Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «бакалавр»

на тему:

ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ РОЗШИРЕНИХ МЕТОДІВ АВТЕНТИФІКАЦІЇ ДЛЯ СИСТЕМ КОНТРОЛЮ ДОСТУПУ

Виконав:
здобувач IV курсу
групи КН-41
спеціальності 122 «Комп’ютерні науки»
Мартиненков Максим Олександрович

Науковий керівник:
к.ю.н., доцент Кіндрат П. В.

Рівне – 2024

ЗМІСТ

СПИСОК ТЕРМІНІВ ТА УМОВНИХ ПОЗНАЧЕНЬ

IoT	Internet of Things – система фізичних об’єктів («речей»), взаємопов’язаних між собою за допомогою вбудованих датчиків, програмного забезпечення та/або інших технологій
AI	Artificial intelligence – розділ комп’ютерної лінгвістики та інформатики і зосереджений на розробці інтелектуальних машин
Blockchain	Blockchain – розподілена база даних, що зберігає впорядкований ланцюжок записів (так званих блоків), який постійно довшає.
IC	Інформаційна система – сукупність організаційних і технічних засобів для збереження та обробки інформації з метою забезпечення інформаційних потреб користувачів.
U2F	Universal 2nd Factor – відкритий, бездрайверний протокол для двофакторної автентифікації, заснований на виклик-відповідь автентифікації

MFA	Багатофакторна автентифікація – метод контролю доступу до комп’ютера в якому користувачеві для отримання доступу до інформації необхідно пред’явити більше одного «доказу механізму аутентифікації».
HTTP	HyperText Transfer Protocol – протокол передачі даних, що використовується в комп’ютерних мережах.
REST	Representational State Transfer – підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів
URL	Uniform Resource Locator – стандартизована адреса певного ресурсу в інтернеті.
Wi-Fi	Wireless Fidelity – загальновживана назва для стандарту IEEE 802.11 передавання цифрових потоків даних по радіоканалах
CRUD	Операції create, read, update, delete
EF	Entity Framework

ВСТУП

Актуальність роботи. З кожним днем вимоги до безпеки в цифровому середовищі зростають, особливо в контексті доступу до конфіденційної інформації та особистих даних. Вірогідність кіберзлочинності, включаючи зловживання даними, фішинг, крадіжку особистих ідентифікаційних даних та хакерські атаки, постійно зростає. У цьому контексті системи контролю доступу, які включають методи аутентифікації, стають критичними для забезпечення безпеки і захисту інформації.

Наприклад, у сфері фінансів, де чутлива інформація про фінансові ресурси та транзакції потребує особливого захисту, недостатньо простих паролів або підписів. Сучасні методи аутентифікації, такі як біометричні дані, двофакторна аутентифікація та мультифакторна аутентифікація, стають все більш важливими для підтвердження ідентичності користувача та запобігання несанкціонованому доступу.

У світі Інтернету речей (IoT) обмін даними між підключеними пристроями стає невід'ємною частиною нашого повсякденного життя. Забезпечення безпеки та конфіденційності в цьому середовищі є критично важливим завданням, оскільки вразливість одного пристрою може викликати ланцюгову реакцію проблем у всій мережі. Тому розробка ефективних методів аутентифікації для пристроїв IoT стає надзвичайно важливою для забезпечення безпеки та недопущення недозволеного доступу до приватних даних.

Також, у сфері медицини, де конфіденційність медичних записів є критичною, надійна система контролю доступу є обов'язковою для забезпечення конфіденційності пацієнтів та запобігання несанкціонованому доступу до медичних даних.

Отже, від розробки нових, надійних та ефективних методів аутентифікації залежить не лише захист конфіденційної інформації, але й відвертається потенційна загроза для безпеки та приватності в сучасному цифровому світі. Тому розробка та реалізація розширених методів аутентифікації для систем контролю доступу є безсумнівно актуальною задачею.

Метою роботи є дослідження, теоретичне обґрунтування та реалізація розширених методів аутентифікації для систем контролю доступу з метою підвищення ефективності та безпеки інформаційних ресурсів. Дослідження спрямоване на проектування та реалізацію нових методів аутентифікації, а також на їх апробацію та аналіз результатів експериментів. Завдання дослідження включають:

- Дослідження сучасних методів аутентифікації та їх аналіз.
- Проектування розширених методів аутентифікації на основі біометричних даних та аналізу поведінки користувачів.
- Розробка та імплементація розширених методів аутентифікації в реальній системі контролю доступу.
- Експериментальне тестування розроблених методів та аналіз отриманих результатів.

Об’єктом дослідження є системи контролю доступу, які використовуються для забезпечення безпеки інформаційних ресурсів та контролю доступу до них. Дослідження спрямоване на аналіз функціонування таких систем та визначення можливостей їх покращення.

Предметом дослідження є розширені методи автентифікації, які використовуються для підвищення рівня безпеки в системах контролю доступу. В рамках дослідження будуть детально вивчені та проаналізовані такі методи з метою їх подальшої реалізації та ефективного використання у практичних застосуваннях.

Методи дослідження:

- Аналіз сучасних технологій автентифікації: Цей метод передбачає детальний аналіз існуючих технологій та методів аутентифікації в системах контролю доступу. Він включає в себе вивчення принципів роботи, сильних і слабких сторін, а також можливостей подальшого вдосконалення.

- Проектування та розробка розширених методів автентифікації: Цей метод включає розробку нових методів аутентифікації на основі вивчення сучасних технологій та інноваційних підходів. Він передбачає аналіз можливих шляхів покращення систем контролю доступу та розробку нових алгоритмів та технологій для забезпечення вищого рівня безпеки.
- Експериментальне тестування та оцінка розроблених методів: Цей метод включає проведення експериментів для перевірки ефективності та надійності розроблених методів аутентифікації. Він передбачає проведення тестів у реальних умовах або за допомогою симуляційних середовищ для оцінки їх працездатності та визначення можливих обмежень чи недоліків.

Практичне значення дослідження полягає в можливості впровадження розроблених розширених методів аутентифікації у реальні системи контролю доступу. Вдосконалення систем аутентифікації може сприяти підвищенню рівня безпеки інформаційних ресурсів, запобіганню несанкціонованого доступу до них та збільшенню зручності користування такими системами для кінцевих користувачів.

Крім того, результати дослідження можуть бути використані в інших галузях, де необхідний високий рівень безпеки та контролю доступу, таких як банківська сфера, медична індустрія, транспортні системи та інші.

Апробація результатів дослідження. Основні теоретичні та практичні результати дослідження доповідалися та обговорювалися на XVII Всеукраїнській науково-практичній конференції здобувачів вищої освіти і молодих учених «Наука, освіта, суспільство очима молодих» (м. Рівне, 17 травня 2024 р.).

Структура роботи. Дипломна робота складається зі вступу, трьох розділів, висновків, переліку використаних джерел та додатків. Загальний обсяг роботи становить 66 сторінок. Вона містить 13 рисунків та 13 таблиць. Список використаних джерел включає 21 найменування. Обсяг додатків – 38 сторінок

РОЗДІЛ 1
ЗАВДАННЯ АУТЕНТИФІКАЦІЇ В СИСТЕМАХ КОНТРОЛЮ ДОСТУПУ

1.1 Методи аутентифікації як елемент систем контролю доступу.

У сучасному світі системи контролю доступу є ключовим елементом забезпечення безпеки в різноманітних середовищах, від комп'ютерних систем до фізичних приміщень. Методи аутентифікації, що використовуються в цих системах, відіграють важливу роль у підтвердженні ідентичності користувачів та забезпеченні конфіденційності даних. Зазначені методи включають в себе різноманітні підходи до перевірки автентичності користувача, такі як використання паролів, біометричних даних, токенів або комбінацій різних методів.

Завдяки розвитку технологій, методи аутентифікації стають все більш надійними та ефективними, забезпечуючи високий рівень безпеки та зручності для користувачів. Крім того, впровадження двофакторної аутентифікації, яка вимагає введення двох різних видів інформації для підтвердження ідентичності користувача, забезпечує додатковий рівень захисту. Наприклад, у такій системі користувач може пред'являти пароль разом із токеном або кодом, отриманим на мобільний пристрій.

Базова автентифікація – це найпростіший спосіб автентифікації, що дійсно використовують у реальних системах. Цей спосіб полягає в тому, що клієнт (поняття з клієнт-серверної архітектури) разом з кожним запитом, що вимагає автентифікації відправляє свій логін та пароль. Для цього у запиті вказується заголовок з назвою Authorization, значенням якого є спочатку назва способу автентифікації – Basic, а потім рядок, що складається з логіну та паролю, які закодовані за допомогою Base64. Схему базової автентифікації представлено на рисунку 1.1.

Рисунок 1.1 – Схема базової автентифікації

Іншим способом є авторизація за допомогою токенів. Токен – це цифровий ключ, який фактично є рядком, який складається з послідовності символів, що випадково згенеровані. Він не містить у собі ніякої інформації про справжній логін чи пароль та використовується лише для ідентифікації користувача. Для того щоб отримати токен клієнт, на спеціальну адресу, повинен відправити запит, що містить інформацію необхідну для автентифікації. Зазвичай – це інформація облікового запису: логін та пароль користувача. Сервер, у свою чергу, згенерує токен та поверне його клієнту. Цей токен буде асоційовано з користувачем і збережено на сервері в його сховищі даних. У подальших викликах клієнт зможе використовувати цей токен, вказавши його у заголовку Authorization, разом із назвою способу – Bearer. А сервер, використовуючи сховище даних, зрозуміє якому користувачу належить цей токен та зможе виконати процес автентифікації та авторизації. Схема, на якій зображено механізм авторизації за допомогою токенів представлено на рисунку 1.2.

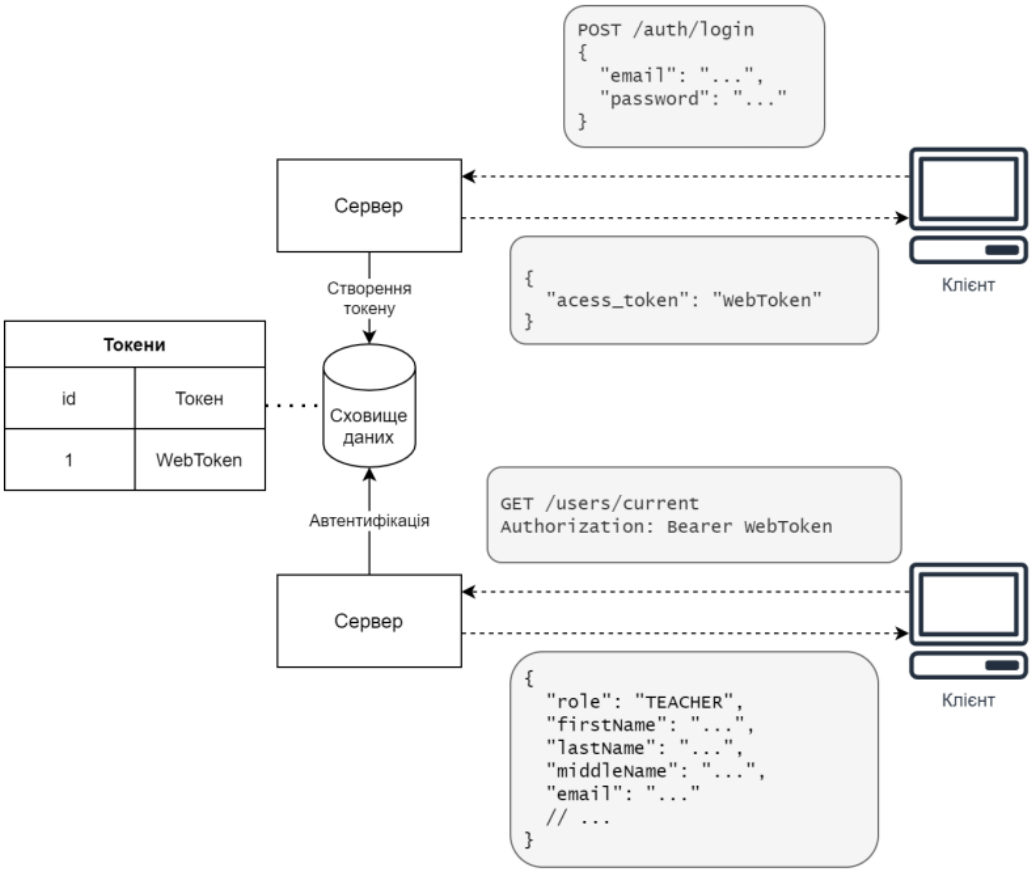


Рисунок 1.2 – Схема авторизації за допомогою токенів

Підхід зі збереженням токенів зручний, але він вимагає зберігати усі токени в одному сховищі даних, що може бути не зручним у розподілених системах. В них кожен модуль вимагає, що вимагає автентифікації, повинен буте звертатись до модулю авторизації, щоб ідентифікувати користувача. Так збільшується зв’язок між модулями, а також навантаження на модуль авторизації. Для вирішення цієї проблеми можна помістити інформацію про користувача у сам токен, та зашифрувати його. Таким чином працює JSON Web Token, або скорочено – JWT.

JWT – це стандарт по створенню токенів доступу, що оснований на форматі JSON. Токен, що створений по стандарту JWT складається з трьох частин: заголовку, корисного навантаження та криптографічного підпису. Схема автентифікації за допомогою JWT зображена на рисунку 1.3.

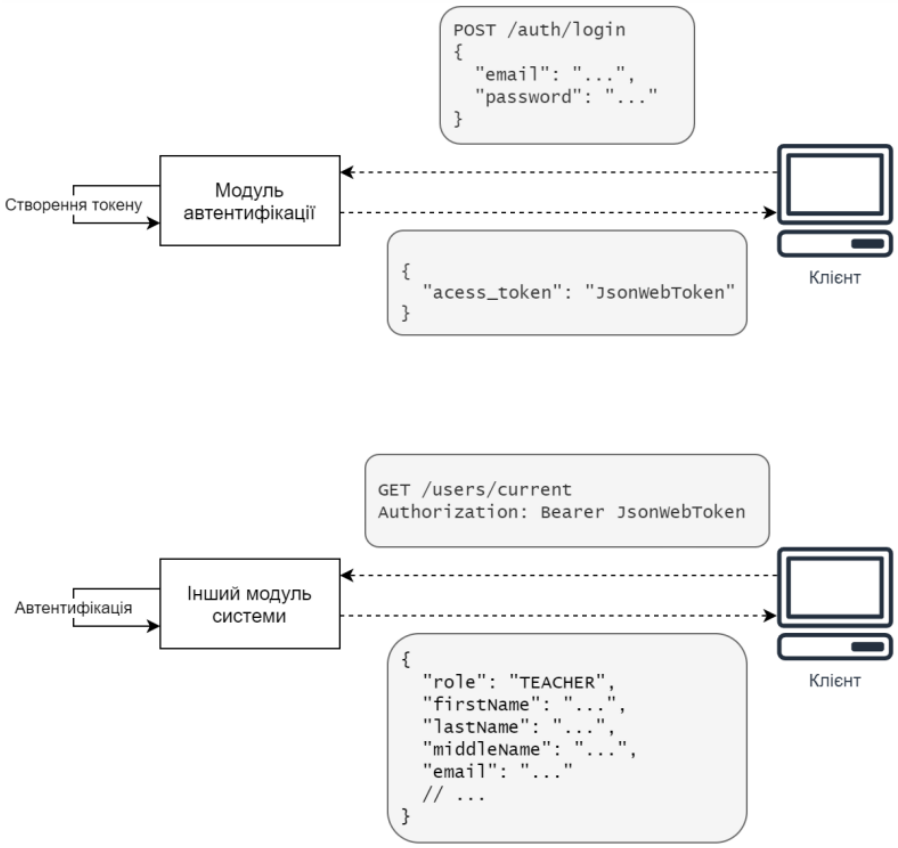


Рисунок 1.3 – Автентифікація за допомогою JWT

З появою нових технологій, таких як IoT, які дозволяють різним пристроям спілкуватися та обмінюватися даними між собою, необхідно забезпечувати не лише безпеку доступу для людей, але й для пристроїв. Інформація, яка обробляється та передається через ці пристрої, може бути такою ж важливою та конфіденційною, як і та, що обробляється людьми. Тому важливо розробляти та використовувати ефективні методи аутентифікації для забезпечення безпеки цієї інформації.

Одним із напрямів розвитку систем контролю доступу є використання біометричних технологій [9], таких як сканування відбитків пальців, розпізнавання обличчя або сканування райдужки ока. Ці технології базуються на унікальних фізичних або поведінкових характеристиках особи, що робить їх важко підробити або обійти. Вони можуть бути ефективними засобами аутентифікації, особливо там, де вимагається високий рівень безпеки, наприклад, у фінансових установах або на об’єктах з високим рівнем конфіденційності.

Ще одним важливим аспектом є розвиток систем адаптивного контролю доступу, які враховують контекст інтеракції користувача з системою прийняття рішень про надання доступу. Такі системи використовують інформацію про поведінку користувача, його місцезнаходження, час доступу та інші фактори для адаптації рівня доступу та заходів безпеки. Наприклад, якщо система виявляє, що користувач намагається отримати доступ до системи з незвичної локації або в незвичний для нього час, вона може запитати додаткові підтвердження або застосувати додаткові заходи безпеки.

Ще одним трендом у розвитку методів аутентифікації є використання штучного інтелекту та машинного навчання для виявлення аномальної поведінки користувачів. Алгоритми можуть аналізувати типові патерни взаємодії користувачів з системою та виявляти незвичайні або підозрілі дії, що можуть свідчити про спроби несанкціонованого доступу.

Трапляються випадки, коли працівникам компанії, або особам, яких компанія залучає до роботи, потрібен доступ до інформації, наприклад, до документів, слайдів або баз даних, що містяться на мережевому сервері, але вони не мають відповідного рівня доступу для читання та / або зміни відповідної інформації. Така ситуація може статися в не найзручніший для компанії час, і таким особам необхідно, або займати специфічні посади, з відповідним рівнем допуску, або ж пройти цілу низку дозвільних процедур, щоб отримати рівень доступу до інформації, необхідний для здійснення своїх функціональних обов’язків. Варто підкреслити, що вжиття компанією заходів контролю доступу до інформації, з одного боку забезпечує певний рівень інформаційної безпеки, в той час як з іншої сторони – може створювати перешкоди для здійснення певних бізнес операцій. Для унеможливлення виникнення таких ситуацій, в рамках компанії повинен бути врегульований порядок доступу до інформації (information access control) на належному рівні, із врахуванням всіх особливостей діяльності компанії, а також особливістю інформації, яка нею використовується.

1.1.1 Контроль доступу до інформації

Контроль доступу (access control) – в загальному розумінні це процес ідентифікації осіб, які виконують певну роботу, а також їхня автентифікація в контексті отриманих результатів ідентифікації. І тільки автентифікованим особам можуть надаватися інструменти доступу до певних засобів (наприклад пароль доступу до комп'ютера чи певної системи). У світі захисту інформатизації, це можна розглядати як надання індивідуального дозволу на потрапляння в мережу через ім'я користувача та пароль, що дозволяє їм отримувати доступ до файлів, комп'ютерів чи іншого апаратного та програмного забезпечення, яке вимагає особа з ціллю виконання поставлених компанією завдань.

Отже, в контексті надання особі потрібного рівня доступу до інформації, необхідно розглянути моделі контролю доступу, які діляться на чотири основні види:

- Модель обов'язкового контролю доступу (Mandatory Access Control (MAC));
- Модель контролю доступу, в контексті визначеної ролі (посади) (Role Based Access Control (RBAC));
- Модель дискреційного контролю доступу (Discretionary Access Control (DAC));
- Модель контролю доступу на основі (встановлених) правил (Rule Based Access Control (RBAC or RB-RBAC)).

