Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризированих технологій
(повна назва)

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

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Удосконалення методу автоматизованого обліку готової продукції з використанням технології IIOT
(тема)

Виконав студент 2 курсу, групи АУТПм-22-2
Вишванюк Світлана Вікторівна
(прізвище, ім'я, по батькові)

Спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва напряму)

Тип програми Освітньо-професійна

Освітня програма Автоматизоване управління технологічними процесами
(назва)

Керівник доц. каф. КІТАР Хрустальова С.В.
(посада, прізвище, ініціали)

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

2024

Я, як студентка ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавала та не одержувала недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

22.01.2024  Вишванюк С.В.

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет АКТ
Кафедра КІТАР
Рівень вищої освіти другий (магістерський)
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
Тип програми Освітньо-професійна
Освітня програма Автоматизоване управління технологічними процесами
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав.кафедри _____
(підпис)
«__»_____2024р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Вишванюк Світлані Вікторівні
(шифр і назва)

1. Тема роботи: Удосконалення методу автоматизованного обліку готової продукції з використанням технології IIoT

Затверджена наказом університету від №1288Ст від 03.11.2023

2. Термін подання студентом роботи до екзаменаційної комісії 19.01.2024р.

3. Вихідні дані до роботи: 3.1 Використання технології IIoT; 3.2 Протокол MQTT або CoAP

4. Перелік питань, що потрібно опрацювати в роботі: 4.1 Вступ; 4.2 Огляд та Аналіз існуючих методів, засобів та автоматизованих систем обліку; 4.3 Сучасні методи обліку готової продукції 4.4 Технології IIoT в системах обліку готової продукції; 4.5 Розроблення структурної схеми та алгоритму роботи автоматизованного обліку готової продукції з використанням технології IioT; 4.6 Розроблення структурної схеми; 4.7 Вибір компонентів для удосконалення методу автоматизованного обліку; 4.8 Розроблення алгоритму роботи; 4.9 Вибір середовища та мови програмування; 4.10 Бази даних; 4.11 Дослідження методу у вигляді програмного забезпечення; 4.12 Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій Графічний демонстраційний матеріал в форматі PowerPoint(*.ppt) формату А4 –13 сторінок.

6. Консультанти розділів роботи

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд та аналіз існуючих методів, засобів та автоматизованих систем обліку	2.09.2023-15.09.2023	виконано
2	Сучасні методи обліку готової продукції	16.09.2023-23.09.2023	виконано
3	Огляд технології IIoT в системах обліку готової продукції	24.09.2023-2.10.2023	виконано
4	Розроблення структурної схеми	3.10.2023-16.10.2023	виконано
5	Вибір компонентів для удосконалення методу автоматизованного обліку	17.10.2023-24.10.2023	виконано
6	Розроблення алгоритму роботи	25.10.2023-3.11.2023	виконано
7	Вибір середовища та мови програмування	4.11.2023-12.11.2023	виконано
8	Створення бази даних	13.11.2023-16.11.2023	виконано
9	Дослідження методу у вигляді програмного забезпечення	17.11.2023-25.11.2023	виконано

Дата видачі завдання 1 вересня 2023р.

Студент _____ Вишванюк С.В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Хрустальова С.В.
(підпис) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 83 с., 0 табл., 39 рис., 3 дод., 16 джерел.

IIOT, RFID, SWIFT UI, ATTRIBUTES, ENTITIES, QR-CODE, ШТРИХ-КОД, SENSORS.

Мета дослідження – підвищення ефективності обліку готової продукції.

Об’єкт дослідження – процес обліку готової продукції.

Предмет дослідження – програмний засіб автоматизованого обліку готової продукції.

У роботі використовується комбінація методів теоретичного аналізу літературних джерел, аналізу існуючих систем виробничого обліку, та експериментальних досліджень для впровадження та тестування розробленого методу.

В результаті виконання магістерської роботи очікується створення ефективного методу автоматизованого обліку готової продукції з використанням технології IIoT, а також створення бази даних, реалізацію методу автоматизованного обліку у вигляді програмного забезпечення за допомогою мови програмування Swift UI. Розроблений метод може знайти застосування в промисловості для підвищення контролю та оптимізації виробничих процесів. Результати цієї роботи можуть бути важливим внеском у розвиток промислової автоматизації та виробничого управління.

ABSTRACT

Explanatory note: 83 pages, 0 tables, 39 figures, 3 app, 16 sources.

IIOT, RFID, SWIFT UI, ATTRIBUTES, ENTITIES, QR CODE, BARCODE, SENSORS.

Research aim: to enhance the efficiency of production accounting.

Research object: the process of production accounting.

Research subject: a software tool for automated production accounting.

The study employs a combination of theoretical analysis of literature sources, analysis of existing production accounting systems, and experimental research for the implementation and testing of the developed method.

The expected outcome of the master's thesis is the creation of an effective method for automated production accounting using IIoT technology, as well as the development of a database and implementation of the automated accounting method in the form of software using the Swift UI programming language. The developed method can be applied in industry to improve control and optimize production processes.

The results of this work can make a significant contribution to the development of industrial automation and production management.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – програмне забезпечення;

RFID (Radio Frequency Identification) – ідентифікація за допомогою радіочастот;

IIoT (Industrial Internet of Things) – промисловий Інтернет речей;

API (Application Programming Interface) – інтерфейс програмування додатків;

ERP (Enterprise Resource Planning) – планування ресурсів підприємства;

UI (User Interface) – інтерфейс користувача.

ВСТУП

У сучасному виробничому середовищі, де важлива кожна деталь виробничого процесу, автоматизація та використання передових технологій виявляються невід'ємною частиною ефективного виробництва. Індустрія Інтернету Речей (IIoT) визначає нові стандарти у сфері автоматизації та моніторингу виробничих процесів.

В умовах постійного росту конкуренції та змін у виробничих умовах, необхідно впроваджувати нові підходи до управління та контролю за виробництвом. IIoT надає можливість збору та обробки великого обсягу даних в режимі реального часу, що дозволяє ефективно реагувати на зміни та оптимізувати виробничі процеси. Предметом дослідження є існуючі методи, технології та системи автоматизованого обліку готової продукції, а також можливості їх удосконалення за допомогою технології Індустрії Інтернету Речей (IIoT).

Мета роботи полягає в розробці структурної схеми системи автоматизованого обліку готової продукції та алгоритму її роботи, враховуючи інноваційний підхід, що передбачає використання технології IIoT, та реалізації програмного засобу.

У ході дослідження ми аналізуватимемо існуючі підходи до обліку готової продукції, розглядаючи їхні переваги та обмеження. Результатом роботи стане розробка структурної схеми та лагоритму роботи автоматизованного обліку, орієнтованої на використання передових технологій, таких як сенсори та мережеві з'єднання, а також розробка алгоритму, який забезпечує ефективний збір, передачу та обробку даних, з урахуванням аспектів кібербезпеки.

Для досягнення поставленої мети перед нами стоять наступні завдання:

– аналіз існуючих методів та технологій: провести докладний огляд та оцінку ефективності поточних методів обліку готової продукції, визначити їх переваги та недоліки;

– поглиблено ознайомитися із технологією індустрії інтернету речей та визначити, як вона може бути використана для покращення методу обліку;

– розробка структурної схеми системи: створити детальну структурну схему автоматизованої системи обліку, враховуючи інтеграцію сенсорів, мережеві з'єднання та засоби збору даних;

– розробка алгоритму роботи системи: розробити ефективний алгоритм, що включає етапи збору, передачі та обробки даних, а також враховує аспекти кібербезпеки;

– зробити вибір компонентів для удосконалення методу автоматизованного обліку;

– розробка структурної схеми автоматизованного обліку готової продукції з використанням технології IIoT;

– реалізувати метод автоматизованного обліку у вігляді програмного засобу;

– вибрати середовище та мову програмування;

– створити базу даних;

– реалізувати метод у вигляді програмного забезпечення, зробити експеримент, та вивести результат у вікно програми на конкретному прикладі з описом реалізації.

Вирішення цих завдань дозволить нам систематизувати та реалізувати покращений метод обліку готової продукції, підвищуючи його точність, ефективність та стійкість до зовнішніх впливів. Ми реалізуємо метод у вигляді програмного забезпечення, що дозволить наглядно побачити реалізацію у вигляді розробки структурної схеми системи та алгоритм роботи, враховуючи інноваційні підходи, що буде передбачати використання технології IIoT.

Дана кваліфікаційна робота виконана згідно ДСТУ 3008 – 15 [1], керуючись навчальним посібником з дипломного проекту [2] та методичними вказівками [3]. Результати даної кваліфікаційної роботи описані в наукових статтях, та їх можна побачити в гугл академії [4].

1 ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ, ЗАСОБІВ ТА АВТОМАТИЗОВАНИХ СИСТЕМ ОБЛІКУ

1.1 Сучасні методи обліку готової продукції

В умовах стрімкого технологічного розвитку та неперервних змін у виробничих процесах, важливість ефективного обліку готової продукції для підприємств набуває ключового значення. Впровадження автоматизованих систем та передових технологій стає необхідністю для оптимізації цього процесу, забезпечуючи точність, швидкість та безпеку управління виробництвом.

Існують різноманітні методи автоматизованого обліку готової продукції, які застосовуються в промислових умовах. Деякі з них включають системи маркування та ідентифікації, системи сканування та RFID, системи візуалізації даних, системи автоматичного збору даних та інтегровані ERP системи. Ці методи сприяють точному обліку кількості та якості продукції, швидкому збору та обробці даних про виробництво, а також в оперативному моніторингу та управлінні виробничими процесами [5].

На першому етапі проводиться аналіз традиційних методів обліку. Ручне введення даних та використання штрих-кодів досліджуються з оцінкою їхніх переваг, таких як низькі витрати та простота використання, а також недоліків, наприклад, обмежена точність та можливі помилки при ручному введенні.

Другий етап присвячений аналізу технологій обліку, зокрема RFID технології. Глибокий розгляд використання RFID в системах автоматизованого обліку готової продукції дозволяє визначити параметри, такі як точність ідентифікації, швидкість зчитування та стійкість до

зовнішніх впливів. Цей етап також включає порівняльний аналіз з традиційними методами з метою визначення найбільш підходящого рішення.

Огляд існуючих систем обліку є третім етапом. Тут детально розглядаються сучасні автоматизовані системи обліку готової продукції, звертаючи увагу на їхню архітектуру, функціональні можливості та обмеження. Також досліджується, як ці системи можуть інтегруватися з існуючими виробничими процесами та сприяти оптимізації управління продукцією.

Четвертий етап включає порівняльний аналіз ефективності та вартості впровадження різних методів та технологій для обліку готової продукції. Також розглядається можливість інтеграції цих методів з іншими виробничими процесами з метою знаходження оптимального рішення для покращення обліку готової продукції в конкретних умовах та потребах підприємства.

На першому етапі аналізу досліджуються два традиційні методи обліку готової продукції: ручне введення даних та використання штрих-кодів. Ручне введення даних відрізняється простотою та доступністю використання. Оператор може вручну вводити дані про готову продукцію в систему, що особливо зручно для невеликого обсягу виробництва. Такий метод також характеризується низькими витратами на впровадження, оскільки не потребує складного обладнання.

Проте ручне введення даних має обмеження у точності та може призводити до помилок. Це особливо актуально для великих обсягів виробництва, де ручне введення даних може призвести до неточностей та помилок.

Щодо використання штрих-кодів, цей метод дозволяє автоматизувати процес введення даних. Кожен виріб має унікальний штрих-код, який може бути зчитаний за допомогою спеціального сканера. Це підвищує швидкість та точність обліку [5].

Проте вартість впровадження системи штрих-кодів може бути високою через потребу у спеціальному обладнанні та програмному забезпеченні. Крім того, штрих-коди можуть бути пошкоджені або втратити читабельність, особливо в умовах виробничого середовища.На другому етапі нашого дослідження ми вдаватимемося в аналіз технології RFID (Radio-Frequency Identification), яку можна побачити на рисунку 1.1, та її використання в системах автоматизованого обліку готової продукції.

Рисунок 1.1 – RFID-мітка

Технологія RFID (Radio Frequency Identification) дозволяє ідентифікувати об'єкти за допомогою безконтактних радіочастотних міток. Вона може бути корисною для вдосконалення процесу обліку готової продукції через свою можливість автоматичного зчитування та ідентифікації інформації [6].

Оцінка точності ідентифікації є ключовою частиною нашого дослідження. Ми детально вивчаємо ефективність RFID технології у розпізнаванні індивідуальних міток на готовій продукції. Важливо визначити, наскільки цей метод може забезпечити точність ідентифікації продукції в різних умовах виробництва. На рисунку 1.2 представлено RFID-мітка та її область застосування.

Рисунок 1.2 – RFID-мітка та її область застосування

Швидкість зчитування важлива у нашому аналізі. Враховуючи великі обсяги готової продукції, ми досліджуємо, наскільки оперативно RFID технологія може обробляти та передавати дані про ідентифікацію. Ми аналізуємо час, необхідний для зчитування інформації з міток та передачі цих даних в систему обліку. Стійкість до зовнішніх впливів, таких як вологість, температурні зміни та електромагнітні поля, також важлива у нашому дослідженні.

Оскільки умови виробництва можуть варіюватись, ми досліджуємо, наскільки надійно RFID технологія може працювати в різних середовищах. Порівняльний аналіз з традиційними методами обліку, такими як ручне введення даних або використання штрих-кодів, дозволяє нам з'ясувати переваги та недоліки кожного методу. Це допомагає визначити найкращий підхід для обліку готової продукції в конкретних умовах виробництва.

Третій етап аналізу – це огляд існуючих систем обліку. Архітектурні особливості та структури: сучасні автоматизовані системи обліку готової продукції мають складну архітектуру, спеціально створену для управління та контролю [6].

Ці системи зазвичай використовують розподілені архітектури для забезпечення ефективності та надійності. Інтеграція компонентів, таких як

сенсори, зчитувачі RFID та інтерфейси з базами даних, реалізована таким чином, щоб забезпечити швидку і точну передачу даних.

Функціональні можливості систем обліку готової продукції охоплюють широкий спектр функцій, включаючи автоматизоване визначення кількості та стану готової продукції, а також взаємодію з іншими виробничими системами. Сучасні рішення часто використовують інтелектуальні алгоритми для аналізу даних, прогнозування запасів та контролю якості. При обмеженнях та інтеграції важливо, щоб системи були здатні інтегруватися з іншими виробничими процесами. Обмеження можуть включати обмежену сумісність з застарілими технологіями, що впливає на швидкість обробки даних та обмін інформацією. Системи обліку готової продукції сприяють оптимізації управління виробництвом, надаючи оперативну інформацію про залишки, терміни придатності та різноманітні параметри якості. Аналітичні інструменти вбудовані в систему допомагають у прийнятті стратегічних рішень на основі зібраних даних.

