МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
Факультет мехатроніки та комп'ютерних технологій
Кафедра комп'ютерних наук

ДИПЛОМНА БАКАЛАВРСЬКА РОБОТА

на тему

РОЗРОБКА ВЕБ ЗАСТОСУНКУ ДЛЯ КОМЕРЦІЙНОЇ ДІЯЛЬНОСТІ

Виконав: студент групи БІТск1-21
спеціальності 122 Комп'ютерні науки
Валерій МИКИТЕНКО
Науковий керівник
Ганна КОРОГОД
Рецензент
Володимир ЩЕРБАНЬ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп’ютерних технологій

Кафедра комп’ютерних наук

Спеціальність 122 Комп’ютерні науки

Освітня програма Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп’ютерних наук

_____Володимир ЩЕРБАНЬ

«_____» _____ 20 __ р.

ЗАВДАННЯ

НА ДИПЛОМНУ БАКАЛАВРСЬКУ РОБОТУ

студенту

Микитенку Валерію Анатолійовичу

1. Тема роботи: Розробка веб застосунку для комерційної діяльності.
Науковий керівник роботи: Корогод Ганна Олександрівна, доцент кафедри комп'ютерних наук, затверджені наказом КНУТД від “08” листопада 2022 року № 224-уч.
2. Строк подання студентом дипломної роботи: 25.05.2023р.
3. Вихідні дані до дипломної бакалаврської роботи: Розробки кафедри комп’ютерних наук, рекомендована література, додатки.
4. Зміст дипломної бакалаврської роботи: Розділ 1. Теоретичні аспекти, Розділ 2. Проектування та алгоритм формування баз даних, Розділ 3. Програмна частина.
5. Дата видачі завдання: 06.02.2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної бакалаврської роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	20.04.2023р.	
2	Розділ 1. Теоретичні аспекти частина	25.04.2023р.	
3	Розділ 2. Проектування та алгоритм формування баз даних	05.05.2023р.	
4	Розділ 3. Програмна частина	15.05.2023р.	
5	Висновки	16.05.2023р.	
6	Оформлення дипломної бакалаврської роботи	20.05.2023р.	
7	Здача дипломної бакалаврської роботи на кафедру для рецензування	25.05.2023р.	
8	Перевірка дипломної бакалаврської роботи на наявність ознак плагіату		
9	Подання дипломної роботи (проекту) на затвердження завідувачу кафедри (за 7 днів до захисту)		

Студент _____ Валерій МИКИТЕНКО

Науковий керівник роботи _____ Ганна КОРОГОД

Рецензент _____ Володимир ЩЕРБАНЬ

АНОТАЦІЯ

Об’єкт дослідження – системи управління взаємовідносинами з клієнтами (CRM - Customer Relationship Management).

Метою роботи було створення комплексного програмного рішення у вигляді веб-застосунку, яке допоможе бізнесу ефективно управляти взаємодією з клієнтами, оптимізувати процеси продажів і підвищити загальну задоволеність клієнтів.

В роботі вирішено завдання по створенню веб-застосунку для комерційної діяльності системи управління взаємовідносинами з клієнтами (CRM - Customer Relationship Management). Для вирішення цієї задачі CRM-система буде побудована з використанням React, використовуючи його компонентну архітектуру та розгалужену екосистему бібліотек і фреймворків. Система забезпечить зручний інтерфейс для управління даними про клієнтів, відстеження потенційних клієнтів, а також сприятиме ефективній комунікації та співпраці між членами команди.

Ключові слова: CRM-система, веб-додаток, база даних, бізнес.

ABSTRACT

Research object – customer relationship management systems (CRM - Customer Relationship Management).

The goal of the work was to create a comprehensive software solution in the form of a web application that will help businesses effectively manage interaction with customers, optimize sales processes and increase overall customer satisfaction.

The work solves the task of creating a web application for the commercial activity of the customer relationship management system (CRM - Customer Relationship Management). To solve this problem, the CRM system will be built using React, using its component architecture and extensive ecosystem of libraries and frameworks. The system will provide a convenient interface for managing customer data, tracking potential customers, and will promote effective communication and collaboration between team members.

Keywords: CRM system, web application, database, business.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CRM *(Customer Relationship Management)* – Система управління відносинами з клієнтами

URL *(Uniform Resource Locator)* – Визначник місцезнаходження сайту в мережі Інтернет

IDE *(Integrated development environment* – Інтегроване середовище розробки

API (Application programming interface) – Програмний інтерфейс додатків

HTTP (HyperText Transfer Protocol) – Протокол передачі даних, що використовується в комп'ютерних мережах

JSON (JavaScript Object Notation) – Текстовий формат, призначений для зберігання структурованих даних

SQL (Structured Query Language) – Мова структурованих запитів для роботи з БД

CRM (Customer Relationship Management) – Система управління взаємовідносинами з клієнтами

XML (eXtensible Markup Language) – Розширювана мова розмітки

БД – База даних

Зміст

ВСТУП

У сучасну цифрову епоху, компанії будь-якого розміру все більше і більше переміщують свої операції в онлайн, що призвело до зростання електронної комерції та онлайн-маркетплейсів, які створили нові можливості у бізнесі для взаємодії з клієнтами, оптимізації операцій та зростання доходів. Таким чином, розробка ефективного веб-додатку для комерційної діяльності стала важливим фактором успіху багатьох компаній.

За останні роки розробка веб-додатків зазнала значного прогресу з точки зору інструментів, технологій та методологій. З появою адаптивного веб-дизайну, хмарних обчислень та інтернет речей (IoT - Internet of Things) компанії тепер можуть створювати високоінтерактивні та зручні для користувача веб-додатки, які можуть задовольнити широкий спектр комерційної діяльності, включаючи онлайн-покупки, обробку платежів, управління запасами та управління відносинами з клієнтами CRM.

CRM передбачає використання технологій для управління взаємодією з клієнтами та отримання інформації про них та про їхню поведінку. Це важлива стратегія для компаній, які прагнуть побудувати міцніші стосунки зі своїми клієнтами, працювати більш ефективно і розвивати свій бізнес. Використовуючи CRM, компанії можуть досягти цих цілей ефективніше, ніж будь-коли раніше.

Дипломна робота має на меті дослідити розробку веб-додатків для комерційної діяльності з особливим акцентом на CRM. У ній будуть розглянуті переваги використання веб-додатків для бізнесу, дано визначення, особливості і переваги CRM, крім того буде представлена розробка власної CRM-системи.

У першому розділі цієї дипломної роботи дано визначення веб-додатків та їх значення для комерційної діяльності. Представлено переваги використання веб-додатків для бізнесу, архітектура веб-додатків і види небезпек та шляхи боротьби з ними.

В другому розділ дипломної роботи дано визначення CRM системі, представлено алгоритм проектування CRM та вибір технологій для його реалізації. Крім того, були представлені шляхи оптимізації бізнес-операцій з клієнтами, а також його значення для бізнесу.

Третій розділ присвячений практичній реалізації CRM системи з використанням HTML, CSS, JS, React та інших технологій. У цьому розділі надано покрокове представлення створення CRM. Програмна реалізація розроблена таким чином, щоб її можна було легко налаштовувати та масштабувати, для успішної комерційної діяльності компанії будь-якого направлення.

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ

1.1. Веб-додатки і їх важливість для сучасного бізнесу

Розглянемо фундаментальні відмінності між вебсайтом, веб-браузером і веб-додатком.

Веб-додаток – це програмне забезпечення, яке працює на веб-сервері і доступ до якого здійснюється через вебсайт, до якого, в свою чергу, надається доступ через веб-браузер. Він надає розширені функціональні можливості, такі як автоматизація процесів, зберігання даних та сповіщення. На відміну від них, вебсайти в основному складаються зі статичного контенту, такого як статті, відео або зображення, що супроводжуються посиланнями на інші вебсайти. Їм бракує можливостей для автоматизації процесів, зберігання даних або самостійного надсилання сповіщень. Види веб-додатків представлені на рис. 1.1.

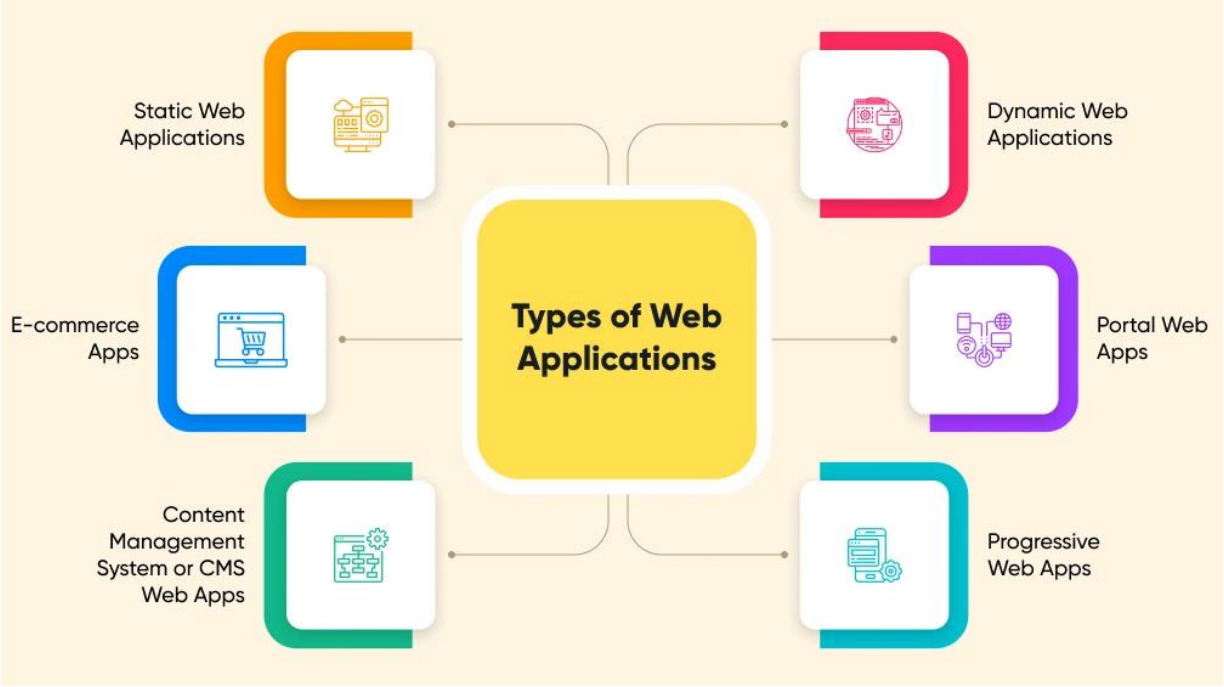

Рисунок 1.1 – Види веб-додатків

Веб-браузер (також відомий як інтернет-браузер) – це програмне забезпечення, що використовується для перегляду веб-сторінок у Всесвітній павутині. Веб-браузери дозволяють користувачам переходити за гіперпосиланнями, вводити URL-адреси, виконувати пошукові запити,

завантажувати та відтворювати різноманітний веб-контент, такий як текст, зображення, відео, аудіофайли та інші медіа. Блок-схема роботи веб-браузера наведена на рис. 1.2. Деякі з найпопулярніших веб-браузерів на сьогоднішній день:

– Google Chrome;

– Mozilla Firefox;

– Microsoft Edge;

– Safari;

– Opera.

