Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система управління поведінкою пристроїв на заданих ділянках
(тема)

Виконав:
здобувач __4___ року навчання
групи ПЗПІ-21-8

Олександр СИНЕЛЬНІКОВ
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. каф. ПІ Володимир ЛЕЩИНСЬКИЙ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Синельнікову Олександру Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система управління поведінкою пристроїв на заданих ділянках

Затверджена наказом по університету від № 397 Ст від 19.05.2025

2. Термін подання студентом роботи до екзаменаційної комісії 19.06.2025

3. Вихідні дані до роботи Розробити програмну систему для управління автономними пристроями, що складається з серверу, клієнтського веб-застосунку та застосунку для симуляції роботи системи в реальному світі

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	14.05.2025	*виконано*
2	Створення специфікації ПЗ	16.05.2025	*виконано*
3	Проектування ПЗ	20.05.2025	*виконано*
4	Розробка ПЗ	23.05.2025	*виконано*
5	Тестування ПЗ	25.05.2025	*виконано*
6	Оформлення пояснювальної записки	31.05.2025	*виконано*
7	Підготовка презентації та доповіді	10.06.2025	*виконано*
8	Попередній захист	12.06.2025	*виконано*
9	Нормоконтроль, рецензування	14.06.2025	*виконано*
10	Здача роботи у електронний архів	16.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	19.06.2025	*виконано*

Дата видачі завдання «25» квітня 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. каф. ПІ Володимир ЛЕЩИНСЬКИЙ
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка містить: 56 стор., 7 рис., 1 табл., 12 джерел.

АВТОНОМНІ ПРИСТРОЇ, АВТОРИЗАЦІЯ, БЕЗПЕКА, КЕРУВАННЯ, ОПЕРАТОР, LORA, NEXTJS, POSTGRESQL, REACTJS, REST API, TOKEN, WEB APP.

Ціллю проекту є створення програмної системи керування автономними пристроями на заданих ділянках. В розробку системи входить її проектування, розробку та розгортання серверної частини системи, а також демонстрація її роботи на прикладі автономних приладів.

Метою розробки є створення надійної та безпечної системи для керування пристроями без постійного підключення до інтернету. Керування ними буде виконано локальними операторами окремих мереж за допомогою певного девайсу контролеру, якому надається тимчасовий доступ сервер у вигляді токену з автономним підтвердженням його дійсності девайсом.

Для імплементації серверної частини було використано фулстак фреймворк Next.js для створення надійного REST API та сучасного Web застосунку для адміністрації системи на основі React.js. Іншу частино виконано у вигляді симуляції за основу якої буде взято поведінку пристроїв, що працюють по протоколам LoRa.

В результаті виконаної роботи було зроблено проектування та реалізацію програмної системи для керування пристроями без постійного підключення до Інтернету з можливістю виконувати свої основні функції автономно (вмикання ліхтарів, полив, військове застосування, тощо).

ABSTRACT

AUTHORIZATION, AUTONOMOUS DEVICES, LORA, MANAGEMENT, NEXTJS, OPERATOR, POSTGRESQL, REACTJS, REST API, SECURITY, TOKEN, WEB APP.

The goal of the project is to create a software system for controlling autonomous devices in specified areas. The development of the system includes its design, development and deployment of the server part of the system, as well as a demonstration of its operation using the example of autonomous devices.

The goal of the development is to create a reliable and secure system for controlling devices without a permanent connection to the Internet. They will be controlled by local operators of individual networks using a specific controller device, which is granted temporary access to the server in the form of a token with autonomous confirmation of its validity by the device.

To implement the server part, the Next.js full-stack framework was used to create a reliable REST API and a modern Web application for system administration based on React.js. The rest of the work was performed in the form of a simulation based on the behavior of devices operating on LoRa protocols.

As a result of the work performed, a software system was designed and implemented for controlling devices without a permanent Internet connection with the ability to perform its main functions autonomously (turning on lights, watering, etc.).

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API (Application Programming Interface) – інтерфейс, що дозволяє програмам взаємодіяти між собою.

JSON (JavaScript Object Notation) – формат зберігання та передачі структурованих даних у вигляді тексту.

ID (Identifier) – унікальний ідентифікатор об'єкта або користувача в системі.

HTTP (HyperText Transfer Protocol) – протокол передачі даних між клієнтом і сервером у веб-мережі.

JWT (JSON Web Token) – стандартизований формат токенів для безпечної передачі даних.

LoRa (Long Range) – протокол бездротового зв’язку з низьким енергоспоживанням.

LoRaWAN – розширення LoRa з підтримкою мережевої архітектури.

LPWAN (Low Power Wide Area Network) – тип мережі, яка дозволяє пристроям передавати дані на великі відстані з мінімальним енергоспоживанням.

OAuth 2.0 – протокол авторизації, що дозволяє безпечно делегувати доступ до ресурсів.

ORM (Object-Relational Mapping) – технологія для взаємодії з базами даних через об'єктно-орієнтовані підходи.

REST API – архітектурний стиль для побудови веб-сервісів.

RSA (Rivest-Shamir-Adleman) – асиметричний криптографічний алгоритм для шифрування і підписів.

