Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: *Комп'ютерна система керування автобусними пасажирськими перевезеннями*

Виконав: студент 4 курсу, групи СІ-42
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

(підпис) *Микитюк А. І.* (прізвище та ініціали)

Керівник (підпис) *Стадник Н. Б.* (прізвище та ініціали)

Нормоконтроль (підпис) *Тиш Є. В.* (прізвище та ініціали)

Завідувач кафедри (підпис) Осухівська Г.М. (прізвище та ініціали)

Рецензент (підпис) *Гром'як Р.С.* (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *комп'ютерних систем та мереж*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
«25» квітня 2026 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студента *Микитюка Андрія Івановича*
(прізвище, ім'я, по батькові)

1. Тема роботи *Комп'ютерна система керування автобусними пасажирськими перевезеннями*

Керівник роботи _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від *« 24 » квітня 2026 року № 4/9-188* .
2. Термін подання студентом завершеної роботи *17.06.2026 р.*
3. Вихідні дані до роботи *Технічне завдання, архітектурні вимоги до клієнт-серверних систем, технічні специфікації мікроконтролера ESP32 та протоколи бездротового обміну* даними HTTP/JSON.
4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз технічного завдання
2. Проєктна частина
3. Практична частина
4. Безпека життєдіяльності, основи охорони праці.
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Блок-схема алгоритму роботи системи.
2. Структурна схема..
3. Схема електрична принципова.
4. Результат роботи у вигляді веб-інтерфейсу.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	*доц. каф. МТ Сенчишин В. С.*		

7. Дата видачі завдання *24.04.2026 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*26.01 – 02.02*	
2.	*Робота над першим розділом «Аналіз технічного завдання»*	*03.02 – 15.02*	
3.	*Робота над другим розділом «Проєктна частина»*	*20.04 – 25.04*	
4.	*Робота над третім розділом «Практична частина»*	*26.04 – 05.05*	
5.	*Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»*	*07.05 – 25.05*	
6.	*Оформлення пояснювальної записки і графічного матеріалу*	*26.05 – 7.06*	
7.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*	
8.	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*	
9.	*Захист кваліфікаційної роботи бакалавра*	*24.06*	

Студент _____ *Микитюк А. І.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Стадник Н. Б.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Микитюк А. І. Комп’ютерна система керування автобусними пасажирськими перевезеннями : робота на здобуття ступеня бакалавра : спец. 123 — комп’ютерна інженерія / канд.тех.наук. Н. Б. Стадник. — Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2026. — 75 с.

Ключові слова: комп’ютерна система, пасажирські перевезення, Інтернет речей, веб-інтерфейс, мікроконтролер, клієнт-серверна архітектура, синхронізація в реальному часі.

Кваліфікаційна робота присвячена проєктуванню та розробці трирівневої комп'ютерної системи для автоматизації обліку вільних і зайнятих місць в автобусах та координації пасажирських рейсів у реальному часі. В межах роботи реалізовано апаратний IoT-модуль на базі мікроконтролера ESP32, який збирає дані про пасажиропотік і поточні GPS-координати. Розроблено бекенд-сервер на базі Node.js (Express), що забезпечує обробку телеметрії та вирішення конфліктів синхронізації при одночасному оновленні даних від датчиків та ручних діях операторів. Створено адаптивний веб-інтерфейс, який надає пасажирам доступ до розкладу й інтерактивної карти салону, а персоналу автостанцій - інструменти оперативної модифікації стану місць у разі виникнення нештатних ситуацій.

ANNOTATION

Mykytiuk A. Computer System for Management of Bus Passenger Transportation: Bachelor thesis „123 - Computer Engineering“ Stabnyk Nataliia - Ternopil, TNTU, 2026 – 75 p.

Keywords: computer system, passenger transportation, Internet of Things, web-interface, microcontroller, client-server architecture, real-time synchronization.

The thesis is devoted to the design and development of a three-tier computer system for automating the allocation of free and occupied seats in buses and coordinating passenger trips in real time. Within the scope of the project, a hardware IoT-node based on the ESP32 microcontroller was implemented to collect passenger flow data and current GPS coordinates. A backend server based on Node.js (Express) was developed to handle telemetry data and resolve synchronization conflicts caused by concurrent updates from hardware sensors and manual staff overrides. An adaptive web interface was created, providing passengers with access to timetables and an interactive seat layout, and enabling bus station personnel to adjust seat availability efficiently in emergency situations.

ЗМІСТ

Додаток А Технічне завдання

Додаток Б Лістинги програмного коду комп'ютерної системи

СПИСОК СКОРОЧЕНЬ

API – Application Programming

CPU – Central Processing Unit

CORS – Cross-Origin Resource Sharing

GND – Ground

GPS – Global Positioning System

HTTP – HyperText Transfer Protocol

IoT – Internet of Things

JSON – JavaScript Object Notation

SaaS – Software as a Service

SSL/TLS – Secure Sockets Layer / Transport Layer Security

URL – Uniform Resource Locator

VLAN – Virtual Local Area Network

WLAN – Wireless Local Area Network

ВСТУП

Сучасний етап розвитку транспортної інфраструктури та пасажирських перевезень характеризується стрімким зростанням пасажиропотоку, ускладненням логістичних маршрутів та підвищенням вимог до якості, безпеки й оперативності надання послуг. Ефективне функціонування автотранспортних підприємств та автостанцій безпосередньо залежить від рівня автоматизації процесів управління. Традиційні підходи до обліку пасажирів та моніторингу рейсів, що базуються на паперовому документообігу або застарілих локальних базах даних, вже не задовольняють вимогам динамічного ринку. Вони призводять до виникнення затримок у синхронізації даних, помилок через людський фактор та відсутності актуальної інформації про наявність вільних місць у салоні під час виконання рейсу. Впровадження концепцій Інтернету речей та розподілених клієнт-серверних архітектур відкриває нові можливості для побудови систем керування пасажирськими перевезеннями в реальному часі, дозволяючи інтегрувати периферійні датчики, модулі геолокації та хмарні серверні платформи у єдиний інформаційний простір.

Об’єктом дослідження виступає процес інформаційного супроводу, автоматизованого обліку завантаженості та диспетчеризації регулярних автобусних пасажирських перевезень. Предметом дослідження є методи, апаратно-програмні засоби, мережеві протоколи взаємодії та алгоритми синхронізації даних у клієнт-серверній комп'ютерній системі моніторингу рейсів у реальному часі. Основна мета роботи полягає у підвищенні оперативності, надійності та прозорості управління автобусними пасажирськими перевезеннями шляхом проєктування і реалізації трирівневої комп'ютерної системи з автоматизованим обліком стану місць та віддаленим веб-доступом для користувачів і персоналу.

Для досягнення поставленої мети в рамках кваліфікаційної роботи було вирішено комплекс взаємопов'язаних завдань. Спочатку було проведено

детальний аналіз існуючих рішень у сфері автоматизації пасажирських перевезень та обґрунтовано вибір структурної трирівневої архітектури. Далі було розроблено структурну схему комп'ютерної системи та спроєктовано схему підключення датчиків пасажиропотоку до бортового обчислювального модуля. Важливим етапом стала розробка унікального алгоритму усунення конфліктів при синхронізації телеметричних і диспетчерських даних. На етапі практичної реалізації було написано мікропрограму збору телеметрії, серверний додаток на базі середовища Node.js та адаптивний веб-інтерфейс, після чого виконано комплексне тестування розробленого апаратно-програмного комплексу.

РОЗДІЛ 1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Огляд сучасних засобів та комп'ютеризованих систем керування пасажирськими перевезеннями

Процес цифровізації та автоматизації пасажирських перевезень в останні роки розвивається у напрямку створення інтегрованих систем Intelligent Transportation Systems (ITS). Сучасні системи управління автопарком (Fleet Management Systems) вирішують широкий спектр завдань: від простого супутникового моніторингу місцезнаходження транспорту до інтелектуального прогнозування часу прибуття та автоматизованого розрахунку рентабельності рейсів.

На сьогодні на ринку представлено кілька комерційних рішень, що використовуються великими автоперевізниками:

- Глобальні GDS-системи (на кшталт Amadeus, Sabre, BusBud). Вони орієнтовані на агрегацію розкладів та продаж квитків у масштабах багатьох країн. Перевагою є охоплення величезної аудиторії, проте недоліком є повна відсутність зв'язку з безпосереднім апаратним станом конкретного автобуса на маршруті. Дані про вільні місця в таких системах змінюються лише в момент купівлі квитка в касі, що унеможливлює облік пасажирів, які зайшли на проміжних зупинках.

- Телематичні платформи моніторингу (наприклад, Wialon, GPS-Tracks). Дані комп'ютерні системи базуються на встановленні бортових GPS/GPRS трекерів. Вони забезпечують диспетчера точними координатами, інформацією про витрату палива та швидкісний режим. Проте ці системи є закритими інженерними платформами, які не мають зручного інтегрованого веб-

інтерфейсу для звичайних пасажирів, що бажають перевірити наявність вільних сидінь у реальному часі.

- Локальні ERP-системи автостанцій. Забезпечують внутрішній облік рейсів та касові операції. Основним недоліком є використання застарілих архітектурних рішень із низькою частотою оновлення даних та відсутністю гнучких модулів для оперативної зміни конфігурації салону (наприклад, швидкого блокування місць касиром через виявлене пошкодження крісла безпосередньо перед посадкою).

Таким чином, існує чітко виражена потреба у розробці спеціалізованої комп'ютерної системи, яка б поєднувала функції апаратного збору телеметрії з транспортного засобу, централізовану обробку даних на сервері та надання гнучкого веб-інтерфейсу з розподілом прав доступу.

1.2 Порівняльний аналіз архітектурних рішень комп'ютерних систем

При проєктуванні комп'ютеризованих систем керування розподіленими об'єктами критично важливим є вибір архітектури програмно-мережевого комплексу. У сучасній інженерії найчастіше розглядаються три варіанти побудови архітектури:

1) Монолітна архітектура (Monolithic Architecture). Усі компоненти системи (інтерфейс, бізнес-логіка, робота з даними) виконуються як єдиний програмний комплекс на одному сервері.

Переваги: Простота початкової розробки та розгортання на етапі прототипування.

Недоліки: Погана масштабованість. При зростанні кількості автобусів та одночасних запитів від пасажирів сервер швидко вичерпує обчислювальні ресурси. Будь-яка помилка в модулі відображення розкладу може призвести до падіння всієї системи, включаючи модуль приймання критично важливої телеметрії.

2) Дволанкова клієнт-серверна архітектура (Two-Tier Architecture). Передбачає безпосередню взаємодію товстого клієнта (додатка оператора) з сервером бази даних через мережеві драйвери.

Переваги: Висока швидкість виконання простих запитів у локальній мережі автостанції.

Недоліки: Непридатна для публічного веб-доступу пасажирів. Пряме підключення тисяч клієнтів до бази даних створює критичні вразливості в системі кібербезпеки та перевантажує СКБД. Окрім того, периферійні IoT-пристрої не мають достатньо ресурсів для підтримки важких постійних з'єднань із базою даних.

3) Триланкова клієнт-серверна архітектура (Three-Tier / Multi-Tier Architecture). Оптимальне інженерне рішення для розроблюваної системи. Вона розділяє комплекс на три ізольованих рівні:

- Рівень представлення (Client Tier / Front-end): Веб-браузери пасажирів та мобільні пристрої персоналу.

- Рівень бізнес-логіки (Application/Web Server Tier / Back-end): Центральний веб-сервер, який приймає HTTP-запити, валідує дані, керує авторизацією та виконує алгоритми обробки інформації.

- Рівень даних (Data Tier): Ізольована база даних, доступ до якої має виключно сервер бізнес-логіки.

Порівняльні характеристики архітектурних рішень наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз архітектурних рішень комп'ютерних систем

Критерій порівняння	Монолітна архітектура	Дволанкова клієнт-серверна	Триланкова клієнт-серверна (Обрана)
Масштабованість	Низька	Середня	Високий (ізольована БД)
Рівень безпеки даних	Середній	Низький (прямий доступ до БД)	Високий (ізольована БД)
Придатність для IoT-модулів	Погана	Критично низька	Ідеальна (через легкі REST API)
Стійкість до відмов	Низька	Середня	Висока
Складність оновлення компонентів	Висока	Середня	Низька (ізольовані шари)

Здійснений аналіз показує, що трирівнева клієнт-серверна архітектура є найбільш ефективною для реалізації комп'ютерної системи керування автобусними перевезеннями, оскільки забезпечує надійний захист інформації, легку інтеграцію периферійного IoT-обладнання та високу швидкість обслуговування кінцевих користувачів.

1.3 Обґрунтування вибору апаратного та програмного стеку технологій

При розробці комп'ютеризованих систем керування транспортними потоками та моніторингу пасажиропотоку в реальному часі критично важливим є вибір збалансованого стеку технологій. Апаратні та програмні засоби повинні забезпечувати високу швидкість обробки інформації, мінімальні мережеві затримки, енергоефективність периферійних вузлів та кросплатформність користувацьких інтерфейсів.

Обґрунтування апаратної платформи бортового модуля. Для реалізації периферійного IoT-вузла, що встановлюється на борту транспортного засобу та виконує функції збору телеметрії (координати, кількість пасажирів), було проведено порівняльний аналіз поширених мікроконтролерів та обчислювальних платформ. Традиційні архітектури на базі мікроконтролерів сімейства AVR (наприклад, Arduino Uno, Nano або Mega 2560) мають суттєві обмеження щодо тактової частоти (до 16 МГц) та обсягу оперативної пам'яті (від 2 до 8 Кб). Головним їхнім недоліком є відсутність вбудованих мережевих інтерфейсів, що потребує підключення зовнішніх громіздких шилдів (Ethernet,Wi-Fi або GSM), збільшує вартість пристрою та знижує надійність апаратної частини при вібраційних навантаженнях в автобусі.

