МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра комп'ютерних наук та прикладної математики

“До захисту допущена”

Зав. кафедри комп'ютерних наук

та прикладної математики

Юрій Турбал

“__” _____20_р.

КВАЛІФІКАЦІЙНА РОБОТА

Апаратний інтерфейс у програмно-апаратних комплексів систем пропуску

Виконав:

Кравчук Василь Олександрович _____

Група ІПЗ-41

Керівник:

Прищепа О.В., к.ф.-м.н., доцент _____

Рівне-2024

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 60 с., 23 рисунків, 24 джерел.

Мета роботи: Розробка апаратно-програмного комплексу автоматизованої системи контролю пропуску для навчального закладу на основі технологій платформи інтернету речей.

Об'єкт дослідження: Автоматизовані системи контролю пропуску для вищих навчальних закладів.

Предмет дослідження: Розробка комплексу автоматизованої системи пропуску для навчального закладу з динамічною системою надання доступу.

Методи вивчення: платформа інтернету речей, NodeJS, NestJS

Здійснено аналіз готових впроваджених систем на ринку та сформовано головні функціональні вимоги до системи. Проаналізовано платформи та апаратні засоби для розробки апаратно-програмного комплексу. Виконано аналіз та порівняння існуючих систем колаборації процесу розробки програмного засобу. Виконано розробку та тестування апаратно-програмної системи термінала автоматичного контролю пропуску.

Ключові слова: апаратний інтерфейс, системи пропуску, інтернет речей, NodeJS, NestJS, автоматизація, безпека.

ВСТУП

Впродовж кількох останніх декад ми бачили велику хвилю тотальної цифровізації, це був тренд зумовлений підвищенням ефективності усього робочого процесу у разі переходу в цифру, починаючи від обміну інформацією, закінчуючи веденням звітності та нарахуванням заробітніх плат, і все завдяки поширенню комп'ютерних систем. Очікувано що цей тренд не оминув і навчальні заклади.

Поява систем націлених на оптимізацію та спрощення управління навчальним процесом була лише питанням часу. Ці системи взяли на себе все від адміністрування до створення навчальної платформи. Вони значно збільшили ефективність управлінського персоналу.

Попри свою беззаперечну користь як для працівників, так і для здобувачів освіти, все ж ці системи також мають свої недоліки та певні аспекти навчального процесу які вони не покривають повністю, або ж лише частково. Одними з таких аспектів є три важливі та пов'язані між собою проблеми: управління розкладом навчального процесу, ведення запису відвідуваності/присутності, пропуск та допуск студентів на територію навчального закладу та, або його територій та приміщень. Зв'язок між цими аспектами може не бути очевидним, проте для чого надавати студенту доступ до лабораторії з дороговартісним науковим обладнанням, якщо він не відвідує там занять, або ж не проводить свою наукову чи дослідну діяльність, також якщо студент був допущений до місця проведення тих чи інших занять, то можна припустити, що студент був присутнім на занятті.

З вищесказаного випливає потреба у створенні певного комплексу, системи для керування розкладом, автоматичного запису відвідуваності та пропуску студентів і захисту від сторонніх. Проект складатиметься з: серверного програмного забезпечення (API), що буде займатися веденням відвідуваності, збереженням і зміною розкладу та прийняттям рішення про пропуск студента;

апаратно-програмної системи аутентифікації користувача, що надсилатиме запит до API для отримання рішення про надання або ж ненадання доступу студенту до тієї чи іншої аудиторії обладнаної таким апаратно-програмним засобом.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРОБЛЕМИ

1.1. Чому навчальним закладам варто впроваджувати автоматичні системи контролю пропуску та запису відвідуваності

Традиційно управління пропуском та ведення журналів відвідуваності відбувалось вручну з використанням класичних паперових носіїв інформації. Працівники навчального закладу особисто вели запис відвідувань та приймали рішення про допуск конкретної особи на територію закладу або до окремого приміщення, території. Використання для цього виключно людської праці створює більшу ймовірність помилки та сповільнює навчальні процеси, а вкрай мала за сучасними мірками мобільність інформації на паперових носіях подеколи взагалі унеможливлювала прийняття коректного рішення про допуск здавалося б авторизованої особи до необхідного забезпечення, чи просто приміщення. Подекуди занадто великі затрати часу на запис відвідуваності, наприклад на відкритих для усіх студентів заходах, створюють невиправдано велике зменшення пропускної здатності охочих відвідувачів, що взагалі змушує відмовитись від запису відвідувань в конкретному випадку, попри потенційну користь наявності такого запису.

Використання автоматизованої системи пропуску дозволить суттєво пришвидшити процес контролю осіб для запобігання проникнення сторонніх, а головне, усуне фактор людської помилки та вирішить проблему прийняття некоректного рішення пов’язану з інформаційною немобільністю. У випадку термінового відсторонення тієї чи іншої особи від навчального процесу, наприклад у разі порушення певних правил чи наражання інших на небезпеку, відповідне рішення одразу набуде впливу, а відповідна особа не матиме змоги

скористатись затримкою в комунікації для отримання вже фактично несанкціонованого доступу.

Автоматизація процесу пропуску дозволяє зробити автоматичним також і ведення відвідуваності, адже знаючи час та місце вдалої спроби авторизації її можна співставити з поточним розкладом занять конкретного студента та зробити про це відповідний запис. Використання комп'ютерних систем для запису відвідувань дозволяє виключити людську помилку, а головне якоюсь мірою спростити роботи для викладачів усунувши потребу ручного та неефективного запису присутніх. Автоматизоване ведення відвідуваності також вирішує проблему значного зменшення пропускної здатності відвідувачів, що робить доцільним доцільним використання системи пропуску в ролі системи запису відвідуваності на добровільних подіях та заходах, в подальшому записи можуть використовуватися для оцінки активності студента чи навіть нарахування додаткових балів за прийняття участі в житті університету.

Наявність такої системи суттєво підвищує загальний рівень безпеки усіх причетних до навчання та ускладнить неавторизований доступ до дороговартісного матеріального забезпечення, що може призвести до невідворотної втрати його функціональних здатностей або навіть викрадення. Сама ж автоматизація дозволяє зменшити загальну кількість необхідного для функціонування персоналу, підвищення його продуктивності та зміни обов'язків, або ж його перехід до ролі спостерігачів, що лише сильніше підвищить рівень безпеки.

Саме через оптимізацію, пришвидшення та поліпшення вищенаведених аспектів навчального процесу вищим навчальним закладам варто впроваджувати подібні системи задля поліпшення зручності для викладачів та студентів, а головне, задля поліпшення безпеки працівників та здобувачів освіти.

1.2. Приклади впровадження аналогічних рішень у вищих навчальних закладах

Попри вкрай рідкісну появу автоматичних систем пропуску серед українських ВНЗ, серед навчальних закладів Європи є звичним явищем впровадження таких систем. У європейських університетах подібні системи не лише відповідають за контроль пропуску, але й за запис відвідуваності, що дозволяє сформувати єдину систему ВНЗ, яка позбавляє освітній процес зайвого клопоту не лише для студентів, але й для викладачів котрим більше не варто перейматись витрачанням дорогоцінного часу ведення списків присутніх на занятті.

Університет Барселони в Іспанії був одним з перших запроваджувачів автоматичних систем турнікетів. Університет інтегрував систему, яка використовує біометричні дані та студентські картки для доступу до кампусів. Ця система надає можливість не лише контролювати доступ до навчальних корпусів, але й автоматично реєструвати присутність студентів на заняттях. Таке рішення значно спростило процес ведення журналу відвідуваності, а також підвищило рівень безпеки на території університету, запобігаючи проникненню сторонніх осіб.

Ще одним прикладом є Мюнхенський університет Людвіга — Максиміліана, що впровадив систему автоматичних турнікетів на усіх входах до університетських будівель. Система включає використання RFID-карток, які студенти та співробітники отримують від навчального закладу. Турнікети зчитують інформацію з карток та надають доступ лише авторизованим користувачам. Окрім цього, система дозволяє відстежувати переміщення людей всередині будівель, що сприяє оперативному реагуванню у випадку надзвичайних ситуацій.

У Франції Університет Сорбонна також впровадив автоматичні турнікетні системи у кількох своїх кампусах. Університет використав систему на основі QR-кодів, що генеруються мобільним застосунком університету. Кожен студент генерує QR-код, що сканується на вході до будівлі. Ця система є зручною для студентів, оскільки не потребує додаткових карток, проте кожен студент має на собі смартфон, також дана система дозволяє адміністрації університету легко оновлювати та керувати доступом до різних зон кампусу.

У Нідерландах Університет Амстердаму встановив систему турнікетів, яка інтегрується в загальну систему безпеки міста. Використовуючи спеціальні студентські картки, система дозволяє не тільки контролювати доступ до університетських будівель, але й надає студентам доступ до міських бібліотек та інших громадських установ. Такий підхід створює єдину “екосистему” для студентів, роблячи їхнє життя більш зручним і безпечним.

Ці приклади демонструють, як автоматичні системи контролю пропуску можуть бути ефективно впроваджені у вищих навчальних закладах, як вони здатні підвищити рівень безпеки та комфорту студентів і співробітників та які варіації методів та підходів можливо застосувати.

РОЗДІЛ 2
РЕАЛІЗАЦІЯ API, ВИКОРИСТАНІ ЗАСОБИ ТА ПІДХОДИ

2.1. Постановка задачі

Проект передбачає розробку апаратно-програмного комплексу для системи пропуску, який включає серверне програмне забезпечення (API) та апаратно-програмну систему аутентифікації користувача. Система повинна забезпечувати:

1. Управління розкладом:
    a. Автоматичне ведення розкладу навчального процесу.
    b. Можливість динамічного оновлення розкладу в режимі реального часу.
2. Запис відвідуваності:
    a. Автоматичне реєстрація відвідуваності студентів на основі даних з апаратно-програмних терміналів.
    b. Збереження та обробка даних про відвідуваність для подальшого аналізу.
3. Контроль пропуску:
    a. Аутентифікація користувачів за допомогою NFC/RFID карток та QR-кодів.
    b. Прийняття рішень про допуск студентів на основі розкладу та записів відвідуваності.
    c. Інтеграція з мобільною мережею для віддаленого доступу.
4. Безпека та захист:
    a. Запобігання несанкціонованому доступу до навчальних приміщень.
    b. Підвищення загального рівня безпеки на території навчального закладу.

Система складатиметься з наступних компонентів:

1. Серверне програмне забезпечення для обробки запитів від терміналів та збереження даних.
2. Апаратно-програмні термінали для аутентифікації користувачів та збору даних про відвідуваність.
3. Інтерфейс для адміністраторів системи для управління розкладом, записами відвідуваності та налаштуваннями системи.

Основні вимоги до системи включають високу надійність, безперебійність роботи, захищеність від зовнішніх втручань та можливість масштабування для роботи з великим об'ємом даних та користувачів.

Цілі проекту:

- Створення ефективного інструменту для управління доступом та записом відвідуваності в навчальному закладі.
- Зниження навантаження на адміністративний персонал.
- Підвищення рівня безпеки та контролю в навчальному закладі.
- Забезпечення точності та своєчасності записів відвідуваності.

2.2. Підготовка системи контролю версій, обміну кодовою базою та автоматизації постачання та розгортання нових змін

Git є потужною системою для контролю версій та змін в кодовій базі проекта, проте сам по собі лише локально ініціалізований git репозиторій не надає можливості віддаленого обміну змінами та синхронізації репозиторію, а ручне внесення змін проекта на віддалений сервер та подальше розгортання шляхом виконання попередньо використаних велику кількість разів інструкцій створює потребу у використанні системи, що об’єднує в собі засоби для централізованого збереження та обміну змінами для синхронізації зусиль розробників, автоматичного вивантаження змін на сервер та їх подальшого розгортання згідно заданих інструкцій. Існує чимало відкритих для публіки подібних рішень, наприклад: GitLab, GitHub, Bitbucket, Gitea, SourceForge та

інші. Найцікавішим вибором серед перелічених є GitLab завдяки його багатому функціоналу та інструментарію для керування репозиторієм, потужні засоби для автоматичного впровадження і розгортання змін та безкоштовна можливість розгортання приватно сервера з GitLab. Приватний GitLab сервер надає можливість швидкого розгортання змін, дозволяє економити кошти для окремих індивідів та малих організацій, надає можливість адміністрування всієї системи сховища репозиторіїв та забезпечення високого рівня безпеки. Звичайно, аналоги такі як Gitea та SourceForge також мають базовий функціонал як для керування репозиторієм, та і для автоматичного розгортання, проте вони переважно покривають лише саме “базовий функціонал”, а їх розробники не мають такий же великий кредит довіри як і у випадку з GitLab.

В якості апаратної платформи для розгортання GitLab було обрано одноплатний комп’ютер Raspberry Pi 4 в комплектації з 4 гігабайтами оперативної пам’яті. Дана платформа була обрана через невисоку вартість, мале енергоспоживання, мінімальне тепловиділення, широку підтримку багатьма популярними дистрибутивами сімейства Linux, надійність, детальну документацію та наявність. Для живлення дана платформа потребує USB блок живлення з вихідними напругою та струмом 5 вольт і 3 ампери відповідно.

Операційною системою для сервера було обрано сучасний дистрибутив Linux openSUSE MicroOS [11]. Дана операційна система має невеликі системні вимоги, автоматичні атомарні оновлення, можливість відновлення системи з попередньої точки в часі на рівні файлової системи BTRFS [12], автоматичне повернення стану операційної системи в разі невдалого оновлення.

Атомарні оновлення – це метод коли оновлення операційної системи відбувається шляхом створення нового снапшота (бінарної гілки) файлової системи куди і встановлюється безпосередньо оновлення. У разі невдалого оновлення відповідний снапшот просто видаляється, а поточний активний стан операційної системи залишається незмінним. Головна перевага такого підходу

криється саме у “незмінності” поточного стану системи, адже поки система виконує певні завдання, виконуючі процеси не будуть перервані для їх синхронізації з оновленою частиною системи, що значно підвищує стабільність та надійність загалом. Для застосування змін до системи потрібно лише здійснити перезапуск, при подальшому запуску система здійснить завантаження з нового снапшота файлової системи, що містить останнє вдале оновлення. Такий підхід робить операційну систему суттєво більш передбачуваною та дозволяє автоматизувати оновлення та зменшити часові витрати на обслуговування. Існують також і протилежні підходи які націлені на повну неперервність роботи системи та забезпечують оновлення системи та її ядра прямо під час роботи. Дійсно, для уникнення тимчасової відмови обслуговування з боку інфраструктури проекта у разі оновлення при використанні атомарних оновлень потрібно використовувати сервер дублер який тимчасово підхопить на себе роль головного сервера, або ж використовувати багатосерверну структуру з розподільником навантаження для можливості поступового оновлення інфраструктури без короткотривалого унеможливлення обслуговування. Проте засоби та методи оновлення системи під час її роботи створюють додатковий рівень комплексності, погіршують передбачуваність, ускладнюють обслуговування та несуть потенційні безпекові ризики, що робить цей підхід субоптимальним.

