КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

на тему: «***Впровадження CRM-системи у роботу інтернет-магазину***»

Хмеленко Євгеній Вадимович

Київ 2023 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій проектування та прикладної математики

(кафедра)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ

Тетяна ГОНЧАРЕНКО
„22” Червня 2023 року

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

на тему: " Впровадження CRM-системи у роботу інтернет-магазину "

Виконав: студент 2 – го курсу, групи КНм – 22

Спеціальності: 122 «Комп’ютерні науки .

(шифр і назва напряму підготовки, спеціальності)

Магістрант Хмеленко Євгеній Вадимович
(прізвище та ініціали)

Керівник Горда Олена Володимирівна
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Київ, 2023 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій .
Кафедра: інформаційних технологій .
Освітній рівень: «магістр за ОПП» .
Спеціальність: 122 «Комп'ютерні науки» .

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
Тетяна ГОНЧАРЕНКО
„22" Червня 2023 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТР

1. Тема роботи: Хмеленко Євгеній Вадимович
затверджена наказом ректора КНУБА № 1280/2 від «26» Червня 2023р.

2. Керівник роботи: Горда Олена Володимирівна.

3. Строк подання студентом роботи до захисту:.

4. Зміст пояснювальної записки за розділами:

Р.1. Теоретичні основи організації взаємодії споживачів з компанією в електронному середовищі

Р.2. Аналіз стану організації взаємодії споживачів з компанією в електронному середовищі.

Р.3. Перспективи розвитку організації взаємодії споживачів з компанією в електронному середовищі.

Р.4. Реалізація програмного продукту.

5. Інформаційні слайди:

6. Календарний план виконання атестаційної випускної роботи

Види робіт та їх зміст	Дата виконання
Р.1. Теоретичні основи CRM-системи	
Р.2. Аналіз ринку CRM-систем та вибір оптимальної платформи.	
Р.3. Опис програмної реалізації CRM-системи.	
Р.4. Програмна реалізація.	
Остаточне оформлення роботи	
Направлення роботи на рецензування, перевірку на плагіат	
Попередній захист роботи на кафедрі	

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис
Розділ 1.			
Розділ 2.			
Розділ 3.			
Розділ 4.			

8. Дата видачі завдання: «22» Червня 2023р.

Керівник _____ Горда О.В.
(підпис) (прізвище та ініціали)

Магістрант _____ Хмеленко Є.В.
(підпис) (прізвище та ініціали)

РЕЗЮМЕ

Київський національний університет будівництва і архітектури

Хмеленко Євгеній Вадимович

факультет автоматизації і інформаційних технологій, група КНм-22

Тема атестаційної випускної роботи: «Впровадження CRM-системи у роботу інтернет-магазину»

освітній рівень: магістр,

спеціальність: 122 «Комп'ютерні технології»,

Науковий керівник: Горда Олена Володимирівна

Обсяг роботи. Атестаційна випускна робота магістра складається: з 4 розділів, стор. 73, 2 таблиць, 2 діаграм, 28 рисунків, завдання, анотації, вступу, висновків, списку використаних джерел та додатків.

Актуальність теми. Впровадження CRM-системи в інтернет-магазині сприяє підвищенню ефективності бізнесу, покращенню якості обслуговування клієнтів, зростанню продажів та оптимізації внутрішніх процесів.

У вступі визначені основні напрямки дослідження, обґрунтовано актуальність теми, сформульовано мету та основні завдання системи.

У першому розділі наведено основні відомості, що таке CRM-система, історія походження, основні функції та переваги CRM-систем, їх класифікація, типізація та основні переваги використання в електронній комерції.

У другому розділі проведено аналіз CRM-ринку світу та Україні, представлені найвідоміші українські платформи та платформи світу, проведено порівняльний аналіз CRM-систем, наведено критерії відбору системи для інтернет-магазину.

У третьому розділі виконано аналіз функціональних вимог, розроблена архітектура програмного зазпечення, проведено аналіз баз даних, що відповідають вимогам та обрано мову програмування.

У четвертому розділі представлен приклад реалізації CRM-системи.

Ключові слова CRM-система, бізнес-процес, управління клієнтським обслуговуванням, інформаційні процеси, економічний ефект від впровадження

Якість оформлення проекту. Атестаційна випускна робота магістра оформлена у відповідності до діючих нормативних документів та методичних вказівок до виконання дипломних робіт для студентів спеціальності 126 «Інформаційні системи і технології». Порушень та зауважень під час розробки та перевірки дипломної роботи не виявлено.

Загальний висновок стосовно роботи та присвоєння авторові освітньо-кваліфікаційного рівня «магістр». Робота виконана якісно та на високому рівні, студент продемонстрував достатній рівень теоретичної підготовки та сформованих практичних навичок в області сучасних інформаційних технологій. Заслуговує оцінки «Відмінно».

Науковий керівник _____/ Горда Олена Володимирівна./

(підпис)

Посада, місце роботи: к.т.н., доц. кафедри інформаційних технологій

«23» Листопада 2022р.

АНОТАЦІЯ

Атестаційна робота присвячена дослідженню ринку CRM-систем, їх вплив на сучасну електронну комерцію та приклад програмної реалізації CRM-системи.

В умовах стрімкого розвитку електронної комерції та зростання конкуренції інтернет-магазини стикаються з необхідністю ефективного управління взаємодією з клієнтами. CRM-системи (Customer Relationship Management) дозволяють оптимізувати процеси взаємодії з клієнтами, підвищити рівень їх задоволеності та лояльності, а також збільшити обсяги продажів.

У роботі проведено комплексний аналіз впровадження CRM-системи у роботу інтернет-магазину, розроблено методичні рекомендації щодо вибору та інтеграції CRM-системи з існуючими бізнес-процесами. Отримані результати можуть бути використані власниками та менеджерами інтернет-магазинів для підвищення ефективності управління клієнтськими взаєминами, збільшення продажів та покращення сервісу.

Робота включає приклад реалізації CRM-системи, який демонструє її основні можливості та перспективи цього напряму програмного забезпечення.

ANNOTATION

The certification work contributes to the research of the market of CRM systems, their impact on modern electronic commerce and an example of software implementation of the CRM system.

In the conditions of the rapid development of e-commerce and the growth of competition, online stores are achieved by ensuring effective management of interaction with customers. CRM systems (Customer Relationship Management) can optimize the processes of interaction with customers, increase their level of satisfaction and loyalty, and also increase the volume of sales.

In the work, a comprehensive analysis of the implementation of the CRM system in the work of the online store was carried out, methodical recommendations were developed for the selection and integration of the CRM system with existing business processes. The

obtained results can be used by the owners and managers of online stores to increase the efficiency of managing customer interactions, increase sales and improve service.

The work includes an example of the implementation of the CRM system, which demonstrates its main capabilities and prospects of this area of software.

РЕЦЕНЗІЯ

на атестаційну випускну роботу

Студента Хмеленка Євгенія Вадимовича

Факультет автоматизації і інформаційних технологій

Спеціальності 122 «Інформаційних технологій»

Тема роботи: Інтеграція CRM-системи у роботі інтернет-магазину.

Обсяг роботи:

Висновок про відповідність завданню: робота виконана у повній відповідності до завдання і у встановлений термін .

Актуальність обраної теми: інтеграція CRM-системи у електронну комерцію збільшує зручність праці, прибуток та надає нові можливості взаємодії з клієнтами.

Використання у роботі сучасних досягнень науки і техніки: розробка проекту базується на використанні сучасних інформаційних комп'ютерних технологій

Використання у роботі комп'ютерних технологій: Next.js, PostgreSQL.

Практичне значення роботи: впровадження сучасних інформаційних технологій має забезпечувати виконання ряду вимог, у тому числі наявність зручного і дружнього інтерфейсу, забезпечення безпеки за допомогою різних методів контролю та розмежування доступу до інформаційних ресурсів, підтримку розподіленої обробки інформації.

Якість оформлення роботи: випускна робота оформлена у відповідності до діючих нормативних документів та методичних вказівок для студентів спеціальності 122 «Комп'ютерні науки»

Зауваження та побажання: Зауважень не виявлено

Загальний висновок стосовно роботи та надання авторові освітнього ступеня "магістр": робота виконана на високому рівні, студент продемонстрував високий рівень теоретичної підготовки та сформованих практичних навичок в області сучасних інформаційних технологій. Заслуговує оцінки «відмінно».

Рецензент _____/ к.ф.-м.н.,

(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Посада, місце роботи: доцент кафедри інформаційних технологій проєктування та прикладної математики КНУБА

«__» _____2023р.

ЗМІСТ

ВСТУП

В сучасному світі електронної комерції інтернет-магазини відіграють все важливішу роль у задоволенні потреб споживачів. Зростання конкуренції на ринку змушує компанії шукати нові шляхи для підвищення ефективності своєї роботи, покращення обслуговування клієнтів та збільшення обсягів продажів.

Одним з таких інструментів є системи управління взаєминами з клієнтами (CRM – Customer Relationship Management). Впровадження CRM-системи дозволяє інтернет-магазину автоматизувати та оптимізувати процеси взаємодії з клієнтами, що в свою чергу підвищує рівень задоволеності клієнтів та їх лояльність, а також сприяє зростанню прибутковості бізнесу.

Мета кваліфікаційної роботи. Основною метою даної атестаційної роботи є дослідження процесу впровадження CRM-системи у роботу інтернет-магазину для покращення управління клієнтськими відносинами та підвищення конкурентоспроможності.

Об'єкт дослідження — інтернет-магазин як суб'єкт електронної комерції.

Предметом дослідження — є процес впровадження CRM-системи у роботу інтернет-магазину.

Методи дослідження. У роботі використано методи аналізу, синтезу, порівняння, моделювання та експертного оцінювання. Це дозволило систематично підходити до вивчення теми, виявляти ключові аспекти та взаємозв'язки, а також розробити рекомендації з впровадження CRM-системи.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ CRM-СИСТЕМИ

1.1 Поняття CRM-системи

Управління взаємовідносинами з клієнтами (Customer relationship management) — це процес, у якому компанія або інша організація керує своєю взаємодією з клієнтами, зазвичай використовуючи аналіз даних для вивчення великих обсягів інформації.

CRM-системи збирають дані з низки різних каналів зв’язку, включаючи веб-сайт компанії, телефон, електронну пошту, живий чат, маркетингові матеріали та останнім часом соціальні мережі. Вони дозволяють компаніям дізнатися більше про свою цільову аудиторію та про те, як краще задовольнити їхні потреби, таким чином утримуючи клієнтів і стимулюючи зростання продажів.

CRM можна використовувати з минулими, теперішніми або потенційними клієнтами. Концепції, процедури та правила, яких дотримується корпорація під час спілкування зі своїми споживачами, називаються CRM. Цей повний зв’язок охоплює прямі контакти з клієнтами, такі як продажі та операції, пов’язані з обслуговуванням, прогнозування та аналіз моделей і поведінки споживачів з точки зору компанії.

1.2 Історія CRM-системи

Історія CRM-систем розпочинається ще у 1980-х роках з появою програмного забезпечення для автоматизації продажів (Sales Force Automation, SFA). Одним з перших інструментів такого роду був ACT!, розроблений компанією Conductor Software у 1987 році. ACT! дозволяв користувачам зберігати контактну інформацію та відстежувати взаємодії з клієнтами, що стало першим кроком до сучасних CRM-систем.

На початку 1990-х років, із розвитком технологій баз даних і зростанням

потреб бізнесу в більш комплексних рішеннях, виникає концепція управління взаємовідносинами з клієнтами. У цей час на ринку з'являються такі компанії, як Siebel Systems, яка стала одним з провідних постачальників CRM-систем у 1993 році. Продукти Siebel надавали більш широкий функціонал, включаючи управління продажами, маркетингом і обслуговуванням клієнтів.

З розвитком Інтернету в кінці 1990-х і початку 2000-х років, CRM-системи починають використовувати веб-технології. У 1999 році була заснована компанія Salesforce.com, яка стала першою компанією, що запропонувала CRM як послугу (SaaS, Software as a Service). Це дозволило компаніям використовувати CRM-системи без необхідності встановлення програмного забезпечення на своїх серверах, що значно знизило витрати і зробило CRM доступним для більш широкого кола бізнесів.

Сучасні CRM-системи є комплексними платформами, які інтегруються з іншими бізнес-додатками, використовують штучний інтелект для аналізу даних і надають інструменти для прогнозування поведінки клієнтів. Серед найпопулярніших CRM-систем сьогодні — Salesforce, Microsoft Dynamics 365, SAP CRM, Oracle CRM, HubSpot та інші.

Таким чином, CRM-системи пройшли значний шлях від простих інструментів для зберігання контактів до багатофункціональних платформ, які допомагають бізнесам ефективно взаємодіяти з клієнтами і покращувати свої процеси.

1.3 Основні функції та переваги CRM-системи

CRM-системи є незамінним інструментом для сучасних компаній, які прагнуть ефективно управляти своїми взаємовідносинами з клієнтами та залишатися конкурентоспроможними на ринку. Їх основними функціями та перевагами являються:

Управління контактами:

- Збереження та організація інформації про клієнтів, включаючи контактні дані, історію взаємодій, уподобання та потреби.
- Централізоване зберігання даних про клієнтів, що дозволяє швидкий доступ до інформації для всіх співробітників компанії.

Управління продажами:

- Автоматизація процесу продажів, включаючи лідогенерацію, кваліфікацію потенційних клієнтів, управління можливостями та прогнозування продажів.
- Моніторинг статусу угод, що допомагає відстежувати прогрес продажів та виявляти вузькі місця в процесі.

Маркетингова автоматизація:

- Підтримка маркетингових кампаній, включаючи розсилки електронних листів, соціальні медіа, аналітику та сегментацію ринку.
- Відстеження ефективності маркетингових заходів, що дозволяє коригувати стратегії для досягнення кращих результатів.

Обслуговування клієнтів:

- Управління запитами клієнтів, включаючи їх реєстрацію, обробку та відстеження.
- Підтримка багатоканального обслуговування, що дозволяє клієнтам взаємодіяти з компанією через різні канали (телефон, електронна пошта, чат тощо).

Аналітика та звітність:

- Збір та аналіз даних про клієнтів, продажі та маркетингові заходи.
- Генерація звітів, що допомагають приймати обґрунтовані рішення на основі даних.

Управління проектами:

- Координація завдань та проектів, пов'язаних з клієнтами.
- Відстеження прогресу проектів та управління командною роботою.

Переваги використання CRM-системи:

Покращення взаємовідносин з клієнтами: CRM-системи допомагають краще розуміти потреби клієнтів, що дозволяє надавати їм персоналізовані послуги та підвищувати рівень їх задоволеності.

Збільшення ефективності роботи: Автоматизація рутинних процесів дозволяє співробітникам зосередитися на важливіших завданнях, що сприяє підвищенню продуктивності.

Оптимізація процесів продажу та маркетингу: CRM-системи надають інструменти для більш ефективного управління продажами та маркетинговими кампаніями, що сприяє збільшенню доходів.

Прийняття обґрунтованих рішень: Аналітичні функції CRM-систем дозволяють отримувати глибокі інсайти та приймати рішення на основі даних.

1.4 Типи та класифікація CRM-системи

CRM-системи можна класифікувати за різними критеріями, залежно від їх функціональності, способу впровадження та призначення. Нижче наведені основні типи та класифікації:

Типи CRM-систем за функціональністю:

1. **Операційні CRM (Operational CRM):**
    - Призначені для автоматизації та оптимізації основних бізнес-процесів, таких як продажі, маркетинг та обслуговування клієнтів.
    - Включають модулі для управління контактами, лідами, угодами, обслуговуванням клієнтів, тощо.
    - Основна мета – підвищення ефективності операційних процесів та покращення взаємодії з клієнтами.
2. **Аналітичні CRM (Analytical CRM):**

- Призначені для збору, зберігання та аналізу даних про клієнтів з метою отримання інсайтів та прийняття обґрунтованих рішень.
- Включають інструменти для аналізу поведінки клієнтів, сегментації ринку, прогнозування продажів, оцінки ефективності маркетингових кампаній тощо.
- Основна мета – глибоке розуміння потреб клієнтів та оптимізація бізнес-стратегій на основі даних.

3. **Колаборативні CRM (Collaborative CRM):**
   - Призначені для покращення співпраці між різними підрозділами компанії та взаємодії з клієнтами через різні канали комунікації.
   - Включають інструменти для управління взаємодією через електронну пошту, чат, соціальні мережі та інші канали.
   - Основна мета – забезпечення безшовної комунікації та підвищення рівня задоволеності клієнтів шляхом узгодженої роботи різних відділів компанії.

Класифікація CRM-систем за способом впровадження:

1. **Локальні CRM (On-Premises CRM):**
   - Встановлюються та працюють на серверах компанії.
   - Потребують значних початкових інвестицій в обладнання та програмне забезпечення.
   - Забезпечують високий рівень контролю та безпеки даних.
2. **Хмарні CRM (Cloud CRM):**
   - Працюють на серверах постачальника послуг і доступні через Інтернет.
   - Не потребують значних початкових інвестицій, працюють за підпискою.
   - Забезпечують високу гнучкість, масштабованість та доступність з будь-якого місця.
3. **Гібридні CRM (Hybrid CRM):**
   - Поєднують елементи On-Premises та Cloud CRM.

- Дозволяють зберігати частину даних локально, а частину – в хмарі.
- Забезпечують баланс між контролем над даними та гнучкістю хмарних рішень.

Класифікація CRM-систем за розміром та типом бізнесу

1. **CRM для малого та середнього бізнесу (SMB CRM):**
   - Призначені для невеликих компаній з обмеженими ресурсами.
   - Зазвичай мають простий інтерфейс та основні функції, які легко налаштовуються.
   - Вартість таких систем зазвичай нижча, і вони пропонують базові можливості для управління взаємодією з клієнтами.
2. **CRM для великого бізнесу (Enterprise CRM):**
   - Призначені для великих компаній з складною структурою та великим обсягом даних.
   - Мають розширені функціональні можливості, інтеграцію з іншими корпоративними системами (ERP, HRM тощо).
   - Потребують більш складного налаштування та управління.

Класифікація CRM-систем за галузевою спеціалізацією

1. **Галузеві CRM (Industry-Specific CRM):**
   - Розроблені з урахуванням специфічних потреб певних галузей (наприклад, охорона здоров'я, фінанси, нерухомість, виробництво).
   - Включають спеціалізовані функції та модулі, які відповідають вимогам конкретної галузі.
2. **Універсальні CRM (Generic CRM):**
   - Підходять для широкого спектру галузей та можуть бути налаштовані під конкретні потреби компанії.
   - Пропонують загальні функціональні можливості для управління взаємодією з клієнтами.

Отже, вибір конкретного типу CRM-системи залежить від специфіки бізнесу, потреб компанії та наявних ресурсів.

1.5 Переваги використання CRM-системи в електронній комерції

Використання CRM-систем в електронній комерції має безліч переваг, які сприяють покращенню взаємодії з клієнтами, підвищенню ефективності бізнес-процесів та збільшенню прибутковості. Основними перевагами використання CRM-систем в електронній комерції являються:

1. Поліпшення взаємин з клієнтами

CRM-системи дозволяють ефективно збирати та аналізувати дані про клієнтів, що допомагає краще розуміти їх потреби та уподобання. Це дозволяє створювати персоналізовані пропозиції та комунікації, що підвищують рівень задоволеності клієнтів.

2. Підвищення рівня обслуговування клієнтів

Завдяки зберіганню всієї інформації про клієнтів в одному місці, співробітники можуть швидко знаходити потрібні дані та оперативно реагувати на запити клієнтів. Це значно покращує якість обслуговування та скорочує час реагування на звернення.

3. Автоматизація маркетингових кампаній

CRM-системи дозволяють автоматизувати маркетингові процеси, такі як розсилки електронних листів, SMS-кампанії, таргетована реклама тощо. Це допомагає ефективніше досягати цільової аудиторії та збільшувати конверсії.

4. Аналіз та сегментація клієнтів

CRM-системи надають потужні інструменти для аналізу даних та сегментації клієнтів за різними критеріями (вік, місцезнаходження, історія покупок тощо). Це дозволяє створювати більш таргетовані маркетингові стратегії та підвищувати ефективність продажів.

5. Покращення процесів продажу

CRM-системи автоматизують та оптимізують процеси продажу, дозволяючи відстежувати статус угод, керувати лідами та можливостями, прогнозувати продажі. Це сприяє підвищенню продуктивності відділу продажів та зменшенню циклу угоди.

6. Інтеграція з іншими системами

CRM-системи можуть інтегруватися з іншими бізнес-системами, такими як системи управління складом, платіжні системи, системи обліку тощо. Це забезпечує єдину екосистему для управління всіма аспектами бізнесу, покращуючи координацію та обмін даними між різними відділами.

7. Підвищення ефективності прийняття рішень

Аналітичні інструменти CRM-систем дозволяють отримувати глибокі інсайти про поведінку клієнтів, ефективність маркетингових кампаній, тренди продажів. Це допомагає приймати обґрунтовані рішення на основі даних, що сприяє оптимізації бізнес-стратегій.

8. Управління лояльністю клієнтів

CRM-системи дозволяють ефективно управляти програмами лояльності, відстежувати активність клієнтів, нараховувати бонуси та пропонувати персоналізовані винагороди. Це допомагає утримувати постійних клієнтів та стимулювати їх до повторних покупок.

9. Підвищення мобільності бізнесу

Багато сучасних CRM-систем надають можливість доступу з мобільних пристроїв, що дозволяє співробітникам мати доступ до необхідної інформації та функцій незалежно від їх місцезнаходження. Це особливо важливо для електронної комерції, де швидкість та гнучкість є ключовими факторами успіху.

10. Покращення прогнозування та планування

Завдяки збору та аналізу великої кількості даних CRM-системи допомагають точно прогнозувати попит, планувати закупівлі та управління запасами. Це сприяє оптимізації логістики та зменшенню витрат.

В цілому, впровадження CRM-системи в електронну комерцію дозволяє покращити взаємодію з клієнтами, оптимізувати бізнес-процеси та збільшити ефективність роботи компанії, що в кінцевому результаті веде до збільшення прибутковості.

РОЗДІЛ 2. АНАЛІЗ РИНКУ CRM-СИСТЕМИ ТА ВИБІР ОПТИМАЛЬНОЇ ПЛАТФОРМИ

2.1 Огляд ринку CRM-систем

Одне з перших запитань, які у вас можуть виникнути, якщо ви розглядаєте використання CRM, це: «Чи хтось ще це робить?» Якщо більшість компаній чудово обходяться без CRM, цілком зрозуміло, що ви не вважаєте це необхідною інвестицією. Однак це зовсім не так, ось деяка статистика:

- 13% компаній стверджують, що інвестиції в платформу CRM є одним із головних пріоритетів продажів.
- 70% клієнтів очікують безперебійної взаємодії з усіма каналами, що робить CRM-системи необхідними для забезпечення постійної взаємодії з клієнтами.
- У 2021 році дохід від програмного забезпечення CRM сягнув 48,7 мільярда доларів США, і прогнозується, що з 2021 по 2028 рік він зростатиме на 14,2% у середньому на рік.
- Компанії, які використовують системи CRM, відзначили збільшення перебігу потенційних клієнтів на 17%, утримання клієнтів – на 16%, а продуктивність агентів – на 21%.
- CRM є найбільшим і найшвидше зростаючим ринком програмного забезпечення з очікуваним глобальним доходом у 114,4 мільярда доларів США до 2027 року.
- 91% підприємств із понад 11 співробітниками зараз використовують CRM-системи порівняно з 50% компаній із 10 чи менше співробітниками.
- У середньому системи CRM пропонують рентабельність інвестицій (ROI) у розмірі 8,71 доларів США за кожен витрачений долар.
- 74% користувачів CRM повідомляють, що їхня система CRM покращила доступ до даних клієнтів
- 65% компаній починають використовувати CRM протягом перших

п’яти років роботи.

Ринок CRM-систем є дуже динамічним і насиченим різноманітними рішеннями, які можуть задовольнити потреби як малих і середніх бізнесів, так і великих корпорацій. Розподілом цього ринку займаються, як великі постачальники програмного забезпечення світого масштабу, так і локальні компанії, чиї CRM-системи використовуються переважно клієнтами з тієї ж країни. Основним монополістом являється компанія **Salesforce** (за різними даними їй належить від 19.5% до 23.9% ринку), але з кожним роком її частка зменшується, на ринок виходять нові пропозиції та збільшується частка інших компаній.

Діаг. 2.1 Розподіл ринку CRM постачальників у 2024 році

Основними CRM платформами являються:

1. Salesforce

- **Опис:** Salesforce — це надзвичайно популярна платформа CRM, яка широко використовується в усіх галузях промисловості по всьому світу. Маючи понад 80 000 співробітників і 150 000 клієнтів різних розмірів і галузей, клієнти

Salesforce використовують його, щоб об’єднати всі бізнес-функції в єдине джерело правди. Деякі з найбільших клієнтів Salesforce включають: Amazon Web Services (80 мільярдів доларів США річного прибутку), The New York Post (9.9 мільярдів доларів США), U.S. Bank (22 мільярдів доларів США), Walmart (611.3 мільярдів доларів США), Macy’s (24.4 мільярдів доларів США), T-Mobile (19.2 мільярдів доларів США), L’Oreal Americas (10 мільярдів доларів США), American Express (55 мільярдів доларів США), The Hershey Company (10.8 мільярдів доларів США)

- **Ключові особливості:**
  - Інтеграція з великою кількістю сторонніх додатків через AppExchange.
  - Потужні аналітичні інструменти та функції прогнозування.
  - Широкі можливості налаштування та автоматизації бізнес-процесів.
  - Хмарне рішення, що забезпечує доступ з будь-якого місця.
- **Переваги:** Гнучкість, масштабованість, велика екосистема додатків і партнерів.

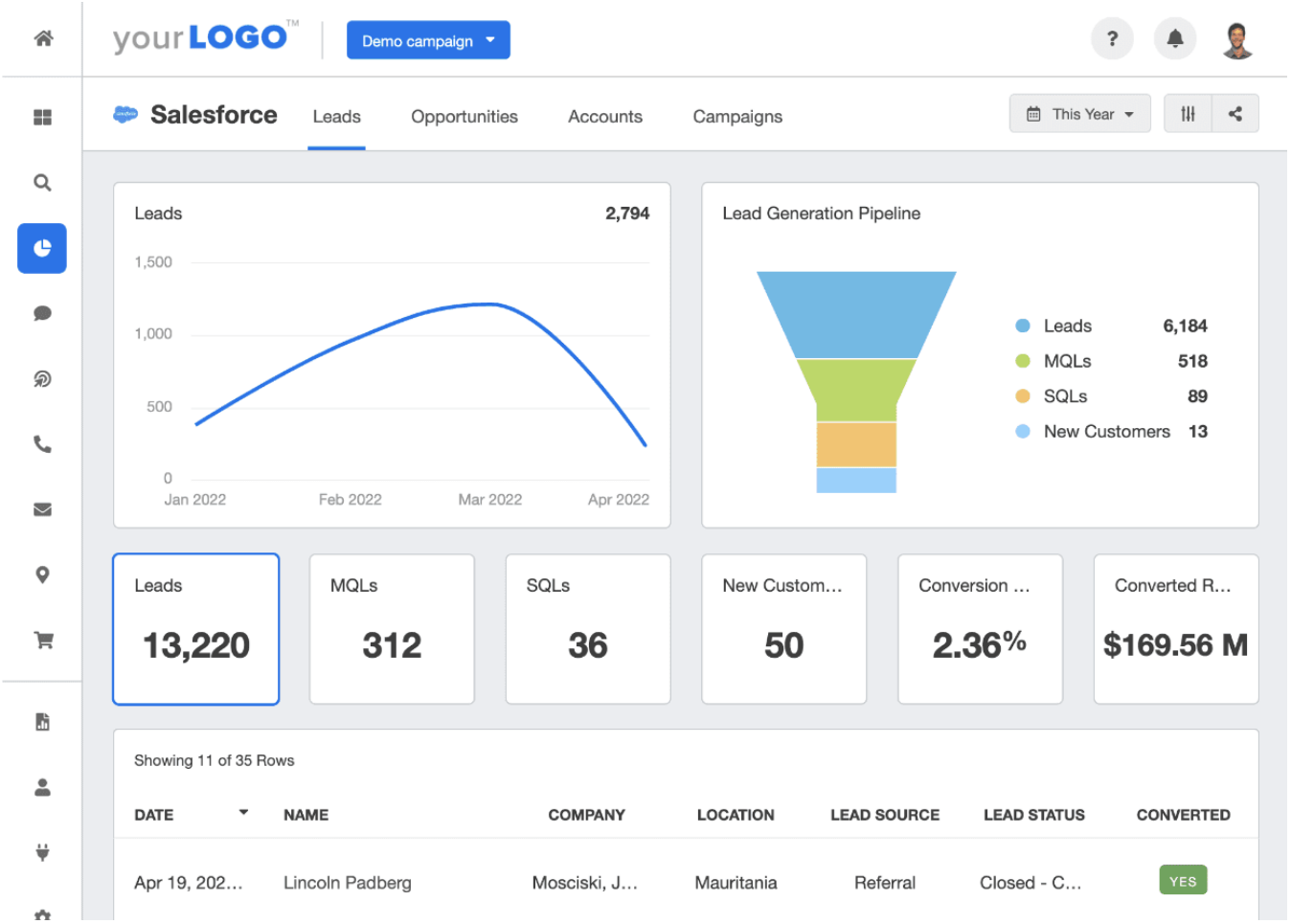


Рис. 2.1 Інтерфейс CRM-системи Salesforce

2. Microsoft Dynamics 365

- **Опис:** Під егідою Microsoft Dynamics кілька продуктів розроблено для різноманітних потреб бізнесу. Dynamics 365 — це всеосяжне хмарне рішення, яке об’єднує різні модулі, наприклад управління взаємовідносинами з клієнтами та планування ресурсів підприємства. Тим не менш, локальні користувачі попередніх версій Dynamics 365, як-от Dynamics CRM, Dynamics 365 CE, Dynamics AX, Dynamics NAV, Dynamics GP і Dynamics SL, залишаються.

Через багато факторів визначити точну кількість клієнтів Microsoft Dynamics складно. Корпорація Майкрософт не завжди розкриває точну кількість клієнтів для своїх продуктів Dynamics. Однак численні джерела пропонують оцінки на основі офіційних оголошень, галузевих звітів і опитувань. У 2021 році Dynamics 365 зайняла 7,3% ринку в сегменті CRM, її перейняли понад 44 000 компаній. У фінансовому році, що закінчився 30 червня 2023 року, Dynamics отримала дохід у розмірі 5,44 мільярда доларів США. Станом на 2023 рік

Dynamics становить 2,5% від усього доходу Microsoft.

- **Ключові особливості:**
  - Інтеграція з продуктами Microsoft (Office 365, Azure, LinkedIn).
  - Можливості для автоматизації продажів, маркетингу та обслуговування клієнтів.
  - Вбудовані аналітичні інструменти та штучний інтелект.
  - Можливість гібридного розгортання (хмарне та локальне).

