МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

ВОВЧАК Богдан Андрійович

Програмна система для ідентифікації користувача на основі технології NFC / Software system for user identification based on NFC technology

спеціальність: 125 – Кібербезпека

освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи КБм -21
Б.А. Вовчак

Науковий керівник
д.т.н., професор М.М.Касянчук

Кваліфікаційну роботу допущено
до захисту:

«____» _____ ___2023 р.

Завідувач кафедри
_____ В.В.Яцків

ТЕРНОПІЛЬ – 2023

Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки
Освітній ступінь "магістр"
спеціальність:125 – Кібербезпека
освітньо-професійна програма – Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ В.В.Яцків
"____"_____20__ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ВОВЧАК Богдан Андрійович
(прізвище, ім'я по-батькові)

1. Тема кваліфікаційної роботи
Програмна система для ідентифікації користувача на основі технології NFC / Software system for user identification based on NFC technology
керівник роботи: д.т.н., проф. М.М.Касянчук

затверджені наказом по університету від «___» _____2023 року № ___

2. Строк подання студентом закінченої кваліфікаційної роботи 1 грудня 2023 р.

3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити
- провести огляд існуючих систем та технологій для ідентифікації користувачів;
- розробити концепцію уніфікованої системи ідентифікації, яка об'єднуватиме різні методи індентифікації користувачів;
- спроектувати систему ідентифікації користувача за допомогою NFC мітки;
- розробити програмну систему ідентифікації користувачів, яка базується на технології NFC;
- реалізувати розроблену систему на практиці, включаючи програмний та апаратний забезпечення;
- провести тести та налаштування системи для забезпечення її надійності та ефективності.

5. Перелік графічного матеріалу у роботі.

- принцип роботи технології NFC;
- схеми атак на дані, що передаються відповідно до технології NFC;
- схема емуляції картки з використанням Secure Element;
- схема емуляції картки з використанням HCE Android;
- стек протоколів, які підтримуються реалізацією HCE;
- процес вибору сервісу-обробника HCE;
- загальна схема прецедентів;
- діаграма бізнес сутностей;
- модельно-орієнтована архітектура програмного забезпечення;
- ER-діаграма бази даних.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання: 8 грудня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назви етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Огляд методів та технологій для ідентифікації користувачів в програмних системах	12.2022 р. – 03.2023 р.	
2	Безпека передачі даних під час ідентифікації користувачів на основі технології NFC	03.2023 р. – 05.2023 р.	
3	Розробка програмної системи для ідентифікації користувача на основі технології NFC	05.2023 р. – 11.2023 р.	

Студент _____ Вовчак Б.А.
(підпис)

Керівник роботи _____ д.т.н., проф. Касянчук М.М.
(підпис)

АНОТАЦІЯ

Випускна кваліфікаційна робота на тему „Програмна система для ідентифікації користувача на основі технології NFC” на здобуття освітнього ступеня «Магістр» зі спеціальності 125 „Кібербезпека” освітньо-професійної програми «Кібербезпека» написана обсягом 99 сторінок і містить 31 ілюстрацію, 13 таблиць, 2 додатоки та 30 джерел за переліком посилань.

Метою випускної кваліфікаційної роботи є підвищення ефективності ідентифікації користувачів за допомогою технології NFC.

Методи дослідження. Методи об’єктно-орієнтованого програмування та математичні методи моделювання.

Результати дослідження: Проведено огляд існуючих систем та технологій для ідентифікації користувачів, що дало змогу розробити концепцію уніфікованої системи ідентифікації, яка об'єднуватиме різні методи індентифікації користувачів. На основі отриманих даних спроектовано систему ідентифікації користувача за допомогою NFC мітки, яка базується на запропонованій у роботі модельно-орієнтованій архітектурі програмного забезпечення. Розроблено програмне та апаратне забезпечення для системи для ідентифікації користувача на основі технології NFC. Загалом, розроблена система допоможе розробникам швидше і ефективніше створювати програми, які взаємодіють з фізичними об'єктами за допомогою технології NFC.

Ключові слова: ТЕХНОЛОГІЯ NFC, NFC МІТКИ, ПРОГРАМНА СИСТЕМА, ІДЕНТИФІКАЦІЯ КОРИСТУВАЧІВ.

ABSTRACT

The graduate work on the topic „ Software System for User Identification Based on NFC Technology” for Master’s degree on speciality 125 "Cybersecurity " is written on 99 pages and contains 31 illustrations, 13 tables, 2 appendixs and 30 references.

The aim of graduate work is is to enhance the efficiency of user identification using NFC technology.

Research methods: Object-oriented programming methods and mathematical modeling methods.

Research results: An overview of existing systems and technologies for user identification has been conducted, which allowed for the development of a concept for a unified identification system that integrates various user identification methods. Based on the obtained data, a user identification system using NFC tags has been designed, which is based on the proposed model-oriented software architecture in the work. Both software and hardware components have been developed for the user identification system based on NFC technology. Overall, the developed system will assist developers in creating programs that interact with physical objects using NFC technology more quickly and efficiently.

Keywords: NFC TECHNOLOGY, NFC TAGS, SOFTWARE SYSTEM, USER IDENTIFICATION

ЗМІСТ

ВСТУП 7
1 ОГЛЯД МЕТОДІВ ТА ТЕХНОЛОГІЙ ДЛЯ ІДЕНТИФІКАЦІЇ КОРИСТУВАЧІВ В ПРОГРАМНИХ СИСТЕМАХ 10
1.1 Аналіз методів ідентифікації користувачів 10
1.2 Огляд технології NFC 17
1.3 Порівняльний аналіз засобів для ідентифікації користувачів на основі технології NFC 19
1.4 Постановка задачі дослідження 25
2 БЕЗПЕКА ПЕРЕДАЧІ ДАНИХ ПІД ЧАС ІДЕНТИФІКАЦІЇ КОРИСТУВАЧІВ НА ОСНОВІ ТЕХНОЛОГІЇ NFC 27
2.1. Загрози при передачі даних із використання технології NFC 27
2.2. Дослідження технології NFC в мобільних пристроях 35
2.3. Проектування NFC- тегу 39
3 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ ДЛЯ ІДЕНТИФІКАЦІЇ КОРИСТУВАЧА НА ОСНОВІ ТЕХНОЛОГІЇ NFC 42
3.1 Проектування програмної системи 42
3.2 Проектування бази даних програмної системи 54
3.3 Тестування програмної системи 66
ВИСНОВКИ 70
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 72
ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО КОДУ 76
ДОДАТОК Б КОПІЇ ПУБЛІКАЦІЇ 86

ВСТУП

У зв'язку з поширенням комп'ютерних технологій, проблема забезпечення безпеки інформації в комп'ютерних інформаційних системах стає все більш актуальною [1-3]. Тому дуже важливими є теоретичні дослідження в галузі захисту комп'ютерної інформації та їх практичне впровадження в конкретних комп'ютерних системах. Мета захисту інформації в комп'ютерних системах полягає в тому, щоб відокремити нормально працюючу інформаційну систему від несанкціонованих дій та доступу з боку сторонніх осіб або програм до захищених комп'ютерних даних [4-5]. Створення єдиної, централізованої системи безпеки є необхідною умовою для функціонування сучасної інформаційної інфраструктури [6].

Технологія бездротової передачі даних NFC - це сучасна технологія автоматичної безконтактної ідентифікації об'єктів через бездротовий зв'язок в реальному часі, яка має безліч застосувань. Основною характеристикою NFC є бездротовий інтерфейс, який обмежується робочою відстанню до 10 сантиметрів і може працювати в кількох режимах [7-9]. NFC знаходить застосування в таких галузях, як логістика, управління персоналом, виробництво, банківські системи, системи контролю та управління доступом, хмарні технології, Інтернет речей, соціальні мережі та інші.

На сьогодні існує безліч систем ідентифікації, які працюють в окремих установах чи інформаційних системах [10-11]. Проте всі вони не пов'язані між собою і вимагають наявності ідентифікатора у користувача (зазвичай пластикової картки), що призводить до необхідності носити з собою велику кількість пластикових карток, яка постійно зростає, і постійно шукати потрібну у даний момент [12-13].

Впровадження єдиної системи ідентифікації дозволить замінити всі пластикові картки єдиною NFC міткою. Тому тема даної роботи є актуальною і потребує подальших досліджень.

Мета і задачі дослідження. Метою дослідження є підвищення ефективності ідентифікації користувачів за допомогою технології NFC.

Досягнення цієї мети передбачає розв’язання таких наукових завдань:

- провести огляд існуючих систем та технологій для ідентифікації користувачів;
- розробити концепцію уніфікованої системи ідентифікації, яка об'єднуватиме різні методи індентифікації користувачів;
- спроектувати систему ідентифікації користувача за допомогою NFC мітки;
- розробити програмну систему ідентифікації користувачів, яка базується на технології NFC;
- реалізувати розроблену систему на практиці, включаючи програмний та апаратний забезпечення;
- провести тести та налаштування системи для забезпечення її надійності та ефективності

Об’єкт дослідження – процес ідентифікації користувачів.

Предмет дослідження – методи та програмні засоби для ідентифікації користувачів на основі технології NFC.

Методи дослідження. В роботі використовуються методи об’єктно-орієнтованого програмування та математичні методи моделювання.

Наукова новизна. У роботі запропоновано модельно-орієнтовану архітектуру програмного забезпечення, яка спрощує створення застосунків на основі технології NFC та надає розробникам готові шаблони вихідного коду.

Практичне значення. Реалізовано програмну систему для ідентифікації користувачів на основі технології NFC.

Публікації та апробація.

1. Вовчак Б.А. Аналіз атак при передачі даних з використанням технології NFC. Збірник матеріалів школи-семінару молодих вчених і

студентів «Комп'ютерні інформаційні технології», Тернопіль. 2023. С. 20-21 [14].

2. Вовчак Б.А. Архітектура програмного забезпечення для розробки застосунків на основі технології NFC // Збірник матеріалів міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ», Вінниця, 2023. С. 54-56 [15].

1 ОГЛЯД МЕТОДІВ ТА ТЕХНОЛОГІЙ ДЛЯ ІДЕНТИФІКАЦІЇ КОРИСТУВАЧІВ В ПРОГРАМНИХ СИСТЕМАХ

1.1 Аналіз методів ідентифікації користувачів

Системи ідентифікації та аутентифікації визнається ключовим складовим елементом інфраструктури для запобігання несанкціонованому доступу (НСД) до будь-якої інформаційної системи [13]. Несанкціонований доступ означає надмірне проникнення до інформації, порушуючи встановлені правила доступу, і використання при цьому офіційних обчислювальних засобів або автоматизованих систем. НСД може мати як ненавмисний, так і навмисний характер [16-17].

Ідентифікація включає в себе пред'явлення користувачем ідентифікаційних ознак, які є унікальними і притаманними лише йому. І аутентифікація полягає в переконанні другої сторони в тому, що суб'єкт насправді є тим, за кого він себе видає. Іноді вживається синонім "перевірка достовірності" для виразу "аутентифікація".

На сьогоднішній день існує кілька методів ідентифікації користувачів [14], кожен з яких має свої переваги та недоліки. Це призводить до ситуації, коли певні технології підходять для одних комп'ютерних систем, а інші - для інших. Проте в багатьох випадках не існує однозначного рішення, і розробники програмного забезпечення, а також користувачі, повинні самостійно визначити, який метод ідентифікації використовувати у своїх інформаційних комп'ютерних системах.

Існує три основних методи ідентифікації:

1) парольна ідентифікація - до недавнього часу цей метод був практично єдиною системою визначення особистості користувача, і це зрозуміло. Парольна ідентифікація є найбільш простою в реалізації та використанні. Суть полягає в тому, що кожен зареєстрований користувач отримує персональний набір ідентифікаційних даних (зазвичай це пара логін-

пароль), які він використовує для входу в систему. Оскільки ці дані унікальні для кожного користувача, система може ідентифікувати його.

Головна перевага парольної ідентифікації полягає в її простоті використання та реалізації. Додатково, вона не вимагає значних витрат, оскільки вона підтримується в більшості програмних продуктів, забезпечуючи доступність і простоту захисту інформації.

Проте існують і значні недоліки. Один із них - висока залежність від надійності користувачів, а саме від обраних ними паролів. Багато людей використовують ненадійні паролі, які можуть бути легко вгадані. Це стосується коротких паролів, загальновідомих комбінацій символів тощо. Тому фахівці з інформаційної безпеки рекомендують використовувати довгі паролі, складені з випадкових комбінацій букв, цифр та символів);

2) апаратна (електронна) ідентифікація базується на визначенні особистості користувача за допомогою предмету або ключа, який перебуває в його виключному користуванні. Це не стосується звичайних ключів, але електронних ключів [9]. На сьогоднішній день найпоширеніші два типи пристроїв: різноманітні карти (проксіміті-карти, смарт-карти, магнітні карти і т.д.) та так звані токени (token), які підключаються безпосередньо до порта комп'ютера.

Головною перевагою апаратної ідентифікації є висока надійність. В інформаційних носіях токенів можуть зберігатися ключі, які важко підібрати. Крім того, вони мають різні захисні механізми і вбудовані мікропроцесори, які дозволяють їм виконувати додаткові функції в процесі ідентифікації користувача [18-19].

Однак є декілька недоліків апаратної ідентифікації. Однією з серйозних небезпек є можливість крадіжки токенів або карт у зареєстрованих користувачів. Крім того, вони можуть бути втрачені, передані іншій особі або навіть здубльовані. Іншим недоліком цієї технології є її вартість. Незважаючи на те, що вартість самого електронного ключа і програмного забезпечення для нього знизилася, для впровадження цієї системи

ідентифікації все одно потрібні певні витрати. Кожному зареєстрованому користувачу потрібно забезпечити персональними токенами, які також можуть витрати ся з часом, бути загублені і т.д. Тобто апаратна ідентифікація потребує певних експлуатаційних витрат;

3) біометрична ідентифікація використовується для визначення особистості за допомогою її унікальних біологічних ознак. Це означає, що біометричні технології розроблялися для точного ідентифікування особи, використовуючи її біологічні характеристики, які є унікальними для кожної людини. Використання біометричної ідентифікації в галузі інформаційної безпеки є логічним рішенням і активно розвивається. На сьогодні існує більше десятка різних біометричних ознак [5]. Для найпоширеніших з них, таких як відбитки пальців і райдужна оболонка ока, розроблено безліч різних типів сканерів. Таким чином, користувачі, які обирають біометричну ідентифікацію, мають можливість вибирати серед різних пристроїв, які працюють за різними принципами.

Біометричні технології відрізняються високою надійністю, оскільки не існує двох людей з однаковими біометричними характеристиками, такими як відбитки пальців. На сьогоднішній день існують спроби обходу біометричних сканерів, такі як перенесення відбитків пальців на плівку або використання фотографій відбитків пальців. Проте сучасні пристрої є значно стійкішими до таких видів обману.

Однак головним недоліком біометричної ідентифікації є висока вартість обладнання. Кожен комп'ютер у такій системі повинен мати свій власний біометричний сканер, що призводить до додаткових витрат. Незважаючи на те, що ціни на біометричні пристрої постійно знижуються, вони все ще залишаються високими.

Також важливо враховувати, що ідентифікація на основі одного фактора, такого як відбиток пальця, не завжди є надійною. Тому сучасні системи часто використовують багатофакторну аутентифікацію для забезпечення вищого рівня безпеки.

Комплексна ідентифікація використовує кілька параметрів для визначення особи користувача в комп'ютерних інформаційних системах. При цьому ці параметри можуть комбінуватися в довільному порядку. У більшості випадків використовують лише одну пару параметрів, таку як пароль і токен. Наприклад, користувачу може потрібно буде ввести пароль і вставити токен, щоб отримати доступ до системи. Такий підхід забезпечує високий рівень безпеки, оскільки обидва фактори є обов'язковими для аутентифікації, і без них доступ до системи надається. Однак в деяких системах застосовуються більш складні процедури ідентифікації, які включають одночасне використання паролів, токенів і біометричних характеристик [20-21].

Парольні системи захисту мають певні переваги, такі як простота та звичність використання. Паролі вже давно вбудовані в багато операційних систем і сервісів, і їх можна легко використовувати. При правильному використанні паролі можуть забезпечити прийнятний рівень безпеки для багатьох організацій. Однак, через ряд обмежень, паролі є найслабкішим засобом перевірки достовірності в контексті комп'ютерної безпеки. Недоліками парольної ідентифікації є велика залежність від користувачів, які часто використовують ненадійні паролі, і можливість їхнього підбору або зламу.

На жаль, символьні паролі ще довго залишатимуться одним з найпоширеніших способів ідентифікації, але багатофакторна ідентифікація дозволяє посилити безпеку, поєднуючи паролі з іншими факторами, такими як токени і біометричні характеристики.

Щоб підвищити надійність захисту паролів, можна застосувати такі заходи [3]:

- встановлення обмежень: Налаштування мінімальної довжини паролів та вимог до їх складності, таких як використання букв, цифр, знаків пунктуації тощо;
- керування строком дії паролів та їх регулярна зміна;

- обмеження доступу до файлу паролів;
- обмеження кількості невдалих спроб входу в систему;
- використання програм, що генерують паролі. Ці програми відповідають певним правилам і створюють паролі, які легко запам'ятовуються.

Для більш детального розгляду принципів побудови систем парольного захисту, ми визначимо основні поняття.

Апаратна (або електронна) ідентифікація включає в себе фізичні пристрої невеликих розмірів, які дозволяють користувачеві підтверджувати свою ідентичність. Ці системи можуть використовувати різні методи ідентифікації, включаючи відбитки пальців, смарт-карти, розпізнавання обличчя, і багато інших.

Переносні аутентифікаційні пристрої:

- асинхронні пристрої: Користувач вводить інформацію в пристрій, отримує відповідь і передає її до комп'ютера;
- PIN/асинхронні пристрої: Вони поєднують асинхронний метод з введенням PIN-коду для забезпечення більш високого рівня безпеки;
- синхронні пристрої: Наприклад, синхронізований токен генерує пароль для користувача у визначену мить, який автоматично вводиться в систему;
- PIN/синхронні пристрої: Ці пристрої поєднують синхронну генерацію паролю з введенням PIN-коду.

Смарткарти різних типів:

- пасивні карти (карти з пам'яттю): Карти, які можуть зберігати інформацію, але не мають обчислювальних можливостей;
- активні карти (інтелектуальні карти) використовуються для генерації одноразових паролів та взаємодії з пристроєм через картридер. Наприклад, користувач вводить PIN-код, картридер взаємодіє з смарт-картою, і процес відбувається автоматично без додаткового втручання

людини. Деякі картки є безконтактними і дозволяють проводити ідентифікацію без фізичного контакту.

Ці технології використовуються для забезпечення безпеки доступу та аутентифікації користувачів у різних системах.

Основні функції eToken включають:

- двофакторна аутентифікація користувачів: Використовується для захищеного доступу до ресурсів, таких як комп'ютери, мережі та додатки;
- безпечне зберігання конфіденційної інформації: eToken може зберігати закриті ключі цифрових сертифікатів, криптографічні ключі, налаштування додатків та інші дані в безпечній незалежній пам'яті;
- апаратне виконання криптографічних операцій: eToken підтримує генерацію ключів шифрування, симетричне і асиметричне шифрування, розрахунок хеш-функцій та формування електронних цифрових підписів (ЕЦП) в надійному апаратному середовищі;
- eToken інтегрується з багатьма операційними системами, бізнес-додатками та засобами забезпечення інформаційної безпеки, що дозволяє використовувати його для різних завдань;
- сувору аутентифікацію користувачів при доступі до серверів, баз даних та розділів веб-сайтів;
- збереження паролів, ключів шифрування та закритих ключів цифрових сертифікатів;
- захист електронної пошти, включаючи цифровий підпис і шифрування;
- захист систем електронної торгівлі, інтернет-банкінгу та додатків для банківських послуг;
- захист комп'ютерів і мереж шляхом створення віртуальних приватних мереж (VPN).

eToken надає важливі переваги, такі як апаратна аутентифікація користувачів, безпечне зберігання конфіденційної інформації та мобільність

користувача для безпечної роботи з конфіденційними даними навіть на ненадійних комп'ютерах.

