Міністерство освіти і науки України

«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
завідувач кафедри кібербезпеки
та захисту інформації
__________ Н.В. Лукова-Чуйко
« » червня 2021р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної роботи
бакалавра
(назва освітнього рівня)

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: «Дослідження вразливостей, шляхів та рекомендацій щодо захисту CRM-систем»

Виконавець: студент IV курсу, групи КБ-41

_______________ **Скрипник Максим Вікторович**
(підпис) (прізвище ім'я по-батькові)

	Прізвище, ініціали	Підпис
Керівник	Шестак Я.В.	

Нормоконтроль	Даков С.Ю.	

Київ 2021

Міністерство освіти і науки України

«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідувач кафедри кібербезпеки
та захисту інформації
_____________ Н.В. Лукова-Чуйко
«10» жовтня 2020 р.

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студенту **КБ-41** **Скрипнику Максиму Вікторовичу**
(група) (прізвище ім'я по-батькові)

Тема дипломної роботи Дослідження вразливостей, шляхів та рекомендацій щодо захисту CRM-систем

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №2 від 08.10.2020 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Концепція систем управління відносин з клієнтами, хмарні CRM-системи

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Необхідно ознайомитися з теорією CRM - систем, їх типами та особливостями, вразливостями з боку безпеки даних, проаналізувати можливі інструменти захисту від загроз, розглянути існуючі рішення в галузі CRM – систем, розробити рекомендації при налаштуванні та роботі з CRM – системою.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Розроблені рекомендації до захисту даних у CRM – системах і обрання найбезпечнішого рішення серед існуючих систем.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання:12 жовтня 2020 року

Завдання видала		Я. В. Шестак
	(підпис)	(ініціали, прізвище)
Завдання прийняв до виконання	(підпис)	М. В. Скрипник (ініціали, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	25.01.2021 – 22.01.2020	*виконано*
2	Аналіз літератури	29.01.2020 – 11.02.2020	*виконано*
3	Обґрунтування вибору рішення	12.02.2020 – 15.02.2020	*виконано*
4	Концепція хмарних обчислень	16.02.2020 – 04.03.2020	*виконано*
5	Аналіз проблем інформаційної безпеки в хмарних технологіях	05.03.2020 – 21.03.2020	*виконано*
6	Дослідження вразливостей та загроз	22.03.2020 – 08.04.2020	*виконано*
7	Вироблення рекомендацій щодо вибору хмарної послуги і методу захисту даних, що використовуються в хмарних сервісах	09.04.2020 – 10.05.2020	*виконано*
8	Оформлення пояснювальної записки	11.05.2020 – 27.05.2020	*виконано*
9	Підготовка до захисту дипломної роботи	28.05.2020 – 08.06.2021	*виконано*

Завдання видала		Я. В. Шестак
	(підпис)	(ініціали, прізвище)
Завдання прийняв до виконання	(підпис)	М. В. Скрипник (ініціали, прізвище)

Термін подання дипломної роботи до ЕК 08 червня 2021 року

РЕФЕРАТ

Дипломна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел та додатків, має 67 сторінки основного тексту, 6 таблиць та 29 рисунків. Список використаних джерел містить 18 найменувань і займає 3 сторінки.

Метою даної роботи є аналіз захищеності сучасних CRM – систем та розробка рекомендацій щодо вибору системи, а також щодо їх налаштування та безпечного використання.

У роботі проаналізована сучасна література та ресурси з захисту CRM – систем, виконаний аналіз документів, порівняння, вивчення та узагальнення вітчизняної та зарубіжної практики з теми захисту CRM – систем, розроблено рекомендації з вибору найбезпечнішого рішення.

Розроблені налаштування і код додатку на базі системи Salesforce, можна використовувати як наочний приклад захищеного додатку на базі готового рішення, використовуючи інструменти які нам надає система.

Розроблені рекомендації призначені для користувачів, що хочуть забезпечити безпеку даних в CRM – системі, яку вони використовують.

Ключові слова: системи управління відносинами, загрози, вразливості, CRM системи, Salesforce, хмарні системи.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

CRM – Customer Relationship Management
ERP – Enterprise Resource Planning
B2C – Business To Commerce
СУБД – Система Управління Базами Даних
IT – Інформаційні Технології
SaaS – Software as a Service
PaaS – Platform as a Service
OWD – Organization Wide Defaults
XSS – Cross-Site Scripting
CSRF – Cross-Site Request Forgery
SQL – Structured Query Language
OS – Operation System
XXE – XML External Entity
LDAP – Lightweight Directory Access Protocol
API – Application Programming Interface
SOAP – Simple Object Access Protocol
XML – eXtensible Markup Language
REST – Representational State Transfer
JSON – JavaScript Object Notation
RPC – Remote Procedure Call
GWT – Google Web Toolkit
LWC – Lightning Web Components
HTML – HyperText Markup Language
DOM – Document Object Model
MIME – Multipurpose Internet Mail Extension
CRUD – Create Read Update Delete
SOQL – Salesforce Object Query Language
AJAX – Asynchronous Javascript and XML

ЗМІСТ

ВСТУП

Актуальність даної роботи визначається тим, що в сьогоднішньому світі існує безліч підприємств, діяльність яких потребує автоматизації процесів. Це стосується як автоматизації в роботі підприємства, так і в обробці даних. Для цього компанії в цілому світі використовують різні рішення, від простих до дуже складних, використовуючи усі засоби для збільшення продуктивності. Особливим рішенням є використання потужних CRM-систем.

Customer relationship management (CRM або управління відносинами з клієнтами) - поняття, що охоплює концепції, котрі використовуються компаніями для управління взаємовідносинами зі споживачами, включаючи збір, зберігання й аналіз інформації про споживачів, постачальників, партнерів та інформації про взаємовідносини з ними. З кінця минулого століття, CRM-системи пройшли шлях від маленьких додатків до багатофункціональних технологій управління даними. Звісно, збільшивши функціонал, у рішень збільшилася кількість потенційних загроз, що є найважливішим фактором в сучасному світі, коли безпека інформації та даних є основним питанням успішності підприємства.

Під час функціонування системи, вона працює з надзвичайно великим обсягом конфіденційної та особистої інформації: імена, номери паспортів, банківських карт, тощо. Залежно від галузі де CRM-система використовується, ризики безпеки даних можуть збільшуватися до неприпустимого рівня.

Тому виходячи з цього, витоки інформації є недопустимими, а впровадження надійного захисту є неймовірно важливим. Саме тому в наш час на це витрачаються великі гроші, а компанії виробники програмного забезпеченням наймають цілі відділи інформаційної безпеки.

Тому *метою роботи* є дослідження та розробка рекомендацій щодо вибору CRM – системи, а також щодо їх налаштування та безпечного використання.

Для досягнення поставленої мети необхідно вирішити такі *завдання*:

- Дослідити можливі загрози CRM – систем;
- Проаналізувати ознаки безпечної CRM – системи;
- Ознайомитися з існуючими засобами захисту від загроз;
- Зробити аналіз інструментарію існуючих рішень CRM – систем;
- Розглянути особливості побудови захисту таких систем;
- Розробити рекомендації для побудови захисту даних у CRM – системах.

Об'єктом дослідження в даній роботі є процес захисту даних у CRM – системах.

Предметом дослідження в даній роботі є методи та засоби захисту інформації у CRM – системах.

Методи дослідження дипломної роботи:

- аналіз літератури та ресурсів;
- аналіз документів;
- порівняння;
- огляд інструментів CRM – систем;
- огляд написання коду;
- вивчення та узагальнення вітчизняних та зарубіжних методів.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО CRM-СИСТЕМИ ТА ЇХ ЗАХИСТ

1.1 Поняття «CRM – система»

CRM-система (Customer Relationship Management або Управління відносинами з клієнтами) - це - прикладне програмне забезпечення для організацій, призначене для автоматизації стратегій взаємодії з замовниками (клієнтами), зокрема, для підвищення рівня продажів, оптимізації маркетингу і поліпшення обслуговування клієнтів шляхом збереження інформації про клієнтів і історію взаємин з ними, встановлення і поліпшення бізнес-процесів і подальшого аналізу результатів.

Принципи CRM-систем:

1. наявність єдиного сховища інформації, звідки в будь-який момент доступні усі відомості про усі випадки взаємодії з клієнтом;
2. синхронізація управління множинними каналами взаємодії;
3. постійний аналіз зібраної інформації про клієнтів та прийняття відповідних організаційних рішень — наприклад, «сортування» клієнтів на основі їхньої значимості для компанії.

1.2 Історія розвитку CRM - систем

До 1993 року ринок CRM складався з двох основних напрямків — автоматизації торгових представників (Sales Force Automation — SFA) та клієнтського обслуговування (Customer Service — CS). Первинне призначення автоматизованих систем управління територіальними продажами полягало в тому, щоб торгові представники могли управляти «точками дотику» своїх клієнтів, а також працювати з планом продаж, узгодженим із календарем. З часом подібні системи збагатилися впровадженням функції управління можливостями, що на практиці означало підтримку тактики та методології продаж, прийнятої в компанії, а також можливість взаємозв'язку з іншими підрозділами компанії, наприклад, із службою клієнтської підтримки чи сервісними службами.

До 2000 року CRM-системи, як правило, були «однобокими» — так звані «менеджери контактів», системи підтримки маркетингових заходів чи системи для автоматизації сервісних служб. В період з 2000 по 2005 роки почав формуватися спільний бізнес компаній із споживачами (Colaborative Commerce — спільна комерція). Спільна комерція характеризується налагоджуванням інтерактивної взаємодії компаній з їхніми постійними партнерами через Інтернет. Така взаємодія передбачає надання зовнішнім користувачам значно ширшого доступу до корпоративної інформації у зв'язку з чим повинна базуватися на принципах гарантії безпеки та довіри до партнера а також на узгоджених правилах роботи.

Після 2005 року наступила друга хвиля Colaborative Commerce, що базується на більшій відкритості ERP-систем. Провідні виробники стали створювати користувацькі інтерфейси для своїх ERP-систем, з'явилися електронні торговельні майданчики B2C, формується нова інфраструктура ведення бізнесу. У цьому випадку, на відміну від першої хвилі, мова іде про взаємодію «багато до багатьох», — підприємства співпрацюють не тільки з постійними партнерами, а й з усіма членами бізнес-суспільства. Практично усі сучасні CRM-системи отримали в більшій чи меншій мірі вказані вище можливості та рівні обробки та надання інформації — обробка і зберігання даних в колективних сховищах, розробка баз знань, інтернет-засоби для інтерактивної взаємодії з клієнтом засобами корпоративних порталів.

1.3 Безпека систем управління відносинами з клієнтами

Захищати дані про комерційну і операційну діяльність та надійно зберігати клієнтську базу - одна з основних задач CRM-системи, і в цьому вона й важливіша ніж усе інше прикладне ПЗ в компанії.

Річ у тому, що інформація, не тільки про клієнтів, а й про методи роботи з ними є дуже корисною для різних типів зловмисників. До того ж небезпека йде не тільки зовні, а й з середини, тому що інформація може бути скомпрометована співробітниками (інсайдерами). Тому крім зовнішнього захисту, дуже важливим є розмежування доступу до інформації у середині системи. Для того, щоб зрозуміти, яка CRM-система є безпечною, нам потрібно дізнатися ознаки небезпечної системи.

CRM бувають хмарні і десктопні. Хмарні - це ті, СУБД (база даних) яких розташовується не у вас в компанії, а в приватній або публічній хмарі в якомусь дата-центрі, коли десктопні базують свою СУБД на ваших власних серверах. Дістати несанкціонований доступ можна до CRM обох типів, але швидкість і простота доступу різні, особливо якщо ми говоримо про системи, розробник якої не сильно дбає про інформаційну безпеку.

Для десктопних систем відповідальність за мережеву безпеку та безпеку серверу падає на розробника системи, тобто на підприємство, якому належить рішення. Таким чином збільшуються витрати не тільки на сервер, бо він має існувати локально, а й на програмне та технічне забезпечення серверів, а також побудову архітектури для забезпечення інформаційної безпеки. Звісно, якщо розробити безпеку своєї системи таким чином, захищеність її від атак на проникнення буде максимальною, але втрата продуктивності системи і загальні витрати будуть досить великими.

Тому, в сучасному світі більшість компаній звертають увагу на використання хмарних рішень. Крім зменшення витрат, ці системи мають дуже потужні готові інструменти для побудови безпечних додатків. Але, як і в кожній системі є і ризики. Серед основних ризиків хмарних CRM-систем, я би виділив такі:

- хмарний ресурс може мати вразливості;

- можуть статися витік даних будь-якої природи (людський фактор, шахрайство, хакери і т.д.);

- такі функції як, наприклад, відновлення даних можуть бути відсутні, або можливі при додаткових витратах;

- небажаний доступ до системи.

Звісно, ризики можуть бути різними, але все таки переваг переходу на хмарні системи набагато більше. Переваги такого переходу очевидні: кожен співробітник, який має право доступу в систему, може працювати з інформацією в будь-якому місці і в будь-який час доби. Наприклад, при необхідності підняти всі дані по VIP клієнту менеджер, який перебуває у відрядженні за кордоном, без проблем входить в систему, підключаючись до виділеного сервера, на якому і знаходиться CRM. Зниження витрат на IT - ще одна перевага переходу на хмарні технології. Як правило, CRM знаходяться на виділеному сервері, і техобслуговування входить в загальну вартість послуг з надання віртуального простору.

1.4 Шляхи захисту CRM - систем

Розглянувши можливі ризики CRM-систем, можна виділити так шляхи їх захисту:

1. Відновлення даних (бекап) - найважливіша річ, про яку нерідко або забувають, або не піклуються. Якщо у вас десктопна система, налаштуйте систему резервного копіювання даних із заданою частотою і організуйте грамотне зберігання копій. Якщо у вас хмарна CRM, обов'язково до укладення договору дізнайтеся, як організована робота з резервних копій: вам потрібні відомості про глибину і частоті, про місце зберігання, про вартість відновлення даних (нерідко безкоштовні тільки бекапи «останніх даних на період», а повноцінне, безпечне резервне копіювання здійснюється як платна послуга). Загалом, тут точно не місце для економії або недбалого ставлення. І так, не забувайте перевіряти те, що відновлюється з резервних копій.

2. Поділ прав доступу на рівні функцій і даних. Теж дуже важливий крок до забезпечення інформаційної безпеки. Ким би не був користувач, клієнтом чи адміністратором, або, можливо, менеджером компанії, він повинен мати доступ лише до тих даних і функцій, до яких це не буде порушувати політику конфіденційності. Це дуже важливо, так як CRM-системи майже завжди працюють з безліччю приватної інформації, такі як платні реквізити, адреси клієнтів, тощо.

3. Безпека на рівні мережі - потрібно дозволити використання CRM тільки всередині офісної підмережі, обмежити доступ для мобільних пристроїв, заборонити

роботу з CRM-системою з дому або, що ще гірше, з публічних мереж (коворкінг, кафе, клієнтських офісів та ін.). Особливо будьте обережні з мобільною версією - нехай вона буде лише сильно усіченим варіантом для роботи.

4. Антивірус зі скануванням в режимі реального часу потрібен в будь-якому випадку, але в разі безпеки корпоративних даних - особливо. Забороніть на рівні політик відключати його самостійно.

5. Навчання співробітників гігієни кіберпростору - не порожня трата часу, а гостра необхідність. Потрібно донести до всіх колег, що їм важливо не тільки попередити, але і правильно зреагувати на інформацію, що надійшла загрозу. Забороняти користуватися інтернетом або своєю поштою в офісі - це минуле століття і причина гострого негативу, тому доведеться попрацювати саме з профілактикою.

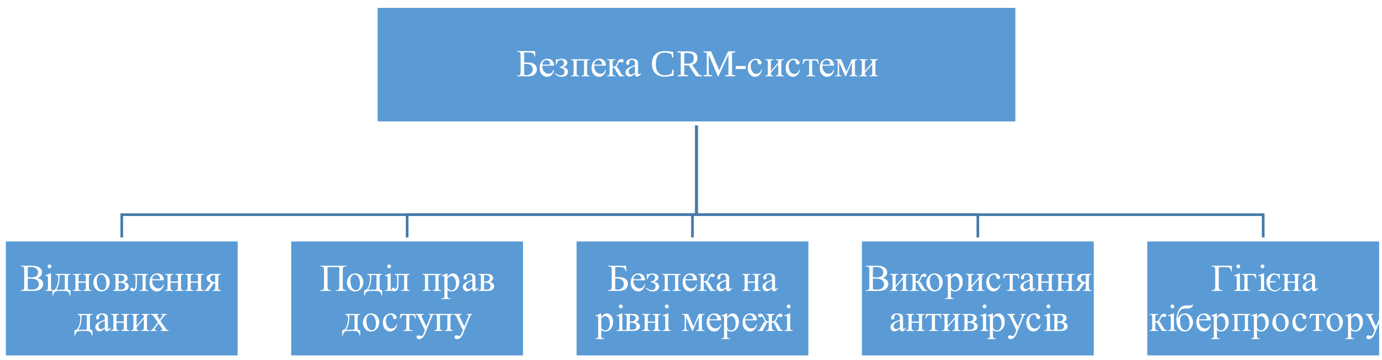

Рисунок 1.1 Складові безпеки CRM систем

Звичайно, використовуючи хмарну систему, можна домогтися достатнього рівня безпеки: використовувати виділені сервера, налаштовувати маршрутизатори і обробити трафік на рівні додатку та рівні баз даних, використовувати приватні підмережі, ввести суворі правила безпеки для адміністраторів, забезпечити безперебійність за рахунок резервного копіювання з максимально необхідної частотою і повнотою, здійснювати цілодобовий моніторинг мережі. Якщо вдуматися, це не так і складно - швидше дорого. Але, як показує практика, такі заходи вживають тільки деякі компанії, в основному великі.