Під четвертим етапом порівняльного аналізу важливо оцінювати ефективність та вартість впровадження різних методів та технологій. Потрібно визначити, наскільки швидко та точно вони здатні збирати, обробляти та аналізувати дані, а також як вони інтегруються з існуючими процесами. Після аналізу важливо визначити оптимальний підхід для конкретних умов та потреб підприємства. Це може включати вибір конкретного методу, технології чи їх комбінації, щоб максимально відповідати вимогам ефективного обліку готової продукції в умовах виробництва [7].

1.2 Технології IIoT в системах обліку готової продукції

Промисловий інтернет речей – це система об'єднаних комп'ютерних мереж і підключених до них промислових (виробничих) об'єктів з

вбудованими датчиками і програмним забезпеченням для збору та обміну даними, з можливістю віддаленого контролю і управління в автоматизованому режимі, без участі людини.

Вибір і застосування технологій Industrial Internet of Things (IIoT) у системах обліку готової продукції стає критичним для підприємств, спрямованих на підвищення ефективності та точності управління виробництвом. Під час цього аналізу враховуються різноманітні аспекти, такі як використання сенсорних технологій для збору точних даних, засоби забезпечення надійного зв'язку та мережі для передачі інформації, а також інноваційні рішення, спрямовані на оптимізацію управління та контролю за готовою продукцією у промислових умовах.

Огляд технологій IIoT включає в себе дослідження передових інструментів, які використовуються для підключення та керування промисловими процесами. Це включає в себе використання різних типів сенсорів, засобів збору та обробки даних, а також мережевих технологій для ефективного обміну інформацією.

Сенсорні технології використовуються для вимірювання різних параметрів виробничих процесів, таких як температура, тиск, вологість і т. д. Ці дані дозволяють отримувати точну і невідкладну інформацію для прийняття управлінських рішень.

Засоби збору та обробки даних використовуються для організації та аналізу великого обсягу інформації, яка надходить від сенсорів та інших джерел. Аналіз цих даних допомагає виявляти тенденції, прогнозувати виробничі сценарії та підвищувати ефективність виробничих процесів [8].

Мережеві технології забезпечують спільний зв'язок між усіма пристроями та системами виробництва, створюючи єдину структуру для обміну даними. Це включає в себе використання хмарних послуг, протоколів IoT та інших інноваційних методів передачі інформації. Технології зв'язку,

такі як 5G, NB-IoT та Wi-Fi, визначають швидкість та надійність передачі даних між пристроями.

Кібербезпека у контексті IIoT є критичним аспектом, оскільки зі збільшенням кількості підключених пристроїв стає необхідним забезпечити безпечний та надійний обмін інформацією в промислових мережах.

Гарантування конфіденційності та цілісності даних є однією з ключових завдань кібербезпеки. Використання заходів шифрування даних є важливим для запобігання несанкціонованому доступу до конфіденційної інформації. Крім того, контроль цілісності даних допомагає уникнути будь-яких змін чи втрати важливих виробничих даних.

Аутентифікація та авторизація відіграють ключову роль у впровадженні кібербезпеки в IIoT. Вони дозволяють перевіряти ідентифікацію пристроїв та користувачів, а також керувати рівнями доступу до систем та даних.

Заходи захисту від атак та вразливостей є важливими для забезпечення безпеки систем IIoT. Систематичний моніторинг та аналіз потенційних загроз дозволяють своєчасно реагувати на можливі атаки та запобігати їхнім наслідкам [9].

Застосування технологій IIoT в обліку готової продукції відкриває широкі можливості для підвищення ефективності та оптимізації виробничих процесів.

Давайте розглянемо ключові переваги цих технологій:

– моніторинг у реальному часі: завдяки технологіям IIoT можливе постійне ведення моніторингу готової продукції у реальному часі, що дозволяє швидко виявляти будь-які відхилення, керувати запасами та реагувати на зміни у виробничому процесі;

– точність та автоматизація обліку: автоматизовані системи, які базуються на технологіях IIoT, забезпечують високу точність в обліку готової

продукції, уникнення помилок, що можуть виникнути при ручному обліку, і надійні дані;

– оптимізація управління запасами: системи IIoT дозволяють ефективно відстежувати та керувати рівнем запасів готової продукції, уникати надмірних запасів, знижувати витрати на утримання складів і покращувати планування виробництва;

– забезпечення якості продукції: моніторинг параметрів якості продукції за допомогою сенсорів та засобів IIoT дозволяє оперативно виявляти відхилення в якості, що сприяє вчасному втручанню та покращенню якості готової продукції;

– зменшення витрат та оптимізація виробничих процесів: Автоматизація через технології IIoT спрощує багато виробничих операцій, що призводить до зменшення витрат на працю та оптимізації робочих процесів;

– вдосконалення рішень на основі даних: Збір та аналіз великого обсягу даних дозволяє створювати продуктивні аналітичні інструменти, що поліпшують прийняття рішень у управлінні готовою продукцією;

– забезпечення кібербезпеки: технології IIoT можуть включати продумані заходи кібербезпеки, що захищають дані про готову продукцію від несанкціонованого доступу та кібератак.

З урахуванням цих переваг, впровадження технологій IIoT у системи обліку готової продукції сприяє підвищенню ефективності та конкурентоспроможності підприємства.

Далі, на рисунку 1.3 можна ознайомитись з прикладами IIoT системи.

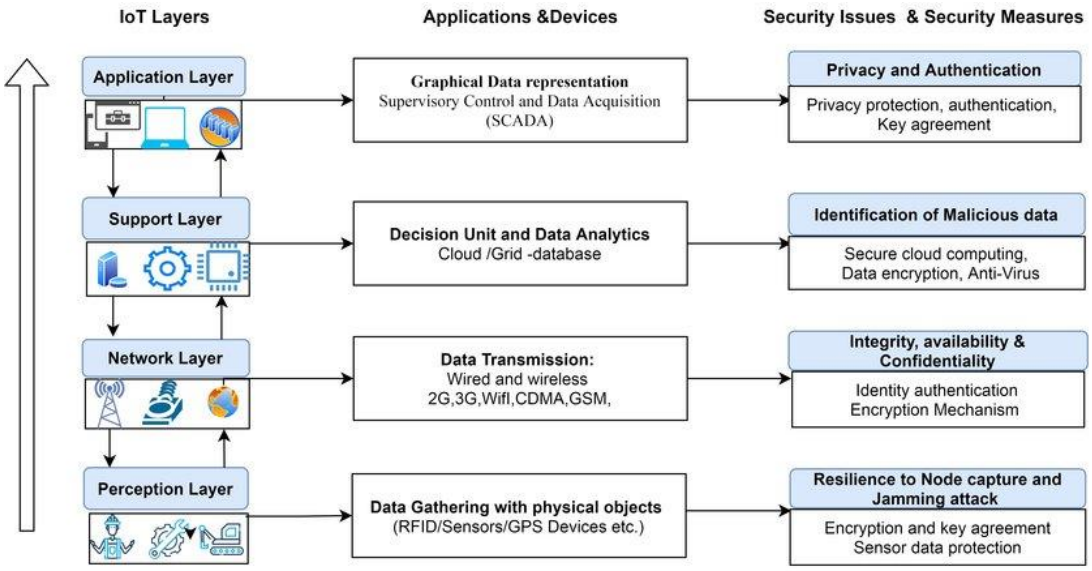


Рисунок 1.3 – Приклади IIoT системи

Технології хмарного зберігання та аналізу відіграють важливу роль у розвитку промислового Інтернету речей (IIoT). Завдяки хмарному зберіганню можна масштабувати обсяги даних, забезпечуючи їх доступність та надійність. Оскільки сенсори та пристрої IIoT генерують величезні обсяги інформації, хмарні платформи стають витратоекономним та ефективним рішенням.

Щодо аналізу даних, хмарні технології надають широкі можливості, включаючи швидку обробку та використання алгоритмів машинного навчання. Гнучкість та масштабованість хмарних рішень дозволяють адаптувати обчислювальні ресурси під потреби виробничих систем.

Незважаючи на переваги, важливо враховувати обмеження, такі як залежність від швидкості та стабільності мережі, аспекти приватності та безпеки даних, а також виклики інтеграції з існуючими системами. Проте все більше підприємств визнають значущий внесок хмарних технологій у вдосконалення управління та оптимізацію виробничих процесів в умовах IIoT.

Шлюз IoT є центральним концентратором для пристроїв IoT, що забезпечує їх зв'язок між собою та з хмарою. Він об'єднує пристрої IoT із хмарою, фільтрує дані та перетворює їх у корисну інформацію [10].

Серверні платформи та кінцеві пристрої підключаються до шлюзу IoT через комунікаційні технології, як представлено на рисунку 1.4. Шлюз має обчислювальну платформу, що дозволяє програмам керувати даними, пристроями, безпекою та іншими функціями, пов’язаними зі шлюзом.

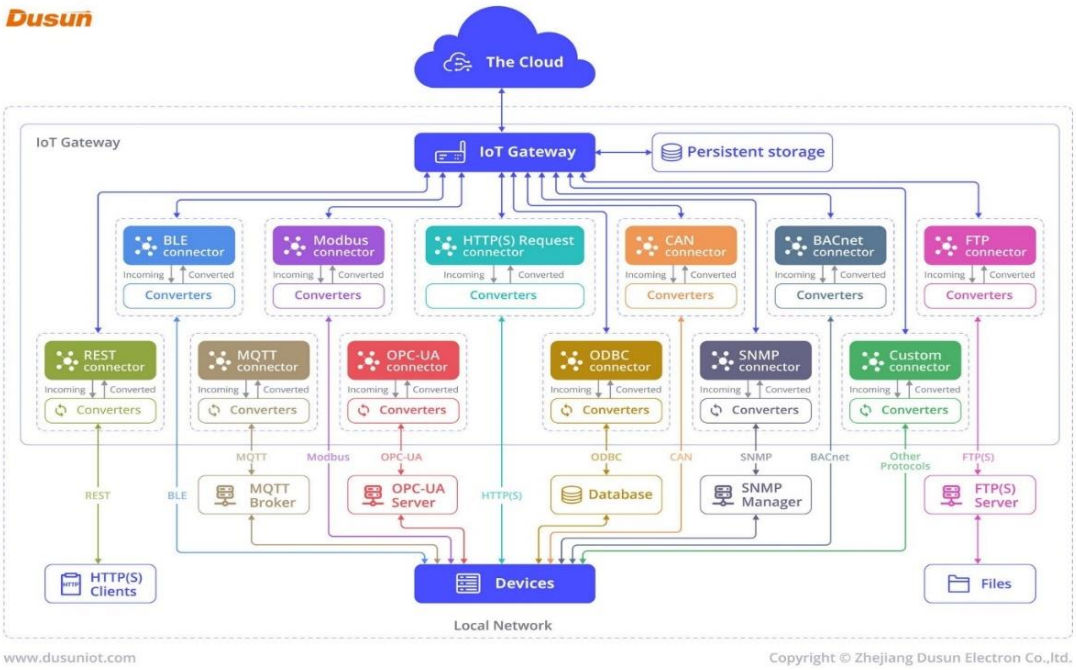


Рисунок 1.4 – Серверні платформи

Велика увага при розробці IoT приділяється встановленню з'єднання і роботі мереж.

До бездротових IoT-мереж/протколів як правило відносяться протоколи Bluetooth, mesh-мережі, Zigbee, Z-Wave. Для IIoT це також Wireless Hart та ISA100. Це яскравий приклад різноманіття бездротових систем зв'язку IoT.

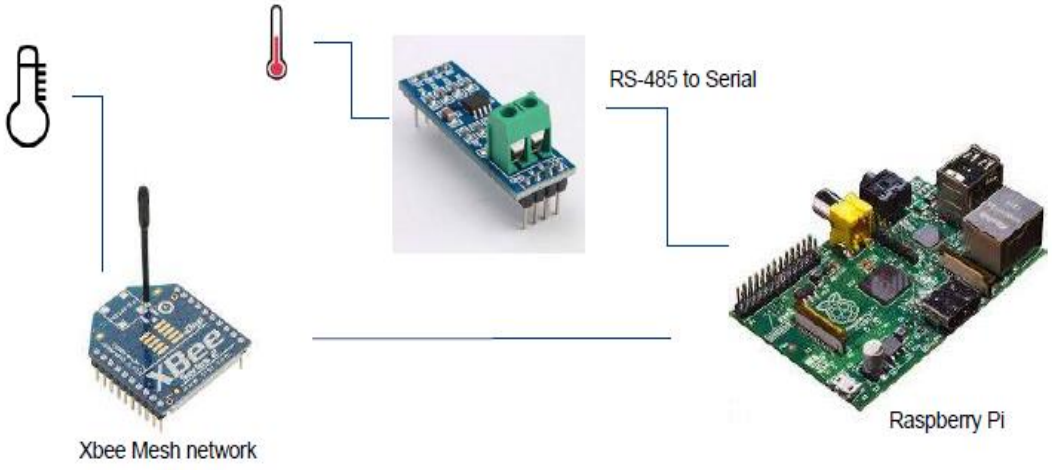


Рисунок 1.5 – Raspberry Pi – одноплатний комп'ютер, Xbee-модуль

Впровадження технологій IIoT у системи обліку готової продукції має потенціал для значних переваг, але водночас стикається з рядом викликів, які потребують уважного вивчення.

З позитивного боку, очікується підвищення ефективності завдяки обробці даних у реальному часі, що дозволить швидко реагувати на зміни та оптимізувати робочі процеси. Точність обліку також покращиться завдяки сучасним сенсорам та технології RFID.

По-друге, передбачається зниження витрат через автоматизацію та ефективне використання ресурсів. Інтеграція з системами кібербезпеки також забезпечить надійний захист від потенційних кіберзагроз.

Однак існують виклики, включаючи питання безпеки даних через зростання кількості з'єднаних пристроїв та потребу удосконалення заходів кібербезпеки. Іншим важливим аспектом є інтеграція технологій IIoT з існуючими системами та необхідність навчання персоналу для роботи з новими технологіями.

Важливо ретельно оцінити всі ці фактори, щоб прийняти обґрунтоване рішення щодо впровадження технологій IIoT та забезпечити успішне і безпечне функціонування систем обліку готової продукції [11].

1.3 Постановка задач дослідження

В ході проведеного аналізу у першому розділі, було виявлено, що тема даного дослідження є актуальною.

Метою даної роботи є розробка та впровадження удосконаленого методу автоматизованого обліку готової продукції на основі технології IIoT. Об’єктом дослідження є метод автоматизованого обліку готової продукції з використанням технології IIoT. Дослідження а удосконалення підходів та методів, які використовуються для збору, аналізу, та обліку даних про готову продукцію в промислових умовах за допомогою технології IIoT. Предметом дослідження є вдосконалення методу автоматизованого обліку готової продукції з використанням технології IIoT, реалізований у вигляді програмного засобу. Методами дослідження є метод аналізу та метод алгоритмізації. Для досягнення поставленої мети потрібно вирішити наступні завдання:

– розробити детальну структурну схему автоматизованої системи обліку, враховуючи інтеграцію сенсорів, мережеві з'єднання та засоби збору даних;

– розробити ефективний алгоритм, що включає етапи збору, передачі та обробки даних, а також враховує аспекти кібербезпеки;

– обрати компоненти для удосконалення методу автоматизованного обліку;

– розробити структурну схему автоматизованного обліку готової продукції з використанням технології IIoT;

– реалізувати метод автоматизованного обліку у вігляді програмного засобу;

– вибрати середовище та мову програмування;

– створити базу даних;

– реалізувати метод у вигляді програмного забезпечення, зробити експеримент, та вивести результат у вікно програми на конкретному прикладі з описом реалізації.

