Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та мехатроніки
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

(тема)
Розробка методу керування електронними пристроями на приладобудівному виробництві у рамках реалізації концепції Індустрії 4.0

Виконав: студент 5 курсу, гр. ІТМРТм-19-1
Мельник Олексій Олегович
(прізвище, ініціали)

Спеціальність
172 – Телекомунікації та радіотехніка
освітньої програми

(код і повна назва напряму)

Тип програми освітньо-професійна
(повна назва освітньої програми)

Керівник доц. Боцман І. В.
(посада, прізвище, ініціали)

Допускається до захисту
зав. кафедри _____ Невлюдов І.Ш.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет	Автоматики і комп'ютеризованих технологій
Кафедра	Комп'ютерно-інтегрованих технологій, автоматизації та мехатроніки
Рівень вищої освіти	другий (магістерський)
Спеціальність	
Тип програми	освітньо-професійна
Освітня програма	

(код і повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____ 2020 р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Мельнику Олексію Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка методу керування електронними пристроями на приладобудівному виробництві у рамках реалізації концепції Індустрії 4.0 затверджена наказом по університету від 02.11. 2020 р. № Ст
2. Термін подання студентом роботи до екзаменаційної комісії . . 2020 р.
3. Вихідні дані до роботи
3.1 Об'єкти керування – електронні пристрої у складі автоматичної лінії;
3.2 Спосіб управління системою – гнучко налаштовуваний;
3.3 Реалізувати метод на основі збору статистики з електроустаткування та візуалізації прийнятих рішень.
4. Перелік питань, що потрібно опрацювати в роботі
4.1 Вступ
4.2 Аналіз предметної області
4.3 Дослідження методів керування електронними пристроями
4.4 Розробка методу керування пристроями у складі автоматичної лінії
4.5 Програмна реалізація запропонованого методу керування електронними пристроями на приладобудівному виробництві
4.6 Вибір заходів з охорони праці на виробництві
4.7 Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Демонстраційний матеріал представлений у форматі презентації PowerPoint (*.ppt) – с. формату А4

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Керівник (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз літератури за темою атестаційної роботи	04.11.2020	виконано
2	Аналіз предметної області	07.11.2020	виконано
3	Дослідження методів керування електронними		виконано
	пристроями	16.11.2020	
4	Розробка програми для управління пристроями		виконано
	у складі автоматичної лінії	18.11.2020	
5	Програмна реалізація запропонованого методу		виконано
	керування електронними пристроями		
6	на приладобудівному виробництві	24.11.2020	виконано
7	Оформлення пояснювальної записки	02.12.2020	виконано
8	Подання роботи у ЕК	10.12.2020	виконано

Дата видачі завдання

Студент _____ Мельник О. О.

(підпис) (прізвище, ініціали)

Керівник роботи _____ доц. Боцман І. В.

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 74 с., 58 рис., 1 табл., 20 джерел, 3 дод.

ІНДУСТРІЯ 4.0, МЕТОД КЕРУВАННЯ, ЕЛЕКТРОННІ ПРИСТРОЇ, ПРИЙНЯТТЯ РІШЕНЬ, ГНУЧКА АВТОМАТИЧНА ЛІНІЯ.

Об’єкт дослідження – процеси керування виробничим обладнанням на приладобудівному підприємстві.

Предмет дослідження – метод автоматизованого керування електронними пристроями на приладобудівному виробництві за допомогою алгоритмів керування електроустаткуванням і візуалізації даних.

Мета роботи – розробка методу керування електронними пристроями на приладобудівному виробництві на основі використання сучасних Інтернет-технологій.

Методи дослідження: теоретичний аналіз, комп’ютерне моделювання, методи автоматичного керування та прийняття рішень.

Проведено аналіз предметної області, зокрема принципів впровадження концепції Індустрія 4.0 на великих приладобудівних підприємствах. Запропоновано метод керування електронними пристроями на приладобудівному виробництві на основі використання сучасних Інтернет-технологій. Зокрема побудовано алгоритм роботи програмного забезпечення для керування електронними пристроями у складі гнучкої автоматичної лінії та визначення надійності її складових. У результаті запропонований метод реалізовано у вигляді програми для керування електронними пристроями та дерева прийняття рішень для керування пристроями у режимі реального часу.

Результати магістерської атестаційної роботи були опубліковані у збірнику матеріалів XIII Всеукраїнської науково практичної WEB конференції аспірантів, студентів та молодих вчених (м. Кривий Ріг, 24-26 березня 2020 р.).

ESSAY

Explanatory note: 74 p., 58 figures, 1 table, 20 sources, 3 applications.

INDUSTRY 4.0, CONTROL METHOD, ELECTRONIC DEVICES, DECISION MAKING, FLEXIBLE AUTOMATIC LINE.

The object of research is the processes of managing production equipment at an instrument-making enterprise.

The subject of research is a method of automated control of electronic devices in instrument-making production using algorithms for controlling electrical equipment and data visualization.

The purpose of the work is to develop a method for controlling electronic devices in instrument-making production based on the use of modern Internet technologies.

Research methods: theoretical analysis, computer modeling, methods of automatic control and decision making.

The analysis of the subject area, in particular, the principles of implementation of the concept of Industry 4.0 at large instrument-making enterprises is carried out. A method for controlling electronic devices in instrument-making production based on the use of modern Internet technologies is proposed. In particular, an algorithm for the operation of software for controlling electronic devices as part of a flexible automatic line and determining the reliability of its components has been built.

As a result, the proposed method is implemented in the form of a program for controlling electronic devices and a decision tree for controlling devices in real time.

The results of the master's attestation work were published in the collection of materials of the XIII All-Ukrainian scientific and practical WEB conference of graduate students, students and young scientists (Kryvyi Rih, 24-26 March 2020).

ЗМІСТ

Перелік скорочень, умовних познак, одиниць і термінів ……. 8
Вступ ……. 9
1 Аналіз предметної області…… 11
1.1 Аналіз завдання на атестаційну роботу…… 11
1.2 Аналіз концепції Індустрія 4.0…… 11
1.3 Хмарні обчислення…… 14
1.4 Інтелектуальні системи прийняття рішень…… 15
1.5 Постановка завдання дослідження ……. 16
2 Розробка програми для управління електронними пристроями 21
2.1 Вибір редактора коду……... 21
2.2 Розробка алгоритму роботи серверної частини програми…….... 22
2.3 Розробка алгоритму клієнтської частини…… 23
2.4 Розробка алгоритму роботи для мікроконтролера……. 24
2.5 Розрахунок надійності системи…… 25
3 Метод керування електронними пристроями на приладобудівному виробництві ……... 27
3.1 Автоматичне керування пристроями на виробничому підприємстві …….. 27
3.2 Програмна реалізації оцінювання надійності автоматизованої системи ……..... 34
3.3 Порядок програмування серверної частини програми……... 37
3.4 Авторизація користувача …… 42
3.5 Розробка API для отримання інформації про пристрої у складі ГАЛ …….. 44

4 Клієнтська частина програми для реалізації запропонованого методу керування електронними пристроями 52
4.1 Програмування клієнтської частини програми..... 52
4.2 Структура програми 54
4.3 Авторизація користувача 56
4.4 Панель керування пристроями 59
4.5 Дані у режимі реального часу 66
5 Охорона праці 68
5.1 Промислова безпека в лабораторії 68
5.2 Виробнича санітарія в лабораторії 68
Висновки 71
Перелік джерел посилання..... 72
Додаток А Код серверної частини програми..... 74
Додаток Б Код клієнтської частини програми..... 94
Додаток В Демонстраційний матеріал..... 134

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ І ТЕРМІНІВ

БД – база даних;

ГАЛ – гнучка автоматична лінія;

ІБР – імовірність безвідмовної роботи.

CSS – cascade style sheets;

Callback – функція зворотного виклику;

HTML – мова гіпертекстової розмітки;

ВСТУП

Четверта промислова революція або Індустрія 4.0 (Industry 4.0) є провідним напрямом розвитку сучасних виробничих технологій.

Характерними рисами Індустрії 4.0 є повністю автоматизовані виробництва, на яких керівництво всіма процесами здійснюється в режимі реального часу та з урахуванням мінливих зовнішніх умов. Кіберфізичні системи створюють віртуальні копії об'єктів фізичного світу, контролюють фізичні процеси та приймають децентралізовані рішення. Вони здатні об'єднуватися в одну мережу, взаємодіяти в режимі реального часу, само налагоджуватися та самонавчатися. Важливу роль у цих процесах відіграють Інтернет-технології, що забезпечують комунікації між персоналом і машинами. Підприємства виробляють продукцію відповідно до вимог індивідуального замовника, оптимізуючи собівартість виробництва.

На даний момент в Україні створено рух Індустрія 4.0 в Україні, велику увагу цим питанням приділяє Асоціація підприємств промислової автоматизації України (АППАУ). На промисловій виставці в Ганновері представники компанії IT-Enterprise відзначили, що модуль Виробництво та інші модулі ERP-системи IT-Enterprise вже вирішують деякі завдання Індустрії 4.0.

Проблеми розвитку Індустрії 4.0 в України полягають у тому, що наразі відсутня чітка спеціалізація та не визначені ті галузі, які будуть розвиватися в новій індустрії на світовій арені, а також у тому, що відсутні інвестиції у цю сферу.

Стан Індустрії 4.0 у світі можна описати, орієнтуючись на наступні країни. В Японії створено Національний інститут просування цифрової економіки і цифрового суспільства (Japan Institute for Promotion of Digital Economy and Community, JIPDEC).

Найбільші компанії США – AT&T, Cisco, GE, IBM і Intely 2014 році створили Консорціум промислового Інтернету (Industrial Internet Consortium

ТМ, ІІС), відкриту некомерційну групу, яка за станом на початок 2017 року об'єднувала вже 250 компаній із 30 країн. Основне завдання Консорціуму – створення екосистеми компаній, наукових центрів і державних структур, сприятливої для впровадження індустріального Інтернету.

Об'єктом дослідження є процеси керування виробничим обладнанням на приладобудівному підприємстві.

Предметом дослідження в даному випадку буде метод автоматизованого керування електронними пристроями на приладобудівному виробництві за допомогою алгоритмів керування електроустаткуванням і візуалізації даних.

Метою атестаційної роботи є розробка методу керування електронними пристроями на приладобудівному виробництві на основі використання сучасних Інтернет-технологій.

Таким чином, у роботі необхідно розробити метод керування електронними пристроями, реалізувати метод прийняття рішень про роботу пристроїв і розробити програму для відображення даних у режимі реального часу.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати предметну область та аналогічні методи керування;

– розробити метод прийняття рішень для керування пристроями;

– запропонувати метод керування електронними пристроями;

– розробити відповідне програмне забезпечення для реалізації запропонованого методу та перевірити його працездатність;

– оформити пояснювальну записку згідно з рекомендаціями [1], вимогами ДСТУ 3008:2015 [2] та згідно з положеннями [3–7].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз завдання на атестаційну роботу

Об’єктом дослідження є процеси керування виробничим обладнанням на приладобудівному підприємстві.

Предметом дослідження в даному випадку буде метод автоматизованого керування електронними пристроями на приладобудівному виробництві за допомогою алгоритмів керування електроустаткуванням і візуалізації даних.

Метою атестаційної роботи є створення автоматизованої системи для керування електроустаткуванням на приладобудівному виробництві на основі використання сучасних Інтернет-технологій.

Для досягнення цієї мети необхідно проаналізувати сучасний стан проблеми та шляхи реалізації концепції Індустрії 4.0 в Україні та світі.

1.2 Аналіз концепції Індустрія 4.0

Підґрунтя Індустрії 4.0 становили три попередні промислові революції. В основі Першої було використання води та пару для механізації виробництва, а також упровадження верстатів і нових технологічних процесів у хімічній і залізорудній промисловості. Результатом Другої промислової революції стала глобалізація, спричинена винайденням електричної енергії, що спричинило зростання масового виробництва та створило основу для вільного руху людей та ідей завдяки розширенню комунальних, телеграфних і залізничних мереж. Третя промислова революція, яку також називають Цифровою революцією, характеризується впровадженням нових цифрових технологій, включаючи персональні комп’ютери, мобільні телефони та Інтернет [8].

Як свідчать вчені, основою Четвертої промислової революції є доступність усієї релевантної інформації в режимі реального часу, що може

бути забезпечено за рахунок поєднання всіх складників ланцюга вартості; Індустрія 4.0 формується на основі кіберфізичних систем виробництва, поєднання дійсної та віртуальної реальності.

Загалом поняття «Індустрія 4.0» можна охарактеризувати як «новий рівень організації та контролю над ланцюгом життєвого циклу товарів, орієнтований на задоволення індивідуальних потреб споживачів. Цей цикл починається з ідеї товару, охоплює розміщення замовлення, розробку товару та його комерційне виробництво, а також постачання товару кінцевим споживачам і завершується утилізацією». Індустрія 4.0 є моделлю, на основі якої світові компанії забезпечують вертикальну інтеграцію «розумних» машин, продуктів і виробничих ресурсів у гнучкі виробничі системи та їх горизонтальну інтеграцію в міжгалузеві мережі цінностей [9].

Концепція впровадження засад Індустрії 4.0 передбачає перехід на повністю автоматизоване цифрове виробництво, кероване інтелектуальними системами у режимі реального часу за постійної взаємодії із зовнішнім середовищем, що виходить за межі одного підприємства, з перспективою об'єднання у глобальну промислову мережу Речей і послуг.

Індустрія 4.0 характеризує поточний тренд розвитку автоматизації та обміну даними, який включає в себе кіберфізичні системи, Інтернет речей і хмарні обчислення. Кіберфізичні системи створюють віртуальні копії об'єктів фізичного світу, контролюють фізичні процеси та приймають децентралізовані рішення. Вони здатні об'єднуватися в одну мережу, взаємодіяти у режимі реального часу, самоналагоджуватися та самонавчатися. Важливу роль відіграють Інтернет-технології, які забезпечують комунікації між персоналом і обладнанням.

Тож, Індустрія 4.0 являє собою новий виток промислової революції, який характеризується інтеграцією виробництва та мережевих комунікацій. Нове покоління технологічного обладнання дозволяє збирати актуальні дані у реальному часі, виготовляти персоналізовані продукти, створюючи прямі зв'язки ланцюжка виробництва від замовлення продукту до отримання його споживачем у

найкоротші терміни з максимальною ефективністю процесу. Зокрема у промисловості планується модернізувати верстати так, щоб вони здатні були оновлювати програми завдяки підключенню до Інтернету, аналізувати власний знос, вдосконалюватися для виконання різноманітних завдань.

Індустрія 4.0 у сучасних умовах дає змогу збирати та аналізувати дані швидше та ефективніше, забезпечуючи при цьому виробництво якісніших товарів за нижчих витрат. Це, у свою чергу, дає змогу підвищити продуктивність виробництва, сприяє зростанню промисловості та змінює профіль робочої сили, внаслідок чого посилюється конкурентоспроможність підприємств і країн. Можна виділити три основні драйвери, що свідчать про доцільність переходу до Індустрії 4.0:

– можливість інтегрувати та краще керувати горизонтальними та вертикальними ланцюгами вартості;

– цифрові технології та взаємозв'язок товарів і послуг (Інтернет речей/послуг);

– нові цифрові бізнес-моделі, що пропонують значну додану цінність для споживачів на основі індивідуальних рішень.

Як показує світовий досвід, Індустрія 4.0 стимулює розвиток нових технологій виробництва, що значно впливають на глобальні виробничі системи рисунку 1.1.