Використання одноплатних мікрокомп'ютерів (наприклад, Raspberry Pi 4 Model B) забезпечує високу обчислювальну потужність завдяки 4-ядерним процесорам ARM Cortex-A72 та великому обсягу ОЗУ. Проте для виконання специфічних завдань опитування датчиків пасажиропотоку в режимі реального часу така платформа є надлишковою, має високе енергоспоживання, тривалий час завантаження операційної системи після подачі живлення та високу ринкову вартість.

Оптимальним інженерним рішенням для даної системи є вибір мікроконтролера ESP32 (зокрема, NodeMCU або ESP32 DevKit). Його техніко-економічні переваги включають:

- наявність двоядерного 32-бітного процесора Xtensa LX6 з тактовою частотою до 240 МГц, що дозволяє паралельно обробляти переривання від датчиків та підтримувати мережеву сесію;

- вбудовані модулі бездротового зв'язку Wi-Fi (802.11 b/g/n) та Bluetooth (v4.2 BR/EDR та BLE), що усуває потребу в додаткових апаратних компонентах для передачі даних;

- достатній обсяг оперативної пам'яті (520 Кб SRAM), необхідний для формування JSON-пакетів та підтримки стеків мережевих протоколів TCP/IP;

- низьке енергоспоживання та підтримку режимів глибокого сну (Deep Sleep), що важливо для бортових систем, які живляться від бортової мережі автомобіля або автономного акумулятора.

Обґрунтування вибору серверної платформи (Back-end). Центральний сервер системи повинен одночасно обробляти високочастотні телеметричні дані від багатьох транспортних засобів та забезпечувати миттєву віддачу інформації на запити користувачів і диспетчерів. Для реалізації бізнес-логіки було обрано програмну платформу Node.js із використанням мінімалістичного фреймворку Express.

Вибір Node.js обґрунтований такими архітектурними особливостями:

- Асинхронна подійна модель введення-виведення (Non-blocking I/O). На відміну від традиційних серверів (наприклад, Apache з PHP), які створюють окремий потік для кожного підключення, Node.js працює в одному потоці з використанням механізму Event Loop. Це дозволяє серверу обслуговувати десятки тисяч одночасних з'єднань від периферійних IoT-пристроїв із мінімальними витратами оперативної пам'яті.

- Висока продуктивність двигуна V8. Компіляція JavaScript у чистий машинний код забезпечує високу швидкість обробки JSON-пакетів телеметрії.

- Екосистема пакетів (NPM). Наявність готових перевірених бібліотек (таких як express для розробки REST API та cors для налаштування політик безпеки) значно прискорює процес розгортання та підвищує надійність коду.

Обґрунтування протоколу обміну даними та формату представлення інформації. Для забезпечення взаємодії між бортовим модулем ESP32 та сервером Node.js обрано прикладний протокол HTTP (метод POST), а як формат передачі даних - JSON (JavaScript Object Notation).

Вибір JSON обумовлений його текстовою структурою, яка є легкою для читання людиною (що спрощує налагодження системи) та має мінімальну надлишковість (оверхед) порівняно з форматом XML. Це дозволяє економити мобільний інтернет-трафік при передачі пакетів через 4G/LTE модеми в

автобусах. Текстовий рядок виду {"busId":"BUS-41-01","passengers":3,"lat":49.5535,"lng":25.5948} легко формується на стороні мікроконтролера за допомогою стандартних функцій форматування рядків і миттєво парситься на сервері Node.js, оскільки є рідним об'єктом для мови JavaScript.

Обґрунтування технологій рівня представлення (Front-end). Веб-інтерфейс системи розроблено з використанням базового стеку веб-технологій: HTML5, CSS3 та чистого JavaScript (Vanilla JS). Відмова від важких клієнтських фреймворків (React, Angular, Vue) на етапі створення прототипу дозволила:

- забезпечити максимальну швидкість завантаження сторінки навіть на мобільних пристроях пасажирів в умовах нестабільного покриття мережі 3G/4G;

- реалізувати легку адаптивну верстку карти салону автобуса за допомогою CSS Grid, яка підлаштовується під будь-які розміри екранів;

- організувати періодичне асинхронне опитування сервера за допомогою технології Fetch API для динамічного оновлення лічильників пасажирів та координат без повного перезавантаження веб-сторінки користувача.

Здійснений аналіз та обґрунтування технологічного стеку показують, що комбінація апаратної платформи ESP32, серверного середовища Node.js та легких веб-технологій дозволяє створити високонадійну, масштабовану комп'ютерну систему керування пасажирськими перевезеннями з низькою вартістю розгортання.

РОЗДІЛ 2 ПРОЄКТНА ЧАСТИНА

2.1 Розробка структурної схеми комп'ютерної системи керування перевезеннями

Проєктування комп'ютерної системи керування автобусними пасажирськими перевезеннями базується на трирівневій клієнт-серверній архітектурі, яка дозволяє ізолювати апаратні засоби збору телеметрії від логіки обробки даних та інтерфейсів кінцевих користувачів. Такий підхід забезпечує гнучкість системи, її масштабованість та стійкість до відмов, що є критично важливим для транспортних систем реального часу.

Для відображення взаємозв'язків між основними апаратними, мережевими та програмними компонентами комплексу було розроблено структурну схему комп'ютерної системи (рисунок 2.1). Згідно з вимогами до оформлення конструкторської документації, цей графічний матеріал також підготовлено у вигляді окремого креслення під десятковим номером КС КРБ 123.187.00.00 С1.

Архітектурна топологія спроєктованої системи включає чотири основні рівні взаємодії:

1) Апаратний рівень (Бортовий IoT-вузол): Розгортається безпосередньо в рухомому складі (автобусі). Базовим обчислювальним елементом є двоядерний 32-бітний мікроконтролер ESP32. До цифрових портів введення-виведення загального призначення (GPIO) підключаються периферійні компоненти для підрахунку пасажиропотоку (апаратно реалізовано кнопками імітації входу/виходу пасажирів на пінах GPIO 12 та GPIO 14). Робота інтегрованого приймача геолокаційних сигналів GPS на етапі прототипування реалізована програмно шляхом математичної імітації зміщення координат у

прошивці мікроконтролера, що дозволило спростити апаратну частину. Основна функція цього рівня - первинний збір даних, фільтрація шумів (усунення брязкоту контактів) та пакування інформації у текстові пакети.

2) Мережевий транспортний рівень: Забезпечує бездротову передачу даних через стек протоколів TCP/IP. Бортовий мікроконтролер використовує вбудований радіомодуль Wi-Fi (або зовнішній 4G/LTE модем) для підключення до глобальної мережі Інтернет. Для зв'язку периферійного пристрою із локальним сервером розробника в умовах апаратно-програмної симуляції застосовується технологія зворотного перенаправлення портів (SSH-тунелювання через провайдерів Serveo або Pinggy). Це дозволяє транслювати вхідні HTTP-запити з публічної адреси в інтернеті на локальний порт 3000 робочої станції.

3) Серверний рівень (Бекенд): Центральна ланка системи, реалізована в програмному середовищі Node.js із використанням веб-фреймворку Express. Бекенд виконує роль координатора та розподільника інформаційних потоків. Він містить у собі:

- Ендпоінт приймання телеметрії (/api/telemetry): Обробляє вхідні POST-запити від автобусів, парсить JSON-рядки, оновлює координати та запускає алгоритм динамічного перерахунку сидінь.

- Ендпоінт синхронізації стану (/api/status): Віддає за запитом типу GET актуальну матрицю салону для відображення на пасажирському табло.

- Ендпоінт диспетчерського керування (/api/seats/toggle): Приймає команди від касирів чи операторів автостанцій для ручного перемикання прапорців службового блокування сидінь.

4) Рівень представлення (Клієнтський фронтенд): Кросплатформовий веб-додаток, побудований на базі стандартів HTML5, CSS3 та JavaScript. Фронтенд розділений на дві функціональні зони: публічну (для пасажирів, що переглядають розклад і мапу вільних місць без реєстрації) та адміністративну (панель оператора автостанції для коригування стану салону в разі поломок чи

службових бронювань).

Взаємозв'язок усіх елементів та логічна структура побудови комп'ютерної системи наведені на рисунку 2.1.

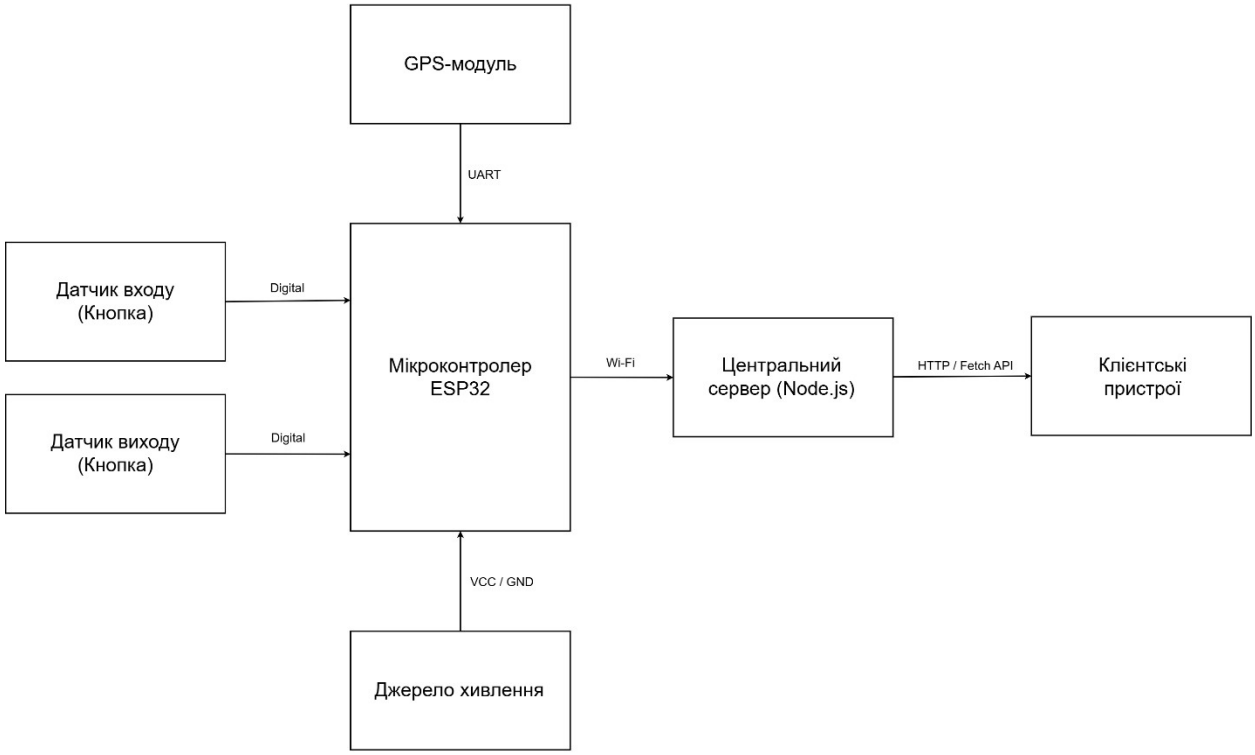

Рисунок 2.1 – Структурна схема комп'ютеризованої системи керування пасажирськими перевезеннями

Перевагою розробленої структурної схеми є повна децентралізація обчислень. Бортовий модуль не обтяжений логікою збереження чи аналізу загальної структури рейсів - він лише надсилає первинні метрики. Веб-інтерфейс, у свою чергу, не здійснює прямих запитів до пам'яті контролера, а взаємодіє виключно з уніфікованим програмним інтерфейсом (API) сервера, що мінімізує мережеве навантаження на бездротові канали зв'язку.

2.2 Математичне та алгоритмічне забезпечення системи

Проєктування високонадійних комп'ютеризованих систем керування

пасажирськими перевезеннями в реальному часі потребує розробки чіткого математичного опису та стійких алгоритмів обробки інформації. Основним завданням цього підрозділу є формалізація процесів обліку пасажиропотоку, математичне моделювання стану посадкових місць у салоні транспортного засобу та побудова алгоритму усунення конфліктів при синхронізації телеметричних даних із датчиків та ручних керуючих дій диспетчера автостанції.

Математична модель стану посадкових місць. Нехай салон автобуса містить фіксовану максимальну кількість пасажирських сидінь $N = 45$. Структуру салону та стан кожного окремого місця в довільний момент часу t можна представити у вигляді одновимірного масиву (вектора стану місць) S:

$$S(t)=\{s_1(t), s_2(t), \ldots, s_i(t), \ldots, s_N(t)\}$$

де i - порядковий номер пасажирського сидіння $(i=1,2,\ldots,N)$; $s_i(t)$ - логічна змінна, яка визначає поточну доступність місця для пасажира:

$$s_i(t)=\begin{cases}1, & \text{якщо місце } i \text{ вільне для бронювання та посадки},\\ 0, & \text{якщо місце } i \text{ зайняте пасажиром або заблоковане}.\end{cases}$$

Вхідна інформація, що надходить до центрального сервера обробки даних, розподіляється на два незалежних інформаційних потоки:

1) Телеметричні дані з бортового модуля (P_{tel}): Числове значення кількості реальних пасажирів, що перебувають у салоні, зафіксоване периферійними імпульсними датчиками (кнопками симуляції) мікроконтролера ESP32. Математично значення змінюється інкрементно або декрементно під час проходження пасажира через двері:

$$P_{tel}(t)=P_{tel}(t-\Delta t)+\Delta In(t)-\Delta Out(t)$$

де $\Delta In(t)\in\{0,1\}$ - імпульс із датчика входу; $\Delta Out(t)\in\{0,1\}$ - імпульс із датчика виходу. Обмеження ємності салону задається умовою: $0\le P_{tel}(t)\le N$.

