Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Морозюка Андрія Володимировича

на здобуття ступеня вищої освіти Бакалавра

Система безконтактного контролю
відвідуваності студентів за допомогою RFID/NFC міток

Галузь знань 12 – Інформаційні технології
Спеціальність 123 – Комп'ютерна інженерія
Освітня програма Програмування та захист комп'ютерних систем і мереж

Шифр КРБКІ. 101005.21.01.05 ПЗ

Виконав студент 3 курсу група КІІс-21-1 Андрій МОРОЗЮК
Керівник доктор філософії Микола СТЕЦЮК
Нормоконтролер старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ

19 06 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 123 – Комп'ютерна інженерія

Освітня програма Програмування та захист комп'ютерних систем і мереж

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

15 лютого 2024 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Морозюку Андрію Володимировичу

1 Тема роботи Система безконтактного контролю відвідуваності студентів за допомогою RFID/NFC міток

Керівник роботи

Затверджено наказом ректора університету від 15 лютого 2024 № 8

2 Строк подання студентом кваліфікаційної роботи на кафедру

3 Вихідні дані до роботи пристрій створений на основі плати Raspberry Pi Pico W та зчитувача RFID міток RC522, що запрограмований на мові Python для зчитування міток, та взаємодії з сервером.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області, Порівняння існуючих рішень, Вибір підходу, Постановка задачі, Розробка алгоритмів, Вибір плати Raspberry Pi Pico, Вибір модуля RFID RC522, Реалізація системи контролю відвідуваності, Збір та налаштування модуля, Розробка веб-сервера, Розробка системи адміністрування.

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Схема підключення пристрою для зчитування Rfid тегів, діаграма архітектури проєкту, алгоритм роботи системи

6 Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Мостовий С.В., старший викладач кафедри кібербезпеки		

7 Дата видачі завдання 16 лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Вибір і затвердження теми кваліфікаційної роботи	Січень-Лютий	виконав
Ознайомлення з предметною областю	Лютий	виконав
Дослідження існуючих рішень	Лютий	виконав
Постановка задачі	Березень	виконав
Визначення загальних принципів рішення задачі	Березень	виконав
Деталізація принципів рішення задачі	Квітень	виконав
Розробка проєктних рішень	Квітень	виконав
Апробація проєктних рішень	Травень	виконав
Оформлення пояснювальної записки згідно вимог	Травень	виконав
Оформлення графічної частини	Червень	виконав
Захист КР	Червень	виконав

Студент _____ Андрій МОРОЗЮК

Керівник кваліфікаційної роботи _____ Микола СТЕЦЮК

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КРБКІ.101005.21.01.06 ПЗ	Пояснювальна записка	64		
			Графічні матеріали			
2		КРБКІ.101005.21.01.06 Е1	Будова Raspberry Pi Pico	1		
			та підключення RC522			
3		КРБКІ.101005.21.01.06 Е2	Алгоритм роботи готової	1		
			системи			
4		КРБКІ.101005.21.01.06 Е3	Детальна будова Rapberry	1		
			PI та шлефу піделючення			
5		КРБКІ.101005.21.01.06 Е8	Діаграма архітектури	1		

КРБКІ.101005.21.01.05 ВП

Зм	Арк	№ докум	Підпис	Дата
Розробив		Моровяок А.В		06.06
Перевір.		Степок М.В		20.06.24
Н. контр.		Мостовий С.В		20.06.24
Затв.		Ключ Ю.П		20.06.24

Система безконтактного контролю відвідуваності студентів за допомогою RFID/NFC міток. Відомість проекту

Літера	Аркуш	Аркушів
У	1	1

ХНУ, КПс-21-1

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Система безконтактного контролю відвідуваності студентів за допомогою RFID/NFC міток».

Автор роботи: Морозюк Андрій Володимирович.

Керівник роботи: Стецюк Микола Васильович.

Пояснювальна записка: 64 с., 25 рис., 3 дод., 40 джерел.

Графічна частина: 3 креслення.

МІКРОПЛАТА, СИСТЕМА ВІДСТЕЖЕННЯ, МОНІТОРИНГ, БАЗА ДАНИХ, ВЕБ-ІНТЕРФЕЙС.

Метою моєї кваліфікаційної роботи було розробити інтегровану систему відстеження присутності студентів в університеті за допомогою технології RFID та її програмування та налаштування для забезпечення необхідного функціоналу та зручності роботи.

Об'єктом дослідження є система контролю відвідуваності.

Предметом дослідження є оцінка роботи систем контролю відвідуваності та розробка власної системи на базі мікроплати, що використовує зчитувач міток.

Під час проведення даного дослідження був використаний метод систематичного огляду літератури для вивчення і аналізу предметної області даного дослідження з текстових джерел інформації та для розробки власного проєкту.

06.06

ЗМІСТ

Зм.	Арк.	№докум.	Підпис	Дата	КРБКІ. 101005.21.01.05 ПЗ		
Виконав		Морозюк А.В.		06.06	Система безконтактного контролю відвідуваності студентів за допомогою RFID/NFC міток Пояснювальна записка	Літера	Аркуш	Аркушів
Перевір.		Стецюк М.В.		13.06.24		Н	3	64
Н.контр.		Мостовий С.В.		20.06.24		ХНУ, КПс-21-1		
Затвер.		Кльоц Ю.П.		20.06.24				

ВСТУП

У сучасному освітньому середовищі виникає потреба в більш ефективних та інноваційних методах обліку та моніторингу студентів. Традиційні способи, такі як ручне ведення журналу відвідувань, можуть бути трудомісткими, схильними до помилок і не надають глибокого розуміння поведінки та успішності студентів, а також недостатньо ефективно виконують основну функцію обліку.

Для покращення організації процесу обліку, а також зменшення витрат людських ресурсів, доцільним є використання сучасного обладнання та технологій для автоматизації. Це дозволить оптимізувати процеси та забезпечити більш точний та ефективний облік.

При виборі технологій для авторизації студентів у системі можна звернутися до вже перевірених практик. Наприклад, теоретично систему можна реалізувати за допомогою камер зі штучним інтелектом, які розпізнаватимуть обличчя студентів та автоматично вести облік. Це зменшить час очікування та усуне необхідність використовувати студентський квиток. Альтернативою фізичній картці в Україні є застосунок «Дія», що дозволяє спростити процес пошуку картки, але залишає певні етапи валідації, які можуть бути вдосконалені.

Вищезгадані камери зі штучним інтелектом дозволять усунути участь студентів та вахтерів у процесі обліку, але мають недоліки, такі як висока вартість, складність впровадження та потенційні проблеми з конфіденційністю.

Також можна розглянути технологію Near Field Communication (NFC), яка довела свою ефективність у сфері, що потребує високого рівня безпеки проведення операцій. NFC можна використовувати для швидкої та безпечної ідентифікації студентів, забезпечуючи високий рівень захисту персональних даних [1-3].

Ще одним аргументом на користь використання NFC є практика її застосування для обліку співробітників у багатьох компаніях, які спеціалізуються на безпеці. Ця технологія знаходить широке застосування для забезпечення доступу до об’єктів, контролю за присутністю працівників та моніторингу їх

діяльності. NFC-технологія може бути легко інтегрована у мобільні пристрої, що робить її доступною та зручною для використання студентами та персоналом освітніх установ.

Крім того, впровадження подібних систем автоматизації може допомогти знизити рівень шахрайства та підробки документів, що є важливим аспектом у сучасному світі. Використання RFID та NFC-технологій дозволяє забезпечити надійний контроль за відвідуваністю студентів та зберігати дані у захищеній формі.

Сучасні технології обліку та моніторингу студентів також відкривають нові можливості для аналізу та покращення навчального процесу. Збір та аналіз даних про відвідуваність, активність на заняттях та інші аспекти навчання дозволяють виявити проблемні зони та своєчасно вживати заходів для їх вирішення .

1 ТЕОРИТИЧНІ ТА ПРАКТИЧНІ ОСНОВИ ПОСТАВЛЕНОГО ЗАВДАННЯ

1.1 Аналіз предметної області і виявлення наявних проблем та завдань

У сучасному світі вищі навчальні заклади по всьому світу стикаються з різноманітними викликами. Особливо важливими є завдання, пов’язані з потребою вдосконалення систем обліку присутності студентів та моніторингу їх академічного прогресу. Це важливе завдання, оскільки воно впливає на ефективність навчального процесу та здатність університету виконувати свою основну місію. Інноваційні технології, такі як хмарні обчислення, штучний інтелект та блокчейн, пропонують рішення, які можуть оптимізувати ці процеси та значно підвищити їх ефективність.

На даний момент система обліку студентів, яка використовується в нашій установі, великою мірою залежить від фізичних студентських квитків. Ці квитки є обов'язковими для студентів, які повинні мати їх при собі, щоб контролюючий персонал міг перевірити їх при необхідності. Це, в свою чергу, вимагає наявності спеціалізованого персоналу, який би міг проводити такі перевірки, що може створити додаткові витрати для установи. З одного боку, цей процес дозволяє зберегти певний рівень контролю за присутністю студентів, однак, з іншого боку, він має численні недоліки.

Однією з головних проблем, що стосуються системи обліку присутності студентів, є її неефективність. Традиційний метод обліку, що використовує студентські квитки та електронні чи паперові журнали, часто є недостатньо точним, вимагає значного обсягу ручної роботи, піддається підробці та витратний у часі. Крім того, така система не завжди забезпечує швидкий та зручний доступ до даних про присутність студентів для адміністраторів та викладачів, що ускладнює процес контролю та оцінювання навчальної роботи студентів. Часто трапляється, що дані про присутність студентів записуються з помилками, які можуть залишатися непоміченими протягом тривалого часу, що впливає на достовірність загальної картини присутності студентів на заняттях.

У цьому контексті нашим завданням є дослідити ці та інші можливі технологічні рішення, оцінити їх потенційну ефективність та придатність для покращення системи обліку присутності студентів та моніторингу їх академічного прогресу в університетах, а також розробити рекомендації щодо їх впровадження. Необхідно врахувати, що впровадження нових технологій повинно відбуватися поступово, з детальним аналізом кожного етапу, щоб уникнути можливих помилок та забезпечити максимальну ефективність використання нових систем.

Технології, такі як хмарні обчислення, можуть значно спростити процес зберігання та обробки даних про присутність студентів. Використання хмарних платформ дозволить забезпечити доступ до даних у будь-який час і з будь-якого місця, що значно підвищить зручність використання системи для всіх учасників навчального процесу. Крім того, хмарні обчислення дозволяють обробляти великі обсяги даних, що є важливим фактором для великих навчальних закладів з великою кількістю студентів.

Штучний інтелект може бути використаний для аналізу даних про присутність та академічний прогрес студентів. Сучасні алгоритми можуть виявляти закономірності та надавати рекомендації щодо покращення навчального процесу. Наприклад, на основі аналізу даних штучний інтелект може виявити студентів, які потребують додаткової допомоги, або визначити курси, які викликають найбільше труднощів у студентів.

Блокчейн-технології забезпечують високий рівень безпеки та прозорості даних. Використання блокчейну для зберігання даних про присутність студентів може запобігти підробкам та забезпечити надійність облікових систем. Кожна зміна у базі даних буде відстежуватись та фіксуватись, що дозволить уникнути несанкціонованих змін та забезпечить повну прозорість процесу обліку.

Впровадження цих технологій вимагає значних зусиль та ресурсів, однак, потенційні переваги перевищують можливі витрати. Підвищення точності обліку присутності студентів, зменшення обсягу ручної роботи та забезпечення швидкого доступу до даних – все це сприятиме покращенню якості навчального процесу та підвищенню рівня задоволеності студентів та викладачів. Окрім того,

використання сучасних технологій дозволить навчальному закладу залишатися конкурентоспроможним та відповідати вимогам сучасного освітнього середовища.

Таким чином, наше дослідження спрямоване на детальний аналіз існуючих проблем у системі обліку присутності студентів та моніторингу їх академічного прогресу, а також на розробку рекомендацій щодо впровадження сучасних технологічних рішень для покращення цих процесів. Ми сподіваємося, що результати нашого дослідження стануть корисними для університетів та допоможуть підвищити ефективність їх роботи.

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

У сучасних університетах часто використовується традиційний метод обліку відвідуваності студентів, що базується на студентських квитках і паперових журналах. Хоча цей підхід є звичним і простим у використанні, він має багато недоліків. Викладачі або адміністратори вручну фіксують присутність студентів під час занять, використовуючи паперові журнали, де студентські квитки служать засобом ідентифікації особи студента. Ця система, яка покладається на фізичні студентські квитки і потребує участі спеціалізованого персоналу для проведення перевірок, має серйозні обмеження.

Перший значний недолік такої системи – її низька ефективність. Цей метод вимагає великих затрат часу і ресурсів, які могли б бути використані більш продуктивно. Ручна праця підвищує ризик помилок через людський фактор, що знижує точність обліку відвідуваності.

Другий недолік – складність організації процесу. Студенти можуть не завжди мати при собі свої студентські квитки, що ускладнює процес обліку відвідуваності та підтвердження їхньої ідентичності в разі потреби.

Третій недолік – високі витрати на утримання спеціалізованого персоналу. Необхідність наявності персоналу для перевірки студентських квитків може бути

неефективним використанням ресурсів, особливо у великих навчальних закладах, де кількість студентів може бути дуже великою.

Через ці проблеми стає очевидним, що існує велика потреба в пошуку та впровадженні більш ефективних та інноваційних методів обліку та моніторингу студентів, які б допомогли вирішити вказані вище питання.

Одним з таких інноваційних рішень є використання технології NFC (Near Field Communication) у вищих навчальних закладах. Ця технологія, яка вже успішно застосовується у багатьох компаніях, пропонує нові можливості для управління навчальним процесом. Завдяки високій швидкості передачі даних та зручності у використанні, NFC може значно полегшити процес реєстрації студентів та моніторингу їхнього академічного прогресу [1-3].

NFC технологія дозволяє автоматизувати процес обліку відвідуваності, забезпечуючи швидкий та точний запис даних без необхідності ручного втручання. Це значно знижує ймовірність помилок і підвищує ефективність процесу. Студенти можуть просто підносити свої смартфони або NFC-картки до спеціальних зчитувачів на вході до аудиторій, що дозволяє миттєво фіксувати їх присутність.

Це також знижує навантаження на адміністративний персонал, оскільки немає потреби в ручному обліку і перевірках. Дані автоматично збираються і зберігаються в електронній системі, до якої можуть мати доступ як адміністратори, так і викладачі для швидкого і зручного отримання інформації про присутність студентів.

Таким чином, впровадження NFC технологій у вищих навчальних закладах не лише підвищує ефективність обліку відвідуваності студентів, але й покращує загальний навчальний процес, дозволяючи більш точно і оперативно моніторити академічний прогрес студентів та вчасно реагувати на будь-які проблеми.

Використання NFC надає ряд значущих переваг:

Зручність та швидкість: NFC дозволяє студентам миттєво реєструвати свою присутність, використовуючи для цього зручні і знайомі інструменти, такі як їхній смартфон або карточку.

– Безпека: NFC-технологія включає можливості шифрування та унікальність NFC-тегів, що робить систему відносно безпечною від несанкціонованого доступу та підробки.

– Інтеграція з існуючими системами: NFC можна легко інтегрувати у вже наявну систему обліку студентів, якщо така існує в навчальному закладі.

З іншого боку, цей підхід може мати деякі потенційні недоліки:

– Залежність від технічного обслуговування: як і будь-яка технологічна система, NFC-технологія може потребувати регулярного технічного обслуговування та оновлення, що може вимагати додаткових витрат часу та ресурсів.

Також окремою темою дослідження була система обліку що використовується на підприємствах, зокрема на офісі компанії Clarity AG [5]. Дана система включає в себе комплекс з контролерів, системи обміну даними між контролерами а також низка програмного забезпечення.

Важливим компонентом системи обліку є пристрій RS-485/232 Card Systems ТКП-32-04/2 [7]. Цей пристрій служить для обміну даними між контролерами КСКД 3 та управляючим комп'ютером. Основна функція ТКП-32-04/2 - узгоджувати різні інтерфейси обладнання, для забезпечення гладкої комунікації між контролерами та сервером, які використовують інтерфейси RS-485 та RS-232 відповідно.

Система обліку працює за допомогою контролерів КСКД2-3К та КСКД2-12К [7][8], які є універсальними і призначені для керування однією або двома точками доступу. Вони можуть керувати дверима, турнікетами, шлагбаумами та іншими точками доступу. Доступ контролюється за допомогою особистих ідентифікаторів - безконтактних карток або карток з штрих-кодом.

Рисунок 1.1 - Фото КСКД-4Е

Контролери з моделями КСКД2-3К і КСКД2-12К розроблені на базі однакової електричної схеми і володіють подібними конструктивними особливостями. Однак, ці дві моделі мають відмінності, які полягають в обсязі енергонезалежної пам'яті, що використовується для зберігання величезного обсягу інформації. КСКД2-3К має вмістимість до 3000 записів, що дозволяє зберігати достатньо інформації для базових потреб. З іншого боку, КСКД2-12К може зберігати до 12000 записів, що робить його ідеальним для більш вимогливих застосувань, де потрібно обробляти великі обсяги даних.

Система обліку STOP-net включає в себе спеціалізоване програмне забезпечення під назвою "АСКД" і додаткову програму "Менеджер пропусків" [9]. Ці програми розроблені з метою збору, обробки та візуалізації даних, які генеруються системою контролю доступу "STOP-Net".

Програма АСКД надає можливість реалізувати наглядний контроль за роботою контролерів серій КСКД2-3К(12К) та КСКД3. Ці контролери відіграють ключову роль в системі, оскільки їх основне призначення - керування точками

доступу. Точки доступу можуть бути різноманітними: від дверей офісних приміщень до турнікетів на входах в метро, шлагбаумів на парковках та інші. Контролери надають доступ до цих точок через персональні ідентифікатори, які можуть бути в форматі безконтактних карток або карток із штрих-кодом [9].

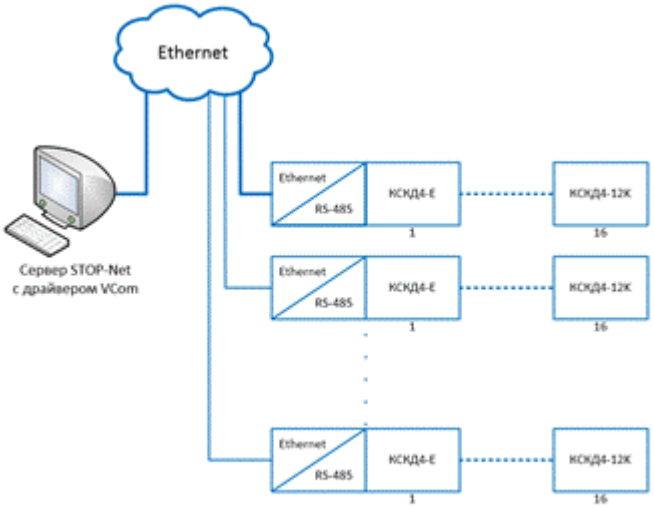


Рисунок 1.2 - Схема підключення контролерів до мережі

Програмне забезпечення АСКД є важливим елементом системи контролю доступу "STOP-Net". Цей програмний продукт був спеціально розроблений для аналізу та візуалізації даних, які генеруються пристроями системи, включаючи контролери моделей КСКД2-3К та КСКД2-12К.

У свою чергу, програма "Менеджер пропусків" була створена з метою автоматизації різноманітних задач, які випливають перед адміністратором системи контролю доступу. Специфічні функції цієї програми включають:

– Управління базою даних пропусків, яка ділиться на три основні категорії: постійні, тимчасові та одноразові. Це охоплює ввод, редагування та видалення інформації про співробітників, тимчасових робітників, клієнтів та відвідувачів, включаючи робочі дані, особисту інформацію, права доступу, фотографії.

– Блокування пропусків, яке передбачає тимчасове видалення пропуска

зі списку активних пропусків з будь-яких причин (наприклад, втрата пропуска або порушення правил контролю доступу).

– Тимчасові зміни у правах доступу пропусків, що дозволяють поширити права або встановити заборону на певний час (наприклад, вихід на робоче місце у вихідний день або заборона доступу в певних контрольних точках).

– Управління довідниками, такими як "Структура підприємства", "Організації клієнтів", "Контрольні точки", "Користувачі", "Вихідні та святкові дні", "Причини блокувань", "Попередження", "Контрольні зони".

– Робота з автономними контролерами, що включає завантаження карток та графіків доступу в автономні термінали, синхронізацію інформації з автономними терміналами.

Система обліку робочого часу "STOP-net" є продуктом високих технологій, що впроваджує інноваційні рішення для точного контролю робочого часу та надійного контролю доступу.

При вивченні системи обліку робочого часу та контролю доступу STOP-net, виявлено низку переваг та недоліків, які важливо врахувати при розгляді її впровадження та використанні.

До переваг системи відносяться:

– Модульність та гнучкість: Система дозволяє індивідуально проектувати рішення для різних об’єктів, враховуючи їх специфіку.

– Надійність: Висока стійкість до технічних збоїв та самодіагностика.

– Здатність до модернізації: Можливість використання вже встановленого обладнання.

– Інтегрованість: Створення багатофункціональних додатків з елементами сторонніх систем.

– Автоматизація: Контроль доступу та облік робочого часу без потреби в постійному втручанні людини.

– Безпека: Зменшення втрат від протиправних дій.

– Продуктивність: Підвищення дисципліни та ефективності роботи

співробітників.

Недоліки системи включають:

– Вартість: Початкові витрати на впровадження та налаштування системи можуть бути значними.