ВСТУП

В нашому світі все більше і більше пристроїв рекламують себе як «смарт», але насправді все, що вони роблять – це обмежують застосування цих пристроїв без підключення до інтернету. Постійне підключення до інтернету обмежує область застосування приладів та збільшує вразливість мереж до атак [1].

При застосуванні приладів на певних територіях ними не можна керувати з сервера, а саме тому вони мають працювати автономно. Метою цієї роботи є проектування системи керування автономними приладами на заданих ділянках. Така система має надавати можливість операторам керувати пристроями, застосовувати автономну систему авторизації та тимчасові ключі-токени для підтвердження повноважень користувача у локальній мережі, використовувати сучасні технології взаємодії та аутентифікації на серверному рівні, бути надійною та застосовувати протоколи дистанційної передачі інформації з мінімальним відбитком у локальному радіо-просторі.

Для виконання цієї задачі ми проаналізуємо можливі імплементації такої системи, готові часткові рішення нашої мети, розглянемо можливі сучасні технології для надання комфортного досвіду користувача, а також забезпечення безпеки даних системи. Така система може бути застосована у великій кількості сфер (агрономічна діяльність, муніципальне управляння, воєнне застосування) з різноманітним застосуванням подібного роду автономних пристроїв (автономний полив, керування освітленням, автономна система спостереженням).

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

При згадці про системи керування приладами одразу на думку приходить девайси IoT, але такі девайси зрідка є повністю автономними, бо вони є лише єдиним компонентом більшої системи, більше за те для повноцінної роботи потребують постійного підключення до інтернету, що дуже сильно обмежує можливі функції та галузі застосування. Більшість доступних готових рішень надають широкі можливості, однак потребують постійного підключення девайсів до мережі, що не підходить для нашого проекту.

Сучасна індустрія автономних пристроїв активно розвивається у напрямі створення систем, які здатні ефективно працювати в умовах обмежених ресурсів: енергії, обчислювальних потужностей, пропускної здатності каналів зв'язку. Пристрої, що функціонують у подібному середовищі, зазвичай розгортаються у важкодоступних місцях або працюють тривалий час без зовнішнього живлення. Як наслідок, традиційні моделі постійного з'єднання та високочастотного обміну даними є непридатними.

1.2 Аналіз конкурентів

Популярні комерційні рішення, такі як AWS IoT, Azure IoT Hub і Google Cloud IoT Core, були створені з урахуванням глобальної доступності хмарної інфраструктури, високої масштабованості, централізованого збору телеметрії та потужних засобів аналітики. Але ці системи мають спільний недолік у контексті автономних пристроїв – повну залежність від постійного зв’язку з інтернетом.

AWS IoT Core пропонує складну екосистему для побудови IoT-додатків, включаючи вбудовану підтримку багатьох протоколів передачі даних, а також можливість підключення до інших сервісів AWS, таких як Lambda, S3 та DynamoDB. Проте всі ці можливості залежать від стійкого з'єднання з хмарою

Amazon. При відсутності інтернету, пристрій втрачає зв'язок з основою логічної системи, оскільки авторизація, обробка даних і навіть обробка даних делеговані хмарі. Крім того, через складність інфраструктури, автономна робота на рівні пристрою вимагає використання додаткових сервісів AWS, що ускладнює архітектуру системи та збільшує вартість її розгортання.

При використанні Azure IoT Hub виникають схожі проблеми. Ця система побудована з фокусом на централізоване керування пристроями, аналітику в реальному часі та інтеграцію з іншими сервісами Azure. При підключенні пристрою до Azure, створюється цифрове відображення пристрою в хмарі, який відображає його стан. Це передбачає, що кожна дія та зміна стану повинна бути синхронізована з хмарою. Локальне прийняття рішень або авторизація користувача без контакту з хмарою в моделі Azure не підтримується.

Google Cloud IoT Core був готовим рішенням для керування пристроями від Google, що виявився ще менш придатний для виконання автономних задач. Google анонсувало про закриття сервісу до серпня 2023 року [2], що демонструє ненадійність використання централізованих сервісів для керування критичної інфраструктури.

1.3 Постановка задачі

В усіх трьох випадках головною перевагою є аналітика великих даних, хмарне масштабування і централізоване керування тисячами пристроїв, але це і стає їх головним недоліком, коли мова заходить про побудову автономної системи. В жодному з рішень немає вбудованої підтримки динамічної делегації повноважень без підключення до сервера, а також неможливо реалізувати контроль доступу між пристроями в межах локальної мережі без використання зовнішніх сервісів.

Для керування більш автономними пристроями нам знадобиться зв’язати пристрої через LPWAN [3] – низько енергетична мережа широкого радіусу, що забезпечить зв’язок між пристроями, що працюють від батареї, на великій відстані

з низькою швидкістю передачі даних, але з найменшими можливим використанням електроенергії.

Однією з відомих технологій, яка вирішує ці виклики, є LoRa та LoRaWAN – бездротова технологія, що дозволяє передавати дані на великі відстані з низьким енергоспоживанням. Вони широко використовується у таких сферах, як моніторинг стану навколишнього середовища, сільське господарство, системи безпеки, інфраструктура розумного міста.