Mandatory Access Control (MAC)

Модель обов'язкового контролю доступу надає право контролю доступу лише власникам компанії та авторизованим менеджерам компанії. Це означає, що кінцевий користувач не має контролю над будь-якими налаштуваннями в рамках інформаційної системи. Зараз є дві моделі безпеки, пов'язані з MAC:

- Biba;
- Bell-LaPadula.

Модель Biba орієнтована на цілісність інформації, тоді як модель Bell-LaPadula орієнтована на конфіденційність інформації.

Biba – це модель, в рамках якої користувач з низьким рівнем допуску до інформації може ознайомлюватися з інформацією більш високого рівня (так званий «read up»), а користувач з високим рівнем допуску має можливість ознайомлювати, в письмовому вигляді, користувачів з нижчим рівнем допуску з інформацією вищого рівня (так званий «write down»). Модель Biba зазвичай використовується в компаніях, де працівники нижчих рівнів можуть читати інформацію вищого рівня, а керівники можуть надавати інформацію в письмовому вигляді з ціллю інформувати працівників нижчого рівня.

Bell-LaPadula – це модель яка визначає рівні доступу в залежності від ступеня конфіденційності інформації. Модель Bell-LaPadula передбачає, що користувач з високим рівнем допуску має можливість розкривати інформацією лише користувачам на цьому ж рівні і не нижче (так званий «write up»), але також може ознайомлюватися з інформацією на нижчих рівнях допуску (так званий «read down»). Таку модель зазвичай використовують в державних або військових установах. Наприклад, в деяких арміях світу, можливо зустріти градацію конфіденційності за рівнями секретності, конфіденційності та доступності (наприклад: «дуже секретно», «секретно», «конфіденційно», «у відкритому доступі»).

Role Based Access Control (RBAC)

Контроль доступу в контексті визначеної ролі забезпечує контроль доступу на основі посади, яку займає окрема особа в компанії. Отже, наприклад, замість отримання додаткових дозволів на доступ до певної інформації, яка може доступна тільки працівникам безпеки, особа отримує такий доступ автоматично в момент призначення його на посаду, пов’язану із забезпеченням безпеки компанії. Зайняття певної посади в рамках компанії автоматично передбачає надання дозволів на доступ до інформації на певному рівні. Ця модель є також доволі практичною для використання компаніями, як володіють інформацією, що не може бути доступною для всіх працівників.

Discretionary Access Control (DAC)

Дискреційна модель контролю доступу до інформації є найменш обмежувальною моделлю порівняно з найбільш обмежувальною моделлю Mandatory Access Control. DAC дозволяє окремій особі здійснювати повний контроль над будь-якими об’єктами, якими вона володіє, а також програмами, пов’язаними з цими об’єктами. Така модель може мати слабкі сторони, які полягають в тому, що така модель дає кінцевому користувачеві повний контроль для встановлення параметрів рівня безпеки для інших користувачів, що може призвести до того, що такі кінцеві користувачі в рамках компанії матимуть більш високі привілеї, ніж вони повинні мати.

Rule Based Access Control (RBAC or RB-RBAC)

Контроль доступу на основі правил – модель, яка передбачає наявність технічних інструментів, які передбачають можливість динамічного присвоєння користувачам ролей на основі критеріїв, визначених, наприклад, системним адміністратором Компанії або іншим відповідальним спеціалістом в сфері інформаційної безпеки. Якщо будь-кому надається доступ до бази даних лише протягом певних годин дня, то такий доступ повинен регламентуватися певними правилами або ж затвердженим порядком, який є обов’язковим для погодження всіма працівниками компанії.

1.1.2 Ознайомлення з методами автентифікації користувачів

У сучасному світі інформаційні системи (ІС) є невід’ємною складовою інфраструктури держави, бізнесу та громадянського суспільства. Їх роль у збереженні та обробці інформації стає все більш суттєвою. Однак, разом з новими можливостями, які надають сучасні технології, зростає і необхідність забезпечення безпеки для захисту цієї інформації.

Статистика свідчить, що понад 25% порушень безпеки в ІС відбуваються через внутрішніх користувачів, партнерів та постачальників послуг, які мають прямий доступ до систем. Близько 70% із них становлять випадки несанкціонованого отримання прав та привілеїв, крадіжки та передачі облікових

даних користувачів ІС, що стає можливим через недосконалість технологій розмежування доступу та автентифікації. Удосконалення методів управління доступом та реєстрації користувачів є одним з пріоритетних напрямків розвитку інформаційних систем.

Однією з основних процедур реєстрації користувачів в ІС є процедура ідентифікації - відповідь на питання «Хто Ви?» та автентифікація - підтвердження того, що «Ви саме той, за кого себе видаєте». Несанкціонований доступ до ІС переважно пов’язаний з порушенням процедур автентифікації.

Автентифікація - це процедура встановлення ідентичності користувача в системі на основі наданого ним ідентифікатора. Один із способів аутентифікації в інформаційній системі полягає в попередній ідентифікації на основі логіну та пароля користувача - конфіденційної інформації, знання якої підтверджує володіння певним ресурсом в мережі. Після введення логіну та пароля, комп’ютер порівнює їх зі значенням, збереженим в захищеній базі даних, і у разі успішної аутентифікації виконує авторизацію користувача для подальшої роботи в системі.

Традиційну автентифікацію за допомогою пароля називають однофакторною або слабкою, оскільки за наявності відповідних ресурсів перехоплення або підбору пароля можливе порушення безпеки. Таким чином, часто виникає необхідність у застосуванні сильної або багатофакторної автентифікації - на основі двох чи більше факторів. Використання багатофакторної автентифікації базується на припущенні, що несанкціонований користувач не зможе надати всі необхідні фактори для доступу. Якщо хоча б один з компонентів відсутній або вказаний невірно, ідентифікація користувача не встановлюється з достатнім ступенем впевненості, і доступ залишається заблокованим.

1.1.3 Методи багатофакторної автентифікації

Фактори багатофакторної автентифікації можуть включати:

- Деякий фізичний об’єкт, що перебуває в у володінні користувача, такий як USB-накопичувач з секретним токеном, банківська карта, ключ тощо.

- Якийсь секрет, відомий лише користувачу, такий як пароль, PIK-код, ТАК і т. д.
- Деякі фізичні характеристики користувача (біометричні дані), такі як відбиток пальця, райдужка ока, голос, швидкість набору тексту, шаблон в інтервалах натискання клавіш і т. д.
- Місцезнаходження, підключення до певної комп'ютерної мережі або використання сигналу GPS для визначення місця розташування.

Фактори знання є найпоширенішою формою автентифікації. У цьому випадку користувач повинен підтвердити знання певного секрету для автентифікації. Пароль - секретне слово або рядок символів, які використовуються для автентифікації користувача. Це найчастіше використовуваний механізм аутентифікації. Багато методів багатофакторної автентифікації покладаються на пароль як один з факторів автентифікації. Варіації включають в себе як більш довгі, сформовані з декількох слів (кодові фрази), так і коротші, числові, персональні ідентифікаційні номери (ПІН), що часто використовуються для доступу до банкоматів. Очікується, що паролі будуть зберігатися в пам'яті користувачів.

Фактори володіння (щось, що належить користувачу і тільки користувачу) використовуються для автентифікації протягом століть, у вигляді ключа до замків. Основний принцип полягає в тому, що ключ уособлює секрет, який поділяється між замком і ключем, цей же принцип лежить в основі аутентифікації за фактором володіння в комп'ютерних системах. Токени - це приклад фактора володіння, вони бувають різних типів:

- Відключені токени - не мають зв'язку з клієнтським комп'ютером. Зазвичай вони використовують вбудований екран для відображення згенерованих даних автентифікації, які користувач вводить вручну.
- Підключені токени - це пристрої, які фізично підключені до використовуваного комп'ютера. Ці пристрої автоматично передають дані на комп'ютер. Існує безліч різних типів таких токенів, включаючи пристрої для читання карток, бездротові токени і ШВ-токени.

- Програмний токен - це тип пристрою двофакторної автентифікації, що може використовуватися для авторизації при використанні комп'ютерних сервісів. Програмні токени можуть зберігатися на будь-якому електронному пристрої, такому як настільний комп'ютер, ноутбук, КПК або мобільний телефон, і його можна дублювати. На відміну від апаратних токенів, де облікові дані зберігаються на виділеному пристрої, і, отже, не можуть бути дубльовані.

Фактори власності. Це фізичні особливості суб'єкта. Це може бути портрет, відбиток пальця або долоні, голос або особливість очка. Для суб'єкта цей спосіб є найпростішим: не потрібно запам'ятовувати пароль, не треба носити з собою пристрій аутентифікації. Однак біометрична система повинна володіти високою чутливістю, щоб підтверджувати авторизованого користувача, але відкидати зловмисника зі схожими біометричними параметрами. Також вартість такої системи досить велика. Але, незважаючи на свої недоліки, біометрика залишається досить перспективним фактором.

Фактор місцезнаходження. Завдяки розвитку технологій все частіше починає застосовуватися четвертий фактор - визначення фізичного розташування користувача. У той час, коли користувач підключений до конкретної корпоративної мережі, він може входити в систему, використовуючи тільки пін-код. Однак при підключенні до іншої мережі може вимагатися використання додаткового фактору автентифікації. Це можна розглядати як прийнятний стандарт, в якому доступ до офісу знаходиться під контролем.

Системи управління доступом до мережі працюють аналогічним чином, коли рівень доступу до системи може залежати від конкретної мережі, до якої підключено ваш пристрій, наприклад, Wi-Fi або дротового зв'язку. Це також дозволяє користувачеві переміщатися між офісами та динамічно отримувати однаковий рівень доступу до мережі в кожному з них.

1.1.4 Двофакторна аутентифікація

Двофакторна аутентифікація (ДФА) є важливим засобом забезпечення безпеки в сучасних інформаційних системах. Вона використовує два різних елементи для перевірки ідентичності користувача: щось, що він знає (наприклад,

пароль), і щось, що він має (наприклад, фізичний токен або мобільний пристрій). Цей підхід робить процес аутентифікації більш надійним, оскільки для несанкціонованого доступу потрібно мати доступ до обох складових, що ускладнює завдання для потенційних зловмисників.

Важливою складовою двофакторної автентифікації є використання мобільних пристроїв як другого фактора ідентифікації. Мобільні телефони стали невід'ємною частиною повсякденного життя, і вони можуть бути ефективно використані для підтвердження ідентичності користувача. Наприклад, за допомогою спеціальних додатків або SMS-повідомлень можуть генеруватися одноразові паролі, які потрібно ввести разом з основним паролем для входу до системи. Це дозволяє забезпечити додатковий шар безпеки без необхідності використання додаткових фізичних пристроїв.

Однією з переваг використання мобільних пристроїв для двофакторної автентифікації є їх масове поширення і доступність. Більшість людей мають мобільний телефон і нерідко носять його з собою весь час. Це робить процес аутентифікації зручним і простим для користувачів, оскільки їм не потрібно носити додаткові пристрої або запам'ятовувати складні паролі.

Однак, важливо враховувати потенційні загрози безпеці, пов'язані з використанням мобільних пристроїв. Наприклад, можливість втрати чи крадіжки телефону може призвести до небажаних наслідків. Також існує ризик компрометації пристрою через віруси або шкідливе програмне забезпечення. Тому важливо вживати заходів безпеки, таких як використання складних паролів і вчасне оновлення програмного забезпечення, щоб забезпечити найвищий рівень захисту.

Є різні комбінації двофакторної автентифікації. Першим фактором є комбінація логіна і пароля, а в ролі другого може

виступати один з приведених нижче конкретних методів двофакторної автентифікації, розглянемо їх переваги та недоліки[7].

SMS

Підтвердження за допомогою SMS-кодів працює дуже просто. Ви, як

завжди, вводите свій логін і пароль, після чого на ваш номер телефону приходить

SMS з кодом, який потрібно ввести для входу в обліковий запис. Це все. При

наступному вході відправляється вже інший SMS-код, дійсний лише для поточної

сесії.

Переваги:

- Генерація нових кодів при кожному вході. Якщо зловмисники перехоплять
- ваш логін і пароль, вони нічого не зможуть зробити без коду.
- Прив’язка до телефонного номеру. Без вашого телефону вхід неможливий.

Недоліки:

- При відсутності сигналу мережі ви не зможете залогінитися.
- Існує теоретична ймовірність підміни номера через послугу оператора або
- працівників салонів зв’язку.
- Якщо ви авторизуєтесь і отримуєте коди на одному і тому ж пристрої (наприклад, смартфоні), то захист перестає бути двофакторним.

U2F Ключі

Універсальний другий фактор (Universal 2nd Factor) – це відкритий

стандарт, який використовується з пристроями USB, пристроями NFC та смарткартками. Щоб автентифікуватися, ви маєте просто підключити його (для USB-ключів), натиснути на нього (для пристроїв NFC) або провести (для смарт-карт) до обладнання.

Переваги:

- Ключ U2F - справжній фізичний фактор. На відміну від SMS-кодів, їх не можна перехопити або перенаправляти. І на відміну від більшості двофакторних методів, ключі U2F є захищеними від фішингу, оскільки вони зареєстровані лише

для роботи з конкретними сайтами, на яких ви зареєстровані. Це один з найнадійніших методів 2FA в даний час.

Недоліки:

- Оскільки U2F – відносно нова технологія, вона ще не так широко підтримується. Наприклад, ключі NFC працюють лише з мобільними пристроями Android, тоді як USB-ключі працюють переважно з браузером Chrome (Firefox працює над цим). Ключі U2F також коштують грошей, часто від 10 до 20 доларів, але можуть збільшитися в залежності від того, наскільки міцний ключ ви хочете.

Додатки-автентифікатори

Цей варіант багато в чому схожий на SMS, з тією лише відмінністю, що, замість отримання кодів по SMS, вони генеруються на пристрої за допомогою спеціального додатку (Google Authenticator, Authy). Під час налаштування ви отримуєте первинний ключ (найчастіше у вигляді QR-коду), на основі якого за допомогою криптографічних алгоритмів генеруються одноразові паролі з терміном дії від 30 до 60 секунд. Навіть якщо припустити, що зловмисники зможуть перехопити 10, 100 або навіть 1 000 паролів, передбачити з їх допомогою, яким буде наступний пароль, просто неможливо.

Переваги:

- Для автентифікатора не потрібен сигнал мережі, тобто достатньо підключення до інтернету лише при первинному налаштуванні.
- Підтримка декількох акаунтів в одному автентифікаторі.

Недоліки:

- Якщо зловмисники отримають доступ до первинного ключа на вашому пристрої або шляхом злому сервера, вони зможуть генерувати паролі в майбутньому.
- При використанні автентифікатора на тому ж пристрої, з якого здійснюється вхід, втрачається двухфакторність.

Перевірка входу за допомогою мобільних додатків

Даний тип автентифікації можна назвати збірною солянкою з усіх попередніх. В цьому випадку, замість запиту кодів або одноразових паролів, ви

повинні підтвердити вхід з вашого мобільного пристрою з встановленим додатком сервісу. На пристрої зберігається приватний ключ, який перевіряється при кожному вході. Це працює в Twitter, Snapchat та різних онлайн-іграх. Наприклад, при вході в ваш Twitter-аккаунт в веб-версії ви вводите логін і пароль, потім на смартфон приходить повідомлення із запитом про вхід, після підтвердження якого в браузері відкривається ваша стрічка.

Переваги:

- Не потрібно нічого вводити при вході.
- Незалежність від мережі.
- Підтримка декількох акаунтів в одному додатку.

Недоліки:

- Якщо зловмисники перехоплять приватний ключ, вони зможуть видавати себе за вас.
- Сенс двофакторної автентифікації втрачається при використанні одного і того ж пристрою для входу.

Апаратні токени

Фізичні (або апаратні) токени є самим надійним способом двофакторно автентифікації. Будучи окремими пристроями, апаратні токени, на відміну від всіх перерахованих вище способів, ні при якому розкладі не втратять своєї двохфакторної складової. Найчастіше вони представлені у вигляді USB-брелоків з власним процесором, що генерує криптографічні ключі, які автоматично вводяться при підключенні до комп’ютера. Вибір ключа залежить від конкретного сервісу. Google, наприклад, рекомендує використовувати маркери стандарту FIDO U2F, ціни на які починаються від 6 доларів без урахування доставки.

Переваги

- Ніяких SMS і додатків.
- Немає необхідності в мобільному пристрої.
- Є повністю незалежним девайсом.

Недоліки

- Потрібно купувати окремо.

- Підтримується не у всіх сервісах.
- При використанні декількох акаунтів доведеться носити цілу в’язку токенів.

Резервні ключі

По суті, це не окремий спосіб, а запасний варіант на випадок втрати або крадіжки смартфону, на який мають приходити одноразові паролі або коди підтвердження. При налаштуванні двофакторної автентифікації в кожному сервісі вам дають кілька резервних ключів для використання в екстрених ситуаціях. З їх допомогою можна увійти в ваш аккаунт, відв’язати налаштовані пристрої та додати нові. Ці ключі варто зберігати в надійному місці, а не у вигляді скріншоту на смартфоні або текстового файлу на комп’ютері.

Обличчя, голос, відбиток пальців

Розпізнавання обличчя, розпізнавання голосу та сканування відбитків пальців підпадають під категорію біометричних даних. Системи використовують біометричну автентифікацію, коли потрібно, щоб ви дійсно були тими, за кого себе видаєте, часто в областях, де потрібна перевірка безпеки (наприклад, уряд).

Переваги:

- Біометрію надзвичайно важко підробити. Навіть відбиток пальців, який, можливо, найпростіший для копіювання, вимагає певного фізичної взаємодії.
- Розпізнавання голосу потребує певного твердження сказаного вашим голосом

- Розпізнавання обличчя потребує чогось радикального, такого як пластична хірургія. Він не незламний, але досить близький до цього.

Недоліки:

- Найбільший недолік і причина, чому біометрія рідко використовується як двофакторний метод, полягає в тому, що скомпрометований біометричний пристрій може ставити під загрозу саме життя.

У цілому, двофакторна аутентифікація з використанням мобільних пристроїв є потужним і ефективним інструментом для забезпечення безпеки в інформаційних системах. Цей підхід дозволяє поєднувати високий рівень захисту зі зручністю і легкістю використання для користувачів.

1.2 Тенденції розвитку систем аутентифікації.

Біометрична аутентифікація.

Одним з найважливіших досягнень у сфері управління доступом є просування у біометричній аутентифікації. Традиційні методи, такі як паролі і PIN-коди, схильні до порушень безпеки, часто через слабкі практики користувачів (наприклад, використання кількома користувачами одного логіну) або складних методів взлому (наприклад, атаки методом перебору паролів). Біометрична аутентифікація використовує унікальні біологічні характеристики, такі як відбитки пальців, обличчя або структура райдужок, для перевірки ідентичності користувача. Розвиток технологій біометрії зробив її більш надійною, безпечною та зручною, надаючи міцне рішення для управління доступом. Розпізнавання обличчя здобуло популярність у різних галузях, пропонуючи зручний та безконтактний метод аутентифікації. Оскільки більшість смартфонів покладаються на FaceID, від Apple, Samsung, і нових учасників ринку, таких як Xiaomi, Redmi та OnePlus, це підтверджує надійність біометричної аутентифікації. При подальшому вдосконаленні технології можна очікувати широкого впровадження та інтеграції біометричної аутентифікації в управління доступом.

Багатофакторна аутентифікація (MFA).