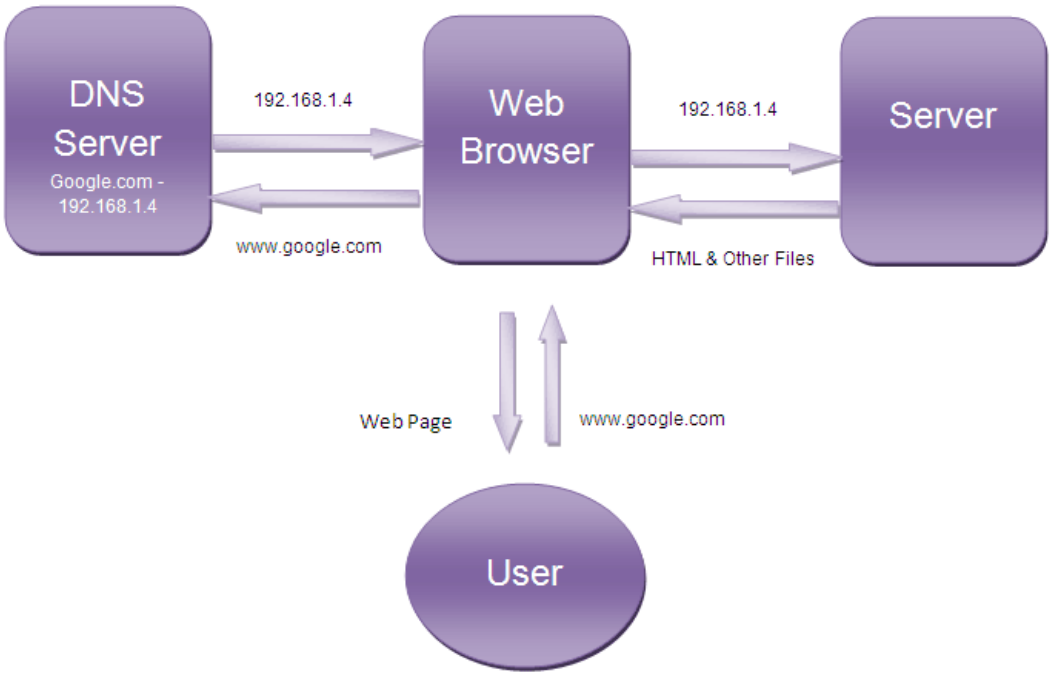

Рисунок 1.2 – Блок-схема роботи веб-браузера

Веб-додатки, які часто називають “динамічними вебсайтами”, використовують БД для виконання більш складних завдань. Вони дають можливість користувачам вносити зміни в режимі реального часу, наприклад, додавати або змінювати контент, і зберігати інформацію безпосередньо на своїх пристроях.

Якщо підсумувати, вебсайти статичні, тоді як веб-додатки динамічні та інтерактивні. Розуміння цієї різниці є життєво важливим для бізнесу, який прагне ефективно використовувати ці цифрові платформи.

1.1.1. Переваги веб-додатків для сучасного бізнесу

Розробка веб-додатків пропонує численні переваги для бізнесу, щоб покращити комунікацію та операційну ефективність з клієнтами. Вона спрощує процес відстеження та аналізу цінних даних, дозволяючи компаніям приймати обґрунтовані рішення та зберігати конкурентну перевагу.

Використовуючи веб-додатки, компанії отримують доступ до потужних інструментів, які сприяють зворотному зв’язку з клієнтами в режимі реального часу. Цей безцінний зворотний зв’язок допомагає виявити нові тенденції та проблеми, які впливають на загальний досвід клієнтів. Як результат, компанії можуть швидко та ефективно вирішувати ці проблеми, забезпечуючи швидке вирішення та підвищення рівня задоволеності клієнтів.

1. Веб-додатки прості у використанні та завжди доступні.

Веб-додатки пропонують зручний інтерфейс, який завжди доступний для користувачів. Завдяки надійному інтернет-з’єднанню користувачі можуть насолоджуватися оптимальним інтерфейсом, в якому легко орієнтуватися.

Порівняно зі статичними вебсайтами, веб-додатки забезпечують кращий користувацький досвід. Вони здатні виконувати складні завдання, що виходять за рамки можливостей вебсайтів, забезпечуючи більш динамічний користувацький досвід.

Численні компанії використовують веб-додатки, щоб надати своїм клієнтам доступ до послуг, еквівалентних тим, що надаються у фізичних магазинах. Це включає в себе такі функції, як бронювання зустрічей та онлайн-покупки. Це особливо вигідно для клієнтів, які не можуть відвідати фізичний магазин, оскільки вони все одно можуть отримати доступ до того ж спектру інформації та послуг.

2. Підвищення ефективності бізнес-операцій за допомогою веб-додатків.

Використання веб-систем може оптимізувати обмін даними та сприяти співпраці між користувачами в єдиному середовищі завдяки хмарному зберіганню інформації.

Використовуючи хмарні робочі процеси та автоматизацію, організації можуть замінити ручні або паперові процеси. Таке вдосконалення бізнес-процесів має потенціал для підвищення продуктивності працівників і зниження витрат.

Для компаній, які значною мірою покладаються на завдання, керовані даними, варто присвятити час і зусилля розробці веб-додатків для подальшої оптимізації своїх операцій. Веб-додатки використовують потужність хмарних обчислень, усуваючи залежність від локального обладнання.

3. Підвищення гнучкості та масштабованості за допомогою веб-додатків.

Щоб відповідати мінливим потребам вашого бізнесу, програмне забезпечення повинно легко адаптуватися до зростання та змін, таких як розширення в нових місцях або оптимізація операцій для підвищення ефективності.

Спеціалізовані веб-додатки забезпечують гнучкість у впровадженні нових функцій і можливостей, а також безперешкодну інтеграцію з іншими системами.

Використання хмарних серверів дає змогу масштабувати обчислювальні потужності та сховища відповідно до ваших конкретних вимог, забезпечуючи оптимальну масштабованість та адаптивність.

4. Спрощене розгортання, оновлення та обслуговування веб-додатків.

Розгортання веб-програми - це простий процес. Після встановлення програмного забезпечення на хост-сервері користувачі можуть отримати доступ до нього за вказаною URL-адресою.

На відміну від мобільних додатків, які потребують окремого завантаження та встановлення оновлень, веб-додатки спрощують цей процес. Оновлення впроваджуються централізовано на хост-сервері, забезпечуючи всім користувачам доступ до однієї і тієї ж версії в будь-який час, не вимагаючи індивідуальних оновлень на пристроях.

Використовуючи веб-додатки, ви можете не тільки легко і швидко впроваджувати нове програмне забезпечення, але й полегшити проблеми з технічним обслуговуванням. Ваша система залишається стабільною, мінімізуючи складнощі з обслуговуванням.

5. Надання клієнтам можливостей самообслуговування за допомогою веб-додатків.

Однією з переконливих переваг вибору веб-додатку над традиційним вебсайтом є можливість запропонувати клієнтам варіанти самообслуговування. За допомогою веб-додатку клієнти можуть виконувати завдання самостійно, усуваючи необхідність прямої взаємодії з представниками служби підтримки.

Розглянемо сценарій продажу полісу автострахування. Хоча цей продукт часто потребує допомоги, використання веб-додатку дозволяє клієнтам керувати своїм полісом без втручання людини.

Ця можливість, зазвичай недосяжна за допомогою стандартного вебсайту, виявляється надзвичайно вигідною для глобальних підприємств, які мають справу з розпорошеною клієнтською базою. Це дозволяє великим компаніям утримувати розгалужену команду з обслуговування клієнтів, одночасно надаючи клієнтам по всьому світу можливість самообслуговування.

6. Підвищення лояльності клієнтів за допомогою веб-додатків.

Створення веб-додатків може суттєво сприяти підвищенню лояльності клієнтів, особливо для компаній зі значною клієнтською базою, які прагнуть забезпечити індивідуальний підхід до кожного клієнта.

Веб-додаток полегшує надання персоналізованого досвіду, пристосованого до конкретних уподобань кожного клієнта. Він дозволяє компаніям пропонувати персоналізований контент та інформацію у форматі, який є легко зрозумілим і резонує з кожним клієнтом.

Використовуючи можливості веб-додатків, компанії можуть гарантувати, що кожен клієнт відчуває, що його цінують, і отримує саме те,

що йому потрібно, створюючи динамічний і цікавий досвід, який зміцнює лояльність клієнтів.

1.2. Архітектура веб додатків

На сучасному етапі розвитку інформаційних технологій статичні веб-сторінок втратили свою актуальність. Інтернет зазнав значної трансформації, надаючи веб-додаткам розширену функціональність. Таким чином, перш ніж розпочати проект, необхідно вибрати відповідний тип архітектури веб-додатків і модель компонентів.

Архітектура веб-додатків – це структура, яка визначає взаємодію між додатками, базами даних і системами проміжного програмного забезпечення. Вона забезпечує безперебійну роботу декількох додатків одночасно. На рис. 1.3 представлено архітектуру веб-додатків при відкритті веб-сторінки.

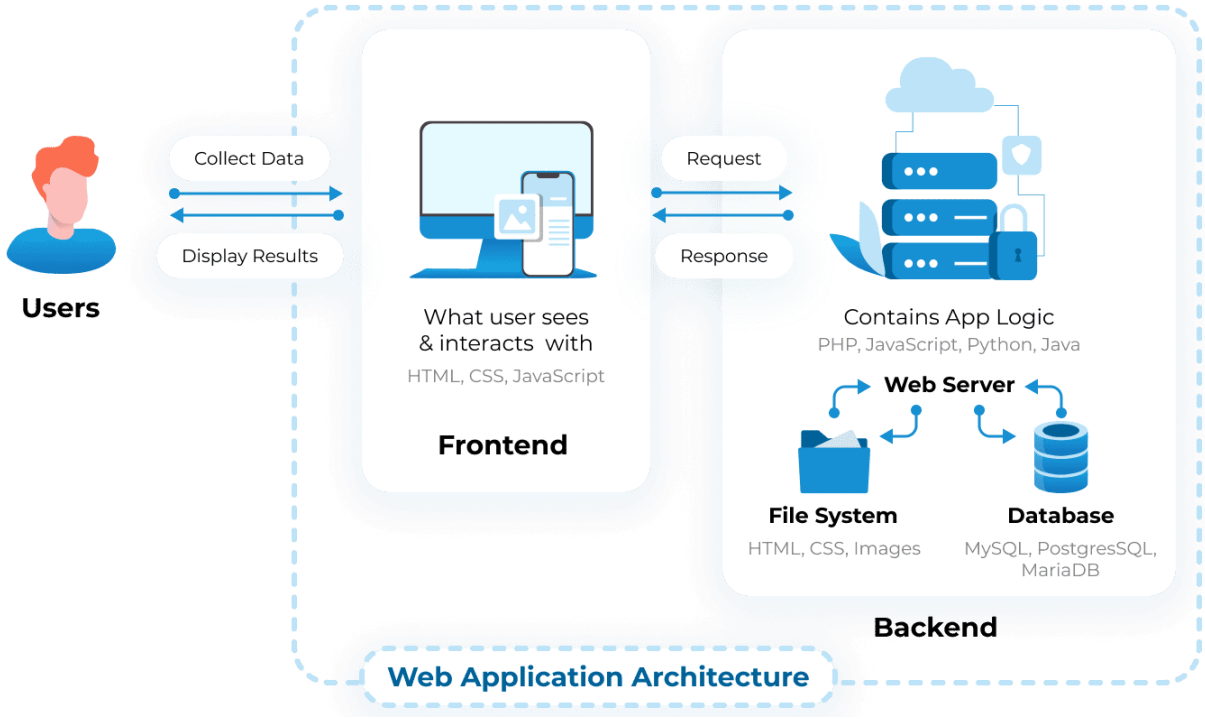

Рисунок 1.3 – Архітектура веб-додатків при відкритті веб-сторінки

Після вводу користувачем URL-адресу в адресному рядку веб-браузера, ініціюється запит на конкретну веб-адресу. У відповідь на запит сервер надсилає браузеру необхідні файли. Потім браузер обробляє надіслані файли

для відображення запитуваної сторінки. Це дозволяє користувачеві (user) взаємодіяти з веб-сторінкою.

Важливим аспектом, на який слід звернути увагу, є код, який інтерпретується веб-браузером. Цей код містить конкретні інструкції, які керують реакцією браузера на різні дії користувача.

Таким чином, у сучасному світі архітектура веб-додатків відіграє важливу роль у поширенні веб-комунікації між додатками та пристроями.