Безконтактні смарт-карти можуть бути розділені на дві основні категорії: ідентифікатори Proximity та смарт-карти, які відповідають міжнародним стандартам ISO/IEC 15693 і ISO/IEC 14443. В основі багатьох пристроїв, що використовують безконтактні смарт-карти, лежить технологія радіочастотної ідентифікації. Головними компонентами цих пристроїв є чип та антена. Ідентифікатори можуть бути активними (з власним джерелом живлення) або пасивними (без власного живлення). Вони мають унікальні 32 або 64-розрядні серійні номери. Проте системи ідентифікації на базі технології Proximity в основному не мають криптографічного захисту, за винятком спеціалізованих систем.

USB-ключі взаємодіють з комп'ютером через USB-порт і можуть використовуватися для різних цілей, таких як аутентифікація користувачів і зберігання конфіденційних даних.

Біометрична ідентифікація - це метод ідентифікації особи на основі її унікальних біометричних характеристик. Ці характеристики є індивідуальними для кожної людини і можуть включати в себе фізіологічні ознаки та поведінкові характеристики. Біометрична ідентифікація використовується для визначення особи та вирішення питань доступу до ресурсів комп'ютерних систем.

Біоелектронні системи - ці системи використовують біометричні дані, такі як відбитки пальця, для ідентифікації користувачів. Іноді їх також комбінують з іншими методами, які роблять системи ще більш надійними та безпечними.

Впровадження комбінованих систем допомагає підвищити безпеку та надійність процесу ідентифікації користувачів в інформаційних системах.

1.2 Огляд технології NFC

Технологія ближнього бездротового зв'язку (NFC) - це стандарт для бездротового обміну даними між пристроями на невеликій відстані, зазвичай менше ніж 4 сантиметри. NFC використовується для безконтактних платежів, обміну контактною інформацією, керування електронними пристроями та інших застосувань [22-23].

Технологія NFC базується на принципах радіочастотної ідентифікації (RFID) та використовує індукцію магнітного поля для забезпечення зв'язку між пристроями.

Основна ідея полягає в тому, що пристрої, які підтримують NFC, можуть спілкуватися, просто наближаючись один до одного або доторкаючись. Ця технологія дозволяє безпечно обмінювати даними, використовувати спільні додатки та спрощує взаємодію між пристроями.

Використання NFC розширюється на багато галузей життя, включаючи безконтактні платежі, обмін контактами та даними, контроль доступу та ідентифікацію користувачів.

Технологія NFC стала особливо популярною завдяки її використанню в смартфонах і дозволяє споживачам зручно та ефективно взаємодіяти з різними пристроями та гаджетами за допомогою всього дотику або точкового з'єднання [24-25].

Мобільні телефони з підтримкою технології NFC надають користувачам можливість виконувати різні дії у середовищі, що підтримує NFC. Ось кілька прикладів:

- завантажте музику чи відео зі смарт-плаката;
- обмінюйтесь візитними картками з іншим телефоном;
- сплачуйте вартість проїзду на автобусі чи поїзді;
- роздрукуйте зображення на принтері;
- використовуйте термінал торгового пункту для оплати покупки, подібно до безконтактної кредитної картки;

- з'єднайте два пристрої Bluetooth.

Телефон з підтримкою NFC може використовуватися для оплати товарів і послуг, аналогічно до безконтактних карток. Технологія NFC також дозволяє забезпечити підвищений рівень безпеки, дозволяючи користувачам захищати захищені програми через функції користувальницького інтерфейсу телефону.

Технологія NFC має два режими роботи:

1) режим пасивного зв'язку: У цьому режимі ініціатор створює поле носія, і цільовий пристрій відповідає, модулюючи це поле. Цільовий пристрій може видобувати енергію з електромагнітного поля, що створюється ініціатором, і, таким чином, виступає як транспондер;

2) режим активної комунікації: В цьому режимі як ініціатор, так і цільовий пристрій активно генерують свої власні радіочастотні поля. Пристрій може вимикати своє радіочастотне поле, коли він не передає дані, для зекономлення енергії. У цьому режимі обидва пристрої, зазвичай, мають джерело живлення.

Щодо кодування даних, NFC використовує два різні методи в залежності від швидкості передачі [26-27]:

- при передачі даних зі швидкістю 106 кбіт/с використовується модифіковане кодування Міллера з 100% модуляцією;
- у всіх інших випадках використовується кодування Манчестера з коефіцієнтом модуляції 10%.

Пристрої NFC здатні одночасно приймати та передавати дані, що дозволяє їм виявляти зіткнення, якщо прийнятий сигнал не збігається з переданим сигналом.

Принцип роботи технології зображено на рисунку 1.1.

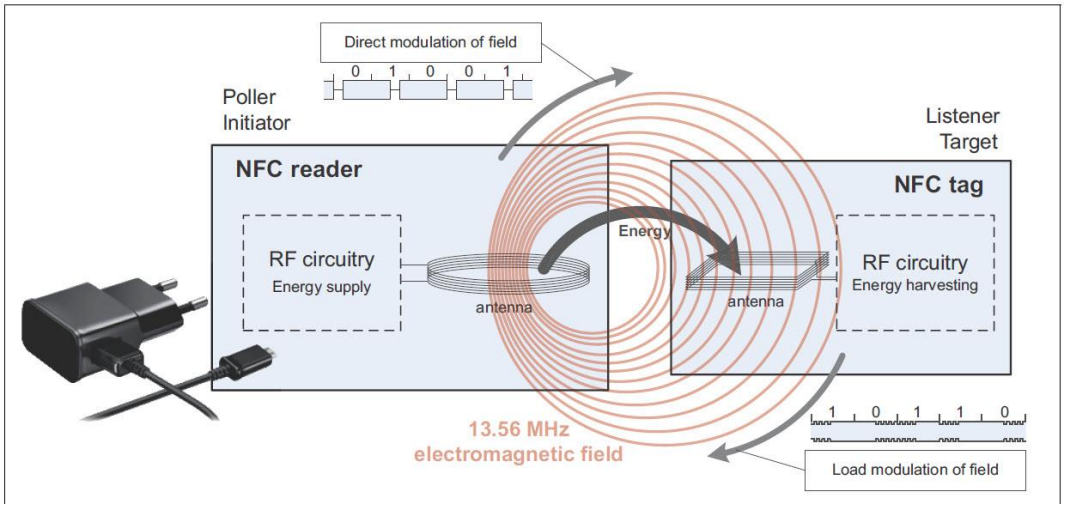

Рисунок 1.1 – Принцип роботи технології NFC

1.3 Порівняльний аналіз засобів для ідентифікації користувачів на основі технології NFC

Технологія NFC вже активно використовується в різних системах ідентифікації, оскільки вона має низку переваг, що роблять її очевидним вибором для вирішення цих завдань [28-29].

Перший банківський пластиковий платіжний картки з унікальними серійними номерами були випущені Національним банком Flatbush у Брукліні в 1946 році та Національним банком Franklin в 1951 році. Проте на початку свого існування, вони були прийняті деякими місцевими магазинами, і доступність для платежів була обмеженою. Картка Diner's Club була створена в 1950 році для використання в ресторанах високого класу, що надихнуло American Express запустити подібну платіжну картку для витрат під час поїздок.

У попередні роки картки виготовлялися з картону, однак на початку 1960-х вони перейшли до пластикових матеріалів, з виходами серійних номерів, що полегшило використання копіювальних форм для реєстрації транзакцій. У 1970-х роках технологія магнітних смужок була вперше використана на банківських картках, і в даний момент майже всі банківські

карти - це смарт-карти, які містять мікросхеми та сховище даних для автентифікації користувачів та транзакцій.

Банківські карти зазвичай мають логотип компанії, яка обробляє платежі, таку як Visa або Mastercard. Банки також можуть співпрацювати з іншими організаціями, щоб випустити спільні карти, на яких може бути розміщений логотип або зображення бренду-партнера [3]. На рисунку 1.2 наведено приклад банківської карти банку "ПриватБанк" [4].

Рисунок 1.2 – Приклад банківської картки

Електронний квиток - це система, в якій право на проїзд (або квиток) зберігається у формі електронних даних на мікрочіпі, а не видається у формі паперового квитка. Зазвичай, мікрочіп, на якому зберігається інформація про квиток, вбудований в смарт-карту.

Електронні квитки використовуються для зручної поїздки між містами та всередині них, а також для різних видів транспорту. Ця система підтримується урядовими специфікаціями та дозволяє пасажирам здійснювати поїздки без необхідності використання паперових квитків. На рисунку 1.3 наведено зображення електронного квитка, який діє в місті Києві.

Рисунок 1.3 – Приклад електронного квитка

Серед ключових переваг смарт-квитків можна виділити наступне:

1) зручність покупки: пасажири можуть придбати квитки перед поїздкою, що прискорює процес посадки і скорочує час очікування у черзі;

2) безпека: Електронні квитки важче підробити, і в разі втрати їх можна миттєво заблокувати, а гроші повернути. Це робить систему більш надійною для пасажирів;

3) лояльність та індивідуальні пропозиції: Оператори можуть створювати власні програми лояльності та пропонувати різні типи квитків, враховуючи потреби та бажання клієнтів;

4) можливість докупу послуг: пасажири можуть докупати додаткові послуги під час поїздки або на місці призначення, що розширює їхні можливості та забезпечує більш комфортний досвід подорожування.

Смарт-квитки дозволяють оптимізувати систему продажу та користування проїздом, підвищуючи зручність та безпеку для пасажирів і надаючи операторам можливість розвивати нові сервіси.

В автобусах з електронною системою квитків, для оплати достатньо прикласти картку до зчитувального пристрою. Система автоматично проводить валідацію карти та стягує оплату за проїзд.

Програми дисконтних карток — це стимулюючі плани, які дозволяють роздрібному бізнесу збирати дані про своїх клієнтів. Учасники отримують повідомлення про знижки, купони, бонуси або інші винагороди у обмін на участь у програмі. Головна мета таких програм — створити лояльність клієнтів та надати їм унікальні пропозиції. Дисконтні карти можуть мати форму пластикових карток, ключових брелків або наклейок (рисунок 1.4). Зазвичай на них є штрих-код або магнітна смужка, які скануються при покупці.

Рисунок 1.4 – Приклад дисконтної карти

Інформація про покупки та звички клієнтів передається до бази даних, що допомагає продавцям зрозуміти потреби своїх клієнтів. Багато покупців використовують дисконтні програми, але це може вимагати від них певної відмови від конфіденційності.

Програми дисконтних карток стали популярними в бізнесі та роздрібній торгівлі як спосіб привернути та утримати клієнтів.

Google Pay та Apple Pay є двома популярними мобільними платіжними системами, які спрощують процес оплати за товари та послуги за допомогою смартфонів. Ось деякі основні відомості про них:

Google Pay:

- Google Pay об'єднав попередні платіжні сервіси Android Pay і Google Wallet;
- дозволяє користувачам Android використовувати свої смартфони для проведення оплати, а також зберігати посадкові талони та квитки;
- забезпечує безпеку операцій за допомогою шифрування та безпечних серверів Google;
- можливість проводити оплати в інтернеті, в додатках та в магазинах, які підтримують Google Pay.

Apple Pay:

- розроблена компанією Apple для користувачів iPhone;
- використовує технологію NFC для проведення безконтактних оплат в магазинах;
- включає програму Apple Cash для надсилання грошей друзям у США;
- введена кредитна картка Apple Card з інтеграцією в Apple Pay та додатком Wallet;
- використовує "токенізацію" для забезпечення безпеки транзакцій та заміни фактичних номерів кредитних карток.

Обидві системи спрощують оплату та забезпечують високий рівень безпеки. Платіжна система Apple Pay також використовує розпізнавання обличчя Face ID для підтвердження платежів. Google Pay і Apple Pay підтримують безконтактні платежі на основі NFC, що робить їх досить зручними для користувачів.

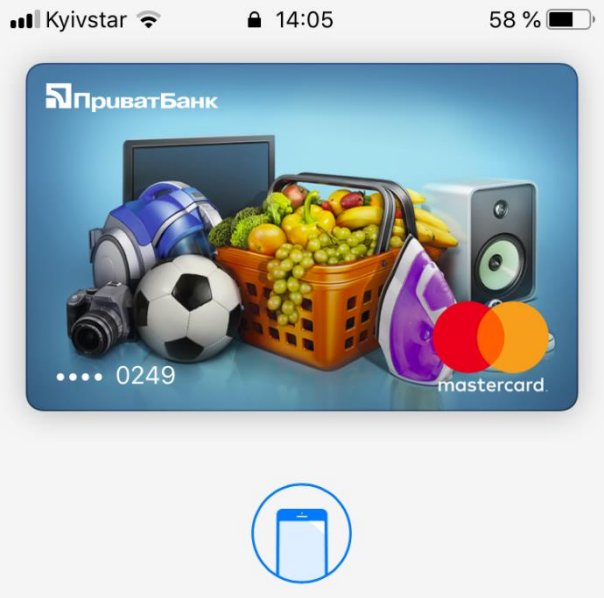

Рисунок 1.5 – Приклад інтерфейсу Apple Pay

Системи контролю доступу включають в себе численні аспекти і можуть принести багато переваг у сфері безпеки та управління доступом. Вам слід також звернути увагу на інші ключові переваги цих систем:

1) моніторинг та аудит: Системи контролю доступу дозволяють вести облік, коли і де персонал входить і виходить. Ця інформація може бути використана для аудиту та вирішення можливих безпекових проблем;

2) захист важливих ресурсів: Важливі об'єкти, такі як серверні кімнати або лабораторії з обмеженим доступом, можуть бути захищені від несанкціонованого доступу завдяки системам контролю доступу;

3) гнучкість і реактивність: Ви можете легко змінити права доступу або скасувати картки доступу у разі необхідності. Це надає вам можливість швидко реагувати на зміни в персоналі або інші обставини;

4) інтеграція з іншими системами: Системи контролю доступу можуть бути легко інтегровані з іншими системами безпеки, такими як системи відеоспостереження або системи виявлення злому, для забезпечення більш високого рівня безпеки;

5) біометричні технології: Біометричні методи, такі як розпізнавання відбитків пальців чи обличчя, забезпечують надійний та зручний спосіб ідентифікації осіб.

Загалом, системи контролю доступу стали невід'ємною частиною сучасних заходів забезпечення безпеки в багатьох сферах, від корпоративних офісів до медичних установ і об'єктів громадського харчування.

1.4 Постановка задачі дослідження

Для вирішення поставленої задачі, запропоновано створити систему, яка об'єднує користувачів і установи, що надають послуги. Кожній установі та користувачу в системі може бути створений обліковий запис. Установа має можливість визначати необхідну інформацію, яку повинен надати користувач, щоб скористатися її послугами. Також установа може створювати власні категорії користувачів і розподіляти користувачів між ними, регулюючи права та можливості клієнтів. У свою чергу, користувач може вибирати установи, якими він бажає скористатися, надавши необхідну інформацію та створивши віртуальну картку установи.

Ідентифікацію користувача пропонується здійснювати за допомогою зчитувального пристрою NFC, підключеного до електронної системи, та NFC мітки, розташованої у користувача. Кожен користувач отримує унікальний ідентифікаційний номер, який записується на NFC мітку. Це дозволяє системі однозначно ідентифікувати користувача, який у даний момент прикладає мітку до зчитувального пристрою і надавати відповідну інформацію про користувача установі чи автоматизованому пристрою.

Для реалізації цієї системи, NFC мітку у формі імпланта можна помістити під шкіру користувача, що зараз вже практикується деякими ентузіастами через її малі розміри. Однак наразі не багато користувачів готові до такої операції. Тому пропонується використовувати NFC мітки у

будь-якій формі, на вибір користувача, наприклад, у вигляді картки, брелока, кільця тощо.

Ця система має потенціал знайти застосування в різних сферах, таких як:

- каси магазинів та супермаркетів, де користувачі можуть швидко та зручно оплачувати покупки та надавати інформацію про свої лояльнісні картки;
- громадський транспорт, де пасажири можуть швидко оплачувати проїзд і надавати необхідну інформацію про свій квиток;
- підприємства, які можуть використовувати систему для контролю доступу та розподілу прав користувачів на своїй території;
- медичні установи, де працівники швидкої допомоги можуть отримувати важливу інформацію про пацієнта при надходженні.

2 БЕЗПЕКА ПЕРЕДАЧІ ДАНИХ ПІД ЧАС ІДЕНТИФІКАЦІЇ КОРИСТУВАЧІВ НА ОСНОВІ ТЕХНОЛОГІЇ NFC

2.1 Загрози при передачі даних із використання технології NFC

В рамках дослідження безпеки технології NFC була розглянута можливість проведення різних атак на канал передачі даних. Перелік атак представлений в таблиці 2.1 [30].

Таблиця 2.1 - Безпека технології NFC

Атаки на бездротові канали	Актуальність для технології NFC	Методи захисту
Пасивне прослуховування	+	Криптографічні методи
Пошкодження даних	+	Криптографічні методи
Модифікація даних	багато обмежень	Криптографічні методи
Вставка даних	багато обмежень	Криптографічні методи
Relay атаки	+	Екранування, Авторизація користувача
MITM атаки	-	-

Наведений перелік відображає можливість атак на незахищений за замовчуванням канал передачі даних. Уникнути багатьох таких атак можливо на рівні додатків, використовуючи криптографію та інші засоби захисту. Оскільки NFC є бездротовою технологією, існує загроза прослуховування каналу. Під час з'єднання двох пристроїв, вони взаємодіють за допомогою

радіохвиль. Потенційний зловмисник може використовувати спрямовану антену для прослуховування передаваних сигналів. Шляхом експериментів або вивчення специфікацій протоколів взаємодії пристроїв атакуючий може виявити, як отримати інформацію з перехопленого сигналу. Слід відзначити, що існують доступні на ринку пристрої для перехоплення та розкодування радіочастотних сигналів.

Також важливо враховувати режим роботи передавача даних. У залежності від активного чи пасивного режиму роботи, методи перехоплення даних можуть значно відрізнятися. Загалом можна сказати, що в активному режимі радіус можливого прослуховування може сягати 10 метрів, тоді як в пасивному режимі ця відстань обмежується 1 метром.

В технології NFC обмін інформацією відбувається лише при безпосередньому контакті пристроїв на відстані не більше 10 см. Однак питання про те, наскільки близько має знаходитися потенційний зловмисник, щоб успішно перехопити радіочастотний сигнал, який було б можливо подальше використовувати, не має однозначної відповіді. Ця невизначеність обумовлена безліччю факторів, що впливають на відстань, на якій можливе прослуховування, таких як характеристики передавача та антени атакуючого, якість обладнання атакуючого, умови локації, наявність бар'єрів у вигляді стін і потужність NFC пристрою. Оцінка цих параметрів робить неможливим надання універсального значення, яке підходило б до більшості ситуацій.

Також важливо враховувати режим роботи передавача даних. У залежності від активного чи пасивного режиму роботи, методи перехоплення даних можуть значно відрізнятися. Загалом можна сказати, що в активному режимі радіус можливого прослуховування може сягати 10 метрів, тоді як в пасивному режимі ця відстань обмежується 1 метром

На рисунку 2.1 зображена схема пасивного прослуховування.