1.5 Більше про хмарні CRM-системи

Термін хмарні технології пішов від англійського словосполучення "cloud technology", так як дослівний переклад такого слова, як "cloud" означає "хмара", але в іншому розумінні це ж саме слово можна перекласти як "розсіяний" або "розподілений".

Саме тому, можна сказати, що хмарні технології – це "розподілені технології", тобто дані опрацьовуються з використанням не лише одного комп'ютера, а опрацювання розподіляється по декількох комп'ютерах, які підключені до мережі Internet.

Хмарні сервіси використовуються великими й маленькими компаніями незалежно від сфери бізнесу. Вони відрізняються зручністю й простотою, а також мобільністю. Користувачам не потрібно встановлювати жодних додатків та мати власних серверів. До роботи можна приступати одразу після реєстрації й оплати вибраного тарифу.

CRM-системи які працюють за допомогою хмарних технологій використовують Saas модель, або сервіс як послуга. SaaS (Software as a Service - Програмне забезпечення як послуга) - це вигідна альтернатива придбання програмного забезпечення. SaaS дозволяє отримувати програмне забезпечення як послугу, а не купувати дорогі ліцензійні програми.

Ми вже описували кроки для захисту CRM – систем, тому для захисту Saas платформ можна виділити такі властивості:

1. Створення хмарних середовищ з урахуванням вимог безпеки

Як правило, витік даних відбувається в результаті того, що кіберзлочинці використовують найслабшу ланку в поверхні атаки. У багатьох організаціях впровадження хмари призводить до розширення поверхні атаки в геометричній прогресії. Для усунення цих слабких ланок потрібно повсюдно забезпечувати єдиний рівень безпеки, навіть коли інфраструктура постійно змінюється.

2. Нативна хмарна безпека

Оскільки дані і робочі процеси повинні переміщатися по всій інфраструктурі і в хмарі, необхідно домагатися узгодженої роботи рішень безпеки. Вибір хмарного брандмауера від того ж постачальника, чиї продукти захищають і фізичні активи організації, зовсім не обов'язково гарантує вирішення цієї проблеми. Подібні рішення повинні безперешкодно взаємодіяти з хмарними сервісами і стежити за роботою цих сервісів, а також вміти ідентифікувати хмарні ресурси таким же логічним чином, як вони ідентифікують інші ресурси. Проте, хоча базова технологія, яка використовується для захисту мереж, значно відрізняється від технологій, які використовуються для захисту хмарних ресурсів, практика управління аспектами безпеки повинна залишатися такою ж. Ось чому вирішальне значення має природна інтеграція в хмарну інфраструктуру.

3. Різні форм-фактори

Для комплексної, узгодженої захисту необхідно, щоб одні й ті ж рішення безпеки були розгорнуті на якомога більшій кількості платформ і в максимально можливій кількості різних форм-факторів. Наприклад, додатки повинні мати можливість звертатися до хмарного рішенням безпеки для ідентифікації та захисту окремих видів даних і транзакцій. Контейнерні додатки повинні мати доступ до контейнерних інструментів безпеки для зручної інтеграції функцій безпеки в ланцюжок додатків. І в ідеалі ці інструменти повинні працювати точно так само, як і рішення, розгорнуті у вашій розподіленої інфраструктури, в тому числі у філіях і на периферійних пристроях.

4. Централізоване управління

Одна з найбільших претензій з боку мережевих адміністраторів полягає в тому, що вони позбавлені єдиної консолі, за допомогою якої вони могли б бачити всю інфраструктуру і управляти всією своєю мережею і яка б забезпечувала прозорість фізичних і віртуальних мереж. Якщо для управління безпекою використовувати рішення, які здатні виявляти атаки і захищати інфраструктуру тільки в окремих сегментах мережі, але не у всій мережі, то, швидше за все, це призведе до компрометації інфраструктури. Щоб усунути прогалини в безпеці, організаціям потрібна єдина панель управління, яка забезпечує наочність і дозволяє сформувати узгоджені політики безпеки у всій інфраструктурі для ефективного управління ризиками. Рішення безпеки повинні обмінюватися і зіставляти відомості про загрози, отримувати і впроваджувати централізовано узгоджені політики і зміни в конфігураціях і координувати всі ресурси для своєчасного реагування на виявлені загрози.

Отже, маючи такі особливості захисту хмарних CRM-систем, ми можемо визначити потреби рішення, яке ми маємо обрати для нашого додатку.

1.6 Аналіз хмарної платформи Salesforce

З попередніх пунктів ми розглянули базові поняття CRM – систем, які вони бувають та що потрібно для їх продуктивної і безпечної роботи. Також ми вивели рекомендації щодо захисту хмарних рішень. Слід зазначити, що в сучасному світі існує безліч готових рішень майже для усіх потреб ринку, їх як правило можна змінювати та кастомізувати під себе, роблячи різного виду додатки і покращуючи продуктивність вашого підприємства. Та серед усіх їх слід відзначити CRM-систему на базі Salesforce.

Salesforce.com — американська компанія, розробник однойменної CRM-системи, що надається замовникам виключно за моделлю SaaS. Під найменування Force.com компанія представляє PaaS-платформу для самостійної розробки додатків, а під брендом Database.com — хмарну систему управління базами даних.

Рисунок 1.2 Логотип компанії Salesforce

Salesforce — це платформа, яка повністю знаходиться на серверах компанії Salesforce в клауді. Компанія Salesforce була заснована в 1999 році колишнім виконавчим директором Oracle Марком Беніоффом. Головна ідея створення — це побудова доступного програмного забезпечення і впровадження його повністю онлайн в якості сервісу. Слід звернути увагу, що Salesforce уже давно вийшла за рамки просто CRM. Salesforce — це хмарна-платформа, на базі якої, крім CRM частини, є багато цікавого.

Salesforce дозволяє створювати та розгортати індивідуальні рішення, автоматизовувати бізнес-процеси, інтегруватися із зовнішніми додатками. Більшість світових ентерпрайз-компаній є клієнтами компанії Salesforce і використовують цю платформу як рішення для своїх бізнес-потреб. Серед них: Adidas, AWS, Canon, Philips, Toyota, American Express, Western Union, Cisco, KLM і багато інших. Для мене стало відкриттям, що візовий центр США використовує Salesforce для збирання запитів на візи, їхньої обробки та видачі. Salesforce, як компанія, є партнером різноманітних представників IT-світу і, відповідно, вони пропонують готові інтеграційні рішення, удосконалені сервіси для задоволення вимог кінцевих клієнтів і ще багато різних можливостей. Серед них — Apple, Microsoft, Google, Amazon.

З 2014 року компанія є найбільшим на ринку продавцем CRM-рішень, що можна побачити на рисунку 1.3. Крім, кращого рішення в галузі управління відносин з клієнтами, компанія надає безліч прикладних послуг, які можуть бути інтегровані в роботи системи та використані для підвищення ефективності та можливостей роботи вашого застосунку. Але для нас в нашому дослідженні важливим є питання безпеки та можливостей для нашого додатку, і що у Salesforce з безпекою?

Можна впевнено сказати, що Salesforce є одним з найбезпечніших рішень у цій галузі і це не лише через її велику популярність. Я би виділив такі особливості щодо безпеки даних та користувача:

1. Система повністю працює у хмарі.

Ми вже досліджували переваги хмарних технологій над десктопними, тому слід зазначити лише те, що система захищена від стандартних десктопних вразливостей.

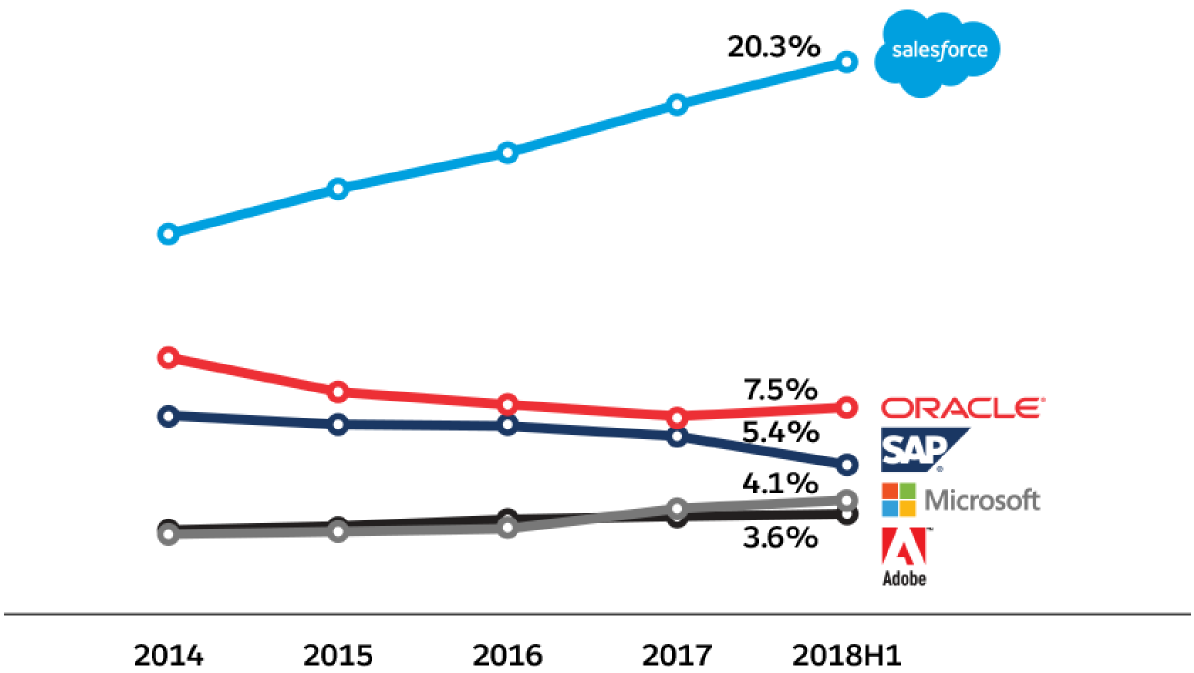

Рисунок 1.3 Світові продавці CRM-рішень

2. Постійні оновлення

Цей пункт є дуже важливим, бо все ж працюючи з хмарними платформами достатній захист систем можна забезпечити тільки при постійному оновленні програмного забезпечення для захисту від нових вразливостей, вірусів, тощо, які виникають кожен день. У Salesforce з цим все дуже добре – маленькі оновлення є постійними, великі ж версії рішення виходять 3 рази на рік.

3. Відновлення даних

На мою думку, найважливіший пункт. Основна функція управління відносин з клієнтами – обробка великої кількості даних, і можливість відновити їх. У Salesforce ця можливість вбудована, до того ж є декілька шляхів для бєкапу ваших даних

4. Інструментарій для розробки рішень та інформаційної безпеки

Основною перевагою системи на іншими є надзвичайно гнучкі інструменти для побудови клієнтських рішень. Крім, багатого вбудованого функціоналу, клієнт можемо розширити своє рішення за допомогою розробки своїх компонентів та об'єктів, а також потужних додатків в інтеграції з готовим функціоналом. Повністю весь інструментарій ми розглянемо у другому розділі, де зможемо дослідити, що система пропонує нам для захисту нашого додатку.

5. Salesforce – готова система

Зразок додатку можливо було би розробити з нуля, але використовуючи готове рішення, ми по-перше економимо час для роботи над іншими аспектами нашого додатку, а по-друге маємо, наприклад, готовий процес авторизації та автентифікації, який ми можем трохи змінити під себе, наприклад, додати двухфакторну автентифікації, тощо. Ми можемо бути впевненими у безпечності готового рішення.

Висновки за розділом 1

У цьому розділі ми розглянули поняття CRM системи, визначили які вони бувають, дослідили переваги кожного з типів, а також обрали який тип рішень підходить нам. Крім цього, ми описали основні цілі захисту інформації у системах управління відносинами з клієнтом, для цього ми визначили основні загрози під час використання таких систем та як їх можна подолати.

Після всього цього ми визначилися з обраним рішенням за яким ми будемо будувати наш додаток і описали чому ми вибрали саме його. Отже, результатом дослідження у цьому розділі стали описані загрози та ризики, можливі методи їх усунення та переваги системи Salesforce яку ми обрали для побудови нашого проекту. У наступному розділі ми більше розглянемо, що саме надає нам Salesforce для побудови ефективної та безпечної CRM– системи і опишемо методи подолання різних загроз на прикладі використання наданих інструментів.

РОЗДІЛ 2
ОСОБЛИВОСТІ ПОБУДОВИ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ У СИСТЕМІ SALESFORCE

2.1 Аналіз інструментів розмежування прав доступу до даних

Головною функцією використання систем управління відносин з клієнтами – це робота з даними. Дуже часто під час розширення клієнтури підприємства, працювати з ними стає все складніше, кількість даних зростає до неймовірності. Тут компанії, як правило, і вирішують використати CRM – системи. Тому очевидним є те, що системи працюють з великим обсягом даних, як простих так і конфіденційних. В наші часи інформація про клієнтів надзвичайно корисна, отже, знайдеться багато зловмисників, які спробують заволодіти нею. Виходячи з цього, на перший план постає інформаційна безпека, а саме забезпечення конфіденційності, цілісності та доступності інформації.

Треба розуміти, що доступ до системи можуть мати багато людей, це можуть бути як адміністратори та керівники компанії, так і звичайні працівники. В таких умовах для нас важливо зберегти конфіденційність даних, тобто забезпечити для користувача доступ до тих даних, якими він володіє або повинен мати доступ.

Як і в багатьох системах, найлегший спосіб забезпечити конфіденційність є розмежування прав доступу. Але в кожному рішенні або додатку, цей спосіб може бути впроваджений по різному. Ми ж розглянемо як працює розмежування доступу в CRM – системі Salesforce.

Salesforce надає дуже потужний інструментарій для побудови інформаційної безпеки на багатьох рівнях. Перш ніж поговорити про саму безпеку, слід відзначити про доступ до самої системи. Коли компанія отримує підписку до системи, вона отримує доступ до організації (*англ.* Organization). Організація це вже готове програмне забезпечення, яке знаходиться в хмарі і доступ до якої ви можете отримати за допомогою автентифікації за посиланням login.salesforce.com. Там ви отримаєте готові інструменти для адміністрування та побудови ваших рішень, при чому розробку можна вести за допомогою не тільки програмного коду, ще й за допомогою готових функцій.

Завдяки гнучкій багаторівневій моделі спільного використання платформи Salesforce легко призначити різні набори даних різним наборам користувачів. Ви можете збалансувати безпеку та зручність, зменшити ризик викрадених або зловживаних даних, і при цьому переконатися, що всі користувачі можуть легко отримати необхідні дані.

Платформа дозволяє легко вказати, які користувачі можуть переглядати, створювати, редагувати або видаляти будь-який запис або поле в програмі. Ви можете контролювати доступ до цілої організації, конкретного об'єкта, конкретного поля чи навіть окремого запису. Поєднуючи засоби керування безпекою на різних рівнях, ви можете забезпечити потрібний рівень доступу до даних тисячам користувачів без необхідності вказувати дозволи для кожного користувача окремо.

На рівні організації ми можемо налаштувати доступ до неї, тобто якщо якісь умови будуть порушені, користувач не зайде в систему взагалі. За допомогою налаштувань організації ви можете:

- Встановити політику щодо паролів

Встановити політику щодо паролів та входу, наприклад, зазначте час до закінчення терміну дії всіх паролів користувачів та рівень складності, необхідний для паролів.

- Встановити термін дії пароля користувача

Термін дії паролів закінчується для всіх користувачів у вашій організації, за винятком користувачів із дозволом «Пароль ніколи не закінчується».

- Скинути пароль користувача

Скинути пароль для вказаних користувачів.

- Побачити спроби входу та періоди блокування

Якщо користувач заблокований через занадто багато невдалих спроб входу, ви можете розблокувати доступ до його облікового запису.

До того ж, майже все це можна налаштувати для окремого користувача чи групи користувачів.

Крім того, дуже важливим є налаштування безпечної системи авторизації та автентифікації, що є, на мою думку, найголовнішим у кожній системі незалежно від галузі використання. Найефективніший спосіб захистити свою організацію та її дані - вимагати, щоб користувачі надавали не лише своє ім’я користувача та пароль. Експерти з безпеки називають це багатофакторною автентифікацією. В системі ми можемо вибрати чи будемо ми її використовувати, якщо вона потрібна, то компанія надає можливість налаштувати багатофакторну автентифікацію за допомогою додатку на телефоні.

При роботі з самими даними, ми можемо використовувати багато інструментів, які існують в системі, але кожний з них має своє призначення. За цією ознакою ми можемо поділити рівні контролю доступу на такі:

1. Рівень об’єктів (Об’єкт у Salesforce – таблиця бази даних)
2. Рівень записів у об’єкті
3. Рівень полів у записі

Кожен з рівнів важливий і використовується при побудові певного функціоналу.

Рівень об’єктів .

Найпростіший спосіб контролю доступу до даних - це встановлення дозволів для певного типу об’єкта. Ви можете контролювати, чи може група користувачів створювати, переглядати, редагувати чи видаляти будь-які записи цього об’єкта.

Ви можете встановити дозволи на об’єкти за допомогою профілів або наборів дозволів. Користувач може мати один профіль і безліч наборів дозволів.

- Профіль користувача визначає об’єкти, до яких вони можуть отримати доступ, і те, що вони можуть робити з будь-яким записом об’єкта (наприклад, створювати, читати, редагувати чи видаляти).

- Набори дозволів надають користувачеві додаткові дозволи та налаштування доступу.

Використовуйте профілі, щоб надати мінімальні дозволи та налаштування, які потрібні всім користувачам певного типу. Потім використовуйте набори дозволів, щоб надавати більше дозволів за потреби. Поєднання профілів та наборів дозволів надає вам велику гнучкість у визначенні доступу на рівні об’єкта.