Загальна структура сервера виглядатиме наступним чином:

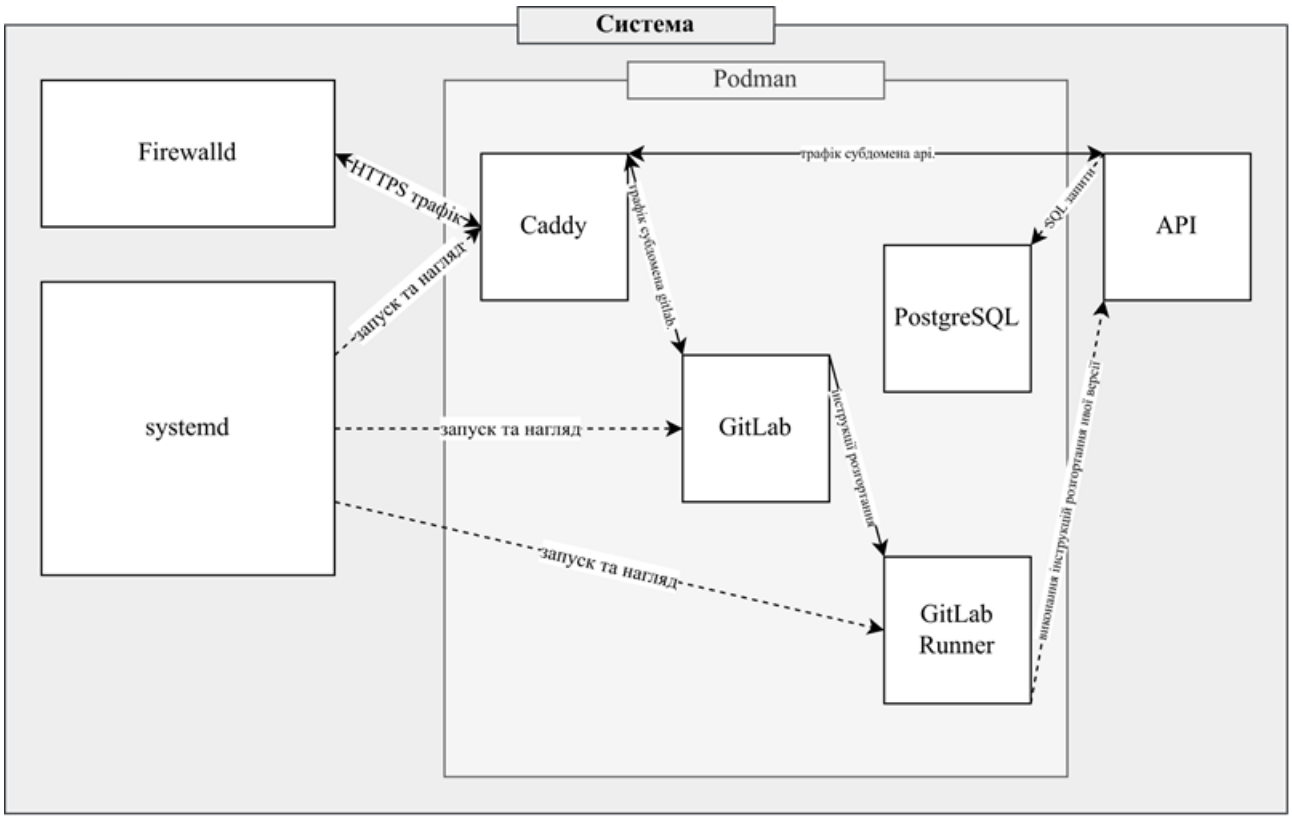

Рис. 2.1. Загальна структура архітектури сервера

Вхідною точкою для клієнтських запитів виступатиме реверсивний проксі Caddy. Caddy – веб-сервер, що дозволяє розгортати кілька мережевих сервісів що використовують HTTP(S) (стандартний протокол зв'язку для веб-сайтів) та бере на себе функцію керування SSL сертифікатами, що необхідні для використання захищеного зв'язку через шифрований протокол HTTPS. Caddy підтримує роботу з HTTP2 та HTTP3, що дозволяє забезпечити швидку передачу інформації між клієнтом і сервером. Окрім цього також варто зазначити невелике використання ресурсів та простоту налаштування при одночасному забезпеченні потужного функціонала. Існує багато інших альтернативних засобів таких як NGINX, Apache, Traefik та інші, проте Caddy через свою простоту використання та невибагливість є найкращим вибором для інфраструктури малого розміру.

Для уможливлення автоматичного розгортання нових версій проекта на сервер буде розгорнуто виконавчий елемент системи автоматичного розгортання GitLab – GitLab Runner. GitLab Runner і є тим елементом системи який виконує

заздалегідь визначені інструкції з розгортання на віддаленому сервері. GitLab Runner постачається як окрема програма та потребує окремого встановлення.

Для встановлення та виконання GitLab, Caddy та GitLab Runner буде використано систему контейнерів. Контейнери дозволяють зменшити витрати часу та ресурсів на обслуговування та розширення інфраструктури, дозволяють спростити процес резервного копіювання та відновлення, дозволяють відділити окремі елементи інфраструктури одне від одного та системи шляхом їх абстрагування за рахунок часткової віртуалізації дочірніх процесів контейнерів, що також дозволяє надавати доступ лише до конкретних необхідних ресурсів системи, що підсилює загальну захищеність інфраструктури.

Традиційно виконання задач у контейнеризованому середовищі відбувається з використання інструменту Docker, що значною мірою вплинув та визначив сучасний концепт контейнеризації, проте з часом на заміну йому був розроблений більш ресурсоефективний Podman. Podman дозволяє виконання контейнерів початково орієнтованих на Docker з використанням меншої кількості системних ресурсів. Також його вагомою особливістю є інший підхід до надання доступу до ресурсів системи контейнеру, Docker завжди виконує контейнери від імені привілейованого користувача демона Docker, що потенційно створює загрозу у разі обходу зловмисником заходів безпеки, наслідком чого може бути виконання шкідливого коду з правами привілейованого користувача. Podman з іншої сторони виконує навантаження від імені поточного системного користувача, що не лише обмежує максимально можливий рівень доступу до системних ресурсів контейнером, але й обмежує доступ одного користувача до контейнерів інших. Єдиним реальним недоліком в деяких ситуаціях є той факт, що Podman не здійснює запуск та нагляд за контейнерами після запуску операційної системи, тобто він не бере на себе обов’язки автозапуску контейнерів та їх перезапуску у разі помилки. Для обходу

цієї особливості використовується менеджер системи та сервісів програмного пакета systemd.

systemd – програмний комплекс розроблений для операційних систем сімейства Linux який керує багатьма частинами системи такими як управління службами, монтування файлових систем, таймери виконання, сокети, управління пристроями, ведення системного журналу, віртуалізація та контейнеризація. У випадку використання Podman цікавість являє можливість написання файлів, що описують контейнери, на основі яких systemd створюватиме описані контейнери із заданими правами, доступом до ресурсів тощо, та якими буде опікуватись автоматично запускаючи на старті системи та перезапускаючи у разі критичної помилки.

Для встановлення openSUSE MicroOS на Raspberry Pi 4 необхідно записати офіційний образ openSUSE MicroOS для Raspberry Pi на системний диск (наприклад microSD картка) наприклад з використанням інструмента dd (див. Рис. 2.2, де “image.img” – розархівований образ, “dev/sdX” – мітка диска)

Рис. 2.2. Приклад запису openSUSE MicroOS на системний диск

Після встановлення MicroOS для реалізації вище наведеної архітектури необхідно встановити інструмент контейнеризації Podman викликом відповідної команди “install” інструмента “transactional-update”, що дозволяє вносити зміни в операційну систему openSUSE MicroOS (див. Рис. 2.3)

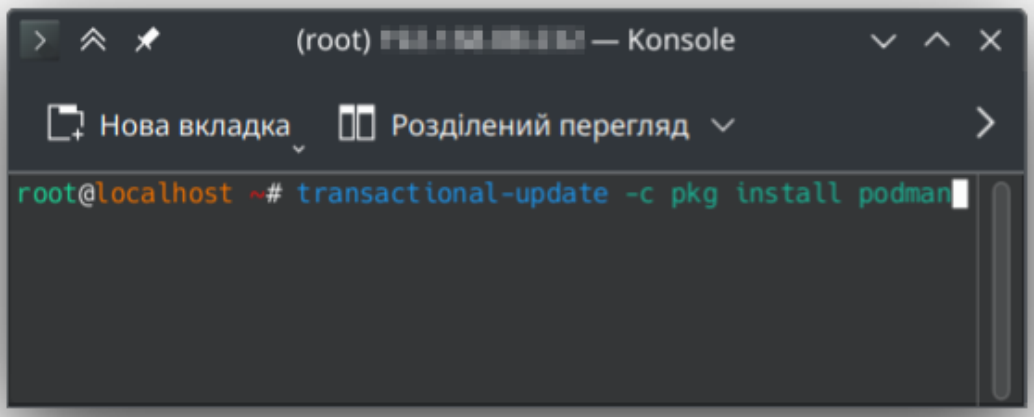

Рис. 2.3. Встановлення Podman на openSUSE MicroOS

Розгортання Caddy з використанням systemd відбувається шляхом написання вищезгаданого файлу, що описує новий контейнер, у нашому випадку для Caddy. Файл містить інформацію про майбутню назву контейнера, безпосередньо посилання на контейнер програми та його версію, визначення доступних для Caddy директорій для збереження конфігурації, сертифікатів та іншої технічної інформації, та у випадку використання Caddy правило розширення можливостей “NET_ADMIN”, що надає доступ Caddy до керування мережевою підсистемою та внесенням необхідних для функціонування та коректної маршрутизації мережевих пакетів змін. Після створення відповідного файлу systemd опікуватиметься виконанням Caddy.

Для розгортання GitLab було використано схожий до розгортання Caddy підхід, проте через відсутність офіційного контейнера GitLab сумісного з архітектурою ARM [13] процесора Raspberry Pi 4 (aarch64) було взято за основу офіційний контейнер операційної системи openSUSE Leap [14] версії 15.5 в редакції для архітектури aarch64 (ARM) що офіційно підтримується командою розробки GitLab. Варто зазначити, що попри наявність неофіційних контейнерів GitLab

для архітектури aarch64 відповідальність за безпеку та надійність подібних контейнерів як правило лежить на третіх лицях, непричетних до офіційного процесу розробки, що робить використання подібних контейнерів потенційно небезпечним та ризикованим, особливо у світлі подій атаки на відкритий проект “XZ Utils” з метою скомпроментації 29 березня 2024 року [15]. Для встановлення GitLab в новостворений контейнер необхідно всередині контейнера виконати додавання офіційного GitLab-CE (GitLab Community Edition) репозиторію для openSUSE Leap 15.5 та виконати встановлення GitLab-CE. Також у зв’язку з тим, що Raspberry Pi 4 є доволі потужною платформою за мірками одноплатних ARM комп’ютерів, проте не володіє вражаючими показниками, варто здійснити базове налаштування GitLab згідно загальних рекомендацій для Raspberry Pi та викликати реконфігурацію GitLab згідно відредагованого конфігураційного файлу “/etc/gitlab/gitlab.rb” (див. рис. 2.4.)

Рис. 2.4. Встановлення та виклик реконфігурації GitLab-CE

Фінальним етапом є розгортання GitLab Runner, що і буде виконувати інструкції з розгортання проекта. Аналогічно до процесу розгортання попередніх контейнерів необхідно створити відповідний описовий файл контейнера для systemd. Після створення та запуску контейнера необхідно виконати реєстрацію новоствореного GitLab Runner для можливості його подальшого використання в проектах. Для реєстрації необхідно перейти у секцію “Runners” у вкладці “CI/CD” в налаштуваннях репозиторію проекту та натиснути кнопку “New project runner” для реєстрації. При реєстрації нового GitLab Runner необхідно

вказати базові налаштування (див. рис. 2.5). Після вказання бажаних налаштувань потрібно натиснути кнопку “Create runner”.