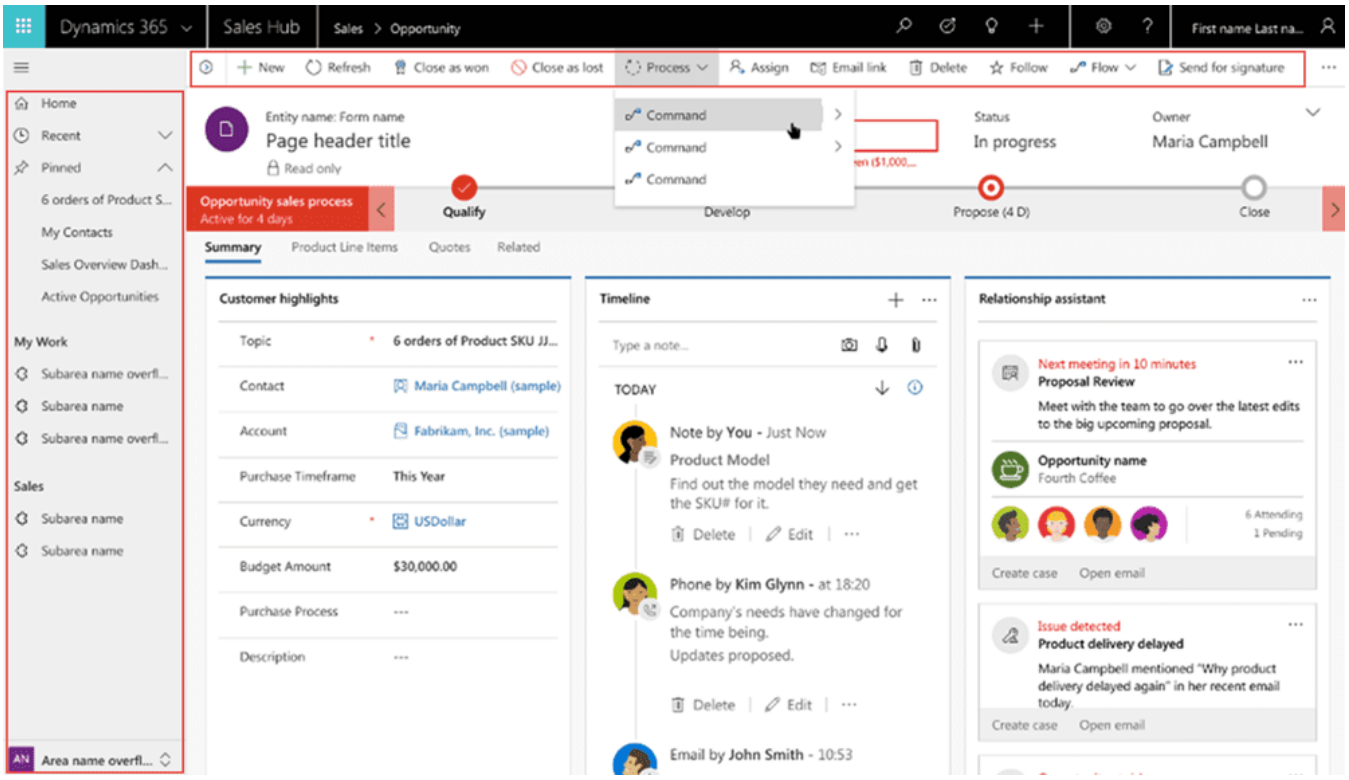


Рис. 2.2 Інтерфейс CRM-системи Microsoft Dynamics 365

- **Переваги:** Інтеграція з іншими продуктами Microsoft, потужні функції для підприємств.

3. HubSpot CRM

- **Опис:** HubSpot — популярна платформа управління взаємовідносинами з клієнтами (CRM) із функціями маркетингу, продажів і взаємодії з клієнтами. Піонер вхідного маркетингу, заснований у 2006 році, у першому кварталі 2024 року збільшив загальну базу користувачів до 216 840 клієнтів.

У 2023 році розробник програмного забезпечення отримав 2,12 мільярда

доларів доходу, що в 2,4 рази більше, ніж у 2020 році.

- **Ключові особливості:**
  - Безкоштовна базова версія з можливістю розширення функціоналу.
  - Інструменти для автоматизації маркетингу, продажів та обслуговування клієнтів.
  - Інтеграція з різними сторонніми додатками.
  - Зручний інтерфейс та легкість у використанні.
- **Переваги:** Доступність, простота в освоєнні, інтегровані інструменти для маркетингу.

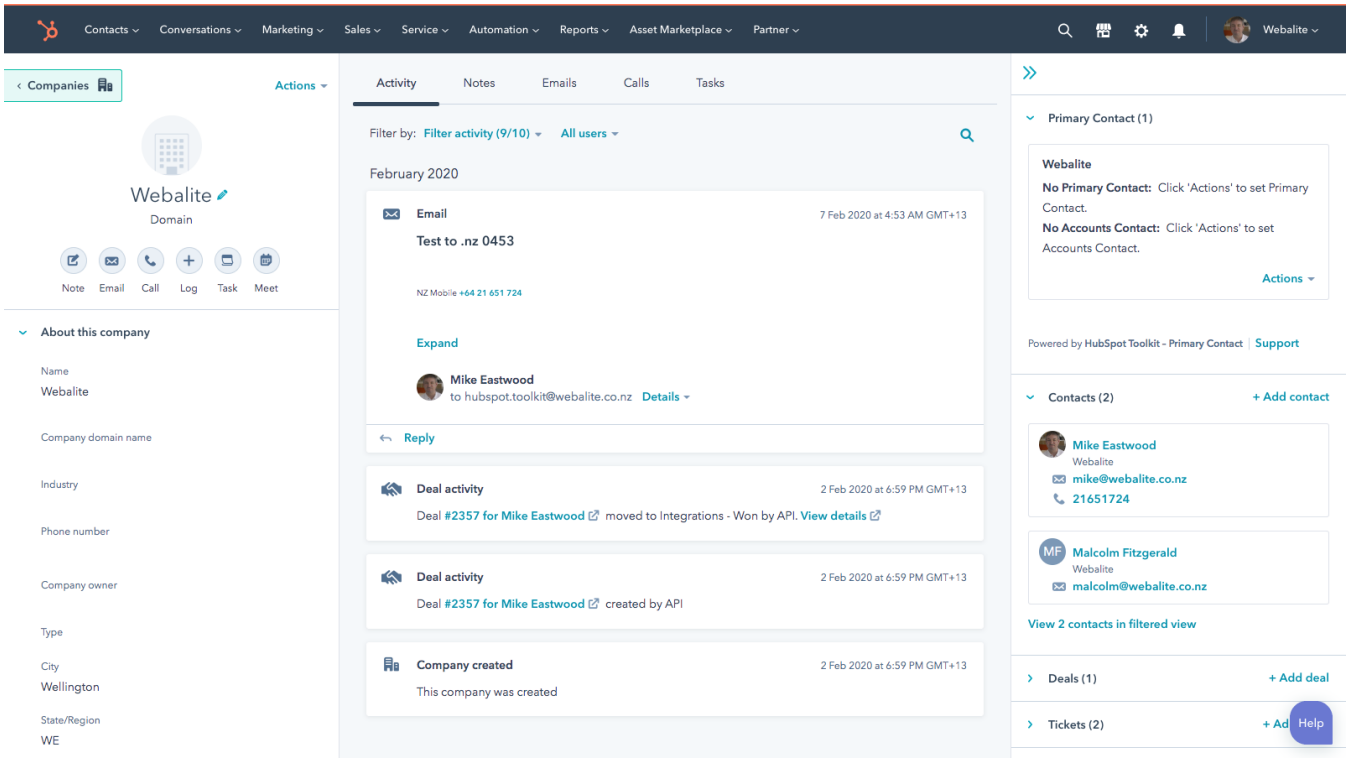


Рис. 2.3 Інтерфейс CRM-системи HubSpot CRM

4. Zoho CRM

- **Опис:** Заснована у 2005 році, Zoho CRM дає змогу глобальній мережі з понад 250 000 компаній у 180 країнах залучати більше потенційних клієнтів, взаємодіяти з клієнтами та збільшувати свої доходи. На даний момент компанія має більше 9000 працівників та 13 млн. користувачів.
- **Ключові особливості:**

- Широкий набір функцій для управління продажами, маркетингом та підтримкою клієнтів.
- Можливості для налаштування та автоматизації бізнес-процесів.
- Інтеграція з іншими продуктами Zoho та сторонніми додатками.
- Доступні різні тарифні плани, включаючи безкоштовну версію.

• **Переваги:** Гнучкість, доступна ціна, багатий функціонал.

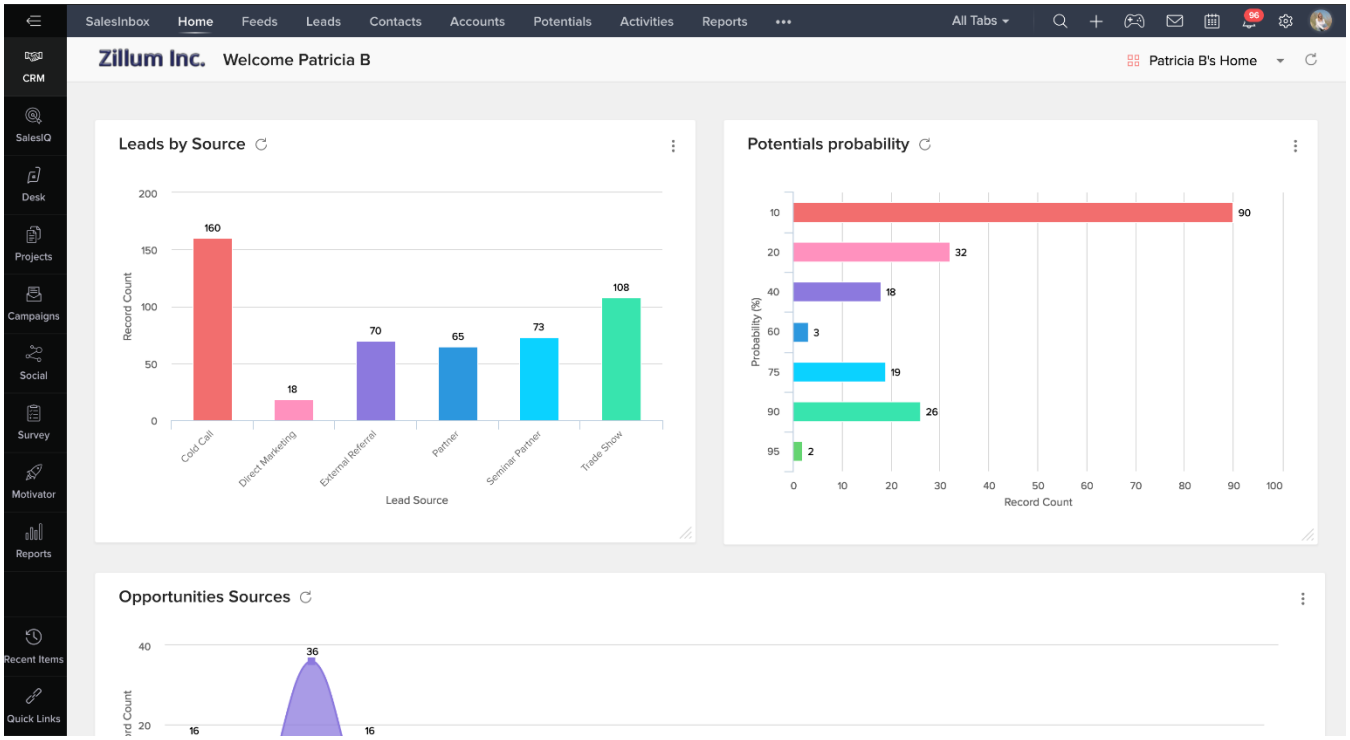


Рис. 2.4 Інтерфейс CRM-системи Zoho CRM

5. SAP Customer Experience (SAP CX)

• **Опис:** SAP Customer Experience - це набір хмарних рішень, які охоплюють весь шлях клієнта, від маркетингу до продажів і обслуговування до комерції. SAP CX допомагає компаніям зрозуміти своїх клієнтів, ефективно залучати їх і забезпечувати цінність у кожній точці взаємодії. SAP CX базується на галузевому досвіді, штучному інтелекті та даних клієнтів.

Відповідно до останнього звіту IDC, SAP CX є лідером на світовому ринку додатків для взаємодії з клієнтами з часткою 9,4% і темпом зростання на 14,8% у 2022 році. SAP CX обслуговує понад 10 000 клієнтів у 25 галузях у понад 120

країнах. Деякі найкращі клієнти SAP включають Apple, Amazon, Audi.

- **Ключові особливості:**
  - Інтеграція з іншими продуктами SAP.
  - Потужні інструменти для аналітики та прогнозування.
  - Рішення для управління продажами, маркетингом, комерцією та обслуговуванням клієнтів.
  - Можливості для персоналізації клієнтського досвіду.
- **Переваги:** Підходить для великих підприємств, потужні аналітичні можливості.

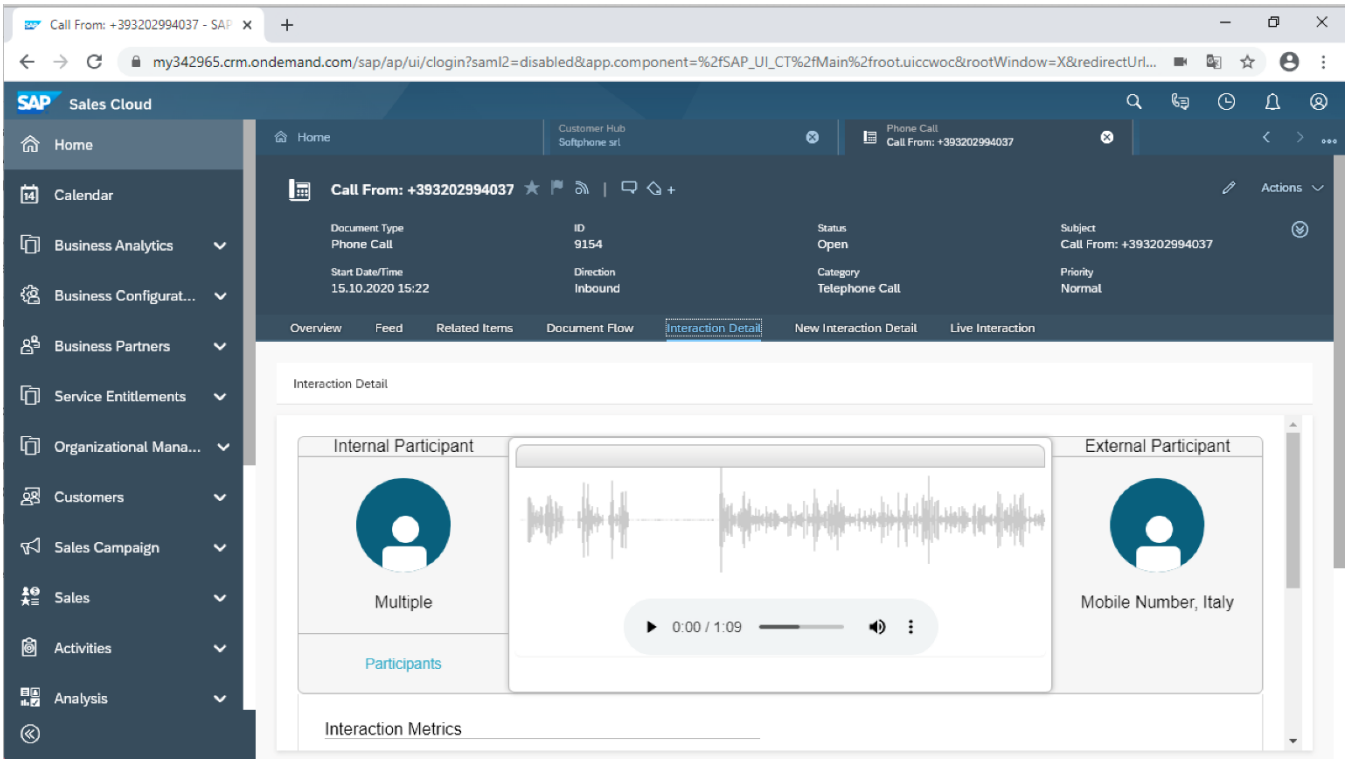


Рис. 2.5 Інтерфейс CRM-системи SAP Customer Experience (SAP CX)

6. Pipedrive

- **Опис:** Завдяки повільному, але дуже обережному зростанню з моменту свого запуску в 2010 році Pipedrive починає робити собі ім’я за останні кілька років як досить надійний CRM.

Цей інструмент управління взаємовідносинами з клієнтами (CRM)

виглядає привабливим і багатообіцяючим для професіоналів із продажу та засновників із слоганом: «Створено, щоб ви продовжували продавати». Привертаючи увагу 13 інвесторів і не демонструючи жодних ознак зниження популярності в найближчому майбутньому, Pipedrive може стати однією з найпопулярніших CRM усіх часів. На даний момент більше 100000 компаній використовують **Pipedrive** у більше ніж 170 країнах, включаючи Антарктику. Станом на 2022 рік дохід компанії складає 65 мільйонів доларів США.

- **Ключові особливості:**
  - Візуальний конвеєр продажів для відстеження угод.
  - Інтеграція з багатьма сторонніми додатками.
  - Інструменти для автоматизації продажів та звітності.
  - Зручний інтерфейс та легкість в освоєнні.
  - **Переваги:** Простота використання, орієнтація на продажі.

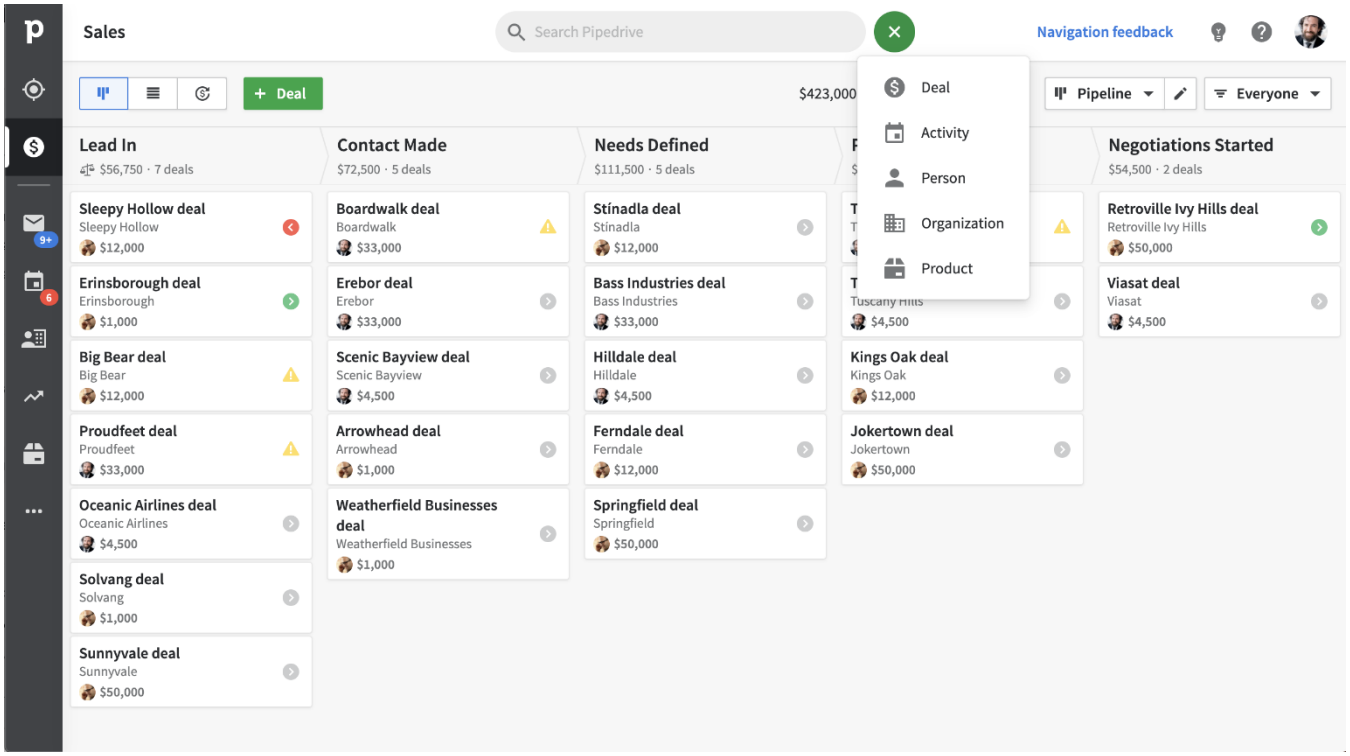


Рис. 2.6 Інтерфейс CRM-системи Pipedrive

7. Freshworks CRM (раніше Freshsales)

- **Опис:** Freshworks — це програмне забезпечення для управління взаємовідносинами з клієнтами (CRM), яке в основному спрямоване на автоматизацію продажів і маркетингу. Воно пропонує відстеження подій, керування угодами, автоматичні електронні листи, розширені звіти та багато іншого. Вбудований телефон і електронна пошта, мобільні застосунки, необмежена кількість контактів та цілодобова безкоштовна підтримка. Станок на 2022 рік ринкова капіталізація компанії становить 4,35 мільярди доларів США та має понад 58100 клієнтів.
- **Ключові особливості:**
  - Вбудовані інструменти для телефонії, електронної пошти та чатів.
  - Автоматизація продажів, управління лідами та контактами.
  - Інтеграція з іншими продуктами Freshworks.
  - Можливість персоналізації робочого процесу.
- **Переваги:** Легкість у використанні, вбудовані комунікаційні інструменти.

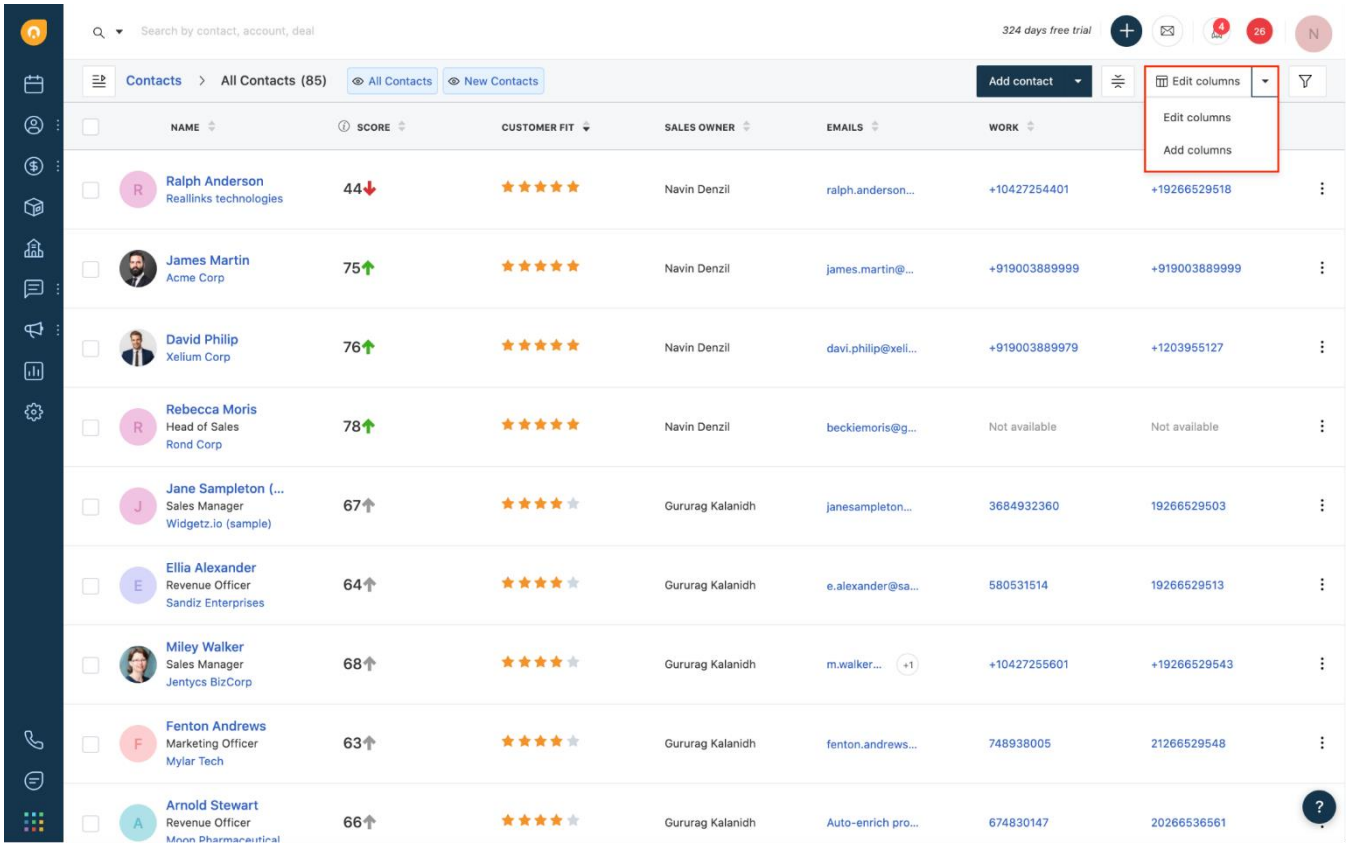


Рис. 2.7 Інтерфейс CRM-системи Freshworks CRM

8. Insightly

- **Опис:** Заснована у 2009 році, Insightly CRM надає програмне забезпечення для управління взаємовідносинами з клієнтами для підприємств будь-якого розміру в різних галузях, таких як виробництво, консалтинг, професійні послуги, ЗМІ та реклама, некомерційні організації, технології та інші. Використовується 25000 компаніями у понад 200 країнах та має дохід близько 14 мільйонів доларів США.
- **Ключові особливості:**
  - Управління контактами та лідами, автоматизація продажів.
  - Інструменти для управління проектами та завданнями.
  - Інтеграція з популярними бізнес-додатками.
  - Потужні аналітичні можливості.
- **Переваги:** Поєднання CRM та управління проектами, інтеграції.

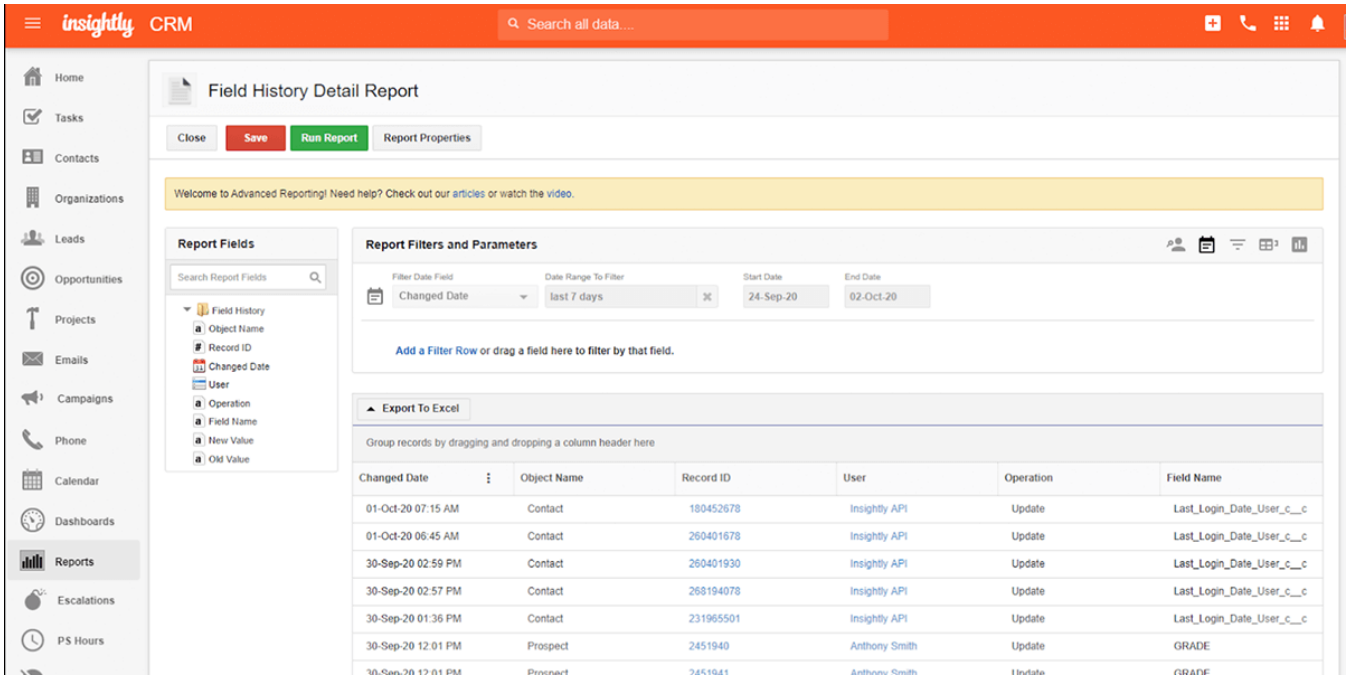


Рис. 2.8 Інтерфейс CRM-системи Insightly

9. SugarCRM

- **Опис:** Заснована у 2004 році, SugarCRM — це програмне забезпечення для управління взаємовідносинами з клієнтами (CRM), яке допомагає компаніям керувати взаємодією з клієнтами, процесами продажів і даними про клієнтів. Платформа пропонує такі функції, як автоматизація продажів, автоматизація маркетингу, обслуговування клієнтів і аналітика, щоб допомогти компаніям покращити відносини з клієнтами та збільшити свої доходи.

Компанія має клієнтів у різних галузях і відома своєю гнучкою архітектурою з відкритим вихідним кодом, яка дозволяє налаштовувати та інтегрувати з іншими бізнес-системами. Станом на 2023 рік має 73,5 мільйонів доларів США доходу та 2 мільйони користувачів.

- **Ключові особливості:**
  - Гнучкі можливості налаштування, що дозволяють адаптувати систему під специфічні потреби бізнесу.
  - Підтримка автоматизації продажів, маркетингу та обслуговування клієнтів.
  - Інтеграція з безліччю сторонніх додатків і сервісів.
  - Відкрита архітектура, що дозволяє розширювати функціональність за допомогою кастомних рішень.
  - **Переваги:** Висока гнучкість та можливість налаштування, масштабованість, підтримка багатьох інтеграцій.

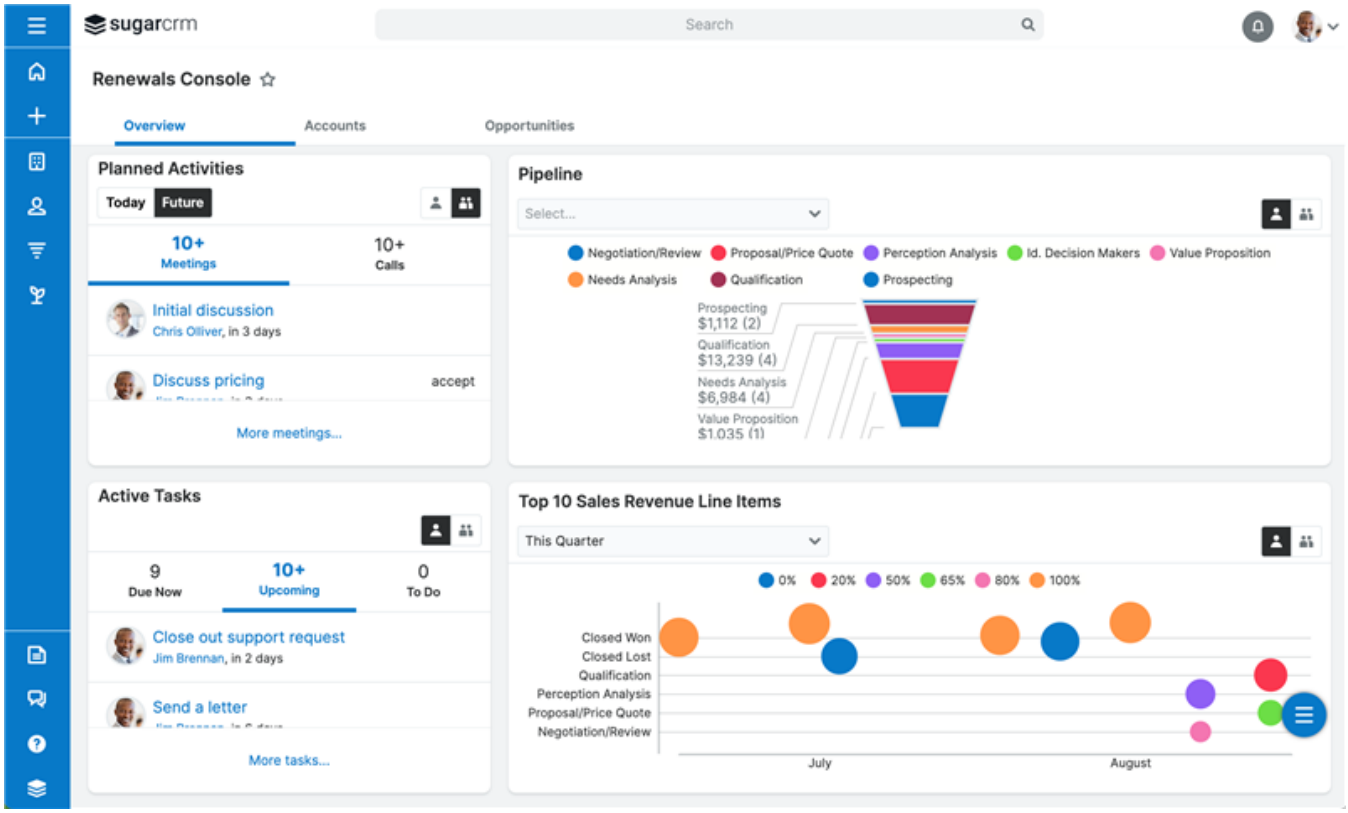


Рис. 2.9 Інтерфейс CRM-системи SugarCRM

10. Keap (раніше Infusionsoft)

- **Опис:** Заснована у 2018 році, Keep — це хмарне CRM-рішення, яке пропонує керування взаємовідносинами з клієнтами, автоматизацію маркетингу та функції електронної комерції в одному пакеті. Це допомагає малим підприємствам оптимізувати продажі та керувати взаємодією з клієнтами, організовуючи потенційних клієнтів, сегментуючи та оцінюючи їх на основі тегів і поведінки, надсилаючи цільові послідовності електронних листів, відстежуючи ціни та угоди, плануючи дзвінки, виставляючи рахунки клієнтам і приймаючи платежі – все в одному інструменті.