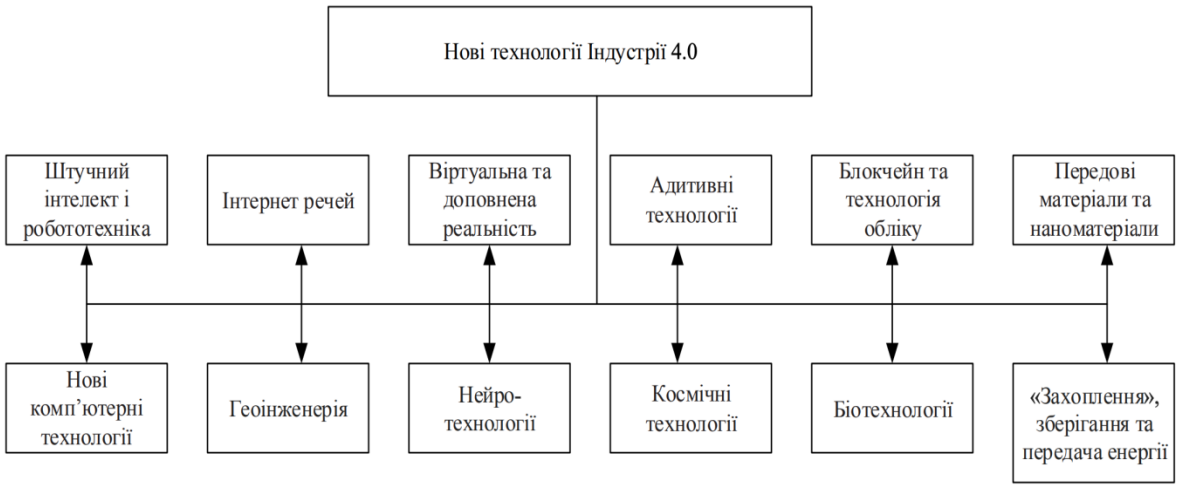


Рисунок 1.1 – Напрями реалізації концепції Індустрія 4.0

1.3 Хмарні обчислення

Невід'ємною частиною Індустрії 4.0 є хмарні обчислення. Хмарні обчислення – це надання різних послуг через Інтернет. Ці ресурси включають інструменти та додатки, такі як зберігання даних, сервери, бази даних (БД), мережі та програмне забезпечення. Хмарні обчислення є популярним варіантом для людей і бізнесу з низки причин, наприклад:

– економія витрат;

– підвищення продуктивності;

– швидкість та ефективність виконання операцій;

– безпека [10].

На рисунку 1.2 наведено сфери, в яких використовують хмарні технології.

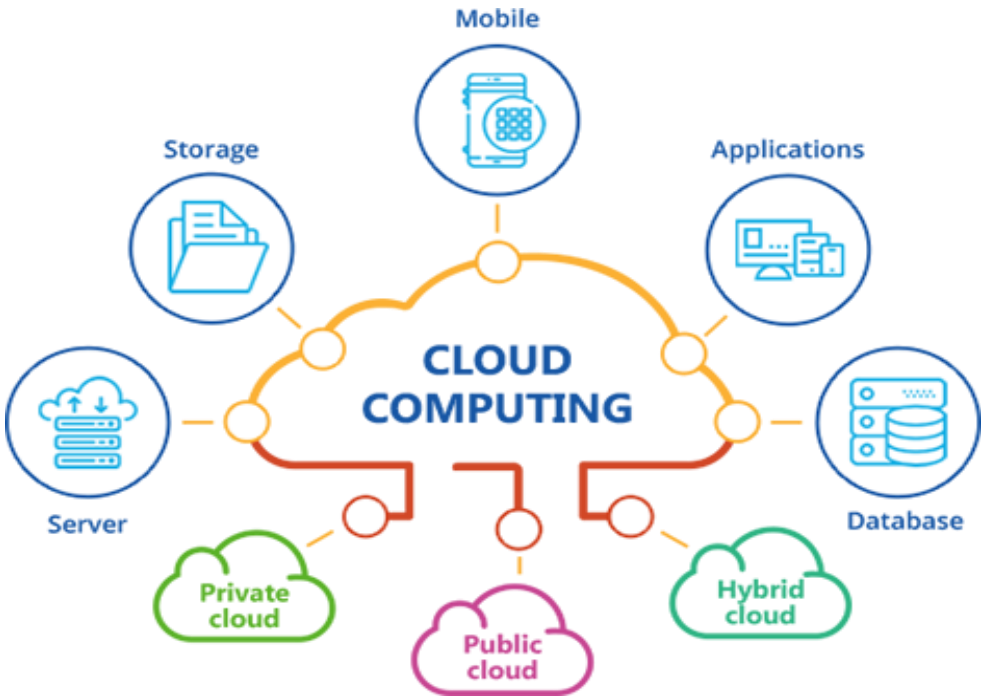


Рисунок 1.2 – Сфери використання хмарних технологій

Реалізація хмарних технологій зводиться до того, що всі операції зберігання, обробки або відправки даних перекладається на серверну частину проекту. Зі сторони клієнта залишається лише взаємодія з вмістом системи.

1.4 Інтелектуальні системи прийняття рішень

Для автоматизації виробництва необхідні інтелектуальні системи прийняття рішень. Для вирішення цієї задачі використовують дерева рішень.

Дерева рішень є одним з найбільш ефективних інструментів інтелектуального аналізу даних та аналітики, які дозволяють вирішувати завдання класифікації та регресії. Вони являють собою ієрархічні деревоподібні структури, що складаються з вирішальних правил виду «Якщо ..., то ...». Правила автоматично генеруються у процесі навчання на навчальній множині. На рисунку 1.3 наведено приклад дерева рішень [11].

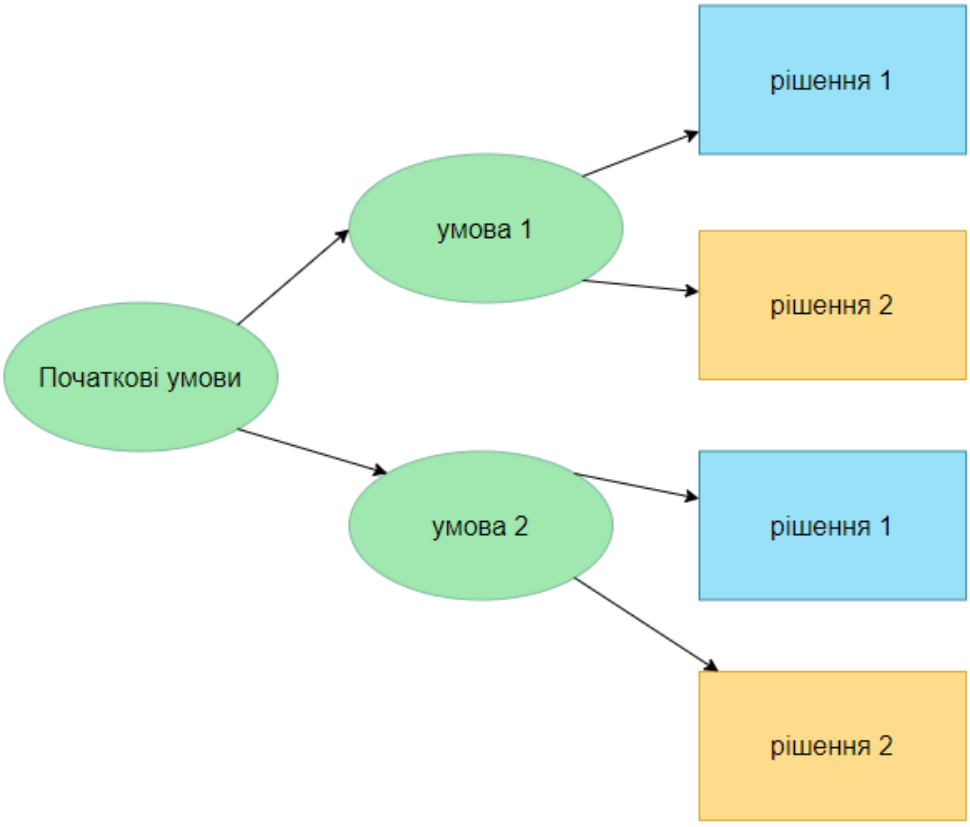


Рисунок 1.3 – Приклад дерева рішень

Наприклад, якщо вночі електроенергія коштує дешевше, то можна збільшити швидкість виробництва. На рисунку 1.4 наведено приклад такого дерева рішень.

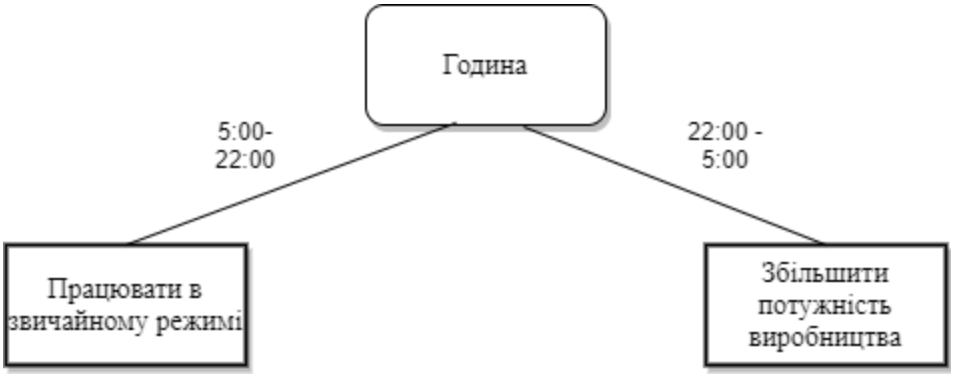


Рисунок 1.4 – Приклад дерева рішень

1.5 Постановка завдання дослідження

Таким чином, на основі проведеного аналізу можна зазначити, що Індустрія 4.0 – найбільша структурна зміна за останні пару сотень років, яка змінює кожен аспект життя й економіки будь-якої країни та галузі. Через свої масштаби і складність, ця трансформація буде не схожа на всі попередні. Нова парадигма поєднує в собі передові технології виробництва й експлуатації з інтелектуальними цифровими технологіями. Все це створюється для того, щоб створити цифрове підприємство, яке буде не тільки взаємопов'язане й автономне, але і зможе обмінюватися даними, аналізувати та використовувати їх для подальшого просування інтелектуальних дій у фізичному світі.

В України є рідкісна можливість вийти на новий рівень. Фактично – це перехрестя, яке визначить майбутнє зростання, процвітання, продуктивності України та її конкурентоспроможності.

Тож, завданням атестаційної роботи буде дослідження методів керування виробництвом у рамках реалізації Індустрії 4.0, а саме автоматизація прийняття рішень і збору статистики з електроустаткування та подальша візуалізація прийнятих рішень.

Для вирішення цього завдання необхідно буде розробити серверну частину та веб-сервіс. Серверна частина буде використовуватися для зберігання, аналізу й оновлення даних у БД, а за допомогою веб-сервісу можна буде керувати обладнанням та візуалізувати дані.

Для написання клієнтської частини був обраний фреймворк React.js та фреймворк Electron.

React-JavaScript – бібліотека для створення користувацьких інтерфейсів. Його головне завдання – забезпечення виведення на екран того, що можна бачити на веб-сторінках. React значно полегшує створення інтерфейсів завдяки розподіленню кожної сторінки на невеликі фрагменти. Такі фрагменти називаються компонентами.

Electron – це фреймворк для розробки настільних додатків із використанням мов HTML, CSS і JavaScript. Такі додатки можуть працювати на різних платформах. Серед них – Windows, Mac і Linux.

В основі Electron лежать проекти Chromium і Node.js, об'єднані в єдине середовище, що забезпечує роботу додатків. Це дає можливість застосовувати веб-технології у процесі розробки настільних програм.

Як платформа для розробки серверної частини було обрано Node.js та фреймворк express.js.

Пакет express – це мінімалістичний і гнучкий фреймворк веб-додатків Node.js, який надає набір функцій для розробки веб і мобільних додатків. Він істотно спрощує розробку веб-додатків на базі Node. Нижче наведені деякі з основних функцій фреймворка Express:

– дозволяє налаштувати посередників для відповіді на запити HTTP;

– визначає таблицю маршрутизації, яка використовується для виконання різних дій на основі методу HTTP і URL-адреси;

– дозволяє динамічно створювати HTML-сторінки на основі передачі аргументів шаблонами;

– інтеграція з базами даних (MySQL, MongoDB, PostgreSQL, Redis та іншими);

– підтримує використання шаблонізаторів, таких як: Pug, EJS, LoDash.

ORM – це інструмент або рівень абстракції, який відображає (перетворює) дані в реляційній БД у програмні об'єкти, якими може маніпулювати програміст за допомогою мови програмування (зазвичай

об’єктно орієнтовані мови). ORM існують виключно для відображення деталей між двома джерелами даних. Обіцянка в Nodejs – це подія, яка дасть результат у майбутньому. Цей результат може бути або успіхом (fulfilled), або невдачею (rejected).

Як ORM-бібліотека використовується Sequelize. Згідно з документацією Sequelize – це ORM на основі обіцянок для Node.js v4 та новіших версій. Вона підтримує діалекти PostgreSQL, MySQL, SQLite та MSSQL і має надійну підтримку транзакцій, відносини, реплікацію читання тощо.

Sequelize, будучи ORM на основі обіцянки, означає, що він підтримує обіцянки NodeJS, використовуючи внутрішньо бібліотеку bluebirdJS (яка є бібліотекою обіцянок NodeJS).

Серед переваг використання ORM систем можна виділити наступні:

– ORM заощаджують час на написання необроблених запитів SQL, тим самим скорочуючи час розробки;

– запити SQL записуються у вигляді рядків і вбудовуються в логіку програми. Під час написання необроблених запитів у програмі дуже важко отримати такі інструменти, як підсвічування синтаксису, автовиправлення або підтримка Intellisence. ORM забезпечують безпечний інтерфейс для взаємодії з базами даних;

– ORM допомагає запобігти ін’єкціям SQL;

– ORM дозволяє писати об’єктно-орієнтований код для своїх моделей та абстрагувати весь процес CRUD, що дозволяє писати декларативний код у своїй програмі [12].

Sequelize містить безліч функцій, нижче наведено перелік ключових переваг цієї ORM системи:

– обіцянки;

– підтримка серверів MySQL,SQLite, PostgreSQL та Microsoft SQL;

– модельні хуки;

– підтримка транзакцій;

– синхронізація бази даних;

– міграції БД;

– перевірка моделі;

– замикання.

Для зберігання даних була обрана база даних PostgreSQL. PostgreSQL – це об’єктно-реляційна система управління БД (СУБД), яка була розроблена в Науковому Комп’ютерному Департаменті Берклі Каліфорнійського Університету. POSTGRES є піонером у багатьох аспектах, які стали доступні в деяких комерційних СУБД набагато пізніше.

PostgreSQL – це продукт із відкритим вихідним кодом, який є нащадком оригінального коду, написаного в Берклі. PostgreSQL підтримує більшу частину стандарту SQL і пропонує безліч сучасних можливостей:

– комплексні запити;

– зовнішні ключі;

– тригери;

– уявлення;

– транзакційна цілісність;

– багатоверсійності управління паралельним доступом.

Також можливості PostgreSQL можуть бути розширені за потребою користувача шляхом додавання нових:

– типів даних;

– функцій;

– операторів;

– агрегатних функцій;

– індексних методів;

– процедурних мов.

Оскільки СУБД PostgreSQL випускається під ліберальною ліцензією, її можна безкоштовно використовувати, модифікувати та поширювати з будь-якою метою, включаючи особисті, комерційні або академічні.

На базі PostgreSQL компанією EnterpriseDB створені потужніші варіанти цієї СУБД, які є платними для комерційного використання – PostgresPlus

(складається цілком тільки з продуктів із відкритими початковими кодами; плата визначається тільки в разі необхідності придбання комерційної підтримки продукту) і PostgresPlusAdvanced Server (розширення PostgreSQL спеціальними можливостями для забезпечення сумісності з OracleDatabase). У комплекті поставки цих товарів міститься великий набір програмного забезпечення для розробників:

– PostgresStudio – більш потужний аналог pgAdmin;

– PostgresPlusDebugger – відладчик для коду на PL/pgSQL, інтегрований із попереднім пакетом;

– MigrationStudio – інструмент для автоматичного перетворення БД з MySQL/Oracle на PostgreSQL.[13]

Для збору даних з пристроїв вирішено використовувати мікроконтролер esp32.