Слід зазначити, що профілі працюють далеко не лише з рівнем об’єкту, та й взагалі не тільки з даними. Набори дозволів можуть існувати, як окремо, так і в групах наборів дозволів, що полегшує присвоєння їх користувачам.

Рівень записів.

Щоб точно контролювати доступ до даних, ви можете дозволити певним користувачам переглядати певні поля в певному об’єкті, але потім обмежити окремі записи, які їм дозволено бачити.

Доступ до записів визначає, які окремі записи користувачі можуть переглядати та редагувати в кожному об’єкті, до якого вони мають доступ у своєму профілі. Спочатку задайте собі такі запитання:

1.Чи повинні ваші користувачі мати відкритий доступ до кожного запису чи просто до підмножини?

2.Якщо це підмножина, які правила повинні визначати, чи може користувач мати до них доступ?

Ви контролюєте доступ на рівні запису чотирма способами. Вони перераховані в порядку збільшення доступу. Ви використовуєте за замовчуванням «організацію за замовчуванням», щоб заблокувати ваші дані до найбільш обмежувального рівня, а потім використовуйте інші засоби захисту рівня запису, щоб надати доступ обраним користувачам, як потрібно.

- За замовчуванням для організації (Organization Wide Defaults) визначається рівень доступу користувачів до записів один одного для усієї організації.

- Ієрархії ролей (Hierarchy role) забезпечують доступ керівників до тих самих записів, що і їхні підлеглі. Кожна роль в ієрархії представляє рівень доступу до даних, необхідний користувачеві або групі користувачів.

- Автоматичні правила спільного доступу (Automatic Sharing rules) - це автоматичні винятки за замовчуванням для організації для певних груп користувачів, щоб надати їм доступ до записів, якими вони не володіють або які зазвичай не бачать.

- Спільний доступ наданий вручну (Manual sharing) дозволяє власникам записів надавати дозволи на читання та редагування користувачам, які можуть не мати доступу до запису будь-яким іншим способом.

Загально рівень доступу до записів можна представити так, як на рисунку 2.1.

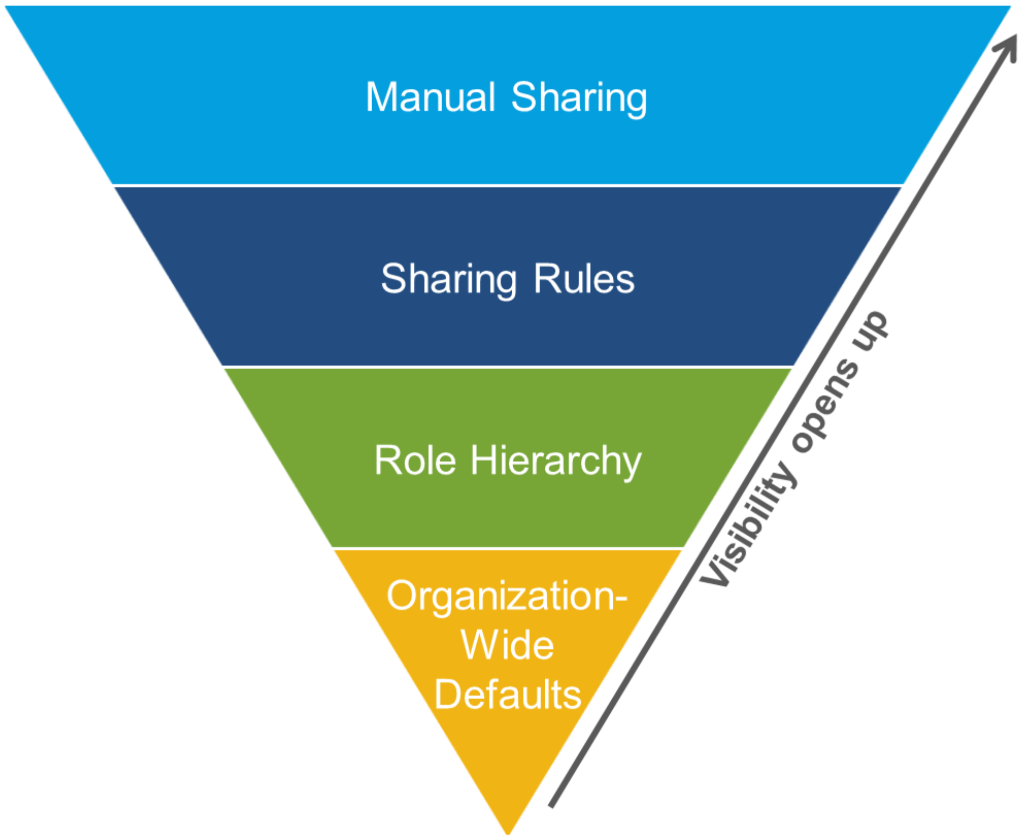

Рисунок 2.1. Рівень доступу до записів об’єкту.

Окремо я хотів би зупинитися на ієрархічних ролях, тому що це, мабуть, найбільш використовувана функція для розмежування прав доступу до записів. Ієрархія ролей працює разом із налаштуваннями спільного доступу для визначення рівнів доступу користувачів до ваших даних Salesforce. Користувачі можуть отримати доступ до даних усіх користувачів безпосередньо під ними в ієрархії.

Користувачі, яким потрібно бачити багато даних (наприклад, генеральний директор, керівники або інше керівництво), часто з’являються вгорі ієрархії. Але ієрархії ролей не повинні відповідати вашій організаційній діаграмі. Кожна роль в ієрархії просто представляє рівень доступу до даних, який потрібен користувачеві або групі користувачів.

Залежно від налаштувань спільного доступу, ролі можуть керувати рівнем видимості даних ваших Salesforce до користувачів. Користувачі на будь-якому рівні ролі можуть переглядати, редагувати та звітувати про всі дані, якими володіють користувачі або які надаються їм нижче за ними в ієрархії ролей, якщо ваша модель спільного використання для об'єкта не визначає інше. Зокрема, у списку, пов'язаному із замовчуванням для організації, якщо для користувацького об'єкта вимкнено параметр Надання доступу за допомогою ієрархій, доступ до записів об'єкта отримують лише власник запису та користувачі, яким надано доступ за замовчуванням для організації.

Рівень доступу до полів записів.

Організація рівнів доступу до полів записів надає нам можливість контролювати, які поля може бачити користувач залежно від його ролі або посади у компанії. У деяких випадках ви хочете, щоб користувачі мали доступ до об'єкта, але обмежили їх доступ до окремих полів у цьому об'єкті. Параметри захисту на рівні поля - або дозволи полів - визначають, чи може користувач бачити, редагувати та видаляти значення для певного поля на об'єкті. Це налаштування, які дозволяють захищати конфіденційні поля, такі як номер соціального страхування кандидата, без необхідності приховувати об'єкт кандидата.

На відміну від макетів сторінок, які контролюють лише видимість полів на деталях та редагують сторінки, безпека на рівні полів контролює видимість полів у будь-якій частині програми, включаючи пов'язані списки, подання списків, звіти та результати пошуку. Насправді, щоб повністю переконатись, що користувач не може отримати доступ до певного поля, важливо використовувати сторінку захисту на рівні поля для даного об'єкта, щоб обмежити доступ до поля. Просто немає інших ярликів, які забезпечують однаковий рівень захисту для певного поля.

Приклад розмежування прав для кожного з рівнів ми можемо розглянути на рисунку 2.2.

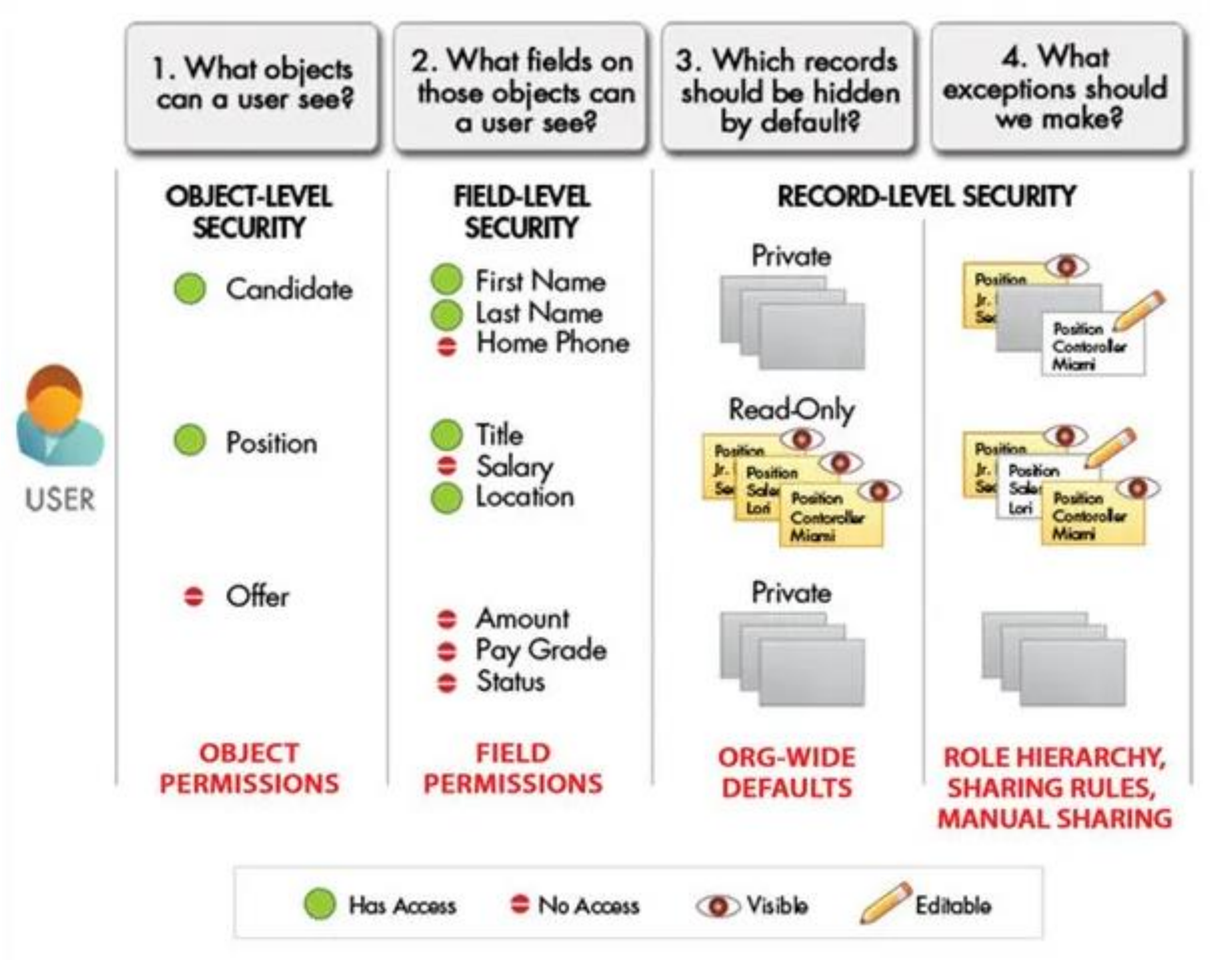

Рисунок 2.2 Приклад доступу користувача до даних

2.2 Аналіз інструментів моніторингу та відновлення даних

В першому розділі ми розглянули рекомендації для захисту CRM - систем. Там ми описували, що дуже важливим є в роботі з даними їх моніторинг, тобто крім прямого захисту, адміністратор і компанія мають добре контролювати ситуації, які можуть виникати. Тому, важливим фактором в виборі рішення для кожного додатку є наявність в системі інструментів для контролю даних, авторизації користувачів, моніторингу їх активності, тощо.

Salesforce надає адміністраторам корисні вбудовані інструменти для моніторингу активності користувачів. Доступ до них мають тільки користувачі, профіль яких стосується адміністрування. Серед них можна виділити, найважливіші:

1. Історія входу користувачів – ви можете переглянути список успішних та невдалих спроб входу користувачів за останні 6 місяців.

2. Журнали відладки – тут ви можете переглянути логи. Дуже потужний інструмент для фахівців з безпеки, бо надає можливість сповіщати адміністраторів про проблеми з коду.

3. Налаштування аудиту перевірки - налаштування журналу перевірки, коли вносяться зміни до конфігурації вашої організації.

Професіональне використання інструментів моніторингу допомагає нам легко і швидко знайти причину неполадок або завчасно знайти можливу загрозу. Але якщо зловмиснику все ж вдалася його атака чи дані були втрачені із-за якоїсь другої причини, їх втрата може бути фатальною для підприємства. Тому як ми і описували в першому розділі в цьому нам знадобляться інструменти для відновлення даних або бекапу.

Резе́рвне копіюва́ння або бека́п (англ. *backup*) — процес створення копії даних з носія (жорсткого диска, дискети тощо), призначений для відновлення цих даних у разі їх пошкодження або видалення. Створення резервної копії даних надає можливість виконати відновлення інформації при втраті оригіналу, з якого було створено резервну копію. При цьому під втратою треба розуміти настання події, що призвела до зміни даних, після чого вони втратили цінність або були видалені з носія. Приклад: умисне завдання шкоди через видалення важливої для підприємства інформації. Об'єкти резервного копіювання — це дані або сукупність даних, з яких можна створити резервну копію. Приклади об'єктів: файли або теки, дані прикладних програм, дані операційної системи чи сама ОС (наприклад Windows System State або AIX System Backup), образи віртуальних машин та дисків віртуальних машин, файлові системи тощо.

Навіть із найкращими намірами користувачі та адміністратори потрапляли в ситуації, коли вони або видаляли великі обсяги даних, або модифікували записи, щоб пізніше зрозуміти, що було допущено помилку. За допомогою таких інструментів, як Data Loader, дуже легко масово видаляти або оновлювати записи. І проста помилка у зіставленні вихідного файлу або поля може спричинити катастрофу для ваших даних. Рекомендується регулярно створювати резервні копії даних і робити вручну резервне копіювання даних у часі, перш ніж продовжувати будь-який основний проект з даними у вашій організації.

У системі присутні такі інструменти для резервного копіювання та відновлення даних:

- Data Export Service - Експорт даних вручну або за графіком за допомогою інтерфейсу користувача.
- Data Loader - Вручну експортуйте дані на вимогу за допомогою API.
- Report Export - Вручну експортуйте дані на вимогу за допомогою звітів.

Кожна утиліта може допомогти у відновленні даних, вони відрізняються інтенсивністю використання, наявністю графічного інтерфейсу, кількістю даних, які можна відновити, тощо. Але якщо через ці інструменти відновити дані неможливо, то можна звернутися до компанії Salesforce у підтримку та запросити відновлення даних у них.

2.3 Аналіз веб загроз

Розглядуючи кожну CRM – систему, особливо хмарну, слід зазначити, що доступ до неї користувач отримує за допомогою мережі Інтернет. В наш час у всесвітній мережі існує безліч загроз та ризиків пов’язаних з інформаційною безпекою користувача. Тому від Salesforce нам потрібні інструменти, які можуть позбавити нас цих проблем з безпекою та запобігти виникнення цих проблем у майбутньому. Та перш ніж розглянути, що нам надає система, ми маємо розглянути основні загрози, які можуть виникнути для даних.

Серед ризиків інформаційної безпеки для веб-додатків, слід поділити їх на ті які виникають на клієнтській стороні та на стороні серверу. На стороні клієнту (браузеру):

1. XSS (англ. Cross Site Scripting — «міжсайтовий скриптинг») — тип вразливості інтерактивних інформаційних систем у вебі. XSS виникає, коли на сторінки, які були згенеровані сервером, з якоїсь причини потрапляють

користувацькі скрипти. Специфіка подібних атак полягає в тому, що замість безпосередньої атаки сервера зловмисники використовують вразливий сервер для атаки на користувача. Для терміну використовують скорочення «XSS», щоб не було плутанини з каскадними таблицями стилів (абревіатура «CSS»). Довгий час програмісти не приділяли їм належної уваги, вважаючи їх безпечними. Однак ця думка помилкова: на сторінці або в HTTP-Cookie можуть бути досить вразливі дані (наприклад, ідентифікатор сесії адміністратора). На популярному сайті скрипт може влаштувати DoS-атаку.

2. Атака CSRF або Cross-Site Request Forgery дозволяє зловмисникові змусити браузер жертви відправити певний HTTP запит, включаючи куки, файли сеансу і будь-яку іншу, автоматично включається інформацію в уразливе веб-додаток. Таким чином, зловмисник отримує можливість генерувати запити з браузера жертви, які додаток вважає правильними і відправленими самою жертвою. Наприклад, ви просто відкриваєте посилання, а сайт вже відправляє вашим друзям повідомлення з рекламним змістом без вашого відома або видаляє ваш аккаунт.

Типи атак описані вище є найрозповсюдженішими серед клієнтських атак, а також несуть найбільшу шкоду для даних. Тепер можемо розглянути атаки, які стосуються більш серверною частини:

1. Ін’єкції. Під ін'єкціями маються на увазі уразливості, які виникають в результаті передачі неперевірених, введених користувачем даних інтерпретатора для виконання. Таким чином, будь-який користувач може виконати довільний код у інтерпретаторі. Найпоширеніші типи ін'єкцій - SQL, OS, XXE і LDAP. Ін'єкції виникають, коли дані, що передаються інтерпретатору не перевіряє на наявність керуючих послідовностей і команд, таких, як лапки в SQL.

2. Проблеми аутентифікації і перевірки сесій. Багатьом додаткам необхідно ідентифікувати користувачів для роботи з ними. Часто функції для перевірки автентичності та управління сесіями реалізуються невірно, що дозволяє зловмисникам отримувати доступ до облікових записів користувачів в обхід паролів. Зловмисники можуть будь-яким чином перехопити ключі або токени

сеансу, які виступають в якості ідентифікаторів користувачів і використовувати їх тимчасово або постійно.