Keap також пропонує настроювані робочі процеси, які можна адаптувати до найкращих практик бізнесу. Його інструменти звітності та аналітики допомагають користувачам аналізувати такі параметри, як електронні листи, ефективність кампанії та дані про рентабельність інвестицій. Станом на 2022 рік має 31.5 тисячу клієнтів та 205.6 тисяч користувачів.

- **Ключові особливості:**
  - Автоматизація маркетингових кампаній, включаючи електронні листи, SMS та інші комунікаційні канали.
  - Інструменти для управління контактами, лідами та угодами.
  - Вбудовані можливості для електронної комерції, включаючи управління платежами та інвойсами.
  - Інтуїтивно зрозумілий інтерфейс та простота використання.
  - **Переваги:** Комплексне рішення для маркетингу та продажів, орієнтоване на малі бізнеси, інтуїтивно зрозумілий інтерфейс.

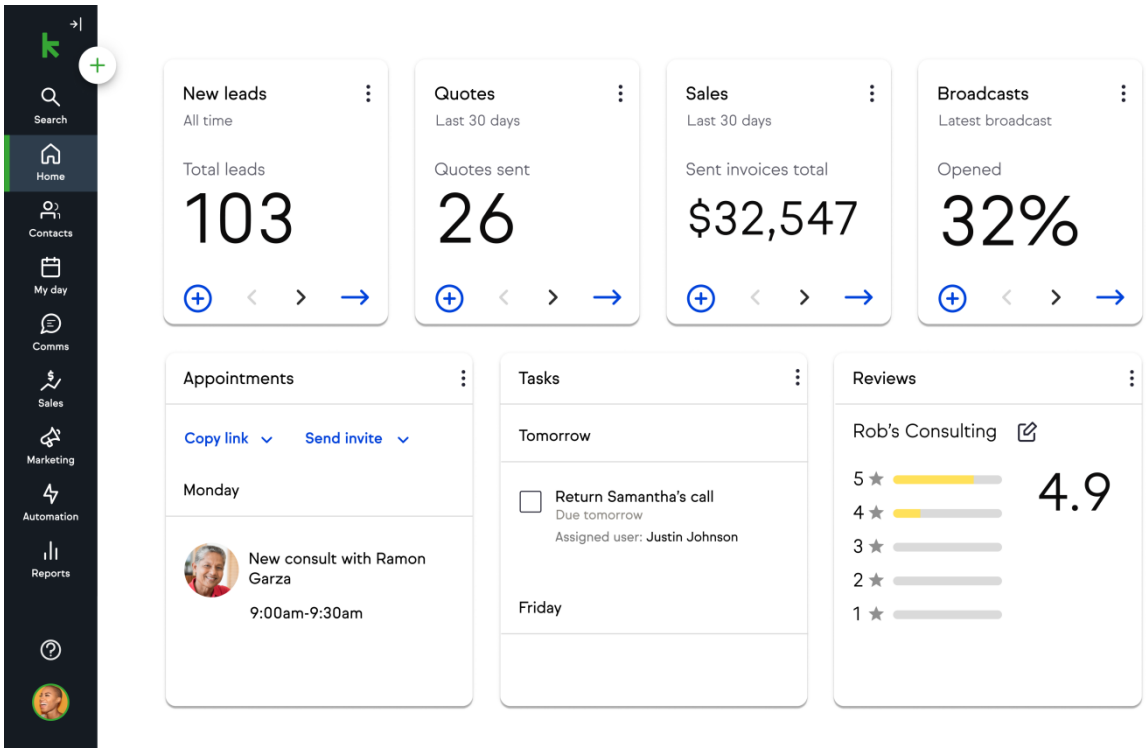


Рис. 2.10 Інтерфейс CRM-системи Keap

2.2 Ринок CRM-систем в Україні

Ринок CRM-систем в Україні розвивається повільніше у порівнянні з міжнародним. Так за дослідженням компанії Бітрікс24 у 2020 році, із 1030 опитаних українських компаній, представників великого, середнього та малого бізнесу, лише 8.4% використовували, впроваджували або планували впровадити

CRM систему.

Діаг. 2.2 Розподіл CRM-систем серед опитуваних компаній на 2020 рік.

Після повномасштабного вторгнення доля ринку українських CRM зросла у 2,5 раза, а російські CRM втратили понад 60% клієнтів в Україні. AmoCRM, retailCRM, YCLIENTS – втратили понад 75%.

У березні 2022 року український офіс Бітрікс24 заявив про розрив стосунків із материнською компанією у рф, але це не допомогло CRM-вендору із російським корінням втримати свої позиції. А 1 червня 2023 року Бітрікс24 потрапили під санкції від РНБО та були змушені остаточно припинити роботу на українському ринку.

Огляд українських CRM-систем:

1. SalesDrive

Опис: SalesDrive – це найпопулярніша українська CRM-система, орієнтована на потреби інтернет-магазинів і компаній, які займаються активними продажами. Система допомагає автоматизувати процеси управління замовленнями, клієнтами та продажами.

- **Ключові особливості:**
  - Управління клієнтською базою та історією взаємодій.
  - Автоматизація процесу обробки замовлень.
  - Інтеграція з популярними службами доставки.
  - Звіти та аналітика продажів.
  - Інтеграція з маркетинговими інструментами.
- **Переваги:**
  - Проста у використанні.
  - Інтуїтивний інтерфейс.
  - Спеціалізація на інтернет-магазинах.
- **Недоліки:**
  - Обмежена функціональність для великих компаній з складними бізнес-процесами.

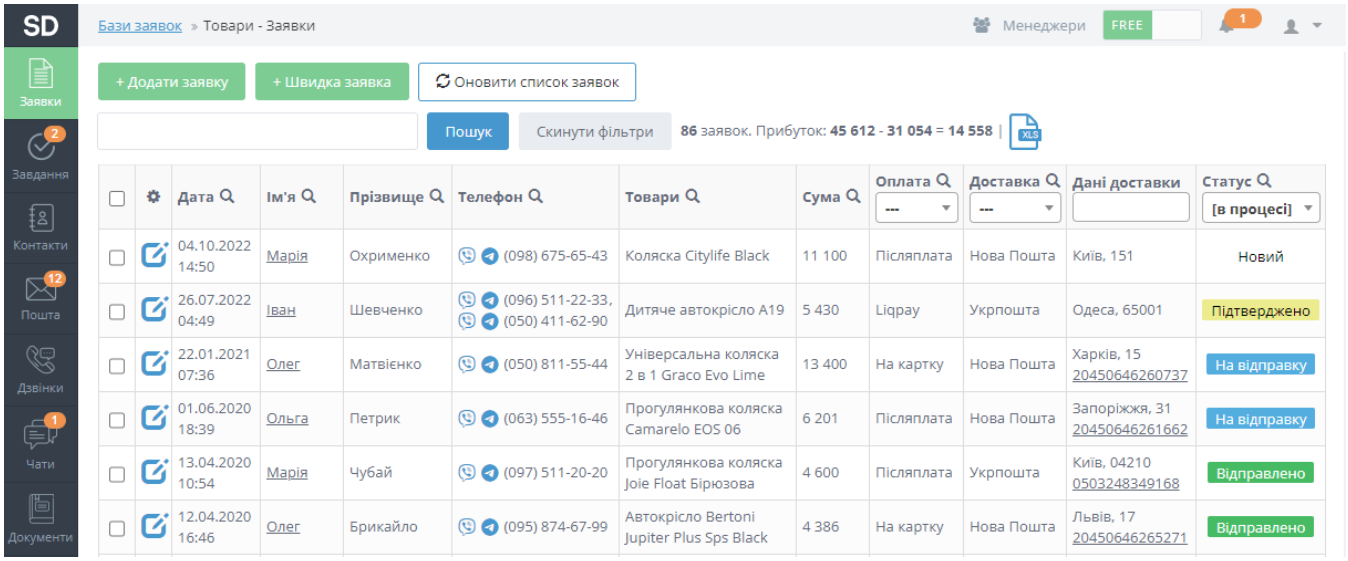


Рис. 2.11 Інтерфейс CRM-системи SalesDrive.

2. KeyCRM

Опис: KeyCRM - українська CRM-система, що буде корисна будь-якому підприємцю (послуги, ecommerce, контакт-центри/онлайн-консультації, сфера краси і здоров'я), так як автоматизує процеси, що є майже у кожному бізнесі.

- **Ключові особливості:**
  - Управління контактами та взаємодіями.
  - Автоматизація маркетингових кампаній.
  - Інтеграція з різними бізнес-додатками та сервісами.
  - Звіти та аналітика.
- **Переваги:**
  - Гнучкість налаштувань.
  - Підтримка багатоканальної комунікації з клієнтами.
  - Інтуїтивний інтерфейс.
- **Недоліки:**
  - Відносно новий продукт на ринку, обмежена кількість користувачів.

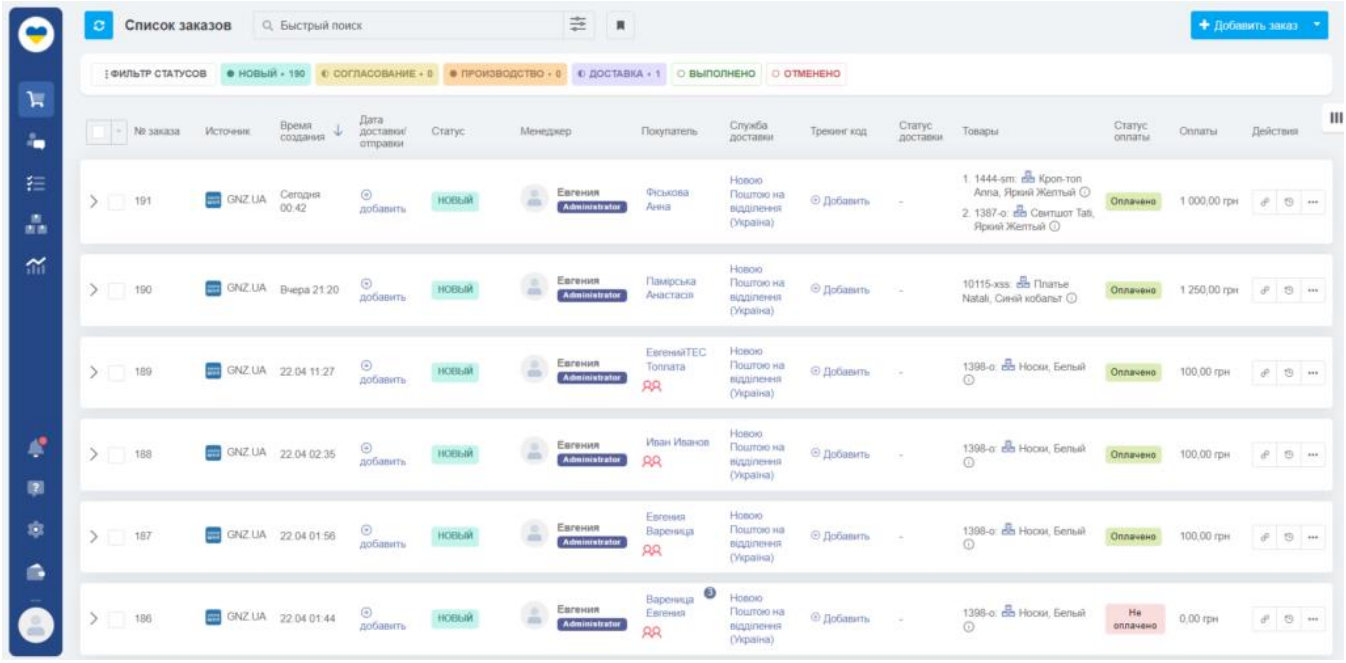


Рис. 2.12 Інтерфейс CRM-системи KeyCRM.

3. KeepinCRM

Опис: KeepinCRM – це система для малого та середнього бізнесу, яка допомагає автоматизувати процеси продажів, управління клієнтами та взаємодіями з ними.

- **Ключові особливості**
  - Управління замовленнями та рахунками.
  - Автоматизація обробки лідів.
  - Інтеграція з популярними сервісами та платформами.
  - Аналітика та звіти.
- **Переваги:**
  - Легкість у використанні.
  - Фокус на малий та середній бізнес.
  - Широкий набір інтеграцій.
- **Недоліки:**
  - Може бути обмежена функціональність для великих компаній.

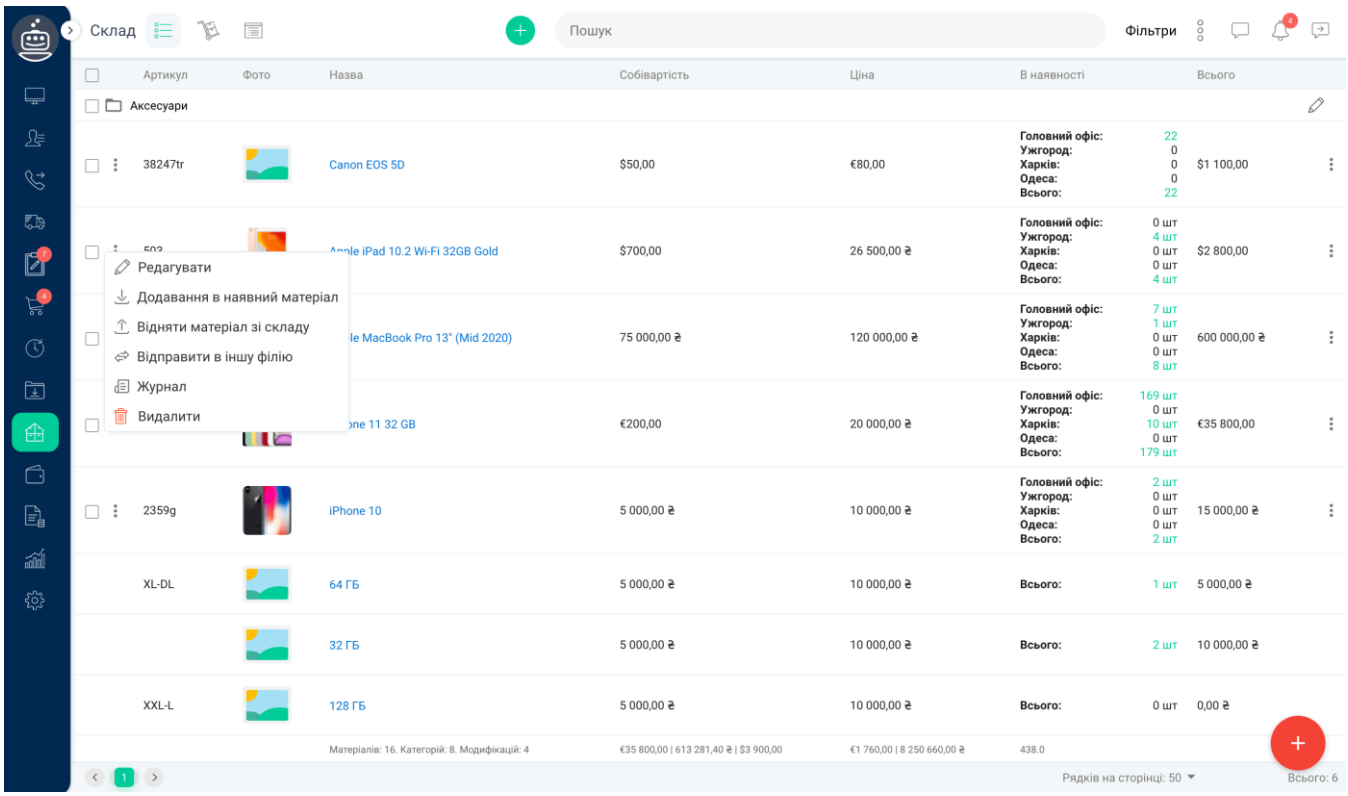


Рис. 2.13 Інтерфейс CRM-системи KeepinCRM.

4. Perfectum

Опис: Perfectum CRM пропонує інструменти для управління клієнтами,

проектами, продажами та маркетингом, орієнтовані на компанії різних розмірів. На ринку з 2017 року, отримувала нагороди як краща українська CRM система 2019, 2020, 2022, 2023 років.

- **Ключові особливості:**
  - Управління контактами та взаємодіями.
  - Проектний менеджмент.
  - Автоматизація продажів та маркетингових кампаній.
  - Звіти та аналітика.
- **Переваги:**
  - Широкий функціонал.
  - Інтеграція з різними бізнес-системами.
  - Підходить для різних типів бізнесів.
- **Недоліки:**
  - Складність налаштування для нових користувачів.

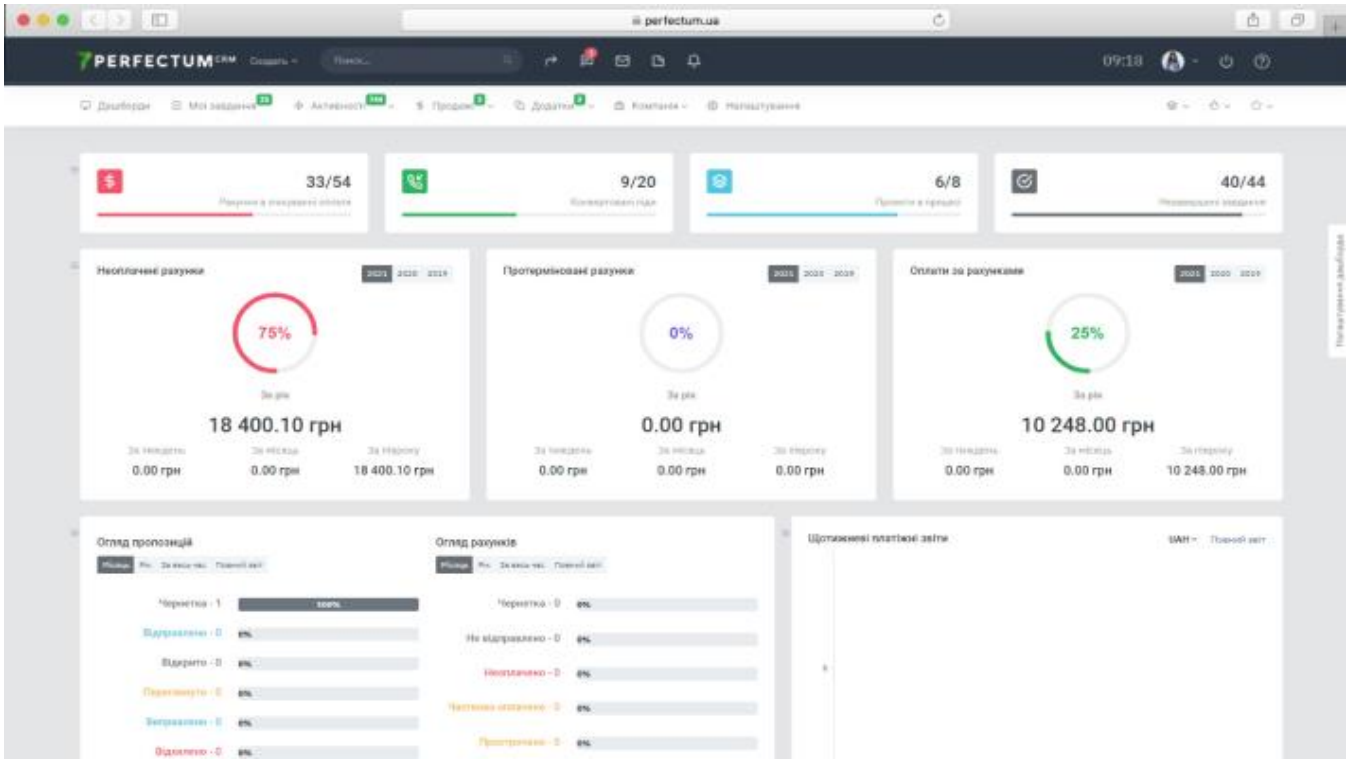


Рис. 2.14 Інтерфейс CRM-системи Perfectum.

5. OneBox

Опис: OneBox – це українська операційна система для бізнесу, яка складається з сотень додатків, включаючи CRM, ERP та інші інструменти для управління бізнес-процесами. OneBox надає гнучке налаштування власної CRM-системи завдяки представленої документації та допомагають з міграцією з інших CRM платформ.

- **Ключові особливості:**
  - Управління клієнтами та взаємодіями.
  - Автоматизація бізнес-процесів.
  - Інтеграція з великою кількістю сторонніх сервісів.
  - Потужні інструменти аналітики та звітності.
- **Переваги:**
  - Висока гнучкість та налаштовуваність.
  - Підходить для великих компаній з складними бізнес-процесами.
  - Інтеграція з різними платформами.
- **Недоліки:**
  - Висока вартість впровадження та підтримки.

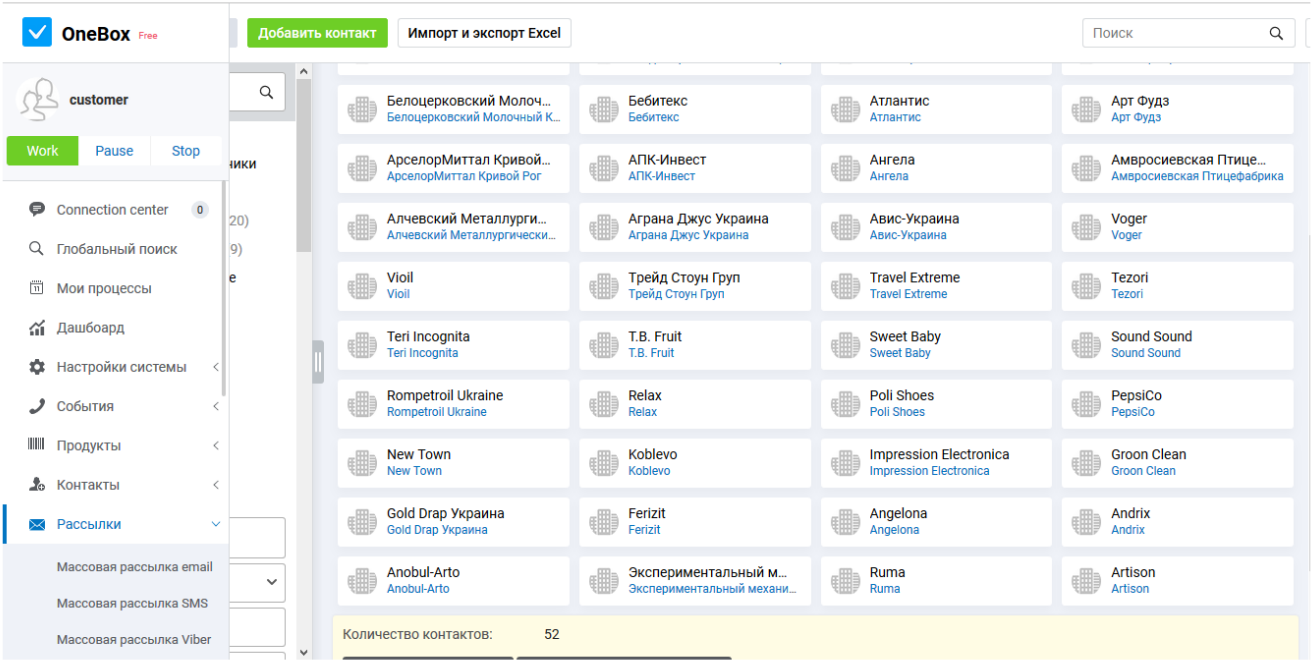


Рис. 2.15 Інтерфейс CRM-системи OneBox.

6. Creatio

Опис: Creatio – єдина платформа для автоматизації CRM, галузевих та внутрішніх процесів за допомогою no-code технологій. Creatio об'єднує можливості системи управління взаємовідносинами з клієнтами та системи управління бізнес-процесами.

- **Ключові особливості:**
  - Управління контактами та взаємодіями.
  - Автоматизація маркетингових кампаній.
  - Управління продажами та сервісом.
  - Інтеграція з іншими бізнес-системами.
- **Переваги:**
  - Потужна платформа з широким функціоналом.
  - Підтримка AI та машинного навчання.
  - Гнучкість налаштувань.
- **Недоліки:**
  - Висока вартість впровадження та підтримки.
  - Складність налаштування для нових користувачів.

Рис. 2.16 Інтерфейс CRM-системи Creatio.

7. TelerCRM

Опис: TelerCRM – це система для автоматизації продажів та управління взаємодіями з клієнтами від компанії Phonet, орієнтована на малий та середній бізнес. Має вбудовану телефонію.

- **Ключові особливості:**
  - Управління контактами та історією взаємодій.
  - Автоматизація обробки лідів.
  - Інтеграція з популярними сервісами.
  - Звіти та аналітика.
- **Переваги:**
  - Доступна вартість.
  - Проста у використанні.
  - Підтримка багатоканальної комунікації.
- **Недоліки:**
  - Обмежена функціональність для великих компаній.

- Менш відомий бренд на ринку.

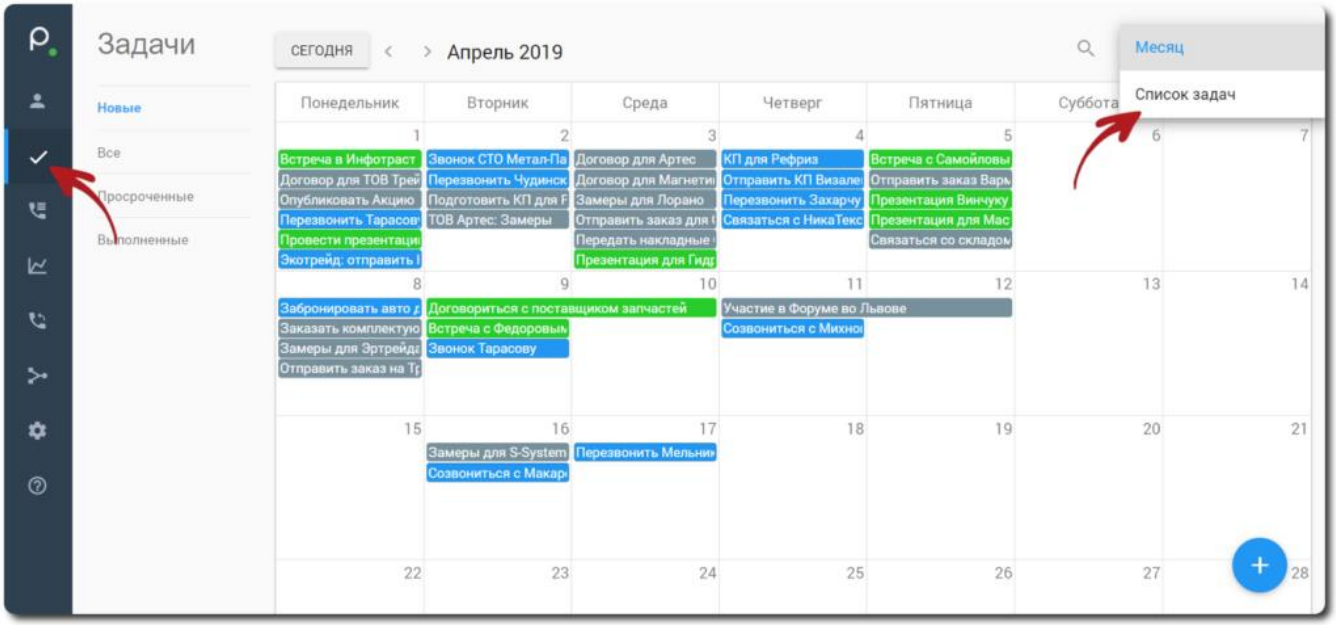


Рис. 2.17 Інтерфейс CRM-системи TelerCRM.

2.3 Порівняльний аналіз популярних CRM-систем

Ринок CRM-систем пропонує широкий вибір рішень для різних потреб та бюджетів. Вибір оптимальної CRM-системи повинен базуватися на ретельному аналізі потреб компанії та функціональних можливостей пропонованих рішень.

Здійснення порівняльного аналізу популярних CRM-платформ допоможе виявити сильні та слабкі сторони кожної системи та вибрати найкраще рішення для конкретних потреб бізнесу. Розглянемо ключові параметри вищезазначених платформ.

Параметр	**Salesforce**	**Microsoft Dynamics 365**	**HubSpot CRM**	**Zoho CRM**	**SAP Customer Experience**
Цільова аудиторія	Всі розміри бізнесу	Великі підприємства	Малі та середні бізнеси	Малі та середні бізнеси	Великі підприємства
Модель впровадження	Хмарне	Хмарне та гібридне	Хмарне	Хмарне та локальне	Хмарне
Основні функції	Управління продажами, маркетингом, обслуговуванням	Управління продажами, маркетингом, обслуговуванням, ERP	Управління контактами, лідами, автоматизація маркетингу	Управління продажами, маркетинг	Управління продажами, маркетингом, обслуговуванням,

		інтеграція		ом, підтримкою клієнтів	персоналізація
Інтеграції	Широкий набір інтеграцій через AppExchange	Інтеграція з продуктами Microsoft, сторонніми додатками	Інтеграція з маркетинговими та комунікаційними інструментами	Інтеграція з Zoho Suite та стороннім и додатками	Інтеграція з SAP та сторонніми додатками
Аналітика та звітність	Потужні аналітичні інструменти та функції прогнозування	Вбудовані аналітичні інструменти, штучний інтелект	Базові аналітичні інструменти	Потужні аналітичні інструмент и	Потужні аналітичні інструменти, персоналізація
Користуваць кий інтерфейс	Гнучкий, але складний	Інтегрований з продуктами Microsoft	Простий та інтуїтивний	Простий та інтуїтивни й	Професійний, може бути складним
Цінова політика	Висока	Висока	Безкоштовни й базовий план, платні розширення	Доступна, є безкоштов ний план	Висока
Підтримка користувачів	24/7 підтримка	24/7 підтримка	Підтримка через електронну пошту та чат	Підтримка через електронн у пошту та чат	24/7 підтримка
Гнучкість та масштабован ість	Висока	Висока	Середня	Висока	Висока

Табл. 2.1 Порівняльний аналіз CRM-систем (частина 1)

Параметр	**Pipedrive**	**Freshworks CRM**	**Insightly**	**SugarCRM**	**Keap**
Цільова аудиторія	Малі та середні бізнеси	Малі та середні бізнеси	Малі та середні бізнеси	Середні та великі бізнеси	Малі бізнеси
Модель впровадження	Хмарне	Хмарне	Хмарне	Хмарне та локальне	Хмарне
Основні функції	Управління продажами	Управління продажами, маркетингом, підтримкою клієнтів	Управління продажами, проектами	Управління продажами, маркетингом, підтримкою клієнтів	Управління контактами, лідами, автоматизація маркетингу
Інтеграції	Інтеграція з популярними інструмента	Інтеграція з продуктами Freshworks та	Інтеграція з популярними інструмента	Інтеграція з багатьма сторонніми додатками	Інтеграція з популярними інструмента

	ми для продажів	сторонніми додатками	ми для продажів		ми для продажів
Аналітика та звітність	Базові аналітичні інструменти	Вбудовані аналітичні інструменти	Потужні аналітичні інструменти	Потужні аналітичні інструменти	Вбудовані аналітичні інструменти
Користувацький інтерфейс	Простий та інтуїтивний	Простий та інтуїтивний	Простий та інтуїтивний	Гнучкий, але може бути складним	Простий та інтуїтивний
Цінова політика	Доступна	Доступна	Доступна	Висока	Доступна
Підтримка користувачів	Підтримка через електронну пошту та чат	Підтримка через електронну пошту та чат	Підтримка через електронну пошту та чат	Підтримка через електронну пошту та чат	Підтримка через електронну пошту та чат
Гнучкість та масштабованість	Середня	Середня	Середня	Висока	Середня

Табл. 2.2 Порівняльний аналіз CRM-систем (частина 2)

Висновки

1. **Salesforce**: Найбільш гнучка та масштабована платформа, підходить для всіх розмірів бізнесу, але має високу вартість. Підходить для компаній, які потребують розширених можливостей та інтеграцій.
2. **Microsoft Dynamics 365**: Ідеально підходить для великих підприємств, особливо тих, що вже використовують продукти Microsoft. Забезпечує потужну інтеграцію з іншими системами Microsoft та великий набір функцій.
3. **HubSpot CRM**: Прекрасний вибір для малого та середнього бізнесу, особливо завдяки безкоштовному базовому плану та простоті використання. Пропонує інтеграції з маркетинговими та комунікаційними інструментами.
4. **Zoho CRM**: Доступне рішення з багатим набором функцій для малого та середнього бізнесу. Пропонує потужні аналітичні інструменти та широкі можливості інтеграції.
5. **SAP Customer Experience (SAP CX)**: Підходить для великих підприємств, які шукають потужні аналітичні можливості та інтеграцію з іншими продуктами SAP.