Передбачається, що розроблювана програма буде використовуватися для дистанційного керування різними приладами на виробництві в рамках реалізації Індустрії 4.0. У ході проведеного аналізу було сформовано вимоги до застосовуваного мікроконтролера та програми для нього. До них належать:

– швидкий обмін даними з сервером;

– максимально простий інтерфейс;

– простота інсталювання;

– низька ціна.

2 РОЗРОБКА ПРОГРАМИ ДЛЯ УПРАВЛІННЯ ЕЛЕКТРОННИМИ ПРИСТРОЯМИ

2.1 Вибір редактора коду

VS code – це open-source редактор коду, розроблений корпорацією Microsoft для операційної системи Windows, Linux і macOS. Він підтримує налагодження коду, має вбудовану підтримку системи контролю версій Git і GitHub, підсвічування синтаксису й інтелектуального завершення коду. Він є дуже гнучким, що дозволяє користувачам змінювати тему, комбінації клавіш, налаштування та встановлення розширень, які дають додаткові функціональні можливості редактору.

Вихідний код є вільним для користувача та випускається під ліцензією MIT. Скомпільовані файли є безкоштовними для приватного або комерційного використання.

Код VS code розроблений з допомогою фреймворку Electron, який використовується для розгортання додатків Node.js для десктопів.

Також VS code надає безкоштовний магазин розширень для спрощення розробки програмних продуктів.

Ключові особливості VS code:

– вбудований магазин розширень;

– інтелектуальне завершення коду;

– ES lint;

– система контролю версій;

– вбудований дебагер;

– гарячі клавіші;

– підсвічування синтаксису;

– підтримка роботи розширення emmet для збільшення швидкості написання розмітки HTML;

– вбудований термінал із можливістю відкриття додаткових терміналів і зміни типу терміналу;

– можливість встановлення та створення власних тем.

2.2 Розробка алгоритму роботи серверної частини програми

Серверна частина проєкту працює за наступним алгоритмом (рисунок 2.1). Після отримання нового запиту express порівнює шлях запиту із вже створеними та якщо знаходить відповідність, то передає об’єкт запиту проміжному обробнику, який у свою чергу відправляє дані.

Рисунок 2.1 – Алгоритм роботи серверної частини

2.3 Розробка алгоритму клієнтської частини

Після першого запуску відкривається сторінка авторизації, після успішного входу до свого профілю можна перейти на сторінку зі статистикою або на сторінку керування пристроями. Алгоритм роботи клієнтської частини наведено на рисунку 2.2.

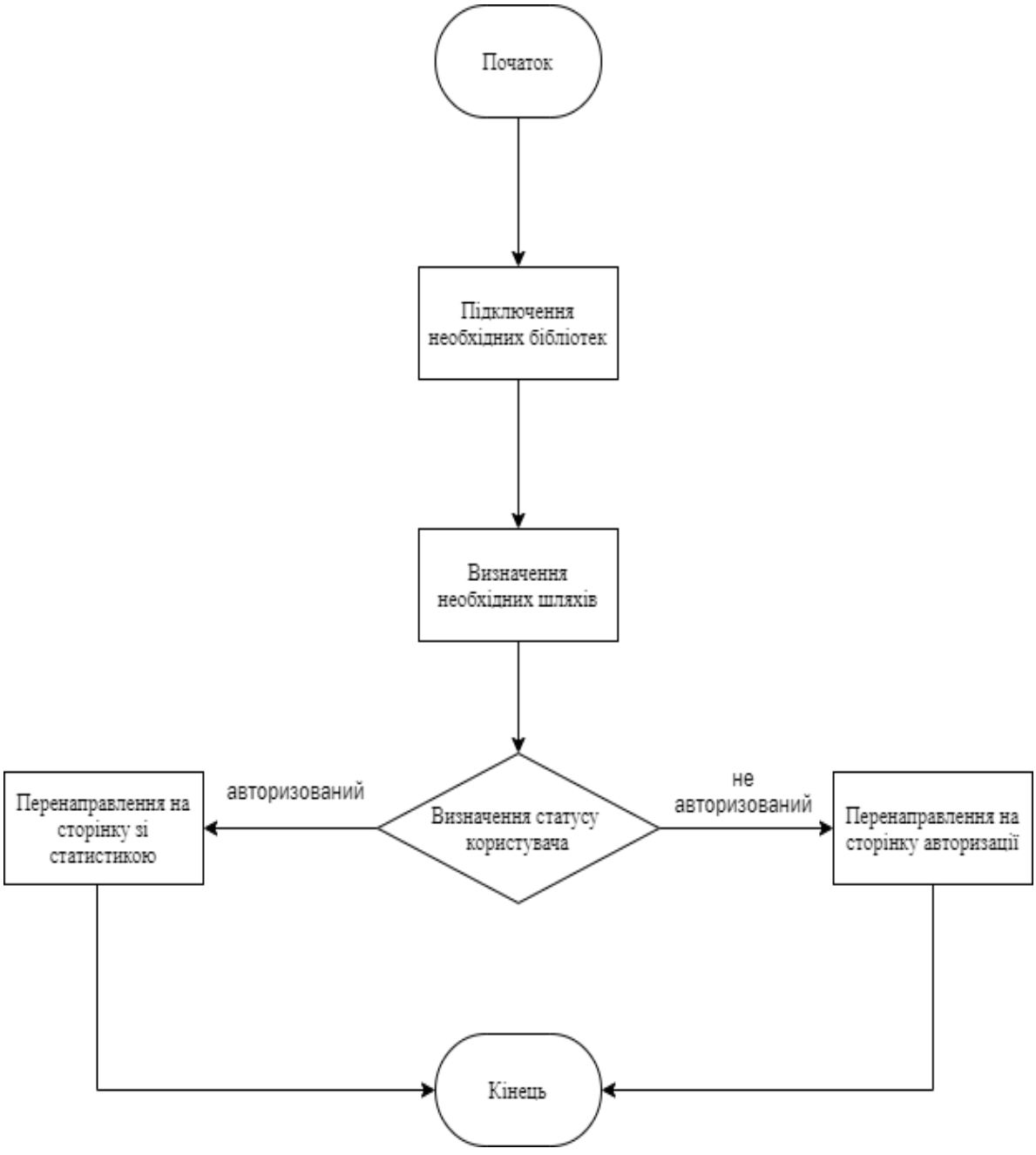


Рисунок 2.2 – Алгоритм роботи клієнтської частини

2.4 Розробка алгоритму роботи для мікроконтролера

Для отримання даних на сервері був розроблений алгоритм оновлення даних і стану мікроконтролера. Через обраний інтервал виконується POST запит до серверу, після чого сервер отримує дані про роботу пристрою, зберігає та обробляє дані. Алгоритм роботи мікроконтролера зображений на рисунку 2.3.

Рисунок 2.3 – Алгоритм роботи мікроконтролера

2.5 Розрахунок надійності системи

Структурна схема для розрахунку надійності будь-якої з підсистем, що входять до складної радіоелектронної системи, розглядається як поєднання послідовно та паралельно з'єднаних елементів.

Послідовне з'єднання відноситься до підсистем і елементів складних систем, що працюють без резерву.

Імовірність виходу з ладу $Ps(t)$ для такої системи відповідно до теорії ймовірностей дорівнює:

$$P_s(t) = \prod_{i=1}^{n} [1 - Pi(t)], \tag{2.2}$$

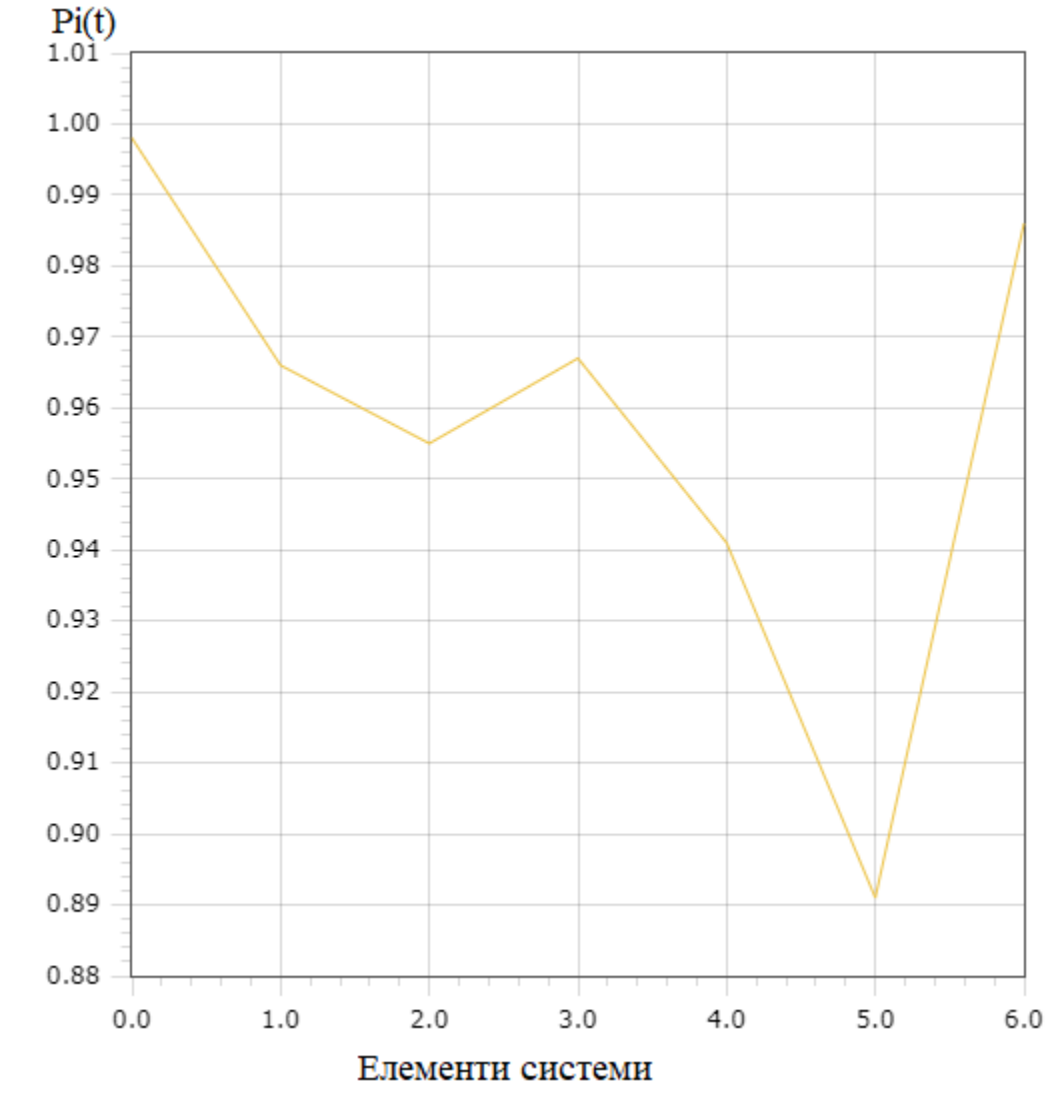
Імовірність безвідмовної роботи (ІБР) системи дорівнює:

$$P^{ПР}s(t) = 1 - \prod_{i=1}^{n} [1 - Pi(t)], \tag{2.3}$$

Наприклад розрахуємо надійність системи за умови послідовного з'єднання елементів, дані надійності елементів наведені в таблиці 2.1. На рисунку 2.5 зображено графік надійності елементів системи

Таблиця 2.1 – Надійність елементів системи

Назва пристрою	ІБР 1	ІБР 2	ІБР 3	ІБР 4	ІБР 5	ІБР 6	ІБР 7
Ймовірність безвідмовної роботи	0,998	0,966	0,955	0,967	0,941	0,891	0,986



.

Рисунок 2.5 – Графік надійності елементів системи

Отже опираючись на дані графіку ми можемо візуально визначити най вразливіший елемент системи.

$$P^{ПР}s(t) = 1-(1-0,998)\times(1-0,996)\times(1-0,995)\times(1-0,967)\times(1-0,941)\times(1-0,986) = 0.736, \quad (3.4)$$

Таким чином, безвідмовність такої системи становить 73,6 %.

Надалі дані про надійність будуть отримуватися з кожного пристрою. Після отримання даних необхідно зберегти їх до бази даних.

3 МЕТОД КЕРУВАННЯ ЕЛЕКТРОННИМИ ПРИСТРОЯМИ НА ПРИЛАДОБУДІВНОМУ ВИРОБНИЦТВІ

3.1 Автоматичне керування пристроями на виробничому підприємстві

У процесі автоматичного регулювання об’єкт управління характеризується одним або декількома керованими параметрами, значення яких можна змінювати. Керовані параметри залежать також від навантажень і перешкод, що діють на об’єкт управління.

Завданням управління об’єктом є забезпечення заданого циклу змін кожного керованого параметра у часі. У найпростішому випадку може, наприклад, вирішуватися завдання стабілізації керованого параметра, тобто автоматична підтримка його значення незалежно від навантажень і перешкод, які впливають на об’єкт.

Управління може здійснюватися як з урахуванням умов функціонування системи, так і без урахування. У другому випадку характеристики управління незмінні, отже, ефективність управління в умовах, що змінилися, може не забезпечуватися. Управління з незмінними характеристиками часто називають жорстким управлінням.

Управління з незмінними характеристиками часто називають регулюванням, а системи, в яких воно здійснюється, – системами автоматичного регулювання. При цьому системи автоматичного управління відносять до систем зі змінними характеристиками управління. Однак даний підхід, за всієї його поширеності, не є загальноприйнятим.

Управління виконується з певною метою. У найпростішому випадку метою управління може бути стабілізація значення керованого параметра, незалежно від навантаження на об’єкт управління та діючих на нього перешкод. Регулятори, що вирішують завдання стабілізації керованого параметра, називають стабілізуючими регуляторами або стабілізаторами.

Наприклад, поширені стабілізатори напруги, стабілізатори температури, стабілізатори тиску та ін.

Метою управління може бути забезпечення деякого заданого закону зміни керованого параметра у часі. Керований параметр у цьому випадку є функцією часу, яка відома та задається задатчиком. Таке управління називають програмним.

Управління може також виконуватися з метою повторення керованим параметром змін задавального впливу, яке заздалегідь є невідомим і формується задатчиком у залежності від зовнішніх впливів. Таке управління називається стежачим управлінням, а системи, в яких воно здійснюється, – стежачими системами.

Серед аналогів систем керування електронними приладами можна виділити шлюз CPX-IoT. Промисловий шлюз Інтернету речей розміщений у стандартному корпусі електричного терміналу CPX. CPX-IoT (рисунок 3.1) та збирає інформацію про пристрої Festo і їх статуси через Ethernet-з'єднання за стандартизованим протоколом зв'язку OPC UA. Потім відправляє цю інформацію на сервер через друге з'єднання Ethernet із використанням таких IoT-протоколів, як AMQP або MQTT. Відповідні механізми інформаційного захисту забезпечують безпеку даних.

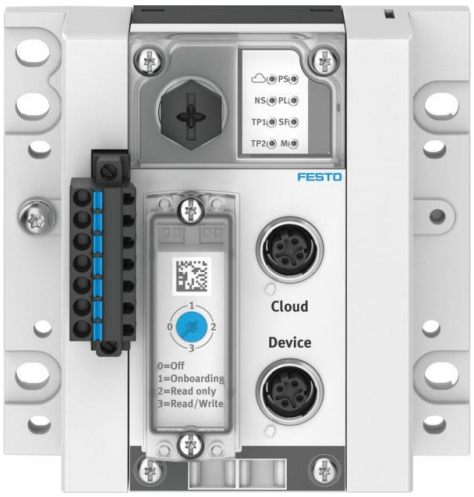


Рисунок 3.1 – Промисловий шлюз CPX-IoT

Таким чином, вироби Festo, зокрема електричні та пневматичні приводи, клапани, пневмоострови, станції віддаленого введення/виведення, блоки підготовки стисненого повітря або датчики можуть за необхідності інтегруватися у підсистеми з традиційної піраміди, скажімо, через децентралізовані контролери, у тому числі CPX або CECC. Або ж вони можуть бути інтегровані безпосередньо, без будь-якої ієрархії, наприклад:

– система переміщення YXMx;
– модуль енергоефективності MSE6-E2M;
– електричний термінал CPX;
– пневмоострови MPA.