При імплементації нашої системи можна використовувати хоча б протоколи LoRa, з власною імплементацією заміни LoRaWAN. LoRaWAN хоч і має систему аутентифікацію і шифрування, але бракує можливості тимчасової делегації прав керування, коли один пристрій має надати іншому обмежений доступ до певної групи девайсів на короткий час, а також мережа є вразливою до скомпрометованих пристроїв LoRaWAN, що можуть бути використанні з лихими намірами (в залежності від ризику в галузі застосування).

Звісно існують такі часткові рішення як система MySensors, але вони в більшості своїй розраховані на любителів та DIY проекти, а не на використання у індустріальному середовищі. Вона надає базову підтримку створення мережі сенсорів з можливістю релейного передавання повідомлень, проте серйозно обмежена в аспектах масштабованості та гнучкості при впровадженні складних логік доступу. Також, ця система особисто нам не підходить через обмеження імплементації системи захисту [4].

Ще одне часткове рішення – використання протоколу MQTT на базі автономних мікроконтролерів. Це дозволяє побудувати відносно легку систему обміну повідомленнями між пристроями, з підтримкою захищеного зв'язку, проте така модель потребує постійної доступності брокера з надійним інтернет підключенням.

Крім того, ще одним суттєвим обмеженням існуючих рішень є складність їх масштабування та адаптації до специфічних сценаріїв використання. У випадках, коли необхідно реалізувати нестандартну логіку взаємодії пристроїв, забезпечити локальну автономну обробку даних, або вбудувати специфічні

механізми контролю доступу, використання готових комерційних платформ стає малоефективним. Такі платформи, як правило, мають фіксовану архітектуру і пропонують обмежену конфігурацію, що робить неможливим реалізацію індивідуальних вимог без значних компромісів.

Таким чином, для побудови надійної та гнучкої системи керування автономними пристроями у розподіленому середовищі з обмеженим енергоспоживанням та зв'язком, виникає необхідність розробити власну архітектуру, яка враховуватиме як переваги наявних технологій, так і їхні обмеження.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Наша система має забезпечувати керування автономними пристроями в умовах, де постійне підключення до інтернету неможливе або небажане. Це вимагає принципово іншого підходу порівняно з традиційними IoT-рішеннями.

Система повинна надавати можливість операторам керувати пристроями через локальну мережу, що створюють між собою пристрої, без необхідності зовнішнього інтернет-з'єднання. Це означає, що всі критичні функції, а саме аутентифікація, передача команд та моніторинг стану мають працювати автономно. Наприклад, коли оператор підходить до теплиці, він має отримувати доступ до системи поливу через свій девайс без необхідності підключатися до серверу.

Система має підтримувати тимчасову делегацію повноважень керування пристроями. Коли адміністратор системи хоче делегувати повноваження помічнику, система має генерувати одноразовий токен з обмеженим терміном дії. Підключення та будь-яка взаємодія з пристроями за допомогою простроченого токену, або девайсом який не має на його використання повноважень в жодному разі не має бути можливим.

Система безпеки має включати кілька рівнів захисту:

– двофакторна аутентифікація для всіх членів організації;

– всі команди, що передаються по радіоканалу, мають бути зашифровані;

– керування пристроями має бути можливим тільки при наявності у оператора відповідного тимчасового токену;

– кожен пристрій повинен мати унікальний ідентифікатор і підтримувати механізм блокування при спробі несанкціонованого доступу.

Важливим фактором є енергоефективність. Пристрої мають працювати від акумуляторів якомога довше без підзарядки. Це буде досягатися за рахунок оптимізації режимів роботи, наприклад, датчики активуються лише у визначені

інтервали часу, радіопередавач активується лише коли до пристрою звертаються, а передача даних відбувається у стислому форматі.

Для зручності моніторингу система має зберігати локальну історію подій. Це дозволить аналізувати роботу пристроїв і виявляти аномалії навіть без постійного підключення до центрального сервера. У разі появи інтернет-з'єднання ці дані можуть синхронізуватися з хмарою для подальшого аналізу.

Система має підтримувати модульний підхід до додавання нових типів пристроїв. Коли в мережі з'являється новий пристрій, його інтеграція не повинна вимагати повної реконфігурації мережі. Достатньо буде встановити пристрій, він автоматично підключиться до найближчої локальної мережі, а пристрій оператора (контролер) додасть їх унікальні ідентифікатори до списку пристроїв мережі і відправить оновлену інформацію на центральний сервер.

Оскільки система розрахована на автономну роботу, важливо передбачити альтернативні способи комунікації в разі виводу з ладу системи радіо. До кожного пристрою має бути можливість підключитись через кабель до необхідної реконфігурації, але з підтримкою тих же механізмів безпеки. Теоретично, деякі пристрої мережі можуть бути підключені до неї повністю дротово. При цьому один пристрій буде підключено через провід до іншого. Не зважаючи на це, такий пристрій буде відображатися в системі як повноцінний член мережі, але втратить зв’язок з іншими пристроями, якщо кабель буде перебито, або радіоприймач пристрою, до якого він підключений вийде з ладу.