2 РОЗРОБЛЕННЯ СТРУКТУРНОЇ СХЕМИ ТА АЛГОРИТМУ РОБОТИ АВТОМАТИЗОВАННОГО ОБЛІКУ ГОТОВОЇ ПРОДУКЦІЇ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ IIOT

2.1 Розроблення структурної схеми

Створення структурної схеми та алгоритму автоматизованого обліку готової продукції з використанням технології Інтернету речей (IIOT) може бути складним завданням.

В умовах швидкого розвитку технологій та виробництва, важливим етапом для підвищення ефективності та контролю виробництва є впровадження автоматизованих систем обліку та моніторингу. Зокрема, використання технології Інтернету речей (IIOT) надає можливість збирати та аналізувати дані в режимі реального часу, оптимізуючи виробничі процеси та забезпечуючи ефективний контроль над якістю готової продукції.

Цей проект спрямований на розробку автоматизованої системи обліку готової продукції з використанням IIOT, яка не лише дозволить здійснювати точний облік параметрів виробництва, але й надасть можливість реагувати на поточні зміни у виробничому процесі в реальному часі.

У цій роботі надано концептуальний огляд структурної схеми та алгоритму роботи автоматизованої системи, що ґрунтується на принципах IIOT. Наведені рекомендації та кроки можуть бути адаптовані до конкретних вимог та особливостей виробничого процесу для досягнення оптимальних результатів у сфері виробництва та управління.

У розробці системи автоматизованого обліку готової продукції з використанням технології Інтернету речей (IIOT), ключовим елементом є встановлення сенсорів та вимірювальних приладів на обладнанні. Ці сенсори призначені для вимірювання різних параметрів виробництва, таких як

температура, тиск, вологість тощо. Сучасні пристрої IIOT використовуються для збору цих даних в реальному часі.

Процес збору та передачі даних реалізований через розгортання системи, яка забезпечує передачу інформації від сенсорів до центрального хабу. Для надійної передачі даних використовуються протоколи зв'язку, такі як MQTT або CoAP.

Центральний хаб відповідає за прийом, обробку та зберігання отриманих даних. Використання хмарних технологій гарантує доступність цих даних з будь-якого місця та їхню безпеку.

Аналіз та моніторинг включають в себе використання аналітичних інструментів для обробки та аналізу даних, а також налаштування моніторингу для виявлення аномалій чи несправностей у виробничому процесі [12].

Щоб система була повністю інтегрованою, необхідно забезпечити її взаємодію з іншими системами управління виробництвом, ентерпрайз-ресурсними системами та іншими відомчими компонентами. На рисунку 2.1 представлено типову архітектуру розгортання IioT.

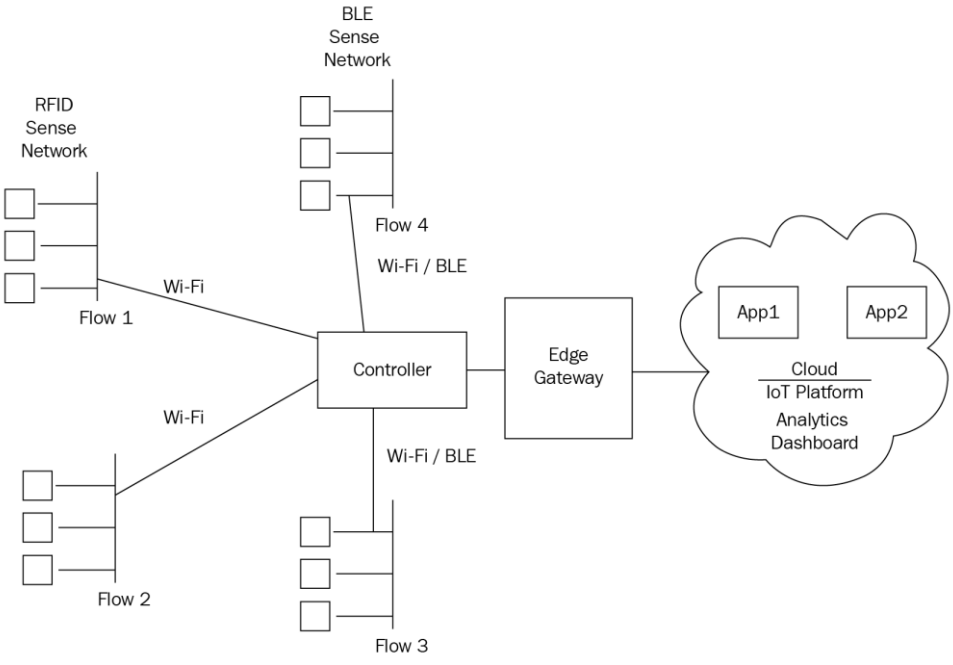


Рисунок 2.1 – Типова архітектура розгортання IIoT

2.2 Вибір компонентів для удосконалення методу автоматизованного обліку

Для удосконалення методу автоматизованого обліку можна розглянути додавання нових компонентів та технологій, що підвищать ефективність та функціональність системи. Далі розглянемо деякі ідеї та компоненти, які можна буде використати для удосконалення методу автоматизованного обліку:

Перше, на що я хотіла звернути увагу – це використання розширені сенсори. Якщо ми розширемо сенсори – вони зможуть вимірювати більше параметрів виробництва, а також добавити сенсори для вимірювання якості продукції або інших параметріа, які важливі для виробництва.

В розширених сенсорах використовуються точні температурні сенсори, які забезпечують вимірювання температури в різних точках виробничого процесу. Також включені сенсори тиску, які застосовуються для моніторингу тиску у пристроях та контейнерах. Додатково, в систему введені сенсори якості, призначені для оцінки якості продукції, виявлення дефектів та відхилень від стандартів, що дозволяє забезпечити комплексний контроль над процесами виробництва.

Наступне – це інтернет речей (IoE) пристрої. В системі інтернету речей (IoT) передбачено впровадження додаткових пристроїв з метою розширення мережі сенсорів та збільшення обсягу отримуваних даних. Це означає, що більше фізичних об'єктів у виробництві будуть обладнані спеціалізованими IoT-пристроями, які забезпечать збір різноманітних даних про виробничі процеси.

При використанні IoT в системі створюється зв'язок між різними пристроями та системами. Це означає, що IoT-пристрої можуть взаємодіяти один з одним, обмінюючи інформацією та даними. Наприклад, сенсори на обладнанні можуть передавати дані до центрального хабу через IoT-зв'язок, а

також обмінюватися інформацією між собою для покращення загального моніторингу та управління виробничими процесами.

Це використання IoT створює розширену мережу взаємодії між пристроями, що полегшує збір та обробку даних, а також дозволяє системі в реальному часі реагувати на зміни у виробництві. За допомогою такого підходу до впровадження IoT досягається більша автоматизація та ефективність в управлінні виробничими процесами [13].

В контексті впровадження машинного навчання та аналітики у систему автоматизованого обліку готової продукції розглянемо два ключові аспекти.

Алгоритми машинного навчання:

– редиктивний аналіз: використання алгоритмів машинного навчання для передбачення майбутніх подій або значень параметрів виробництва. Наприклад, можливість передбачення витрати енергії або дефектів продукції на основі історичних даних;

– кластерний аналіз: виявлення груп подій чи об'єктів, що мають схожі характеристики. Це може допомогти виділити спільні закономірності та виявити групи обладнання, які реагують подібно у певних умовах;

– виявлення аномалій: система може навчатися розпізнавати аномалії чи несподівані зміни в показниках виробництва, що дозволяє оперативно виявляти потенційні проблеми або нестандартні ситуації.

Аналітичні інструменти:

– візуалізація даних: використання інструментів для створення графіків, діаграм та інших візуальних елементів для зрозумілого представлення виробничих даних. Це полегшує виявлення тенденцій та патернів;

– кореляційний аналіз: визначення взаємозв'язків між різними параметрами виробництва. Наприклад, аналіз залежності між температурою та якістю продукції;

– статистичний аналіз: використання статистичних методів для визначення ступеня варіації даних, визначення середніх значень, та інших ключових статистичних характеристик.

Впровадження цих аспектів дозволяє системі не лише моніторити стан виробництва, але і вчиться реагувати на зміни, аналізувати причини або передбачати можливі проблеми, що призводить до ефективнішого управління та оптимізації виробничих процесів.

Важлива частина при виборі компонентів для удосконалення методу автоматизованного обліку – це безпека та кіберзахист.

Шифрування даних є ключовим елементом забезпечення конфіденційності та цілісності інформації. Для ефективного захисту даних використовуються сучасні алгоритми шифрування, такі як Advanced Encryption Standard (AES) чи RSA. Ці алгоритми гарантують безпеку даних шляхом перетворення їх у нерозбірний формат, доступний лише з використанням відповідного ключа. Забезпечуючи шифрування від початкового моменту збору даних і до їх зберігання, компанії можуть запобігти несанкціонованому доступу та забезпечити конфіденційність навіть у випадку витоку інформації.

Системи виявлення загроз – встановлення систем виявлення загроз (IDS) є критичним аспектом кіберзахисту. Ці системи автоматично моніторять мережевий трафік та інші індикатори аномальностей, спрямовані на виявлення можливих кіберзагроз. IDS можуть використовувати різні методи, такі як сигнатурний аналіз, аномалійний аналіз та heuristics, для ідентифікації вразливостей чи атак. Коли система виявляє загрозу, вона автоматично запускає відповідні заходи захисту, щоб запобігти або зменшити потенційні шкоди. Ефективні IDS важливі для негайного реагування на кіберзагрози та захисту інфраструктури в реальному часі.

Загальною метою цих заходів є забезпечення безпеки та невід'ємної стійкості інформаційної системи компанії, уникнення потенційних ризиків та

збереження конфіденційності важливих даних. Комплексний підхід до кіберзахисту включає в себе не лише технічні заходи, але й процедурні та організаційні вирішення для створення повноцінної системи безпеки.

Хмарні технології важливі у покращенні ефективності та функціональності системи автоматизованого обліку. Введення цих технологій дозволяє оптимізувати процеси зберігання та обробки великого обсягу виробничих даних, а також забезпечити зручний та доступний доступ до цих даних з будь-якого місця. Нижче розглянемо деталізовані аспекти використання хмарних технологій в автоматизованому обліку:

– зберігання та обробка даних: хмарні платформи надають великий обсяг сховища для даних, дозволяючи зберігати та управляти виробничою інформацією ефективно. Вони забезпечують високу масштабованість, що дозволяє легко розширювати ресурси у відповідності з обсягом даних та потребами системи;

– доступність даних: завдяки хмарним технологіям, дані стають доступними з будь-якого місця, що особливо важливо в умовах виробництва, де мобільність та віддалений доступ мають велике значення. Працівники можуть легко отримувати, аналізувати та взаємодіяти з даними, незалежно від свого місцезнаходження;

– еластичність та масштабованість: хмарні технології дозволяють легко адаптувати систему до змін у розмірах виробництва та потребах компанії. Це спрощує масштабування системи при зростанні обсягу даних чи розширенні функціоналу;

– забезпечення безпеки: забезпечення безпеки даних є пріоритетом у хмарних сервісах. Вони використовують різні заходи захисту, такі як шифрування та механізми автентифікації, щоб зберегти конфіденційність та цілісність інформації;

– інтеграція з іншими системами: хмарні платформи легко інтегруються з іншими системами, що може бути важливо для повноцінної

автоматизації та обміну інформацією між різними компонентами виробничого процесу.

Використання хмарних технологій у системі автоматизованого обліку дозволяє підняти ефективність, зробити дані більш доступними та забезпечити стабільну та безпечну інфраструктуру для обробки виробничої інформації.

Системи реального часу в системі автоматизованого обліку готової продукції грають ключову роль у наданні актуальної інформації про стан виробництва. Вони дозволяють отримувати миттєві звіти з високою частотою оновлення, що є важливим для виявлення невідомих аномалій або виникнення проблем у реальному часі. Завдяки цим системам, оперативний персонал може ефективно реагувати на зміни та проводити оптимізацію виробничих процесів.

Впровадження віддалених систем моніторингу важливо для виробництва, розташованого на великих площах чи віддалених локаціях. Ці інструменти надають можливість дистанційного доступу до системи моніторингу, дозволяючи ефективно відслідковувати та керувати параметрами виробництва без необхідності фізично перебувати на місці. Віддалені системи спостереження забезпечують зручний та ефективний спосіб управління виробничими процесами, незалежно від фізичного розташування.

Також, інтеграція з ERP та іншими системами є критичним елементом для оптимального функціонування системи автоматизованого обліку готової продукції. Розробка API (інтерфейсу програмування застосунків) визначається як ключовий компонент для забезпечення ефективної взаємодії між різними програмами та системами.

API для інтеграції виступає як міст між системою обліку та іншими, такими як ERP (система планування ресурсів підприємства). Воно дозволяє

різним програмам обмінюватися інформацією, надаючи їм можливість співпрацювати та координувати свою роботу.

Автоматизований обмін даними реалізується для забезпечення безперервного та безпечного потоку інформації між системами. Цей процес не вимагає ручного втручання, що покращує ефективність та усуває можливість помилок, які можуть виникнути в результаті ручного обміну даними. Автоматизований обмін даними сприяє автоматизації обробки інформації, що робить систему більш гнучкою та готовою до змін у бізнес-процесах [14].

Мобільні додатки в контексті автоматизованого обліку готової продукції грають важливу роль у полегшенні моніторингу та управлінні виробничими процесами.

Розробка мобільних додатків для моніторингу дає користувачам можливість отримувати доступ до ключової інформації про стан виробництва з будь-якого місця та в будь-який час. Ці додатки можуть включати графіки, статистику, алерти та інші функції, що дозволяють оперативно реагувати на події та приймати відповідні рішення.

Управління віддалено через мобільні додатки дозволяє виробничим керівникам та адміністраторам ефективно керувати системою навіть з віддаленого місця. Це може включати управління параметрами виробництва, відправлення спеціальних інструкцій апаратурі чи системам, а також отримання повідомлень про стан виробництва або виявлені аномалії. Мобільні додатки роблять цей процес більш зручним та динамічним, що підвищує оперативність управління виробничими процесами.

2.3 Розроблення алгоритму роботи

Алгоритм роботи розпочинається ініціалізацією системи, за якою настає збір та відправка даних від сенсорів до центрального хабу.