Для управління пристроями на виробничому підприємстві в автоматичному режимі запропоновано метод керування пристроями на основі інформації про поточний стан пристроїв у складі гнучкої автоматичної лінії (ГАЛ), який дозволяє здійснити, зокрема:

– прийняття рішень про встановлення стану пристрою у складі ГАЛ;
– управління потужністю пристрою на основі зовнішніх факторів.

У загальному випадку досліджуваний технічний об’єкт кількісно можна охарактеризувати векторами зовнішніх, внутрішніх і вихідних параметрів відповідно:

$$x \in R^k, q \in R^m, y \in R^n,$$

Одні і ті ж фізичні, механічні або інформаційні характеристики у моделях різного рівня та змісту можуть виконувати роль як зовнішніх або внутрішніх, так і вихідних параметрів.

Для більшості випадків абстрактна модель системи керування пристроями може бути представлена за допомогою рисунку 3.2. Система не існує сама по собі, а виділяється з навколишнього середовища за якою-небудь системоутворюючою ознакою, в якості якого найчастіше виступає мета

системи. Взаємодія системи із зовнішнім середовищем здійснюється через вхід і вихід системи (безліч вхідних і вихідних параметрів).

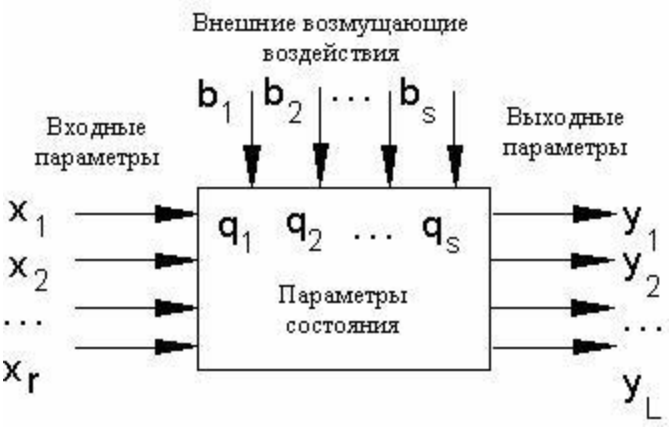


Рисунок 3.2 – Абстрактна модель системи керування пристроями

Під вхідними параметрами системи розуміється комплекс параметрів зовнішнього середовища (у тому числі вихідні параметри систем, зовнішніх по відношенню до даної, наприклад, систем управління), які суттєво впливають на стан і значення вихідних параметрів даної системи та підлягають обліку й аналізу засобами, наявними у розпорядженні дослідника.

Вихідні параметри – це комплекс параметрів системи, які безпосередньо впливають на стан зовнішнього середовища та є значущими з точки зору мети дослідження.

Важливою особливістю функціонування складних систем є принципова невизначеність істинного стану зовнішнього середовища у кожен момент часу. Природа цієї невизначеності пов'язана з наявністю ряду причин, найважливіші з яких обумовлені наступними факторами:

– про деякі, можливо, безпосередньо впливаючі на поведінку системи параметри зовнішнього середовища (тобто параметри, які слід було б віднести до категорії вхідних), дослідник часто не знає, і, отже, не може їх враховувати;

– деякі параметри зовнішнього середовища не можуть бути виміряні у силу технічної непристосованості інформаційних засобів;

– чисельні значення врахованих параметрів оцінюються з помилками вимірювань, які визначаються, з одного боку, внутрішніми шумами вимірювальних пристроїв, а з іншого – зовнішніми перешкодами.

Вплив на систему подібних неврахованих факторів компенсується введенням у модель додаткових зв'язків – зовнішніх обурювальних впливів або шумів.

Система може знаходитися у різних станах. Стан будь-якої системи у певний момент часу можна з певною точністю охарактеризувати сукупністю значень параметрів стану q.

Таким чином, система характеризується трьома групами змінних:

– вхідними змінними, які генеруються системою:

$$\overline{x} = x_1, x_2, \ldots x_n,$$

– вихідними змінними, що визначають вплив досліджуваної системи на навколишнє середовище:

$$\overline{y} = y_1, y_2, \ldots y_n,$$

– параметрами стану (внутрішніми параметрами), що характеризують динамічну поведінку досліджуваної системи:

$$\overline{q} = q_1, q_2, \ldots q_n,$$

У порівняно простому випадку математична модель може являти собою співвідношення:

$$y = \int (x,1); x \in R^k; q \in R^m; y \in R^n,$$

де $\int$ – векторна функція векторного аргументу.

Теоретичний шлях побудови математичної моделі полягає у встановленні зв'язку між *y*, *x* і *q* у вигляді операційного рівняння:

$$L(u(z)) = 0,$$

де L – деякий оператор (у загальному випадку нелінійний);

u – вектор фазових змінних, що включає ті параметри, які характеризують його стан;

z – вектор незалежних змінних, у загальному випадку включає час і просторові координати;

0 – нульовий елемент простору, в якому діє цей оператор.

На рисунку 3.3 зображено графік залежності продуктивності системи від часу доби. На осі абсцис відкладено час доби, а на осі ординат – потужність виробничого пристрою у складі ГАЛ.

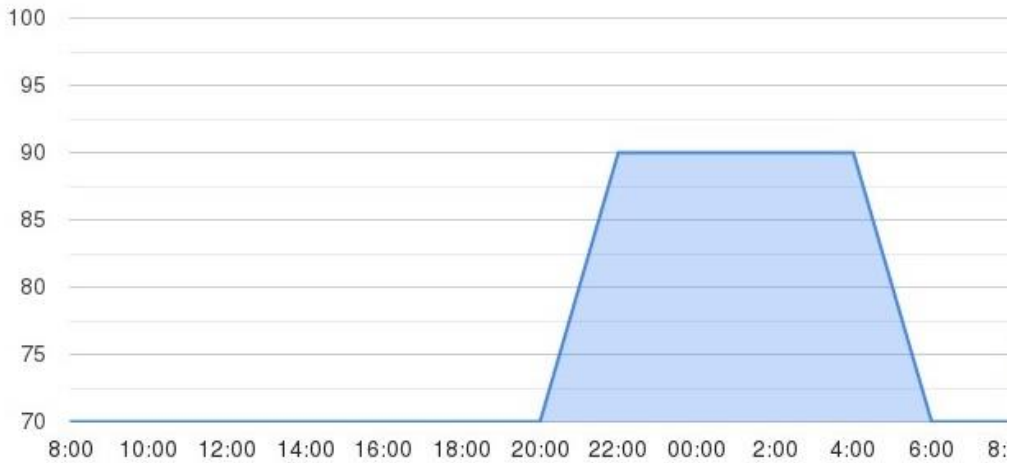


Рисунок 3.3 – Графік залежності потужності системи від часу доби

Метод керування електронними пристроями у складі ГАЛ реалізується на основі розробленого дерева рішень, алгоритм роботи якого наведено на рисунку 3.4.

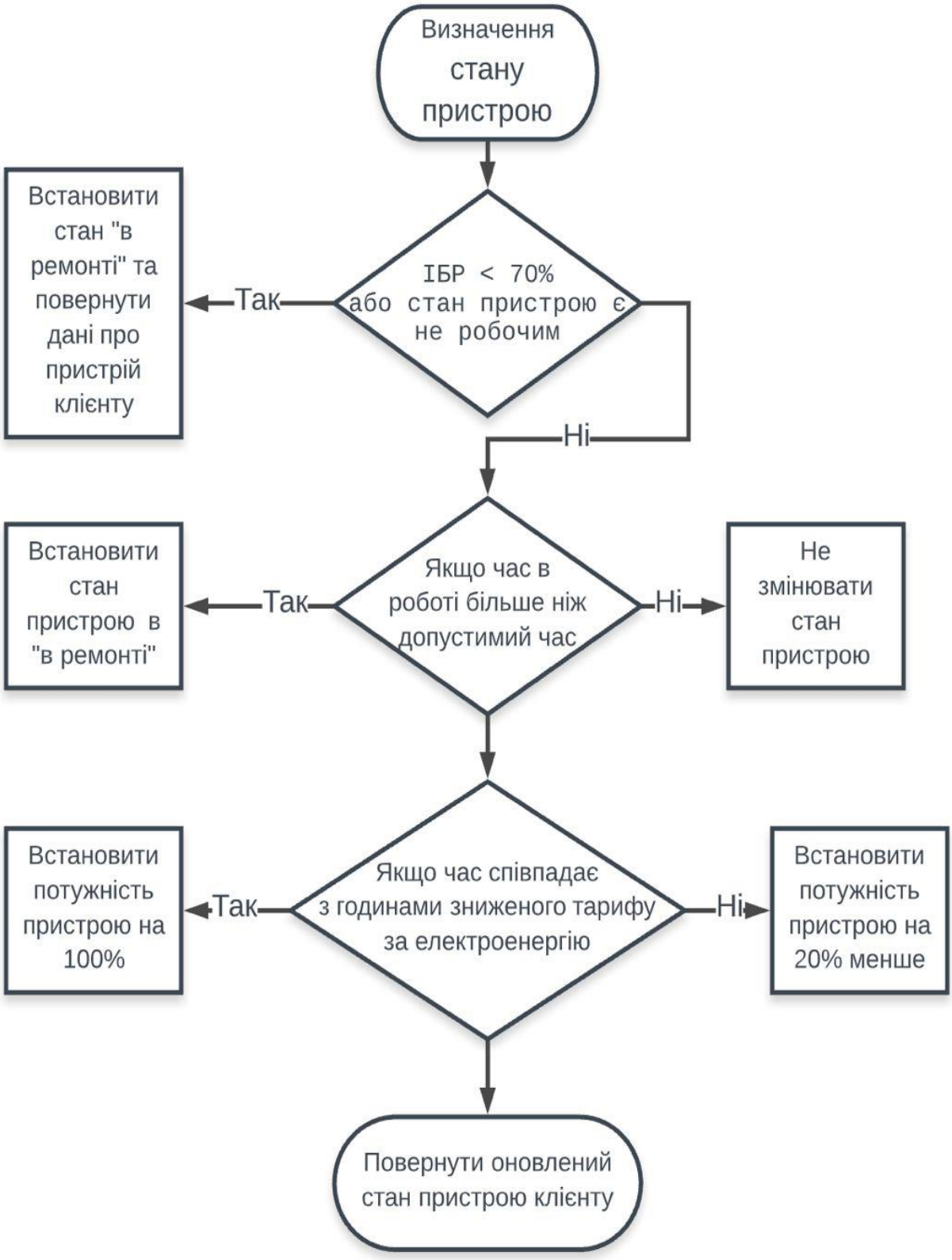


Рисунок 3.4 – Алгоритм прийняття рішень про роботу пристрою у складі ГАЛ

3.2 Програмна реалізації оцінювання надійності автоматизованої системи

Розрахунок надійності являє собою процедуру визначення значень показників надійності об’єкта з використанням методів, заснованих на їх обчисленні за довідковими даними про надійність елементів об’єкта, за даними про надійність об’єктів-аналогів, даними про властивості матеріалів та іншої інформації, наявної на момент розрахунку.

У результаті розрахунку визначаються кількісні значення показників надійності.

Імовірність безвідмовної роботи (ІБР) елемента, агрегату, системи – це ймовірність того, що у межах заданого напрацювання або заданого інтервалу часу відмова об’єкта не відбудеться.

Надійність – властивість об’єкта зберігати у часі у встановлених межах значення всіх параметрів, що характеризують здатність виконувати необхідні функції у заданих режимах і умовах застосування, технічного обслуговування, зберігання та транспортування [14].

Існує два способи підключення пристроїв до системи керування – паралельний і послідовний. Приклад паралельного з’єднання пристроїв зображено на рисунку 3.5.

Програмна реалізація розрахунку імовірності безвідмовності системи із паралельним з’єднанням наведена на рисунку 3.6.

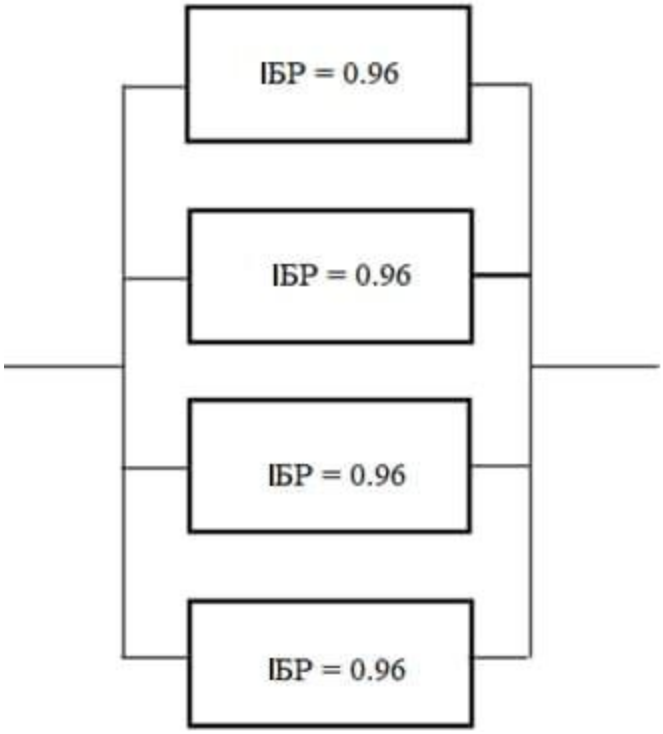


Рисунок 3.5 – Приклад системи з паралельним з’єднанням пристроїв ІБР

```
const countVBR = (deviceVBR: number[]) => {
  const totalVBR = deviceVBR.reduce((acc, vbr) => acc * (1 - vbr / 100));

  return 1 - totalVBR;
};
```

Рисунок 3.6 – Реалізація розрахунку надійності системи із паралельним з’єднанням елементів

Системою із послідовним з’єднанням елементів називається система, в якій відмова будь-якого елемента призводить до відмови всієї системи. Таке поєднання елементів у техніці зустрічається більш часто, тому його називають основним з’єднанням.

У системі з послідовним з’єднанням для безвідмовної роботи протягом деякого напрацювання t необхідно і досить, щоб кожен з її n елементів працював безвідмовно протягом часу напрацювання. Приклад системи з послідовним з’єднанням наведено на рисунку 3.7.

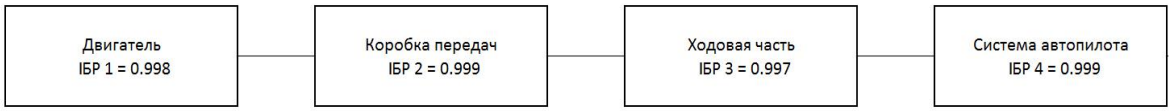


Рисунок 3.7 – Приклад системи з послідовним з'єднанням елементів ІБР

У цьому випадку для розрахунку надійності системи використовується наступна формула:

$$Ps(t) = \prod_{i=1}^{n} [Pi(t)],$$

де Pi(t) – імовірність безвідмовної роботи і-го елемента.

Реалізація розрахунку імовірності безвідмовності для системи з послідовним з'єднанням наведена на рисунку 3.8.

```
const countParalelVBR = (deviceVBR: number[]) => {
  return deviceVBR.reduce((acc, vbr) => acc * (vbr / 100));
};
```

Рисунок 3.8 – Реалізація розрахунку імовірності безвідмовності для системи з послідовним з'єднанням елементів

Розрахунок надійності системи для паралельного з'єднання елементів у разі, якщо відома інтенсивність відмов елементів (failure rate), кількість цих елементів і час роботи системи, виконується за формулою:

$$Pi(t) = \exp(-\Delta t),$$

де Δ – це інтенсивність відмов.