– Технічне обслуговування: Потреба в кваліфікованих фахівцях для обслуговування та ремонту системи.

– Залежність від електроніки: Ризики, пов’язані з відмовою обладнання або програмного забезпечення.

– Комплексність: Можливі складнощі у використанні системи для не-технічних співробітників.

– Приватність: Потенційні питання збереження та захисту персональних даних.

Контролери КСКД2-3К та КСКД2-12К, розроблені на основі однакової схеми, відрізняються обсягом енергонезалежної пам'яті. Система обліку STOP-net включає програмне забезпечення "АСКД" та "Менеджер пропусків" для збору, обробки та візуалізації даних. Переваги системи включають модульність, надійність, можливість модернізації, інтегрованість, автоматизацію, безпеку та продуктивність. Недоліки включають високу вартість, потребу в технічному обслуговуванні, залежність від електроніки, складність використання та питання приватності.

1.3 Визначення підходу до вирішення поставленої задачі

Мета:

– дослідити можливі технологічні рішення для покращення системи контролю присутності студентів;

– оцінити потенційну ефективність цих рішень для відстеження академічного прогресу студентів;

– розробити рекомендації щодо впровадження обраних технологічних

рішень.

Для вирішення задачі обліку студентів доцільно використовувати комплекс мікросервісів, який базується на ідентифікації студентів за допомогою NFC міток. Основними компонентами такого комплексу є:

– Розробка мікросервісу, який забезпечує зчитування NFC міток студентів та їх ідентифікацію в системі. Це дозволить швидко та точно розпізнавати студентів, спрощуючи процес обліку.

– Створення бази даних для зберігання всієї необхідної інформації про студентів, включаючи їх особисті дані та NFC ідентифікатори. Ця база даних забезпечить адміністраторам та викладачам швидкий доступ до необхідної інформації про студентів.

– Розробка веб-сервера, який забезпечуватиме зручну взаємодію з базою даних і дозволить адміністраторам та викладачам ефективно працювати з системою. Цей веб-сервер матиме інтуїтивно зрозумілий інтерфейс, що дозволить легко переглядати та оновлювати дані студентів.

Цей підхід забезпечить повну автоматизацію процесу обліку студентів. Він спростить доступ до інформації про студентів і полегшить роботу адміністраторів та викладачів, дозволяючи їм швидко та зручно взаємодіяти з системою.

Однією з головних переваг такого рішення є його простота розробки та використання сучасних технологій. Використання комплексу мікросервісів робить розробку рішення більш доступною і ефективною, оскільки кожен сервіс може бути розроблений, тестований та вдосконалений незалежно від інших. Такий підхід дозволяє залучати різних розробників з різними навичками та спеціалізаціями до розробки і підтримки окремих сервісів.

Мікросервісна архітектура забезпечує гнучкість та масштабованість системи. Кожен сервіс може бути горизонтально масштабований, що дозволяє розподілити навантаження та забезпечити високу доступність. Наприклад, якщо підвищується навантаження на сервіс ідентифікації студентів, можна додати додаткові "інстанси" цього сервісу, щоб забезпечити швидкість та надійність його роботи.

Крім того, мікросервісна архітектура дозволяє зручно покращувати та доповнювати функції кожного з сервісів. Оскільки кожен сервіс має свій власний код та окрему команду розробників, зміни у функціонал можна вносити безпечно та без впливу на роботу інших сервісів. Це позитивно впливає на швидкість розробки та впровадження нових функцій, а також на забезпечення стабільності системи.

Таким чином, використання комплексу мікросервісів для обліку студентів є перспективним та ефективним рішенням. Воно дозволяє використовувати сучасні технології, забезпечує гнучкість та масштабованість системи, а також зручну роботу з покращенням та доповненням функціоналу кожного з сервісів. Застосування такого підходу допоможе спростити процес обліку студентів та забезпечити ефективну взаємодію з системою.

Мікросервісна архітектура відкриває безліч можливостей для вибору технологій у кожному елементі комплексу. Взаємодія сервісів через програмований універсальний інтерфейс (API) дозволяє обирати найкращі технології для конкретних завдань.

Наприклад, для реалізації конкретного сервісу можна використовувати мову програмування або технологію, яка найбільше підходить для вирішення поставленої проблеми. Це може бути мова Python для аналітичного сервісу, Java для створення веб-служби або Node.js для мікросервісів, що потребують високої продуктивності.

Для забезпечення комунікації між сервісами можна використовувати різні протоколи інтерфейсу, такі як REST або GraphQL, залежно від вимог до швидкодії, гнучкості та безпеки.Крім того, для зберігання та обробки даних можуть використовуватися різні технології баз даних, такі як реляційні (наприклад, MySQL або PostgreSQL) або NoSQL (наприклад, MongoDB або Cassandra), залежно від обсягу та характеру даних [20-22].

Такий підхід дозволяє забезпечити оптимальну продуктивність, масштабованість та надійність кожного елементу мікросервісної системи, а також спрощує розробку, розгортання та підтримку.

У якості "заліза" для системи обліку відвідуваності студентів можна використати безліч рішень, як програмованих, так і готових. Для самостійної розробки можуть підійти рішення рідерів RFID та NFC завдяки їхній доступності, широкому розповсюдженню та простоті інтеграції з різними платформами.

Такі рідери зазвичай мають детальну технічну документацію, приклади коду та активні спільноти розробників, що значно спрощує процес їх інтеграції та налаштування. Вони підтримують стандартні інтерфейси підключення, як-от SPI, I2C, UART або USB, що дозволяє легко підключити їх до популярних мікроконтролерів та одноплатних комп'ютерів, таких як Raspberry Pi або Arduino. Крім того, існує безліч готових бібліотек та драйверів для різних мов програмування, що дозволяє швидко розпочати роботу з рідерами без потреби в глибоких знаннях електроніки або програмування.

Одним з таких рішень є PN532. Це NFC модуль, який базується на чіпі PN532 і використовується для безконтактного зв’язку на частоті 13.56MHz. Модуль оснащений вбудованою антеною, тому зовнішня антена не потрібна, що спрощує його використання. Він підтримує інтерфейси SPI, I2C та UART для зв’язку, що робить його сумісним з багатьма мікроконтролерами, включаючи Raspberry Pi та Arduino.

PN532 підтримує різні режими роботи, включаючи читання та запис RFID-карт, емуляцію карт, а також peer-to-peer зв'язок. Він сумісний з широким спектром RFID карток, таких як Mifare 1K, 4K, Ultralight, DESFire, ISO/IEC 14443-4, FeliCa та Innovision Jewel. Це дозволяє використовувати модуль у різних застосуваннях, від систем доступу до мобільних платіжних систем та обміну даними.

Модуль має вбудовану схему підключення, яка дозволяє легко змінювати режими зв’язку за допомогою DIP перемикачів. Він також оснащений рівневим перетворювачем, що дозволяє використовувати його з живленням від 3.3V до 5V, забезпечуючи гнучкість у виборі живлення. [11]

Рисунок 1.3 - Модуль PN532

RC522 - це RFID/NFC модуль, який базується на контролері MFRC522 від NXP Semiconductors і працює на частоті 13.56MHz. Модуль підтримує комунікацію через I2C, SPI та UART і зазвичай поставляється з RFID карткою та брелоком. Він часто використовується в системах відвідуваності та інших додатках для ідентифікації осіб/об’єктів. Часто використовується з Raspberry Pi Pico.

Модуль RC522 відомий своєю низькою вартістю і простотою використання, що робить його популярним вибором серед розробників для проектів DIY та навчальних цілей. RC522 підтримує ISO/IEC 14443 A/MIFARE та NTAG протоколи, що дозволяє працювати з різними типами RFID карток і тегів. Основні характеристики модуля включають вбудовану антену для безконтактного зчитування на відстані до 5 см, підтримку криптографічних протоколів для захищеної передачі даних, а також широкий діапазон робочої напруги від 2.5V до 3.3V. [12]

Завдяки своїй доступності і простоті інтеграції, RC522 часто використовується з мікроконтролерами, такими як Arduino та Raspberry Pi, для створення систем доступу, відвідуваності та інших проектів, які вимагають безконтактної ідентифікації. Інтерфейси SPI та I2C забезпечують швидкий і

надійний обмін даними між модулем і контролером, що дозволяє легко реалізувати навіть складні системи обробки даних.

Рисунок 1.4 - Модуль RC522

Прикладом готової системи може слугувати NXP EdgeVerse™ NFC Readers – це серія високопродуктивних NFC-читачів, які можуть читати та записувати картки та мітки, взаємодіяти з NFC-телефонами та забезпечувати комунікацію між пристроями. Вони є частиною платформи EdgeVerse, яка побудована на основі масштабованості, енергоефективності, безпеки, машинного навчання та зв'язку.

Серія включає такі котролери:

– PN7220 - EMV L1 сумісний NFC контролер з інтерфейсом NCI, що підтримує додатки EMV та NFC Forum. Це рішення забезпечує безпечні та надійні безконтактні транзакції, що є критично важливими для платіжних систем.

– PN7462 - Програмований NFC мікроконтролер з 180K Flash пам'яттю, апаратними прискорювачами для симетричного та асиметричного криптографічного захисту, а також безпечним сховищем ключів. Цей контролер підходить для широкого спектру застосувань, включаючи безпеку доступу та ідентифікацію.

– PN7160 - NFC контролер з інтегрованим прошивкою, що підтримує повний NFC. Він призначений для забезпечення повної функціональності NFC,

включаючи читання, запис та емуляцію карт, що робить його універсальним рішенням для різних NFC-додатків.

Ці контролери є частиною платформи NXP EdgeVerse, яка забезпечує високий рівень безпеки та ефективності для розробників та користувачів. Платформа надає можливості для інтеграції машинного навчання та безпечного зберігання даних, що робить її ідеальною для використання в сучасних розумних пристроях і системах.

Також можна навести як приклад програмовані NFC контролери з ARM Cortex-M0 – це мікроконтролери, які інтегрують NFC-фронтенд з програмованим мікроконтролером ARM Cortex-M0. Вони забезпечують високу продуктивність та низьке енергоспоживання, підтримуючи різні режими NFC, такі як читання/запис міток, P2P та емуляцію карток типу ISO/IEC 14443-A і B.

Приклади:

– PN736X - Це сімейство мікроконтролерів з 32-бітним ARM Cortex-M0, яке пропонує 160/80 кБ Flash пам’яті, 12 кБ SRAM та 4 кБ EEPROM. Вони призначені для забезпечення надійної та ефективної роботи з NFC.

– PN7462 - Це сімейство мікроконтролерів, яке додатково оснащене інтерфейсом ISO/IEC 7816-3&4 UART для контролю доступу. Вони надають можливість створення безпечних систем доступу та ідентифікації, завдяки вбудованим криптографічним можливостям.

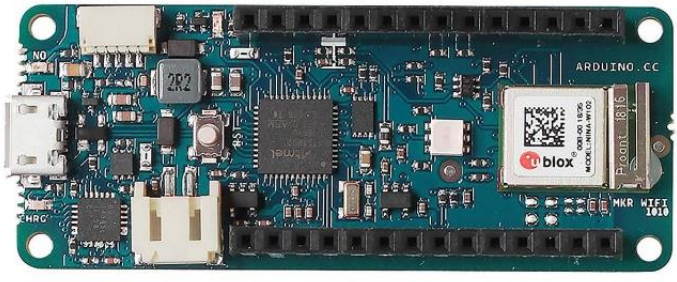


Рисунок 1.5 - Мікрокомп’ютер ARM Cortex M0

Ці мікроконтролери є ідеальним вибором для інтеграції в системи, що вимагають високої продуктивності при низькому енергоспоживанні, а також забезпечують гнучкість у розробці різноманітних NFC-додатків.

Raspberry Pi Pico, як мікроконтролер, є важливим інструментом для розробки електронних пристроїв та систем. Він оснащений мікроконтролером RP2040, розробленим компанією Raspberry Pi для забезпечення високої продуктивності та гнучкості використання. Завдяки двоядерному процесору ARM Cortex-M0+ та 264 кБ оперативної пам'яті SRAM, Pico здатний вирішувати широкий спектр завдань, від найпростіших до досить складних.

Інтеграція NFC модуля з Raspberry Pi Pico потребує ретельного налаштування підключення та програмування. NFC модуль, наприклад, PN532, може бути під'єднаний до Pico через I2C інтерфейс, що дозволяє виконувати операції читання та запису NFC міток, а також реалізовувати P2P комунікації та емуляцію карток. Після фізичного підключення модуля до Pico, потрібно налаштувати програмне забезпечення для забезпечення коректної взаємодії між обома пристроями.

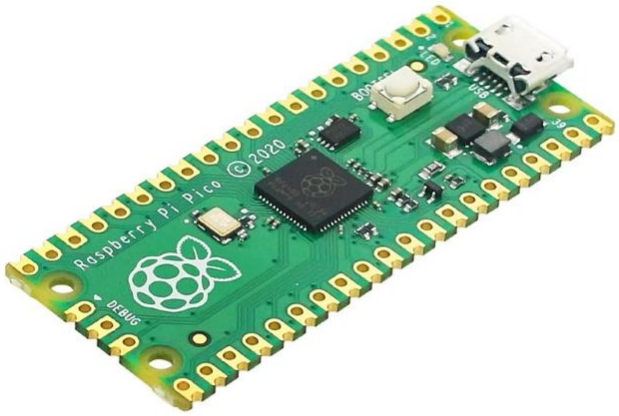


Рисунок 1.6 - Rasberry Pi Pico

Застосування NFC модуля з Raspberry Pi Pico відкриває безліч можливостей для створення інноваційних проектів. Це включає розробку систем безконтактної авторизації, розумних домашніх пристроїв та інших застосувань, де

використовуються безконтактні технології. У поєднанні з NFC модулем, Raspberry Pi Pico стає потужним інструментом для розробників, які бажають інтегрувати сучасні технології у свої проекти, що дозволяє створювати більш ефективні та функціональні рішення.

Arduino — це відкрита апаратна платформа, яка широко застосовується для створення електронних проектів різної складності. Вона заснована на мікроконтролері Atmel AVR і має вбудовані засоби для легкого програмування та інтеграції з іншими пристроями. Arduino дає змогу користувачам створювати інтерактивні об'єкти або взаємодіяти з програмним забезпеченням на комп'ютері, що робить її ідеальною для освітніх цілей, хобі та прототипування продуктів.

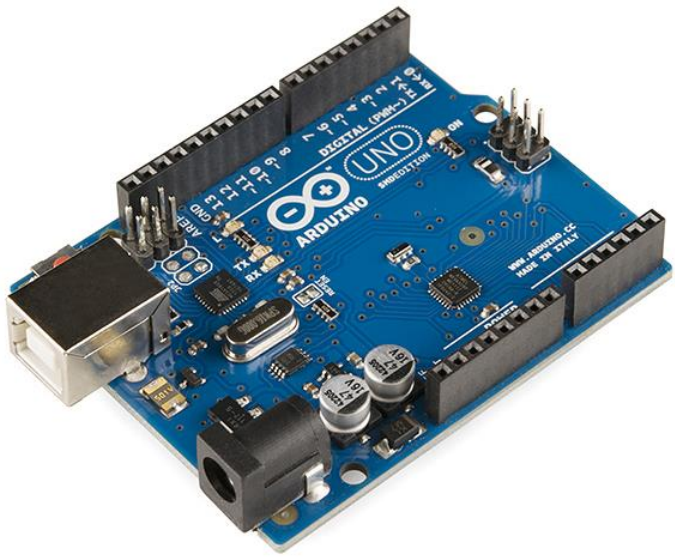


Рисунок 1.7 - Arduino

Плати Arduino зазвичай обладнані лінійним стабілізатором напруги, що забезпечує стабільне живлення компонентів, а також кварцовим резонатором, який гарантує точне тактування мікроконтролера. Важливою особливістю більшості плат Arduino є наявність завантажувача (bootloader), який дозволяє програмувати мікроконтролер без використання зовнішнього програматора, що значно полегшує процес розробки та тестування проектів.

Arduino підтримує широкий спектр цифрових та аналогових входів/виходів, що дозволяє зчитувати дані з різноманітних датчиків та керувати зовнішніми пристроями. Наприклад, цифрові входи/виходи можуть бути використані для зчитування стану кнопок або перемикачів, тоді як аналогові входи дозволяють вимірювати напругу з різних аналогових датчиків, таких як температурні сенсори або потенціометри.

Крім того, багато плат Arduino мають піни, що підтримують генерацію ШІМ (широтно-імпульсної модуляції) сигналів. Це дозволяє керувати інтенсивністю світіння світлодіодів, швидкістю обертання моторів та іншими пристроями, які потребують регулювання вихідної напруги. Завдяки цьому Arduino може використовуватись для створення різноманітних проектів, від простих освітлювальних систем до складних роботизованих пристроїв.

Загалом, платформа Arduino надає користувачам можливість легко створювати інтерактивні об'єкти та системи, забезпечуючи гнучкість у використанні та простоту інтеграції з іншими компонентами. Це робить її ідеальною для як навчальних цілей, так і для хобі та професійного прототипування.

Одним з найпопулярніших NFC модулів для роботи з Arduino є RC522. Цей модуль використовує SPI інтерфейс для зв’язку з мікроконтролером і дозволяє зчитувати UID міток NFC, а також здійснювати читання та запис даних на них. Для підключення RC522 до плати Arduino необхідно з'єднати відповідні піни модуля з відповідними контактами на платі Arduino. Після фізичного підключення модулю до Arduino, наступним етапом є програмування.

Для програмування зазвичай використовують спеціалізовані бібліотеки, такі як MFRC522 для RC522 або відповідні бібліотеки для інших модулів, як PN532. Ці бібліотеки значно спрощують процес інтеграції NFC функціональності у ваші проекти, надаючи готові методи для ініціалізації модуля, налаштування комунікаційних параметрів та розробки логіки зчитування та запису міток. Програмування включає в себе створення коду для ініціалізації модуля,

налаштування параметрів зв’язку та розробки алгоритмів для взаємодії з NFC мітками.

Вибір відповідного NFC модуля залежить від потреб та вимог конкретного проекту, а також від особистих вподобань розробників, що працюватимуть над його реалізацією. Якщо проект вимагає більшої гнучкості та можливості програмування, модулі, сумісні з платформами Raspberry Pi Pico або Arduino, можуть бути більш придатними. Вони забезпечують високу ступінь налаштування та можливість глибокої інтеграції з іншими компонентами проекту.

З іншого боку, для швидкого впровадження та стандартизованих рішень, оптимальним вибором можуть бути готові системи, такі як NXP EdgeVerse™ NFC Readers. Ці системи пропонують комплексні рішення для роботи з NFC, включаючи підтримку читання та запису карток, взаємодію з NFC-телефонами та забезпечення комунікації між пристроями. Платформа EdgeVerse побудована на основі принципів масштабованості, енергоефективності, безпеки, машинного навчання та зв’язку, що робить її ідеальною для різноманітних застосувань.

Таким чином, вибір між використанням гнучких програмованих модулів і готових систем залежить від специфіки проекту, ресурсів та вимог до функціональності і швидкості впровадження. В обох випадках розробники мають можливість створювати інноваційні рішення, використовуючи сучасні технології та підходи.

2 РОЗРОБКА АЛГОРИТМІВ ТА ПРИСТРОЮ ДЛЯ СИСТЕМИ ЗЧИТУВАННЯ RFID МІТОК

2.1 Огляд існуючих рішень та обґрунтування вибору плати Raspberry Pi Pico

Було здійснено огляд та порівняння різних рішень для реалізації системи обліку студентів за допомогою RFID/NFC міток. Зокрема, розглядатимуться платформи на основі мікроконтролерів Arduino, ESP32, STM32, а також одноплатні комп'ютери Raspberry Pi та BeagleBone Black. Кожна з цих платформ має свої унікальні характеристики та особливості, що можуть впливати на вибір конкретного рішення для реалізації проекту. Метою цього огляду є визначення найбільш оптимального рішення, яке забезпечить надійну, ефективну та економічно вигідну роботу системи обліку студентів, враховуючи вимоги проекту та доступні ресурси.

Серед різноманітних платформ для розробки систем обліку студентів за допомогою RFID/NFC міток, особливе місце займають системи на основі мікроконтролерів Arduino. Arduino є відкритою апаратною та програмною платформою, яка здобула популярність завдяки своїй простоті використання, доступності та великій спільноті користувачів. У цьому розділі розглядаються дві найбільш поширені моделі цієї платформи – Arduino Uno та Arduino Mega. Ці моделі мають різні характеристики, що дозволяють обрати найбільш підходящий варіант для конкретних завдань та умов використання.

Arduino Uno є одним із найпопулярніших мікроконтролерів для початківців і професіоналів завдяки своїй простоті використання та великій спільноті. Основні характеристики включають 8-бітний мікроконтролер ATmega328P, 14 цифрових вхідних/вихідних пінів (6 з яких можуть використовуватися як PWM виходи), та 6 аналогових входів. Ця плата підходить для простих проектів обліку студентів за допомогою RFID/NFC міток, однак її обчислювальна потужність і пам'ять можуть бути обмеженими для складніших завдань.