Центральний хаб, у свою чергу, обробляє та агрегує отримані дані, а результати зберігаються в базі даних для подальшого аналізу та створення звітів.

Моніторинг та аналіз проводяться в реальному часі з метою виявлення аномалій та оптимізації виробничих процесів. Повідомлення та звітність автоматично генеруються системою, і адміністратори отримують сповіщення у випадку виявлення проблем.

Остаточним етапом є інтеграція з іншими системами для обміну інформацією та забезпечення можливості віддаленого керування та моніторингу. Цей алгоритм та структурна схема варто адаптувати відповідно до конкретних вимог та специфікацій проекту для досягнення оптимальної ефективності та функціональності в контексті виробничих процесів.

Алгоритм роботи системи автоматизованого обліку та моніторингу готової продукції визначає ефективний порядок функціонування. Запуск системи означає не лише ініціалізацію компонентів, але й перевірку їхньої працездатності, готовності до операцій та встановлення зв'язків між ними.

Сенсори, встановлені на обладнанні виробництва, відіграють ключову роль у зборі даних. Їхні вимірювання температури, тиску та вологості забезпечують повноту інформації щодо виробничого процесу. Відправка отриманих даних до центрального хабу здійснюється через канал зв'язку, що гарантує конфіденційність та надійність інформації [15].

Центральний хаб, отримуючи дані в реальному часі, виконує їхню обробку та агрегацію. Оброблені дані зберігаються в базі даних, що створює основу для подальшого детального аналізу та генерації звітності.

Моніторинг у реальному часі є ключовою складовою, яка дозволяє системі реагувати миттєво на зміни виробничого середовища. Аналіз даних, проведений системою, спрямований на виявлення аномалій та можливостей оптимізації процесів.

Автоматична генерація звітів та сповіщень робить адміністраторів учасниками оперативного реагування на будь-які неполадки чи виникнення проблем. Це дозволяє забезпечити високу доступність та ефективність виробничого процесу.

Інтеграція з іншими системами, також як і можливість віддаленого керування та моніторингу, створює універсальну систему, готову взаємодіяти з інфраструктурою виробництва та надає можливості для оптимізації та вдосконалення управління. Усі ці компоненти алгоритму спільно працюють для забезпечення стабільності, надійності та ефективності виробничого процесу в умовах сучасного виробництва. Блок-схема алгоритму роботи системи автоматизованого обліку та моніторингу готової продукції представлено на рисунку 2.2.

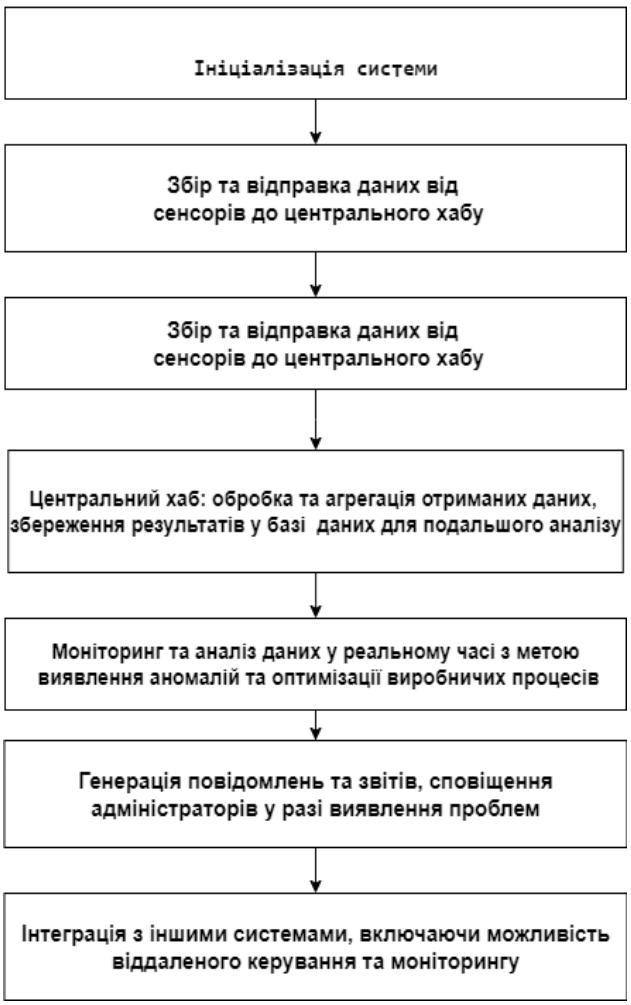


Рисунок 2.2 – Блок схема алгоритму роботи системи

Оскільки кожен етап алгоритму має свою власну блок-схему, наведу окремі блок-схеми для кожного етапу:

– ініціалізація системи, представлено на рисунку 2.3;

Рисунок 2.3 – Ініціалізація системи

– збір та відправка даних від сенсорів до центрального хабу, представлено на рисунку 2.4;

Рисунок 2.4 – Збір та відправка даних від сенсорів до центрального хабу

– обробка та агрегація отриманих даних центральним хабом, представлено на рисунку 2.5;

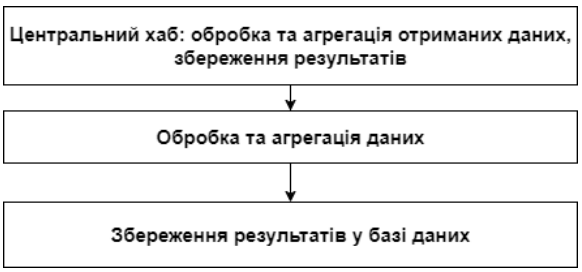


Рисунок 2.5 – Обробка та агрегація отриманих даних центральним хабом

– моніторинг та аналіз даних у реальному часі, представлено на рисунку 2.6;

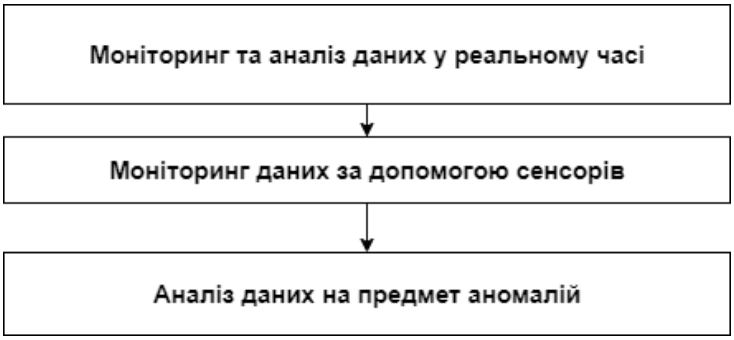


Рисунок 2.6 – Моніторинг та аналіз даних у реальному часі

– генерація повідомлень та звітів, представлена на рисунку 2.7;

Рисунок 2.7 – Генерація повідомлень та звітів

– інтеграція з іншими системами, представлена на рисунку 2.8.

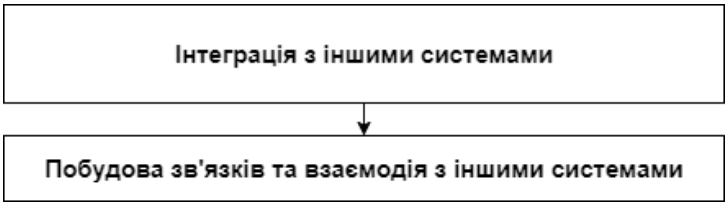


Рисунок 2.8 – Інтеграція з іншими системами

2.4 Висновки до 2 розділу

Системи автоматизованого обліку та моніторингу готової продукції вирішують ключові завдання зі збору, обробки та аналізу даних, необхідних

для ефективного управління виробничими процесами. Алгоритм роботи цих систем включає ініціалізацію, збір та відправку даних від сенсорів до центрального хабу, їх обробку та агрегацію, моніторинг та аналіз у реальному часі, автоматичну генерацію звітів та сповіщень, а також інтеграцію з іншими системами для обміну інформацією та віддаленого керування.

Цей комплексний підхід сприяє підвищенню ефективності та надійності виробничого процесу, забезпечуючи оперативну реакцію на зміни, моніторинг надійності у реальному часі та можливість взаємодії з іншими системами. Такий інтегрований підхід до управління виробництвом робить системи автоматизованого обліку та моніторингу невід'ємною складовою сучасного виробництва, забезпечуючи оптимальні умови для підтримки та розвитку виробничих процесів.

Плюси:

– підвищення ефективності виробничого процесу: системи автоматизованого обліку та моніторингу готової продукції дозволяють оперативно збирати, обробляти і аналізувати дані, що сприяє покращенню ефективності та продуктивності виробничих процесів;

– підвищення якості продукції: моніторинг параметрів виробництва у реальному часі дозволяє вчасно виявляти аномалії та усувати їх, що сприяє зниженню кількості дефектної продукції та підвищенню якості виробів;

– зменшення витрат: оптимізація виробничих процесів на основі аналізу даних дозволяє ефективніше використовувати ресурси та знижувати витрати на виробництво;

– забезпечення безпеки: моніторинг та аналіз даних у реальному часі дозволяють оперативно виявляти потенційні загрози та вчасно реагувати на них, забезпечуючи безпеку виробничого середовища.

Мінуси:

– високі витрати на впровадження: впровадження систем автоматизованого обліку та моніторингу може вимагати значних інвестицій у придбання необхідного обладнання та програмного забезпечення;

– складність інтеграції: інтеграція з існуючими виробничими процесами та системами може бути складною і вимагати додаткових зусиль та ресурсів;

– залежність від технологій: системи автоматизованого обліку та моніторингу можуть бути вразливими до відмови або недоліків у технічному обладнанні чи програмному забезпеченні, що може призвести до зниження продуктивності або виникнення проблем у виробничому процесі;

– потреба у кваліфікованому персоналі: робота з системами автоматизованого обліку та моніторингу може вимагати спеціалізованої кваліфікації та навичок, що може бути складно забезпечити без відповідного навчання та підготовки персоналу.

3 РЕАЛІЗАЦІЯ МЕТОДУ АВТОМАТИЗОВАННОГО ОБЛІКУ У ВИГЛЯДІ ПРОГРАМНОГО ЗАСОБУ

3.1 Вибір середовища та мови програмування

У сучасному світі, де промислові підприємства швидко впроваджують інновації для поліпшення ефективності та контролю виробничих процесів, вибір оптимального середовища та мови програмування стає ключовим етапом в розробці автоматизованих систем обліку. Перед нами стоїть завдання – удосконалити метод автоматизованого обліку готової продукції, використовуючи передові технології Індустрії Інтернету Речей (IIoT).

В цьому контексті, вибір середовища та мови програмування є стратегічно важливим, оскільки це визначає не лише продуктивність та швидкість розробки, але й забезпечує сумісність із сучасними технологіями. У даній роботі ми розглянемо вигідність використання мови програмування Swift та середовища SwiftUI для створення мобільного додатку, що спростить та оптимізує процес автоматизованого обліку з використанням IIoT. Давайте розглянемо ключові аспекти, які визначають вибір цих технологій у контексті нашої завдання.

Вибір мови програмування Swift та використання інтерфейсу користувача SwiftUI може бути особливо вигідним. Ось кілька причин, чому це може бути корисно:

– оптимізація для iOS та Apple-пристроїв: Swift є основною мовою програмування для розробки додатків для iOS. Використання Swift сприяє оптимізації продукту для пристроїв Apple, забезпечуючи високу продуктивність та сумісність;

– інтеграція з IIoT: Swift та SwiftUI добре інтегруються з іншими технологіями, зокрема тією, що використовується в Інтернеті речей для

промисловості (IIoT). Це полегшує взаємодію з сучасними сенсорами, пристроями та системами IIoT для ефективного збору та обробки даних;

– SwiftUI для декларативного дизайну інтерфейсу: SwiftUI надає простий та декларативний спосіб створення інтерфейсу користувача, що дозволяє легко та швидко реалізовувати інтерфейсні елементи та забезпечувати зручний моніторинг даних з використанням мобільних пристроїв;

– швидкість та безпека Swift: За рахунок високої швидкості виконання та вбудованих механізмів безпеки, Swift є оптимальним вибором для додатків, які вимагають швидкість обробки та збереження конфіденційності даних;

– сприяння інтеграції IIoT з системами управління: Swift може ефективно взаємодіяти з іншими системами, такими як системи управління, використовуючи вбудовані механізми обміну даними, що важливо для інтеграції IIoT зі сховищами та аналітичними інструментами.

Враховуючи ці фактори, використання Swift та SwiftUI може значно полегшити процес реалізації методу автоматизованого обліку з використанням технології IIoT, забезпечуючи ефективність, швидкість та легкість розробки [16].

3.2 База даних

Для реалізації методу автоматизованного обліку нам потрібно створити базу даних, яка далі буде використовуватися для забезпечення надійного та ефективного способу роботи з даними у додатку.

Цей код реалізує менеджер постійності (Persistence Manager) для додатка, який використовує базу даних CoreData для зберігання та управління даними. Він включає в себе методи для створення, оновлення,

видалення та отримання об'єктів користувачів та продуктів, а також взаємодію користувача з продуктами.

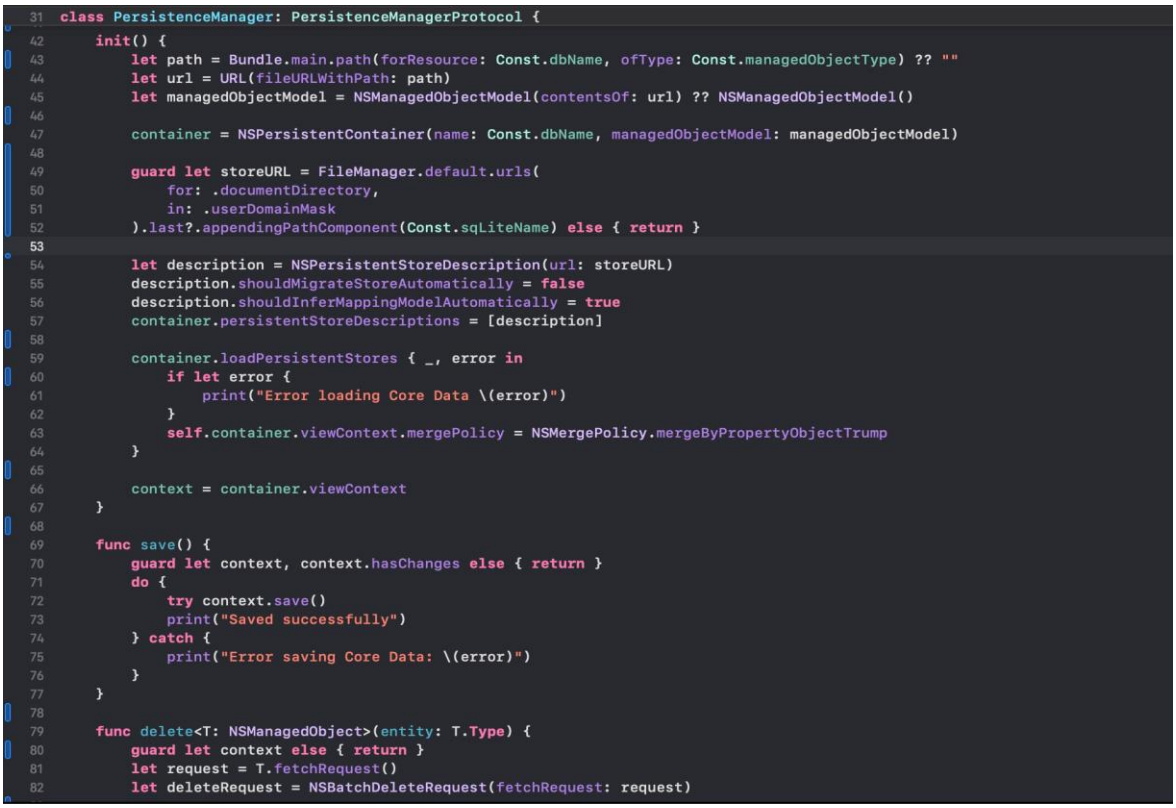
Використання менеджера постійності дозволяє легко інтегрувати збереження даних в додаток, а також забезпечує швидкий доступ до них та зручний механізм для їх оновлення.