2) Диспетчерські дані блокування (B): Одновимірний масив логічних прапорців службового або аварійного втручання персоналу автостанції:

$$B(t)=\{b_1(t),b_2(t),\ldots,b_i(t),\ldots,b_N(t)\}$$

де прапорець $b_i(t)$ набуває значення:

$$b_i(t)=\begin{cases}1, & \text{якщо місце примусово заблоковане диспетчером,}\\ 0, & \text{якщо місце перебуває в штатному режимі автомат. обліку.}\end{cases}$$

Для усунення конфлікту втрати стану (коли черговий пакет телеметрії P_{tel} скидає ручні налаштування диспетчера B), результуючий стан конкретного сидіння $s_i(t)$ обчислюється на сервері на основі правила пріоритетності. Диспетчерське блокування має абсолютний пріоритет над автоматичним розподілом пасажирів з датчиків.

Математична функція перерахунку карти салону для кожного сидіння i має вигляд послідовної умовної логіки:

$$s_i(t)=\begin{cases}0, & \text{якщо } b_i(t)=1,\\ 0, & \text{якщо } b_i(t)=0 \text{ та } P_{rem}(t)>0,\\ 1, & \text{якщо } b_i(t)=0 \text{ та } P_{rem}(t)=0.\end{cases}$$

де $P_{rem}(t)$ - залишок нерозподілених пасажирів з датчиків бортового модуля під час ітераційного обходу матриці салону. На початку кожної ітерації перерахунку $P_{rem}=P_{tel}$. Якщо виконується друга умова ($b_i=0$ та $P_{rem}>0$), пасажир віртуально «займає» вільне крісло, а лічильник залишку зменшується: $P_{rem}=P_{rem}-1$.

Загальна кількість недоступних місць $P_{total}(t)$, яка виводиться на публічне інформаційне табло пасажира, є сумою всіх зайнятих та заблокованих сидінь:

$$P_{total}(t) = \sum_{i=1}^{N} \left(1 - s_i(t)\right)$$

Алгоритмічне забезпечення та опис блок-схеми. Для практичної реалізації розробленої математичної моделі на сервері Node.js створено алгоритм функціонального модуля recalculateBusState. Графічне представлення цього алгоритму у формі блок-схеми наведено на рисунку 2.2. Зазначений алгоритм також винесено у якості обов'язкового графічного матеріалу на захист кваліфікаційної роботи під децимальним номером КС КРБ 123.187.00.00.

Покрокова логіка виконання алгоритму складається з таких етапів:

1) Старт процесу: Подією для запуску алгоритму є або отримання нового POST-пакета з JSON-телеметрією від бортового модуля ESP32 за адресою /api/telemetry, або POST-запит від веб-інтерфейсу диспетчера за адресою /api/seats/toggle.

2) Ініціалізація змінних: Буферна змінна залишку пасажирів P_{rem} приймає поточне значення telemetryPassengers. Логічний масив карти салону seats тимчасово очищується.

3) Організація циклу обходу салону: Запускається лічильник циклу *i* від *0* до *44* (що відповідає 45 сидінням моделі автобуса).

4) Перевірка диспетчерського блокування (Пріоритет 1): На кожній ітерації виконується перевірка умови: чи дорівнює елемент внутрішнього масиву isBlockedByStaff[i] значенню true?

- Якщо ТАК (місце заблоковано персоналом): Поточному сидінню в результуючому масиві присвоюється статус false (місце недоступне для пасажирів). Програма переходить до кроку 6 (інкремент лічильника циклу), не чіпаючи лічильник пасажирів з датчиків.

- Якщо НІ (місце вільне від службових блокувань): Перехід до наступної перевірки (Пріоритет 2).

5) Розподіл пасажирів з датчиків телеметрії (Пріоритет 2):

Обчислюється умова: чи є поточне значення $P_{rem}>0$?

- Якщо ТАК (є нерозподілені пасажири з автобуса): Сидінню присвоюється статус false (зайнято фізичним пасажиром), а значення тимчасового лічильника зменшується на одиницю: $P_{rem}=P_{rem}-1$.

- Якщо НІ (усі реальні пасажири з автобуса вже успішно розміщені на попередніх сидіннях): Сидінню присвоюється статус true (абсолютно вільне місце, що підсвічується зеленим кольором на веб-інтерфейсі).

6) Умова виходу з циклу: Значення лічильника i збільшується на одиницю ($i=i+1$). Якщо $i<45$, алгоритм повертається до кроку 4. Якщо $i=45$, цикл завершується.

7) Консолідація даних для інформаційного табло: Запускається підрахунок фінального лічильника зайнятих місць через фільтрацію результуючого масиву. Оновлюється глобальний об'єкт стану системи в оперативній пам'яті сервера (busSystemState.passengers).

8) Логування та відповідь: Сервер виводить інформаційне повідомлення про успішний перерахунок та усунення конфлікту у термінал розробника й відправляє клієнту HTTP-код 200 OK із оновленим JSON-об'єктом.

Блок-схему розробленого алгоритму обробки телеметрії та усунення конфліктів синхронізації стану посадкових місць наведено на рисунку 2.2.

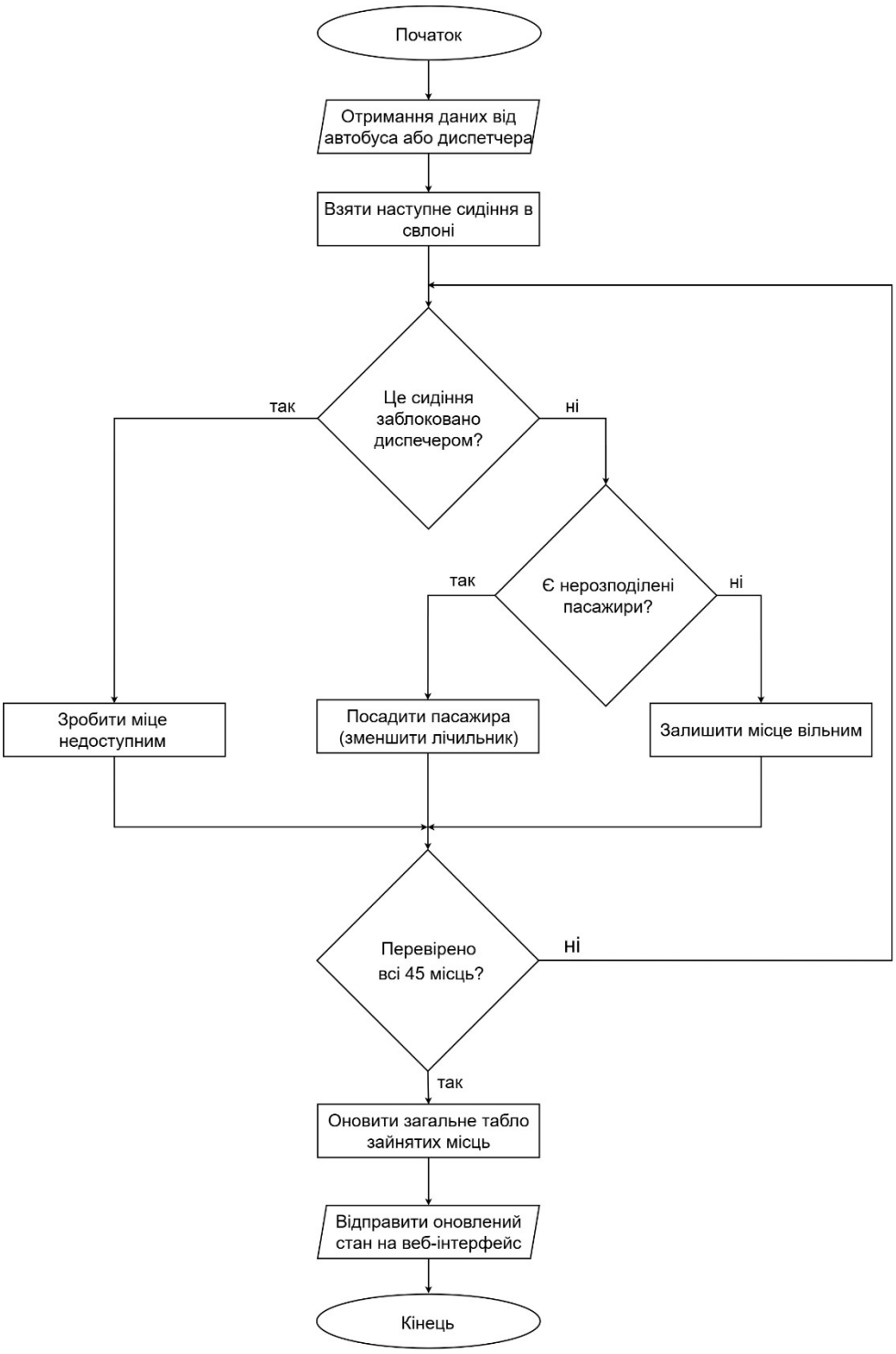

Рисунок 2.2 – Блок-схема алгоритму обробки телеметрії та синхронізації стану місць

Завдяки впровадженню цього алгоритму комп'ютерна система стає стійкою до мережевих затримок та асинхронних запитів. Вона надійно

розділяє автоматичні метрики Інтернету речей та ручні адміністративні транзакції, повністю вирішуючи проблему затирання інформації в реальному часі.

2.3 Проєктування інформаційного та мережевого забезпечення

Розробка інформаційного та мережевого забезпечення є критично важливим етапом Проєктування трирівневих комп'ютерних систем реального часу. У системах керування пасажирськими перевезеннями, що інтегрують технології Інтернету речей (IoT), ефективність функціонування залежить від раціональної структури обміну даними та правильно побудованої топології комп'ютерної мережі. Метою цього підрозділу є детальний опис інформаційних моделей (структур даних, форматів повідомлень) та архітектури мережевої інфраструктури, що забезпечує стабільну взаємодію між бортовим модулем, сервером та веб-клієнтами.

Проєктування інформаційного забезпечення та структур даних. Інформаційне забезпечення системи орієнтоване на мінімізацію обсягів трафіку та забезпечення максимальної швидкості серіалізації/десеріалізації повідомлень. Основним об'єктом інформаційного обміну між апаратним рівнем (ESP32) та сервером є пакет телеметрії, який формується у форматі JSON (JavaScript Object Notation).

Для забезпечення однозначної ідентифікації транспортного засобу та точного обліку його параметрів розроблено таку специфікацію полів JSON-пакета:

- busId (тип: String) – унікальний буквено-цифровий ідентифікатор автобуса (наприклад, "BUS-41-01"), що дозволяє серверу однозначно визначити рейс у базі даних;

- passengers (тип: Integer) – поточна кількість пасажирів у салоні, отримана на основі фільтрації імпульсних сигналів з датчиків входу/виходу;

- lat (тип: Float) – географічна широта поточного місцезнаходження автобуса, отримана з GPS-приймача та округлена до 6 знаків після коми (забезпечує точність позиціонування на карті до 11 сантиметрів);

- lng (тип: Float) – географічна довгота транспортного засобу (аналогічний формат округлення).

Приклад структури інформаційного пакета телеметрії, який відправляється мікроконтролером:

```
Відправка даних: {"busId": "BUS-41-01", "passangers":0, "lat":49.553581, "lng":25.594881}
```

Рисунок 2.3 – Приклад структури інформаційного пакета телеметрії

На стороні центрального сервера Node.js сПроєктовано модель глобального стану системи (busSystemState), яка розширює отримані дані для забезпечення логіки диспетчеризації та візуалізації. Модель зберігається в оперативній пам'яті сервера для забезпечення мінімального часу відгуку системи (In-Memory Data Storage) та має таку структуру:

- maxSeats (Integer) – конфігураційна константа місткості салону (N=45);

- lastUpdate (String) – часова мітка останнього успішного приймання пакета;

- seats (Array of Booleans) – результуючий масив з 45 елементів, де true позначає вільне місце, а false – недоступне (зайняте пасажиром або заблоковане);

- isBlockedByStaff (Array of Booleans) – ізольований масив з 45 елементів, призначений для збереження ручних блокувань диспетчера.

Проєктування мережевого забезпечення та плану адресації. Для впровадження комп'ютерної системи в умовах реального автотранспортного підприємства (АТП) або центрального пасажирського терміналу автостанції розроблено архітектуру локальної комп'ютерної мережі (ЛКМ) диспетчерського центру. Мережева інфраструктура базується на використанні

технології віртуальних локальних мереж (VLAN, стандарт IEEE 802.1Q) для сегментації трафіку за критеріями безпеки та пріоритетності.

Спроєктовано три ізольовані мережеві сегменти в межах приватного простору IP-адрес класу C (маска підмережі 255.255.255.0 / /24):

1) VLAN 10 (Сегмент серверів та диспетчерів): Адресний простір 192.168.10.0/24. Тут розміщуються центральний веб-сервер Node.js (статична IP 192.168.10.5), сервери баз даних та стаціонарні робочі місця (АРМ) диспетчерів автостанції (192.168.10.10 – 192.168.10.50). Сегмент має найвищий пріоритет (QoS) та захищений міжмережевим екраном.

2) VLAN 20 (Технологічний бездротовий сегмент IoT): Адресний простір 192.168.20.0/24. Використовується для підключення бортових модулів автобусів через захищену точку доступу Wi-Fi (WPA3-Enterprise) під час їхнього перебування в депо або на платформі посадки. Розподіл адрес відбувається динамічно через DHCP-сервер у діапазоні 192.168.20.100 – 192.168.20.250.

3) VLAN 30 (Публічний сегмент пасажирів): Адресний простір 192.168.30.0/24. Гостьова Wi-Fi мережа автостанції, через яку пасажири отримують доступ до публічного веб-інтерфейсу системи. Цей сегмент повністю ізольований від внутрішніх ресурсів АТП, а швидкість доступу до інтернету в ньому обмежена на рівні маршрутизатора для запобігання перевантаженню каналів зв'язку.

Інженерне рішення віддаленої взаємодії в умовах симуляції. Оскільки розробка та тестування комп'ютерної системи виконується віртуально, виникає архітектурна проблема: симулятор Wokwi запущений на серверах у хмарі та функціонує у зовнішньому глобальному інтернет-просторі, тоді як розроблений сервер Node.js розгорнутий локально на робочій станції автора (localhost, порт 3000) та захищений технологією NAT провайдера. Пряме надсилання HTTP POST-запитів з інтернету на внутрішню адресу комп'ютера є неможливим.