6. **Pipedrive**: Проста у використанні система, орієнтована на управління продажами. Підходить для малого та середнього бізнесу, які шукають інтуїтивний інтерфейс.

7. **Freshworks CRM**: Легке у використанні рішення для малого та середнього бізнесу з інтегрованими інструментами для комунікацій.

8. **Insightly**: Поєднує CRM та управління проектами, підходить для малого та середнього бізнесу. Пропонує потужні аналітичні можливості.

9. **SugarCRM**: Гнучка система з можливістю високого рівня налаштування, підходить для середніх та великих бізнесів. Відкрита архітектура дозволяє розширювати функціональність.

10. **Keap**: Комплексне рішення для малого бізнесу, що поєднує CRM з автоматизацією маркетингу. Простий та інтуїтивний інтерфейс, доступна ціна.

2.4 Критерії вибору CRM-системи для інтернет-магазину

Вибір конкретної CRM-системи для інтернет-магазину повинен базуватися на аналізі специфічних потреб бізнесу та можливостей різних платформ. Припустимо, що інтернет-магазин має такі основні вимоги:

1. **Інтеграція з популярними eCommerce платформами (наприклад, Shopify, WooCommerce).**
2. **Автоматизація маркетингу (email, SMS, соціальні мережі).**
3. **Інструменти для підтримки клієнтів (живий чат, управління зверненнями).**
4. **Детальна аналітика та звітність.**
5. **Доступна ціна та простота використання.**

На основі цих вимог розглянемо вибір конкретної CRM-системи з уже проаналізованих:

Кандидати:

- **HubSpot CRM**
- **Zoho CRM**
- **Freshworks CRM**
- **Pipedrive**

Порівняння:

1. HubSpot CRM

- **Інтеграція з eCommerce платформами:** Добре інтегрується з Shopify та WooCommerce через власні додатки.
- **Автоматизація маркетингу:** Пропонує потужні інструменти для автоматизації маркетингу, включаючи email-маркетинг, соціальні мережі та SMS-розсилки.
- **Підтримка клієнтів:** Вбудовані інструменти для підтримки клієнтів, живий чат та управління зверненнями.
- **Аналітика та звітність:** Пропонує базові аналітичні інструменти з можливістю розширення у платних планах.
- **Ціна:** Безкоштовний базовий план, платні розширення доступні для розширених функцій.
- **Простота використання:** Інтуїтивно зрозумілий інтерфейс, простий у налаштуванні та використанні.

2. Zoho CRM

- **Інтеграція з eCommerce платформами:** Інтегрується з популярними eCommerce платформами через Zoho Marketplace.
- **Автоматизація маркетингу:** Пропонує інструменти для автоматизації email-маркетингу та соціальних мереж.
- **Підтримка клієнтів:** Вбудовані засоби для підтримки клієнтів, включаючи живий чат.

- **Аналітика та звітність:** Потужні аналітичні інструменти та можливість детального звітування.
- **Ціна:** Доступні різні тарифні плани, включаючи безкоштовну версію.
- **Простота використання:** Простий та інтуїтивний інтерфейс, легко освоїти навіть новачкам.

3. Freshworks CRM

- **Інтеграція з eCommerce платформами:** Інтегрується з популярними eCommerce платформами через Freshworks Marketplace.
- **Автоматизація маркетингу:** Пропонує інструменти для автоматизації email-маркетингу та соціальних мереж.
- **Підтримка клієнтів:** Вбудовані засоби для підтримки клієнтів, включаючи живий чат та підтримку через електронну пошту.
- **Аналітика та звітність:** Вбудовані аналітичні інструменти для відстеження ключових показників.
- **Ціна:** Доступні тарифні плани для малого та середнього бізнесу.
- **Простота використання:** Інтуїтивно зрозумілий інтерфейс, легко освоїти.

4. Pipedrive

- **Інтеграція з eCommerce платформами:** Добре інтегрується з популярними eCommerce платформами через сторонні додатки.
- **Автоматизація маркетингу:** Обмежені можливості для автоматизації маркетингу в базовій версії.
- **Підтримка клієнтів:** Фокусується більше на управлінні продажами, обмежена підтримка клієнтів.
- **Аналітика та звітність:** Базові аналітичні інструменти.
- **Ціна:** Доступна ціна, різні тарифні плани.
- **Простота використання:** Простий та інтуїтивний інтерфейс.

Рекомендована CRM-система: HubSpot CRM

1. **Широкі можливості інтеграції:** Легко інтегрується з популярними платформами eCommerce, такими як Shopify та WooCommerce.
2. **Потужна автоматизація маркетингу:** Пропонує розширені інструменти для автоматизації email-маркетингу, соціальних мереж та SMS.
3. **Інструменти підтримки клієнтів:** Включає живий чат, управління зверненнями та інші інструменти для підтримки клієнтів.
4. **Простота використання:** Інтуїтивно зрозумілий інтерфейс, що дозволяє швидко налаштувати систему без потреби в спеціалізованому навчанні.
5. **Цінова доступність:** Безкоштовний базовий план з можливістю розширення функціоналу через платні додатки.

HubSpot CRM є ідеальним рішенням для інтернет-магазину, який шукає комплексну систему з потужними можливостями для автоматизації маркетингу, підтримки клієнтів та детальної аналітики, при цьому залишаючись доступною та легкою у використанні.

РОЗДІЛ 3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СТВОРЕННЯ CRM-СИСТЕМИ

3.1 Аналіз функціональних вимог

Аналіз базових функціональних вимог CRM-системи є важливим кроком у створенні власної. Розглянемо базові функціональні вимоги до CRM-системи:

1. Управління контактами

Система повинна дозволяти зберігати та оновлювати інформацію про клієнтів, включаючи контактні дані, історію комунікацій, інтереси та інші релевантні дані, а також мати можливість сегментувати клієнтів за різними критеріями для цільових маркетингових кампаній або персоналізованих пропозицій.

2. Управління продажами

Потрібно забезпечити функціональність для управління угодами на всіх етапах

– від початкового контакту до закриття угоди, інструменти для прогнозування майбутніх продажів на основі поточних угод і попереднії даних. Додати звіти і аналітику для оцінки ефективності продажів, продуктивності команди та інших показників.

3. Управління взаємодіями з клієнтами

Забезпечити ведення повної історії всіх комунікацій з клієнтами (дзвінки, листування тощо) та планування майбутніх контактів з клієнтами та автоматичні нагадування про заплановані події.

4. Аналітика та звітність

Зробити налаштовувані дашборди для відстеження ключових показників ефективності (KPI).

5. Користувацький інтерфейс

Розробити інтуїтивно зрозумілий та зручний інтерфейс, який дозволяє швидко навчатися та ефективно працювати. Гарним плюсом буде доступ до системи через мобільні пристрої для роботи в полі.

6. Безпека

Забезпечити різні рівні доступу для користувачів залежно від їх ролей і обов'язків та захист даних через шифрування та інші методи безпеки.

3.2 Архітектура розробленого програмного забезпечення

Для CRM-системи архітектура може бути складною через потребу управління різними типами даних, інтеграцію з іншими системами та забезпечення високої доступності та безпеки. Архітектура програми буде мати декілька рівнів, кожний із своєю зоною відповідальності:

- Презентаційний рівень – відповідає за інтерфейс користувача, включаючи веб-інтерфейс, інтерфейс мобільного додатку, а також інтерфейс для інтеграції з іншими системами.
- Бізнес-логіка – тут знаходяться всі логічні компоненти системи, такі як модулі управління контактами, продажами, маркетингом та

іншими. Цей рівень відповідає за обробку та маніпулювання даними.

- Рівень доступу до даних – цей рівень відповідає за доступ до бази даних або інших джерел даних. Включає в себе моделі даних, об'єктно-реляційне відображення (ORM), запити до бази даних та інші механізми доступу до даних.

CRM-система буде мати модульну структуру, де кожен функціональний блок, такий як управління контактами, продажами, маркетингом, має власний модуль. Це дозволяє забезпечити високу міцність, масштабованість так модульність системи.

Також розділення функціональності на окремі мікросервіси може забезпечити більшу гнучкість та можливість розгортання. Кожен мікросервіс може бути незалежним застосунком, що спрощує розробку, тестування і супровід.

Для безпеки застосунку будемо використовувати такі інструменти як автентифікація та авторизація, що забезпечить відповідний рівень доступу для користувачів. Шифрування даних допоможе захистити конфіденційну інформацію під час зберігання та передачі даних. Від атак реалізуємо такі заходи безпеки як захист від введення з відкритим кодом (SQL injection), міжсайтових атак(XSS), перехоплення сесій та інших загроз.

3.3 Аналіз баз даних, що відповідають вимогам

MySQL — вільна система керування реляційними базами даних. Ця система керування базами даних (СКБД) з відкритим кодом була створена як альтернатива комерційним системам. MySQL з самого початку була дуже схожою на mSQL, проте з часом вона все розширювалася і зараз MySQL — одна з найпоширеніших систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних вебсторінок, оскільки має чудову підтримку з боку різноманітних мов програмування.

MySQL характеризується високою швидкістю, стійкістю і простотою

використання. MySQL вважається гарним рішенням для малих і середніх застосунків. Сирцеві коди сервера компілюються на багатьох платформах. Найповніше можливості сервера виявляються в UNIX-системах, де є підтримка багатопоточності, що підвищує продуктивність системи в цілому.

Переваги MySQL:

Продуктивність: Висока продуктивність для операцій читання.

Простота: Легка у налаштуванні та використанні.

Спільнота та підтримка: Широка спільнота користувачів, багато ресурсів та документації.

Інтеграція: Добре інтегрується з багатьма веб-фреймворками та мовами програмування.

Недоліки:

Масштабованість: Може мати обмеження у вертикальній масштабованості для дуже великих даних.

Обмежена гнучкість: Менш гнучка модель даних порівняно з NoSQL базами.

PostgreSQL, також відома як Postgres, є потужною, об'єктно-реляційною системою управління базами даних (СУБД) з відкритим вихідним кодом. Вона була розроблена для надійності, повноти функціональності та продуктивності, що робить її однією з найпопулярніших баз даних у світі.

Поширюється під ліцензією PostgreSQL, яка є вільною ліцензією, яка дозволяє користувачам використовувати, змінювати та розповсюджувати програмне забезпечення безкоштовно.

Переваги PostgreSQL:

Продуктивність: Висока продуктивність для складних запитів та великих обсягів даних.

Гнучкість: Підтримка JSON, багаті можливості розширення.

Масштабованість: Добре масштабується, підтримка асинхронної та

синхронної реплікації.

Транзакції: Повна підтримка ACID, надійне управління транзакціями.

Недоліки:

Складність: Може бути складніша у налаштуванні та адмініструванні порівняно з MySQL.

Oracle Database, розроблена компанією Oracle Corporation, є однією з найбільш популярних і потужних систем управління базами даних (СУБД) у світі. Вона широко використовується в корпоративних середовищах завдяки своїм можливостям у галузі продуктивності, масштабованості, безпеки та управління даними. Oracle Database є комерційним продуктом, що передбачає наявність платних ліцензій, але також включає підтримку та послуги від Oracle Corporation. Вона є однією з найпотужніших і найнадійніших СУБД на ринку, надаючи багатий набір функцій для задоволення потреб найвибагливіших користувачів.

Переваги Oracle Database:

Продуктивність: Висока продуктивність для великих корпоративних систем.

Надійність: Висока надійність, масштабованість та безпека.

Функціональність: Багаті можливості, включаючи потужний SQL та аналітичні функції.

Недоліки:

Вартість: Висока вартість ліцензії та підтримки.

Складність: Високі вимоги до адміністрування та налаштування.

MongoDB – це популярна документоорієнтована база даних з відкритим вихідним кодом, розроблена для гнучкості та масштабованості. Вона відноситься до категорії NoSQL баз даних і широко використовується в розробці сучасних додатків, які вимагають роботи з великими обсягами даних і високої продуктивності.

MongoDB є потужним інструментом для розробників, яким необхідна

гнучкість у структурі даних, висока продуктивність та можливість масштабування.

Переваги MongoDB:

Гнучкість: Гнучка модель даних, яка добре підходить для роботи з документами.

Масштабованість: Горизонтальне масштабування, підходить для великих обсягів ненормалізованих даних.

Швидкість: Висока продуктивність для операцій запису та читання.

Недоліки:

Транзакції: Обмежена підтримка багатодокументних транзакцій порівняно з реляційними базами.

Цілісність даних: Менш підходить для додатків, які потребують високої транзакційної цілісності.

Обрання бази даних. Провівши порівняння різних СУБД для CRM-системи інтернет-магазину, яка потребує високої транзакційної цілісності, підтримки складних запитів та гнучкої моделі даних, найкращим вибором буде **PostgreSQL**. Ця база даних забезпечує високу продуктивність, гнучкість та надійність, що важливо для управління великими обсягами даних про клієнтів, замовлення та інші важливі аспекти CRM-системи.

3.4 Вибір мови програмування та необхідних бібліотек

Обраною мовою програмування є **JavaScript**, оскільки вона добре підходить для розробки як фронтенд, так і бекенд частини. JavaScript має наступні переваги для написання проєкту:

- JavaScript можна використовувати як для фронтенда (з використанням фреймворків, таких як React, Angular, Vue.js або NextJS), так і для бекенда (з Node.js). Це дозволяє розробникам використовувати одну мову для всього проекту, що спрощує розробку і підтримку коду.

- JavaScript підтримує асинхронне програмування через колбеки, проміси і async/await, що дозволяє ефективно обробляти запити та покращує продуктивність системи.
- **Node.js**, популярне середовище виконання для JavaScript на серверній стороні, використовує подієво-орієнтовану архітектуру, що дозволяє обробляти велику кількість одночасних з'єднань без блокування.
- Існує велика кількість бібліотек та фреймворків для JavaScript, які можуть бути використані для розробки CRM-систем, включаючи бібліотеки для роботи з UI, базами даних, обробкою запитів та іншими аспектами розробки.
- Node Package Manager (NPM) забезпечує доступ до мільйонів пакетів, що можуть бути використані для додавання нових функцій або оптимізації існуючих.
- JavaScript дозволяє створювати модульні додатки, де компоненти можуть бути легко повторно використані та тестовані окремо, що значно пришвидшує процес розробки.
- JavaScript дозволяє створювати модульні додатки, де компоненти можуть бути легко повторно використані та тестовані окремо, що значно пришвидшує процес розробки.

У розробці будуть використовуватись такі основні бібліотеки та технології:

React — відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими стикаються в розробці односторінкових застосунків.

Next.js — це вебфреймворк з відкритим вихідним кодом, створений приватною компанією Vercel, що дозволяє розробляти веб-додатки на основі React із рендерингом на стороні сервера та генерацією статичних веб-сайтів.

Vercel — це платформа для хостингу і розгортання веб-додатків, яка спеціалізується на простоті використання і швидкості. Вона забезпечує повний цикл розробки, включаючи побудову, попередній перегляд і розгортання веб-додатків. Також платформа надає API для взаємодії з БД PostgreSQL.

РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Опис програмного забезпечення

Створення власної CRM-системи важка робота, яка потребує багато сил та часу на реалізацію і повна реалізація усіх можливостей сучасних систем наодинці може зайняти місяці розробки, тому для прикладу у нашій CRM-системі будуть реалізовані базові можливості:

Управління контактами: Можливість додавати, переглядати, редагувати та видаляти дані контактів клієнтів. Це включає в себе базову інформацію про контакт (ім'я, прізвище, електронна пошта).

Управління рахунками: Можливість додавати, переглядати, редагувати та видаляти рахунки клієнтів.

Аналітика: Відображення загальної аналітики про активність клієнтів, рахунки, загальний дохід, тощо.

Система керування користувачами та доступом: Можливість налаштовувати права доступу до різних функціональних частин CRM-системи.

4.2 Налаштування проекту

Для початку створимо новий проект Next.js. Для цього потрібно мати встановлений Node.js та npm. Створення нового проекту відбувається за допомогою команди **npx create-next-app** «назва папки» після виконуючи команду cd «назва папки» переходимо до неї.

```
PS C:\Users\Eugene> npx create-next-app crm
```

Далі нам необхідно встановити всі необхідні бібліотеки командою **npm install** «назва бібліотеки», а саме react, react-dom, next-auth, tailwindcss, typescript, zod, autoprefixer, bcrypt, use-debounce, clsx, postcss.

React-dom — це пакет, що надає DOM-специфічні методи для роботи з React. Він є окремим пакетом поруч з react, і використовується для маніпулювання DOM-деревом у веб-додатках, побудованих з використанням React.

NextAuth.js — це бібліотека для аутентифікації в React-програмах, яка була розроблена для роботи з Next.js. Вона пропонує простий спосіб інтеграції аутентифікації в додатки, підтримуючи різні методи аутентифікації, такі як OAuth, email/password, та соціальні логіни (наприклад, Google, Facebook).

Tailwind CSS — це утилітарний CSS-фреймворк для швидкого створення сучасних, адаптивних інтерфейсів користувача. Основна ідея Tailwind CSS полягає в тому, щоб розробники використовували невеликі, класи-утиліти (utility-first classes), які можна комбінувати для побудови компонентів без необхідності написання кастомного CSS.

TypeScript — це надбудова JavaScript, яка додає статичну типізацію до мови. Він розроблений Microsoft і надає розробникам можливість писати більш надійний та підтримуваний код, зберігаючи при цьому всі переваги JavaScript.

Zod — це бібліотека для валідації схем та даних у JavaScript і TypeScript. Вона дозволяє легко описувати структури даних і перевіряти, чи відповідають їм

об'єкти під час виконання. Основною перевагою Zod є його проста і зрозуміла API для створення та валідації схем.

Autoprefixer — це популярний інструмент для автоматичного додавання вендорних префіксів до CSS властивостей, що забезпечує кросбраузерну сумісність стилів. Він аналізує ваш CSS-код і додає необхідні префікси на основі даних про підтримку браузерами, що дозволяє уникнути ручного додавання префіксів та спрощує процес розробки.

bcrypt — це алгоритм шифрування, призначений для безпечного хешування паролів. Він є одним з найбільш широко використовуваних алгоритмів для забезпечення безпеки паролів у веб-додатках та системах.

use-debounce — це бібліотека для React, яка надає хук useDebounce для зручної реалізації дебаунс-логіки в компонентах. Дебаунс (debounce) — це техніка програмування, яка обмежує частоту виклику функції. Вона особливо корисна для оптимізації обробки подій, які можуть спрацьовувати занадто часто, наприклад, при введенні тексту в інпут або при прокручуванні сторінки.

clsx — це невелика утиліта JavaScript для побудови умовних класів у компонентах. Вона призначена для спрощення маніпуляцій з класами CSS у коді, особливо у React-додатках, але може використовуватися з будь-якими JavaScript-фреймворками або без них.

PostCSS - це інструмент для обробки CSS. Він дозволяє розширювати функціональність CSS, використовуючи плагіни, які міняють або додають правила вихідного CSS. PostCSS дозволяє автоматизувати і оптимізувати роботу з CSS, дозволяючи розробникам використовувати сучасні функції, які можуть не бути підтримані у всіх браузерах.

Heroicons - це набір векторних іконок, призначених для використання в веб-проектах. Вони мають простий і сучасний дизайн і добре підходять для багатьох типів проектів, включаючи веб-додатки, веб-сайти та інші інтерфейси.

vercel/postgres - це офіційний пакет підтримки PostgreSQL для Vercel, який надає можливість легко підключатися до баз даних PostgreSQL з ваших веб-проектів, розгорнутих на платформі Vercel.

4.3 Проектування бази даних

Для початку визначмо, які сутності буде використано у нашому проекті:

Користувач: інформація про користувача (ID, ім’я, електронна пошта, пароль).

Клієнт: інформація про клієнта (ID, ім’я, електронна пошта).

Рахунок: дані про рахунок (ID, клієнт, сума, статус, дата)

Дохід: дані про дохід (місяць, сума доходу)

Далі ми визначаємо, відносини між нашими таблицями:

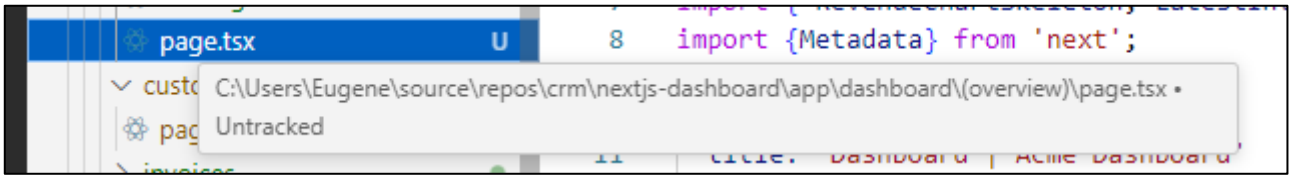
Клієнт та рахунок: один клієнт може мати багато рахунків, тому в таблиці рахунки може бути зовнішній ключ, який посилається на ID клієнта.

4.4 Створення маршрутизації

Маршрутизація у Next.js забезпечує простоту і зручність при створенні динамічних веб-додатків. Вона ґрунтується на файловій системі проекту і дозволяє автоматично створювати маршрути на основі структури папок і файлів у директорії pages. Ось детальний огляд того, як влаштована маршрутизація у Next.js:

Файлова система як основа маршрутизації:

- Кожен файл у папці ***pages*** автоматично стає маршрутом.
- Наприклад, dashbord/page.tsx — маршруту /dashboard.



Для початку ми створимо маршрути для основних сторінок у CRM системі:

- /dashboard — основний маршрут, який вкладає у себе інші основні маршрути

- /customers — основний маршрут для взаємодії з клієнтами
- /invoices — основний маршрут для взаємодії з рахунками

Далі створюємо маршрути для авторизації та маршрути, які вкладемо в основні:

- /login — маршрут для сторінки авторизації
- /customers/[id] — маршрут для перегляду конкретного клієнту
- /customers/create — маршрут для створення нового клієнту
- /invoices/[id] — маршрут для перегляда конкретного рахунку
- /invoices/create — маршрут для створення нового рахунку

4.4 Створення інтерфейсу користувача

Перш за все нам треба подумати за навігацію нашим застосунком. Для цього у лівій частині нашої сторінки ми будемо відображати список доступних сторінок. Щоб реалізувати це ми створюємо компонент **Layout**, який вміщає компонент навігації **SideNav** та простий **div**, у якому ми будемо рендерити решту сторінки.

SideNav складається з компонентів **CRMLogo**, який відображає логотип нашої CRM, та **NavLinks**, що відповідає за рендер списку доступних сторінок.

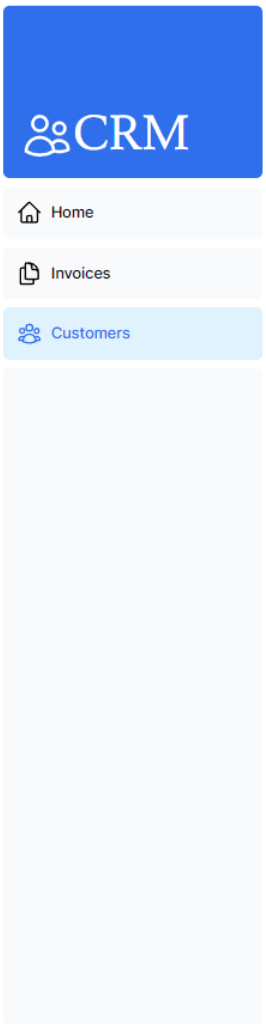


Рис. 4.1 Навігація

Основною сторінкою нашої CRM-системи являється сторінка **dashboard**, або ж приладова дошка. По суті, dashboard - це звіт. Ключове відмінність його від класичного аналога в динамічному форматі. Традиційний звіт складається на конкретний момент часу. Вже через кілька хвилин дані можуть бути застарілими і не актуальними.

Dashboard є програмним рішенням, яке забезпечує постійний збір і постійне оновлення інформації. Тобто на будь-який момент часу найважливіші показники будуть свіжими і актуальними.

Для створення сторінки dashboard ми визначаємо, яку саме інформацію вона буде відтворювати – останній дохід, останні рахунки, скільки всього зароблено, яка сума очікує підтвердження, загальна кількість рахунків та клієнтів. Також важливою частиною являється навігація по сторінці.

Розділимо dashboard на декілька частин, верхню частину виділимо для інформації, яка не займає багато місця (дані про прибуток, клієнтів, рахунки) і обернемо її у контейнер. Основну частину залишимо для діаграми доходу та останніх рахунків.

Для реалізації цього йдемо з найменшого компоненту до цілої сторінки:

1. Створюємо компонент **card** – маленьку цеглину, яка здатна відображати просту інформацію, наприклад загальну кількість клієнтів.
2. Далі створюємо компонент **CardWrapper** і наповнюємо його компонентами **cards** із інформацією, яку ми витягуємо з нашої БД.
3. Створюємо компонент **RevenueChart** з інформацією про прибуток за останні 12 місяці.
4. Створюємо компонент **LatestInvoices** з інформацією про останні сплачені рахунки.
5. Створюємо сторінку **Page** нашого dashboard та додаємо компоненти CardWrapper, RevenueChart, LatestInvoices попередньо обернувши їх компонентом **Suspense** – компонентом React, який дозволяє вказувати запасний компонент, поки інформація не завантажиться.

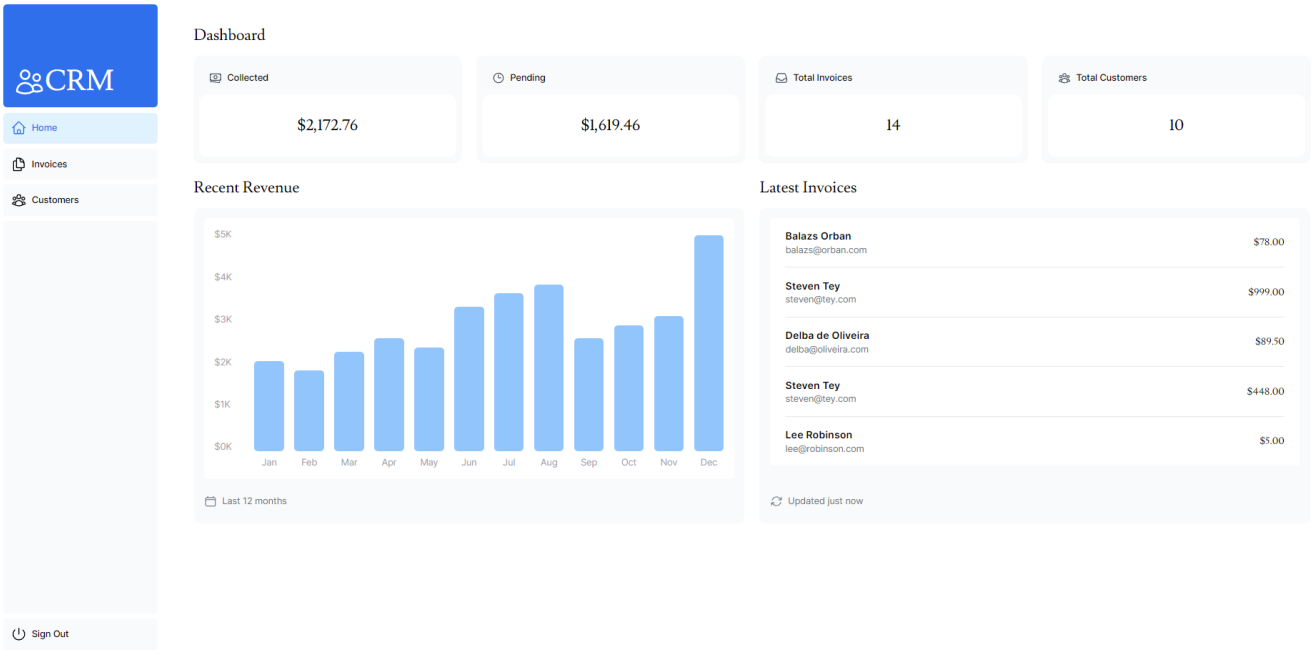


Рис. 4.2 Сторінка dashboard

Іншою важливою сторінкою нашого застосунка являється сторінка **customers –** вона відображає усіх наших клієнтів у вигляді таблиці з їх даними. Орієнтування здійснюється за допомогою пагінації та пошуку.

1. Створюємо компонент **Search**, який відповідає за пошук певного клієнта.
2. Далі створюємо компонент з кнопкою **CreateCustomer**, яка дозволяє створити нового клієнта.
3. Компонент **Table** із списком клієнтів, який обертаємо у **Suspense**
4. Компонент пагінації – **Pagination**, до складу якого входять компоненти **PaginationNumber** та **PaginationArrow**
5. Створюємо компонент сторінки – **Page** та розміщаємо попередньо створені компоненти всередину.

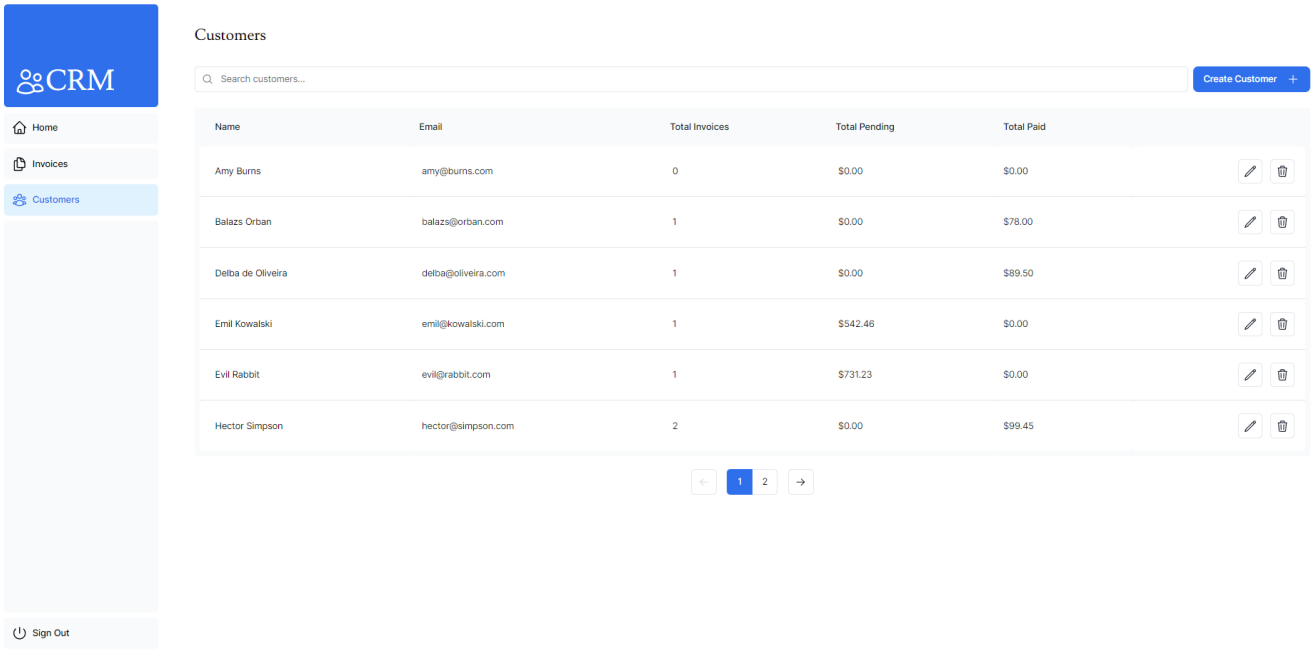


Рис. 4.3 Сторінка customers

Третьою важливою ланкою нашого проекту являється **invoices** – сторінка, що виводить дані по усіх рахунках клієнтів, які так само можна відфільтрувати за допомогою пошуку.

Ця сторінка дуже подібна до попередньої, тому деякі компоненти ми просто використовуємо без змін.

1. Компонент **Search** ми вже попередньо створили, тому просто беремо його.
2. З компонентом **Pagination** так само.
3. Створюємо компонент з кнопкою **CreateInvoice**, для створення нового рахунку.
4. Створюємо **Table** із списком рахунків, обертаємо у **Suspense**
5. Далі йде **Page** із розміщеними компонентами.