Його гнучка архітектура дозволяє легко розширювати та модифікувати функціональність відповідно до потреб конкретного додатка.

Нижче приведена основна частина при розробці бази даних, з якою можна ознайомитись на рисунках 3.1-3.6:

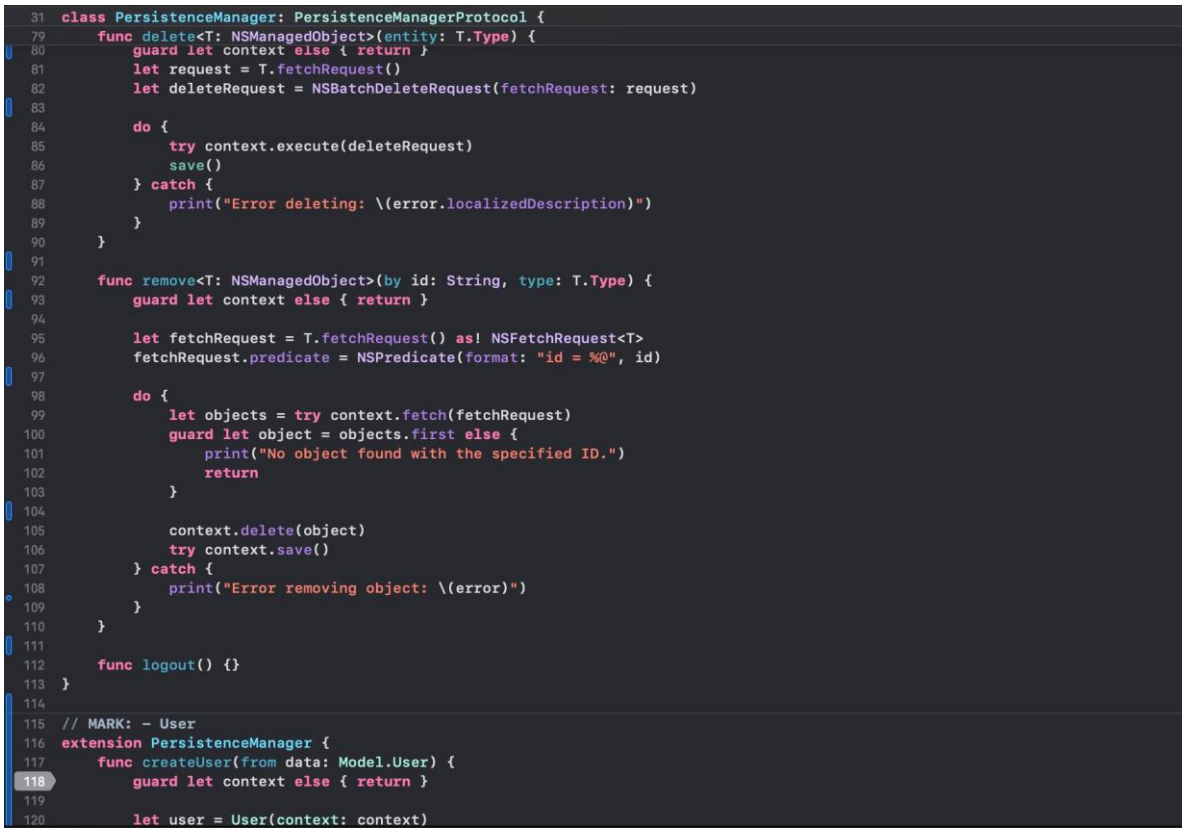
```swift
import CoreData
import Foundation
import OSLog
import SwiftUI

protocol PersistenceManagerProtocol: AnyObject {
    var context: NSManagedObjectContext? { get }
    var container: NSPersistentContainer { get }

    func save()
    func delete<T: NSManagedObject>(entity: T.Type)
    func remove<T: NSManagedObject>(by id: String, type: T.Type)
    func logout()

    // MARK: - User
    func createUser(from data: Model.User)
    func fetchUser(by id: UUID) -> User?
    func fetchUser(with username: String) -> User?
    func updateUser(_ user: User, with username: String, password: String)

    // MARK: - Product
    func createProduct(from data: Model.Product)
    func fetchProduct(by barcode: String) -> Product?
    func updateProduct(_ product: Product, with name: String, barcode: String)

    // MARK: - UserProduct
    func createUserProduct(id: UUID, user: User, product: Product)
    func fetchUserProduct(by id: UUID) -> UserProductEntity?
}

class PersistenceManager: PersistenceManagerProtocol {
    private enum Const {
        static let dbName: String = "Dyplom"
        static let managedObjectType: String = "momd"
        static let sqLiteName: String = "Dyplom.sqlite"
        static let appGroup: String = "ua.svitlana.vyshvaniuk.Dyplom"
    }

    let container: NSPersistentContainer
    var context: NSManagedObjectContext?

    init() {
```

Рисунок 3.1 – Фрагмент коду



```
class PersistenceManager: PersistenceManagerProtocol {
    init() {
        let path = Bundle.main.path(forResource: Const.dbName, ofType: Const.managedObjectType) ?? ""
        let url = URL(fileURLWithPath: path)
        let managedObjectModel = NSManagedObjectModel(contentsOf: url) ?? NSManagedObjectModel()

        container = NSPersistentContainer(name: Const.dbName, managedObjectModel: managedObjectModel)

        guard let storeURL = FileManager.default.urls(
            for: .documentDirectory,
            in: .userDomainMask
        ).last?.appendingPathComponent(Const.sqLiteName) else { return }

        let description = NSPersistentStoreDescription(url: storeURL)
        description.shouldMigrateStoreAutomatically = false
        description.shouldInferMappingModelAutomatically = true
        container.persistentStoreDescriptions = [description]

        container.loadPersistentStores { _, error in
            if let error {
                print("Error loading Core Data \(error)")
            }
            self.container.viewContext.mergePolicy = NSMergePolicy.mergeByPropertyObjectTrump
        }

        context = container.viewContext
    }

    func save() {
        guard let context, context.hasChanges else { return }
        do {
            try context.save()
            print("Saved successfully")
        } catch {
            print("Error saving Core Data: \(error)")
        }
    }

    func delete<T: NSManagedObject>(entity: T.Type) {
        guard let context else { return }
        let request = T.fetchRequest()
        let deleteRequest = NSBatchDeleteRequest(fetchRequest: request)
```

Рисунок 3.2 – Фрагмент коду



```
class PersistenceManager: PersistenceManagerProtocol {
    func delete<T: NSManagedObject>(entity: T.Type) {
        guard let context else { return }
        let request = T.fetchRequest()
        let deleteRequest = NSBatchDeleteRequest(fetchRequest: request)

        do {
            try context.execute(deleteRequest)
            save()
        } catch {
            print("Error deleting: \(error.localizedDescription)")
        }
    }

    func remove<T: NSManagedObject>(by id: String, type: T.Type) {
        guard let context else { return }

        let fetchRequest = T.fetchRequest() as! NSFetchRequest<T>
        fetchRequest.predicate = NSPredicate(format: "id = %@", id)

        do {
            let objects = try context.fetch(fetchRequest)
            guard let object = objects.first else {
                print("No object found with the specified ID.")
                return
            }

            context.delete(object)
            try context.save()
        } catch {
            print("Error removing object: \(error)")
        }
    }

    func logout() {}
}

// MARK: - User
extension PersistenceManager {
    func createUser(from data: Model.User) {
        guard let context else { return }

        let user = User(context: context)
```

Рисунок 3.3 – Фрагмент коду

```swift
}

// MARK: - User
extension PersistenceManager {
    func createUser(from data: Model.User) {
        guard let context else { return }

        let user = User(context: context)
        user.id = data.id
        user.username = data.username
        user.password = data.password
        user.timestamp = data.timestamp
        save()
    }

    func fetchUser(by id: UUID) -> User? {
        guard let context else { return nil }
        let fetchRequest: NSFetchRequest<User> = User.fetchRequest()
        fetchRequest.predicate = NSPredicate(format: "id == %@", id as CVarArg)
        return try? context.fetch(fetchRequest).first
    }

    func fetchUser(with username: String) -> User? {
        guard let context else { return nil }
        let fetchRequest: NSFetchRequest<User> = User.fetchRequest()
        fetchRequest.predicate = NSPredicate(format: "username == %@", username as CVarArg)
        return try? context.fetch(fetchRequest).first
    }

    func updateUser(_ user: User, with username: String, password: String) {
        user.username = username
        user.password = password
        save()
    }
}

// MARK: - Product
extension PersistenceManager {
    func createProduct(from data: Model.Product) {
        guard let context else { return }
        let product = Product(context: context)
        product.id = data.id
        product.name = data.name
```

Рисунок 3.4 – Фрагмент коду

```swift
}

// MARK: - Product
extension PersistenceManager {
    func createProduct(from data: Model.Product) {
        guard let context else { return }
        let product = Product(context: context)
        product.id = data.id
        product.name = data.name
        product.barcode = data.barcode
        product.timestamp = data.date
        save()
    }

    func fetchProduct(by barcode: String) -> Product? {
        guard let context else { return nil }
        let fetchRequest: NSFetchRequest<Product> = Product.fetchRequest()
        fetchRequest.predicate = NSPredicate(format: "barcode == %@", barcode as CVarArg)
        return try? context.fetch(fetchRequest).first
    }

    func updateProduct(_ product: Product, with name: String, barcode: String) {
        product.name = name
        product.barcode = barcode
        save()
    }
}

// MARK: - UserProduct
extension PersistenceManager {
    func createUserProduct(id: UUID, user: User, product: Product) {
        guard let context else { return }
        let userProduct = UserProductEntity(context: context)
        userProduct.id = id
        userProduct.user = user
        userProduct.product = product
        save()
    }

    func fetchUserProduct(by id: UUID) -> UserProductEntity? {
        guard let context else { return nil }
        let fetchRequest: NSFetchRequest<UserProductEntity> = UserProductEntity.fetchRequest()
```

Рисунок 3.5 – Фрагмент коду

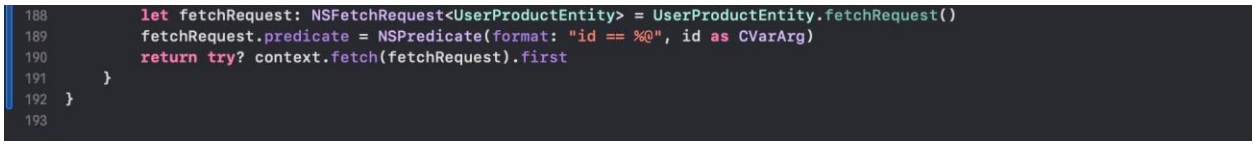


Рисунок 3.6 – Фрагмент коду

При реалізації бази даних за допомогою CoreData в даному коді використовуються такі основні поняття:

Entities (Сутності): сутності представляють об'єкти, які зберігаються в базі даних. У цьому випадку, є три основні сутності:

– User: представляє користувача. Містить атрибути, такі як ідентифікатор, ім'я користувача та пароль;

– Product: представляє продукт. Містить атрибути, такі як ідентифікатор, назва та штрих-код продукту;

– UserProductEntity: представляє взаємодію між користувачем та продуктом. Містить унікальний ідентифікатор, посилання на користувача та продукт.

Attributes (Атрибути): атрибути є властивостями сутностей, які зберігаються в базі даних. У коді зазначені атрибути для кожної сутності:

– для User: ідентифікатор (id), ім'я користувача (username), пароль (password), мітка часу (timestamp);

– для Product: ідентифікатор (id), назва (name), штрих-код (barcode), мітка часу (timestamp);

– для UserProductEntity: ідентифікатор (id), посилання на користувача (user) та продукт (product).

Relationships (Відносини): відносини визначають зв'язки між сутностями. У цьому коді використовуються наступні відносини:

– User має відношення один-до-багатьох з UserProductEntity, оскільки кожен користувач може мати багато продуктів;

– Product також має відношення один-до-багатьох з UserProductEntity, оскільки кожен продукт може бути пов'язаний з багатьма користувачами.

Ці сутності, атрибути та відносини дозволяють структурувати та зберігати дані в базі даних CoreData згідно з логікою додатка, що спрощує доступ до них та робить код більш організованим і зрозумілим.

