідне стабільне інтернет-з'єднання, що може бути проблемою в деяких регіонах.

Keap є потужним інструментом для малого та середнього бізнесу, який допомагає автоматизувати маркетингові та продажні процеси, покращити управління клієнтськими взаєминами та підвищити ефективність роботи команди. Завдяки простоті використання та комплексності рішення, Keap може значно полегшити роботу підприємств і допомогти їм досягати кращих результатів. Однак, потенційно висока вартість і потреба в навчанні можуть бути стримуючими факторами для деяких компаній.

В таблиці 1.1 показано порівняння присутності найважливіших функцій системи для використання в галузі 3D-друку в розглянутих програмних рішеннях та розроблюваного додатку.

Таблиця 1.1 - Порівняння існуючих аналогів (таблиця виконана самостійно)

| Критерій | Print Orders | Creatio | Salesforce Sales Cloud | Keap |
|---|---|---|---|---|
| Управління замовленнями | + | + | + | + |
| Управління клієнтською БД | + | + | + | + |
| Звітність | + | + | + | + |
| Мобільний додаток | - | + | + | + |
| Інтеграція з іншими системами | - | + | + | + |
| Автоматизація | + | + | + | + |
| Простота інтерфейсу | + | - | - | - |
| Простота налаштування | + | - | - | - |
| Вартість | + | - | - | - |

Всі розглянуті системи спрямовані на те, щоб охопити різні сфери діяльності, і тому мають надлишковий функціонал, який не потрібен для діяльності в сфері 3D-друку.

## 1.3 Виявлення та вирішення проблем

На основі проведеного аналізу існуючих аналогів, розглянувши всі позитивні і негативні аспекти, можна зробити висновок, що наразі в сфері аддитивних технологій, яка насьогодні стрімко розвивається, немає CRM-системи, яка б була спрямована на вирішення проблем і задач саме в 3D-друці. Тому задача створення зручного, простого в використанні додатку, є актуальною в даний час.

## 1.4 Постановка задачі

У рамках кваліфікаційної роботи необхідно розробити веб-додаток для власників 3D-принтерів, що спеціалізуються на виготовленні виробів на замовлення.

Основна ідея, яка розглядається в даному проекті, полягає у створенні додатку, який сфокусований на конкретній сфері, тобто 3D-друку, і має тільки необхідний функціонал, що спрощує його використання.

Для досягнення мети необхідно вирішити декілька наступних завдань:

- проаналізувати предметну область та визначити необхідні вимоги до додатку;
- визначити основні сутності, необхідний набір атрибутів та спроектувати схему бази даних;
- створити базу даних та міграції для розгортання спроектованої бази даних;
- розробити серверну частину застосунку;
- розробити систему, що має на меті забезпечення безпеки та захисту. Що буде забезпечувати заборону на виконання певних дій у системі без певних прав доступу;
- створити інтерфейс користувача.

# 2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

При розробці будь-якої системи необхідно чітко визначити вимоги, що відображають потреби та очікування користувачів і зацікавлених сторін, а також врахувати обмеження, які впливають на проектування та реалізацію системи. Вимоги до системи є необхідним елементом, оскільки вони встановлюють функціональні та нефункціональні характеристики, які система повинна мати для задоволення потреб користувачів та бізнес-вимог.

Оскільки архітектура системи передбачає поділ на дві компоненти, необхідно розробити вимоги як для системи в цілому, так і для кожної окремої компоненти. Це включає створення функціональних та нефункціональних специфікацій для обох частин системи, перелік яких наведено нижче.

Функціональні вимоги (визначають конкретні функції та можливості, які повинна мати система):

а) управління користувачами:

1) реєстрація Користувачів: система повинна дозволяти новим користувачам реєструватися, заповнюючи форму з необхідними даними (ім'я, електронна пошта, пароль тощо);

2) автентифікація: система повинна дозволяти користувачам входити в систему за допомогою своїх облікових даних;

3) управління ролями: система повинна підтримувати різні ролі користувачів, такі як адміністратор та звичайний користувач. Адміністратори можуть управляти іншими користувачами та їх ролями;

б) управління замовленнями:

1) створення замовлень: користувачі повинні мати можливість створювати нові замовлення, вводячи інформацію про вироби (ціна, вага, матеріал, час виготовлення, кількість), а також дані про клієнта;

2) редагування замовлень: користувачі повинні мати можливість редагувати існуючі замовлення для внесення змін у деталі замовлення;

3) видалення замовлень: користувачі повинні мати можливість видаляти замовлення;

4) перегляд замовлень: всі користувачі повинні мати можливість переглядати список всіх замовлень, включаючи деталі кожного замовлення;

в) управління клієнтами:

1) створення клієнтів: користувачі повинні мати можливість додавати нових клієнтів до системи, заповнюючи відповідні дані (П.І.Б., адреса, контактні дані);

2) редагування клієнтів: користувачі повинні мати можливість редагувати інформацію про існуючих клієнтів;

3) видалення клієнтів: користувачі повинні мати можливість видаляти клієнтів із системи;

4) перегляд клієнтів: користувачі повинні мати можливість переглядати список всіх клієнтів, включаючи їх деталі та історію замовлень;

г) управління принтерами:

1) додавання принтерів: користувачі повинні мати можливість додавати нові принтери до системи, вводячи інформацію про область друку, встановлене сопло, час напрацювання;

2) редагування принтерів: користувачі повинні мати можливість редагувати інформацію про існуючі принтери;

3) видалення принтерів: користувачі повинні мати можливість видаляти принтери із системи;

4) перегляд принтерів: користувачі повинні мати можливість переглядати список всіх принтерів, включаючи їх деталі;

д) адміністрування:

1) управління користувачами: адміністратори повинні мати можливість переглядати, додавати, редагувати та видаляти користувачів;

2) моніторинг дій користувачів: адміністратори повинні мати можливість переглядати журнал дій користувачів для моніторингу активності;

е) інші функції:

1) пошук та фільтрація: користувачі повинні мати можливість здійснювати пошук замовлень та клієнтів за різними критеріями (ім'я, ІД, статус замовлення);

2) календар подій: користувачі повинні мати доступ до календаря подій для створення нагадувань та важливих подій;

3) експорт даних: користувачі повинні мати можливість експортувати дані замовлень у формат PDF.

Нефункціональні вимоги (визначають характеристики системи, які не пов'язані з конкретною функціональністю, але важливі для користувацького досвіду та продуктивності системи):

а) продуктивність:

1) швидкість відповіді: система повинна забезпечувати швидкий відгук на користувацькі запити, зокрема створення, редагування та видалення записів не повинно займати більше 3-4 секунд;

2) підтримка користувачів: система повинна підтримувати одночасну роботу до 500 користувачів без значної втрати продуктивності;

б) безпека:

1) захист даних: система повинна забезпечувати захист даних користувачів, включаючи хешування паролів;

2) автентифікація та авторизація: система повинна використовувати безпечні механізми автентифікації та авторизації для контролю доступу до даних та функцій;

в) зручність використання:

1) інтуїтивний інтерфейс: система повинна мати мінімалістичний та інтуїтивний інтерфейс, який легко використовувати навіть недосвідченим користувачам;

2) адаптивний дизайн: інтерфейс системи повинен бути адаптивним та коректно відображатися на різних пристроях (ПК, планшети, смартфони).

Ці вимоги забезпечують створення функціонального, надійного та зручного для користувачів веб-додатку, який відповідає потребам власників 3D-принтерів та їх клієнтів.

# 3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ

Контекстна діаграма є одним з видів діаграм в системному аналізі і проектуванні. Вона використовується для зображення зовнішнього середовища системи і її взаємодії з іншими системами або сутностями. Основна мета контекстної діаграми полягає в уточненні меж системи і визначенні зовнішніх сутностей, з якими вона взаємодіє (див. рис. 3.1).


Рисунок 3.1 - Контекстна діаграма ПС «Облік замовлень з 3D-друку» (рисунок виконаний самостійно)

Основні елементи контекстної діаграми зазвичай включають:

- основна система (ПС «Облік замовлень з 3D-друку») - це об'єкт або система, яка досліджується чи проектується. Вона зображується у центрі діаграми;
- зовнішні сутності (Користувач, База клієнтів, PDF і т.п.) - це інші системи, користувачі, апаратне чи програмне забезпечення, які взаємодіють з основною системою. Вони зображуються навколо основної системи;
- взаємодії (Інформація, Авторизація і т.п.) - вони показують типи взаємодій між основною системою та зовнішніми сутностями. Це можуть бути потоки даних, комунікації, передачі інформації тощо.

Контекстна діаграма допомагає зрозуміти, як основна система взаємодіє зі своїм зовнішнім середовищем і визначається як перший крок при аналізі або проектуванні системи.

## 3.1 UML-проектування

Мета розробки варіантів використання полягає у визначенні завершеного аспекту або фрагменту поведінки певної сутності без розкриття її внутрішньої структури.

Варіанти використання призначені насамперед для визначення функціональних вимог до системи і управляють усім процесом розробки. Всі основні види діяльності, такі як аналіз, проектування та тестування, виконуються на основі варіантів використання. Під час аналізу та проектування варіанти використання допомагають зрозуміти, як результати, які користувач хоче отримати, впливають на архітектуру системи, і як повинні поводитися компоненти системи, щоб забезпечити необхідну функціональність для користувача.

Діаграма варіантів використання (use case diagram) призначена для відображення зовнішнього функціонування системи, що проектується, та її взаємодії з користувачами. Основне завдання діаграм використання полягає у специфікації вимог до системи на початкових етапах проектування, коли вирішуються найбільш загальні питання, пов'язані з призначенням системи, що розробляється.

Діаграма складається з наступних елементів:

- зовнішні користувачі (actors) – це об'єкти, які передають або отримують інформацію для системи. Ними можуть бути як фізичні об'єкти, такі як люди і механізми, так і програмні системи. Один фізичний об'єкт може бути представлений кількома користувачами, якщо він взаємодіє з різними функціями системи;
- блоки використання (use cases) – це групи функцій системи, які об'єднуються в єдине ціле для зовнішнього користувача;

– зв'язки між блоками використання та зв'язки між блоками використання і зовнішніми користувачами.

Перед початком розробки веб-додатку «Print Orders» були визначені основні функції зі сторони користувача. Після детального аналізу була створена Use-case діаграма (див. рис. 3.2), яка описує функції розроблюваного програмного продукту доступні кожній групі користувачів.

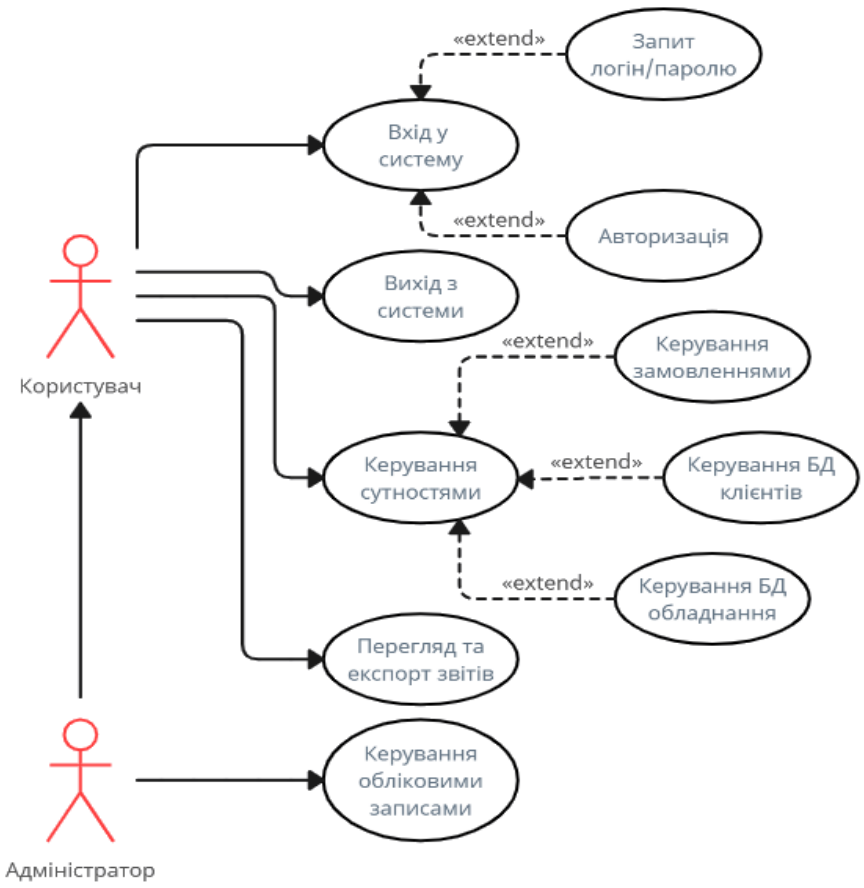



Рисунок 3.2 – USE-CASE діаграма веб-додатку «Print Orders» (рисунок виконаний самостійно)

У програмному забезпеченні передбачено два типи зовнішніх користувачів, які взаємодіють із системою: користувачі та адміністратори.

Користувачі можуть ввійти в додаток, вийти з нього, керувати сутностями, такими, як замовленнями, обладнанням та клієнтами.

Під «керуванням» розуміється можливість створювати, переглядати, редагувати та видаляти замовлення, дані про обладнання та клієнтів.

Адміністратори мають доступ до всіх функцій, які доступні звичайним користувачам, а також у них є можливість управління користувачами, тобто створення, редагування та видалення облікових записів користувачів, забезпечуючи контроль над доступом до системи.

## 3.2 Опис процесу діяльності

За допомогою діаграм діяльності варіантів використання можна детально ознайомитись з процесом роботи та діями що проходять всередині цих процесів.

Діаграма діяльності варіанту використання «Авторизація», містить в собі перелік подій та необхідних дій для авторизації в систему під своїм обліковим записом з використанням існуючих даних для введення їх у поля логіну та паролю (див. рис. 3.3).

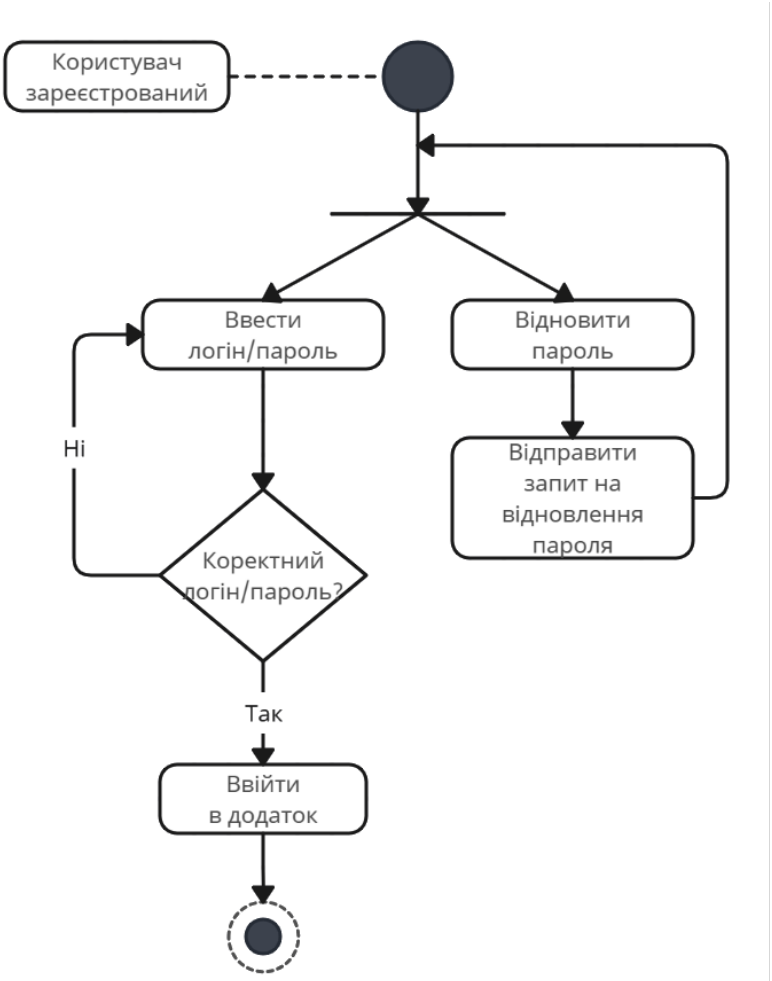



Рисунок 3.3 – Діаграма діяльності варіанту використання «Авторизація» (рисунок виконаний самостійно)

Діаграма діяльності варіанту використання «Створення замовлення», містить в собі перелік дій необхідних для вдалого виконання процесу додавання запису про замовлення до бази даних (див. рис. 3.4). При заповненні усіх необхідних полів замовлення, потрібно натиснути кнопку «Зберегти», після чого дані будуть внесені до бази даних.

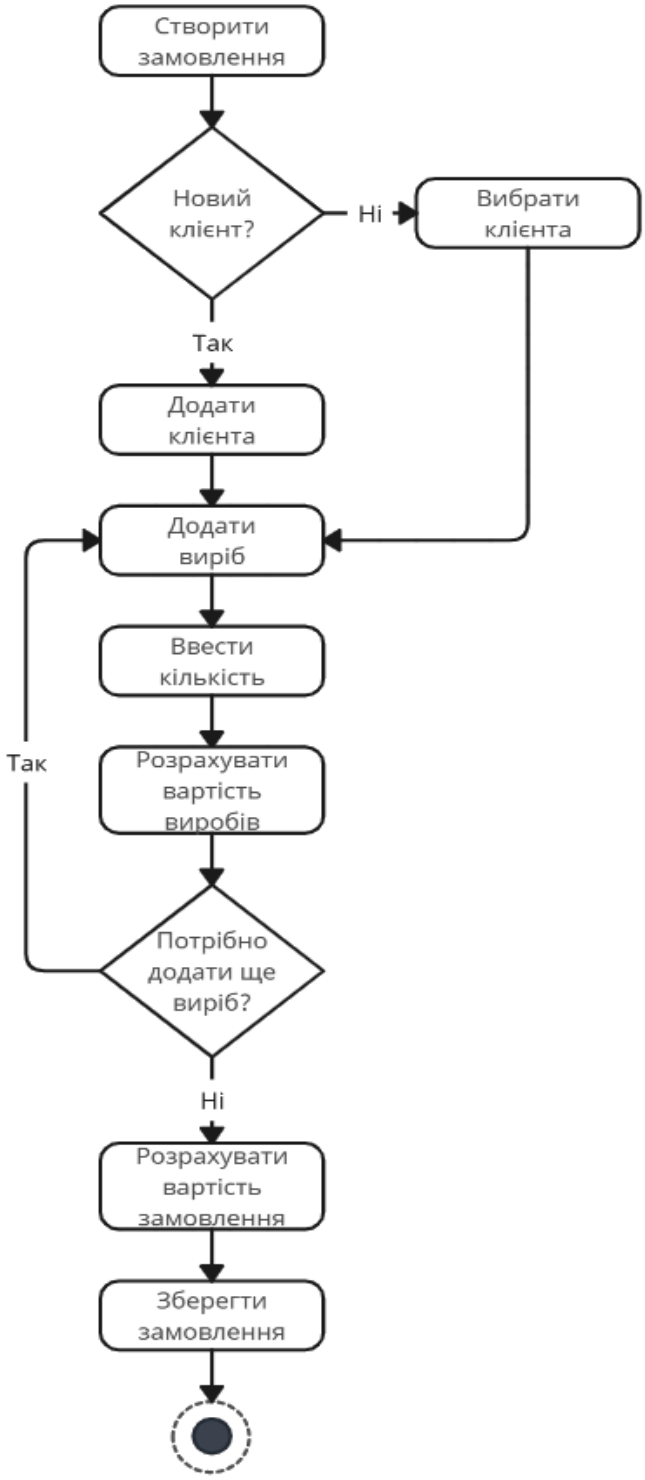



Рисунок 3.4 – Діаграма діяльності варіанту використання «Створення замовлення» (рисунок виконаний самостійно)

## 3.3 Проектування архітектури ПЗ

Взаємодія користувача з системою здійснюється за моделлю клієнт-сервер (див. рис. 3.5). Це означає, що сервер не може надсилати дані клієнту без попереднього запиту. Сервер не зберігає проміжних даних про клієнта, тому кожен запит клієнта повинен містити повну інформацію про нього [9]. Це робить систему незалежною від кількості серверів, дозволяє швидко масштабуватися та забезпечує гнучкість.