3. Незахищені API. Більшість сучасних додатків часто включають в себе клієнтські додаток і багаті API інтерфейси, доступні через JavaScript в браузері або з мобільних додатків. Вони можуть працювати по протоколах SOAP / XML, REST / JSON, RPC, GWT і так далі. Ці API дуже часто не захищені і теж містять безліч помилок, які призводять до вразливостей.

Ознайомившись з потенційними загрозами, ми тепер можемо розглянути, що надає Salesforce, як готова CRM – система, для боротьби з ними.

2.4 Боротьба з веб загрозами у Salesforce

Salesforce Shield

Salesforce Shield - це набір інструментів безпеки, які адміністратори та розробники можуть використовувати для захисту критично важливих для бізнесу програм із такими можливостями, як розширене шифрування та моніторинг подій. Shield дозволяє вибудувати новий рівень довіри, прозорості, відповідності та управління за допомогою набору простих інструментів "наведи і натисніть". Ці інструменти включають шифрування платформи, моніторинг подій та стежку аудиту на місцях.

Оскільки все більше клієнтів використовують Salesforce для зберігання особистої інформації, включаючи конфіденційні або власні дані, важливо забезпечити безпеку доступу до цих даних. Інструменти, що входять до складу Shield, дозволяють відповідати як зовнішній, так і внутрішній політиці дотримання даних.

Шифрування платформи

Шифрування платформи призначене для збереження критично важливих функціональних можливостей програми, таких як пошук, робочий процес та правила перевірки, завдяки чому ви зберігаєте повний контроль над ключами шифрування та можете встановлювати дозволи на зашифровані дані для захисту конфіденційних даних від неавторизованих користувачів. Шифрування на платформі дозволяє вам власним чином зашифрувати свої найбільш конфіденційні дані в спокої у всіх ваших програмах Salesforce.

Моніторинг подій

Моніторинг подій дає вам доступ до детальних даних про продуктивність, безпеку та використання у всіх ваших програмах Salesforce. Моніторинг подій схожий на вікно, яке відображає всі детальні дані про діяльність користувачів у вашій організації. Ми розглядаємо ці дії користувачів як події, які фіксуються у чомусь, що називається журналом подій. Ви можете переглядати інформацію про окремі події або відстежувати тенденції подій, щоб швидко виявити ненормальну поведінку та захистити дані вашої компанії.

Польовий аудит

Поле аудиту дає змогу дізнатися стан та цінність ваших даних на будь-яку дату та в будь-який час. Ви можете використовувати його для дотримання нормативних вимог, внутрішнього управління, аудиту або обслуговування споживачів. Перевірка поля дозволяє визначити політику збереження заархівованих даних історії полів протягом 10 років з моменту архівування даних. Ця функція допомагає дотримуватися галузевих норм, що стосуються можливостей аудиту та збереження даних.

2.5 Клієнтська безпека у Salesforce

Клієнтська безпека для Lightning Web Components (LWC)

Веб-компоненти Lightning (LWC) - це інтерфейс JavaScript Salesforce. Захист на стороні клієнта набуває все більшого значення, оскільки розробники продовжують створювати додатки на платформі Salesforce, де все більше взаємодії з користувачами. Тому слід більше розглянути, як саме Це допоможе вам запобігти уразливості безпеки у ваших програмах та забезпечить набагато кращий досвід роботи користувачів.

Salesforce рекомендує LWC як основну структуру розробки інтерфейсу для використання на платформі Salesforce з двох причин. По-перше, LWC за замовчуванням має захищений Locker, що запобігає доступу компонентів до даних та об’єктної моделі документа (DOM) інших компонентів з інших просторів імен. Це означає, що вам не доведеться турбуватися про те, що одна програма витікає або отримує доступ до інформації з іншої. По-друге, LWC пропонує власні вбудовані функції безпеки, такі як дезінфекція неправильно сформованого HTML-коду та блокування коду, несумісного з політикою безпеки вмісту.

LWC використовує Lightning Locker, потужну архітектуру захисту компонентів, яка підвищує безпеку, ізолюючи компоненти Lightning в окремих просторах імен. На відміну від фреймворків, які покладаються на iframes для ізоляції (як і більшість подібних продуктів), Lightning Locker можна розглядати як бібліотеку ізоляції простору імен JavaScript. Він працює шляхом ведення списку пісочниць, приєднаних до певних просторів імен, та запобігання виконанню JavaScript, яке виходить за межі цих пісочниць.

Lightning Locker також просуває найкращі практики, що покращують безпеку програми, дозволяючи доступ лише до ретельно перевірених API.

До функцій безпеки Lightning Locker належать:

- Застосування строгого режиму JavaScript: Lightning Locker неявно включає жорсткий режим JavaScript. Вам не потрібно вказувати "use strict" у своєму коді. Суворий режим JavaScript робить код більш безпечним, надійним та підтримуваним.

- Зберігання доступу до DOM: Компонент може обходити лише DOM та елементи доступу, створені компонентом у тому самому просторі імен. Така поведінка заважає анти-шаблону потрапляти в елементи DOM, що належать компонентам в іншому просторі імен.

- Захищені обгортки: Обмежує використання глобальних об'єктів, приховуючи об'єкт або обертаючи його в захищену версію об'єкта. Наприклад, захищеною версією об'єкта вікна є SecureWindow. Locker перехоплює виклики до вікна і замість цього використовує SecureWindow. Деякі методи та властивості мають різну поведінку або недоступні для захищених об’єктів.

- Обмеження функції eval (): Використання функції eval () підтримується, щоб дозволити використання сторонніх бібліотек, які динамічно оцінюють код. Однак він обмежений роботою лише в глобальному масштабі простору імен. Функція eval () не може отримати доступ до локальних змінних у межах області, в якій вона викликається.

- Аналіз типів MIME: аналізуються типи файлів MIME, що надсилаються через браузер. Locker дозволяє деякі типи MIME, дезінфікує деякі, а решту блокує.

- Обмежений доступ до фреймворку API: Ви можете отримати доступ лише до опублікованих підтримуваних методів фреймворку API JavaScript.

Використання LWC допомагає нам легко захищатися від XSS та CSRF атак. Найпростіший спосіб захистити своїх користувачів від атак сценаріїв між сайтами - це забезпечити, щоб ви дозволяли LWC маніпулювати DOM, коли це можливо, і уникати ручного маніпулювання DOM. LWC та захисна оболонка Lightning Locker забезпечують найкращу санітарну обробку DOM у цій галузі. Однак це означає, що ви повинні залишити LWC, щоб маніпулювати власною DOM.

Спиратися на LWC для маніпуляцій DOM просто. По-перше, уникайте використання анотації lwc: dom = ”manual”, де це можливо. Ще одна найкраща практика, якої потрібно дотримуватися, коли вам потрібно прив’язати дані JavaScript до свого інтерфейсу, скористатися анотаціями LWC, такими як @api та @track. Коли вам потрібно переглядати список елементів, використовуйте ітератори LWC замість того, щоб генерувати DOM-вузли в JavaScript.

2.6 Серверна безпека у Salesforce

Apex - це власна мова, розроблена Salesforce.com. Згідно з офіційним визначенням, Apex - це строго типізована об'єктно-орієнтована мова програмування, який дозволяє розробникам виконувати оператори управління потоком і транзакціями на сервері платформи Force.com в поєднанні з викликами API-інтерфейсу Force.com.

Вона має Java-подібний синтаксис і діє як збережені процедури бази даних. Мова дозволяє розробникам додавати бізнес-логіку до більшості системних подій, включаючи натискання кнопок, поновлення пов'язаних записів і сторінки Visualforce.

Коли ви використовуєте Apex, безпека вашого коду є критично важливою. За замовчуванням класи Apex мають можливість читати та оновлювати всі дані в організації. Тому надзвичайно важливо дотримуватися правил спільного використання, встановлювати дозволи на об’єкти та поля та захищати від CRUD та FLS. Правила спільного використання Apex використовуються для визначення “контексту виконання”, згідно з яким виконується ваш код. Вам потрібно буде визначити, який код слід запускати як “системний режим” - тобто з привілеями доступу до багатьох ресурсів - і який код запускати як “користувацький режим”, в якому дозволи, захист на рівні поля та спільний доступ застосовуються правила поточного користувача.

Розробники, які використовують Apex, повинні переконатись, що вони ненавмисно не викривають конфіденційні дані, які зазвичай будуть приховані від користувачів за дозволами користувачів, захистом на рівні поля або за замовчуванням для всієї організації. Слід звернути ретельну увагу на веб-служби, які можуть бути обмежені дозволами, але виконуватися в контексті системи після їх запуску.

Apex зазвичай працює в системному контексті. У системному контексті код Apex має доступ до всіх об’єктів і полів - дозволи об’єктів, захист на рівні поля та правила спільного доступу не застосовуються до поточного користувача. Ця стратегія гарантує, що код не запуститься через приховані для користувача поля або об’єкти.

Однак правила обговорення не завжди обходять: Клас повинен бути оголошений ключовим словом без спільного доступу, щоб гарантувати, що правила спільного використання не застосовуються. Здебільшого системний контекст забезпечує правильну поведінку для системних операцій, таких як тригери та веб-служби, які потребують доступу до всіх даних в організації. Однак ви також можете вказати, що певні класи Apex повинні застосовувати правила спільного використання, які застосовуються до поточного користувача.

Є три ключові слова, які слід пам'ятати для правил спільного використання. Ви використовуєте параметри "with sharing" чи "without sharing" ключових слів для класу, щоб вказати, чи повинні застосовуватися правила спільного доступу. Ви використовуєте "inherited sharing" ключове слово спільного доступу в класі Apex для запуску класу в режимі спільного використання класу, який його викликав.

Щодо баз даних, то взаємодія з ними у мові використовується за допомогою мови запитів SOQL. Мова запитів дуже схожа нам звичайний SQL, але використання у коді є легшим і зразу приводиться до об'єктів, що є більш зручним для роботи з ними.

Ще однією перевагою до SQL є стійкість до ін'єкцій. Для цього є декілька способів захиститися від них:

- Статичні запити зі змінними прив'язки;
- String.escapeSingleQuotes ();
- Заміна символів;
- Перелік дозволених символів.

Використання цих технік допоможе нам захиститися від ін'єкцій, що забезпечить повну безпеку від них, а використання модифікаторів доступу забезпечить нам вірну роботу прав розмежування доступу.

Висновки за розділом 2

У цьому розділі ми розглянули найважливіші загрози, які можуть виникнути під час використання CRM – систем. Описавши їх, а також використавши інформацію з попереднього розділу, ми розглянули інструментарій, який пропонує нам Salesforce для боротьби з цими ризиками.

Ми описали безпеку на кожному рівні, так ми розглянули розмежування доступу у першому пункті, у другому розглянули моніторинг та відновлення даних у системі. У інших пунктах ми розглянули безпеку програмного коду, які інструменти та мови програмування використовуються, який у них захист, а також рекомендації до написання безпечного і потужного коду.

Аналізуючи можливості рішення, ми впевнилися, що використання системи є доцільним для побудови інформаційно безпечного рішення, а ті техніки з якими ми ознайомилися будуть у нагоді нам, при побудові нашого власного додатку.

РОЗДІЛ 3
ПОБУДОВА БЕЗПЕЧНОГО CRM – ДОДАТКУ НА БАЗІ ПЛАТФОРМИ SALESFORCE

3.1 Загальний опис додатку

Як ми і описували раніше, системи управління відносинами з клієнтами є надзвичайно потужними інструментами для роботи підприємства і допомагають вирішувати багато проблем, які можуть виникнути при збільшенню обсягу виробництва.

Припустімо, що у нас є компанія по виробництву та продажу якоїсь продукції. Підприємство є досить розвиненим і має великий обсяг продукції, яка виробляється. То ж яка потреба є у побудові власного додатку? Я би описав такі пункти:

1. Полегшення замовлення товарів для користувачів. Клієнти, в нашому випадку це власники магазинів, супермаркетів, тощо, які займаються розповсюдженням нашої техніки. Побудова додатку, для них, в першу чергу це полегшення замовлення продукції до пари натискань.

2. Полегшений процес відправлення. Маючи систему управління відносин з клієнтами, підприємство може набагато легше обробляти запити, а також швидше передавати їх між відділами, таким чином покращуючи ефективність.

3. Збір статистики для підприємства. Для підприємства, отримувати чітку статистику про свій продукт є одним з найважливіших чинників успішного функціонування. При побудові CRM – системі, ми можемо легко автоматизувати цей процес.

4. Побудова безпечного процесу обробки даних. Автоматизувавши всі процеси ми можемо забезпечити безпечну її обробку, що і є нашою найважливішою задачею.

То ж впевнившись в доцільності побудови та використання нашого рішення, можемо перейти до його опису.

Для нашого підприємства існують такі типи користувачів :

- Адміністратори системи;

- Менеджери продажів;
- Клієнти (власники магазинів).

Головною функцією нашого додатку є обробка запиту від клієнта, передача цього запиту потрібному менеджеру по продажам і від нього передача далі вже повністю обробленого замовлення до транспортного відділу. Адміністратори існують для контролю та моніторингу бізнес процесів, а також інформаційної безпеки.

Із особливостей можна виділити, що залежно від типи користувача він буде бачити різні компоненти додатку, крім того, кожен менеджер та клієнт буде мати свою територію до якої він буде прив’язаний. Територія, звісно, залежить від географічного розташування користувача, наприклад, наше підприємство може працювати в усіх областях країни, або навіть у декількох країнах. Тому, до кожної території буде підкріплений менеджер, який буде опрацьовувати замовлення, ну і звісно кожен клієнт буде пов’язаний з територією де він знаходиться і його замовлення буде йти до потрібного менеджеру. Таким, чино ми зможемо покращити ефективність роботи з запитами і розмежувати дані між користувачами.

3.2 Характеристика додатку

Маючи загальний опис, ми можемо перейти до характеристики додатку. Обравши в минулих розділах платформу Salesforce та впевнившись в доцільності її використання, ми будемо розробляти рішення на її базі. За допомогою використання цієї платформи, ми отримуємо перевагу в використанні її можливостей, бо Salesforce як система, надає нам можливість використовувати вже готові компоненти та інструменти для побудови додатку, так і будувати свої компоненти для вирішення спеціальних потреб.

Як ми описували, у попередньому пункті, із-за розмежування доступу, у кожного типу користувачів буде доступ до певних частин додатку та компонентів. Таким чином можна побудувати таблицю, яка буде показувати залежність типу користувачів від доступу до функцій додатку.

Таблиця 3.1

Можливості користувачів системи

Тип користувача	Доступ до яких функцій має
Клієнт	Має доступ до перегляду таблиці продуктів, додавання продуктів до кошику, створення замовлення, редагування замовлення
Менеджер по продажам	Має доступ до перегляду замовлень пов'язаних з його територією, може скасувати або відправити замовлення до транспортного відділу, може додавати нові продукти до списку
Адміністратор	Має усі можливості описані вище, крім того може виконувати моніторинг подій у системі, має доступ до змін в базі даних, налаштуваннях компонентів тощо.

Розмежування доступу буде реалізоване за допомогою інструментів описаних у другому розділі, наприклад, тип користувача буде дорівнювати профілю, доступ до записів буде реалізований за допомогою доступу до об'єктів.

Також присутнім буде і доступ до полів на продуктах. Клієнт не зможе бачити кількість продуктів, яка є наявною на складах в даний момент. А доступ до записів за територіями, буде реалізований за допомогою правил обміну.

Адміністратор же буде мати повний доступ до консолі адміністратора, зможе змінювати налаштування та контролювати правильну роботу додатку. Для функціонування додатку буде створені також і кастомні компоненти за допомогою програмних застосунків, які будуть написані теж на інструментарії Salesforce. При написанні програмного коду, ми використаємо методи захисту та рекомендації описані у попередніх розділах.

3.3 Побудова об'єктів та полів бази даних, а також розмежування доступу до них

У попередніх розділах ми описували можливості Salesforce як готової системи, розглядали, які вбудовані можливості існують, а також що вона нам надає для розробки власних компонентів системи. Мабуть, основними речами з інструментарію є вбудовані засоби розмежування доступу. Але спочатку, слід розглянути більш докладно як система працює з базою даних.

У Salesforce представлений, мабуть, найзручніший інструмент роботи з базою даних, це об'єктна модель роботи з таблицями. Таким чином, у нас є об'єкт – таблиця в базі даних, поле – стовпці в таблиці, а також записи – рядки в таблиці. Така модель полегшує роботу з базами даних з коду та в ручний спосіб. В Salesforce є безліч вбудованих об'єктів з якими ви можете починати працювати одразу, кастомізувати їх тощо, а також штучні або кастомні об'єкти, це ті, які користувачі (адміністратори) можуть створювати самі. До того ж, ви можете їх пов'язувати між собою, а з коду працювати і з тим, і з тим типом. Різниця між ними в тому, що до закінчення в назві об'єктів, які ми створили самі, завжди додається «__c».

Розглянувши те як побудована база даних у системі, слід зазначити, що на рівні кожного компоненту можна працювати з розмежуванням доступу. Ми вже досліджували, як побудований інструментарій на кожному рівні у другому розділі, тому зупинятися на цьому не будемо, а опишемо, як буде організована база даних та які саме функції ми використаємо.

Маючи підприємство, яке виготовляє продукцію електроніки, нам потрібен об'єкт «Продукт». Зрозуміло, що у кожного продукту буде існувати певна кількість, ціна, назва, зображення для клієнтів тощо.

Також для покупки продукту нам потрібен об'єкт «Замовлення», а також «Продукт замовлення». З замовленнями все зрозуміло, ми матимемо поле Адреси, Країни, Статус, також матимемо поле, яке буде показувати хто створив замовлення (створено автоматично системою). «Продукт замовлення» ж більш складний, він означає продукти, які клієнт буде додавати до замовлення, тобто для кожного такого запису, буде вказана кількість продукту, посилання на продукт, а також посилання на продукт до якого він прикріплений.