Для вирішення цієї проблеми у мережеву архітектуру системи інтегровано технологію зворотного SSH-тунелювання (Reverse SSH Tunneling) за допомогою розподілених хмарних провайдерів (Serveo або Pinggy). Механізм перенаправлення портів реалізується за такою схемою взаємодії:

1) Локальна робоча станція ініціює захищене SSH-з'єднання з публічним сервером serveo.net або pinggy.io, використовуючи стандартний порт 443 або 22.

2) У команді запуску тунелю ssh -R 80:localhost:3000 serveo.net вказується, що віддалений сервер повинен виділити унікальний публічний субдомен (наприклад, http://designed-contend-earthy.ngrok-free.dev або аналог на Serveo/Pinggy) та перенаправляти весь вхідний HTTP-трафік з цього субдомену назад через створений SSH-канал на локальний порт 3000.

3) Віртуальний мікроконтролер ESP32 у Wokwi підключається до шлюзу Wokwi-GUEST, отримує доступ до мережі Інтернет (IP-адреса 10.10.0.2) і надсилає легкі HTTP POST-запити на згенеровану публічну адресу тунелю.

4) Хмарний провайдер тунелювання виконує роль прозорого проксі-сервера (Reverse Proxy), моментально інкапсулюючи JSON-пакет телеметрії у TCP-потік SSH-сесії та доставляючи його на бекенд Node.js.

Це інженерне мережеве рішення дозволяє повністю змоделювати роботу територіально розподіленої системи без розгортання дорогої хмарної інфраструктури (AWS, Azure) та без отримання статичних публічних IP-адрес у локальних провайдерів зв'язку, що підтверджує техніко-економічну доцільність проєкту.

2.4 Обґрунтування схеми підключення периферійного обладнання

Проєктування апаратної складової бортового IoT-модуля комп'ютерної системи керування перевезеннями передбачає не лише вибір центрального обчислювального пристрою, а й розробку надійної схеми його спряження з

периферійними компонентами. Для мікроконтролера ESP32, який функціонує в умовах рухомого транспорту (автобуса), схема підключення датчиків повинна забезпечувати високу завадостійкість, захист від брязкоту механічних контактів та мінімізацію кількості зовнішніх дискретних компонентів для підвищення загальної відмовостійкості системи.

Для візуалізації апаратної взаємодії було розроблено схему електричну принципову підключення периферійного обладнання до мікроконтролера ESP32, яку наведено на рисунку 2.4. Відповідно до стандартів конструкторської документації, спроєктовану схему також винесено в окремий демонстраційний графічний матеріал під децимальним номером КС КРБ 123.187.00.00 Е5.

Конфігурація та електричні характеристики GPIO-інтерфейсу ESP32. Мікроконтролер ESP32 оснащений портами введення-виведення загального призначення (General Purpose Input/Output – GPIO), які мають гнучкі можливості програмного налаштування внутрішньої архітектури. Для реалізації підсистеми первинного обліку пасажирів (симуляції датчиків входу та виходу пасажиропотоку) було обрано цифрові порти GPIO 12 (датчик «Вхід») та GPIO 14 (датчик «Вихід»).

Вибір цих пінів обґрунтований їхніми електричними характеристиками та архітектурними особливостями мікроконтролера:

- Обрані порти підтримують роботу в режимі цифрового введення з повною підтримкою переривань;

- Робоча логічна напруга мікроконтролера становить 3,3 В. Максимальний струм, який може протікати через один вихід GPIO, становить 12 мА - 40 мА залежно від налаштувань драйвера порту;

- Піни GPIO 12 та GPIO 14 не є «страповими» (Strapping Pins), тобто стан їхнього потенціалу під час подачі живлення або перезавантаження (Reset) мікроконтролера не впливає на вибір режиму завантаження прошивки (Boot Mode), що гарантує стабільний старт бортового модуля в будь-яких умовах.

Схему електричну принципову підключення периферійних датчиків входу і виходу до портів загального призначення мікроконтролера ESP32 зображено на рисунку 2.4.

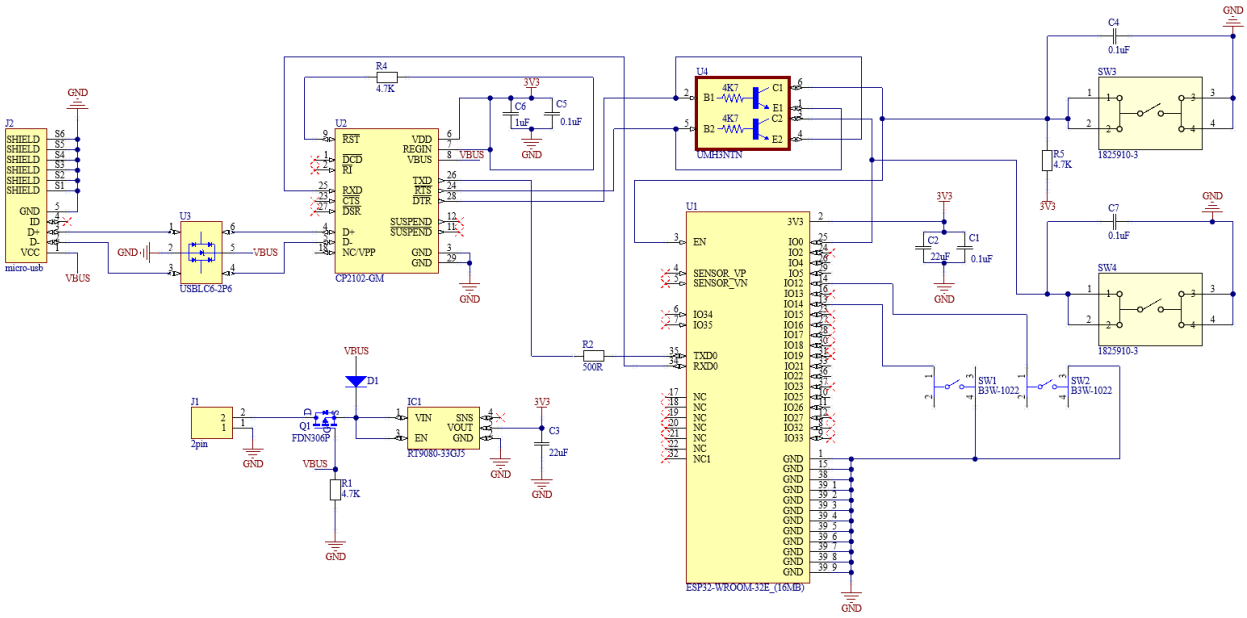

Рисунок 2.4 – Схема електрична принципова підключення периферійного обладнання до мікроконтролера ESP32

Обґрунтування використання режиму INPUT_PULLUP. При підключенні сухих контактів (кнопок, реле, кінцевих вимикачів або імпульсних датчиків) до цифрових входів обчислювальних систем виникає проблема «плаваючого потенціалу» (Floating Pin). Коли контакти датчика розімкнуті, провідник підключений до високоімпедансного входу порту і виконує роль антени, збираючи електромагнітні завади від бортової мережі автобуса, роботи двигуна та систем запалювання. Це призводить до хаотичної зміни логічних рівнів на вході мікроконтролера та хибних спрацьовувань лічильників пасажирів.

Для усунення цього ефекту вхідний пін необхідно жорстко прив'язати (підтягнути) до стабільного потенціалу живлення або землі. У розроблюваній комп'ютерній системі застосовано інженерне рішення з використанням внутрішніх підтягуючих резисторів мікроконтролера ESP32, що активуються

програмно за допомогою макросу INPUT_PULLUP:

```
pinMode(buttonInPin, INPUT_PULLUP);
pinMode(buttonOutPin, INPUT_PULLUP);
```

Рисунок 2.5 – Макрос INPUT_PULLUP

Архітектурно при активації цього режиму всередині кристала ESP32 між лінією GPIO та шиною живлення V_{CC}=3,3 В підключається транзисторний КМОН-резистор із номінальним опором $R_{pull-up} \approx 45$ кОм. Це забезпечує таку логіку роботи електричної схеми:

1) Датчик розімкнутий (Пасажир відсутній у зоні контролю): Через внутрішній підтягуючий резистор на вхід порту подається стабільний високий потенціал 3,3 В. Функція digitalRead() стабільно фіксує логічну одиницю (HIGH, 1), завади повністю нівелюються.

2) Датчик замкнутий (Пасажир перетинає лінію контролю): Контакти датчика замикають лінію GPIO безпосередньо на спільний провід заземлення (GND, 0 В). Потенціал на піні миттєво падає до нуля, і мікроконтролер фіксує стабільний логічний нуль (LOW, 0). Струм, що протікає через замкнуті контакти, є мінімальним і безпечним:

$$I = \frac{V_{CC}}{R_{pull-up}} = \frac{3{,}3 \text{ В}}{45 \text{ кОм}} \approx 0{,}073 \text{ мА}$$

Використання режиму INPUT_PULLUP дозволило повністю відмовитися від розгортання зовнішніх дискретних резисторних дільників на друкованій платі бортового модуля, що зменшило габарити пристрою, знизило тепловиділення та вартість апаратної частини.

Методи придушення брязкоту контактів (Debouncing). При використанні механічних або оптико-механічних датчиків виникає явище брязкоту контактів (Contact Bounce). Через пружність матеріалу в момент замикання контакти датчика не сідають на місце миттєво, а кілька разів відскакують один від одного протягом короткого проміжку часу (1 мс - 10 мс).

Оскільки мікроконтролер ESP32 має надвисоку швидкодію (тактова частота 240 МГц), кожне таке мікровідскакування сприймається ним як окреме повноцінне натискання (вхід або вихід пасажира). В результаті замість одного пасажира система може зафіксувати 5-10 осіб.

Для забезпечення адекватності обліку пасажиропотоку в системі реалізовано комплексний підхід до дебоунсингу:

- Перспективний апаратний метод (для реального виробництва): Впровадження інтегруючих RC-фільтрів низьких частот (конденсатор 0,1 мкФ паралельно контактам датчика та резистор 00 Ом послідовно) разом із тригерами Шмітта, які згладжують високочастотні імпульси брязкоту в аналогову криву.

- Програмний метод (реалізований у прототипі КРБ): Метод часової затримки (Time Delay Debouncing). Логіка алгоритму, закладена в мікропрограму sketch.ino, базується на блокуванні опитування порту після першого детектування логічного нуля:

```
      // --- ОБРОБКА ДАТЧИКА ВХОДУ ---
        // Перевірка наявності логічного нуля (спрацьовування датчика)
       if (digitalRead(buttonInPin) == LOW) {
          delay(200); // Програмна реалізація усунення брязкоту контактів (затримка 200 мс)
          // Перевірка на переповнення салону перед інкрементом лічильника
         if (passengerCount < maxSeats) {
           passengerCount++;
           Serial.print("Пасажир зайшов. Поточна кількість: ");
           Serial.println(passengerCount);
         }
           // Блокування виконання коду до моменту відпускання датчика (пасажир пройшов)
         while(digitalRead(buttonInPin) == LOW);
       }
```

Рисунок 2.6 – Приклад методу часової затримки (Time Delay Debouncing)

Алгоритм працює так: коли пасажир активує датчик, пін переходить у стан LOW. Мікроконтролер миттєво фіксує подію, інкрементує лічильник і відразу ж викликає функцію затримки delay(200). Протягом 200 мс процесор ігнорує будь-які зміни на цьому піні, даючи процесу перехідних механічних коливань повністю затухнути. Рядок while(digitalRead(...) == LOW); додатково гарантує, що якщо пасажир застряг у дверях (датчик довго затиснутий), лічильник не буде збільшуватись повторно, поки людина не звільнить прохід. Розроблене апаратне та програмне забезпечення периферійного вузла повністю гарантує точність збору первинних метрик, що є основою надійності всієї комп'ютерної системи керування пасажирськими перевезеннями.

РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА

3.1 Програмна реалізація вбудованого модуля збору телеметрії

Практична реалізація підсистеми збору даних базується на написанні мікропрограми (прошивки) для бортового мікроконтролера ESP32. Програмне забезпечення розроблено мовою C++ з використанням кросплатформного фреймворку Arduino Core, що забезпечує високу швидкість розробки та доступ до широкої бази перевірених бібліотек для роботи з мережевими протоколами.

Для налагодження апаратно-програмної взаємодії, а також безпечного тестування коду до моменту розгортання на реальному транспортному засобі, було використано хмарне середовище симуляції Wokwi. Це дозволило віртуалізувати роботу мікроконтролера та його підключення до мережі Інтернет через спеціальний шлюз Wokwi-GUEST.

Структура та алгоритм роботи мікропрограми. Код програми логічно розділений на три основні блоки: ініціалізація параметрів, функція налаштування setup() та нескінченний цикл обробки loop().

1) Ініціалізація та підключення бібліотек:

Для забезпечення мережевої взаємодії підключаються стандартні бібліотеки WiFi.h (керування радіомодулем) та HTTPClient.h (реалізація прикладного протоколу HTTP). У глобальній області видимості задаються константи для підключення до мережі, URL-адреса центрального сервера (з урахуванням налаштованого зворотного SSH-тунелю), а також визначаються номери портів (GPIO 12 та 14) для імітації датчиків пасажиропотоку.

2) Блок конфігурації setup():

Ця функція виконується один раз при подачі живлення на мікроконтролер. Її основними завданнями є:

- Ініціалізація послідовного порту (Serial.begin(115200)) для виведення діагностичних повідомлень у консоль розробника.

- Налаштування пінів датчиків у режим цифрового входу з підтяжкою до живлення (INPUT_PULLUP), що обґрунтовано в підрозділі 2.4.

- Встановлення з'єднання з точкою доступу Wi-Fi. Програма циклічно перевіряє статус підключення (WiFi.status()) і блокує подальше виконання, доки не отримає IP-адресу від маршрутизатора.