Ефективна архітектура веб-додатків повинна враховувати не лише обсяг глобального мережевого трафіку, але й масштабованість та безпеку. Ці аспекти мають вирішальне значення при забезпеченні оптимальної продуктивності та зручності для користувачів.

1.2.1. Функціональність архітектури веб-додатків

Код на стороні клієнта. Цей код знаходиться у веб-браузері користувача і реагує на вхідні дані користувача. Він відповідає за відображення користувацького інтерфейсу та обробку взаємодії на стороні клієнта. Для розробки клієнтського коду зазвичай використовуються такі технології, як CSS, HTML і JavaScript. На відміну від коду на стороні сервера, код на стороні клієнта може переглядати і змінювати користувач. Він взаємодіє з сервером за допомогою HTTP-запитів.

Код на стороні сервера. Цей код працює на сервері і відповідає на HTTP-запити, надіслані клієнтським кодом. Веб-розробники або команди розробників визначають поведінку коду на стороні сервера по відношенню до коду на стороні клієнта. Для розробки серверного коду використовуються такі популярні мови програмування, як C#, Java, JavaScript, Python, PHP та Ruby.

Код на стороні сервера відповідає за створення запитуваної веб-сторінки та обробку різних типів даних, включаючи профілі користувачів і введення. Однак він залишається прихованим від кінцевого користувача, працюючи за лаштунками.

Важливо відзначити, що код на стороні клієнта і код на стороні сервера співпрацюють через HTTP-запити, але код на стороні клієнта не має прямого

доступу до файлів сервера. Він може спілкуватися з сервером лише за допомогою встановленого механізму запитів-відповідей.

Можна виділити два основні компоненти веб-додатків:

1) **UI/UX компоненти** – журнали активності, інформаційні панелі, сповіщення, налаштування, статистика тощо. Дані компоненти в першу чергу зосереджені на візуальних та інтерактивних аспектах дизайну інтерфейсу веб-додатку. UI/UX компоненти відокремлені від основної роботи архітектури веб-додатку;

2) **структурні компоненти**, що, в свою чергу, складаються з клієнтської та серверної частин:

– *клієнтський компонент*, що знаходиться у веб-браузері користувача. Розроблений з використанням CSS, HTML і JavaScript, він не потребує спеціальних налаштувань. Клієнтський компонент представляє функціональні елементи веб-додатку, з якими безпосередньо взаємодіє кінцевий користувач.

– *серверний компонент*, який побудовано з використанням різних мов програмування і фреймворків, таких як Java, .NET, Node.js, PHP, Python і Ruby on Rails. Він складається з двох основних частин: логіки програми та БД. Логіка додатку служить центральним центром управління веб-додатком, обробляючи обробку та бізнес-логіку. З іншого боку, база даних служить сховищем для постійних даних, що використовуються веб-додатком.

Співпраця між клієнтськими та серверними компонентами формує основу веб-додатку, забезпечуючи безперебійний зв’язок та взаємодію між інтерфейсом користувача та базовою функціональністю.

1.2.2. Моделі компонентів веб-додатків

Модель веб-додатку залежить від кількості задіяних серверів і баз даних. Існує три основні моделі:

Один веб-сервер, одна база даних. Це проста, але ненадійна модель, де використовується один сервер і одна база даних. Якщо сервер виходить з ладу, веб-додаток стає недоступним. Ця модель зазвичай використовується для тестових проектів або навчальних цілей, а не для реальних веб-додатків.

Кілька веб-серверів, одна база даних (бездержавна архітектура). У цій моделі використовується декілька веб-серверів, але база даних управляється окремо. Веб-сервери обробляють клієнтську інформацію і записують її в зовнішню базу даних. Така бездержавна архітектура забезпечує кращу надійність, оскільки один сервер може взяти на себе роботу, якщо інший вийде з ладу. Однак, якщо база даних виходить з ладу, це впливає на веб-додаток.

Кілька веб-серверів, кілька баз даних. Ця модель є найбільш ефективною та надійною. І веб-сервери, і бази даних мають надлишковість, що виключає єдину точку відмови. Дані можуть зберігатися однаково в усіх базах даних або рівномірно розподілятися між ними. Для великих масштабів рекомендується використовувати балансувальники навантаження для управління розподілом запитів між серверами.

Ці моделі пропонують різні рівні надійності та масштабованості, і вибір залежить від конкретних потреб і вимог веб-додатку.

1.2.3. Типи архітектур веб-додатків

Тип архітектури визначається тим, як логіка програми розподіляється між клієнтською та серверною сторонами. Існує три основні типи архітектури:

SPA (Single Page Application - Односторінкові додатки). SPA динамічно оновлюють вміст поточної сторінки замість того, щоб завантажувати з сервера абсолютно нові сторінки на кожну дію користувача. Вони використовують AJAX (асинхронний JavaScript і XML) для безперебійної взаємодії сторінок, нагадуючи традиційні десктопні додатки. SPA забезпечують інтуїтивно зрозумілу та інтерактивну взаємодію з користувачем, запитуючи лише необхідний контент та інформаційні елементи.

Мікросервіси. Мікросервіси – це невеликі, незалежні сервіси, які виконують специфічні функції. Переваги мікросервісів – підвищена продуктивність і швидке розгортання. Компоненти в додатку на основі мікросервісів не пов’язані жорстко, що дозволяє розробникам використовувати різні мови програмування і технологічні стеки. Така гнучкість спрощує і прискорює розробку додатків.

Безсерверні архітектури. У безсерверній архітектурі розробник програми передає управління сервером та інфраструктурою сторонньому постачальнику хмарних послуг. Такий підхід дозволяє розробникам зосередитися виключно на написанні логіки коду, не турбуючись про обслуговування серверів та інфраструктури. Безсерверні архітектури ідеально підходять, коли компанія-розробник хоче зняти з себе обов'язки з управління серверами та апаратним забезпеченням.

Кожен тип архітектури веб-додатків має свої переваги і підходить для різних сценаріїв, залежно від конкретних вимог програми та уподобань команди розробників.

1.3. Безпека веб-додатків

Безпека веб-додатків – це комплекс заходів і практик, спрямованих на захист вебсайтів, додатків і API від потенційних кіберзагроз і атак. Ця комплексна дисципліна спрямована на забезпечення безперебійного функціонування веб-додатків, одночасно захищаючи бізнес від різних негативних наслідків, таких як кібер-вандалізм, крадіжка даних і недобросовісна конкуренція.

Враховуючи глобальну природу Інтернету, веб-додатки та API є вразливими до атак, які можуть здійснюватися з різних місць, мати різні масштаби та складність. Отже, безпека веб-додатків охоплює широкий спектр стратегій і охоплює різні етапи ланцюжка постачання програмного забезпечення. Впроваджуючи надійні заходи безпеки, організації можуть зменшити ризики та підвищити загальний рівень захищеності своїх веб-додатків.

1.3.1. Типові ризики пов'язані з безпекою веб-додатків

Веб-додатки вразливі до різних типів атак, які залежать від цілей зловмисників, характеру діяльності організації, на яку спрямована атака, та конкретних вразливостей безпеки додатку. Деякі з найпоширеніших типів атак включають:

Вразливості нульового дня: Це невідомі вразливості, які розробники програми ще не усунули. Як наслідок, для них не існує доступного рішення. За оцінками, щороку виявляють понад 20 000 вразливостей нульового дня. Зловмисники оперативно використовують ці вразливості і часто намагаються обійти заходи безпеки, що впроваджуються постачальниками програмного забезпечення.

XSS (Cross site scripting - Міжсайтовий скриптинг). XSS - це вразливість, яка дозволяє зловмисникам впроваджувати шкідливі скрипти на стороні клієнта у веб-сторінки. Це дозволяє зловмисникам отримати прямий доступ до критично важливої інформації, видавати себе за користувачів або обманом змусити їх розкрити важливі дані.

SQLi (SQL Injection - SQL-ін'єкції). SQLi передбачає використання зловмисниками вразливостей у способі виконання пошукових запитів у базі даних веб-додатків. Впроваджуючи шкідливий SQL-код, зловмисники можуть отримати несанкціонований доступ до конфіденційної інформації, маніпулювати або створювати нові дозволи користувачів, або навіть маніпулювати і знищувати цінні дані.

DoS (Denial of Service - Атаки на Відмову в Обслуговуванні) та DDoS (Distributed Denial of Service - Розподілені Атаки на Відмову в Обслуговуванні). Зловмисники використовують різні методи, щоб перевантажити цільовий сервер або його інфраструктуру надмірним атакуючим трафіком. Це призводить до того, що сервер не реагує або працює повільно, що в кінцевому підсумку призводить до відмови в обслуговуванні для законних користувачів.

Пошкодження пам'яті. Пошкодження пам'яті відбувається, коли в певну ділянку пам'яті вносяться ненавмисні зміни, що призводить до непередбачуваної поведінки програмного забезпечення. Зловмисники намагаються використати пошкодження пам'яті за допомогою таких методів, як ін'єкції коду або атаки на переповнення буфера.

Переповнення буфера. Переповнення буфера - це аномалія, яка виникає, коли програма записує дані у визначену область пам'яті, відому як буфер.

Коли ємність буфера перевищується, сусідні комірки пам’яті перезаписуються даними. Це може бути використано для впровадження шкідливого коду в пам’ять, що потенційно створює вразливість у цільовій системі.

CSRF (Cross-Site Request Forgery - Підробка міжсайтових запитів). Підробка міжсайтових запитів полягає в тому, що жертву обманом змушують зробити запит, який використовує її автентифікацію або авторизацію. Використовуючи привілеї облікового запису користувача, зловмисник може надіслати запит, маскуючись під користувача. Після того, як обліковий запис користувача скомпрометований, зловмисник може витягти, маніпулювати або знищити важливу інформацію. Зазвичай об’єктом атаки стають облікові записи з високими привілеями, наприклад, адміністраторів або керівників.

Підстановка облікових даних. Зловмисники можуть використовувати ботів для швидкого введення великої кількості викрадених комбінацій імен користувачів та паролів на порталі входу до веб-додатку. У разі успіху зловмисник отримує доступ до облікового запису законного користувача і може викрасти його дані або здійснити шахрайські транзакції від його імені.

Скрейпінг сторінок. Зловмисники можуть використовувати ботів для незаконного вилучення контенту з веб-сторінок у великих масштабах. Цей викрадений контент може бути використаний для отримання конкурентної цінової переваги, зловмисної імітації або в інших зловмисних цілях.

Зловживання API. Інтерфейси прикладного програмування (API) полегшують комунікацію між двома програмами. Як і будь-яке програмне забезпечення, API можуть мати вразливості, які дозволяють зловмисникам впроваджувати шкідливий код в одну з програм або перехоплювати конфіденційні дані під час передачі. Зловживання API стає все більш поширеним явищем, оскільки використання API продовжує зростати. Список OWASP API Top Ten надає стислий огляд ключових ризиків безпеки API, з якими стикаються організації сьогодні.

Тіньові API. Команди розробників часто створюють і публікують API, не повідомляючи про це служби безпеки, що призводить до появи невідомих API,

які можуть розкривати конфіденційні дані компанії. Ці "тіньові" API працюють непомітно, оскільки команди безпеки, відповідальні за захист API, не знають про їхнє існування.

Зловживання стороннім кодом. Багато сучасних веб-додатків включають різні сторонні інструменти, наприклад, інструмент обробки платежів для сайту електронної комерції. Якщо зловмисники виявляють вразливість в одному з цих інструментів, вони можуть скомпрометувати його, викрасти оброблені дані, порушити його функціональність або впровадити шкідливий код в інші частини програми. Прикладом такого типу атак є Magecart-атаки, які полягають у зчитуванні даних кредитних карток у платіжних процесорів. Ці атаки також відомі як атаки на ланцюжок постачання браузера.