Таким чином, ми можемо побудувати таку таблицю, яка буде показувати:

- об'єкт,
- його назву у системі,
- поля,
- с ким він зв'язаний.

Таблиця 3.2

Організація бази даних

Об'єкт	**Назва у системі**	**Поля**	**Об'єкти з якими зв'язаний**
Продукт	ProductItem__c	• Кількість (Quantity__c) • Ціна (Price__c) • Назва (Name) • Зображення (Image__c)	InvoiceLineItem__c (Продукт замовлення)
Замовлення	Invoice__c	• Адреса (Address__c) • Країна (Country__c) • Статус (Status__c) • Створено користувачем (CreatedBy)	User (Користувач) InvoiceLineItem__c (Продукт замовлення)
Продукт замовлення	InvoiceLineItem__c	• Посилання на Продукт (ProductItem__c) • Посилання на Замовлення (Invoice__c) • Кількість (Quantity__c)	Invoice__c (Замовлення) ProductItem__c (Продукт)

Слід зазначити, що крім об'єктів розміщених у таблиці, в базу даних в об'єкт User (Користувач) нам потрібно додати поле Країна (Country__c). За цим полем ми будем розмежовувати, які записи буде бачити кожен менеджер залежно від країни.

Тепер, коли ми розглянули організацію бази даних, об’єктів та їх полів, ми можемо розглянути, як саме це робиться в системі Salesforce. Однією з головних зручностей системи є те, що ми можемо створити всі об’єкти та полі вручну, використовуючи інтерфейс. На наступних рисунках 3.1 – 3.4, будуть показані приклади створення.

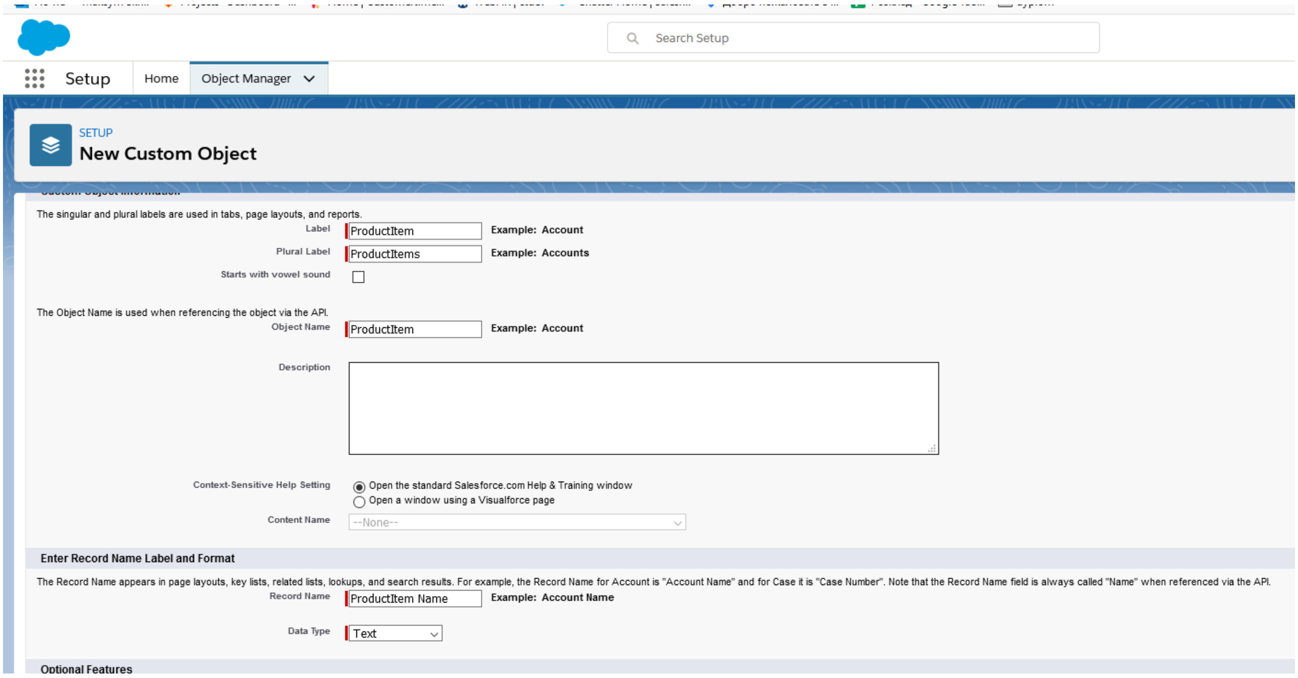

Рисунок 3.1 Створення кастомного об’єкту ProductItem

Такі ж налаштування ми використовуємо і для інших об’єктів, після цього ми можемо перейти до створення полів. Перед створенням слід зазначити, що для кожного поля існують типи, їх досить багато, саме обрання типу для поля ми можемо побачити на рисунку 3.3.

Після обрання типу ми переходимо до самого опису поля, де ми вказуємо його назву, чи є це поле обов’язковим або унікальним, а також деякі налаштування, які залежать від типу поля. Приклад створення текстового поля ми можемо побачити на рисунку 3.4.

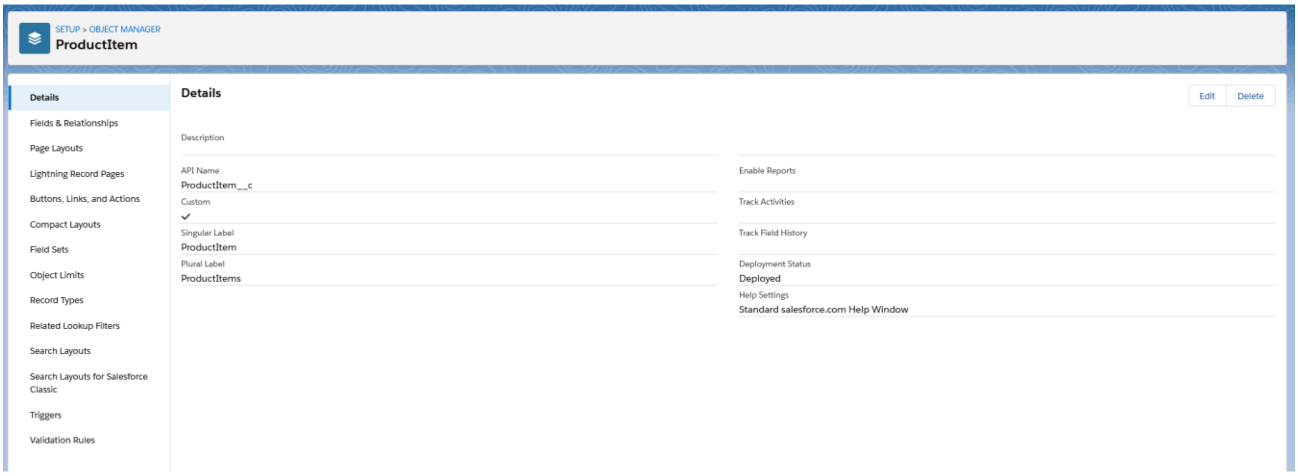

Рисунок 3.2 Сторінка об’єкту ProductItem

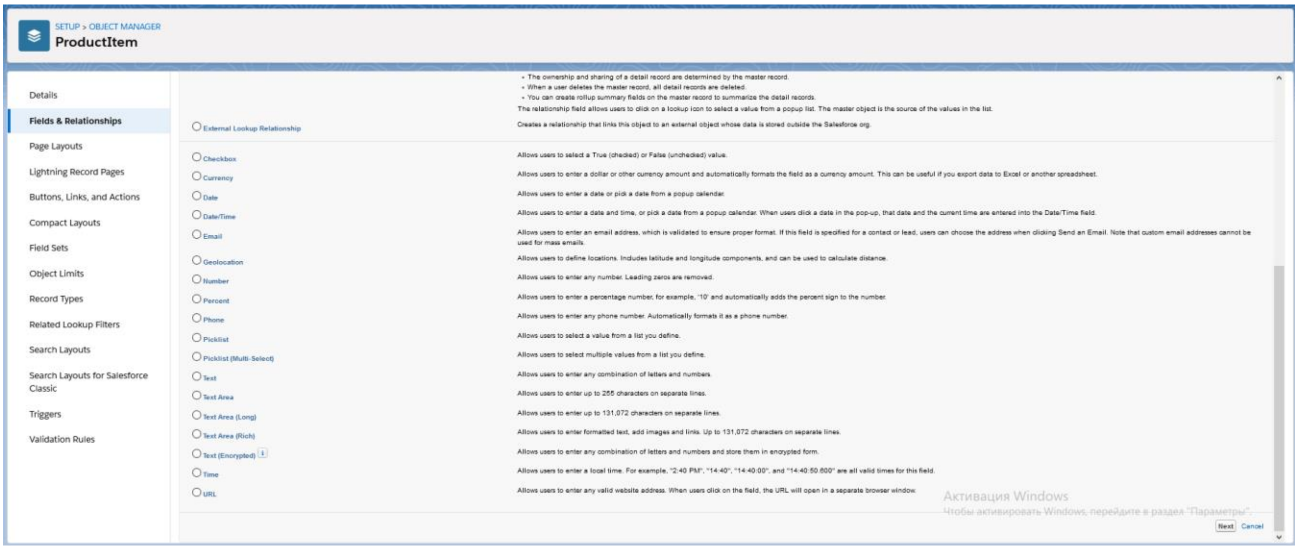

Рисунок 3.3 Сторінка обрання типу поля об’єкту

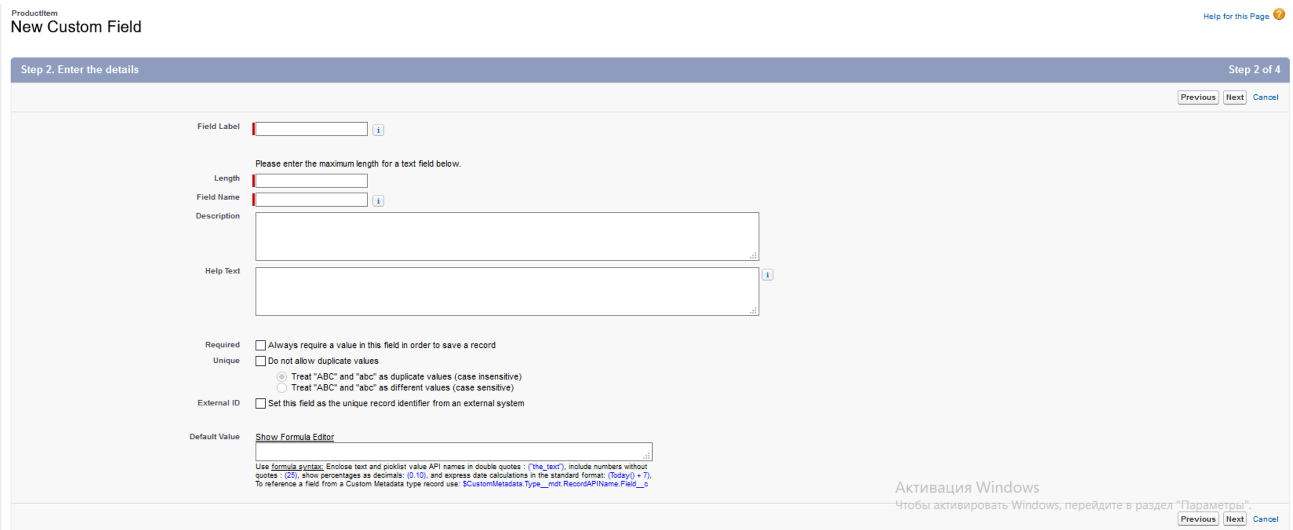

Рисунок 3.4 Приклад створення текстового поля

Тепер коли ми створили об'єкти і полі в базі даних, ми можемо розглянути розмежування доступу до цих об'єктів. В минулому пункті цього розділу ми вже розглянули типи користувачів та залежність їх до певних функцій. Тому використавши матеріали з другого розділу, ми можемо перенести це в нашу систему.

Коли ми описували інструментарій який є наявним у системі, я розповідав про те, що доступ до даних створюється на різних рівнях. Всього їх три:

- Рівень доступу до об'єкту;
- Рівень доступу до полів;
- Рівень доступу до записів.

В нашому проекті нам знадобляться здебільшого два рівні: до об'єктів та до записів. Звісно, залежно від додатку який ми проектуємо, ми можемо використовувати усі рівні, які є дуже потужними, але в нашому випадку ми маємо три типи користувачів, що дорівнюють профілям в Salesforce та існують на рівні доступу до об'єктів. В рівні доступу до записів ми використаємо «Sharing rules» або «Правила обміну». Правила обміну будуть використовуватися до записів замовлень для тих користувачів (менеджерів), які знаходяться в такій же країні, як і та що вказана в замовлені, а також для користувачів (клієнтів), які будуть бачити тільки ті замовлення, які створили самі.

Таблиця 3.3

Профілі у додатку

Тип користувача	**Назва профілю**	**До яких об'єктів надано доступ**
Клієнт	Customer	ProductItem__c (тільки на перегляд) Invoice__c
Менеджер з продажів	Product Manager	ProductItem__c, Invoice__c
Адміністратор	System Administrator (системний профіль)	Має доступ до усіх об'єктів у системі

Описавши таблицю профілів, ми можемо зробити схожу і для правил обміну:

Таблиця 3.4

Таблиця правил обміну для додатку

Тип користувача	**Об'єкт до якого буде задіяний**	**Поле за яким буде працювати**
Менеджер з продажів	Invoice__c	Country__c
Клієнт	Invoice__c	CreatedBy

Після того, як ми розглянули організацію розмежування доступу до даних, ми можемо реалізувати це в системі. Як і в роботі з базою даних, всі налаштування можна проводити вручну, хоча також з інструментарієм можна працювати і із коду.

Спочатку налаштуємо профілі. Профіль адміністратора вже існує в системі і має назву «System Administrator». Він є стандартним, тому використаємо саме його. Для двох інших ж профілів ми створимо два кастомних профілі, приклад створення яких можемо побачити на рисунку.

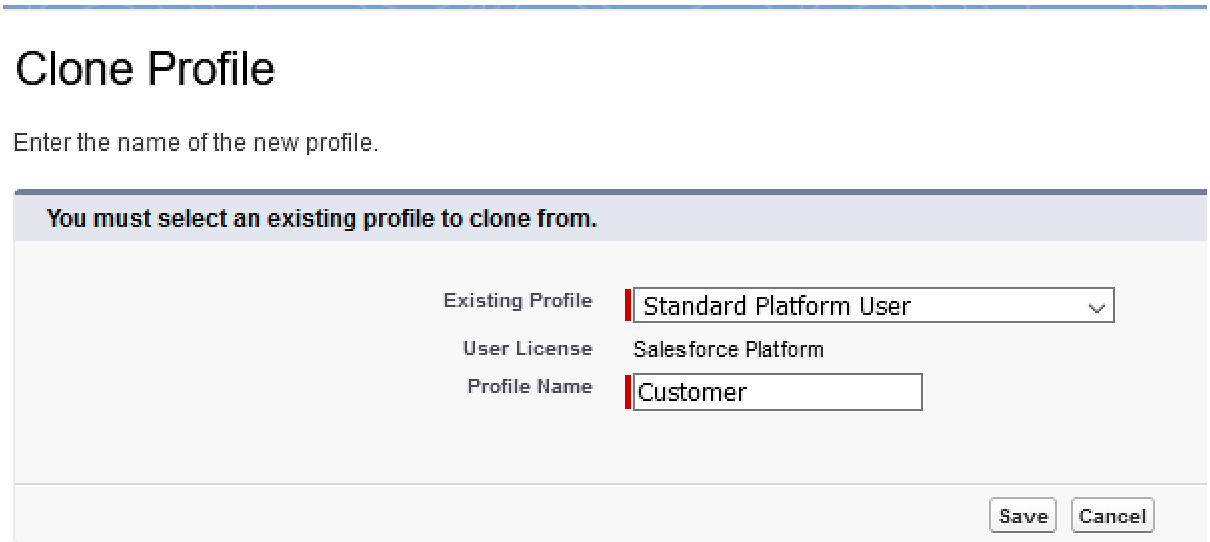

Рисунок 3.5 Приклад створення профілю

Після створення самого профілю ми можемо почати налаштування доступу до різних об'єктів, сторінку налаштувань можна побачити на рисунку 8.

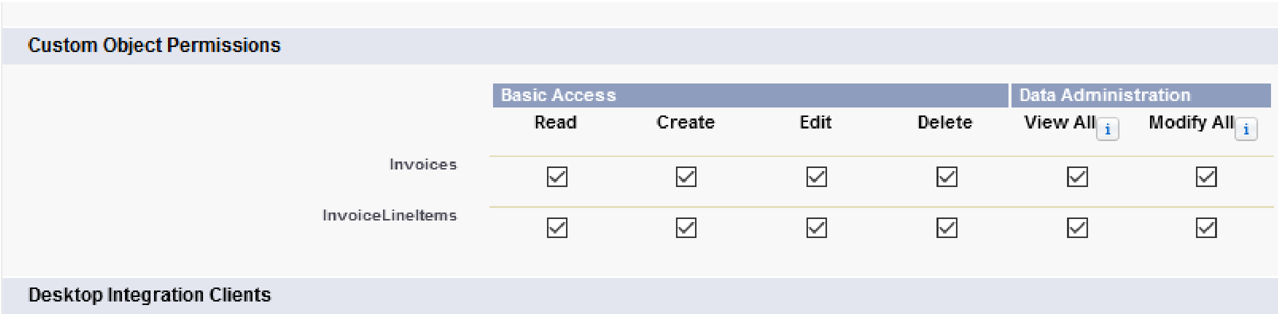

Рисунок 3.6 Приклад налаштування доступу профілю до об’єкту

Звісно, показаний фрагмент на рисунку є лише малою частиною того, що можна налаштувати в профілі, все це ми вже описували в другому розділі, але для нашого додатку поки що вистачить саме цього.