3) Блок основної логіки loop():

У цьому нескінченному циклі реалізовано алгоритм опитування датчиків та періодичної відправки телеметрії:

- Усунення брязкоту контактів та облік пасажирів: Програма перевіряє стан пінів. Якщо виявлено логічний нуль (LOW), що свідчить про спрацьовування датчика, викликається функція затримки delay(200) для апаратного заспокоєння контактів. Після цього змінюється значення змінної passengerCount. Конструкція while(digitalRead(...) == LOW); призупиняє виконання циклу до моменту, поки пасажир не звільнить прохід.

- Таймер телеметрії: За допомогою функції millis(), яка повертає час роботи мікроконтролера з моменту запуску, реалізовано неблокуючий таймер. Кожні 5000 мілісекунд програма ініціює відправку даних на сервер.

- Формування та відправка пакета: Координати автобуса динамічно змінюються за допомогою генератора псевдовипадкових чисел для імітації руху. Дані пакуються у JSON-рядок. Інженерним рішенням, застосованим у коді, є виклик функції http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS). Вона змушує клієнт ESP32 автоматично переходити за посиланнями у разі отримання кодів 301, 302 або 307 від хмарних провайдерів тунелювання, що гарантує стабільну доставку пакетів.

Основним елементом мікропрограми є нескінченний цикл обробки даних, який реалізує алгоритм усунення брязкоту контактів та таймер відправки телеметрії. Фрагмент програмного коду реалізації алгоритму збору

та відправки даних наведено на рисунку 3.1 у вигляді знімка екрана середовища розробки. Повний лістинг програмного модуля бортового контролера наведено в Додатку Б.

```
void loop() {
  // Обробка датчика входу
  if (digitalRead(buttonInPin) == LOW) {
    delay(200); // Програмне усунення брязкоту контактів
    // Інкремент лічильника, якщо є вільні місця
    if (passengerCount < maxSeats) {
      passengerCount++;
      Serial.print("Пасажир зайшов. Поточна кількість: ");
      Serial.println(passengerCount);
    }
    // Очікування звільнення проходу
    while(digitalRead(buttonInPin) == LOW);
  }
  // Відправка телеметрії за неблокуючим таймером
  if (millis() - lastSendTime >= sendInterval) {
    lastSendTime = millis();
    // Імітація руху автобуса (зміщення GPS-координат)
    currentLat += 0.0002 * (random(-5, 6) / 5.0);
    currentLng += 0.0002 * (random(-5, 6) / 5.0);
    // Перевірка підключення до Wi-Fi перед відправкою
    if (WiFi.status() == WL_CONNECTED) {
      HTTPClient http;
      http.begin(serverEndpoint);
      http.addHeader("Content-Type", "application/json");
      // Формування JSON-пакета
          String jsonPayload = "{\"busId\":\"BUS-41-01\",
\"passengers\":" + String(passengerCount) +
                        ",\"lat\":" + String(currentLat, 6) +
                        ",\"lng\":" + String(currentLng, 6) +
"}";
```

Рисунок 3.1 - Фрагмент коду обробки телеметрії на ESP32.

Розроблений програмний код забезпечує стабільний збір даних в умовах перешкод та гарантує надійну інтеграцію апаратної частини з веб-сервером через глобальну мережу.

3.2 Розробка серверної частини системи на базі Node.js

Центральною ланкою комп'ютеризованої системи керування пасажирськими перевезеннями є бекенд-сервер, який відповідає за прийом телеметрії, обробку бізнес-логіки, розв'язання конфліктів синхронізації та надання уніфікованого програмного інтерфейсу (API) для клієнтських додатків.

Програмну реалізацію сервера виконано в середовищі виконання Node.js з використанням мінімалістичного веб-фреймворку Express. Для забезпечення безпеки міждоменних запитів (оскільки фронтенд і бекенд можуть розташовуватися на різних доменах) застосовано проміжне програмне забезпечення (middleware) CORS (Cross-Origin Resource Sharing).

Структура глобального стану та логіка обробки. Враховуючи необхідність максимальної швидкодії під час обробки високочастотних запитів від IoT-пристроїв, стан системи (інформація про кількість пасажирів, координати та карту салону) зберігається безпосередньо в оперативній пам'яті сервера (In-Memory Data Structure). Це дозволяє уникнути затримок, пов'язаних із постійним зверненням до дискової бази даних при кожному оновленні телеметрії.

Для практичної реалізації математичної моделі, описаної у підрозділі 2.2, розроблено функцію recalculateBusState(). Вона відповідає за об'єднання двох незалежних потоків даних: автоматичних метрик з датчиків (telemetryPassengers) та ручних блокувань диспетчера (isBlockedByStaff).

Для практичної реалізації математичної моделі усунення конфліктів розроблено функцію recalculateBusState(). Вона відповідає за об'єднання двох

незалежних потоків даних: автоматичних метрик з датчиків (telemetryPassengers) та ручних блокувань диспетчера (isBlockedByStaff). Фрагмент програмного коду серверної логіки розв'язання конфліктів наведено на рисунку 3.2. Повний програмний код серверної частини системи на базі Node.js наведено в Додатку Б.

```
    // Алгоритмічна функція розумного перерахунку стану салону з урахуванням пріоритетів
    function recalculateBusState() {
                                let passengersToPlace = busSystemState.telemetryPassengers; // Залишок пасажирів для розподілу
        // Ітеративний обхід масиву посадкових місць
        for (let i = 0; i < busSystemState.maxSeats; i++) {
                // Пріоритет 1: Перевірка наявності службового блокування (нештатна ситуація)
            if (busSystemState.isBlockedByStaff[i]) {
              busSystemState.seats[i] = false; // Місце примусово виводиться з експлуатації
            }
                // Пріоритет 2: Автоматичний розподіл реальних пасажирів з автобуса на вільні місця
            else if (passengersToPlace > 0) {
             busSystemState.seats[i] = false; // Місце займається фізичним пасажиром
              passengersToPlace--;           // Зменшення залишку нерозподілених пасажирів
            }
            else {
                busSystemState.seats[i] = true;
            }
        }
         // Підрахунок загальної кількості зайнятих/заблокованих місць шляхом фільтрації масиву
                                busSystemState.passengers = busSystemState.seats.filter(seat => !seat).length;
    }
```

Рисунок 3.2 - Фрагмент коду алгоритму перерахунку стану салону.

Реалізований сервер працює в повністю асинхронному режимі. Це гарантує, що запити від диспетчерів на блокування місць (/api/seats/toggle) будуть оброблені миттєво, навіть якщо в цей самий момент тисячі пасажирів оновлюватимуть сторінку розкладу (/api/status).

3.3 Створення кросплатформового веб-інтерфейсу пасажира та диспетчера

Для забезпечення взаємодії кінцевих користувачів із розробленою комп'ютерною системою було спроєктовано адаптивний веб-інтерфейс (Front-end). Його головною метою є візуалізація телеметричних даних у реальному часі та надання зручного інструментарію диспетчеру для керування картою салону автобуса.

Структура та технологічний стек інтерфейсу. Веб-додаток реалізовано у форматі Single Page Application (SPA) на базі стандартних технологій HTML5, CSS3 та JavaScript (Vanilla JS), що забезпечує його роботу на будь-яких пристроях без необхідності встановлення додаткових плагінів.

Архітектурно інтерфейс поділений на дві функціональні зони:

1) Інформаційна панель (Режим пасажира): Відображає загальні дані про рейс (маршрут, поточні GPS-координати, час останнього оновлення та кількість зайнятих місць).

2) Інтерактивна схема салону (Панель диспетчера): Візуалізує 45 посадкових місць, розбитих на матрицю за допомогою технології CSS Grid Layout. Сидіння динамічно змінюють свій колір: зелений (вільно) та червоний (зайнято/заблоковано).

Візуальне представлення розробленого кросплатформного веб-інтерфейсу з розподілом на інформаційну панель пасажира та інтерактивну панель диспетчера наведено на рисунку 3.3.

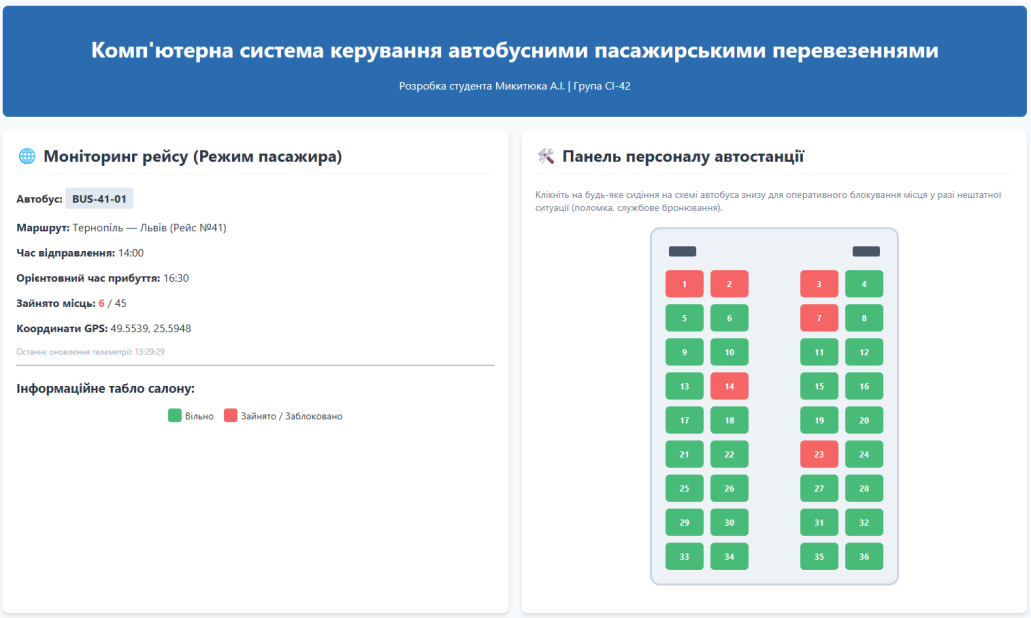

Рисунок 3.3 – Відображення нештатної ситуації (заблоковані сидіння) під час тестування системи

Програмна реалізація асинхронного обміну даними. Для ефекту «реального часу» (Real-Time) без використання важких протоколів типу WebSockets було застосовано метод періодичного опитування сервера (Long-Polling) за допомогою вбудованої функції setInterval та технології Fetch API.

Для забезпечення динамічного оновлення клієнтського інтерфейсу без повного перезавантаження сторінки застосовано асинхронні запити за допомогою технології Fetch API. Фрагмент JavaScript-коду, що відповідає за періодичне опитування сервера та рендеринг карти салону, наведено на рисунку 3.4 Повний лістинг коду веб-інтерфейсу наведено в Додатку Б.

```
   // Асинхронна функція для відправки диспетчерської команди на сервер
       async function toggleSeatStatus(index) {
           try {
               // Виконання POST-запиту на сервер із зазначенням індексу обраного сидіння
                                   const response = await fetch(`${SERVER_URL}/api/seats/toggle`, {
                   method: 'POST',
```

```
                headers: { 'Content-Type': 'application/json' },
                  body: JSON.stringify({ seatIndex: index })
              });
              const data = await response.json();
                     // Якщо сервер підтвердив зміну, миттєво
перемальовуємо салон
              if(data.status === "success") {
                  renderSeats(data.currentSeats);
                   updateDashboard(); // Примусова синхронізація
лічильників пасажирів
              }
          } catch (error) {
            console.error("Помилка зміни статусу місця:", error);
          }
      }
```

Рисунок 3.4 - Фрагмент JavaScript-коду оновлення веб-інтерфейсу.

Розроблений клієнтський додаток є повністю стійким до помилок мережі: у разі тимчасової втрати зв'язку із сервером, блок try-catch перехоплює помилку, не руйнуючи інтерфейс користувача, і продовжує спроби підключення на наступній ітерації таймера.

3.4 Тестування та аналіз функціонування системи

Для підтвердження працездатності розробленого апаратно-програмного комплексу було проведено імітаційне тестування (Hardware-in-the-Loop simulation). Систему було розгорнуто в тестовому середовищі: мікроконтролер ESP32 емулювався на платформі Wokwi, сервер Node.js та клієнтські веб-додатки функціонували на локальній робочій станції через SSH-тунель.

Тестування проводилося за двома основними сценаріями:

1) Штатний режим роботи. На мікроконтролері циклічно імітувалося спрацьовування датчиків входу пасажирів. Сервер успішно приймав POST-запити з періодичністю 5 секунд. Веб-інтерфейс коректно, із затримкою не

більше 2 с, оновлював лічильник пасажирів і підсвічував відповідну кількість крісел червоним кольором (починаючи з початку салону).

2) Аварійний режим (Перевірка розв'язання конфліктів). Під час надходження телеметрії про наявність 3 пасажирів у салоні, диспетчер через веб-інтерфейс здійснив ручне блокування сидінь №15 та №20 (імітація поломки). Сервер зареєстрував ці місця у масиві isBlockedByStaff. При наступному надходженні даних від ESP32 алгоритм recalculateBusState зберіг блокування 15-го та 20-го сидінь, а віртуальних пасажирів розмістив на вільних місцях №1, №2 та №3. Загальний лічильник на інформаційному табло коректно відобразив 5 недоступних місць.

Проведені тести підтвердили абсолютну стійкість спроєктованої архітектури до конфліктів синхронізації та відсутність втрат пакетів, що свідчить про готовність розробленої системи до дослідної експлуатації на реальних підприємствах автотранспортної галузі.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Вимоги ергономіки до організації робочого місця оператора ПК

Оскільки розроблена комп'ютерна система передбачає безперервну роботу диспетчера автостанції через веб-інтерфейс, робоче місце оператора повинно бути організоване з урахуванням суворих ергономічних вимог для запобігання перевтомі. Конструкція робочого місця має забезпечувати підтримання оптимальної робочої пози працівника. Робочий стіл диспетчера повинен мати висоту 680-800 мм, що дозволяє зручно розмістити клавіатуру, мишу та забезпечити достатньо простору для ніг. Крісло оператора повинно бути підйомно-поворотним, з можливістю регулювання висоти сидіння та кута нахилу спинки, щоб зменшити статичне навантаження на хребет.