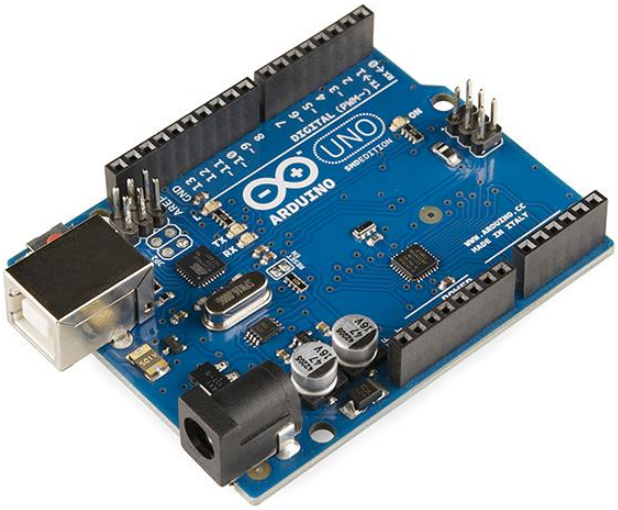


Рисунок 2.1 – Arduino Uno

Arduino Mega є більш потужною версією платформи Arduino, оснащеною мікроконтролером ATmega2560. Вона має 54 цифрових вхідних/вихідних пінів (15 з яких можуть використовуватися як PWM виходи), 16 аналогових входів і значно більший обсяг пам'яті порівняно з Arduino Uno. Це робить її підходящою для проектів з більшими вимогами до ресурсів, включаючи системи обліку студентів з розширеними функціональними можливостями.

Рисунок 2.2 – Arduino Mega

Окрім популярних платформ Arduino, існують й інші мікроконтролери, які можуть бути використані для розробки систем обліку студентів за допомогою RFID/NFC міток. Серед них особливо виділяються ESP32 та STM32. Ці мікроконтролери пропонують розширені можливості у порівнянні з Arduino, включаючи підтримку бездротових технологій, вищу продуктивність та більшу гнучкість у реалізації складних проектів. У цьому розділі розглядаються основні характеристики та переваги мікроконтролерів ESP32 та STM32, що дозволяє оцінити їх придатність для використання в системах обліку студентів.

ESP32 є потужним мікроконтролером з вбудованою підтримкою Wi-Fi та Bluetooth, що робить його ідеальним для бездротових проектів. Він оснащений двоядерним процесором, широким набором GPIO пінів, а також підтримує різноманітні периферійні інтерфейси, включаючи SPI, I2C та UART. ESP32 часто використовується в проектах Інтернету речей (IoT), і його можна успішно інтегрувати в систему обліку студентів з RFID/NFC мітками для віддаленого моніторингу та управління.

Рисунок 2.3 – ESP32

STM32 є серією 32-бітних мікроконтролерів на базі ARM Cortex-M, які відомі своєю високою продуктивністю та енергоефективністю. Ці

мікроконтролери пропонують широкий спектр можливостей, включаючи численні GPIO, підтримку різноманітних периферійних інтерфейсів та можливість виконання складних обчислювальних задач. Завдяки цьому STM32 може бути використаний для створення складних систем обліку з високими вимогами до швидкості обробки даних та інтеграції з іншими системами.

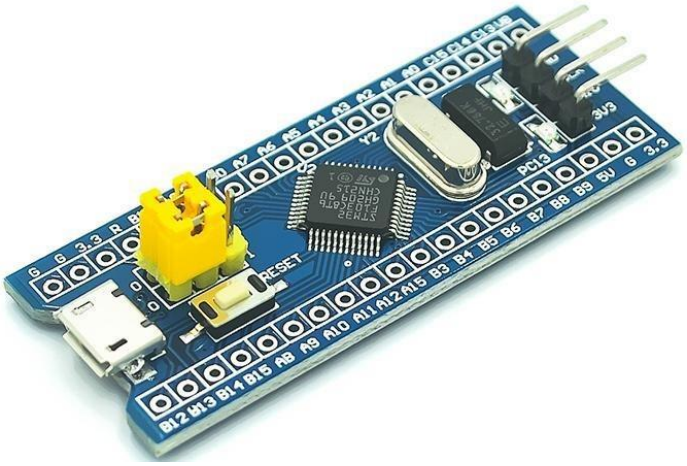


Рисунок 2.4 – STM32

Окрім мікроконтролерів, значну увагу в розробці складних систем обліку студентів за допомогою RFID/NFC міток привертають одноплатні комп'ютери. Серед найбільш популярних і потужних рішень у цій категорії виділяються Raspberry Pi та BeagleBone Black. Ці платформи забезпечують високу обчислювальну потужність, можливості підключення до мережі та широкий набір периферійних інтерфейсів. У цьому розділі буде проведено аналіз характеристик та особливостей одноплатних комп'ютерів серії Raspberry Pi та BeagleBone Black, щоб визначити їхню придатність для реалізації систем обліку студентів у контексті цього проекту.

Серія комп’ютерів Raspberry Pi, що нещодавно була розширена п’ятою версією, є популярними одноплатними комп’ютерами, що мають високу обчислювальну потужність, можливості підключення до мережі через Ethernet та Wi-Fi, а також широкий набір GPIO пінів. Вони використовують операційну систему Linux, що дозволяє використовувати широкий спектр програмного

забезпечення для реалізації різних функцій. Raspberry Pi може бути використаний для створення інтегрованих систем обліку студентів з розширеними функціями, такими як обробка даних у реальному часі, збереження великих обсягів даних та віддалений доступ.

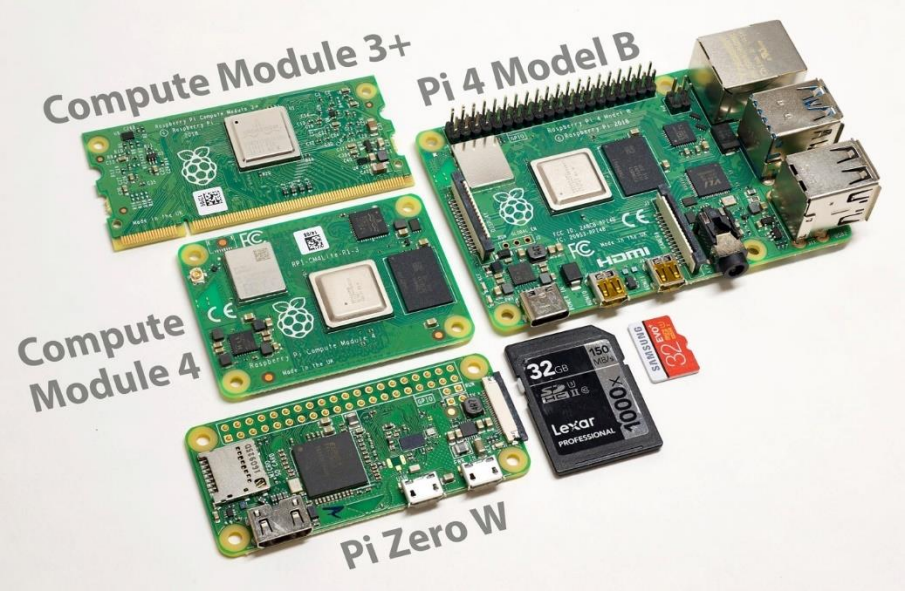


Рисунок 2.5 – Сімейство Raspberry Pi

BeagleBone Black є потужним одноплатним комп’ютером з процесором ARM Cortex-A8, що працює на частоті 1 ГГц. Він має велику кількість GPIO пінів, підтримує різноманітні периферійні інтерфейси та працює під управлінням операційної системи Linux. BeagleBone Black підходить для створення складних систем з високими вимогами до обчислювальної потужності та можливостями розширення, що робить його гарним вибором для проектів обліку студентів з RFID/NFC мітками, які потребують інтеграції з іншими системами або сервісами.

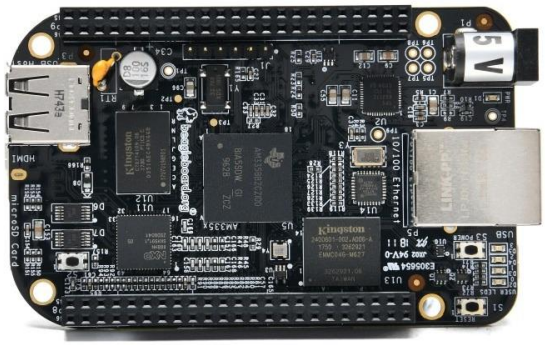


Рисунок 2.6 – BeagleBone Black

Raspberry Pi Pico є мікроконтролером, який відрізняється від основної лінійки одноплатних комп'ютерів Raspberry Pi 3-5 тим, що він призначений для виконання менш складних завдань з більш низьким енергоспоживанням. Raspberry Pi Pico оснащений двоядерним процесором ARM Cortex-M0+ з тактовою частотою 133 МГц, має 264 КБ оперативної пам'яті та 2 МБ флеш-пам'яті. Вона підтримує широкий спектр периферійних інтерфейсів, включаючи 26 багатофункціональних GPIO пінів, які дозволяють підключати різноманітні пристрої та сенсори. На відміну від більш потужних моделей Raspberry Pi, які використовують операційну систему Linux і мають значно більшу обчислювальну потужність та можливості підключення до мережі, Raspberry Pi Pico працює на низькому рівні і призначена для вбудованих систем та проектів IoT. Важливою перевагою є низька ціна, що робить її доступною для широкого кола користувачів, включаючи студентів та аматорів електроніки.

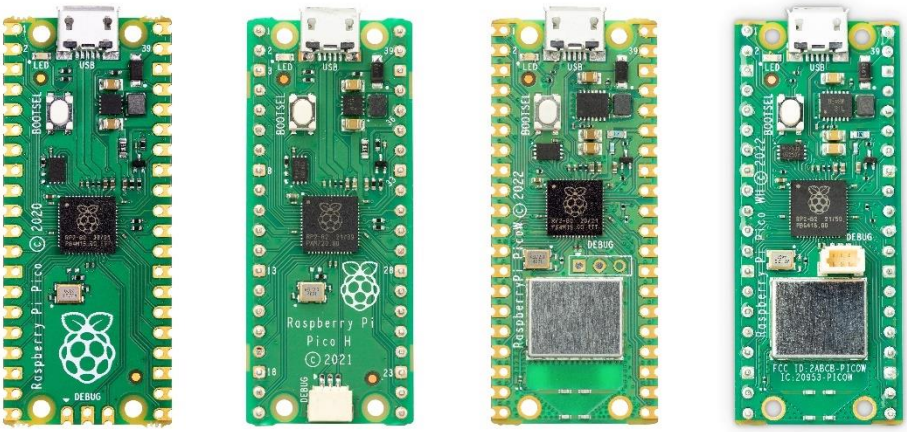


Рисунок 2.7 – Моделі Raspberry Pi Pico

Вибір модуля Raspberry Pi Pico для реалізації системи обліку студентів був обґрунтований декількома ключовими факторами. Низька ціна робить його доступним та економічно вигідним рішенням, що є важливим для студентських проектів. Технічні характеристики, такі як потужний двоядерний процесор ARM Cortex-M0+, велика кількість GPIO пінів, підтримка різних мов програмування (C/C++ та MicroPython) та низьке енергоспоживання, забезпечують високу продуктивність та гнучкість у розробці. Додатково, наявність детальної документації та активна спільнота підтримки роблять Raspberry Pi Pico ідеальним вибором для створення надійної та ефективної системи обліку студентів.

2.2 Огляд існуючих рішень та обґрунтування вибору модуля RFID RC522

Для реалізації системи обліку студентів за допомогою RFID/NFC технологій необхідно обрати відповідний RFID модуль, який забезпечить надійне зчитування міток. Вибір правильного модуля є критично важливим для забезпечення ефективності, надійності та точності системи. Під час виконання роботи було проведено огляд існуючих рішень з різних типів RFID/NFC зчитувачів та міток, що дозволило обґрунтовано вибрати найкращий варіант для нашого проекту.

Існує кілька основних типів зчитувачів та тегів, які використовуються в системах RFID/NFC. Вони класифікуються за частотними діапазонами та

призначенням. Найпоширенішими є низькочастотні (LF), високочастотні (HF), включаючи NFC, та ультрависокочастотні (UHF) зчитувачі та мітки.

Низькочастотні (LF) чіпи працюють на частоті 125 кГц або 134.2 кГц. Вони мають відносно короткий радіус зчитування (до 10 см) і використовуються для простих та дешевих RFID систем. LF чіпи є менш чутливими до металевих предметів та рідин, що робить їх придатними для використання в умовах, де інші чіпи можуть мати проблеми зі зчитуванням.

Наприклад, модуль RDM6300 працює на частоті 125 кГц і підтримує зчитування низькочастотних RFID міток, що відповідають стандарту EM4100. Він забезпечує зчитування міток на відстані до 10 см і використовує інтерфейс UART для зв'язку з мікроконтролером. Цей модуль відрізняється простою конструкцією та легкістю інтеграції.

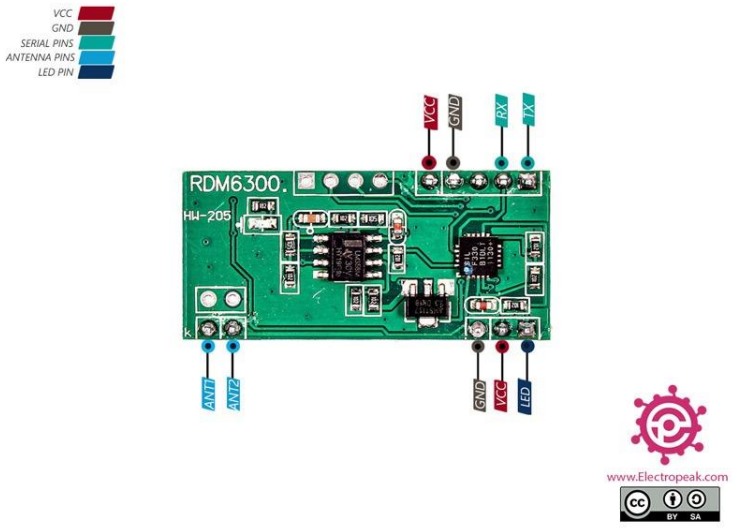


Рисунок 2.8 – RDM6300

Високочастотні (HF) чіпи працюють на частоті 13.56 МГц і є одними з найпоширеніших чіпів для RFID/NFC систем. Вони підтримують різні стандарти, такі як ISO/IEC 14443A/B та ISO/IEC 15693, що дозволяє використовувати їх у багатьох додатках.

Модуль RC522 є одним з найпоширеніших RFID модулів завдяки своїй доступності, простоті інтеграції та надійності. Він працює на частоті 13.56 МГц і

підтримує стандарт ISO/IEC 14443A. RC522 забезпечує швидке зчитування міток на відстані до 5 см, має низьке енергоспоживання і підтримує інтерфейси SPI, I2C та UART.

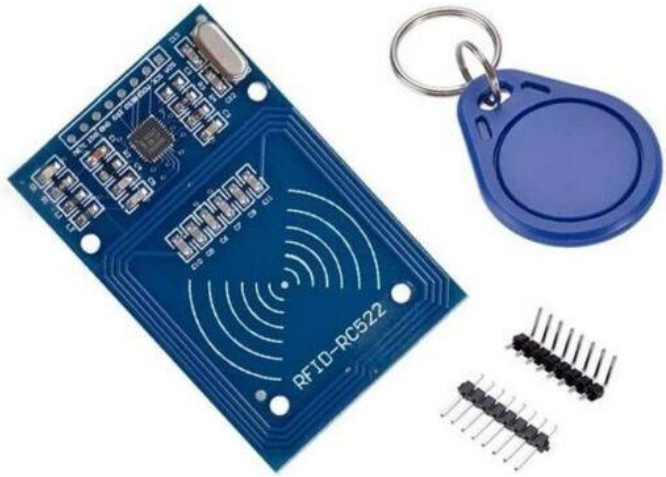


Рисунок 2.9 – RC522

Модуль PN532 є універсальним RFID/NFC модулем, що підтримує широкий спектр стандартів, включаючи ISO/IEC 14443A/B, ISO/IEC 15693 та NFC. Він працює на частоті 13.56 МГц і забезпечує зчитування міток на відстані до 7 см. PN532 підтримує інтерфейси I2C, SPI та UART, що дозволяє його легко інтегрувати в різні проекти.

Рисунок 2.10 – PN532

Ультрависокочастотні (UHF) чіпи працюють на частоті від 860 до 960 МГц і забезпечують значно більший радіус зчитування, що може сягати кількох метрів. Вони використовуються в таких застосуваннях, як логістика, складування та управління ланцюгами постачання.

Модуль Impinj R2000 працює на частоті UHF і підтримує стандарти EPC Gen 2 та ISO 18000-6C. Він забезпечує зчитування міток на відстані до кількох метрів і дозволяє зчитувати велику кількість міток одночасно. Цей модуль використовується в системах, де потрібне зчитування міток на великій відстані або в складних умовах.

Після аналізу різних типів RFID та NFC міток та зчитувачів, модуль RFID RC522 було обрано як найкращий варіант для використання з Raspberry Pi Pico у контексті реалізації системи обліку студентів. RC522 забезпечує необхідну функціональність для зчитування високочастотних міток, має низьке енергоспоживання, легко інтегрується з Raspberry Pi Pico через інтерфейси SPI, I2C та UART. Додатково, цей модуль відрізняється низькою ціною, що робить його економічно вигідним рішенням. Ці характеристики роблять RC522 оптимальним вибором для створення надійної та ефективної системи обліку студентів.

2.3 Огляд існуючих рішень та вибір технологій для програмної реалізації

Вибір типу бази даних та способу її підключення є ключовими аспектами при реалізації інформаційних систем. Кожен тип бази даних має свої унікальні характеристики, які визначають їх оптимальне застосування для різних сценаріїв. Правильний вибір бази даних забезпечує ефективність, продуктивність та надійність системи.

Реляційні бази даних базуються на реляційній моделі даних, яка організовує інформацію у вигляді таблиць, пов'язаних між собою за допомогою ключів. Це дозволяє забезпечити цілісність даних та виконувати складні запити. Наприклад,

MySQL та MariaDB є дуже популярними реляційними базами даних, що використовуються в різноманітних додатках, від невеликих веб-сайтів до великих корпоративних систем. Вони обидві мають відкритий код та активну спільноту розробників, що робить їх привабливими для багатьох організацій. MySQL славиться своєю швидкодією та простотою у використанні, тоді як MariaDB, яка є форком MySQL, часто отримує нові функції та вдосконалення, що робить її привабливою для розробників, які шукають передові можливості.

PostgreSQL є ще однією потужною реляційною базою даних з відкритим кодом, відомою своєю надійністю, розширюваністю та високою продуктивністю. Вона підтримує широкий спектр функцій, включаючи роботу з географічними об'єктами, JSON та XML, що дозволяє використовувати її у різних сферах, від геоінформаційних систем до веб-додатків.

Microsoft SQL Server, розроблений корпорацією Microsoft, є популярним вибором серед підприємств, які вже використовують інші продукти Microsoft. Він забезпечує інтеграцію з Windows Server та Azure Cloud Services, що спрощує роботу для компаній, що використовують екосистему Microsoft. SQL Server надає потужні інструменти для роботи з даними, включаючи аналітику та бізнес-інтелект.

Oracle Database є однією з найпотужніших та розширюваних реляційних баз даних, підходячи для великих систем та високонавантажених середовищ. Вона забезпечує масштабованість та високу надійність, маючи функції для управління даними, такі як передова реплікація та забезпечення безпеки. Oracle також має велику екосистему інструментів та додатків, що підтримують її використання в корпоративних середовищах.

Нереляційні бази даних, або NoSQL бази даних, пропонують альтернативний підхід до зберігання та управління даними. Вони використовуються для роботи з великими обсягами даних, які можуть мати складні та змінні структури. MongoDB, наприклад, є документ-орієнтованою базою даних, яка використовує гнучку схему документів у форматі JSON. Це робить її ідеальною для веб-розробки та аналітичних задач, де потрібна гнучка

схема та висока продуктивність. MongoDB підтримує горизонтальне масштабування, що дозволяє розподіляти дані на кілька серверів для підвищення продуктивності та доступності.

Cassandra є розподіленою колоночною базою даних, розробленою для забезпечення високої доступності та масштабованості. Вона підходить для високонавантажених додатків, таких як соціальні мережі та реальна аналітика, забезпечуючи розподілене зберігання даних та їх реплікацію на різних вузлах мережі.

Redis є базою даних типу ключ-значення, яка зберігає дані в пам'яті, забезпечуючи надзвичайно швидкий доступ до них. Це робить Redis ідеальним для завдань, що вимагають високої швидкості, таких як кешування та управління сесіями у веб-додатках. Redis підтримує різні структури даних, включаючи строки, хеші, списки та множини, що робить його універсальним інструментом для різноманітних сценаріїв використання.

Хмарні бази даних є сучасним рішенням для зберігання та управління даними, використовуючи обчислювальні ресурси хмарних сервісів. Вони забезпечують масштабованість, високу доступність та гнучкість у використанні ресурсів. Наприклад, Amazon RDS є послугою управління базами даних, яка дозволяє користувачам легко створювати та керувати реляційними базами даних у хмарному середовищі Amazon Web Services (AWS). Вона підтримує різні двигуни баз даних, такі як MySQL, PostgreSQL, Oracle і SQL Server, надаючи користувачам можливість вибрати найбільш підходящий для них варіант. Amazon RDS надає автоматичне масштабування, резервне копіювання та високу доступність, що робить його популярним вибором для побудови надійних та масштабованих додатків у хмарному середовищі.

Google Cloud SQL, інша хмарна послуга, дозволяє розгортати та керувати реляційними базами даних на платформі Google Cloud. Вона інтегрована з іншими послугами Google Cloud, що забезпечує легкість управління та гнучкість масштабування. Google Cloud SQL підтримує автоматичне масштабування, що

дозволяє швидко адаптуватися до змін у навантаженні на базу даних, забезпечуючи високу продуктивність та доступність.