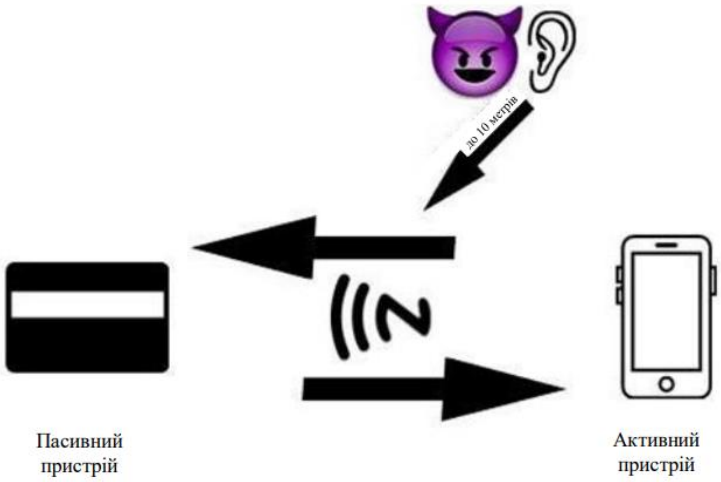

Рисунок 2.1 - Схема пасивного прослуховування

Крім перехопленням даних зловмисники можуть також намагатися модифікувати передачу даних через NFC канал. У простому випадку, атакуючий може спробувати перервати комунікацію між пристроями, переповнивши канал випадковими даними, заважаючи таким чином приймачеві розрізнити коректні дані, що надходять від законного пристрою. На схемі, представленій на рисунку 2.2, проілюстровано такий можливий сценарій атаки.

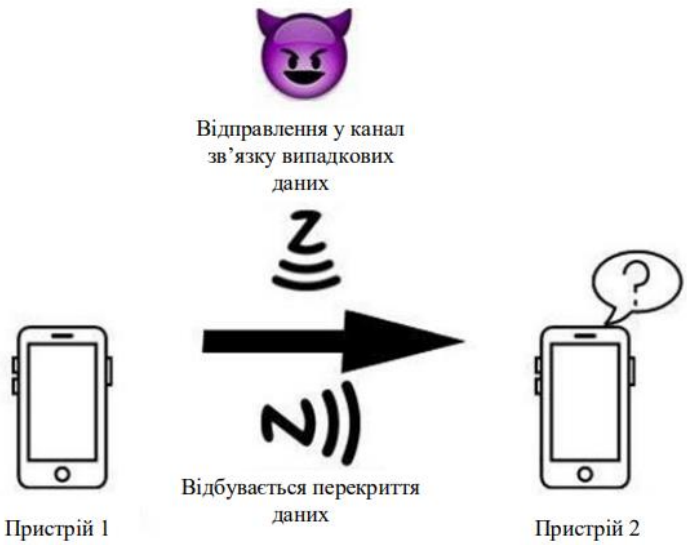

Рисунок 2.2 - Схема атаки пошкодження даних

Задля пошкодження даних, атакуючий може відправляти правильні радіосигнали в конкретний проміжок часу. Цей проміжок часу визначається залежно від використовуваної модуляції та кодування. Така атака використовується для переривання комунікації між пристроями, переповнюючи канал випадковими даними. В результаті цього діяльності приймаючий пристрій не може коректно розрізнити дані, надіслані від легітимного пристрою, і не може їх обробити. Така атака може бути вважається підвидом атаки відмови в обслуговуванні.

Для модифікації передачі даних атакуючий має на увазі зміну отримуваних даних на їх шляху. Залежно від коефіцієнта модуляції мають місце різні підходи до цієї атаки. Наприклад, для 100% модуляції, декодер перевіряє наявність радіочастотного сигналу (відсутність паузи) або відсутність сигналу (пауза) на двох послідовних полубітах.

Атака "вставка даних" полягає в тому, що зловмисник намагається вставити своє повідомлення в канал передачі даних між двома пристроями, які здійснюють NFC-зв'язок. NFC передача даних відбувається на принципі "повідомлення-відповідь". Якщо пристрій, який має відповісти, потребує додатковий час для генерації відповіді, то зловмисник може вставити своє повідомлення в канал, і пристрій, який ініціює зв'язок, сприймає це як легітимну відповідь. Схема цієї атаки показана на рисунку 2.3.

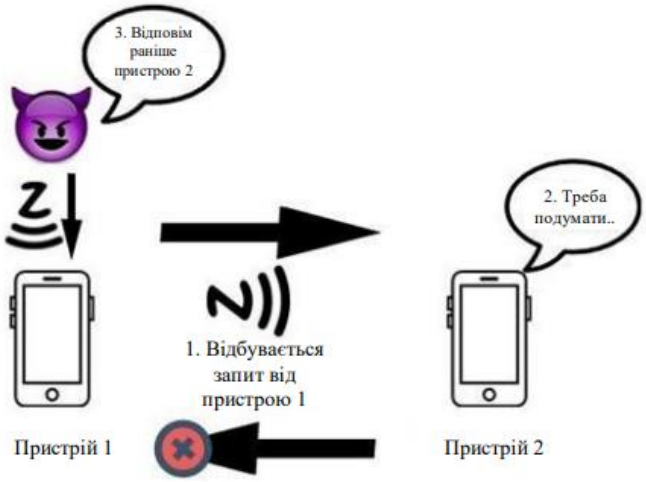

Рисунок 2.3 -Схема атаки за допомогою вставки даних

Атака "людина посередині", або MITM (Man-in-the-Middle), включає в себе ситуацію, коли два суб'єкти, наприклад, Аліса і Боб, спілкуються один з одним, і зловмисник Єва стає "посередині" їхнього зв'язку. Це означає, що Єва може перехоплювати та читати їхні повідомлення, а також модифікувати дані. Ні Аліса, ні Боб не підозрюють про цю атаку, оскільки вони взаємодіють безпосередньо один з одним, не знаючи про присутність Єви в середині зв'язку. Схему цієї атаки можна побачити на рисунку 2.4.

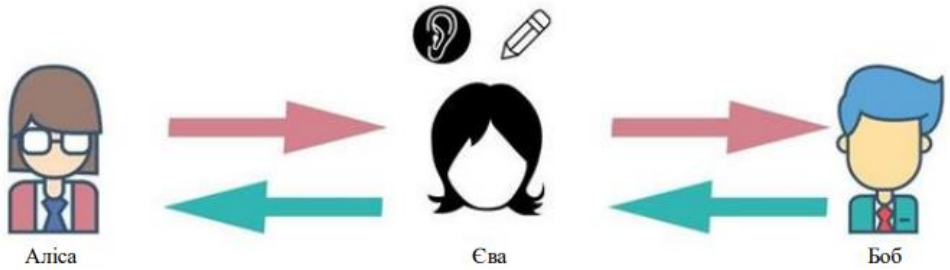

Рисунок 2.4 - Загальна схема атаки MITM

Даний сценарій є типовим для вразливих каналів, які не зашифровані або де відсутні аутентифікація та обмін ключами. Наприклад, в алгоритмі Діффі-Хеллмана можлива ситуація, де Аліса і Боб обмінюються ключами, але зловмисник Єва може встановити окремий ключ із Алісою і окремий із Бобом. У подальшому, коли Аліса намагається відправити дані Бобу, Єва може перехоплювати ці дані, розшифровувати їх за допомогою ключа Аліси, потім знову шифрувати за ключем Боба і передавати Бобу. Це призводить до того, що як Аліса, так і Боб вважають, що вони взаємодіють у безпечному каналі, тоді як насправді всі дані проходять через посередника.

Зрозуміло, що схожий сценарій може виникнути і для NFC-каналу. Наприклад, якщо Аліса працює в активному режимі, а Боб в пасивному, і Єва знаходиться вблизьк, вона може прослуховувати дані, які відправляються від Аліси до Боба, і, при бажанні, перервати цю передачу. Але такий вид атаки

може бути виявлений, оскільки Аліса може відзначити спробу перервати передачу та призупинити обмін даними.

Однак, якщо враховувати, що на стороні Аліси відсутня перевірка на переривання передачі даних, і обмін ключами продовжується, то наступний етап полягає у передачі даних від Єви до Боба. Але через те, що РЧ поле, створене Алісою, все ще існує, і для передачі даних Єва повинна генерувати власне РЧ поле, що викликає проблему накладання полів. На практиці це ускладнює можливість Єви надіслати повідомлення Бобу, оскільки їй складно створити РЧ поле, яке коректно зрозуміється Бобом.

В іншій конфігурації, коли і Аліса і Боб працюють в активному режимі, Єва може знову намагатися перервати повідомлення від Аліси. Якщо перевірка на переривання відсутня і Аліса не помічає спробу перервання, то Єва може продовжити наступним кроком. Тепер Єва має передати повідомлення Бобу. Але коли Аліса завершує передачу та вимикає своє РЧ поле, вона переходить в режим прослуховування. В цьому режимі вона може отримати повідомлення, яке Єва відправляє Бобу, оскільки вона все ще знаходиться в зоні зчитування. У цьому випадку Єва не може надіслати повідомлення тільки Бобу або тільки Алісі; обидва пристрої отримають повідомлення.

Для здійснення атаки зловмиснику потрібно мати фізичний доступ до смарт-картки [6]. Схему relay-атаки зображено на рисунку 2.5.

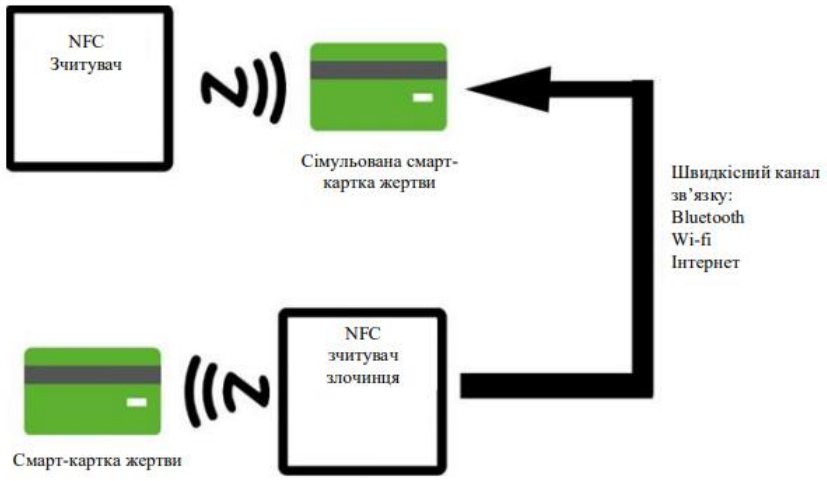

Рисунок 2.5 - Схема relay-атаки

Атака ускладнюється через можливі тимчасові затримки, що виникають внаслідок використання стороннього каналу, відповідно до стандарту ISO 14443-4, які коливаються від 302 мікросекунд до 4949 мілісекунд. Для захисту від розглянутих раніше атак зазвичай використовується криптографія на рівні додатку.

Захист від модифікації даних може бути досягнутий різними способами. Один із них - використання активного режиму і швидкості в 106 кілобіт на секунду, що може запобігти зміні бітів в деяких ситуаціях. Активний режим також може бути використаний для часткового захисту від модифікації даних, але він слабко захищений від прослуховування. Крім того, передавальний NFC-пристрій може слідкувати за РЧ полем під час передачі даних і при виявленні підозрілих атак припиняти передачу. Найбільш ефективним методом захисту від модифікації даних є використання криптографічно захищеного каналу.

Щодо захисту від атаки "вставка даних", можна вживати такі заходи, як надсилання відповіді без затримки, щоб ускладнити проведення атаки, де довгий час генерації відповіді є обов'язковим. Проте, цей метод може бути неефективним, якщо для роботи програми необхідні обчислення. Варіантом є також перевірка каналу під час підготовки відповіді і припинення спілкування при виявленні сторонніх повідомлень. Все-таки, найбільш дієвим і рекомендованим способом захисту є використання криптографічно захищеного каналу.

Щодо захисту смарт-карток від цього типу атак, можуть бути використані наступні методи:

- клітки Фарадея: Екранування карток, коли вони не використовуються, може запобігти незаконному зчитуванню даних;

- застосування цифрового підпису для передачі даних через NFC може забезпечити їх цілісність та аутентифікацію. Це важливо для гарантування безпеки обміну інформацією через цей канал.

Використання протоколів визначення відстані також може виявитися корисним для перевірки відстані між двома NFC-пристроями, що уникне можливих атак, спрямованих на встановлення "людини посередині" шляхом розширення діапазону дії NFC.

Щоб створити безпечний канал для обміну даними через NFC, рекомендується використовувати стандартні протоколи обміну ключами, такі як алгоритм Діффі-Хеллмана, базований на RSA або еліптичних кривих. Після установки загального секретного ключа можна використовувати симетричні алгоритми шифрування, такі як 3DES або AES, для передачі секретних даних. Це забезпечить конфіденційність, цілісність та аутентифікацію передаваних даних.

Загалом, NFC відкриває можливості для створення безпечного каналу обміну даними, а використання захищених протоколів і методів гарантує високий рівень безпеки під час комунікації через цей канал.

Ця концепція передбачає використання випадкових даних та синхронізації між двома взаємодіючими пристроями для підвищення безпеки обміну даними і ускладнення можливості атаки "людина посередині". Під час налаштування з'єднання ці пристрої синхронізують свою передачу щодо швидкості, амплітуди та фази радіочастотного сигналу. Це досягається завдяки тому, що обидва пристрої починають передавати дані одночасно. Після такої синхронізації пристрої розпочинають синхронну передачу даних та одночасно слухають сигнали, передані іншим пристроєм.

Якщо обидва пристрої відправляють одиниці, сумарний сигнал буде підсумком двох одиниць, і атакуючий може легко перехопити цей сигнал. Аналогічно, атакуючий може перехопити одночасну передачу нулів, оскільки сумарний сигнал буде рівним нулю. Проте, якщо обидва пристрої відправляють різні сигнали, атакуючому стане важко визначити, який саме сигнал відправив кожен пристрій. Тим часом обидва пристрої, які знають власні дані, зможуть розпізнати та інтерпретувати сигнали правильно. Вищезгадані випадки проілюстровані на рисунку 2.6.

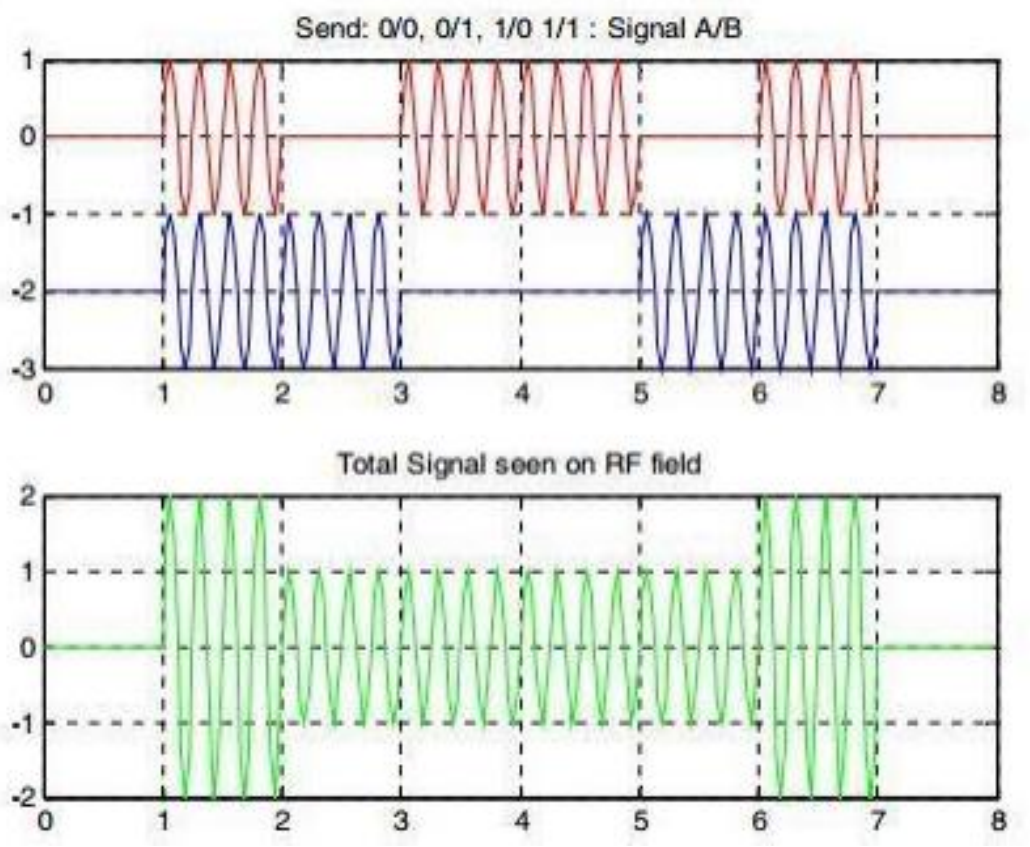

Рисунок 2.6 - Загальний сигнал, одержуваний на виході

На верхньому графіку можна спостерігати сигнал пристрою 1 (відображений червоним кольором) та сигнал пристрою 2 (відображений синім кольором). На нижньому графіку представлений результуючий сигнал, який може бути зафіксований атакуючим.

2.2 Дослідження технології NFC в мобільних пристроях

Більшість Android-смартфонів, які підтримують NFC, також мають можливість використовувати технологію HCE (Host Card Emulation), щоб емулювати NFC картки. Ця функція була впроваджена в ОС Android, починаючи з API версії 19 (KitKat 4.4). Раніше емуляція карток була можлива, але використовувалася технологія Secure Element, яку показано на рисунку 2.7.

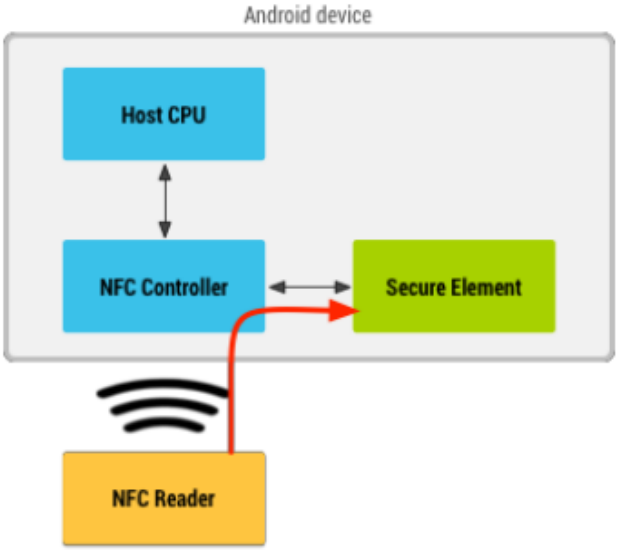

Рисунок 2.7 - Емуляція картки з використанням Secure Element

Емуляція картки з використанням Secure Element - це процес, при якому NFC-смартфон використовує вбудований захищений елемент (Secure Element) для емуляції NFC картки. Цей метод передбачає зберігання конфіденційної інформації, такої як дані кредитних карток, в спеціально забезпеченому місці на пристрої, що надає додатковий рівень безпеки.

За допомогою цієї технології, користувач може використовувати свій смартфон для безконтактних платежів або автентифікації на пристроях, підтримуючи NFC, якщо їх сумісність підтримує технологію Secure Element.

Згодом технологія HCE (Host Card Emulation) стала популярною альтернативою для емуляції NFC карток, яка не вимагає використання Secure Element. HCE дозволяє емулювати NFC картку, використовуючи програмне забезпечення на смартфоні, замість фізичного Secure Element. Це робить процес емуляції карток більш доступним і зручним для розробників додатків та користувачів.

Додаток, який встановлює образ карти на Secure Element, працює наступним чином: коли користувач подносить пристрій до зчитувача, NFC дані передаються безпосередньо в Secure Element. Після цього обробка транзакції відбувається в безпосередньому забезпеченому середовищі, і до

закінчення цього процесу користувач не має доступу до даних та дій, пов'язаних з транзакцією. Це забезпечує високий рівень безпеки.

Натомість, технологія HCE (Host Card Emulation) перенаправляє NFC дані безпосередньо до процесора пристрою, де запущені інші програми. Дані обробляються за допомогою компонентів, відомих як HCE-сервіси. Ця обробка відбувається в менш захищеному середовищі, де користувач може мати деякий рівень доступу до ходу транзакції. Отже, основна відмінність полягає в тому, де відбувається обробка NFC транзакцій і наскільки захищеним є середовище для цієї обробки. Secure Element забезпечує вищий рівень фізичної безпеки, тоді як HCE використовує технічні засоби для забезпечення безпеки у програмному режимі (рисунок 2.8).