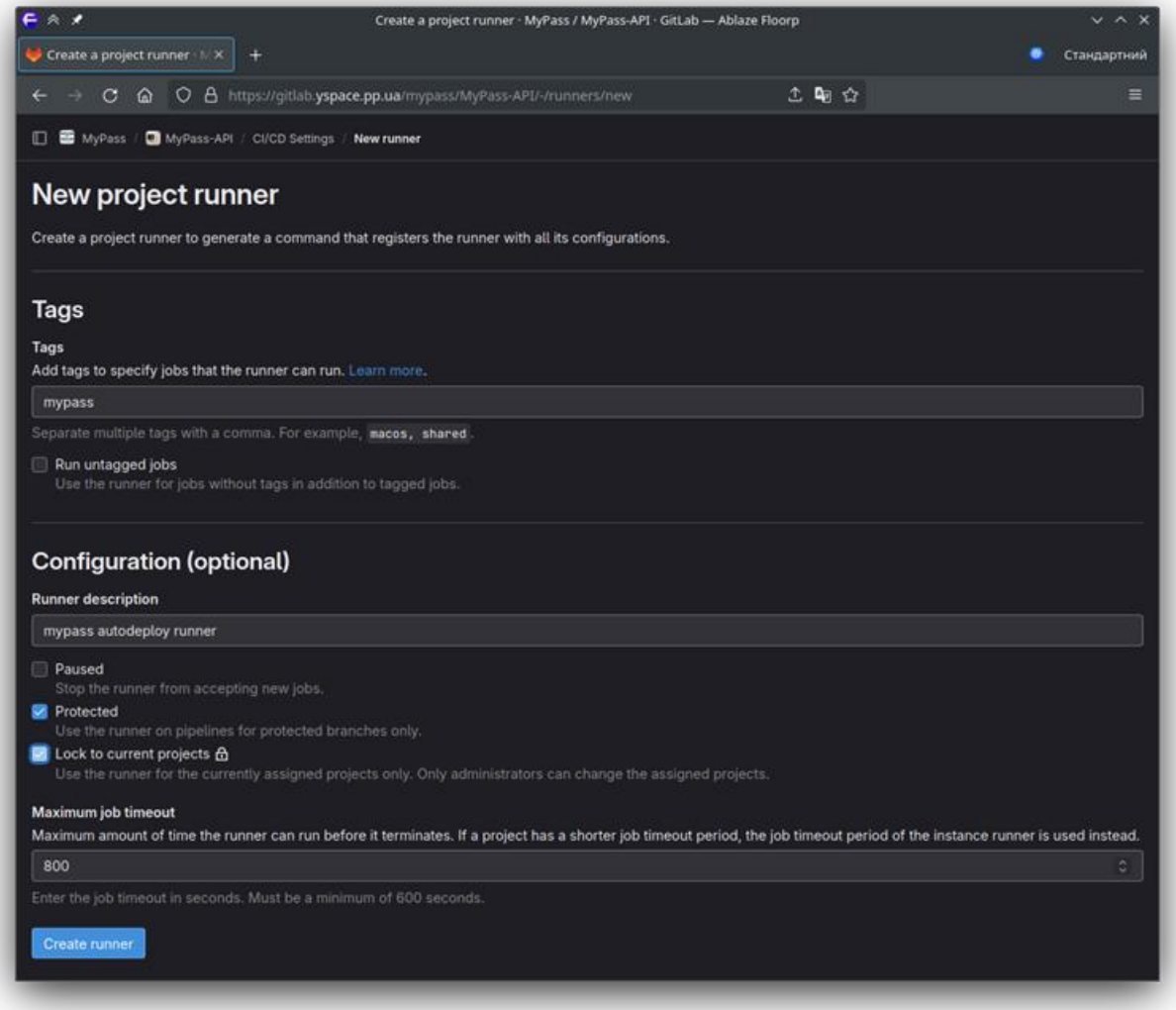

Рис. 2.5. Базова конфігурація нового GitLab Runner при його реєстрації

Останнім кроком є безпосередня реєстрація нового GitLab Runner. Для цього потрібно виконати команду реєстрації всередині нового контейнера з GitLab Runner зі вказанням токена аутентифікації, що буде вказаний у наступному вікні реєстрації GitLab Runner після натискання “Create runner” (див. рис. 2.6 та рис. 2.7).

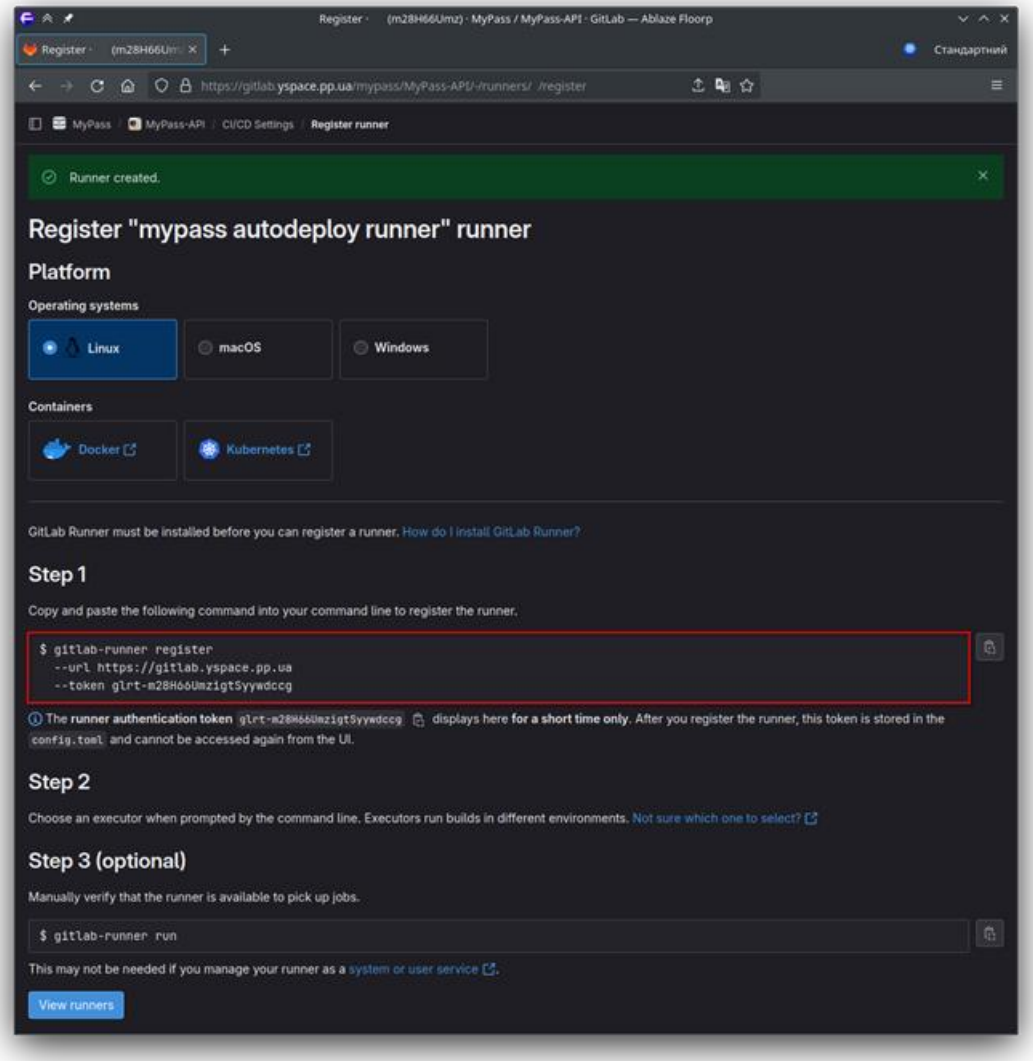

Рис. 2.6. Інструкції надані GitLab для реєстрації GitLab Runner

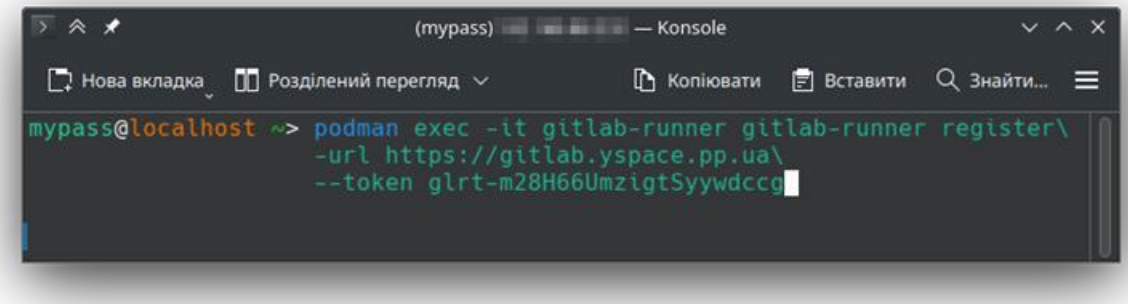

Рис. 2.7. Виконання фінальних інструкцій реєстрації нового GitLab Runner

2.3. Підбір інструментів платформи та засобів для розробки

Для реалізації API проекту було використано гнучкий та надійний стек технологій на основі NodeJS. Головною мовою розробки виступав TypeScript, що фактично є надбудовою з системою типізації над стандартним JavaScript для кращого контролю формату та сумісності вхідних та вихідних типів даних. Головним рушійним комплексом та системою розробки REST API було використано NestJS, за його здатність покривати всі сценарії, потреби та типи API, а також надійність, сучасність та дружні до розробника підходи та структура, що надає певний рівень свободи при написанні кодової бази. Для середовища запуску та виконання коду проекта було обрано Bun за його миттєву систему керування пакетами та залежностями проекта, багатократне пришвидшення виконання коду у порівнянні з NodeJS, раціональне використання системної пам'яті та повну сумісність з класичними інструментами виконання та розгортання проекта. Збереження, структурування та підтримання цілісності інформації покладено на базу даних PostgreSQL, що задовольняє всі сучасні вимоги, надає широкий вибір типів даних та має малі затримки при отриманні та записі інформації.

JavaScript (ECMAScript), спочатку створений як мова програмування для додавання інтерактивних елементів до веб-сторінок. Його розробив Брендан Айк у 1995 році, працюючи в компанії Netscape. Спочатку JavaScript використовувався виключно на стороні клієнта, тобто в браузерах, де він додавав динаміку до статичних HTML-сторінок.

З часом, завдяки своїй простоті та функціональності, JavaScript отримав широке визнання серед розробників. Це призвело до того, що у 2009 році була створена платформа Node.js, яка дозволила використовувати JavaScript на стороні сервера, тобто для бекенду. Node.js була розроблена Раяном Далем і

відразу стала популярною через свою здатність обробляти великий обсяг одночасних запитів з високою швидкістю.

Використання JavaScript для бекенду дозволяє розробникам використовувати одну мову програмування для всього циклу розробки веб-застосунків, як на фронтенді (вебінтерфейс користувача), так і на бекенді (серверне програмне забезпечення). Це значно спрощує процес розробки, підтримки коду та процес розвитку всієї команди.

JavaScript є високорівневою та інтерпретованою мовою програмування яка підтримує кілька парадигм програмування, включаючи об'єктно-орієнтоване, функціональне та імперативне програмування. Це означає, що розробники можуть вибирати найбільш зручний для них стиль написання коду, або комбінувати їх, залежно від вимог проекту. JavaScript використовує динамічну типізацію, що дозволяє змінювати типи змінних під час виконання програми, надаючи більше гнучкості в написанні коду.

Node.js базується на асинхронній архітектурі, що дозволяє ефективно обробляти кілька запитів одночасно (паралельно), це дозволяє покращити здатність програми задіювати всі наявні ресурси системи для швидшого виконання отриманих задач та надання відповідей. Тому Node.js є оптимальним рішенням для створення високонавантажених веб-сайтів і застосунків таких як онлайн конференц-зали, стрімінгові сервіси, соціальні платформи, великі онлайн магазини тощо.

Завдяки Node.js та численним бібліотекам і фреймворкам, таким як Express.js, NestJS, TypeORM.., JavaScript перетворився на потужний інструмент для створення повноцінних веб-застосунків, що дало змогу багатьом компаніям значно пришвидшити розробку та зменшити витрати використовуючи одну мову програмування на всіх етапах роботи з кодовою базою усього проекта.

JavaScript став дуже важливим елементом сучасної веб-розробки, використовуючись як на стороні клієнта, так і на стороні сервера, завдяки своїй гнучкості, асинхронності та простоті вивчення.

TypeScript – це мова програмування створена для розширення можливостей JavaScript. Вона була розроблена компанією Microsoft і вперше представлена в 2012 році. Основною причиною її створення було бажання вирішити проблеми масштабованості та підтримки великих проектів на JavaScript. JavaScript, яка спочатку розроблялась як мова для додавання інтерактивності на вебсторінки, не була пристосована для великих і складних програм. TypeScript додає можливість типізації, що допомагає програмістам уникати багатьох помилок на етапі написання коду, а не під час виконання програми.

У контексті бекенду, TypeScript стає все популярнішим завдяки своїй здатності покращити надійність і читабельність коду. Він дозволяє розробникам чіткіше описувати структуру даних і логіку застосунків що працюють на сервері, шляхом впровадження системи типізації даних, що запобігає випадковій передачі некоректно типізованої інформації з одного блоку коду в інший. Оскільки при збиранні проекта TypeScript компілюється в JavaScript, він повністю сумісний з усіма існуючими інструментами та бібліотеками останнього, що робить його дуже зручним для використання.

Однією з особливих можливостей TypeScript є декоратори, що дозволяють розробникам додавати спеціальні метадані до класів та методів. Наприклад, на бекенд проекті декоратори можуть використовуватися для налаштування маршрутизації HTTP-запитів. Декоратор може прикріпити до методу певну URL-адресу та HTTP-метод (GET, POST тощо), що дозволяє легше організовувати та підтримувати код.

Декоратор можна уявити як якусь позначку або ярлик, який ми прикріплюємо до частини коду, щоб сигналізувати, що поведінку, функціонал або випадки виконання цього коду потрібно змінити згідно логіки визначеної в декораторі. Наприклад, якщо у нас код який обробляє запити на отримання даних користувачів, ми можемо використати декоратор щоб сказати: "Цей метод обробляє GET-запити за адресою /users". Це значно спрощує роботу розробника, оскільки вся логіка обробки запитів зосереджена у самому коді, а не розкидана по всьому проекту. Також декоратори уможливлюють надання тих чи інших властивостей окремому блоку коду без його безпосередньої модифікації.

Таким чином, TypeScript надає розробникам потужні інструменти для створення більш зрозумілого, надійного, передбачуваного та підтримуваного коду, особливо в контексті серверного програмного забезпечення. Завдяки своїй інтеграції з JavaScript, він дозволяє користуватись усіма перевагами існуючої екосистеми JavaScript, роблячи перехід на TypeScript відносно безболісним.

NestJS – це прогресивний фреймворк для створення серверних застосунків на основі TypeScript або JavaScript. Він створений для того, щоб розробникам було легше будувати масштабовані та ефективні серверні програми. NestJS використовує архітектурні принципи, натхненні іншими фреймворками, такими як Angular, що робить його більш простим і зрозумілим, для багатьох розробників, що вже працювали з цими інструментами.

Розробка NestJS розпочалася у 2017 році, коли польський розробник Каміль Мичек зрозумів, що існуючі на той час фреймворки для Node.js не задовольняють його потреби в побудові масштабованих серверних застосунків. Мичек вирішив створити фреймворк, який би поєднував кращі практики з об’єктно-орієнтованого програмування (ООП), функціонального програмування (ФП) та метапрограмування. В результаті з'явився NestJS, який швидко завоював

популярність завдяки своїй простоті, модульності та підтримці проекта зі сторони самого Каміля.

База архітектури NestJS базується на модулях. Модулі – це основні будівельні блоки програми, які допомагають організувати код у логічні одиниці, що опікуються певною частиною частиною функціоналу програми. Кожен модуль може містити контролери, провайдери, сервіси та інші модулі. Це дозволяє легко розширювати функціональність програми та забезпечує високу гнучкість, що сприяє кращій організації коду.

Контролери в NestJS відповідають за обробку запитів від клієнтів і повернення відповідей. Вони використовують роути, шляхи для визначення того, які дії виконуються на різних URL. Контролери тісно співпрацюють з сервісами, які містять так звану “бізнес-логіку”.