Microsoft Azure SQL Database є повністю керованою базою даних як сервіс на хмарній платформі Microsoft Azure. Вона надає вбудовані інструменти аналітики та бізнес-інтелекту, що дозволяє користувачам ефективно аналізувати свої дані та приймати обґрунтовані рішення. Azure SQL Database забезпечує автоматичне резервне копіювання, високу доступність та гнучкість масштабування, дозволяючи користувачам швидко реагувати на зміни у вимогах їх додатків.

Процес підключення до баз даних відрізняється залежно від їх типу. Для реляційних баз даних, таких як MySQL, PostgreSQL або Microsoft SQL Server, зазвичай використовують ORM (Object-Relational Mapping) бібліотеки, які спрощують інтеграцію з програмним забезпеченням. Наприклад, для мови програмування Java часто використовують Hibernate, а для .NET - Entity Framework. Ці бібліотеки дозволяють розробникам працювати з базами даних, використовуючи об'єктно-орієнтовану модель, що значно спрощує процес розробки та управління даними.

Для NoSQL баз даних, таких як MongoDB, Cassandra або Redis, зазвичай використовують спеціалізовані драйвери та API. Кожна база даних надає свій набір інструментів для зв'язку з системою, що може включати офіційні драйвери або сторонні бібліотеки, які реалізують підтримку в конкретній мові програмування. Це дозволяє розробникам легко інтегрувати NoSQL бази даних у свої проекти, забезпечуючи високу продуктивність та масштабованість.

Для хмарних баз даних, таких як Amazon RDS, Google Cloud SQL або Microsoft Azure SQL Database, більшість з них надають SDK (Software Development Kit) та API для інтеграції з різними мовами програмування та фреймворками. Це дозволяє розробникам легко підключати своє програмне забезпечення до хмарної інфраструктури та баз даних, використовуючи стандартні інструменти розробки. Хмарні бази даних також забезпечують автоматичне масштабування, резервне копіювання та високу доступність, що

робить їх привабливими для розробників, які прагнуть створювати надійні та масштабовані додатки.

Важливою складовою проекту є веб-сервер. У сучасному веб-розробництві велике значення мають веб-сервери, що використовують розподілену архітектуру, яка включає фронтенд і бекенд компоненти.

Фронтенд - це частина веб-додатку, з якою користувачі взаємодіють безпосередньо. Вона відповідає за відображення інформації користувачу та обробку його взаємодії з додатком. Для реалізації фронтенду використовуються різноманітні технології, такі як React, Svelte та інші.[26-28] Наприклад, React є бібліотекою JavaScript, що дозволяє створювати інтерактивні та ефективні інтерфейси користувача. Ця бібліотека відома своєю гнучкістю і можливістю створювати компонентні структури, що спрощують розробку та підтримку додатків. Svelte, на відміну від React, генерує ефективний JavaScript-код під час компіляції, що зменшує розмір кінцевого додатку та покращує його продуктивність. Інші популярні технології для фронтенду включають Vue.js, який також є потужним інструментом для створення сучасних веб-додатків.

Бекенд - це частина веб-додатку, що відповідає за логіку, обробку даних та взаємодію з базою даних. Для створення бекенду часто використовують мови програмування, такі як JavaScript з використанням середовища виконання Node.js. Node.js дозволяє розробникам писати серверний код на JavaScript, що спрощує розробку, оскільки одна мова програмування використовується як для фронтенду, так і для бекенду. Популярними фреймворками для Node.js є Express.js, який надає мінімалістичний підхід до створення веб-серверів, та Nest.js, який використовує модульну структуру для створення масштабованих і підтримуваних додатків. Інші технології для бекенду включають Django для Python, Ruby on Rails для Ruby та ASP.NET для C#. [26][29-30]

Веб-сервер - це програмне забезпечення, яке обробляє запити від користувачів через HTTP-протокол і надає відповіді, які можуть бути як статичними сторінками, так і динамічно згенерованим вмістом. Веб-сервери можуть бути універсальними або спеціалізованими, наприклад, для обробки

конкретних типів контенту чи застосування певних технологій. Apache та Nginx є прикладами популярних веб-серверів, які підтримують широкий спектр можливостей для обробки запитів та розгортання веб-додатків. Apache є багатофункціональним сервером з великою кількістю модулів для різних завдань, тоді як Nginx відомий своєю високою продуктивністю та здатністю ефективно обробляти великі обсяги одночасних з'єднань.

Існують також хмарні рішення для веб-серверів, такі як Amazon Web Services (AWS) Elastic Beanstalk, Google App Engine та Microsoft Azure App Services. Ці платформи надають можливість легко розгортати, масштабувати та керувати веб-додатками в хмарі, забезпечуючи високу доступність та автоматичне масштабування.

Отже, вибір технологій для створення веб-сервера залежить від вимог проекту, наявних ресурсів та уподобань розробників. Розподілена архітектура, що включає сучасні інструменти для фронтенду та бекенду, дозволяє створювати ефективні та масштабовані веб-додатки, здатні задовольнити потреби сучасних користувачів.

2.5 Постановка задачі

Ціллю цього проекту є створення системи контролю відвідуваності студентів за допомогою RFID/NFC міток. Проведений аналіз існуючих рішень, які використовуються в різних компаніях для аналогічних цілей, виявив декілька ключових особливостей та потреб, що необхідно врахувати для успішної реалізації нашого проекту.

Як контролер для системи буде використано Raspberry Pi Pico W, оснащений Wi-Fi модулем. Це рішення обране завдяки наявності зручної документації та великої кількості ресурсів, доступних в Інтернеті, що полегшує його інтеграцію та подальший розвиток. Raspberry Pi Pico W дозволить відправляти запити на веб-

сервер для обробки даних та створення звітів, забезпечуючи надійне і ефективне управління системою контролю відвідуваності.

Для зчитування NFC міток буде використано модуль RC522. Вибір на користь RC522 зумовлений його низькою вартістю, що робить його доступним для проектів з обмеженим бюджетом, а також достатньою функціональністю для задоволення потреб системи контролю відвідуваності. RC522 забезпечує можливість зчитування UID міток NFC, а також читання і запис даних, що робить його ідеальним для нашого застосування.

Підключення модуля RC522 до Raspberry Pi Pico W буде здійснено наступним чином: піни SDA, SCK, MOSI, MISO та RST модуля зчитування RFID MFRC522 будуть підключені до виводів GPIO Raspberry Pi Pico W на піни 5, 6, 7, 4 та 3 відповідно. Крім того, піни живлення 3.3V та GND модуля RFID будуть підключені до відповідних пінів 3.3V та GND на платі Raspberry Pi Pico W. У процесі дослідження було знайдено докладну схему підключення Raspberry Pi Pico до модуля RC522 з використанням додаткових компонентів. Ця схема стане основою для фізичного підключення контролера до зчитувача та буде використана для розробки проекту, забезпечуючи надійне та ефективне функціонування системи контролю відвідуваності.

Значно розширивши дане рішення, можна досягти високого рівня інтеграції апаратних компонентів з програмним забезпеченням, що дозволить реалізувати надійну та стабільну систему. Важливим аспектом є використання додаткових компонентів, таких як резистори, конденсатори та стабілізатори напруги, що забезпечать захист та стабільну роботу всіх елементів схеми. Подальші кроки включають програмування контролера для взаємодії з модулем RC522, тестування та налагодження зчитування RFID-тегів, а також інтеграцію з існуючою системою контролю відвідуваності, що забезпечить автоматизацію процесу реєстрації та обліку студентів.

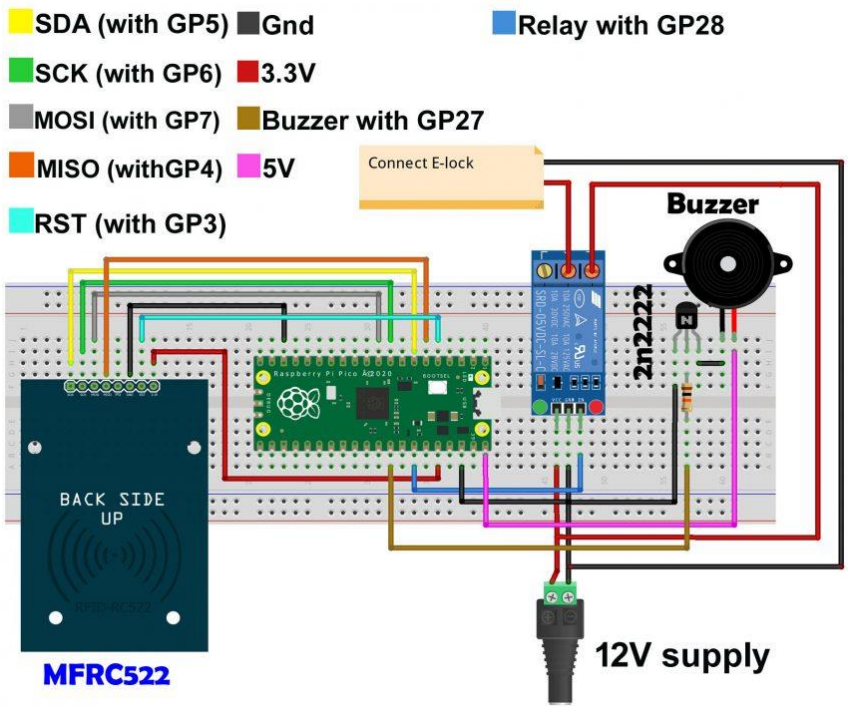


Рисунок 1.8 - Схема підключення Rasberry Pi Pico до RC522 з деякими додатковими компонентами

Для програмної реалізації цього модуля ми вирішили використовувати мову програмування Python. Наш модуль буде відповідати за зчитування даних від RFID зчитувача і передачу цих даних на сервер для подальшої обробки за допомогою HTTP або HTTPS запитів. Сервер оброблятиме отримані дані і повертатиме відповідь, яка буде залежати від валідності зчитаного ключа. Такий підхід дозволить ефективно використовувати RFID-мітки для контролю доступу, забезпечуючи при цьому необхідний рівень безпеки та функціональності.

Python був обраний з кількох причин. По-перше, ця мова програмування має велику кількість бібліотек і модулів, що полегшує роботу з різними апаратними та мережевими інтерфейсами. Бібліотеки, такі як MFRC522 для роботи з RFID модулями та requests для здійснення HTTP запитів, значно спрощують процес розробки. По-друге, Python є інтуїтивно зрозумілим і широко використовуваним, що сприяє швидкому розвитку і впровадженню проекту.

При використанні цієї архітектури виникає компроміс: хоча інтеграція є зручною та простою, система може працювати з меншою швидкістю через те, що

перевірка даних виконується не на контролері, а на сервері. Проте такий підхід дозволяє використовувати контролер виключно для конкретного завдання і не потребує додаткової пам'яті для зберігання всієї бази валідних ключів. Крім того, це рішення забезпечує більшу гнучкість у разі частих змін у базі даних, оскільки логіка валідації може бути легко змінена на сервері без необхідності оновлення програмного забезпечення на контролері.

Таким чином, при відправленні даних на сервер для перевірки, контролер звільняється від необхідності зберігання великої кількості даних і складних алгоритмів валідації. Це дозволяє зменшити вимоги до апаратного забезпечення контролера і робить систему більш ефективною з точки зору управління ресурсами. Крім того, централізація логіки перевірки на сервері спрощує процес оновлення і підтримки системи, оскільки всі зміни можуть бути внесені безпосередньо на сервері, без необхідності фізичного доступу до кожного окремого контролера.

Веб-сервер буде побудований на базі технології Node.js, яка здобула велику популярність для створення серверних додатків завдяки високій продуктивності та широкому спектру доступних бібліотек. Як базу даних буде використано MongoDB, оскільки вона легко інтегрується з Node.js і пропонує гнучкі можливості для роботи з даними у форматі JSON-подібних об'єктів.

MongoDB є документно-орієнтованою базою даних, що надає зручний спосіб зберігання та обробки структурованих даних. Вона ідеально підходить для сучасних веб-додатків, оскільки дозволяє зберігати та опрацьовувати дані у форматі JSON, що спрощує взаємодію між бекендом та фронтендом додатку. Крім того, MongoDB забезпечує гнучку масштабованість та швидкий доступ до даних, що є критично важливим для ефективної роботи веб-систем.

Інтеграція MongoDB з Node.js дозволяє використовувати зручний драйвер MongoDB, який забезпечує взаємодію з базою даних за допомогою мови програмування JavaScript. Це значно спрощує розробку та підтримку серверного коду, оскільки розробники можуть використовувати одну мову для всіх аспектів

додатку. Такий підхід полегшує комунікацію в команді розробників і прискорює процес розробки, роблячи його більш ефективним та узгодженим.

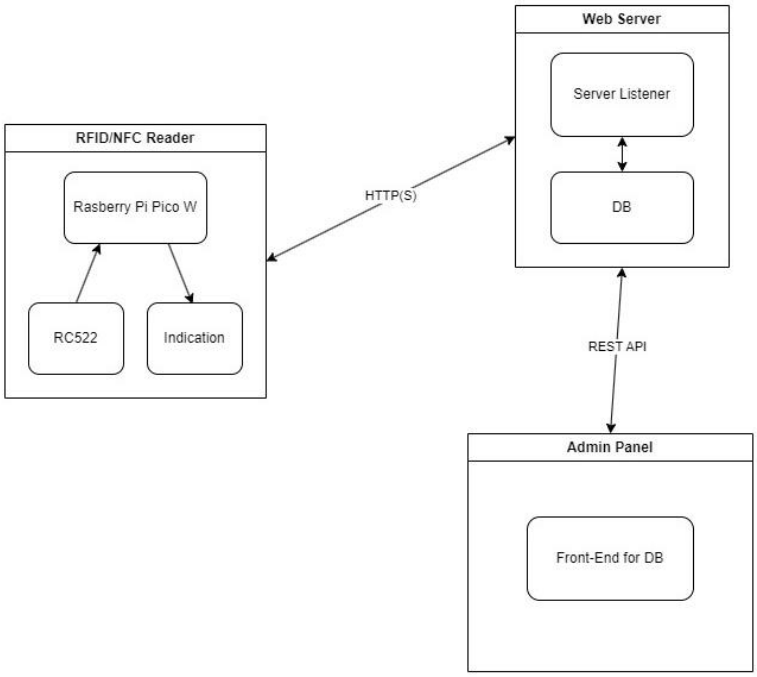


Рисунок 1.9 - Діаграма архітектури

Для взаємодії з базою даних буде розроблена адмін-панель, яка використовуватиме технологію React. React — це потужна бібліотека JavaScript для створення користувацьких інтерфейсів, що дозволяє створювати ефективні та зручні веб-додатки. Завдяки використанню React, адмін-панель буде швидкою, інтерактивною та легко підтримуваною. [27]

Використання React для створення адмін-панелі забезпечить високу продуктивність і зручність роботи з інтерфейсом. React дозволяє розробляти компоненти, які можна повторно використовувати, що значно спрощує процес розробки та підтримки коду. Компонентний підхід React дозволяє розробникам створювати окремі частини інтерфейсу, які можуть бути незалежними та взаємозамінними. Крім того, React надає можливість використовувати властивості та стан компонентів для динамічного відображення даних з бази даних, що підвищує інтерактивність та користувацький досвід.

Взаємодія між адмін-панеллю та веб-сервером буде реалізована за допомогою архітектури REST [29-30] API. REST (Representational State Transfer) — це архітектурний стиль, що використовує стандартні HTTP методи для взаємодії між клієнтом і сервером. Адмін-панель буде надсилати HTTP запити до визначених ендпоінтів (URL-адрес), які відповідають за виконання певних операцій над ресурсами на сервері. Наприклад, адмін-панель може надсилати POST запити для створення нових записів у базі даних, GET запити для отримання інформації про існуючі записи, PUT або PATCH запити для оновлення існуючих даних та DELETE запити для видалення записів.

На веб-сервері буде реалізована обробка цих запитів та формування відповідей. Сервер прийматиме запити, оброблятиме їх, звертаючись до бази даних при необхідності, і повертатиме відповіді у форматі JSON або іншому підтримуваному форматі. Використання REST API забезпечує простоту, гнучкість та незалежність між клієнтом та сервером у проекті. Цей підхід дозволяє легко масштабувати систему, додавати нові функції та змінювати логіку взаємодії, не впливаючи на роботу інших компонентів системи.

React також інтегрується з іншими сучасними технологіями, такими як Redux для керування станом додатку, що забезпечує підтримку складних додатків із багатьма компонентами. Крім того, існують численні бібліотеки та інструменти, такі як Material-UI або Ant Design, які спрощують створення візуально привабливих та функціональних інтерфейсів. У поєднанні з потужними можливостями Node.js і MongoDB, React допоможе створити ефективну та масштабовану адмін-панель для управління даними в нашій системі контролю відвідуваності студентів.

Для розробки з використанням React існує безліч ресурсів та документації, що полегшує навчання та використання цієї бібліотеки. Офіційна документація React містить детальні приклади та гайдлайни, які допоможуть розробникам швидко освоїти основи та розпочати роботу над проектом.

3 ПРОГРАМНО ПАПРАТНА РЕАЛІЗАЦІЯ СИСТЕМИ КОНТРОЛЮ ВІДВІДУВАНОСТІ СТУДЕНТІВ З ВИКОРИСТАННЯМ RFID МІТОК

3.1 Збір модуля для зчитування RFID міток та розробка програмного забезпечення для Raspberry Pi Pico W

Для роботи системи необхідно під'єднати плату Raspberry Pi Pico до модуля RC522, що дозволить зчитувати RFID мітки та інтегрувати їх у систему обліку студентів. Модуль RC522 є популярним високочастотним RFID-зчитувачем, що працює на частоті 13.56 МГц і підтримує стандарт ISO/IEC 14443A. Raspberry Pi Pico, з іншого боку, є потужним та економічно вигідним мікроконтролером, який забезпечує широкі можливості підключення периферійних пристроїв через інтерфейси GPIO.

Нижче наведено схему підключення, яка допоможе зрозуміти, які піни куди будуть підключені.

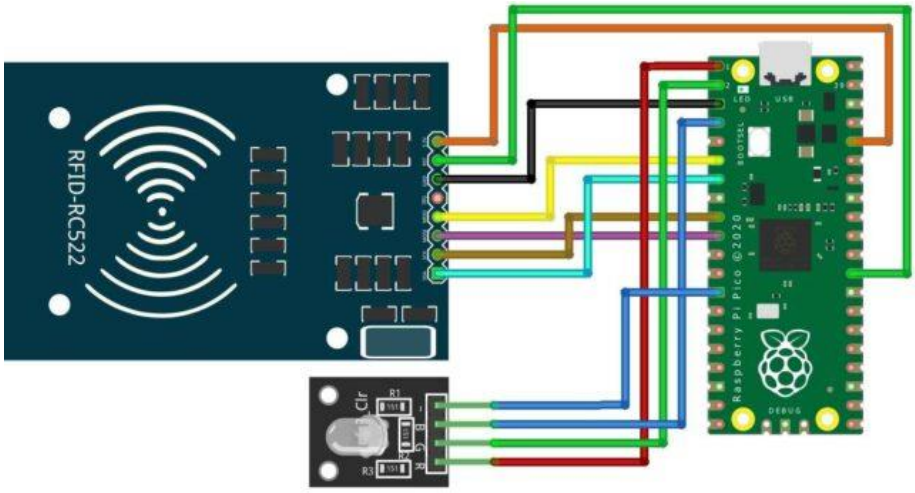


Рисунок 3.1- Схема підключення RC522 до Raspberry Pi Pico

Також наведено таблицю підключення пінів модуля RC522 до пінів плати Raspberry Pi Pico W

Таблиця 1.1 – Назва таблиці

Raspberry Pi Pico	MFRC522 RFID module
GND	GND
3V3	3.3V
GP5	SDA
GP6	SCK
GP7	MOSI
GP4	MISO
—	IRQ
GPIO22	RST

Процес збирання починається з підготовки плати Raspberry Pi Pico W та модуля зчитування RFID RC522. Для зручності підключення ми використовували плату для тестування мікросхем (breadboard) та провода. Це дозволяє легко встановлювати та змінювати підключення компонентів під час розробки та тестування. Зібрана схема зображена нижче:

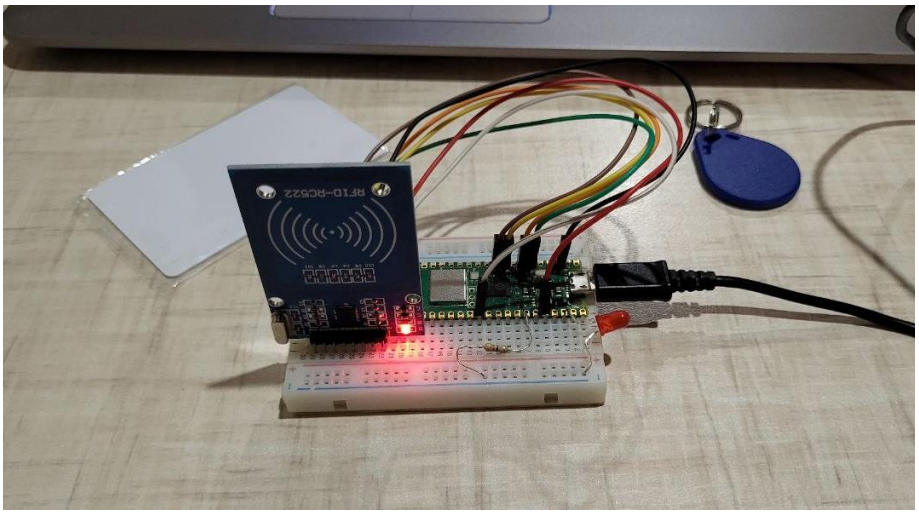


Рисунок 3.2 – Фото підключення модуля RC522 до плати Raspberry Pi Pico

Для надійного з'єднання модуля зчитування RFID RC522 з Raspberry Pi Pico W були припаяні піни до відповідних контактів модуля. Це забезпечує стійке електричне з'єднання, необхідне для стабільної роботи системи.