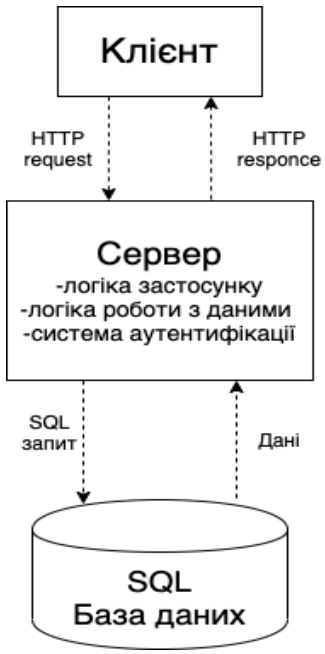



Рисунок 3.5 – Схема клієнт-серверної архітектури (за даними [9])

Сервер відповідає за центральну бізнес-логіку додатку. Він не зберігає даних безпосередньо, але керує їх обробкою та передачею. На сервері розташовані алгоритми обробки даних, що включають в себе різноманітні аналітичні та обчислювальні функції, необхідні для функціонування додатку. Однією з ключових функцій сервера є система автентифікації, яка забезпечує безпеку доступу до додатку і його функцій.

SQL база даних служить основним сховищем для зберігання інформації, що генерується та оброблюється додатком. Вона використовується для ефективного зберігання та організації даних, необхідних для роботи додатку. База даних

інтегрована з сервером, що дозволяє ефективно звертатися до даних під час виконання операцій та запитів, що стосуються користувачів і їх даних.

Клієнтська частина додатку забезпечує інтерфейс для взаємодії з користувачем. Вона відповідає за відображення інформації, яку обробляє сервер, та за передачу введених користувачем даних на сервер для подальшої обробки. Клієнтська частина додатку забезпечує зручність інтерфейсу та взаємодії, що є критичним для задоволення потреб користувачів.

### 3.4 Проектування структури зберігання даних

База даних є невід'ємною та критично важливою складовою такої системи. Вона виконує роль центрального сховища для збереження, організації та керування великим обсягом даних, що використовуються в системі.

Діаграма бази даних, зображена на рисунку 3.6, надає візуальне відображення структури всіх таблиць і полів, які використовуються для збереження інформації.

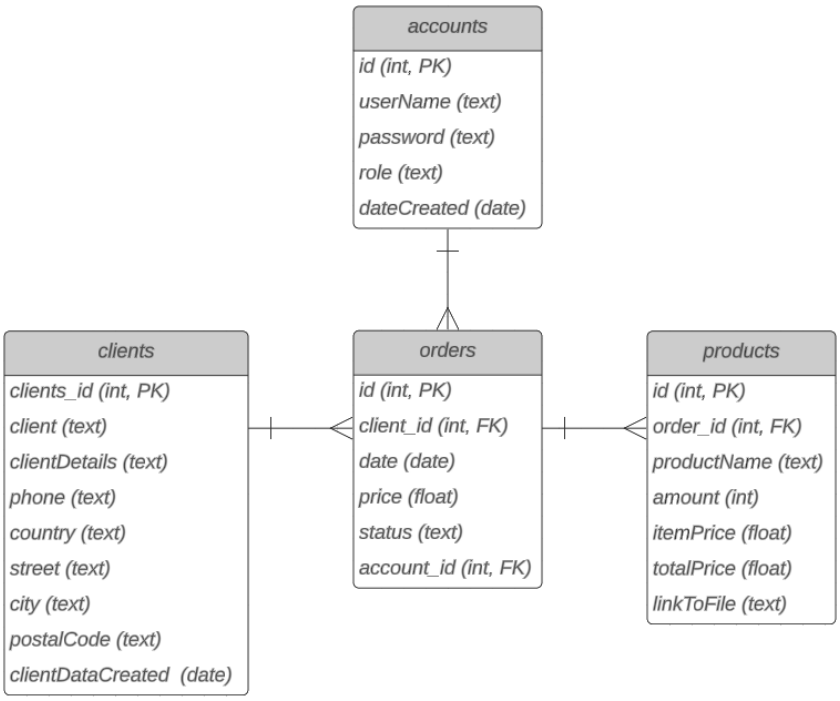



Рисунок 3.6 – Діаграма БД (рисунок виконаний самостійно)

У складі бази даних присутні 4 таблиць, які містять сутності, що відповідають їхньому природному зв'язку. Кожна таблиця має унікальний первинний ключ, який ідентифікує кожен запис у таблиці.

а) сутність «accounts» представляє модель облікових записів.

1) атрибут «id» унікальний ідентифікатор облікового запису та його первинний ключ;

2) атрибут «username» зберігає ім'я користувача;

3) атрибут «password» зберігає хеш пароля користувача;

4) атрибут «role» зберігає роль користувача (наприклад, admin);

5) атрибут «dateCreated» зберігає дату створення облікового запису.

б) сутність «clients» представляє модель клієнтів.

1) атрибут «client_id» унікальний ідентифікатор клієнта та його первинний ключ;

2) атрибут «client» зберігає ім'я клієнта;

3) атрибут «clientDetails» зберігає деталі клієнта;

4) атрибут «phone» зберігає номер телефону клієнта;

5) атрибут «country» зберігає країну клієнта;

6) атрибут «street» зберігає вулицю клієнта;

7) атрибут «city» зберігає місто клієнта;

8) атрибут «postalCode» зберігає поштовий індекс клієнта;

9) атрибут «clientDateCreated» зберігає дату створення запису клієнта.

в) сутність «orders» представляє модель замовлень.

1) атрибут «id» унікальний ідентифікатор замовлення та його первинний ключ.

2) атрибут «client_id» є зовнішнім ключем, який посилається на клієнта, що зробив замовлення;

3) атрибут «date» зберігає дату створення замовлення;

4) атрибут «price» зберігає ціну замовлення;

5) атрибут «status» зберігає статус замовлення;

6) атрибут «account_id» є зовнішнім ключем, який посилається на користувача, що створив замовлення.

г) сутність «products» представляє модель продуктів.

1) атрибут «id» унікальний ідентифікатор продукту та його первинний ключ;

2) атрибут «order_id» є зовнішнім ключем, який посилається на замовлення, до якого належить продукт;

3) атрибут «productName» зберігає назву продукту;

4) атрибут «amount» зберігає кількість продукту;

5) атрибут «itemPrice» зберігає ціну за одиницю продукту;

6) атрибут «totalPrice» зберігає загальну ціну продукту;

7) атрибут «linkToFile» зберігає посилання на файл моделі для друку .

## 3.5 Створення дизайну системи

Програмний додаток "Print Orders" має сучасний та інтуїтивно зрозумілий інтерфейс, що забезпечує зручне та ефективне користування. Кольорова гама додатку переважно складається з білого, червоного та сірого кольорів. Білий колір є основним фоном, що створює чистий і зрозумілий вигляд, а також полегшує сприйняття інформації. Червоний колір використовується для акцентування важливих елементів, таких як логотип, кнопки та повідомлення, привертаючи увагу користувача до ключових дій та інформації. Сірий колір використовується для тексту та фону меню, забезпечуючи необхідний контраст та гармонійний вигляд (див. рис. 3.7).

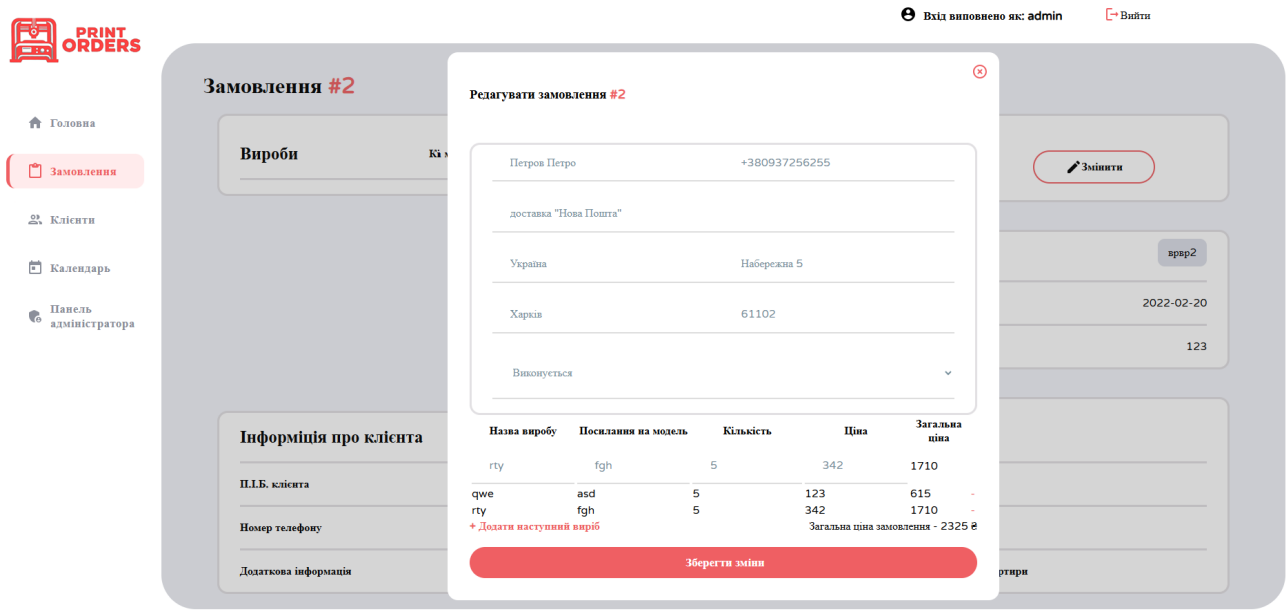



Рисунок 3.7 – Приклад дизайну додатку PrintOrders (рисунок виконаний самостійно)

Інтерфейс додатку містить прості та зрозумілі форми і кнопки. Кнопки мають заокруглені кути, що додає їм сучасного вигляду. Кнопка входу знаходиться під полями для введення логіна та пароля, що є логічним та інтуїтивно зрозумілим розташуванням. У формі авторизації поля для вводу логіна та пароля розташовані вертикально одне під одним, що є звичним для користувачів (див. рис. 3.8).

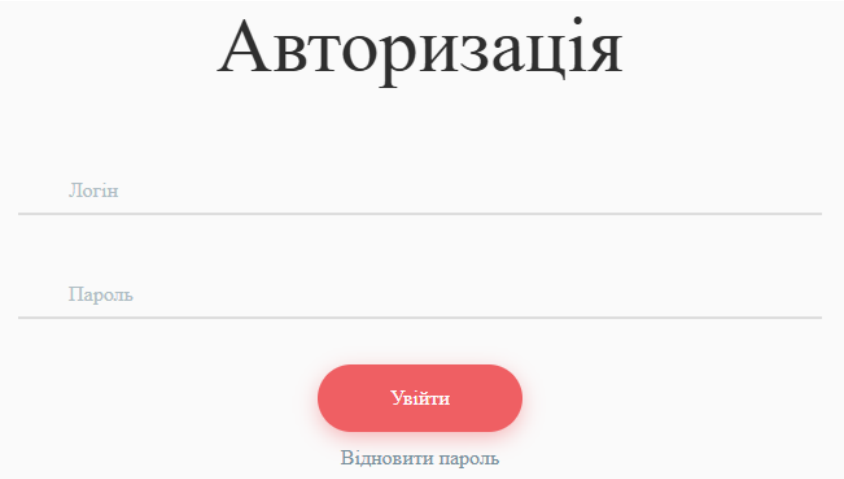



Рисунок 3.8 – Приклад розміщення елементів форми авторизації (рисунок виконаний самостійно)

Кнопка додавання нових замовлень розташована праворуч у вікні для створення нового замовлення, що полегшує доступ до цієї функції (див. рис. 3.9).

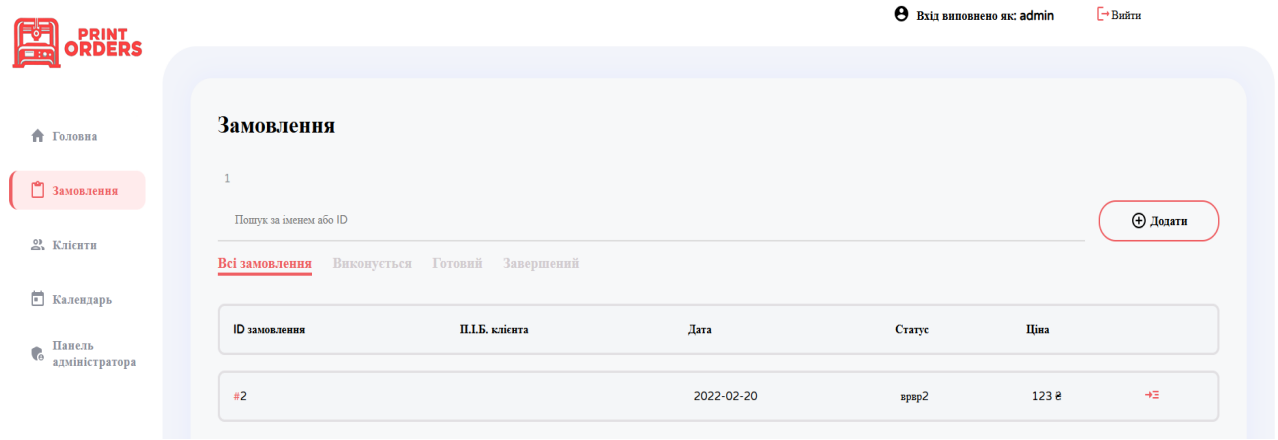



Рисунок 3.9 – Приклад розміщення кнопок (рисунок виконаний самостійно)

Поля вводу також мають прямокутну форму із заокругленими кутами, що гармонійно поєднується з формою кнопок. У формі додавання нового замовлення поля розташовані вертикально, що забезпечує зручність введення даних (див. рис. 3.10).

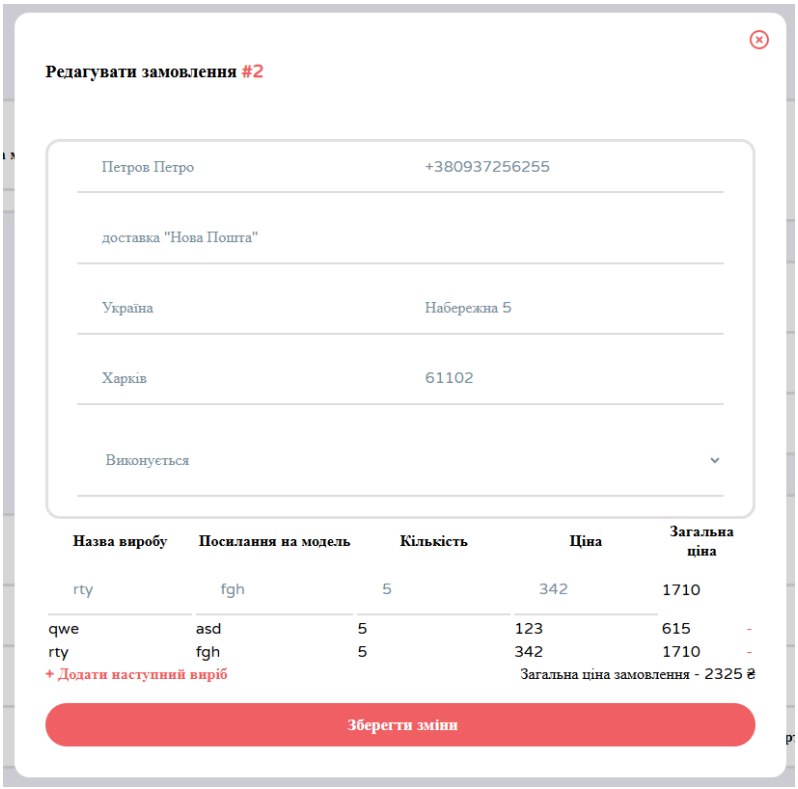



Рисунок 3.10 – Дизайн вікна редагування замовлення (рисунок виконаний самостійно)

Меню додатку розташоване зліва і містить чотири основні розділи: "Головна", "Замовлення", "Клієнти", "Календар", а також "Панель адміністратора". Іконки та текст у меню розташовані горизонтально, кожен пункт меню має піктограму та текстовий опис, що робить навігацію простою та зрозумілою (див. рис.3.11).

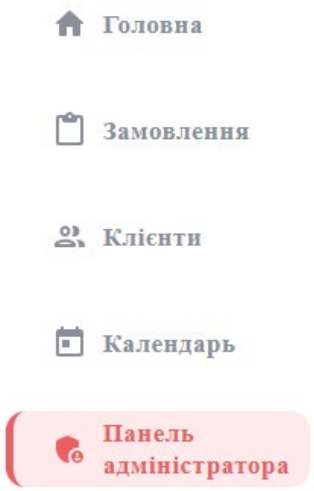



Рисунок 3.11 – Вигляд головного меню додатку (рисунок виконаний самостійно)

Важливі елементи, такі як логотип та привітання, розташовані у верхньому лівому куті та праворуч від форми авторизації відповідно. Це забезпечує логічний і зрозумілий розподіл інформації. Повідомлення, виконані червоним кольором та великим шрифтом, привертають увагу користувача до важливих деталей.