Сервіси є основною частиною програми, вони виконують основні операції такі як доступ до бази даних, обробка інформації, виклик логіки з інших сервісів, або ж виконання будь-яких інших дій необхідних для виконання вказаної контролером задачі.

Провайдери в NestJS – це класи, частини проекта що, як правило, містять набір певної логіки. Вони визначаються і існують в межах модулів та можуть бути введені та використані в іншій частині проекта за допомогою механізму залежностей. Цей підхід дозволяє швидко та ефективно розширювати, доповнювати та створювати новий функціонал.

Для використання функціоналу та можливостей фреймворка NestJS широко використовується функціонал декораторів з TypeScript. Декоратори дозволяють визначати тип блоку кода, його роль, властивості, умову виклика та багато іншого. Таким чином уникається потреба використовувати більш комплексні та менш придатні для підтримки методи організації коду та виклику відповідних до події механізмів, сервісів, що в рази покращує досвід розробки на NestJS.

Завдяки модульній архітектурі, простоті використання та потужним інструментам для розробки, NestJS став популярним вибором серед розробників які шукають ефективний спосіб побудови масштабованих серверних застосунків.

Bun – сучасний швидкісний рушій для JavaScript та Typescript який було створено з метою підвищення продуктивності та ефективності в порівнянні з Node.js. Розробка Bun розпочалась через потребу замінити повільну та неефективну за сучасними мірками Node.js, при цьому зберігаючи повну сумісність з проектами орієнтованими на Node.js.

Bun розроблявся з акцентом на використання сучасних технологій та підходів для досягнення максимальної продуктивності. В основі Bun лежить Zig – мова програмування, яка надає змогу досягти високої продуктивності, завдяки меншому рівню абстракції та більш безпосередньому управлінню пам’яттю та системними ресурсами.

Основною причиною, чому Bun швидший за Node.js, є його численні архітектурні рішення. Bun використовує власні високопродуктивні модулі роботи з HTTP запитами та процесами вводу-виводу, що мінімізує затримки при обробці запитів. Ще одним вдосконаленням є покращене управління пам'яттю, що також зменшує ресурсозатрати та збільшує продуктивність.

TypeORM – система управління базами даних для TypeScript та JavaScript, створена для роботи з реляційними базами даних в застосунках на базі Node.js. TypeORM підтримує багато популярних баз даних, таких як PostgreSQL, MySQL, SQLite, MariaDB, Oracle, Microsoft SQL Server і інші.

Однією з основних характеристик TypeORM є використання декораторів, що дозволяють визначати моделі, що описують структуру бази даних, безпосередньо у класах, що робить код чистішим і зрозумілішим, полегшуючи його підтримку. Також вона надає інструмент для роботи з міграціями, що спрощує керування змінами в структурі бази даних і, залежно від потреб та підходу, автоматизувати створення міграцій.

TypeORM чудово інтегрується з іншими бібліотеками і фреймворками такими як Express або NestJS, здатна забезпечити хорошу продуктивність і масштабованість, що дозволяє використовувати її як для маломасштабних проектів, так і для великих корпоративних систем та застосунків.

PostgreSQL – об'єктно-реляційна база даних, відома своєю високою надійністю та масштабованістю. Її розробка почалася у 1986 році в Каліфорнійському університеті в Берклі під керівництвом професора Майкла Стоунбрейкера. З часом, проект отримав значну популярність і підтримку від спільноти розробників, що сприяло його постійному вдосконаленню та розвитку.

Однією з ключових особливостей PostgreSQL є її архітектура, що підтримує як реляційні, так і об'єктно-орієнтовані моделі даних. Це дозволяє зберігати комплексні дані, такі як масиви, JSON, XML, вектори та інші структури, що робить її гнучкою у застосуванні для широкого списку сценаріїв.

Завдяки своїй архітектурі, PostgreSQL забезпечує високу продуктивність та ефективність. Наявна підтримка багатоверсійності (MVCC) дозволяє уникати блокувань при читанні даних і забезпечує високий ступінь паралельності. Це

особливо корисно і навіть необхідно для великих систем, де одночасно працює багато користувачів.

PostgreSQL також вирізняється своєю потужною системою обробки транзакцій, яка гарантує повну відповідність ACID-властивостям (атомарність, консистентність, ізоляція, витривалість). Це забезпечує високу надійність збереження даних і відновлення після збоїв. Інші важливі функції включають підтримку процедурних мов, таких як PL/pgSQL/SQL, що дозволяє створювати складні збережені процедури і тригери.

Розробники цінують PostgreSQL за її стабільність та здатність до масштабування. Крім того, наявність великої та активної спільноти користувачів забезпечує постійний розвиток та підтримку проекту. Таким чином, PostgreSQL продовжує залишатися одним з найбільш надійних і гнучких рішень, коли це стосується реляційних SQL сумісних баз даних.

2.4. Розробка REST API (прикладного програмного інтерфейсу) для роботи з системою

База даних репрезентує собою певну модель, що складається з таблиць з інформацією та зв'язків між ними. Кожна таблиця являє базову інформаційну інтерпретацію певного об'єкта з його важливими для конкретного сценарію застосування властивостями. Властивості таблиці можуть мати різні типи, наприклад числовий, текстовий, булевий (хиба або ж істина) чи більш конкретні такі як час, вектор, координати, масиви тощо.

Для задоволення функціональних потреб проекту база даних має містити наступні види записів: користувач, ключ-картка, група, дисципліна, захід, подія (наприклад конкретно взята пара), локація, термінал, запис відвідуваності з відповідними до логічної ієрархії стану речей в реальному світі зв'язками.

Існує три головних види зв'язків у класичній реляційній моделі баз даних. Вид зв'язку "one-to-one" використовується для реалізації відношення одного запису A_x до одного запису B_y. При такому типі зв'язку запис A_x, одночасно не може бути пов'язаним з B_y та B_{y+n}, а лише з одним з них і навпаки. "many-to-one" дозволяє відносити кілька записів $A_{\{1, 2, \dots x\}}$ до одного запису B_y, але не навпаки. Зв'язок "many-to-many" дозволяє пов'язувати довільну кількість записів $A_{\{1, 2, \dots x\}}$ з довільною кількістю записів $B_{\{1, 2, \dots y\}}$ в довільному порядку кожен з яких пов'язаний з довільними записами $B_{\{1, 2\dots y\}}$ і одночасно навпаки довільні записи $B_{\{1, 2, \dots y\}}$ можуть бути пов'язаними з довільними записами $A_{\{1, 2, \dots x\}}$ в довільному порядку кожен з яких пов'язаний із довільними $B_{\{1, 2, \dots y\}}$.

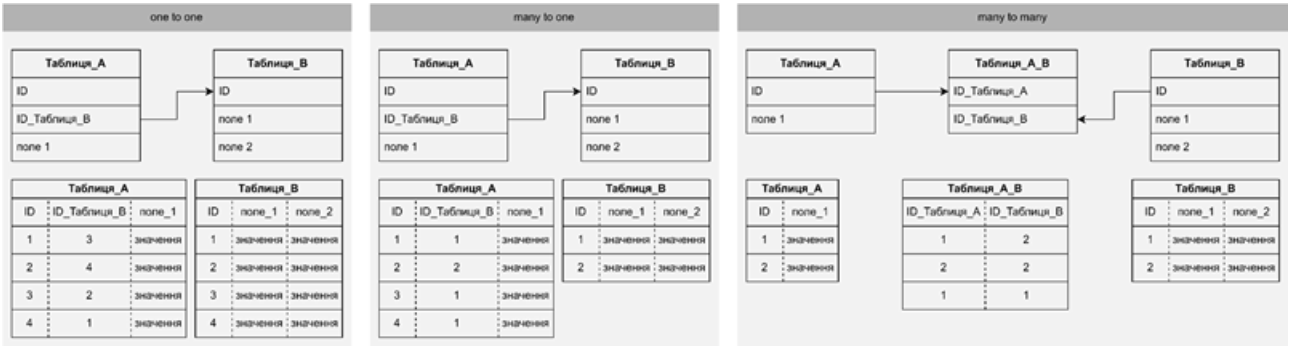

Рис. 2.8. Види зв’язків у реляційних базах даних

У нижченаведеній таблиці можна більш детально побачити необхідні для поточного сценарію застосування види записів, їх необхідні для репрезентації властивості та інші пов’язані з ними записи:

Таблиця 2.1

Необхідні типи записів майбутньої бази даних

Запис	**Властивості**	**Пов’язані записи**
користувач	email, роль, ім’я, прізвище, патронім	ключ, група, дисципліна, подія, відвідуваність
ключ (відповідає коду ключ-картки)	ключ, тип	користувач
група	ім’я	користувач, дисципліна
дисципліна	ім’я, курс, години, тип (екзамен, залік…)	група, подія, користувач
захід	тип	дисципліна, користувач
подія	тип, дата, місце	захід, відвідуваність, термінал

Продовження таблиці 2.1

відвідуваність	*лише технічні*	користувач, термінал
термінал	розташування, авторизовані ролі, прапорець дозволу допуску всіх ролей після розблокування, публічний ключ	відвідуваність, подія
локація	опис	подія, термінал

Планування структури бази даних для API є одним з найважливіших етапів розробки, адже відбувається він безпосередньо на початку і значною мірою визначає технічні можливості проекта та його придатність до подальшого масштабування та розвитку з нарощуванням функціоналу. Некоректне проектування бази даних може призвести до аномалії наявності невідповідної дійсності інформації та навіть такої, що перечить сама ж собі, або ж інших аномалій роботи бази даних при певних операціях, наприклад видаленні, що може призвести до невідворотної втрати даних. Для забезпечення цілісності інформації та уникнення подібних аномалій існують різні форми нормалізації, яким має відповідати структура бази даних. Кожна наступна з основних форма нормалізації передбачає дотримання попередньої та впроваджує нові правила й вимоги для забезпечення цілісності інформації. Перші три форми нормалізації:

- 1 форма нормалізації:
    - властивість не може містити більше ніж 1 значення заданого типу
    - кожен запис повинен мати свій унікальний ключ для ідентифікації
    - недопущення дублікатів записів
    - кожен стовпець містить власний вид інформації та не дублює інші
- 2 форма нормалізації:
    - дотримана 1 форма нормалізації
    - всі поля в записах повністю залежать від унікального ключа
- 3 форма нормалізації:
    - дотримана 2 форма нормалізації
    - відсутність часткової проміжної залежності від унікального ключа (наприклад залежність лише від частини унікального ключа, або ж опосередкована залежність від нього)

На практиці дотримання перших трьох форм нормалізації дозволяє запобігти потенційним аномаліям та забезпечити грамотний дизайн бази даних у переважній більшості випадків та проектів.

На основі вищенаведених принципів та правил нормалізації баз даних та загальних вимог проекта було розроблено базу даних даних наступної структури (див. рис. 2.9).

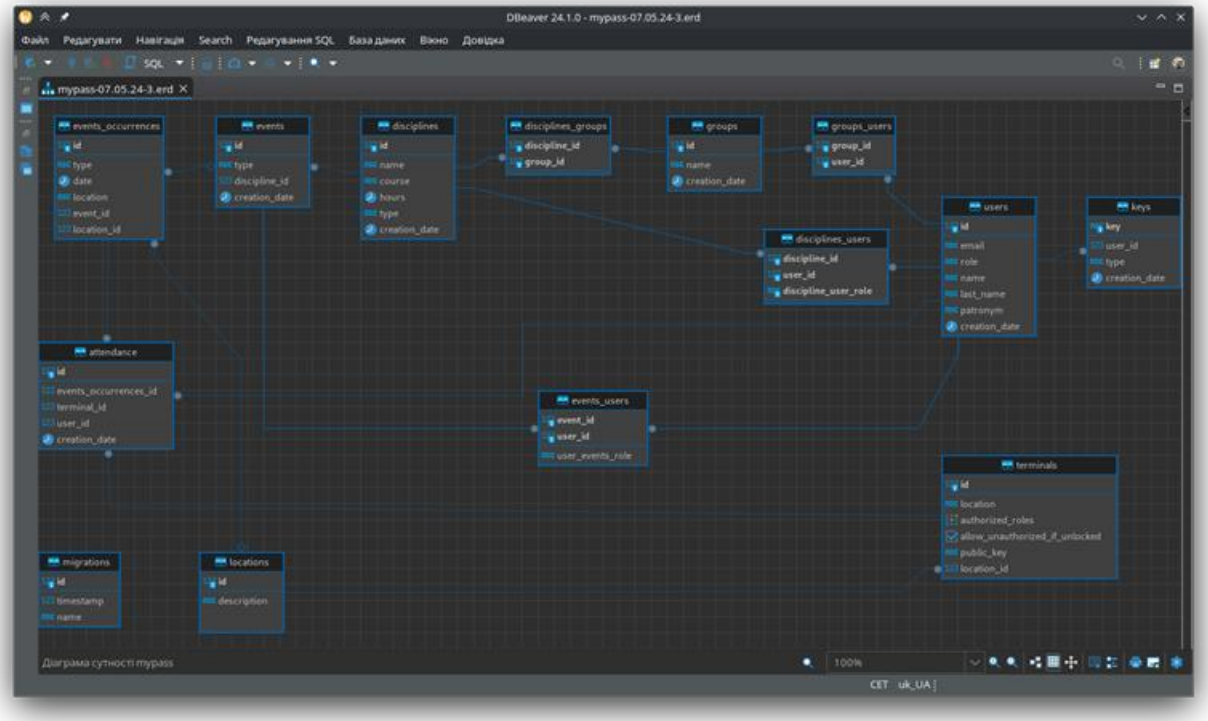

Рис. 2.9. Готова структура бази даних проекта

Наведена на рисунку 2.9 структура відповідає вимогам третьої нормальної форми та репрезентує логічний зв’язок всіх потрібних для системи елементів. При створенні увага приділялась також і можливості подальшого розширення структури та можливостей для покриття різних сценаріїв подальшого розвитку проекта та задоволення потреб нового функціоналу.

Розробка та закладення основи кодової бази API є

Першим етапом після проектування структури API є ініціалізація нового проекту, яку ми можемо здійснити використовуючи Nest CLI та його команду “new”. По її закінченню ми отримуємо пустий проект з директорією “src” яка містить головний модуль програми (кореневий).

Після планування бази даних та створення основи кодової бази перед безпосереднім початком розробки конкретного функціонала потрібно забезпечити можливість взаємодії програми із базою даних. *(TypeORM)*

Для цього ми створюємо модуль TypeORM використовуючи команду “forRoots” об’єкта “TypeOrmModule” та імпортуємо в кореневий модуль. При створенні вказуємо тип бази даних (“postgres”), порт, хост, дані користувача, а також шлях до теки з міграціями після виконання збирання проекту. Додатково вказуємо стратегію найменування як “SnakeNamingStategy”.