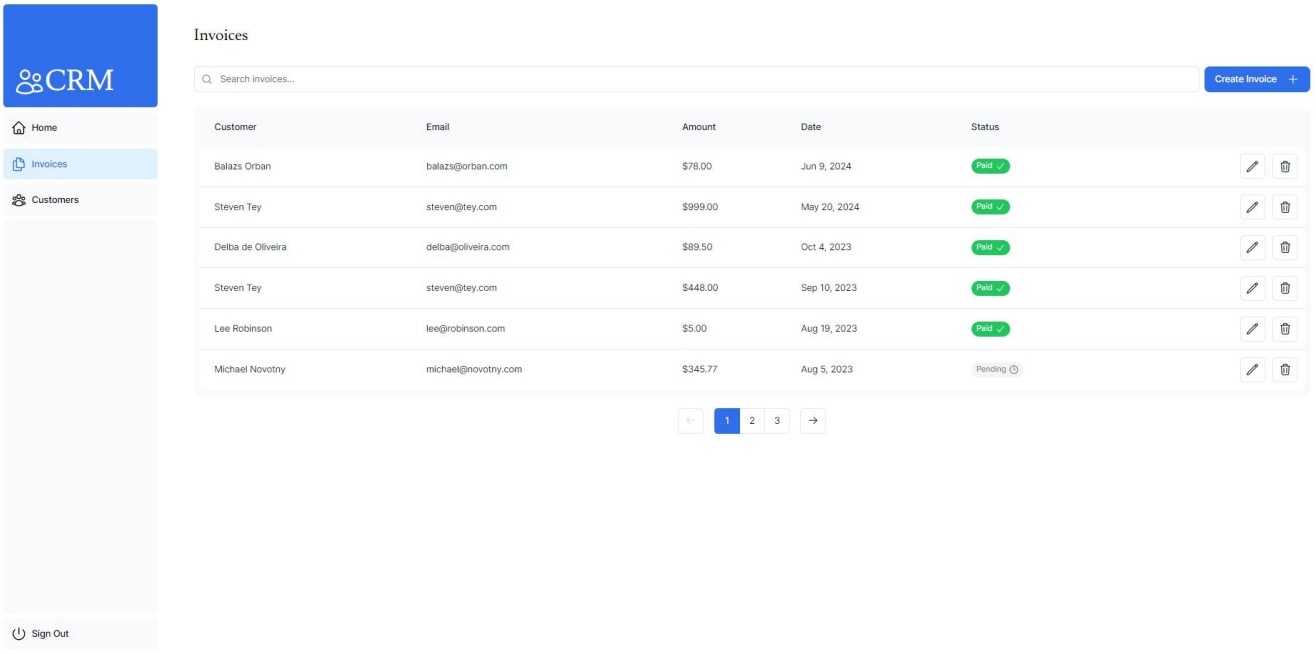


Рис. 4.4 Сторінка invoices

Ще однією важливою сторінкою є сторінка авторизації – **login**, яка буде зустрічати нових користувачів CRM-системи.

1. Вже створений компонент з логотипом CRM **CRMLogo**
2. Компонент **LoginForm** із формою авторизацією.
3. **LoginButton** компонент кнопка для входу до системи, яку розміщуємо в **LoginForm**.
4. На останньому компоненті **LoginPage** розміщаємо **LoginForm** та **CRMLogo.**

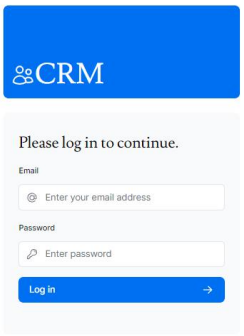


Рис. 4.5 Сторінка login

4.5 Підключення до бази даних

Для підключення нашої CRM-системи до обраної бази даних ми використовуємо **Vercel:**

1. Створюємо обліковий запис **GitHub**, або авторизуємся, якщо вже створений.
2. Розміщуємо свій проект у репозиторії на **GitHub**
3. Таким же чином створюємо обліковий запис **Vercel** та прив’язуємо його до акаунту **GitHub.**
4. Імпортуємо наш репозиторій та розміщуємо його на **Vercel.**

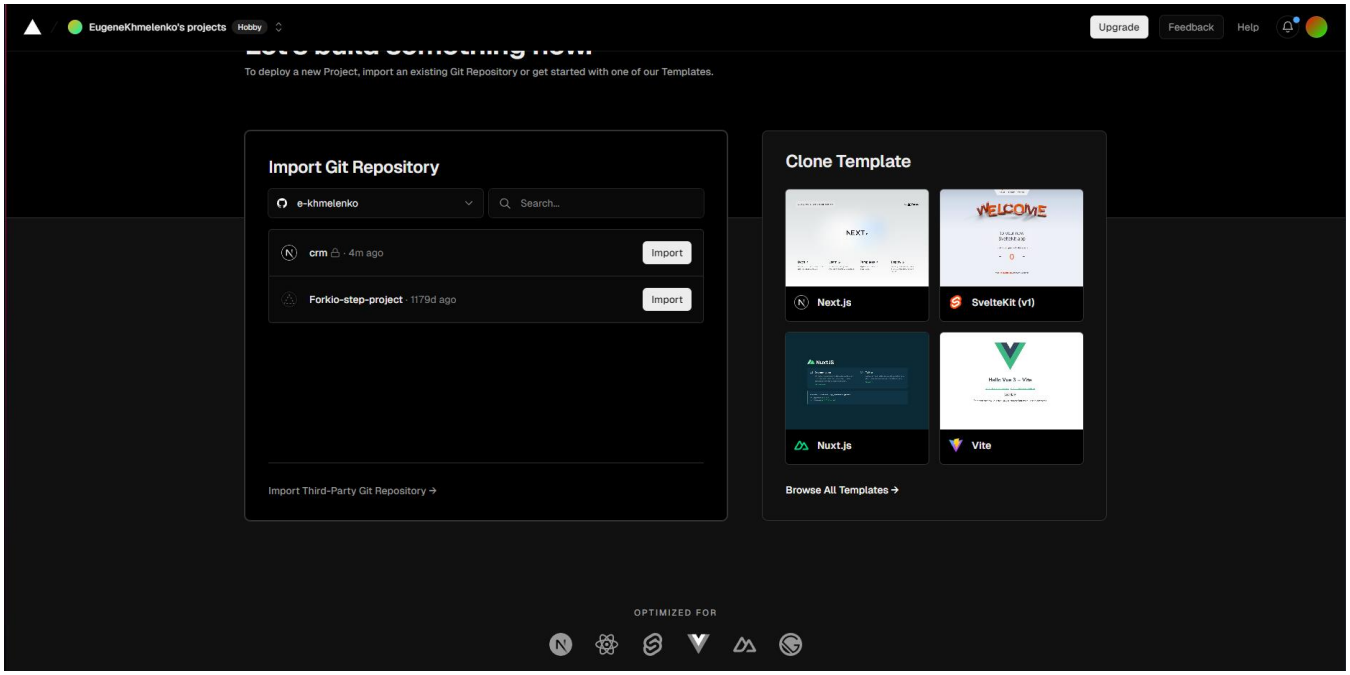


Рис. 4.6 Процес підключення бази даних (1/3)

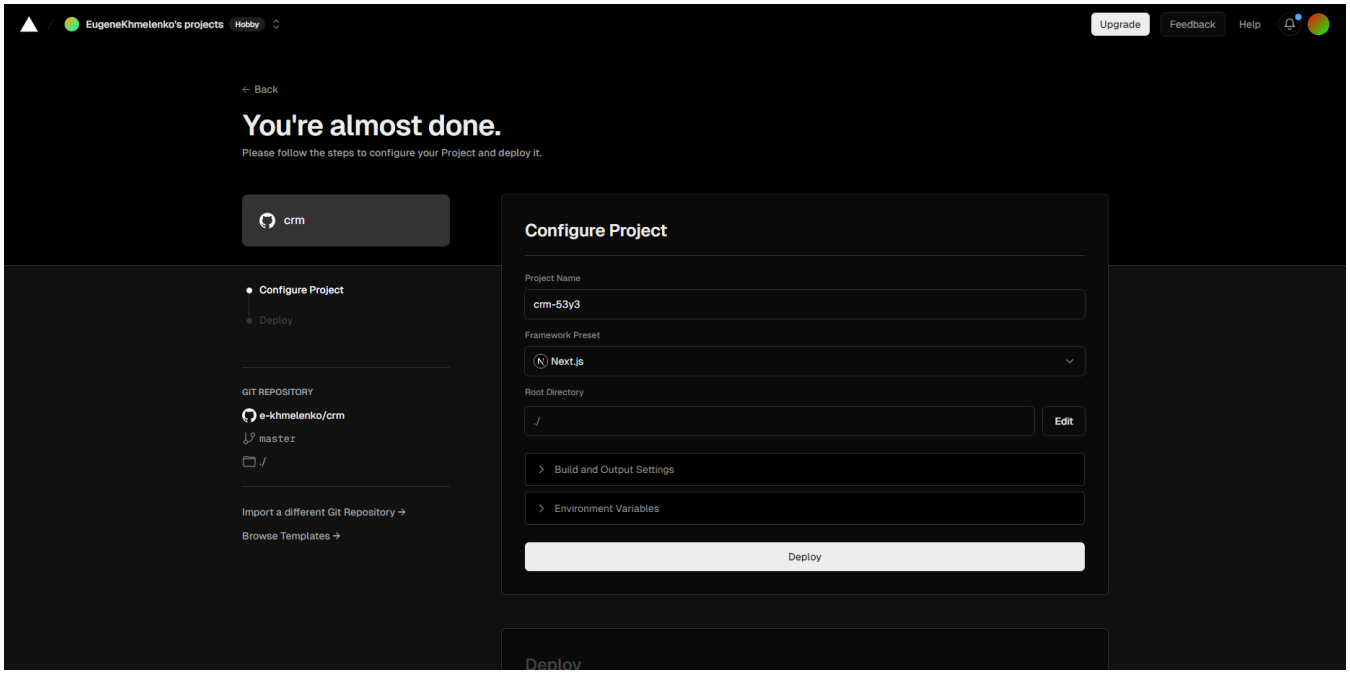


Рис. 4.7 Процес підключення бази даних (2/3)

5. Далі налаштовуємо базу даних (Connect Store → Create New → Postgres → Continue).

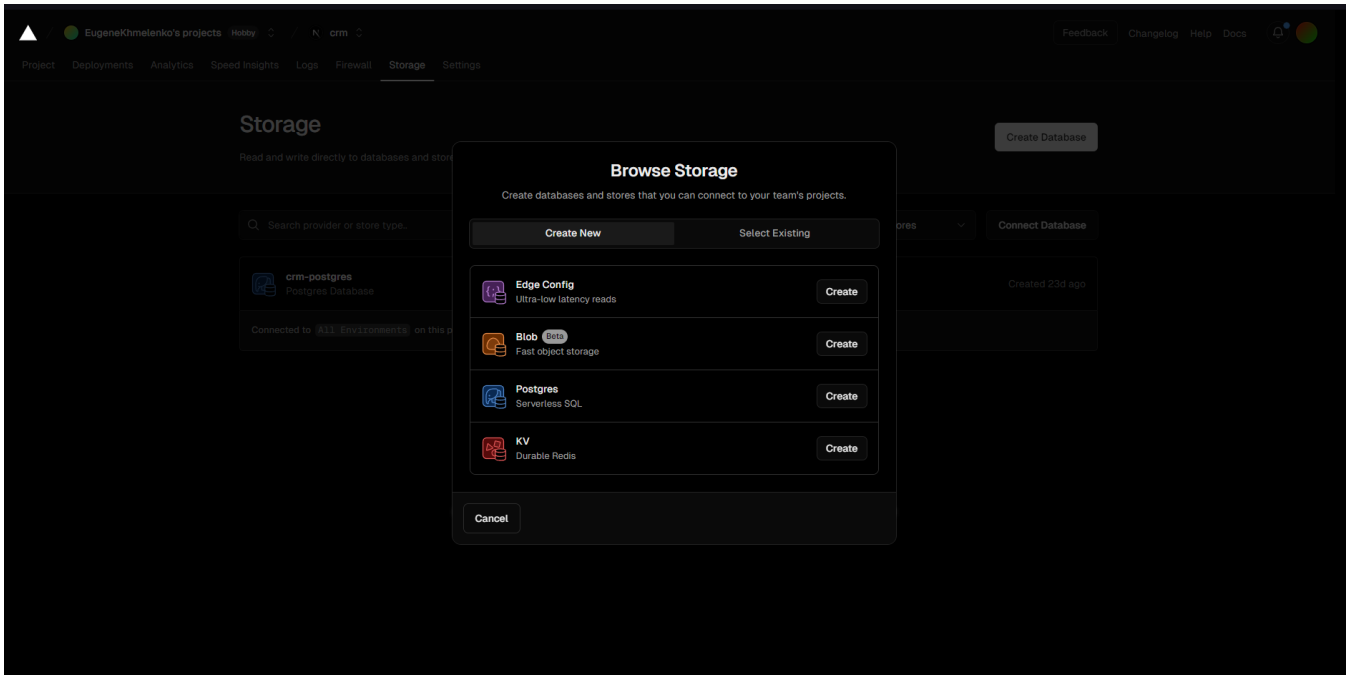


Рис. 4.8 Процес підключення бази даних (3/3)

6. Налаштовуємо змінну середовища (.env).
7. Командою ***npm i @vercel\postgres*** встановлюємо API Vercel Postgres SDK, яке надає можливість взаємодіяти з нашою БД Postgres.

4.6 Реалізація бізнес-логіки

Створивши інтерфейс сторінок, наступним кроком буде наповнення їх логікою. Наша CRM-система матиме можливість створювати клієнтів, рахунки, редагувати та видаляти їх.

Для створення рахунку нам потрібна відповідна форма. Створюємо її у компоненті **Form**. Основними даними, які ми будемо заповнювати – це ім'я клієнту, сума рахунку, дата та статус рахунку.

1. Ім’я клієнту – select із можливістю обрати будь-якого клієнта із нашої бази даних.
2. Сума рахунку – input із типом number.
3. Статус рахунку – input із типом radio.
4. Дата – автоматично заповнюється при створенні.
5. Додаємо кнопку, яка відправляє дані та кнопку відміни.

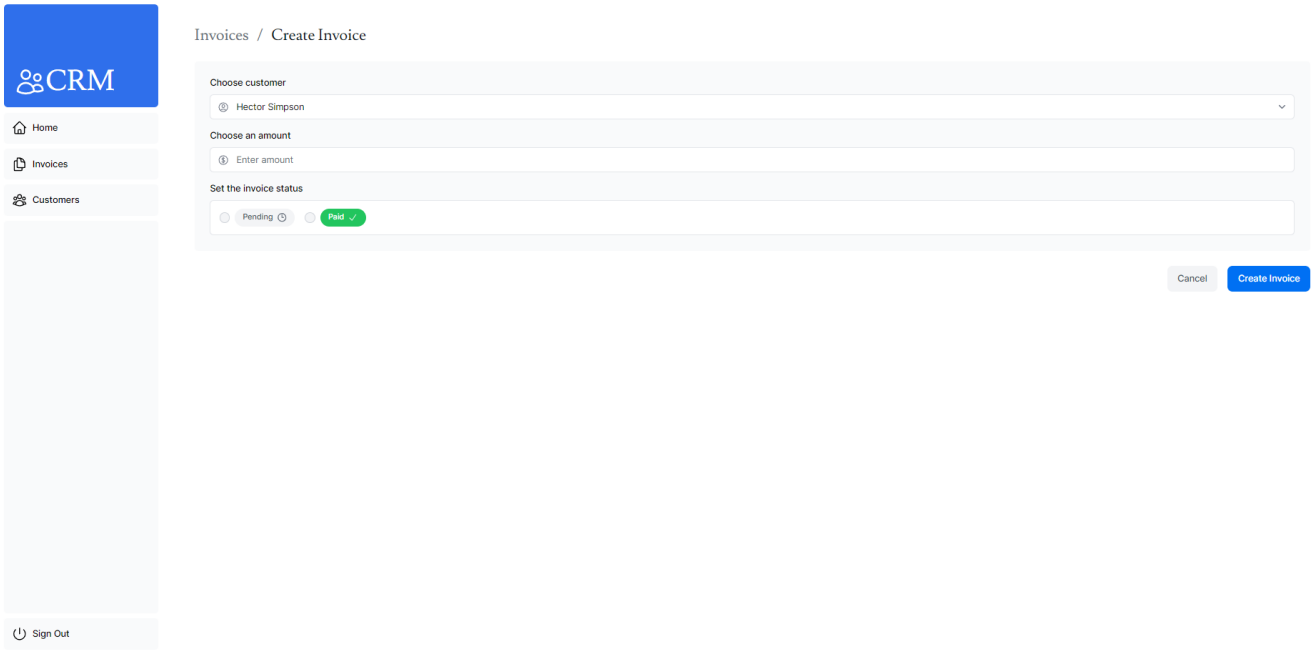


Рис. 4.9 Сторінка invoices/create

Редагування рахунку відбувається при натисканні відповідної кнопки на таблиці із рахунками на сторінці invoices. Для цього створюємо форму **EditInvoiceForm**, яка дуже схожа на попередню, за вийнятком попередньо заповнених даних у формі.

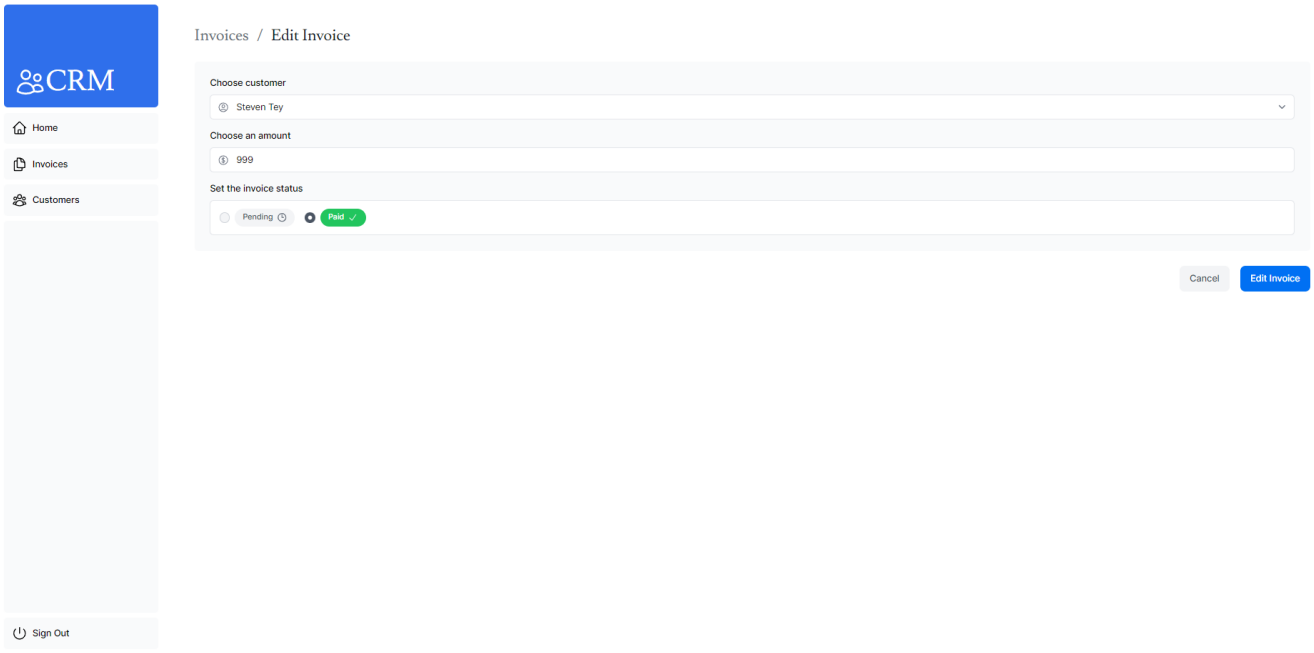


Рис. 4.10 Сторінка invoices/[id]/edit

Аналогічно для створення клієнта створюємо відповідну форму **Form**:

1. Ім’я клієнту – input із типом string.
2. Електронна пошта – input із типом email.
3. Кнопка для відправки даних і кнопка відміни.

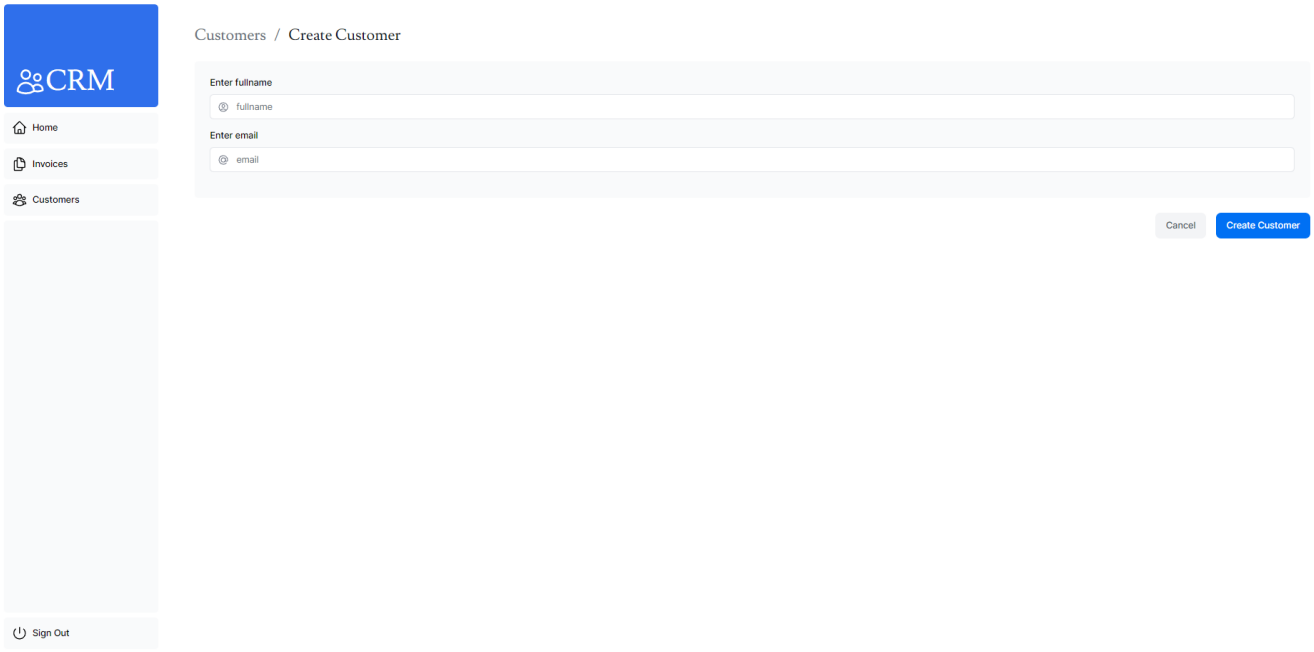


Рис. 4.11 Сторінка customers/create

ВИСНОВКИ

Із проведеної роботи можна зробити висновок, що впровадження CRM-системи в роботу інтернет-магазину є стратегічно важливим кроком, який дозволяє підвищити ефективність бізнес-процесів, покращити якість обслуговування клієнтів та збільшити продажі. Розгляд світових та українських CRM-систем показав, що ринок пропонує широкий вибір рішень, які можуть задовольнити потреби компаній різного розміру та галузей. Основними перевагами CRM-систем є автоматизація рутинних завдань, інтеграція з іншими бізнес-системами, потужні аналітичні інструменти та можливість персоналізації взаємодії з клієнтами. Загалом, впровадження CRM-системи в роботу інтернет-магазину є інвестицією, яка окупається за рахунок підвищення ефективності бізнес-процесів та збільшення прибутків.

Завдяки дослідженню зв’ясовано, що CRM є найбільшим ринком програмного забезпечення і має тенденцію тільки зростати. Незважаючи на низький рівень впровадження CRM-систем в Україні, існують значні перспективи для зростання ринку. Основними викликами є економічна нестабільність, недостатня поінформованість та інфраструктурні обмеження. Проте, зростаюча цифровізація та інтеграція нових технологій відкривають великі можливості для українських компаній.

Провівши аналіз існуючих рішень, надано рекомендації щодо вибору CRM-системи для інтеграції у роботу інтернет-магазину. На основі досліджених вимог було створено приклад реалізації CRM-системи з базовим функціоналом.

Список використаних джерел

1. How Many Companies Use Salesforce? Total Customer Number in 2024 **https://ascendix.com/blog/how-many-companies-use-salesforce/**
2. How Many Microsoft Dynamics Customers Are There? **https://uds.systems/blog/how-many-microsoft-dynamics-customers-are-there/**
3. HubSpot User and Revenue Stats **https://backlinko.com/hubspot-users**
4. Zoho CRM – 22 Amazing Stats and Facts **https://www.helloleads.io/blog/stats-facts/zoho-crm-22-amazing-stats-facts/**
5. Zoho CRM **https://www.zoho.com/crm/all-in-one-crm.html**
6. Diving Deep Into SAP CX: Exploring The Five SAP CX Solutions And Their Applications **https://www.spadoom.com/en/blog/exploring-the-five-core-sap-cx-solutions/**
7. 19 Pipedrive CRM Stats **https://startupgeek.com/blog/pipedrive/**
8. Freshworks Revenue and Growth Statistics (2024) **https://www.usesignhouse.com/blog/freshworks-stats**
9. Insightly CRM – 25 Amazing Stats and Facts **https://www.helloleads.io/blog/stats-facts/insightly-crm-25-amazing-stats-and-facts/**
10. How SugarCRM hit $73.5M revenue and 2M customers in 2023. **https://getlatka.com/companies/sugarcrm#customers**
11. SugarCRM Revenue and Competitors **https://growjo.com/company/SugarCRM**
12. We understand entrepreneurs because that’s how we started **https://keap.com/about#**
13. Keap CRM Review 2023: Overview, Features, & Pricing **https://ubiquedigitalsolutions.com/blog/keap-crm-review-features-pricing/**
14. Customer Relationship Management Market Size, Share, & Trends Analysis

Report, By Component, By Solution, By Deployment, By Enterprise Size, By End Use, And Segment Forecasts, 2024 – 2030 **https://www.grandviewresearch.com/industry-analysis/customer-relationship-management-crm-market**

15. 20 Impressive CRM Statistics You Need to Know in 2024 **https://www.webfx.com/blog/marketing/crm-statistics/**
16. CRM Statistics: The Uses, Benefits & Challenges of Customer Relationship Management Platforms **https://martech.zone/crm-statistics/**
17. CRM Market Share Report https://crmsearch.com/crm/crm-software-market-share/
18. "CRM at the Speed of Light: Social CRM Strategies, Tools, and Techniques for Engaging Your Customers" by Paul Greenberg.
19. "Customer Relationship Management: Concepts and Technologies" by Francis Buttle, Stan Maklan
20. "The Ultimate CRM Handbook" by Barton Goldenberg
21. Як змінився рейтинг CRM в Україні за час війни у 2022 році https://ain.ua/2023/03/10/yak-zminyvsya-rejtyng-crm-v-ukrayini-za-chas-vijny-u-2022-roczi/
22. Що відбувається на українському ринку CRM. Огляд у розрізі iForum https://speka.media/shho-vidbuvajetsya-na-ukrayinskomu-rinku-crm-oglyad-u-rozrizi-iforum-9xdorv
23. CRM для продажу товарів, для інтернет-магазину https://salesdrive.ua/
24. Чому KeyCRM? https://ua.keycrm.app/our-story
25. Зрозуміла CRM для керування компанії https://keepincrm.com/
26. PERFECTUM CRM+ERP СИСТЕМА ДЛЯ ВСІЄЇ КОМПАНІЇ https://perfectum.ua/ua/
27. Українські CRM системи для бізнесу https://1b.app/ua/
28. Як автоматизувати бізнес за допомогою CRM-системи https://business.diia.gov.ua/cases/sistematizacia-biznes-procesiv/ak-avtomatizuvati-biznes-za-dopomogou-crm-sistemi