Після налаштувань профілів, почнемо працювати з правилами обміну. При формуванні таблиці ми зазначили об’єкти, щодо яких відносяться правила, це тому що правила обміну встановлюються не на користувачі, а на об’єкті, доступ до записів якого потрібно налаштувати. Приклад створення правила можна побачити на рисунку 9.

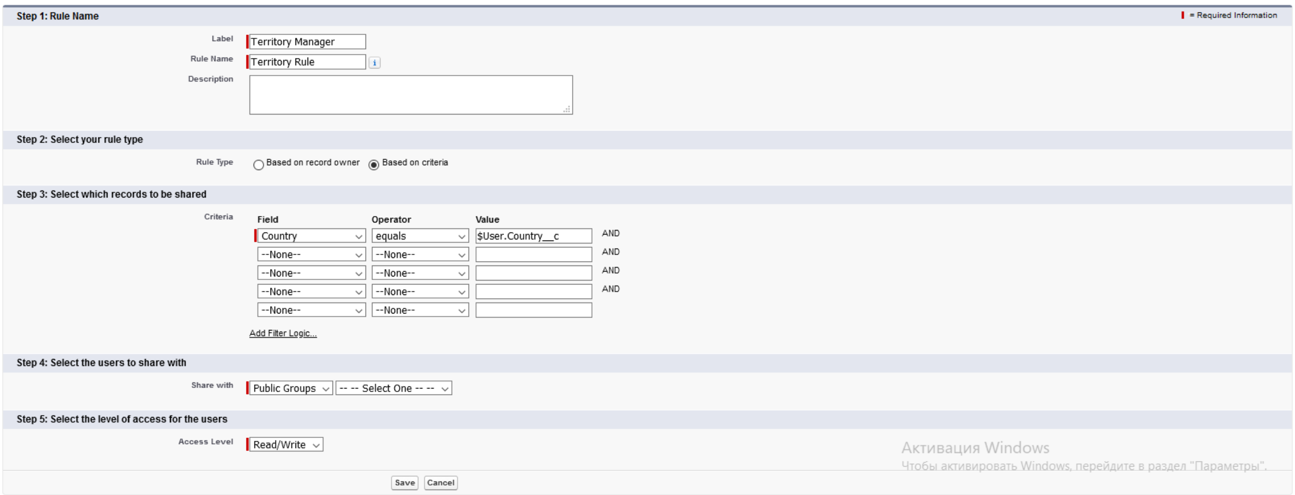

Рисунок 3.7 Приклад створення правила обміну

Таким чином, налаштувавши профілі та правила обміну ми створили розмежування доступу до даних, після цього ми можемо створити користувачів для кожної ролі і розмежування прав буде працювати. Але тепер нам потрібно створити компоненти для роботи з даними.

3.4 Розробка коду для компонентів додатку

У попередніх розділах ми описували систему Salesforce та на її базі розглянули можливість налаштувань компонентів для роботи з даними у системі. Тоді, у другому розділі, ми описували, що у системі наявна вже велика кількість готових потужних компонентів не тільки для відображення даних, а й для їх налаштування, для роботи зі звітами тощо. Також ми розглянули можливість розробки своїх компонентів.

Розробку для системи Salesforce ми можемо умовно поділити на дві частини: Back-end та Front-end. За назвою зрозуміло, що Back-end - це серверна розробка, тобто код, який працює з базою даних та маніпулює процесами, а Front-end займається клієнтською розробкою і відповідає здебільшого за відображення даних, а також маніпулювання ними на стороні клієнту (браузеру). Ми в цьому пункті теж поділимо розробку коду на дві умовних частини, і розповімо про них окремо.

Почнемо з серверної частини. Так, як основна функція наших компонентів буде робота з базою даних, ми можемо пов’язати створення класів з кожним об’єктом. В другому розділі ми вже описували мову Apex за якою працює наша система, тому нам відомо, що класи є конструкціями для побудови наших програм. В цьому проекті ми використаємо два типи класів, насправді це поділення теж умовне, бо відрізнятися вони будуть лише своїм функціоналом. Перший тип – це класи сервіси, вони будуть містити в собі функції (методи) для роботи з базою даних, тобто вибірки даних та збереження. Другий тип – контролери, вони будуть працювати з класами-сервісами та передавати дані у клієнтські компоненти.

Таким чином ми можемо утворити таблицю, в якій будуть знаходиться наші класи.

Створивши таблицю класів (Таблиця 3.5), ми можемо побачити, що їх назви пов’язані з назвами об’єктів бази даних, які ми створили у попередньому пункті.

Після того, як ми розглянули в цілому серверну частину, ми можемо перейти до реалізації.

Таблиця 3.5

Список серверних класів

Назва класу	**Тип класу**	**Що клас робить**
ProductItemService	Сервіс	Здійснює вибірку даних об’єкту Продукти
InvoiceService	Сервіс	Здійснює вибірку даних об’єкту Замовлення, створює нові записи об’єкту Замовлення у базі даних
InvoiceController	Контролер	Зв’язує InvoiceService клас з клієнтськими компонентами
ProductListController	Контролер	Зв’язує ProductListController клас з клієнтськими компонентами

Спочатку реалізуємо сервіси, для цього створимо два класи ProductItemService та InvoiceService. Перший клас буде містити метод getProductList який буде робити запит до бази даних та повертати список продуктів. Код буде мати такий вигляд:

```
public with sharing class ProductItemsService {

    public List<ProductItem__c> getProductList() {
        return [
            SELECT Id, Name, PriceOfItem__c, ImageUrl__c
            FROM ProductItem__c
            WHERE Quantity__c > 0
        ];
    }

}
```

Рисунок 3.8 Код класу ProductItemsService

Другий клас буде дещо складніший, бо крім вибірки даних, він матиме метод, який буде відповідати за збереження нового замовлення. Як параметри до нього будуть приходити нове замовлення та список продуктів замовлення. Код класу матиме такий вигляд:

```
public with sharing class InvoiceService {

    public List<Territory__c> getAvailableTerritories() {
        return [
            SELECT Id, Name
            FROM Territory__c
        ];
    }

    public void saveInvoice(Invoice__c invoice, List<InvoiceController.ProductItemWrapper> p
roductList) {
        List<InvoiceLineItem__c> lineItems = new List<InvoiceLineItem__c>();
        for(InvoiceController.ProductItemWrapper item : productList) {
            InvoiceLineItem__c newItem = new InvoiceLineItem__c();
            newItem.QuantityProductItems__c = item.quantity;
            newItem.ProductItem__c = item.Id;
            newItem.Invoice__c = invoice.Id;
            lineItems.add(newItem);
        }
        insert lineItems;
    }
}
```

Рисунок 3.9 Код класу InvoiceService

Після написання коду, треба докладніше розібрати його. В обох класах ми маємо модифікатори доступу “with sharing”. Цей модифікатор відповідає за те, що клас працює у контексті користувача, тобто повністю підпорядковується правилам розмежування доступу. Більше про модифікатори доступу ми розповідали у другому розділі. Всі вибірки даних виконуються за допомогою SOQL запитів, використання таких запитів дозволяє нам захиститися від ін’єкцій, до того ж такий запит повертає вже готовий об’єкт, тому нам не потрібно працювати з сереалізацією, що допомагає безпечності нашого коду.

Далі слід звернути увагу на рядок “insert lineItems”, він відповідає за збереження продуктів замовлення, а конструкція “insert *variable*” є зручним вбудованим синтаксисом для збереження записів у базу даних. Також у класі наявний код зв’язаний з заповненням полів у новому записі бази даних.

Після створення сервісів, ми можемо перейти до створення контролерів. Їх у нас буде теж 2, перший буде мати назву ProductListController та відповідатиме за зв’язок ProductItemService та компоненту списку продуктів, другий матиме назву

InvoiceController та зв'язуватиме компоненти замовлення та клас InvoiceService. Код класу ProductListController буде мати такий вигляд:

```
public with sharing class ProductListController {

    private static ProductItemsService service = new ProductItemsService();

    @AuraEnabled
    public static List<ProductItem__c> getProducts(){
        try {
            return service.getProductList();
        } catch (Exception e) {
            throw new AuraHandledException(e.getMessage());
        }
    }

}
```

Рисунок 3.10 Код класу ProductListController

Код не дуже сильно відрізняється від коду класу сервісів, маємо такий же модифікатор доступу, в методі getProducts викликається метод getProductList, який ми реалізували в сервісі трохи раніше. Звернути увагу слід лише на анотацію над методом «@AuraEnabled». Вона означає видимість методу до клієнтських компонентів. Таким чином ми можемо самі обирати, які методи ми можемо використовувати з клієнтського коду, а які ні.

Після написання серверного коду, можемо почати написання клієнтських компонентів. Як ми і описували раніше у другому розділі, для написання візуальних компонентів в Salesforce існує фреймворк LWC (Lightning Web Component). Також там же ми описали його можливості та стійкість до різних типів атак. Тому ми можемо одразу взятися за їх написання, але спершу побудуємо таблицю (Таблиця 3.6), де покажемо усі потрібні нам компоненти та їх функції.

Клієнтські компоненти міститимуть трохи більше коду ніж серверні. Це пояснюється тім, що в компоненті крім коду, ще наявна і розмітка. Тобто наші компоненти будуть мінімально складатися з Html та JavaScript файлів. Повний код компонентів ми можемо спостерігати у Додатку Б, а в цьому пункті ми розглянемо як приклад реалізацію одної з компонент – invoiceForm.

Таблиця 3.6

Візуальні компоненти та їх функції

Назва компоненту	**Функція**
productList	Відображення списку продуктів
productItem	Відображення одного продукту
cart	Відображення замовлення
cartItem	Відображення одного продукту замовлення
invoiceForm	Форма заповнення замовлення, а також його підтвердження

Як ми описували у попередніх розділах, в Salesforce для написання додатків, а саме клієнтських компонентів, ми маємо безліч вже написаних частин, функцій та тегів для використання у нашому коді. Крім, того що це скорочує написання програм, ми ще й отримуємо стійкі до більшості типів загроз компоненти, що є найголовнішим для нас у проектуванні нашого додатку. Більше про це ми вже писали у другому розділі, коли розглядали Lightning Web Components Framework.

Тому використовуючи такий підхід ми можемо написати наш код розмітки:

В коді розмітки ми використали деякі вже написані компоненти, так наприклад, lightning-layout та lightning-layout-item відповідають за розташування різних складових на сторінці, а lightning-input ми використали для створення поля заповнення адреси. Звісно ми могли використати і стандартні компоненти html-розмітки, але це було б небезпечно для нашого додатку, і для забезпечення захисту до загроз потрібно було би писати свій власний код.

```
<template>

    <lightning-layout multiple-rows="true">

        <lightning-layout-item class="slds-p-around_medium" size="12">

            <lightning-combobox
                label="Territory"
                options={selectOptions}
                value={territory}
                onchange={handleTerritory}
            ></lightning-combobox>

        </lightning-layout-item>
```

Рисунок 3.11 HTML код компоненту invoiceForm

```
            <lightning-layout-item class="slds-p-around_medium" size="12">

                <lightning-input
                    label="Address"
                    value={address}
                    onchange={handleAddress}
                ></lightning-input>

            </lightning-layout-item>

            <lightning-layout-item class="slds-p-around_medium" size="12">
                <lightning-button
                    variant="brand"
                    label="Book products"
                    onclick={bookProducts}
                ></lightning-button>
            </lightning-layout-item>

        </lightning-layout>


    </template>
```

Рисунок 3.12 Продовження HTML коду компоненту invoiceForm

Далі ми реалізуємо код JavaScript файлу:

```
import { LightningElement, api, track } from 'lwc';
import getTerritories from '@salesforce/apex/InvoiceController.getTerritories';
import saveRecord from '@salesforce/apex/InvoiceController.saveRecord';

export default class InvoiceForm extends LightningElement {
    @api
    products;
    @track
    territories = [];
    territory;
    address = '';

    get selectOptions() {
        let options = [];
        this.territories.forEach(item => {
            options.push({
                label : item.Name,
                value : item.Name
            });
        });
        return options;
    }
```

Рисунок 3.13 JavaScript код компоненту invoiceForm

```
async getTerritories() {
        try{
            const response = await getTerritories();
            this.territories = response;
        }
        catch(error) {
            console.log(error);
        }
    }

    handleTerritory(event) {
        this.territory = event.detail.value;
    }

    async bookProducts() {
        try{
            await saveRecord(
                {
                    productsJSON : JSON.stringify(this.wrapProducts(this.products)),
                    territory : this.territory,
                    address : this.address
                }
            );
            this.refreshCartAfterSave();
        }
        catch(error) {
            console.log(error);
        }
    }

    wrapProducts() {
        let productList = [];
        this.products.forEach(element => {
            productList.push({
                ...element.item,
                quantity : element.quantity
            });
        });
        return productList;
    }
```

Рисунок 3.14 Продовження JavaScript коду компоненту invoiceForm

Код цього файлу дещо більший за розмітку і містить всю клієнтську логіку цього компоненту. Так, наприклад, у методі getTerritories ми отримуємо усі можливі країни з серверу, а у методі bookProducts відправляємо сформоване замовлення до нього у форматі JSON.

Повний код усіх компонентів можна побачити у Додатку Б, а ми спроектувавши код та налаштування нашого додатку, можемо перейти до його захисту.

3.5 Аналіз захисту програмного коду додатку

Після написання коду додатку, нам потрібно розібратися з його захистом. Ми вже описували у другому розділі, що якщо використовувати усі техніки описані там, не відмовлятися від використання готових компонентів, використовувати захищений DOM у LWC, використовувати SOQL запити до бази даних тощо, ми матимемо основний захист до більшості веб-загроз.

При написанні коду у попередньому пункті, ми використовували максимально техніки описанні у попередніх розділах. Спершу для серверного коду при написанні класів, ми використовували модифікатор доступу «with sharing». Таким чином код завжди працює у контексті користувача, тобто дані, які буде бачити користувач будуть відноситься лише до тих до яких він має доступ, так же і фунціонал буде доступний відносно доступу користувача.

Для зв'язку між клієнтською та серверною частиною використовується JSON. При використанні https (використовується за замовчуванням у Salesforce), ми отримуємо більшу захищеність ніж при використані, наприклад XML. До того ж швидкість JSON набагато вища.

При написанні клієнтських компонентів ми використали вбудовані компоненти LWC і як ми описували раніше, цех підхід допомагає нам в захисті від загроз типу XSS, CSRF тощо. В коді JavaScript ми також виконали перевірку полів, таким чином ми можемо впевнитися в валідації наших даних і в їх якості.

Таким чином, після розробки коду та налаштування розмежування доступу, ми отримали на виході цілком безпечний додаток.

3.6 Тестування розробленого додатку

Після розробки програмного коду та налаштування розмежування доступу, треба впевнитися що наші компоненти працюють вірно. Спершу ми маємо розглянути налаштування розмежування доступу до даних. Як ми пам'ятаємо з попередніх розділів у нас є три різних профілі з різним рівнем доступу. Таким чином наш додаток буде виглядати по різному для різних типів користувачів. На наступних рисунках можна побачити види домашньої сторінки для різних профілів. Адміністратор буде мати доступ до панелі адміністратора, менеджери з продажів будуть бачити список замовлень, а у клієнтів будуть видні компоненти для відображення товарів і форма замовлення. Таким чином ми створюючи користувачів для різних профілів можемо протестувати роботу розмежування доступу.

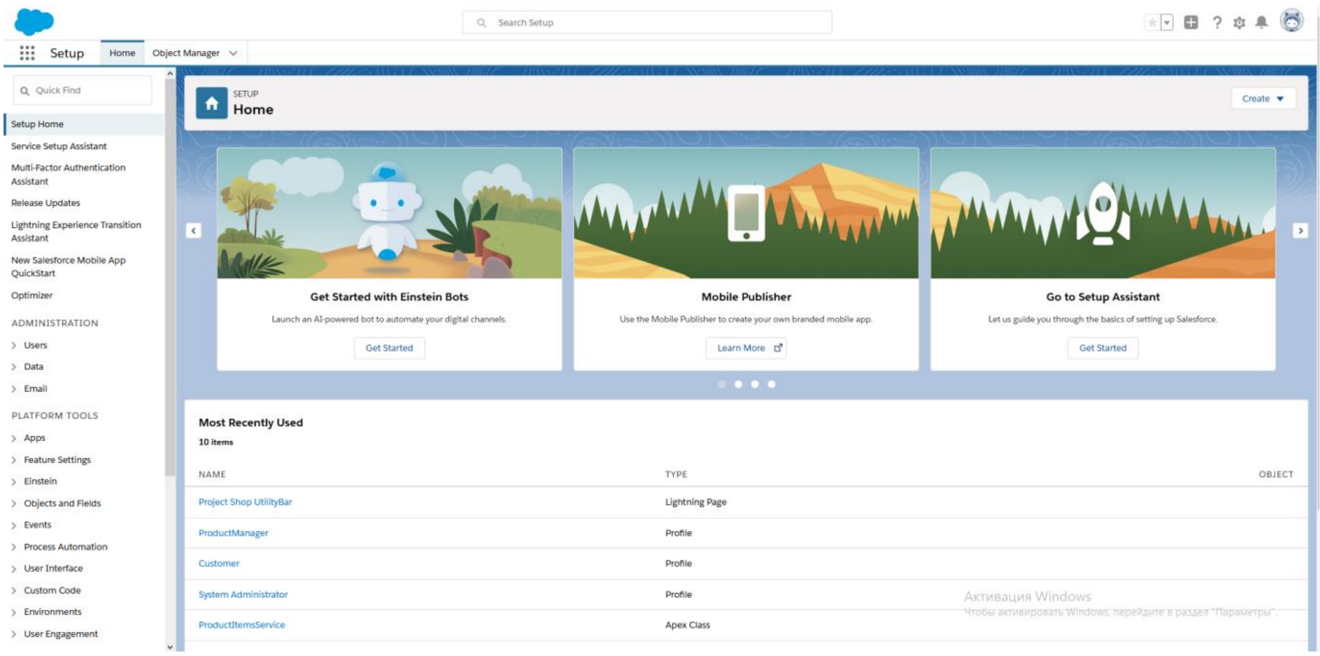

Рисунок 3.15 Панель адміністратора

На рисунку 3.15 ми можемо бачити панель адміністратора, тут ми можемо бачити різні інструменти за допомогою яких ми можемо контролювати роботу додатка, змінювати його, проводити моніторинг тощо. Доступ до панелі матимуть лише користувачі у яких профіль буде «System Administrator».