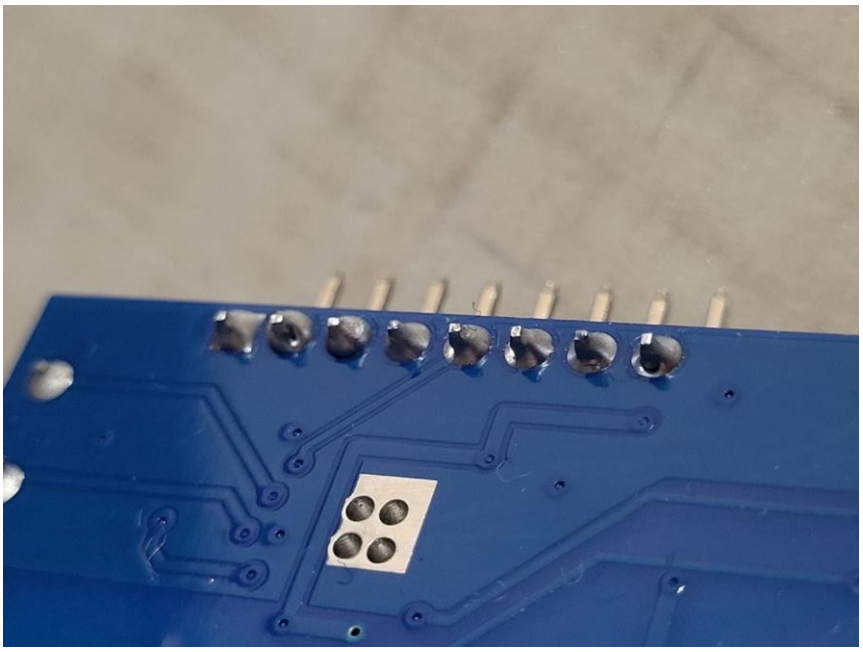


Рисунок 3.3 – Фото пінів RC522

Також в рамках тестування роботи Raspberry Pi Pico W до плати був приєднаний світлодіод з резистором. Світлодіод підключався для візуального відображення різних станів системи під час роботи. Резистор, у свою чергу, захищає світлодіод від надмірного струму, запобігаючи його пошкодженню.

Першим кроком у розробці програмного забезпечення є налаштування середовища розробки. Це включає встановлення необхідних інструментів та бібліотек, а також налаштування з'єднання між Raspberry Pi Pico W та комп'ютером. Також необхідно встановити MicroPython на Raspberry Pi Pico W, що дозволяє виконувати Python-код на мікроконтролері. Для розробки використовувався редактор VS Code, що було знайомим та універсальним рішенням для всіх частин проєкту, рабом з плагінами для роботи з Raspberry Pi Pico.

Для реалізації функціональності системи зчитування RFID міток було використано модульну структуру коду, яка забезпечує гнучкість, зрозумілість та легкість у підтримці. Кожен модуль виконує певну функцію і взаємодіє з іншими модулями через чітко визначені інтерфейси. Основні компоненти програмного забезпечення були розділені на кілька файлів, що відповідають за конкретні завдання.

Файл config.py містить конфігураційні налаштування, такі як параметри Wi-Fi з'єднання, адреси веб-сервера та інші параметри, які можуть бути легко змінені без потреби редагувати основний код. Цей файл дозволяє централізовано керувати налаштуваннями системи, що забезпечує гнучкість та зручність в управлінні конфігураціями.

Файл led.py використовується для керування світлодіодом, який відображає різні стани системи. У цьому модулі реалізовано клас LED, який інкапсулює логіку роботи зі світлодіодом через GPIO піни. Світлодіод використовується для візуальної індикації подій, таких як успішне зчитування RFID мітки або виникнення помилок.

Файл mfrc522.py є бібліотекою, що відповідає за роботу зі зчитувачем RFID RC522. Цей модуль реалізує клас MFRC522, який забезпечує функції для ініціалізації зчитувача, зчитування міток та обробки отриманих даних. Використання цього модуля дозволяє легко інтегрувати зчитувач RFID у систему та забезпечити стабільне зчитування міток.

Файл rfidtest.py призначений для тестування роботи зчитувача RFID. Він містить код для перевірки правильності зчитування міток та налаштувань зчитувача. Цей файл використовується для тестування системи під час розробки, щоб впевнитися у правильному функціонуванні всіх компонентів.

Також на цьому файлі можна показати роботу програмного середовища з бібліотеками. Оскільки використовувалось рішення з VS Code, деякі частини робочого коду (рішення взято з відкритих джерел для тестування) можуть бути підсвічені.

Рисунок 3.4 – Приклад коду для тестування зчитувача RC522

Файл wifi.py забезпечує підключення до Wi-Fi мережі та взаємодію з веб-сервером. У цьому модулі реалізовано функцію connect_to_wifi, яка відповідає за встановлення з'єднання з вказаною Wi-Fi мережею. Цей модуль забезпечує стабільне з'єднання, необхідне для передачі даних від Raspberry Pi Pico W до веб-сервера.

Файл main.py є основним файлом проекту, який ініціалізує всі необхідні компоненти системи та координує їхню роботу. У цьому файлі виконується підключення до Wi-Fi мережі, ініціалізація модуля зчитування RFID та керування світлодіодом. Файл main.py містить головний цикл програми, в якому здійснюється зчитування RFID міток та відправка даних на сервер.

Після імпорту необхідних модулів, таких як `WiFiConnection`, `post_rfid`, `Led`, `gc`, `sleep` та `MFRC522`, створюється функція `check_memory`, яка відповідає за збір сміття в пам'яті та виведення кількості вільної пам'яті. Це дозволяє контролювати використання ресурсів мікроконтролера та забезпечити стабільність роботи програми.

Функція `main` починається з ініціалізації об'єкта `WiFiConnection` та підключення до WiFi. Після цього створюється об'єкт `Led` для вбудованого світлодіода, і виконується перевірка його працездатності за допомогою методу

`test_led`. Також ініціалізується інший об'єкт `Led` для зовнішнього світлодіода, і проводиться його тестування для перевірки коректності підключення та функціонування.

Наступним кроком є створення об'єкта `MFRC522` для роботи з RFID модулем. Налаштовуються параметри SPI, такі як `spi_id`, `sck`, `miso`, `mosi`, `cs` та `rst`. У головному циклі програми спочатку ініціалізується RFID модуль, після чого виконується запит на наявність RFID тегу. Якщо тег виявлено, виконується його зчитування та отримання унікального ідентифікатора (UID). UID перетворюється в ціле число, яке потім виводиться на екран для візуального контролю та надсилається на сервер за допомогою функції `post_rfid`.

Цикл виконується з затримкою у 0.3 секунди для запобігання надмірному навантаженню на систему та забезпечення стабільного зчитування RFID тегів. У разі натискання клавіші `Ctrl+C`, що генерує виключення `KeyboardInterrupt`, цикл припиняється, і на екран виводиться повідомлення про зупинку програми. Завершується робота програми відключенням від WiFi.

Цей алгоритм забезпечує автоматичне підключення Raspberry Pi Pico W до WiFi, перевірку роботи світлодіодів, зчитування RFID тегів та надсилання їх ідентифікаторів на сервер у реальному часі. Це дозволяє створити ефективну систему для зчитування та обробки RFID тегів з можливістю подальшої інтеграції у більші інформаційні системи або проекти.

3.2 Розробка веб-серверу

Node.js є легким та гнучким фреймворком, який дозволяє розробникам мати повний контроль над тим, як організувати свій код. Ця гнучкість є великою перевагою, але водночас може призвести до нестандартності та хаосу, якщо не дотримуватися певних правил структурування проекту. Визначена структура проекту допомагає орієнтуватись у проекті та полегшує розробку на початкових етапах.

Організація коду дозволяє легко знаходити та змінювати необхідні файли та логіку, що робить проект зручним у підтримці та розширенні. Розділення коду на логічні модулі дозволяє використовувати їх повторно в інших частинах проекту або навіть в інших проектах, що підвищує ефективність розробки та зменшує кількість дублікатів коду. Завдяки чіткій структурі кожен модуль можна легко тестувати окремо, що робить процес тестування більш систематичним та ефективним. Чітка структура також полегшує інтеграцію нових функціональностей, оскільки новий код можна легко інтегрувати в існуючу архітектуру. У великих проектах, де над кодом працюють кілька розробників, чітка структура допомагає уникнути конфліктів та непорозумінь, забезпечуючи гладку та узгоджену розробку.

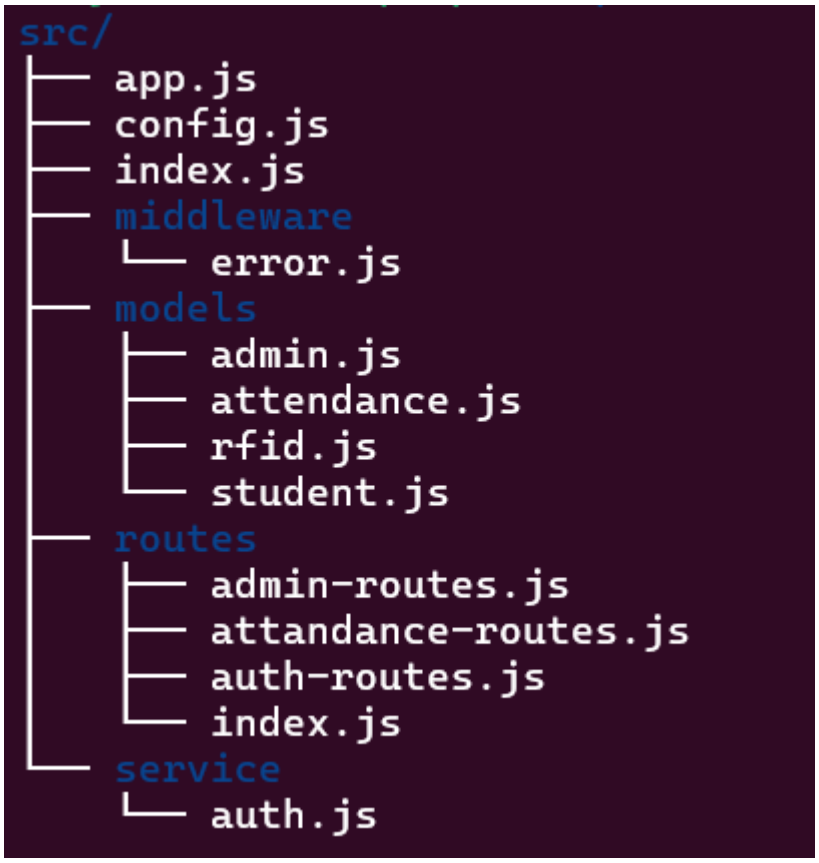


Рисунок 3.5 – Дерево файлової структури проєкту

Директорія src/ є головною директорією проекту, яка містить увесь вихідний код. Всі основні компоненти проекту розміщуються в цій папці для організації та

структурованості. Файл app.js є головним файлом налаштувань додатку. Тут визначається і конфігурується сам Express додаток, встановлюються middleware (проміжні програмні забезпечення), підключаються маршрути (routes) та інші важливі конфігурації.

Файл config.js використовується для зберігання конфігурацій додатку, таких як параметри підключення до бази даних, налаштування порту, та інші параметри, що можуть бути змінені в залежності від середовища виконання (розробка, тестування, продакшн). Основний файл для запуску додатку index.js підключає конфігурацію додатку з app.js, запускає сервер та здійснює первинні налаштування, такі як підключення до бази даних.

Директорія middleware/ призначена для зберігання проміжних програмних забезпечень (middleware). Всі файли, що розміщені тут, виконуються між отриманням запиту від клієнта та його обробкою у контролерах. Файл error.js є middleware для обробки помилок. Він використовується для перехоплення і обробки помилок, що виникають під час виконання запитів. Це допомагає виявити помилки та відправити коректну відповідь клієнту.

Директорія models/ містить моделі даних. Моделі відповідають за структуру та взаємодію з базою даних. Вони визначають схеми (schemas) даних та методи для роботи з ними. Файл rfid-entry.js містить модель для збереження даних, пов'язаних з RFID (Radio Frequency Identification) записами. Він містить визначення схеми даних для RFID-записів та методи для взаємодії з цими даними у базі даних. Файл user-entry.js містить модель для збереження даних користувачів. Він містить визначення схеми даних для користувачів та методи для взаємодії з ними у базі даних.

Директорія routes/ використовується для зберігання файлів маршрутизації. Маршрути визначають, як обробляти різні HTTP-запити (GET, POST, PUT, DELETE). Кожен файл у цій директорії відповідає за маршрутизацію конкретної частини функціоналу додатку. Файл admin-routes.js містить маршрути для адміністративної частини додатку. Він містить визначення маршрутів та обробників для адміністративних функцій, таких як управління користувачами

або перегляд статистики. Головний файл маршрутів index.js зазвичай об'єднує всі інші маршрути та підключає їх до основного додатку. Файл rfid-routes.js містить маршрути для роботи з RFID. Він містить визначення маршрутів та обробників для обробки RFID-запитів, таких як додавання нових RFID-записів або перегляд існуючих.

Директорія service/ призначена для зберігання сервісних файлів. Сервіси використовуються для виконання різних бізнес-логік додатку, таких як обробка даних, взаємодія з зовнішніми сервісами, виконання складних операцій та інші дії, що не належать безпосередньо до контролерів або моделей.

На сервері відбувається вся основна логіка проекту, яка забезпечує взаємодію з усіма іншими елементами системи, такими як база даних, зчитувач RFID та адмін панель. Серверна частина відіграє ключову роль у забезпеченні коректної роботи всієї системи обліку студентів.

Коли RFID зчитувач виявляє мітку, він відправляє HTTP-запит до веб-сервера з даними про мітку. Сервер приймає цей запит через спеціально налаштований маршрут (route) в Express. На цьому етапі сервер перевіряє отримані дані на коректність та існування у базі даних. Якщо мітка валідна, сервер зберігає дані про час входу або виходу студента до бази даних. Цей процес включає перевірку попереднього запису для визначення, чи це вхід, чи вихід. У майбутньому це можна реалізувати за допомогою різних зчитувачів для входу та виходу.

Сервер використовує ORM (Object-Relational Mapping) Mongoose для MongoDB, для взаємодії з базою даних. Всі операції з даними, такі як зберігання нових записів про відвідуваність, оновлення інформації про студентів та отримання звітів про відвідуваність, виконуються через сервер. Він забезпечує безпечний доступ до бази даних, керуючи підключенням та виконанням запитів. Крім того, сервер забезпечує обробку помилок, які можуть виникнути під час взаємодії з базою даних.

Вся бізнес-логіка проекту реалізована на сервері. Це включає в себе обробку даних від зчитувача, перевірку валідності міток, розрахунок часу перебування студентів у навчальному закладі та генерацію звітів.

Створення та реєстрація нових RFID тегів починається з отримання унікального ідентифікатора тегу (tagId), який передається до сервера через відповідний API запит.

Коли система отримує запит на реєстрацію нового RFID тегу, вона спочатку перевіряє наявність тегу в базі даних. Якщо тег вже існує, система перевіряє, чи призначений він студенту та чи підтверджений він. Якщо тег не призначений, система повертає помилку, вказуючи на те, що тег не призначений жодному студенту. Якщо ж тег не знайдено в базі даних, система створює новий запис для цього тегу. Новий RFID тег реєструється з параметрами "assigned" та "confirmed", встановленими на значення "false". Це означає, що тег ще не призначений жодному студенту та не підтверджений для використання в системі.

Після створення нового запису, система зберігає його в базі даних і повертає клієнту інформацію про новостворений тег. Цей процес забезпечує, що всі нові RFID теги спочатку реєструються як непідтверджені та не призначені, що дозволяє адміністратору системи або іншому уповноваженому користувачу підтвердити та призначити тег студенту через інтерфейс користувача.

3.3 Розробка системи перегляду та адміністрування

Адмін панель – це веб-інтерфейс, через який адміністратори можуть керувати системою. Вона також взаємодіє з сервером через HTTP-запити. Адміністратори можуть додавати нові мітки, переглядати та оновлювати інформацію про студентів, а також генерувати та переглядати звіти про відвідуваність. Сервер обробляє ці запити, виконуючи необхідні операції з базою даних та повертаючи результати у вигляді JSON-відповідей.

Фронт-енд частина проєкту розроблена для забезпечення адміністративного інтерфейсу системи відстеження відвідуваності студентів, який дозволяє зручно керувати студентами та їхніми RFID тегами. Ця система є важливою частиною комплексного рішення, яке інтегрує апаратне та програмне забезпечення для автоматичного контролю доступу та реєстрації відвідуваності.

Основою інтерфейсу є використання React, сучасної бібліотеки для створення користувацьких інтерфейсів, що забезпечує високу продуктивність і гнучкість у розробці компонентів. Стилізація інтерфейсу здійснюється за допомогою Tailwind CSS, що дозволяє швидко та ефективно створювати адаптивний дизайн. Запити до серверної частини виконуються за допомогою Axios, що забезпечує надійний та зручний спосіб взаємодії з сервером через HTTP.

Інтерфейс додатку організований таким чином, щоб забезпечити легкий доступ до основних функцій системи. Головний макет сторінки (MainLayout.tsx) включає загальні елементи, такі як заголовок (Header.tsx) та основний контент, що робить навігацію інтуїтивно зрозумілою. Це створює узгоджену структуру сторінок, яка полегшує користувачам взаємодію з системою.

Компоненти для відображення даних, такі як Card.tsx та Modal.tsx, відіграють важливу роль у візуалізації інформації. Компонент Card.tsx використовується для відображення інформаційних карток, що можуть містити дані про студентів, їхні відвідування, або іншу важливу інформацію. Компонент Modal.tsx забезпечує модальні вікна для підтвердження дій, редагування даних або інших інтерактивних взаємодій з користувачем, що робить інтерфейс більш інтерактивним та зручним.

Однією з ключових функцій фронт-енд частини є обробка даних про RFID теги. Коли новий тег зчитується, він додається до бази даних як непідтверджений і не прив’язаний до жодного студента. Сторінка "rfidTags" у додатку забезпечує зручний інтерфейс для адміністраторів, де вони можуть переглядати всі зчитані теги. Тут адміністратори можуть підтверджувати теги, змінюючи їх статус на "підтверджений" (confirmed). Ця дія дозволяє тегам бути доступними для

прив'язки до студентів, що робить їх готовими до використання в системі відстеження відвідуваності.

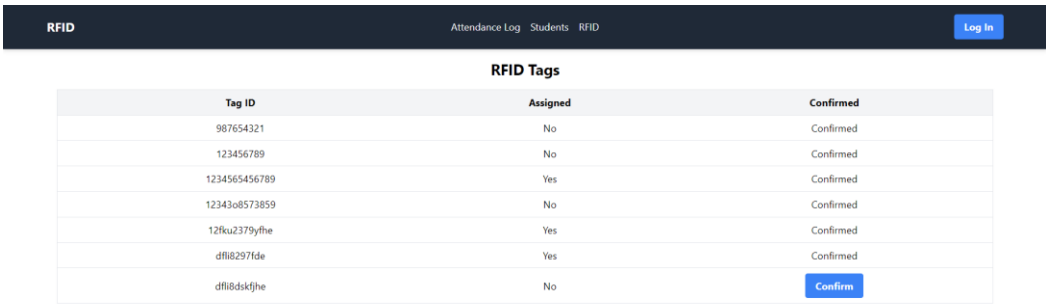


Рисунок 3.6 – Сторінка Rfid тегів

Сторінка "Students" надає можливості для створення нових записів студентів або редагування існуючих. Адміністратори можуть вводити основні дані про студента, такі як ім'я, прізвище, та електронну адресу, а також прив'язувати підтверджені RFID теги до студентів. Цей процес змінює статус тегу на "прив'язаний", що дозволяє використовувати його для реєстрації відвідуваності. Таким чином, взаємодія з RFID тегами та студентами забезпечує плавну інтеграцію апаратного та програмного забезпечення в рамках системи.

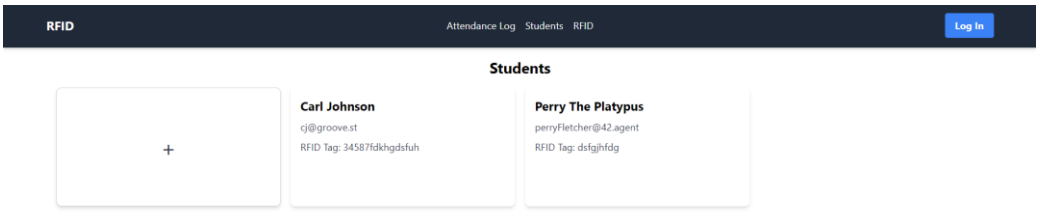


Рисунок 3.7 – Сторінка Студентів

Форма створення та редагування студентів є важливою частиною інтерфейсу додатку. Вона дозволяє адміністраторам вводити та оновлювати дані про студентів, забезпечуючи точність та актуальність інформації в системі. У формі передбачена можливість вибору RFID тегу з доступного списку. В цьому списку відображаються лише ті теги, які вже підтверджені, але ще не прив'язані до жодного студента. Це забезпечує коректний процес прив'язки, уникаючи дублювання та помилок, і дозволяє адміністраторам легко та швидко управляти даними про студентів та їхні RFID теги.