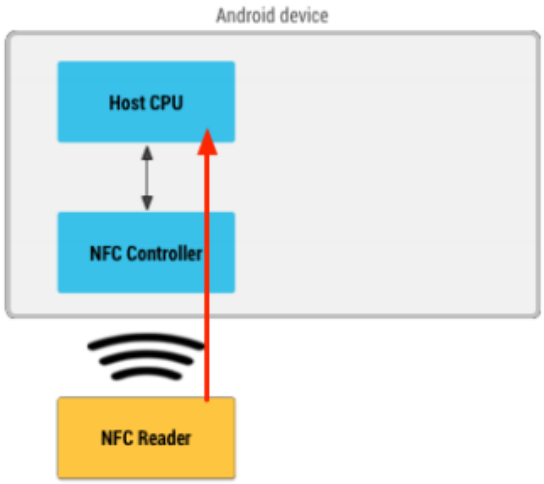

Рисунок 2.8 - Емуляція картки за допомогою HCE Android

HCE підтримується на смартфонах, що працюють під управлінням операційних систем Windows та Blackberry. Ця технологія спрощує емуляцію NFC карток і дозволяє реалізувати різноманітні протоколи, які широко використовуються в сучасних системах зв'язку.

Стек протоколів (рисунок 2.9), підтримуваних реалізацією HCE в ОС Android, включає різні комунікаційні та безпекові протоколи, які дозволяють взаємодіяти з різними типами NFC-читачів та інших пристроїв. Ця гнучкість та розширення можливостей робить HCE привабливим вибором для

розробників, які прагнуть впровадити NFC-функціональність у свої додатки, незалежно від операційної системи пристрою.

Коли пристрій взаємодіє з NFC-зчитувачем, операційна система Android автоматично визначає, який із HCE-сервісів повинен взаємодіяти з NFC-зчитувачем. Це визначення відбувається за допомогою унікального ідентифікатора, відомого як AID (Application Identifier). Кожному сервісу HCE надається свій унікальний AID, який вказує, який обробник слід активувати. Процес вибору сервісу обробника показаний на рисунку 2.10.

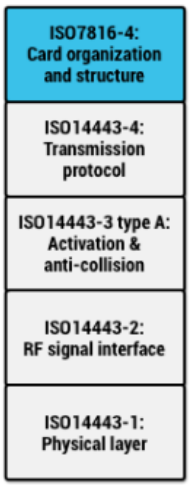

Рисунок 2.9 – Стек протоколів, які підтримуються реалізацією HCE

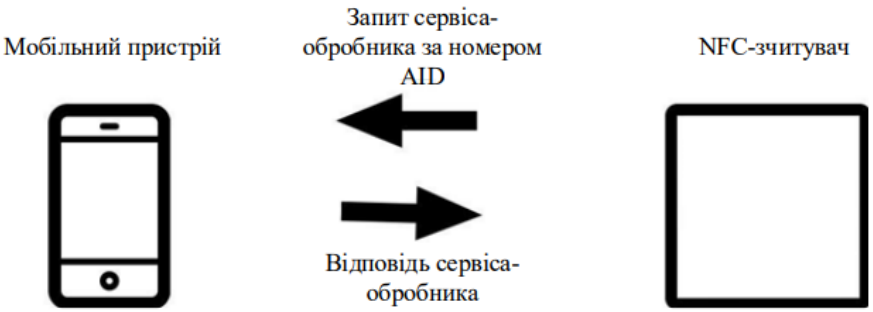

Рисунок 2.10 - Процес вибору сервісу-обробника HCE

Розмір AID (Application Identifier) може сягати до 16 байтів і використовується для ідентифікації конкретних NFC-сервісів. Є спеціальні групи AID, які зарезервовані для використання в інфраструктурах, таких як

Google Wallet, Visa і Master Card, а також ідентифікатори, які можна використовувати для власних цілей.

2.3 Проектування NFC- тегу

Проектування NFC-тегу - це процес розробки і створення пристрою, який може взаємодіяти з пристроями, обладнаними NFC-зчитувачами. Цей процес включає в себе декілька важливих етапів, що дозволяють створити NFC-тег, який відповідає конкретним потребам і вимогам проекту.

Початковим кроком є визначення цілей та функціональності NFC-тегу. Важливо зрозуміти, яку саме роль він виконуватиме в системі. Наприклад, це може бути передача URL-адреси, контактних даних або використання тегу для контролю доступу. Вибір правильного типу NFC-тегу, такого як мітка, картка або брелок, залежить від конкретних вимог проекту.

Далі слід вибрати підходящий NFC-чіп, який найкраще відповідає потребам проекту. Різні NFC-чіпи мають різні характеристики та можливості, тому вибір важливий.

Збереження та форматування даних на NFC-тезі - ще один важливий аспект проектування. Визначення, які саме дані будуть збережені на тезі (текст, URL, контактні дані тощо) і їх форматування відповідно до обраних стандартів важливо для забезпечення сумісності.

Захист та безпека також потребують уваги. Розгляньте можливості захисту даних на NFC-тезі від несанкціонованого доступу, такі як методи шифрування чи захист паролем, якщо це важливо для вашого застосування.

Програмування NFC-тегу включає в себе запис даних та функціональності на тег. Використовуйте відповідне програмне забезпечення або інструменти для цього.

Перевірка та тестування важливі для переконання, що NFC-тег працює належним чином та відповідає вимогам проекту. Важливо провести тестування на різних NFC-зчитувачах та пристроях.

Після успішного завершення проекту можна перейти до виробництва NFC-тегів і їх розповсюдження для використання в конкретному застосуванні. Детальні інструкції та документація для користувачів також важливі для забезпечення правильного використання NFC-тегу.

Теги NFC та RFID отримують свою енергію від магнітного поля зчитувача, через взаємодію індуктивностей між тегом та антеною зчитувача, аналогічно трансформатору. Ефективність передачі енергії від зчитувача до мітки залежить від характеристик петлевої антени, яка налаштована на несучу частоту, зазвичай 13,56 МГц.

Антени 13,56 МГц можуть мати різну форму, відповідно до вимог конкретного застосування. Основним параметром є еквівалентна індуктивність Lant антени 13,56 МГц. Ємність зазвичай складає всього кілька пікофарад для типових продуктів NFC / RFID. Відповідні значення індуктивності та ємності зазвичай наводяться в документації до NFC-чіпу.

Оскільки основні форми NFC-міток, такі як картки, брелоки або браслети, доступні для вільного продажу, розглянемо проектування мітки у формі кільця. Надалі наведено формули для розрахунку індуктивності антени, яка відповідає обраній формі.

Кругова антена:

$L_{ant} = \mu_0 \times N^{1.9} \times r \times \ln\left(\frac{r}{r_0}\right)$, де:

r – середній радіус антени в міліметрах,

r_0 – діаметр провідника в міліметрах,

N – кількість обертів,

$\mu_0 = 4\pi \times 10^{-7} H/m$,

L вимірюється в Генрі.

Спіральна антена:

$L_{ant} = 31.33 \times \mu_0 \times N^2 \times \frac{a^2}{8a+11c}$, де (рисунок 2.11):

$a = (r_{in} + r_{out})/2$ (середній радіус, в метрах)

$c = r_{out} - r_{in}$, в метрах

$$\mu_0 = 4\pi \times 10^{-7} H/m,$$

L вимірюється в Генрі [12].

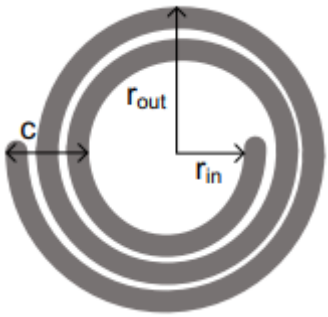

Рисунок 2.11 – Спіральна антена

Для забезпечення зчитування NFC тегу ви можете використовувати універсальний модуль PN532 NFC RFID разом із мікроконтролером Arduino Nano. Ця комбінація дозволить вам створити NFC-зчитувач та інтегрувати його з Arduino для взаємодії з NFC-тегами.

Ось деякі кроки, які можна виконати для реалізації цього проекту:

1) підключіть модуль PN532 до Arduino Nano. Зазвичай, PN532 підключається через інтерфейс SPI або I2C. Переконайтесь, що ви підключили всі необхідні піни (наприклад, SDA, SCL для I2C або MOSI, MISO, SCK, і SS для SPI);

2) встановіть бібліотеку PN532 для Arduino. Ви можете знайти багато доступних бібліотек для роботи з модулем PN532 в середовищі Arduino. Встановіть відповідну бібліотеку, щоб спростити роботу з модулем;

3) напишіть програмний код для Arduino, який буде взаємодіяти з модулем PN532 і зчитувати дані з NFC-тегів. Ваш код повинен включати функції для ініціалізації PN532, зчитування NFC-тегів і обробки отриманих даних;

4) завантажте програмний код на Arduino Nano та переконайтесь, що модуль PN532 правильно ініціалізується і зчитує дані з NFC-тегів.

Після успішного завершення проекту можна використовувати Arduino Nano та модуль PN532 для зчитування NFC-тегів у додатку.

3 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ ДЛЯ ІДЕНТИФІКАЦІЇ КОРИСТУВАЧА НА ОСНОВІ ТЕХНОЛОГІЇ NFC

3.1 Проектування програмної системи

Проектування програмної системи є складним процесом, що передбачає визначення архітектури, компонентів, інтерфейсів та інших ключових характеристик системи для досягнення бажаного кінцевого результату.

На першому етапі проектування, ми створюємо діаграму прецедентів, використовуючи UML (Unified Modeling Language). Ця діаграма відображає взаємозв'язок між акторами (користувачами системи) та прецедентами (функціональними можливостями системи). Діаграма прецедентів є частиною прецедентної моделі і служить для концептуального опису системи.

У цьому контексті, прецеденти визначають можливості, які надає система, і яким чином користувач може досягти конкретних, вимірюваних результатів за допомогою цих можливостей. Кожен прецедент відповідає окремому сервісу або функції системи і описує типовий спосіб взаємодії користувача з системою. Він допомагає визначити зовнішні системні вимоги та потреби користувачів.

Основною метою діаграми прецедентів є опис функціональності та поведінки системи, що дозволяє замовнику, кінцевому користувачеві та розробникам спільно обговорювати та розуміти проектовану систему. Цей інструмент допомагає зрозуміти, як користувачі взаємодіють з системою та як вона задовольняє їхні потреби та очікування.

На підставі вимог до системи були виділені наступні актори системи:

- незареєстрований користувач: це особа, яка відвідує сайт системи вперше. Вона має можливість ознайомитися з системою та, в подальшому, зареєструватися для отримання доступу до основних функцій системи;

- зареєстрований користувач: Це користувач, який вже має створений профіль у системі. Він має доступ до основного функціоналу системи після авторизації;
- адміністратор: Адміністратор є одним із зареєстрованих користувачів, але він має повний доступ до системи та бази даних. Адміністратор може переглядати інформацію та статистику роботи системи, а також налагоджувати її роботу;
- менеджер установи: Менеджер установи є також зареєстрованим користувачем і може бути директором або уповноваженою особою установи. Він має можливість створити власну установу в системі, налаштувати картку установи, а також призначити інших менеджерів та співробітників для роботи з установою;
- співробітник установи: Цей користувач також зареєстрований у системі і має дозвіл від менеджера установи. Він працює з клієнтами установи, отримує інформацію з картки клієнта та має можливість змінювати категорію, до якої відноситься клієнт;
- клієнт установи: Це також зареєстрований користувач, який створив картку установи. Він має можливість бути ідентифікованим як співробітник установи та отримувати певні послуги від установи;
- кожен із цих акторів взаємодіє з системою відповідно до своїх ролей і функцій у контексті проекту.

На основі опису акторів системи та вимог до системи було розроблено загальну схему прецедентів, яка наведена на рисунку 3.1.

Сценарії прецедентів можуть бути виражені в різних формах, таких як структурований неформальний текст, формалізований структурований текст, псевдокод, таблиці або діаграми активностей. Кожен сценарій описує завершену та конкретну взаємодію, спрямовану на досягнення певної мети з точки зору користувача. У випадку використання табличної форми для представлення сценарію, лінія, яка розділяє ліву та праву частину таблиці, відділяє дії користувача від дій системи.

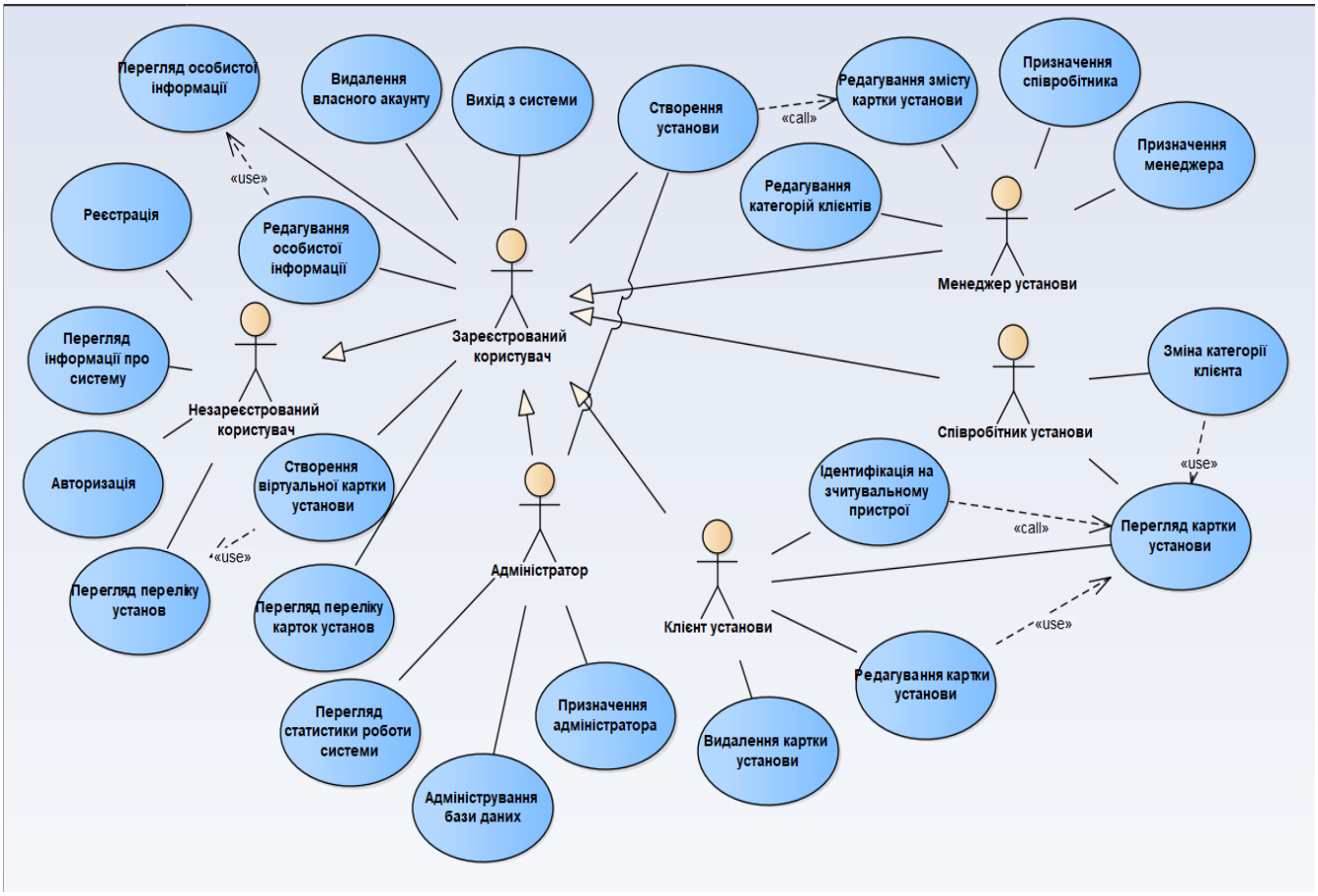

Рисунок 3.1 – Загальна схема прецедентів

Використання табличної форми особливо підкреслює роль користувача, що є критичним аспектом при розробці користувацького інтерфейсу. Саме тому для детального опису сценаріїв функціонування розроблюваної системи було обрано табличний підхід. При складанні табличних сценаріїв важливо дотримуватися чіткості та конкретності, вказувати вхідні та вихідні дані, передбачати різні сценарії взаємодії з системою та враховувати різні варіанти введення даних. Такий підхід полегшить розробку та тестування системи, сприяючи високій якості користувацького інтерфейсу та загальної ефективності системи.

Нижче в таблицях 3.1 - 3.10 подано основні сценарії прецедентів системи.

Таблиця 3.1 - Сценарій реєстрації користувача

Назва	Реєстрація		
Учасники	Незареєстрований користувач, Система		
Результат	Користувач зареєстрований		
Передумови	Немає		
Основний сценарій		Дії користувача	Дії системи
	1	Перехід на сторінку Реєстрації	
	2		Повернення форми
	3	Заповнення полей «Ім’я», «Прізвище», «По-батькові», «Дата народження», «Місце проживання», «Логін», «Номер телефону», «Email», «Пароль».	
	4	Підтвердження правильності введення	
	5		Проведення валідації даних
	6		Перехід на головну
Виключні ситуації	Данні не пройшли валідацію, користувач з такими даними існує		

Таблиця 3.2 - Сценарій авторизації користувача

Назва	Авторизація		
Учасники	Зареєстрований користувач, Система		
Результат	Користувач увійшов в систему		
Передумови	Користувач зареєстрований		
Основний сценарій		Дії користувача	Дії системи
	1	Перехід на сторінку Вхід	
	2		Повернення форми для Входу
	3	Заповнення полей «Логін», «Пароль»	

Продовження таблиці 3.2

Основний сценарій		Дії користувача	Дії системи
	4	Підтвердження правильності введення даних	
	5		Проведення валідації даних
	6		Перехід на головну сторінку
Виключні ситуації	Данні не пройшли валідацію, користувач з такими данними не існує		

Таблиця 3.3 - Сценарій перегляду переліку карток установ

Назва	Перегляд переліку карток установ		
Учасники	Зареєстрований користувач, Система		
Результат	Користувач отримує дані установ, клієнтом яких він є		
Передумови	Користувач виконав авторизацію		
Основний сценарій		Дії користувача	Дії системи
	1	Натискання на кнопку «Мої картки»	
	2		Перехід на сторінку з переліком карток установ
	3		Відображення списку установ, клієнтом яких є користувач, зі стислою інформацією про установу та її логотип
Виключні ситуації	Користувач не має картки жодної з установ		

Таблиця 3.4 - Сценарій створення віртуальної картки установи

Назва	Створення віртуальної картки установи		
Учасники	Зареєстрований користувач, Система		
Результат	Створена нова картка установи		
Передумови	Користувач перейшов на сторінку установ		
Основний сценарій		Дії користувача	Дії системи
	1	Введення назви установи в поле «Пошук»	
	2		Відображення установ, які відповідають критеріям пошуку

Продовження таблиці 3.4

Основний сценарій		Дії користувача	Дії системи
	3	Вибір установи	
	4	Натискання кнопки «Створити картку»	
	5		Повернення форми для створення картки
	6	Заповнення всіх обов’язкових полів	
	7		Проведення валідації даних
	8		Створення віртуальної картки установи
Виключні ситуації	Дані не пройшли валідацію, картка даної установи вже існує у користувача		

Таблиця 3.5 - Сценарій створення установи

Назва	Створення установи			
Учасники	Зареєстрований користувач, Система, Адміністратор			
Результат	Створена нова установа			
Передумови	Користувач перейшов на сторінку партнерства, адміністратор перейшов на сторінку заявок			
Основний сценарій		Дії користувача	Дії системи	Дії адміністратора
	1	Натискання кнопки «Створити установу»		
	2		Повернення форми для створення установи	
	3	Заповнення полей «Назва», «Опис», «Контакти», «Логотип», «Посилання»		
	4	Натискання кнопки «Створити установу»		
	5		Проведення валідації даних	