1.3.2. Найкращі практики з безпеки веб додатків

Веб-розробники мають можливість проектувати і створювати додатки таким чином, щоб запобігти несанкціонованому доступу до приватних даних, шахрайським діям та іншим зловмисним діям. Для усунення найпоширеніших ризиків безпеки додатків, розробники повинні застосовувати такі найкращі практики:

Впровадження валідації вхідних даних. Блокуючи неправильно відформатовані дані від проходження через робочі процеси додатку, розробники можуть запобігти ін’єкційним атакам та впровадженню шкідливого коду в додаток.

Використання сучасного шифрування. Зберігання даних користувачів у зашифрованому форматі та використання HTTPS (HyperText Transfer Protocol Secure - Безпечний Протокол Передачі Гіпертексту)для безпечної передачі вхідного та вихідного трафіку допомагає захиститися від крадіжки даних зловмисниками.

Забезпечення надійної автентифікації та авторизації. Впровадження засобів контролю надійних паролів, багатофакторної автентифікації (наприклад, апаратних ключів), механізмів контролю доступу та інших

методів автентифікації та авторизації ускладнює зловмисникам шахрайський доступ до облікових записів користувачів та навігацію в додатку.

Здійснення контролю над API. Використовуючи інструменти, які виявляють невидимі "тіньові API", що можуть слугувати потенційними об’єктами атак, організації можуть підвищити безпеку API, забезпечивши всебічну видимість та управління всіма API.

Документування змін коду. Ефективне документування змін коду сприяє ефективній співпраці між командами безпеки та розробників, дозволяючи швидко виявляти та усувати нові вразливості.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА АЛГОРИТМ ФОРМУВАННЯ БАЗ ДАНИХ

2.1. CRM та стек технології

2.1.1. Призначення CRM-системи

Призначення CRM-системи – динамічне оперування контактною інформацією клієнтів і їх замовлень. Початково CRM-система повинена включати:

– панелі з відображенням мінімальної інформації про кожного з клієнтів;

– можливість створення карток клієнтів з їх контактною інформацією і інформацією про їх останнє замовлення;

– можливість видалення картки клієнта;

– можливість видалення всіх карток клієнтів;

– можливість редагування карток клієнтів.

2.1.2. Проектування CRM-системи

Для початку, CRM-система повинна мати можливість працювати на актуальних версіях всіх популярних веб-браузерів. Повністю відповідати поставленим вимогам, тобто виконувати всі вищезазначені функції. CRM-система повинна мати зрозумілий і лаконічний інтерфейс, зберігати і відображати: номер замовлення, контакти клієнта, його фотографію, статус виконання замовлення і пріоритет у виконанні замовлення.

Задачі, що мають бути виконані:

1. наявність системи керування картками, де користувачі зможуть створювати, оновлювати та переглядати картки;

2. кожна картка клієнта повинна мати наступні атрибути: назва замовлення, опис замовлення, категорія, пріоритет, статус, прогрес, контактні дані клієнта та аватар;

3. користувачі повинні мати можливість вибрати категорію з попередньо визначеного списку або ввести нову категорію;

4. поле пріоритету має дозволяти користувачам призначати рівень пріоритету для кожної картки в діапазоні від 1 (найнижчий) до 5 (найвищий);

5. поле статусу має вказувати поточний статус заявки з такими опціями, як “не розпочато”, “у процесі”, “на утриманні” та “завершено”;

6. у полі прогресу має відображатися індикатор прогресу, що вказує на рівень виконання замовлення зі значеннями в діапазоні від 0% до 100%;

7. система повинна дозволяти користувачам завантажувати фотографію клієнта;

8. форма для створення та оновлення карток повинна містити перевірку на стороні клієнта для обов’язкових полів і правильність формату електронної пошти і номеру телефону;

9. після надсилання форма має надсилати запит до БД для створення або оновлення картки у серверній системі;

10. після успішного надсилання користувач повинен бути перенаправлений на сторінку з усіма картками;

11. система повинна отримувати наявні дані про картки з БД для редагування та перегляду;

12. інтерфейс користувача повинен забезпечувати швидкий і зручний інтерфейс для безперебійного керування заявками.

2.2. Система управління взаємовідносинами із клієнтами

Суть управління взаємовідносинами з клієнтами полягає в розумінні клієнтів і їх потреб. CRM – це стратегія яка використовується для отримання інформації про потреби, бажання та очікування клієнтів, щоб сприяти зміцненню відносин з ними (рис. 2.1). Вона вважається ключовим фактором успіху в бізнесі, втілюючи філософію, яка ставить клієнта в центр культури, процесів і дій організації.

CRM – це інтегрована стратегія управління клієнтами фірми, спрямована на ефективне управління клієнтами шляхом надання індивідуалізованих товарів і послуг та максимізації їхньої життєвої цінності.

Також CRM характеризують як "стратегічне використання інформації, процесів, технологій і людей для управління відносинами клієнта з компанією протягом усього життєвого циклу клієнта".

Дехто визначає CRM як "корпоративний підхід до розуміння і впливу на поведінку клієнтів через змістовні комунікації з метою покращення залучення, утримання, лояльності та прибутковості клієнтів".

Рисунок 2.1 – Графічне представлення роботи CRM

Якість обслуговування клієнтів визначається і оцінюється самими клієнтами, що впливає на бажаність відносин з організацією. Зустрічі з клієнтами створюють моменти істини, і для організацій дуже важливо ефективно управляти цими зустрічами.

Мета CRM – отримати уявлення про поведінку клієнтів та їхню цінність для підвищення лояльності. Кінцевою метою є досягнення конкурентної переваги в управлінні клієнтами та підвищення прибутковості.

Підтримка відносин з клієнтами: Організаціям також необхідно надавати пріоритет існуючим клієнтам, щоб забезпечити постійні закупівлі та постійну підтримку своїх продуктів.

Підвищивши рівень утримання клієнтів лише на 5%, організації можуть збільшити прибутковість від 20% до 125%.

Прибутковість клієнтів - це фінансові результати діяльності клієнтів по відношенню до всіх витрат, пов’язаних з діловою операцією. У контексті CRM прибутковість визначається на основі довічної цінності клієнта для організації, враховуючи прибутки і витрати, пов’язані з кожним клієнтом і його конкретними транзакціями з плином часу.

2.2.1. Основні переваги CRM

Спочатку вважалося, що основні переваги управління взаємовідносинами з клієнтами (CRM) залежать від типу організації, оскільки методи і технології CRM пристосовані до конкретних організаційних процесів. Однак зараз визнано, що бажані переваги CRM в основному однакові для всіх організацій і країн. Комплексний підхід до CRM може забезпечити численні переваги практично для будь-якого бізнесу, зокрема

Для організації: Впровадження CRM-системи, яка централізує всю необхідну інформацію в одній доступній базі даних, зменшує ймовірність помилок при взаємодії з клієнтами. Маючи точні адреси та контактну інформацію, а також можливість керувати договорами на обслуговування в CRM-системі, продавці можуть зосередитися на наданні відмінного обслуговування клієнтів, що призведе до підвищення прибутковості.

CRM-системи можуть впорядкувати й автоматизувати різні функції в організації, зокрема й комунікацію. За допомогою автовідповідачів і запрограмованих функцій CRM-системи можуть надсилати клієнтам інформацію, інформаційні бюлетені, оновлення продуктів, нагадування і релевантні новини відповідно до їхніх запитів або уподобань. Цей автоматизований процес скорочує час, необхідний для інформування клієнтів про організацію. Крім того, консолідуючи дані в одному місці, продавці можуть ефективно управляти цінами, кошторисами та розуміти купівельні моделі клієнтів.

Переваги для клієнтів включають підвищення ефективності та результативності обслуговування клієнтів. Замість того, щоб чекати на конкретну людину, будь-який співробітник, який має доступ до CRM-системи, може надати допомогу, покращуючи час реагування та сприяючи лояльності клієнтів. Клієнти відчувають, що організація розуміє їхні потреби, бажання та очікування, створюючи більш тісні відносини.

Співробітники також отримують вигоду від CRM-систем. Надаючи працівникам можливість професійно допомагати клієнтам, CRM підвищує продуктивність і задоволеність роботою, зменшуючи джерела незадоволеності. Взаємодія між працівниками і клієнтами та її результати, зафіксовані в CRM-системі, дозволяють цілеспрямовано працювати над удосконаленням і забезпечують раннє попередження про потенційні проблеми. Це дає змогу працівникам і керівникам інвестувати час у ті сфери, які потребують покращення, перш ніж проблеми стануть глибоко вкоріненими.

CRM-системи пропонують продавцям кращий і зручніший доступ до важливої інформації. Маркетинговий персонал може більш ефективно керувати та відстежувати кампанії та рекламні заходи. Працівникам відділу обслуговування клієнтів CRM надає прямий доступ до інформації про клієнтів та їхніх контактів, що дозволяє їм вирішувати потенційні проблеми на випередження, до того, як вони загостряться.

2.2.2. Приклади CRM-систем

У контексті комерційної діяльності системи управління взаємовідносинами з клієнтами (CRM) відіграють вирішальну роль в управлінні взаємодією з клієнтами, вдосконаленні процесів продажу та підвищенні загальної ефективності бізнесу. Існує безліч прикладів CRM-систем, які набули популярності в різних галузях.

Одним із яскравих прикладів є *Salesforce CRM*, хмарна CRM-платформа, яка пропонує комплексний набір інструментів і функцій для оптимізації процесів продажу, маркетингу та обслуговування клієнтів. Інтерфейс цієї

CRM можна побачити на рис 2.2. Salesforce надає централізовану базу даних для управління інформацією про клієнтів, що дозволяє компаніям відстежувати потенційних клієнтів, управляти можливостями та автоматизувати робочі процеси продажів.

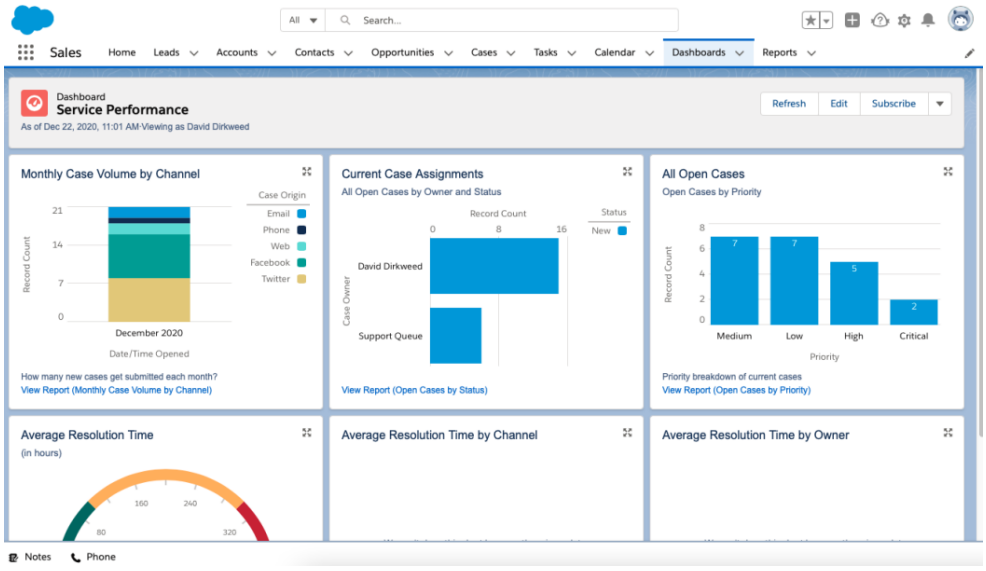

Рисунок 2.2 – Інтерфейс Salesforce CRM

Ще однією широко використовуваною CRM-системою є *Microsoft Dynamics 365*. Побудована на базі екосистеми Microsoft, Dynamics 365 поєднує в собі функції CRM та ERP (планування ресурсів підприємства), забезпечуючи цілісне рішення для управління бізнесом. Інтерфейс цієї CRM можна побачити на рис 2.3. Вона дозволяє компаніям управляти відносинами з клієнтами, оптимізувати операції та стимулювати продажі й маркетингові зусилля.