Багатофакторна аутентифікація вже деякий час є основою управління доступом, але останні тенденції вказують на перехід до більш складних і безперервних рішень MFA. Традиційна MFA зазвичай включає комбінацію чогось, що ви знаєте (пароль), що ви маєте (смарт-карту) або що ви є (біометричний показник). Проте, нові тенденції акцентують на контекстних факторах, додаючи додатковий рівень безпеки. MFA, що враховує контекст, враховує місцезнаходження користувача, тип пристрою та поведінкові особливості, щоб визначити легітимність запитів на доступ. Деякі провідні постачальники MFA, такі як Okta, Ping Identity та Duo, пропонують контекстно-орієнтовану MFA як частину додаткових прав. Аналізуючи ці контекстні фактори, системи управління доступом можуть динамічно адаптуватися та застосовувати відповідні методи аутентифікації. Ця тенденція покращує безпеку та забезпечує більш зручний досвід, балансуючи надійний захист та зручність користувача.

Аутентифікація без пароля.

З появою вразливостей, пов’язаних з традиційними паролями, зростає популярність переходу до аутентифікації без пароля. Аутентифікація без пароля усуває потребу користувачів у запам’ятовуванні складних паролів, зменшуючи ризик порушень безпеки, пов’язаних з обліковими даними. До методів аутентифікації без пароля належать біометричні дані, одноразові паролі (OTP) та аутентифікація на основі пристрою. Найбільшою новиною стала пропозиція Google щодо безпарольного доступу до G-Suite, мета якої полягає у забезпеченні безпечного та зручного досвіду користувача, зменшуючи залежність від легко компрометованих паролів. Ця тенденція відповідає стрімкому розвитку галузі у напрямку більш безшовної та безпечної аутентифікації користувачів.

Модель безпеки нульового довір’я.

Традиційний підхід до безпеки ґрунтується на периметральній моделі, вважаючи, що раз користувач отримав доступ до мережі, йому можна довіряти. Однак зростаюча складність кіберзагроз спонукала до парадигмального зміщення до моделі безпеки нульового довір’я. Основна ідея цієї моделі полягає в тому, що «ніколи не довіряйте, завжди перевіряйте.» Модель нульового довір’я підкреслює

постійну перевірку ідентичності користувача та статусу пристрою, незалежно від місцезнаходження чи мережі. Аутентифікація є ключовим компонентом цієї моделі, з управлінням доступом, яке застосовує строгі протоколи для перевірки користувачів та пристроїв під час кожної взаємодії. Цей підхід забезпечує превентивний та адаптивний стан безпеки, зменшуючи ризик несанкціонованого доступу та потенційних порушень безпеки. Хоча мережа нульового довір'я наразі пропонується підприємствами-постачальниками, такими як Cisco, Skyhigh Security, Zscaler, Cloudflare та Akamai, альтернативність використання ZTNA як альтернативи VPN виявляється важливою для моделі нульового довір'я.

Аутентифікація на основі блокчейну.

Технологія блокчейну має децентралізований та стійкий до втручань характер і застосовується не лише у сфері криптовалют. Автентифікація на основі блокчейну виступає як безпечне та прозоре рішення в управлінні доступом. Використовуючи технологію розподілених реєстрів, процеси аутентифікації можуть бути децентралізованими, що зменшує ризик однієї точки відмови, яка була загрозою для традиційних механізмів аутентифікації, таких як Kerberos, LDAP, NTLM і т. д. Аутентифікація на блокчейні забезпечує цілісність записів про ідентичність та права доступу, роблячи маніпулювання або компрометацію конфіденційної інформації складними для зловмисників. Крім того, прозорість транзакцій блокчейну покращує аудиторію, забезпечуючи організаціям чіткий запис подій доступу та потенційних подій безпеки. Хоча ця технологія ще на ранньому етапі розвитку, очікується, що її використання буде зростати з розвитком технологій.

Постійна адаптивна оцінка ризиків та довіри.

Потреба в реальному часі оцінки ризиків та адаптивних механізмів довіри стала надзвичайною в постійно змінній кібербезпеці. Постійна адаптивна оцінка ризиків та довіри (CARTA) є новою тенденцією, яка динамічно впроваджує оцінку ризиків у процес управління доступом. На відміну від статичних методів аутентифікації, CARTA оцінює ризикові фактори протягом сесії користувача, коригуючи права доступу на основі змінюючихся обставин. CARTA включає

контекстні дані, включаючи поведінку користувача, стан пристрою та потоки розвідки щодо загроз. Системи управління доступом можуть виявляти аномалії та потенційні загрози безпеки, аналізуючи ці фактори в реальному часі. Наприклад, користувач, який намагається отримати доступ до чутливої інформації з незнайомого місця або пристрою. У такому випадку CARTA може запропонувати додаткові заходи аутентифікації або тимчасово обмежити доступ, поки ризик не буде зменшено. Організації можуть покращити свою позицію в галузі безпеки та оперативно реагувати на можливі порушення, постійно оцінюючи ризики, пов’язані з взаємодією користувачів. CARTA доповнює існуючі протоколи аутентифікації, надаючи додатковий рівень захисту перед новими та динамічними викликами кібербезпеки.

1.3 Відомі вразливості методів аутентифікації та засоби їх усунення.

У зв’язку зі зростанням комплексності та обсягу даних, які обробляються в сучасних інформаційних системах, питання забезпечення безпеки і конфіденційності стають все більш актуальними. Методи аутентифікації, які використовуються для підтвердження ідентичності користувачів перед наданням доступу до ресурсів, є одними з ключових складових в цьому процесі. Однак існують вразливості, які можуть бути використані зловмисниками для обхіду або компрометації систем аутентифікації.

Однією з найпоширеніших вразливостей є парольна атака. Це коли зловмисники використовують програмні засоби для послідовного перебору всіх можливих комбінацій паролів з метою вгадати правильний пароль. Щоб запобігти цьому, рекомендується використовувати довгі паролі, які складаються з комбінації різних символів, цифр та спеціальних символів, а також використовувати інші методи аутентифікації, такі як біометричні дані або одноразові паролі.

Іншою важливою вразливістю є фішинг - це соціально-інженерний метод, за допомогою якого зловмисники намагаються отримати конфіденційну інформацію, таку як паролі чи особисті дані, шляхом видавання себе за довірену особу чи організацію. Для боротьби з цим методом важливо підвищувати

освіченість користувачів щодо типових методів атак та надавати їм інструменти для розпізнавання шахрайських спроб.

Ще однією важливою вразливістю є викрадення сесій. Зловмисники можуть перехопити або підробити ідентифікатори сесій, щоб отримати несанкціонований доступ до облікових записів. Для захисту від цього можна використовувати шифрування сесійних токенів та використання механізмів одноразових токенів або двофакторної аутентифікації.

Окрім того, існують слабкі методи аутентифікації, такі як використання простих паролів або старих алгоритмів шифрування, які можуть стати об’єктом атак з використанням сучасних обчислювальних ресурсів. Для захисту від цього важливо використовувати сучасні методи аутентифікації та постійно оновлювати та підвищувати безпеку системи.

Загалом, для забезпечення безпеки методів аутентифікації важливо поєднувати технічні заходи безпеки, такі як шифрування та захист від атак, з соціально-педагогічними заходами, такими як навчання користувачів та забезпечення свідомості про ризики безпеки. Тільки комплексний підхід може гарантувати ефективний рівень захисту від потенційних загроз.

Варто також зазначити, що на додаток до технічних вразливостей, існують соціальні атаки на аутентифікацію, такі як інсайдерські загрози або атаки з метою перехоплення ідентифікаційних даних під час їх передачі по мережі. У цих випадках важливо впроваджувати строгі правила безпеки, такі як обмеження прав доступу для працівників та шифрування даних під час їх транспортування через мережу.

Для забезпечення безпеки методів аутентифікації також важливо постійно оновлювати програмне забезпечення та застосовувати патчі безпеки для усунення виявлених вразливостей. Крім того, слід регулярно проводити аудит безпеки, який дозволяє виявляти потенційні проблеми та ризики безпеки та вчасно приймати заходи для їх вирішення.

Нарешті, не слід забувати про захист від внутрішніх загроз. Інсайдерські атаки можуть завдати серйозної шкоди системі, тому важливо ретельно

контролювати доступ до конфіденційної інформації та вживати заходів для виявлення та запобігання несанкціонованим діям власних співробітників.

1.4. Інноваційні методи розширеної аутентифікації.

Завдяки стрімкому розвитку технологій і зростаючій складності вимог до безпеки, інноваційні методи автентифікації стають надзвичайно важливими для сучасних систем контролю доступу. У цьому розділі розглянемо дослідження та аналіз методів біометричної, двофакторної та мультифакторної автентифікації, щоб з’ясувати їхню придатність, переваги та недоліки.

Дослідження методів біометричної аутентифікації. Біометрична аутентифікація використовує унікальні фізичні характеристики особи, такі як відбитки пальців, розпізнавання обличчя, голосу тощо, для ідентифікації особи. Цей підхід є одним із найбільш перспективних у сфері безпеки.

Дослідження різних типів біометричних ідентифікаторів і їх ефективності допоможе визначити найбільш надійні та зручні методи для впровадження в системи контролю доступу. Аналіз точності та надійності різних методів біометричної ідентифікації в різних умовах допоможе зрозуміти їхню придатність для різних сценаріїв застосування. Додатково, розглянемо можливості удосконалення біометричних систем шляхом використання новітніх алгоритмів обробки даних та технологій розпізнавання.

Дослідження методів двофакторної аутентифікації. Двофакторна аутентифікація використовує комбінацію двох різних методів для підтвердження особи. Це може бути комбінація пароля та одноразового коду, біометричних даних та фізичного об’єкта тощо.

Аналіз різних комбінацій факторів аутентифікації та їх впливу на безпеку та зручність використання допоможе визначити найефективніші стратегії використання двофакторної аутентифікації для різних сценаріїв застосування. Крім того, дослідимо можливості використання різних типів аутентифікаційних методів у комбінації з урахуванням їхнього взаємодії та сумісності.

Дослідження методів мультифакторної аутентифікації. Мультифакторна аутентифікація використовує комбінацію трьох або більше різних факторів для

підтвердження особи. Це може включати комбінацію пароля, біометричних даних та фізичного об’єкта.

Дослідження різних підходів до використання комбінацій факторів аутентифікації та їх впливу на безпеку та надійність допоможе визначити найбільш ефективні методи для використання в сучасних системах контролю доступу. Аналіз технологічних рішень та методологій реалізації мультифакторної аутентифікації допоможе зрозуміти їхню придатність та переваги для сучасних систем контролю доступу.

Дослідження інноваційних підходів до аутентифікації на основі штучного інтелекту (ШІ). Дослідження та аналіз інноваційних підходів до аутентифікації на основі штучного інтелекту є важливим аспектом розвитку сучасних систем контролю доступу. Використання методів машинного навчання, глибокого навчання та інших технік ШІ може допомогти підвищити безпеку та зручність процесу автентифікації.

Аналіз можливостей застосування ШІ в аутентифікаційних системах та їхній вплив на безпеку допоможе визначити перспективні напрямки розвитку. Вивчення сучасних досліджень і розробок у галузі алгоритмів машинного навчання для автентифікації допоможе зрозуміти можливості та обмеження таких підходів.

Висновки до розділу 1

Було проведено дослідження методів аутентифікації як ключового елемента систем контролю доступу. Було розглянуто різні методи аутентифікації, їхні переваги та недоліки, а також вплив цих методів на безпеку систем контролю доступу.

Також було проаналізовано сучасні тенденції в розвитку систем аутентифікації. Особливу увагу було приділено біометричним та багатофакторним методам, які вважаються перспективними в контексті забезпечення безпеки.

В рамках розділу було виявлено відомі вразливості різних методів аутентифікації та розглянуто можливі способи їх усунення. Це дозволило підкреслити важливість постійного оновлення та вдосконалення методів аутентифікації.

Нарешті, було досліджено новітні методи розширеної аутентифікації, які можуть забезпечити більшу безпеку в системах контролю доступу. Це підкреслює важливість інновацій у цій області.

Таким чином, цей розділ надає глибоке розуміння важливості аутентифікації в системах контролю доступу та підкреслює необхідність постійного оновлення та вдосконалення методів аутентифікації для забезпечення безпеки.

РОЗДІЛ 2
МОДЕЛЮВАННЯ РОЗШИРЕНИХ МЕТОДІВ АУТЕНТИФІКАЦІЇ ДЛЯ СИСТЕМ КОНТРОЛЮ ДОСТУПУ

2.1 Вплив методів розширеної автентифікації на безпеку та зручність використання систем контролю доступу.

Автентифікація є важливим елементом будь-якої системи контролю доступу. Вона дозволяє системі переконатися, що користувач, який намагається отримати доступ, дійсно є тим, за кого він себе видає. Це важливо для забезпечення безпеки системи та захисту даних користувачів.

Однак, не всі методи автентифікації створені однаково. Деякі методи можуть бути більш безпечними, але менш зручними для користувачів. Інші методи можуть бути більш зручними, але менш безпечними. Тому важливо знайти оптимальний баланс між безпекою та зручністю.

В рамках цього підрозділу розглядається вплив розширених методів автентифікації на безпеку та зручність використання систем контролю доступу. Розширені методи автентифікації - це методи, які використовують додаткові фактори аутентифікації або використовують більш складні технології для перевірки ідентичності користувача.

Одним з прикладів розширених методів автентифікації є використання фізичних об’єктів для ідентифікації користувачів. Це може бути, наприклад, система турнікетів в метро, де користувачі ідентифікуються за допомогою спеціальних карток. Цей метод є досить безпечним, оскільки фізичні об’єкти важко вкрасти або скопіювати. Водночас, він є досить зручним для користувачів, оскільки вони просто повинні мати при собі картку для отримання доступу.

Також можливе використання більш складних технологій, таких як біометрична аутентифікація. Цей метод включає використання унікальних біологічних характеристик користувача, таких як відбитки пальців або риси обличчя, для ідентифікації. Цей метод є дуже безпечним, оскільки біометричні дані важко вкрасти або скопіювати. Однак, він може бути менш зручним для

користувачів, оскільки він вимагає спеціального обладнання та може бути пов’язаний з проблемами приватності.

Одним з можливих напрямків для подальшої програмно-апаратної імплементації є використання комбінації різних технологій, таких як мікроконтролери, бездротові модулі та технології ідентифікації, що базуються на фізичних об’єктах, подібних до системи турнікетів в метро, для створення ефективних систем аутентифікації. Цей підхід відрізняється від традиційних методів аутентифікації, таких як використання паролів або біометричних даних, оскільки він включає використання фізичних об’єктів для ідентифікації користувачів, що може забезпечити додатковий рівень безпеки. Важливо зазначити, що використання розширених методів автентифікації вимагає врахування ряду факторів. Наприклад, необхідно враховувати можливість втрати або крадіжки фізичних об’єктів, які використовуються для ідентифікації. Також важливо враховувати можливість компрометації біометричних даних, які можуть бути викрадені або скопійовані. Крім того, необхідно враховувати можливість втрати доступу до веб-сервісів або баз даних, які використовуються для обробки запитів аутентифікації. Також важливо враховувати потреби користувачів. Деякі користувачі можуть віддавати перевагу зручності перед безпекою, тоді як інші можуть віддавати перевагу безпеку над зручністю. Тому важливо розробити систему, яка може бути налаштована відповідно до потреб конкретного користувача або організації.

Крім того, при розробці системи автентифікації важливо враховувати майбутні технологічні тенденції. Наприклад, з розвитком Інтернету речей (IoT), штучного інтелекту (AI) та блокчейну можуть з’явитися нові можливості для автентифікації. Наприклад, AI може аналізувати шаблони поведінки користувача, такі як час входу в систему, частота використання певних функцій та інші характеристики, щоб визначити, чи є користувач тим, за кого він себе видає. Це може допомогти виявити підозрілу поведінку та запобігти несанкціонованому доступу.

Блокчейн, з іншого боку, може бути використаний для створення безпечного та прозорого механізму ведення записів. Блокчейн є розподіленою базою даних, яка зберігає інформацію в ланцюгу блоків, які захищені криптографією. Це означає, що інформація, яка записана в блокчейн, не може бути змінена або видалена без відповідних дозволів. Це може бути особливо корисним для зберігання даних про автентифікацію, оскільки це забезпечує високий рівень безпеки та прозорості.

Використання AI та блокчейну в системах автентифікації є активним напрямком досліджень, і ці технології можуть значно покращити безпеку та ефективність цих систем в майбутньому. Однак, важливо враховувати, що використання цих технологій також вимагає врахування ряду викликів, таких як проблеми з приватністю, етикою та регулюванням. Тому важливо продовжувати дослідження та розробку в цій області, щоб забезпечити, що ці технології використовуються відповідально та ефективно.

Це підкреслює важливість постійного оновлення та вдосконалення методів аутентифікації для забезпечення безпеки в системах контролю доступу. Розширені методи автентифікації можуть значно покращити безпеку та зручність використання цих систем.

2.2 Формулювання завдань розробки

Основним завданням розробки є створення системи, яка балансує між безпекою та зручністю користувачів. Це означає, що система повинна бути достатньо надійною, щоб захистити дані користувачів, але водночас вона повинна бути зручною та простою у використанні. Також вирішальним фактором є створення легко розширюваної системи, що може використовуватись для різних цілей, без затрат на її розгортання.

Для досягнення цієї мети, одним з ключових завдань є інтеграція різних методів автентифікації в одну систему. Це може включати використання традиційних методів, таких як паролі, а також більш сучасних технологій, таких як біометрична аутентифікація та автентифікація за допомогою фізичних об’єктів.

Іншим важливим завданням є розробка програмного забезпечення та апаратної частини системи. Це включає в себе створення веб-сервісів для обробки запитів аутентифікації, розробку баз даних для зберігання інформації про користувачів та їхні права доступу, а також розробку апаратної частини системи, яка включає в себе мікроконтролери, бездротові модулі та інші компоненти. Під час розробки системи автентифікації також важливо провести детальний аналіз та опис багаторівневої архітектури системи перед початком розробки. Багаторівнева архітектура [12] дозволяє розділити систему на окремі компоненти або рівні, кожен з яких відповідає за певні функціональні аспекти. Це сприяє збереженню модульності, підвищує масштабованість та спрощує підтримку системи в майбутньому.

Опис багаторівневої архітектури перед розробкою дозволяє розробникам краще зрозуміти взаємозв’язки між компонентами системи, виявити потенційні точки з’єднання та ризики, а також ефективно розподілити робочі завдання між командами. Крім того, це допомагає уникнути проблем, пов’язаних з монолітним підходом до розробки, де всі компоненти системи об’єднуються в одному моноліті.

Після завершення розробки, важливим завданням є тестування та валідація системи. Це включає в себе перевірку надійності та безпеки системи, а також оцінку її зручності для користувачів. На цьому етапі можуть бути виявлені та виправлені різні помилки та проблеми, що допоможе забезпечити високу якість кінцевого продукту. Після завершення тестування та валідації, система готова до розгортання та моніторингу. На цьому етапі важливо забезпечити стабільну роботу системи, виявляти та вирішувати будь-які проблеми, що можуть виникнути, а також регулярно оновлювати систему для відповідності найновішим стандартам безпеки та зручності.

Додатково, важливо також врахувати питання сумісності з різними платформами та пристроями, оскільки користувачі можуть використовувати різні пристрої для доступу до системи. Також слід розглянути питання масштабованості системи, оскільки з часом кількість користувачів може зростати,

і система повинна бути готова до такого росту без втрати продуктивності та безпеки.

Крім того, важливо враховувати питання забезпечення конфіденційності даних користувачів під час їх передачі та зберігання в системі. Це може включати розробку механізмів шифрування для захисту конфіденційної інформації під час передачі через мережу та зберігання в базі даних. Також важливо регулярно оновлювати та аудитувати заходи безпеки, щоб виявляти та усувати потенційні вразливості.

Нарешті, важливо не забувати про постійне вдосконалення та підтримку системи після її розгортання. Регулярні оновлення, тестування та моніторинг дозволяють підтримувати стабільну роботу системи та вчасно реагувати на будь-які потенційні проблеми.