Для менеджеру з продажів додаток буде мати інший вигляд, бо цей тип користувачів має доступ лише до списку замовлень та продуктів. Тому в меню додатку він матиме лише два посилання як це видно на рисунку 3.16.

Список продуктів буде мати схожий вигляд. Якщо ми хочемо побачити більш детальну інформацію про продукт, ми можемо натиснути на нього і перейти на сторінку про продукт. Там ми побачимо компонент як на малюнку 3.17. Там будуть показані поля, до яких має доступ профіль користувача.

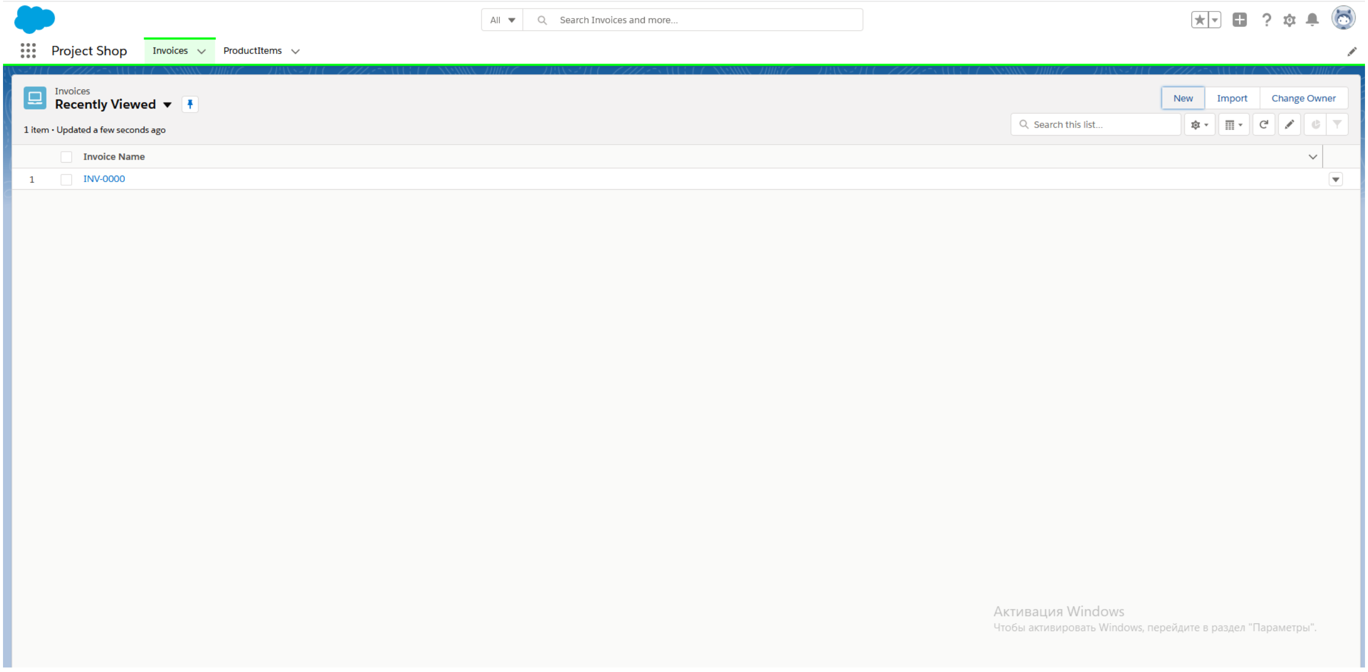

Рисунок 3.16 Вигляд списку замовлень

Слід зазначити, що так як система має досить багато готових компонентів, на рисунку вище ми використовуємо саме них. Їх також можна змінити при потребі, але якщо нам це не потрібно ми одразу можемо використовувати їх. Таким чином ми збільшуємо швидкість розробки додатку, що є досить великим плюсом.

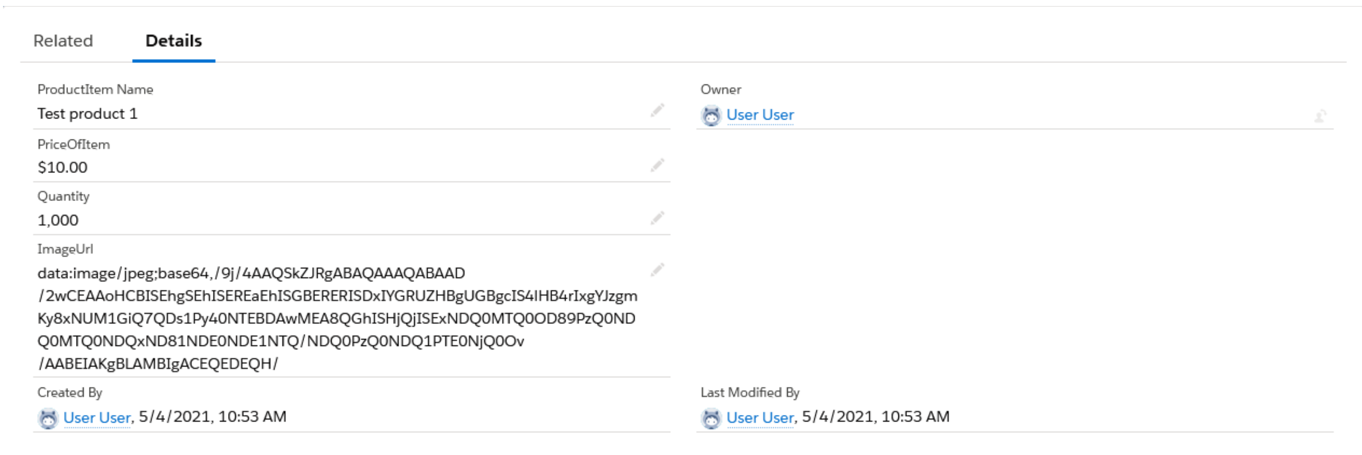

Рисунок 3.17 Приклад компоненту для детальної інформації

Далі ми можемо перейти до вигляду додатку, який бачить клієнт. В цілому у нього буде доступ лише до однієї сторінки, на якій будуть знаходиться розроблені нами раніше компоненти. Який вигляд має домашня сторінка для клієнта ми можемо побачити на малюнку 3.18.

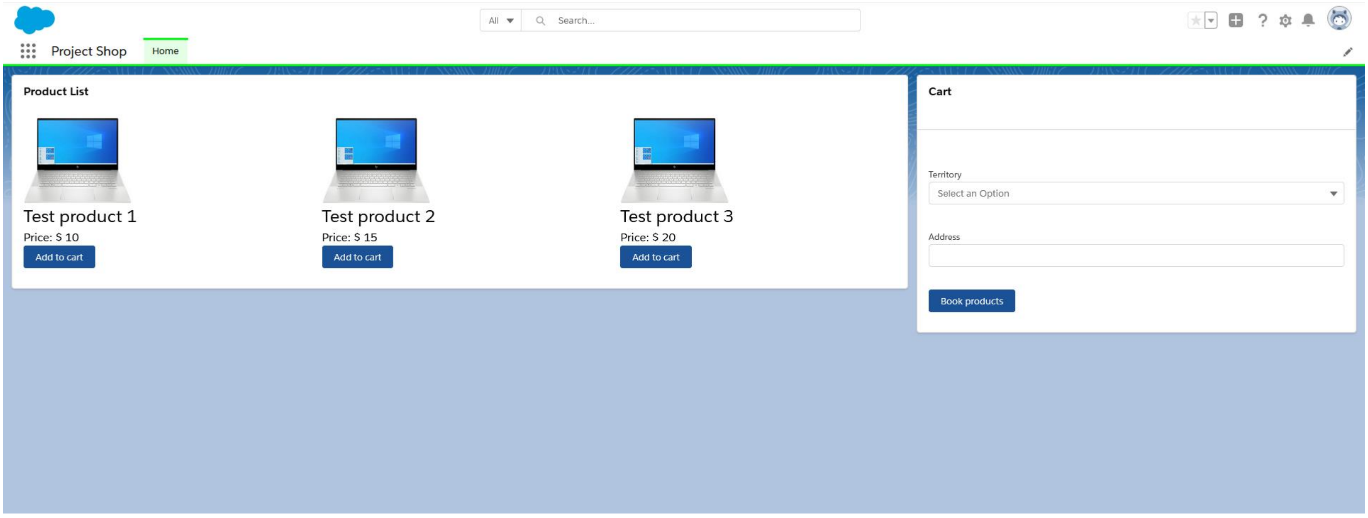

Рисунок 3.18 Приклад домашньої сторінки клієнта

Звісно, на рисунку показані тестові дані для компонентів, ми створили 3 тестових продукти і бачимо, що за допомогою компоненту написаного раніше ми можемо бачити їх на сторінці, а також додавати до замовлення. При додаванні декількох продуктів до замовлення і заповнення форми ми можемо натиснути на кнопку і замовлення автоматично створиться.

Але протестувавши вигляд ми протестували лише компоненти на клієнтській стороніта частково протестували розмежування доступу для неї. Крім цього ми ще писали серверний код. Його ми теж можемо протестувати за допомогою «Консолі розробника». Ми вже описували її у другому розділі. Це надзвичайно потужний інструмент для розробника, тому що за допомогою консолі ми можемо писати програмний код, дивитися логи, робити запити для бази даних і використовувати інструменти налагодження.

Таким чином за допомогою консолі ми можемо використати код написаний раніше і побачити що саме він нам поверне. До того ж консоль можна налаштувати, щоб вона працювала у контексті певного користувача, таким чином ми можемо протестувати доступ до даних з коду і чи працюють для нього правила розмежування. На рисунку 3.20 ми можемо побачити приклад написання коду для тестування класу.

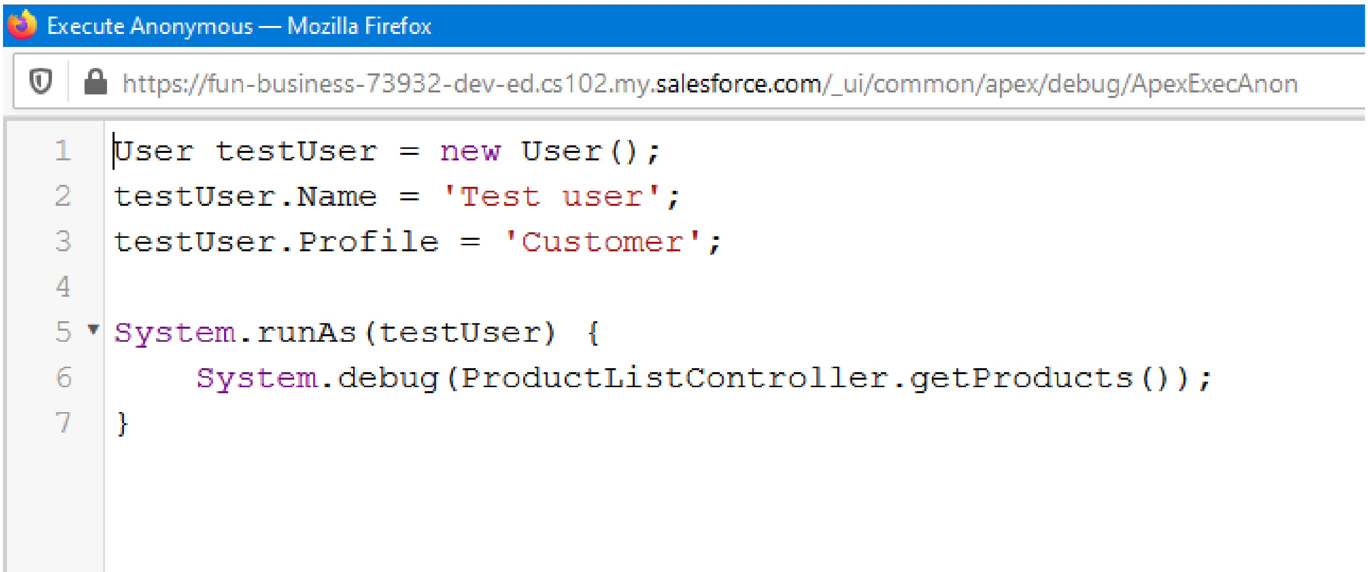

Рисунок 3.19 Приклад тестування класу ProductListController

На рисунку ми можемо побачити код в якому ми створюємо нового користувача і надаємо йому профіль «Customer» (Клієнт), а потім запускаємо код від його імені. Після запуску коду ми побачимо результат запуску у вигляді логів і там повинні будемо побачити 3 тестових продукти, як на малюнку 3.15.

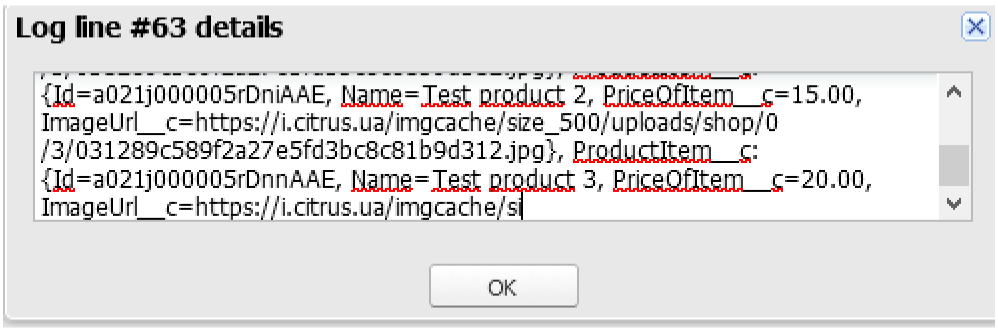

Рисунок 3.20 Результат запуску коду для тестування

Звісно, клас ProductListController можна було протестувати і за допомогою клієнтських компонентів, бо вони відображалися вірно і правила розмежування доступу до продуктів надають доступ усім користувачам незалежно від типу. Але ми маємо клас InvoiceController, який для клієнтів не має повертати записів, а для менеджерів має повертати лише ті замовлення які належать до їх території. То ж спочатку протестуємо код який запуститься для клієнту.

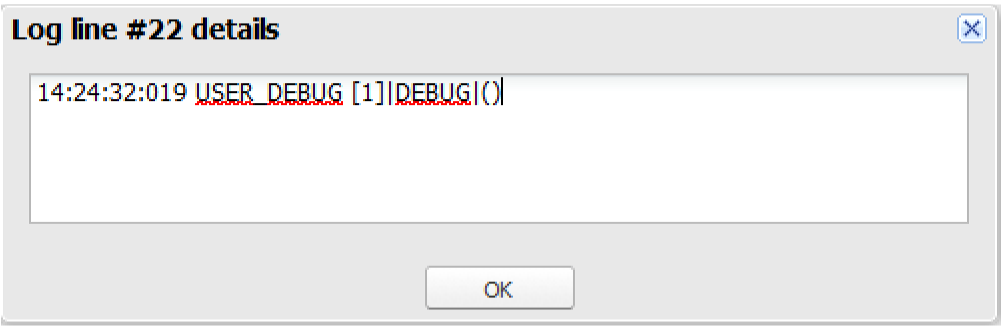

Рисунок 3.21 Приклад роботи класу InvoiceController для клієнту

Як результат бачимо пустий лог, таким чином ми впевнилися, що правила розмежування доступу працюють для клієнтів. Для менеджерів ж щоб протестувати нам потрібно через клієнтський компонент створити декілька замовлень на різні території, після чого менеджер буде здатний бачити лише ті записи з якими співпадає його територія.

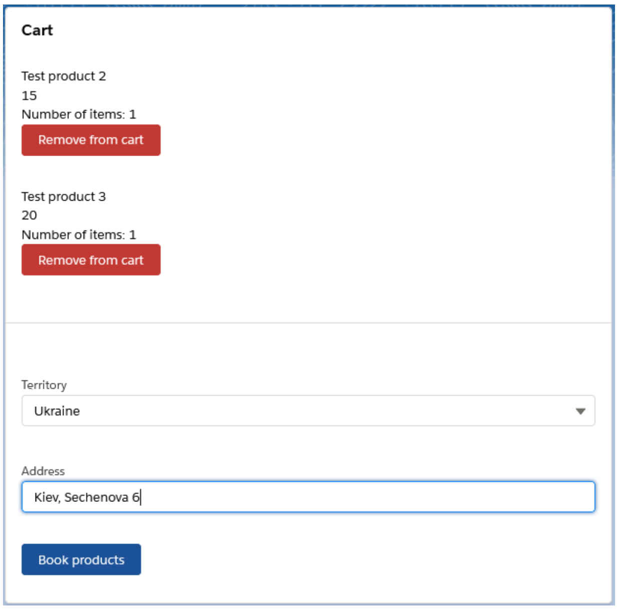

Рисунок 3.22 Приклад заповнення форми замовлення

На рисунку 3.22 ми бачимо заповнення форми замовлення, таким чином ми створюємо два замовлення, одне на Україну, інше для Польщі. Таким чином при створенні користувача з територією «Україна», він повинен бачити лише одне замовлення за правилами розмежування. Для цього напишемо такий код як на рисунку 3.23, а результат одразу буде видно на рисунку 3.24.

```
User testUser = new User();
testUser.Name = 'test user';
testUser.Territory__c = 'Ukraine';
testUser.Profile = 'Product Manager';

System.debug(InvoiceController.getInvoices());
```

Рисунок 3.16 Код для тестування класу InvoiceController для менеджеру

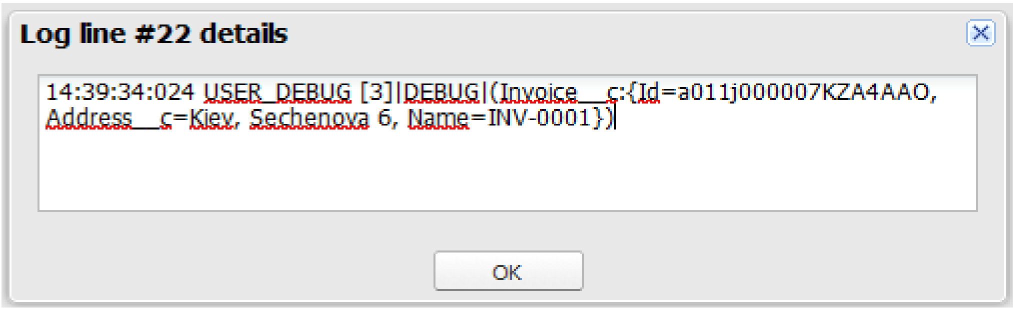

Рисунок 3.23 Результат тестування класу InvoiceController

Якщо ж ми зайдемо з під користувача адміністратора, ми повинні бачити усі замовлення і побачимо, що у системі їх 2, як на рисунку 3.18.