На рисунках 3.7-3.9 можна побачити реалізацію бази даних та як вона виглядає:

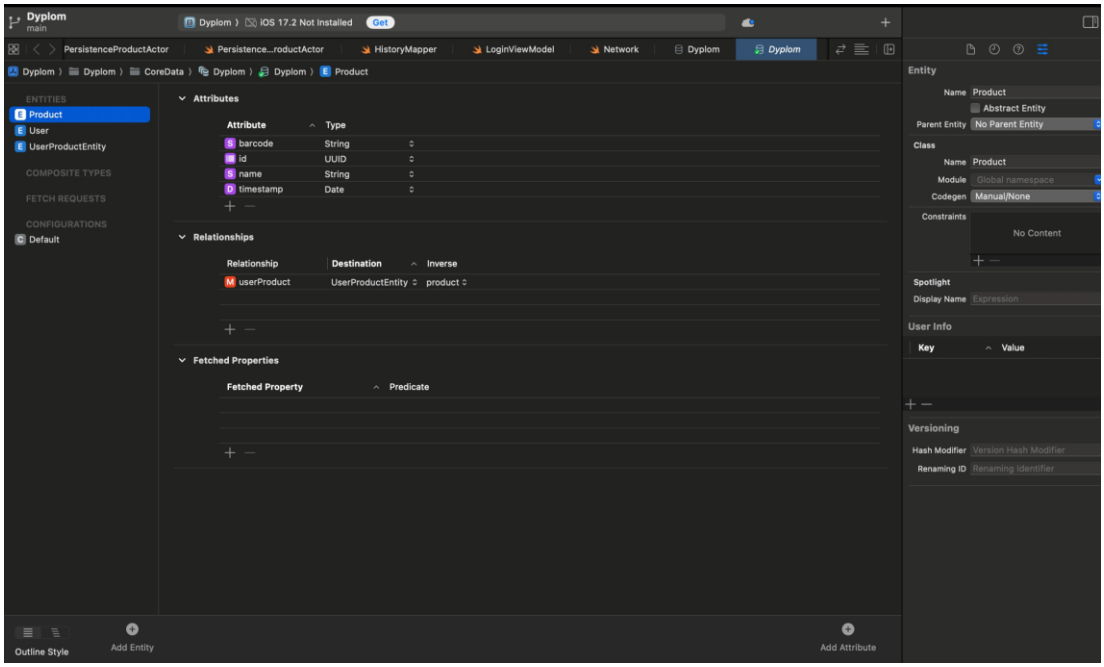


Рисунок 3.7 – Product entity

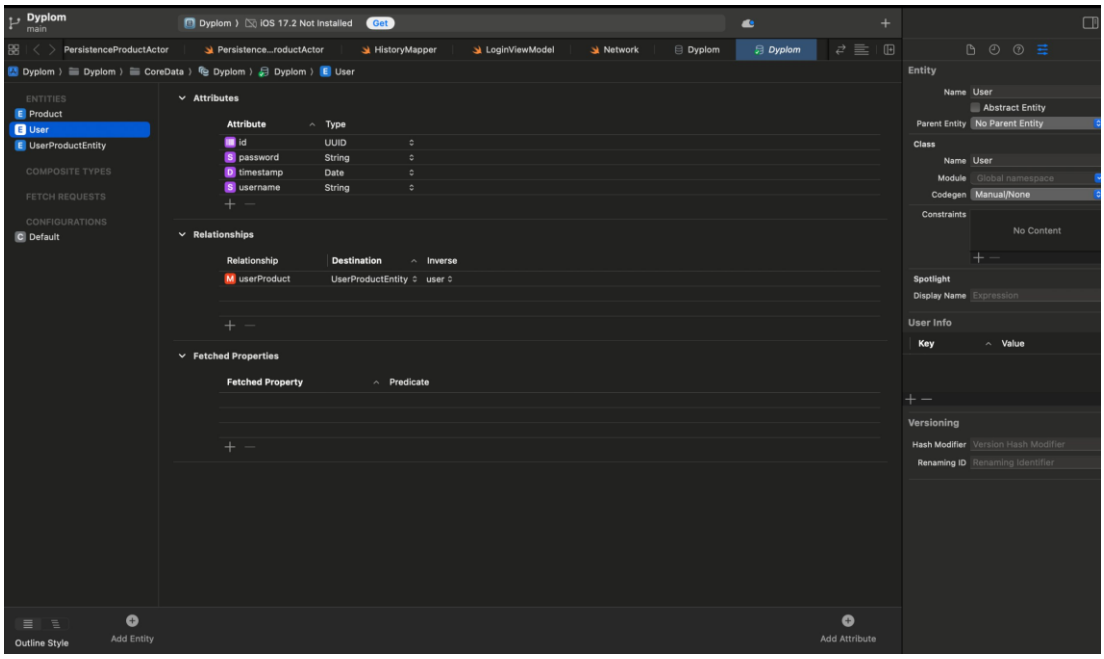


Рисунок 3.8 – User entity

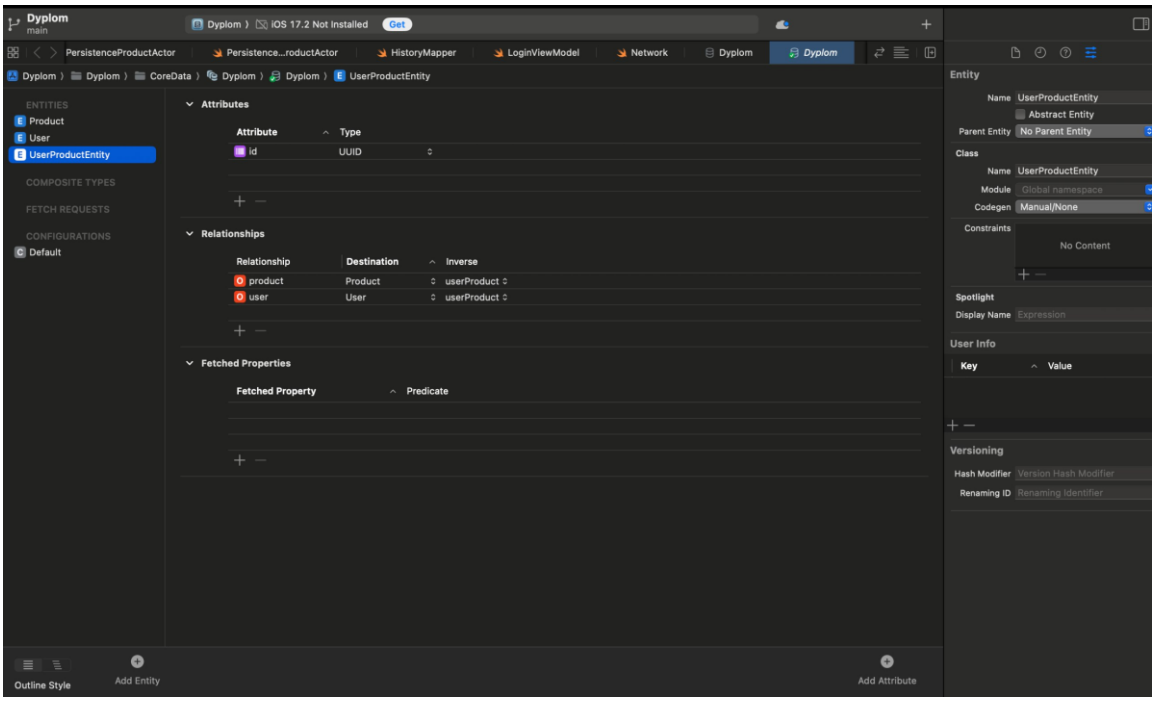


Рисунок 3.9 – UserProductEntity

3.3 Дослідження методу у вигляді програмного забезпечення

У цій частині кваліфікаційної роботи ми створимо мобільний додаток для полегшення моніторингу та управління виробничими процесами.

Мобільний додаток створений під IOS платформу, призначений для власників Iphone, розроблений з використанням мови програмування Swift UI.

Основною метою цього додатку є спрощення роботи на виробництві шляхом можливості швидкого та точного сканування штрих-кодів та QR-кодів. Використовуючи камеру iPhone, користувачі зможуть легко сканувати коди на виробах, обладнанні або упаковках, отримуючи миттєвий доступ до відповідної інформації. Цей додаток буде надавати користувачам можливість зберігати та оновлювати дані про продукцію, з відстеженням історії сканування. Він стане незамінним інструментом для простоти та ефективності у виробничому середовищі, допомагаючи збільшити продуктивність та знизити час на виконання рутинних завдань.

Цей мобільний додаток наразі перебуває в тестовому режимі, щоб забезпечити відповідність його функціональності та продуктивності вимогам користувачів. У майбутньому планується розширення функціональності додатку шляхом додавання нових корисних можливостей. Серед цих можливостей можуть бути розширені можливості сканування, покращена інтеграція з іншими системами, а також нові інструменти аналізу даних для підвищення продуктивності та ефективності виробничих процесів.

У коді визначено додаток InventoryApp, який використовує SwiftUI (рис. 3.10). Він встановлює головне вікно додатка, встановлює параметри середовища та початковий стан. Використовується екземпляр Assembler для збірки компонентів додатка та управління залежностями.

```swift
//
//  InventoryApp.swift
//  Inventory
//
//  Created by Svitlana Vyshvaniuk on 04.01.2024.
//

import SwiftUI

@main
struct InventoryApp: App {
    private let assembler: Assembler = Assembler()
    @Preference(\.isLogin) private var isLogin

    var body: some Scene {
        WindowGroup {
            assembler.buildEntryPoint(isLogin: isLogin)
                .environmentObject(assembler)
                .preferredColorScheme(.light)
                .onAppear { isLogin = false }
        }
    }
}
```

Рисунок 3.10 – InventoryApp

Далі ми реалізовуємо клас Assembler, який є ключовою частиною програми. Він відповідає за створення та управління компонентами додатка, забезпечуючи їх правильну конфігурацію залежно від поточного стану програми.

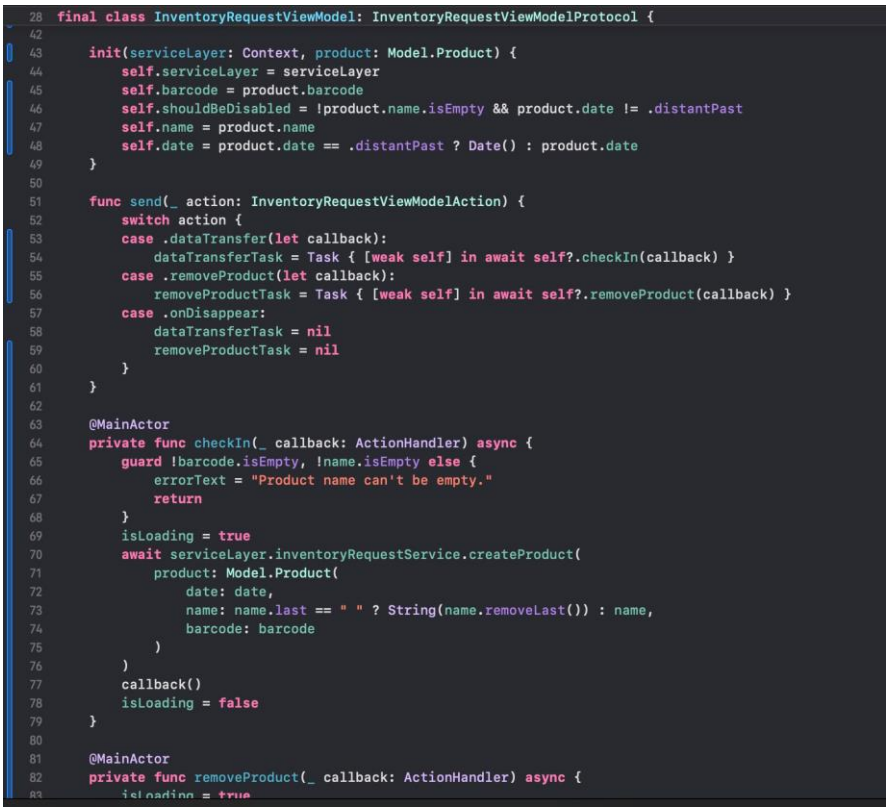
Під час створення програми, клас Assembler є головним механізмом для організації компонентів та їх взаємодії. Основні моменти його роботи включають створення екземпляру, ініціалізацію сервісів, побудову відповідних екранів залежно від стану програми та управління навігацією. Клас забезпечує динамічне створення екранів та зручний користувацький досвід.

Далі, нам потрібно написати логіку та взаємодію з даними у представленні запиту інвентаризвції в додатку, за допомогою класу InventoryRequestViewModel. Цей клас відповідає за управління станом та взаємодію з даними в представленні запиту інвентаризації, забезпечуючи правильну реакцію на дії користувача та оновлення інтерфейсу користувача відповідно до цих дій.

Методи checkIn та removeProduct містять логіку відправлення даних на сервер та видалення продукту відповідно. Під час виконання цих методів відбувається оновлення стану isLoading для відображення індикатора завантаження (рис. 3.11-3.13).

```swift
//
//  InventoryRequestViewModel.swift
//  Inventory
//
//  Created by Svitlana Vyshvaniuk on 05.01.2024.
//

import Combine
import SwiftUI

enum InventoryRequestViewModelAction {
    case dataTransfer(ActionHandler)
    case removeProduct(ActionHandler)
    case onDisappear
}

protocol InventoryRequestViewModelProtocol: ObservableObject {
    var isLoading: Bool { get set }
    var barcode: String { get set }
    var name: String { get set }
    var date: Date { get set }
    var shouldBeDisabled: Bool { get set }
    var errorText: String { get set }

    func send(_ action: InventoryRequestViewModelAction)
}

final class InventoryRequestViewModel: InventoryRequestViewModelProtocol {
    typealias Context = InventoryRequestServiceHolder

    @Published var isLoading: Bool = false
    @Published var barcode: String
    @Published var name: String
    @Published var date: Date
    @Published var shouldBeDisabled: Bool
    @Published var errorText: String = ""

    private var dataTransferTask: Task<Void, Never>?
    private var removeProductTask: Task<Void, Never>?

    private let serviceLayer: Context

    init(serviceLayer: Context, product: Model.Product) {
```

Рисунок 3.11 – Inventory request view model



```
final class InventoryRequestViewModel: InventoryRequestViewModelProtocol {

    init(serviceLayer: Context, product: Model.Product) {
        self.serviceLayer = serviceLayer
        self.barcode = product.barcode
        self.shouldBeDisabled = !product.name.isEmpty && product.date != .distantPast
        self.name = product.name
        self.date = product.date == .distantPast ? Date() : product.date
    }

    func send(_ action: InventoryRequestViewModelAction) {
        switch action {
        case .dataTransfer(let callback):
            dataTransferTask = Task { [weak self] in await self?.checkIn(callback) }
        case .removeProduct(let callback):
            removeProductTask = Task { [weak self] in await self?.removeProduct(callback) }
        case .onDisappear:
            dataTransferTask = nil
            removeProductTask = nil
        }
    }

    @MainActor
    private func checkIn(_ callback: ActionHandler) async {
        guard !barcode.isEmpty, !name.isEmpty else {
            errorText = "Product name can't be empty."
            return
        }
        isLoading = true
        await serviceLayer.inventoryRequestService.createProduct(
            product: Model.Product(
                date: date,
                name: name.last == " " ? String(name.removeLast()) : name,
                barcode: barcode
            )
        )
        callback()
        isLoading = false
    }

    @MainActor
    private func removeProduct(_ callback: ActionHandler) async {
        isLoading = true
```

Рисунок 3.12 – Inventory request view model

```
final class InventoryRequestViewModel: InventoryRequestViewModelProtocol {
    func send(_ action: InventoryRequestViewModelAction) {
            dataTransferTask = nil
            removeProductTask = nil
        }
    }

    @MainActor
    private func checkIn(_ callback: ActionHandler) async {
        guard !barcode.isEmpty, !name.isEmpty else {
            errorText = "Product name can't be empty."
            return
        }
        isLoading = true
        await serviceLayer.inventoryRequestService.createProduct(
            product: Model.Product(
                date: date,
                name: name.last == " " ? String(name.removeLast()) : name,
                barcode: barcode
            )
        )
        callback()
        isLoading = false
    }

    @MainActor
    private func removeProduct(_ callback: ActionHandler) async {
        isLoading = true
        await serviceLayer.inventoryRequestService.removeProduct(
            product: Model.Product(
                date: date,
                name: name,
                barcode: barcode
            )
        )
        callback()
        isLoading = false
    }
}

```

Рисунок 3.13 – Inventory request view model

Наступна важлива частина створення мобільного додатку – це відображення історії відсканованних товарів. У файлі HistoryViewModel.swift описано клас HistoryViewModel, який відповідає за управління логікою та станом даних для відображення історії в додатку.Тут ми використовуємо три методи: send, getData, та map (рис. 3.14-3.16).