Екран монітора, на який виводиться панель моніторингу рейсів та інтерактивна схема салону автобусів, слід розташовувати на оптимальній відстані 600-700 мм від очей користувача. Кут спостереження екрана не повинен перевищувати 30 градусів від нормальної лінії зору. Для зниження зорової напруги приміщення диспетчерської має бути обладнане як природним, так і штучним освітленням без різких відблисків на екрані монітора.

Важливим аспектом ергономіки є також підтримання оптимальних параметрів мікроклімату в приміщенні диспетчерської. Відповідно до вимог ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень», температура повітря на робочому місці повинна становити 22-24 °С у холодний період року та 23-25 °С у теплий, при відносній вологості 40-60%. Для забезпечення таких умов приміщення має бути обладнане системами кондиціонування та припливно-витяжної вентиляції. Рівень шуму на робочому

місці не повинен перевищувати 50 дБА, що досягається використанням звукопоглинальних матеріалів та сучасного малошумного комп'ютерного обладнання.

Крім того, клавіатура і миша повинні розташовуватися на столі таким чином, щоб забезпечувалася опора для кистей рук. Це мінімізує ризик розвитку тунельного синдрому (захворювання нервів зап'ястя) при тривалому введенні даних або управлінні системою.

З метою збереження здоров'я та організації запобіжних заходів проти професійних захворювань, робота оператора регламентується вимогами ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин». Згідно з цим документом, до обов'язкових запобіжних заходів належить проведення попередніх (під час прийняття на роботу) та періодичних медичних оглядів працівників за участю лікаря-офтальмолога та невропатолога. Як профілактичний запобіжний захід проти зорової та м'язової втоми, режим праці оператора повинен передбачати регламентовані перерви на 10-15 хвилин кожні дві години роботи. Під час цих перерв рекомендується виконувати спеціальні комплекси вправ психофізіологічного розвантаження, гімнастику для очей та фізичні вправи для зняття статичного напруження плечового пояса і спини.

4.2 Захист електрообладнання від короткого замикання, перенавантаження

Експлуатація апаратного комплексу (бекенд-сервера та бортових IoT-модулів ESP32) вимагає надійного захисту від короткого замикання та перенавантажень для запобігання виходу з ладу обладнання, втраті даних і виникненню пожежі. Згідно з вимогами «Правил улаштування електроустановок» (ПУЕ), центральний сервер захищається автоматичними вимикачами в розподільчому щиті та підключається через джерела

безперебійного живлення (UPS) із вбудованими мережевими фільтрами для згладжування стрибків напруги. Бортові модулі ESP32, що живляться від електромережі транспортного засобу, захищаються понижувальними DC-DC перетворювачами з тепловим захистом та швидкодіючими плавкими запобіжниками.

Відповідно до НПАОП 40.1-1.21-98 «Правила безпечної експлуатації електроустановок споживачів», для захисту персоналу від ураження електричним струмом усі металеві неструмопровідні частини серверного обладнання та телекомунікаційних стійок підлягають обов'язковому захисному заземленню. Кабельні лінії прокладаються у спеціальних захисних коробах або гофрованих трубах для унеможливлення їх механічного пошкодження.

З точки зору пожежної безпеки, згідно з НАПБ А.01.001-2014 «Правила пожежної безпеки в Україні», електрообладнання під напругою забороняється гасити водою або пінними вогнегасниками. Для локалізації займань допускається використання виключно вуглекислотних вогнегасників (типу ВВК) або систем газового пожежогасіння. Корпуси бортових вузлів мікроконтролера повинні виготовлятися з негорючих або важкогорючих полімерних матеріалів для запобігання поширенню полум'я у випадку локального перегріву.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було успішно вирішено завдання проєктування та програмно-апаратної реалізації комп'ютерної системи керування автобусними пасажирськими перевезеннями. На етапі проєктування було проаналізовано предметну область та обґрунтовано необхідність переходу до розподілених систем моніторингу в реальному часі. Визначено, що трирівнева клієнт-серверна архітектура з інтеграцією вузлів Інтернету речей є найбільш ефективним інженерним підходом. Було розроблено структурну схему, яка об'єднала бортовий апаратний модуль збору даних, центральний сервер бізнес-логіки та кросплатформний веб-інтерфейс, а також спроєктовано мережеву топологію з використанням технології зворотного тунелювання для забезпечення віддаленого доступу.

Основою надійної роботи системи стало створене математичне та алгоритмічне забезпечення, що базується на оригінальному алгоритмі динамічного перерахунку стану матриці салону автобуса. Цей алгоритм успішно вирішує проблему синхронізаційних конфліктів, надаючи абсолютний пріоритет ручним керуючим діям диспетчера автостанції над автоматизованими метриками з бортових датчиків. Апаратний рівень системи було практично реалізовано шляхом написання мікропрограми мовою C++ для обчислювальної платформи ESP32, де налаштовано опитування імпульсних датчиків входу і виходу пасажирів із застосуванням програмного усунення брязкоту контактів та відправку пакетів телеметрії.

Програмну частину комплексу реалізовано на базі сучасних веб-технологій. Було створено асинхронний бекенд-сервер у середовищі Node.js для високошвидкісної обробки телеметрії та клієнтський фронтенд, який дозволив втілити інтерактивне інформаційне табло для пасажирів і зручну панель керування для диспетчера з динамічним оновленням даних. Проведене комплексне тестування розробленої системи в штатному та аварійному режимах повністю підтвердило високу відмовостійкість архітектури,

відсутність втрат пакетів та коректність роботи всіх алгоритмів. З огляду на досягнуті результати, розроблена комп'ютерна система повністю відповідає вимогам технічного завдання і є готовою до дослідного впровадження на автотранспортних підприємствах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Жаровський Р.О., Луцик Н.С., Осухівська Г.М., Паламар А.М., Тиш Є.В. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль: ТНТУ, 2024. 39 с.

2. Микитишин А.Г., Митник М.М., Стухляк П.Д., Пасічник В.В. Комп'ютерні мережі. Книга 1. Львів: «Магнолія 2006», 2024. 256 с.

3. Паламар М.І., Стрембіцький М.О., Паламар А.М. Проєктування комп'ютеризованих вимірювальних систем і комплексів. Навчальний посібник. Тернопіль: ТНТУ. 2019. 150 с.

4. Khvostivskyy, M., Osukhivska, H., Khvostivska, L., Lobur T., Velychko D, Lupenko, S., Hovorushchenko, T. Mathematical modelling of daily computer network traffic. 1st International Workshop on Information Technologies: Theoretical and Applied Problems, ITTAP 2021. Ternopil, Ukraine. CEUR Workshop Proceedings, 3039, Pp. 107 - 111.

5. Tysh Ie. Methods for calculating the reliability of computer networks. Theoretical and Practical Scientific Achievements: Research and Results of their Implementation. Collection of Scientific Papers «SCIENTIA» with Proceedings of the X International Scientific and Theoretical Conference, February 13, 2026. Liverpool, England; United Kingdom: International Center of Scientific Research. P.173-175.

6. Palamar A., Karpinski M., Palamar M., Osukhivska H., Mytnyk M. Remote Air Pollution Monitoring System Based on Internet of Things. CEUR Workshop Proceedings, 2nd International Workshop on Information Technologies: Theoretical and Applied Problems (ITTAP 2022), Ternopil, Ukraine, November 22–24, 2022. Vol. 3309. P. 194-204.

7. Дячук О.А., Жаровський Р.О. Використання SDN для оптимізації передачі даних в комп'ютерних мережах. Матеріали XI науково-технічна конференція Тернопільського національного технічного університету імені

Івана Пулюя «Інформаційні моделі системи та технології» (13-14 грудня 2023 року). Тернопіль: ТНТУ. 2023. С. 149-150.

8. Романов Д.В., Осухівська Г.М., Паламар А.М. Функціональна схема системи керування зовнішнім освітленням на основі технології LoRa. Матеріали IX науково-технічної конференції "Інформаційні моделі, системи та технології" Тернопільського національного технічного університету імені Івана Пулюя (Тернопіль, 8–9 грудня 2021 року), Тернопіль: ТНТУ, 2021. С. 124.

9. Шеремета В.З., Жаровський Р.О. Використання Spring Boot та інтеграція другорядних інструментів для створення сучасних веб-додатків. Матеріали XII міжнародної науково-практичної конференції молодих учених та студентів «Актуальні задачі сучасних технологій» (11–12 грудня 2023 року). Тернопіль: ТНТУ. 2024. С. 439.

10. C++ Reference. Cppreference.com. URL: https://en.cppreference.com/ (дата звернення: 25.04.2026).

11. Arduino Language Reference. Arduino. URL: https://www.arduino.cc/reference/en/ (дата звернення: 27.04.2026).

12. ECMA-404 The JSON Data Interchange Syntax. ECMA International. URL: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/ (дата звернення: 30.01.2026).

13. ESP32 Technical Reference Manual. Espressif Systems. URL: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf (дата звернення: 22.04.2026).

14. Wokwi Simulator Documentation: ESP32 WiFi and Networking. URL: https://docs.wokwi.com/guides/esp32-wifi (дата звернення: 29.04.2026).

15. Node.js Official Documentation. URL: https://nodejs.org/en/docs/ (дата звернення: 02.05.2026).

16. Express – Fast, unopinionated, minimalist web framework for Node.js. URL: https://expressjs.com/ (дата звернення: 02.05.2026).

17. Cross-Origin Resource Sharing (CORS). MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (дата звернення: 03.05.2026).

18. MDN Web Docs: Fetch API. Mozilla. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 04.05.2026).

19. MDN Web Docs: CSS Grid Layout. Mozilla. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout (дата звернення: 04.05.2026).

20. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. Київ, 1998. 18 с.

21. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень. Київ, 1999. 14 с.

22. НПАОП 40.1-1.21-98. Правила безпечної експлуатації електроустановок споживачів. Київ, 1998. 396 с.

23. Правила пожежної безпеки в Україні. Затв. Мін-вом внутріш. справ України 30.12.2014. Чинний від 10.04.2015. Київ: Парламентське видавництво, 2015. 192 с.

24. Правила улаштування електроустановок. Затв. Міненерговугілля України 21.07.2017. Київ, 2017. 617 с.

Додаток А

Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем і програмної інженерії

Кафедра комп’ютерних систем та мереж

“Затверджую”

Завідувач кафедри КС

_____ Осухівська Г.М.

“ 2 ” лютого 2026 р.

Комп’ютерна система керування автобусними пасажирськими перевезеннями

ТЕХНІЧНЕ ЗАВДАННЯ

на _9_ листках

Вид робіт: Кваліфікаційна робота

На здобуття освітнього ступеня «Бакалавр»

Спеціальність 123 «Комп’ютерна інженерія»

«УЗГОДЖЕНО»	«ВИКОНАВЕЦЬ»
Керівник кваліфікаційної роботи	Студент групи СІ-42
_____ к.т.н., ст. викл. Стадник Н. Б.	_____ Микитюк А. І.
“ 2 ” лютого 2026 р.	“ 2 ” лютого 2026 р.

Тернопіль 2026

1 Загальні відомості

1.1 Повна назва та її умовне позначення

Повна назва теми кваліфікаційної роботи: «Комп’ютерна система керування авто бусними пасажирськими перевезеннями».

Умовне позначення кваліфікаційної роботи: КС КРБ 123.187.00.00

1.2 Виконавець

Студент групи СІ-42, факультету комп’ютерно-інформаційних систем і програмної інженерії, кафедри комп’ютерних систем та мереж, Тернопільського національного технічного університету імені Івана Пулюя, Микитюк А. І.

1.3 Підстава для виконання роботи

Підставою для виконання кваліфікаційної роботи є наказ по університету №4/9-188 від 24.04.2026 р.

1.4 Планові терміни початку та завершення роботи

Плановий термін початку виконання кваліфікаційної роботи – 26.01.2026 р.

Плановий термін завершення виконання кваліфікаційної роботи – 21.06.2026 р.

1.5 Порядок оформлення та пред’явлення результатів роботи

Порядок оформлення пояснювальної записки та графічного матеріалу здійснюється у відповідності до чинних норм та правил ISO, ЕСКД, ЕСПД та ДСТУ.

Пред’явлення проміжних результатів роботи з виконання кваліфікаційної роботи здійснюється у відповідності до графіку, затвердженого керівником роботи. Попередній захист кваліфікаційної роботи відбувається при готовності роботи – наявності пояснювальної записки та графічного матеріалу.

Пред’явлення результатів кваліфікаційної роботи відбувається шляхом захисту на відповідному засіданні ЕК, ілюстрацією основних досягнень за допомогою графічного матеріалу.

2 Призначення і цілі створення системи

2.1 Призначення системи

Комп'ютерна система, що розробляється, призначена для:

– Автоматизованого збору телеметрії (кількість пасажирів, GPS-координати) з транспортних засобів.

– Централізованого управління станом посадкових місць та розв'язання конфліктів синхронізації.

– Надання публічного доступу пасажирам до інформації про рейс.

– Надання інструментів диспетчерам для оперативного блокування місць у разі нештатних ситуацій.

2.2 Мета створення системи

Метою кваліфікаційної роботи є розробка та налаштування розподіленої комп'ютерної системи моніторингу пасажиропотоку на основі мікроконтролера ESP32, сервера Node.js та веб-технологій.

2.3 Характеристика об'єкту

Розроблювану систему призначено для використання на підприємствах автотранспортної галузі, автостанціях та безпосередньо в рухомому складі (автобусах).

3 Вимоги до системи

3.1 Вимоги до системи в цілому

3.1.1 Вимоги до структури та функціонування системи

Комп'ютерна система повинна складатись з:

– Апаратного бортового модуля збору телеметрії.

– Серверної частини обробки бізнес-логіки.

– Клієнтського веб-додатку (панель пасажира та панель диспетчера).

3.1.2 Вимоги до способів та засобів зв’язку між компонентами системи

Основна вимога, яка ставиться до способів та засобів інформаційного обміну – це їх узгодженість. Обмін даними повинен здійснюватися бездротовим шляхом з використанням протоколу HTTP та формату JSON.