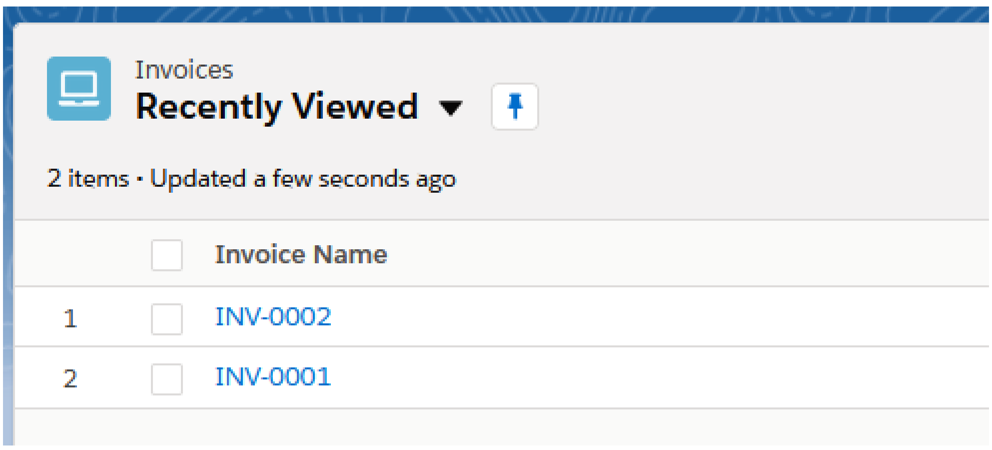

Рисунок 3.24 Список замовлень який може бачити адміністратор

Таким чином, протестувавши наші розроблені компоненти та налаштування ми можемо впевнитися в правильності роботи додатку. Під час тестування компонентів ми впевнилися, що вони відображаються вірно, а правила розмежування доступу працюють як для клієнтських компонентів, так і для серверного коду, а також для усіх профілів. Отже, ми протестували побудований нами приклад додатку в системі Salesforce.

Висновки за розділом 3

У третьому розділі ми розглянули побудову безпечного додатку на базі CRM – системи Salesforce. Під час розробки даного рішення, ми використовували вбудовані інструменти для побудови додатків цієї системи, а також написали програмний код для компонентів, яких не вистачало нам для нашого рішення. Використавши техніки, які ми описали в попередніх розділах, ми отримали стійку до більшості типів загроз систему, яку можна використовувати на підприємстві.

Також ми розглянули побудову розмежування доступу до системи, яка базується на різних типах користувачів, створили об’єкти в базі даних з певними полями, розробили програмний інтерфейс для роботи з клієнтами, а також серверний код для роботи бізнес логіки.

Звісно, ми використали тільки малу частину реального функціоналу системи, який ми описували раніше, але при цьому ми отримали захищений приклад додатку на базі системи Salesforce, тому використання цієї системи було цілком доцільним.

ВИСНОВКИ

Під час виконання дипломної роботи був проведений аналіз вразливостей, шляхів та рекомендацій захисту CRM систем. Як основу для аналізу ми обрали одну з популярних систем управління відносин з клієнтами, а саме Salesforce. Salesforce є хмарною системою і надає доступ до уже готових інструментів для розробки додатків. Таким чином, при розробці прикладу додатку ми використовували Salesforce, як платформу, її інструментарій був використаний у ході будування розмежування доступу. Також розроблюючи програмний код для компонентів роботи з клієнтами, ми отримали навички у побудові безпечних до веб загроз рішень.

За результатами розробки проекту додатку, ми отримали стійку до загроз систему з правилами розмежування доступу і готовими для використання компонентами для використання на базі системи Salesforce. Під час розробки ми оцінили методики та шляхи захисту систем управління відносин з клієнтами, розглянули переваги хмарних рішень, а також саме системи Salesforce. Загалом після проведеного дослідження, можна зробити висновок, що питання кібербезпеки є завжди актуальним у таких типах систем, при чому саме обрана нами система є доцільним прикладом вирішення цього питання.

Для досягнення поставленої мети в дипломній роботі було виконано такі завдання:

- Досліджено можливі загрози CRM – систем.
- Проаналізовано ознаки безпечної CRM – системи.
- Ознайомлено з існуючими засобами захисту від загроз.
- Зроблено аналіз інструментарію існуючих рішень CRM – систем.
- Розглянуто особливості побудови захисту таких систем.
- Розроблено рекомендації для побудови захисту даних у CRM – системах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Что такое CRM – системы и как их правильно выбирать? [Електронний ресурс] //Habr.com – 2015. – Режим доступу до ресурсу: https://habr.com/ru/company/trinion/blog/249633/ .
2. Облачная CRM - надуманные страхи [Електронний ресурс] // Блог компанії WireCRM – Режим доступу до ресурсу: https://wirecrm.com/articles/oblachnaya-crm-nadumannyye-strakhi .
3. Майбутнє за хмарними CRM [Електронний ресурс] // Блог компанії Bitrix24 – Режим доступу до ресурсу: https://www.bitrix24.ua/blogs/maybutn-za-khmarnimi-crm.php .
4. Четыре основные концепции безопасности облачных технологий [Електронний ресурс] // Anti-malware.ru – 2019. – Режим доступу до ресурсу: https://www.anti-malware.ru/analytics/Technology_Analysis/4-cloud-security-concepts .
5. Salesforce [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Salesforce.com .
6. Що таке Salesforce і чим вона цікава для досвідчених розробників [Електронний ресурс] // Dou.ua – 2019. – Режим доступу до ресурсу: https://dou.ua/lenta/articles/what-salesforce-is/ .
7. Overview of Data Security [Електронний ресурс] // Trailhead – Режим доступу до ресурсу:https://trailhead.salesforce.com/en/content/learn/modules/data_security/data_security_overview .
8. Control Access to the Org [Електронний ресурс] // Trailhead – Режим доступу до ресурсу:https://trailhead.salesforce.com/en/content/learn/modules/data_security/data_security_org?trail_id=security .
9. Control Access to Objects [Електронний ресурс] // Trailhead – Режим доступу до

ресурсу:https://trailhead.salesforce.com/content/learn/modules/data_security/data_security_objects?trail_id=security .

10. Create a Role Hierarchy [Електронний ресурс] // Trailhead – Режим доступу до ресурсу:https://trailhead.salesforce.com/content/learn/modules/data_security/data_security_roles?trail_id=security .

11. Control Access to Records [Електронний ресурс] // Trailhead – Режим доступу до ресурсу:https://trailhead.salesforce.com/content/learn/modules/data_security/data_security_records .

12. Control Access to Fields [Електронний ресурс] // Trailhead – Режим доступу до ресурсу:https://trailhead.salesforce.com/content/learn/modules/data_security/data_security_fields .

13. Best practices to backup Salesforce data [Електронний ресурс] // help.salesforce.com – Режим доступу до ресурсу: https://help.salesforce.com/articleView?id=000334121&type=1&mode=1 .

14. Типы уязвимостей сайтов [Електронний ресурс] // Losst.ru – 2017. – Режим доступу до ресурсу: https://losst.ru/tipy-uyazvimostej-sajtov\ .

15. Protect Apps with Shield [Електронний ресурс] // Trailhead – Режим доступу до ресурсу: https://trailhead.salesforce.com/content/learn/modules/secure-salesforce-configuration/protect-apps-with-shield?trail_id=security_developer .

16. Build Secure Apps with Lightning Web Components [Електронний ресурс] // Trailhead – Режим доступу до ресурсу: https://trailhead.salesforce.com/en/content/learn/modules/secure-clientside-development/build-secure-apps-with-lightning-web-components?trail_id=security_developer .

17. Апекс — Краткое руководство [Електронний ресурс] // Coderlessons.com – 2019. – Режим доступу до ресурсу:

https://coderlessons.com/tutorials/kompiuternoe-programmirovanie/izuchite-apex-programmirovanie/apeks-kratkoe-rukovodstvo .

18. Write Secure Apex Controllers [Електронний ресурс] // Trailhead – Режим доступу до ресурсу: https://trailhead.salesforce.com/en/content/learn/modules/secure-serverside-development/write-secure-apex-controllers?trail_id=security_developer

ДОДАТОК А

Код класу ProductListController

```
public with sharing class ProductListController {
    private static ProductItemsService service = new ProductItemsService();
    @AuraEnabled
    public static List<ProductItem__c> getProducts(){
        try {
            return service.getProductList();
        } catch (Exception e) {
            throw new AuraHandledException(e.getMessage());
        }
    }
}
```

Код класу InvoiceController

```
public with sharing class InvoiceController {
    private static InvoiceService service = new InvoiceService();
    @AuraEnabled
    public static void saveRecord(String productsJSON, String territory, String address){
        try {
            Invoice__c newInvoice = new Invoice__c(Country__c = territory, Address__c = addr
ess);
            insert newInvoice;
            List<ProductItemWrapper> productList = (List<ProductItemWrapper>) JSON.deseriali
ze(productsJSON, List<ProductItemWrapper>.class);
            service.saveInvoice(newInvoice, productList);
        } catch (Exception e) {
            throw new AuraHandledException(e.getMessage());
        }
    }
    @AuraEnabled
    public static List<Territory__c> getTerritories(){
        try {
            return service.getAvailableTerritories();
        } catch (Exception e) {
            throw new AuraHandledException(e.getMessage());
        }
```

```
    }
    @AuraEnabled
    public static List<Invoice__c> getInvoices(){
        try {
            return [
                SELECT Name, Address__c, Id
                FROM Invoice__c
            ];
        } catch (Exception e) {
            throw new AuraHandledException(e.getMessage());
        }
    }
    public class ProductItemWrapper{
        public String Id;
        public Integer quantity;
    }
}
```

Код класу InvoiceService

```
public with sharing class InvoiceService {
    public List<Territory__c> getAvailableTerritories() {
        return [
            SELECT Id, Name
            FROM Territory__c
        ];
    }
    public void saveInvoice(Invoice__c invoice, List<InvoiceController.ProductItemWrapper> p
roductList) {
        List<InvoiceLineItem__c> lineItems = new List<InvoiceLineItem__c>();
        for(InvoiceController.ProductItemWrapper item : productList) {
            InvoiceLineItem__c newItem = new InvoiceLineItem__c();
            newItem.QuantityProductItems__c = item.quantity;
            newItem.ProductItem__c = item.Id;
            newItem.Invoice__c = invoice.Id;
            lineItems.add(newItem);
        }
        insert lineItems;
    }
    public static void insertPriceToLineItem(List<InvoiceLineItem__c> lineItems) {
```

```
        Set<Id> productIds = new Set<Id> ();
        for(InvoiceLineItem__c item : lineItems) {
            productIds.add(item.ProductItem__c);
        }
        List<ProductItem__c> products = [
            SELECT Id, PriceOfItem__c
            FROM ProductItem__c
            WHERE Id IN :productIds
        ];
        for(InvoiceLineItem__c lineItem : lineItems) {
            for(ProductItem__c product : products) {
                if(product.Id == lineItem.ProductItem__c) {
                    lineItem.LineItemPrice__c = product.PriceOfItem__c;
                }
            }
        }
    }
}
```

Код класу ProductItemsService

```
public with sharing class ProductItemsService {
    public List<ProductItem__c> getProductList() {
        return [
            SELECT Id, Name, PriceOfItem__c, ImageUrl__c FROM ProductItem__c
            WHERE Quantity__c > 0
        ];
    }
}
```

ДОДАТОК Б

HTML код компоненту Cart

```
<template>
    <lightning-card title="Cart">
        <lightning-layout multiple-rows="true">
            <template for:each={cartProducts} for:item="product">
                <lightning-layout-item size="12" key={product.Name} class="slds-p-
around_medium">
                    <c-cart-item
                        product={product}
                        cart-products={cartProducts}
                        onrefresh={refreshProducts}
                    ></c-cart-item>
                </lightning-layout-item>
            </template>
        </lightning-layout>
        <hr />
        <c-invoice-form
            products={cartProducts}
            onrefresh={refreshProducts}
        ></c-invoice-form>
    </lightning-card>
</template>
```

JavaScript код компоненту Cart

```
import { LightningElement, track } from 'lwc';
export default class Cart extends LightningElement {
    @track
    cartProducts = [];
    constructor() {
        super();
        this.getProductsFromLocal();
    }
    getProductsFromLocal() {
        this.cartProducts = JSON.parse(localStorage.getItem("products"));
    }
    refreshProducts() {
        this.getProductsFromLocal();
```

```
    }
}
```

HTML код компоненту CartItem

```
<template>
    <div>
        <h5>{product.item.Name}</h5>
        <p>{product.item.PriceOfItem__c}</p>
        <p>Number of items: {product.quantity}</p>
        <lightning-button
            label="Remove from cart"
            variant="destructive"
            onclick={removeFromLocal}
        ></lightning-button>
    </div>
</template>
```

JavaScript код компоненту CartItem

```
import { api, LightningElement } from 'lwc';
export default class CartItem extends LightningElement {
    @api
    product;
    @api
    cartProducts;
    removeFromLocal() {
        let arr = this.cartProducts.filter(element => {
            if(element.item.Name !== this.product.item.Name){
                return true;
            }
            else{
                return false;
            }
        });
        localStorage.setItem("products", JSON.stringify(arr));
        this.fireRefresh();
    }
    fireRefresh() {
        this.dispatchEvent(new CustomEvent('refresh'));
    }
}
```

HTML код компоненту InvoiceForm

```
<template>
    <lightning-layout multiple-rows="true">
        <lightning-layout-item class="slds-p-around_medium" size="12">
            <lightning-combobox
                label="Territory"
                options={selectOptions}
                value={territory}
                onchange={handleTerritory}
            ></lightning-combobox>
        </lightning-layout-item>
        <lightning-layout-item class="slds-p-around_medium" size="12">
            <lightning-input
                label="Address"
                value={address}
                onchange={handleAddress}
            ></lightning-input>
        </lightning-layout-item>
        <lightning-layout-item class="slds-p-around_medium" size="12">
            <lightning-button
                variant="brand"
                label="Book products"
                onclick={bookProducts}
            ></lightning-button>
        </lightning-layout-item>
    </lightning-layout>
</template>
```

JavaScript код компоненту InvoiceForm

```
import { LightningElement, api, track } from 'lwc';
import getTerritories from '@salesforce/apex/InvoiceController.getTerritories';
import saveRecord from '@salesforce/apex/InvoiceController.saveRecord';
export default class InvoiceForm extends LightningElement {
    @api
    products;
    @track
    territories = [];
    territory;
    address = '';
    get selectOptions() {
```

```
    let options = [];
    this.territories.forEach(item => {
        options.push({
            label : item.Name,
            value : item.Name
        });
    });
    return options;
}
constructor() {
    super();
    this.getTerritories();
}
async getTerritories() {
    try{
        const response = await getTerritories();
        this.territories = response;
    }
    catch(error) {
        console.log(error);
    }
}
handleTerritory(event) {
    this.territory = event.detail.value;
}
async bookProducts() {
    try{
        await saveRecord(
            {
                productsJSON : JSON.stringify(this.wrapProducts(this.products)),
                territory : this.territory,
                address : this.address
            }
        );
        this.refreshCartAfterSave();
    }
    catch(error) {
        console.log(error);
    }
}
```

```
    wrapProducts() {
        let productList = [];
        this.products.forEach(element => {
            productList.push({
                ...element.item,
                quantity : element.quantity
            });
        });
        return productList;
    }
    handleAddress(event) {
        this.address = event.detail.value;
    }
    refreshCartAfterSave() {
        localStorage.setItem('products', JSON.stringify([]));
        this.dispatchEvent(new CustomEvent('refresh'));
        this.territory = null;
        this.address = '';
    }
}
```

HTML код компоненту ProductList

```
<template>
    <lightning-card title="Product List">
            <lightning-layout multiple-rows="true">
                <template for:each={products} for:item="product">
                    <lightning-layout-item class="slds-p-
around_medium" size="4" key={product.Name}>
                        <c-product-item
                            product={product}
                        ></c-product-item>
                    </lightning-layout-item>
                </template>
            </lightning-layout>
    </lightning-card>
</template>
```

JavaScript код компоненту ProductList

```
import { LightningElement, track } from 'lwc';
import getProducts from '@salesforce/apex/ProductListController.getProducts';
export default class ProductList extends LightningElement {
```

```
@track
products;
constructor() {
    super();
    this.getProducts();
}
async getProducts() {
    try{
        const response = await getProducts();
        this.products = response;
    }catch(error) {
        console.log(error);
    }}}
```