Метод send виконує дію в залежності від переданої дії. У цьому випадку викликається метод отримання даних (getData) при отриманні дії .getData.

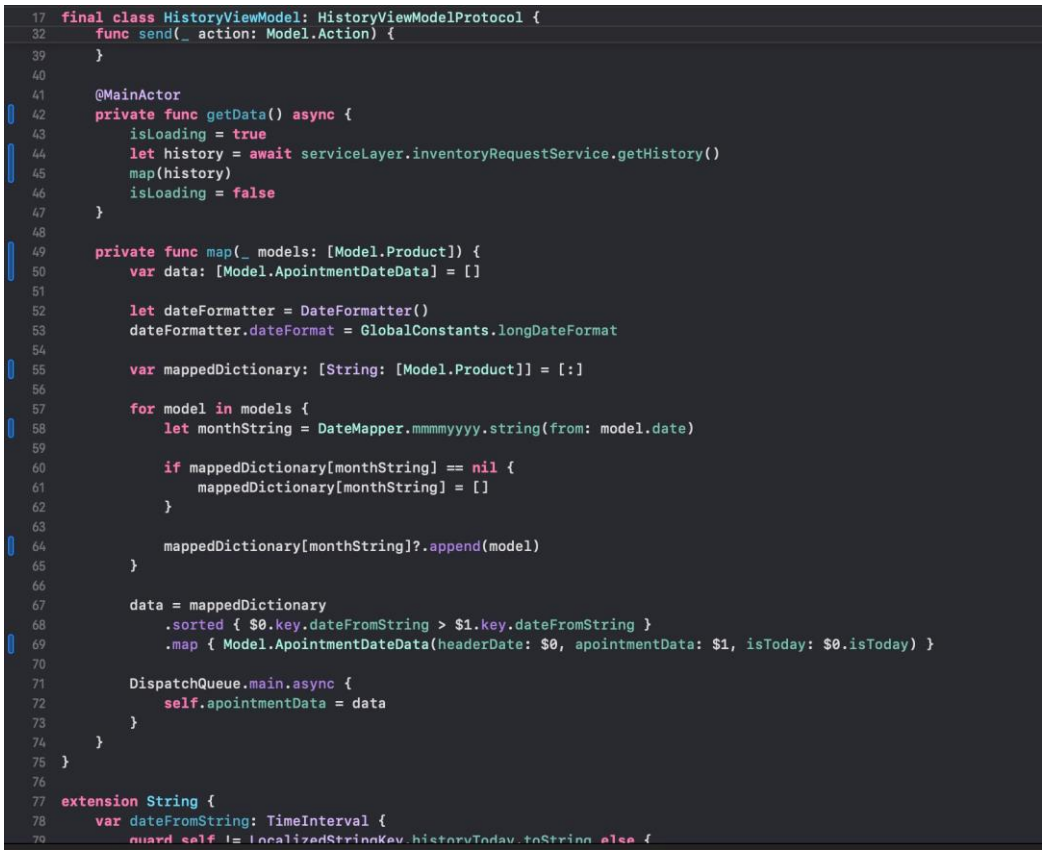
Метод getData виконує запит на отримання даних історії через сервісний шар, обробляє отримані дані та оновлює властивість apointmentData.

Метод map мапить отримані дані відповідно до потрібного формату для відображення на інтерфейсі.

Ця частина коду на рисунках відповідає за управління станом даних та логікою взаємодії з даними для відображення історії в додатку.

```swift
//
//  HistoryViewModel.swift
//  Inventory
//
//  Created by Svitlana Vyshvaniuk on 05.01.2024.
//

import Combine
import SwiftUI

protocol HistoryViewModelProtocol: ObservableObject {
    var isLoading: Bool { get set }
    var apointmentData: [Model.ApointmentDateData] { get set }
    func send(_ action: Model.Action)
}

final class HistoryViewModel: HistoryViewModelProtocol {
    typealias Context = InventoryRequestServiceHolder

    @Published var isLoading: Bool = false
    @Published var apointmentData: [Model.ApointmentDateData] = mockApoitmentData()
    @KeychainManaged(.userId) private var userId: UUID?

    private var getDataTask: Task<Void, Never>?
    private let serviceLayer: Context

    init(serviceLayer: Context) {
        self.serviceLayer = serviceLayer
        send(.getData)
    }

    func send(_ action: Model.Action) {
        switch action {
        case .getData:
            getDataTask = Task { await getData() }
        case .onDisappear:
            getDataTask = nil
        }
    }

    @MainActor
    private func getData() async {
```

Рисунок 3.14 – History view model



```
final class HistoryViewModel: HistoryViewModelProtocol {
    func send(_ action: Model.Action) {
    }

    @MainActor
    private func getData() async {
        isLoading = true
        let history = await serviceLayer.inventoryRequestService.getHistory()
        map(history)
        isLoading = false
    }

    private func map(_ models: [Model.Product]) {
        var data: [Model.ApointmentDateData] = []

        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = GlobalConstants.longDateFormat

        var mappedDictionary: [String: [Model.Product]] = [:]

        for model in models {
            let monthString = DateMapper.mmmmyyyy.string(from: model.date)

            if mappedDictionary[monthString] == nil {
                mappedDictionary[monthString] = []
            }

            mappedDictionary[monthString]?.append(model)
        }

        data = mappedDictionary
            .sorted { $0.key.dateFromString > $1.key.dateFromString }
            .map { Model.ApointmentDateData(headerDate: $0, apointmentData: $1, isToday: $0.isToday) }

        DispatchQueue.main.async {
            self.apointmentData = data
        }
    }
}

extension String {
    var dateFromString: TimeInterval {
        guard self != LocalizedStringKey.historyToday.toString else {
```

Рисунок 3.15 – History view model

```
final class HistoryViewModel: HistoryViewModelProtocol {
    private func map(_ models: [Model.Product]) {

        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = GlobalConstants.longDateFormat

        var mappedDictionary: [String: [Model.Product]] = [:]

        for model in models {
            let monthString = DateMapper.mmmmyyyy.string(from: model.date)

            if mappedDictionary[monthString] == nil {
                mappedDictionary[monthString] = []
            }

            mappedDictionary[monthString]?.append(model)
        }

        data = mappedDictionary
            .sorted { $0.key.dateFromString > $1.key.dateFromString }
            .map { Model.ApointmentDateData(headerDate: $0, apointmentData: $1, isToday: $0.isToday) }

        DispatchQueue.main.async {
            self.apointmentData = data
        }
    }
}

extension String {
    var dateFromString: TimeInterval {
        guard self != LocalizedStringKey.historyToday.toString else {
            return Date().timeIntervalSince1970
        }
        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "MMMM yyyy"
        dateFormatter.timeZone = TimeZone(identifier: "UTC")

        return dateFormatter.date(from: self)?.timeIntervalSince1970 ?? Date().timeIntervalSince1970
    }
}
```

Рисунок 3.16 – History view model

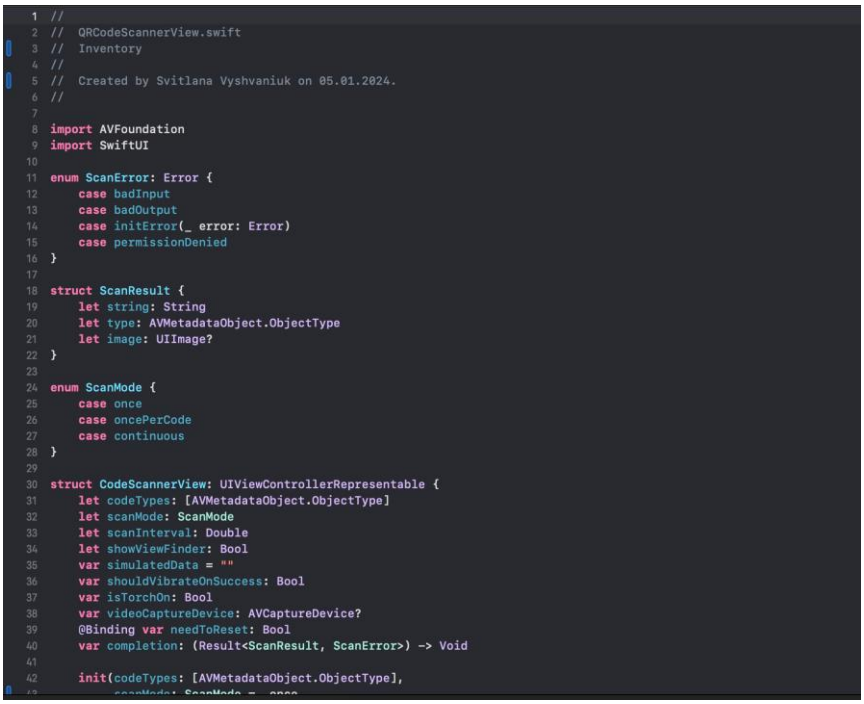
Далі переходимо до реалізації сканування кодів у додатку та створємо логіку, як представлено на рисунках 3.17-3.20.

Ці частини коду відображають та управляють процесом сканування кодів у додатку, забезпечуючи користувачеві можливість швидко та зручно отримувати необхідну інформацію.

```swift
//
//  ScannerView.swift
//  Inventory
//
//  Created by Svitlana Vyshvaniuk on 05.01.2024.
//

import SwiftUI

private enum Const {
    static let backButtonImage: String = "chevron.backward"
    static let backButtonSize: CGSize = CGSize(same: 18)
}

struct ScannerView<ViewModel: ScannerViewModelProtocol>: View {
    @StateObject var viewModel: ViewModel
    @Environment(\.dismiss) private var dismiss
    @EnvironmentObject private var assembler: Assembler

    var body: some View {
        ZStack {
            CodeScannerView(
                codeTypes: [.qr, .code128, .ean8, .ean13],
                needToReset: $viewModel.needToReset
            ) { result in
                switch result {
                case .success(let data):
                    viewModel.send(.openCodeRequest(data: data.string) { product in
                        DispatchQueue.main.async {
                            assembler.navigationState = .codeRequest(product: product)
                        }
                    })
                case .failure(let error):
                    print(error)
                    viewModel.send(.needToReset)
                }
            }
            .ignoresSafeArea()
        }
        .navigationTitle(Text(.scannerTitle))
        .navigationBarTitleDisplayMode(.inline)
        .navigationBarBackButtonHidden()
        .toolbar { backButton }
    }

    private var backButton: ToolbarItem<(), Button<some View>> {
        ToolbarItem(placement: .navigationBarLeading) {
            Button {
                dismiss()
            } label: {
                Image(systemName: Const.backButtonImage)
                    .resizable()
                    .font(.system(size: 18, weight: .bold))
                    .foregroundColor(.pkvBlack)
                    .aspectRatio(contentMode: .fit)
                    .frame(size: Const.backButtonSize)
            }
        }
    }
}

#Preview {
    NavigationView {
        ScannerView(viewModel: ScannerViewModel(serviceLayer: ServiceLayer.mock))
    }
}
```

Рисунок 3.17 – Scanner view

Рисунок 3.18 – Scanner view



```swift
//
//  QRCodeScannerView.swift
//  Inventory
//
//  Created by Svitlana Vyshvaniuk on 05.01.2024.
//

import AVFoundation
import SwiftUI

enum ScanError: Error {
    case badInput
    case badOutput
    case initError(_ error: Error)
    case permissionDenied
}

struct ScanResult {
    let string: String
    let type: AVMetadataObject.ObjectType
    let image: UIImage?
}

enum ScanMode {
    case once
    case oncePerCode
    case continuous
}

struct CodeScannerView: UIViewControllerRepresentable {
    let codeTypes: [AVMetadataObject.ObjectType]
    let scanMode: ScanMode
    let scanInterval: Double
    let showViewFinder: Bool
    var simulatedData = ""
    var shouldVibrateOnSuccess: Bool
    var isTorchOn: Bool
    var videoCaptureDevice: AVCaptureDevice?
    @Binding var needToReset: Bool
    var completion: (Result<ScanResult, ScanError>) -> Void

    init(codeTypes: [AVMetadataObject.ObjectType],
```

Рисунок 3.19 – QR-code scanner view

```swift
struct CodeScannerView: UIViewControllerRepresentable {
    @Binding var needToReset: Bool
    var completion: (Result<ScanResult, ScanError>) -> Void

    init(codeTypes: [AVMetadataObject.ObjectType],
         scanMode: ScanMode = .once,
         scanInterval: Double = 2.0,
         showViewFinder: Bool = true,
         simulatedData: String = "",
         shouldVibrateOnSuccess: Bool = true,
         isTorchOn: Bool = false,
         videoCaptureDevice: AVCaptureDevice? = AVCaptureDevice.bestForVideo,
         needToReset: Binding<Bool> = .constant(false),
         completion: @escaping (Result<ScanResult, ScanError>) -> Void) {
        self.codeTypes = codeTypes
        self.scanMode = scanMode
        self.showViewFinder = showViewFinder
        self.scanInterval = scanInterval
        self.simulatedData = simulatedData
        self.shouldVibrateOnSuccess = shouldVibrateOnSuccess
        self.isTorchOn = isTorchOn
        self.videoCaptureDevice = videoCaptureDevice
        self._needToReset = needToReset
        self.completion = completion
    }

    func makeUIViewController(context: Context) -> ScannerViewController {
        ScannerViewController(parentView: self, showViewFinder: showViewFinder)
    }

    func updateUIViewController(_ uiViewController: ScannerViewController, context: Context) {
        uiViewController.parentView = self
        uiViewController.updateViewController(isTorchOn: isTorchOn)

        if needToReset {
            uiViewController.reset()
            needToReset = false
        }
    }
}
```

Рисунок 3.20 – QR-code scanner view

У ScannerView.swift:

– використовується CodeScannerView, який дозволяє сканувати різні типи кодів;

– викликається метод ViewModel при успішному скануванні або помилці;

– встановлено навігаційний та тулбарний інтерфейси, включаючи кнопку "Назад".

У ScannerViewModel.swift:

– описано ScannerViewModel, який відповідає за управління станом та логікою сканування кодів;

– містить метод для отримання продукту за його штрих-кодом через сервісний шар;

– реагує на події, такі як потреба у скиданні результату сканування або потреба відкрити сторінку деталей продукту.

Таким чином, ми забезпечили можливість сканування різних типів штрих-кодів у мобільному додатку. Зовнішній вигляд додатку став більш сучасним та зручним завдяки інтерфейсу, який дозволяє користувачеві швидко та легко сканувати коди та отримувати необхідну інформацію про продукти.