З точки зору UX дизайну, додаток має високу інтуїтивність завдяки використанню знайомих іконок та логічного розташування елементів. Поля та кнопки мають зрозумілі написи та інструкції, що полегшує користування додатком навіть для новачків. Дизайн додатку є простим та чистим, з мінімумом відволікаючих елементів, що дозволяє користувачам швидко знайти потрібну інформацію або виконати необхідну дію.

Загалом, додаток "Print Orders" має привабливий та зручний дизайн, що забезпечує позитивний досвід користувача та ефективність виконання завдань.

# 4 ОПИС ПРОГРАМНИХ РІШЕНЬ

## 4.1 Засоби розробки

### 4.1.1. Засоби розробки серверної частини

Розробка серверних застосунків набуває все більшої популярності в сучасний час, і це не випадково. Потреба в створенні таких застосунків зростає через розширення цифрової сфери та збільшення кількості користувачів онлайн-сервісів. Ці застосунки повинні забезпечувати максимальну швидкодію та продуктивність при виконанні користувацьких запитів. Це досягається шляхом вибору оптимальних технологій, які забезпечують ефективну обробку даних та оптимізовані алгоритми обробки запитів. Для досягнення цієї мети була використана технологія Node.js.

Node.js – вільне та відкрите середовище виконання JavaScript, яке побудоване на двигуні JavaScript V8, розробленому Google для використання в браузері Google Chrome. Воно дозволяє виконувати код JavaScript на сервері та створювати масштабовані та ефективні мережеві додатки [10].

Основна ідея Node.js полягає в тому, що розробники можуть використовувати JavaScript, як мову програмування для побудови серверних додатків. Це робить Node.js ідеальним рішенням для розробки веб-додатків, додатків чату, API та інших типів програмного забезпечення.

У даному середовищі можна виділити декілька переваг, зокрема:

- висока продуктивність: Node.js використовує асинхронну модель програмування, яка дозволяє обробляти велику кількість запитів одночасно без блокування потоку виконання. Це забезпечує високу продуктивність та швидку відповідь на запити.
- масштабованість: завдяки здатності обробляти багато запитів одночасно, Node.js добре підходить для створення масштабованих додатків. Він також легко масштабується горизонтально шляхом додавання серверів.

- єдина мова програмування: використання JavaScript на стороні як клієнта, так і сервера спрощує розробку та підтримку коду, оскільки розробники можуть працювати з єдиною мовою програмування.
- широкий вибір модулів та пакетів: Node.js має велику кількість модулів, доступних через npm, що дозволяє розробникам легко використовувати готові рішення для розширення функціональності своїх додатків.

Також варто враховувати деякі недоліки цього середовища, а саме:

- однопоточність: Node.js використовує один потік виконання, що може стати обмеженням для обробки CPU-інтенсивних завдань. Довгі операції можуть блокувати потік виконання, що призводить до зниження продуктивності.
- нестабільність деяких модулів: через велику кількість сторонніх модулів у Node.js існує ризик неповної сумісності, помилок або вразливостей у деяких з них. Це може вплинути на стабільність і безпеку додатків.

Node.js залишається популярним вибором для створення серверних додатків завдяки своїм численним перевагам, незважаючи на деякі недоліки. Правильне використання цієї технології дозволяє створювати високопродуктивні та масштабовані рішення для сучасних онлайн-сервісів.

Також для розробки серверної частини був використаний Express.js. Це мінімалістичний і гнучкий веб-фреймворк для Node.js, призначений для побудови серверних додатків та API. Він став одним із найпопулярніших фреймворків для Node.js завдяки своїй простоті, продуктивності та широким можливостям.

### 4.1.2. Системи управління базами даних

У сучасному світі використання баз даних для збереження, організації та управління великим обсягом структурованої інформації стало характерною рисою систем, які працюють з великими обсягами даних. Реляційні системи управління базами даних вважаються невід'ємною складовою сучасної розробки програмного забезпечення. Вони забезпечують потужність та ефективність управління

великими обсягами даних, що використовуються у різноманітних додатках та системах.

У сучасних реляційних СУБД реалізовано безліч функцій для ефективного управління базами даних. Вони дозволяють створювати, змінювати та видаляти таблиці, індексувати дані для швидшого пошуку, контролювати транзакції для забезпечення цілісності даних і виконувати резервне копіювання для захисту даних. Також підтримують розширені функції, такі як тригери та збережені процедури, що дозволяють автоматизувати дії та зберігати набори інструкцій на стороні сервера.

Одним із ключових аспектів реляційних СУБД є продуктивність. Швидкодія та ефективність операцій з базою даних залежать від оптимізації запитів, індексування, кешування, паралельного виконання та оптимізації обсягу даних. Сучасні реляційні СУБД застосовують різні техніки та алгоритми для підвищення продуктивності та забезпечення швидкого доступу до даних.

Надійність також є важливим аспектом СУБД. Вони включають механізми для забезпечення надійності та цілісності даних, такі як транзакційний контроль, відновлення після збоїв і резервне копіювання. Резервне копіювання даних дозволяє відновити базу у разі втрати або пошкодження інформації.

Серед основних причин використання реляційних СУБД можна виділити наступні:

- структуроване збереження даних: зберігання даних у вигляді таблиць забезпечує логічну організацію та зв'язки між даними.
- запити та аналітика даних: підтримка мови SQL дозволяє виконувати різноманітні операції з даними, включаючи вибір, фільтрацію, сортування, об'єднання та аналіз.
- цілісність та надійність даних: обмеження на поля (унікальність, зовнішні ключі) та транзакції допомагають підтримувати даних.
- безпека даних: рівні доступу, обмеження прав користувачів та шифрування даних забезпечують захист від несанкціонованого доступу та зламів.

- масштабованість: реляційні СУБД можуть обробляти великі обсяги даних та підтримувати багато користувачів одночасно, з можливістю реплікації та розподіленої обробки.
- резервне копіювання та відновлення даних: засоби для резервного копіювання та відновлення забезпечують захист даних та можливість відновлення у разі втрати або пошкодження.

Хоча реляційні системи управління базами даних мають багато переваг, усі вони також мають деякі недоліки, які варто враховувати:

- гнучкість: потреба у визначенні схеми бази даних заздалегідь може стати проблемою при необхідності внесення змін або додавання нових елементів.
- обробка складних структур даних: реляційні СУБД не завжди ефективно обробляють складні структури даних, такі як вкладені об'єкти, масиви чи географічні дані.
- обмежена масштабованість: деякі реляційні СУБД можуть стикатися з обмеженнями масштабованості при великому обсязі даних або високих навантаженнях.
- високі вимоги до апаратного забезпечення: для оптимальної продуктивності та швидкодії деякі реляційні СУБД можуть вимагати потужного апаратного забезпечення.

Реляційні системи управління базами даних залишаються ключовим елементом сучасної розробки програмного забезпечення, забезпечуючи потужні інструменти для збереження, організації та управління даними.

Тому, з огляду на потрібність у продуктивності, безпеці та надійності, для розробки системи обліку замовлень було прийнято рішення використовувати класичну реляційну СУБД, а саме MySQL.

MySQL – це система управління реляційними базами даних з відкритим кодом, що базується на структурованій мові запитів (SQL). MySQL доступний у всіх основних операційних системах, включаючи Windows, Linux тощо.

MySQL, як і інші реляційні бази даних, зберігає дані в таблицях, стовпцях та рядках. Кожен запис визначається унікальним ідентифікатором. Її основою завжди були продуктивність та надійність бази даних. MySQL була розроблена та оптимізований для арени веб-розробок; це, мабуть, найпоширеніша база даних, яка використовується при розгортанні веб-серверів.

### 4.1.3. Засоби розробки клієнтської частини

Завдання створення якісного та зручного у використанні інтерфейсу є вкрай важливим, адже саме інтерфейс визначає значну частину користувацького досвіду та впливає на успіх онлайн-проекту.

Веб-інтерфейс є обличчям веб-додатку або сайту, від якого залежить, як користувачі взаємодіють з продуктом і як вони його сприймають. Якість веб-інтерфейсу безпосередньо впливає на задоволеність користувачів, їх залученість, утримання та конверсію.

Для створення інтерфейсу користувача зазвичай використовують такі сучасні технології:

- HTML: це основна мова розмітки для створення структури веб-сторінок.
- CSS: використовується для визначення зовнішнього вигляду веб-сторінок.
- JavaScript: мова програмування, що надає динамічність веб-сторінкам, дозволяючи створювати взаємодію з користувачем, валідацію форм, анімацію, асинхронне завантаження даних (за допомогою AJAX) та багато іншого.
- UI-бібліотеки: бібліотеки, такі як Bootstrap, Foundation, Material-UI, надають готові компоненти та стилі, що допомагають швидко створювати привабливі веб-інтерфейси.

Одним з ключових аспектів при розробці веб-додатків є вибір фреймворку для користувацької частини. Існує безліч фреймворків, серед яких для розробки інтерфейсу був обраний React.

React – це одна з найпопулярніших JavaScript-бібліотек для розробки інтерфейсу користувача. Розроблена компанією Facebook і поширюється як відкрите програмне забезпечення, React дозволяє створювати ефективні та масштабовані веб-додатки. Компоненти інтерфейсу оновлюються лише при зміні їх стану, що дозволяє ефективно керувати відображенням даних.

Зазвичай React використовують для розробки таких типів додатків:

- односторінкові додатки (SPA): React дозволяє створювати потужні SPA, які завантажуються один раз, а далі взаємодіють з користувачем, оновлюючи лише необхідні частини сторінки.
- мобільні додатки: за допомогою React Native, можна розробляти нативні мобільні додатки для iOS та Android, використовуючи знання та навички з React.
- веб-розширення: React може бути використаний для створення веб-розширень для різних браузерів, розширюючи функціональність браузера та створюючи користувацькі інструменти та інтерфейси.

Основними перевагами React є:

- ефективність: використання віртуального DOM (Virtual DOM) дозволяє оновлювати тільки необхідні частини сторінки при зміні даних, що підвищує продуктивність.
- JSX: синтаксичне розширення JavaScript, яке дозволяє використовувати HTML-подібний синтаксис для опису компонентів, роблячи код більш зрозумілим.
- компонентна архітектура: компоненти можна повторно використовувати, що прискорює розробку та полегшує супровід коду.
- односторонній потік даних: полегшує управління станом додатка, роблячи його простішим і передбачуваним.

Однак, React має і недоліки:

- висока крива навчання: може бути викликом для початківців через складні концепції.

- висока кількість абстракцій: використання багатьох абстракційних шарів може ускладнювати налагодження та розуміння роботи додатка.
- нестабільність API: часті зміни в API можуть вимагати оновлення коду для збереження сумісності з останніми версіями.

Загалом, React є потужною бібліотекою JavaScript для створення складних інтерфейсів користувача. Він підходить для розробки як невеликих, так і великих проектів, завдяки компонентній архітектурі, високій продуктивності та широкій підтримці спільноти..

## 4.2 Середовище розробки

WebStorm – це інтегроване середовище розробки (IDE), розроблене компанією JetBrains, яке спеціалізується на розробці JavaScript, але також підтримує HTML, CSS та інші веб-технології. Цей інструмент відомий своєю потужною функціональністю та зручним інтерфейсом, що робить його одним з найпопулярніших серед веб-розробників.

WebStorm надає інтелектуальне автодоповнення коду, що допомагає розробникам швидше писати код та уникати помилок. Завдяки вбудованому налагоджувачу, можна легко відстежувати виконання коду, ставити точки зупинки та аналізувати змінні, що значно полегшує виявлення та виправлення помилок. Він також пропонує потужні інструменти для рефакторингу коду, які допомагають безпечно змінювати структуру коду, перейменовувати змінні, методи та класи, не порушуючи логіку програми. Підтримка популярних фреймворків, таких як React, Angular, Vue.js та Node.js, робить розробку з ними більш ефективною.

Швидка навігація по файлах, класах, методах та інших елементах проекту значно прискорює процес розробки. Крім того, WebStorm інтегрується з різними інструментами для збірки проектів, такими як Webpack, Gulp та Grunt, що дозволяє автоматизувати процеси розробки.

Завдяки великій кількості плагінів, які можна додати для розширення функціональності, WebStorm стає ще більш потужним і гнучким інструментом для розробників.

MySQL Workbench – це офіційний інструмент від Oracle для роботи з базами даних MySQL, який надає широкі можливості для адміністрування, проектування та розробки схем баз даних, а також для виконання SQL-запитів.

Однією з основних функцій MySQL Workbench є моделювання даних, що дозволяє створювати візуальні моделі баз даних. Це включає в себе схеми таблиць, зв'язки між ними та інші структури, що значно полегшує процес проектування та документування баз даних.

Редактор SQL-запитів в MySQL Workbench дозволяє писати, редагувати та виконувати SQL-запити. Зручний інтерфейс з автодоповненням, підсвічуванням синтаксису та іншими інструментами значно спрощує написання запитів.

Інструменти для адміністрування баз даних дозволяють керувати користувачами, налаштовувати параметри сервера, моніторити продуктивність та аналізувати логи. Завдяки функціям резервного копіювання та відновлення, можна легко створювати резервні копії баз даних та відновлювати дані у разі потреби.

Можливість написання сценаріїв для автоматизації рутинних завдань адміністрування та розробки робить MySQL Workbench ще більш зручним інструментом. Інтуїтивно зрозумілий інтерфейс та підтримка різних платформ (Windows, macOS, Linux) роблять його доступним для широкого кола користувачів.

## 4.3 Програмна реалізація додатку

### 4.3.1. Програмна реалізація серверної частини

Серверна частина продукту розроблена на платформі Node.js з використанням фреймворку Express.js для створення HTTP-сервера. В якості бази даних використовується MySQL. Аутентифікація реалізована за допомогою бібліотеки Passport.js.

Файл server.js є головним файлом, який відповідає за налаштування і запуск серверу. Він виконує декілька важливих функцій. По-перше, він ініціалізує та налаштовує додаток на основі Express:

```
var app = express();
```

По-друге, він налаштовує з'єднання з базою даних MySQL, забезпечуючи інтеграцію сервера з базою даних. Це відбувається за допомогою mysql.createPool:

```
var connection = mysql.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DATABASE,
  multipleStatements: true,
  timezone: 'utc'
});
```

Далі, він імпортує та використовує модулі для аутентифікації користувачів та роботи з сесіями, що дозволяє забезпечити безпеку та керування користувацькими сесіями. Приклад – використання body-parser для обробки даних, що надходять у запитах:

```
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
```

Нарешті, він визначає маршрути для обробки HTTP-запитів, що дозволяє серверу реагувати на різні запити клієнтів і виконувати відповідні дії.

В маршруті для логіну використовується стратегія local для аутентифікації користувачів. Після успішної аутентифікації, сервер повертає повідомлення про успіх:

```
app.post('/login', passport.authenticate('local'), (req, res) => {
  res.send("success")
});
```

Файл queries.js містить функції для взаємодії з базою даних. Кожна функція виконує певний SQL-запит і повертає результат. Як приклад, розглянемо функцію для додавання нового замовлення:

```
function addNewOrder(orderDetails, clientId, totalCost, date) {
  const queryString = "INSERT into orders VALUES ('', ?, ?, ?, ?, ?)";
  const passedValues = [clientId, date, totalCost, orderDetails.status,
orderDetails.worker];
  return new Promise((resolve, reject) => {
    connection.query(queryString, passedValues, function(error, results) {
      if(error) {
        console.log(error);
      }
      resolve(results.insertId);
    })
  })
}
```

Функція addNewOrder приймає деталі замовлення, ID клієнта, загальну вартість і дату створення. SQL-запит INSERT into orders VALUES (', ?, ?, ?, ?, ?) додає новий запис у таблицю orders. Проміс використання для обробки результату запиту або помилки.

Файл passport-config.js відповідає за конфігурацію модуля Passport, який використовується для аутентифікації користувачів. Файл містить функцію initialize, яка приймає два параметри: connection та passport, і налаштовує стратегію аутентифікації, забезпечуючи правильну взаємодію з базою даних та обробку користувацьких даних:

```
function initialize(connection, passport) {
  const authenticateUser = async (username, password, done) => {
    connection.query('SELECT * FROM accounts WHERE username = ?',
[username], async function(error, results, fields) {
      if (results.length > 0) {
        try {
          if (await bcrypt.compare(password, results[0].password)) {
            const user = {
              id: results[0].id,
              username: results[0].username,
              password: results[0].password
            };
            return done(null, user)
          } else {
            return done(null, false, {
              message: "Wrong password"
            })
          }
        } catch (e) {
          return done(e);
        }
      } else {
        return done(null, false, {
          message: "User not found"
        });
      }
    })
  }
  passport.use(new LocalStrategy({
    usernameField: 'username',
    passwordField: "password"
  }, authenticateUser));
  passport.serializeUser((user, done) => done(null, user.id));
  passport.deserializeUser((id, done) => connection.query("select *
from accounts where id = ?", [id], function(err, results) {
    done(err, results[0]);
  }));
}
```

Інша функція - authenticateUser. Ця функція призначена для аутентифікації користувача. Вона перевіряє, чи існує користувач у базі даних, і порівнює введений пароль із збереженим хешем паролю, забезпечуючи таким чином безпеку входу.

Файл також містить налаштування стратегії LocalStrategy, де встановлюються поля, які використовуються для аутентифікації, а саме username та password. Це дозволяє чітко визначити, які дані повинні бути введені користувачем для входу в систему.

### 4.3.2. Програмна реалізація клієнтської частини

Користувацький інтерфейс веб-додатка побудований на основі бібліотеки React та за допомогою JavaScript. Файлова структура проекту складається з деяких частин що представлено на рисунку 4.1.

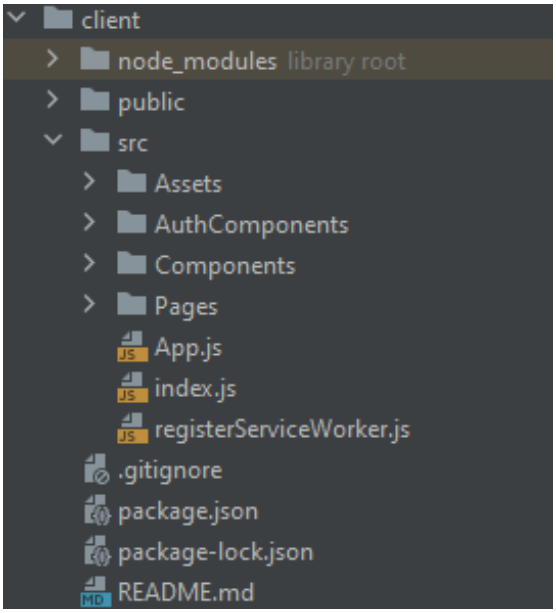



Рисунок 4.1 – Загальна файлова структура клієнтської частини (рисунок виконаний самостійно)

Папка «src» – коренева папка, що містить всі основні компоненти вашого проекту React.