Також треба забезпечити пристрій сценаріями дій у випадку від’єднання від локальної мережі (повної автономної роботи), а також в разі розрядження акумулятора, або фатального збою в системі. Девайсу потрібно записати що сталося у щоденник подій, зробити останні дії та коректно завершити свою роботу. Порядок дій при такому сценарії визначається відповідно до моделі причинно-наслідкового зв'язку між внутрішніми станами пристрою, для побудови використання цих залежностей для формування послідовності дій. [5]

Система хоч, в першу чергу і розрахована на автономні пристрої, але в якості додаткової функціональності, має підтримувати варіант дистанційного

керування пристроями з серверу, через стаціонарний операторський девайс (gateway), розрахований на стабільне підключення до інтернету. Така мережа буде автоматично переходити на стандартний автономний режим (або вимикатися) в разі відсутності зв’язку з сервером.

Також важливо відстежувати коректні ланцюжки подій, навіть при автономній роботі, для цього можливо було б зберігати логи локально, а потім зіставляти з найменшим впливом суперечливих знань на пояснення в системі рекомендацій для підтвердження дійсності подій [6]. Це важливо зробити для уникнення можливості відправки хибної інформації на сервер компроментованими девайсами або контролерами, навіть якщо за замовченням ми довіряємо кожному оператору, важливо розраховувати на те, що нашою довірою закортять скористатися.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ

3.1 UML проектування системи

Розглянемо можливі прецеденти системи (рис. А.1). Ми маємо 3 основних актори: член організації, оператор контролеру та адміністратор організації.

Оператор контролеру напряму відповідає за контролер – девайс, через який він може керувати автономними пристроями. Для того щоб це робити контролер потрібно авторизувати в системі, а потім отримати токен на керування певними пристроями. Оператор може продивитися усі девайси, що належать організації в виді списку або мапи, а також їх статус та належність то локальних мереж. Оператору надається обмежений доступ за виділеними пристроями на певний час, що залежить від токену, який йому було видано.

Член організації керує системою через веб застосунок. Після авторизації в системі, йому буде надано доступ до панелі керування організацією, її пристроями, а також усіма іншими об’єктами, що відносяться до цієї організації. Головною задачею члена організації є керування запитами, куди входять запити на авторизацію та на генерацію токенів. Член організації також може передивитися інших членів організації, переглянути історію виданих токенів, а також переглянути автономні пристрої організації в форматі списку або на мапі.

Адміністратор організації – людина, яка створила організацію в системі і відповідає за керування її членами – має всі повноваження звичайного члену організації, але також може керувати усіма іншими членами організації (запросити, вигнати, змінити роль або посаду), а також налаштуваннями організації в системі (її назву, опис, правила, тощо).

За структурою кожна організація є ієрархічною структурою з адміністратором на чолі та її членами, що керують розподіленням тимчасових повноважень операторам. Усі оператори та члени організації за стандартом є рівними, але їх особисті повноваження можуть відрізнятися в залежності від назначеної їм ролі.

3.2 Проектування архітектури системи

Система призначена бути максимально динамічною, з декількома типами кінцевих девайсів, до яких буде надаватися доступ через девайс-посередник. Єдиною постійною, а також центральною частиною системи є основний сервер, на якому будуть зберігатися записи подій, буде прямий доступ до бази даних усіх зареєстрованих девайсів, а також будуть генеруватися тимчасові токени для керування автономними пристроями.

Особливо ускладнює нашу задачу той факт, що ми маємо розраховувати на те, що користувачі можуть встановити девайси на особливо віддаленій ділянці. Для нас це означає те, не тільки кінцеві девайси не матимуть прямого доступу до інтернету, але й девайс оператора може мати проблеми з постійним зв’язком з сервером. Структура нашої системи (рис. А.2) буде складатися з головного сервера, в якого буде доступ до бази даних. На головний сервер будуть приходити запити на отримання тимчасового доступу до керування пристроями через REST API.

Головний сервер буде хостити Web-застосунок, через який адміністратори системи (члени організацій) зможуть схвалювати запити на доступ, а також дивитися за діями локальних операторів та слідкувати за станом системи, з підтримкою оновлень у реальному часі тих для пристроїв, що мають доступ до інтернету.

Керування пристроями, як і зв’язок пристроїв між собою, буде відбуватися через радіо сигнал, в якості прикладу низько енергетичної мережі широкого радіусу, ми можемо використати набір протоколів LoRa, але потенційно його прийдеться модифікувати для забезпечення рівня захищеності, на який ми розраховуємо, з використання асиметричного шифрування, системи автономної авторизації, тощо.

Пристрої можуть спілкуватися між собою по зашифрованому радіо-каналу, тим самим формуючи між собою локальну автономну мережу. Всі пристрої, що

складають локальну мережу ієрархічно рівні, окрім тих моментів, коли до мережі підключається контролер, що фактично робить топологію мережі зіркою.

Створення локальної мережі є важливим, бо це надає можливості оператору звертатися до будь-якого пристрою мережі, навіть якщо в зоні ефективної досяжності контролеру присутній лише один девайс. Контролер, який підключається до цієї мережі, виступає посередником між сервером і локальними пристроями.

Архітектура системи поєднує централізоване управління через головний сервер з децентралізованою автономною роботою локальних пристроїв, що робить її надійною та ефективною навіть у складних умовах.

3.3 Проектування структури зберігання даних

Логічна структура нашої системи повинна віддзеркалювати її фізичну структуру. Кожна з її частин має мати відповідний об’єкт в базі даних.

В базі даних (рис. А.3) усіх користувачів в системі розподілено на організації. Кожна організація має свій унікальний ідентифікатор та назву.