ДОДАТКИ

Додаток А

```
{
  "private": true,
  "scripts": {
    "build": "next build",
    "dev": "next dev",
    "prettier": "prettier --write --ignore-unknown .",
    "prettier:check": "prettier --check --ignore-unknown .",
    "start": "next start",
    "seed": "node -r dotenv/config ./scripts/seed.js",
    "lint": "next lint"
  },
  "dependencies": {
    "@heroicons/react": "^2.0.18",
    "@tailwindcss/forms": "^0.5.7",
    "@types/node": "20.5.7",
    "@vercel/postgres": "^0.5.1",
    "autoprefixer": "10.4.15",
    "bcrypt": "^5.1.1",
    "clsx": "^2.0.0",
    "next": "^14.0.2",
    "next-auth": "^5.0.0-beta.18",
    "postcss": "8.4.31",
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "tailwindcss": "3.3.3",
    "typescript": "5.2.2",
    "use-debounce": "^10.0.0",
    "zod": "^3.22.2"
  },
  "devDependencies": {
    "@types/bcrypt": "^5.0.1",
    "@types/react": "18.2.21",
    "@types/react-dom": "18.2.14",
    "@vercel/style-guide": "^5.0.1",
    "dotenv": "^16.3.1",
    "eslint": "^8.52.0",
    "eslint-config-next": "^14.0.0",
    "eslint-config-prettier": "9.0.0",
    "prettier": "^3.0.3",
    "prettier-plugin-tailwindcss": "0.5.4"
  },
  "engines": {
    "node": ">=18.17.0"
  }
}
```

```
import NextAuth from 'next-auth';
```

```ts
import { authConfig } from './auth.config';
import Credentials from 'next-auth/providers/credentials';
import {z} from 'zod';
import { sql } from '@vercel/postgres';
import type { User } from '@/app/lib/definitions';
import bcrypt from 'bcrypt';

async function getUser(email: string): Promise<User | undefined> {
    try {
      const user = await sql<User>`SELECT * FROM users WHERE email=${email}`;
      return user.rows[0];
    } catch (error) {
      console.error('Failed to fetch user:', error);
      throw new Error('Failed to fetch user.');
    }
  }

  export const { auth, signIn, signOut } = NextAuth({
    ...authConfig,
    providers: [
      Credentials({
        async authorize(credentials) {
          const parsedCredentials = z
            .object({ email: z.string().email(), password: z.string().min(6) })
            .safeParse(credentials);

          if (parsedCredentials.success) {
            const { email, password } = parsedCredentials.data;
            const user = await getUser(email);
            if (!user) return null;
            const passwordsMatch = await bcrypt.compare(password, user.password);

            if(passwordsMatch) return user;
          }

          console.log('Invalid credentials')
          return null;
        },
      }),
    ],
  });
```

```ts
import type { NextAuthConfig } from 'next-auth';

export const authConfig = {
  pages: {
    signIn: '/login',
  },
  callbacks: {
    authorized({ auth, request: { nextUrl } }) {
```

```
      const isLoggedIn = !!auth?.user;
      const isOnDashboard = nextUrl.pathname.startsWith('/dashboard');
      if (isOnDashboard) {
        if (isLoggedIn) return true;
        return false; // Redirect unauthenticated users to login page
      } else if (isLoggedIn) {
        return Response.redirect(new URL('/dashboard', nextUrl));
      }
      return true;
    },
  },
  providers: [], // Add providers with an empty array for now
} satisfies NextAuthConfig;
```

```
import NextAuth from 'next-auth';
import { authConfig } from './auth.config';

export default NextAuth(authConfig).auth;

export const config = {
  // https://nextjs.org/docs/app/building-your-
application/routing/middleware#matcher
  matcher: ['/((?!api|_next/static|_next/image|.*\\.png$).*)'],
};
```

```
{
  "compilerOptions": {
    "target": "es5",
    "lib": ["dom", "dom.iterable", "esnext"],
    "allowJs": true,
    "skipLibCheck": true,
    "strict": true,
    "noEmit": true,
    "esModuleInterop": true,
    "module": "esnext",
    "moduleResolution": "bundler",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "jsx": "preserve",
    "incremental": true,
    "plugins": [
      {
        "name": "next"
      }
    ],
    "paths": {
      "@/*": ["./*"]
    }
  },
  "include": [
```

```
    "next-env.d.ts",
    "**/*.ts",
    "**/*.tsx",
    ".next/types/**/*.ts",
    "app/lib/placeholder-data.js",
    "scripts/seed.js"
  ],
  "exclude": ["node_modules"]
}
```

```
import type { Config } from 'tailwindcss';

const config: Config = {
  content: [
    './pages/**/*.{js,ts,jsx,tsx,mdx}',
    './components/**/*.{js,ts,jsx,tsx,mdx}',
    './app/**/*.{js,ts,jsx,tsx,mdx}',
  ],
  theme: {
    extend: {
      gridTemplateColumns: {
        '13': 'repeat(13, minmax(0, 1fr))',
      },
      colors: {
        blue: {
          400: '#2589FE',
          500: '#0070F3',
          600: '#2F6FEB',
        },
      },
    },
    keyframes: {
      shimmer: {
        '100%': {
          transform: 'translateX(100%)',
        },
      },
    },
  },
  plugins: [require('@tailwindcss/forms')],
};
export default config;
```

```
const styleguide = require('@vercel/style-guide/prettier');

module.exports = {
  ...styleguide,
  plugins: [...styleguide.plugins, 'prettier-plugin-tailwindcss'],
```

```
};
module.exports = {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  },
};
```

```
POSTGRES_URL="postgres://default:TUMgGuD7Bv4r@ep-shy-feather-a2o7qkcc-pooler.eu-central-1.aws.neon.tech:5432/verceldb?sslmode=require"
POSTGRES_PRISMA_URL="postgres://default:TUMgGuD7Bv4r@ep-shy-feather-a2o7qkcc-pooler.eu-central-1.aws.neon.tech:5432/verceldb?sslmode=require&pgbouncer=true&connect_timeout=15"
POSTGRES_URL_NO_SSL="postgres://default:TUMgGuD7Bv4r@ep-shy-feather-a2o7qkcc-pooler.eu-central-1.aws.neon.tech:5432/verceldb"
POSTGRES_URL_NON_POOLING="postgres://default:TUMgGuD7Bv4r@ep-shy-feather-a2o7qkcc.eu-central-1.aws.neon.tech:5432/verceldb?sslmode=require"
POSTGRES_USER="default"
POSTGRES_HOST="ep-shy-feather-a2o7qkcc-pooler.eu-central-1.aws.neon.tech"
POSTGRES_PASSWORD="TUMgGuD7Bv4r"
POSTGRES_DATABASE="verceldb"

# `openssl rand -base64 32`
AUTH_SECRET= 'Qw/BxTPm/5D+FbXVtm+sqBFGIaP1G2hRGCvcEmpUzZ4='
AUTH_URL=http://localhost:3000/api/auth
```

```
// Loading animation
const shimmer =
  'before:absolute before:inset-0 before:-translate-x-full before:animate-[shimmer_2s_infinite] before:bg-gradient-to-r before:from-transparent before:via-white/60 before:to-transparent';

export function CardSkeleton() {
  return (
    <div
      className={`${shimmer} relative overflow-hidden rounded-xl bg-gray-100 p-2 shadow-sm`}
    >
      <div className="flex p-4">
        <div className="h-5 w-5 rounded-md bg-gray-200" />
        <div className="ml-2 h-6 w-16 rounded-md bg-gray-200 text-sm font-medium" />
      </div>
      <div className="flex items-center justify-center truncate rounded-xl bg-white px-4 py-8">
        <div className="h-7 w-20 rounded-md bg-gray-200" />
      </div>
    </div>
  );
```

```
}

export function CardsSkeleton() {
  return (
    <>
      <CardSkeleton />
      <CardSkeleton />
      <CardSkeleton />
      <CardSkeleton />
    </>
  );
}

export function RevenueChartSkeleton() {
  return (
    <div className={`${shimmer} relative w-full overflow-hidden md:col-span-4`}>
      <div className="mb-4 h-8 w-36 rounded-md bg-gray-100" />
      <div className="rounded-xl bg-gray-100 p-4">
        <div className="mt-0 grid h-[410px] grid-cols-12 items-end gap-2 rounded-md
bg-white p-4 sm:grid-cols-13 md:gap-4" />
        <div className="flex items-center pb-2 pt-6">
          <div className="h-5 w-5 rounded-full bg-gray-200" />
          <div className="ml-2 h-4 w-20 rounded-md bg-gray-200" />
        </div>
      </div>
    </div>
  );
}

export function InvoiceSkeleton() {
  return (
    <div className="flex flex-row items-center justify-between border-b border-
gray-100 py-4">
      <div className="flex items-center">
        <div className="mr-2 h-8 w-8 rounded-full bg-gray-200" />
        <div className="min-w-0">
          <div className="h-5 w-40 rounded-md bg-gray-200" />
          <div className="mt-2 h-4 w-12 rounded-md bg-gray-200" />
        </div>
      </div>
      <div className="mt-2 h-4 w-12 rounded-md bg-gray-200" />
    </div>
  );
}

export function LatestInvoicesSkeleton() {
  return (
    <div
      className={`${shimmer} relative flex w-full flex-col overflow-hidden md:col-
span-4`}
    >
```

```
      <div className="mb-4 h-8 w-36 rounded-md bg-gray-100" />
      <div className="flex grow flex-col justify-between rounded-xl bg-gray-100 p-
4">
        <div className="bg-white px-6">
          <InvoiceSkeleton />
          <InvoiceSkeleton />
          <InvoiceSkeleton />
          <InvoiceSkeleton />
          <InvoiceSkeleton />
          <div className="flex items-center pb-2 pt-6">
            <div className="h-5 w-5 rounded-full bg-gray-200" />
            <div className="ml-2 h-4 w-20 rounded-md bg-gray-200" />
          </div>
        </div>
      </div>
    </div>
  );
}

export default function DashboardSkeleton() {
  return (
    <>
      <div
        className={`${shimmer} relative mb-4 h-8 w-36 overflow-hidden rounded-md
bg-gray-100`}
      />
      <div className="grid gap-6 sm:grid-cols-2 lg:grid-cols-4">
        <CardSkeleton />
        <CardSkeleton />
        <CardSkeleton />
        <CardSkeleton />
      </div>
      <div className="mt-6 grid grid-cols-1 gap-6 md:grid-cols-4 lg:grid-cols-8">
        <RevenueChartSkeleton />
        <LatestInvoicesSkeleton />
      </div>
    </>
  );
}

export function TableRowSkeleton() {
  return (
    <tr className="w-full border-b border-gray-100 last-of-type:border-none
[&:first-child>td:first-child]:rounded-tl-lg [&:first-child>td:last-child]:rounded-
tr-lg [&:last-child>td:first-child]:rounded-bl-lg [&:last-child>td:last-
child]:rounded-br-lg">
      {/* Customer Name and Image */}
      <td className="relative overflow-hidden whitespace-nowrap py-3 pl-6 pr-3">
        <div className="flex items-center gap-3">
          <div className="h-8 w-8 rounded-full bg-gray-100"></div>
          <div className="h-6 w-24 rounded bg-gray-100"></div>
```

```
        </div>
      </td>
      {/* Email */}
      <td className="whitespace-nowrap px-3 py-3">
        <div className="h-6 w-32 rounded bg-gray-100"></div>
      </td>
      {/* Amount */}
      <td className="whitespace-nowrap px-3 py-3">
        <div className="h-6 w-16 rounded bg-gray-100"></div>
      </td>
      {/* Date */}
      <td className="whitespace-nowrap px-3 py-3">
        <div className="h-6 w-16 rounded bg-gray-100"></div>
      </td>
      {/* Status */}
      <td className="whitespace-nowrap px-3 py-3">
        <div className="h-6 w-16 rounded bg-gray-100"></div>
      </td>
      {/* Actions */}
      <td className="whitespace-nowrap py-3 pl-6 pr-3">
        <div className="flex justify-end gap-3">
          <div className="h-[38px] w-[38px] rounded bg-gray-100"></div>
          <div className="h-[38px] w-[38px] rounded bg-gray-100"></div>
        </div>
      </td>
    </tr>
  );
}

export function InvoicesMobileSkeleton() {
  return (
    <div className="mb-2 w-full rounded-md bg-white p-4">
      <div className="flex items-center justify-between border-b border-gray-100
pb-8">
        <div className="flex items-center">
          <div className="mr-2 h-8 w-8 rounded-full bg-gray-100"></div>
          <div className="h-6 w-16 rounded bg-gray-100"></div>
        </div>
        <div className="h-6 w-16 rounded bg-gray-100"></div>
      </div>
      <div className="flex w-full items-center justify-between pt-4">
        <div>
          <div className="h-6 w-16 rounded bg-gray-100"></div>
          <div className="mt-2 h-6 w-24 rounded bg-gray-100"></div>
        </div>
        <div className="flex justify-end gap-2">
          <div className="h-10 w-10 rounded bg-gray-100"></div>
          <div className="h-10 w-10 rounded bg-gray-100"></div>
        </div>
      </div>
    </div>
```

```
  );
}

export function InvoicesTableSkeleton() {
  return (
    <div className="mt-6 flow-root">
      <div className="inline-block min-w-full align-middle">
        <div className="rounded-lg bg-gray-50 p-2 md:pt-0">
          <div className="md:hidden">
            <InvoicesMobileSkeleton />
            <InvoicesMobileSkeleton />
            <InvoicesMobileSkeleton />
            <InvoicesMobileSkeleton />
            <InvoicesMobileSkeleton />
            <InvoicesMobileSkeleton />
          </div>
          <table className="hidden min-w-full text-gray-900 md:table">
            <thead className="rounded-lg text-left text-sm font-normal">
              <tr>
                <th scope="col" className="px-4 py-5 font-medium sm:pl-6">
                  Customer
                </th>
                <th scope="col" className="px-3 py-5 font-medium">
                  Email
                </th>
                <th scope="col" className="px-3 py-5 font-medium">
                  Amount
                </th>
                <th scope="col" className="px-3 py-5 font-medium">
                  Date
                </th>
                <th scope="col" className="px-3 py-5 font-medium">
                  Status
                </th>
                <th
                  scope="col"
                  className="relative pb-4 pl-3 pr-6 pt-2 sm:pr-6"
                >
                  <span className="sr-only">Edit</span>
                </th>
              </tr>
            </thead>
            <tbody className="bg-white">
              <TableRowSkeleton />
              <TableRowSkeleton />
              <TableRowSkeleton />
              <TableRowSkeleton />
              <TableRowSkeleton />
              <TableRowSkeleton />
            </tbody>
          </table>
```

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

export function CustomersMobileSkeleton() {
  return (
    <div className="mb-2 w-full rounded-md bg-white p-4">
      <div className="flex items-center justify-between border-b border-gray-100
pb-8">
        <div className="flex items-center">
          <div className="mr-2 h-8 w-8 rounded-full bg-gray-100"></div>
          <div className="h-6 w-16 rounded bg-gray-100"></div>
        </div>
        <div className="h-6 w-16 rounded bg-gray-100"></div>
      </div>
      <div className="flex w-full items-center justify-between pt-4">
        <div>
          <div className="h-6 w-16 rounded bg-gray-100"></div>
          <div className="mt-2 h-6 w-24 rounded bg-gray-100"></div>
        </div>
        <div className="flex justify-end gap-2">
          <div className="h-10 w-10 rounded bg-gray-100"></div>
          <div className="h-10 w-10 rounded bg-gray-100"></div>
        </div>
      </div>
    </div>
  );
}


export function CustomersTableSkeleton() {
  return (
    <div className="mt-6 flow-root">
      <div className="inline-block min-w-full align-middle">
        <div className="rounded-lg bg-gray-50 p-2 md:pt-0">
          <div className="md:hidden">
            <CustomersMobileSkeleton />
            <CustomersMobileSkeleton />
            <CustomersMobileSkeleton />
            <CustomersMobileSkeleton />
            <CustomersMobileSkeleton />
            <CustomersMobileSkeleton />
          </div>
          <table className="hidden min-w-full text-gray-900 md:table">
            <thead className="rounded-lg text-left text-sm font-normal">
              <tr>
                <th scope="col" className="px-4 py-5 font-medium sm:pl-6">
                  Customer
                </th>
```

```
                <th scope="col" className="px-3 py-5 font-medium">
                  Email
                </th>

                <th
                  scope="col"
                  className="relative pb-4 pl-3 pr-6 pt-2 sm:pr-6"
                >
                  <span className="sr-only">Edit</span>
                </th>
              </tr>
            </thead>
            <tbody className="bg-white">
              <TableRowSkeleton />
              <TableRowSkeleton />
              <TableRowSkeleton />
              <TableRowSkeleton />
              <TableRowSkeleton />
              <TableRowSkeleton />
            </tbody>
          </table>
        </div>
      </div>
    </div>
  );
}
```

```
'use client';

import { MagnifyingGlassIcon } from '@heroicons/react/24/outline';
import { useSearchParams, usePathname, useRouter } from 'next/navigation';
import { useDebouncedCallback } from 'use-debounce';

export default function Search({ placeholder }: { placeholder: string }) {
  const searchParams = useSearchParams();
  const pathname = usePathname();
  const {replace} = useRouter();

  const handleSearch = useDebouncedCallback((term:string)=>{
    console.log(`Searching... ${term}`)
    const params = new URLSearchParams(searchParams)
    params.set('page','1');
    if(term){
      params.set('query',term);
    } else {
      params.delete('query')
    }
    replace(`${pathname}?${params.toString()}`);
  },300)
  return (
```

```
    <div className="relative flex flex-1 flex-shrink-0">
      <label htmlFor="search" className="sr-only">
        Search
      </label>
      <input
        className="peer block w-full rounded-md border border-gray-200 py-[9px] pl-
10 text-sm outline-2 placeholder:text-gray-500"
        placeholder={placeholder}
        onChange={(e)=>{
          handleSearch(e.target.value)
        }}
        defaultValue={searchParams.get('query')?.toString()}
      />
      <MagnifyingGlassIcon className="absolute left-3 top-1/2 h-[18px] w-[18px] -
translate-y-1/2 text-gray-500 peer-focus:text-gray-900" />
    </div>
  );
}
```

```
'use client';

import { lusitana } from '@/app/ui/fonts';
import {
  AtSymbolIcon,
  KeyIcon,
  ExclamationCircleIcon,
} from '@heroicons/react/24/outline';
import { ArrowRightIcon } from '@heroicons/react/20/solid';
import { Button } from './button';
import { useFormState, useFormStatus } from 'react-dom';
import { authenticate } from '@/app/lib/actions';

export default function LoginForm() {
  const [errorMessage, dispatch] = useFormState(authenticate, undefined)

  return (
    <form action={dispatch} className="space-y-3">
      <div className="flex-1 rounded-lg bg-gray-50 px-6 pb-4 pt-8">
        <h1 className={`${lusitana.className} mb-3 text-2xl`}>
          Please log in to continue.
        </h1>
        <div className="w-full">
          <div>
            <label
              className="mb-3 mt-5 block text-xs font-medium text-gray-900"
              htmlFor="email"
            >
              Email
            </label>
```

```
            <div className="relative">
              <input
                className="peer block w-full rounded-md border border-gray-200 py-
[9px] pl-10 text-sm outline-2 placeholder:text-gray-500"
                id="email"
                type="email"
                name="email"
                placeholder="Enter your email address"
                required
              />
              <AtSymbolIcon className="pointer-events-none absolute left-3 top-1/2
h-[18px] w-[18px] -translate-y-1/2 text-gray-500 peer-focus:text-gray-900" />
            </div>
          </div>
          <div className="mt-4">
            <label
              className="mb-3 mt-5 block text-xs font-medium text-gray-900"
              htmlFor="password"
            >
              Password
            </label>
            <div className="relative">
              <input
                className="peer block w-full rounded-md border border-gray-200 py-
[9px] pl-10 text-sm outline-2 placeholder:text-gray-500"
                id="password"
                type="password"
                name="password"
                placeholder="Enter password"
                required
                minLength={6}
              />
              <KeyIcon className="pointer-events-none absolute left-3 top-1/2 h-
[18px] w-[18px] -translate-y-1/2 text-gray-500 peer-focus:text-gray-900" />
            </div>
          </div>
        </div>
        <LoginButton />
        <div
          className="flex h-8 items-end space-x-1"
          aria-live='polite'
          aria-atomic='true'
        >
          {errorMessage && (
            <>
              <ExclamationCircleIcon className="h-5 w-5 text-red-500" />
              <p className="text-sm text-red-500">{errorMessage}</p>
            </>
          )}
        </div>
      </div>
```

```
    </form>
  );
}

function LoginButton() {
  const {pending} = useFormStatus();

  return (
    <Button className="mt-4 w-full" aria-disabled={pending}>
      Log in <ArrowRightIcon className="ml-auto h-5 w-5 text-gray-50" />
    </Button>
  );
}
```

```
@tailwind base;
@tailwind components;
@tailwind utilities;

input[type='number'] {
  -moz-appearance: textfield;
  appearance: textfield;
}

input[type='number']::-webkit-inner-spin-button {
  -webkit-appearance: none;
  margin: 0;
}

input[type='number']::-webkit-outer-spin-button {
  -webkit-appearance: none;
  margin: 0;
}
```

```
import {Inter,Lusitana} from 'next/font/google'

export const inter = Inter({subsets:['latin']});
export const lusitana = Lusitana({weight:['400','700'],subsets:['latin']});
```

```
import { UsersIcon } from '@heroicons/react/24/outline';
import { lusitana } from '@/app/ui/fonts';

export default function CRMLogo() {
  return (
    <div
      className={`${lusitana.className} flex flex-row items-center leading-none
text-white`}
    >
```

```
      <UsersIcon className="h-12 w-12*" />
      <p className="text-[44px]">CRM</p>
    </div>
  );
}
import clsx from 'clsx';

interface ButtonProps extends React.ButtonHTMLAttributes<HTMLButtonElement> {
  children: React.ReactNode;
}

export function Button({ children, className, ...rest }: ButtonProps) {
  return (
    <button
      {...rest}
      className={clsx(
        'flex h-10 items-center rounded-lg bg-blue-500 px-4 text-sm font-medium
text-white transition-colors hover:bg-blue-400 focus-visible:outline focus-
visible:outline-2 focus-visible:outline-offset-2 focus-visible:outline-blue-500
active:bg-blue-600 aria-disabled:cursor-not-allowed aria-disabled:opacity-50',
        className,
      )}
    >
      {children}
    </button>
  );
}
```

```
import Image from 'next/image';
import { UpdateInvoice, DeleteInvoice } from '@/app/ui/invoices/buttons';
import InvoiceStatus from '@/app/ui/invoices/status';
import { formatDateToLocal, formatCurrency } from '@/app/lib/utils';
import { fetchFilteredInvoices } from '@/app/lib/data';

export default async function InvoicesTable({
  query,
  currentPage,
}: {
  query: string;
  currentPage: number;
}) {
  const invoices = await fetchFilteredInvoices(query, currentPage);

  return (
    <div className="mt-6 flow-root">
      <div className="inline-block min-w-full align-middle">
        <div className="rounded-lg bg-gray-50 p-2 md:pt-0">
          <div className="md:hidden">
            {invoices?.map((invoice) => (
```

```
            <div
              key={invoice.id}
              className="mb-2 w-full rounded-md bg-white p-4"
            >
              <div className="flex items-center justify-between border-b pb-4">
                <div>
                  <div className="mb-2 flex items-center">
                    <Image
                      src={invoice.image_url}
                      className="mr-2 rounded-full"
                      width={28}
                      height={28}
                      alt={`${invoice.name}'s profile picture`}
                    />
                    <p>{invoice.name}</p>
                  </div>
                  <p className="text-sm text-gray-500">{invoice.email}</p>
                </div>
                <InvoiceStatus status={invoice.status} />
              </div>
              <div className="flex w-full items-center justify-between pt-4">
                <div>
                  <p className="text-xl font-medium">
                    {formatCurrency(invoice.amount)}
                  </p>
                  <p>{formatDateToLocal(invoice.date)}</p>
                </div>
                <div className="flex justify-end gap-2">
                  <UpdateInvoice id={invoice.id} />
                  <DeleteInvoice id={invoice.id} />
                </div>
              </div>
            </div>
          ))}
        </div>
        <table className="hidden min-w-full text-gray-900 md:table">
          <thead className="rounded-lg text-left text-sm font-normal">
            <tr>
              <th scope="col" className="px-4 py-5 font-medium sm:pl-6">
                Customer
              </th>
              <th scope="col" className="px-3 py-5 font-medium">
                Email
              </th>
              <th scope="col" className="px-3 py-5 font-medium">
                Amount
              </th>
              <th scope="col" className="px-3 py-5 font-medium">
                Date
              </th>
              <th scope="col" className="px-3 py-5 font-medium">
```