Продовження таблиці 3.5

Основний сценарій		Дії користувача	Дії системи	Дії адміністратора
	6		Відображення установи на сторінці заявок	
	7			Натискання кнопки «Підтвердити»
	8		Створення установи	
Виключні ситуації	Дані не пройшли валідацію, адміністратор відмовляє у створенні установи			

Таблиця 3.6 - Сценарій перегляду статистики роботи системи

Назва	Перегляд статистики роботи системи		
Учасники	Адміністратор, Система		
Результат	Отримано інформацію про роботу системи		
Передумови	Адміністратор перейшов на сторінку адміністрування		
Основний сценарій		Дії адміністратора	Дії системи
	1	Натискання кнопки «Статистика»	
	2		Відображення інформації про кількість використаної пам’яті, запущених потоків, запитів користувачів
	3	Натискання кнопки «Ативні користувачі»	
	4		Відображення списку активних користувачів системи зі стислою інформацією про них
	5	Натискання кнопки «Стан системи»	
	6		Відображення інформації про кількість вільного місця на диску, стану бази даних
Виключні ситуації	В системі немає інших активних користувачів		

Таблиця 3.7 - Сценарій адміністрування бази даних

Назва	Адміністрування бази даних		
Учасники	Адміністратор, Система		
Результат	Внесено зміни до бази даних		
Передумови	Адміністратор перейшов на сторінку адміністрування		
Основний сценарій		Дії адміністратора	Дії системи
	1	Натискання кнопки «Таблиці»	
	2		Відображення списку таблиць бази даних
	3	Вибір таблиці бази даних	
	4		Відображення рядків таблиці
	5	Вибір рядку таблиці	
	6	Натискання кнопки «Редагувати»	
	7		Відображення форми редагування рядку таблиці
	8	Внесення змін у поля форми	
	9		Проведення валідації даних
	10		Збереження результатів до бази даних
Виключні ситуації	Дані не пройшли валідацію, зміни не можуть бути застосовані		

Таблиця 3.8 - Сценарій редагування змісту картки установи

Назва	Редагування змісту картки установи		
Учасники	Менеджер установи, Система		
Результат	Картка установи налаштована		
Передумови	Менеджер установи перейшов на сторінку картки установи		
Основний сценарій		Дії менеджера установи	Дії системи
	1	Натискання кнопки «Додати поле»	
	2		Відображення списку стандартних полів системи
	3	Вибір поля	
Основний сценарій		Дії менеджера установи	Дії системи
	4	Натискання кнопки «Підтвердити»	
	5		Додавання поля до картки установи
	6	Повторення попередніх дій доки картка не буде містити всі необхідні поля	
	7	Натискання кнопки «Зберегти картку»	
	8		Збереження змісту картки установи
Виключні ситуації	Серед списку стандартних полів немає необхідного		

Таблиця 3.9 - Сценарій редагування категорій клієнтів

Назва	Редагування категорій клієнтів
Учасники	Менеджер установи, Система
Результат	Додана нова категорія
Передумови	Менеджер установи перейшов на сторінку категорій клієнтів

Продовження таблиці 3.9

Основний сценарій		Дії менеджера установи	Дії системи
	1	Натискання кнопки «Додати категорію»	
	2		Повернення форми для додавання категорії
	3	Заповнення полів «Назва» та «Опис»	
	4	Підтвердження правильності введення даних	
	5		Проведення валідації даних
	6		Створення нової категорії
Виключні ситуації	Дані не пройшли валідацію		

Таблиця 3.10 - Сценарій зміни категорії клієнта

Назва	Зміна категорії клієнта		
Учасники	Співробітник установи, Система		
Результат	Клієнта віднесено до певної категорії		
Передумови	Клієнт установи пройшов валідацію на зчитувальному пристрої		
Основний сценарій		Дії співробітника установи	Дії системи
	1	Натискання на кнопку «Віднести до категорії»	
	2		Відображення списку категорій клієнтів установи
	3	Вибір категорії	
	4	Натискання кнопки «Підтвердити»	
	5		Додавання категорії до списку категорій картки користувача
Виключні ситуації	Клієнт вже відноситься до даної категорії, в установи немає категорій		

На основі отриманої інформації, яка була виявлена на етапах бізнес-моделювання, проводиться розробка концептуальної моделі даних, яка буде використовуватися в розроблюваній системі.

Основні сутності та взаємозв'язки між ними відображаються на діаграмі бізнес-сутностей (рисунок 3.2).

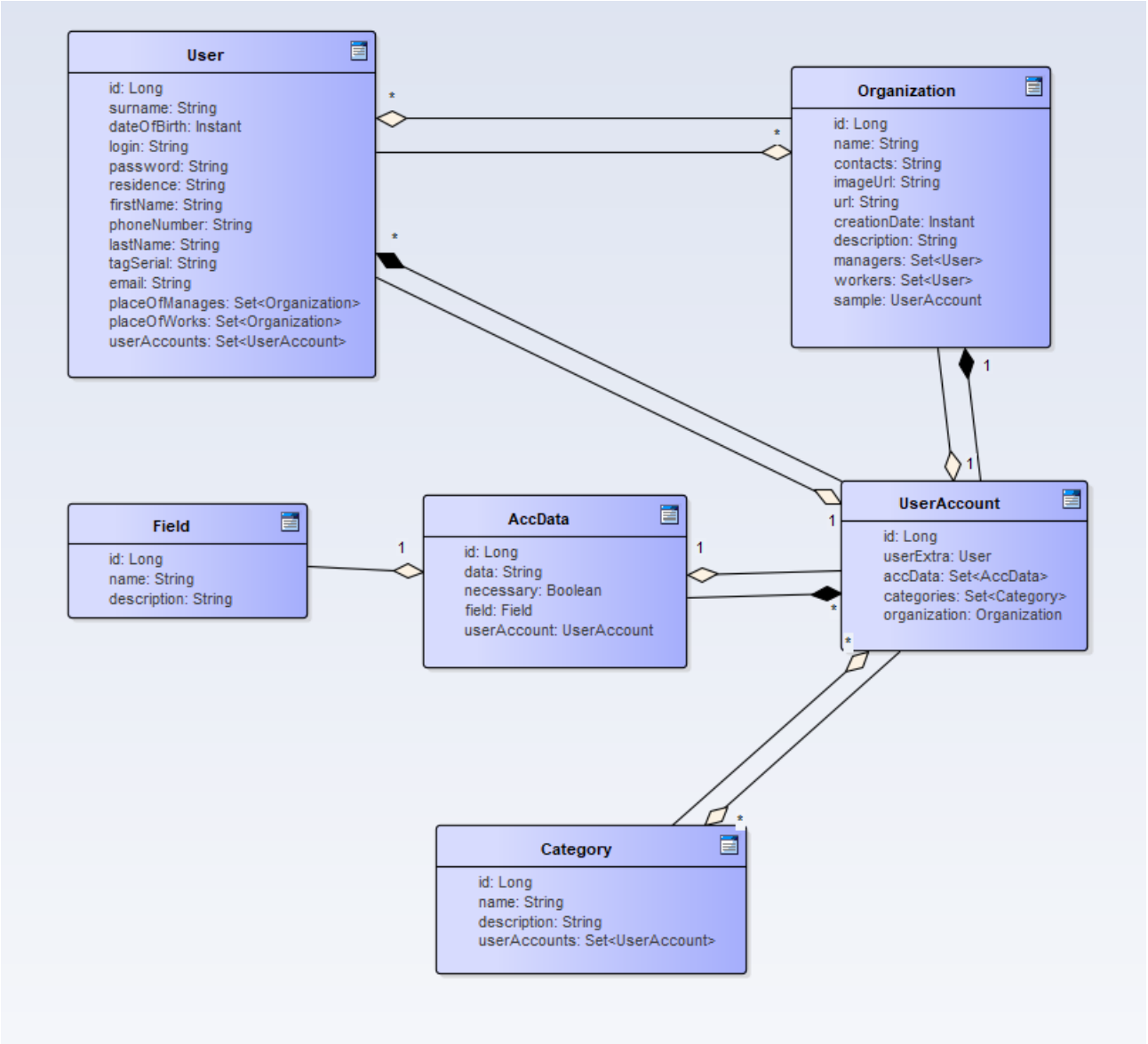

Рисунок 3.2 – Діаграма бізнес сутностей

В даній системі були визначені наступні основні сутності:

- користувач: Ця сутність містить інформацію про особисті дані користувача системи, а також перелік карток установ, клієнтом яких є користувач, перелік установ, працівником яких є користувач, і перелік установ, менеджером яких є користувач;
- установа: Установа зберігає інформацію про свою назву, опис, контакти, логотип, посилання на сайт, дату створення, перелік користувачів,

які працюють в установі, перелік менеджерів установи та приклад картки установи, яку мають заповнювати клієнти;

- картка: Ця сутність містить інформацію про користувача, який є власником картки, перелік даних картки, перелік категорій, до яких відноситься користувач, та організацію, якій належить ця картка;
- дані картки: Сутність містить інформацію про поля картки та інформацію, надану користувачем;
- поле картки: Ця сутність містить інформацію про свою назву та опис;
- категорія: Категорія містить інформацію про свою назву та опис.

На цьому етапі завершуються процедури бізнес-моделювання, і це дозволяє надати команді проектувальників інформацію в єдиному форматі, яка буде необхідна для створення системи.

Архітектура програмного забезпечення відіграє ключову роль у створенні ефективних розподілених мережевих застосувань, які забезпечують взаємодію та обмін даними. Результати аналізу та принципи роботи найбільш популярних архітектур для клієнт-серверних систем наведено у роботах [2-5].

У даній роботі розроблено модельно-орієнтовану архітектуру програмного забезпечення, яка дозволяє розробникам створювати NFC-застосунки з урахуванням вимог до незалежності від платформи, доступності та гнучкості (рисунок 3.3).

Головні модулі запропонованою архітектури включають NFCSurface (поверхні фізичних об'єктів), Model (поведінку застосунку), Router (асоціації між NFCSurface та HotSpot), і HotSpot (області на поверхні NFCSurface). Метамодель також підтримує визначення різних Model для розподілу поведінки системи. Кожна Model визначає набір Action, які можуть бути виконані в цій моделі. HotSpot пов'язані з Action за допомогою Route, який визначає зв'язок між фізичною поверхнею та обчислювальною дією. Вони групуються в Router, який визначає, як пов'язані HotSpot та Action.

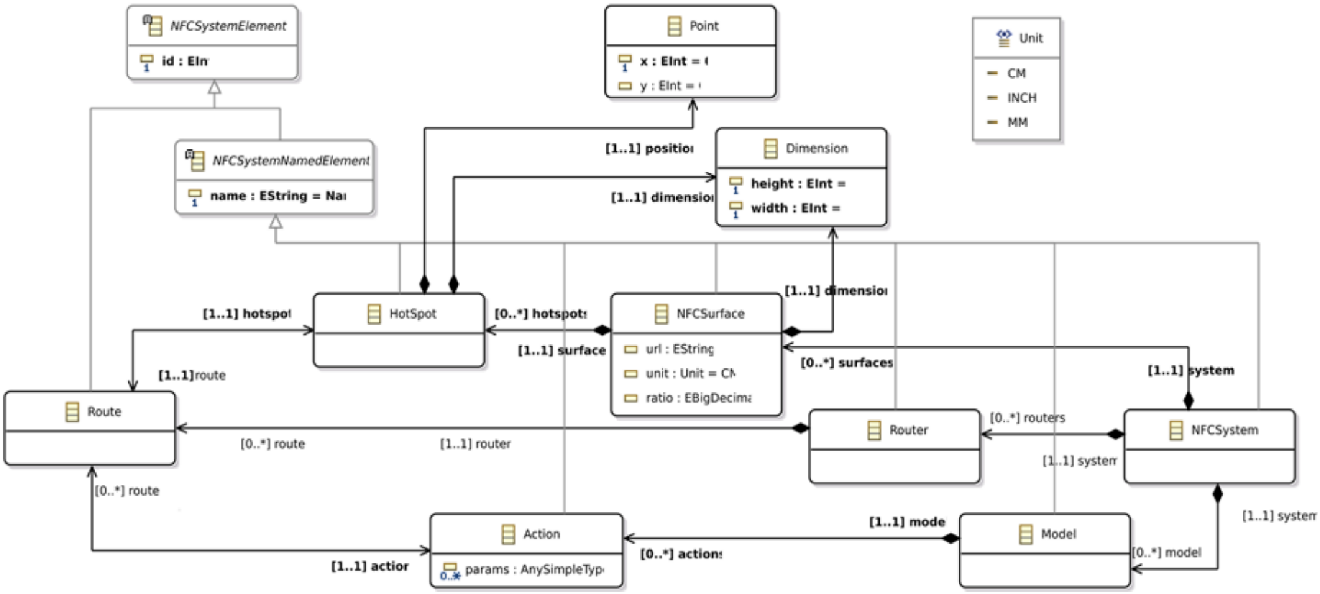

Рисунок 3.3 - Модельно-орієнтована архітектура програмного забезпечення

Загалом, запропонована архітектура допоможе розробникам швидше і ефективніше створювати програми, які взаємодіють з фізичними об'єктами за допомогою технології NFC.

3.2 Проектування бази даних програмної системи

Логічне проектування бази даних включає наступні етапи:

- перетворення локальної концептуальної моделі даних в локальну логічну модель. На цьому етапі виконується видалення зв'язків M:H, складних зв'язків, рекурсивних зв'язків, зв'язків з атрибутами, а також видалення множинних атрибутів;
- визначення набору відносин на основі структури локальної логічної моделі даних;
- перевірка моделі за допомогою правил нормалізації, що допомагає забезпечити ефективну та оптимізовану структуру бази даних;

- перевірка моделі щодо транзакцій користувачів, що гарантує правильність та цілісність даних під час роботи користувачів з системою;
- створення діаграми сутність-зв'язок (ER-діаграми), яка графічно відображає сутності та їх зв'язки в базі даних;
- визначення вимог для підтримки цілісності даних, такі як обов'язкові дані, обмеження для доменів атрибутів, цілісність сутностей (наприклад, первинний ключ не може бути NULL), а також бізнес-правила, які повинні дотримуватися в системі.

Ці етапи допомагають створити логічну модель бази даних, яка є незалежною від фізичних умов та може служити основою для подальшої фізичної реалізації бази даних.

Після створення ER-діаграми реалізуйте базу даних в системі управління базами даних (СУБД) відповідно до розробленої моделі. Додайте SQL-запити та програмний функціонал для роботи з базою даних та проведіть тестування для забезпечення її правильної роботи.

Для побудови ER-діаграми, відповідно до наведених правил, виконуються наступні дії (рисунок 3.4):

- зв'язок 1:1 і обов'язковий клас приналежності обох об'єктів: В цьому випадку потрібна лише одна таблиця. Первинним ключем цієї таблиці може бути первинний ключ будь-якого з двох об'єктів;
- зв'язок 1:1 і клас приналежності однієї сутності необов'язковий: Тут потрібна окрема таблиця для кожної сутності. Первинний ключ сутності стає первинним ключем відповідної таблиці. Первинний ключ сутності, для якої клас приналежності необов'язковий, додається як атрибут до таблиці сутностей з обов'язковим класом приналежності;
- зв'язок 1:1 і клас приналежності обох сутностей є необов'язковими: В даному випадку створюється три таблиці: по одній для кожної сутності та одна для зв'язку. Первинний ключ кожної сутності стає первинним ключем відповідної таблиці. Таблиця зв'язків містить ключі обох сутностей;

- зв'язок 1:M і обов'язковий клас приналежності M-сторони: У цьому випадку також потрібна окрема таблиця для кожної сутності. Первинний ключ сутності на стороні 1 стає атрибутом таблиці сутності на стороні M;

- зв'язок M:M: Для зв'язку типу M:M також потрібно створити три таблиці: по одній для кожної сутності та одну для зв'язку. Таблиця зв'язків містить ключі обох сутностей.

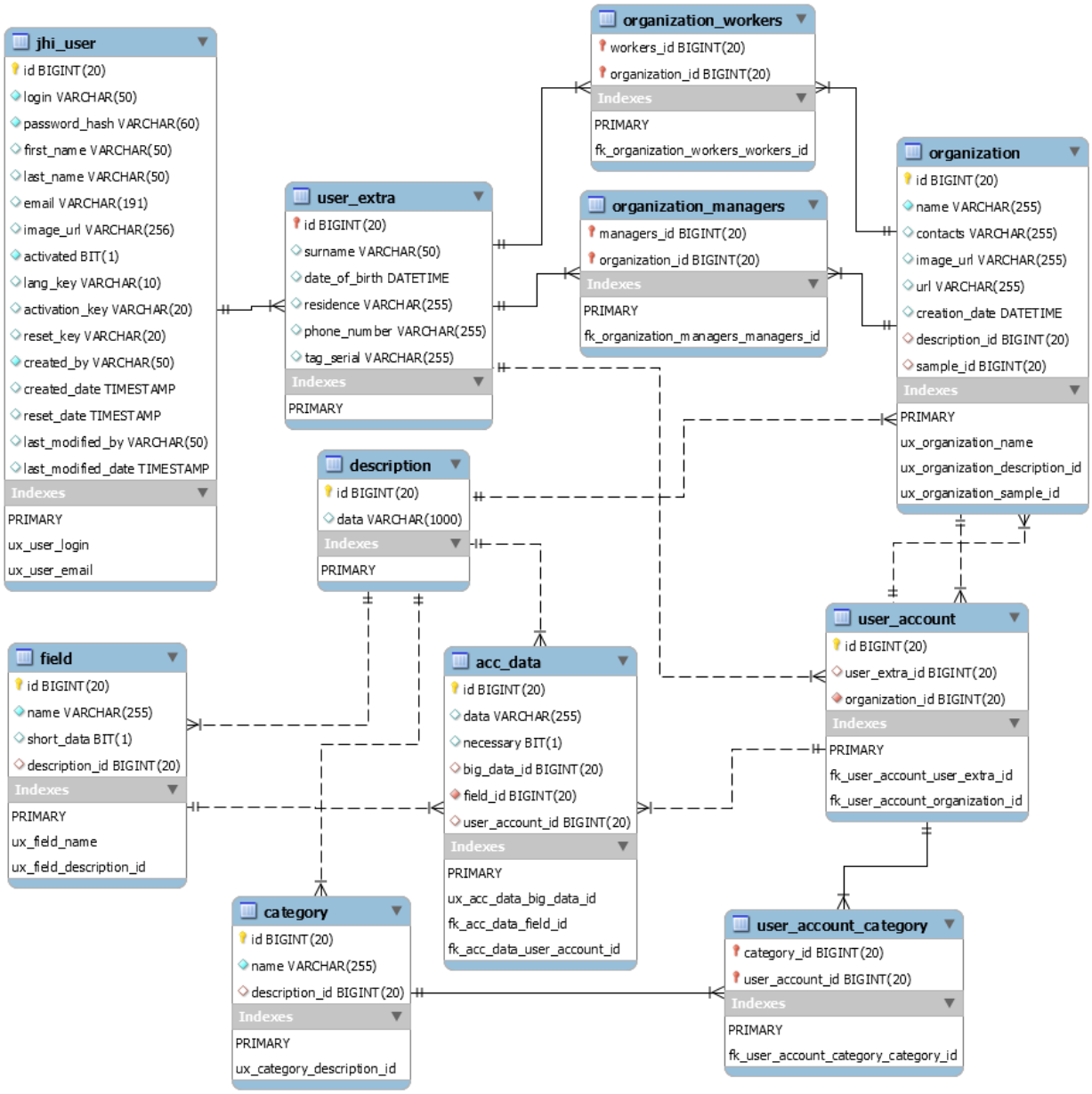

Рисунок 3.4 – ER-діаграма бази даних

Зв’язки між сутностями проектованої системи наведено в таблиці 3.11.