Реалізація розрахунку надійності системи, якщо відома інтенсивність

відмов, наведена на рисунку 3.9.

```
const countDeviceRejectIntensity = (intensity: number, time: number) => {
  return Math.exp(-intensity * time);
};
```

Рисунок 3.9 – Реалізація розрахунку надійності системи у випадку відомої інтенсивності відмов

3.3 Порядок програмування серверної частини програми

Проаналізувавши завдання та визначивши функції веб-сервісу, які потрібно реалізувати, можна приступити до написання коду програми. Спочатку необхідно оголосити всі необхідні бібліотеки, змінні та глобальні функції. Бібліотеки для подальшої реалізації функцій веб-сервісу наведені на рисунку 3.10.

```
"dependencies": {
  "@types/bcrypt": "^3.0.0",
  "@types/body-parser": "^1.19.0",
  "@types/cors": "^2.8.8",
  "@types/express": "^4.17.8",
  "@types/jsonwebtoken": "^8.5.0",
  "@types/morgan": "^1.9.2",
  "@types/socket.io": "^2.1.11",
  "bcrypt": "^5.0.0",
  "body-parser": "^1.19.0",
  "cors": "^2.8.5",
  "crypto": "^1.0.1",
  "dotenv": "^8.2.0",
  "express": "^4.17.1",
  "express-validation": "^3.0.6",
  "joi": "^17.3.0",
  "jsonwebtoken": "^8.5.1",
  "morgan": "^1.10.0",
  "pg": "^8.4.2",
  "pg-hstore": "^2.3.3",
  "sequelize": "^6.3.5",
  "socket.io": "^3.0.3",
  "socket.io-client": "^3.0.3",
  "uuid": "^8.3.1"
},
```

Рисунок 3.10 – Модулі програми

Node.js – це середовище для серверної розробки на мові JavaScript. Воно є кросплатформеним і має відкритий вихідний код. Використовується здебільшого для написання веб-серверів, але має і багато інших можливостей.

З'явившись у 2009 році, коли JavaScript вже почав вважатися серйозною мовою, Node завоювала велику популярність і фактично стала лідером у сфері веб-розробки.

Переваги мови Node js:

– платформа передбачає власні інструменти та особливості, наприклад, тут немає браузерних API, coockie або DOM, зате присутні власні бібліотеки та інші рішення. Але в основному використовуються можливості та синтаксис всім звичної мови JavaScript;

– багата стандартна бібліотека. Платформа від самого початку мала широкий набір можливостей, а в нових версіях бібліотека поповнюється та поліпшується;

– велика кількість зовнішніх бібліотек і готових модулів. Використання пакетного менеджера NPM дозволяє постійно розвивати екосистему Node. Сьогодні число опенсорсних пакетів у ньому вже більше за 500 тисяч і постійно зростає;

– двигун V8. Мова JavaScript створювалась як інтерпретована скриптова мова. Але процес її інтерпретації не є швидким і простим. При цьому мова розвивається, вона давно стала повноцінною, на JavaScript можна писати великі програми. А тому наявність компілятора стала не просто плюсом, але – необхідністю [15].

Розглянемо пакетний менеджер npm (node package manager). За допомогою npm можна встановлювати пакети локально або глобально. У локальному режимі пакети встановлюються в каталог node_modules батьківського каталогу. Власником каталогу є поточний користувач. Глобальні пакети встановлюються у каталог {prefix} / lib / node_modules /, власником якого є root (префіксом у даному випадку зазвичай є каталог / usr / або / usr /

local). Для глобального встановлення пакету використовується прапор --global, його можна записувати скорочено -g. Майже всі пакети були встановлені за допомогою npm, для цього використовується команда npm install або npm для скорочення [16].

Пакет express – це мінімалістичний і гнучкий фреймворк веб-додатків Node.js, який надає набір функцій для розробки веб і мобільних додатків. Він істотно спрощує розробку веб-додатків на базі Node. Нижче наведені деякі з основних функцій фреймворка Express:

– дозволяє налаштувати посередників для відповіді на запити HTTP;

– визначає таблицю маршрутизації, яка використовується для виконання різних дій на основі методу HTTP і URL-адреси;

– дозволяє динамічно створювати HTML-сторінки на основі передачі аргументів шаблонами;

– інтеграція з базами даних (MySQL, MongoDB, PostgreSQL, Redis та іншими);

– підтримує використання шаблонізаторів таких як: Pug, EJS, LoDash.

Для шифрування паролю користувача використовується модуль bcrypt.

Bcrypt – це функція хешування паролів, розроблена Niels Provos і David Mazières, заснована на шифрі Blowfish і представлена в USENIX у 1999 році.

Bcrypt є адаптивною функцією: з плином часу кількість ітерацій може бути збільшена, щоб зробити її більш повільною, тому вона залишається стійкою до атаки з використанням brute-force навіть за умови збільшення потужності обчислення.

Характерною ознакою для bcrypt є префікс "$2a$", "$2b$" або "$2y$" у хеш-рядку, у файлі тіньового пароля вказує, що хеш-рядок – це bcrypt хеш у модульному форматі шифрування. Інша частина хеш-рядка включає в себе параметр вартості, 128-бітну salt (Radix-64 кодується як 22 символи) і 184 біти результуючого хеш-значення (Radix-64 кодується як 31 символ). Кодування Radix-64 використовує алфавіт unix / crypt і не є стандартним для кодування у форматі base64 [17].

Для отримання доступу до змінних оточення оточення був використаний модуль dotenv. Цей модуль дозволяє отримувати змінні оточення або .env файлу у вигляді об'єкта та потім використовувати їх у програмі, у таких файлах зазвичай вказується порт для прослуховування програмою, хост, протокол передачі даних, URL посилання на інші веб-сервіси, приватні ключі та інші налаштування програми. Приклад .env файлу наведено на рисунку 3.11.

```
DATABASE=database
DATABASE_USER=user
DATABASE_PASSWORD=password
PORT=5001
NODE_ENV=dev
#one year
REFRESH_TOKEN_LIFE=977616000
```

Рисунок 3.11 – Приклад .env файлу

Після ініціалізації .env файлу в об'єкті process.env з'являться всі змінні для подальшого використання їх для налаштування роботи серверної частини програми.

Модуль body-parser дозволяє отримувати тіло запиту для POST запитів у вигляді об'єкту, доступ до якого можна отримати через req.body.

Пакет cors надає можливість отримувати запити від додатків зі сторонніх програм, він встановлює необхідні заголовки для обміну даними.

Валідація та створення токенів реалізована з використанням модулю jsonwebtoken. JSON Web Token (JWT) – це JSON об'єкт, який визначений у відкритому стандарті RFC 7519. Він вважається одним із найбезпечніших способів передачі інформації між двома учасниками. Для його створення необхідно визначити заголовок (header) із загальною інформацією по токену, корисні дані (payload), такі як id користувача, його роль і т. д. і підписи (signature) [18].

Для відловлювання помилок та логування запитів використовується

модуль morgan, він дозволяє налаштувати формат виводу логів, встановити, які запити потрібно логувати та вибрати статус-коди. Також можна записувати логи до файлу.

Для розробки серверної частини програми було вирішено використовувати проміжні функції (middleware). Вони надають можливість виконувати валідацію запиту, додавати до запиту необхідні параметри і т. д.

Проміжні функції (middleware) – це функції, які мають доступ до об'єкта запиту (req), об'єкта відповіді (res) і до наступної функції тимчасової роботи у циклі запит-відповідь додатка. Перехід до наступної проміжної функції можна виконати за допомогою функції next.

Проміжні функції можуть виконувати такі завдання:

– виконання будь-якого коду;
– внесення змін до об'єктів запитів і відповідей;
– завершення циклу запит-відповідь;
– виклик наступної функції тимчасової роботи зі стека.

Якщо поточна функція тимчасової роботи завершує цикл запит-відповідь, вона повинна викликати next для передачі управління наступної функції тимчасової роботи. В іншому випадку запит не буде передано до наступного обробника.

3.4 Авторизація користувача

У головному файлі серверу було визначено два шляхи для запитів, вони вказують, які проміжні обробники використовуються для вказаних шляхів. Шляхи проміжних обробників реалізовані у вигляді модулів. На рисунку 3.12 зображено використання проміжних обробників.

```
app.use('/api/auth', authRote);
app.use('/api/device', deviceRoute);
```

Рисунок 3.12 – Використання проміжних обробників для обробки запиту

У проміжному обробнику authRoute міститься два роути – для реєстрації та авторизації користувача. На рисунку 3.13 зображено роути авторизації.

```
router.post('/register', registerValidate, async (req, res, next) => {…

router.post('/login', loginValidate, async (req, res) => {…
```

Рисунок 3.13 – Роути авторизації корситувача

Реєстрація користувача починається з валідації запиту, тіло запиту спочатку потрапляє до обробника registerValidate. Валідатор передає тіло запиту схемі валідації, після чого отримує об’єкт із можливими помилками, якщо в об’єкті знаходяться помилки, то валідатор повертає статус «код 400» та повідомлення з описанням про помилку. Валідація запиту та схема валідації зображені на рисунку 3.14.

```
export const registerValidate = (req: Request, res: Response, next: () => void) => {
  const { error } = registerSchema.validate(req.body);

  if (error)
    return res.status(400).json({
      error: error.details[0].message,
    });

  next();
};
```

а)

```
export const registerSchema = joi.object({
  userName: joi.string().min(4).max(16).alphanum().required(),
  email: joi.string().email().required(),
  password: joi.string().pattern(new RegExp('^[a-zA-Z0-9]{6,30}$')),
});
```

б)

Рисунок 3.14 – Валідація запиту (а) та схема (б)

Якщо валідація пройшла успішно, то наступним кроком буде виконаний пошук користувача у БД за email, якщо аккаунт на таку електронну пошту вже зареєстрований, то буде встановлений статус код 400 та буде відправлено

повідомлення про помилку.

Якщо такий email ще не зареєстровано, то за допомогою модулю bcrypt буде зашифровано пароль, після чого виконається запит до БД та до таблиці User буде додано новий запис. З результату виконання запиту до БД можна отримати uuid (унікальний id користувача), у свою чергу uuid користувача буде додано в токен із підписом та відправлено на клієнтську частину програми. Реєстрація користувача зображена на рисунку 3.15.

```
router.post('/register', registerValidate, async (req, res, next) => {
  const isExist = await User.findOne({
    where: {
      email: req.body.email
    }
  });

  if (isExist) return res.status(400).json({error: 'user already in DB'});

  const hash = await bcrypt.hash(req.body.password, 10).then(hash => hash);

  req.body.password = hash;

  const { uuid } = await User.create(req.body) as unknown as IUser;

  res.json({token: createJWT({ uuid }) })

  next()

})
```

Рисунок 3.15 – Реєстрація користувача

Вхід користувача до особистого аккаунту починається з валідації тіла запиту. За допомогою loginSchema виконується валідація паролю та електронної пошти користувача, якщо схема повертає помилку, то у відповідь встановлюється статус код 400 та відправляється повідомлення про помилку.

Якщо валідація пройшла успішно, то з тіла запиту за допомогою деструктуризації дістаються пароль та пошта користувача. Після отримання даних виконується пошук користувача за email, якщо користувач з таким email вже знаходиться у БД, то перевіряється пароль, отриманий з тіла запиту із

зашифрованим паролем, отриманим після запиту до БД. У випадку, якщо паролі співпадають, користувачу відправляється токен, а якщо паролі не співпадають, то у відповідь встановлюється статус код 400 та відправляється повідомлення про помилку. Якщо користувач не був знайдений, то встановлюється статус код 404 та відправляється повідомлення про помилку. Авторизація користувача зображена на рисунку 3.16.

```
router.post('/login', loginValidate, async (req, res) => {
  const { email, password } = req.body;

  const user = await User.findOne({
    where: {
      email
    }
  }) as unknown as IUser

  if (user) {
    const compare = await bcrypt.compare(password, user.password).then(result => result);

    if (compare) {
      return res.json({token: createJWT({ uuid: user.uuid}) })
    }

    return res.status(400).json({error: 'wrong password'});
  }

  res.status(404).json({ error: 'user not found' });

})
```

Рисунок 3.16 – Авторизація користувача

3.5 Розробка API для отримання інформації про пристрої у складі ГАЛ

Для обробки запитів для отримання інформації про електричний прилад було розроблено модуль deviceRoute. За допомогою даного модулю можна отримати кількість пристроїв, загальну статистику, дані щодо кожного пристрою, а також є можливість керування пристроями. На рисунку 3.17 зображено роути керування пристроями.

```
router.post('/create', deviceValidation, async (req, res) => {…
router.get('/count', async (req, res) => {…
router.get('/data-all', async(req, res) => {…
router.get('/get-data/:uuid', async (req, res) => {…
router.get('/info/:uuid', async (req, res) => {…
router.get('/list', async (req, res) => {…
router.get('/avg-data/:type', async (req, res) => {…

router.post('/update-state', deviceInfoValidation, async (req, res) => {…
```

Рисунок 3.17 – Роути для керування пристроями

Перш за все необхідно додати пристрій до БД, цей функціонал реалізовано за допомогою роуту /create. Під час отримання запиту виконується валідація тіла запиту. На рисунку 3.18 зображено схему та валідацію запиту.

```
export const deviceValidation = (req: Request, res: Response, next: () => void) => {
  const { error } = deviceSchema.validate(req.body);

  if (error)
    return res.status(400).json({
      error: error.details[0].message,
    });

  next();
};
```

а)

```
export const deviceSchema = joi.object({
  name: joi.string().min(4).max(500).required(),
  performance: joi.number().required(),
});
```

б)

Рисунок 3.18 – Валідація (а) та схема валідації (б)

Якщо всі дані пройшли валідацію, то за допомогою моделі Device виконується

метод Device.create(req.body), який записує дані про пристрій у БД і відправляє результат користувачу. Додавання пристрою зображено на рисунку 3.19.

```
router.post('/create', deviceValidation, async (req, res) => {
  const device = await Device.create(req.body);

  res.json({ device });
});
```

Рисунок 3.19 – Додавання пристрою до БД

Для отримання кількості пристроїв було створено роут /count, метод Device.count() повертає кількість пристроїв, які знаходяться у БД та у відповідь передає значення, отримане після виконання запиту до БД. Отримання кількості пристроїв зображено на рисунку 3.20.

```
router.get('/count', async (req, res) => {
  const count = await Device.count();

  res.json(count);
});
```

Рисунок 3.20 – Отримання кількості пристроїв

Отримати стан кожного пристрою у складі ГАЛ можна за допомогою роуту /data-all, під час звернення до цього роуту виконується два запити до БД. У першому запиті отримуються ідентифікатори uuid пристроїв, після чого виконується другий запит, який отримує останній запис для кожного пристрою та за допомогою синтаксису es6 виконується операція spread, у результаті якої в об’єкті знаходяться дані про стан пристрою, його максимальну потужність та назва пристрою (рисунок 3.21). Після виконання всіх запитів користувач отримує масив даних із даними про всі пристрої. Роут для отримання даних за всіма пристроями зображено на рисунку 3.22.

```
{
    "performance": 44,
    "uuid": "56bf14be-d28d-4449-a8bf-3f0e18ccc829",
    "temp": 77,
    "velocity": 2462,
    "uptime": 25745858,
    "state": "working",
    "vbr": 94,
    "createdAt": "2020-11-25T07:39:56.808Z",
    "updatedAt": "2020-11-25T07:39:57.187Z",
    "name": "станок 1"
},
```


Рисунок 3.21 – Результат запиту до БД

```
router.get('/data-all', async(req, res) => {

    const allDevices = await Device.findAll({
      attributes: ['uuid', 'name', 'createdAt'],
      raw: true,
    });

    const data = await Promise.all(allDevices.map(async device => {
      const lastRecord = await DeviceData.findAll({
        limit: 1,
        where: { uuid: device.uuid },
        order: [[ 'createdAt', 'DESC' ]],
        attributes: { exclude: ['id'] },
        raw: true,
      })

      return { ...lastRecord[0], ...device };
    }));

    res.json(data);

})
```

Рисунок 3.22 – Роут для отримання стану пристроїв

Для отримання даних для конкретного пристрою було розроблено роут /get-data/:uuid, він дозволяє отримати дані про стан пристрою у хронологічному

порядку. Через GET-параметри передається uuid потрібного пристрою, отримати його можна з об'єкту req.params. Після отримання запиту виконується запит до БД, у результаті виконання запиту буде отримано дані пристрою, після чого вони будуть відправлені користувачу, якщо uuid не був вказаний, то користувач отримає помилку зі статус-кодом 400. Роут для отримання даних про пристрій зображено на рисунку 3.23.

```
router.get('/get-data/:uuid', async (req, res) => {
  const { uuid } = req.params;

  if (uuid) {
    const device = await DeviceData.findAll({ where: { uuid } });

    return res.json(device);
  }

  res.sendStatus(400);
});
```

Рисунок 3.23 – Роут для отримання даних про пристрій

Для отримання стану конкретного пристрою було розроблено роут /state/:uuid. Через GET-параметри передається uuid потрібного пристрою, отримати його можна з об'єкту req.params. Після отримання запиту виконується запит до БД, у результаті виконання запиту буде отримано стан пристрою, після чого стан буде відправлений користувачу, якщо uuid не був вказаний, то користувач отримає помилку зі статус-кодом 400. Роут для отримання стану пристрою зображено на рисунку 3.24.