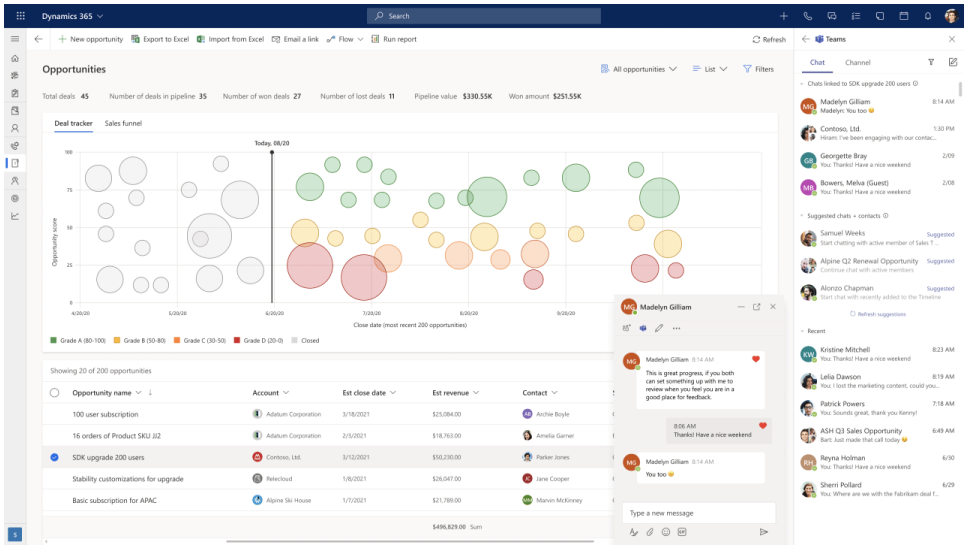

Рисунок 2.3 – Інтерфейс Microsoft Dynamics 365

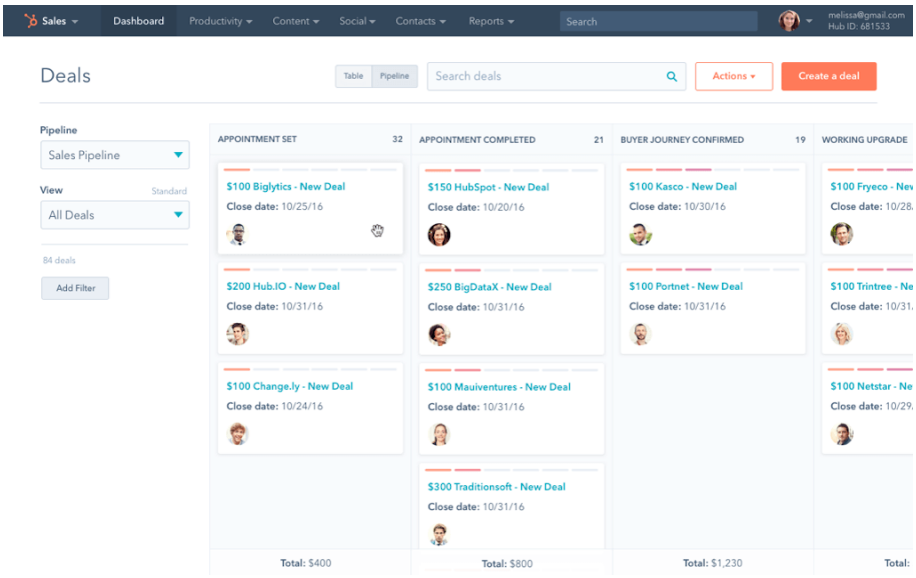

Рисунок 2.3 – Інтерфейс HubSpot CRM

HubSpot CRM - популярний вибір серед малого та середнього бізнесу. Це безкоштовна CRM-платформа з функціями для управління контактами, відстеження потенційних клієнтів та електронного маркетингу. Інтерфейс цієї CRM можна побачити на рис 2.4. HubSpot CRM інтегрується з іншими інструментами HubSpot, такими як автоматизація маркетингу та підтримка клієнтів, що дозволяє компаніям управляти всім життєвим циклом клієнта в рамках єдиної платформи.

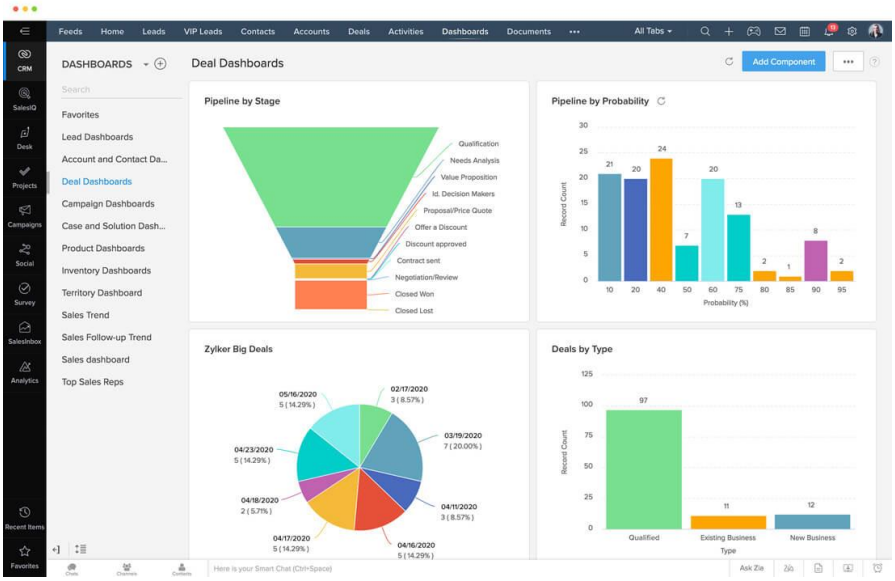

Рисунок 2.4 – Інтерфейс Zoho CRM

Zoho CRM - ще один вартий уваги приклад, що пропонує низку CRM-рішень, пристосованих для бізнесу будь-якого розміру. Інтерфейс цієї CRM можна побачити на рис 2.4. Вона надає функції для управління потенційними клієнтами, управління контактами, прогнозування продажів та інтеграції з електронною поштою. Zoho CRM також пропонує широкі можливості кастомізації, що дозволяє компаніям адаптувати систему до своїх унікальних процесів і вимог.

2.3. Стек інструментів для реалізації

Реалізація CRM-системи передбачає використання стеку різноманітних інструментів і технологій веб-розробки. Ці інструменти були ретельно підібрані, щоб забезпечити ефективну розробку, зручний користувацький досвід та надійну функціональність. Проект був побудований з використанням HTML, CSS, JavaScript, React, Node.js і бази даних ASTRASTAX, кожен з них відіграє важливу роль у різних аспектах програми.

2.3.1. HyperText Markup Language

HTML, скорочено від HyperText Markup Language (мова розмітки гіпертексту), є широко визнаною мовою розмітки, яка використовується для створення веб-сторінок. Її основне призначення - структурувати та

організовувати розділи, абзаци та посилання за допомогою елементів HTML, які включають теги та атрибути.

Застосування HTML різноманітне і охоплює різні сфери:

1. веб-розробка: HTML має вирішальне значення для розробників, оскільки дозволяє їм визначати зовнішній вигляд і розташування елементів веб-сторінки, таких як текст, гіперпосилання і медіафайли, забезпечуючи належне відображення у веб-браузерах;

2. навігація в Інтернеті: завдяки вбудованій підтримці HTML гіперпосилань користувачі можуть легко переміщатися між веб-сторінками і встановлювати зв’язки між пов’язаним контентом, забезпечуючи безперешкодний перегляд сторінок;

3. веб-документація: HTML слугує цінним інструментом для організації та форматування документів в Інтернеті, подібно до функціональності, що надається текстовими редакторами, такими як Microsoft Word;

Важливо зазначити, що HTML не вважається мовою програмування, оскільки їй бракує можливості створювати динамічну функціональність. Натомість вона визнана офіційним веб-стандартом. Консорціум Всесвітньої павутини (W3C) здійснює нагляд за підтримкою, розвитком і регулярним оновленням специфікацій HTML.

2.3.2. Cascading Style Sheets

CSS, що розшифровується як Cascading Style Sheets (каскадні таблиці стилів), – це мова таблиць стилів, яка відіграє життєво важливу роль у визначенні дизайну та презентації документів. У контексті веб-розробки CSS в першу чергу використовується для покращення візуальної привабливості та естетичних аспектів веб-сторінок, написаних мовами розмітки, такими як HTML або SVG.

CSS слугує для того, щоб зробити веб-сторінки візуально привабливими та зручними для користувача. Вона дозволяє розробникам застосовувати різні елементи дизайну до веб-контенту, включаючи розташування зображень, вибір кольорів шрифтів, налаштування стилів фону і навіть регулювання

інтервалів між абзацами. Ці покращення дизайну досягаються завдяки використанню тегів CSS.

Використовуючи CSS, веб-розробники можуть відокремити рівень презентації від рівня контенту, що дозволяє зробити веб-дизайн більш ефективним і гнучким. CSS забезпечує систематичний і організований підхід до стилізації веб-документів, забезпечуючи послідовний і візуально приємний досвід для користувачів на різних платформах і пристроях.

2.3.3. JavaScript

JavaScript - це мова програмування, яка використовується розробниками для створення інтерактивних веб-сторінок. Вона покращує користувацький досвід, надаючи різні функціональні можливості, такі як оновлення стрічок соціальних мереж, відображення анімації та реалізація інтерактивних мап. Як мова сценаріїв на стороні клієнта, вона є фундаментальною технологією Всесвітньої павутини.

JavaScript виник як засіб зробити веб-сторінки більш динамічними та інтерактивними. Раніше веб-сторінки були статичними і не мали тієї динамічної функціональності, яку ми очікуємо сьогодні. JavaScript дозволив браузерам реагувати на дії користувача і змінювати вміст веб-сторінок на льоту. З часом JavaScript вийшов за межі веб-браузерів, і розробники почали використовувати його для розробки як на стороні клієнта, так і на стороні сервера. Він став універсальною мовою, що застосовується в різних сферах.

Деякі з найпоширеніших випадків використання JavaScript.

Написання сценаріїв на стороні клієнта: JavaScript забезпечує написання сценаріїв на стороні клієнта, дозволяючи браузерам виконувати код на пристрої користувача і динамічно змінювати вміст веб-сторінок.

Розробка на стороні сервера: З появою таких фреймворків, як Node.js, JavaScript тепер можна використовувати для розробки на стороні сервера, керуючи серверною логікою та операціями з базами даних.

Розробка мобільних додатків: Фреймворки JavaScript, такі як React Native та Ionic, полегшують розробку крос-платформних мобільних додатків.

Розробка ігор: Ігрові рушії JavaScript, такі як Phaser та Three.js, дають можливість розробникам створювати браузерні ігри та інтерактивні програми.

Код JavaScript інтерпретується JavaScript- рушієм, який перекладає код на машинну мову, яку може виконати операційна система. На відміну від компільованих мов, JavaScript відноситься до категорії скриптових мов і не потребує окремого етапу компіляції. Рушій JavaScript, присутній у веб-браузерах та інших середовищах, і динамічно перетворює код JavaScript у виконуваний машинний код. Сучасні рушії використовують такі методи, як компіляція "точно вчасно" або компіляція під час виконання для оптимізації продуктивності.

2.3.4. React

React.js - це фреймворк і бібліотека JavaScript з відкритим вихідним кодом, розроблена компанією Facebook. Вона спрощує процес створення інтерактивних користувацьких інтерфейсів та веб-додатків, зменшуючи кількість необхідного коду порівняно з традиційною ванільною розробкою на JavaScript.

За допомогою React додатки створюються з використанням багаторазових компонентів, які діють як незалежні будівельні блоки. Ці компоненти представляють окремі елементи кінцевого інтерфейсу і можуть бути зібрані для створення всього користувацького інтерфейсу програми.