Наступним етапом є створення CRUD модулів для кожного самостійного типу запису в таблиці бази даних. CRUD – це акронім кожне слово якого відповідає за чотири основні функції які можна виконувати над записами в базі даних, а саме: створити, зчитати, оновити та видалити. Для генерації модулів використовується команда “nest g resaurse” в кінці якої ми прописуємо назву модулі. Після її виконання створюється тека яка має в собі сервіс, контролер, теку з пов’язаними DTO та файл з розширенням “modules” який є головним і відповідає за експортування та налаштування даного модуля. Для обробки запитів до бази в контролері було створено п’ять основних функцій з відповідними декораторами:

- findAll – повертає всі елементи таблиці при виклику запиту ”GET”
- findOne – повертає конкретний запис з таблиці за вказаним id. Значення параметра id передається через декоратора “Param”.
- crete – створює новий запис в таблиці за вказаними даними при виклику запиту ”POST”. Для отримання даних використовується декоратор “Body”
- update – оновлює запит за вказаними id відповідно до переданих даних при виклику методу “PATCH”. Для отримання даних використовується декоратор “Body”
- remove – видаляє запит з бази даних при виклику запиту “DELETE”.

Кожна функція викликає свій відповідник в сервісі, які виконують запити до бази даних на основі переданих даних. Логіка кожної функції, для роботи з базою даних, не змінюється між модулями тому буде описана узагальнено для всіх. В кожен сервіс було імпортовано об’єкт “DataSource”, який має метод

“query”, за допомогою якого ми будемо надсилати запити в базу даних. Додатково в кожному модулі який має зв’язки з іншими модулям ми створюємо відповіді функції для отримання пов’язаних даних. Дані функції будуть викликатись при отриманні або створенні запитів в базі даних.

Переходимо до реалізації методів для звернення до бази даних. Першим буде метод для видалення записів. Даний метод надсилає запит “DELETE” з параметром “WHERE” який дозволить видалити тільки один запис за переданим id.

Наступними методами є “findOne” та “findAll” які мають спільні “SELECT” запити, але перший метод замість того щоб дістати всі записи, фільтрує їх на основі переданого в нього id і повертає отриманий запит. В кінці методів, викликаються функції для отримання інформації з суміжних модулів при потребі користувача.

Метод “create” робить запит “INSERT INTO” для додавання нового запису в таблицю. Якщо дана таблиця має певні залежності і інформація про них була передана до методу, то ми викликаємо додаткові методи для налаштування даних залежностей. Дані методи будуть видаляти попередні спільні записи та створювати нові на основі переданих даних.

Метод “update” має схожу логіку до попереднього методу тільки замість “INSERT INTO” робиться запис “UPDATE”. Методи оновлення залежностей працюють ідентично до методу “create”.

2.5. Підбір апаратного забезпечення для реалізації апаратно-програмної системи термінала

Згідно вимог та потреб для реалізації апаратно-програмного комплексу термінала пропуску майбутній пристрій повинен мати наступний базовий функціонал:

- зчитування NFC та RFID карток
- зчитування 1D, 2D кодів (Barcode, QR code)
- під'єднання до мобільної стільникової мережі для доступу до інтернету
- робота з Wi-Fi для доступу до інтернету
- виведення інформації на дисплей для користувача

Обране для реалізації можливостей апаратне забезпечення має відповідати наступним трьом критеріям: надійність, відносна недороговартісність, доступність. Під доступністю мається на увазі можливість повторного придбання цієї ж моделі виробу у необхідній кількості.

ESP32-WROOM-32E – модуль побудований на базі мікроконтролера ESP32 розроблений Espressif Systems, що призначений для застосування у продуктах інтернету речей (IoT). Мікроконтролер оснащений модулем Wi-Fi для роботи з домашніми та офісними безпровідними мережами та підтримує всі базові інтерфейси для взаємодії з додатковими модулями: чотири SPI, два I^2C, два I^2S, три UART, десять портів для ємнісних сенсорних інтерфейсів. Також на борту наявний 12-бітний аналогово-цифровий перетворювач для зчитування аналогових сигналів та 8-бітний цифро-аналоговий перетворювач для генерації примітивних аналогових сигналів малої точності для управління аналоговими пристроями. Для виконання обчислень ESP32-WROOM-32E використовує двоядерний 32-бітний процесор Xtensa LX6, що працює в діапазоні частот від 160 до 240 МГц та оснащений 520 Кб оперативної пам'яті, модуль має 16 мегабайт постійної програмної пам'яті.

Мікроконтролери сімейства ESP32 дозволяють здійснювати програмування з використанням стандартних методів та підходів, а також підтримують переважну більшість різноманітних написаних для платформи Arduino бібліотек для роботи з платами розширення, що не залишить розробника без великого масиву документацій та практичних прикладів, що допоможуть у розробці.

Впродовж років даний модуль зарекомендував себе як надійна, потужна, дешева, багатофункціональна платформа для розробки засобів платформи інтернету речей. Мікроконтролери серії ESP від Espressif Systems використовуються великою кількістю компаній та ентузіастів з усього світу для розробки домашнього, офісного та індустріального устаткування.

Гарні потужнісні характеристики, великий об’єм постійної пам’яті в 16 мегабайт, невисока ціна [16], детальна документація [17] та висока надійність роблять даний мікроконтролер чудовим вибором для розробки комплексного апаратного рішення платформи інтернету речей з базовим користувацьким інтерфейсом.

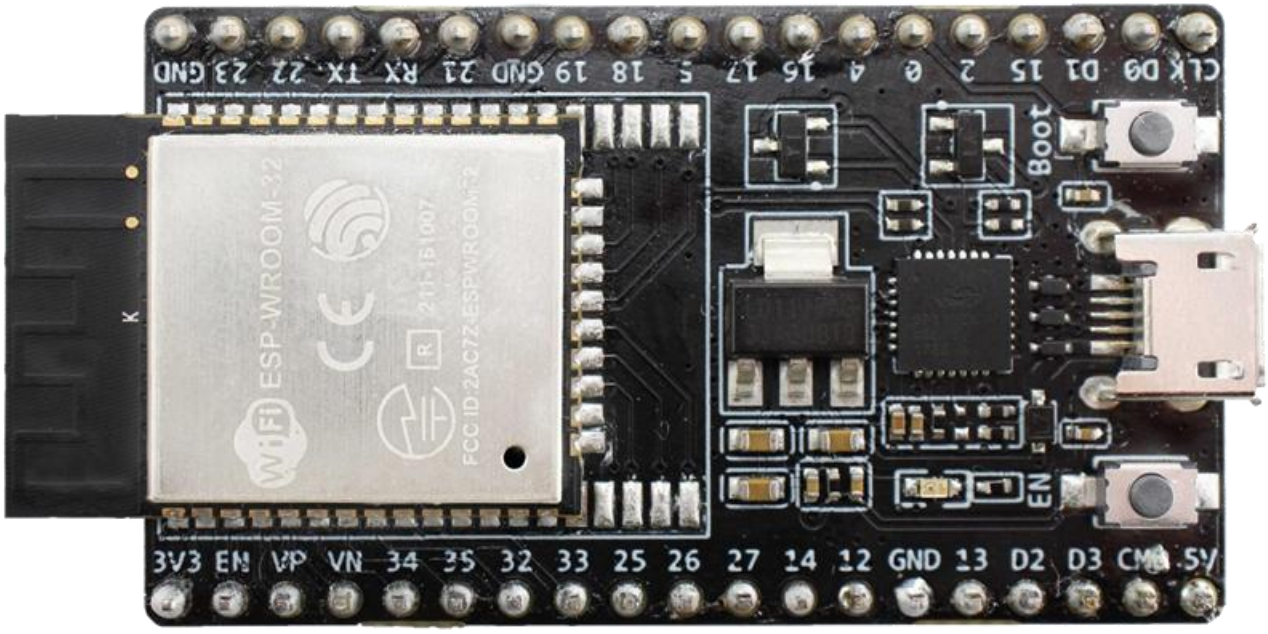

Рис. 2.10. Плата для розробки на базі ESP32-WROOM-32E

PN532 – простий у використанні NFC/RFID модуль розроблений для роботи з платформою Arduino та іншими сумісними з бібліотеками Arduino мікроконтролерами (наприклад ESP32). PN532 дозволяє працювати з NFC та RFID картками в різних режимах їх роботи. Цей модуль підтримує три методи зв'язку з головною платою мікроконтролера: UART, SPI та I^2C. Для застосування пристрою в проектах використовуються офіційні бібліотеки наявні в середовищі розробки Arduino IDE. Модуль використовується для виконання різноманітних функцій, таких як передача довільних даних через NFC/RFID, робота як NFC/RFID картка та NFC/RFID зчитувач/термінал. Пристрій дозволяє працювати з іншими картками та пристроями на відстані до 5 см. Завдяки низькому споживанню енергії та високій швидкості, цей модуль став популярним серед багатьох компаній для інтеграції у власні розробки, що доводить його відповідність заявленим виробником характеристикам та високу надійність.

Висока надійність, гнучкість у методах інтеграції, підтримка роботи у різних режимах протоколів NFC/RFID та невелика вартість [18] роблять даний виріб чудовим для застосування у пристрої термінала для зчитування NFC та RFID карток.

Рис. 2.11. Модуль PN532 для роботи з NFC та RFID

Waveshare 1D/2D Codes Reader – модуль розроблений Waveshare для сканування одно та двовимірних кодів, таких як Barcode та QR code. Модуль налічує два оптичних сканера: лазер та камера. Лазер модуля забезпечує швидке та стабільне сканування одновимірних кодів таких як: код 11, код 39 / код 93, UPC / EAN, код 128 / EAN128, панель даних GS1 (RSS) та інші. Невеличка камера вмонтована в блок сканера дозволяє декодувати такі двовимірні коди як: QR-код, матриця даних та PDF417. Кут сканування одновимірних кодів складає 60° по горизонталі з допустимим перекосом коду відносно сканера до 65°. Кут огляду сканера двовимірних кодів складає 28° по горизонталі та 21,5° по вертикалі. Для підключення модуля до мікроконтролера використовується серійний інтерфейс UART.

Через свою надійність спрацьовування та швидкість роботи Waveshare 1D/2D Codes Reader є оптимальним вибором для розробки надійного та передбачуваного пристрою, попри свою помірну, проте не велику вартість [19].

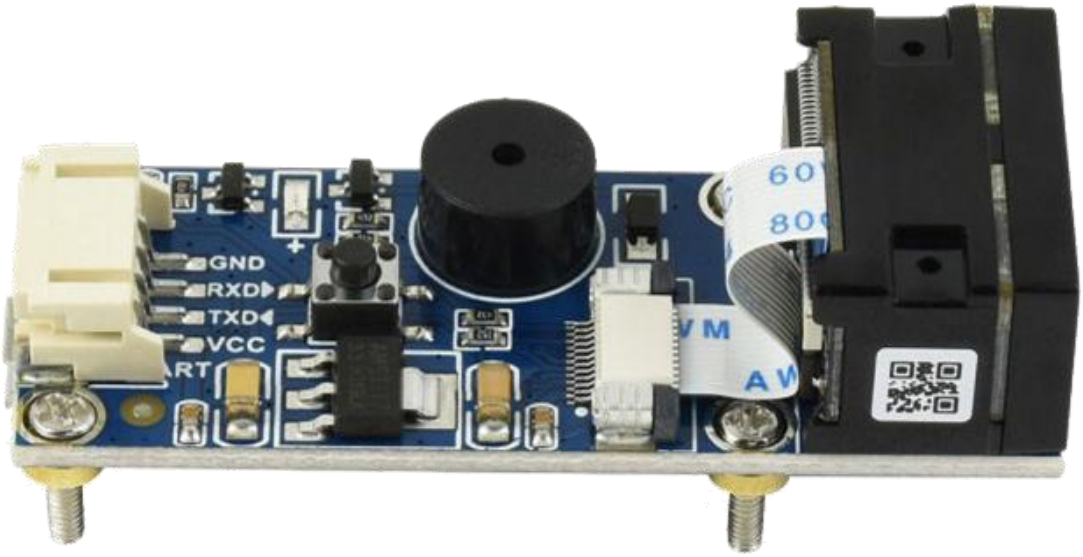

Рис. 2.12. Сканер штрих-кодів, QR-кодів 1D/2D від Waveshare

SIM800L – GSM модуль від Simcom, який дозволяє реалізувати пітримку мобільних стільникових мереж (GSM), тобто він може підключатися до мобільної мережі для прийому дзвінків та відправлення й отримання текстових повідомлень, а також встановлювати з'єднання з інтернетом за допомогою протоколів GPRS, TCP та IP. Даний модуль підтримує кілька різних частот стільникових мереж, що дозволяє одній версії модуля працювати з усіма мобільними мережами світу. Для взаємодії з платою мікроконтролера використовується серійний UART зв'язок, надсилання команд та отримання інформації від модуля відбувається у "текстовому" форматі.

Даний модуль доволі часто можна побачити інтегрованим у різноманітні проекти завдяки його простоті використання, надійності та дешевизні [20], що робить його чудовим вибором для використання у проектах де необхідно надсилати або ж отримувати інформацію з віддаленого сервера через інтернет.

Рис. 2.13. Модуль стільникового зв’язку SIM800L

Waveshare 3.5inch RPi LCD (A) – це кольоровий рідкокристалічний дисплей з діагоналлю 3,5 дюйма, роздільністю 480 на 320 пікселів та сенсорною панеллю початково розроблений компанією Waveshare. Модуль був створений для використання з Raspberry Pi 3 та 4, проте дисплей можливо використовувати і з іншими пристроями, наприклад мікроконтролерами, завдяки бібліотекам комунікації з TFT дисплеями через протокол SPI.

Дисплей має достатню роздільну здатність, задовільну для пристрою термінала діагональ, а головне, відомі та доступні для мікроконтролера ESP32 методи підключення та бібліотеки для виведення зображення. Серед нечисленної кількості подібних дисплеїв на ринку даний модуль є найбільш оптимальним

завдяки своїй якості виготовлення, сукупності характеристик та поміркованій вартості [21].