Папка «Pages» – містить компоненти сторінок, які відповідають за різні маршрути в додатку.

Папка «AuthComponents» – містить компоненти, які забезпечують аутентифікацію і авторизацію.

Папка «Components» – містить загальні компоненти, які використовуються на різних сторінках.

Папка «Assets» – містить статичні ресурси, такі як стилі CSS та зображення.

Файл App.js є головним файлом, який визначає структуру та маршрути додатку. Він включає компоненти для обробки аутентифікації та авторизації, а також різні сторінки додатку:

```
function App() {
    const ctx = useContext(AuthLoginInfo);
    console.log(ctx);
    return (
      <BrowserRouter>
        <Sidebar />
        <Routes>
          <Route path='/' exact element={
              <PrivateRoute>
                <Homepage />
              </PrivateRoute>
            } />
…
          <Route path='/login' element={
              <LoginRoute>
                <Login />
              </LoginRoute>
            } />
        </Routes>
      </BrowserRouter>
    );
}
```

Функція useContext(AuthLoginInfo) отримує інформацію про аутентифікацію з контексту.

Routes і Route визначають маршрути для різних сторінок додатку, використовуючи компоненти PrivateRoute, LoginRoute та AdminRoute для обробки аутентифікації та авторизації.

Файл Sidebar.js містить компонент Sidebar, який відповідає за відображення бічної панелі навігації в додатку. Цей компонент включає різні функції та елементи інтерфейсу, що дозволяють користувачам переміщуватися між сторінками додатку, а також забезпечують доступ до профілю користувача та можливості виходу з системи:

```
const logout = () => {
  axios
    .get("http://localhost:5000/logout", {
      withCredentials: true,
    })
    .then((res) => {
      if (res.data === "success") {
        window.location.href = "/login";
      }
    });
};
```

Компонент SidebarSection відповідає за відображення бічної панелі навігації, яка включає логотип та список навігаційних посилань:

```
const SidebarSection = ({ ctx, sidebarCollapse }) => {
  return (
    <div
      className={`Sidebar ${sidebarCollapse ? "SidebarOpen" : "SidebarClosed"}`}
    >
      <div classNameName="SidebarLogoWrap">
        <div classNameName="SidebarLogo">
          <img src={logo} classNameName="logo" alt="" />
        </div>
      </div>
      <ul classNameName="SidebarList">
        {SidebarData.map((val, key) => {
          if (val?.role !== undefined && val?.role !== ctx?.role) {
            return null;
          }
          return (
            <NavLink
              to={val.link}
              key={key}
              classNameName={({ isActive }) =>
                isActive ? "sidebar-active-link" : "sidebar-link"
              }
            >
              <li classNameName="SidebarRow">
                <div classNameName="RowIcon">{val.icon}</div>
                <div classNameName="RowTitle">{val.title}</div>
              </li>
            </NavLink>
          );
        })}
      </ul>
    </div>
  );
};
```

Цей компонент відображає логотип та список посилань, які генеруються на основі даних з SidebarData. Якщо роль користувача не відповідає ролі, вказаній в даних, відповідне посилання не відображається.

Основний компонент Sidebar об'єднує всі попередні компоненти та забезпечує їхню функціональність:

```
function Sidebar() {
  const ctx = useContext(AuthLoginInfo);
  const [sidebarCollapse, setSidebarCollapse] = useState(true);
  const handleSidebarCollapse = () => {
    sidebarCollapse ? setSidebarCollapse(false) :
setSidebarCollapse(true);
  };
  return (
    <div className="SidebarWrapper">
      <NavbarSection
        ctx={ctx}
        handleOnClick={handleSidebarCollapse}
        sidebarCollapse={sidebarCollapse}
      />
      <SidebarSection ctx={ctx} sidebarCollapse={sidebarCollapse} />
    </div>
  );
}
```

У цьому компоненті використовується контекст AuthLoginInfo для отримання інформації про залогіненого користувача. Стан sidebarCollapse визначає, чи бічна панель розгорнута чи згорнута. Функція handleSidebarCollapse змінює цей стан при натисканні на кнопку гамбургер-меню. Компонент Sidebar повертає JSX, що включає NavbarSection та SidebarSection, забезпечуючи їхню взаємодію та відображення відповідно до стану та контексту.

Файл Clients.js реалізує компонент Clients, який відповідає за відображення та керування списком клієнтів у додатку. Цей компонент включає різні функції та елементи інтерфейсу, що дозволяють користувачам шукати, переглядати та додавати нових клієнтів.

Функція handleSearchChange оновлює відфільтровані дані при зміні значення пошуку:

```
const handleSearchChange = (newFilteredData) => {
  setFilteredData(newFilteredData);
};
```

Використання useEffect дозволяє завантажувати дані про клієнтів при завантаженні компонента або при зміні стану newOrderSubmitted і об'єднує інформацію з різних джерел у масив clientsData та filteredData:

```
useEffect(() => {
  setNewOrderSubmitted(false);
  axios
    .get("http://localhost:5000/clients", { withCredentials: true })
    .then((res) => {
      if (res.data != null) {
        setClientsData(
          res.data[0].map((t1) => ({
            ...t1,
            ...res.data[1].find((t2) => t2.client_id === t1.client_id),
          }))
        );
        setFilteredData(
          res.data[0].map((t1) => ({
            ...t1,
            ...res.data[1].find((t2) => t2.client_id === t1.client_id),
          }))
        );
      }
    });
}, [newOrderSubmitted]);
```

Компонент AddClients відповідає за форму додавання нових клієнтів:

```
const AddClients = () => {
  const [clientDetails, setClientDetails] = useState({
    clientName: "",
…
  });
  const addNewOrder = () => {...};
  return (
    <Popup trigger={buttonPopup} setTrigger={setButtonPopup}>
      <div className="popupWrap">
        <div className="productsSummary">
          <h3 className="productSummaryLeft">Додати нового клієнта</h3>
        </div>
        <div className="addNewOrderWrap">
          <div className="addNewOrderForm">
            <div className="orderDetails">
              <div className="input-group">
                <input
                  type="text"
                  placeholder="П.І.Б клієнта"
                  className="orderDetailsInput orderDetailsInputHalf"
                  value={clientDetails.clientName}
                  onChange={(e) =>
                    setClientDetails({clientName: e.target.value...})
                  }
                  required="required"
                />
```

```
                <input
                  type="text"
                  placeholder="Номер телефону"
                  className="orderDetailsInput orderDetailsInputHalf"
                  value={clientDetails.phone}
                  onChange={(e) =>
                    setClientDetails({phone: e.target.value...})
                  }
                  required="required"
                />
              </div>
…
            </div>
          </div>
        </div>
        <div className="submitWrap"...>
      </div>
    </Popup>
  );
};
```

Файл UsersSetting.js містить компонент React, який використовується для керування налаштуваннями користувачів у додатку. Його основне призначення полягає у відображенні списку користувачів, наданні можливості додавання нових користувачів та видалення існуючих.

Функція formatIsoDate використовується для форматування дати з ISO формату до простішого формату, який містить лише дату без часу:

```
function formatIsoDate(date) {
  return date.split("T")[0];
}
```

Компонент UsersSetting є основним компонентом, який використовує декілька хуків для керування станом: usersData – для зберігання даних користувачів, usersUpdated – для відслідковування оновлень даних користувачів, popup – для керування спливаючим вікном видалення користувача, newUserPopup – для керування спливаючим вікном додавання нового користувача:

```
function UsersSetting() {
  const [usersData, setUsersData] = useState();
  const [usersUpdated, setUsersUpdated] = useState(false);
  const [popup, setPopup] = useState({
    show: false,
    id: null,
  });
  const [newUserPopup, setNewUserPopup] = useState(false);
…
```

Хук useEffect для завантаження даних користувачів з сервера при першому завантаженні компонента або при зміні стану usersUpdated:

```
useEffect(() => {
  axios
    .get(`http://localhost:5000/getusers`, { withCredentials: true })
    .then((res) => {
      if (res.data != null) {
        setUsersUpdated(false);
        setUsersData(res.data);
      }
    });
}, [usersUpdated]);
```

Цей код виконує HTTP-запит до сервера для отримання списку користувачів і оновлює стан usersData.

Компонент AddNewUserSection відображає кнопку для додавання нового користувача, яка відкриває спливаюче вікно для введення даних нового користувача:

```
const AddNewUserSection = () => {
  return (
    <div className="addNewUserWrap">
      <span className="addNewUserText">Додати нового користувача</span>
      <button
        className="addOrder"
        onClick={() => {
          setNewUserPopup(true);
        }}
      >
        <AddCircleOutlineRoundedIcon /> Додати
      </button>
    </div>
  );
};
```

### 4.3.3. Програмна реалізація бази даних

У даному веб-додатку використовується реляційна база даних MySQL для зберігання та управління даними. База даних складається з наступних таблиць: accounts, calendar, clients, orders та products. Кожна таблиця має свою структуру і взаємозв'язки з іншими таблицями.

Для створення бази даних використовується команда CREATE DATABASE:

```
CREATE DATABASE `db-print-order`;
USE `db-print-order`;
```

Для створення таблиць використовується команда CREATE TABLE:

```
CREATE TABLE `accounts` (
  `id` int(11) NOT NULL,
  `username` text DEFAULT NULL,
  `password` text DEFAULT NULL,
  `role` text DEFAULT NULL,
  `dateCreated` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Заповнення таблиці відбувається за допомогою команди INSERT INTO:

```
INSERT INTO `accounts` (`id`, `username`, `password`, `role`, `dateCreated`) VALUES
(1, 'admin', '$2b$10$J1JTnk4KtLylMznNjX.xyP77EY/Gq1JpccSnr3qa', 'admin', '2022-10-25');
```

Для додавання первинного ключа та автоінкремента використовуються команди ADD PRIMARY KEY та AUTO_INCREMENT:

```
ALTER TABLE `accounts`
  ADD PRIMARY KEY (`id`);

ALTER TABLE `accounts`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;
```

## 4.4 Розгортання та запуск системи

Після копіювання вихідного коду додатку, потрібно встановити залежності в кореневій папці та папці client за допомогою команди, яку необхідно ввести в терміналі IDE:

```
npm install
```

Після цього в файлі .env, який використовується для зберігання конфіденційних параметрів проекту або програми, таких як паролі, ключі API, та інші налаштування середовища, треба вказати всі необхідні дані для підключення до бази даних, заповнивши наступні змінні оточення:

```
DB_HOST=
DB_PORT=
DB_NAME=
DB_USER=
DB_PASSWORD=
```

На наступному кроці необхідно встановити програмного забезпечення MySQL Workbench і MySQLServer та імпортувати шаблон БД – db-print-order.sql.

Далі в терміналі IDE запустити систему командою:

```
npm run dev | npm run build
```

У випадку, якщо все налаштовано правильно, система стане доступною через браузер за посиланням http://localhost:3000/login.

## 4.5 Опис інтерфейсу користувача

Вхід в систему для зареєстрованих користувачів відбувається за допомогою вводу логіну та паролю. На рисунку 4.2 представлено вікно авторизації.

Якщо користувач не пам’ятає пароль, він має змогу відновити його за допомогою пошти.

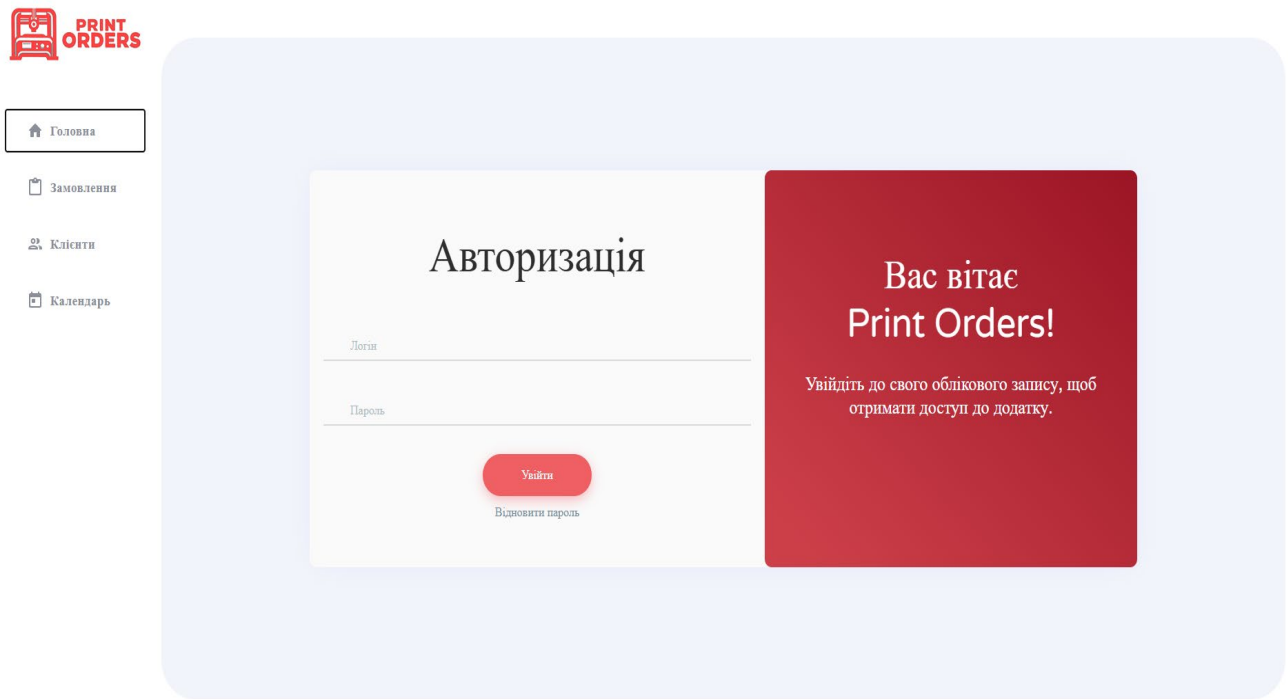



Рисунок 4.2 – Вікно авторизації (рисунок виконаний самостійно)

Після авторизації користувачу стає доступним інтерфейс системи, і він потрапляє на головну сторінку (див. рис. 4.3) де має можливість керувати замовленнями, клієнтами та бачити статистики.

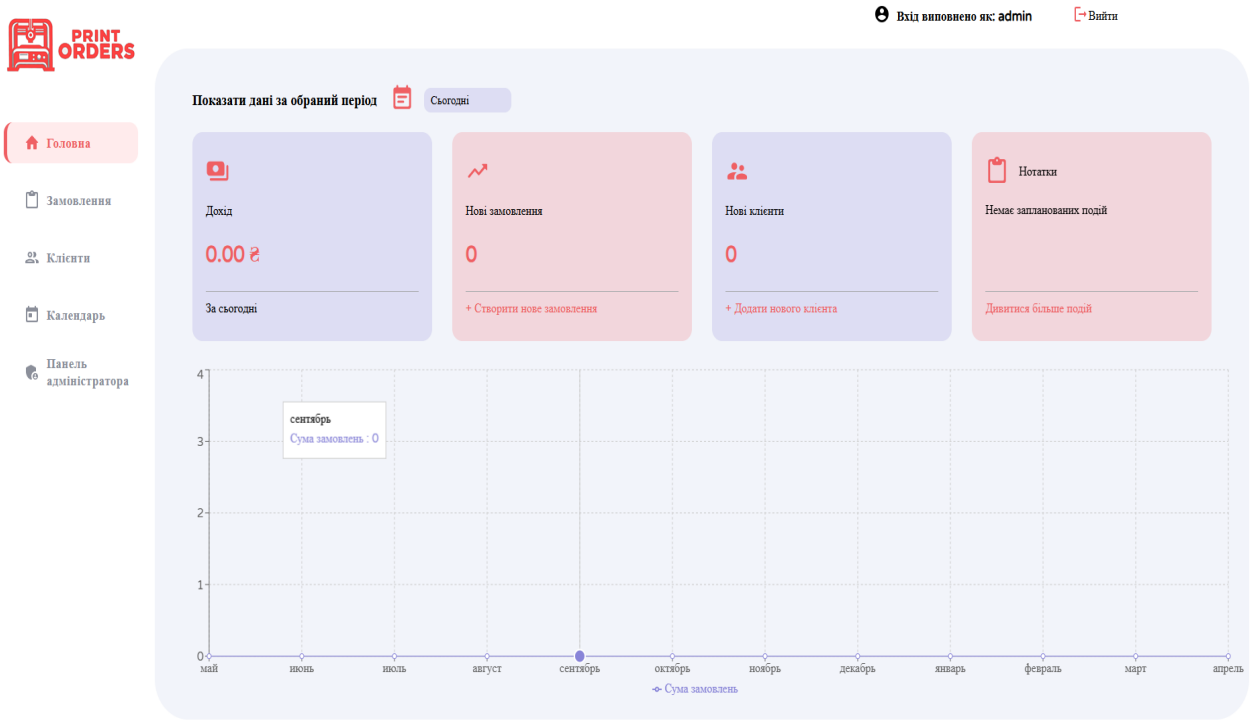



Рисунок 4.3 – Головна сторінка додатку (рисунок виконаний самостійно)

На рисунку 4.4 можна побачити вікно сторінки «Замовлення», куди користувач повинен перейти натиснувши відповідну кнопку головного меню, , щоб додати нове замовлення. Щоб розпочати створення нового замовлення необхідно натиснути кнопку «Додати».

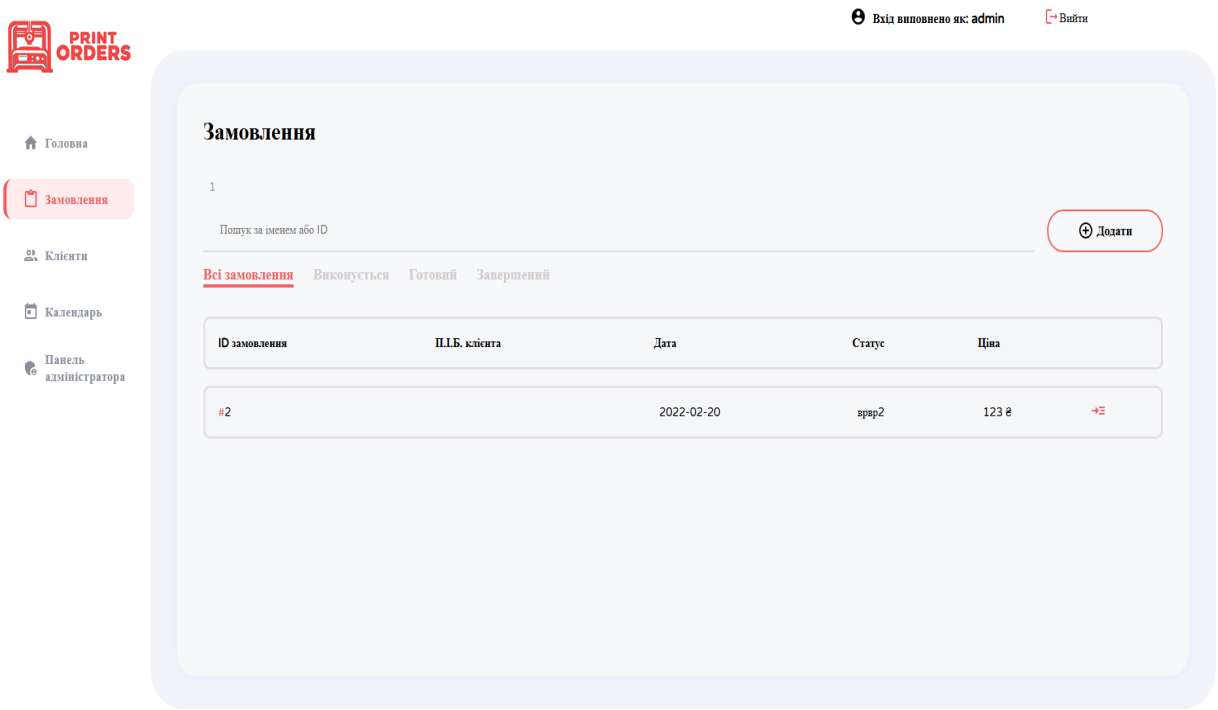



Рисунок 4.4 – Вікно сторінки «Замовлення» (рисунок виконаний самостійно)

В результаті з’явиться форма додавання нового замовлення (див. рис. 4.5), де необхідно перемикачем вибрати – новий клієнт, чи ні. Якщо новий – необхідно ввести інформацію про клієнта, заповнюючи відповідні поля форми. Якщо ж клієнт не новий – треба буде його знайти в списку клієнтів та обрати. Наступною дією користувача буде додавання виробів до замовлення – потрібно ввести назву виробу, його кількість та ціну. В результаті додаток підрахує ціну кожного з виробів і загальну суму замовлення. Після цього залишиться тільки натиснути кнопку «Додати», щоб зберегти замовлення.

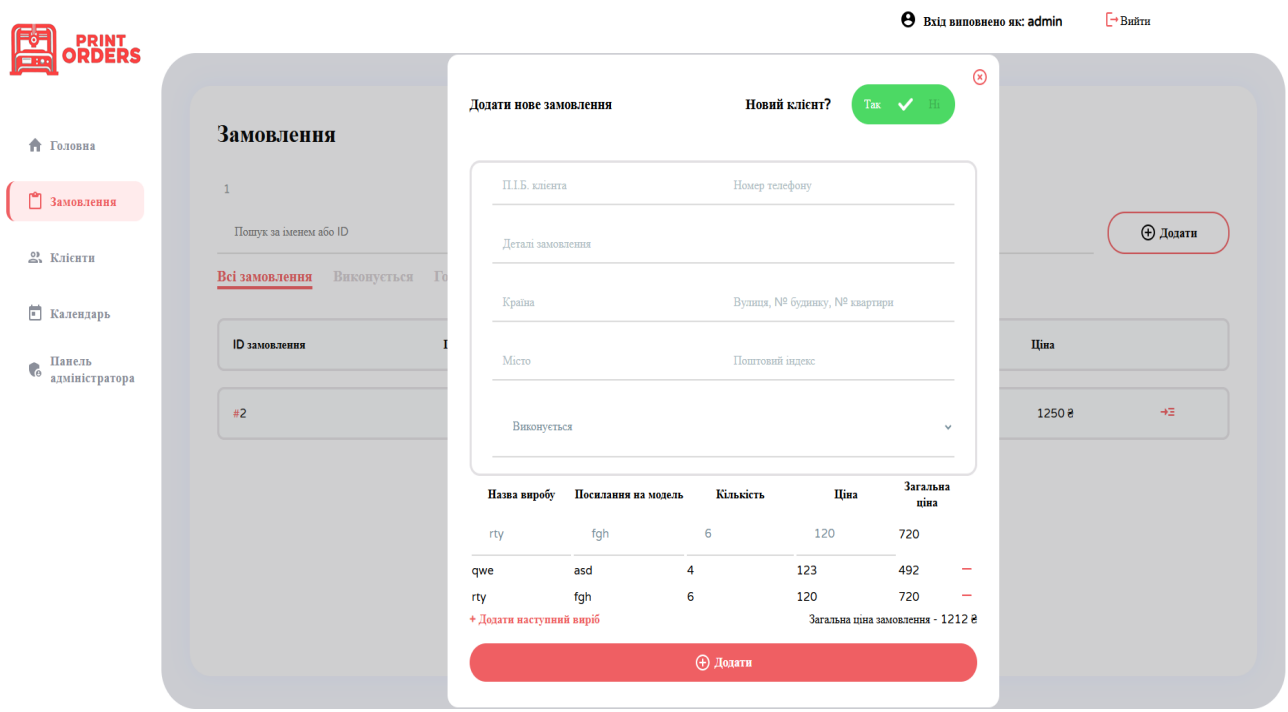



Рисунок 4.5 – Форма додавання нового замовлення (рисунок виконаний самостійно)

Додавання нового клієнта відбувається аналогічним способом, тому докладний опис цього процесу описуватися не буде.

Якщо користувач увійшов в додаток, як адміністратор, то в головному меню з’явиться розділ «Панель адміністратора», де у користувача є можливість управляти обліковими записами звичайних користувачів (див. рис. 4.6). На панелі адміністратора всі облікові записи розділені на 2 групи: адміністратори і звичайні користувачі. На екрані відображається ім’я користувача та дата створення його облікового запису, а також біля кожного запису є кнопка видалення.

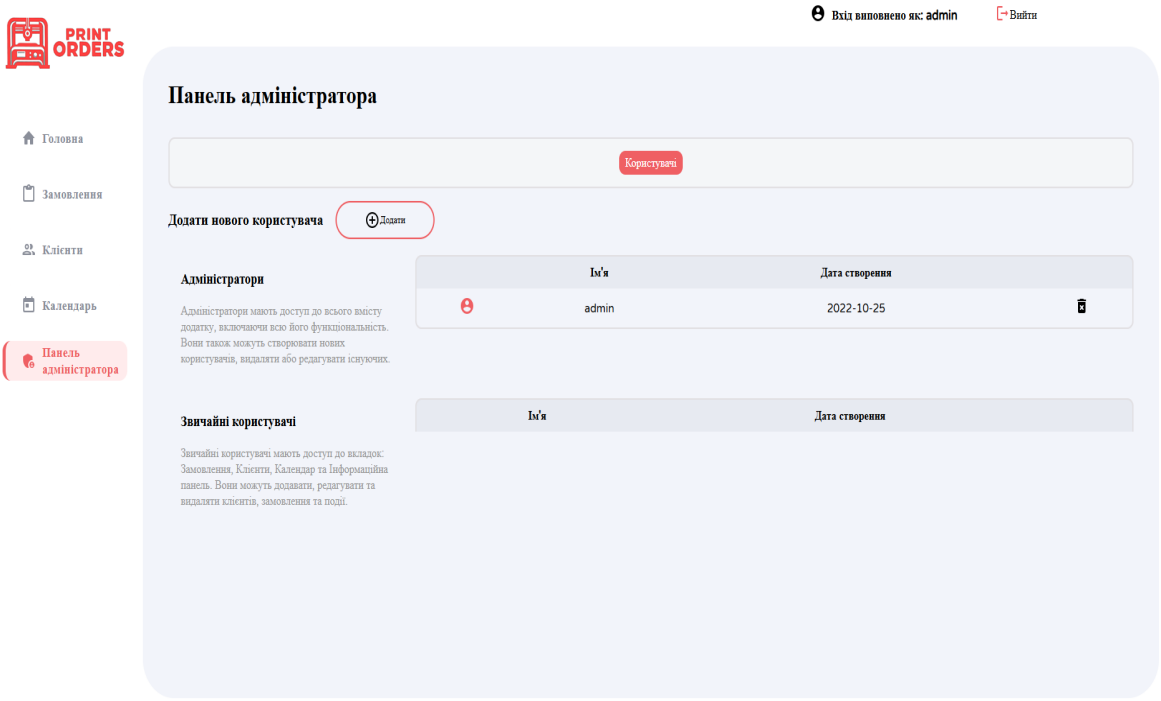



Рисунок 4.6 – Панель адміністратора (рисунок виконаний самостійно)

# 5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Забезпечення якості (QA) є невід'ємною частиною процесу розробки ПЗ, яка спрямована на надання замовникам найкращого продукту або послуги.

Функціональне тестування є важливою складовою забезпечення якості, при якій перевіряється відповідність програмного забезпечення функціональним вимогам та специфікаціям. Цей тип тестування передбачає введення певних вхідних даних та перевірку відповідних вихідних результатів, щоб переконатися, що система працює належним чином.

В результаті тестування була перевірена функціональність додатку. Тести були проведені в таких браузерах: Google Chrome ver.126.0.6478.127, Microsoft Edge ver.126.0.2592.87. Деякі з проведених тестів показані в таблицях 5.1 і 5.2.

Таблиця 5.1 – Тест-кейс №1 (таблиця виконана самостійно)

| Інформація про тест-кейс | | | |
|---|---|---|---|
| Ідентифікатор тесту: | Тест-кейс №1 | | |
| Опис функції: | Створення фінансової та волонтерської заявки | | |
| Власник тесту: | Пивовар Євген Валентинович | | |
| Дата створення: | 05.07.2024 | | |
| Мета тесту: | Перевірити коректність реалізації авторизації користувача | | |
| Авторизуватися в веб-додатку | | | |
| № | Опис випадку | Очікуваний результат | Висновок |
| 1 | Відкрити веб-застосунок | Користувач має доступ до сторінки авторизації | Пройдено |
| 2 | Ввести облікові дані: "Email" та "Пароль" | Введені дані відображаються коректно | Пройдено |
| 3 | Натиснути кнопку "Увійти" | Користувач авторизований, перенаправлено на головну сторінку | Пройдено |
| 4 | Ввести некоректні облікові дані | Відображається повідомлення про помилку | Пройдено |
| 5 | Натиснути кнопку "Забули пароль? " | Перенаправлено на сторінку відновлення пароля | Пройдено |
| 6 | Ввести зареєстрований email у формі відновлення пароля | Відправлено пароль на вказаний email | Пройдено |
| Результати тестування | | | |
| Тестувальник: Пивовар Є. В. | Дата прогону тесту: 05.07.2024 | Результат тесту (P/F/B): ПРОЙДЕНО (P) | |

Таблиця 5.2 – Тест-кейс №2 (таблиця виконана самостійно)

| Інформація про тест-кейс | | | |
|---|---|---|---|
| Ідентифікатор тесту: | Тест-кейс №2 | | |
| Опис функції: | Додавання нового замовлення | | |
| Власник тесту: | Пивовар Євген Валентинович | | |
| Дата створення: | 05.07.2024 | | |
| Мета тесту: | Перевірити коректність реалізації створення нового замовлення користувачем | | |
| Передумова | | | |
| № | Опис випадку | Очікуваний результат | Висновок |
| 1 | Відкрити веб-застосунок | Користувач має доступ до сайту, який відкритий | Пройдено |
| 2 | Має обліковий запис та хоче авторизуватися | Перенаправлено на головну сторінку | Пройдено |
| Додати нове замовлення | | | |
| № | Опис випадку | Очікуваний результат | Висновок |
| 1 | Перейти на сторінку замовлення | Відбувається перехід на сторінку замовлення | Пройдено |
| 2 | Натиснути кнопку «Додати» | З’являється форма для створення замовлення | Пройдено |
| 3 | Перевести перемикач «Новий клієнт» в положення «Так» | Перемикач зеленого кольору, всі поля для заповнення даних про нового клієнта доступні для заповнення | Пройдено |
| 4 | Заповнити обов’язкові поля «П.І.Б. клієнта» та «Номер телефону» | Введені дані відображаються коректно | Пройдено |
| 5 | Заповнити поля «Деталі замовлення», «Країна», «Вулиця, № будинку, № квартири» «Місто» та «Поштовий індекс» | Введені дані відображаються коректно | Пройдено |
| 6 | Обрати статус замовлення - «Виконується» | Обрано статус «Виконується» | Пройдено |
| 7 | Заповнити обов’язкові поля «Назва виробу», «Кількість» та «Ціна» | Кількість символів у полях вказана правильно, загальна ціна та загальна ціна замовлення рахуються правильно | Пройдено |
| 8 | Натиснути кнопку «Додати» | В списку з’являється створене замовлення | Пройдено |

Кінець таблиця 5.2

| | | | |
|---|---|---|---|
| 9 | Заповнити обов’язкове поле форми некоректно | Відображається повідомлення про некоректне заповнення поля | Пройдено |
| 10 | Перевести перемикач «Новий клієнт» в положення «Ні» | Перемикач червоного кольору, всі поля для заповнення даних про нового клієнта недоступні для заповнення, з’являється поле пошуку клієнтів | Пройдено |
| 11 | Ввести в поле пошуку декілька символів з П.І.Б існуючого клієнта | Відображається список клієнтів, в П.І.Б яких є введені символи | Пройдено |
| 12 | Обрати клієнта зі списку | Всі поля для заповнення даних заповнені і мають заборону на редагування | Пройдено |
| Результати тестування | | | |
| Тестувальник: Пивовар Є. В. | Дата прогону тесту: 05.07.2024 | Результат тесту (P/F/B): ПРОЙДЕНО (P) | |

В результаті тестування було виявлено декілька багів, таких як:

– при введення некоректного пароля при авторизації система не відображала повідомлення про неправильний пароль;
– не зберігаються зміни у профілі користувача при їх редагуванні.

Тест на продуктивність проводиться для оцінки швидкості, стабільності та масштабованості веб-додатку під різним навантаженням. Тест дозволить визначити, як система поводиться під час високих навантажень і чи здатна вона обробляти запити користувачів у прийнятний час без помилок або затримок.

Для проведення тесту на продуктивність буде використано інструмент Apache JMeter ver.5.6.1. Цей інструмент дозволяє моделювати різні сценарії навантаження та збирати статистику про час відгуку, кількість оброблених запитів, помилки та інші показники продуктивності.

На рисунку 5.1 наведені результати тестів для таких сценаріїв, як авторизація користувача та додавання нового замовлення з такими параметрами:

– кількість користувачів: 500;

– тривалість тесту: 5 хвилин;

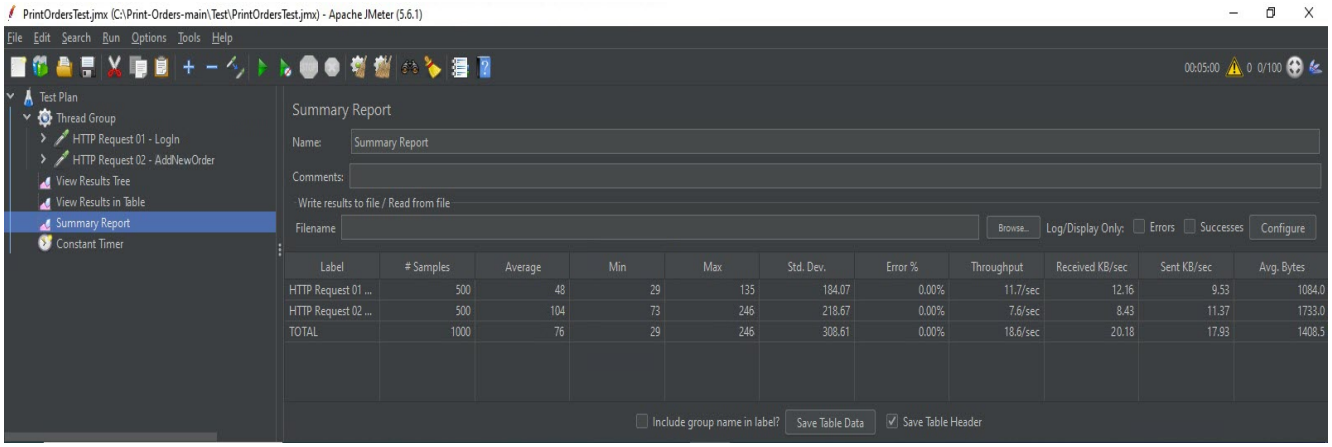



Рисунок 5.1 – Результати тестів на продуктивність (рисунок виконаний самостійно)

Результати тесту на продуктивність показали, що веб-додаток здатний обробляти запити авторизації та додавання замовлень у прийнятний час під навантаженням.

# ВИСНОВКИ

Кваліфікаційна робота присвячена розробці веб-додатку для власників 3D-принтерів, які спеціалізуються на виготовленні виробів на замовлення. Тема є актуальною у зв'язку з зростанням популярності 3D-технологій і потребою в управлінні замовленнями та ресурсами.

Був проведений аналіз існуючих аналогів програмного забезпечення для вирішення поставлених задач. Аналіз показав, що існуючі системи вирішують проблему не у повному обсязі. Деякі з них мають громіздкий інтерфейс та мають високі апаратні вимоги до пристроїв користувачів або високу ціну.

На основі проведеного аналізу були сформульовані функціональні і нефункціональні вимоги до розроблюваного веб-додатку.

Результатом роботи стала програмна система для обліку замовлень з 3D-друку, яка розроблена у вигляді веб-додатку.

У процесі розробки було реалізовано усі необхідні функції веб-додатку. Обрано оптимальні засоби розробки та підходи до проектування. Для реалізації функціоналу додатку використано сучасні технології веб-розробки, зокрема, клієнтська частина реалізована з використанням фреймворка React.js, а серверна частина — з використанням Node.js та Express.js. Для зберігання даних використовується реляційна база даних MySQL.

Додаток пропонує конкретний спектр функцій, що включає створення, видалення та редагування замовлень, управління клієнтською базою даних та базою обладнання та матеріалів, а також адміністрування користувачів.

Систему було налаштовано, протестовано та випробувано, що підтвердила правильна робота програмного забезпечення. Система ефективна та вирішує всі поставлені завдання.

Потенційними користувачами додатку є власники 3D-принтерів, або невеликого бізнесу, що займаються виготовленням різних виробів під замовлення і стикаються з необхідністю у впорядкуванні замовлень, управлінні клієнтською базою та контролі якості продукції.

Завдяки виконанню цих завдань створено ефективний інструмент для управління процесами виготовлення виробів на замовлення за допомогою 3D-принтерів, що сприятиме підвищенню продуктивності та якості обслуговування клієнтів. Програмне забезпечення може працювати на будь-якій операційній системі ПК за умови наявності браузера, який підтримує останні веб-стандарти, та постійного доступу до інтернету.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Creatio URL: https://www.creatio.com/ (дата звернення: 25.05.2024);

2. Salesforce Sales Cloud. – URL: https://www.salesforce.com/ (дата звернення: 25.05.2024);

3. Keap URL: https://keap.com/ (дата звернення: 25.05.2024);

4. MySQL documentation. – URL: https://www.mysql.com/ (дата звернення: 25.05.2024);

5. JavaScript documentation [Електронний ресурс] – URL: https://developer.mozilla.org/ru/docs/Web/JavaScript (дата звернення:5.06.2024);

6. Node.js documentation [Електронний ресурс] – URL: https://nodejs.org/en/docs/ (дата звернення: 6.06.2024);

7. Express.js documentation [Електронний ресурс] – URL: https://expressjs.com/ (дата звернення: 5.06.2024);

8. React.js documentation [Електронний ресурс] – URL: https://reactjs.org/docs/getting-started.html (дата звернення: 6.06.2024);

9. Nebrass Lamouchi – Introduction to the Monolithic Architectur [Електронний ресурс] – URL: https://www.academia.edu/67859996/Introduction_to_the_Monolithic_Architecture (дата звернення: 5.06.2024);

10. Mike Cantelon, Marc Harter, T.J. Holowaychuk, and Nathan Rajlich. Node.js in Action. Second Edition : Manning Publications, 2017. 392 p.;

# ДОДАТОК А

## Фрагмент коду програмного коду

Код компонента server.js:

```
require("dotenv").config();

var mysql = require('mysql');
var express = require('express');
var session = require('express-session');
const flash = require('express-flash');
var app = express();
var bodyParser = require('body-parser');
var path = require('path');
var bcrypt = require('bcrypt');
const passport = require("passport");
const cors = require("cors");

const {
  getAllDataFromTarget,
  getOrdersById,
  getClientById,
  getOrdersByClientId,
  getProductsBySingleOrderId,
  getProductsByMultipleOrderId,
  deleteProductsById,
  addMultipleProducts,
  updateOrderById,
  updateClientById,
  getAllClientsWithOrdersCount,
  addNewClient,
  getDasboardData,
  addNewEvent,
  addNewOrder,
  deleteUserById,
  getUsersForAdminPanel,
  createNewUser
} = require("./queries.js");

var connection = mysql.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DATABASE,
  multipleStatements: true,
  timezone: 'utc'
});

const corsOptions = {
  origin: "http://localhost:3000",
  credentials: true,
  optionSuccessStatus: 200
};

const initializePassport = require('./passport-config');
initializePassport(connection, passport);
```

```
app.use(cors(corsOptions));
app.use(flash());
app.use(session({
  secret: 'secret',
  resave: false,
  saveUninitialized: false
}));
app.use(passport.initialize());
app.use(passport.session());
app.use(bodyParser.urlencoded({
  extended: true
}));
app.use(bodyParser.json());

app.post('/login', passport.authenticate('local'), (req, res) => {
  res.send("success")
});

app.get('/logout', (req, res) => {
  req.logout();
  res.send("success");
});

app.get('/user', (req, res) => {
  res.send(req.user)
});

app.get("/orders", async (req, res) => {
  const queryAllOrders = await getAllDataFromTarget("orders");
  const queryAllClients = await getAllDataFromTarget("clients");
  const mergedQueries = queryAllOrders.map(order => ({
    ...order,
    ...queryAllClients.find(client => client.client_id === order.client_id)
  }));
  res.send(mergedQueries)
});

app.get("/order_by_id", async (req, res) => {
  const orderId = req.query.id;
  const queryOrderById = await getOrdersById(orderId);
  const queryProductsByOrderID = await getProductsBySingleOrderId(orderId);
  const queryClientById = await getClientById(queryOrderById.client_id);

  Promise.all([queryOrderById, queryProductsByOrderID,
queryClientById]).then((results) => {
    res.send(results)
  })
});

app.post("/neworder", async (req, res) => {
  let newOrderData = req.body.clientDetails;
  // if new client was selected
  let isNewClient = req.body.isNewClient;
  // if existing client was selected
  let oldClientId = req.body.oldClientId;
  let dataDetails = newOrderData;
```

```
  let productsData = newOrderData.products;
  let totalCostOfProducts = productsData.reduce(
    (accumulator, product) =>
      accumulator + (product.itemPrice * product.amount || 0),
    0
  );
  let currentDate = new Date();

  // if new client is being created
  if(isNewClient) {
    const queryAddNewClient = await addNewClient(dataDetails);
    const queryAddNewOrder = await addNewOrder(dataDetails,
queryAddNewClient, totalCostOfProducts, currentDate);
    const queryAddMultipleProducts = await
addMultipleProducts(queryAddNewOrder, productsData);
    Promise.all([queryAddNewClient, queryAddNewOrder,
queryAddMultipleProducts]).then(() => {
      res.send("success");
    })
  } else {
    const queryAddNewOrder = await addNewOrder(dataDetails, oldClientId,
totalCostOfProducts, currentDate);
    const queryAddMultipleProducts = await
addMultipleProducts(queryAddNewOrder, productsData);
    Promise.all([queryAddNewOrder, queryAddMultipleProducts]).then(() => {
      res.send("success");
    })
  }
})

app.post("/updateorder", async (req, res) => {
  let orderId = req.body.orderId;
  let clientData = req.body.clientDetails.client;
  let orderData = req.body.clientDetails.order;
  let productsData = req.body.clientDetails.products;
  let deletedIds = req.body.deletedItems.ids;
  let totalCostOfProducts = productsData.reduce(
    (accumulator, product) =>
      accumulator + (product.itemPrice * product.amount || 0),
    0
  );
  let currentDate = new Date();

  const queryUpdateClientById = await updateClientById(clientData,
orderData.client_id);
  const queryUpdateOrderById = await updateOrderById(totalCostOfProducts,
orderData.status, orderId);
  const queryAddMultipleProducts = await addMultipleProducts(orderId,
productsData);

  // if user is deleteing some products from order
  if(deletedIds.length > 0) {
    const queryDeleteProductsById = await deleteProductsById(deletedIds);
    Promise.all([queryUpdateClientById, queryUpdateOrderById,
queryAddMultipleProducts, queryDeleteProductsById]).then(() => {
      res.send("success");
    });
```

```
  } else {
    Promise.all([queryUpdateClientById, queryUpdateOrderById,
queryAddMultipleProducts]).then(() => {
      res.send("success");
    });
  }
})

app.get("/clients", async (req, res) => {
  const queryAllClientsWithOrdersCount = await
getAllClientsWithOrdersCount();
  Promise.resolve(queryAllClientsWithOrdersCount).then((results) => {
    res.send(results);
  })
})

app.post("/newclient", async (req, res) => {
  const clientDetails = req.body.clientDetails;
  const queryAddingNewClient = await addNewClient(clientDetails);
  Promise.resolve(queryAddingNewClient).then(() => {
    res.send("success");
  })
})

app.get("/client_by_id", async (req, res) => {
  const clientId = req.query.id;

  const queryClientById = await getClientById(clientId);
  const queryOrdersByClientId = await getOrdersByClientId(clientId);
  const queryProductsByOrderId = await
getProductsByMultipleOrderId(queryOrdersByClientId);

  Promise.all([queryClientById, queryOrdersByClientId,
queryProductsByOrderId]).then((results) => {
    res.send(results);
  });
});

app.get("/dashboard_data", async (req,res) => {
  const queryDashboardData = await getDasboardData();
  Promise.resolve(queryDashboardData).then((results) => {
    res.send(results);
  })
})

app.post("/newevent", async (req, res) => {
  const eventData = req.body.eventData;
  const queryAddNewEvent = await addNewEvent(eventData);
  Promise.resolve(queryAddNewEvent).then(() => {
    res.send("success");
  })
})

app.get("/events", async (req, res) => {
  const queryAllEvents = await getAllDataFromTarget("calendar");
  Promise.resolve(queryAllEvents).then((results) => {
    res.send(results);
```

```
  })
})

app.get("/getusers", async (req, res) => {
  const queryGetUsersForAdminPanel = await getUsersForAdminPanel();
  Promise.resolve(queryGetUsersForAdminPanel).then((results) => {
    res.send(results);
  })
})

app.post("/deleteuser", async (req, res) => {
  let userId = req.body.userId;
  const queryDeleteUser = await deleteUserById(userId);
  Promise.resolve(queryDeleteUser).then(() => {
    res.send("success");
  })
});

app.post("/newuser", async (req, res) => {
  const userDetails = req.body.userDetails;
  const hashedPassword = await bcrypt.hash(userDetails.password, 10);
  const queryCreateNewUser = await createNewUser(userDetails,
hashedPassword);
  Promise.resolve(queryCreateNewUser).then(() => {
    res.send("success");
  })
})

const port = 5000;
app.listen(port, () => `Server running on port ${port}`);
```

Код компонента passport-config.js:

```
const LocalStrategy = require('passport-local').Strategy;
const bcrypt = require("bcrypt");
const mysql = require('mysql');

function initialize(connection, passport) {
  const authenticateUser = async (username, password, done) => {
    connection.query('SELECT * FROM accounts WHERE username = ?',
[username], async function(error, results, fields) {
      //console.log(await bcrypt.hash('112233gg', 10));
      if (results.length > 0) {
        try {
          if (await bcrypt.compare(password, results[0].password)) {
            const user = {
              id: results[0].id,
              username: results[0].username,
              password: results[0].password
            };
            return done(null, user)
          } else {
            return done(null, false, {
              message: "Wrong password"
            })
          }
        } catch (e) {
          return done(e);
```

```
        }
      } else {
        return done(null, false, {
          message: "User not found"
        });
      }
    })
  }
  passport.use(new LocalStrategy({
    usernameField: 'username',
    passwordField: "password"
  }, authenticateUser));
  passport.serializeUser((user, done) => done(null, user.id));
  passport.deserializeUser((id, done) => connection.query("select * from
accounts where id = ?", [id], function(err, results) {
    done(err, results[0]);
  }));
}

module.exports = initialize;
```

Код компонента App.js:

```
import React, { useContext } from 'react';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Homepage from './Pages/Homepage';
import Login from './Pages/Login';
import Orders from './Pages/Orders';
import OrderPage from './Pages/OrderPage';
import Clients from './Pages/Clients';
import ClientPage from './Pages/ClientPage';
import CalendarEvents from './Pages/CalendarEvents';
import AdminPanel from './Pages/AdminPanel';
import PrivateRoute  from './AuthComponents/PrivateRoute';
import LoginRoute  from './AuthComponents/LoginRoute';
import AdminRoute  from './AuthComponents/AdminRoute';
import Sidebar from './Components/Sidebar';
import { AuthLoginInfo }  from './AuthComponents/AuthLogin';

function App() {
    const ctx = useContext(AuthLoginInfo);
    console.log(ctx)
    return (
      <BrowserRouter>
        <Sidebar>
          </Sidebar>
            <Routes>
              <Route path='/' exact element={
                  <PrivateRoute>
                    <Homepage />
                  </PrivateRoute>
                } />
              <Route path='/orders' element={
                    <PrivateRoute>
                      <Orders />
                    </PrivateRoute>
                  } />
                <Route path='/orders/:orderId' element={
```

```
                    <PrivateRoute>
                      <OrderPage />
                    </PrivateRoute>
                  } />
                <Route path='/clients' element={
                    <PrivateRoute>
                      <Clients />
                    </PrivateRoute>
                  } />
                <Route path='/clients/:clientId' element={
                    <PrivateRoute>
                      <ClientPage />
                    </PrivateRoute>
                  } />
                <Route path='/calendar' element={
                    <PrivateRoute>
                      <CalendarEvents />
                    </PrivateRoute>
                  } />
                <Route path='/adminPannel' element={
                    <AdminRoute>
                      <AdminPanel />
                    </AdminRoute>
                  } />
              <Route path='/login' element={
                  <LoginRoute>
                    <Login />
                  </LoginRoute>
                } />
            </Routes>
      </BrowserRouter>
    );
}

export default App;
```

Код компонента index.js:

```
import React from 'react';
import ReactDOM from 'react-dom';
import axios from 'axios';
import './Assets/Styles/index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
import { AuthLogin } from './AuthComponents/AuthLogin';

ReactDOM.render(
  <AuthLogin>
    <App />
  </AuthLogin>
    ,document.getElementById('root'));
registerServiceWorker();
```

Код компонента AdminPanel.js:

```
import React, { useState } from "react";
import "./Styles/adminPanel.css";
import UsersSetting from
```

```
"./../Components/AdminPanelComponents/UsersSetting";
import ConfigureSetting from
"./../Components/AdminPanelComponents/ConfigureSetting";

function AdminPanel() {
  const [selectedSetting, setSelectedSetting] = useState(0);
  const componentsMap = {
    usersSetting: UsersSetting,
  };
  const adminNavData = [
    {
      id: 0,
      title: "Користувачі",
      component: "usersSetting",
    },
  ];

  const CalendarWrap = (props) => {
    return (
      <div className="adminPanelWrap">
        <div className="adminPanelHeader">
          <h1>Панель адміністратора</h1>
        </div>
        {props.children}
      </div>
    );
  };

  const AdminPanelNav = () => {
    return (
      <div className="adminNav">
        <ul className="adminNavList">
          {adminNavData.map((val) => {
            return (
              <li
                key={val.id}
                className={`adminNavItem ${
                  selectedSetting === val.id ? "active-setting" : ""
                }`}
                onClick={() => setSelectedSetting(val.id)}
              >
                {val.title}
              </li>
            );
          })}
        </ul>
      </div>
    );
  };

  const ContentWrap = (props) => {
    return (
      <div className="contentWrap">
        {adminNavData.map((val) => {
          if (val.id === selectedSetting) {
            const Component = componentsMap[val.component];
            return <Component key={val.id} />;
```

```
          }
        })}
      </div>
    );
  };

  return (
    <div className="bodyWrap">
      <CalendarWrap>
        <AdminPanelNav />
        <ContentWrap />
      </CalendarWrap>
    </div>
  );
}

export default AdminPanel;
```

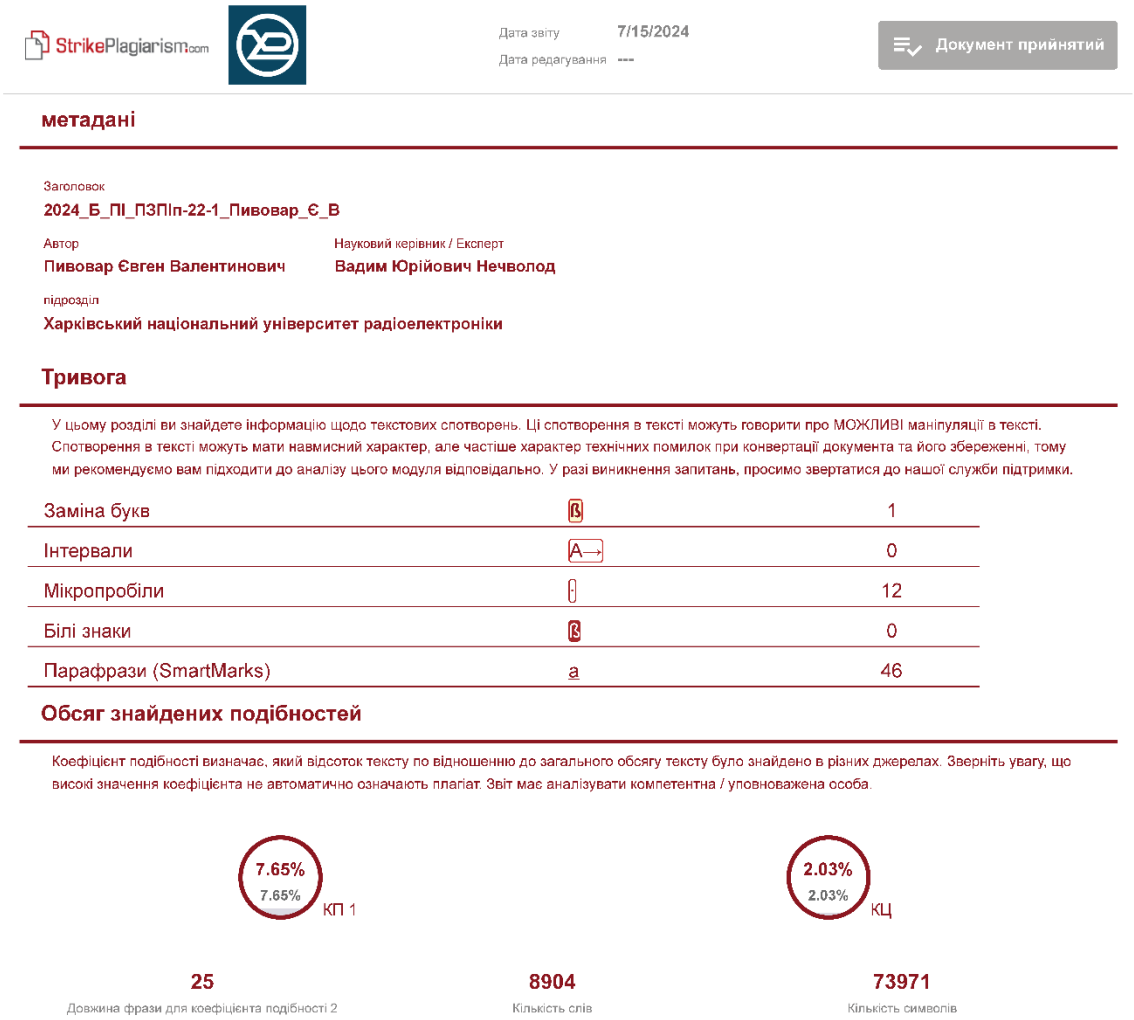
Код компонента SearchBar.js:

```
import react, { useState, useEffect } from "react";

import React from "react";

function SearchBar({ data, handleSearchChange, dataType, status }) {
  const [searchInput, setSearchInput] = useState("");

  const clientsDataFiltered = () => {
    const dataFiltered = data
      ?.filter((val) => {
        if (
          [val.client.toLowerCase(), val.client_id + ""].some((r) =>
            r.includes(searchInput.toLowerCase())
          )
        ) {
          return val;
        }
      })
      .reverse();
    handleSearchChange(dataFiltered);
  }

  const ordersDataFiltered = () => {
    const dataFiltered = data
      ?.filter((val) => {
        if (
          val.status.includes(status) &&
          [val.client.toLowerCase(), val.id + ""].some((r) =>
            r.includes(searchInput.toLowerCase())
          )
        ) {
          return val;
        }
      })
      .reverse();
    handleSearchChange(dataFiltered);
  }
```

```
  useEffect(() => {
    if(dataType === "orders") {
        ordersDataFiltered();
    }
    if(dataType === "clients") {
        clientsDataFiltered();
    }

  }, [searchInput, status]);

  return (
    <>
      <input
        type="text"
        placeholder="Пошук за іменем або ID"
        onChange={(e) => setSearchInput(e.target.value)}
        value={searchInput}
       />
    </>
  );
}