```
                Status
              </th>
              <th scope="col" className="relative py-3 pl-6 pr-3">
                <span className="sr-only">Edit</span>
              </th>
            </tr>
          </thead>
          <tbody className="bg-white">
            {invoices?.map((invoice) => (
              <tr
                key={invoice.id}
                className="w-full border-b py-3 text-sm last-of-type:border-none
[&:first-child>td:first-child]:rounded-tl-lg [&:first-child>td:last-child]:rounded-
tr-lg [&:last-child>td:first-child]:rounded-bl-lg [&:last-child>td:last-
child]:rounded-br-lg"
              >
                <td className="whitespace-nowrap py-3 pl-6 pr-3">
                  <div className="flex items-center gap-3">
                    <Image
                      src={invoice.image_url}
                      className="rounded-full"
                      width={28}
                      height={28}
                      alt={`${invoice.name}'s profile picture`}
                    />
                    <p>{invoice.name}</p>
                  </div>
                </td>
                <td className="whitespace-nowrap px-3 py-3">
                  {invoice.email}
                </td>
                <td className="whitespace-nowrap px-3 py-3">
                  {formatCurrency(invoice.amount)}
                </td>
                <td className="whitespace-nowrap px-3 py-3">
                  {formatDateToLocal(invoice.date)}
                </td>
                <td className="whitespace-nowrap px-3 py-3">
                  <InvoiceStatus status={invoice.status} />
                </td>
                <td className="whitespace-nowrap py-3 pl-6 pr-3">
                  <div className="flex justify-end gap-3">
                    <UpdateInvoice id={invoice.id} />
                    <DeleteInvoice id={invoice.id} />
                  </div>
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
```

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

import { CheckIcon, ClockIcon } from '@heroicons/react/24/outline';
import clsx from 'clsx';

export default function InvoiceStatus({ status }: { status: string }) {
  return (
    <span
      className={clsx(
        'inline-flex items-center rounded-full px-2 py-1 text-xs',
        {
          'bg-gray-100 text-gray-500': status === 'pending',
          'bg-green-500 text-white': status === 'paid',
        },
      )}
    >
      {status === 'pending' ? (
        <>
          Pending
          <ClockIcon className="ml-1 w-4 text-gray-500" />
        </>
      ) : null}
      {status === 'paid' ? (
        <>
          Paid
          <CheckIcon className="ml-1 w-4 text-white" />
        </>
      ) : null}
    </span>
  );
}
```

```
'use client';

import { ArrowLeftIcon, ArrowRightIcon } from '@heroicons/react/24/outline';
import clsx from 'clsx';
import Link from 'next/link';
import { generatePagination } from '@/app/lib/utils';
import { usePathname, useSearchParams } from 'next/navigation';

export default function Pagination({ totalPages }: { totalPages: number }) {
  const pathname = usePathname();
  const searchParams = useSearchParams();
  const currentPage = Number(searchParams.get('page')) || 1;

  const allPages = generatePagination(currentPage, totalPages);
```

```
  const createPageURL = (pageNumber: number|string)=>{
    const params = new URLSearchParams(searchParams);
    params.set('page', pageNumber.toString());
    return `${pathname}?${params.toString()}`;
  }

  return (
    <>
       <div className="inline-flex">
        <PaginationArrow
          direction="left"
          href={createPageURL(currentPage - 1)}
          isDisabled={currentPage <= 1}
        />

        <div className="flex -space-x-px">
          {allPages.map((page, index) => {
            let position: 'first' | 'last' | 'single' | 'middle' | undefined;

            if (index === 0) position = 'first';
            if (index === allPages.length - 1) position = 'last';
            if (allPages.length === 1) position = 'single';
            if (page === '...') position = 'middle';

            return (
              <PaginationNumber
                key={page}
                href={createPageURL(page)}
                page={page}
                position={position}
                isActive={currentPage === page}
              />
            );
          })}
        </div>

        <PaginationArrow
          direction="right"
          href={createPageURL(currentPage + 1)}
          isDisabled={currentPage >= totalPages}
        />
      </div>
    </>
  );
}

function PaginationNumber({
  page,
  href,
  isActive,
```

```
  position,
}: {
  page: number | string;
  href: string;
  position?: 'first' | 'last' | 'middle' | 'single';
  isActive: boolean;
}) {
  const className = clsx(
    'flex h-10 w-10 items-center justify-center text-sm border',
    {
      'rounded-l-md': position === 'first' || position === 'single',
      'rounded-r-md': position === 'last' || position === 'single',
      'z-10 bg-blue-600 border-blue-600 text-white': isActive,
      'hover:bg-gray-100': !isActive && position !== 'middle',
      'text-gray-300': position === 'middle',
    },
  );

  return isActive || position === 'middle' ? (
    <div className={className}>{page}</div>
  ) : (
    <Link href={href} className={className}>
      {page}
    </Link>
  );
}

function PaginationArrow({
  href,
  direction,
  isDisabled,
}: {
  href: string;
  direction: 'left' | 'right';
  isDisabled?: boolean;
}) {
  const className = clsx(
    'flex h-10 w-10 items-center justify-center rounded-md border',
    {
      'pointer-events-none text-gray-300': isDisabled,
      'hover:bg-gray-100': !isDisabled,
      'mr-2 md:mr-4': direction === 'left',
      'ml-2 md:ml-4': direction === 'right',
    },
  );

  const icon =
    direction === 'left' ? (
      <ArrowLeftIcon className="w-4" />
    ) : (
      <ArrowRightIcon className="w-4" />
```

```
    );

  return isDisabled ? (
    <div className={className}>{icon}</div>
  ) : (
    <Link className={className} href={href}>
      {icon}
    </Link>
  );
}
```

```
'use client';

import { CustomerField, InvoiceForm } from '@/app/lib/definitions';
import {
  CheckIcon,
  ClockIcon,
  CurrencyDollarIcon,
  UserCircleIcon,
} from '@heroicons/react/24/outline';
import Link from 'next/link';
import { Button } from '@/app/ui/button';
import { updateInvoice } from '@/app/lib/actions';
import { useFormState } from 'react-dom';

export default function EditInvoiceForm({
  invoice,
  customers,
}: {
  invoice: InvoiceForm;
  customers: CustomerField[];
}) {
  const initialState = { message: null, errors: {} };
  const updateInvoiceWithId = updateInvoice.bind(null, invoice.id);
  const [state, dispatch] = useFormState(updateInvoiceWithId, initialState);

  return (
    <form action={dispatch}>
      <div className="rounded-md bg-gray-50 p-4 md:p-6">
        {/* Customer Name */}
        <div className="mb-4">
          <label htmlFor="customer" className="mb-2 block text-sm font-medium">
            Choose customer
          </label>
          <div className="relative">
            <select
              id="customer"
              name="customerId"
```

```
              className="peer block w-full cursor-pointer rounded-md border border-
gray-200 py-2 pl-10 text-sm outline-2 placeholder:text-gray-500"
              defaultValue={invoice.customer_id}
            >
              <option value="" disabled>
                Select a customer
              </option>
              {customers.map((customer) => (
                <option key={customer.id} value={customer.id}>
                  {customer.name}
                </option>
              ))}
            </select>
            <UserCircleIcon className="pointer-events-none absolute left-3 top-1/2
h-[18px] w-[18px] -translate-y-1/2 text-gray-500" />
          </div>
        </div>

        {/* Invoice Amount */}
        <div className="mb-4">
          <label htmlFor="amount" className="mb-2 block text-sm font-medium">
            Choose an amount
          </label>
          <div className="relative mt-2 rounded-md">
            <div className="relative">
              <input
                id="amount"
                name="amount"
                type="number"
                step="0.01"
                defaultValue={invoice.amount}
                placeholder="Enter USD amount"
                className="peer block w-full rounded-md border border-gray-200 py-2
pl-10 text-sm outline-2 placeholder:text-gray-500"
              />
              <CurrencyDollarIcon className="pointer-events-none absolute left-3
top-1/2 h-[18px] w-[18px] -translate-y-1/2 text-gray-500 peer-focus:text-gray-900"
/>
            </div>
          </div>
        </div>

        {/* Invoice Status */}
        <fieldset>
          <legend className="mb-2 block text-sm font-medium">
            Set the invoice status
          </legend>
          <div className="rounded-md border border-gray-200 bg-white px-[14px] py-
3">
            <div className="flex gap-4">
              <div className="flex items-center">
```

```
              <input
                id="pending"
                name="status"
                type="radio"
                value="pending"
                defaultChecked={invoice.status === 'pending'}
                className="h-4 w-4 cursor-pointer border-gray-300 bg-gray-100
text-gray-600 focus:ring-2"
              />
              <label
                htmlFor="pending"
                className="ml-2 flex cursor-pointer items-center gap-1.5 rounded-
full bg-gray-100 px-3 py-1.5 text-xs font-medium text-gray-600"
              >
                Pending <ClockIcon className="h-4 w-4" />
              </label>
            </div>
            <div className="flex items-center">
              <input
                id="paid"
                name="status"
                type="radio"
                value="paid"
                defaultChecked={invoice.status === 'paid'}
                className="h-4 w-4 cursor-pointer border-gray-300 bg-gray-100
text-gray-600 focus:ring-2"
              />
              <label
                htmlFor="paid"
                className="ml-2 flex cursor-pointer items-center gap-1.5 rounded-
full bg-green-500 px-3 py-1.5 text-xs font-medium text-white"
              >
                Paid <CheckIcon className="h-4 w-4" />
              </label>
            </div>
          </div>
        </div>
      </fieldset>
    </div>
    <div className="mt-6 flex justify-end gap-4">
      <Link
        href="/dashboard/invoices"
        className="flex h-10 items-center rounded-lg bg-gray-100 px-4 text-sm
font-medium text-gray-600 transition-colors hover:bg-gray-200"
      >
        Cancel
      </Link>
      <Button type="submit">Edit Invoice</Button>
    </div>
  </form>
);
```

```
}
```

```
'use client';
import { CustomerField } from '@/app/lib/definitions';
import Link from 'next/link';
import {
  CheckIcon,
  ClockIcon,
  CurrencyDollarIcon,
  UserCircleIcon,
} from '@heroicons/react/24/outline';
import { Button } from '@/app/ui/button';
import {createInvoice } from '@/app/lib/actions';
import { useFormState } from 'react-dom';

export default function Form({ customers }: { customers: CustomerField[] }) {
  const initialState = {message:null, errors:{}};
  const [state, dispatch] = useFormState(createInvoice, initialState)

  return (
    <form action={dispatch}>
      <div className="rounded-md bg-gray-50 p-4 md:p-6">
        {/* Customer Name */}
        <div className="mb-4">
          <label htmlFor="customer" className="mb-2 block text-sm font-medium">
            Choose customer
          </label>
          <div className="relative">
            <select
              id="customer"
              name="customerId"
              className="peer block w-full cursor-pointer rounded-md border border-
gray-200 py-2 pl-10 text-sm outline-2 placeholder:text-gray-500"
              defaultValue=""
              aria-describedby='customer-error'
            >
              <option value="" disabled>
                Select a customer
              </option>
              {customers.map((customer) => (
                <option key={customer.id} value={customer.id}>
                  {customer.name}
                </option>
              ))}
            </select>
            <UserCircleIcon className="pointer-events-none absolute left-3 top-1/2
h-[18px] w-[18px] -translate-y-1/2 text-gray-500" />
          </div>
          <div id="customer-error" aria-live="polite" aria-atomic="true">
```

```
            {state.errors?.customerId &&
              state.errors.customerId.map((error: string) => (
                <p className="mt-2 text-sm text-red-500" key={error}>
                  {error}
                </p>
              ))}
          </div>
        </div>

        {/* Invoice Amount */}
        <div className="mb-4">
          <label htmlFor="amount" className="mb-2 block text-sm font-medium">
            Choose an amount
          </label>
          <div className="relative mt-2 rounded-md">
            <div className="relative">
              <input
                id="amount"
                name="amount"
                type="number"
                step="0.01"
                placeholder="Enter USD amount"
                className="peer block w-full rounded-md border border-gray-200 py-2
pl-10 text-sm outline-2 placeholder:text-gray-500"
                aria-describedby='amount-error'
              />
              <CurrencyDollarIcon className="pointer-events-none absolute left-3
top-1/2 h-[18px] w-[18px] -translate-y-1/2 text-gray-500 peer-focus:text-gray-900"
/>
            </div>
          </div>
          <div id="amount-error" aria-live="polite" aria-atomic="true">
            {state.errors?.amount &&
              state.errors.amount.map((error: string) => (
                <p className="mt-2 text-sm text-red-500" key={error}>
                  {error}
                </p>
              ))}
          </div>
        </div>

        {/* Invoice Status */}
        <fieldset>
          <legend className="mb-2 block text-sm font-medium">
            Set the invoice status
          </legend>
          <div className="rounded-md border border-gray-200 bg-white px-[14px] py-
3">
            <div className="flex gap-4">
              <div className="flex items-center">
                <input
```

```
                id="pending"
                name="status"
                type="radio"
                value="pending"
                className="h-4 w-4 cursor-pointer border-gray-300 bg-gray-100
text-gray-600 focus:ring-2"
                aria-describedby='status-error'
              />
              <label
                htmlFor="pending"
                className="ml-2 flex cursor-pointer items-center gap-1.5 rounded-
full bg-gray-100 px-3 py-1.5 text-xs font-medium text-gray-600"
              >
                Pending <ClockIcon className="h-4 w-4" />
              </label>
            </div>
            <div className="flex items-center">
              <input
                id="paid"
                name="status"
                type="radio"
                value="paid"
                className="h-4 w-4 cursor-pointer border-gray-300 bg-gray-100
text-gray-600 focus:ring-2"
                aria-describedby='status-error'
              />
              <label
                htmlFor="paid"
                className="ml-2 flex cursor-pointer items-center gap-1.5 rounded-
full bg-green-500 px-3 py-1.5 text-xs font-medium text-white"
              >
                Paid <CheckIcon className="h-4 w-4" />
              </label>
            </div>
          </div>
          <div id="status-error" aria-live="polite" aria-atomic="true">
          {state.errors?.status &&
            state.errors.status.map((error: string) => (
              <p className="mt-2 text-sm text-red-500" key={error}>
                {error}
              </p>
            ))}
          </div>
        </div>
      </fieldset>
    </div>
    <div className="mt-6 flex justify-end gap-4">
      <Link
        href="/dashboard/invoices"
        className="flex h-10 items-center rounded-lg bg-gray-100 px-4 text-sm
font-medium text-gray-600 transition-colors hover:bg-gray-200"
```

```
        >
          Cancel
        </Link>
        <Button type="submit">Create Invoice</Button>
      </div>
    </form>
  );
}
```

```
import { PencilIcon, PlusIcon, TrashIcon } from '@heroicons/react/24/outline';
import Link from 'next/link';
import {deleteInvoice} from '@/app/lib/actions';

export function CreateInvoice() {
  return (
    <Link
      href="/dashboard/invoices/create"
      className="flex h-10 items-center rounded-lg bg-blue-600 px-4 text-sm font-
medium text-white transition-colors hover:bg-blue-500 focus-visible:outline focus-
visible:outline-2 focus-visible:outline-offset-2 focus-visible:outline-blue-600"
    >
      <span className="hidden md:block">Create Invoice</span>{' '}
      <PlusIcon className="h-5 md:ml-4" />
    </Link>
  );
}

export function UpdateInvoice({ id }: { id: string }) {
  return (
    <Link
      href={`/dashboard/invoices/${id}/edit`}
      className="rounded-md border p-2 hover:bg-gray-100"
    >
      <PencilIcon className="w-5" />
    </Link>
  );
}

export function DeleteInvoice({ id }: { id: string }) {
  const deleteInvoiceWithId = deleteInvoice.bind(null,id);
  return (
    <form action={deleteInvoiceWithId}>
      <button className="rounded-md border p-2 hover:bg-gray-100">
        <span className="sr-only">Delete</span>
        <TrashIcon className="w-5" />
      </button>
    </form>
  );
}
```

```tsx
import { clsx } from 'clsx';
import Link from 'next/link';
import { lusitana } from '@/app/ui/fonts';

interface Breadcrumb {
  label: string;
  href: string;
  active?: boolean;
}

export default function Breadcrumbs({
  breadcrumbs,
}: {
  breadcrumbs: Breadcrumb[];
}) {
  return (
    <nav aria-label="Breadcrumb" className="mb-6 block">
      <ol className={clsx(lusitana.className, 'flex text-xl md:text-2xl')}>
        {breadcrumbs.map((breadcrumb, index) => (
          <li
            key={breadcrumb.href}
            aria-current={breadcrumb.active}
            className={clsx(
              breadcrumb.active ? 'text-gray-900' : 'text-gray-500',
            )}
          >
            <Link href={breadcrumb.href}>{breadcrumb.label}</Link>
            {index < breadcrumbs.length - 1 ? (
              <span className="mx-3 inline-block">/</span>
            ) : null}
          </li>
        ))}
      </ol>
    </nav>
  );
}
```

```tsx
import Link from 'next/link';
import NavLinks from '@/app/ui/dashboard/nav-links';
import CRMLogo from '@/app/ui/crm-logo';
import { PowerIcon } from '@heroicons/react/24/outline';
import { signOut } from '@/auth';

export default function SideNav() {
  return (
    <div className="flex h-full flex-col px-3 py-4 md:px-2">
      <Link
```

```
        className="mb-2 flex h-20 items-end justify-start rounded-md bg-blue-600 p-
4 md:h-40"
        href="/"
      >
        <div className="w-32 text-white md:w-40">
          <CRMLogo />
        </div>
      </Link>
      <div className="flex grow flex-row justify-between space-x-2 md:flex-col
md:space-x-0 md:space-y-2">
        <NavLinks />
        <div className="hidden h-auto w-full grow rounded-md bg-gray-50
md:block"></div>
        <form
          action={async ()=>{
            'use server';
            await signOut();
          }}
        >
          <button className="flex h-[48px] w-full grow items-center justify-center
gap-2 rounded-md bg-gray-50 p-3 text-sm font-medium hover:bg-sky-100 hover:text-
blue-600 md:flex-none md:justify-start md:p-2 md:px-3">
            <PowerIcon className="w-6" />
            <div className="hidden md:block">Sign Out</div>
          </button>
        </form>
      </div>
    </div>
  );
}
```

```
import { generateYAxis } from '@/app/lib/utils';
import { CalendarIcon } from '@heroicons/react/24/outline';
import { lusitana } from '@/app/ui/fonts';
import { fetchRevenue } from '@/app/lib/data';


export default async function RevenueChart() {
  const revenue = await fetchRevenue();

  const chartHeight = 350;

  const { yAxisLabels, topLabel } = generateYAxis(revenue);

  if (!revenue || revenue.length === 0) {
    return <p className="mt-4 text-gray-400">No data available.</p>;
  }

  return (
```

```
    <div className="w-full md:col-span-4">
      <h2 className={`${lusitana.className} mb-4 text-xl md:text-2xl`}>
        Recent Revenue
      </h2>

      { <div className="rounded-xl bg-gray-50 p-4">
        <div className="sm:grid-cols-13 mt-0 grid grid-cols-12 items-end gap-2
rounded-md bg-white p-4 md:gap-4">
          <div
            className="mb-6 hidden flex-col justify-between text-sm text-gray-400
sm:flex"
            style={{ height: `${chartHeight}px` }}
          >
            {yAxisLabels.map((label) => (
              <p key={label}>{label}</p>
            ))}
          </div>

          {revenue.map((month) => (
            <div key={month.month} className="flex flex-col items-center gap-2">
              <div
                className="w-full rounded-md bg-blue-300"
                style={{
                  height: `${(chartHeight / topLabel) * month.revenue}px`,
                }}
              ></div>
              <p className="-rotate-90 text-sm text-gray-400 sm:rotate-0">
                {month.month}
              </p>
            </div>
          ))}
        </div>
        <div className="flex items-center pb-2 pt-6">
          <CalendarIcon className="h-5 w-5 text-gray-500" />
          <h3 className="ml-2 text-sm text-gray-500 ">Last 12 months</h3>
        </div>
      </div> }
    </div>
  );
}
```

```
'use client';

import {
  UserGroupIcon,
  HomeIcon,
  DocumentDuplicateIcon,
} from '@heroicons/react/24/outline';
import clsx from 'clsx';
```

```
import Link from 'next/link';
import { usePathname } from 'next/navigation';

const links = [
  { name: 'Home', href: '/dashboard', icon: HomeIcon },
  {
    name: 'Invoices',
    href: '/dashboard/invoices',
    icon: DocumentDuplicateIcon,
  },
  { name: 'Customers', href: '/dashboard/customers', icon: UserGroupIcon },
];

export default function NavLinks() {
  const pathname = usePathname();
  return (
    <>
      {links.map((link) => {
        const LinkIcon = link.icon;
        return (
          <Link
            key={link.name}
            href={link.href}
            className={clsx('flex h-[48px] grow items-center justify-center gap-2
rounded-md bg-gray-50 p-3 text-sm font-medium hover:bg-sky-100 hover:text-blue-600
md:flex-none md:justify-start md:p-2 md:px-3',
            {'bg-sky-100 text-blue-600': pathname === link.href})}
          >
            <LinkIcon className="w-6" />
            <p className="hidden md:block">{link.name}</p>
          </Link>
        );
      })}
    </>
  );
}
```

```
import { ArrowPathIcon } from '@heroicons/react/24/outline';
import clsx from 'clsx';
import Image from 'next/image';
import { lusitana } from '@/app/ui/fonts';
import { LatestInvoice } from '@/app/lib/definitions';
import { fetchLatestInvoices } from '@/app/lib/data';
export default async function LatestInvoices() {
  const latestInvoices = await fetchLatestInvoices();
  return (
    <div className="flex w-full flex-col md:col-span-4">
      <h2 className={`${lusitana.className} mb-4 text-xl md:text-2xl`}>
        Latest Invoices
```

```
      </h2>
      <div className="flex grow flex-col justify-between rounded-xl bg-gray-50 p-
4">
        {/* NOTE: comment in this code when you get to this point in the course */}

        { <div className="bg-white px-6">
          {latestInvoices.map((invoice, i) => {
            return (
              <div
                key={invoice.id}
                className={clsx(
                  'flex flex-row items-center justify-between py-4',
                  {
                    'border-t': i !== 0,
                  },
                )}
              >
                <div className="flex items-center">
                  <Image
                    src={invoice.image_url}
                    alt={`${invoice.name}'s profile picture`}
                    className="mr-4 rounded-full"
                    width={32}
                    height={32}
                  />
                  <div className="min-w-0">
                    <p className="truncate text-sm font-semibold md:text-base">
                      {invoice.name}
                    </p>
                    <p className="hidden text-sm text-gray-500 sm:block">
                      {invoice.email}
                    </p>
                  </div>
                </div>
                <p
                  className={`${lusitana.className} truncate text-sm font-medium
md:text-base`}
                >
                  {invoice.amount}
                </p>
              </div>
            );
          })}
        </div> }
        <div className="flex items-center pb-2 pt-6">
          <ArrowPathIcon className="h-5 w-5 text-gray-500" />
          <h3 className="ml-2 text-sm text-gray-500 ">Updated just now</h3>
        </div>
      </div>
    </div>
  );
```

```
}
```

```
import {
  BanknotesIcon,
  ClockIcon,
  UserGroupIcon,
  InboxIcon,
} from '@heroicons/react/24/outline';
import { lusitana } from '@/app/ui/fonts';
import { fetchCardData } from '@/app/lib/data';

const iconMap = {
  collected: BanknotesIcon,
  customers: UserGroupIcon,
  pending: ClockIcon,
  invoices: InboxIcon,
};

export default async function CardWrapper() {
  const {
    numberOfInvoices,
    numberOfCustomers,
    totalPaidInvoices,
    totalPendingInvoices
  } = await fetchCardData();
  return (
    <>

      <Card title="Collected" value={totalPaidInvoices} type="collected" />
      <Card title="Pending" value={totalPendingInvoices} type="pending" />
      <Card title="Total Invoices" value={numberOfInvoices} type="invoices" />
      <Card
        title="Total Customers"
        value={numberOfCustomers}
        type="customers"
      />
    </>
  );
}

export function Card({
  title,
  value,
  type,
}: {
  title: string;
  value: number | string;
  type: 'invoices' | 'customers' | 'pending' | 'collected';
}) {
```

```
  const Icon = iconMap[type];

  return (
    <div className="rounded-xl bg-gray-50 p-2 shadow-sm">
      <div className="flex p-4">
        {Icon ? <Icon className="h-5 w-5 text-gray-700" /> : null}
        <h3 className="ml-2 text-sm font-medium">{title}</h3>
      </div>
      <p
        className={`${lusitana.className}
          truncate rounded-xl bg-white px-4 py-8 text-center text-2xl`}
      >
        {value}
      </p>
    </div>
  );
}
```

```
import Image from 'next/image';
import { lusitana } from '@/app/ui/fonts';
import { fetchFilteredCustomers } from '@/app/lib/data';

export default async function CustomersTable({
  query,
  currentPage,
}: {
  query: string,
  currentPage: number;
}) {
  const customers = await fetchFilteredCustomers(query, currentPage)
  return (
    <div className="w-full">
      <div className="mt-6 flow-root">
        <div className="overflow-x-auto">
          <div className="inline-block min-w-full align-middle">
            <div className="overflow-hidden rounded-md bg-gray-50 p-2 md:pt-0">
              <div className="md:hidden">
                {customers?.map((customer) => (
                  <div
                    key={customer.id}
                    className="mb-2 w-full rounded-md bg-white p-4"
                  >
                    <div className="flex items-center justify-between border-b pb-
4">
                      <div>
                        <div className="mb-2 flex items-center">
                          <div className="flex items-center gap-3">
                            <Image
                              src={customer.image_url}
```

```
                            className="rounded-full"
                            alt={`${customer.name}'s profile picture`}
                            width={28}
                            height={28}
                          />
                          <p>{customer.name}</p>
                        </div>
                      </div>
                      <p className="text-sm text-gray-500">
                        {customer.email}
                      </p>
                    </div>
                  </div>
                  <div className="flex w-full items-center justify-between
border-b py-5">
                    <div className="flex w-1/2 flex-col">
                      <p className="text-xs">Pending</p>
                      <p className="font-medium">{customer.total_pending}</p>
                    </div>
                    <div className="flex w-1/2 flex-col">
                      <p className="text-xs">Paid</p>
                      <p className="font-medium">{customer.total_paid}</p>
                    </div>
                  </div>
                  <div className="pt-4 text-sm">
                    <p>{customer.total_invoices} invoices</p>
                  </div>
                </div>
              ))}
            </div>
            <table className="hidden min-w-full rounded-md text-gray-900
md:table">
              <thead className="rounded-md bg-gray-50 text-left text-sm font-
normal">
                <tr>
                  <th scope="col" className="px-4 py-5 font-medium sm:pl-6">
                    Name
                  </th>
                  <th scope="col" className="px-3 py-5 font-medium">
                    Email
                  </th>
                  <th scope="col" className="px-3 py-5 font-medium">
                    Total Invoices
                  </th>
                  <th scope="col" className="px-3 py-5 font-medium">
                    Total Pending
                  </th>
                  <th scope="col" className="px-4 py-5 font-medium">
                    Total Paid
                  </th>
                </tr>
```

```
              </thead>

              <tbody className="divide-y divide-gray-200 text-gray-900">
                {customers.map((customer) => (
                  <tr key={customer.id} className="group">
                    <td className="whitespace-nowrap bg-white py-5 pl-4 pr-3
text-sm text-black group-first-of-type:rounded-md group-last-of-type:rounded-md
sm:pl-6">
                      <div className="flex items-center gap-3">
                        <Image
                          src={customer.image_url}
                          className="rounded-full"
                          alt={`${customer.name}'s profile picture`}
                          width={28}
                          height={28}
                        />
                        <p>{customer.name}</p>
                      </div>
                    </td>
                    <td className="whitespace-nowrap bg-white px-4 py-5 text-sm">
                      {customer.email}
                    </td>
                    <td className="whitespace-nowrap bg-white px-4 py-5 text-sm">
                      {customer.total_invoices}
                    </td>
                    <td className="whitespace-nowrap bg-white px-4 py-5 text-sm">
                      {customer.total_pending}
                    </td>
                    <td className="whitespace-nowrap bg-white px-4 py-5 text-sm
group-first-of-type:rounded-md group-last-of-type:rounded-md">
                      {customer.total_paid}
                    </td>
                  </tr>
                ))}
              </tbody>
            </table>
          </div>
        </div>
      </div>
    </div>
  );
}
```

```
'use client';
import { CustomerField } from '@/app/lib/definitions';
import Link from 'next/link';
import {
  AtSymbolIcon,
```

```
  CameraIcon,
  UserCircleIcon,
} from '@heroicons/react/24/outline';
import { Button } from '@/app/ui/button';
import { createCustomer } from '@/app/lib/actions';
import { useFormState } from 'react-dom';

export default function Form() {
  const initialState = {message:null, errors:{}};
  const [state, dispatch] = useFormState(createCustomer, initialState)

  return (
    <form action={dispatch}>
      <div className="rounded-md bg-gray-50 p-4 md:p-6">
        {/* Customer fullname */}
        <div className="mb-4">
          <label htmlFor="customer" className="mb-2 block text-sm font-medium">
            Enter fullname
          </label>
          <div className="relative">
            <input
              id="name"
              name="name"
              type='string'
              placeholder='fullname'
              className="peer block w-full cursor-pointer rounded-md border border-
gray-200 py-2 pl-10 text-sm outline-2 placeholder:text-gray-500"
              defaultValue=""
              aria-describedby='customer-error'
            />
            <UserCircleIcon className="pointer-events-none absolute left-3 top-1/2
h-[18px] w-[18px] -translate-y-1/2 text-gray-500" />
          </div>
          <div id="customer-error" aria-live="polite" aria-atomic="true">
            {state.errors?.name &&
              state.errors.name.map((error: string) => (
                <p className="mt-2 text-sm text-red-500" key={error}>
                  {error}
                </p>
              ))}
          </div>
        </div>

        {/* Customer email */}
        <div className="mb-4">
          <label htmlFor="email" className="mb-2 block text-sm font-medium">
            Enter email
          </label>
          <div className="relative mt-2 rounded-md">
            <div className="relative">
              <input
```

```
                id="email"
                name="email"
                type="email"
                placeholder="email"
                className="peer block w-full rounded-md border border-gray-200 py-2
pl-10 text-sm outline-2 placeholder:text-gray-500"
                aria-describedby='email-error'
              />
              <AtSymbolIcon className="pointer-events-none absolute left-3 top-1/2
h-[18px] w-[18px] -translate-y-1/2 text-gray-500 peer-focus:text-gray-900" />
            </div>
          </div>
          <div id="email-error" aria-live="polite" aria-atomic="true">
            {state.errors?.email &&
              state.errors.email.map((error: string) => (
                <p className="mt-2 text-sm text-red-500" key={error}>
                  {error}
                </p>
              ))}
          </div>
        </div>

        {/* Customer photo
        <div className="mb-4">
          <label htmlFor="image_url" className="mb-2 block text-sm font-medium">
            Upload photo
          </label>
          <div className="relative mt-2 rounded-md">
            <div className="relative">
              <input
                id="image_url"
                name="image_url"
                type="file"
                placeholder="image_url"
                accept="image/*"
                className="peer block w-full rounded-md border border-gray-200 py-2
pl-10 text-sm outline-2 placeholder:text-gray-500"
                aria-describedby='image_url-error'
              />
              <CameraIcon className="pointer-events-none absolute left-3 top-1/2 h-
[18px] w-[18px] -translate-y-1/2 text-gray-500 peer-focus:text-gray-900" />
            </div>
          </div>
          <div id="image_url-error" aria-live="polite" aria-atomic="true">
            {state.errors?.image_url &&
              state.errors.image_url.map((error: string) => (
                <p className="mt-2 text-sm text-red-500" key={error}>
                  {error}
                </p>
              ))}
          </div>
```

```
        </div>*/}

      </div>
      <div className="mt-6 flex justify-end gap-4">
        <Link
          href="/dashboard/invoices"
          className="flex h-10 items-center rounded-lg bg-gray-100 px-4 text-sm
font-medium text-gray-600 transition-colors hover:bg-gray-200"
        >
          Cancel
        </Link>
        <Button type="submit">Create Customer</Button>
      </div>
    </form>
  );
}
```

```
import { PencilIcon, PlusIcon, TrashIcon } from '@heroicons/react/24/outline';
import Link from 'next/link';
import {deleteCustomer} from '@/app/lib/actions';

export function CreateCustomer() {
  return (
    <Link
      href="/dashboard/customers/create"
      className="flex h-10 items-center rounded-lg bg-blue-600 px-4 text-sm font-
medium text-white transition-colors hover:bg-blue-500 focus-visible:outline focus-
visible:outline-2 focus-visible:outline-offset-2 focus-visible:outline-blue-600"
    >
      <span className="hidden md:block">Create Customer</span>{' '}
      <PlusIcon className="h-5 md:ml-4" />
    </Link>
  );
}

export function UpdateCustomer({ id }: { id: string }) {
  return (
    <Link
      href={`/dashboard/customers/${id}/edit`}
      className="rounded-md border p-2 hover:bg-gray-100"
    >
      <PencilIcon className="w-5" />
    </Link>
  );
}

export function DeleteCustomer({ id }: { id: string }) {
  const deleteCustomerithId = deleteCustomer.bind(null,id);
  return (
    <form action={deleteCustomerithId}>
```

```
      <button className="rounded-md border p-2 hover:bg-gray-100">
        <span className="sr-only">Delete</span>
        <TrashIcon className="w-5" />
      </button>
    </form>
  );
}
```

```
import CRMLogo from '@/app/ui/crm-logo';
import LoginForm from '@/app/ui/login-form';
import {Metadata} from 'next';

export const metadata: Metadata ={
  title: 'Login | CRM Dashboard'
}

export default function LoginPage() {
  return (
    <main className="flex items-center justify-center md:h-screen">
      <div className="relative mx-auto flex w-full max-w-[400px] flex-col space-y-
2.5 p-4 md:-mt-32">
        <div className="flex h-20 w-full items-end rounded-lg bg-blue-500 p-3 md:h-
36">
          <div className="w-32 text-white md:w-36">
            <CRMLogo />
          </div>
        </div>
        <LoginForm />
      </div>
    </main>
  );
}
```

```
import { Revenue } from './definitions';

export const formatCurrency = (amount: number) => {
  return (amount / 100).toLocaleString('en-US', {
    style: 'currency',
    currency: 'USD',
  });
};

export const formatDateToLocal = (
  dateStr: string,
  locale: string = 'en-US',
) => {
  const date = new Date(dateStr);
  const options: Intl.DateTimeFormatOptions = {
    day: 'numeric',
```

```
    month: 'short',
    year: 'numeric',
  };
  const formatter = new Intl.DateTimeFormat(locale, options);
  return formatter.format(date);
};

export const generateYAxis = (revenue: Revenue[]) => {
  const yAxisLabels = [];
  const highestRecord = Math.max(...revenue.map((month) => month.revenue));
  const topLabel = Math.ceil(highestRecord / 1000) * 1000;

  for (let i = topLabel; i >= 0; i -= 1000) {
    yAxisLabels.push(`$${i / 1000}K`);
  }

  return { yAxisLabels, topLabel };
};

export const generatePagination = (currentPage: number, totalPages: number) => {

  if (totalPages <= 7) {
    return Array.from({ length: totalPages }, (_, i) => i + 1);
  }

  if (currentPage <= 3) {
    return [1, 2, 3, '...', totalPages - 1, totalPages];
  }
  if (currentPage >= totalPages - 2) {
    return [1, 2, '...', totalPages - 2, totalPages - 1, totalPages];
  }

  return [
    1,
    '...',
    currentPage - 1,
    currentPage,
    currentPage + 1,
    '...',
    totalPages,
  ];
};
```

```
export type User = {
  id: string;
  name: string;
  email: string;
  password: string;
};
```

```ts
export type Customer = {
  id: string;
  name: string;
  email: string;
  image_url: string;
};

export type Invoice = {
  id: string;
  customer_id: string;
  amount: number;
  date: string;
  status: 'pending' | 'paid';
};

export type Revenue = {
  month: string;
  revenue: number;
};

export type LatestInvoice = {
  id: string;
  name: string;
  image_url: string;
  email: string;
  amount: string;
};

export type LatestInvoiceRaw = Omit<LatestInvoice, 'amount'> & {
  amount: number;
};

export type InvoicesTable = {
  id: string;
  customer_id: string;
  name: string;
  email: string;
  image_url: string;
  date: string;
  amount: number;
  status: 'pending' | 'paid';
};

export type CustomersTableType = {
  id: string;
  name: string;
  email: string;
  image_url: string;
  total_invoices: number;
  total_pending: number;
```

```
  total_paid: number;
};

export type FormattedCustomersTable = {
  id: string;
  name: string;
  email: string;
  image_url: string;
  total_invoices: number;
  total_pending: string;
  total_paid: string;
};

export type CustomerField = {
  id: string;
  name: string;
};

export type InvoiceForm = {
  id: string;
  customer_id: string;
  amount: number;
  status: 'pending' | 'paid';
};
```

```
import { sql } from '@vercel/postgres';
import {
  CustomerField,
  CustomersTableType,
  InvoiceForm,
  InvoicesTable,
  LatestInvoiceRaw,
  User,
  Revenue,
} from './definitions';
import { formatCurrency } from './utils';
import { unstable_noStore as noStore} from 'next/cache';

export async function fetchRevenue() {
  noStore();
  try {

    const data = await sql<Revenue>`SELECT * FROM revenue`;


    return data.rows;
  } catch (error) {
    console.error('Database Error:', error);
    throw new Error('Failed to fetch revenue data.');
```

```
  }
}

export async function fetchLatestInvoices() {
  noStore();
  try {
    const data = await sql<LatestInvoiceRaw>`
      SELECT invoices.amount, customers.name, customers.image_url, customers.email,
invoices.id
      FROM invoices
      JOIN customers ON invoices.customer_id = customers.id
      ORDER BY invoices.date DESC
      LIMIT 5`;

    const latestInvoices = data.rows.map((invoice) => ({
      ...invoice,
      amount: formatCurrency(invoice.amount),
    }));
    return latestInvoices;
  } catch (error) {
    console.error('Database Error:', error);
    throw new Error('Failed to fetch the latest invoices.');
  }
}

export async function fetchCardData() {
  noStore();
  try {
    const invoiceCountPromise = sql`SELECT COUNT(*) FROM invoices`;
    const customerCountPromise = sql`SELECT COUNT(*) FROM customers`;
    const invoiceStatusPromise = sql`SELECT
         SUM(CASE WHEN status = 'paid' THEN amount ELSE 0 END) AS "paid",
         SUM(CASE WHEN status = 'pending' THEN amount ELSE 0 END) AS "pending"
         FROM invoices`;

    const data = await Promise.all([
      invoiceCountPromise,
      customerCountPromise,
      invoiceStatusPromise,
    ]);

    const numberOfInvoices = Number(data[0].rows[0].count ?? '0');
    const numberOfCustomers = Number(data[1].rows[0].count ?? '0');
    const totalPaidInvoices = formatCurrency(data[2].rows[0].paid ?? '0');
    const totalPendingInvoices = formatCurrency(data[2].rows[0].pending ?? '0');

    return {
      numberOfCustomers,
      numberOfInvoices,
      totalPaidInvoices,
      totalPendingInvoices,
```

```
    };
  } catch (error) {
    console.error('Database Error:', error);
    throw new Error('Failed to fetch card data.');
  }
}

const ITEMS_PER_PAGE = 6;
export async function fetchFilteredInvoices(
  query: string,
  currentPage: number,
) {
  noStore();
  const offset = (currentPage - 1) * ITEMS_PER_PAGE;

  try {
    const invoices = await sql<InvoicesTable>`
      SELECT
        invoices.id,
        invoices.amount,
        invoices.date,
        invoices.status,
        customers.name,
        customers.email,
        customers.image_url
      FROM invoices
      JOIN customers ON invoices.customer_id = customers.id
      WHERE
        customers.name ILIKE ${`%${query}%`} OR
        customers.email ILIKE ${`%${query}%`} OR
        invoices.amount::text ILIKE ${`%${query}%`} OR
        invoices.date::text ILIKE ${`%${query}%`} OR
        invoices.status ILIKE ${`%${query}%`}
      ORDER BY invoices.date DESC
      LIMIT ${ITEMS_PER_PAGE} OFFSET ${offset}
    `;

    return invoices.rows;
  } catch (error) {
    console.error('Database Error:', error);
    throw new Error('Failed to fetch invoices.');
  }
}

export async function fetchInvoicesPages(query: string) {
  noStore();
  try {
    const count = await sql`SELECT COUNT(*)
    FROM invoices
    JOIN customers ON invoices.customer_id = customers.id
    WHERE
```

```
      customers.name ILIKE ${`%${query}%`} OR
      customers.email ILIKE ${`%${query}%`} OR
      invoices.amount::text ILIKE ${`%${query}%`} OR
      invoices.date::text ILIKE ${`%${query}%`} OR
      invoices.status ILIKE ${`%${query}%`}
  `;

    const totalPages = Math.ceil(Number(count.rows[0].count) / ITEMS_PER_PAGE);
    return totalPages;
  } catch (error) {
    console.error('Database Error:', error);
    throw new Error('Failed to fetch total number of invoices.');
  }
}

export async function fetchInvoiceById(id: string) {
  noStore();
  try {
    const data = await sql<InvoiceForm>`
      SELECT
        invoices.id,
        invoices.customer_id,
        invoices.amount,
        invoices.status
      FROM invoices
      WHERE invoices.id = ${id};
    `;

    const invoice = data.rows.map((invoice) => ({
      ...invoice,
      // Convert amount from cents to dollars
      amount: invoice.amount / 100,
    }));

    return invoice[0];
  } catch (error) {
    console.error('Database Error:', error);
    throw new Error('Failed to fetch invoice.');
  }
}

export async function fetchCustomerPages(query: string) {
  noStore();
  try {
    const count = await sql`SELECT COUNT(*)
    FROM customers
    WHERE
      customers.name ILIKE ${`%${query}%`} OR
      customers.email ILIKE ${`%${query}%`}
  `;
```

```
    const totalPages = Math.ceil(Number(count.rows[0].count) / ITEMS_PER_PAGE);
    return totalPages;
  } catch (error) {
    console.error('Database Error:', error);
    throw new Error('Failed to fetch total number of invoices.');
  }
}

export async function fetchCustomers() {
  try {
    const data = await sql<CustomerField>`
      SELECT
        id,
        name
      FROM customers
      ORDER BY name ASC
    `;

    const customers = data.rows;
    return customers;
  } catch (err) {
    console.error('Database Error:', err);
    throw new Error('Failed to fetch all customers.');
  }
}

export async function fetchFilteredCustomers(query: string) {
  noStore();
  try {
    const data = await sql<CustomersTableType>`
    SELECT
      customers.id,
      customers.name,
      customers.email,
      customers.image_url,
      COUNT(invoices.id) AS total_invoices,
      SUM(CASE WHEN invoices.status = 'pending' THEN invoices.amount ELSE 0 END) AS
total_pending,
      SUM(CASE WHEN invoices.status = 'paid' THEN invoices.amount ELSE 0 END) AS
total_paid
    FROM customers
    LEFT JOIN invoices ON customers.id = invoices.customer_id
    WHERE
      customers.name ILIKE ${`%${query}%`} OR
        customers.email ILIKE ${`%${query}%`}
    GROUP BY customers.id, customers.name, customers.email, customers.image_url
    ORDER BY customers.name ASC
    `;

    const customers = data.rows.map((customer) => ({
      ...customer,
```

```
      total_pending: formatCurrency(customer.total_pending),
      total_paid: formatCurrency(customer.total_paid),
    }));

    return customers;
  } catch (err) {
    console.error('Database Error:', err);
    throw new Error('Failed to fetch customer table.');
  }
}

export async function getUser(email: string) {
  try {
    const user = await sql`SELECT * FROM users WHERE email=${email}`;
    return user.rows[0] as User;
  } catch (error) {
    console.error('Failed to fetch user:', error);
    throw new Error('Failed to fetch user.');
  }
}
```

```
'use server';

import {z} from 'zod';
import { sql } from '@vercel/postgres';
import { revalidatePath } from 'next/cache';
import { redirect } from 'next/navigation';
import { signIn } from '@/auth';
import { AuthError } from 'next-auth';

const FormSchema = z.object({
    id: z.string(),
    customerId: z.string({
      invalid_type_error:'Please select a customer.',
    }),
    amount: z.coerce.number().gt(0,{message:'Please enter an amount greater than
$0.'}),
    status: z.enum(['pending','paid'],{
      invalid_type_error:'Please select an invoice status.',
    }),
    name: z.string(),
    email: z.string(),
    image_url: z.string(),
    date: z.string(),
})

const CreateInvoice = FormSchema.omit({id: true, date: true});
const CreateCustomer = FormSchema.omit({id: true, date: true});
const UpdateCustomer = FormSchema.omit({id: true, date: true});
```

```ts
const UpdateInvoice = FormSchema.omit({ id: true, date: true });

export type State = {
  errors?:{
    customerId?: string[];
    name?: string[];
    amount?: string[];
    status?: string[];
    email?: string[];
    //image_url?: string[];
  };
  message?: string |null;
};

export async function createInvoice(prevState : State, formData:FormData){
    const validatedFields = CreateInvoice.safeParse({
        customerId: formData.get('customerId'),
        amount: formData.get('amount'),
        status: formData.get('status'),
    });

    if (!validatedFields.success) {
      return {
        errors: validatedFields.error.flatten().fieldErrors,
        message: 'Missing Fields. Failed to Create Invoice.',
      };
    }

    const { customerId, amount, status } = validatedFields.data;
    const amountInCents = amount *100;
    const date = new Date().toISOString().split('T')[0];

    try {
      await sql`
        INSERT INTO invoices (customer_id, amount, status, date)
        VALUES (${customerId}, ${amountInCents}, ${status}, ${date})
      `;
    } catch (error) {
      return {
        message: 'Database Error: Failed to Create Invoice.',
      };
    }

  revalidatePath('/dashboard/invoices');
  redirect('/dashboard/invoices');
}


export async function createCustomer(prevState : State, formData:FormData){
  const validatedFields = CreateCustomer.safeParse({
      name: formData.get('name'),
```

```
      email: formData.get('email'),
      //image_url: formData.get('image_url')
  });

  if (!validatedFields.success) {
    return {
      errors: validatedFields.error.flatten().fieldErrors,
      message: 'Missing Fields. Failed to Create customer.',
    };
  }

  const { name, email, image_url } = validatedFields.data;

  try {
    await sql`
      INSERT INTO customers (customer_id, email,image_url)
      VALUES (${name}, ${email}, ${image_url})
    `;
  } catch (error) {
    return {
      message: 'Database Error: Failed to Create Customer.',
    };
  }

revalidatePath('/dashboard/customers');
redirect('/dashboard/customers');
}

export async function updateInvoice(id: string, prevState: State, formData:
FormData) {
    const validatedFields  = UpdateInvoice.safeParse({
      customerId: formData.get('customerId'),
      amount: formData.get('amount'),
      status: formData.get('status'),
    });

    if (!validatedFields.success) {
      return {
        errors: validatedFields.error.flatten().fieldErrors,
        message: 'Missing Fields. Failed to Update Invoice.',
      };
    }

    const { customerId, amount, status } = validatedFields.data;
    const amountInCents = amount * 100;


    try {
      await sql`
          UPDATE invoices
```

```
        SET customer_id = ${customerId}, amount = ${amountInCents}, status =
${status}
        WHERE id = ${id}
      `;
  } catch (error) {
    return { message: 'Database Error: Failed to Update Invoice.' };
  }

  revalidatePath('/dashboard/invoices');
  redirect('/dashboard/invoices');
}

export async function updateCustomer(id: string, prevState: State, formData:
FormData) {
  const validatedFields  = UpdateInvoice.safeParse({
    customerId: formData.get('customerId'),
    amount: formData.get('amount'),
    status: formData.get('status'),
  });

  if (!validatedFields.success) {
    return {
      errors: validatedFields.error.flatten().fieldErrors,
      message: 'Missing Fields. Failed to Update Invoice.',
    };
  }

  const { customerId, amount, status } = validatedFields.data;
  const amountInCents = amount * 100;


  try {
    await sql`
        UPDATE invoices
        SET customer_id = ${customerId}, amount = ${amountInCents}, status =
${status}
        WHERE id = ${id}
      `;
  } catch (error) {
    return { message: 'Database Error: Failed to Update Invoice.' };
  }

  revalidatePath('/dashboard/invoices');
  redirect('/dashboard/invoices');
}

export async function deleteInvoice(id: string) {
  try {
    await sql`DELETE FROM invoices WHERE id = ${id}`;
    revalidatePath('/dashboard/invoices');
    return { message: 'Deleted Invoice.' };
```

```
    } catch (error) {
      return { message: 'Database Error: Failed to Delete Invoice.' };
    }
  }

  export async function deleteCustomer(id: string) {
    try {
      await sql`DELETE FROM customers WHERE id = ${id}`;
      revalidatePath('/dashboard/customers');
      return { message: 'Deleted customer.' };
    } catch (error) {
      return { message: 'Database Error: Failed to Delete customer.' };
    }
  }

  export async function authenticate(
    prevState: string | undefined,
    formData: FormData,
  ) {
    try {
      await signIn('credentials', formData);
    } catch (error) {
      if (error instanceof AuthError) {
        switch (error.type) {
          case 'CredentialsSignin':
            return 'Invalid credentials.';
          default:
            return 'Something went wrong.';
        }
      }
      throw error;
    }
  }
```

```
import SideNav from '@/app/ui/dashboard/sidenav';

export default function Layout({ children }: { children: React.ReactNode }) {
  return (
    <div className="flex h-screen flex-col md:flex-row md:overflow-hidden">
      <div className="w-full flex-none md:w-64">
        <SideNav />
      </div>
      <div className="flex-grow p-6 md:overflow-y-auto md:p-12">{children}</div>
    </div>
  );
}
```

```
import Pagination from '@/app/ui/invoices/pagination';
import Search from '@/app/ui/search';
import Table from '@/app/ui/invoices/table';
```

```tsx
import { CreateInvoice } from '@/app/ui/invoices/buttons';
import { lusitana } from '@/app/ui/fonts';
import { InvoicesTableSkeleton } from '@/app/ui/skeletons';
import { Suspense } from 'react';
import { fetchInvoicesPages } from '@/app/lib/data';
import {Metadata} from 'next';

export const metadata: Metadata ={
  title: 'Invoices | CRM Dashboard'
}

export default async function Page({
    searchParams,
}:{
    searchParams?:{
        query?: string;
        page?: string;
    };
}) {
  const query = searchParams?.query || '';
  const currentPage = Number(searchParams?.page) || 1;

  const totalPages = await fetchInvoicesPages(query);

  return (
    <div className="w-full">
      <div className="flex w-full items-center justify-between">
        <h1 className={`${lusitana.className} text-2xl`}>Invoices</h1>
      </div>
      <div className="mt-4 flex items-center justify-between gap-2 md:mt-8">
        <Search placeholder="Search invoices..." />
        <CreateInvoice />
      </div>
      <Suspense key={query + currentPage} fallback={<InvoicesTableSkeleton />}>
        <Table query={query} currentPage={currentPage} />
      </Suspense>
      <div className="mt-5 flex w-full justify-center">
        <Pagination totalPages={totalPages} />
      </div>
    </div>
  );
}
```

```tsx
'use client';

import { useEffect } from 'react';

export default function Error({
  error,
  reset,
```

```
}: {
  error: Error & { digest?: string };
  reset: () => void;
}) {
  useEffect(() => {
    // Optionally log the error to an error reporting service
    console.error(error);
  }, [error]);

  return (
    <main className="flex h-full flex-col items-center justify-center">
      <h2 className="text-center">Something went wrong!</h2>
      <button
        className="mt-4 rounded-md bg-blue-500 px-4 py-2 text-sm text-white
transition-colors hover:bg-blue-400"
        onClick={
          // Attempt to recover by trying to re-render the invoices route
          () => reset()
        }
      >
        Try again
      </button>
    </main>
  );
}
```

```
import Form from '@/app/ui/invoices/create-form';
import Breadcrumbs from '@/app/ui/invoices/breadcrumbs';
import { fetchCustomers } from '@/app/lib/data';
import {Metadata} from 'next';

export const metadata: Metadata ={
  title: 'Create Invoice | CRM Dashboard'
}

export default async function Page() {
  const customers = await fetchCustomers();

  return (
    <main>
      <Breadcrumbs
        breadcrumbs={[
          { label: 'Invoices', href: '/dashboard/invoices' },
          {
            label: 'Create Invoice',
            href: '/dashboard/invoices/create',
            active: true,
          },
        ]}
      />
```

```
      <Form customers={customers} />
    </main>
  );
}
```

```
import Form from '@/app/ui/invoices/edit-form';
import Breadcrumbs from '@/app/ui/invoices/breadcrumbs';
import {fetchInvoiceById ,fetchCustomers } from '@/app/lib/data';
import { notFound } from 'next/navigation';
import {Metadata} from 'next';

export const metadata: Metadata ={
  title: 'Edit Invoice | CRM Dashboard'
}

export default async function Page({params}:{params:{id:string}}) {
  const id = params.id;
  const [invoice,customers]= await Promise.all([
    fetchInvoiceById(id),
    fetchCustomers(),
  ]);

  if (!invoice){
    notFound();
  }

  return (
    <main>
      <Breadcrumbs
        breadcrumbs={[
          { label: 'Invoices', href: '/dashboard/invoices' },
          {
            label: 'Edit Invoice',
            href: `/dashboard/invoices/${id}/edit`,
            active: true,
          },
        ]}
      />
      <Form invoice={invoice} customers={customers} />
    </main>
  );
}
```

```
import Link from 'next/link';
import { FaceFrownIcon } from '@heroicons/react/24/outline';

export default function NotFound() {
  return (
    <main className="flex h-full flex-col items-center justify-center gap-2">
      <FaceFrownIcon className="w-10 text-gray-400" />
```

```
      <h2 className="text-xl font-semibold">404 Not Found</h2>
      <p>Could not find the requested invoice.</p>
      <Link
        href="/dashboard/invoices"
        className="mt-4 rounded-md bg-blue-500 px-4 py-2 text-sm text-white
transition-colors hover:bg-blue-400"
      >
        Go Back
      </Link>
    </main>
  );
}
```

```
import Pagination from '@/app/ui/invoices/pagination';
import Search from '@/app/ui/search';
import Table from '@/app/ui/customers/table';
import { CreateCustomer } from '@/app/ui/customers/buttons';
import { lusitana } from '@/app/ui/fonts';
import { CustomersTableSkeleton } from '@/app/ui/skeletons';
import { Suspense } from 'react';
import { fetchCustomerPages } from '@/app/lib/data';
import {Metadata} from 'next';

export const metadata: Metadata ={
  title: 'Customers | CRM Dashboard'
}

export default async function Page({
    searchParams,
}:{
    searchParams?:{
        query?: string;
        page?: string;
    };
}) {
  const query = searchParams?.query || '';
  const currentPage = Number(searchParams?.page) || 1;

  const totalPages = await fetchCustomerPages(query);

  return (
    <div className="w-full">
      <div className="flex w-full items-center justify-between">
        <h1 className={`${lusitana.className} text-2xl`}>Customers</h1>
      </div>
      <div className="mt-4 flex items-center justify-between gap-2 md:mt-8">
        <Search placeholder="Search customers..." />
        <CreateCustomer />
      </div>
      <Suspense key={query + currentPage} fallback={<CustomersTableSkeleton />}>
```

```
        <Table query={query} currentPage={currentPage} />
      </Suspense>
      <div className="mt-5 flex w-full justify-center">
        <Pagination totalPages={totalPages} />
      </div>
    </div>
  );
}
```

```
import Form  from '@/app/ui/customers/create-form';
import Breadcrumbs from '@/app/ui/invoices/breadcrumbs';
import { fetchCustomers } from '@/app/lib/data';
import {Metadata} from 'next';

export const metadata: Metadata ={
  title: 'Create Invoice | CRM Dashboard'
}

export default async function Page() {

  return (
    <main>
      <Breadcrumbs
        breadcrumbs={[
          { label: 'Customers', href: '/dashboard/customers' },
          {
            label: 'Create Customer',
            href: '/dashboard/customers/create',
            active: true,
          },
        ]}
      />
      <Form />
    </main>
  );
}
```

```
import Form from '@/app/ui/invoices/edit-form';
import Breadcrumbs from '@/app/ui/invoices/breadcrumbs';
import {fetchInvoiceById ,fetchCustomers } from '@/app/lib/data';
import { notFound } from 'next/navigation';
import {Metadata} from 'next';

export const metadata: Metadata ={
  title: 'Edit Invoice | CRM Dashboard'
}

export default async function Page({params}:{params:{id:string}}) {
  const id = params.id;
  const [invoice,customers]= await Promise.all([
```

```
    fetchInvoiceById(id),
    fetchCustomers(),
  ]);

  if (!invoice){
    notFound();
  }

  return (
    <main>
      <Breadcrumbs
        breadcrumbs={[
          { label: 'Invoices', href: '/dashboard/invoices' },
          {
            label: 'Edit Invoice',
            href: `/dashboard/invoices/${id}/edit`,
            active: true,
          },
        ]}
      />
      <Form invoice={invoice} customers={customers} />
    </main>
  );
}
```