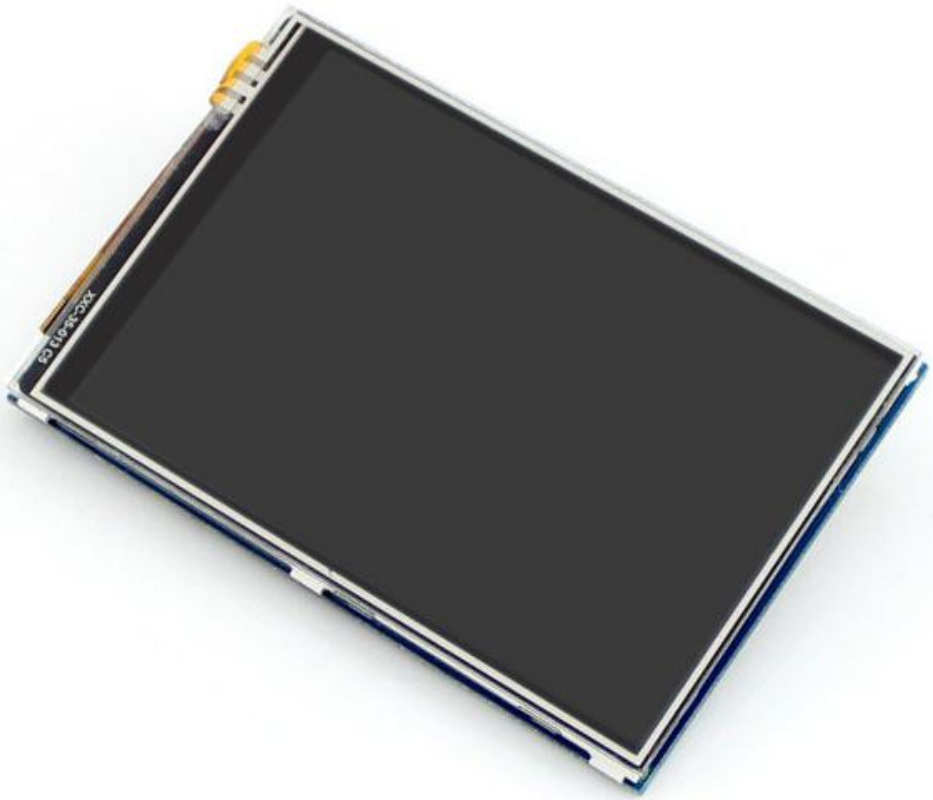

Рис. 2.14. Дисплей Wavashare 3.5inch RPi LCD (A), 480x320

Протокол UART – універсальний асинхронний протокол комунікації, що дозволяє реалізувати комунікацію між головним та залежним пристроями. На відміну від більшості інших протоколів зв'язку UART не є синхронним, що не передбачає синхронізації частоти внутрішнього годинника залежного пристрою з частотою годинника головного пристрою, натомість для синхронізації отримуючого та надсилаючого пристроїв використовується бітова швидкість, що визначається як кількість переданих біт за одну секунду. Для зв'язку UART використовує 2 контакти: RX (отримання) та TX (надсилання). RX залежного пристрою з'єднується з TX головного пристрою і навпаки, RX головного пристрою з'єднується з TX залежного пристрою. UART підтримує одночасний

двонаправлений режим комунікації (повний дуплекс), коли одночасно залежний та головний пристрої надсилають певну інформацію та одночасно отримують певну інформацію.

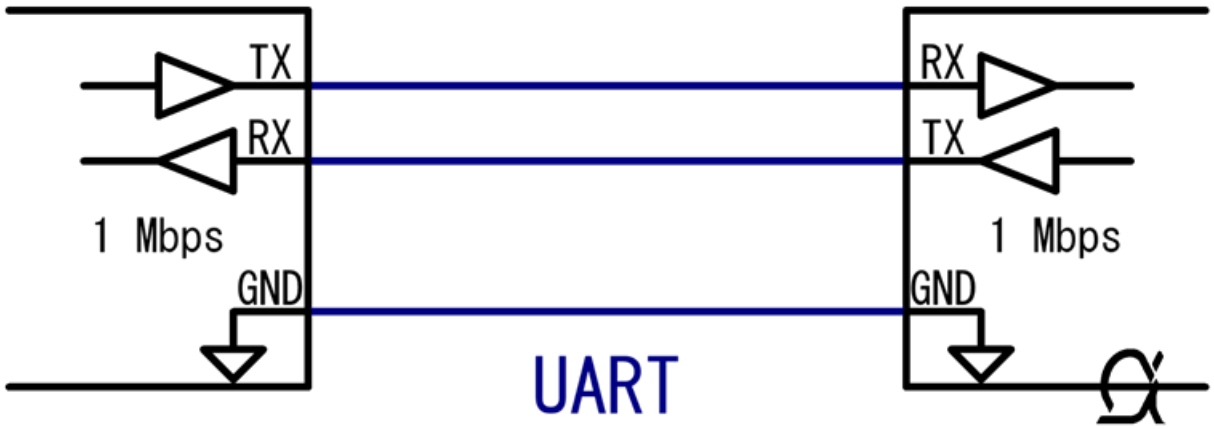

Рис. 2.15. Схематичний приклад UART зв’язку

Протокол I²C (також відомий як I2C) – послідовний синхронний протокл. Даний протокол використовує 2 контакти: SCL (сигнал годинника головного пристрою для синхронізації залежних пристроїв) та SDA (сигнал для передачі даних). I^2C опирається на синхронізацію синхронізацію отримуючих та надсилаючих пристроїв для передачі даних з коректною частотою, швидкістю та в правильні інтервали, коли інформація очікується бути надісланою/отриманою. Протокол I^2C дозволяє використовувати одну лінію зв’язку одночасно кількома залежними пристроями, на початку звернення головний пристрій надсилає адресу залежного пристрою якому адресується інформація, серед залежних пристроїв інформацію обробляє та опціонально надсилає відповідь лише той, чия адреса вказана на початку звернення.

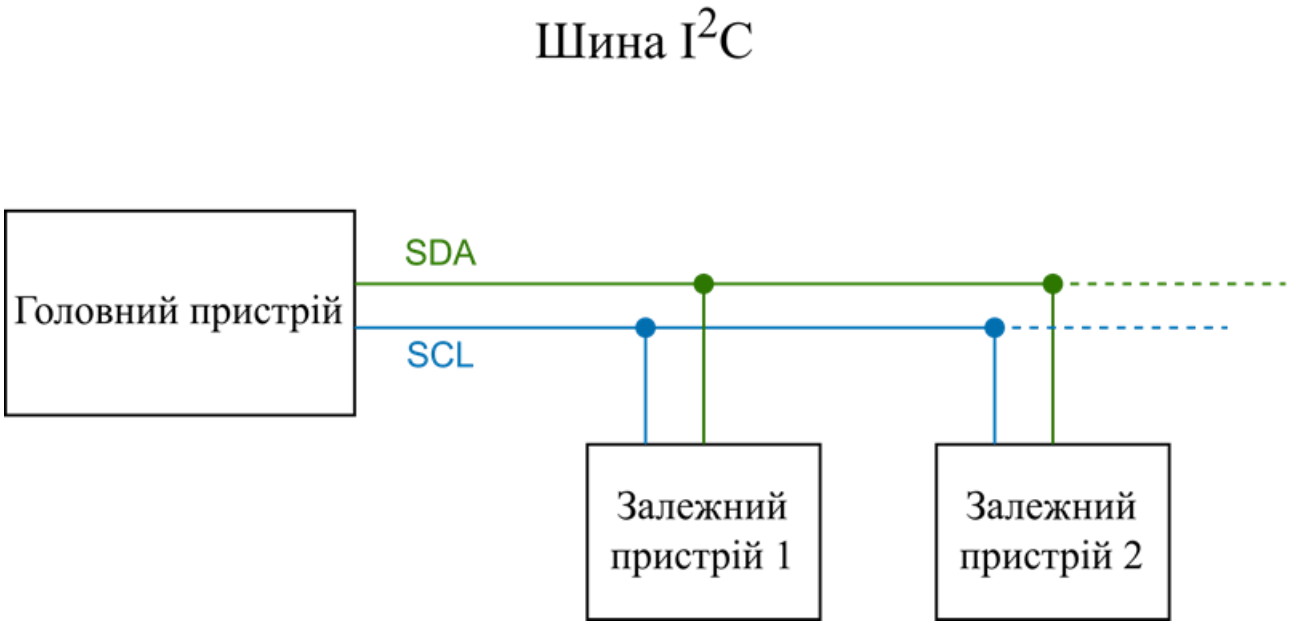

Рис. 2.16. Схематичне зображення протоколу I²C

Протокол SPI – послідовний синхронний протокол. Для зв'язку SPI використовує 4 контакти: CS (вибір залежного пристрою), SCLK (частота годинника головного пристрою для синхронізації), MOSI (вихідний потік від головного пристрою до залежного), MISO (вхідний потік до головного пристрою від залежного). На відміну від інших протоклів зв'язку контроллера з периферійними пристроями SPI дозволяє передачу відносно великих масивів інформації, що уможливлює його використання для, наприклад, модулів дисплеїв. Як і у випадку протоколу I²C, SPI використовує синхронізацію годинника між пристроями для коректної передачі інформації з правильними інтервалами та в коректні моменти.

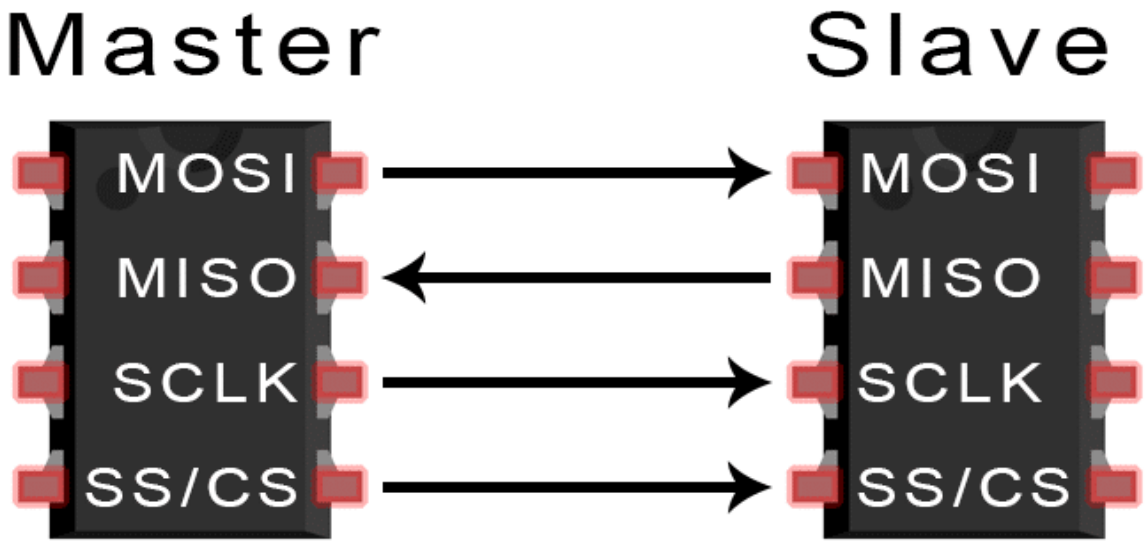

Рис. 2.17. Схематичний приклад протоколу SPI, де “Master” – головний пристрій, “Slave” – залежний пристрій

2.6. Виготовлення прототипу апаратно-програмної системи термінала

Розробка апаратної частини системи термінала починається з оцінки загальної кількості пристроїв та кількості і типів необхідних для їх роботи з’єднань. Сумарно проект використовує 5 модулів: ESP32-WROOM-32E Dev Module (мікроконтролер), PN532 (сканер NFC та RFID карток), Waveshare сканер штрих-кодів та QR кодів, SIM800L (модуль зв’язку через стільникову мережу) та Waveshare 3.5inch RPi LCD (A) (рідкокристалічний дисплей), кожен з цих модулів потребує 2 з’єднання для живлення та додаткові лінії для передачі даних.

Модуль PN532 має два варіанти підключення: SPI та I^2C, через простоту використання I^2C та відсутність потреби використання саме SPI, I^2C є більш оптимальним рішенням, також на відміну від SPI, I^2C потребує лише додаткових 2 з'єднання для передачі інформації. Живиться PN532 від 5 вольт. Модуль SIM800L використовує з’єднання через UART, що потребує два контакти для передачі даних. Для живлення модуль SIM800L використовує 3,3 вольта. Сканер одновимірних та двовимірних кодів від Waveshare також використовує UART для комунікації з платою контролера, живиться даний модуль від 5 вольт.

Дисплей Waveshare 3.5inch RPi LCD (A) згідно офіційної документації [22] використовує протокол SPI для отримання зображення для виводу. Для коректної та ефективної роботи дисплея без використання панелі розпізнавання дотиків достатньо задіяти лише 5 з'єднань (CS, DC, MOSI, SCK, RST), з'єднання DC використовується для задання режиму передачі даних (режим передачі налаштувань дисплея, та режим передачі графічної інформації), контакт RST (RESET) дозволяє виконати скидання попередньо встановлених налаштувань дисплея до початкових значень (може бути використано у разі невдалого налаштування, або повторної ініціалізації), живиться дисплей від 5 або ж 3,3 вольт на вибір.

Тож сумарно для під'єднання усіх необхідних модулів необхідно задіяти 19 з'єднань для живлення та передачі даних. Через відносно велику кількість з'єднань та модулів оптимальним рішенням є розбиття усієї системи на окремі модулі, що спростить виготовлення та обслуговування виробу, а для з'єднання цих модулів з головною платою мікроконтролера використовувати певний вид конектора для надійного, проте за потреби не перманентного з'єднання, що спростить збирання та транспортування виробу. Оптимальним вибором конектора є роз'єм XH2.54-2P, що існує у різних варіаціях з різною кількістю контактів та просту у монтуванні на стандартні макетні перфоплати з кроком монтажних точок у 2,54 мм, на яких і буде з міркувань зручності та відповідності кроку монтажних точок на платах розширення реалізовані модулі системи термінала.

Окремої уваги при виготовлені плат модулів було приділено головному модулю мікроконтролера та модулю дисплея у зв'язку з їх комплексністю та кількістю з'єднань.