У додатку React в першу чергу фокусується на управлінні шаром представлення, подібно до літери "V" у патерні "модель-вид-контролер" (MVC). Він досягає успіху в ефективному рендерингу користувацького інтерфейсу, використовуючи віртуальну DOM, техніку, яка дозволяє оптимізовано оновлювати реальну DOM. Розбиваючи складні користувацькі інтерфейси на менші багаторазові компоненти, React сприяє модульному підходу до розробки. Це не тільки використовує швидкість та ефективність JavaScript, але й дозволяє швидше рендерити веб-сторінки та створювати високодинамічні та адаптивні веб-додатки.

2.3.5. Node.js

Node.js – це кросплатформне середовище виконання JavaScript з відкритим вихідним кодом". - Документація Nodejs.dev.

Проте, для загального розуміння, розберемо це докладніше.

Node.js - це платформа з відкритим вихідним кодом, це означає, що її вихідний код є загальнодоступним. Він підтримується і вдосконалюється спільнотою розробників з усього світу. Тобто, будь-хто зацікавлений, може зробити свій внесок у розвиток Node.js, дотримуючись інструкцій, наведених у посібнику з розробки Node.js.

Node.js розроблений для роботи на різних операційних системах, таких як Linux, macOS та Windows. Він не обмежується програмним забезпеченням якоїсь конкретної операційної системи, що робить його дуже універсальним і гнучким.

Коли розробник пише код JavaScript у текстовому редакторі, цей код сам по собі не може виконувати жодних завдань, доки його не буде виконано або запущено. Для запуску коду JavaScript потрібне середовище виконання. Зазвичай веб-браузери, такі як Chrome та Firefox, мають вбудоване середовище виконання, яке дозволяє їм виконувати JavaScript-код. Однак Node.js надає середовище виконання поза браузером. Він використовує JavaScript-рушій Chrome V8, що дозволяє вам використовувати JavaScript для створення не тільки інтерфейсних, але й бекенд-додатків.

2.3.6. Datastax Astra Data Base.

БД Astra DB, що розроблена DataStax, - це глобально розподілена, безсерверна і багатомодельна служба БД. Вона виділяється як перша і єдина безсерверна, мультирегіональна служба БД, заснована на відкритій базі даних NoSQL, Apache Cassandra.

Astra DB спрощує процес впровадження та управління базою даних Apache Cassandra, усуваючи пов'язані з нею проблеми та складнощі.

Однією з основних особливостей AstraDB є її хмарна архітектура, що означає, що вона спеціально створена для розгортання та роботи в хмарних

середовищах. Ця архітектура використовує переваги хмарних обчислень, такі як еластичність, масштабованість і легка інтеграція з іншими хмарними сервісами. Використовуючи AstraDB, організації можуть більше зосередитися на розробці своїх додатків і використанні можливостей бази даних, а не на управлінні базовою інфраструктурою і конфігурацією кластерів Cassandra. AstraDB доступна на основних хмарних платформах, включаючи Amazon Web Services (AWS), Microsoft Azure і Google Cloud. Це також забезпечує перевагу географічної близькості, дозволяючи зберігати дані ближче до користувачів і зменшуючи мережеві затримки.

2.4. Алгоритм формування БД

Початком для розробки CRM-системи, стало створення БД і отримання токену та посилання для керування нею. Після вдалої реєстрації на сайті astra.datastax.com і освоєння інтерфейсу, почалось формування БД. Форма створення наведена на рис. 2.5.

1. створення БД: Після входу в систему було розпочато створення нової БД, натиснувши кнопку "Створити базу даних". Було вказано: унікальне ім'я БД, вибрано хмарного провайдера Google Cloud та обрано Германію, як регіон, де буде розміщена база даних.

2. отримання токену БД: У розділі налаштувань БД було отримано токен БД. Який буде використовуватися для під'єднання до БД із веб-застосунку та задля налаштування Swagger UI.

3. вибір API БД: DataStax AstraDB пропонує кілька варіантів API, таких як REST, GraphQL і Document. Було обрано Document API задля зберігання і редагування JSON документів.

4. налаштування Swagger UI: за допомогою візуальної частини Swagger UI, було отримано посилання і налаштовані запити для взаємодії з даними БД у реальному часі. Запити, які були налаштовані:

– отримання даних з БД шляхом виконання GET-запитів.

– додавання нових даних до БД за допомогою POST-запитів.

– оновлення існуючих даних в БД за допомогою PUT запитів

– видалення даних з БД за допомогою запитів DELETE.

5. зберігання отриманих токенів і посилання з Swagger у окремому файлі для подальшої взаємодії.

6. взаємодія з БД на серверній частині проекту.

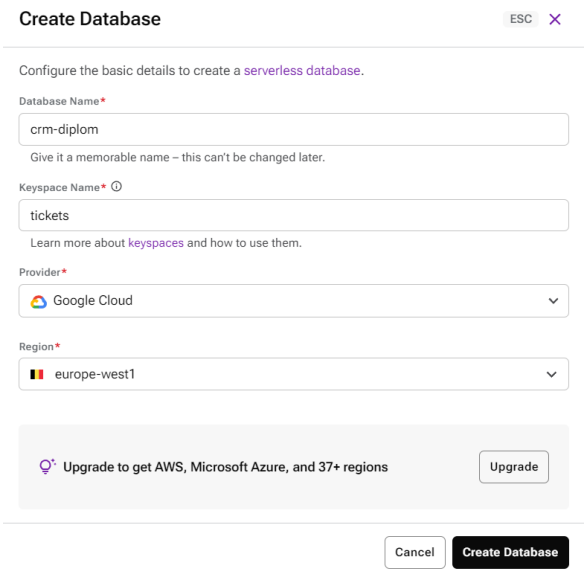

Рисунок 2.5 – Форма створення БД

Swagger UI дозволяє будь-кому - чи то команді розробників, чи кінцевим споживачам - візуалізувати та взаємодіяти з ресурсами API без необхідності мати будь-яку логіку реалізації. Він автоматично генерується з специфікації OpenAPI (раніше відомої як Swagger), а візуальна документація спрощує реалізацію та використання на стороні клієнта.

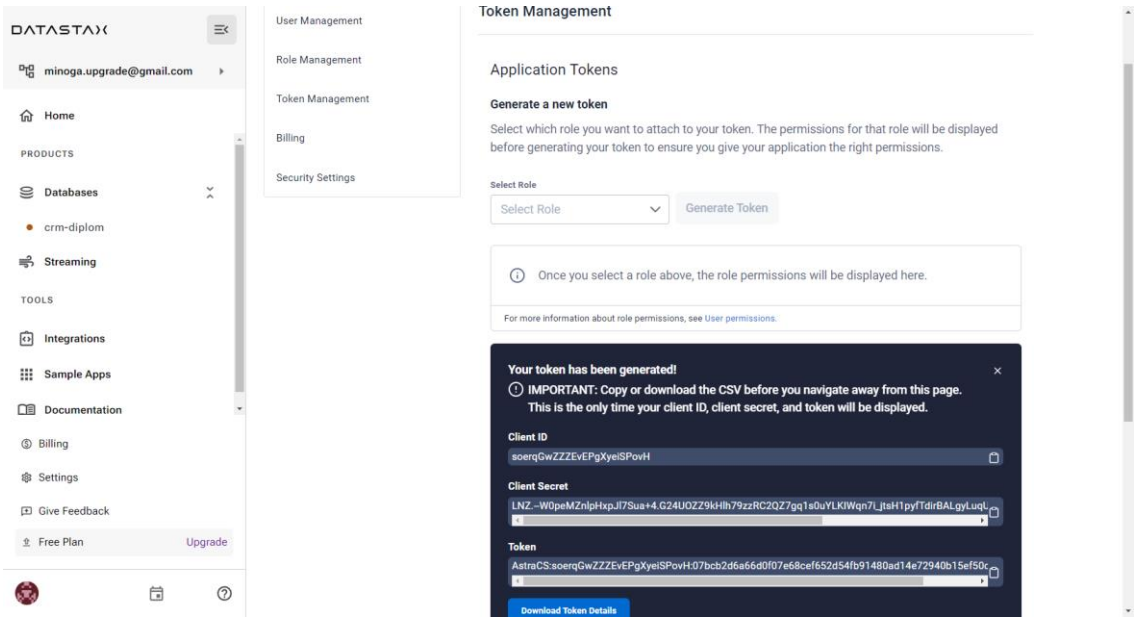

Рисунок 2.6 – Інтерфейс AstraDB з отриманим токеном

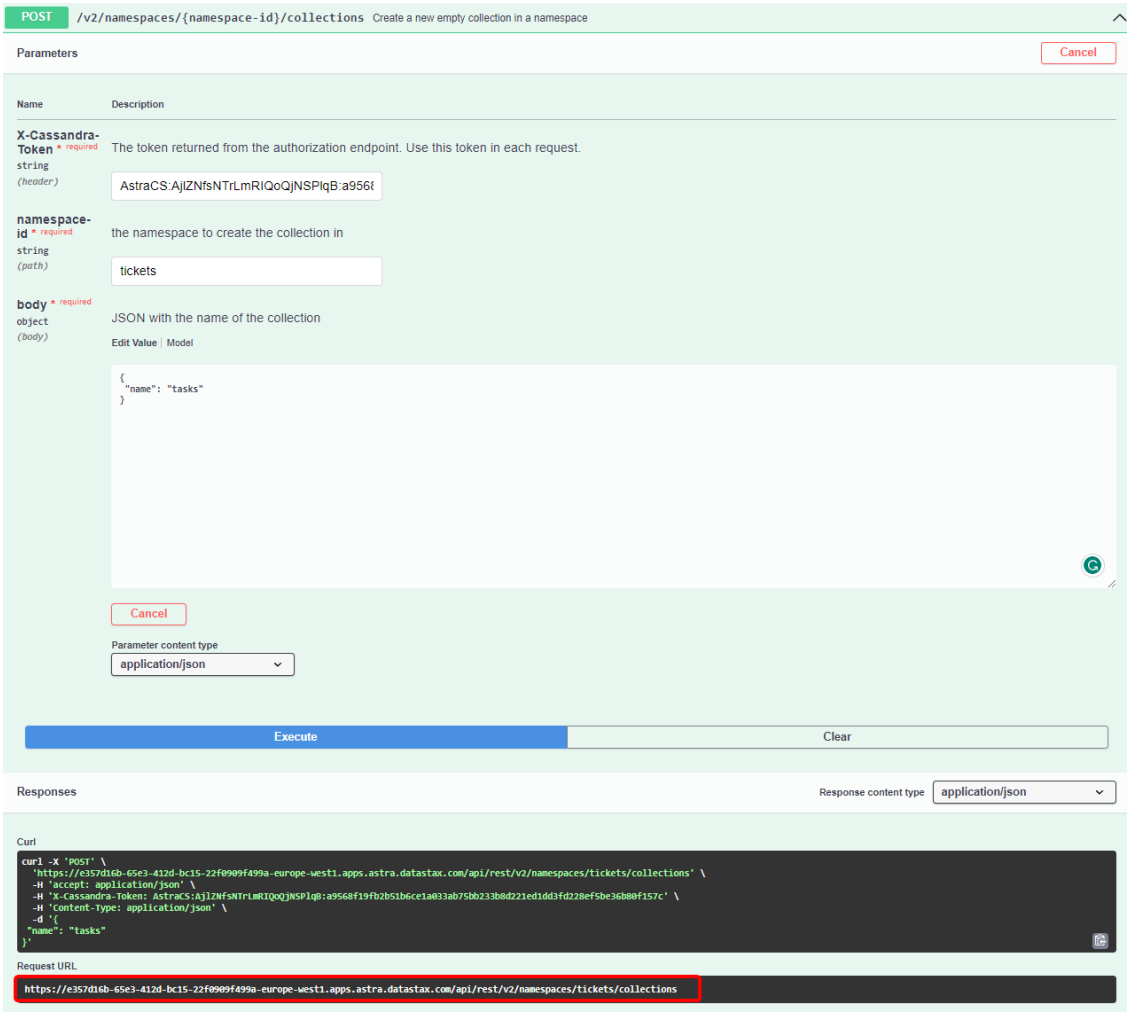

Рисунок 2.7 – Інтерфейс Swagger UI з отриманням посилання

РОЗДІЛ 3. ПРОГРАМНА ЧАСТИНА

3.1. Реалізація візуальної і функціональної частини проекту

Для програмної реалізації було створено React проект у середовищі розробки WebStorm від компанії JetBrains. Лістинг коду наведено у ДОДАТКУ А. Організація структури проекту представлена на рис.3.1.