2.3 Проектування моделі та алгоритмів

У даному розділі буде розглянуто алгоритм вирішення задачі автентифікації користувача в системі контролю доступу, а також проведено аналіз спроектованої моделі, визначено її сильні сторони та вразливості.

Алгоритм вирішення задачі:

- Отримання ідентифікатора користувача: Система отримує від користувача ідентифікатор, який може бути представлений, наприклад, у вигляді логіна та пароля, біометричних даних або унікального токену.
- Перевірка ідентифікатора: Отриманий ідентифікатор перевіряється системою на відповідність збереженим даним в базі даних або іншому засобі збереження інформації. Якщо ідентифікатор вірний, користувачу надається доступ до системи, в іншому випадку відбувається відмова в доступі.
- Надання доступу: У разі успішної перевірки ідентифікатора, користувачу надається доступ до ресурсів або функцій системи, на які він має право.
- Реєстрація подій: Система реєструє кожну спробу входу користувача, що дозволяє проводити аналіз доступу та виявляти можливі загрози безпеки.

Сильні сторони:

- Різноманітність методів: Використання різних методів автентифікації дозволяє вибрати оптимальний захист в залежності від конкретних потреб і вимог безпеки.
- Гнучкість: Модель може бути адаптована під різні сценарії використання та вимоги безпеки, що забезпечує її універсальність.
- Можливість комбінації методів: Використання двофакторної аутентифікації або комбінації різних методів забезпечує більший рівень безпеки.

Вразливості:

- Атаки перебору: Використання логін-парольної аутентифікації може бути вразливим до атак перебору паролів.
- Підробка біометричних даних: Біометричні дані можуть бути підроблені або обмануті, що може вплинути на їхню ефективність.
- Втрата або крадіжка карт доступу: Карти доступу можуть бути втрачені або скопійовані, що може призвести до несанкціонованого доступу до системи.

У системі фігурують дві основні ролі: адміністратор і користувач (наприклад, працівник підприємства). Адміністратор має повний доступ до адміністративних функцій системи, в той час як користувач обмежений в своїх можливостях відповідно до встановлених прав доступу.

Розрізнення користувачів за їхніми характеристиками, такими як місце роботи чи посада, відіграє ключову роль у системі контролю доступу. Це дозволяє ефективно керувати правами доступу та забезпечує безпеку інформації, що зберігається та обробляється системою.

Враховуючи це, всередині системи буде необхідно використовувати механізми для розрізнення користувачів за їхніми ролями та характеристиками. Наприклад, адміністратор системи може мати розширені права доступу порівняно зі звичайним користувачем. Роль адміністратора полягатиме у керуванні користувачами, налаштуванні прав доступу та моніторингу системи. З іншого боку, звичайні користувачі можуть мати обмежений доступ лише до певних ресурсів або функцій системи.Розрізнення користувачів також може включати визначення прав доступу в залежності від їхньої посади або місця роботи.

Наприклад, співробітник виробництва може мати доступ лише до ресурсів, необхідних для виконання його робочих обов’язків, в той час як керівник відділу може мати розширені права доступу для управління персоналом та аналізу даних.

Враховуючи необхідність легкої інтеграції та розширення системи, можна ввести різні сутності, що допоможуть у зручному та ефективному обліку робочого часу персоналу. Ось детальний опис цих сутностей:

- Посади: Ця сутність визначає різні посади або ролі, які можуть мати працівники в організації. Вона містить інформацію про назву посади, опис обов’язків та права доступу до різних ресурсів системи.
- Відпустки: Ця сутність дозволяє вести облік відпусток працівників. Вона містить інформацію про дату початку та завершення відпустки, тип відпустки (оплачена, неоплачена тощо) та інші додаткові дані.
- Відділення підприємства чи навчального закладу: Ця сутність описує різні відділення або підрозділи організації. Вона містить інформацію про назву відділення, його місцезнаходження, контактну інформацію та інші характеристики.
- Графік роботи працівника: Ця сутність визначає графік роботи кожного працівника в організації. Вона містить інформацію про робочі години, дні відпочинку, робочі зміни тощо.
- Логи автентифікації: Ця сутність відображає історію спроб входу користувачів до системи. Вона зберігає інформацію про кожну спробу автентифікації, включаючи ідентифікатор користувача, дату та час спроби, результат (успішна чи неуспішна), а також можливі додаткові дані, наприклад, IP-адресу чи тип використаного пристрою. Крім того, можуть бути збережені додаткові параметри, такі як код статусу, що вказує на причину неуспішної спроби, або інші відомості, корисні для аналізу та виявлення потенційних загроз безпеці.
- Картки доступу: Ця сутність існує для впровадження системи контролю доступу. Вона містить інформацію про працівника і відділення в якому він працює.

Введення цих сутностей дозволить системі бути більш гнучкою та адаптивною до потреб користувачів, а також спростить процеси обліку робочого часу та управління персоналом. Крім того, це забезпечить можливість легкої інтеграції нових функцій та розширення функціональності системи в майбутньому. Запровадження в систему вищезазначених сутностей дозволить значно поліпшити управління робочим часом персоналу та забезпечити більш гнучке налаштування прав доступу. Крім того, введення різноманітних методів автентифікації дозволить забезпечити надійний контроль доступу до системи, підвищуючи загальний рівень безпеки інформації.

Спробуємо формалізувати описані вище сутності. Для цього використовують діаграму сутностей – модель даних, за допомогою якої концептуально описують схему предметної області. Створивши на початковому етапі цю діаграму в подальшому її використовують при проектуванні бази даних, та діаграми класів, оскільки на ній зображені основні сутності та відношення між ними.

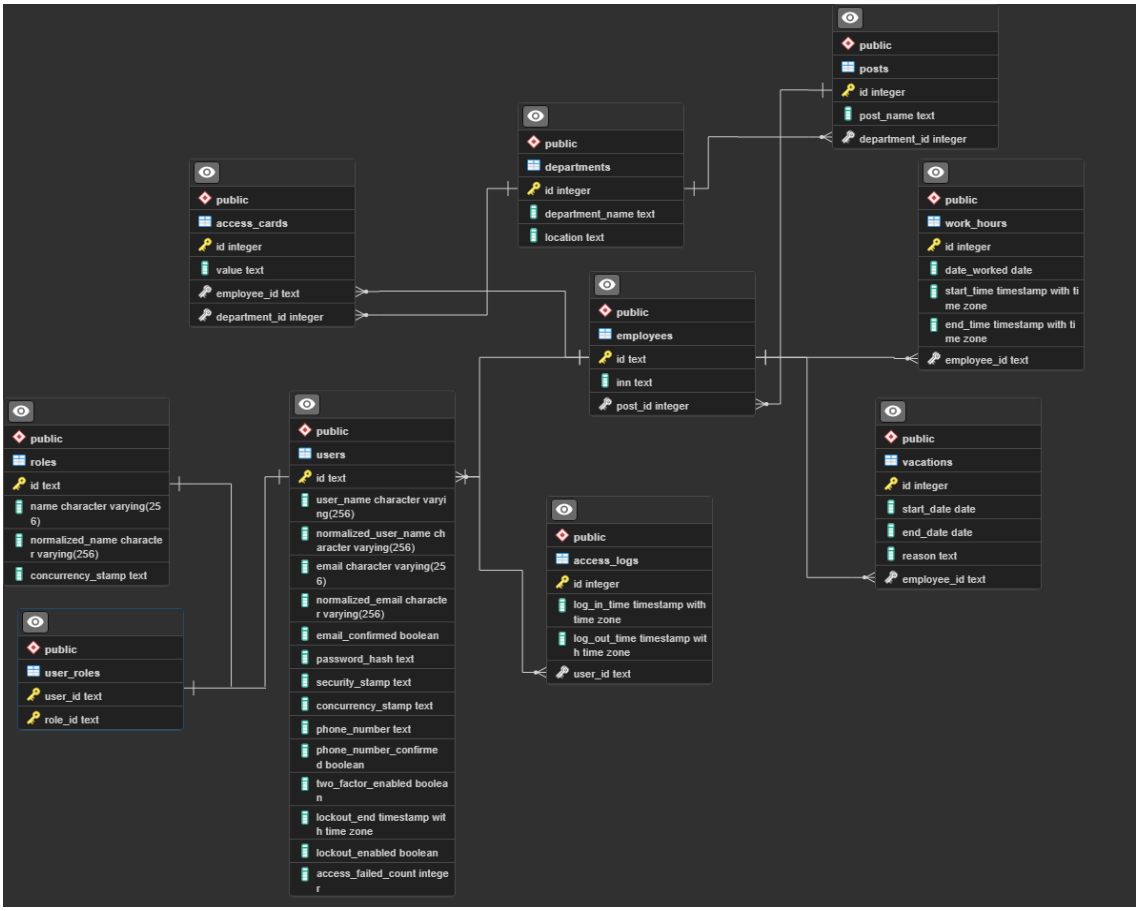


Рисунок 2.1 – Діаграма сутностей

В результаті аналізу предметної області можна вивести діаграму сутностей, на якій будуть відображені усі необхідні сутності та зв’язки між ними. Діаграма зображена на рисунку 2.1. Для наочності вона зображена в дещо спрощеному вигляді. Деталі які ускладнювали схему, але не мали важливого впливу на розуміння діаграми прибрані.

Висновки до розділу 2

У цьому розділі було досліджено та описано реальні приклади реалізації розширених методів автентифікації. З досліджень випливає, що ефективне використання розширених методів автентифікації може значно підвищити рівень безпеки систем контролю доступу, зберігаючи при цьому високий рівень зручності для користувачів. Проаналізовано які вразливості наявні, та які переваги можна отримати при побудові гнучкої системи, яку можна легко розширювати.

Після чого, враховуючи всі особливості, було спроектовано модель предметної області. Були сформовані необхідні вимоги до системи контролю доступу.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1 Вибір та обґрунтування засобів реалізації системи

При розробці програмного забезпечення великий вплив мають правильно підібрані під задачу засоби розробки. В загальному випадку задачу можна виконати майже будь якими засобами, але якщо інструмент створювали для вирішення відмінних від поставленої задачі задач, то багато часу буде витрачено дарма.

Також важливо підібрати актуальні інструменти, які мають довготривалу підтримку та велике ком'юніті. Це полегшить написання системи, та її підтримку у майбутньому, адже буде багато спеціалістів знайомих з вибраним засобом розробки.

3.1.1 Обґрунтування вибору СКБД PostgreSQL

SQL (Structured Query Language) - це специфічна мова програмування за допомогою якої можна створювати, змінювати, та керувати даними в реляційних базах даних (мова структурованих запитів).

SQL в його вихідному вигляді є інформаційно-логічною мовою, а не мовою програмування, але разом SQL передбачає можливість його процедурних розширень, з урахуванням яких мова вже цілком може розглядатися в якості мови програмування [17].

У наш час існує багато видів систем баз даних. За моделлю організації даних їх поділяють на наступні:

- Ієрархічна. Зазвичай представлена у вигляді дерева.
- Мережева. Схожа на ієрархічну, але кожний вузол дерева може мати більше ніж одного предка.
- Реляційна. Дані зберігаються у вигляді відношень, які по своїй сутті схожі на таблиці.
- Об'єктно-орієнтована. Зберігає дані у вигляді об'єктів певного класу.

Ієрархічні та мережеві бази даних підходять для специфічних випадків, коли необхідно зберігати деревовидну структуру даних, проте в нашому випадку вони б не принесли вигоди, а лише ускладнили б вибірку та збереження даних.

Об’єктно-орієнтована база даних може бути зручною у випадку використання об’єктно-орієнтованої мови програмування, проте вона може бути менш ефективною та займати більше місця на сховищі даних через неоптимальне зберігання.

Реляційна база даних [8 с.68], найпопулярніше рішення, є простою в розумінні, надійною та підходить для широкого спектру задач. Вона відповідає вимогам ACID (Atomicity - Атомарність, Consistency - Узгодженість, Isolation - Ізольованість, Durability - Довговічність), що забезпечує надійну роботу транзакцій та цілісність даних. До переваг реляційних баз даних також відноситься добра структурованість даних, що зменшує кількість помилок при використанні системи.

У нашому випадку, де потрібна надійність даних, було вирішено обрати реляційну базу даних. Серед реляційних СКБД найпопулярнішими є MySQL та PostgreSQL [4]. Вибір PostgreSQL обумовлений його більшою надійністю, швидкістю та потужністю. Ця СКБД має краще організоване індексування даних, що дозволяє здійснювати ефективний доступ до інформації, а також більше типів для зберігання даних, що робить її більш гнучкою та відповідальною за різноманітні завдання. Крім того, PostgreSQL відома своєю явною та передбачуваною поведінкою у більшості ситуацій порівняно з MySQL.

Однією з ключових переваг PostgreSQL є його розширені можливості для роботи з JSON-даними. JSON (JavaScript Object Notation) є популярним форматом обміну даними у сучасних веб-додатках та API. Можливість зберігати та опрацьовувати JSON-дані безпосередньо у базі даних дозволяє нам ефективно взаємодіяти з іншими компонентами системи, такими як веб-сервери та клієнтські додатки. Крім того, PostgreSQL має активну спільноту розробників, яка постійно вдосконалює та розвиває цю систему. Це забезпечує нам доступ до нових функцій, поліпшень та заходів безпеки.

Таким чином, вибір PostgreSQL як бази даних для нашого проекту обумовлений його перевагами у надійності, швидкості та гнучкості, що робить його ідеальним вибором для забезпечення ефективної та надійної роботи нашої системи.

3.1.2 Обґрунтування вибору ASP.NET Core MVC

Платформа ASP.NET обирається через свою широку можливість інтеграції з різноманітними технологіями та мовами програмування. Це дозволяє легко впроваджувати систему контролю доступу з іншими додатками та сервісами, що може бути важливо для взаємодії з різними пристроями та іншими системами безпеки.

Крім того, ASP.NET відомий своєю високою продуктивністю та швидкодією. Це дозволяє системі контролю доступу ефективно обробляти великі обсяги даних та надавати миттєвий доступ до інформації про доступ користувачів.

Ще однією перевагою вибору ASP.NET є широкий спектр інструментів розробки, які надаються цією платформою. Різноманітні бібліотеки та фреймворки спрощують процес розробки та забезпечують високу якість коду, що є важливим аспектом для створення надійної та функціональної системи контролю доступу.

Також слід зазначити, що ASP.NET [1] є популярним вибором у корпоративному середовищі, завдяки чому він має велику кількість досвідчених розробників та широку спільноту підтримки. Це робить його привабливим вибором для корпоративних проєктів, включаючи системи контролю доступу.

Щодо аспекту безпеки, ASP.NET надає потужні засоби для забезпечення безпеки додатків. Зокрема, він має вбудовані можливості для автентифікації, авторизації та захисту від атак. У системах контролю доступу безпека є критично важливою, тому вибір ASP.NET може допомогти забезпечити високий рівень захисту даних та доступу. По-перше, ASP.NET надає ряд вбудованих засобів для автентифікації та авторизації, таких як ASP.NET Identity та рольова авторизація, які забезпечують потужні механізми безпеки та контролю доступу. Зокрема, використання ASP.NET Identity дозволяє налаштовувати різні методи

аутентифікації, включаючи стандартний логін та пароль, а також зовнішні постачальники, такі як Azure Active Directory. Це дозволяє легко інтегрувати систему контролю доступу з іншими сервісами та додатками. Крім того, ASP.NET має вбудовані можливості для створення API за допомогою ASP.NET Web API, що дозволяє взаємодіяти з іншими додатками та сервісами. Це може бути корисно для інтеграції з іншими системами безпеки або моніторингу доступу. Окрім цього, можна використовувати різні рішення для управління ідентичністю та доступом, такі як Azure Active Directory або Microsoft Entra, для додаткового забезпечення безпеки та управління доступом до ресурсів.

За допомогою цих технологій та сервісів, ви можете побудувати потужну, безпечну та гнучку систему контролю доступу, яка відповідає вашим потребам і може легко інтегруватися з іншими додатками та сервісами.

ASP.NET Core MVC – потужний фреймворком для розробки веб-додатків на мові програмування C#. Побудований на основі архітектурного шаблону MVC (Model-View-Controller), ASP.NET Core MVC надає структурований підхід до розробки веб-додатків, що сприяє чіткому розділенню логіки програми та її представлення.

MVC - це архітектурний шаблон, що використовується для розробки програмного забезпечення, де логіка програми розділена на три взаємодіючі компоненти:

- Модель (Model): Представляє собою об’єктно-орієнтований клас, що містить дані додатку та логіку їх обробки. Модель відповідає за доступ до даних та їх оновлення.
- Представлення (View): Представляє собою інтерфейс користувача, що відображає дані з моделі та забезпечує можливість взаємодії з користувачем. Представлення відповідає за візуальне представлення даних та взаємодію з користувачем.
- Контролер (Controller): Координує взаємодію між моделлю та представленням. Контролер приймає запити від користувача, виконує відповідні дії з моделлю та вибирає відповідне представлення для відображення результатів.

За допомогою цього підходу MVC забезпечує легку підтримку, розширення та тестування веб-додатків, роблячи код більш структурованим та легко зрозумілим. Переваги використання ASP.NET Core MVC для створення програмного інтерфейсу адміністратора:

- Модель MVC для структурованості коду: Використання шаблону проектування MVC дозволяє розділити логіку додатку на логічні компоненти: модель (дані), представлення (відображення) та контролер (логіка обробки запитів). Це спрощує розробку, підтримку і розуміння коду, особливо у великих проектах.
- Розширюваність та модульність: MVC дозволяє легко розширювати та модифікувати функціональність за допомогою розділення додатку на невеликі модулі. Це особливо корисно для адміністративних інтерфейсів, де можуть знадобитися різні компоненти для управління різними аспектами системи.
- Вбудована підтримка маршрутизації: ASP.NET Core MVC має вбудований механізм маршрутизації, який дозволяє визначати, які URL-адреси мають бути пов’язані з якими діями контролера. Це дозволяє легко налаштувати адреси URL для кожного функціонального елемента в адміністративному інтерфейсі.
- Безпека та аутентифікація: ASP.NET Core має вбудовану підтримку системи аутентифікації та авторизації, що дозволяє забезпечити захист доступу до адміністративного інтерфейсу та його функціональних можливостей.
- Підтримка стандартів індустрії: ASP.NET Core є сучасним та широко використовується фреймворком, який підтримується Microsoft і має активну спільноту розробників. Це забезпечує вас підтримкою, оновленнями та документацією, що дозволяє підтримувати ваш адміністративний інтерфейс в актуальному стані згідно з найновішими технологіями та стандартами.

У порівнянні з іншими можливими технологіями для реалізації адміністративного інтерфейсу, такими як Node.js, Django або Ruby on Rails, ASP.NET Core MVC має кілька ключових переваг.

ASP.NET Core MVC відзначається великою продуктивністю завдяки інтегрованому середовищу розробки (IDE) Visual Studio та великій кількості сторонніх бібліотек та компонентів, які сприяють швидкій розробці. У порівнянні з Node.js, який базується на JavaScript, ASP.NET Core MVC може забезпечити більш структурований та прогностичний підхід до розробки завдяки використанню мови програмування C# та шаблону проектування MVC.

Однією з переваг ASP.NET Core MVC є його вбудована підтримка безпеки та аутентифікації. Порівняно з Django або Ruby on Rails, які також мають підтримку безпеки, ASP.NET Core забезпечує більш гнучкі та налаштовані можливості для реалізації різних сценаріїв аутентифікації та авторизації.

Крім того, ASP.NET Core MVC має вбудовану підтримку маршрутизації, яка дозволяє легко налаштувати адреси URL для різних дій контролерів. У порівнянні з Ruby on Rails, де маршрутизація зазвичай відбувається за допомогою файлу конфігурації, ASP.NET Core надає більш прозорий та декларативний підхід до визначення маршрутів.