export default SearchBar;
```

# ДОДАТОК Б

## Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



StrikePlagiarism.com

Дата звіту 7/15/2024
Дата редагування ---

Документ прийнятий

**метадані**

Заголовок
**2024_Б_ПІ_ПЗПІп-22-1_Пивовар_Є_В**

Автор
**Пивовар Євген Валентинович**

Науковий керівник / Експерт
**Вадим Юрійович Нечволод**

підрозділ
**Харківський національний університет радіоелектроніки**

**Тривога**

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

| | | |
|---|---|---|
| Заміна букв | | 1 |
| Інтервали | | 0 |
| Мікропробіли | | 12 |
| Білі знаки | | 0 |
| Парафрази (SmartMarks) | | 46 |

**Обсяг знайдених подібностей**

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

7.65% 7.65% КП 1

2.03% 2.03% КЦ

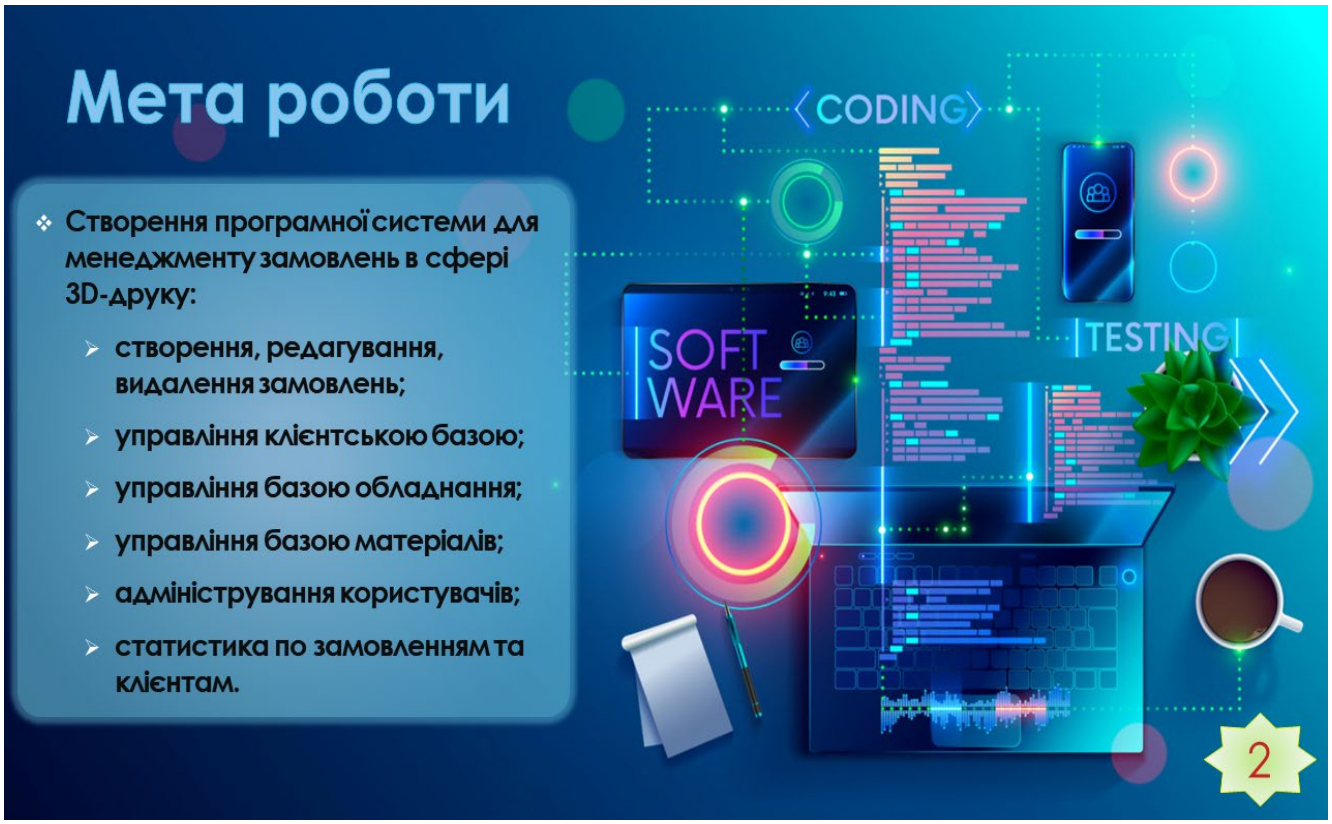
**25**
Довжина фрази для коефіцієнта подібності 2

**8904**
Кількість слів

**73971**
Кількість символів

ДОДАТОК В

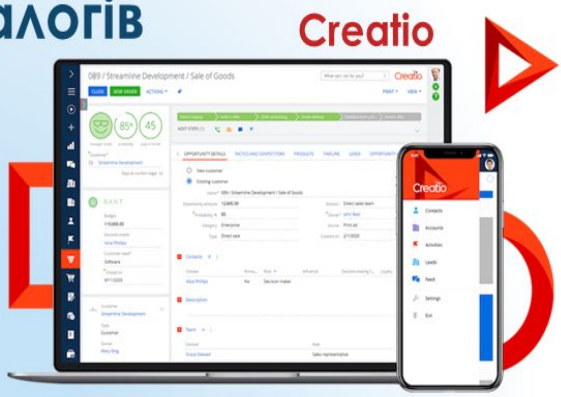
Слайди презентації

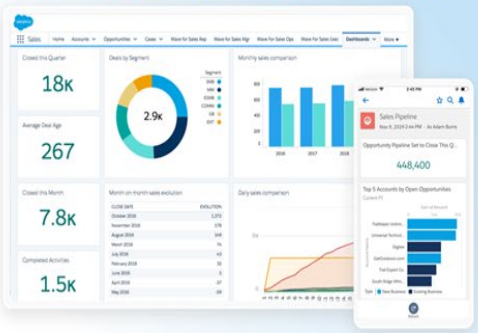
# Аналіз аналогів

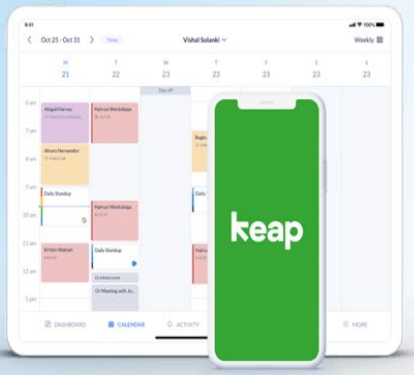
| Критерій | Print Orders | Creatio | Salesforce Sales Cloud | Keap |
|---|---|---|---|---|
| Управління замовленнями | + | + | + | + |
| Управління клієнтською БД | + | + | + | + |
| Звітність | + | + | + | + |
| Мобільний додаток | - | + | + | + |
| Інтеграція з іншими системами | - | + | + | + |
| Автоматизація | + | + | + | + |
| Простота інтерфейсу | + | - | - | - |
| Простота налаштування | + | - | - | - |
| Вартість | + | - | - | - |

Creatio



Salesforce Sales Cloud

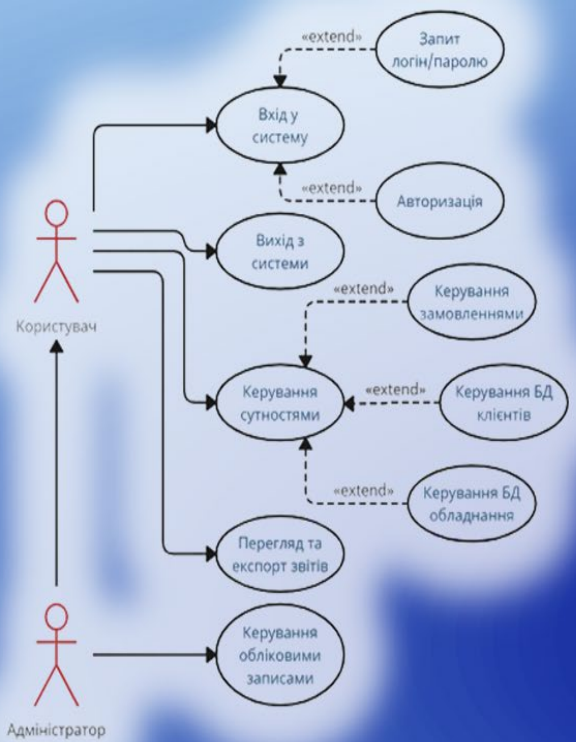




Keap

3

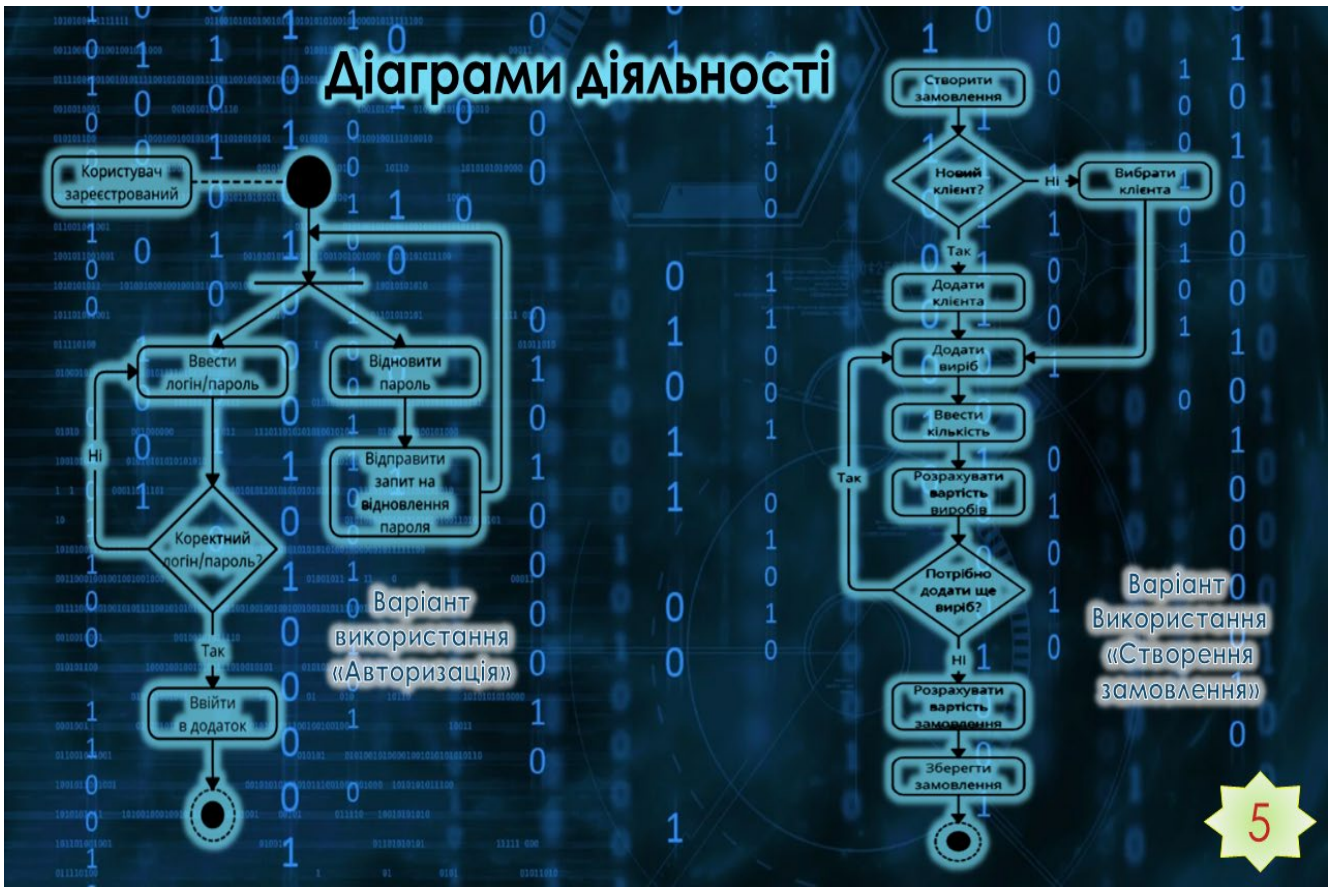
# USE-CASE діаграма

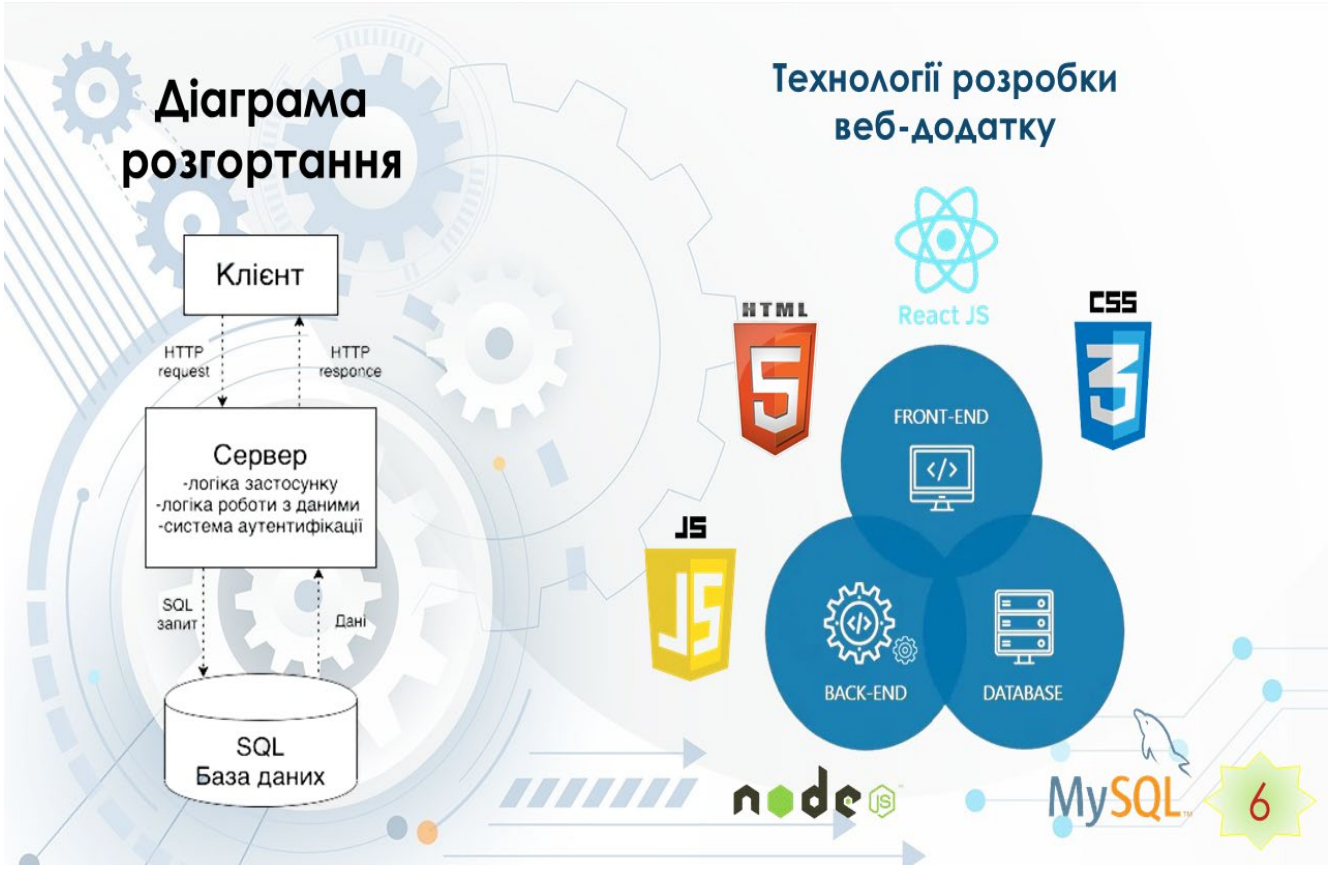


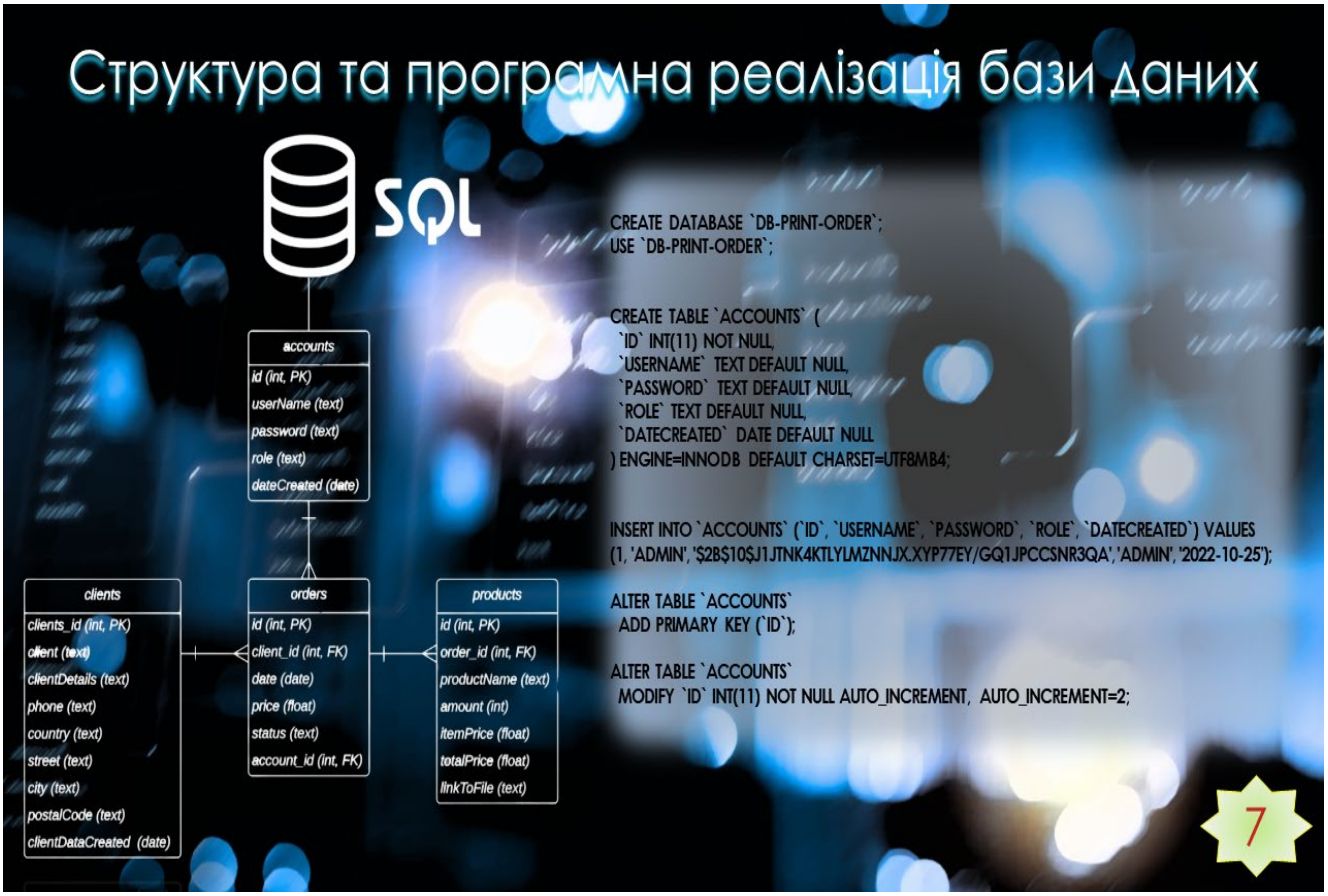

У програмному забезпеченні передбачено два типи зовнішніх користувачів, які взаємодіють із системою:

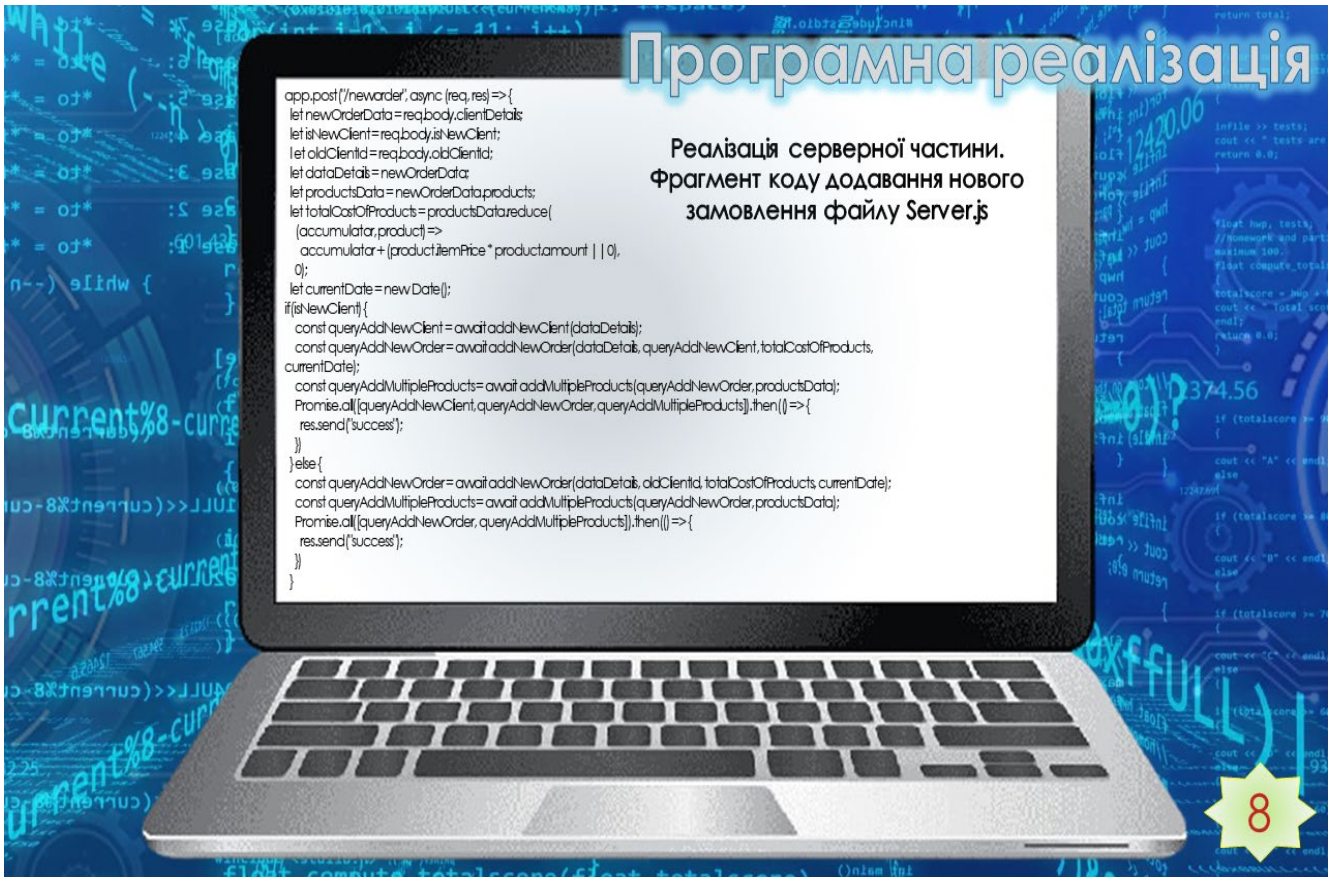
- користувачі можуть ввійти в додаток, вийти з нього, керувати сутностями, такими, як замовленнями, обладнанням та клієнтами.
- адміністратори мають доступ до всіх функцій, які доступні звичайним користувачам, а також у них є можливість управління користувачами.

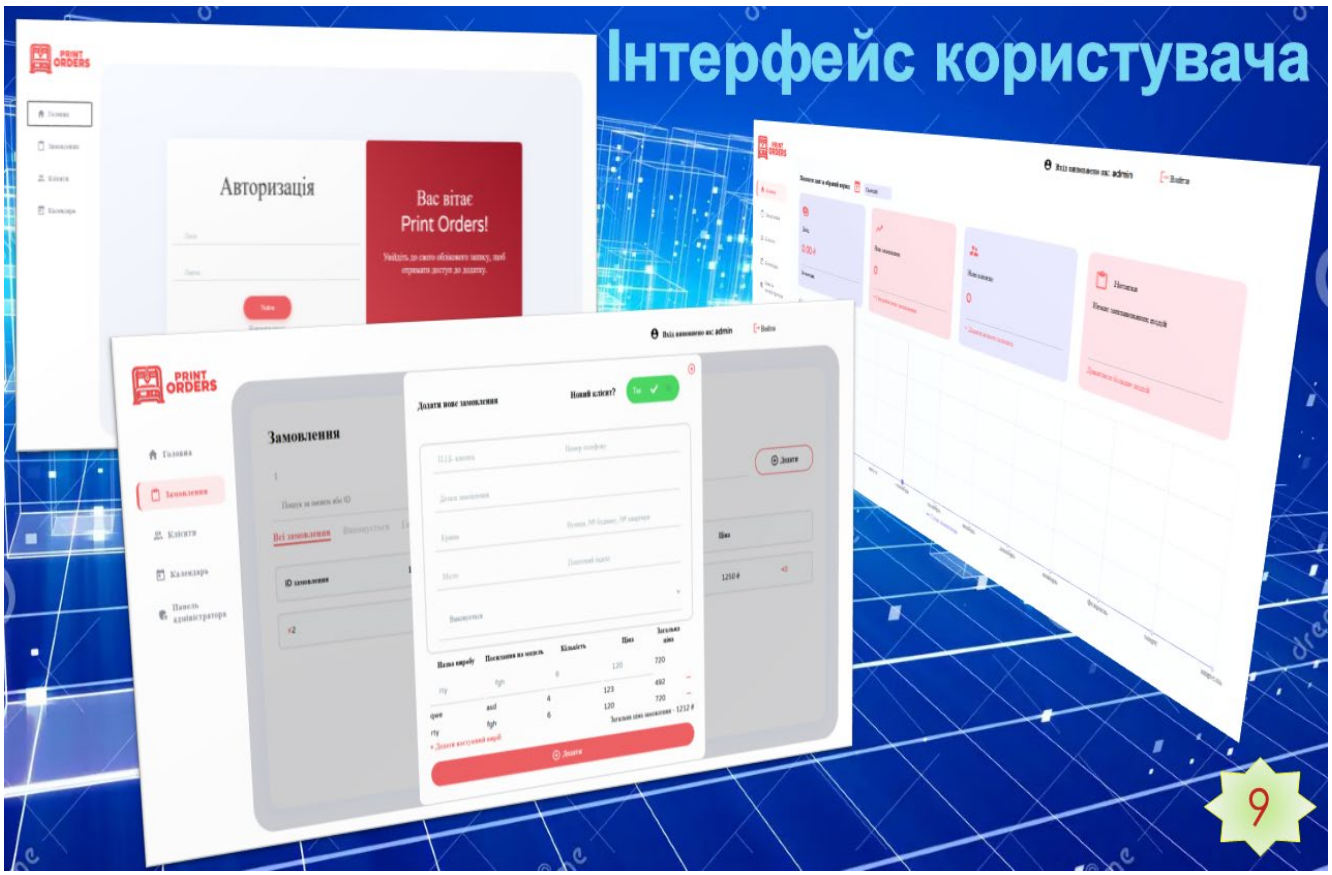
4


Діаграми діяльності
Користувач зареєстрований
Ввести логін/пароль
Відновити пароль
Відправити запит на відновлення пароля
Коректний логін/пароль?
Ні
Так
Ввійти в додаток
Варіант використання «Авторизація»
Створити замовлення
Новий клієнт?
Ні
Вибрати клієнта
Так
Додати клієнта
Додати виріб
Ввести кількість
Розрахувати вартість виробів
Так
Потрібно додати ще виріб?
Ні
Розрахувати вартість замовлення
Зберегти замовлення
Варіант Використання «Створення замовлення»
5


Діаграма розгортання
Клієнт
HTTP request
HTTP responce
Сервер
-логіка застосунку
-логіка роботи з даними
-система аутентифікації
SQL запит
Дані
SQL
База даних
Технології розробки веб-додатку
HTML
React JS
CSS
FRONT-END
JS
BACK-END
DATABASE
node
MySQL
6


Структура та програмна реалізація бази даних
SQL
accounts
id (int, PK)
userName (text)
password (text)
role (text)
dateCreated (date)
clients
clients_id (int, PK)
client (text)
clientDetails (text)
phone (text)
country (text)
street (text)
city (text)
postalCode (text)
clientDataCreated (date)
orders
id (int, PK)
client_id (int, FK)
date (date)
price (float)
status (text)
account_id (int, FK)
products
id (int, PK)
order_id (int, FK)
productName (text)
amount (int)
itemPrice (float)
totalPrice (float)
linkToFile (text)
CREATE DATABASE `DB-PRINT-ORDER`;
USE `DB-PRINT-ORDER`;
CREATE TABLE `ACCOUNTS` (
`ID` INT(11) NOT NULL,
`USERNAME` TEXT DEFAULT NULL,
`PASSWORD` TEXT DEFAULT NULL,
`ROLE` TEXT DEFAULT NULL,
`DATECREATED` DATE DEFAULT NULL
) ENGINE=INNODB DEFAULT CHARSET=UTF8MB4;
INSERT INTO `ACCOUNTS` (`ID`, `USERNAME`, `PASSWORD`, `ROLE`, `DATECREATED`) VALUES
(1, 'ADMIN', '$2B$10$J1JTNK4KTLYLMZNNJX.XYP77EY/GQ1JPCCSNR3QA', 'ADMIN', '2022-10-25');
ALTER TABLE `ACCOUNTS`
ADD PRIMARY KEY (`ID`);
ALTER TABLE `ACCOUNTS`
MODIFY `ID` INT(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;
7


Програмна реалізація
Реалізація серверної частини.
Фрагмент коду додавання нового замовлення файлу Server.js
app.post("/neworder", async (req, res) => {
let newOrderData = req.body.clientDetails;
let isNewClient = req.body.isNewClient;
let oldClientId = req.body.oldClientId;
let dataDetails = newOrderData;
let productsData = newOrderData.products;
let totalCostOfProducts = productsData.reduce(
(accumulator, product) =>
accumulator + (product.itemPrice * product.amount || 0),
0);
let currentDate = new Date();
if(isNewClient) {
const queryAddNewClient = await addNewClient(dataDetails);
const queryAddNewOrder = await addNewOrder(dataDetails, queryAddNewClient, totalCostOfProducts,
currentDate);
const queryAddMultipleProducts = await addMultipleProducts(queryAddNewOrder, productsData);
Promise.all([queryAddNewClient, queryAddNewOrder, queryAddMultipleProducts]).then(() => {
res.send("success");
})
} else {
const queryAddNewOrder = await addNewOrder(dataDetails, oldClientId, totalCostOfProducts, currentDate);
const queryAddMultipleProducts = await addMultipleProducts(queryAddNewOrder, productsData);
Promise.all([queryAddNewOrder, queryAddMultipleProducts]).then(() => {
res.send("success");
})
}
8




# Тестування

**Функціональне тестування авторизації в веб-додатку**

| Опис випадку | Очікуваний результат | Висновок |
|---|---|---|
| Відкрити веб-застосунок | Користувач має доступ до сторінки авторизації | Пройдено |
| Ввести облікові дані: "Email" та "Пароль" | Введені дані відображаються коректно | Пройдено |
| Натиснути кнопку "Увійти" | Користувач авторизований, перенаправлено на головну сторінку | Пройдено |
| Ввести некоректні облікові дані | Відображається повідомлення про помилку | Пройдено |
| Натиснути кнопку "Забули пароль? " | Перенаправлено на сторінку відновлення пароля | Пройдено |
| Ввести зареєстрований Email у формі відновлення пароля | Відправлено пароль на вказаний Email | Пройдено |

**Результати тестування на продуктивність**

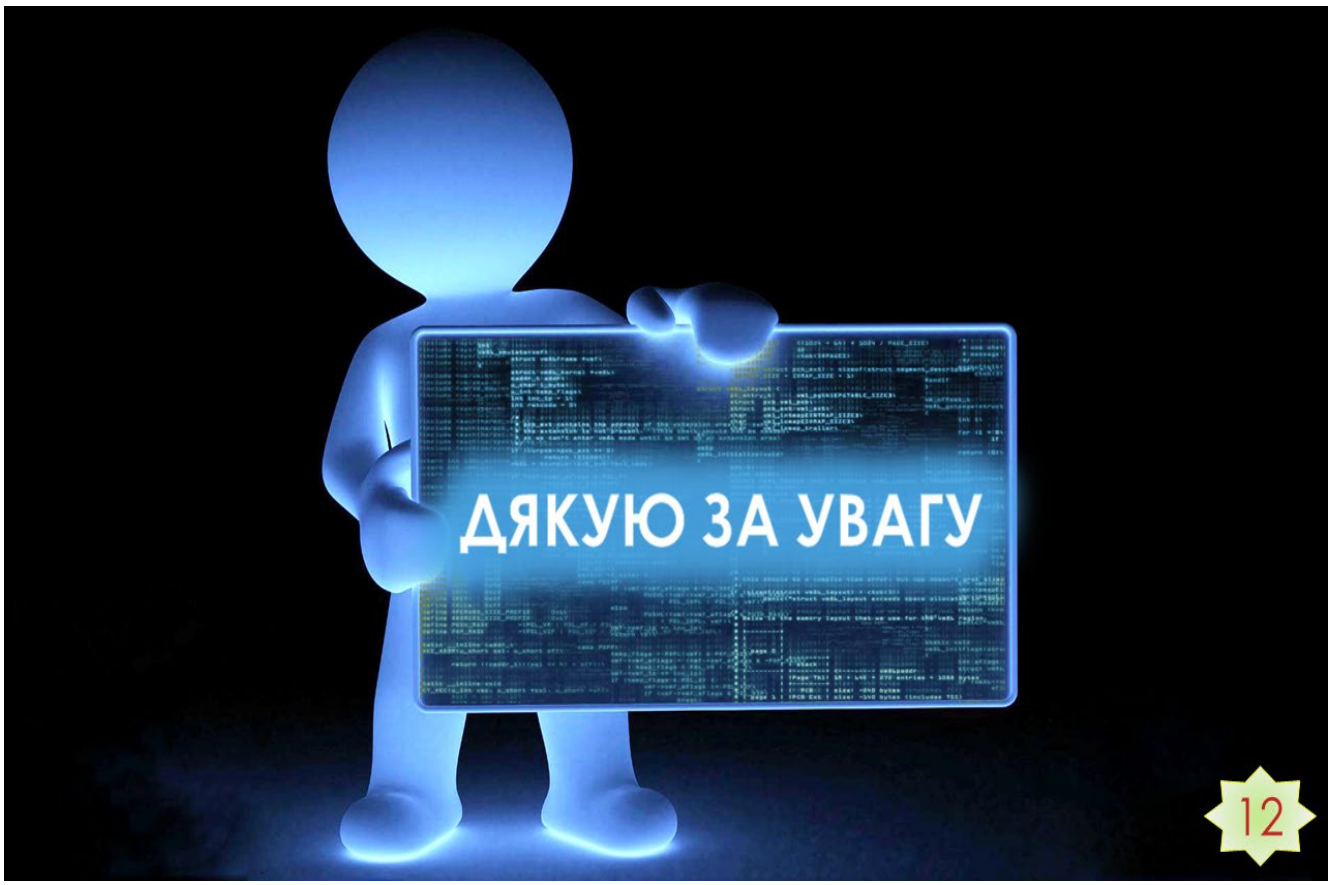
| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | Received KB/sec | Sent KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| HTTP Request 01 ... | 500 | 48 | 29 | 135 | 184.07 | 0.00% | 11.7/sec | 12.16 | 9.53 | 1084.0 |
| HTTP Request 02 ... | 500 | 104 | 73 | 246 | 218.67 | 0.00% | 7.6/sec | 8.43 | 11.37 | 1733.0 |
| TOTAL | 1000 | 76 | 29 | 246 | 308.61 | 0.00% | 18.6/sec | 20.18 | 17.93 | 1408.5 |

10

# Висновки

- Проведений аналіз предметної галузі;
- Спроектована архітектура програмної системи;
- Реалізована та протестована програмна система;
- Закладені основи розвитку програмної системи для роботи в сфері 3D-друку.

11




12