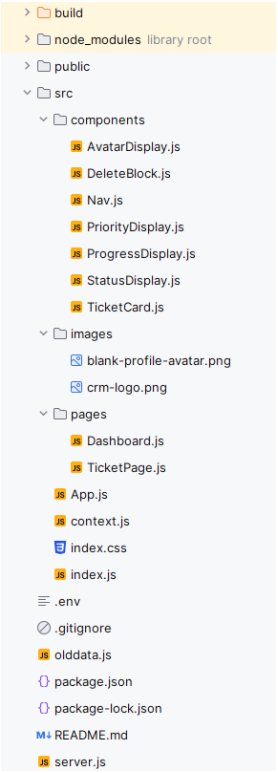

Рисунок 3.1 – Файлова структура CRM-системи

Для розробки маршрутизації було використано модуль App.js, який слугує основною точкою входу в додаток.

3.1.1. Розробка маршрутизації

У React маршрутизація передбачає перенаправлення користувачів на різні сторінки залежно від їхніх дій або запитів. Для роботи з маршрутизацією в React було використано сторонню бібліотеку під назвою React Router Dom.

До модуля App імпортуються необхідні залежності, такі як BrowserRouter, Route, Routes і useState з бібліотек react-router-dom і react. До нього також імпортуються: компонент навігаційної панелі Nav, сторінка з картками клієнтів Dashboard, сторінка для створення і редагування карток

TicketPage і компонент CategoriesContext з модуля context.js для керування категоріями на головній сторінці.

Елемент BrowserRouter обгортає весь додаток і забезпечує функціональність маршрутизації.

Компонент Nav рендериться всередині елемента BrowserRouter.

Елемент Routes використовується для визначення конфігурації маршрутизації. Всередині нього є три елементи Route:

– перший відповідає кореневому URL (“/”), тобто за головну сторінку і рендерить її;

– другий відповідає URL “/ticket” і рендерить сторінку для створення карток;

– третій відповідає URL “/ticket/:id” і рендерить сторінку для редагування вибраної картки.

Таким чином, модуль App налаштовує маршрутизацію і контекст для додатку і рендерить необхідні компоненти на основі поточного маршруту.

3.1.2. Створення компонентів головної сторінки

Для виводу картки клієнта були розроблені компоненти, що відтворюють деякі секції для карток клієнтів на головній сторінці проекту. Їх кінцевий вигляд можна побачити на рис 3.2, де кожному номеру на рисунку відповідає номер списку нижче:

1. *компонент AvatarDisplay* відповідає за відображення фотографії клієнта на головній сторінці;

2. *компонент StatusDisplay* відповідає за відтворення поточного статусу замовлення із вже готових варіантів і також надає відповідний колір фону для статусу;

3. *компонент PriorityDisplay* візуально представляє рівень пріоритету у виконанні замовлення за допомогою зірочок, що дозволяє користувачам швидко;

4. *компонент ProgressDisplay* відповідає за смугу прогресу виконання замовлення, надаючи візуальну інформацію про відсотковий рівень прогресу;

5. *компонент DeleteBlock* відповідає за кнопку, яка запускає запит на видалення конкретної картки клієнта з бази даних. Після успішного видалення сторінка перезавантажується.

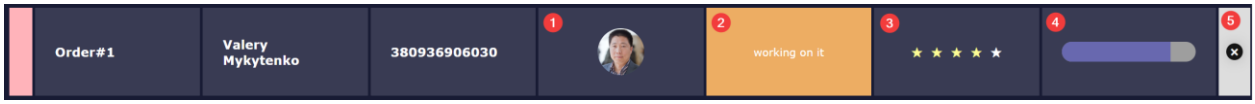

Рисунок 3.2 – Картка клієнта

Контактна інформація клієнта і найменування замовлення виводяться як звичайний текст.

Бічна панель навігації, де знаходиться: логотип, кнопка створення і кнопка видалення усіх карток – розташована на головній сторінці рис 3.3.а і створена у *модулі Nav.js.*

Про модуль *TicketCard.js*, що відповідає за *повноцінну картку клієнта*, потрібно розповісти докладніше бо він виконує функцію об'єднання всіх компонентів наведених вище рис 3.3.б, окрім бічної панелі, що були створенні для відображення картки певного клієнта на головній сторінці.

На початку модуля було імпортовано елемент Link з бібліотеки react-router-dom та компоненти з інформацією про клієнта і замовлення, що будуть відображатись на сторінці з усіма картками. Елемент Link був використаний, щоб створити посилання, яке буде спрямовувати користувачів до детального перегляду картки з можливістю редагування, тобто до модуля сторінки TicketPage.js, про який буде окрема частина розділу. Пункт призначення посилання, для подальшого редагування, визначається унікальним номером картки - documentId.

У межах посилання було відображену відповідну інформацію про картку, таку як пріоритет, прогрес, статус замовлення, номер замовлення, ім'я клієнта, номер телефону і фото клієнта.

Кожен з компонентів, отримують вхідні дані з бази даних за допомогою мітки на певного клієнта.

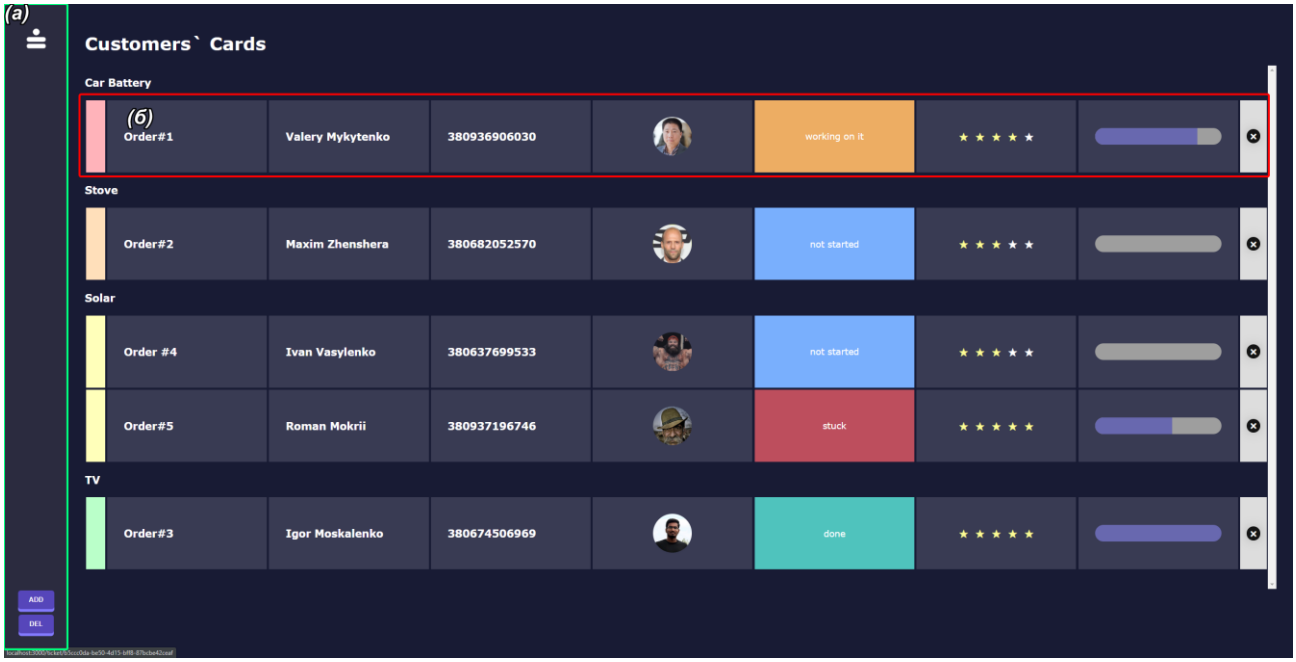

Рисунок 3.3 – Головна сторінка: а - бічна панель; б - картка клієнта

3.1.3. Реалізація головної сторінки

Головна сторінка рис 3.3, на якій знаходяться усі картки клієнтів, розташована у модулі Dashboard.js. До якого імпортуються компоненти TicketCard, CategoriesContext з модуля context.js і бібліотека Axios.

Коротко про компонент CategoriesContext: він був підключений для передавачі й оновлення інформації, пов’язаної з категоріями, у всьому дереві компонентів.

Всередині модуля сторінки є дві частини стану, визначені за допомогою хука useState. Стан “tickets” містить дані для карток клієнтів, а стан “categories” отримується з CategoriesContext за допомогою хука useContext.

Хуки React - це прості функції JavaScript, які можна використовувати, щоб ізолювати повторно використовувану частину від функціонального компонента. Хуки можуть мати стан і керувати побічними ефектами.

У модуля сторінки використовується хук useEffect для отримання даних про картки з сервера, коли компонент монтується. Він робить HTTP GET-запит до серверу, для отримання даних, використовуючи бібліотеку Axios. Дані відповіді обробляються для вилучення необхідної інформації та

форматування її в масив об’єктів карток. Відформатовані дані карток зберігаються у стані “tickets” за допомогою функції setTickets.

Інший хук useEffect використовується для оновлення стану “categories” щоразу, коли змінюється стан “tickets”. Він витягує унікальні категорії з даних карток і оновлює стан категорій за допомогою функції setCategories.

Усередині тегу div з класом ticket-container відображаються унікальні категорії і для кожної з них виводиться окремий розділ. Для кожної категорії у ньому фільтруються картки на основі категорії і відображаються відфільтровані картки, щоб створити компонент TicketCard для кожної картки. Компонент TicketCard отримує набір даних, такі як дані картки, унікальний ідентифікатор і колір на основі індексу категорії.

Загалом, сторінка отримує дані про картки з сервера, впорядковує картки за категоріями і відображає їх у вигляді карток клієнтів.

3.1.4. Реалізація сторінки для створення і редагування карток клієнтів

Сторінка для інтерфейсу створення, вигляд якого видно на рис. 3.4, і для інтерфейсу редагування карток, вигляд якого видно на рис. 3.5, клієнтів, представляє модуль TicketPage.js. Поведінка сторінки залежить від переданого їй реквізиту editMode.

Усередині сторінки було отримано доступ до функцій categories і setCategories з модуля context.js за допомогою хука useContext. Це дозволяє отримувати та оновлювати дані категорій з контексту.

Сторінка ініціалізує стан “formData” за допомогою хука useState і містить такі властивості, як статус, прогрес, категорія, необхідні для створення або редагування картки. Початкові значення встановлюються на основі наявних даних або значень за замовчуванням.

Для управління навігацією в додатку було використано хук useNavigate з бібліотеки react-router-dom. Він дозволяє мені програмно переходити за різними маршрутами, які були прописані у компоненті App.