3.1.3 Вимоги до режимів функціонування системи

Для системи визначено два режими функціонування:

– нормальний режим функціонування (штатний автоматичний збір даних від датчиків);

– аварійний (нештатний) режим функціонування (виникнення поломки сидіння або службового блокування диспетчером).

Основним режимом функціонування є нормальний режим.

Для забезпечення нормального режиму функціонування системи необхідно виконувати вимоги і дотримуватись умов експлуатації програмного забезпечення і комплексу технічних засобів системи.

Аварійний режим функціонування системи характеризується відмовою одного або декількох компонент (наприклад, поломкою сидіння). При цьому функції роботи системи продовжують підтримувати роботу системи моніторингу шляхом ручного втручання диспетчера (блокування місця), що має абсолютний пріоритет над автоматичним розподілом пасажирів.

3.1.4 Вимоги по діагностуванню системи

Для діагностування системи використовуються інструменти діагностування основних процесів системи, які вмонтовані в операційну систему і програмне забезпечення, а також засоби для діагностики апаратного забезпечення.

Інструменти повинні забезпечувати зручний інтерфейс для можливості перегляду діагностичних подій (виведення логів телеметрії у консоль сервера), моніторингу процесу виконання програм.

3.1.5 Перспективи розвитку, проєктування системи

Дана система може бути розширена завдяки використанню додаткових програмних компонентів, а також передбачено розширення бази даних для збільшення кількості маршрутів та підключення нових автобусів.

3.2 Показники призначення

Система повинна передбачати можливість масштабування. Можливості масштабування повинні забезпечуватися засобами використовуваного базового програмного і технічного забезпечення (додавання нових ідентифікаторів busId).

3.2.1 Вимоги до надійності

Система повинна забезпечувати працездатність та відновлення своїх функцій при виникненні наступних ситуацій:

– при збоях в системі електропостачання апаратної частини;

– при помилках в роботі апаратних засобів (наприклад, брязкіт контактів датчиків);

– при помилках, пов'язаних з програмним забезпеченням та втратою інтернет-з'єднання.

Для захисту апаратури від стрибків напруги і комутаційних завад повинні застосовуватися мережні фільтри.

3.3 Вимоги до безпеки

Зовнішні елементи технічних засобів системи, що перебувають під напругою, повинні мати захист від випадкового дотику, а самі технічні засоби мати занулення або захисне заземлення. Бортовий модуль повинен живитися від безпечної напруги.

Система електроживлення повинна забезпечувати захисне вимикання при перевантаженнях і коротких замиканнях в колах навантаження, а також аварійне ручне вимикання.

Загальні вимоги пожежної безпеки повинні відповідати нормам на побутове електрообладнання. У разі пожежі не мають виділятися отруйні гази і дим. Після зняття електроживлення має бути доступне застосування будь-яких засобів пожежогасіння.

3.3.1 Вимоги до експлуатації, технічного обслуговування, ремонту і зберігання компонентів системи

Мікроклімат в приміщеннях (на робочому місці диспетчера) повинен відповідати нормам виробничого мікроклімату по ДСН 3.3.6.042-99:

– температуру повітря в межах від +10°С до +35°С;

– відносну вологість повітря при 25°С в межах від 30% до 80%;

– атмосферний тиск 760±25 мм рт. ст.

Періодичне технічне обслуговування використовуваних технічних засобів має проводитися відповідно до вимог технічної документації, але не рідше ніж один раз на рік. Періодичне технічне обслуговування і тестування технічних засобів повинні включати обслуговування і тестування всіх використовуваних засобів, датчики, контролери, системи передачі даних.

3.4 Вимоги до захисту інформації від несанкціонованого доступу

Система повинна забезпечувати захист від несанкціонованого доступу на рівні не нижче встановленого вимогами, що пред'являються до категорії 1Д по класифікації документа, що діє, “Автоматизовані системи. Захист від несанкціонованого доступу до інформації. Класифікація автоматизованих систем”.

Компоненти підсистеми захисту від НСД повинні забезпечувати:

– ідентифікацію користувача (оператора/диспетчера);

– перевірку повноважень користувача при роботі з системою (доступ до ендпоінту /api/seats/toggle);

– розмежування доступу користувачів (пасажири мають право лише на перегляд даних).

Рівень захищеності від несанкціонованого доступу засобів обчислювальної техніки повинен відповідати нормативним вимогам.

3.4.1 Вимоги по збереженню інформації при аваріях

Інформація (конфігурація салону, логіка заблокованих місць), при виникненні аварійних ситуацій повинна бути збережена на резервних носіях або в енергонезалежній пам'яті сервера.

3.4.2 Вимоги по стандартизації і уніфікації

Система повинна відповідати вимогам ергономіки і зручності користування за умови комплектування високоякісним обладнанням, використання кросплатформного адаптивного веб-інтерфейсу (CSS Grid).

3.4.3 Вимоги до функцій (завдань), що виконуються системою:

– зчитування сигналів з апаратних датчиків (вхід/вихід);

– генерація та передача пакетів телеметрії з борту автобуса;

– вирішення алгоритмічних конфліктів при синхронізації місць;

– забезпечення зручного інтерфейсу для пасажира (табло) та диспетчера (панель керування).

– забезпечення високої швидкодії (асинхронні запити).

4 Вимоги до документації

Документація повинна відповідати вимогам ЄСКД та ДСТУ

Комплект документації повинен складатись з:

– пояснювальної записки;

– графічного матеріалу:

а) Блок-схема алгоритму роботи системи.

б) Структурна схема.

в) Схема електрична принципова.

г) Результат роботи у вигляді веб-інтерфейсу.

5 Стадії та етапи Проєктування

Таблиця 1 – Стадії та етапи виконання кваліфікаційної роботи бакалавра

№ етапу	Назва етапу виконання кваліфікаційної роботи бакалавра	Термін виконання
1	*Розробка технічного завдання*	*26.01 – 02.02*
2	*Робота над першим розділом «Аналіз технічного завдання»*	*03.02 – 15.02*
3	*Робота над другим розділом «Проєктна частина»*	*20.04 – 25.04*
4	*Робота над третім розділом «Практична частина»*	*26.04 – 05.05*
5	*Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»*	*07.05 – 25.05*
6	*Оформлення пояснювальної записки і графічного матеріалу*	*26.05 – 7.06*
7	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*
8	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*
9	*Захист кваліфікаційної роботи бакалавра*	*24.06*

Позн.	*Найменування*	*К-сть*	*Примітка*
	Модулі		
U1	Плата ESP32-DevKitC V4 WROOM-32U	1	
	Мікросхеми		
IC1	Регулятор напруги (LDO) RT9080-33GJ5	1	
U3	Конвертер USB-UART CP2102-GM	1	
U4	Збірка TVS-діодів USBLC6-2P6	1	
U5	Збірка цифрових транзисторів UMH3NTN	1	
	Кнопки		
SW1,SW2	Вимикач кнопковий B3W-1022	2	
SW3,SW4	Вимикач кнопковий 1825910-3	2	

Зм.	*Арк.*	*№ документа*	*Підпис*	*Дата*	*КС КРБ 123.187.00.00 ПЕ*		
Розробив		*Микитюк А. І.*			Комп'ютерна система керування автобусними пасажирськими перевезеннями *Перелік елементів*	*Літ*	*Аркуш* 1 / *Аркушів* 1
Перевірив		*Стадник Н. Б.*					
Рецензент		*Гром'як Р.С.*				ТНТУ СІ-42 м. Тернопіль	
Н. контр.		*Тиш Є. В.*					
Зав. каф.		*Осухівська Г. М.*					

Додаток Б

Лістинги програмного коду комп'ютерної системи

Б.1 Лістинг коду бортового модуля збору телеметрії (C++, мікроконтролер ESP32)

```
#include <WiFi.h>             // Бібліотека для роботи з вбудованим радіомодулем Wi-Fi
#include <HTTPClient.h>     // Бібліотека для реалізації клієнтських запитів за протоколом HTTP

// Налаштування параметрів доступу до віртуальної Wi-Fi мережі у середовищі Wokwi
const char* ssid = "Wokwi-GUEST";
const char* password = "";

// URL-адреса центрального бекенд-сервера (прокинута через зворотний SSH-тунель Serveo)
const char* serverEndpoint = "http://ecaee575da765c0a-217-196-165-208.serveousercontent.com/api/telemetry";

// Конфігурація цифрових портів введення-виведення (GPIO) для підключення датчиків
const int buttonInPin = 12;  // Пін для датчика входу пасажира
const int buttonOutPin = 14; // Пін для датчика виходу пасажира

// Ініціалізація глобальних змінних стану бортової системи
int passengerCount = 0;                // Поточний лічильник пасажирів у салоні
const int maxSeats = 45;                   // Максимальна експлуатаційна місткість автобуса
unsigned long lastSendTime = 0;           // Змінна для зберігання часу останньої відправки даних (у мілісекундах)
const unsigned long sendInterval = 5000; // Інтервал відправки телеметрії (5000 мс = 5 секунд)

// Ініціалізація базових географічних координат для імітації роботи GPS-приймача
float currentLat = 49.5535;
float currentLng = 25.5948;

void setup() {
     // Ініціалізація апаратного UART-інтерфейсу для виведення налагоджувальної інформації (швидкість 115200 бод)
  Serial.begin(115200);

  // Конфігурація пінів датчиків у режим входу з активацією внутрішніх підтягуючих резисторів (усунення наведень)
  pinMode(buttonInPin, INPUT_PULLUP);
```

```
  pinMode(buttonOutPin, INPUT_PULLUP);

  // Процес встановлення з'єднання з бездротовою мережею
  Serial.print("Підключення до Wi-Fi");
  WiFi.begin(ssid, password);

  // Блокуючий цикл очікування успішного підключення до точки доступу
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // Виведення отриманої від DHCP-сервера локальної IP-адреси
  Serial.println("\nWi-Fi підключено успішно!");
  Serial.print("IP адреса ESP32: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  // --- БЛОК 1: ОБРОБКА ДАТЧИКА ВХОДУ ---
  // Перевірка наявності логічного нуля (спрацьовування датчика)
  if (digitalRead(buttonInPin) == LOW) {
      delay(200); // Програмна реалізація усунення брязкоту контактів
(затримка 200 мс)

    // Перевірка на переповнення салону перед інкрементом лічильника
    if (passengerCount < maxSeats) {
      passengerCount++;
      Serial.print("Пасажир зайшов. Поточна кількість: ");
      Serial.println(passengerCount);
    }

    // Блокування виконання коду до моменту відпускання датчика (пасажир
пройшов)
    while(digitalRead(buttonInPin) == LOW);
  }

  // --- БЛОК 2: ОБРОБКА ДАТЧИКА ВИХОДУ ---
  if (digitalRead(buttonOutPin) == LOW) {
    delay(200); // Програмне усунення брязкоту контактів

    // Захист від від'ємного значення пасажирів
    if (passengerCount > 0) {
      passengerCount--;
      Serial.print("Пасажир вийшов. Поточна кількість: ");
      Serial.println(passengerCount);
    }

    // Блокування до звільнення датчика
    while(digitalRead(buttonOutPin) == LOW);
  }

  // --- БЛОК 3: НЕБЛОКУЮЧИЙ ТАЙМЕР ТА ВІДПРАВКА ДАНИХ ---
   // Перевірка чи минув заданий інтервал часу з моменту останньої
відправки
  if (millis() - lastSendTime >= sendInterval) {
    lastSendTime = millis(); // Оновлення часової мітки
```

```
   // Імітація руху автобуса шляхом додавання псевдовипадкового зміщення
до GPS-координат
    currentLat += 0.0002 * (random(-5, 6) / 5.0);
    currentLng += 0.0002 * (random(-5, 6) / 5.0);

       // Перевірка наявності фізичного з'єднання з мережею перед
ініціалізацією HTTP-клієнта
    if (WiFi.status() == WL_CONNECTED) {
      HTTPClient http;

      // Ініціалізація HTTP-з'єднання з центральним сервером
      http.begin(serverEndpoint);

      // Додавання HTTP-заголовка, що вказує на формат даних JSON
      http.addHeader("Content-Type", "application/json");

      // Серіалізація зібраних даних у формат JSON-рядка
      String jsonPayload = "{\"busId\":\"BUS-41-01\",\"passengers\":" +
String(passengerCount) +
                           ",\"lat\":" + String(currentLat, 6) +
                           ",\"lng\":" + String(currentLng, 6) + "}";

      Serial.println("\nВідправка даних: " + jsonPayload);

        // Виконання POST-запиту та отримання коду відповіді сервера
(наприклад, 200 OK)
      int httpResponseCode = http.POST(jsonPayload);

      // Аналіз успішності доставки пакета
      if (httpResponseCode > 0) {
        String response = http.getString(); // Отримання тіла відповіді
        Serial.print("Код відповіді сервера: ");
        Serial.println(httpResponseCode);
        Serial.print("Відповідь: ");
        Serial.println(response);
      } else {
        Serial.print("Помилка відправки запиту, код: ");
        Serial.println(httpResponseCode);
      }

      // Коректне закриття з'єднання та звільнення пам'яті
      http.end();
    } else {
      Serial.println("Помилка: відсутнє підключення до Wi-Fi");
    }
  }
}
```