Таблиця 3.11 - Зв’язки між сутностями проектованої системи

Сутність	Чисельність	Напрямок	Чисельність	Сутність
Данні картки	M	→	1	Поле картки
Користувач	1	←→	M	Картка
Користувач	M	←→	M	Установа
Користувач	M	←→	M	Установа
Установа	1	→	1	Картка
Сутність	Чисельність	Напрямок	Чисельність	Сутність
Картка	M	→	1	Установа
Картка	1	←→	M	Данні картки
Картка	M	←→	M	Категорія

Перейдемо до створення таблиць бази даних на основі ER-діаграми. Далі наведено подробну інформацію про характеристики кожної таблиці, включаючи назви колонок, їхні типи даних, вказання на те, чи вони є первинними ключами, чи поле може залишатися порожнім, чи значення автоматично збільшується, і короткий опис призначення кожної колонки.

На рисунку 3.5 зображено інформацію про таблицю «User»

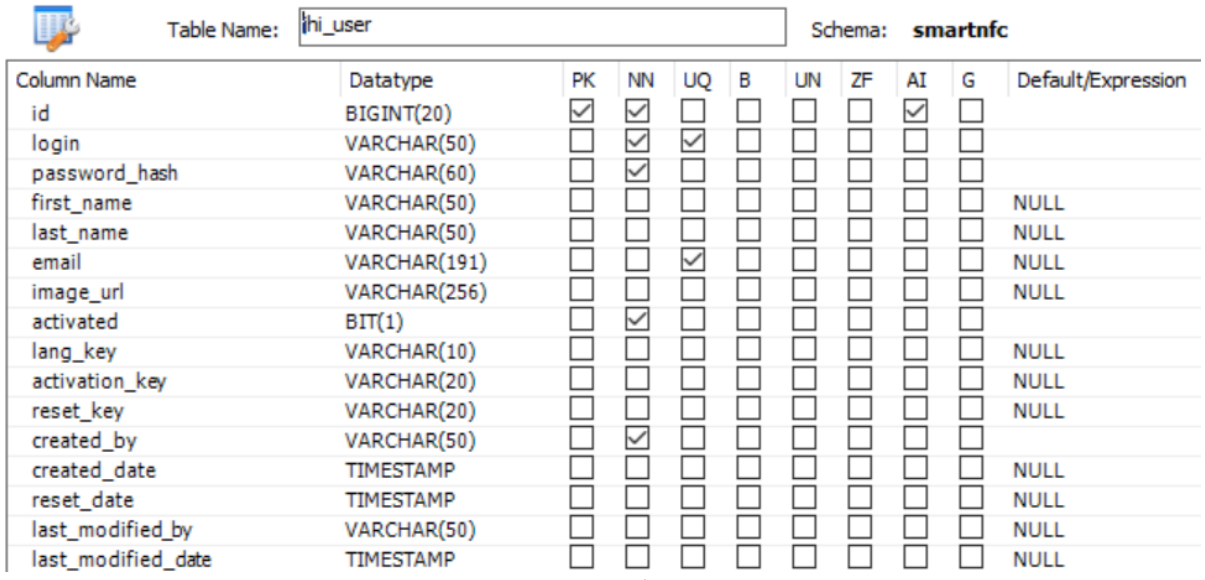
Table Name: hi_user Schema: smartnfc

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	BIGINT(20)	☑	☑	☐	☐	☐	☐	☑	☐	
login	VARCHAR(50)	☐	☑	☑	☐	☐	☐	☐	☐	
password_hash	VARCHAR(60)	☐	☑	☐	☐	☐	☐	☐	☐	
first_name	VARCHAR(50)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
last_name	VARCHAR(50)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
email	VARCHAR(191)	☐	☐	☑	☐	☐	☐	☐	☐	NULL
image_url	VARCHAR(256)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
activated	BIT(1)	☐	☑	☐	☐	☐	☐	☐	☐	
lang_key	VARCHAR(10)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
activation_key	VARCHAR(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
reset_key	VARCHAR(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
created_by	VARCHAR(50)	☐	☑	☐	☐	☐	☐	☐	☐	
created_date	TIMESTAMP	☐	☐	☐	☐	☐	☐	☐	☐	NULL
reset_date	TIMESTAMP	☐	☐	☐	☐	☐	☐	☐	☐	NULL
last_modified_by	VARCHAR(50)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
last_modified_date	TIMESTAMP	☐	☐	☐	☐	☐	☐	☐	☐	NULL

Рисунок 3.5 – Таблиця «User»

Призначення стовпців таблиці:

- Id – основний ключ таблиці;
- Login – поле, яке зберігає логін користувача;
- Password_hash – поле, яке зберігає пароль користувача;
- First_name – поле, яке зберігає ім’я користувача;
- Last_name – поле, яке зберігає прізвище користувача;
- Email – поле, яке зберігає електронну пошту користувача;
- Image_url – поле, яке зберігає посилання на фото користувача;
- Activated – поле, яке зберігає інформацію про те чи активований акаунт користувача;
- Lang_key – поле, яке зберігає інформацію про обрану мову користувача;
- Activation_key – поле, яке зберігає ключ для активації акаунта користувача;
- Reset_key – поле, яке зберігає ключ для зміни пароля користувача;
- Created_by – поле, яке зберігає інформацію про користувача, який створив цей акаунт;
- Created_date – поле, яке зберігає дату створення акаунту;
- Reset_date – поле, яке зберігає дату останньої зміни пароля;
- Last_modified_by – поле, яке зберігає інформацію про користувача, який останній вносив зміни до акаунту;
- Last_modified_date – поле, яке зберігає дату останніх змін акаунту.

На рисунку 3.6 зображено інформацію про таблицю «User_extra». Призначення стовпців таблиці:

- Id – основний ключ таблиці;
- Surname – поле, яке зберігає по-батькові користувача;
- Date_of_birth – поле, яке зберігає дату народження користувача;
- Residence – поле, яке зберігає місце проживання користувача;

- Phone_number – поле, яке зберігає номер телефону користувача;
- Tag_serial – поле, яке зберігає серійний номер NFC тегу, виданого користувачеві;

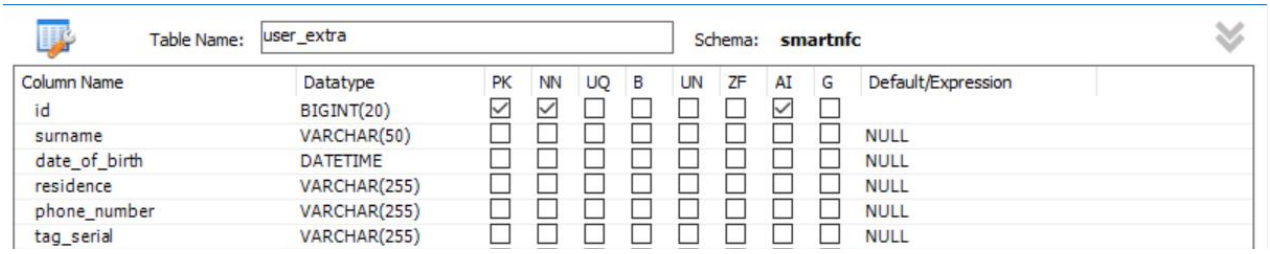
Table Name: user_extra Schema: smartnfc

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	BIGINT(20)	☑	☑	☐	☐	☐	☐	☑	☐	
surname	VARCHAR(50)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
date_of_birth	DATETIME	☐	☐	☐	☐	☐	☐	☐	☐	NULL
residence	VARCHAR(255)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
phone_number	VARCHAR(255)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
tag_serial	VARCHAR(255)	☐	☐	☐	☐	☐	☐	☐	☐	NULL

Рисунок 3.6 – Таблиця «User_extra»

На рисунку 3.7 зображено інформацію про таблицю «Acc_data»

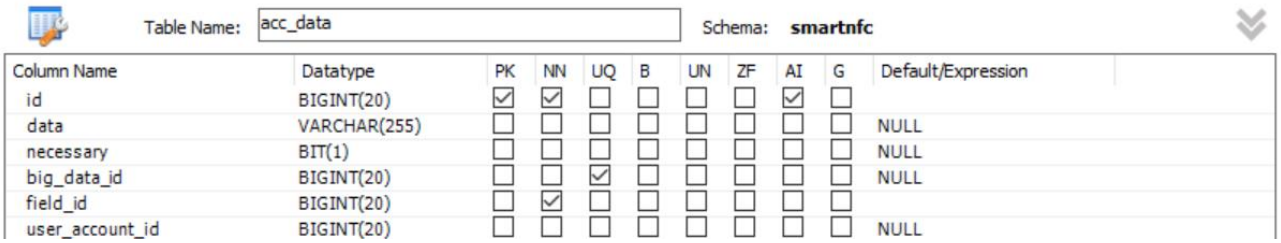
Table Name: acc_data Schema: smartnfc

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	BIGINT(20)	☑	☑	☐	☐	☐	☐	☑	☐	
data	VARCHAR(255)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
necessary	BIT(1)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
big_data_id	BIGINT(20)	☐	☐	☑	☐	☐	☐	☐	☐	NULL
field_id	BIGINT(20)	☐	☑	☐	☐	☐	☐	☐	☐	
user_account_id	BIGINT(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL

Рисунок 3.7 – Таблиця «Acc_data»

Призначення стовпців таблиці:

- Id – основний ключ таблиці;
- Data – поле, яке зберігає дані поля картки, заповнені користувачем;
- Necessary – поле, яке зберігає інформацію про те чи є це поле картки обов’язковим для заповнення;
- Big_data_id – поле, яке зберігає первинний ключ рядка таблиці описів, який зберігає об’ємну текстову інформацію за необхідності;
- Field_id – поле, яке зберігає первинний ключ поля картки, яке заповнює користувач;
- User_account_id – поле, яке зберігає первинний ключ картки.

На рисунку 3.8 зображено інформацію про таблицю «Category»

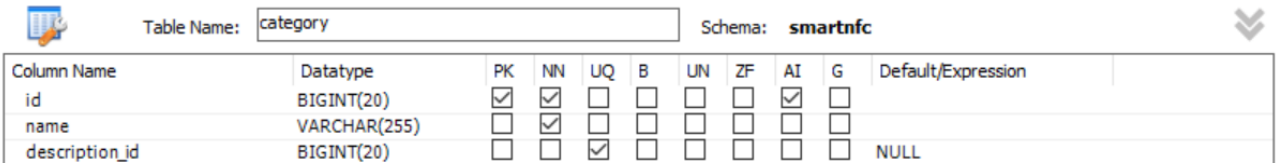
Table Name: category Schema: smartnfc

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	BIGINT(20)	☑	☑	☐	☐	☐	☐	☑	☐	
name	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	☐	
description_id	BIGINT(20)	☐	☐	☑	☐	☐	☐	☐	☐	NULL

Рисунок 3.8 – Таблиця «Category»

Призначення стовпців таблиці:

- Id – основний ключ таблиці;
- Name – поле, яке зберігає назву категорії користувачів;
- Description_id – поле, яке зберігає первинний ключ опису.

На рисунку 3.9 зображено інформацію про таблицю «Description»

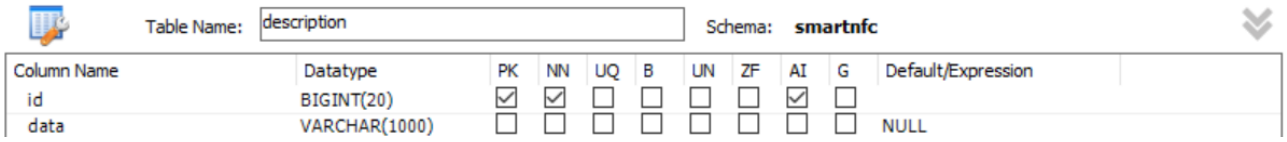
Table Name: description Schema: smartnfc

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	BIGINT(20)	☑	☑	☐	☐	☐	☐	☑	☐	
data	VARCHAR(1000)	☐	☐	☐	☐	☐	☐	☐	☐	NULL

Рисунок 3.9 – Таблиця «Description»

Призначення стовпців таблиці:

- Id – основний ключ таблиці;
- Data – поле, яке зберігає текст опису.

На рисунку 3.10 зображено інформацію про таблицю «Field»

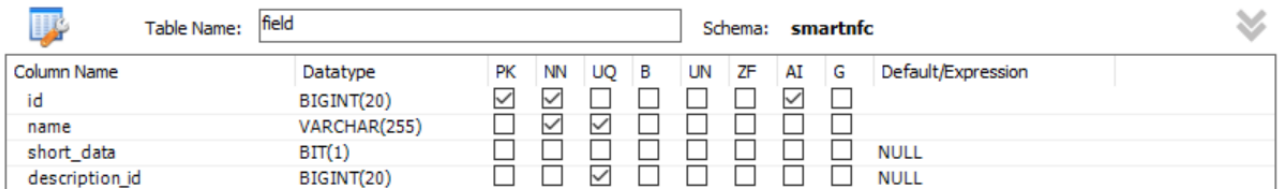
Table Name: field Schema: smartnfc

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	BIGINT(20)	☑	☑	☐	☐	☐	☐	☑	☐	
name	VARCHAR(255)	☐	☑	☑	☐	☐	☐	☐	☐	
short_data	BIT(1)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
description_id	BIGINT(20)	☐	☐	☑	☐	☐	☐	☐	☐	NULL

Рисунок 3.10 – Таблиця «Field»

Призначення стовпців таблиці:

- Id – основний ключ таблиці;

- Name – поле, яке зберігає назву поля;
- Short_data – поле, яке зберігає інформацію про необхідність зберігання об’ємної текстової інформації для цього поля;
- Description_id – поле, яке зберігає первинний ключ опису.

На рисунку 3.11 зображено інформацію про таблицю «Organization».

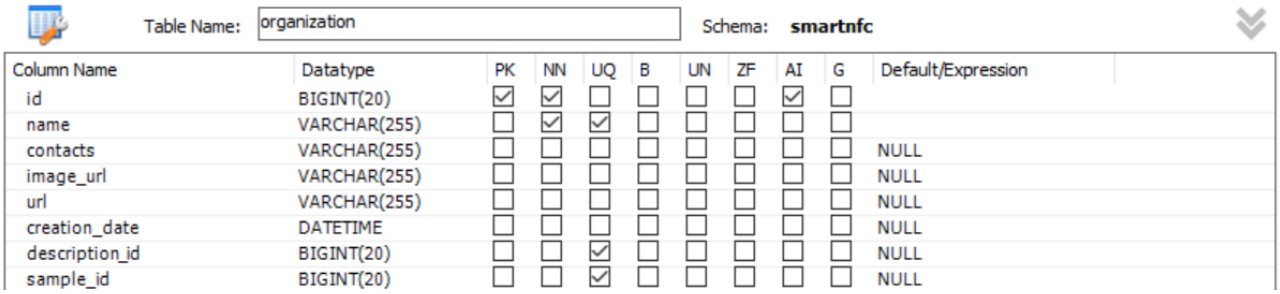
Table Name: organization Schema: smartnfc

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	BIGINT(20)	☑	☑	☐	☐	☐	☐	☑	☐	
name	VARCHAR(255)	☐	☑	☑	☐	☐	☐	☐	☐	
contacts	VARCHAR(255)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
image_url	VARCHAR(255)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
url	VARCHAR(255)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
creation_date	DATETIME	☐	☐	☐	☐	☐	☐	☐	☐	NULL
description_id	BIGINT(20)	☐	☐	☑	☐	☐	☐	☐	☐	NULL
sample_id	BIGINT(20)	☐	☐	☑	☐	☐	☐	☐	☐	NULL

Рисунок 3.11 – Таблиця «Organization»

Призначення стовпців таблиці:

- Id – основний ключ таблиці;
- Name – поле, яке зберігає назву установи;
- Contacts – поле, яке зберігає контакти установи;
- Image_url – поле, яке зберігає посилання на файл логотипу установи;
- Url – поле, яке зберігає посилання на сайт установи;
- Creation_date – поле, яке зберігає дату реєстрації установи в системі;
- Description_id – поле, яке зберігає первинний ключ опису;
- Sample_id – поле, яке зберігає первинний ключ карки користувача, яка є прикладом картки даної установи.

На рисунку 3.12 зображено інформацію про таблицю «Organization_managers»

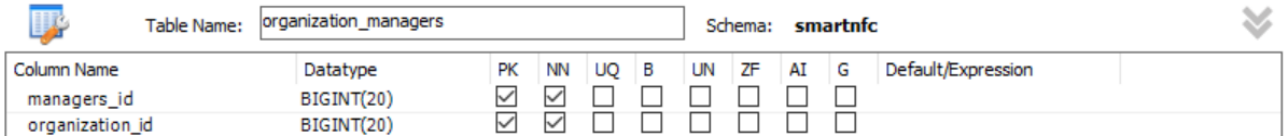

Рисунок 3.12 – Таблиця «Organization_managers»

Призначення стовпців таблиці:

- Managers_id – поле, яке зберігає первинний ключ користувача, який працює менеджером в установі;
- Organization_id – поле, яке зберігає первинний ключ установи, в якій працює користувач.

На рисунку 3.13 зображено інформацію про таблицю «Organization_workers»

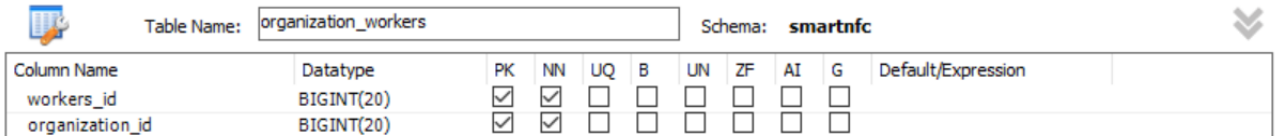

Рисунок 3.13 – Таблиця «Organization_workers»

Призначення стовпців таблиці:

- Workers_id – поле, яке зберігає первинний ключ користувача, який працює в установі;
- Organization_id – поле, яке зберігає первинний ключ установи, в якій працює користувач.

На рисунку 3.14 зображено інформацію про таблицю «User_account»

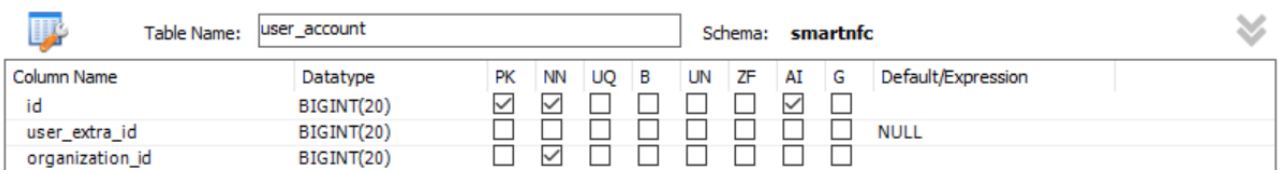

Рисунок 3.14 – Таблиця «User_account»

Призначення стовпців таблиці:

- Id – основний ключ таблиці;
- User_extra_id – поле, яке зберігає первинний ключ користувача, який створив картку;
- Organization_id – поле, яке зберігає первинний ключ установи, якій належить картки.

На рисунку 3.15 зображено інформацію про таблицю «User_account_category»

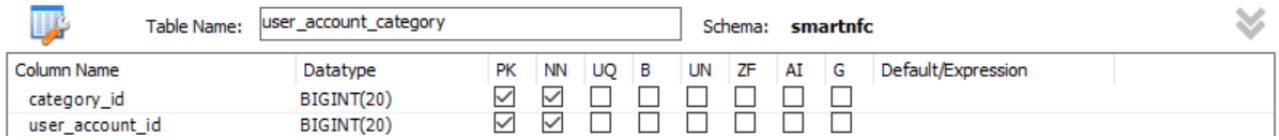

Рисунок 3.15 – Таблиця «User_account_category»

Призначення рядків таблиці:

- Category_id – поле, яке зберігає первинний ключ категорії, до якої належить власник картки;
- User_account_id – поле, яке зберігає первинний ключ картки.

Для реалізації реляційно-об'єктного відображення було використано JPA та бібліотеку Hibernate. Це надає можливість легко встановити зв’язок з будь-якою базою даних та встановити відображення між об’єктно-орієнтованою моделлю та традиційною реляційною моделлю баз даних.