Для отримання списку всіх пристроїв було розроблено роут /list, який виконує запит до БД та повертає масив всіх пристроїв користувачу. Отримання пристроїв зображено на рисунку 3.25.

```
router.get('/state/:uuid', async (req, res) => {
  const { uuid } = req.params;

  if (uuid) {
    const info = await Device.findOne({
      where: {
        uuid,
      },
    });
    const state = await (
      await DeviceData.findOne({
        where: { uuid },
        order: [['createdAt', 'DESC']],
        limit: 1,
      })
    ).state;

    return res.json({ info, state });
  }

  res.sendStatus(400);
});
```

Рисунок 3.24 – Роут для отримання стану пристрою

```
router.get('/list', async (req, res) => {
  const allDevices = await Device.findAll({
    attributes: ['name', 'uuid', 'createdAt'],
  });

  res.json(allDevices);
});
```

Рисунок 3.25 – Отримання списку пристроїв

Для отримання середніх значень стану пристрою (температури, продуктивності і т. д.) було розроблено роут /avg-data/:type. Тип необхідних даних передається до функції avgData, дана функція у свою чергу конструює запит до БД у залежності від типу, якщо тип дорівнює total, то параметри запиту залишаються незмінними, у випадку, якщо тип дорівнює single, до параметрів запиту додаються необхідні атрибути та дані групуються за uuid.

Після виконання всіх запитів до БД отримані дані передаються до функції для приведення даних у потрібний формат. Функції для отримання даних і для приведення даних до потрібного формату наведені на рисунку 3.26.

```
const format = (data: any[]) => {
  let result: any = {};

  data.forEach((i: any) => {
    result[Object.keys(i)[0]] = i[Object.keys(i)[0]];
  });

  return result;
};
```

а)

```
const avgData = async (type: 'total' | 'single' = 'total') => {
  const metrics = ['temp', 'uptime', 'vbr', 'velocity'];

  const promice = metrics.map(async (metric) => {
    const params: FindOptions<any> = {
      attributes: [[fn('avg', col(metric)), `avg${metric}`]],
    };

    if (type == 'single') {
      params.attributes = ['uuid', [fn('avg', col(metric)), `avg${metric}`]];
      params.group = 'uuid';
    }

    return { [metric]: await DeviceData.findAll(params) };
  });

  const result = await Promise.all(promice);

  return format(result);
};
```

б)

Рисунок 3.26 – Функція для приведення даних до потрібного формату (а) та функція для отримання даних (б)

Для обміну даними у режимі реального часу була використана технологія websocket, після запиту до роуту /update-state виконується запит до БД, який записує оновлений стан і викликає на клієнті подію state_update, яка оновлює

стан пристроїв на клієнті. Оновлення даних у режимі реального часу зображено на рисунку 3.27.

```
router.post('/update-state', deviceInfoValidation, async (req, res) => {
  const deviceData = req.body;

  try {
    io.emit('state_update', JSON.stringify(deviceData));
    await DeviceData.create(deviceData);

    res.sendStatus(200);
  } catch (error) {
    res.sendStatus(400);
  }
});
```

Рисунок 3.27 – Оновлення стану пристрою

4 КЛІЄНТСЬКА ЧАСТИНА ПРОГРАМИ ДЛЯ РЕАЛІЗАЦІЇ ЗАПРОПОНОВАНОГО МЕТОДУ КЕРУВАННЯ ЕЛЕКТРОННИМИ ПРИСТРОЯМИ

4.1 Програмування клієнтської частини програми

Для написання клієнтської частини було обрано фреймворк React.js.

React являє собою JavaScript-бібліотеку для створення користувацьких інтерфейсів. Його головне завдання – забезпечення виведення на екран того, що можна бачити на веб-сторінках. React значно полегшує створення інтерфейсів завдяки розподіленню кожної сторінки на невеликі фрагменти. Такі фрагменти називаються компонентами.

Приклад розбиття сторінки на компоненти наведено на рисунку 4.1.

Рисунок 4.1 – Приклад розбиття сторінки на компоненти

Компонент React – це код, який представляє частину веб-сторінки. Кожен компонент – це JavaScript-функція, яка повертає частину коду, що представляє фрагмент сторінки.

У React є два способи створення компонентів – функціональні та класи компонентів. Приклад функціонального компонента наведено на рисунку 4.2, приклад класового компонента наведено на рисунку 4.3.

```
function OurFirstComponent() {
  return (
    <h1>Hello, I am a React Component!</h1>
  );
}
const placeWeWantToPutComponent = document.getElementById('hook');
ReactDOM.render(OurFirstComponent(), placeWeWantToPutComponent);
```

Рисунок 4.2 – Приклад функціонального компонента

```
class Container extends React.Component {
  render() {
    return (
      <div>
        <h1>I am the parent!</h1>
        <OurFirstComponent />
      </div>
    );
  }
}
const placeWeWantToPutComponent = document.getElementById('hook');
ReactDOM.render(<Container />, placeWeWantToPutComponent);
```

Рисунок 4.3 – Приклад класового компонента

Класи компонентів мають містити функцію, що має назву render (). Ця функція повертає JSX-код компонента. Їх можна використовувати так само, як функціональні компоненти, наприклад, звертаючись до них за допомогою конструкції <AClassComponent />.

Компоненти, засновані на класах, можуть зберігати інформацію про поточний стан веб-додатку. Ця інформація називається станом (state), вона зберігається в JS-об’єкті. На рисунку 4.4 показано об’єкт, який представляє стан компонента. Його ключ – це showPopup, з ним пов’язано значення,

протилежне поточному стану [19-20].

```
this.setState({
  showPopup: !this.state.showPopup
});
```

Рисунок 4.4 – Встановлення стану компонента

Користувач може взаємодіяти з компонентом, натискаючи на кнопку або елементи веб-сторінки. Якщо потрібно реагувати на подію, яка відбувається на веб-сторінці, то це виконується за допомогою функції, яка займається обробкою подій. Ці функції так і називаються – обробники подій.

Коли користувач натискає на кнопку, яка представлена тегом <button>, компонент викликає функцію togglePopup(). Функція отримує об'єкт події в якості аргументу, а це значить, що вона може, за необхідності, ним користуватися. Ми використовуємо метод .bind функції handleClick для того, щоб ключове слово this посилалося на весь компонент, а не тільки на <button>.

4.2 Структура програми

Для створення додатку на React використано пакет create-react-app.

Він дозволяє згенерувати стандартний додаток, для його встановлення потрібно у терміналі ввести команду npm і create-react-app, після чого почнеться завантаження пакету. Для генерації вводимо команду create-react-app та назву додатку, після чого буде створена структура веб-додатку.

Структура програми:

– node_modules / – в цій папці знаходяться всі залежності проєкту, які перераховані в package.json і встановлюються після запуску npm install;

– public / – вміст цієї папки – це те, що потрібно, щоб візуалізувати сторінку з додатком: public / index.html – шаблон додатку і favicon.ico.;

– src / – тут знаходиться весь код проєкту. Файл index.tsx – вхідна точка проєкту.

Після того, як створено стандартний додаток, можна приступати до розробки власного додатку. Головний файл – App.js, до нього підключаються всі компоненти додатку, частини додатку знаходяться в директорії /components.

Перш за все потрібно підключити необхідні компоненти та модулі (рисунок 4.5).

```
import React from 'react';
import { HashRouter as Router, Switch, Route, Redirect } from 'react-router-dom';
import { connect } from 'react-redux';

import './styles/style.scss';

import Login from './pages/auth/login';
import Register from './pages/auth/register';
import Dashboard from './pages/dashboard';
import DevicesList from './pages/devicesList';
import DashboardWrapper from './components/dashboardWrapper';
import DeviceInfo from './pages/deviceInfo';
import ChartPage from './pages/chartPage';
```

Рисунок 4.5 – Необхідні компоненти та модулі додатку

Перший рядок – це підключення компонентів React зі встановленого node-модуля.

Модуль react-router-dom дозволяє створювати багатосторінкові додатки.

Він надає можливість використовувати наступні модулі:

– Redirect – цей компонент відповідає за переспрямування користувача на іншу сторінку;

– Router – низькорівневий інтерфейс для всіх компонентів маршрутизатора;

– Switch – відповідає за переключення компонентів у залежності від url;

– Route – дозволяє візуалізувати компоненти у залежності від url.

Використовуючи модуль React router можна не завантажувати заново вже готові компоненти, а просто оновлювати необхідні, це дозволить підвищити швидкість завантаження сторінки.

Завдяки модулю react-redux можна задавати глобальний стан для програми, наприклад дані про авторизацію користувача або його привілеї.

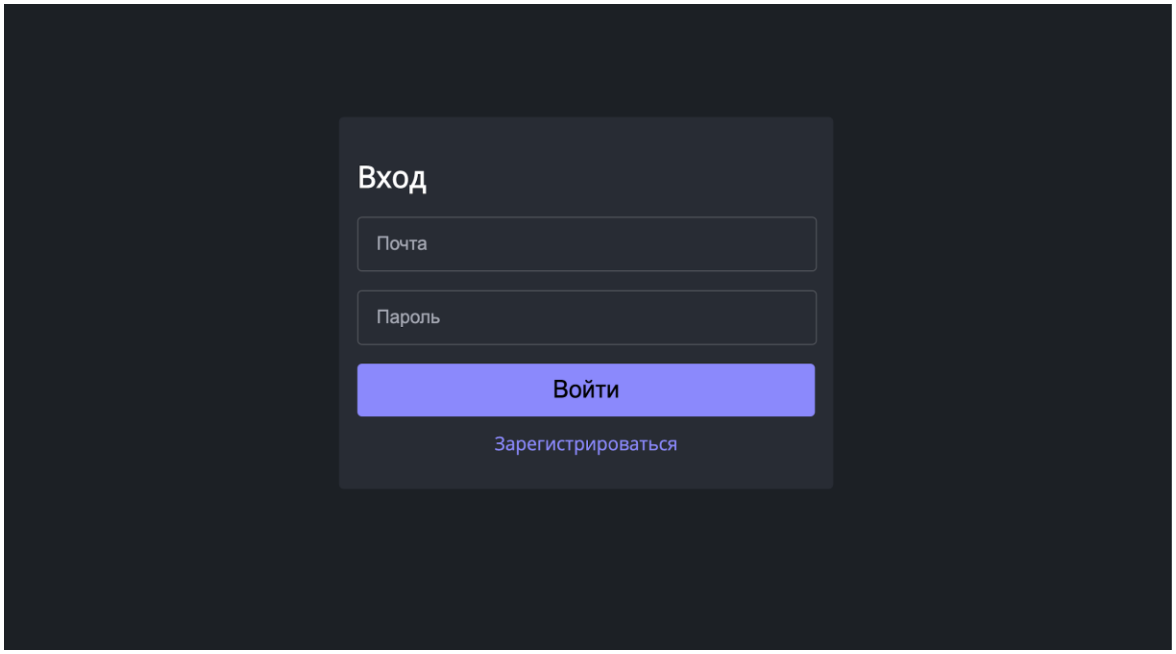
Модулю Route в якості атрибуту передається компонент та url адреса для подальшої візуалізації його під час переходу на інші сторінки. Якщо встановлений атрибут exact, то використовується тільки суворе порівняння url, наприклад, якщо exact встановлений для '/example', то за url '/example/something' сторінка не буде доступною. Фрагмент коду наведено на рисунку 4.6.

```
<Router>
  <Switch>
    <Route exact path="/register" component={Register} />
    <Route exact path="/login" component={Login} />
    <Route exact path="/">
      <Redirect to={page} />
    </Route>
    {isLogged ? (
      <DashboardWrapper>
        <Route path="/dashboard/device/:uuid" component={DeviceInfo} />
        <Route path="/dashboard/chart" component={ChartPage} />
        <Route exact path="/dashboard" component={Dashboard} />
        <Route exact path="/dashboard/list" component={DevicesList} />
      </DashboardWrapper>
    ) : <Route path="/login" component={Login} />}
  </Switch>
</Router>
```

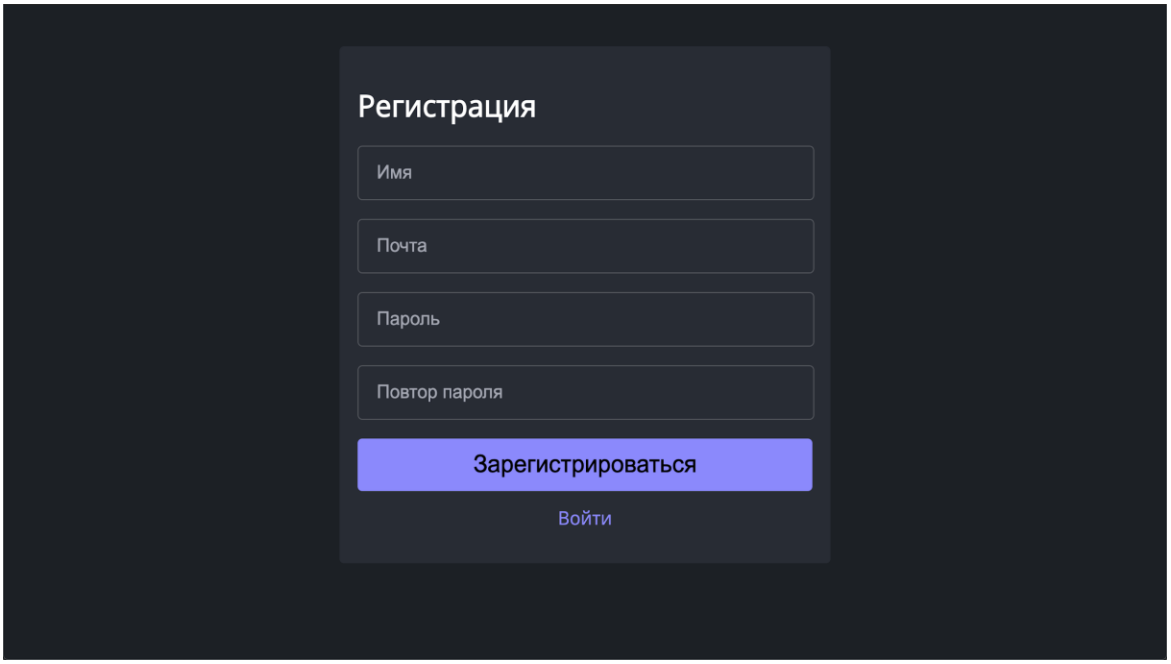
Рисунок 4.6 – Маршрутизація програми

4.3 Авторизація користувача

Після запуску програми буде перевірено, чи авторизований користувач, якщо вхід або реєстрація були виконані, то відкривається сторінка з показниками приладів, в іншому випадку буде відкрито сторінку авторизації, де користувач може зайти в уже зареєстрований аккаунт або створити новий. На рисунку 4.7 зображено форми авторизації.