Щодо масштабованості, ASP.NET Core MVC демонструє хорошу продуктивність та масштабованість у великих проектах завдяки своїй асинхронній природі та вбудованій підтримці асинхронного програмування. У порівнянні з Node.js, який також використовує асинхронний підхід, ASP.NET Core може бути більш стабільним у великих завантажених системах завдяки своєму менеджеру пам’яті та оптимізації під високі навантаження.

У порівнянні з ASP.NET Web Forms, який також розроблений компанією Microsoft, ASP.NET Core MVC відзначається суттєвою модернізацією та покращеною продуктивністю. ASP.NET Web Forms є застарілим технологічним рішенням, яке використовувалося в основному для створення веб-додатків на початку 2000-х років. Однією з ключових відмінностей між ними є архітектура. ASP.NET Web Forms використовує подійно-орієнтовану модель програмування, що часто може призводити до змішання бізнес-логіки та представлення. З іншого боку, ASP.NET Core MVC ґрунтується на шаблоні проектування MVC, що сприяє розділенню логіки за допомогою моделі (дані), представлення (відображення) та

контролера (логіка обробки запитів), що робить код більш чистим, структурованим і легко зрозумілим. Також, ASP.NET Core MVC надає більшу гнучкість та швидкість у порівнянні з ASP.NET Web Forms. Оскільки ASP.NET Core може працювати на різних операційних системах і має підтримку Docker, розгортання та масштабування додатків стає набагато простішим та ефективнішим. Крім того, ASP.NET Core забезпечує кращу продуктивність завдяки оптимізації пам’яті, асинхронному програмуванню та більш швидким шаблонам.

Також ключевим фактором при виборі MVC слугує рішення Asp.Net Core Identity, що надає інструменти для розширення звичайного функціоналу авторизації.

3.1.3 Обґрунтування вибору мікроконтролера Arduino

Arduino - це відкрите апаратне та програмне забезпечення для створення електронних пристроїв. Ця платформа забезпечує можливість швидко та ефективно створювати різноманітні електронні пристрої, починаючи від простих світлодіодних маячків до складних систем автоматизації.

Основна платформа Arduino [10 с.31] має мікроконтролер, який може керувати різними компонентами, такими як світлодіоди, датчики, реле та інші пристрої. Вона пропонує широкий спектр різноманітних моделей та конфігурацій, що дозволяє розробникам вибрати оптимальний варіант для своїх потреб.

Arduino IDE - це середовище розробки для програмування платформи Arduino за допомогою мови програмування C/C++. Це інтуїтивно зрозуміла та легка у використанні інтегрована середа, яка надає розробникам доступ до багатьох корисних функцій та бібліотек для роботи з Arduino.

Безпосередньо для ідентифікації користувача вибір впав на RFID-мітки [19], що використовуються в інтернеті речей (IoT) для безконтактного ідентифікування та взаємодії з різними пристроями та системами. Вони дозволяють вбудовувати ідентифікаційні можливості у різні об’єкти та обладнання, що робить їх ідеальними для застосування в IoT-системах.

RFID (Radio-Frequency Identification) - це технологія, яка дозволяє безконтактно зчитувати дані, збережені на спеціальних мітках або картках, за допомогою радіочастотного сигналу. Ці мітки можуть бути вбудовані у різні предмети або використовуватися як окремі ідентифікаційні пристрої.

У IoT-системах RFID-мітки можуть використовуватися для:

- Ідентифікації та відстеження об’єктів: RFID-мітки можуть бути прикріплені до різних предметів, щоб відстежувати їх розташування та стан.
- Автоматичного управління запасами: Вони можуть бути використані для автоматичного відстеження запасів та управління інвентарем в реальному часі.
- Контролю доступу: RFID-мітки можуть служити як ідентифікатори для доступу до приміщень, об’єктів або послуг.
- Управління та моніторинг систем: Вони можуть бути використані для ідентифікації та взаємодії з різними пристроями та системами у складі IoT.

Для обміну даних між мікроконтролером та API очевидним вибором буде ESP-01. Це невеликий інтернет-модуль, який базується на мікроконтролері ESP8266 і забезпечує можливість підключення до бездротової мережі Wi-Fi. Цей модуль є популярним в середовищі розробки IoT (інтернету речей) через свою низьку вартість, компактність та високий рівень функціональності.

Основні характеристики ESP-01:

- Wi-Fi підключення: ESP-01 дозволяє пристроям підключатися до бездротової мережі Wi-Fi, що робить його ідеальним для взаємодії з Інтернетом та іншими пристроями у IoT-системах.
- Низька вартість: ESP-01 - один з найдешевших модулів з Wi-Fi на ринку, що робить його доступним для широкого кола розробників та любителів.
- Компактний розмір: Модуль ESP-01 має дуже компактні розміри, що дозволяє легко вбудовувати його у різні пристрої та системи, не займаючи багато місця.

- Низьке споживання енергії: ESP-01 відомий своєю ефективністю щодо споживання енергії, що дозволяє використовувати його в батарейних пристроях та пристроях з обмеженим живленням.

ESP-01 часто використовується для забезпечення зв’язку між різними пристроями та системами в IoT-системах, а також для реалізації віддаленого керування та моніторингу через Інтернет. Його простота використання та низька вартість роблять його популярним вибором для розробки широкого спектру IoT-пристроїв та застосувань.

3.2 Архітектура системи

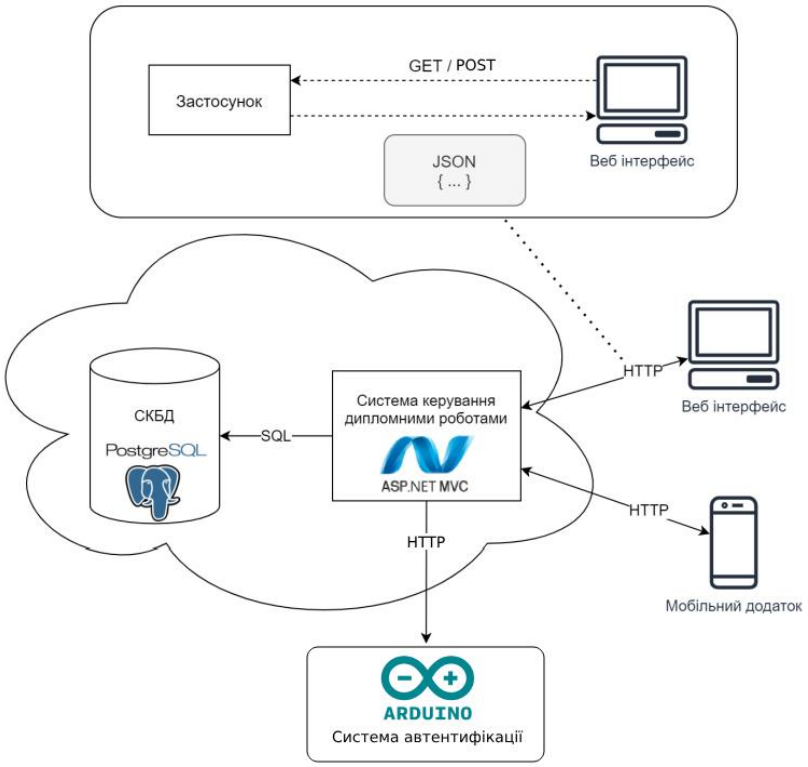


Рисунок 3.1 – Схема високорівневої архітектури

Високорівнева модель архітектури представлена на рисунку 3.1. Центральним елементом є програмний компонент системи керування дипломними тами, реалізований на основі мови програмування C#, що виконується на платформі .NET Framework.

Для збереження та обробки даних система використовує СКБД PostgreSQL і взаємодіє з нею за допомогою мови запитів SQL. Ця мова дозволяє керувати даними: зберігати, оновлювати, вибирати за різними критеріями та видаляти. Клієнтські додатки взаємодіють з системою через програмний інтерфейс, відправляючи HTTP-запити на сервер.

Для створення програмного інтерфейсу, що працює за HTTP-протоколом, ми використали REST підхід. Він передбачає передачу стану представлення. REST - це архітектурний стиль для створення мережевих програмних інтерфейсів, розроблений Роєм Філдінгом у 2000 році. Він поєднує принципи функціонування Інтернету з можливостями протоколу HTTP.

Як і будь який архітектурний підхід, REST, хоч і не диктує способи реалізації, але накладає певні архітектурні обмеження зокрема, відділення відповідальності між сервером та клієнтом, відсутність стану клієнта на сервері, підтримка кешування та ідемпотентність методів HTTP. Це спрощує взаємодію між клієнтом та сервером (рисунок 3.2), полегшує налагодження програми та забезпечує масштабованість. Однак, може знизити продуктивність через збільшення об’єму передаваних даних.

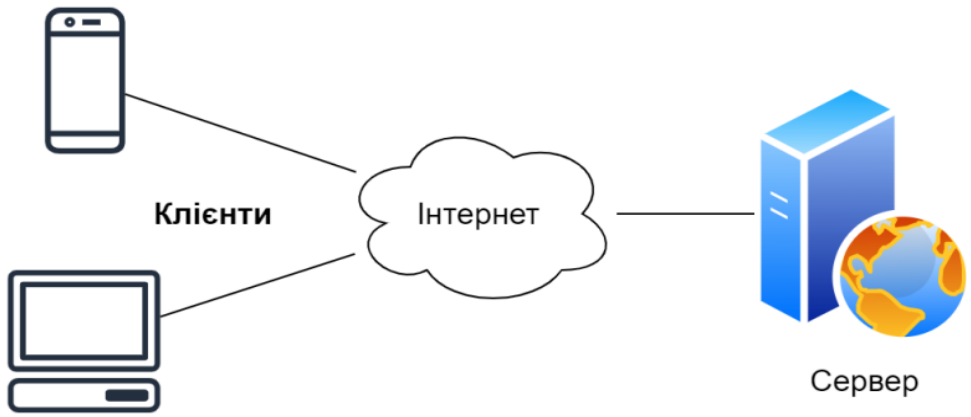


Рисунок 3.2 – Клієнт-серверна архітектура

REST підхід передбачає роботу з ресурсами, які можуть бути будь-чим: зображеннями, документами, інформацією про користувачів або динамічними значеннями цін акцій на біржі. Взаємодія клієнта з сервером базується на

можливості клієнта запросити або створити ресурс на сервері. Це підтримується ідентифікаторами ресурсів, які використовуються для доступу до них.

У REST-інтерфейсі на основі HTTP протоколу, ресурси ідентифікуються за допомогою URL (Uniform Resource Locator), що містять протокол, доменне ім’я або IP-адресу разом з портом та шляхом до ресурсу. Над ресурсом можуть виконуватись п’ять основних дій, відомих як методи HTTP:

- GET: отримати представлення ресурсу;
- PUT: повністю замінити інформацію про ресурс;
- PATCH: частково змінити інформацію про ресурс;
- POST: створити новий ресурс;
- DELETE: видалити (знищити) ресурс.

Методи GET, PUT і DELETE повинні бути ідемпотентними, тобто, незалежно від кількості запитів, результат для одного і того ж ресурсу завжди буде однаковим.

POST метод у загальному випадку не є ідемпотентним, що означає, що якщо надіслати два однакових запити, ресурс буде створений двічі. Однак у деяких випадках метод POST може бути зроблений ідемпотентним, додавши до кожного запиту спеціальний рядок, який генерується клієнтом - ключ ідемпотентності. Сервер перевірить, чи існує ресурс з таким ключем ідемпотентності, і тільки після цього створить його. В іншому випадку сервер поверне інформацію про вже існуючий ресурс, не створюючи його повторно. Хоча цей підхід забезпечує зручність, його уникатимо, оскільки він ускладнює розробку системи. Проте можемо розглянути використання його у майбутньому для покращення стабільності роботи системи.

Ідемпотентність відіграє важливу роль в мережах, де запит може загубитись і не дістатись до отримувача. У такому випадку клієнт не знає, чи виконався запит, і, ймовірно, відправить його ще раз.

Використання HTTP у системі забезпечує роботу мережевого інтерфейсу, що реалізує REST підхід.

В свою чергу прикладний інтерфейс може бути розроблено за допомогою підходу MVC (Model-View-Controller), що є популярним архітектурним шаблоном для розробки веб-додатків. У цьому шаблоні модель (Model) представляє дані та бізнес-логіку, представлення (View) відповідає за відображення інтерфейсу користувача, а контролер (Controller) обробляє запити користувача і взаємодіє як з моделлю, так і з представленням.

Використання підходу MVC дозволяє зробити код більш організованим і легким для розуміння, а також полегшує тестування та підтримку додатку. Крім того, цей підхід дозволяє розділити логіку програми на окремі компоненти, що сприяє покращенню повторного використання коду та забезпечує більшу гнучкість у внесенні змін.

За допомогою MVC прикладний інтерфейс може бути легко розширюваним та модифікованим без значних змін у інших частинах додатку. Кожен компонент MVC виконує свою функцію і може бути модифікований незалежно від інших, що робить процес розробки більш прозорим та ефективним.

3.3 Розробка системи для автентифікації користувача з використанням Arduino та RFID-карток

Базуючись на дослідженнях проведених у попередніх розділах можемо прийти до висновку, що система повинна володіти наступними властивостями:

- Забезпечення зручності для користувача при роботі
- Взаємодія з БД, для збереження даних
- Контроль доступу після ідентифікації користувача

Розробка апаратної частини буде відбуватися за допомогою плати Arduino, WiFi-модуля на базі мікросхеми ESP8266 та RFID-модуля на базі мікросхеми MFRC522. Розробка буде відбуватись у середовищі Arduino IDE, з використанням бібліотек MFRC522 [5], ArduinoJson [6] та ESP8266.

Основний функціонал необхідний для нашої задачі, це зчитування інформації з RFID-картки, після цього обробка і передача даних через компонент

ESP-01 за допомогою запиту на API [18], наступним кроком буде отримання і обробка відповіді з серверу та візуалізація для користувача (рисунок 3.3).

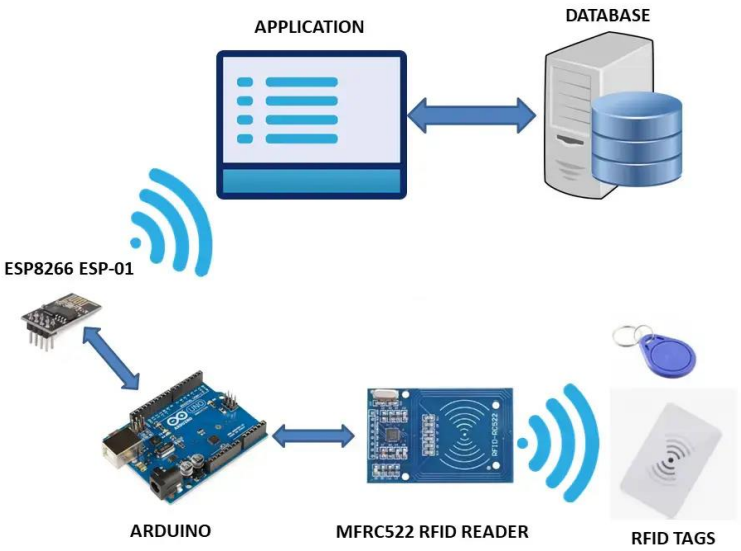


Рисунок 3.3 – Схема системи безпеки RFID на базі Arduino.

Розпочати необхідно із тестування працездатності RFID модуля та його карток, для цього потрібно встановити бібліотеку для роботи з RFID та скласти невелику схему (рисунок 3.4). Також важливо відмітити, що живлення повинне становити 3.3V.

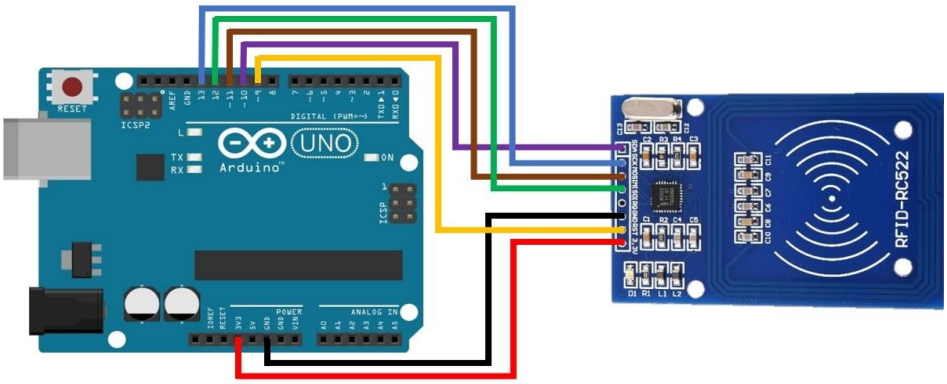


Рисунок 3.4 – Схема підключення пінів для тестування роботи RFID

Після того, як схема готова, потрібно перейти в розділ прикладів Arduino IDE, а саме MFRC522, далі обрати приклад DumpInfo і завантажити код. Далі потрібно відкрити монітор порту, якщо бачимо інформацію в консолі, схожу на рисунок 3.5 і при контакті з RFID-картками ми спостерігаємо вивід інформації

про картку – значить модуль робочий і можна переходити до наступного кроку розробки.

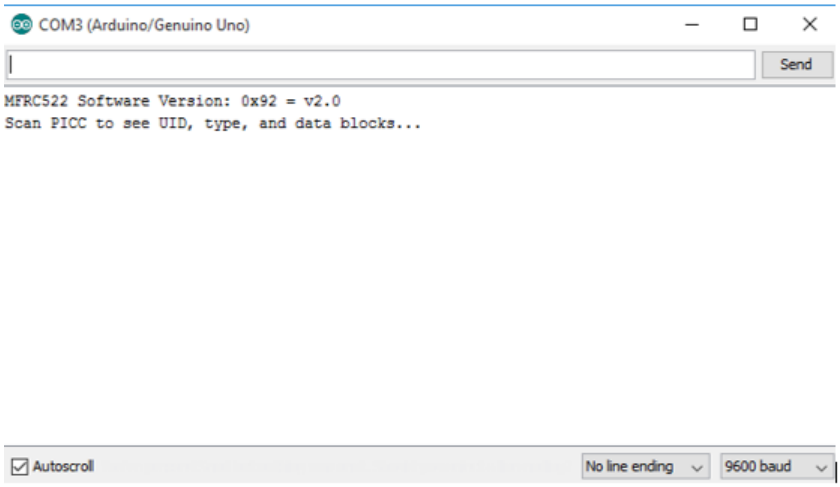
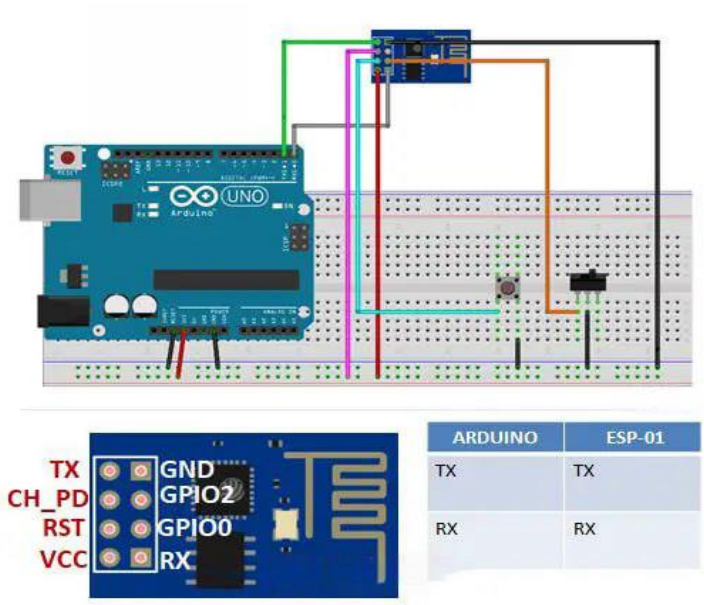


Рисунок 3.5 – Результат роботи бібліотеки MFRC522

Наступний крок – це програмування модуля ESP-01, незважаючи на те, що модуль при покупці він постачається з попередньо встановленою власною «прошивкою» і може виконувати команди TCP/IP через те, що ми називаємо AT-командами. Однак, команди трохи громіздкі для виконання, тому потрібно розробити власну «прошивку».