На рисунку 3.16 представлена загальна архітектура системи. У цій архітектурі рівень відображення відповідає за графічний інтерфейс користувача та обробку запитів. Рівень відображення передає всі запити рівню прикладної логіки. Даний рівень є центральною частиною застосунку та містить основну логіку. Іншими словами, прикладна логіка – рівень, де розташовані компоненти, що виконують різні прикладні операції, такі як підрахунки, пошук і впорядкування даних, обслуговування профілів

користувачів. Рівень прикладної логіки зберігає та отримує дані з бази даних за допомогою рівня доступу до даних. Рівень доступу до даних забезпечує високорівневі, об’єктно-орієнтовані абстракції поверх рівня бази даних. Рівень бази даних зазвичай включає систему управління базами даних.

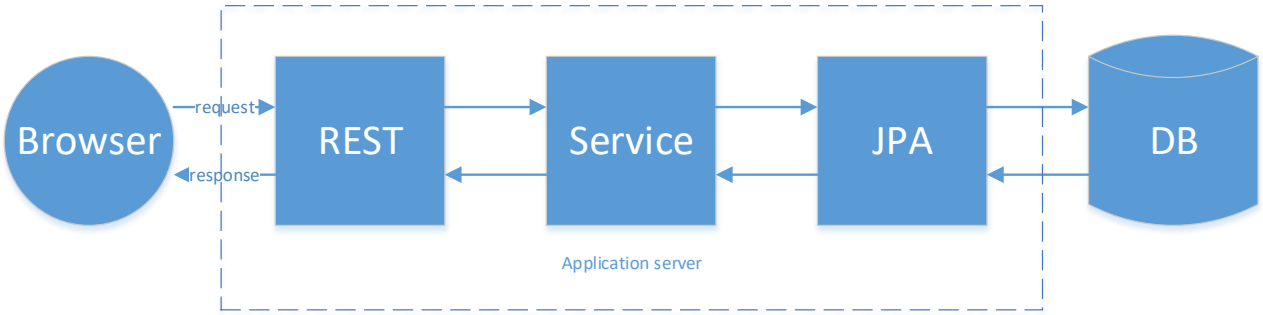

Рисунок 3.16 – Загальна архітектура проекту

У даному проекті рівень відображення реалізований у вигляді модуля обробки запитів користувача з використанням засобів Spring MVC REST. Рівень прикладної логіки представлений модулем бізнес-логіки та сервісів. Рівень доступу до даних представлений модулем доступу до даних з використанням засобів JPA.

Оскільки для реалізації рівня доступу до даних було обрано технологію реляційно-об'єктного відображення засобами JPA, розглянемо ключові особливості роботи з даною технологією.

Робота з JPA в середовищі JavaEE відбувається зазвичай в таких режимах:

- контексти персистентності управляються контейнерами, а не самими додатками;
- використовуються JTA-транзакції, а не локальні транзакції;
- Фабрики з'єднань з базою даних (інтерфейс DataSource) задаються через JNDI-імена;
- контексти персистентності можуть бути як EXTENDED-, так і TRANSACTION-контекстами.

Цей режим відрізняється від роботи в середовищі JavaSE. У цьому режимі додаток не створює явно фабрику менеджерів персистентності та самі менеджери. Контейнер створює їх при створенні екземплярів session-компонентів, які управляють ходом виконання програми (зазвичай з використанням dependency injection). Кожен такий менеджер персистентності працює не з власним контекстом, а з "загальним" контекстом, який автоматично передається між компонентами, що беруть участь в транзакції, разом з контекстами транзакції і безпеки.

Одним із важливих виборів, який розробник повинен зробити, є вибір між TRANSACTION-контекстами, які створюються заново для кожної глобальної транзакції і закриваються після її завершення, і EXTENDED-контекстами, які можуть брати участь в декількох послідовних транзакціях, поки їх розробник явно не закриє. Проте навіть у цьому випадку вибір обмежений: використання session stateless-компонентів природно передбачає використання TRANSACTION-контекстів, а session stateful-компонентів - EXTENDED-контекстів. Інший підхід пов'язаний зі створенням штучних і складних програмних конструкцій. Архітектура додатка суттєво впливає на рішення на низькому рівні.

Ще одним важливим аспектом є вплив прикладних і системних виключень EJB на стан об'єктів в контекстах персистентності та саміх контекстів. Зазвичай традиційним є відкат транзакції EJB-контейнером, і розробнику завжди потрібно мати на увазі можливість виникнення таких виключень.

Також важливо розглядати питання багатопоточності. У компонентній моделі EJB зазвичай не рекомендується створювати власні потоки або закривати існуючі потоки - це обов'язок контейнера. Архітектура JPA передбачає, що робота з кожним контекстом персистентності виконується в однопоточному режимі. Такий підхід гармонійно поєднується з моделлю багатопоточності EJB.

Крім того, слід зазначити, що додаток може мати кілька модулів персистентності для однієї JVM, якщо потрібно використовувати різні сховища даних. Проте це не завжди є необхідним і може ускладнити програму. Контекст персистентності не передається при віддалених викликах, тому ми обмежуємо себе окремим JavaEE-додатком. Використання декількох сховищ даних, як правило, ускладнює програму, тому потрібно внимати цьому аспекту.

У додатку описані інтерфейси JpaRepository, PagingAndSortingRepository, QueryByExampleExecutor та CrudRepository, які надають CRUD-операції, що уніфіковані для всіх сутностей системи. Це дозволяє спростити роботу з сутностями шляхом створення окремого інтерфейсу для кожної сутності, який наслідує вказані інтерфейси.

У модулі "Модуль обробки запитів користувача" також описані класи сервісів, які забезпечують логіку користувацького інтерфейсу. Для зчитування мітки був розроблений програмний код мікропроцесора, який додається у додатку В. Для обробки запитів з клієнтської частини була використана технологія Spring MVC REST.

3.3 Тестування програмної системи

Тестування програмної системи - це процес перевірки та оцінки властивостей системи з метою визначення відповідності її вимогам. Цей процес допомагає виявити помилки, дефекти та недоліки у програмному забезпеченні перед його випуском або впровадженням.

Після реєстрації або аутентифікації, користувачу надається можливість виконати відповідні дії на веб-сайті.

Користувач переходить на example.com та ініціює вхід. Після цього отримує повідомлення від веб-переглядача з запитом: "Будь ласка, завершіть цю дію на своєму телефоні." На телефоні користувач виконує наступні дії:

1) бачить дискретні підказки або сповіщення: "Увійти до example.com.";

2) вибирає цю підказку / сповіщення;

3) відображається список ідентифікаторів example.com, наприклад, "Увійти як Аліса / Увійти як Боб";

4) вибирає ідентифікатор, запитується про жест руху ідентифікації і надає його;

5) веб-сторінка на комп'ютері показує, що вибраний користувач успішно увійшов у систему, а також переходить до сторінки, на якій він підписаний.

Наступний сценарій використання демонструє, як довірена сторона може використовувати комбінацію роумінг-автентифікатора (USB-ключа) і аутентифікатора платформи (вбудованого датчика відбитків пальців):

1) користувач відвідує веб-сайт example.com та увійшов до свого облікового запису;

2) переходить до налаштувань безпеки облікового запису та обирає опцію "Реєстрація USB-ключа безпеки";

3) веб-сайт запитує користувача підключити USB-ключ безпеки, і користувач це робить;

4) після підключення USB-ключа, він вказує користувачеві, що потрібно натиснути кнопку на ньому. Користувач натискає на кнопку;

5) після успішного завершення реєстрації, веб-сайт відображає повідомлення: "Реєстрацію завершено."

Пізніше, на комп'ютері, який обладнаний аутентифікатором платформи, користувач виконує наступні кроки для входу на веб-сайт example.com:

1) відвідує веб-сайт example.com у своєму веб-переглядачі та ініціює процес входу;

2) веб-сайт пропонує використати NFC для аутентифікації;

3) користувач підключається через NFC;

4) після успішного підключення, веб-сайт відображає повідомлення, що користувач успішно увійшов до системи, та переходить на сторінку, яку користувач бажає підписати;

5) веб-сайт запитує користувача, чи бажає він зареєструвати цей комп'ютер для використання з example.com. Користувач погоджується;

6) ноутбук пропонує користувачеві попередньо налаштований ідентифікатор, і користувач надає це підтвердження;

7) після успішного завершення процесу реєстрації, веб-сайт відображає повідомлення "Реєстрація завершена";

8) користувач успішно підписується.

У засобі безпеки і програмному забезпеченні важливо мати прості, зрозумілі і легко інтегровані інтерфейси, які гармонійно взаємодіють з інтерфейсом користувача та робочим процесом. На рисунку 3.17 показані основні екранні форми розробленої комп'ютерно-інтегрованої системи.

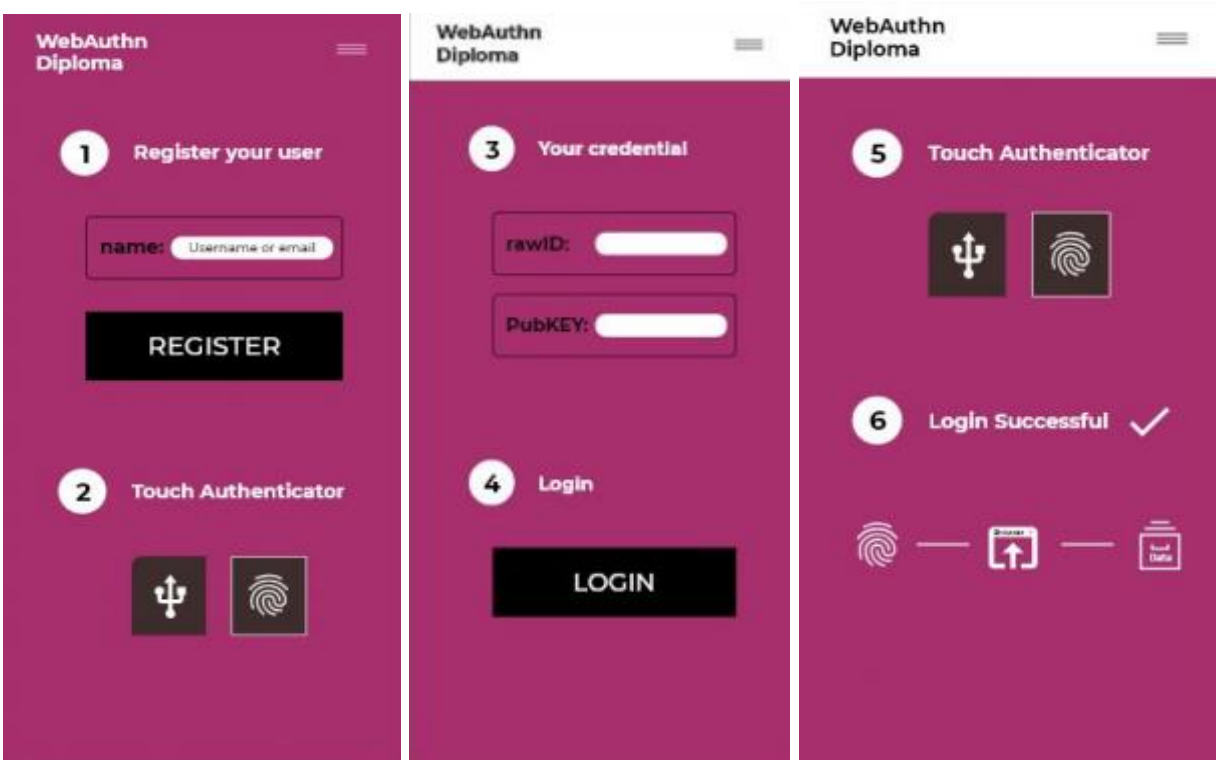

Рисунок 3.17 – Екранна форми комп’ютерно інтегрованої системи ідентифікації користувачів на основі технології NFC

Пізніше, знову на комп'ютері:

1) користувач відвідує веб-сайт example.com у своєму веб-переглядачі та ініціює вхід;

2) веб-сайт відображає повідомлення "Будь ласка, виконайте вказівки вашого комп'ютера для завершення входу";

3) ноутбук пропонує користувачеві ідентифікатор, і користувач надає це підтвердження;

4) після успішного підтвердження, веб-сайт відображає повідомлення, що користувач успішно увійшов до системи та переходить на сторінку, яку користувач бажає підписати.

ВИСНОВКИ

У кваліфікаційній роботі було проведено аналіз поняття ідентифікації особи та показано потребу у використанні різних засобів ідентифікації та ідентифікаторів для підтвердження особи або надання конкретної інформації. Це стало актуальним в контексті надмірної кількості пластикових карток та спричиненого дискомфорту цим обмеженням. Для розв'язання цієї проблеми була розроблена концепція уніфікованої системи ідентифікації, яка дозволить застосовувати єдиний стандартизований ідентифікатор в різних сферах життя.

Під час аналізу предметної області було розглянуто різні існуючі системи ідентифікації, зокрема, ті, які базуються на технології NFC, та вивчив їхні принципи роботи. Це дозволило описати спосіб реалізації системи ідентифікації користувача за допомогою NFC-міток та розглянути деякі варіанти її застосування.

Далі у роботі проводилася проектування програмної системи. Була виконана об'єктна декомпозиція на основі способу реалізації системи, описано основних акторів системи, виділені прецеденти системи та побудована загальна схема прецедентів, яка служить основою для розробки будь-якого застосунку. Далі були описані основні сценарії прецедентів, що значно спрощує процес розробки інтерфейсу системи. Для наглядності сценаріїв була використана таблична форма, яка ілюструє послідовність роботи користувача з системою. Також була розроблена концептуальна модель даних на основі об'єктної декомпозиції системи, визначено основні сутності, які є основою подальшого створення бази даних. Крім того, був описаний приклад проектування антени для NFC-мітки у формі кільця.

На основі концептуальної моделі даних була розроблена база даних, де були встановлені зв'язки між сутностями системи, побудована ER-діаграма та створений SQL-скрипт для генерації бази даних. Також була описана призначення стовпців таблиць. Для спрощення роботи з даними була

використана технологія об'єктно-реляційного відображення за допомогою бібліотеки Hibernate і реалізовані відповідні класи сутностей. Для структури проекту і зменшення зв'язності була використана традиційна чотирьохрівнева архітектура проекту, включаючи модуль доступу до даних, який реалізує основні операції з даними, модуль бізнес-логіки, який відповідає за прикладні операції та підрахунки, і модуль обробки запитів користувача, який забезпечує обробку запитів зі сторони клієнта.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Sophisticated RFID Application on MultiFactor Authentication / J. C. Liou a. o. // Information Technology: New Generations (ITNG), 2011 Eighth International Conference on. IEEE, 2011. C. 180–185.

2. Haselsteiner E., Breitfuss K. Security in near field communication (NFC) // Workshop on RFID Security RFIDSec. 2006. 9. http://nfcukraine.com/ 10. http://www.i-free.com/J.Padgette, K. Scarfone, L.Chen. Guide To Bluetooth Security. 2021. URL: http://csrc.nist.gov/publications/drafts/800-121r1/ 121_Rev1.pdf/.

3. R.Kilani, K.Jensen. Mobile Authentication with NFC enabled Smartphones. 2021. URL: http://ojs.statsbiblioteket.dk/index.php/

4. Information technology - Telecommunications and information exchange; between systems — Near Field Communication — Interface and Protocol (NFCIP1). 2014. URL: http://www.iso.org/

5. Near Field Communication - Interface and Protocol (NFCIP-1). 2021. URL: http://www.ecma-international.org/

6. E.Haselsteiner, K.Breitfuß.Security in Near Field Communication (NFC). URL: http://events.iaik.tugraz.at/

7. Relay Attacks in EMV Contactless Cards with Android OTS Devices. URL: https://conference.hitb.org/hitbsecconf2015ams/wpcontent/uploads/2014/12/D1T1-R.-Rodriguez-P.-Vila-Relay-Attacks-in-EMVContactless-Cards-with-Android-OTS-Devices.pdf.

8. Near Field Communication (NFC) Technology, Vulnerabilities and Principal Attack Schema. 2013. URL: http://resources.infosecinstitute.com/nearfield- communication-nfc-technology-vulnerabilities-and-principal-attack-schema/

9. Gerhard P. Hancke, Markus G. Kuhn. An RFID Distance Bounding Protocol. 2015. URL: https://www.cl.cam.ac.uk/~mgk25/sc2005-distance.pdf /

10. Android Security Tips. URL: https://developer.android.com/

11. Вихорев С. В., Кобцев Р. Ю. Як дізнатися - звідки напасти або звідки виходить загроза безпеці інформації // Захист інформації. Конфідент, № 2, 2002.

12. Wireless LANs - - Security measures. I. Mouchtaris, Petros. II. Title TK5105. 59. A54 2007 005.8- -dc22.

13. Максим М. Безпека бездротових мереж / Меріт Максим, Девід Поліно; Пер. з англ. Семенова О.В. - М .: Компанія АйТі; ДМК Пресс, 2004.- 288с.

14. Вовчак Б.А. Аналіз атак при передачі даних з використанням технології NFC. Збірник матеріалів школи-семінару молодих вчених і студентів «Комп'ютерні інформаційні технології», Тернопіль. 2023. С. 20-21.

15. Вовчак Б.А. Архітектура програмного забезпечення для розробки застосунків на основі технології NFC // Збірник матеріалів міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ», Вінниця, 2023. С. 54-56/

16. Goodman M. International Dimensions of Cybercrime // S. Ghosh and E. Turrini (eds.), Cybercrimes: A Multidisciplinary Analysis. Berlin, Heidelberg, 2010.

17. Управління інформаційною безпекою сучасних ІКСМ на базі стандартів ISO. [Електрон. ресурс].- http://www.rusnauka.com/22_PNR_2010/Informatica/70334.doc.htm

18. Vedat Coskun. Overview of Near Field Communication [Текст] / Kerem Ok, Busra Ozdenizci // PROFESSIONAL NFC Application Development for Android. – 2013. – С. 5 – 22. 31.

19. Lakatos. #WIBattack: Vulnerability in WIB sim-browser can let attackers globally take control of hundreds of millions of the victim mobile phones worldwide to make a phone call, send SMS to any phone numbers, send victim’s location, launch WAP browser, etc. [Елетронний ресурс] // Ginno Security Laboratory. – 2019 – Режим доступу до ресурсу:

https://ginnoslab.org/2019/09/21/wibattack-vulnerabilityin-wib-sim-browser-can-let-attackers

20. Zak Doffman. New SIM Card Attacks: Both Android And iOS Impacted— Are You Vulnerable? [Елетронний ресурс] // Z. Doffman. – 2019 – Режим доступу до ресурсу: https://www.forbes.com/sites/zakdoffman/2019/09/28/ how-many-millions-of-phones-risk-sim-based-attacks-new-report

21. Simon Blythe. Layout Reconstruction of Complex Silicon Chips [Текст] / Beatrice Fraboni, Haroon Ahmed // IEEE Journal of Solid-State Circuits. – 2013. – С. 138 – 145.

22. Ross Anderson. Tamper Resistance — a Cautionary Note [Текст] / Markus Kuhn // Sixth USENIX Security Symposium Proceedings. – 2011. – С. 65 – 73.

23. WolfgangRankl. Overview about attacks on smart cards [Текст] // Information Security Technical Report Volume 8, Issue 1. – 2003. – С. 3 – 18.

24. Michael Roland. Practical Attack Scenarios on Secure Element-enabled Mobile Devices [Текст] / Josef Langer, Josef Scharinger // Information Security Technical Report. – 2012. – С. 4 – 6.

25. Salvador Mendoza. Intro to NFC Payment Relay Attacks. [Елетронний ресурс] // salmg. – 2018 – Режим доступу до ресурсу: https://salmg.net/ 2018/12/01/intro-to-nfc-payment-relay-attacks

26. Muhammad, M.H.; Sagheer, A.; Muhammad, A.K.; Ehab, M.M. A Novel and Efficient Multiple RGB Images Cipher Based on Chaotic System and Circular Shift Operations. IEEE Access 2020, 8, 146408–146427.