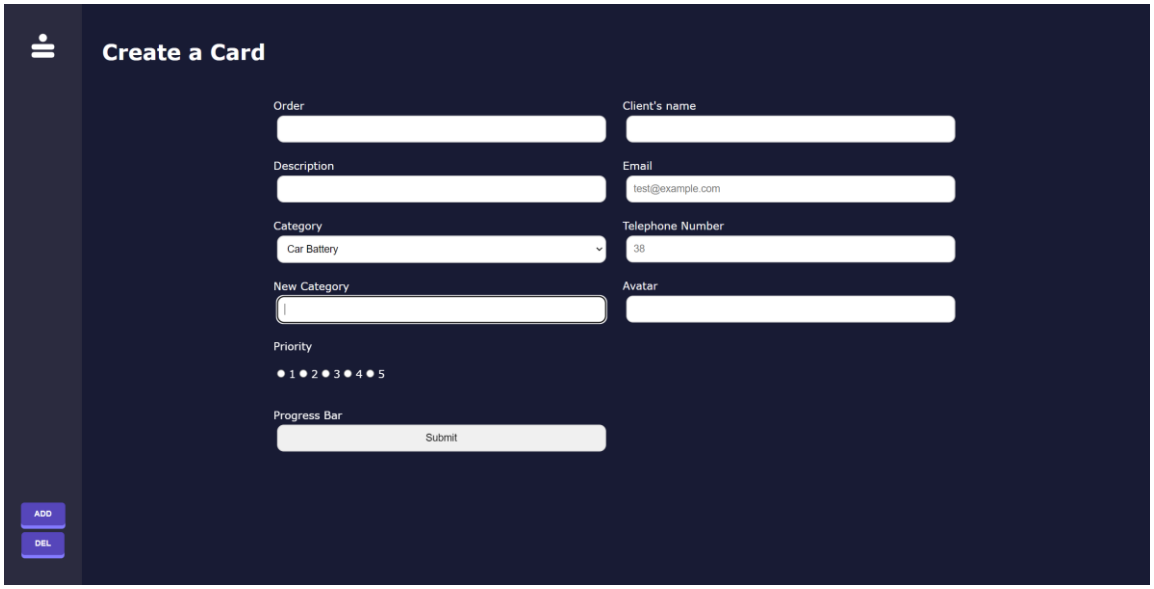

Рисунок 3.4 – Сторінка для створення карток клієнта

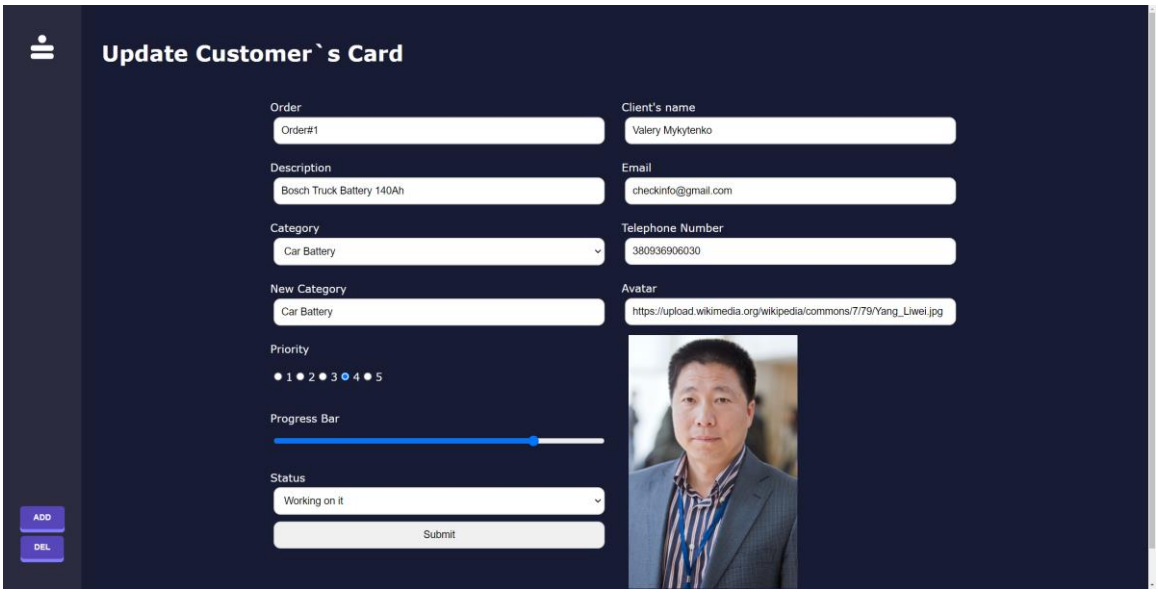

Рисунок 3.5 – Сторінка для редагування карток клієнта

Якщо режим редагування увімкнено, то компонент отримує наявні дані про картки з сервера, використовуючи бібліотеку axios та наданий параметр унікального номера картки з URL-адреси. Отримані дані потім використовуються для заповнення стану “formData”.

Компонент містить форму з полями для введення таких даних, як замовлення, опис, категорія, пріоритет, прогрес, статус, ім’я клієнта, електронна пошта, номер телефону та аватар. Поля форми є керованими

компонентами, які оновлюють стан “formData” після введення даних користувачем.

Після надсилання форми компонент робить HTTP-запит на створення нової картки або оновлення існуючої на основі прапорця editMode. Відповідна кінцева точка API викликається за допомогою axios, і після успішної відповіді користувач повертається назад на головну сторінку.

Також, було реалізовано додаткові функції, такі як відображення існуючих категорій, попередній перегляд фотографії і перевірка полів введення за допомогою HTML-атрибутів.

Загалом, компонент TicketPage дозволяє користувачам створювати або редагувати квитки, заповнюючи форму і надсилаючи дані. Він взаємодіє з сервером за допомогою HTTP-запитів, використовує CategoriesContext для роботи з категоріями і забезпечує зручний користувацький інтерфейс для керування картками клієнтів.

3.2. Реалізація серверної частини проекту

Модуль server.js містить код для сервера Node.js з використанням фреймворку Express, для виконання CRUD-операцій над колекцією карток.

Сервер прослуховує порт 8000 для вхідних HTTP-запитів. Коли запит отримано, сервер використовує проміжне програмне забезпечення Express для обробки CORS (Cross-Origin Resource Sharing - Спільне Використання Ресурсів з Різних Джерел) і розбору JSON-даних, вигляд структури даних JSON-документу можна побачити на рис 3.6.


```
{
  "data": {
    "b5ccc0da-be50-4d15-bff8-87bcbe42ceaf": {
      "avatar": "https://upload.wikimedia.org/wikipedia/commons/7/79/Yang_Liwei.jpg",
      "category": "Car Battery",
      "description": "Bosch Truck Battery 140Ah",
      "email": "checkinfo@gmail.com",
      "owner": "Valery Mykytenko",
      "priority": "4",
      "progress": "81",
      "status": "working on it",
      "telNo": "380936906030",
      "timestamp": "2023-05-12T09:04:58.345Z",
      "title": "Order#1"
    },
```


Рисунок 3.6 – Структура даних JSON-документу

Сервер визначає кілька маршрутів для обробки різних типів запитів. Ці маршрути включають:

- маршрут GET “/tickets”. Цей маршрут отримує список карток із зовнішнього API шляхом надсилання HTTP GET-запиту. Сервер надсилає запит до кінцевої точки API і повертає дані у відповідь;

- маршрут POST “/tickets”. Цей маршрут створює новий карток, надсилаючи HTTP POST запит до зовнішнього API. Сервер витягує дані форми із запиту, готує параметри запиту, відправляє запит і повертає дані відповіді;

- маршрут GET “/tickets/:documentId”. Цей маршрут отримує конкретну картку за його ідентифікатором. Сервер витягує ідентифікатор з параметрів запиту, надсилає HTTP GET-запит до кінцевої точки API з ідентифікатором і повертає дані відповіді;

- маршрут PUT “/tickets/:documentId”. Цей маршрут оновлює конкретну картку за ідентифікатором документа. Сервер витягує ідентифікатор і дані з параметрів і тіла запиту, надсилає HTTP PUT-запит до кінцевої точки API з оновленими даними і повертає дані у відповідь;

- маршрут DELETE “/tickets/:documentId”. Цей маршрут видаляє конкретну картку за його ідентифікатором документа. Сервер витягує ідентифікатор з параметрів запиту, надсилає HTTP-запит DELETE до кінцевої точки API з ідентифікатором і повертає дані у відповідь.

Для створення HTTP-запитів до зовнішнього API сервер використовує бібліотеку Axios. Вона включає необхідні заголовки, такі як токен автентифікації, і обробляє потенційні помилки.

Загалом, модуль server.js налаштовує сервер, який обробляє CRUD-операції для колекції карток. Він зв’язується із зовнішнім API бази даних AstraDB для виконання цих операцій і повертає відповідні відповіді.

ВИСНОВКИ

У даній роботі було досліджено розробку веб-додатків для комерційної діяльності і розроблено CRM-систему з використанням HTML, CSS, JavaScript, React та інших технологій. У ході дослідження було вивчено важливість веб-додатків у сучасному бізнесі та переваги, які вони пропонують, такі як покращення взаємодії з клієнтами, оптимізація процесів та підвищення ефективності.

Впровадивши запропоновану CRM-систему, було продемонстровано практичне застосування технологічного стеку, зокрема компонентної структури React та робота з хмарною БД AstraDB. Функціональність, масштабованість та дизайн користувацького інтерфейсу системи були ретельно розроблені з урахуванням вимог.

Інформація, отримана в результаті цього дослідження, стане у нагоді як розробникам, так і компаніям, які прагнуть використати можливості веб-додатків для досягнення комерційного успіху. Використовуючи стек технологій та найкращі практики, розглянуті в цій дипломній роботі, організації можуть оптимізувати управління взаємовідносинами з клієнтами, впорядкувати процеси продажів і стимулювати зростання бізнесу в цифрову епоху.

Як і в будь-якому дослідженні, під час розробки CRM-системи були певні обмеження та виклики, з якими довелось зіткнутись. Серед них – необхідність постійної адаптації до технологій, що розвиваються, забезпечення безпеки та конфіденційності даних, а також задоволення специфічних бізнес-вимог. Однак завдяки ретельному плануванню, впровадженню та оцінці ці виклики були успішно подолані, що призвело до створення CRM-системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДІЯ БІЗНЕС [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://business.diia.gov.ua/handbook/prodazi/so-take-crm-sistema-ak-obrati-j-pracuvati-z-crm-so-vazlivo-zamiruvati.

2. ClickITtech [Електронний ресурс] – Режим доступу до ресурсу: https://www.clickittech.com/devops/.

3. Трофименко О. Г. Веб-дизайн та HTML-програмування / О. Г. Трофименко. – Одеса: Фенікс, 2018. – 194 с.

4. Пасічник О. В. Веб-дизайн. / О. В. Пасічник, В. В. Пасічник. – Львів: Магнолія, 2019. – 520 с.

5. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

6. FreeCodeCamp [Електронний ресурс] – Режим доступу до ресурсу: www.freecodecamp.org.

7. React [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/learn.

8. DataStax Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.datastax.com/products/datastax-astra.

9. Node.js Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en/docs.

10. Zhang, Y., & Ali, B. (2019). A comprehensive study of web application development frameworks. Journal of Systems and Software, 158, 110-128.

11. Rahman, M. M., & Kumar, A. (2020). A comparative study of front-end web development frameworks. International Journal of Computer Applications, 975, 1-5.

12. Gaur, S., & Choudhary, A. (2018). A review of popular web development frameworks and their comparison. International Journal of Advanced Research in Computer Science, 9(4), 174-178.

13. Flanagan, D. (2018). JavaScript: The Definitive Guide (7th ed.). O'Reilly Media.

14. Freeman, A., Robson, E., & Bates, B. (2020). Head First JavaScript Programming: A Brain-Friendly Guide (2nd ed.). O'Reilly Media.

15. Martin, R. C. (2009). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

16. Gamble, J., Helm, R., Johnson, R., & Vlissides, J. (2021). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.

17. Elliott, J., & Freeman, E. (2020). Head First JavaScript: A Brain-Friendly Guide (2nd ed.). O'Reilly Media.

18. Brown, J. (2020). Fullstack React: The Complete Guide to ReactJS and Friends. Independently published.

19. Simpson, K. (2021). You Don't Know JS Yet: Get Started. O'Reilly Media.

20. Duckett, J. (2014). HTML & CSS: Design and Build Websites. John Wiley & Sons.

21. Mead, A. (2018). The Complete Node.js Developer Course (3rd ed.). Udemy.