Кожна організація може мати багато членів, кожен з об’єктів яких підв’язано до користувача за допомогою адреси електронної пошти авторизованих користувачів. Кожен член організації має різні повноваження в залежності від призначеної йому ролі, але загалом розраховується, що кожен член організації, якого зареєстровано в системі має право доступу до записів подій, а також генерації токенів доступу. Повні права по управлінню організації, а також додаванню та видаленню її членів має лише засновник організації, тобто користувач, що створив організацію у системі.

Кожній організації належать певні девайси, які заносяться в базу даних при встановленні, а також контролери – пристрої, що будуть застосовувати локальні оператори за керуванням автономними девайсами. Контролери мають бути авторизовані членами організації до того як запитувати доступ на керування і репрезентують собою авторизовані сесії операторів при використанні контролеру,

яких може бути декілька по відношенню до кожного окремого фізичного девайсу контролеру.

Об’єкт збереження девайсів зберігає в собі велику кількість інформації про девайс: хто та коли його встановив, який контролер при цьому було використано і ким було надано повноваження на це інформація та доступний функціонал девайсу, координати за якими його було встановлено, його унікальний ідентифікатор, а також унікальний ідентифікатор локальної мережі, що створено при встановленні декількох пристроїв поруч.

Також в базі даних потрібно зберігати мінімальну інформацію про локальні мережі, що об’єднують між собою пристрої. Кожна мережа має унікальний ідентифікатор, назву для полегшення взаємодії між членами організації, а також повинна зберігати інформацію для генерації токену, що надаватиме доступ до пристроїв, що належать їй. Для цього в кожному об’єкті мережі зберігатиметься приватний ключ для шифрування токенів.

Все це має зберігатися в системі та буде застосовано при генерації токену для надання доступу виключно певних девайсів. Також дуже важливо зберігати точний час встановлення окремих девайсів в системі. Час на сервері, а також час на лічильнику девайсу має бути синхронізований для проведення правильної автономної авторизації. Навіть, якщо при встановлені лічильники часу було розсинхровано, теоретично, при генерації токену можна ввести поправку на різницю часу на лічильниках.

Кожний об’єкт контролеру відповідає авторизованій сесії контролера при прив’язці його до авторизації, а не окремому фізичному пристрою. Так, один фізичний контролер може відповідати одразу за декілька об’єктів в базі даних. Це дозволяє краще відстежувати дії операторів, а в разі втрати контролеру, чи зміни власності контролеру дозволяє авторизувати його знов, залишивши минулий об’єкт в якості посилання для записів дій.

3.4 Шифрування та генерація токена

Безпека програмної системи основана на принципі генерації одноразових токенів. Цей токен дозволяє керувати лише певним набором пристроїв і лише протягом встановленого періоду часу. Після закінчення цього терміну доступ автоматично блокується, а пристрій не відповідає на команди оператора.

Шифрування виконується за алгоритмом RSA. Для кожної окремої групи девайсів згенеровано унікальну пару ключів для шифрування та дешифрування. Ключі генеруємо на сервері за допомогою вбудованого модулю crypto. (див. В.1)

Після генерації пари ключів, приватний ключ записується в базу даних, до відповідної мережі, де він буде зберігатися в текстовому вигляді, конвертуючи його з бінарного вигляду у форматі «pem» типу «pkcs1». А публічний ключ, в свою чергу в тому ж форматі, буде надіслано до контролеру, що записує його на девайс при його установці.

Відмінністю нашого алгоритму від типового алгоритму RSA є те, що ми шифруємо наш токен за допомогою приватного ключа, а розшифрувати його будемо за допомогою публічного. Таким чином алгоритм шифрування працює більше як алгоритм електронного підпису. Розшифрування токена буде саме по собі служити підтвердженням його дійсності.

Токен являє собою невеликий JSON документ в якому збережено усю інформацію про токен, включаючи повноваження оператора та термін його дії. Після генерації цього токену, інформацію про його генерацію записано в базу даних, після чого він проходить на етап шифрування.

Для маніпуляції токенами на сервері ми користуватимось наступними функціями для шифрування за допомогою приватного ключа та дешифрування за допомогою публічного. (див. В.2 та В.3)

Звісно, ми не хочемо щоб наш токен перехопили та використовували в поганих намірах, тому його зашифровано і без відповідного публічного ключа неможливо дізнатися жодну інформацію про нього, а оскільки токени

персональні, його неможливо буде використати за допомогою жодного іншого пристрою.

Сам токен в системі не зберігається, це б була дуже вразливою практикою, навіть якщо час роботи токену вже збіг. Але в системі буде зберігатися інформація для його генерації, а головне ким, для кого та для керування якими об’єктами він надавав доступ. Кожен токен матиме свій унікальний ідентифікатор, його присутність зберігається для структурування бази даних, а також для відстеження дій, які було зроблено за допомогою нього.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Опис серверної частини

Серверну частину було розроблено на основі Next.js 15, потужного та гнучкого фулстак фреймворку для Node.js, індустріального стандарту для веб розробки, яким користуються найбільші світові корпорації. [7]

Бекенд частина складається з набору REST API кінцівок (див табл. Б.1), що підключено до бази даних, або систем авторизації чи генерації токенів. Самі кінцівки розраховані на підключення контролерами для отримання обмеженої інформації, а також відправки оновленої інформації про стан пристроїв на сервер.