Поширений варіант перепрограмування модуля – це використання USB програматора для ESP8266, проте можна обійтися і без його покупки, для цього потрібно скласти схему, як на рисунку 3.6.



ARDUINO	ESP-01
TX	TX
RX	RX

ARDUINO	BREADBOARD
3.3 V	Breadboard Power Rail
GND	Breadboard GND Rail
RESET	Breadboard GND Rail

ESP-01	BREADBOARD
CH_PD	Breadboard Power Rail
RST	To Switch then GND
VCC	Breadboard Power Rail
GND	Breadboard GND Rail
GPIO0	To Switch then GND
GPIO2	<Not Connected>

Рисунок 3.6 – Схема підключення пінів для «прошивки» ESP-01

ESP-01 має два режими роботи:

- Звичайний режим
- Режим завантажувача

Для того, щоб завантажити нову прошивку, ми повинні перевести його в режим завантажувача. Для цього CH_PD повинен бути встановлений в HIGH, а GPIO0 підключений до GND. Саме тому у нас є SPST-перемикач між GPIO0 і GND. Те ж саме застосовується і тоді, коли ми хочемо створити нашу програму і завантажити її у флеш-пам’ять ESP-01.

Окремо необхідно підкреслити, що модулі ESP обмінюються даними з монітором порту зі швидкістю передачі даних, починаючи з 9600 біт/сек. В результаті тестів, при завантаженні власної програми на модуль, рекомендовано обирати швидкість завантаження – 115200 біт/сек, розмір флеш-пам’яті – 512 КБ.

Після переведення модуля в режим завантажувача та успішного завантаження власного коду потрібно перевести модуль в звичайний, робочий режим, якщо все виконано вірно – можна спостерігати індикації про пошук мережі вказаної в програмі, та подальше підключення до мережі.

При потребі switch-кнопку можна замінити на звичайну, але в такому випадку після завантаження власної програми у режимі завантажувача потрібно переконатися, що GPI0 порт модуля, не з'єднаний із заземленням.

Перед завантаженням програми з лістингу 1 потрібно переконатися, що в рядках 6 – 7 вказані ssid та пароль відповідно до налаштувань Wi-Fi.

Також потрібно звернути увагу на рядки 10 – 16, в них реалізований «макрос», що дозволить швидко змінити режим відладки для виводу додаткових логів у моніторі порту.

У методі *setup* (рядки 18 – 36) описано підключення до безпровідної мережі та налаштування монітору портів для комунікації з платою Arduino.

Спостерігаючи код у рядках 43 – 77 можемо побачити функціонал завдяки якому модуль ESP8266 ESP-01 зв'язується з Arduino через SoftwareSerial, тому ми аналізуємо дані, що надходять, і витягуємо RFID-код, отриманий з зчитувача MRFC522.

У методі *isUserAuthorized* (рядки 81 – 123) розроблений функціонал, який буде викликати наше REST API, завдяки HTTP запиту. Разом з цим необхідно переконатись, що IP-адреса в рядку 91 відповідає хосту API.

Переконайтеся, що ви змінили IP-адресу в рядку 10 на IP-адресу, на якій ви створили додаток.

Метод *urlencode* (рядки 125 – 158) відповідає за кодування url-параметрів, а саме, щоб при виклику кінцевої точки на API параметр rf_id_code мав шаблонний вигляд.

Останній, але не менш важливий метод *sendDataBackToArduino* (рядки 160 – 171) відповідає за повернення відповіді у форматі JSON назад на плату Arduino. В реалізації використовується бібліотека ArduinoJson.

Після успішного завантаження власної програми в модуль ESP-01 переходимо до наступного кроку – збір схеми на платі Arduino та програмування компонентів для роботи з RFID, як на рисунку 3.7.

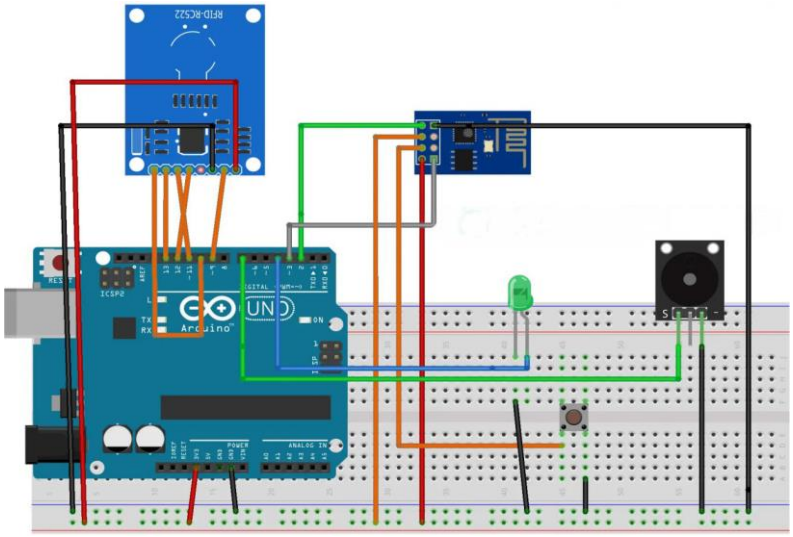


Рисунок 3.7 – Схема підключення для повної збірки системи автентифікації

Рекомендовано використовувати резистор 220 Ом при підключенні світлодіодної лампочки, щоб знизити вольтаж.

Після збірки схеми залишається розробити ПЗ для обміну даними з ESP-01 та зчитування даних з карток. Аналізуючи лістинг 2, спостерігаємо у рядках 7 та 9 ініціалізацію *SoftwareSerial* і *MFRC522* відповідно до пінів на схемі. Потрібно врахувати, що швидкість передачі даних в серіальному інтерфейсі мусить співпадати з обраною для програми ESP-01.

У методі *setup* (рядки 24 – 40) налаштовується серіальний інтерфейс. *MySerial* - це програмний серіальний інтерфейс, який ми використовуємо для зв’язку з ESP8266 ESP-01. Налаштовуємо інтерфейс SPI для RFID-зчитувача MFRC522.

У методі *loop* (рядки 43 – 122) послідовно перевіряється, чи не розміщено новий RFID біля нашого зчитувача MRFC522, за тим якщо була відсканована одна і та ж RFID-мітка, ми запобігаємо її багаторазовому зчитуванню за допомогою перевірки інтервалу між зчитуваннями. Цей функціонал запобігає повторному зчитуванню та покращує зручність користування. Функціонал з рядків 84 – 90 записує в серіальний порт ESP8266 ESP-01 відсканований RFID-код. Після чого очікується відповідь від ESP-01 (рядки 92 – 98). Пізніше, якщо дані доступні, витягується та обробляється відповідь JSON (рядки 100 – 109).

У методі *isUserAuthorized* (рядки 124 – 145) відбувається обробка відповіді API, а саме, десеріалізація JSON за допомогою бібліотеки ArduinoJSON. Якщо користувач має доступ – загоряється світлодіодна лампочка, якщо ж у доступі відмовлено – відтворюється звуковий сигнал через динамік.

3.4 Реалізація прикладного програмного інтерфейсу для адміністратора та API для взаємодії з Arduino

Перш ніж приступити до розробки REST API буде логічно сформувати архітектуру БД, для того щоб відштовхнутися від обмежень спроектованих специфікацій та не зайти в тупик при розробці застосунку.

При розробці структури бази даних інформацію раціонально представити у вигляді сукупності таблиць.

Оскільки в реалізації прикладного інтерфесу буде використовуватись Asp.Net Core Identity [14], не потрібно додатково описувати таблиці про членство, логін та дані користувача – вони будуть успадковані від системи, потрібно лише додати, специфічні для розробки, поля. Повністю ця бібліотека розкриває при використанні вже імплементованих інструментів, завдяки яким спрощується реалізація механізму авторизації та реєстрації [15] користувачів у системі.

При реалізації REST API та програмного інтерфейсу для адміністратора буде проілюстрований новий підхід до секціонування веб-додатку ASP.NET Core на невеликі функціональні групи, кожна з яких містить власний набір Razor Pages, контролерів, уявлень і моделей, за допомогою областей(Areas) [13].

Варто звернути увагу на використання Fluent API [3] конфігурацій для налаштування БД (зовнішніх зв’язків, властивостей). БД у цьому проекті можна створити за допомогою підходу Code First, завдяки моделям та зіставленню на основі класів сутностей і додаткової конфігурації моделі.

При використанні підходу Code First [21] EF Core API створює базу даних і таблиці за допомогою міграції на основі угод і конфігурації, передбачених у ваших доменних класах. Цей підхід корисний при доменно-орієнтованому проектуванні (DDD).

У проекті підготований функціонал для розширення застосунку і публікації його у різних середовищах, завдяки файлам налаштувань в *appsettings.json*.

Реалізація шаблону MVC [11] представлена у звичному використанні контроллера, а модель [16] і представлення у свою чергу згруповані угодою, що файл класу має те саме ім’я як і представлення, вони знаходяться в одному просторі імен, завдяки цьому ми все ще можемо розділяти логіку сторінки та її представлення.

Для адміністраторів потрібно врахувати розширені можливості, такі як CRUD-операції [20] (створення, читання, оновлення, видалення) для керування користувачами та їхнім доступом. Адміністраторський API повинен забезпечувати зручні та потужні інструменти для ефективного управління персоналом. CRUD є основними операціями у будь-якій програмі. Додатки, в яких не виконується жодна з цих операцій, зустрічаються рідко. Для демонстрації реального функціоналу теми ми створимо додаток для зберігання та управління даними про співробітників.

Отже, перш за все, нам потрібно встановити відповідні Nuget-пакети NuGet.Microsoft.EntityFrameworkCore та залежні від нього. У Entity Framework [2] фактичні фізичні сутності в базі даних створюються і управляються з класу DBContext, налаштуємо його із врахуванням майбутнього розширення і контролем середовищем розробки.

Наступний крок - налаштування ін’єкції залежностей, для виконання принципу інверсії залежностей. Для параметрів конструктора класу необхідна інформація про БД, а саме *connectionString*, який ми додаємо у *appsettings.Development*. Після цього можна реалізувати створення екземпляру *DbContext* при старті програми. Досі ми моделювали нашу БД, тепер можна створити саму БД, для цього нам потрібно виконати дві EF-команди: *Add-Migration "Initial"* та *Update-Database*. Тепер нам потрібно спроектувати інтерфейс користувача з Razor Views для відображення отриманих даних або формою/кнопкою для відправки даних методам контроллера.

Вивід списку записів у представленні MVC. Список записів з таблиці *employees* можна легко отримати за допомогою властивості *DbSet<Users>*. Колекція записів повертається функцією View. Отже, для відображення колекції записів повинен існувати файл Razor View *index.cshtml* (з такою ж назвою, як і у методу *index action*). Вивід інформації виконаний за допомогою HTML-таблиці з використанням циклу *foreach*.

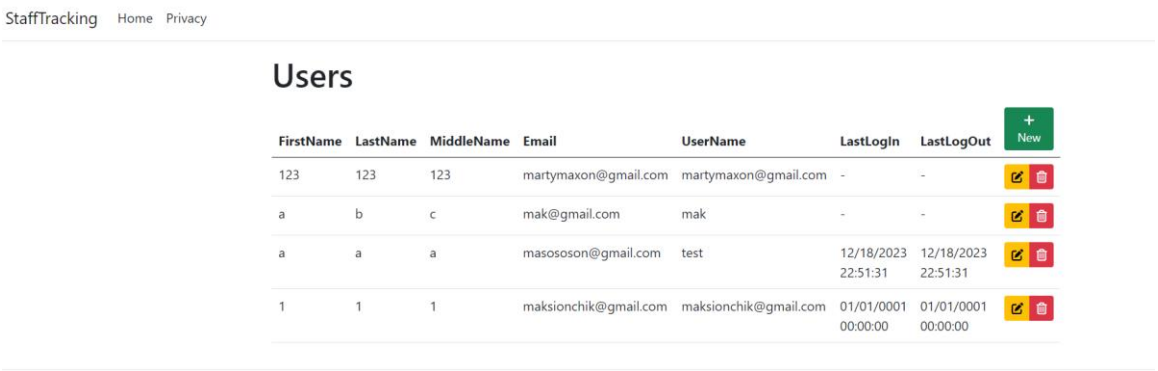


Рис. 3.8 Візуалізація вибірки списку користувачів

Також налаштуємо панель навігації з нашого глобального подання макета *Shared/_Layout.cshtml*. До нього слід додати посилання на таблицю стилів для Bootstrap і Font Awesome.

Додавання або редагування запису. Зазвичай за замовчуванням існують окремі методи для операцій вставки та оновлення. У цьому проекті вони об’єднані в один. Перш за все, потрібно визначити методи GET і POST для операцій вставки і оновлення за допомогою методу дії *AddOrEdit*.

Представлення *AddOrEdit.cshtml* повертається з методу операції *AddOrEdit* типу GET. Метод повертає новий екземпляр моделі працівника, якщо параметр *id* дорівнює пустий. В іншому випадку повертається відповідний запис працівника з заданим *id*. Нарешті, повернута модель заповнюється всередині представлення.

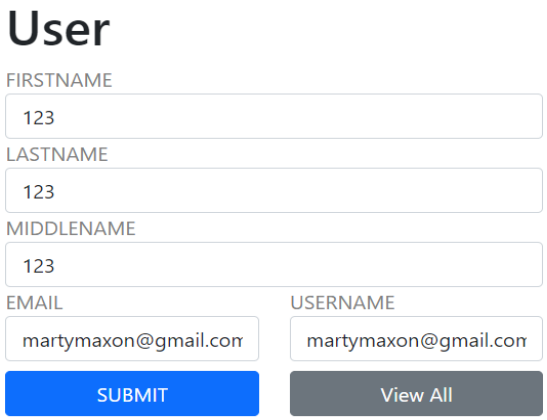


Рис. 3.9 Візуалізація форми для операції редагування

Перевірка форми. У моделі *EmployeeProfile* ми визначили властивість *FirstName* з атрибутом *Required*, що означає, що ця властивість повинна містити значення перед відправленням форми. Entity Framework Core Validation підтримує багато подібних атрибутів. Ми перевіряємо статус валідації моделі в методі *AddOrEdit* за допомогою властивості *ModelState.IsValid*. Якщо є помилка валідації, повертається та сама форма з тією самою моделлю, оновленою з повідомленням про помилку валідації.

Видалення запису. Функціонал видалення реалізований за допомогою кнопки видалення, це буде зроблено методом *Delete*. Усередині методу, завдяки, *SingleAsync* отримуємо відповідний запис із заданим ідентифікатором, нарешті викликається метод *Remove* із *DbContext*.

3.5 Аналіз тестування програмно-апаратного комплексу.

Усі ці види тестування в сукупності допомагають гарантувати, що система обліку робочого часу працює ефективно, надійно та відповідає вимогам користувачів та бізнесу:

- Тестування відповідності вимогам бізнесу: Тестування орієнтується на перевірку того, чи відповідає система визначеним вимогам бізнесу. Кожна функція та можливість системи перевіряється відповідно до специфікацій, які визначені у технічному завданні. Забезпечення відповідності цим вимогам гарантує, що система виконує завдання, для яких вона була створена.

● Тестування відновлення та резервного копіювання: Система перевіряється на можливість відновлення після втрати даних або системних збоїв. Проводяться тести на ефективність резервного копіювання та відновлення, щоб забезпечити безперебійність роботи системи в умовах непередбачуваних ситуацій.

● Тестування масштабованості: Система перевіряється на можливість масштабування під зростання обсягу даних або користувачів. Це важливо для забезпечення того, що система зможе ефективно працювати в умовах росту бізнесу та збільшення обсягу введених даних.

● Тестування міграції даних: Перевіряється коректність та безпека процесу міграції даних при необхідності перенесення системи на іншу платформу чи оновлення бази даних. Тести включають перевірку цілісності та точності даних під час міграції.

● Тестування сумісності: Перевіряється сумісність системи з різними операційними системами, браузерами та іншими програмами, які можуть впливати на її функціональність. Це важливо для забезпечення оптимальної роботи системи в різних середовищах.

● Тестування користувацького досвіду: Оцінюється користувацький досвід та зручність використання системи. Тести включають оцінку інтерфейсу користувача, швидкість реакції системи та загальну задоволеність користувачів взаємодією з програмним продуктом.

● Тестування апаратних вимог: Перевіряється відповідність системи апаратним вимогам. Це включає в себе перевірку налаштувань обладнання та його здатності працювати з програмним продуктом без перебоїв.

● Тестування системи авторизації за допомогою Arduino: Проводиться тестування системи авторизації, яка використовує плату Arduino. Оцінюється точність та надійність системи в процесі реєстрації та авторизації персоналу на підприємстві. Порівнюючи з іншими методами авторизації, такими як біометричні системи чи RFID-карти, визначається ефективність та швидкодія системи на базі Arduino.

● Операції CRUD для адміністратора

○ Тест на створення нового користувача: Перевірка, чи коректно додається новий користувач до системи після виклику відповідного API-методу.

○ Тест на оновлення існуючого користувача: Визначення правильності оновлення даних користувача та перевірка, чи відбувається це безпомилково.

○ Тест на видалення користувача: Перевірка, чи коректно видаляється користувач після виклику API-методу видалення.

Більшість функцій системи, включаючи автентифікацію користувача з використанням Arduino та RFID-карток, а також прикладний програмний інтерфейс для адміністратора та API для взаємодії з Arduino, були успішно реалізовані. Система працює стабільно та надійно.

Під час тестування було виявлено деякі потенційні вразливості, зокрема, можливість перехоплення та підробки RFID-сигналів. Ця проблема може становити загрозу безпеці системи та конфіденційності даних. Для усунення виявлених вразливостей пропонується використати додаткові заходи безпеки, такі як шифрування даних, використання токенів для аутентифікації та впровадження механізмів перевірки цілісності сигналів RFID.

Система успішно інтегрована в локальну мережу, що забезпечує зручну та швидку взаємодію між пристроями та користувачами. Проте це також створює певні потенційні ризики безпеки, які варто врахувати:

● Мережеві атаки: У разі недостатнього захисту мережі, система може стати ціллю для різноманітних атак, таких як перехоплення даних, перехват з’єднань або злам системи. Необхідно вживати заходів безпеки, таких як використання захищених протоколів зв’язку та мережевого шифрування, для запобігання таким атакам.

● Несанкціонований доступ: Існує ризик, що несанкціоновані користувачі можуть отримати доступ до системи через локальну мережу. Це може призвести до витоку конфіденційної інформації або порушення функціональності системи. Забезпечення надійної аутентифікації та авторизації користувачів є важливим для мінімізації цього ризику.

● Загрози фізичної безпеки: В локальній мережі існує ризик фізичного доступу до обладнання системи, що може призвести до недостатнього контролю над пристроями або навіть їх втрати. Важливо забезпечити фізичну безпеку обладнання за допомогою захищених приміщень та механізмів контролю доступу.

Висновки до розділу 3

У третьому розділі нашої роботи ми зосередилися на практичній реалізації та експериментальному дослідженні системи контролю доступу. Почали ми з вибору та обґрунтування засобів реалізації системи, визначивши відповідні технології, такі як СКБД PostgreSQL, ASP.NET Core MVC та мікроконтролер Arduino. Цей вибір був зроблений на основі їхньої потужності, гнучкості та можливості ефективної інтеграції.

Розроблена архітектура системи, яка включає розробку системи для автентифікації користувача з використанням Arduino та RFID-карток, а також прикладного програмного інтерфейсу для адміністратора та API для взаємодії з Arduino. Ці компоненти були створені з метою забезпечення зручного та безпечного управління доступом.