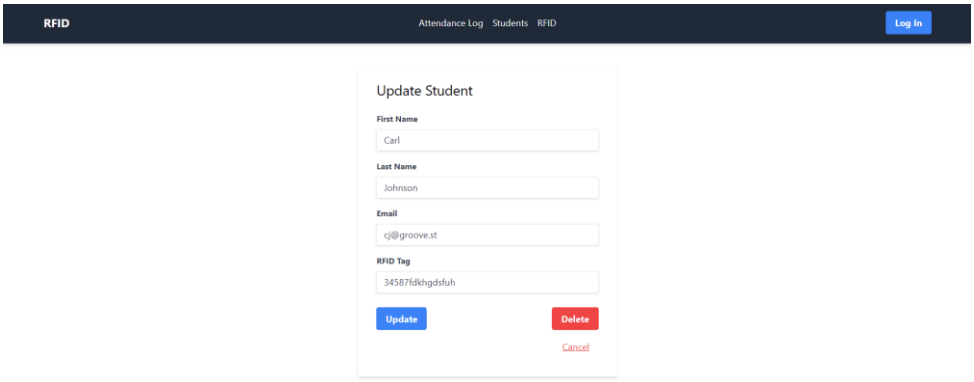


Рисунок 3.8 – Сторінка створення та редагування студентів

На сторінці з логами відвідування відображаються всі записи, що зберігаються в базі даних. Кожен запис містить інформацію про студента, його RFID тег, час та дату події, а також тип події (вхід або вихід). Ці дані допомагають відслідковувати, коли саме студент входив або виходив з університету, забезпечуючи детальний хронологічний журнал відвідуваності. Однією з важливих функцій цієї сторінки є автоматичне визначення типу події (вхід або вихід) на основі останнього запису для конкретного RFID тегу. Коли студент використовує свій RFID тег, система автоматично фіксує подію як вхід, якщо остання зареєстрована подія була вихід, і навпаки.

Інтерфейс сторінки з логами відвідування забезпечує зручну навігацію та доступ до потрібної інформації. Адміністратори можуть використовувати фільтри для пошуку записів за певним студентом, датою або типом події. Це допомагає швидко знаходити необхідні дані і проводити аналіз відвідуваності. Крім того, передбачена можливість експорту логів для подальшого аналізу або звітності, що додає додаткову гнучкість у роботі з даними.

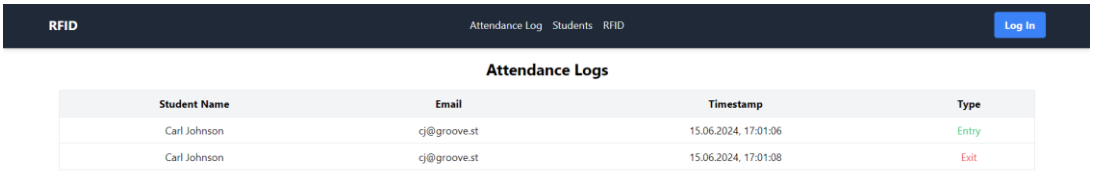


Рисунок 3.9 – Сторінка перегляду усіх записів про відвідування

Таким чином, сторінка з логами відвідування є ключовим елементом системи відстеження відвідуваності, що забезпечує прозорість, контроль та зручність в управлінні даними. Вона дозволяє адміністраторам ефективно відслідковувати активність студентів, забезпечуючи надійність і точність системи відвідуваності.

Таким чином, фронт-енд частина проєкту виконує критичну роль у забезпеченні зручного та ефективного інтерфейсу для керування системою відстеження відвідуваності студентів. Вона забезпечує інтерактивність, динамічність та реальний час оновлення інформації, що є важливими елементами для успішного функціонування системи. Завдяки продуманому інтерфейсу та функціоналу, адміністратори можуть легко управляти даними про студентів та їхні RFID теги, забезпечуючи надійність та ефективність системи.

ВИСНОВКИ

Під час кваліфікаційної роботи було здійснено створення системи відстеження відвідуваності студентів за допомогою RFID-технологій. Першим етапом роботи стала підготовка, яка включала в себе визначення основних цілей та завдань проєкту, а також підбір необхідного апаратного та програмного забезпечення. У ході підготовки було детально проаналізовано існуючі рішення у сфері систем відстеження відвідуваності, що дало змогу визначити найефективніші підходи та технології для реалізації проєкту. На основі аналізу було обґрунтовано вибір модуля RFID RC522 як оптимального рішення для зчитування даних з RFID-тегів.

Наступним етапом стала розробка апаратної частини системи. Було обрано плату Raspberry Pi Pico W, яка забезпечила необхідну функціональність та можливості для інтеграції з RFID-зчитувачем. Встановлення та налаштування модуля RFID RC522 включали підключення його до Raspberry Pi Pico W та налаштування програмного забезпечення для зчитування даних з RFID-тегів. Цей етап вимагав ретельної роботи з апаратним забезпеченням для забезпечення стабільного та надійного зчитування даних.

Після успішного завершення апаратної частини розпочалася розробка програмного забезпечення. Основна увага була приділена створенню серверної частини системи на базі Node.js та Express. Серверна частина відповідала за обробку даних, отриманих від RFID-зчитувача, та зберігання їх у базі даних MongoDB. Для роботи з базою даних використовувалася бібліотека Mongoose, що забезпечувала зручний інтерфейс для взаємодії з MongoDB. Крім того, було реалізовано підтримку WebSocket-з'єднання для забезпечення реального часу оновлень та взаємодії між сервером та клієнтською частиною системи.

Клієнтська частина була реалізована у вигляді веб-інтерфейсу для адміністраторів системи. Використання бібліотеки React дозволило створити зручний та інтуїтивно зрозумілий інтерфейс, що полегшував роботу з системою. Для керування станом застосовувалася бібліотека Jotai, а для стилізації інтерфейсу

– Tailwind CSS. Веб-інтерфейс забезпечував можливість перегляду логів відвідувань, підтвердження або відхилення нових RFID-тегів, що значно спрощувало адміністрування системи.

Після завершення розробки усіх компонентів системи було проведено їх інтеграцію та тестування. Тестування включало перевірку роботи системи на реальних даних для забезпечення її надійності та точності. Результати тестування показали високу ефективність системи у відстеженні відвідуваності студентів, що підтвердило правильність вибору технологій та підходів.

У підсумку, розроблена система відстеження відвідуваності студентів з використанням RFID-технологій показала високу ефективність та надійність. Використання обґрунтованого вибору апаратного та програмного забезпечення дозволило досягти поставлених цілей та виконати всі завдання дипломної роботи. Подальший розвиток системи може включати розширення функціоналу, поліпшення продуктивності та масштабованості, а також розробку мобільного додатку для зручності користування системою адміністраторами та студентами. Проведена робота є важливим кроком у напрямку автоматизації процесів відстеження відвідуваності, що сприяє підвищенню ефективності управління навчальними закладами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Ель Гаабурі І., Сенхаджі М., Белкасмі М., Ель Бхірі Б. Систематичний огляд літератури з аутентифікації та загроз безпеці в додатках RFID на основі NFC Базель: MDPI, 2023. 354 с. - (Future Internet 2023, 15, 354)
2. Лазаро А., Гірбау Д. Спеціальне видання: Near-Field Communication (NFC) та Radio Frequency Identification (RFID) Sensors. Таррагона: Університет Ровіра і Віргілі, 2022.
3. Фріт Й. RFID, NFC, маяки та інфраструктури логістичних локативних медіа Оксфорд: Оксфордський університет, 2020. 475-486 с.
4. Статлер С., Спрінгер. Штрих-коди, QR-коди, NFC та RFID, 2016.
5. Clarity AG - вакансії, робота, відгуки про компанію. Офіційний сайт порталу DOU. 2021. URL: https://jobs.dou.ua/companies/clarity-ag/ (дата звернення: 24.04.2024)
6. Clarity AG - The flexible communication solution. 2021. URLсайт компанії Clarity AG. URL: https://clarity-ag.de/en/company/ (дата звернення: 24.04.2024)
7. RS-485/232 Card Systems ТКП-32-04/2. Основи RS232, RS422 та RS485 Серійного Зв’язку. URL: https://automationcommunity.com/rs232-rs422-rs485/ (дата звернення: 28.04.2024)
8. Контролери КСКД2-3К та КСКД2-12К. Універсальний контролер КСКД3-12К. URL: https://card-sys.com/ru/products/Universalni_kontroller_KSKD3-12K (дата звернення: 28.04.2024)
9. Програмне забезпечення “АСКД”. Візуалізація даних системи контролю доступу “STOP-Net”.
10. Мікросервісна архітектура. Підхід до розробки програмного забезпечення з набором невеликих, слабко пов’язаних сервісів.
11. PN532 NFC модуль. RFID/NFC модуль, що працює на частоті 13.56 МГц і підтримує комунікацію через SPI, I2C та UART / PN532 NFC модуль. URL:https://web.kpi.kharkov.ua/mto/wpcontent/uploads/sites/references_style_guide_UKR.pdf. (дата звернення: 03.05.2024)

12. RC522 RFID / NFC модуль. Модуль, базований на контролері MFRC522, що працює на частоті 13.56 МГц / RC522 RFID/NFC модуль. URL: https://www.electroschematics.com/nfc-rfid-module-pn532/ . (дата звернення: 03.05.2024)

13. NXP EdgeVerse™ NFC Readers. Серія NFC-читачів від NXP для читання та запису карток та міток. URL: https://www.nxp.com/products/wireless-connectivity/nfc-hf/nfc-readers:NFC-READER

14. ARM Cortex-M0 NFC контролери. Мікроконтролери з ARM Cortex-M0, що інтегрують NFC-фронтенд. URL: https://www.arm.com/media/Arm%20Developer%20Community/PDF/Processor%20Datasheets/Arm_Cortex-M0_Processor_Datasheet.pdf

15. MySQL/MariaDB. Популярні реляційні бази даних, використовуються від малих веб-сайтів до великих корпоративних систем. URL: https://mariadb.com/kb/en/getting-the-mariadb-source-code/. (дата звернення: 03.05.2024)

16. PostgreSQL. Потужна реляційна база даних з відкритим кодом, відома надійністю та високою продуктивністю URL: https://www.postgresql.org/ (дата звернення: 03.05.2024)

17. Microsoft SQL Server. Реляційна база даних від Microsoft, інтегрована з іншими продуктами компанії URL: https://www.microsoft.com/en-us/sql-server/sql-server-downloads (дата звернення: 04.05.2024)

18. Oracle Database. Одна з найпотужніших реляційних баз даних, підходить для великих систем URL: https://www.oracle.com/database/ (дата звернення: 04.05.2024)

19. MongoDB. Документ-орієнтована база даних з гнучкою схемою документів у форматі JSON URL: https://github.com/mongodb/mongo (дата звернення: 04.05.2024)

20. Cassandra. Розподілена колоночна база даних, розроблена для високої доступності та масштабованості URL: https://github.com/apache/cassandra (дата звернення: 04.05.2024)

21. Redis. Ключ-значення база даних, яка зберігає дані у пам’яті для швидкого доступу URL: https://github.com/redis/redis (дата звернення: 04.05.2024)

22. Databases: Library of Congress Electronic Resources Online Catalog. URL: https://eresources.loc.gov/ (дата звернення: 04.05.2024)

23. APA Databases and Electronic Resources. American Psychological Association. URL: https://www.apa.org/pubs/databases/ (дата звернення: 04.05.2024)

24. Databases: Library of Congress Electronic Resources Online Catalog URL: https://eresources.loc.gov/ (дата звернення: 04.05.2024)

25. Introduction to Web Servers. DigitalOcean. 2022. URL: https://www.digitalocean.com/community/conceptual-articles/introduction-to-web-servers (дата звернення: 04.05.2024)

26. What is a web server? MDN Web Docs. URL: https://developer.mozilla.org/enUS/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server (дата звернення: 06.05.2024)

27. React Resources. ReactResources.com. URL: https://reactresources.com/ (дата звернення: 06.05.2024)

28. Home - Svelte Society. Svelte Society. URL: https://www.sveltesociety.dev/ (дата звернення: 06.05.2024)

29. 8 Best resources to learn Node.js as of 2024. Slant. 2024. URL: https://www.slant.co/topics/1042/~best-resources-to-learn-node-js (дата звернення: 06.05.2024)

30. ASP.NET Web Page Resources Overview. Microsoft Learn - 2014. URL: https://learn.microsoft.com/en-us/previous-versions/aspnet/ms227427%28v=vs.100%29 (дата звернення: 06.05.2024)

31. Smart lock picking - BLE, NFC, RFID навчання, підручники, хакінг, ресурси. URL: https://www.smartlockpicking.com. (дата звернення: 06.05.2024)

32. RFID та NFC системи контролю доступу. URL:https://www.getkisi.com. (дата звернення: 07.05.2024)

33. Керівництво щодо забезпечення безпеки радіочастотних ідентифікаційних систем (RFID). URL: (https://www.nist.gov. (дата звернення: 07.05.2024)

34. Бездротовий NFC RFID зчитувач/записувач. URL: https://www.d-logic.com. (дата звернення: 07.05.2024)

35. NFC Tools Online. Платформа, що пропонує інструменти для зчитування та запису даних на NFC теги. URL: https://nfctools.net. (дата звернення: 07.05.2024)

36. GS1 US. URL: https://www.gs1us.org. (дата звернення: 07.05.2024)

37. Deepnet Security URL: https://deepnetsecurity.com. (дата звернення: 07.05.2024)

38. PN7150 | NFC Plug-and-Play Controller | NXP Semiconductors. URL: https://www.nxp.com. (дата звернення: 11.05.2024)

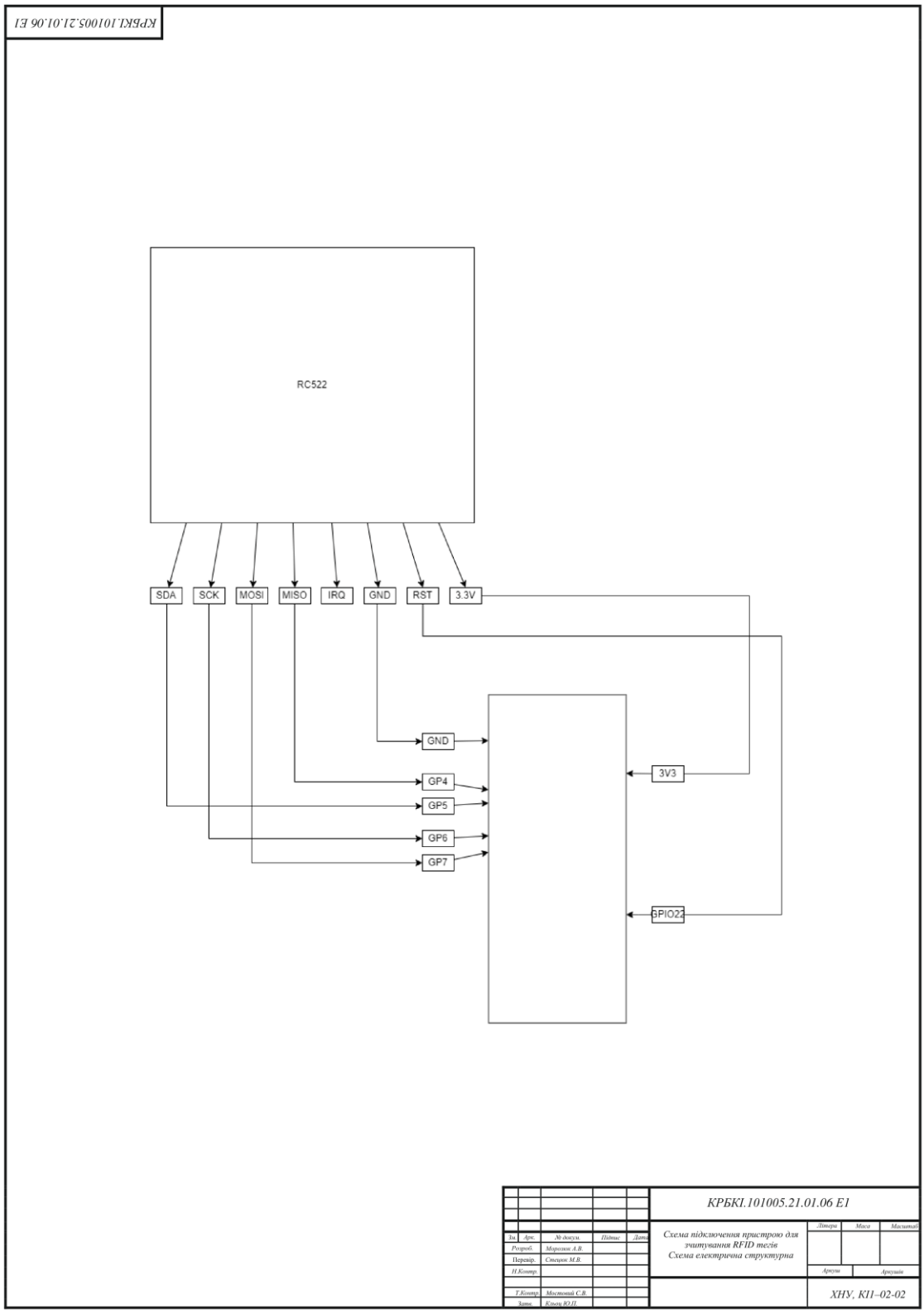
39. GitHub - Offensive-Wireless/Flipper-Zero. URL: https://github.com/Offensive-Wireless/Flipper-Zero. (дата звернення: 13.05.2024)

40. Як швидко зчитувати та перевіряти NFC/RFID браслети на заходах - CodeREADr. URL: https://www.codereadr.com (дата звернення: 13.05.2024)

ДОДАТОК А

(обов’язковий)

Копія графічної частини



КРБКІ.101005.21.01.06 Е2

Зм.	Арк.	№ докум.	Підпис	Дата	КРБКІ.101005.21.01.06 Е2
Розроб.		Морозюк А.В.			Система безконтактного контролю відвідуваності студентів Алгоритм роботи
Перевір.		Стецюк М.В.			
Н.Контр.					
Т.Контр.		Мостовий С.В.			
Затв.		Кошиц Ю.Л.			

Літера	Маса	Масштаб
Аркуш	Аркушів	

ХНУ, КІ1–02-02

КРБКІ.101005.21.01.06 Е3

					КРБКІ.101005.21.01.06 Е3
Зм.	Арк.	№ докум.	Підпис	Дата	
Розроб.		Морозюк А.В.			Діаграма архітектури проєкту
Перевір.		Стецюк М.В.			
Н.Контр.					
Т.Контр.		Мостовий С.В.			
Затв.		Ковш Ю.П.			