Головний модуль мікроконтролера має містити всі порти передачі даних, що використовують плати розширення та необхідну кількість роз'ємів живлення 5 вольт, та живлення 3,3 вольта разом з одним додатковим роз'ємом на кожну

лінію для подальшої можливості живлення мікроконтролера та всієї системи як від 5, так і від 3,3 вольт. Усі роз'єми живлення під'єднані до відповідних контактів живлення 5 вольт та 3,3 вольти. У разі живлення мікроконтролера від 5 вольт він здійснюватиме перетворення живлення для подачі струму 3,3 вольта на відповідні контакти і навпаки, контролер здійснюватиме перетворення живлення у разі живлення від 3,3 вольт для формування лінії 5 вольт. Через відсутність на ESP32-WROOM-32E достатньої кількості апаратних UART портів, було використано контакти D32 та D33 для використання в ролі програмного UART порта, що забезпечує ту ж функціональність що і апаратний.

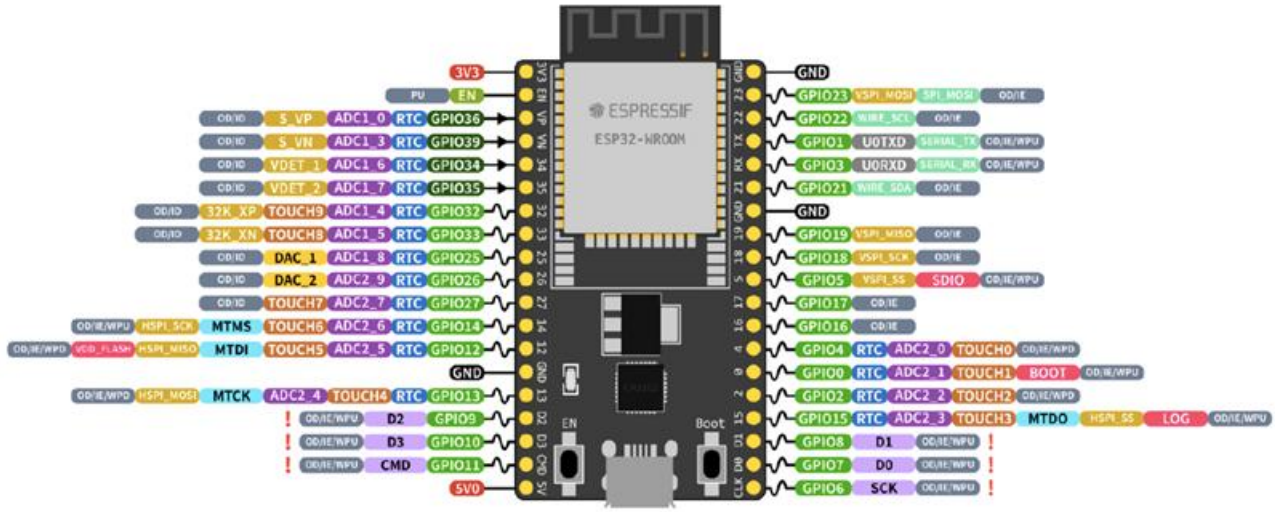

Рис. 2.18. Розшифровка окремих контактів на платі ESP32-WROOM-32E-Dev-Board

Таблиця 2.2

Функції та відповідності контактів ESP32-WROOM-32E-Dev-Board

Контакт	**Функція та використання**
D32	програмний UART RX порт (для модуля SIM800L)
D33	програмний UART TX порт (для модуля SIM800L)
RX	апаратний контакт UART RX (для налагодження)
TX	апаратний контакт UART TX (для налагодження)
D16	апаратний UART RX порт (для сканера 1D/2D кодів)
D17	апаратний UART TX порт (для сканера 1D/2D кодів)
D15	апаратний SPI CS порт (для модуля дисплея)
D2	програмований порт DC (визначення режиму дисплея)
D23	апаратний SPI MOSI порт (для модуля дисплея)
D18	апаратний SPI SCK(SCLK) порт (для модуля дисплея)
EN	контакт “увімкнено” на ESP32 (для синхронізації дисплея через контакт RST)
D21	апаратний порт I^2C SDA (для сканера RFID та NFC карток)
D22	апаратний порт I^2C SCL (для сканера RFID та NFC карток)
5v	контакт лінії вхідного/вихідного живлення 5 вольт
3v3	контакт лінії вхідного/вихідного живлення 3,3 вольта
GND	контакт від’ємної полярності живлення (спільні для 5 та 3,3 вольт)

Модуль дисплея було створено відповідно до офіційної документації [22], розшифровки пінів дисплея, інструкцій до бібліотеки роботи з подібними дисплеями [23], адже наявність додаткових пінів DC та RST, що виходять за межі базового SPI, коректний метод під’єднання дисплея є менш очевидним. Для

приєднання дисплея на плату модуля було змонтовано подвійний ряд пінів “тато-тато” з кроком 2,54 мм, що з’єднуються з двадцяти-шести контактним роз’ємом дисплея.

Оскільки модуль сканування одновимірних та двовимірних кодів в заводському виконанні являє собою зручний для інтеграції модуль з шатним роз’ємом, було використано чотирьох контактний кабель PH2.0 з двома двохконтактними штекерами XH2.54-2P для приєднання до UART порта мікроконтролера та лінії живлення.

Фінальний дизайн апаратного забезпечення системи термінала включає в себе чотири окремих модулі зі змонтованими платами розширення та роз’ємами XH2.54-2P для підключення, а також заводський модуль зчитування штрих-кодів та QR кодів (див. рис. 2.19).

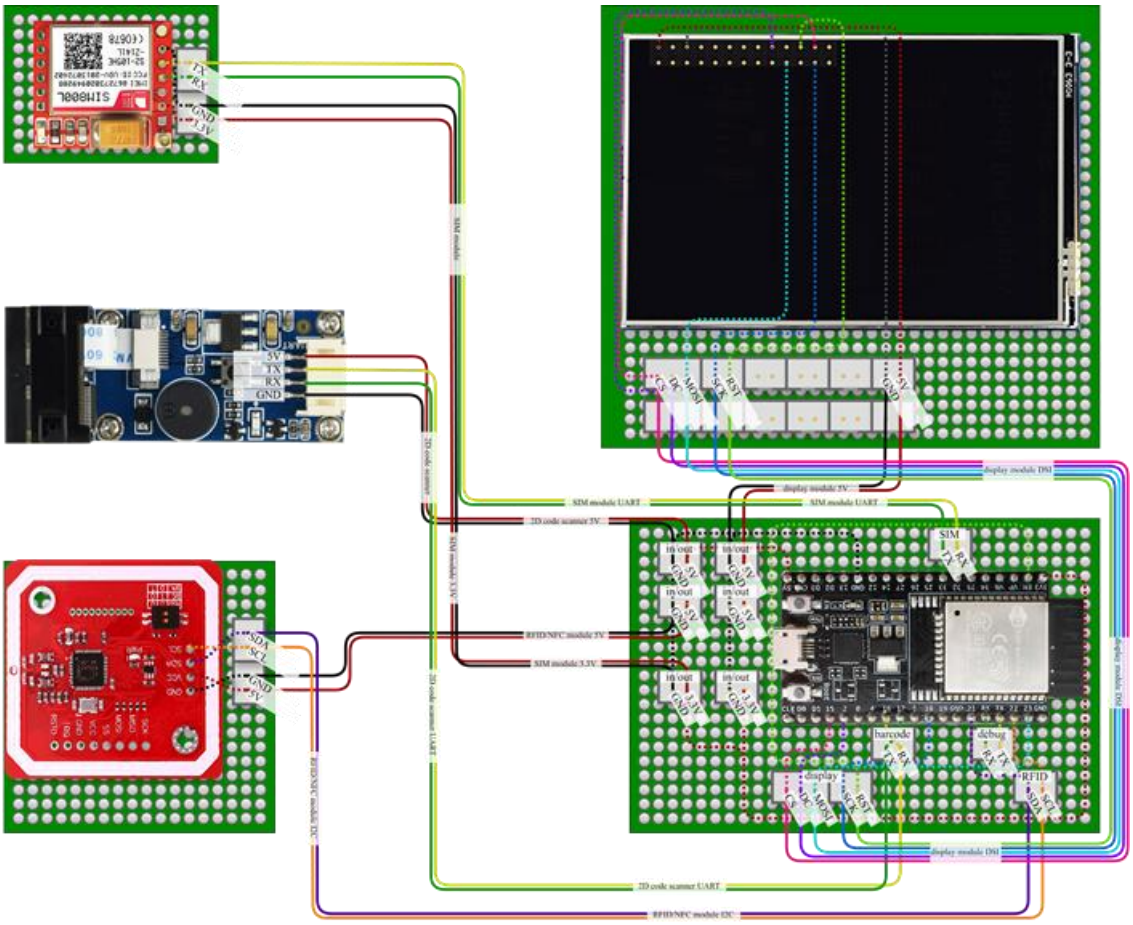

Рис. 2.19. Схематичне зображення апаратного забезпечення системи термінала

Для реалізації програмного забезпечення системи термінала було використано середовище розробки для мікроконтролерів Arduino IDE 2. Arduino IDE 2 надає інструменти для написання, відлагодження, компіляції та вивантаження коду на мікроконтролер. Також Arduino IDE надає засоби для моніторингу серійного з'єднання з мікроконтролером, встановлення параметрів роботи мікроконтролера, оновлення його прошивки (операційної системи). Проте головною особливістю цього середовища є можливість швидко та ефективно встановлювати нові бібліотеки та пакети підтримки нових моделей мікроконтролерів. Arduino IDE 2 є простим для освоєння та використання середовищем, адже бере на себе значну кількість дрібних технічних складностей та дозволяє сфокусуватись на реалізації проекту.

Можливість роботи з ESP32 в середовищі Arduino IDE 2 реалізується за рахунок встановлення додаткових пакетів шляхом додавання посилання "https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json" до списку "додаткових менеджерів плат" у налаштуваннях Arduino IDE 2. Після додавання необхідно встановити однойменний пакунок "esp32" в менеджері плат та вибрати тип плати "ESP32 Dev Module" при виявленні мікроконтролера середовищем розробки при його під'єднанні до ПК.

Для роботи мікроконтролера з дисплеєм та виведенням на нього інформації було використано наступні бібліотеки: SPI (реалізація SPI протоколу) та TFT_eSPI (бібліотека для роботи з TFT дисплеями через протокол SPI). Для початку роботи з дисплеєм необхідно викликати метод "begin" з бібліотеки TFT_eSPI для ініціалізації дисплея та задати його базове налаштування у вигляді положення за допомогою метода "setRotation", що приймає числові значення від 0 до 7, де 0 – стандартна орієнтація, а 3 – поворот на 270° за годинниковою стрілкою, значення від 4 та вище встановлюють режим віддзеркалення. Для

виведення певного тексту на екран, необхідно задати положення курсора на поверхні дисплея через метод “setCursor”, передавши кординати x та y, та викликати метод “println” для виведення на екран стрічки тексту передавши у параметри бажаний текст. Подібним чином викликаючи базові інструкції у вигляді методів можна виводити не лише текст, але й інші графічні зображення, примітиви тощо.

Робота з модулем PN532 потребує використання бібліотек Wire, PN532_I2C, PN532 та NfcAdapter. Ініціалізація модуля здійснюється шляхом виклику метода “begin” з бібліотеки PN532, встановленням максимальної кількості спроб очікування та зчитування карти передавши кількість спроб у шістнадцятковій нотації до метода “setPassiveActivationRetries” та перевівши модуль у режим зчитування карток простим викликом методі “SAMConfig”. Спроба зчитування картки відбувається шляхом виклику метода “readPassiveTargetID” та передачі чотирьох параметрів: швидкість передачі даних згідно стандарту, змінна масиву для збереження зчитаної з карти інформації, змінна для збереження довжини зчитаної інформації та максимальний час очікування картки. У разі повернення методом “readPassiveTargetID” значення true (істина), зчитування картки пройшло вдало, після чого зчитану інформацію можна піддати подальшій обробці та відправці на сервер проекта для запиту отримання доступу.

Реалізація комунікації зі сканером штрих-кодів та QR кодів є найпростішою серед перелічених, адже для ініціалізації даного сканера необхідно лише викликати команду “Serial2.begin”, та передати швидкість передачі даних по UART (9600 для даного сканера згідно документації [24]), формат передачі даних (“SERIAL_8N1” для даного сканера згідно документації [24]), нумерування піна RX та нумерування піна TX. Для перевірки наявності зчитаного значення закодованого у відповідному коді необхідно викликати команду “Serial2.available”, у разі повернення true (істина), можна здійснити

отримання декодованого значення викликом команди “Serial2.readString” та збереженням отриманої стрічки в змінну для подальшого використання зчитаного ідентифікатора картки.

Через нестачу на платі ESP32-WROOM-32E-Dev-Board апаратних UART портів, для роботи з модулем SIM800L необхідно використати бібліотеку SoftwareSerial, що дозволяє ініціалізувати програмний UART порт на вільних програмованих контактах плати мікроконтролера. Ініціалізація програмного UART порта для модуля SIM800L виглядає наступним чином: виклик команди “SoftwareSerial softSerial(RX_SIM, TX_SIM);”, де “RX_SIM” та “TX_SIM” – це програмовані контакти для з’єднань TX та RX відповідно; виклик команди “softSerial.begin(9600)” для безпосередньої ініціалізації програмного UART порта на швидкості сумісній з модулем SIM800L. Після ініціалізації програмного UART необхідно здійснити базове налаштування модуля в мобільній мережі шляхом надсилання параметрів та атрибутів до модуля використовуючи метод виведення нової стрічки “println”. Надсилання запитів через мобільну мережу до сервера також полягає у відправці на SIM800L відповідних атрибутів та параметрів, що містять тип протоколу, адресу звернення, тип вмісту тощо, разом з безпосереднім корисним навантаженням запиту. Отримання ж відповіді від сервера відбувається аналогічно до процесу отримання декодованого значення сканованого модулем сканера одновимірних та двовимірних кодів.