Деякі дії, наприклад авторизація контролерів, або генерація токенів не може бути виконана автоматично, через розрахунки безпеки. Через це усі подібні дії, які надсилаються на сервер, зберігаються в вигляді запитів (об’єкт бази даних Request), що очікують обробки (статус pending) членами організації через веб-застосунок. Після надсилання запиту на сервер, в якості відповіді буде відправлено id доданого запиту, яким може скористатися контролер, або користувач під час очікування його обробки для перевірки статусу. Після обробки запиту можна буде скористатися його id для отримання обробленої інформації.

Після узгодження, або відмови у виконанні запиту, його статус буде змінено на відповідний, після чого запит буде оброблено відповідною системою на сервері, в залежності від його типу.

Уся інформація зберігається у реляційній та масштабованій базі даних PostgreSQL, яку запущено на тому ж комп’ютері. При необхідності масштабувати систему БД може зберігатися на окремому сервері та підключатися віддалено. За рахунок структури бази даних, окремі організації та всі належні їм компоненти можуть бути розгорнені на окремих серверах, що надає більшу гнучкість за потреби.

Для зв’язку та управління з базою даних використовується Drizzle ORM – невелика бібліотека, розроблена для TypeScript, для спрощення взаємодії з базою

даних. Вона підтримує велику кількість баз даних, надаючи узагальнений SQL-подібний інтерфейс для взаємодії з збереженими даними, що надає можливість замінити базу даних після розробки. Drizzle ORM є open source проектом, заснованим українськими розробниками, який зараз набирає популярність серед фулстак та бекенд веб-інженерів. [8]

Керування даними, створення та керування таблицями може повністю відбуватися без взаємодії з SQL, для цього використовуються комбінації функцій та об’єктів JavaScript, при використанні ми можемо імпортувати лише необхідні нам об’єкти бази даних та функції ORM у необхідний нам файл. (див. В.4)

Не дивлячись на те, що ми користуємось фулстак фреймворком, що означає, що клієнтські та серверні файли знаходяться в одній директорії, а компоненти тисло пов’язані один з одним, користуватися підключенням з базою даних на пряму, звісно, є можливим лише на серверній частині застосунку. Отримання даних з БД на клієнті можна лише через відкриті API, або через спеціалізовані Next.js actions.

Система авторизації була розроблена на основі бібліотеки Auth.js, що надає можливість підключити готове рішення для авторизації у нашому веб-додатку. Auth.js також підтримує велику кількість OAuth 2.0 провайдерів (такі як GitHub, Google, Meta, Discord), що надає можливість авторизуватися в застосунку без реєстрації через електронну пошту. В нашому застосунку ми підтримуємо можливість входу через аккаунт Google, GitHub, бо наша система розраховує на участь розробників у кожній організації, а також реєстрацію через пошту та пароль. При вході через один з провайдерів, ми отримуємо доступ до унікального ідентифікатора входу, імені, аватару (не для всіх провайдерів), а також адреси електронної пошти, якою ми користуватимось для ідентифікації користувача в системі.

Система реєстрації працює на основі технології JavaScript Web Token та дає користувачам до захищеної сесії, яку ми можемо підтвердити в коді як на сервері так і на клієнті.

4.2 Опис клієнтської частини

Оскільки в якості фремворку для розробки нашого Web-застосунку було обрано Next.js, для розробки клієнтської частини використався React.js – найпопулярніший фрейворк для розробки динамічних веб-інтерфейсів за рахунок використання компонентів та технології віртуального DOM (Document Object Model). [9]

При розробці інтерфейсу було використано відносне нововведення до у React, а саме серверні компоненти. Раніше реакт був повністю клієнтською бібліотекою, генеруючи інтерфейс на ходу. Цей підхід надавав можливість робити надзвичайно динамічні інтерфейси, але нажаль, швидкість застосунку страждала, особливо на більш повільних комп’ютерах. Серверні компоненти надають можливість генерувати HTML-елементи заздалегідь на сервері, що значно збільшує працездатність сайту. Що найголовніше – Next.js дозволяє комбінувати серверні та клієнтські компоненти один з одним, що дозволяє створювати ефективну розгортку на сервері та доповняти її динамічними клієнтськими компонентами без жодних компромісів інтерактивності.

Для створення інтерфейсів було використано Material UI – це бібліотека компонентів React з відкритим кодом, яка реалізує Material Design від Google. [10] Material Design є ефективною та елегантною системою розробки цифрових інтерфейсів, якими керуються такі корпорації як Google для розробки свої застосунків (рис. А.4). Використання бібліотеки UI компонентів надасть нам можливість побудувати красивий та сучасний користуватський інтерфейс з універсальних елементів, які вже знайомі користувачам. Зручний та доступний інтерфейс надасть нам можливість зосередити увагу на впровадженні функціоналу та надійності системи.