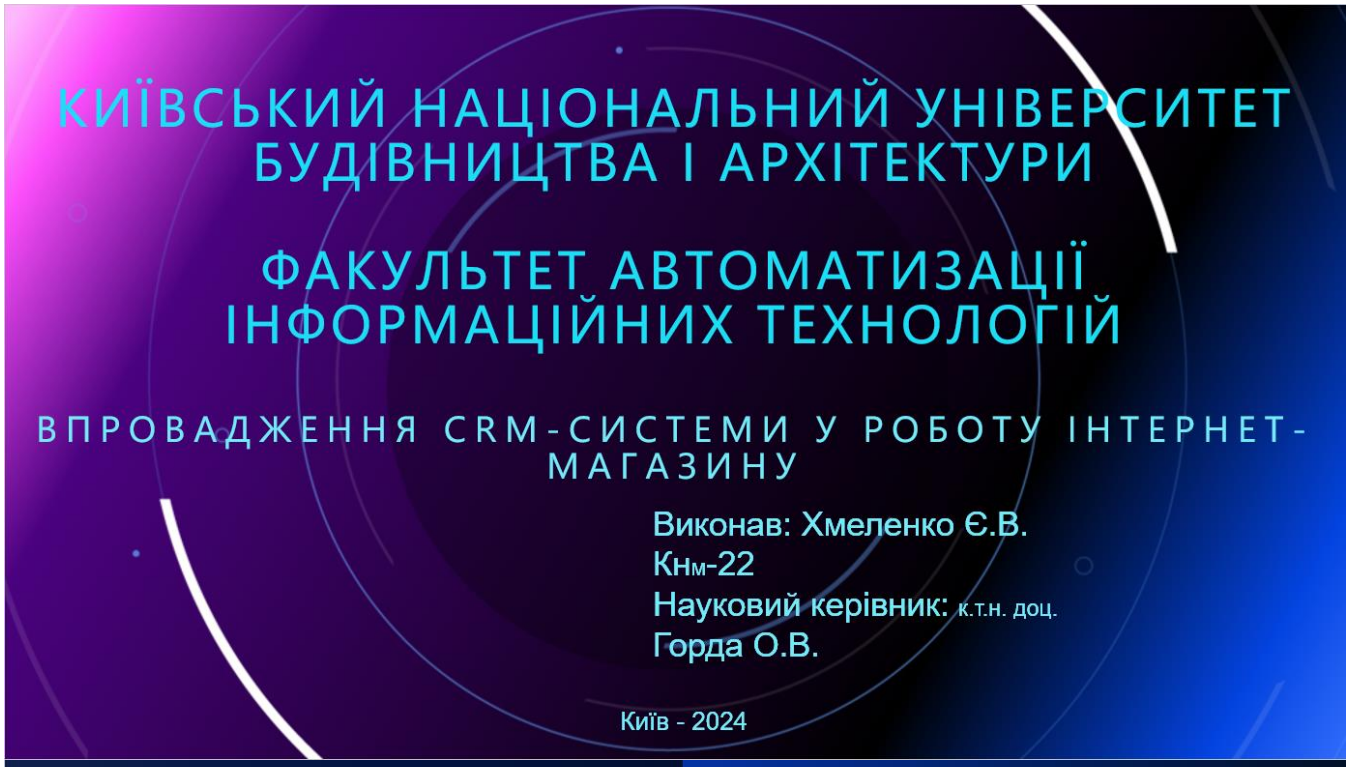
```
import Link from 'next/link';
import { FaceFrownIcon } from '@heroicons/react/24/outline';

export default function NotFound() {
  return (
    <main classNam="flex h-full flex-col items-center justify-center gap-2">
      <FaceFrownIcon className="w-10 text-gray-400" />
      <h2 className="text-xl font-semibold">404 Not Found</h2>
      <p>Could not find the requested customer.</p>
      <Link
        href="/dashboard/customers"
        className="mt-4 rounded-md bg-blue-500 px-4 py-2 text-sm text-white
transition-colors hover:bg-blue-400"
      >
        Go Back
      </Link>
    </main>
  );
}
```

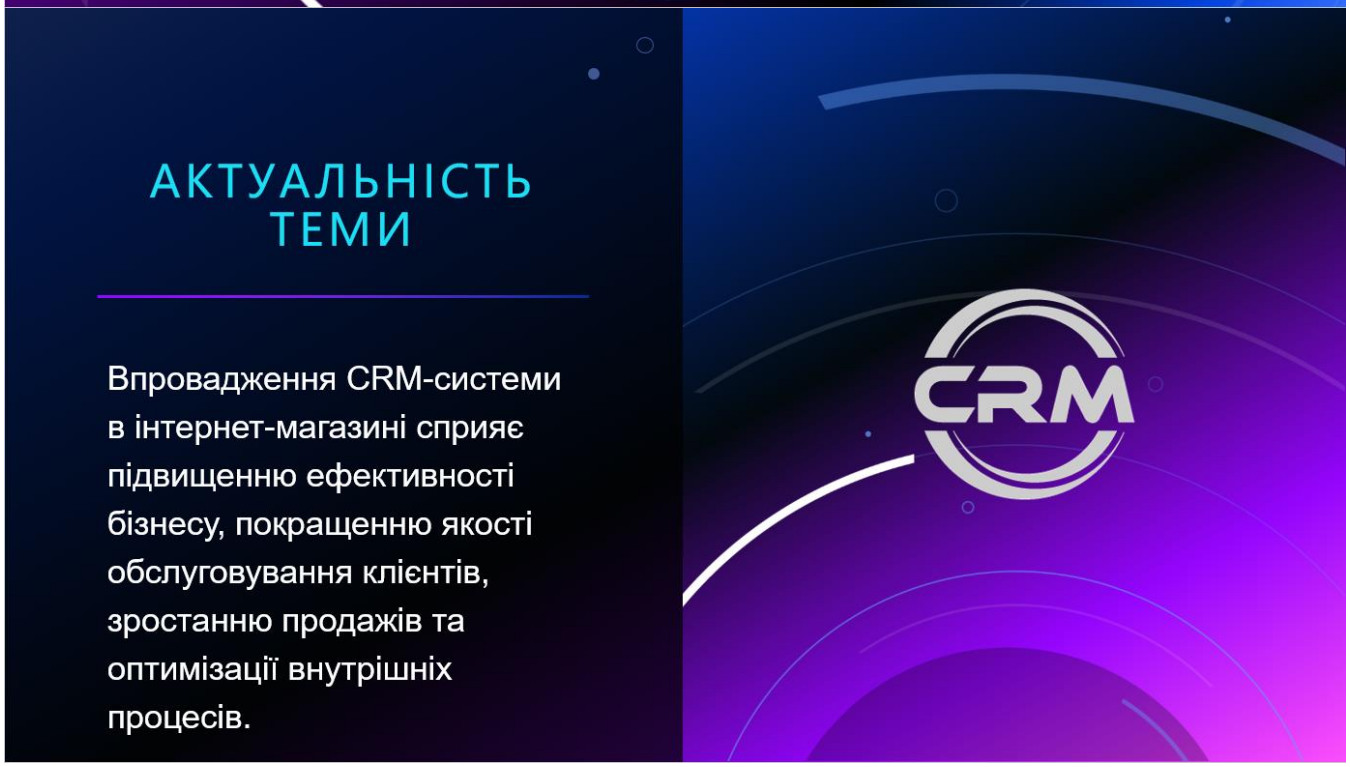
```
import RevenueChart from '@/app/ui/dashboard/revenue-chart';
import LatestInvoices from '@/app/ui/dashboard/latest-invoices';
import CardWrapper from '@/app/ui/dashboard/cards';
import { lusitana } from '@/app/ui/fonts';
import {fetchCardData } from '@/app/lib/data';
```

```tsx
import { Suspense } from 'react';
import { RevenueChartSkeleton, LatestInvoicesSkeleton, CardsSkeleton, CardSkeleton
} from '@/app/ui/skeletons';
import {Metadata} from 'next';

export const metadata: Metadata ={
  title: 'Dashboard | CRM Dashboard'
}

export default async function Page() {
  const {totalPaidInvoices, totalPendingInvoices, numberOfCustomers,
numberOfInvoices} = await fetchCardData()
  return (
    <main>
      <h1 className={`${lusitana.className} mb-4 text-xl md:text-2xl`}>
        Dashboard
      </h1>
      <div className="grid gap-6 sm:grid-cols-2 lg:grid-cols-4">
        <Suspense fallback={<CardSkeleton/>}>
          <CardWrapper/>
        </Suspense>
      </div>
      <div className="mt-6 grid grid-cols-1 gap-6 md:grid-cols-4 lg:grid-cols-8">
        <Suspense fallback={<RevenueChartSkeleton/>}>
          <RevenueChart />
        </Suspense>
        <Suspense fallback={<LatestInvoicesSkeleton/>}>
          <LatestInvoices/>
        </Suspense>
      </div>
    </main>
  );
}
```

```tsx
import DashboardSkeleton from "@/app/ui/skeletons"

export default function Loading(){
    return <DashboardSkeleton/>
}
```

Додаток В


КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ
ФАКУЛЬТЕТ АВТОМАТИЗАЦІЇ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
ВПРОВАДЖЕННЯ CRM-СИСТЕМИ У РОБОТУ ІНТЕРНЕТ-МАГАЗИНУ
Виконав: Хмеленко Є.В.
Кнм-22
Науковий керівник: к.т.н. доц.
Горда О.В.
Київ - 2024


АКТУАЛЬНІСТЬ ТЕМИ
Впровадження CRM-системи в інтернет-магазині сприяє підвищенню ефективності бізнесу, покращенню якості обслуговування клієнтів, зростанню продажів та оптимізації внутрішніх процесів.
CRM

НАУКОВЕ ЗАВДАННЯ

Дослідити ринок CRM-систем та вплив CRM-системи на роботу інтернет-магазину

НАУКОВА МЕТА

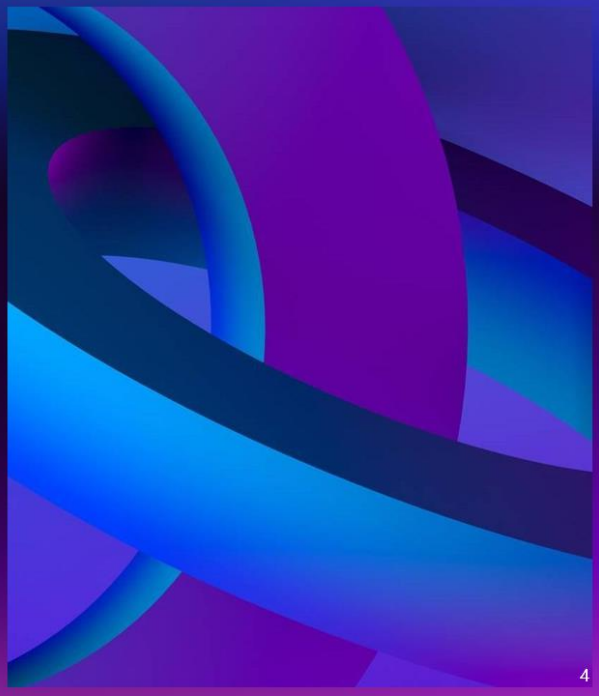
Розробити CRM-систему задовільну основним функціональним вимогам

3

ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕНЬ

Об'єктом дослідження є інтернет-магазин як суб'єкт електронної комерції.

Предметом дослідження є процес впровадження CRM-системи у роботу інтернет-магазину.



4

ТЕРМІНОЛОГІЯ

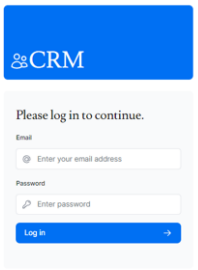
- CRM-система
- KPI (Key performance indicators)
- Leads
- UX (User Experience)
- Омніканальність (Omnichannel)
- Клієнтська база
- Аналітика продажів
- Сегментація клієнтів

5

ПРОБЛЕМАТИКА

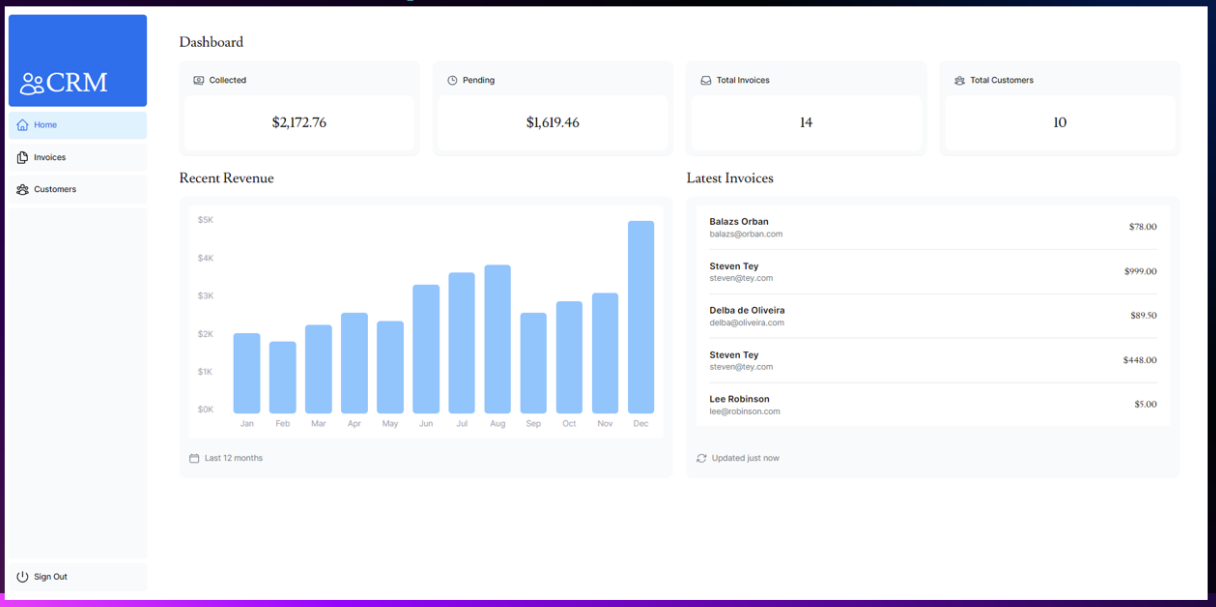
1. Неефективне управління клієнтською базою
2. Втрата потенційних клієнтів
3. Складнощі в управлінні продажами
4. Низький рівень обслуговування клієнтів
5. Неможливість аналізу та звітності
6. Відсутність автоматизації маркетингу
7. Проблеми з інтеграцією

CRM-СИСТЕМА



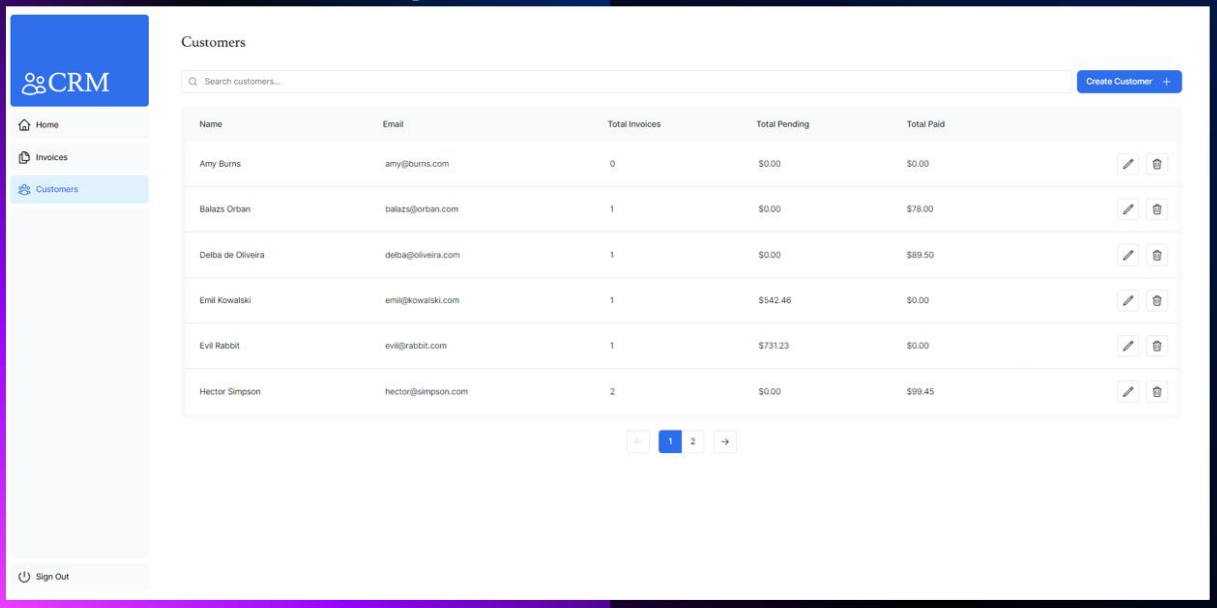
7

Сторінка Dashboard



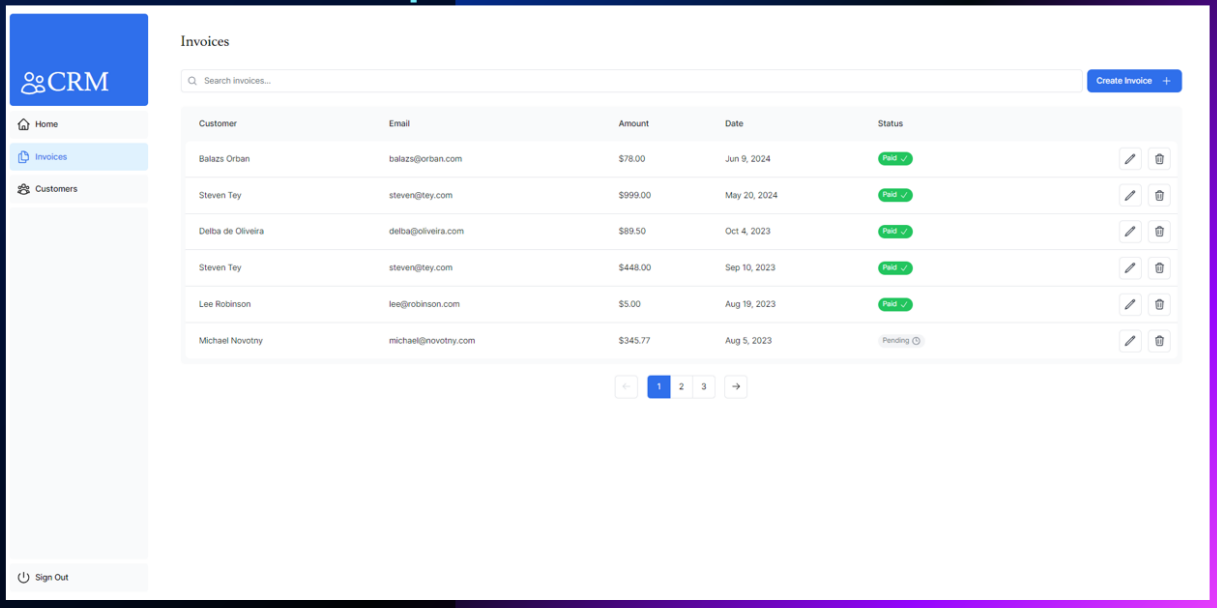
8

Сторінка Customers



9

Сторінка Invoices



10

ВИКОРИСТАНІ ТЕХНОЛОГІЇ

- JavaScript
- React
- Next.js
- Vercel
- PostgreSQL
- Node.Js
- Git
- TailwindCSS

11

ВИСНОВКИ

Впровадження CRM-системи в роботу інтернет-магазину є стратегічно важливим кроком, який дозволяє підвищити ефективність бізнес-процесів, покращити якість обслуговування клієнтів та збільшити продажі.

Основними перевагами CRM-систем є автоматизація рутинних завдань, інтеграція з іншими бізнес-системами, потужні аналітичні інструменти та можливість персоналізації взаємодії з клієнтами.

CRM-системи – майбутнє не тільки електронної комерції, але й загальне майбутнє всього бізнесу.