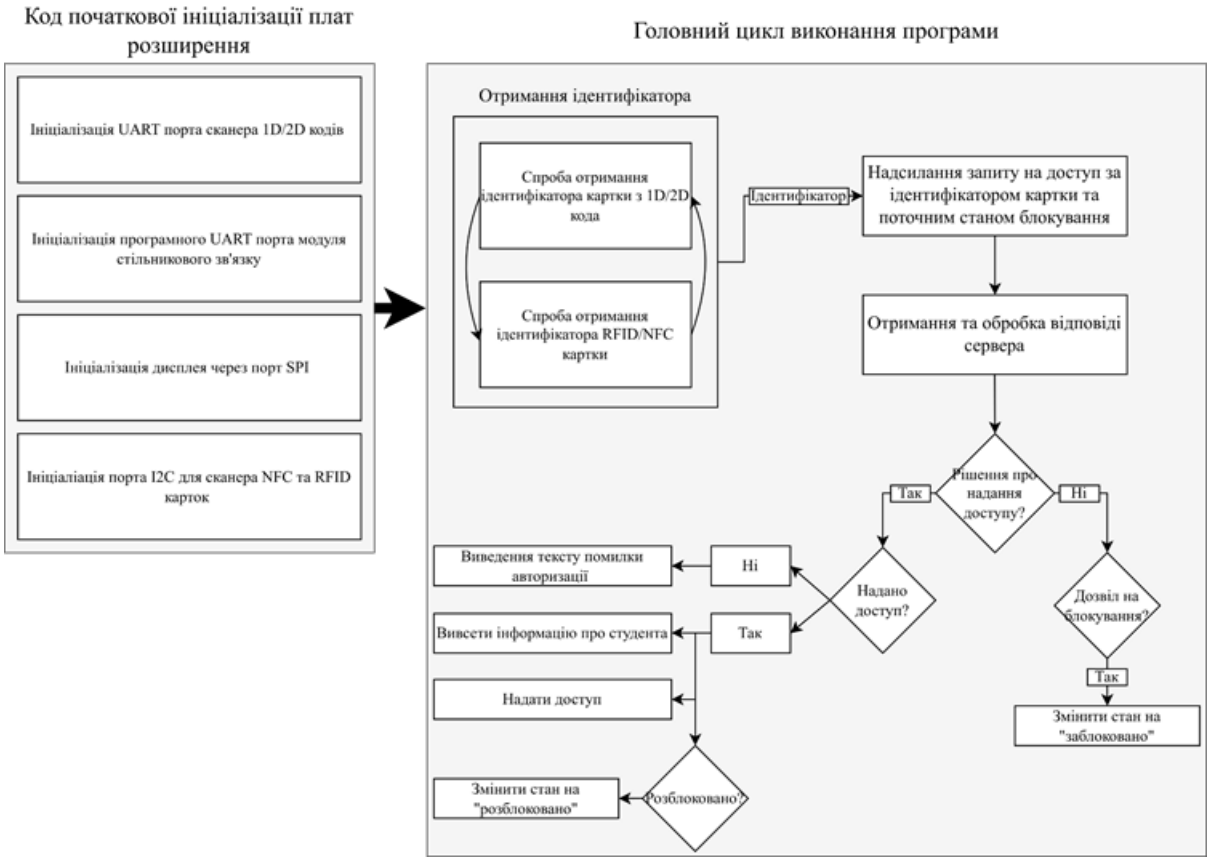

Рис. 2.20. Схематичне зображення алгоритму прошивки термінала.

РОЗДІЛ 3
ТЕСТУВАННЯ АПАРАТНО-ПРОГРАМНОГО КОМПЛЕКСУ АВТОМАТИЗОВАНОЇ СИСТЕМИ ПРОПУСКУ

Кінцевим продуктом розробки є автоматична система контролю пропуску відвідувачів. Дана система об'єднує в собі апаратний комплекс та серверне програмне забезпечення, що взаємодіють один з одним. Сценарій використання системи виглядає наступним чином: користувач приходить до термінала контролю пропуску та сканує штрих-код на своєму студентському квитку, або ж RFID ключ-картку, що також як і студентський квиток містить унікальний ідентифікатор; після сканування термінал відправляє запит на сервер та вміщає в нього відсканований унікальний ідентифікатор користувача та свій поточний стан (стан термінала: заблоковано, розблоковано); при отриманні інформації від термінала сервер приймає рішення про надання доступу користувачеві на основі наступних даних: "чи розблоковано зараз термінал?", "чи дозволено цьому користувачеві бути в цьому приміщенні будь-коли?", "чи відбувається зараз в даному приміщенні певна подія до якої причетний користувач?", "чи було аутентифіковано користувача в системі?", "чи користувач що покидає приміщення був єдиним у ньому?"; після прийняття рішення про дозвіл, або ж заборону доступу чи блокуванню приміщення сервер надсилає його у відповідь на початковий запит до терміналу; після отримання відповіді термінал виконує відповідні дії згідно отриманого рішення та виводить на екран повідомлення.

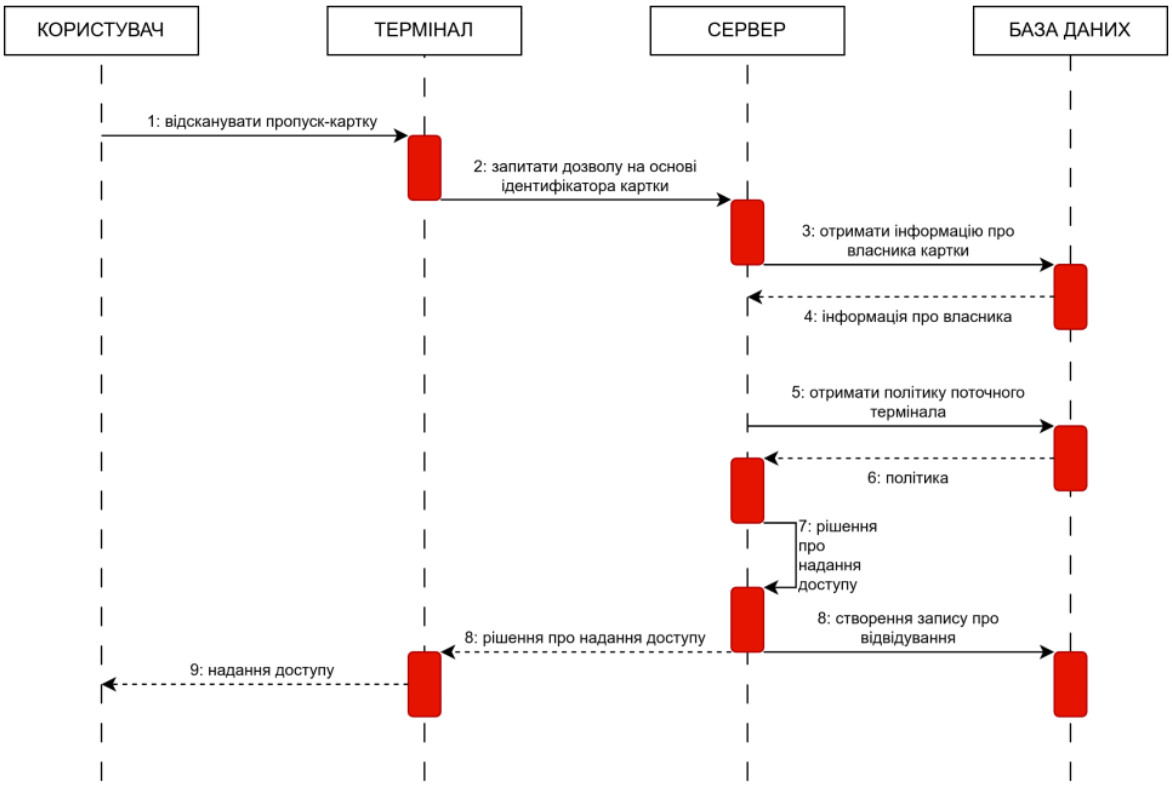

Рис. 3.1. Графічне зображення взаємодії користувача з системою

Подальше практичне тестування системи продемонструвало відповідність фактичних характеристик. Залежно від успішності процесу отримання доступу користувачеві представлено відповідне повідомлення на дисплеї термінала.

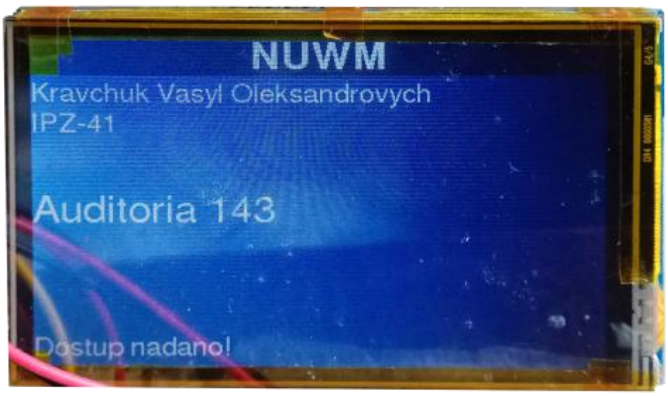

Рис. 3.2. Успішна спроба отримання доступу, повідомлення з термінала

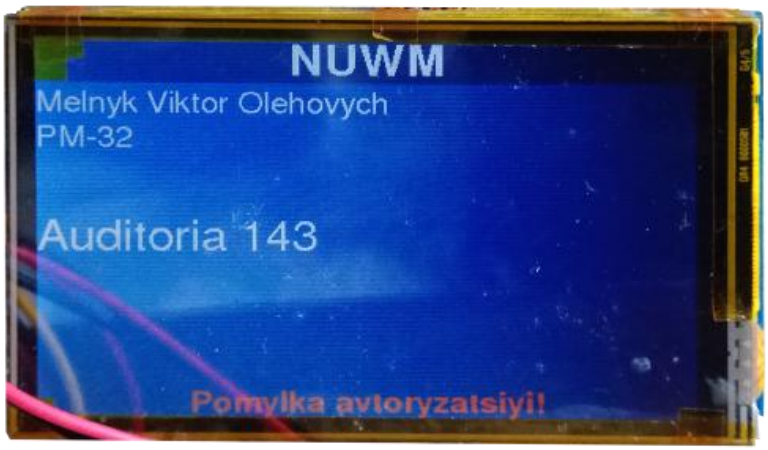

Рис. 3.3. Невдала спроба отримання доступу, повідомлення з термінала

ВИСНОВКИ

В межах даного проєкту було розроблено апаратно-програмний комплекс системи автоматичного контролю пропуску та запису відвідуваності. Проєкт зосереджується на покращенні рівня безпеки учасників навчального процесу водночас покращуючи та пришвидшуючи певні процеси, як от автоматичний запис відвідуваності та оперативне та динамічне надання доступу до приміщень та територій навчального закладу.

Апаратно-програмна складова проекту була розроблена довкола сучасного мікроконтролера для розробки рішень платформи інтернету речей – ESP32. Розроблений апаратно-програмний комплекс являє собою термінал, що дозволяє швидко та ефективно здійснювати сканування студентського квитка, або ж спеціальних ключ-карток. Важливою особливістю даної системи є не лише здатність швидко та надійно здійснювати контроль пропуску, але й здатність даної системи автоматичного ведення запису присутності, що потребує усього лише відсканувати картку з відповідним ідентифікатором студента. Такий підхід дозволяє економити час як викладачів, так і студентів. Додатково автоматизація контролю пропуску дозволяє зменшити навантаження обслуговуючого персоналу.

Для подальшого розвитку теми кваліфікаційної роботи можна зосередитись на реалізації таких можливостей системи:

- інтеграція з вже наявними навчальними платформами та системами управління навчальним процесом;
- створення нового функціоналу та розширення можливостей для задоволення потреб більшої кількості навчальних закладів та розширення цільової аудиторії проєкта.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Фреймворк NestJS. URL:https://nestjs.com (дата звернення 21.05.24)
2. ECMAScript (JavaScript). URL:https://ecma-international.org/publications-and-standards/standards/ecma-262 (дата звернення 21.05.24)
3. Typescript система типізації для ECMAScript. URL:https://www.typescriptlang.org (дата звернення 21.05.24)
4. Bun (середовище виконання коду ECMAScript та TypeScript) URL:https://bun.sh (дата звернення 21.05.24)
5. PostgreSQL (реляційна база даних) URL:https://www.postgresql.org/?ref=jameswarrick.com (дата звернення 21.05.24)
6. ESP32 (стаття на Wikipedia) URL:https://uk.wikipedia.org/wiki/ESP32 (дата звернення 21.05.24)
7. ESP32 (сайт розробника) URL:https://www.espressif.com/en/products/socs/esp32 (дата звернення 21.05.24)
8. NodeMCU (англомовна стаття на Wikipedia) URL:https://en.wikipedia.org/wiki/NodeMCU (дата звернення 21.05.24)
9. NodeMCU (сайт розробника) URL:https://www.nodemcu.com/index_en.html (дата звернення 21.05.24)
10. Moodle (офіційний сайт проекта) URL:https://moodle.org/ (дата звернення 21.05.24)
11. openSUSE MicroOS (офіційний сайт проекта) URL:https://microos.opensuse.org/ (дата звернення 22.05.24)

12. BTRFS (офіційна документація на порталі документації ядра Linux) URL:https://docs.kernel.org (дата звернення 22.05.24)

13. ARM процесорна архітектура (офіційний сайт розробника архітектури) URL:https://www.arm.com/architecture/cpu (дата звернення 22.05.24)

14. Портал:Leap (офіційний портал openSUSE Leap) URL:https://en.opensuse.org/Portal:Leap (дата звернення 22.05.24)

15. Бекдор XZ Utils (стаття на Wikipedia про даний випадок) URL:https://en.m.wikipedia.org/wiki/XZ_Utils_backdoor (дата звернення 22.05.24)

16. Модуль розробника на базі ESP32-WROOM-32E URL:https://arduino.ua/prod2041-wi-fi-modyl-luanode32-s-esp-32 (дата звернення 25.05.24)

17. Офіційна документація модуля ESP32-WROOM-32E/UE URL:https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf (дата звернення 25.05.24)

18. Модуль сканера RFID/NFC V3 на PN532 URL:https://arduino.ua/prod2844-modyl-skanera-rfidnfc-v3-na-pn532 (дата звернення 25.05.24)

19. Сканер штрих-кодів, QR-кодів 1D/2D від Waveshare URL:https://arduino.ua/prod3012-skaner-shtrih-kodov-qr-kodov-1d2d-ot-waveshare (дата звернення 25.05.24)

20. GSM модуль на SIM800L URL:https://arduino.ua/prod1665-gsm-modyl-na-sim800l (дата звернення 25.05.24)

21. 3.5" 480x320 TFT сенсорний дисплей RR035 для Raspberry Pi URL:https://arduino.ua/prod2001-3-5-480x320-tft-sensornii-displei-rr035-dlya-raspberry-pi-ot-elecrow (дата звернення 25.05.24)

22. 3.5inch RPi LCD (A) URL:https://www.waveshare.com/wiki/3.5inch_RPi_LCD_(A) (дата звернення 26.05.24)

23. TFT_eSPI URL:https://github.com/Bodmer/TFT_eSPI (дата звернення 26.05.24)

24. Barcode Scanner Module User Manual URL:https://files.waveshare.com/upload/d/dd/Barcode_Scanner_Module_Setting_Manual_EN.pdf (дата звернення 26.05.24)