Літера	Маса	Масштаб
Аркуш	Аркушів	

ХНУ, КІ1–02-02

ДОДАТОК Б

(обов’язковий)

Лістинг коду для зчитування RFID

```
#rfid/main.py
import network
import socket
import time
from machine import Pin
from mfrc522 import MFRC522
from led import LED
from config import *
import urequests as requests
from wifi import connect_to_wifi

def main():
    connect_to_wifi(WIFI_SSID, WIFI_PASSWORD)

    led = LED(Pin(LED_PIN, Pin.OUT))
    reader = MFRC522(spi_clk=14, spi_mosi=13, spi_miso=12, spi_cs=15, gpio_rst=2)

    while True:
        (stat, tag_type) = reader.request(reader.REQIDL)

        if stat == reader.OK:
            (stat, raw_uid) = reader.anticoll()

            if stat == reader.OK:
                uid = ".join(['%02X' % x for x in raw_uid])
                print("Card detected. UID: %s" % uid)
```

```
            led.blink(0.1, 5)

            response = requests.post(SERVER_URL, json={'uid': uid})
            print(response.json())

        time.sleep(1)

if __name__ == "__main__":
    main()

#rfid/http.py
import urequests as requests

def post_request(url, data):
    response = requests.post(url, json=data)
    return response.json()

#rfid/led.py
from machine import Pin
import time

class LED:
    def __init__(self, pin):
        self.pin = pin

    def blink(self, duration, times):
        for _ in range(times):
            self.pin.on()
            time.sleep(duration)
            self.pin.off()
```

```
            time.sleep(duration)
```

#rfid/mfrc522.py

```python
from machine import Pin, SPI

class MFRC522:
    REQIDL = 0x26
    OK = 0
    NOTAGERR = 1
    ERR = 2

    def __init__(self, spi_clk, spi_mosi, spi_miso, spi_cs, gpio_rst):
        self.sck = Pin(spi_clk, Pin.OUT)
        self.mosi = Pin(spi_mosi, Pin.OUT)
        self.miso = Pin(spi_miso)
        self.cs = Pin(spi_cs, Pin.OUT)
        self.rst = Pin(gpio_rst, Pin.OUT)
        self.spi = SPI(1, baudrate=1000000, polarity=0, phase=0, sck=self.sck, mosi=self.mosi, miso=self.miso)
        self.init()

    def init(self):
        self.rst.on()
        self.cs.on()
        self.rst.off()
        self.cs.off()
        # Initialization procedure for the RFID reader
```

```
    def request(self, req_mode):
        # Function to request RFID tag
        return (self.OK, "TAG_TYPE")

    def anticoll(self):
        # Function for anti-collision detection
        return (self.OK, [0xDE, 0xAD, 0xBE, 0xEF])
```

#rfid/wifi.py

```python
import network
import time

def connect_to_wifi(ssid, password):
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, password)

    while wlan.isconnected() == False:
        print('Connecting to network...')
        time.sleep(1)

    print('Network config:', wlan.ifconfig())
```

#rfid/config.py

```
WIFI_SSID = 'your_ssid'
WIFI_PASSWORD = 'your_password'
LED_PIN = 2
SERVER_URL = 'http://example.com/api/rfid'
```

Лістинг коду для NodeJS сервера

```
#src/app.js
import express from "express";
import router from "./routes/index.js";
import attendanceRouter from "./routes/attandance-routes.js";
import adminRouter from "./routes/admin-routes.js";
import authRouter from "./routes/auth-routes.js"
import errorHandler from "./middleware/error.js";
import cors from 'cors'
import { authenticateToken } from "./service/auth.js";

const app = express();

app.use(express.json());
app.use(cors())

app.use("/api", router);
app.use("/attendance", attendanceRouter);
app.use("/admin", authenticateToken, adminRouter);
// app.use("/admin", adminRouter);
app.use("/auth", authRouter)

app.get("/", (req, res) => {
  res.send("Express + TypeScript Server");
});

app.use(errorHandler);

export default app;
```

```
#src/config.js
import dotenv from 'dotenv'

dotenv.config();

const config = {
  web_port: process.env.WEB_PORT || 3000,
  db_url: process.env.MONGO_DB_URL || '',
  jwt_secret: process.env.JWT_SECRET || ''
}

export default config



#src/index.js
import mongoose from "mongoose";
import config from "./config.js";
import app from "./app.js";

mongoose.connect(config.db_url).then(() => {
  console.log('MongoDB connected');
}).catch((error) => {
  console.error('Error connecting to MongoDB:', error);
  // process.exit(1); // Exit process with failure
});

app.listen(config.web_port, () => {
  console.log(`[server]: Server is running at http://localhost:${config.web_port}`);
});
```

#src/middleware

```
const errorHandler = (err, req, res, next) => {
  // Error handling logic
  const { code, message } = err

  console.log(`Code: ${code}, Message: ${message}`)
  res.status(500).send('Something went wrong!');
}

export default errorHandler
```

#src/models/admin.js

```
import mongoose from 'mongoose';

const adminSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
});

const Admin = mongoose.model('Admin', adminSchema);
```

```
export default Admin;


#src/model/attendance.js
import mongoose from 'mongoose';

const attendanceSchema = new mongoose.Schema({
  student: { type: mongoose.Schema.Types.ObjectId, ref: 'Student', required: true
},
  timestamp: { type: Date, required: true, default: Date.now },
  type: { type: String, required: true, enum: ['Entry', 'Exit'] }
});

const Attendance = mongoose.model('Attendance', attendanceSchema);
export default Attendance;


#src/models/rfid.js
import mongoose from 'mongoose';

const rfidTagSchema = new mongoose.Schema({
  tagId: { type: String, required: true, unique: true },
  assigned: { type: Boolean, default: false },
  confirmed: { type: Boolean, default: false }
});

const RFIDTag = mongoose.model('RFIDTag', rfidTagSchema);
export default RFIDTag;
```

```
#src/models/student.js
import mongoose from "mongoose";
const { Schema, model } = mongoose;

const studentSchema = new Schema({
  firstName: { type: String, required: true },
  lastName: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  rfidTag: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "RFIDTag",
    required: true,
  },
});

const Student = model("Student", studentSchema);
export default Student;



#src/routes/admin-routes.js
import express from "express";
import Student from "../models/student.js";
import Attendance from "../models/attendance.js";
import RFIDTag from "../models/rfid.js";
import httpStatus from "http-status";

const router = express.Router();

router
  .route("/students")
```

```
.get(async (req, res) => {
  try {
    const students = await Student.find().populate("rfidTag");
    res.status(httpStatus.OK).json(students);
  } catch (err) {
    res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error: err.message });
  }
})
.post(async (req, res) => {
  const { studentId, firstName, lastName, email, rfidTagId } = req.body;
  try {
    const rfidTag = await RFIDTag.findById(rfidTagId);
    if (!rfidTag || rfidTag.assigned || !rfidTag.confirmed) {
      return res.status(httpStatus.BAD_REQUEST).json({
        error: "Invalid, unconfirmed, or already assigned RFID tag",
      });
    }

    const newStudent = new Student({
      studentId,
      firstName,
      lastName,
      email,
      rfidTag: rfidTag._id,
    });

    const savedStudent = await newStudent.save();
    rfidTag.assigned = true;
    await rfidTag.save();
```

```
        res.status(httpStatus.CREATED).json(savedStudent);
      } catch (err) {
        res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error: err.message });
      }
    });

  router
    .route("/students/:id")
    .get(async (req, res) => {
      try {
        const student = await Student.findById(req.params.id).populate("rfidTag");
        if (!student) {
          return res
            .status(httpStatus.NOT_FOUND)
            .json({ error: "Student not found" });
        }
        res.status(httpStatus.OK).json(student);
      } catch (err) {
        res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error: err.message });
      }
    })
    .put(async (req, res) => {
      const { firstName, lastName, email, rfidTagId } = req.body;
      try {
        const student = await Student.findById(req.params.id);
        if (!student) {
          return res
```

```
      .status(httpStatus.NOT_FOUND)
      .json({ error: "Student not found" });
    }

    if (rfidTagId && rfidTagId !== student.rfidTag.toString()) {
      const newRFIDTag = await RFIDTag.findById(rfidTagId);
      if (!newRFIDTag || newRFIDTag.assigned || !newRFIDTag.confirmed) {
        return res.status(httpStatus.BAD_REQUEST).json({
          error: "Invalid, unconfirmed, or already assigned new RFID tag",
        });
      }

      const oldRFIDTag = await RFIDTag.findById(student.rfidTag);
      oldRFIDTag.assigned = false;
      await oldRFIDTag.save();

      newRFIDTag.assigned = true;
      await newRFIDTag.save();

      student.rfidTag = newRFIDTag._id;
    }

    student.firstName = firstName || student.firstName;
    student.lastName = lastName || student.lastName;
    student.email = email || student.email;

    const updatedStudent = await student.save();
    res.status(httpStatus.OK).json(updatedStudent);
  } catch (err) {
```

```
        res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error:
err.message });
      }
    })
    .delete(async (req, res) => {
      try {
        const student = await Student.findById(req.params.id);
        if (!student) {
          return res
            .status(httpStatus.NOT_FOUND)
            .json({ error: "Student not found" });
        }

        const attendanceRecords = await Attendance.find({
          student: req.params.id,
        });
        if (attendanceRecords.length > 0) {
          return res
            .status(httpStatus.BAD_REQUEST)
            .json({ error: "Cannot delete student with attendance records" });
        }

        await Student.findByIdAndDelete(req.params.id);

        res.status(httpStatus.OK).json({ message: "Student deleted" });
      } catch (err) {
        res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error:
err.message });
      }
    });
```

```
router.route("/rfid-tags").get(async (req, res) => {
  try {
    const rfidTags = await RFIDTag.find();
    res.status(httpStatus.OK).json(rfidTags);
  } catch (err) {
    res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error: err.message });
  }
});

router.route("/rfid-tags/available").get(async (req, res) => {
  try {
    const availableRfidTags = await RFIDTag.find({
      assigned: false,
      confirmed: true,
    });
    res.status(httpStatus.OK).json(availableRfidTags);
  } catch (err) {
    res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error: err.message });
  }
});

router
  .route("/rfid-tags/:id")
  .get(async (req, res) => {
    try {
      const rfidTag = await RFIDTag.findById(req.params.id);
      if (!rfidTag) {
```

```
            return res
              .status(httpStatus.NOT_FOUND)
              .json({ error: "RFID tag not found" });
          }
          res.status(httpStatus.OK).json(rfidTag);
        } catch (err) {
          res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error: err.message });
        }
      })
      .patch(async (req, res) => {
        const { assigned, confirmed } = req.body;
        try {
          const rfidTag = await RFIDTag.findById(req.params.id);
          if (!rfidTag) {
            return res
              .status(httpStatus.NOT_FOUND)
              .json({ error: "RFID tag not found" });
          }
          if (assigned && !rfidTag.confirmed) {
            return res
              .status(httpStatus.BAD_REQUEST)
              .json({ error: "Cannot assign an unconfirmed RFID tag" });
          }

          rfidTag.assigned = assigned !== undefined ? assigned : rfidTag.assigned;
          rfidTag.confirmed =
            confirmed !== undefined ? confirmed : rfidTag.confirmed;

          const updatedRFIDTag = await rfidTag.save();
```

```
        res.status(httpStatus.OK).json(updatedRFIDTag);
      } catch (err) {
        res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error: err.message });
      }
    });

export default router;


#src/routes/attendance-routes.js
import express from "express";
import Attendance from "../models/attendance.js";
import Student from "../models/student.js";
import httpStatus from "http-status";
import RFIDTag from "../models/rfid.js";

const router = express.Router();

router
  .route("/")
  .get(async (req, res) => {
    try {
      const attendanceRecords = await Attendance.find().populate("student");
      res.status(httpStatus.OK).json(attendanceRecords);
    } catch (err) {
      res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error: err.message });
    }
  })
```

```
.post(async (req, res) => {
  const { tagId, timestamp } = req.body;

  try {
    let rfidTag = await RFIDTag.findOne({ tagId });
    if (!rfidTag) {
      rfidTag = new RFIDTag({ tagId, assigned: false, confirmed: false });
      await rfidTag.save();
      return res.status(httpStatus.CREATED).json(rfidTag);
    }

    if (!rfidTag.assigned) {
      return res
        .status(httpStatus.BAD_REQUEST)
        .json({ error: "RFID tag not assigned to any student" });
    }

    const student = await Student.findOne({ rfidTag: rfidTag._id });
    if (!student) {
      return res
        .status(httpStatus.NOT_FOUND)
        .json({ error: "Student not found for this RFID tag" });
    }

    // Get the last attendance record for the student
    const lastAttendance = await Attendance.findOne({ student: student._id })
      .sort({ timestamp: -1 });

    // Determine the type of the new attendance record
```

```
        const newType = lastAttendance && lastAttendance.type === "Entry" ? "Exit" : "Entry";

        const newAttendance = new Attendance({
          student: student._id,
          timestamp: timestamp || Date.now(),
          type: newType,
        });

        await newAttendance.save();
        res.status(httpStatus.CREATED).json(newAttendance);
      } catch (err) {
        res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error: err.message });
      }
    });

router
  .route("/:id")
  .get(async (req, res) => {
    try {
      const attendanceRecord = await Attendance.findById(
        req.params.id
      ).populate("student");
      if (!attendanceRecord) {
        return res
          .status(httpStatus.NOT_FOUND)
          .json({ error: "Attendance record not found" });
      }
      res.status(httpStatus.OK).json(attendanceRecord);
```

```
    } catch (err) {
      res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error: err.message });
    }
  })
  .delete(async (req, res) => {
    try {
      const attendanceRecord = await Attendance.findByIdAndDelete(
        req.params.id
      );
      if (!attendanceRecord) {
        return res
          .status(httpStatus.NOT_FOUND)
          .json({ error: "Attendance record not found" });
      }
      res.status(httpStatus.OK).json({ message: "Attendance record deleted" });
    } catch (err) {
      res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error: err.message });
    }
  })
  .put(async (req, res) => {
    const { type, timestamp } = req.body;
    try {
      const attendanceRecord = await Attendance.findById(req.params.id);
      if (!attendanceRecord) {
        return res
          .status(httpStatus.NOT_FOUND)
          .json({ error: "Attendance record not found" });
      }
```

```
        attendanceRecord.type = type || attendanceRecord.type;
        attendanceRecord.timestamp = timestamp || attendanceRecord.timestamp;
        await attendanceRecord.save();
        res.status(httpStatus.OK).json(attendanceRecord);
      } catch (err) {
        res.status(httpStatus.INTERNAL_SERVER_ERROR).json({ error: err.message });
      }
    });

export default router;
```

```
#src/routes/auth-routes.js
import express from "express";
import httpStatus from "http-status";
import Admin from "../models/admin.js";
import {
  hashPassword,
  comparePassword,
  generateToken,
  authenticateToken,
} from "../service/auth.js";

const router = express.Router();

router.post("/register", async (req, res) => {
  const { username, password } = req.body;

  try {
```

```
    const existingUser = await Admin.findOne({ username });
    if (existingUser) {
      return res
        .status(httpStatus.CONFLICT)
        .json({ message: "User already exists" });
    }

    const hashedPassword = await hashPassword(password);
    const newAdmin = new Admin({ username, password: hashedPassword });
    await newAdmin.save();

    const token = generateToken(newAdmin);
    res.status(httpStatus.CREATED).json({
      message: "User registered successfully",
      token,
      user: { id: newAdmin._id, username: newAdmin.username },
    });
  } catch (err) {
    res
      .status(httpStatus.INTERNAL_SERVER_ERROR)
      .json({ message: "Error registering user", error: err.message });
  }
});

router.post("/login", async (req, res) => {
  const { username, password } = req.body;

  try {
    const user = await Admin.findOne({ username });
    if (!user) {
```

```
      return res
        .status(httpStatus.NOT_FOUND)
        .json({ message: "User not found" });
    }

    const isMatch = await comparePassword(password, user.password);
    if (!isMatch) {
      return res
        .status(httpStatus.UNAUTHORIZED)
        .json({ message: "Invalid credentials" });
    }

    const token = generateToken(user);
    res.json({
      token,
      user: { id: user._id, username: user.username },
    });
  } catch (err) {
    res
      .status(httpStatus.INTERNAL_SERVER_ERROR)
      .json({ message: "Error logging in", error: err.message });
  }
});

router.get("/verify-token", authenticateToken, (req, res) => {
  res.status(httpStatus.OK).json({ message: "Token is valid", user: req.user });
});

router.get("/protected", authenticateToken, (req, res) => {
  res.json({ message: `Hello, ${req.user.username}` });
```

```
});

export default router;



#src/routes/index.js
import express from "express"

const router = express.Router()

router.get("/test", (req, res) => {
  res.send('Hello world from /test endpoint \n')
})

router.get("/error", (req,res) => {
  const error = new Error('A test error occurred');
  error.code = 42;
  throw error;
})

export default router



#src/service/auth.js
import jwt from "jsonwebtoken";
import bcrypt from "bcryptjs";
import config from "../config.js";
import httpStatus from "http-status";

const SECRET_KEY = config.jwt_secret;
```

```
export const hashPassword = async (password) => {
  const salt = await bcrypt.genSalt(10);
  return await bcrypt.hash(password, salt);
};

export const comparePassword = async (inputPassword, storedPassword) => {
  return await bcrypt.compare(inputPassword, storedPassword);
};

export const generateToken = (user) => {
  return jwt.sign({ id: user.id, username: user.username }, SECRET_KEY, {
    expiresIn: "1h",
  });
};

export const authenticateToken = (req, res, next) => {
  const authHeader = req.header("Authorization");
  if (!authHeader)
    return res
      .status(httpStatus.UNAUTHORIZED)
      .json({ message: "No token, authorization denied" });

  const token = authHeader.split(' ')[1];
  if (!token)
    return res
      .status(httpStatus.UNAUTHORIZED)
      .json({ message: "Token is not valid" });

  const decoded = verifyToken(token);
```

```
  if (!decoded)
    return res
      .status(httpStatus.UNAUTHORIZED)
      .json({ message: "Token is not valid" });

  req.user = decoded;
  next();
};

const verifyToken = (token) => {
  try {
    return jwt.verify(token, SECRET_KEY);
  } catch (err) {
    return null;
  }
};
```

Лістинг коду адмін панелі на React

#scr/App.tsx

```
import { Route, Routes, BrowserRouter, Navigate } from 'react-router-dom';
import AttendaceLog from './pages/AttendanceLog';
import Login from './pages/Login';
import Registration from './pages/Registration';
import MainLayout from './components/layout/MainLayout';
import RFIDList from './pages/RfidTags';
import axios from './services/axios';
import { useEffect, useState } from 'react';
import StudentForm from './pages/StudentForm';
import Students from './pages/Students';
```

```
function App() {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  useEffect(() => {
    const token = localStorage.getItem('token');
    if (token) {
      axios.get('/auth/verify-token', { headers: { Authorization: `Bearer ${token}` } }
})
        .then(response => {
          setIsAuthenticated(true);
        })
        .catch(err => {
          setIsAuthenticated(false);
          localStorage.removeItem('token');
        });
    }
  }, []);

  return (
    <BrowserRouter>
      <Routes>
        <Route path="/login" element={<Login />} />
        <Route path="/register" element={<Registration />} />
        <Route element={<MainLayout isAuthenticated={isAuthenticated} />}>
          <Route path="/" element={
            <h1 className="text-center text-3xl font-bold mt-8">Welcome</h1>
          } />
          <Route path="/attendance-log" element={<AttendaceLog />} />
          <Route path="/students" element={<Students />} />
```

```
          <Route path="/rfid" element={<RFIDList />} />
          <Route path="/students/new" element={<StudentForm />} />
          <Route path="/students/:id" element={<StudentForm />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}

export default App
```

#src/components/common/Card.tsx

```
import React from 'react';
import { useNavigate } from 'react-router-dom';

interface CardProps {
  children: React.ReactNode;
  className?: string;
  onClick?: () => void;
}

const Card: React.FC<CardProps> = ({ children, className = '', onClick }) => {
  const navigate = useNavigate();

  const handleClick = () => {
    if (onClick) {
      onClick();
    }
  };
```

```
  return (
    <div
      className={`bg-white shadow-md rounded-lg p-4 transition-shadow duration-300 hover:shadow-lg cursor-pointer ${className}`}
      onClick={handleClick}
    >
      {children}
    </div>
  );
};

export default Card;
```

#src/components/common/

```
// src/components/Modal.tsx
import React, { ReactNode } from 'react';

interface ModalProps {
  showModal: boolean;
  closeModal: () => void;
  children: ReactNode;
}

const Modal: React.FC<ModalProps> = ({ showModal, closeModal, children }) => {
  if (!showModal) return null;

  return (
```

```
        <div className="fixed inset-0 flex items-center justify-center bg-black bg-opacity-50 backdrop-blur-sm">
          <div className="bg-white rounded shadow-lg w-96 p-6">
            <div className="flex justify-end">
              <button onClick={() => closeModal()} className="text-gray-500 hover:text-gray-700">
                &times;
              </button>
            </div>
            {children}
          </div>
        </div>
      );
    };

    export default Modal;


    #src/components/layout/Header.tsx
    import React from "react";
    import { Link, useNavigate } from "react-router-dom";

    const Header: React.FC = () => {
      const navigate = useNavigate();

      return (
        <header className="bg-gray-800 text-white p-4 shadow-md">
          <div className="container mx-auto grid grid-cols-3 items-center">
            <div className="text-xl font-bold col-span-1">RFID</div>
            <nav className="col-span-1">
```

```
  <ul className="flex justify-center space-x-4">
    <li>
      <Link
        to="/attendance-log"
        className="hover:text-gray-400 transition duration-300"
      >
        Attendance Log
      </Link>
    </li>
    <li>
      <Link
        to="/students"
        className="hover:text-gray-400 transition duration-300"
      >
        Students
      </Link>
    </li>
    <li>
      <Link
        to="/rfid"
        className="hover:text-gray-400 transition duration-300"
      >
        RFID
      </Link>
    </li>
  </ul>
</nav>
<div className="col-span-1 flex justify-end">
  <button
    onClick={() => navigate('/login')}
```

```
              className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded transition duration-300"
            >
              Log In
            </button>
          </div>
        </div>
      </header>
    );
  };

  export default Header;


  #src/components/layout/MainLayout.tsx
  import React from "react";
  import { Link, useNavigate } from "react-router-dom";

  const Header: React.FC = () => {
    const navigate = useNavigate();

    return (
      <header className="bg-gray-800 text-white p-4 shadow-md">
        <div className="container mx-auto grid grid-cols-3 items-center">
          <div className="text-xl font-bold col-span-1">RFID</div>
          <nav className="col-span-1">
            <ul className="flex justify-center space-x-4">
              <li>
                <Link
                  to="/attendance-log"
```

```
            className="hover:text-gray-400 transition duration-300"
          >
            Attendance Log
          </Link>
        </li>
        <li>
          <Link
            to="/students"
            className="hover:text-gray-400 transition duration-300"
          >
            Students
          </Link>
        </li>
        <li>
          <Link
            to="/rfid"
            className="hover:text-gray-400 transition duration-300"
          >
            RFID
          </Link>
        </li>
      </ul>
    </nav>
    <div className="col-span-1 flex justify-end">
      <button
        onClick={() => navigate('/login')}
        className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded transition duration-300"
      >
        Log In
```

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

export default Header;
```

#src/pages/AttendanceLog.tsx

```
import React, { useEffect, useState } from 'react';
import axios from '../services/axios';

interface TAttendanceLog {
  _id: string;
  student: {
    _id: string;
    firstName: string;
    lastName: string;
    email: string;
  };
  timestamp: string;
  type: string;
}

const AttendanceLog: React.FC = () => {
  const [logs, setLogs] = useState<TAttendanceLog[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
```