Б.2 Лістинг серверної частини обробки даних (JavaScript, середовище Node.js)

```
// Підключення необхідних модулів: веб-фреймворку Express та middleware CORS
const express = require('express');
const cors = require('cors');

// Ініціалізація екземпляра сервера та визначення локального порту прослуховування
const app = express();
const PORT = 3000;

// Активація політики міждоменного доступу (CORS) для запитів з веб-браузерів клієнтів
app.use(cors());
// Вбудований middleware для автоматичного парсингу тіла вхідних запитів у форматі JSON
app.use(express.json());

// Глобальний об'єкт стану системи (In-Memory Data Store) для надшвидкого доступу до даних
let busSystemState = {
    busId: "BUS-41-01",
    passengers: 0,              // Загальна кількість недоступних місць (виводиться на публічне табло)
   maxSeats: 45,             // Архітектурна константа місткості салону
    lat: 49.5535,               // Поточна широта
    lng: 25.5948,               // Поточна довгота
   lastUpdate: null,         // Часова мітка останнього синхронізованого пакета
      seats: Array(45).fill(true), // Результуюча матриця сидінь для фронтенду (true = вільно, false = зайнято)

    // --- ЗМІННІ ДЛЯ РОЗВ'ЯЗАННЯ КОНФЛІКТІВ СИНХРОНІЗАЦІЇ ---
    telemetryPassengers: 0,               // Чисте значення лічильника, отримане виключно з апаратних датчиків
       isBlockedByStaff: Array(45).fill(false) // Окремий масив для збереження ручних (службових) блокувань диспетчера
```

```
};

// Алгоритмічна функція розумного перерахунку стану салону з урахуванням пріоритетів
function recalculateBusState() {
      let passengersToPlace = busSystemState.telemetryPassengers; // Залишок пасажирів для розподілу

    // Ітеративний обхід масиву посадкових місць
    for (let i = 0; i < busSystemState.maxSeats; i++) {
            // Пріоритет 1: Перевірка наявності службового блокування (нештатна ситуація)
        if (busSystemState.isBlockedByStaff[i]) {
          busSystemState.seats[i] = false; // Місце примусово виводиться з експлуатації
        }
          // Пріоритет 2: Автоматичний розподіл реальних пасажирів з автобуса на вільні місця
        else if (passengersToPlace > 0) {
          busSystemState.seats[i] = false; // Місце займається фізичним пасажиром
              passengersToPlace--;                // Зменшення залишку нерозподілених пасажирів
        }
        // Місце повністю вільне і готове до використання
        else {
            busSystemState.seats[i] = true;
        }
    }

   // Підрахунок загальної кількості зайнятих/заблокованих місць шляхом фільтрації масиву
     busSystemState.passengers = busSystemState.seats.filter(seat => !seat).length;
}

// --- REST API ЕНДПОІНТИ ---
```

```
// 1. Ендпоінт приймання телеметрії (POST /api/telemetry) від бортового модуля ESP32
app.post('/api/telemetry', (req, res) => {
   // Встановлення спеціального заголовка для проходження через захисний екран Ngrok/Serveo
    res.setHeader('ngrok-skip-browser-warning', 'true');

    // Деструктуризація вхідного JSON-тіла запиту
    const { busId, passengers, lat, lng } = req.body;

    // Валідація наявності критично важливого ідентифікатора
    if (!busId) {
       return res.status(400).json({ status: "error", message: "Missing busId" });
    }

    // Оновлення буферних змінних чистими метриками з датчиків
    busSystemState.telemetryPassengers = passengers;
    busSystemState.lat = lat;
    busSystemState.lng = lng;
      busSystemState.lastUpdate = new Date().toLocaleTimeString(); // Фіксація системного часу

    // Запуск алгоритму перерахунку карти салону на основі нових даних
    recalculateBusState();

    // Виведення логів до консолі сервера для моніторингу
      console.log(`[${busSystemState.lastUpdate}] Телеметрія ${busId}. Пасажирів на борту: ${passengers}. Службових блоків: ${busSystemState.isBlockedByStaff.filter(b => b).length}`);

    // Повернення статусу успішного виконання мікроконтролеру
       res.status(200).json({ status: "success", message: "Telemetry processed successfully" });
});

// 2. Ендпоінт віддачі статусу системи (GET /api/status) для клієнтських веб-додатків
```

```
app.get('/api/status', (req, res) => {
    res.setHeader('ngrok-skip-browser-warning', 'true');
    // Віддача глобального об'єкта стану у форматі JSON
    res.json(busSystemState);
});

// 3. Ендпоінт управління (POST /api/seats/toggle) для диспетчерської панелі автостанції
app.post('/api/seats/toggle', (req, res) => {
     const { seatIndex } = req.body; // Отримання індексу сидіння, яке клікнув диспетчер

    // Валідація коректності індексу (захист від виходу за межі масиву)
    if (seatIndex >= 0 && seatIndex < busSystemState.maxSeats) {
          // Інверсія (перемикання) прапорця службового блокування для вказаного сидіння
                    busSystemState.isBlockedByStaff[seatIndex] = !busSystemState.isBlockedByStaff[seatIndex];

         // Негайний перерахунок загального стану салону для врахування нової конфігурації
         recalculateBusState();

       console.log(`[Диспетчер] Зміна статусу сидіння №${seatIndex + 1}. Поточний стан блокувань: ${busSystemState.isBlockedByStaff.filter(b => b).length} місць.`);

         // Відповідь клієнту з оновленою матрицею сидінь для миттєвого перемальовування інтерфейсу
                return res.json({ status: "success", currentSeats: busSystemState.seats });
    }

    // Повернення помилки 400 Bad Request при невалідному індексі
   res.status(400).json({ status: "error", message: "Invalid seat index" });
});
```

```
// Запуск веб-сервера на вказаному порту
app.listen(PORT, () => {
    console.log(`=====================================================`)
;
    console.log(`Бекенд-сервер системи успішно запущено!`);
    console.log(`Слухає локальний порт: http://localhost:${PORT}`);
    console.log(`=====================================================`)
;
});
```

Б.3 Лістинг коду клієнтського веб-інтерфейсу (HTML/CSS/JS, index.html)

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Система керування перевезеннями | ТНТУ</title>
    <style>
          /* Оголошення глобальних CSS-змінних для кольорової палітри
(Design System) */
        :root {
            --primary: #2b6cb0;
            --success: #48bb78;
            --danger: #f56565;
            --dark: #2d3748;
            --light: #f7fafc;
        }

        /* Базові стилі для тіла документа */
        body {
           font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            background-color: var(--light);
            color: var(--dark);
            margin: 0;
          padding: 20px; /* Зменшено відступ для коректного відображення
*/
        }

        /* Стилізація головного заголовка сторінки (Header) */
        header {
            background-color: var(--primary);
            color: white;
            padding: 20px;
            text-align: center;
            border-radius: 8px;
            margin-bottom: 20px;
        }

       /* Система сіток (CSS Grid) для розташування панелей у дві колонки
*/
        .container {
            display: grid;
            grid-template-columns: 1fr 1fr;
            gap: 20px;
        }

       /* Медіазапит для забезпечення адаптивності на мобільних пристроях
(перехід в одну колонку) */
        @media (max-width: 768px) {
            .container { grid-template-columns: 1fr; }
        }

        /* Стилізація інформаційних карток */
        .card {
            background: white;
            padding: 20px;
            border-radius: 8px;
```

```
            box-shadow: 0 4px 6px rgba(0,0,0,0.1);
        }
       h2 { border-bottom: 2px solid var(--light); padding-bottom: 10px;
margin-top: 0; }

        /* Значки статусів */
        .status-badge {
            display: inline-block;
            padding: 5px 10px;
            background: #e2e8f0;
            border-radius: 4px;
            font-weight: bold;
        }

        /* --- Стилізація інтерактивної схеми салону автобуса --- */
        .bus-layout {
            background: #edf2f7;
            padding: 20px;
            border-radius: 15px;
            border: 3px solid #cbd5e0;
            max-width: 320px;
            margin: 20px auto;
        }

        /* Декоративні елементи коліс автобуса */
        .bus-wheel {
            width: 40px;
            height: 15px;
            background: #4a5568;
            margin: 5px;
            border-radius: 3px;
            display: inline-block;
        }
      .wheels-container { display: flex; justify-content: space-between;
margin-bottom: 15px; }

        /* Створення матриці для посадкових місць (5 колонок) */
        .grid-seats {
            display: grid;
            grid-template-columns: repeat(5, 1fr);
            gap: 10px;
        }

        /* Базовий стиль для вільного сидіння (зелений) */
        .seat {
            height: 40px;
            background-color: var(--success);
            color: white;
            display: flex;
            align-items: center;
            justify-content: center;
            font-size: 12px;
            font-weight: bold;
            border-radius: 6px;
            cursor: pointer;
            transition: all 0.2s ease-in-out;
            user-select: none;
```

```
        }

       /* Модифікатор для зайнятого/заблокованого сидіння (червоний) */
        .seat.occupied { background-color: var(--danger); }

        /* Модифікатор для проходу між рядами сидінь */
        .seat.aisle { background: transparent; cursor: default; }

        /* Легенда кольорів */
        .legend {
            display: flex;
            justify-content: center;
            gap: 15px;
            margin-top: 15px;
        }
      .legend-item { display: flex; align-items: center; gap: 5px; font-
size: 14px; }
        .legend-box { width: 20px; height: 20px; border-radius: 4px; }
    </style>
</head>
<body>

<header>
        <h1>Комп'ютерна система керування автобусними пасажирськими
перевезеннями</h1>
    <p>Розробка студента Микитюка А.І. | Група СІ-42</p>
</header>

<div class="container">
    <div class="card">
    <h2> 🌐 Моніторинг рейсу (Режим пасажира)</h2>
       <p><strong>Автобус:</strong> <span id="bus-id" class="status-
badge">BUS-41-01</span></p>
    <p><strong>Маршрут:</strong> Тернопіль — Львів (Рейс №41)</p>
    <p><strong>Час відправлення:</strong> 14:00</p>
    <p><strong>Орієнтовний час прибуття:</strong> 16:30</p>
       <p><strong>Зайнято місць:</strong> <span id="passenger-count"
style="font-weight: bold; color: var(--danger);">0</span> / 45</p>
     <p><strong>Координати GPS:</strong> <span id="gps-coords">49.5531,
25.5948</span></p>
      <p style="font-size: 12px; color: #a0aec0;">Останнє оновлення
телеметрії: <span id="last-update">-</span></p>

        <hr>
        <h3>Інформаційне табло салону:</h3>
        <div class="legend">
                    <div class="legend-item"><div class="legend-box"
style="background: var(--success);"></div> Вільно</div>
                    <div class="legend-item"><div class="legend-box"
style="background: var(--danger);"></div> Зайнято / Заблоковано</div>
        </div>
    </div>

    <div class="card">
        <h2>🛠️ Панель персоналу автостанції</h2>
```

```
        <p style="color: #718096; font-size: 14px;">Клікніть на будь-яке сидіння на схемі автобуса знизу для оперативного блокування місця у разі нештатної ситуації (поломка, службове бронювання).</p>

        <div class="bus-layout">
            <div class="wheels-container">
                <div class="bus-wheel"></div>
                <div class="bus-wheel"></div>
            </div>
            <div class="grid-seats" id="seats-container"></div>
        </div>
    </div>
</div>

<script>
    // URL адреса локального сервера для звернень API
    const SERVER_URL = 'http://localhost:3000';

    // Асинхронна функція (AJAX) для отримання актуального стану системи з Node.js сервера
    async function updateDashboard() {
        try {
            // Виконання GET-запиту до сервера за допомогою Fetch API
            const response = await fetch(`${SERVER_URL}/api/status`);
            const data = await response.json(); // Десеріалізація JSON-відповіді

           // Оновлення текстових елементів DOM-дерева отриманими даними
            document.getElementById('bus-id').innerText = data.busId;
                document.getElementById('passenger-count').innerText = data.passengers;
                 document.getElementById('gps-coords').innerText = `${data.lat.toFixed(4)}, ${data.lng.toFixed(4)}`;
                   document.getElementById('last-update').innerText = data.lastUpdate || 'Немає даних';

            // Виклик функції для візуального оновлення матриці сидінь
            renderSeats(data.seats);
        } catch (error) {
            console.error("Помилка отримання даних з сервера:", error);
        }
    }

     // Функція для алгоритмічної побудови сітки сидінь (9 рядів по 5 місць)
    function renderSeats(seatsArray) {
        const container = document.getElementById('seats-container');
             container.innerHTML = ''; // Очищення контейнера перед перемальовуванням

        let seatCounter = 0; // Індекс поточного сидіння

        // Цикл генерації рядків (9) та колонок (5) салону
        for (let row = 0; row < 9; row++) {
            for (let col = 0; col < 5; col++) {
             // Логіка створення проходу: центральний стовпчик (індекс 2) залишається порожнім
```

```
                if (col === 2) {
                    const aisle = document.createElement('div');
                    aisle.className = 'seat aisle';
                    container.appendChild(aisle);
                } else {
                    const currentSeatIndex = seatCounter;
                   const isAvailable = seatsArray[currentSeatIndex]; //
Отримання статусу з сервера

                   // Створення DOM-елемента для сидіння та призначення
класів
                    const seatDiv = document.createElement('div');
                        seatDiv.className = `seat ${isAvailable ? '' :
'occupied'}`;
                 seatDiv.innerText = currentSeatIndex + 1; // Нумерація
місця

                    // Прив'язка обробника подій: клік ініціює запит на
блокування місця диспетчером
                                                seatDiv.onclick = () =>
toggleSeatStatus(currentSeatIndex);

                    // Додавання згенерованого місця у контейнер
                    container.appendChild(seatDiv);
                    seatCounter++;
                }
            }
        }
    }

    // Асинхронна функція для відправки диспетчерської команди на сервер
    async function toggleSeatStatus(index) {
        try {
             // Виконання POST-запиту на сервер із зазначенням індексу
обраного сидіння
          const response = await fetch(`${SERVER_URL}/api/seats/toggle`,
{
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ seatIndex: index })
            });
            const data = await response.json();

           // Якщо сервер підтвердив зміну, миттєво перемальовуємо салон
            if(data.status === "success") {
                renderSeats(data.currentSeats);
              updateDashboard(); // Примусова синхронізація лічильників
пасажирів
            }
        } catch (error) {
            console.error("Помилка зміни статусу місця:", error);
        }
    }

    // Реалізація механізму Long-Polling: автоматичне опитування сервера
кожні 2000 мс
    setInterval(updateDashboard, 2000);
```

```
      // Ініціалізація інтерфейсу (первинне завантаження даних) при
відкритті сторінки
    updateDashboard();
</script>
</body>
</html>
```