а)



б)

Рисунок 4.9 – Форми входу (а) та реєстрації (б) користувача

Після натискання на кнопку Зареєструватися спрацьовує подія handleSubmit, яка отримує два об’єкта (actions, history), після чого зі стану компонента отримує email, password, confirmPassword, unserName, після чого порівнює між собою password та confirmPassword. Якщо паролі не співпадають, то до стану компоненту записується повідомлення про помилку та закінчується

виконання функції. Якщо паролі співпадають, то виконується подія actions.register, яка виконується асинхронно. Якщо після виконання події виникне помилка, або аккаунт із такими даними буде вже зареєстровано, то користувач побачить повідомлення про помилку. Після успішної реєстрації користувач автоматично потрапляє на сторінку панелі керування пристроями. Метод реєстрації користувача наведено на рисунку 4.10.

```
handleSubmit = async (e: React.SyntheticEvent) => {
  e.preventDefault();
  const { actions, history } = this.props;
  const { email, password, confirmPassword, userName } = this.state;

  if (password !== confirmPassword) {
    return this.setState({
      error: 'Пароли не совпадают',
    });
  }

  try {
    await actions.register(email, userName, password);

    history.push('/dashboard');
  } catch (error) {
    this.setState({
      error: 'Произошла ошибка',
    });
  }
};
```

Рисунок 4.10 – Метод реєстрації користувача

Після натискання на кнопку Увійти, буде виконаний метод, який асинхронно виконає запит до API через подію actions.login, буде виконано запит, у результаті якого буде отримано токен користувача та здійсниться переспрямування користувача на сторінку панелі керування. Якщо в ході виконання запиту з’явиться помилка, то користувачу буде показано повідомлення про помилку. Метод авторизації користувача зображено на рисунку 4.11.

```
handleSubmit = async (e: React.SyntheticEvent) => {
  e.preventDefault();

  const { email, password } = this.state;
  const { actions, history } = this.props;

  try {
    await actions.login(password, email);

    history.push('/dashboard');
  } catch (error) {
    this.setState({
      error: 'Произошла ошибка',
    });
  }
};
```

Рисунок 4.11 – Метод авторизації користувача

4.4 Панель керування пристроями

Після успішної авторизації користувач потрапляє на панель керування пристроями, на ній з лівої сторони знаходяться панель навігації та кнопка виходу з аккаунту, з правої сторони знаходяться головні середньостатистичні дані пристроїв, а саме:

– імовірність безвідмовної роботи (ІБР);

– загальна кількість пристроїв;

– середня температура пристроїв;

– середній час у роботі;

– середня швидкість роботи.

На момент відкриття сторінки користувачем спрацьовує подія componentDidMount та ініціалізує метод getData, який асинхронно виконує

запит до API та додає результат виконання у стан компонента. Отримання середньостатистичних даних про температуру, ІБР і продуктивність пристрою зображено на рисунку 4.12.

```
getData = async () => {
  const total = await Devices.countDevices();
  const avgData: IAvgData = await Devices.avgData();
  console.log(avgData);

  this.setState({
    total,
    avgData,
    loading: false,
  });
};

componentDidMount() {
  this.getData();
}
```

Рисунок 4.12 – Отримання даних даних про температуру, ІБР і продуктивність пристрою

Після успішно виконаного запиту результат записується до стану компонента, а далі вони відображаються у вигляді картки у правій частині програми. На рисунку 4.13 зображено приклад картки з даними про пристрої.

На вкладці Пристрої перелічено всі пристрої, а саме їх назву та дату створення. За умови приєднання компонента спрацьовує подія componentDidMount та запускається метод loadDevices, цей метод виконує асинхронний запит до API та отримує список всіх пристроїв, після чого додає пристрої до стану компонента. На рисунку 4.14 зображено метод для отримання списку пристроїв.

```
<Card style={{ width: '100%' }}>
  <div className="wrapper">
    <h2>Всего устройств</h2>
    <span>{total}</span>
  </div>
  <FiHardDrive />
</Card>
```

Рисунок 4.13 – Приклад картки

```
async loadDevices() {
  const devices = await Devices.allDevices();

  this.setState({
    devices
  })
}
```

Рисунок 4.14 – Метод для отримання списку пристроїв

Після отримання пристроїв виконується метод renderDevices, який відображає пристрої користувачу, для переходу на сторінку пристрою достатньо натиснути на назву пристрою. Метод для відображення пристроїв наведено на рисунку 4.15. Інтерфейс користувача зображено на рисунку 4.16.

```
renderDevices = (devices: IDevice[]) => {
  return devices.map((device) => (
    <StyledLink key={device.uuid} path={`/dashboard/device/${device.uuid}`}>
      {device.name}
      <span className="creation-date">
        {new Date(device.createdAt).toLocaleDateString()}
      </span>
    </StyledLink>
  ));
}
```

Рисунок 4.15 – Метод для відображення списку пристроїв

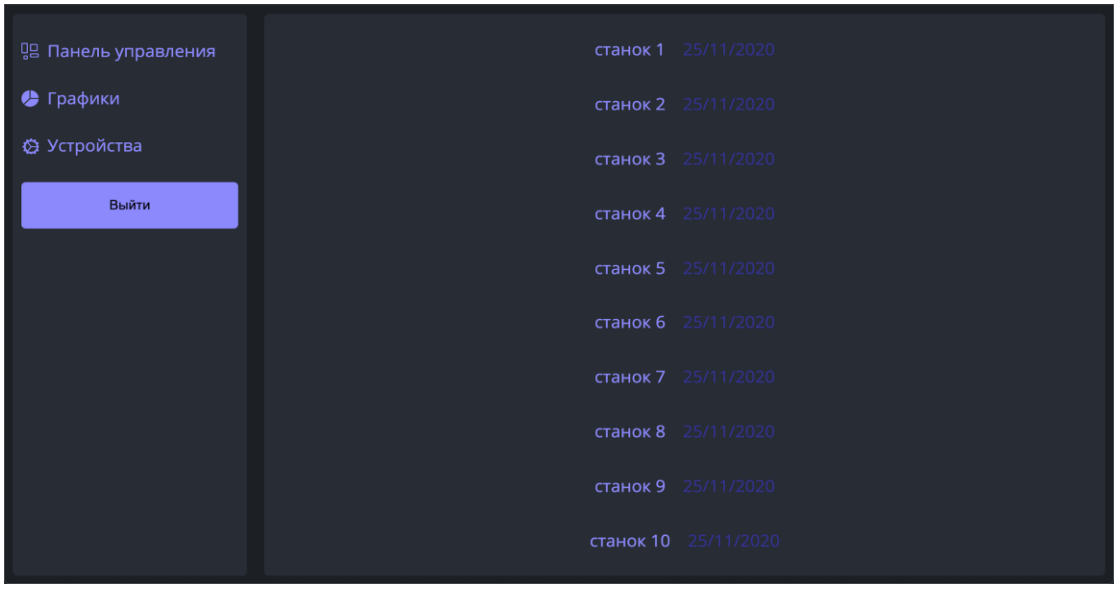


Рисунок 4.16 – Інтерфейс користувача

Сторінка пристрою складається з компонента керування пристроєм та чотирьох графіків, які відображають статистику у хронологічному порядку за такими показниками як:

– швидкість;
– ІБР;
– температура;
– час в роботі.

Компонент з даними пристрою зображено на рисунку 4.17.

```
const DeviceInfo: React.FC<IProps> = ({ match }) => {
  const { uuid } = match.params;

  return (
    <div className="info-wrapper">
      <DeviceTitle uuid={uuid} />
      <DeviceChart uuid={uuid} mark="velocity" />
      <DeviceChart uuid={uuid} mark="temp" />
      <DeviceChart uuid={uuid} mark="uptime" />
      <DeviceChart uuid={uuid} mark="vbr" />
    </div>
  );
};
```

Рисунок 4.17 – Компонент з даними пристрою

Для відображення даних про температуру, ІБР і продуктивність пристрою у режимі реального часу у компонент передається пропс deviceInfo, який знаходиться у сховищі redux store, завдяки redux можна отримати доступ до стану пристрою у будь-якому компоненті, необхідно лише відобразити компонент за допомогою функції connect, ця функція приймає такі параметри:

– mapStateToProps;
– mapDispatchToProps;
– компонент для відображення.

MapStateToProps передає пропси до компоненту за допомогою функції, яка зображена на рисунку 4.18. Ця функція повертає об’єкт зі значеннями, які отримуються зі сховища redux.

```
const mapStateToProps = (state: { devices: any }) => {
  return { deviceInfo: state.devices.data };
};
```

Рисунок 4.18 – Приклад функції MapStateToProps

MapDispatchToProps передає події до компоненту за допомогою функції, яка зображена на рисунку 4.19. Ця функція повертає об’єкт, у якому знаходяться події.

```
const mapDispatchToProps = (dispatch: Dispatch<AnyAction>) => {
  return bindActionCreators({ deviceData }, dispatch);
};
```

Рисунок 4.19 – Приклад функції mapDispatchToProps

Після підключення компоненту спрацьовує подія componentDidMount та запускається метод loadData, який шукає у масиві пристроїв необхідний пристрій за ідентифікатором uuid та оновлює стан компонента. Метод loadData зображено на рисунку 4.20.

```
loadData() {
  const { uuid, deviceInfo } = this.props;
  const deviceIndex = deviceInfo.findIndex((device) => device.uuid === uuid);
  const deviceData = deviceInfo[deviceIndex];

  this.setState({
    ...deviceData,
    deviceIndex,
  });
}
```

Рисунок 4.20 – Реалізація метода loadData

У компоненті DeviceTitle відображено стан пристрою, дані у режимі реального часу та реалізована можливість керування пристроєм, присутня можливість встановлення стану пристрою та регулювання продуктивності. Якщо користувач змінює стан пристрою, то з'являється кнопка для оновлення даних про пристрій. Інтерфейс користувача наведено на рисунку 4.21.

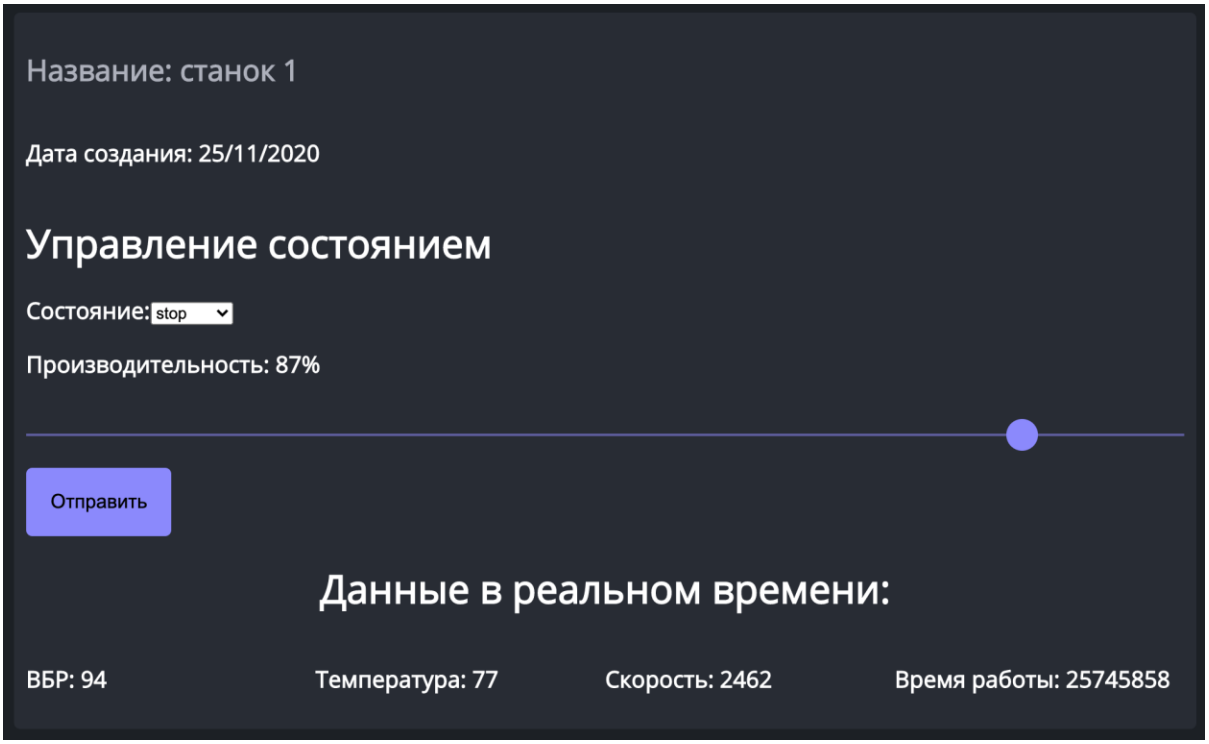


Рисунок 4.21 – Інтерфейс для керування станом пристрою

Також на сторінці інформації про пристрій присутні чотири графіки, вони відображають стан пристрою у хронологічному порядку. На осі абсцис

відображається дата, а на осі ординат зображено показники температури і т. д. Приклад такого графіку зображено на рисунку 4.22.

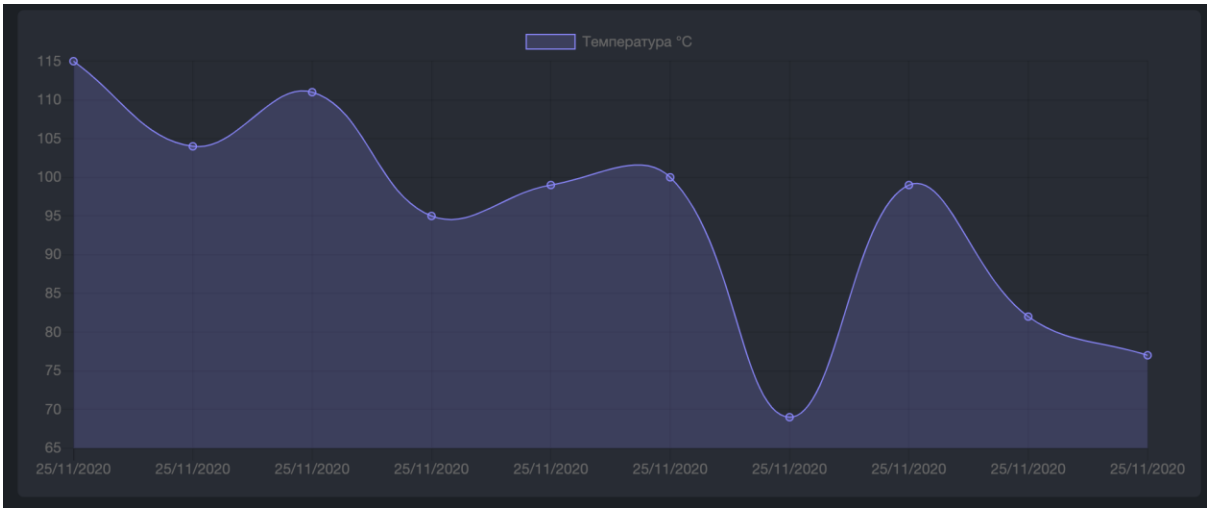


Рисунок 4.22 – Приклад хронологічного графіку температури пристрою

На сторінці Графіки відображено середні показники за всіма пристроями у вигляді стовпців, кожен пристрій – це стовпець. На рисунку 4.23 зображено графік з даними за всіма пристроями.