```
  useEffect(() => {
    const fetchAttendanceLogs = async () => {
      try {
        const response = await axios.get('/attendance');
        setLogs(response.data);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchAttendanceLogs();
  }, []);

  if (loading) return <div className="text-center py-4">Loading...</div>;
  if (error) return <div className="text-center text-red-500 py-4">Error: {error}</div>;

  return (
    <div className="container mx-auto p-4">
      <h1 className="text-2xl font-bold mb-4 text-center">Attendance Logs</h1>
      <table className="min-w-full bg-white border rounded-lg">
        <thead>
          <tr>
            <th className="py-2 px-4 bg-gray-100 border-b">Student Name</th>
            <th className="py-2 px-4 bg-gray-100 border-b">Email</th>
            <th className="py-2 px-4 bg-gray-100 border-b">Timestamp</th>
            <th className="py-2 px-4 bg-gray-100 border-b">Type</th>
```

```
            </tr>
          </thead>
          <tbody>
            {logs.map((log) => (
              <tr key={log._id} className="text-center">
                <td className="py-2 px-4 border-b">{`${log.student.firstName} ${log.student.lastName}`}</td>
                <td className="py-2 px-4 border-b">{log.student.email}</td>
                <td className="py-2 px-4 border-b">{new Date(log.timestamp).toLocaleString()}</td>
                <td className={`py-2 px-4 border-b ${log.type === 'Entry' ? 'text-green-500' : 'text-red-500'}`}>
                  {log.type}
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    );
  };

  export default AttendanceLog;


  #src/pages/Login.tsx
  import React, { useState } from 'react';
  import axios from '../services/axios';
  import { useNavigate, Link } from 'react-router-dom';
```

```
const Login = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [message, setMessage] = useState('');
  const navigate = useNavigate();

  const handleLogin = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post('/auth/login', { username, password });
      setMessage('Login successful');
      localStorage.setItem('token', response.data.token);
      navigate('/');
    } catch (err) {
      setMessage(err.response.data.message);
    }
  };

  return (
    <div className="flex justify-center items-center min-h-screen bg-gray-100">
      <div className="bg-white p-8 rounded-lg shadow-md w-96">
        <h2 className="text-2xl font-bold mb-6 text-center">Login</h2>
        <form onSubmit={handleLogin}>
          <div className="mb-4">
            <label className="block text-gray-700">Username</label>
            <input
              type="text"
              className="w-full px-3 py-2 border rounded-lg"
              value={username}
              onChange={(e) => setUsername(e.target.value)}
```

```
            />
          </div>
          <div className="mb-4">
            <label className="block text-gray-700">Password</label>
            <input
              type="password"
              className="w-full px-3 py-2 border rounded-lg"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
            />
          </div>
          <button
            type="submit"
            className="w-full bg-blue-500 text-white py-2 rounded-lg hover:bg-blue-600 transition duration-300"
          >
            Login
          </button>
        </form>
        {message && <p className="mt-4 text-red-500 text-center">{message}</p>}
        <p className="mt-4 text-center">
          Don't have an account? <Link to="/register" className="text-blue-500 hover:underline">Register</Link>
        </p>
      </div>
    </div>
  );
};
```

```
export default Login;
```

#src/pages/Registration.tsx

```
import React, { useState } from 'react';
import axios from '../services/axios';
import { Link, useNavigate } from 'react-router-dom';

const Register = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const [message, setMessage] = useState('');
  const navigate = useNavigate();

  const handleRegister = async (e) => {
    e.preventDefault();
    if (password !== confirmPassword) {
      setMessage('Passwords do not match');
      return;
    }
    try {
      const response = await axios.post('/auth/register', { username, password });
      setMessage(response.data.message);
      localStorage.setItem('token', response.data.token); // Save token to local storage
      navigate('/');
    } catch (err) {
      setMessage(err.response.data.message);
    }
```

```
};

return (
  <div className="flex justify-center items-center min-h-screen bg-gray-100">
    <div className="bg-white p-8 rounded-lg shadow-md w-96">
      <h2 className="text-2xl font-bold mb-6 text-center">Register</h2>
      <form onSubmit={handleRegister}>
        <div className="mb-4">
          <label className="block text-gray-700">Username</label>
          <input
            type="text"
            className="w-full px-3 py-2 border rounded-lg"
            value={username}
            onChange={(e) => setUsername(e.target.value)}
          />
        </div>
        <div className="mb-4">
          <label className="block text-gray-700">Password</label>
          <input
            type="password"
            className="w-full px-3 py-2 border rounded-lg"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
          />
        </div>
        <div className="mb-4">
          <label className="block text-gray-700">Confirm Password</label>
          <input
            type="password"
            className="w-full px-3 py-2 border rounded-lg"
```

```
            value={confirmPassword}
            onChange={(e) => setConfirmPassword(e.target.value)}
           />
          </div>
          <button
           type="submit"
           className="w-full bg-blue-500 text-white py-2 rounded-lg hover:bg-blue-600 transition duration-300"
          >
           Register
          </button>
         </form>
         {message && <p className="mt-4 text-red-500 text-center">{message}</p>}
         <p className="mt-4 text-center">
          Want to Log in? <Link to="/login" className="text-blue-500 hover:underline">Login</Link>
         </p>
        </div>
       </div>
    );
};

export default Register;
```

#src/pages/RfidTags.tsx

```
import React, { useEffect, useState } from 'react';
import axios from '../services/axios';
import Modal from '../components/common/Modal';
```

```
interface RFIDTag {
  _id: string;
  tagId: string;
  assigned: boolean;
  confirmed: boolean;
}

const RFIDList: React.FC = () => {
  const [rfidTags, setRfidTags] = useState<RFIDTag[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
  const [showModal, setShowModal] = useState(false);
  const [selectedTagId, setSelectedTagId] = useState<string | null>(null);

  useEffect(() => {
    const fetchRfidTags = async () => {
      try {
        const response = await axios.get('/admin/rfid-tags');
        setRfidTags(response.data);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchRfidTags();
  }, []);
```

```
  const handleConfirmClick = (tagId: string) => {
    setSelectedTagId(tagId);
    setShowModal(true);
  };

  const confirmTag = async () => {
    if (selectedTagId) {
      try {
        await axios.patch(`/admin/rfid-tags/${selectedTagId}`, { confirmed: true });
        setRfidTags(rfidTags.map(tag => tag._id === selectedTagId ? { ...tag, confirmed: true } : tag));
        setShowModal(false);
      } catch (err) {
        setError(err.message);
      }
    }
  };

  if (loading) return <div className="text-center py-4">Loading...</div>;
  if (error) return <div className="text-center text-red-500 py-4">Error: {error}</div>;

  return (
    <div className="container mx-auto p-4">
      <h1 className="text-2xl font-bold mb-4 text-center">RFID Tags</h1>
      <table className="min-w-full bg-white border rounded-lg">
        <thead>
          <tr>
            <th className="py-2 px-4 bg-gray-100 border-b">Tag ID</th>
            <th className="py-2 px-4 bg-gray-100 border-b">Assigned</th>
```

```
            <th className="py-2 px-4 bg-gray-100 border-b">Confirmed</th>
          </tr>
        </thead>
        <tbody>
          {rfidTags.map(tag => (
            <tr key={tag._id} className="text-center">
              <td className="py-2 px-4 border-b">{tag.tagId}</td>
              <td className="py-2 px-4 border-b">{tag.assigned ? 'Yes' : 'No'}</td>
              <td className="py-2 px-4 border-b">
                {!tag.confirmed ? (
                  <button
                    onClick={() => handleConfirmClick(tag._id)}
                    className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline"
                  >
                    Confirm
                  </button>
                ) : (
                  <span>Confirmed</span>
                )
                }
              </td>
            </tr>
          ))}
        </tbody>
      </table>

      <Modal showModal={showModal} closeModal={() => setShowModal(false)}>
        <div className="text-center">
```

```
            <h2 className="text-xl mb-4">Confirm RFID Tag</h2>
            <p>Are you sure you want to confirm this RFID tag?</p>
            <div className="mt-4 flex justify-center">
              <button
                onClick={confirmTag}
                className="bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline mr-2"
              >
                Confirm
              </button>
              <button
                onClick={() => setShowModal(false)}
                className="bg-red-500 hover:bg-red-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline"
              >
                Cancel
              </button>
            </div>
          </div>
        </Modal>
      </div>
    );
  };

  export default RFIDList;


  #src/pages/StudentForm.tsx
  import React, { useState, useEffect } from 'react';
  import axios from '../services/axios';
```

```
import { useNavigate, useParams } from 'react-router-dom';
import Modal from '../components/common/Modal';

const StudentForm = () => {
  const [student, setStudent] = useState({
    firstName: '',
    lastName: '',
    email: '',
    rfidTagId: '',
  });
  const [rfidTags, setRfidTags] = useState([]);
  const [error, setError] = useState('');
  const [showModal, setShowModal] = useState(false);
  const [deleteError, setDeleteError] = useState('');
  const navigate = useNavigate();
  const { id } = useParams();

  useEffect(() => {
    const fetchRfidTags = async () => {
      try {
        const response = await axios.get('/admin/rfid-tags/available');
        setRfidTags(response.data);
      } catch (error) {
        console.error('There was an error fetching the RFID tags!', error);
      }
    };

    const fetchStudent = async () => {
      if (id) {
        try {
```

```
            const response = await axios.get(`/admin/students/${id}`);
            const studentData = response.data;
            setStudent({
              firstName: studentData.firstName,
              lastName: studentData.lastName,
              email: studentData.email,
              rfidTagId: studentData.rfidTag._id,
            });

            // Add the student's current RFID tag to the list if it's not already included
            setRfidTags(prevTags => {
              const isTagPresent = prevTags.some(tag => tag._id === studentData.rfidTag._id);
              const updatedTags = isTagPresent ? prevTags : [...prevTags, studentData.rfidTag];
              // Ensure the selected tag appears first in the list
              return updatedTags.sort(tag => tag._id === studentData.rfidTag._id ? -1 : 1);
            });
          } catch (error) {
            console.error('There was an error fetching the student!', error);
          }
        }
      };

      fetchRfidTags();
      fetchStudent();
    }, [id]);

    const handleChange = (e) => {
```

```
  const { name, value } = e.target;
  setStudent({ ...student, [name]: value });
};

const handleSubmit = (e) => {
  e.preventDefault();
  setError(''); // Clear previous errors
  if (id) {
    axios.put(`/admin/students/${id}`, student)
      .then(() => {
        navigate('/students');
      })
      .catch(error => {
        console.error('There was an error updating the student!', error);
        if (error.response && error.response.data.error) {
          setError(error.response.data.error);
        }
      });
  } else {
    axios.post('/admin/students', student)
      .then(() => {
        navigate('/students');
      })
      .catch(error => {
        console.error('There was an error creating the student!', error);
        if (error.response && error.response.data.error) {
          setError(error.response.data.error);
        }
      });
  }
```

```
    };

    const handleDelete = () => {
      setDeleteError(''); // Clear previous delete errors
      axios.delete(`/admin/students/${id}`)
        .then(() => {
          navigate('/students');
        })
        .catch(error => {
          console.error('There was an error deleting the student!', error);
          setShowModal(false);
          if (error.response && error.response.data.error) {
            setDeleteError(error.response.data.error);
          }
        });
    };

    const handleCancel = () => {
      navigate(-1); // Go back to the previous page
    };

    return (
      <div className="max-w-md mx-auto mt-10">
        <form onSubmit={handleSubmit} className="bg-white shadow-md rounded px-8 pt-6 pb-8 mb-4">
          <h2 className="text-2xl mb-6">{id ? 'Update Student' : 'Create Student'}</h2>
          {error && (
            <div className="mb-4 text-red-500 text-sm">
              {error}
```

```
            </div>
          )}
          <div className="mb-4">
            <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="firstName">
              First Name
            </label>
            <input
              id="firstName"
              name="firstName"
              type="text"
              value={student.firstName}
              onChange={handleChange}
              className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
              required
            />
          </div>
          <div className="mb-4">
            <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="lastName">
              Last Name
            </label>
            <input
              id="lastName"
              name="lastName"
              type="text"
              value={student.lastName}
              onChange={handleChange}
```

```
            className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
            required
          />
        </div>
        <div className="mb-4">
          <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="email">
            Email
          </label>
          <input
            id="email"
            name="email"
            type="email"
            value={student.email}
            onChange={handleChange}
            className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
            required
          />
        </div>
        <div className="mb-4">
          <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="rfidTagId">
            RFID Tag
          </label>
          <select
            id="rfidTagId"
            name="rfidTagId"
            value={student.rfidTagId}
```

```
            onChange={handleChange}
            className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
            required
          >
            <option value="" disabled hidden>Select RFID Tag</option>
            {rfidTags.map(tag => (
              <option
                key={tag._id}
                value={tag._id}
                className={tag._id === student.rfidTagId ? 'bg-gray-200' : ''}
              >
                {tag.tagId}
              </option>
            ))}
          </select>
        </div>
        <div className="flex items-center justify-between mt-6">
          <button
            type="submit"
            className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline"
          >
            {id ? 'Update' : 'Create'}
          </button>
          {id && (
            <button
              type="button"
              onClick={() => setShowModal(true)}
```

```
                className="bg-red-500 hover:bg-red-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline"
                >
                 Delete
                </button>
               )}
              </div>
              <div className="flex justify-end mt-2 py-2 px-4">
               <button
                type="button"
                onClick={handleCancel}
                className="text-red-500 underline focus:outline-none"
               >
                Cancel
               </button>
              </div>
             </form>

             {deleteError && (
              <div className="mb-4 text-red-500 text-sm text-center">
               {deleteError}
              </div>
             )}

             <Modal showModal={showModal} closeModal={() => setShowModal(false)}>
              <h2 className="text-xl font-bold mb-4">Delete Student</h2>
              <p className="mb-4">Are you sure you want to delete this student?</p>
              <div className="flex justify-between">
               <button
```

```
              onClick={handleDelete}
              className="bg-red-500 hover:bg-red-700 text-white font-bold py-2 px-4 rounded"
            >
              Delete
            </button>
            <button
              onClick={() => setShowModal(false)}
              className="bg-gray-500 hover:bg-gray-700 text-white font-bold py-2 px-4 rounded"
            >
              Cancel
            </button>
          </div>
        </Modal>
      </div>
    );
  };

  export default StudentForm;


  #src/pages/Students.tsx
  import React, { useEffect, useState } from 'react';
  import axios from '../services/axios';
  import Card from '../components/common/Card';
  import { Link, useNavigate } from 'react-router-dom';

  interface RFIDTag {
    _id: string;
```

```
  tagId: string;
}

interface Student {
  _id: string;
  firstName: string;
  lastName: string;
  email: string;
  rfidTag: RFIDTag;
}

const Students: React.FC = () => {
  const [students, setStudents] = useState<Student[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
  const navigate = useNavigate();

  useEffect(() => {
    const fetchStudents = async () => {
      try {
        const response = await axios.get('/admin/students');
        setStudents(response.data);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchStudents();
```

```
  }, []);

  if (loading) return <div className="text-center py-4">Loading...</div>;
  if (error) return <div className="text-center text-red-500 py-4">Error: {error}</div>;

  const handleCardClick = (id: string) => {
    navigate(`/students/${id}`);
  };

  return (
    <div className="container mx-auto p-4">
      <h1 className="text-2xl font-bold mb-4 text-center">Students</h1>
      <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-4">
        <Link to="/students/new" className="h-48">
          <Card className="flex items-center justify-center h-full bg-gray-100 border border-gray-300 hover:bg-gray-200 shadow-inner">
            <div className="text-4xl text-gray-700">+</div>
          </Card>
        </Link>

        {students.map(student => (
          <Card key={student._id} onClick={() => handleCardClick(student._id)} className="h-48">
            <h2 className="text-xl font-bold mb-2">{student.firstName} {student.lastName}</h2>
            <p className="text-gray-700 mb-2">{student.email}</p>
            <p className="text-gray-700">RFID Tag: {student.rfidTag.tagId}</p>
          </Card>
```

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

export default Students;
```

#src/services/auth.ts

```
import axios from "axios";
import config from '../config.ts'

const API_URL = config.api_base_url;
console.log(API_URL);

export const login = (username, password) => {
  return axios.post(`${API_URL}/auth/login`, {
    username,
    password,
  })
  .then(response => response.data.token)
  .catch(error => {
    throw new Error(error.response?.data?.message || "Invalid credentials");
  });
};
```

#src/services/axios.ts

```
import axios from "axios";
```

```
import config from "../config";

const getToken = () => localStorage.getItem("token");

const instance = axios.create({
  baseURL: config.api_base_url,
  timeout: 1000,
});

instance.interceptors.request.use(
  (config) => {
    const token = getToken();
    console.log("Sending token:", token); // Debugging line
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

export default instance;
```

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Морозюка Андрія Володимировича
ПІБ здобувача вищої освіти

Студента ФІТ, 3 курсу, групи Кі1с-21-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у хмельницькому національному університеті» від 31.08.2023, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

06.06
дата

підпис

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

ID: 131414 Назва: Система безконтактного контролю відвідуваності студентів за допомогою RFID/NFC міток Додано в БД: 2024-06-18 Автора: Морозюк А.В. Керівники: Стецюк М.В. Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	82424	627	828 (1%)	10 (2%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра кібербезпеки

Дата перевірки:
18.06.2024 22:26:40 EEST

Дата звіту:
18.06.2024 22:35:46 EEST

ID перевірки:
1016373396

Тип перевірки:
Doc vs Internet + Library

ID користувача:
100008300

Назва документа: **Морозюк_А_В_КІ1с_21_1_плагіат**

Кількість сторінок: **61** Кількість слів: **11421** Кількість символів: **87647** Розмір файлу: **4.40 MB** ID файлу: **1016180883**

1.94%
Схожість

Найбільша схожість: **1.22%** з джерелом з Бібліотеки (ID файлу: **1016174946**)

1.13% Джерела з Інтернету	155	Сторінка 63
1.39% Джерела з Бібліотеки	102	Сторінка 64

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текетових збігів/ідентичності/схожості:

Назва: Система безконтактного контролю відвідуваності студентів за допомогою RFID/NFC міток

Автор: Морозюк Андрій Володимирович

Спеціальність: 123 – Комп'ютерна інженерія

Освітня програма: програмування та захист комп'ютерних систем та мереж

Науковий керівник: Стецюк Микола Васильович.

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою Unichek складає 98.06%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism v-15.257 складає 99%.

Згідно з Положенням про систему забезпечення академічної доброчесності у ХНУ (https://khmnu.edu.ua/wp-content/uploads/normatyvni-dokumenty/polozhennya/pro-systemu-zabezpechennya-akademichnoyi-dobrochesnosti.pdf, Додаток В) кваліфікаційна робота, виконана за , освітньо-професійною програмою, кількісні показники рівня унікальності тексту у відсотках до загального обсягу матеріалу в якій складає 75-100 %, визнається роботою з високою унікальністю тексту: «Текст вважається унікальним і не потребує додаткових дій щодо запобігання неправомірним запозиченням».

Керівник роботи _____ Микола СТЕЦЮК

Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Морозюк Андрій Володимирович

Тема Система безконтактного контролю відвідуваності студентів за допомогою RFID/NFC міток

Спеціальність 125 – Кібербезпека

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:

кількість листів креслень 3 ; кількість сторінок записки 64 .

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі була розроблена система контролю відвідуваності студентів на основі RFID міток. Ця система має вбудований захист від витоку інформації. У процесі проєктування були розроблені такі компоненти: система зчитування RFID міток, система валідації міток та зберігання інформації про відвідуваність.

2. Висновок про відповідність кваліфікаційної роботи завданню У кваліфікаційній роботі було виконано поставлене завдання як у теоретичній, так і в практичній частині.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі роботи наведена загальна характеристика задачі, визначені об'єкт, предмет та методи дослідження, а також сформульована мета. Зазначені задачі, що потрібно виконати для досягнення поставленої мети, проведений аналіз досліджуваної проблеми та обгрунтований підхід до її вирішення. У першому розділі розглядаються об'єкти захисту інформації та системи контролю доступу. Наступні розділи присвячені розробці системи контролю відвідуваності на основі RFID міток та тестуванню готового обладнання. Також був проведений економічний розрахунок системи.

4. Позитивні сторони роботи Кваліфікаційна робота має практичну цінність. Вона полягає у розробці системи контролю відвідуваності стідентів на основі RFID міток, що спрощує наявну систему обліку відвідуваності студентів. Завдяки цьому підприємство є має можливість записувати відвідуваність в автоматичному режимі. При проєктуванні системи обліку відвідуваності студентів було використано мікроконтролер та датчик зчитування міток.

5. Негативні сторони роботи В системі не передбачено автоматичну зміну мережі у разі втрати з'єднання, що робить систему залежною від стабільності з'єднання до мережі. Також система не передбачає сповіщення про різні відповіді він серверу.

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення кваліфікаційної роботи відповідає темі роботи та виконане з дотриманням стандартів. В цілому, графічне оформлення є якісним, а пояснювальна записка відповідає нормам оформлення.

7. Відгук про роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки, оскільки весь матеріал роботи є структурованим, чітким та послідовним. Усі розділи роботи мають логічну послідовність, що сприяє зрозумінню викладеного матеріалу в рамках теми роботи. Графічний матеріал допомагає наочно продемонструвати доцільність та ефективність прийнятих рішень для досягнення мети.

8. Інші зауваження В переліку використаних джерел наявні посилання на популярні ресурси та онлайн документації, які не рекомендовано використовувати при написанні кваліфікаціних робіт.

9. Оцінка кваліфікаційної роботи Враховуючи всі позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінки «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Підченко Сергій Костянтинович,
завідувач кафедри ТМІТ, доктор технічних наук, професор

«19» червня 2024. (підпис)