Наприкінці, проведений аналіз тестування програмно-апаратного комплексу, під час якого виявили деякі проблеми та вразливості, що потребують подальшого уточнення та виправлення. Однак незважаючи на це, наша реалізація системи контролю доступу має потенціал стати ефективним інструментом для забезпечення безпеки та контролю доступу в різних сферах, від корпоративних офісів до приватних будинків.

У подальшому планується активне вдосконалення системи, зосереджуючись на виправленні виявлених проблем та розширенні функціональності. Це включає в себе впровадження нових можливостей, які підвищать безпеку та зручність використання. Наприклад, можливе розширення методів аутентифікації для забезпечення більшого рівня захисту, вдосконалення інтерфейсу користувача для забезпечення зручності взаємодії, а також додавання нових функцій для

задоволення різноманітних потреб користувачів. Такі кроки допоможуть підвищити ефективність та цінність системи в майбутньому використанні.

ВИСНОВКИ

У ході дослідження було проведено аналіз та порівняння різноманітних методів аутентифікації в системах контролю доступу, визначено їх переваги, недоліки та потенційні ризики. Вивчення актуальних тенденцій у цій галузі дозволило нам зрозуміти, як швидко змінюється ландшафт технологій та як важливо постійно оновлювати та вдосконалювати методи аутентифікації для забезпечення високого рівня безпеки.

Розробку практичного прототипу системи контролю доступу було здійснено використовуючи мікроконтролер Arduino та технологію RFID. Цей прототип не лише дозволив нам випробувати теоретичні концепції на практиці, але й став основою для подальшого вдосконалення та розширення. Отриманий пристрій може слугувати відмінним стартовим пунктом для розробки більш складних та спеціалізованих систем контролю доступу в майбутньому.

Цей шаблон є гнучким та легко адаптованим, що дозволяє впроваджувати нові функції та інтегрувати додаткові засоби безпеки з мінімальними зусиллями. Таким чином, він може слугувати відмінним інструментом для компаній та організацій, які шукають швидке та ефективне рішення для підвищення рівня безпеки та контролю доступу в їхніх приміщеннях.

В цілому, розроблений прототип є важливим кроком у напрямку створення майбутніх систем контролю доступу, які відповідають сучасним вимогам безпеки та приватності, і він має потенціал стати стандартом в цій області.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація щодо ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0
2. Офіційна документація щодо Entity Framework Core, технології ORM в .NET [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/ef/core/
3. Офіційна документація Fluent API в Entity Framework Core [Електронний ресурс] – Режим доступу до ресурсу: https://www.entityframeworktutorial.net/efcore/fluent-api-in-entity-framework-core.aspx
4. Офіційна документація PostgreSQL. [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/15/tutorial-start.html
5. Бібліотека Arduino для MFRC522 та інших RFID RC52-базованих модулів [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/miguelbalboa/rfid
6. Бібліотека ArduinoJson для Arduino та IoT [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/miguelbalboa/rfid
7. Multi-Factor Authentication: A Survey [Електронний ресурс] / A. Ometov, S. Bezzateev, N. Mäkitalo — 2018 — Режим доступу до ресурсу: https://www.researchgate.net/publication/322288752_Multi-Factor_Authentication_A_Survey
8. Steven Morris. Database Systems: Design, Implementation, & Management. [Текст] – 2018; 816 с.
9. A Survey on Cyber Security Evolution and Threats: BiometricAuthentication Solutions. In Biometric Security and Privacy; Springer: Berlin, Germany [Текст] / Benarous, L.; Kadri, B.; Bouridane, A. — 2017; 371–411 с.
10. Arduino Programming in 24 Hours, Sams Teach Yourself. [Текст] Richard Blum. – 2014; 432 с.
11. Pro ASP.NET Core MVC. [Текст] Adam Freeman. – 2016; 1018 с.

12. Чиста архітектура. [Текст] Роберт Сесіл Мартін. – 2019; 368 с.
13. Документація щодо особливостей областей в ASP.NET Core 8.0 з використанням Razor Pages [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/aspnet/core/mvc/controllers/areas?view=aspnetcore-8.0
14. Документація щодо налаштувань ASP.NET Core Identity [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity-configuration?view=aspnetcore-7.0
15. Документація щодо додавання, завантаження та видалення користувацьких даних користувача до Identity у проекті ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/add-user-data?view=aspnetcore-7.0&tabs=visual-studio
16. Документація щодо налаштування моделі Identity в ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/customize-identity-model?view=aspnetcore-7.0#entity-types
17. TRANSACT-SQL [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Transact-SQL
18. Обробка HTTP-запитів за допомогою ESP-01 [Електронний ресурс] – Режим доступу до ресурсу: https://randomnerdtutorials.com/esp8266-nodemcu-http-get-post-arduino/#http-post
19. Технологія RFID [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Радіочастотна_ідентифікація
20. CRUD-операції [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/crud?view=aspnetcore-8.0
21. Застосування підходу Code First [Електронний ресурс] – Режим доступу до ресурсу:

https://learn.microsoft.com/uk-ua/ef/ef6/modeling/code-first/workflows/new-database

ДОДАТКИ

ДОДАТОК А

Таблиці з’єднання пінів Arduino-схеми

Таблиця А. 1 – Підключення модуля RFID до плати Arduino для тестування

Пін (pin)	Підключення до Arduino Uno
SDA	PIN 10
SCK	PIN 13
MOSI	PIN 11
MISO	PIN 12
IRQ	Не потрібно приєднувати
GND	GND (заземлення)
RST	PIN 9
3.3V	3.3V (живлення)

Таблиця А. 2 – Підключення модуля RFID до плати Arduino, фінальна збірка

MFRC522 RFID	Arduino Uno
SDA	PIN 10
SCK	PIN 13
MOSI	PIN 11
MISO	PIN 12

IRQ	Не потрібно приєднувати
GND	GND
RST	PIN 9
VCC	3.3 V

Таблиця А. 3 – Підключення активного динаміка до плати Arduino

Arduino Uno	Активний динамік
PIN 7	SIGNAL (+)
GND	GND

Таблиця А. 4 – Підключення світлодіодної лампочка до плати Arduino

Arduino Uno	Світлодіодна лампочка
PIN 4	Анод
GND	Катод

Таблиця А. 5 – Підключення модуля ESP-01 до плати Arduino, фінальна збірка

ESP8266 ESP-01	Arduino Uno
CH_PD	3.3V

RST	Switch To GND
VCC	3.3V
GND	GND
GPIO 0	Не потрібно приєднувати
GPIO 2	Не потрібно приєднувати
TX	PIN 2
RX	PIN 3

ДОДАТОК Б

Таблиці БД

Таблиця Б. 1 – Специфікація таблиці departments

Назва стовпця	Тип даних	Призначення
id	SERIAL	Унікальний ідентифікатор відділу компанії
department_name	VARCHAR(50)	Назва відділу компанії
location	VARCHAR(100)	Місцезнаходження відділу компанії

Таблиця Б. 2 – Специфікація таблиці posts

Назва стовпця	Тип даних	Призначення
id	SERIAL	Унікальний ідентифікатор посади
post_name	VARCHAR(50)	Назва посади
department_id	INT	Зовнішній ключ, пов’язаний з ідентифікатором відділу в таблиці departments.

Таблиця Б. 3 – Специфікація таблиці employees

Назва стовпця	Тип даних	Призначення
id	SERIAL	Унікальний ідентифікатор відділу компанії
post_id	INT	Зовнішній ключ, пов’язаний з ідентифікатором посади в таблиці posts

inn	VARCHAR(12)	Ідентифікаційний номер працівника
profile_id	INT	Зовнішній ключ, пов’язаний з ідентифікатором профілю в таблиці employee_profiles
access_card	VARCHAR(50)	Відповідний працівнику токен, шо використовується при фізичній ідентифікації

Таблиця Б. 4 – Специфікація таблиці employee_profiles

Назва стовпця	Тип даних	Призначення
id	SERIAL	Унікальний ідентифікатор працівника, до якого прив’язаний профіль
employee_id	INT	Назва відділу компанії
sex	VARCHAR(10)	Стать працівника (наприклад, "Male", "Female")
birth_date	DATE	Дата народження працівника
image	VARCHAR(100)	Шлях до зображення або фотографії працівника
home_address	VARCHAR(255)	Домашня адреса працівника
family_status	VARCHAR(50)	Сімейний стан працівника (наприклад, "Single", "Married", "Divorced", тощо)

Таблиця Б. 5 – Специфікація таблиці work_hours

Назва стовпця	Тип даних	Призначення

id	SERIAL	Унікальний ідентифікатор робочого часу
employee_id	INT	Зовнішній ключ, пов’язаний з ідентифікатором працівника в таблиці employees
date_worked	DATE	Дата проведення робочого часу
start_time	TIME	Початковий час роботи
end_time	TIME	Закінчений час роботи

Таблиця Б. 6 – Специфікація таблиці vacations

Назва стовпця	Тип даних	Призначення
id	SERIAL	Унікальний ідентифікатор відпустки
employee_id	INT	Зовнішній ключ, пов’язаний з ідентифікатором працівника в таблиці employees
start_date	DATE	Дата початку відпустки
end_date	DATE	Дата закінчення відпустки
reason	VARCHAR(255)	Причина відпустки

Таблиця Б. 7 – Специфікація таблиці users

Назва стовпця	Тип даних	Призначення
id	SERIAL	Унікальний ідентифікатор користувача
username	VARCHAR(50)	Ім’я користувача для входу в систему
password	VARCHAR(255)	Захешований пароль користувача
employee_id	INT	Зовнішній ключ, пов’язаний з ідентифікатором працівника в таблиці employees

Таблиця Б. 8 – Специфікація таблиці access_log

Назва стовпця	Тип даних	Призначення
id	SERIAL	Унікальний ідентифікатор відпустки

user_id	INT	Зовнішній ключ, пов’язаний з ідентифікатором користувача в таблиці users
log_in_time	TIMESTAMP	Час входу в систему
log_out_time	TIMESTAMP	Час виходу з системи

ДОДАТОК В

Лістинг програми

Лістинг 1

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h>

// NOTE:  CHANGE THIS TO YOUR WIFI SSID AND PASSWORD
const char* ssid     = "****";
const char* password = "****";

//#define DEBUG    //If you comment this line, the DPRINT & DPRINTLN lines are defined as blank.
#ifdef DEBUG    //Macros are usually in all capital letters.
  #define DPRINT(...)     Serial.print(__VA_ARGS__)     //DPRINT is a macro, debug print
  #define DPRINTLN(...)   Serial.println(__VA_ARGS__)     //DPRINTLN is a macro, debug print with new line
#else
  #define DPRINT(...)     //now defines a blank line
  #define DPRINTLN(...)   //now defines a blank line
#endif

void setup() {
  delay(10);
  Serial.begin(115200);

  // We start by connecting to a WiFi network
```

```
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

const byte numChars = 20;
char receivedChars[numChars];   // an array to store the received data

boolean newData = false;

void loop() {
    recvWithEndMarker();
    processRFIDCode();
}
void recvWithEndMarker() {
    static byte ndx = 0;
    char endMarker = '\n';
    char rc;

    while (Serial.available() > 0 && newData == false) {
        rc = Serial.read();

        if (rc != endMarker) {
            receivedChars[ndx] = rc;
            ndx++;
            if (ndx >= numChars) {
                ndx = numChars - 1;
            }
        }
        else {
            receivedChars[ndx] = '\0'; // terminate the string
            ndx = 0;
            newData = true;
```

```
        }
    }
}

void processRFIDCode() {
    if (newData == true) {
        DPRINT("This just in ... ");
        DPRINTLN(receivedChars);
        newData = false;
        isUserAuthorized(receivedChars);
    }
}



void isUserAuthorized(String rfIdCode){
    // wait for WiFi connection
  if ((WiFi.status() == WL_CONNECTED)) {

    WiFiClient client;

    HTTPClient http;

    // NOTE:  CHANGE THIS TO THE IP ADDRESS WHERE YOUR APPLICATION IS RUNNING
    String url = "http://****/auth/authorized?rf_id_code=";
    String encodedRFIDCode = urlencode(rfIdCode);
    url+=encodedRFIDCode;
    DPRINT("url :: ");
    DPRINTLN(url);

    http.setTimeout(20000);
    if (http.begin(client, url)) {  // HTTP

      // start connection and send HTTP header
      int httpCode = http.GET();
      DPRINT("httpCode :: ");
      DPRINTLN(httpCode);

      // httpCode will be negative on error
      if (httpCode > 0) {
        // HTTP header has been send and Server response header has been handled

        if (httpCode == HTTP_CODE_OK || httpCode == HTTP_CODE_MOVED_PERMANENTLY) {
```

```
          String payload = http.getString();
          DPRINTLN(payload);
          sendDataBackToArduino(payload);

        }
      } else {
        Serial.printf("[HTTP]       GET...       failed,       error:       %s\n",
http.errorToString(httpCode).c_str());
      }

      http.end();
    } else {
      Serial.printf("[HTTP} Unable to connect\n");
    }
  }
}

String urlencode(String str)
{
    String encodedString="";
    char c;
    char code0;
    char code1;
    char code2;
    for (int i =0; i < str.length(); i++){
      c=str.charAt(i);
      if (c == ' '){
        encodedString+= '+';
      } else if (isalnum(c)){
        encodedString+=c;
      } else{
        code1=(c & 0xf)+'0';
        if ((c & 0xf) >9){
            code1=(c & 0xf) - 10 + 'A';
        }
        c=(c>>4)&0xf;
        code0=c+'0';
        if (c > 9){
            code0=c - 10 + 'A';
        }
        code2='\0';
        encodedString+='%';
        encodedString+=code0;
```

```
        encodedString+=code1;
        //encodedString+=code2;
      }
      yield();
    }
    return encodedString;

}

void sendDataBackToArduino(String payload){
  const size_t capacity = JSON_OBJECT_SIZE(1) + 30;
  DynamicJsonDocument doc(capacity);
    // Parse JSON object
  DeserializationError error = deserializeJson(doc, payload);
  if (error) {
    Serial.print(F("deserializeJson() failed: "));
    Serial.println(error.c_str());
    return;
  }
  serializeJson(doc, Serial);
}
```

Лістинг 2

```
#include <StreamUtils.h>
#include <SPI.h>
#include <MFRC522.h>
#include <ArduinoJson.h>
#include <SoftwareSerial.h>

SoftwareSerial mySerial(2, 3); // TX, RX

MFRC522 mfrc522(10, 9);   // Create Instance of MFRC522 mfrc522(SS_PIN, RST_PIN)

const int buzzerPin = 7; //buzzer attach to digital 7
const int ledPin = 4; //led attach to digital 4

bool isRead = false;
bool isNewCard = false;
String tagContent = "";
String currentUID = "";

// Interval before we process the same RFID
int INTERVAL = 2000;
unsigned long previousMillis = 0;
unsigned long currentMillis = 0;

void setup()
{
  Serial.begin(115200);
  mySerial.begin(115200);
  //mySerial.setTimeout(5000);
  //delay(5000);
  // Setup the buzzer
  pinMode(buzzerPin, OUTPUT);
  // Setup the led
  pinMode(ledPin, OUTPUT);

  SPI.begin();
  mfrc522.PCD_Init();

  Serial.println("Detecting RFID Tags");
  Serial.println(mySerial.readString());
}
```

```
void loop()
{
  // Look for new cards
  if (mfrc522.PICC_IsNewCardPresent())
  {

    // Select one of the cards
    if (mfrc522.PICC_ReadCardSerial())
    {

      isRead = true;

      byte letter;
      for (byte i = 0; i < mfrc522.uid.size; i++)
      {
        tagContent.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
        tagContent.concat(String(mfrc522.uid.uidByte[i], HEX));
      }

      tagContent.toUpperCase();
    }
    if (isRead) {
      currentMillis = millis();
      // If current UID is not equal to current UID..meaning new card, then
validate if it has access
      if (currentUID != tagContent) {
        currentUID =  tagContent;
        isNewCard = true;
      } else {
        // If it is the same RFID UID then we wait for interval to pass before we
process the same RFID
        if (currentMillis - previousMillis >= INTERVAL) {
          isNewCard = true;
        } else {
          isNewCard = false;
        }
      }
      if (isNewCard) {
        if (tagContent != "") {
          // turn off LED if it is on for new card
          turnOffLED();
```

```
          previousMillis = currentMillis;
          //Send the RFID Uid code to the ESP-01 for validation
          Serial.println(mySerial.readString());
          Serial.print("Sending data to ESP-01 :: ");
          Serial.println(tagContent);

          mySerial.println(tagContent);
          Serial.println("Waiting for response from ESP-01....");

          int iCtr = 0;
          while (!mySerial.available()) {
            iCtr++;
            if (iCtr >= 100)
              break;
            delay(50);
          }

          if (mySerial.available()) {
            bool isAuthorized = isUserAuthorized(tagContent);
            Serial.println(isAuthorized);
            // If not authorized then sound the buzzer
            if (!isAuthorized) {
              playNotAuthorized();
            } else { //light up the Green LED
              turnOnLED();
            }
          }
          Serial.println("Finished processing response from ESP-01....");
        }

      }

    } else {
      Serial.println("No card details was read!");
    }
    tagContent = "";
    isNewCard = false;
  }

}

bool isUserAuthorized(String tagContent) {
```

```
  const size_t capacity = JSON_OBJECT_SIZE(1) + 30;
  DynamicJsonDocument doc(capacity);

  // Use during debugging
    ReadLoggingStream loggingStream(mySerial, Serial);
    DeserializationError error = deserializeJson(doc, loggingStream);

  // Use during actual running
  //DeserializationError error = deserializeJson(doc, mySerial);
  if (error) {
    //    Serial.print(F("deserializeJson() failed: "));
    Serial.println(error.c_str());
    //return error.c_str();
  }

  bool   is_authorized = doc["is_authorized"] == "true";
  Serial.print("is_authorized :: ");
  Serial.println(is_authorized);

  return is_authorized;
}

void playNotAuthorized() {
  tone(buzzerPin, 2000);
  delay(500);
  noTone(7);
}

void turnOffLED() {
  digitalWrite(ledPin, LOW);
}

void turnOnLED() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
}
```

Лістинг 3

Program.cs

```
using Microsoft.EntityFrameworkCore;
using StaffTracking.Data.DbContext;
using StaffTracking.Data.Entities;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
//DI for DbContext
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'AuthDbContextConnection' not found.");

builder.Services.AddDbContext<ApplicationDbContext>(options => options.UseNpgsql(connectionString).UseSnakeCaseNamingConvention());

builder.Services.AddDefaultIdentity<User>(options => options.SignIn.RequireConfirmedAccount = false)
    .AddEntityFrameworkStores<ApplicationDbContext>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
}
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();
app.UseSwagger();
app.UseSwaggerUI();
```

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=User}/{action=Index}/{id?}");

app.Run();
```

HomeController.cs

```
using System.Diagnostics;
using Microsoft.AspNetCore.Mvc;
using StaffTracking.Models;

namespace StaffTracking.Controllers
{
   public class HomeController : Controller
   {
       private readonly ILogger<HomeController> _logger;

       public HomeController(ILogger<HomeController> logger)
       {
           _logger = logger;
       }

       public IActionResult Index()
       {
           return View();
       }

       public IActionResult Privacy()
       {
           return View();
       }

       [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore
= true)]
       public IActionResult Error()
       {
           return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
       }
   }
}
```