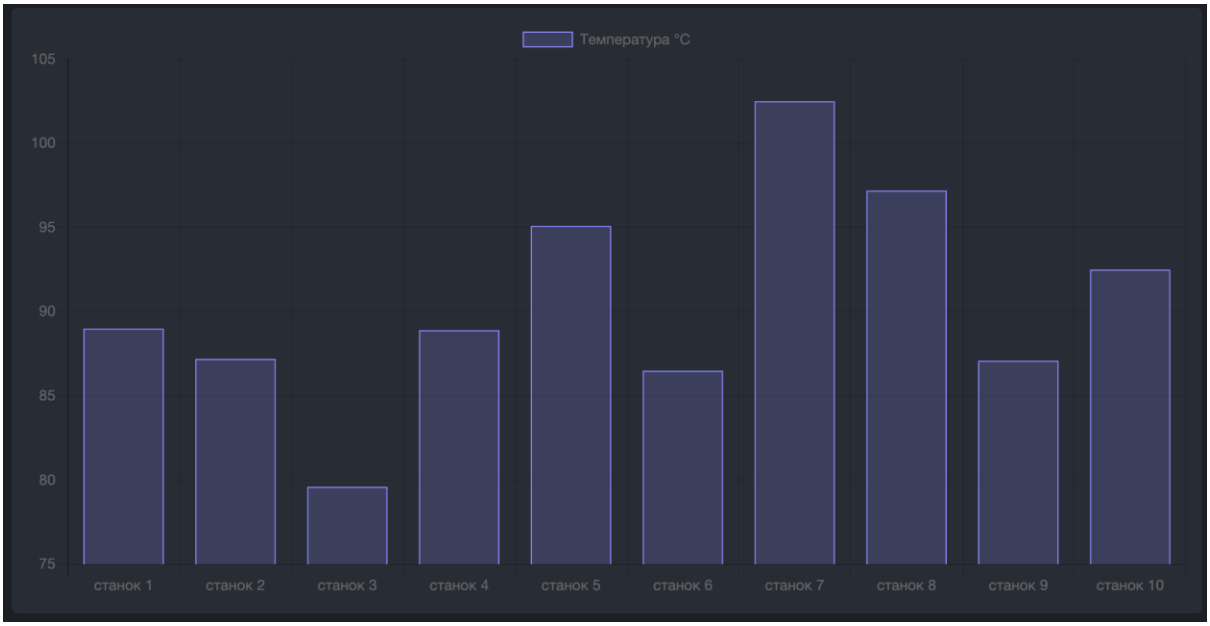


Рисунок 4.23 – Графік середніх показників пристроїв

4.4 Дані у режимі реального часу

Для отримання даних у режимі реального часу було використано технологію webScocket. WebSocket (Веб-сокет) – це протокол повнодуплексного зв'язку, який працює на основі TCP-з'єднання. Тобто за допомогою цього протоколу можна передавати та приймати повідомлення одночасно. Він дозволяє у режимі реального часу обмінюватися повідомленнями між сервером і клієнтом.

WebSocket встановлює одне єдине постійне з'єднання клієнта з сервером, за яким відбувається двосторонній обмін інформацією. Оскільки з'єднання з клієнтом і сервером не закривається (він тримається відкритим постійно), це дозволяє уникнути передачі зайвих даних (HTTP-заголовки). Так само у стандарті WebSockets немає ніяких обмежень за кількістю відкритих з'єднань і черги запитів. Для відправлення повідомлення нетекстових даних можна також використовувати об'єкти Blob і ArrayBuffer.

Переваги websockets:

– двостороннє з'єднання;
– зниження використовуваного трафіку;
– безпека;
– мінімальна затримка передачі даних.

Ці переваги є суттєвими для реалізації засад концепції Індустрії 4.0 на промислових підприємствах, для яких розробляється наша система.

Найпростіший варіант використання websocket – це чат. Однак не тільки чатами обмежується використання вебсокетів, вони потрібні скрізь, де є постійний двосторонній зв'язок між клієнтом (браузером) і сервером. Припустимо, цим інтерактивним інтерфейсом може бути веб-сайт, на якому постійно відбуваються зміни окремих блоків або всієї сторінки у залежності від дії користувача. Схему роботи websocket зображено на рисунку 4.24.

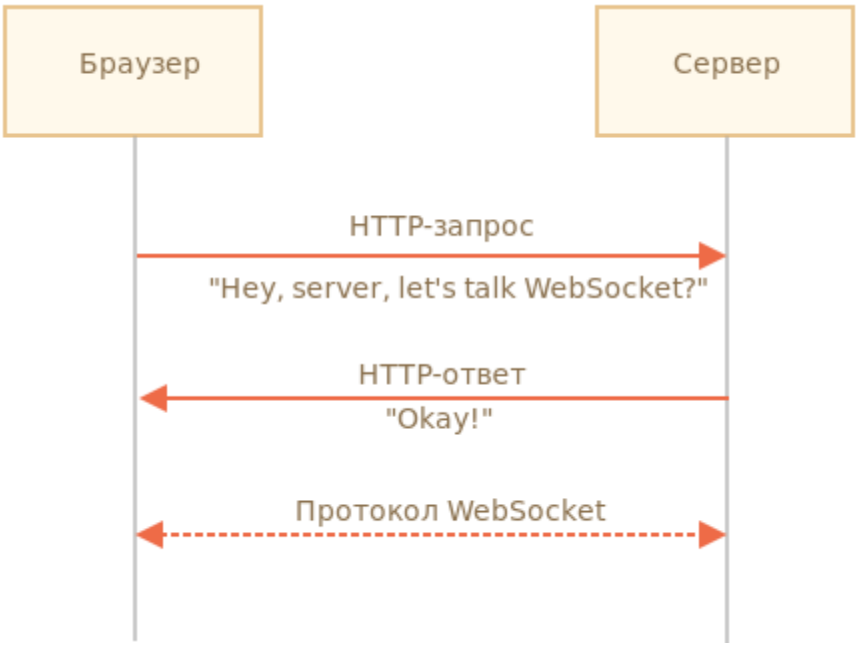


Рисунок 4.24 – Схема роботи websocket

У програмі реалізовано оновлення даних про пристрої у сервісі Socket.ts за допомогою модуля socket.io-client, у процесі ініціалізації програми встановлюється з'єднання з сервером і встановлюється прослуховування на подію data_update. Коли спрацьовує подія data_update, виконується подія updateData, ця подія оновлює стан пристрою у глобальному сховищі. Реалізація отримання даних з серверу у режимі реального часу наведена на рисунку 4.25.

```
const socket = io('ws://localhost:5001');

socket.on('data_update', (data: string) => {
  const parsedData = JSON.parse(data);

  store.dispatch(updateData(parsedData));
})
```

Рисунок 4.25 – Реалізація отримання даних з серверу у режимі реального часу

5 ОХОРОНА ПРАЦІ

5.1 Промислова безпека в лабораторії

Живлення комп'ютерів здійснюється від трифазної електричної мережі змінного струму з глухо-заземленою нейтраллю і напругою 380/220 В, і частотою 50 Гц.

Згідно НПАОП 40.1-1.21-98 робоче місце можна віднести до категорії без підвищеної небезпеки, оскільки у приміщенні відсутні чинники, які викликають підвищену або особливу небезпеку.

Для створення безпечних умов праці необхідно провести низку організаційних і технічних заходів. Згідно НПАОП 40.1-1.32-01 для запобігання ураження людини електричним струмом у приміщенні застосовується заземлення.

Згідно з вимогами НПАОП 0.00-4.12-05 проводиться вступний, первинний на робочому місці, повторний, цільовий і позаплановий інструктажі. Зміст інструктажу відповідає вимогам НПАОП 0.00-4.12-05. Факт проведення інструктажу відзначається у відповідних журналах з підписами інструктованих і інструктуючого.

5.2 Виробнича санітарія в лабораторії

Оптимальні норми температури, відносної вологості та швидкості руху повітря в робочій зоні виробничих приміщень для категорії робіт Іа є наступними відповідно до ДСН 3.3.6.042-99:

– у холодний період: температура 28-32 °С, відносна вологість 40-60 %, швидкість руху повітря менше або дорівнює 0,1 м/с;

– у теплий період: температура 30-34 °С, відносна вологість 40-60 %, швидкість руху менше або дорівнює 0,1 м/с.

Зорова робота користувача з програмним засобом є роботою високої точності, найменший розмір об'єкта розрізнення 0,3-0,5 мм і розряд зорової роботи – III.

Згідно вимог ДБН В.25-28-2006 величина коефіцієнта природного освітлення дорівнює 2 %. Природне світло проникає до приміщення через бічні вікна, що відповідає вимогам. Штучне освітлення встановлюємо у вигляді суцільних або переривчастих ліній світильників, розташованих паралельно лінії зору операторів. Освітленість під час роботи з екраном, у поєднанні з роботою над документами, – не менше 300 лк. В якості джерел світла застосовані люмінесцентні лампи.

Одним з варіантів контролювання температурних показників є використання системи кондиціонування.

Мета перевірочного розрахунку – вибір кондиціонера, який забезпечить оптимальні значення мікрокліматичних параметрів у приміщенні.

За умови виділення надлишкової теплоти кількість повітря визначається з умови асиміляції надлишків цієї теплоти. Кількість необхідного припливного повітря можна визначити.

За умови виділення надлишкової теплоти кількість повітря визначається з умови асиміляції надлишків цієї теплоти. Кількість необхідного припливного повітря можна визначити так:

$$L_{пр} = \frac{Q_{изб}}{c \cdot p \cdot (t_{уд.} - t_{пр.})}, \qquad (5.1)$$

де $Q_{изб.}$ – надлишкове виділення явної теплоти;

c – питома теплоємність повітря за постійного тиску (1,009 кДж/кг·град або 0,24 ккал/кг·град);

p – питома вага повітря у приміщенні (за нормальних умов 1,2928 кг/м^3);

$t_{уд.}$ – температура повітря, що видаляється;

$t_{пр.}$ – температура припливного повітря (повинна бути на 5-8 °С нижчою за температуру повітря у робочій зоні).

У приміщенні джерелами надлишкового тепла $Q_{изб.}$ є:

– надходження тепла від людей;

– виділення тепла від ПК;

– надходження тепла від сонячної радіації;

– виділення тепла від електричного освітлення і т.п.

Вхідні дані для розрахунку: $Q_{изб.}$ для розглянутої лабораторії дорівнює 968 ккал/г; $t_{уд.}$ = 30 °С, $t_{пр.}$ = 23 °С.

$$L_{\text{пр}} = \frac{968}{0{,}24 \cdot 1{,}2928 \cdot (30-18)} = 365 м^3 / г \quad . \qquad (5.2)$$

З розрахованого показника необхідної кількості припливного повітря отримали значення 365 м^3/г. Під час вибору кондиціонера слід віддавати перевагу моделям з показником продуктивності 200-300 м^3/г.

ВИСНОВКИ

Під час виконання атестаційної роботи проведено аналіз концепції Індустрія 4.0 та стану її реалізації в Україні. Також розглянуто типи гнучких автоматичних ліній і способи розрахунку надійності для різних типів автоматизованих систем, а саме систем із послідовним з'єднанням та паралельним елементів.

Проаналізувавши засади Індустрії 4.0 та розглянувши аналогічні системи керування компонентами систем автоматизації виробництва запропоновано метод для керування електронними пристроями у складі ГАЛ у режимі реального часу та розроблено алгоритм прийняття рішень для автоматичної роботи обладнання на виробництві.

У атестаційній роботі для реалізації запропонованого методу розроблено відповідне програмне забезпеченя для керування електронними пристроями в режимі реального часу, за допомогою цієї програми можна керувати станом пристроїв та слідкувати за статистикою стану обладнання

Зокрема, реалізовано можливість відображення статистики обладнання на графіках в панелі керування пристроями та реалізовано обмін даними між користувачем і пристроями в режимі реального часу та керування пристроями використовуючи хмарні технології.

Шляхами подальшого удосконалення розробленого програмного забезпечення є можливість додати такий функціонал:

- створення графіку роботи пристроїв;
- авторизація користувача через соціальні мережі;
- двухфакторна аутентифікація;
- групування пристроїв за їхнім типом;
- візуальне редагування дерева рішень;
- система для повідомлень користувачів про стан системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Невлюдов І. Ш. Методичні вказівки з розробки й оформлення магістерської атестаційної роботи другого (магістерського) рівня вищої освіти галузі знань 15 Автоматизація та приладобудування за спеціальністю 172 Телекомунікації та радіотехніка / І. Ш. Невлюдов, В. В. Косенко, В. В. Євсєєв. – Харьков: ХНУРЕ, 2019. – 55 с.

2. ДСТУ 3008-15. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. – К. Держстандарт України, 2017. – 29 с.

3. Основи наукових досліджень / І. Ш. Невлюдов [та ін.]: навч. посіб. – Кривий Ріг: КК НАУ, 2017. – 344 с.

4. Положення про організацію освітнього процесу в ХНУРЕ [Електронний ресурс]. – Режим доступу: https://nure.ua/polozhennya-pro-organizatsiyu-osvitnogo-protsesu-v-hnure.

5. Положення про протидію академічному плагіату в ХНУРЕ [Електронний ресурс]. – Режим доступу: https://nure.ua/wp-content/uploads/Main_Docs_NURE/Polozhennya-pro-protidiyu-akademichnomu-plagiatu-v-HNURE----290-vid-28.04.2017.pdf.

6. Положення про роботу екзаменаційних комісій ХНУРЕ [Електронний ресурс]. – Режим доступу: https://nure.ua/wp-content/uploads/Main_Docs_NURE/

1. Polozhennya-pro-poryadok-stvorennya-ta-organizatsiyu-roboti-ekzamenatsiynih-komisiy....pdf.

7. Положення про авторське право в ХНУРЕ [Електронний ресурс]. – Режим доступу: https://nure.ua/wp-content/uploads/Main_Docs_NURE/ Polozhennya-pro-avtorske-pravo-v-HNURE.pdf.

8. Мельник О. О., Боцман І. В. Розробка веб-додатку для керування електронними пристроями на приладобудівному виробництві у рамках реалізації концепції Індустрії 4.0 // Комп’ютерні інтелектуальні системи та мережі. Матеріали XIII Всеукраїнської науково практичної WEB конференції

аспірантів, студентів та молодих вчених (24-26 березня 2020 р.). – Кривий Ріг: Криворізький національний університет, 2020. – С. 196-199.

9. Аналіз Індустрії 4.0 [Електронний ресурс]. Режим доступу: http://www.economyandsociety.in.ua/journal/17_ukr/9.pdf.

10. Cloud Computing [Електронний ресурс]. Режим доступу: https://www.investopedia.com/terms/c/cloud-computing.asp.

11. Деревья решений: общие принципы [Електронний ресурс]. Режим доступу: https://loginom.ru/blog/decision-tree-p1.

12. Introduction to Sequelize [Електронний ресурс]. Режим доступу: https://medium.com/the-javascript-dojo/introduction-to-sequelize-1cbfc2d2d1bf.

13. Обзор СУБД PostgresSQL [Електронний ресурс]. Режим доступу: https://studfile.net/preview/1504673/.

14. Расчет надежности [Електронний ресурс]. Режим доступу: https://areliability.com/onlajn-kalkulyator-nadyozhnosti/.

15. Переваги Node.js [Електронний ресурс]. Режим доступу: https://techrocks.ru/2019/01/20/why-do-you-need-node-js.

16. Описання пакетного менеджеру npm [Електронний ресурс]. Режим доступу: http://prgssr.ru/development/vvedenie-v-paketnyj-menedzher-npm-dlya-nachinayushih.html.

17. Описання шифрування Bcrypt [Електронний ресурс]: Режим доступу: https://en.wikipedia.org/wiki/Bcrypt.

18. Пять простых шагов для понимания JWT [Електронний ресурс]: Режим доступу: https://habr.com/ru/post/340146/.

19. Документація React.js [Електронний ресурс]: Режим доступу: https://reactjs.org.

20. Введення в React.js [Електронний ресурс]: Режим доступу: https://habr.com/ru/company/ruvds/blog/343022.