Код будування вітальної сторінки, в виді прикладу розробки, як серверного компоненту React.js з використанням комбінації готових компонентів Material UI, кастомних компонентів Next.js, а також звичайних елементів HTML. (див. В.5)

Не зважаючи на те, що Material UI використовує свою власну систему кастомізації компонентів, для розробки власних елементів інтерфейсу, ми використовуємо Tailwind – новітній спосіб стилізувати HTML-елементи. Tailwind дає нам можливість стилізувати компоненти за рахунок власної системи CSS класів, що надають нам можливість прикрашати наш сайт комбінуючи класи без жодної взаємодії з каскадними листами стилів. Не зважаючи на те, що в Tailwind класів присутня величезна кількість, навіть не рахуючи їх комбінації, а також власні класи, їх кількість не є проблемою для кінцевого користувача бо в процесі компіляції проекту PostCSS вкладає в пакет виключно використані нами класи.

4.3 Безпека системи

Для входу до керувальної панелі організації потрібна обов’язкова авторизація. Для авторизація можна використати аккаунт з підтримкою системи авторизації OAuth 2.0, це можна зробити одразу на вітальній сторінці веб-застосунку. На даний можливе використання входу в систему через акаунт Google, або GitHub (рис. А.5).

Система авторизації була імлементована з використанням бібліотеки Auth.js, що розрахована на створення авторизації для будь-якого веб-фреймворку на основі Node.js з підтримкою сучасних технологій авторизації, таких як різноманітних провайдерів OAuth 2.0, а також авторизації через пошту.

В Auth.js сесія може зберігатися або через кукі, або у вигляді JWT, що дозволяє балансувати між безпекою та продуктивністю. Користувачі можуть входити до системи за допомогою різних облікових записів, і все це гарно інтегрується з Node.js.

Вона створена як універсальне рішення і чудово інтегрується з фреймворками, такими як Next.js, забезпечуючи як серверну, так і клієнтську частини аутентифікації. За кулісами бібліотека автоматично обробляє такі завдання, як створення сесій, керування токенами, обробка редіректів після

авторизації чи виходу з системи, а також захист маршрутів до сторінок чи API, не дозволяючи доступ не авторизованим в системі користувачам.

В нашій системі авторизації використовуються JWT – токени, що призначені для передачі підписаних заяв між службами (як зовнішніми, так і внутрішніми для вашого застосунку). Заява, в цьому контексті – це частина інформації, яку інші можуть переглядати та перевіряти, але не змінювати, [11] в даному випадку для перевірки авторизації користувачів.

Для доступу користувача до організації також потрібно подати запит до відповідної організації, який моє бути схвалений її засновником. Альтернативно засновник авторизації власноруч може запросити користувача у організацію, надіславши листа на імейл користувача, якого буде автоматично додано до організації при вході в систему.

5 ТЕСТУВАННЯ СИСТЕМИ ЧЕРЕЗ СИМУЛЯЦІЮ

Наша система є дуже комплексною, з декількома типами пристроїв з різними типами взаємодії. Також, наша система повинна мати можливість підтримувати велику кількість кінцевих клієнтів (автономних пристроїв), а це дуже складно та просто не раціонально відтворювати в реальному світі.

Для проведення повноцінного функціонального тестування було розроблено 3D симуляцію керування автономними приладами в реальному часі в середовищі розробки комп’ютерних ігор Godot (рис. А.6).

У симуляції головними елементами є різні види взаємодії з системою через контролер, а саме: авторизація контролеру в системі, запит токена, а також базовий функціонал керування автономними пристроями. Взаємодія з контролером відбувається через панелі, що з’являються на екрані по натисненню відповідних клавіш.

Також важливою частиною є симуляція системи зв’язку пристроїв між собою, те як пристрої в’яжуться між собою у локальну мережу і передають повідомлення між собою, що повинно відображати поведінку пристроїв на практиці.

При запуску, симуляція надсилає запит на сервер, звідки бере інформацію про девайси певної організації через API і ініціалізує усі девайси у вигляді об’єктів та візуалізує взаємодію пристроїв між собою, яку інакше не можливо б було помітити.

Така симуляція служить не тільки способом тестування системи в реальному часі та способом функціонального аналізу можливих сценаріїв взаємодії пристроїв між собою, а також може послужити наглядною демонстрацією діяльності системи і способом візуалізації її окремих внутрішніх процесів.

Керування девайсами в симуляції виконується через набір контрольних панелей (рис. А.7), що повинні репрезентувати собою інтерфейс контролеру в реальному житті.

Присутні 3 основні панелі: панель авторизації контролеру, панель запиту токенів, а також панель керування пристроями. У оператора нема можливості отримати доступ до панелі не отримавши відповідь на запит з сервера попередньої панелі, тобто спочатку оператору потрібно авторизуватись в системі, потім отримати тимчасовий токен, а вже після цього з'являється можливість керувати окремими пристроями.

Спочатку, оператору треба відкрити панель авторизації, де йому потрібно ввести ID організації, а також ім'я під яким авторизований девайс можна буде побачити у системі. Після натискання на кнопку «Send request», програма надсилає на сервер запит на авторизацію контролеру, який має бути узгоджений одним з членів організації через веб-додаток. До тих пір програма перевіряє статус запиту через API. Після того як запит було узгоджено, оновлений статус авторизації відображається на тій же панелі.