27. Banik, S.; Pandey, S.K.; Peyrin, T.; Sasaki, Y.; Sim, S.M.; Todo, Y. GIFT: A small present-Towards reaching the limit of lightweight encryption. In Proceedings of the Cryptographic Hardware and Embedded Systems (CHES), Taipei, Taiwan, 25–28 September 2017

28. Shen, H.; Shan, X.; Xu, M.; Tian, Z. A New Chaotic Image Encryption Algorithm Based on Transversals in a Latin Square. Entropy 2022, 24, 1574.

29. Xu, M.; Tian, Z. A novel image encryption algorithm based on self-orthogonal Latin squares. Optik 2018, 171, 891–903.

30. Zhang, X.; Zhang, L. Multiple-image encryption algorithm based on chaos and gene fusion. Multimed. Tools Appl. 2022, 81, 20021–20042.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
#define SDA_PIN 2
#define SCL_PIN 3
#define SS_PIN 10
#define MOSI_PIN 5
#define MISO_PIN 6
#define RED_LED_PIN 7
#define GREEN_LED_PIN 8
#define BLUE_LED_PIN 9

Adafruit_PN532 nfc(SS_PIN);

void setup(void) {
  while (!Serial);
  Serial.begin(115200);
  Serial.println("Hello!");
  nfc.begin();
  uint32_t versiondata = nfc.getFirmwareVersion();
  if (!versiondata) {
    Serial.print("Didn't find PN53x board");
    while (1); // halt
  }
  Serial.print("Found chip PN5"); Serial.println((versiondata >> 24) & 0xFF, HEX);
  Serial.print("Firmware ver. "); Serial.print((versiondata >> 16) & 0xFF, DEC);
  Serial.print('.'); Serial.println((versiondata >> 8) & 0xFF, DEC);
  nfc.setPassiveActivationRetries(0xFF);
  nfc.SAMConfig();
  pinMode(RED_LED_PIN, OUTPUT);
  pinMode(GREEN_LED_PIN, OUTPUT);
  pinMode(BLUE_LED_PIN, OUTPUT);
  Serial.println("Waiting for an ISO14443A card");
}

void loop(void) {
  unsigned long currentMillis = millis();
  digitalWrite(RED_LED_PIN, LOW);
  digitalWrite(BLUE_LED_PIN, LOW);
  digitalWrite(GREEN_LED_PIN, HIGH);
  boolean success;
  uint8_t uid[] = {0, 0, 0, 0, 0, 0, 0};
  uint8_t uidLength;
  success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);
  if (success) {
    digitalWrite(RED_LED_PIN, HIGH);
    digitalWrite(BLUE_LED_PIN, HIGH);
    digitalWrite(GREEN_LED_PIN, LOW);
```

```
    Serial.println("Found a card!");
    Serial.print("UID Length: "); Serial.print(uidLength, DEC); Serial.println(" bytes");
    Serial.print("UID Value: ");
    for (uint8_t i = 0; i < uidLength; i++) {
      Serial.print(" 0x"); Serial.print(uid[i], HEX);
    }
    Serial.println("");
    delay(1000);
  } else {
    Serial.println("Timed out waiting for a card");
  }
}  }
  // Got ok data, print it out!
  Serial.print("Found chip PN5"); Serial.println((versiondata>>24) & 0xFF, HEX);
  Serial.print("Firmware ver. "); Serial.print((versiondata>>16) & 0xFF, DEC);
  Serial.print('.'); Serial.println((versiondata>>8) & 0xFF, DEC);
  // Set the max number of retry attempts to read from a card
  // This prevents us from waiting forever for a card, which is
  // the default behaviour of the PN532.
  nfc.setPassiveActivationRetries(0xFF);
  // configure board to read RFID tags
  nfc.SAMConfig();
  //configure as an output the pin connected to the LED
  pinMode(LED_PIN_RED, OUTPUT);
  pinMode(LED_PIN_GREEN, OUTPUT);
  pinMode(LED_PIN_BLUE, OUTPUT);
  Serial.println("Waiting for an ISO14443A card");
}
void loop(void) {
  unsigned long currentMillis = millis();
  digitalWrite(LED_PIN_RED, 0);
  digitalWrite(LED_PIN_BLUE, 0);
  digitalWrite(LED_PIN_GREEN, 60);
  boolean success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };          // Buffer to store the returned UID
  uint8_t uidLength;                                // Length of the UID (4 or 7 bytes depending on
ISO14443A card type)
  // Wait for an ISO14443A type cards (Mifare, etc.).  When one is found
  // 'uid' will be populated with the UID, and uidLength will indicate
  // if the uid is 4 bytes (Mifare Classic) or 7 bytes (Mifare Ultralight)
  success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);
  if (success) {
    digitalWrite(LED_PIN_RED, 60);
    digitalWrite(LED_PIN_BLUE, 60);
    digitalWrite(LED_PIN_GREEN, 0);
    Serial.println("Found a card!");
    Serial.print("UID Length: ");Serial.print(uidLength, DEC);Serial.println(" bytes");
    Serial.print("UID Value: ");
    for (uint8_t i=0; i < uidLength; i++)
    {
      Serial.print(" 0x");Serial.print(uid[i], HEX);
    }
```

```
    Serial.println("");
            // Wait 1 second before continuing
            delay(1000);
  }
  else
  {
    // PN532 probably timed out waiting for a card
    Serial.println("Timed out waiting for a card");
  }
}
```

LoggingAspect.java

```
package com.demianenko.aop.logging;
import io.github.jhipster.config.JHipsterConstants;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.core.env.Environment;
import org.springframework.core.env.Profiles;
import java.util.Arrays;
/**
 * Aspect for logging execution of service and repository Spring components.
 *
 * By default, it only runs with the "dev" profile.
 */
@Aspect
public class LoggingAspect {
    private final Logger log = LoggerFactory.getLogger(this.getClass());
    private final Environment env;
    public LoggingAspect(Environment env) {
        this.env = env;
    }
    /**
     * Pointcut that matches all repositories, services and Web REST endpoints.
     */
    @Pointcut("within(@org.springframework.stereotype.Repository *)" +
        " || within(@org.springframework.stereotype.Service *)" +
        " || within(@org.springframework.web.bind.annotation.RestController *)")
    public void springBeanPointcut() {
        // Method is empty as this is just a Pointcut, the implementations are in the advices.
    }
    /**
     * Pointcut that matches all Spring beans in the application's main packages.
     */
    @Pointcut("within(com.demianenko.repository..*)"+
        " || within(com.demianenko.service..*)"+
```

```
      " || within(com.demianenko.web.rest..*)")
   public void applicationPackagePointcut() {
      // Method is empty as this is just a Pointcut, the implementations are in the advices.
   }
   /**
    * Advice that logs methods throwing exceptions.
    *
    * @param joinPoint join point for advice.
    * @param e exception.
    */
   @AfterThrowing(pointcut = "applicationPackagePointcut() && springBeanPointcut()",
throwing = "e")
   public void logAfterThrowing(JoinPoint joinPoint, Throwable e) {
      if
(env.acceptsProfiles(Profiles.of(JHipsterConstants.SPRING_PROFILE_DEVELOPMENT))) {
         log.error("Exception in {}.{}() with cause = \'{}\' and exception = \'{}\'",
joinPoint.getSignature().getDeclaringTypeName(),
            joinPoint.getSignature().getName(), e.getCause() != null? e.getCause() : "NULL",
e.getMessage(), e);
      } else {
         log.error("Exception in {}.{}() with cause = {}",
joinPoint.getSignature().getDeclaringTypeName(),
            joinPoint.getSignature().getName(), e.getCause() != null? e.getCause() : "NULL");
      }
   }
   /**
    * Advice that logs when a method is entered and exited.
    *
    * @param joinPoint join point for advice.
    * @return result.
    * @throws Throwable throws {@link IllegalArgumentException}.
    */
   @Around("applicationPackagePointcut() && springBeanPointcut()")
   public Object logAround(ProceedingJoinPoint joinPoint) throws Throwable {
      if (log.isDebugEnabled()) {
         log.debug("Enter: {}.{}() with argument[s] = {}",
joinPoint.getSignature().getDeclaringTypeName(),
            joinPoint.getSignature().getName(), Arrays.toString(joinPoint.getArgs()));
      }
      try {
         Object result = joinPoint.proceed();
         if (log.isDebugEnabled()) {
            log.debug("Exit: {}.{}() with result = {}",
joinPoint.getSignature().getDeclaringTypeName(),
               joinPoint.getSignature().getName(), result);
         }
         return result;
      } catch (IllegalArgumentException e) {
         log.error("Illegal argument: {} in {}.{}()", Arrays.toString(joinPoint.getArgs()),
            joinPoint.getSignature().getDeclaringTypeName(),
joinPoint.getSignature().getName());
         throw e;
```

```
        }
    }
}
```

ApplicationWebXml.java

```
package com.demianenko;
import com.demianenko.config.DefaultProfileUtil;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;
/**
 * This is a helper Java class that provides an alternative to creating a {@code web.xml}.
 * This will be invoked only when the application is deployed to a Servlet container like Tomcat,
JBoss etc.
 */
public class ApplicationWebXml extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        /**
         * set a default to use when no profile is configured.
         */
        DefaultProfileUtil.addDefaultProfile(application.application());
        return application.sources(SmartNfcApp.class);
    }
}
```

ApplicationProperties.java

```
package com.demianenko.config;
import org.springframework.boot.context.properties.ConfigurationProperties;
/**
 * Properties specific to Smart NFC.
 * <p>
 * Properties are configured in the {@code application.yml} file.
 * See {@link io.github.jhipster.config.JHipsterProperties} for a good example.
 */
@ConfigurationProperties(prefix = "application", ignoreUnknownFields = false)
public class ApplicationProperties {
}
```

AsyncConfiguration.java

```
package com.demianenko.config;
import io.github.jhipster.async.ExceptionHandlingAsyncTaskExecutor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.aop.interceptor.AsyncUncaughtExceptionHandler;
import org.springframework.aop.interceptor.SimpleAsyncUncaughtExceptionHandler;
import org.springframework.boot.autoconfigure.task.TaskExecutionProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.AsyncConfigurer;
```

```
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;
import java.util.concurrent.Executor;
@Configuration
@EnableAsync
@EnableScheduling
public class AsyncConfiguration implements AsyncConfigurer {
   private final Logger log = LoggerFactory.getLogger(AsyncConfiguration.class);
   private final TaskExecutionProperties taskExecutionProperties;
   public AsyncConfiguration(TaskExecutionProperties taskExecutionProperties) {
      this.taskExecutionProperties = taskExecutionProperties;
   }
   @Override
   @Bean(name = "taskExecutor")
   public Executor getAsyncExecutor() {
      log.debug("Creating Async Task Executor");
      ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
      executor.setCorePoolSize(taskExecutionProperties.getPool().getCoreSize());
      executor.setMaxPoolSize(taskExecutionProperties.getPool().getMaxSize());
      executor.setQueueCapacity(taskExecutionProperties.getPool().getQueueCapacity());
      executor.setThreadNamePrefix(taskExecutionProperties.getThreadNamePrefix());
      return new ExceptionHandlingAsyncTaskExecutor(executor);
   }
   @Override
   public AsyncUncaughtExceptionHandler getAsyncUncaughtExceptionHandler() {
      return new SimpleAsyncUncaughtExceptionHandler();
   }
}
```

AuditEventConverter.java

```
package com.demianenko.config.audit;
import com.demianenko.domain.PersistentAuditEvent;
import org.springframework.boot.actuate.audit.AuditEvent;
import org.springframework.security.web.authentication.WebAuthenticationDetails;
import org.springframework.stereotype.Component;
import java.util.*;
@Component
public class AuditEventConverter {
   /**
    * Convert a list of {@link PersistentAuditEvent}s to a list of {@link AuditEvent}s.
    *
    * @param persistentAuditEvents the list to convert.
    * @return the converted list.
    */
   public List<AuditEvent> convertToAuditEvent(Iterable<PersistentAuditEvent>
persistentAuditEvents) {
      if (persistentAuditEvents == null) {
         return Collections.emptyList();
      }
      List<AuditEvent> auditEvents = new ArrayList<>();
```

```
        for (PersistentAuditEvent persistentAuditEvent : persistentAuditEvents) {
            auditEvents.add(convertToAuditEvent(persistentAuditEvent));
        }
        return auditEvents;
    }
    /**
     * Convert a {@link PersistentAuditEvent} to an {@link AuditEvent}.
     *
     * @param persistentAuditEvent the event to convert.
     * @return the converted list.
     */
    public AuditEvent convertToAuditEvent(PersistentAuditEvent persistentAuditEvent) {
        if (persistentAuditEvent == null) {
            return null;
        }
        return new AuditEvent(persistentAuditEvent.getAuditEventDate(),
persistentAuditEvent.getPrincipal(),
            persistentAuditEvent.getAuditEventType(),
convertDataToObjects(persistentAuditEvent.getData()));
    }
    /**
     * Internal conversion. This is needed to support the current SpringBoot actuator {@code
AuditEventRepository} interface.
     *
     * @param data the data to convert.
     * @return a map of {@link String}, {@link Object}.
     */
    public Map<String, Object> convertDataToObjects(Map<String, String> data) {
        Map<String, Object> results = new HashMap<>();
        if (data != null) {
            for (Map.Entry<String, String> entry : data.entrySet()) {
                results.put(entry.getKey(), entry.getValue());
            }
        }
        return results;
    }
    /**
     * Internal conversion. This method will allow to save additional data.
     * By default, it will save the object as string.
     *
     * @param data the data to convert.
     * @return a map of {@link String}, {@link String}.
     */
    public Map<String, String> convertDataToStrings(Map<String, Object> data) {
        Map<String, String> results = new HashMap<>();
        if (data != null) {
            for (Map.Entry<String, Object> entry : data.entrySet()) {
                // Extract the data that will be saved.
                if (entry.getValue() instanceof WebAuthenticationDetails) {
                    WebAuthenticationDetails authenticationDetails = (WebAuthenticationDetails)
entry.getValue();
                    results.put("remoteAddress", authenticationDetails.getRemoteAddress());
```

```
                results.put("sessionId", authenticationDetails.getSessionId());
            } else {
                results.put(entry.getKey(), Objects.toString(entry.getValue()));
            }
        }
    }
    return results;
  }
}
```

package-info.java

```
/**
 * Audit specific code.
 */
package com.demianenko.config.audit;
```

CacheConfiguration.java

```
package com.demianenko.config;
import java.time.Duration;
import org.ehcache.config.builders.*;
import org.ehcache.jsr107.Eh107Configuration;
import org.hibernate.cache.jcache.ConfigSettings;
import io.github.jhipster.config.JHipsterProperties;
import org.springframework.boot.autoconfigure.cache.JCacheManagerCustomizer;
import org.springframework.boot.autoconfigure.orm.jpa.HibernatePropertiesCustomizer;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.context.annotation.*;
@Configuration
@EnableCaching
public class CacheConfiguration {
  private final javax.cache.configuration.Configuration<Object, Object> jcacheConfiguration;
  public CacheConfiguration(JHipsterProperties jHipsterProperties) {
    JHipsterProperties.Cache.Ehcache ehcache = jHipsterProperties.getCache().getEhcache();
    jcacheConfiguration = Eh107Configuration.fromEhcacheCacheConfiguration(
      CacheConfigurationBuilder.newCacheConfigurationBuilder(Object.class, Object.class,
        ResourcePoolsBuilder.heap(ehcache.getMaxEntries()))

.withExpiry(ExpiryPolicyBuilder.timeToLiveExpiration(Duration.ofSeconds(ehcache.getTimeT
oLiveSeconds())))
          .build());
  }
  @Bean
  public HibernatePropertiesCustomizer
hibernatePropertiesCustomizer(javax.cache.CacheManager cacheManager) {
    return hibernateProperties -> hibernateProperties.put(ConfigSettings.CACHE_MANAGER,
cacheManager);
  }
  @Bean
  public JCacheManagerCustomizer cacheManagerCustomizer() {
    return cm -> {
```

```
            createCache(cm,
com.demianenko.repository.UserRepository.USERS_BY_LOGIN_CACHE);
            createCache(cm,
com.demianenko.repository.UserRepository.USERS_BY_EMAIL_CACHE);
            createCache(cm, com.demianenko.domain.User.class.getName());
            createCache(cm, com.demianenko.domain.Authority.class.getName());
            createCache(cm, com.demianenko.domain.User.class.getName() + ".authorities");
            createCache(cm, com.demianenko.domain.Description.class.getName());
            createCache(cm, com.demianenko.domain.Category.class.getName());
            createCache(cm, com.demianenko.domain.Field.class.getName());
            createCache(cm, com.demianenko.domain.AccData.class.getName());
            createCache(cm, com.demianenko.domain.UserAccount.class.getName());
            createCache(cm, com.demianenko.domain.UserAccount.class.getName() + ".accData");
            createCache(cm, com.demianenko.domain.UserAccount.class.getName() +
".categories");
            createCache(cm, com.demianenko.domain.Category.class.getName() + ".userAccounts");
            createCache(cm, com.demianenko.domain.Organization.class.getName());
            createCache(cm, com.demianenko.domain.Organization.class.getName() + ".managers");
            createCache(cm, com.demianenko.domain.Organization.class.getName() + ".workers");
            createCache(cm, com.demianenko.domain.UserExtra.class.getName());
            createCache(cm, com.demianenko.domain.UserExtra.class.getName() +
".placeOfManages");
            createCache(cm, com.demianenko.domain.UserExtra.class.getName() +
".placeOfWorks");
            createCache(cm, com.demianenko.domain.UserExtra.class.getName() +
".userAccounts");
            // jhipster-needle-ehcache-add-entry
        };
    }
    private void createCache(javax.cache.CacheManager cm, String cacheName) {
        javax.cache.Cache<Object, Object> cache = cm.getCache(cacheName);
        if (cache != null) {
            cm.destroyCache(cacheName);
        }
        cm.createCache(cacheName, jcacheConfiguration);
    }
}
```

CloudDatabaseConfiguration.java

```
package com.demianenko.config;
import io.github.jhipster.config.JHipsterConstants;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.cloud.config.java.AbstractCloudConfig;
import org.springframework.context.annotation.*;
import javax.sql.DataSource;
import org.springframework.boot.context.properties.ConfigurationProperties;
@Configuration
@Profile(JHipsterConstants.SPRING_PROFILE_CLOUD)
public class CloudDatabaseConfiguration extends AbstractCloudConfig {
    private final Logger log = LoggerFactory.getLogger(CloudDatabaseConfiguration.class);
```

```
    private static final String CLOUD_CONFIGURATION_HIKARI_PREFIX =
"spring.datasource.hikari";
    @Bean
    @ConfigurationProperties(CLOUD_CONFIGURATION_HIKARI_PREFIX)
    public DataSource dataSource() {
        log.info("Configuring JDBC datasource from a cloud provider");
        return connectionFactory().dataSource();
    }
}
```

Constants.java

```
package com.demianenko.config;
/**
 * Application constants.
 */
public final class Constants {
    // Regex for acceptable logins
    public static final String LOGIN_REGEX = "^[_.@A-Za-z0-9-]*$";
    public static final String SYSTEM_ACCOUNT = "system";
    public static final String DEFAULT_LANGUAGE = "en";
    public static final String ANONYMOUS_USER = "anonymoususer";
    private Constants() {
    }
}
```

DatabaseConfiguration.java

```
package com.demianenko.config;
import io.github.jhipster.config.JHipsterConstants;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.transaction.annotation.EnableTransactionManagement;
@Configuration
@EnableJpaRepositories("com.demianenko.repository")
@EnableJpaAuditing(auditorAwareRef = "springSecurityAuditorAware")
@EnableTransactionManagement
public class DatabaseConfiguration {
    private final Logger log = LoggerFactory.getLogger(DatabaseConfiguration.class);
}
```

ДОДАТОК Б

КОПІЇ ПУБЛІКАЦІЙ