AuthController.cs

```
using System.Text.RegularExpressions;
using Microsoft.AspNetCore.Mvc;
```

```
using Microsoft.EntityFrameworkCore;
using StaffTracking.Data.DbContext;
using StaffTracking.Data.Entities;
using StaffTracking.Models;

namespace StaffTracking.Controllers;

[Route("auth")]
public class AuthController: Controller
{
    private readonly ILogger<HomeController> _logger;
    private readonly ApplicationDbContext _context;

    public AuthController(ILogger<HomeController> logger, ApplicationDbContext
context)
    {
        _logger = logger;
        _context = context;
    }

    [HttpGet]
    [Route("authorized")]
    public async Task<AuthResponseDto> Authorized([FromQuery] string rf_id_code)
    {
        var employee = await _context.Employees.SingleOrDefaultAsync(x =>
x.AccessCard.Trim() == rf_id_code.Trim());

        if (employee != null)
        {
            _context.AccessLogs.Add(new AccessLog
            {
                LogInTime = DateTime.UtcNow,
                UserId = employee.Id
            });

            await _context.SaveChangesAsync();

            return new AuthResponseDto
            {
                IsAuthorized = "true"
            };
        }
```

```
        return new AuthResponseDto
        {
            IsAuthorized = "false"
        };
    }
}
```

UserController.cs

```
#nullable disable
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using StaffTracking.Data.DbContext;
using StaffTracking.Data.Entities;
using StaffTracking.Models;

namespace StaffTracking.Controllers
{
   public class UserController : Controller
   {
       private readonly ApplicationDbContext _context;
       private readonly UserManager<User> _userManager;
       private readonly IUserStore<User> _userStore;
       private readonly IConfiguration _config;
       private readonly IUserEmailStore<User> _emailStore;

       public   UserController(ApplicationDbContext   context,   IUserStore<User>
userStore, UserManager<User> userManager, IConfiguration config)
       {
           _context = context;
           _userStore = userStore;
           _userManager = userManager;
           _config = config;
           _emailStore = GetEmailStore();
       }

       // GET: User
       public async Task<IActionResult> Index()
       {
```

```
            var users = await _context.Users.Include(x => x.Employee).ThenInclude(x
=> x.EmployeeProfile)
                .Include(x => x.AccessLogs)
                .Select(x => new UserDto
                {
                    Id = x.Id,
                    UserName = x.UserName,
                    Email = x.Email,
                    LastLogIn          =          x.AccessLogs.OrderBy(y          =>
y.Id).LastOrDefault().LogInTime,
                    LastLogOut         =          x.AccessLogs.OrderBy(y          =>
y.Id).LastOrDefault().LogOutTime,
                    FirstName = x.Employee.EmployeeProfile.FirstName,
                    LastName = x.Employee.EmployeeProfile.LastName,
                    MiddleName = x.Employee.EmployeeProfile.MiddleName
                }).ToListAsync();

            return View(users);
        }


        // GET: User/AddOrEdit
        public async Task<IActionResult> AddOrEdit(string? id)
        {
            if (string.IsNullOrEmpty(id))
                return View(new UserDto());
            var user = await _context.Users.Include(x => x.Employee).ThenInclude(x
=> x.EmployeeProfile)
                .Include(x => x.AccessLogs).SingleAsync(x => x.Id == id);
            return View(new UserDto
            {
                Id = user.Id,
                UserName = user.UserName,
                Email = user.Email,
                LastLogIn = user.AccessLogs.MaxBy(y => y.Id)?.LogInTime,
                LastLogOut = user.AccessLogs.MaxBy(y => y.Id)?.LogOutTime,
                FirstName = user.Employee.EmployeeProfile.FirstName,
                LastName = user.Employee.EmployeeProfile.LastName,
                MiddleName = user.Employee.EmployeeProfile.MiddleName
            });
        }

        // POST: User/AddOrEdit
```

```
        // To protect from overposting attacks, enable the specific properties you
want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public                        async                        Task<IActionResult>
AddOrEdit([Bind("Id,UserName,Email,FirstName,LastName,MiddleName")]        UserDto
userDto)
        {
            /*if (!ModelState.IsValid)
                return View(userDto);*/

            if (string.IsNullOrEmpty(userDto.Id))
            {
                var user = new User
                {
                    Employee = new Employee
                    {
                        Inn = Guid.NewGuid().ToString(),
                        EmployeeProfile = new EmployeeProfile
                        {
                            FirstName = userDto.FirstName,
                            LastName = userDto.LastName,
                            MiddleName = userDto.MiddleName
                        }
                    },
                    AccessLogs = new List<AccessLog>
                    {
                        new ()
                        {
                            LogInTime = DateTime.UtcNow
                        }
                    }
                };

                await      _userStore.SetUserNameAsync(user,      userDto.UserName,
CancellationToken.None);
                await         _emailStore.SetEmailAsync(user,         userDto.Email,
CancellationToken.None);
                await                                    _userManager.CreateAsync(user,
_config.GetSection("DefaultPassword").Get<string>());
            }
            else
```

```
            {
                var user = await _context.Users.Include(x => x.Employee).ThenInclude(x => x.EmployeeProfile)
                    .Include(x => x.AccessLogs).SingleAsync(x => x.Id == userDto.Id);

                user.UserName = userDto.UserName;
                user.Employee.EmployeeProfile.FirstName = userDto.FirstName;
                user.Employee.EmployeeProfile.LastName = userDto.LastName;
                user.Employee.EmployeeProfile.MiddleName = userDto.MiddleName;
                user.Email = userDto.Email;

                _context.Users.Update(user);
                await _context.SaveChangesAsync();
            }

            return RedirectToAction(nameof(Index));
        }

        // POST: User/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(string id)
        {
            var user = await _context.Users.SingleAsync(x => x.Id == id);
            _context.Users.Remove(user);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private IUserEmailStore<User> GetEmailStore()
        {
            if (!_userManager.SupportsUserEmail)
            {
                throw new NotSupportedException("The default UI requires a user store with email support.");
            }
            return (IUserEmailStore<User>)_userStore;
        }
    }
}
```

UserDto.cs

```
namespace StaffTracking.Models;

public class UserDto
{
   public string Id { get; set; }
   public string Email { get; set; }
   public string UserName { get; set; }
   public DateTime? LastLogIn { get; set; }
   public DateTime? LastLogOut { get; set; }
   public string FirstName { get; set; }
   public string LastName { get; set; }
   public string MiddleName { get; set; }
}
```

AddOrEdit.cshtml

```
@model StaffTracking.Models.UserDto

@{
   ViewData["Title"] = "User";
}

<div class="row">
   <div class="col-md-6 offset-md-3">
       <h1>User</h1>
       <form asp-action="AddOrEdit" autocomplete="off">
           <div asp-validation-summary="ModelOnly" class="text-danger"></div>
           <input type="hidden" asp-for="Id" />
           <div class="form-group">
               <label asp-for="FirstName" class="control-label"></label>
               <input asp-for="FirstName" class="form-control" />
               <span asp-validation-for="FirstName" class="text-danger"></span>
           </div>
           <div class="form-group">
               <label asp-for="LastName" class="control-label"></label>
               <input asp-for="LastName" class="form-control" />
               <span asp-validation-for="LastName" class="text-danger"></span>
           </div>
           <div class="form-group">
```

```
                <label asp-for="MiddleName" class="control-label"></label>
                <input asp-for="MiddleName" class="form-control" />
                <span asp-validation-for="MiddleName" class="text-danger"></span>
            </div>
            <div class="row">
                <div class="col-md-6">
                    <div class="form-group">
                        <label asp-for="Email" class="control-label"></label>
                        <input asp-for="Email" class="form-control" />
                        <span        asp-validation-for="Email"        class="text-
danger"></span>
                    </div>
                </div>
                <div class="col-md-6">
                    <div class="form-group">
                        <label asp-for="UserName" class="control-label"></label>
                        <input asp-for="UserName" class="form-control" />
                        <span       asp-validation-for="UserName"       class="text-
danger"></span>
                    </div>
                </div>
            </div>

            <div class="row mt-2">
                <div class="col-md-6">
                    <div class="form-group">
                        <input  type="submit"  value="Submit"  class="btn  btn-primary
w-100" />
                    </div>
                </div>
                <div class="col-md-6">
                    <a  asp-action="Index"  class="btn  btn-secondary    w-100">View
All</a>
                </div>
            </div>
        </form>
    </div>
</div>


@section Scripts {
    @{
    await Html.RenderPartialAsync("_ValidationScriptsPartial");
```

```
}
}
```

Index.cshtml

```
@using System.Globalization
@model IEnumerable<StaffTracking.Models.UserDto>

@{
   ViewData["Title"] = "Index";
}
<div class="row">
   <div class="col-md-12">
       <h1>Users</h1>

       <table class="table">
           <thead>
               <tr>
                   <th>
                       @Html.DisplayNameFor(model => model.FirstName)
                   </th>
                   <th>
                       @Html.DisplayNameFor(model => model.LastName)
                   </th>
                   <th>
                       @Html.DisplayNameFor(model => model.MiddleName)
                   </th>
                   <th>
                       @Html.DisplayNameFor(model => model.Email)
                   </th>
                   <th>
                       @Html.DisplayNameFor(model => model.UserName)
                   </th>
                   <th>
                       @Html.DisplayNameFor(model => model.LastLogIn)
                   </th>
                   <th>
                       @Html.DisplayNameFor(model => model.LastLogOut)
                   </th>
                   <th>
                       <a asp-action="AddOrEdit" class="btn btn-sm btn-success">
                           <i class="fa-solid fa-plus"></i> New
```

```
                    </a>
                </th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td>
                        @Html.DisplayFor(modelItem => item.FirstName)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.LastName)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.MiddleName)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.Email)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.UserName)
                    </td>
                    <td>
                        @{
                            var lastLogIn = item.LastLogOut.HasValue
                                ?
item.LastLogOut.Value.ToString(CultureInfo.InvariantCulture)
                                : "-";
                        }
                        @Html.DisplayFor(modelItem => lastLogIn)
                    </td>
                    <td>
                        @{
                            var lastLogOut = item.LastLogOut.HasValue
                                ?
item.LastLogOut.Value.ToString(CultureInfo.InvariantCulture)
                                : "-";
                        }
                        @Html.DisplayFor(modelItem => lastLogOut)
                    </td>
                    <td>
                        <form asp-action="Delete" asp-route-id="@item.Id">
```

```
                                <div class="btn-group btn-group-sm">
                                    <a       asp-action="AddOrEdit"       asp-route-
id="@item.Id" class="btn btn-warning">
                                        <i class="fa-solid fa-pen-to-square"></i>
                                    </a>
                                    <button type="submit" class="btn btn-danger"
                                        onclick="return  confirm('Are  you  sure  to
delete this user?')">
                                        <i class="far fa-trash-alt"></i>
                                    </button>
                                </div>
                            </form>
                        </td>
                    </tr>
                }
            </tbody>
        </table>
    </div>
</div>
```

Index.cshtml

```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn  about  <a  href="https://docs.microsoft.com/aspnet/core">building  Web
apps with ASP.NET Core</a>.</p>
</div>
```

```
appsettings.Development.json
{
 "Logging": {
   "LogLevel": {
     "Default": "Information",
     "Microsoft.AspNetCore": "Warning"
   }
 },
```

```
 "DefaultPassword": "Qwerty-1",
 "ConnectionStrings": {
   "DefaultConnection":
"server=localhost;port=5432;database=auth_system;userid=postgres;password=1111"
 }
}
```

Error.cshtml

```
@model StaffTracking.Models.ErrorViewModel
@{
   ViewData["Title"] = "Error";
}

<h1 class="text-danger">Error.</h1>
<h2 class="text-danger">An error occurred while processing your request.</h2>

@if (Model.ShowRequestId)
{
   <p>
       <strong>Request ID:</strong> <code>@Model.RequestId</code>
   </p>
}

<h3>Development Mode</h3>
<p>
   Swapping to <strong>Development</strong> environment will display more detailed
information about the error that occurred.
</p>
<p>
   <strong>The Development environment shouldn't be enabled for deployed
applications.</strong>
   It can result in displaying sensitive information from exceptions to end users.
   For local debugging, enable the <strong>Development</strong> environment by
setting the <strong>ASPNETCORE_ENVIRONMENT</strong> environment variable to
<strong>Development</strong>
   and restarting the app.
</p>
```

Лістинг 4

AccessLog.cs

```
namespace AuthSystem.Areas.Identity.Data.Entities;

public class AccessLog
{
   public int Id { get; set; }
   public DateTime LogInTime { get; set; }
   public DateTime LogOutTime { get; set; }

   public string UserId { get; set; }
   public User User { get; set; }
}
```

Department.cs

```
namespace AuthSystem.Areas.Identity.Data.Entities;

public class Department
{
   public int Id { get; set; }
   public string DepartmentName { get; set; }
   public string Location { get; set; }

   public List<AccessCard> AccessCards { get; set; }
   public List<Post> Posts { get; set; }
}
```

Employee.cs

```
using System.ComponentModel.DataAnnotations.Schema;

namespace AuthSystem.Areas.Identity.Data.Entities;

public class Employee
{
   [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
   public string Id { get; set; }
```

```
    public string Inn { get; set; }
    public string AccessCard { get; set; }

    public int? PostId { get; set; }
    public Post Post { get; set; }

    public User User { get; set; }
    public EmployeeProfile EmployeeProfile { get; set; }
    public List<AccessCard> AccessCards { get; set; }
    public List<Vacation> Vacations { get; set; }
    public List<WorkHour> WorkHours { get; set; }
    public List<MilitaryRegistration> MilitaryRegistrations { get; set; }
}
```

EmployeeProfile.cs

```
namespace AuthSystem.Areas.Identity.Data.Entities;

public class EmployeeProfile
{
    public string Id { get; set; }
    public string? Sex { get; set; }
    public DateOnly? BirthDate { get; set; }
    public string? Image { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string MiddleName { get; set; }
    public string? HomeAddress { get; set; }
    public string? FamilyStatus { get; set; }
    public string? PassportNumber { get; set; }
    public DateOnly? PassportIssueDate { get; set; }
    public string? PassportIssueBy { get; set; }
    public DateOnly? PassportExpirationDate { get; set; }
    public Employee Employee { get; set; }
}
```

Post.cs

```
namespace AuthSystem.Areas.Identity.Data.Entities;

public class Post
{
    public int Id { get; set; }
```

```
    public string PostName { get; set; }

    public int DepartmentId { get; set; }
    public Department Department { get; set; }
    public List<Employee> Employees { get; set; }
}
```

User.cs

```
using Microsoft.AspNetCore.Identity;

namespace AuthSystem.Areas.Identity.Data.Entities;

// Add profile data for application users by adding properties to the ApplicationUser class
public class User : IdentityUser
{
   public Employee Employee { get; set; }
   public List<AccessLog> AccessLogs { get; set; }
}
```

Vacation.cs

```
namespace AuthSystem.Areas.Identity.Data.Entities;

public class Vacation
{
   public int Id { get; set; }
   public DateOnly StartDate { get; set; }
   public DateOnly EndDate { get; set; }
   public string Reason { get; set; }

   public string EmployeeId { get; set; }
   public Employee Employee { get; set; }
}
```

WorkHour.cs

```
namespace AuthSystem.Areas.Identity.Data.Entities;

public class WorkHour
{
   public int Id { get; set; }
   public DateOnly DateWorked { get; set; }
   public DateTime StartTime { get; set; }
   public DateTime EndTime { get; set; }

   public string EmployeeId { get; set; }
   public Employee Employee { get; set; }
}
```

AccessCard.cs

```
namespace StaffTracking.Data.Entities;

public class AccessCard
{
    public int Id { get; set; }
    public string Value { get; set; }

    public string EmployeeId { get; set; }
    public Employee Employee { get; set; }

    public int DepartmentId { get; set; }
    public Department Department { get; set; }
}
```

AccessLogConfiguration.cs

```
using AuthSystem.Areas.Identity.Data.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace AuthSystem.Areas.Identity.Data.Configurations;

public class AccessLogConfiguration : IEntityTypeConfiguration<AccessLog>
{
   public void Configure(EntityTypeBuilder<AccessLog> builder)
   {
       builder.HasKey(u => u.Id);

       builder.HasOne(p => p.User).WithMany(u => u.AccessLogs)
           .HasForeignKey(x => x.UserId);
   }
}
```

EmployeeConfiguration.cs

```
using AuthSystem.Areas.Identity.Data.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
```

```
namespace AuthSystem.Areas.Identity.Data.Configurations;

public class AccessLogConfiguration : IEntityTypeConfiguration<AccessLog>
{
   public void Configure(EntityTypeBuilder<AccessLog> builder)
   {
       builder.HasKey(u => u.Id);

       builder.HasOne(p => p.User).WithMany(u => u.AccessLogs)
           .HasForeignKey(x => x.UserId);
   }
}
```

PostConfiguration.cs

```
using AuthSystem.Areas.Identity.Data.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace AuthSystem.Areas.Identity.Data.Configurations;

public class PostConfiguration : IEntityTypeConfiguration<Post>
{
   public void Configure(EntityTypeBuilder<Post> builder)
   {
       builder.HasKey(u => u.Id);

       builder.HasOne(p => p.Department).WithMany(u => u.Posts)
           .HasForeignKey(x => x.DepartmentId);
   }
}
```

UserConfiguration.cs

```
using AuthSystem.Areas.Identity.Data.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace AuthSystem.Areas.Identity.Data.Configurations;

public class UserConfiguration : IEntityTypeConfiguration<User>
```

```
{
   public void Configure(EntityTypeBuilder<User> builder)
   {
       builder.HasKey(u => u.Id);
       builder.HasIndex(u => u.Email);

       builder.HasOne(p => p.Employee).WithOne(u => u.User)
           .HasForeignKey<User>(x => x.Id);
   }
}
```

AccessCardConfiguration.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using StaffTracking.Data.Entities;

namespace StaffTracking.Data.Configurations;

public class AccessCardConfiguration : IEntityTypeConfiguration<AccessCard>
{
    public void Configure(EntityTypeBuilder<AccessCard> builder)
    {
        builder.HasKey(x => x.Id);

        builder.HasOne(x => x.Employee).WithMany(x => x.AccessCards)
            .HasForeignKey(x => x.EmployeeId);

        builder.HasOne(x => x.Department).WithMany(x => x.AccessCards)
            .HasForeignKey(x => x.DepartmentId);
    }
}
```

ApplicationDbContext.cs

```
using AuthSystem.Areas.Identity.Data.Configurations;
using AuthSystem.Areas.Identity.Data.Entities;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;

namespace AuthSystem.Areas.Identity.Data.DbContext;
```

```
public class ApplicationDbContext : IdentityDbContext<User>
{
   public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
       : base(options)
   {
   }

   public DbSet<AccessLog> AccessLogs { get; set; }
   public DbSet<Department> Departments { get; set; }
   public DbSet<Employee> Employees { get; set; }
   public DbSet<EmployeeProfile> EmployeeProfiles { get; set; }
   public DbSet<MilitaryRegistration> MilitaryRegistrations { get; set; }
   public DbSet<Post> Posts { get; set; }
   public DbSet<Vacation> Vacations { get; set; }
   public DbSet<WorkHour> WorkHours { get; set; }
   public DbSet<AccessCard> AccessCards { get; set; }

   protected override void OnModelCreating(ModelBuilder builder)
   {
       base.OnModelCreating(builder);

       builder.ApplyConfiguration(new AccessLogConfiguration());
       builder.ApplyConfiguration(new EmployeeConfiguration());
       builder.ApplyConfiguration(new PostConfiguration());
       builder.ApplyConfiguration(new UserConfiguration());
       builder.ApplyConfiguration(new AccessCardConfiguration());
   }

   protected override void OnConfiguring(DbContextOptionsBuilder builder)
   {
       var envName = Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT");

       var config = new ConfigurationBuilder()
           .SetBasePath(AppDomain.CurrentDomain.BaseDirectory)
           .AddJsonFile("appsettings.json", false)
           .AddJsonFile($"appsettings.{envName}.json", false)
           .Build();

       var connection = config.GetConnectionString("DefaultConnection");
       builder.UseNpgsql(connection).UseSnakeCaseNamingConvention();
   }
}
```