На панелі запиту токена оператор може зробити запит на генерацію токена для отримання доступу до керування девайсами. Тут оператор може обрати права на керування, які йому потрібні, а потім натиснути на кнопку «Request token». Після цього програма надсилає запит на сервер. При надсиланні, користувачу не потрібно вводити ідентифікатор локальної мережі, бо він визначається програмою автоматично. Через обмежені рамки симуляції, на виділеній території може бути лише одна локальна мережа автономних девайсів. Після того як запит на токен було узгоджено, у оператора з'являться доступ до панелі керування пристроями.

На панелі керування пристроями, оператор може обрати окремий пристрій, а через додаткове вікно проводити з ним дії. Оператор може або виконати на пристроєм базові дії (вмикання, вимикання), або ввести команду на дію, яка є специфічною для цього пристрою. Також через панель керування пристроями можна ввімкнути виконання заздалегідь програмованої процедури для певного типу девайсів, наприклад автоматичний полив, або автоматичне вмикання ліхтарів при наближенні оператора.

При розробці сервер було запущено в режимі тестування, в якому замість бази даних було використано SQLite для зменшеного навантаження, а для API

було вимкнено авторизацію. При виробництві в загальному доступі система в жодному разі не повинна бути запущена у тестовому режимі. Інформація збережена в основній базі даних хоч і не буде модифікована, але може бути отримана зацікавленою стороною.

Розробка симуляції відбувалася на скриптовій мові програмування GDScript, спеціально розробленою для середовища Godot та за своєю структурою нагадує Python. При взаємодії з віртуальним контролером за потреби він буде звертатися до серверу через внутрішній компонент HTTPrequest та відповідні API кінцівки, що було виділено.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було розроблено програмну систему керування автономними пристроями.

Було виконано аналіз можливих готових рішень та альтернатив для майбутньої системи. Це дозволило нам виділити їх слабкі місця, що ми використаємо як перевагу над нашими майбутніми конкурентами, такі як: постійне підключення до інтернету, захищеність передачі даних, тимчасові права доступу, захищена делегація повноважень, тощо. Також було проведено аналіз сучасних протоколів локального обміну інформації через радіо і використання надійних криптографічних графік для генерації захищених токенів.

В результаті розробки, було розроблено бекенд частину системи на основі Next.js, що включає собою реляційну базу даних PostgreSQL, надійних систему REST API, а також систему авторизації Auth.js з підтримкою OAuth 2.0.

Було розроблено сучасний та динамічний Web-інтерфейс на основі бібліотеки React.js керуючись системою принципів розробки Material Design від Google для створення зручного та доступного користувацького інтерфейсу.

Для демонстрації можливостей системи в реальному світі було розроблено 3D симуляцію керування автономними приладами за допомогою віртуального контролеру в середі розробки комп’ютерних ігор Godot.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Inside the Smart Home: IoT Device Threats and Attack Scenarios | Trend Micro (US). URL: https://www.trendmicro.com/vinfo/us/security/news/ internet-of-things/inside-the-smart-home-iot-device-threats-and-attack-scenarios (дата звернення 13.05.2025).

2. Google IoT Core is shutting down - akenza | Self-Service IoT Platform to Build Smart Solutions. URL: https://akenza.io/blog/google-iot-core-shutdown (дата звернення: 12.06.2025).

3. Low-Power Wide-Area Networks: Opportunities, Challenges, Risks and Threats / ed. by Ismail Butun, Ian F. Akyildiz. Springer Cham, 2023. 212 p.

4. Security & Signing | MySensors - Create your own Connected Home Experience. URL: https://www.mysensors.org/about/signing (дата звернення: 13.05.2025).

5. Chalyi S., Leshchynskyi V. TEMPORAL-ORIENTED MODEL OF CAUSAL RELATIONSHIP FOR CONSTRUCTING EXPLANATIONS FOR DECISION-MAKING PROCESS. Advanced Information Systems. 2022. Т. 6, № 3. С. 60–65. URL: https://doi.org/10.20998/2522-9052.2022.3.09 (дата звернення: 12.06.2025).

6. Chalyi S., Leshchynskyi V., Leshchynska I. Detailing explanations in the recommender system based on matching temporal knowledge. Eastern-European Journal of Enterprise Technologies. 2020. Т. 4, № 2 (106). С. 6–13. URL: https://doi.org/10.15587/1729-4061.2020.210013 (дата звернення: 12.06.2025).

7. Next.js by Vercel - The React Framework. URL: https://nextjs.org/ (дата звернення: 31.05.2025).

8. Drizzle ORM - Why Drizzle?. URL: https://orm.drizzle.team/docs/overview (дата звернення: 31.05.2025).

9. Що таке React JS? - Run-It. URL: https://www.run-it.com.ua/shcho-take-react-js/ (дата звернення: 31.05.2025).

10. Overview - Material UI. URL: https://mui.com/material-ui/getting-started/ (дата звернення: 31.05.2025).

11. Як використовувати JSON Web Tokens (JWT) для автентифікації | DevZone. URL: Як використовувати JSON Web Tokens (JWT) для автентифікації | DevZone (дата звернення: 31.05.2025).

12. GitHub NureSynelnikovOleksandr/2025_B_PI_PZPI-21-8_Synelnikov_O_O. URL: https://github.com/NureSynelnikovOleksandr/2025_B_PI_PZPI-21-8_Synelnikov_O_O (дата звернення: 12.06.2025).