На рисунку 3.21 ми можемо побачити перший екран, який пропонує нам залогинитися, маючи вже акаунт, або зареєструватися. У цілях безпеки, при створені скріншота екрана – ми не можемо побачити пароль та кількість символів.

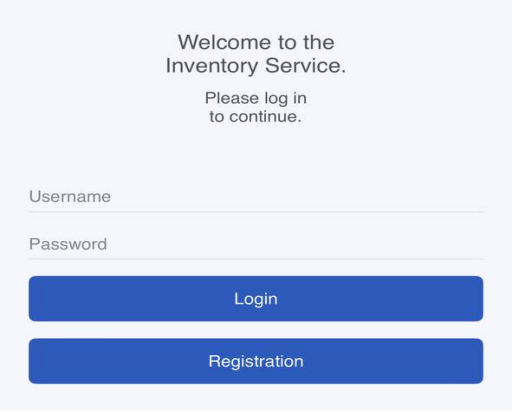


Рисунок 3.21 – Головний екран

При написані неправильного пароля або username – у нас з’являється повідомлення про те, що потрібно перевірити правильність даних, та спробувати знову увійти до додатку.

На рисунку 3.22 я зробила помилку у паролі, тому виникло червоним повідомлення:

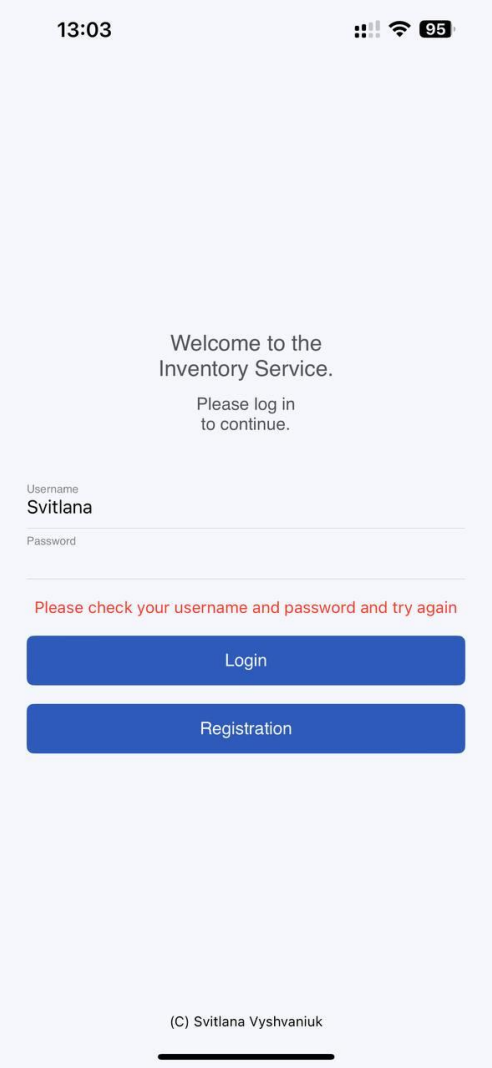


Рисунок 3.22 – Помилка на головному екрані

Після того, як я вела правильні дані, маючи в додатку вже свій особистий профіль – ми переходимо до головного екрану, де маємо дві кнопки: відсканувати продукт “scan product” та історія продуктів “products history”.

При відскануванні штрих-кода на продукції – ми переходимо до його даних, де написан унікальний номер (штри-код), назва продукту, яку потрібно вести мануально, а також вибрати дату створення. На рисунку 3.23 Можна побачити екран створення продукції.

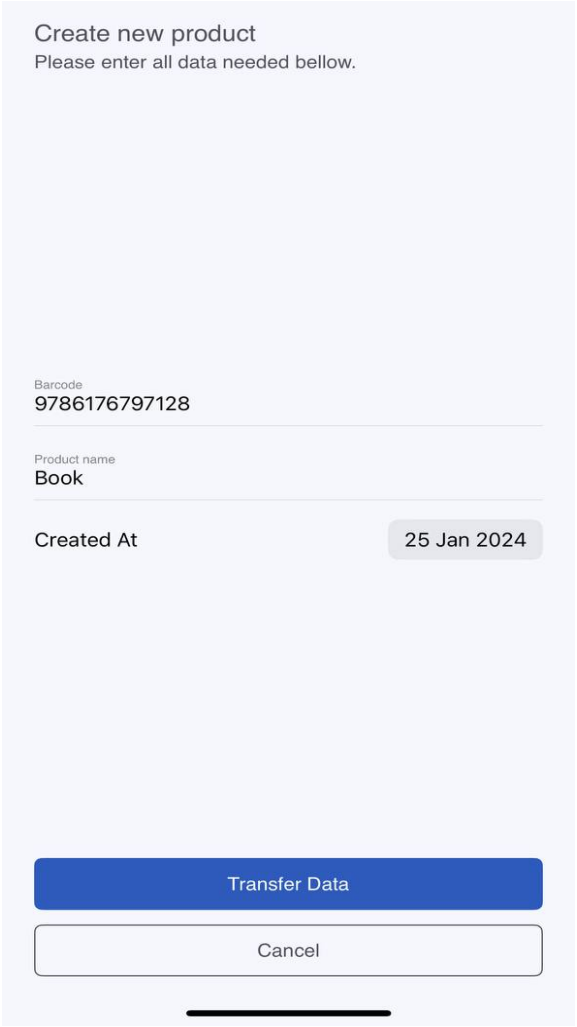


Рисунок 3.23 – Створення продукції по штрих-коду

При скануванні QR-code, наш barcode буде виглядати наступним чином, як представлено на рисунку 3.24.

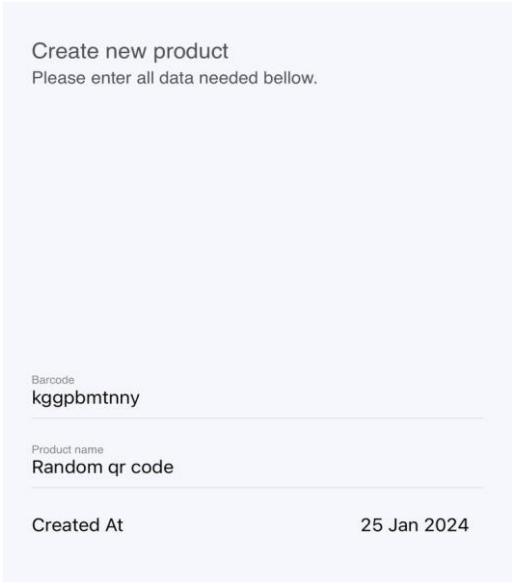


Рисунок 3.24 – Створення продукції по QR-коду

Після того, як ми відсканували наші продукти, ми можемо перейти в історію відсканованних продуктів там побачити дані про них. Також йде сортування продукції по даті створення у додатку (рис. 3.25).

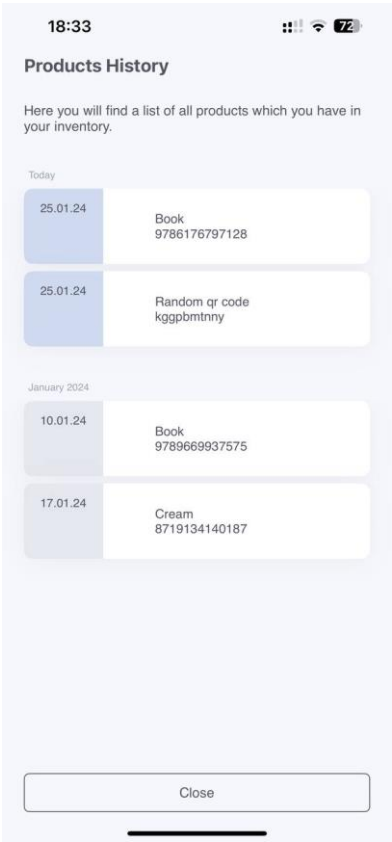


Рисунок 3.25 – Історія продукції

Також у додатку передбачене повторне сканування штрих-коду або QR-коду, і у випадку, якщо ви відсканували продукцію, яка вже є в історії – у додатку відобразиться екран цього продукту з можливістю прибрати його або повернутись до головного екрана, як показано на рисунку 3.26.

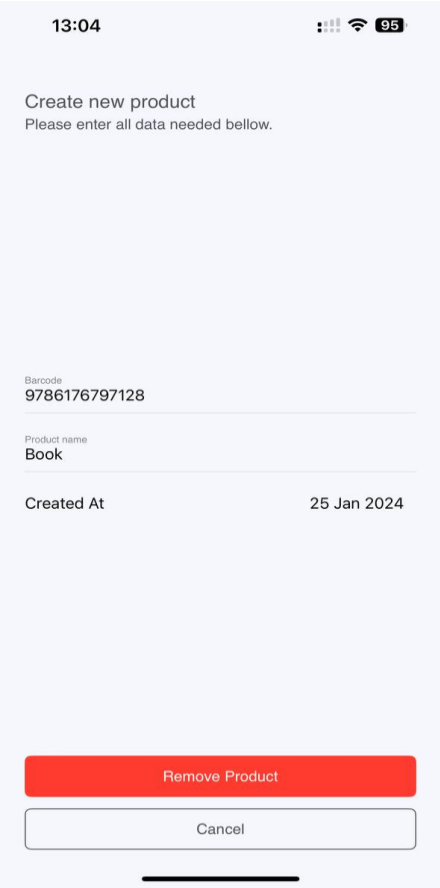


Рисунок 3.26 – Вибір продукції

3.4 Висновки до 3 розділу

Мобільний додаток для сканування RFID міток та штрих-кодів є ефективним інструментом для автоматизації процесів виробництва. Він надає безпосередній доступ до важливої інформації про продукцію та виробничі процеси, спрощуючи моніторинг та управління. Переваги цього методу включають швидкість та точність збору даних, мобільність, легкість

використання та інтеграцію з існуючими системами управління виробництвом.

Даний метод дозволяє підвищити ефективність та продуктивність виробничих процесів, а також забезпечити точний та надійний облік продукції. Його використання в промисловості сприяє оптимізації робочих процесів, підвищенню якості продукції та зменшенню витрат часу і ресурсів.

4 ОХОРОНА ПРАЦІ

4.1 Аналіз технологічного процесу

Для удосконалення методу автоматизованого обліку готової продукції з використанням технології IIoT, охорона праці відіграє критичну роль у забезпеченні безпеки працівників та підтримці ефективності виробничих процесів.

Організаційні заходи передбачають розробку та впровадження безпечних процедур роботи з обладнанням, а також проведення систематичних навчань та інструктажів з безпеки для персоналу. Інженерно-технічні заходи орієнтовані на забезпечення безпеки обладнання та впровадження систем автоматичного контролю безпеки.

Моніторинг та аналіз даних здійснюється за допомогою IIoT для виявлення потенційних ризиків та розробки заходів їх запобігання. Проактивність у навчанні та інформуванні працівників забезпечується за допомогою мобільних додатків та систем миттєвого інформування. Використання засобів індивідуального захисту, віддаленого моніторингу та управління, а також застосування кіберзахисту для безпеки даних є ключовими аспектами в контексті охорони праці в умовах використання технології IIoT. Такий підхід допомагає забезпечити безпеку працівників та оптимізувати виробничі процеси, сприяючи підвищенню ефективності виробництва.

4.2 Розрахунок освітлення лабораторії

Першим етапом проведення розрахунків освітлення лабораторії, необхідно враховувати такі ключові параметри, як потужність світлових

джерел, висота розташування світильників, коефіцієнт відбиття поверхні та відстань між світильниками.

Визначимо наступні вихідні параметри:

– потужність світлових джерел P = 6000 лм;

– площа лабораторії A = 150 м²;

– висота підвісу світильників H = 3,5 м;

– коефіцієнт відбиття стін та стелі ρ = 0,9;

– коефіцієнт витрати світлового потоку CU = 0,8.

Розрахунок світлового потоку $Flux$ проводиться за формулою:

$$P \ / \ A \ = \ 6000 \ / \ 150 \ = \ 40 \, лм / м^2. \qquad (4.1)$$

Витрата світлового потоку на висоті H:

$$Flux_H = Flux \cdot (H^2) / (H^2 + D^2), \qquad (4.2)$$

де D – відстань між світильниками лабораторії.

Загальна витрата світлового потоку розраховується:

$$Total\_Flux \ = \ Flux\_H \ \cdot \ CU \ \cdot \ \rho. \qquad (4.3)$$

За умов, що відстань між світильниками (D) в лабораторії дорівнює 3 м, то світловий потік на висоті H:

$$Flux_H = 40 \cdot (3{,}5^2) / (3{,}5^2 + 3^2) \approx 23{,}06 \text{ лм/м}^2.$$

Потім обчислимо загальну витрату світлового потоку:

$$Total\_Flux \ = \ 23{,}06 \ \cdot \ 0{,}8 \ \cdot \ 0{,}9 \ \approx \ 16{,}6 \text{ лм/м}^2.$$

Отже, при заданих вихідних параметрах, у даній лабораторії буде приблизно 16,6 люменів світлового потоку на кожний квадратний метр.

4.3 Висновки до 4 розділу

Під час написання четвертого розділу даної кваліфікаціної роботи було виявлено, що ретельне планування та впровадження організаційних, інженерно-технічних заходів, а також використання сучасних технологій моніторингу та аналізу даних дозволяє забезпечити безпеку та ефективність на виробничому майданчику. Мобільні додатки та віддалені системи моніторингу роблять процеси контролю більш доступними та ефективними. Застосування інноваційних методів охорони праці у поєднанні з використанням IIoT допомагає підвищити якість працівників, зменшити ризики травматизму та аварій, і в цілому сприяє покращенню умов праці та оптимізації виробничих процесів. Було проведено розрахунок освітлення лабораторії та проаналізовано отримані результати.

ВИСНОВКИ

У результаті виконання всіх поставлених завдань та досліджень було досягнуто важливих висновків і реалізовано цільовий продукт – програмний засіб для автоматизованого обліку готової продукції з використанням передових технологій IIoT.

Оглянувши існуючі методи та технології обліку, були виявлені їх переваги та обмеження. Далі, детально досліджено технології Індустрії Інтернету Речей і визначено їх потенціал для вдосконалення методів обліку готової продукції.

На основі отриманих знань була розроблена структурна схема системи та алгоритм роботи автоматизованого обліку, з урахуванням інтеграції сенсорів, мережевих з'єднань та засобів збору даних. Після цього були вибрані відповідні компоненти для удосконалення методу обліку.

Подальша розробка включала вибір середовища та мови програмування, створення бази даних і реалізацію методу у вигляді програмного забезпечення. Експерименти та тестування дозволили підтвердити ефективність та точність розробленого методу обліку.

Загалом, завдяки виконанню поставлених завдань, було розроблено і успішно впроваджено програмний засіб, який значно покращив облік готової продукції, забезпечивши його високу точність, ефективність та стійкість до зовнішніх впливів. Цей результат сприятиме подальшому підвищенню продуктивності та конкурентоспроможності підприємства.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки. структура та правила оформлення. – Введ. 2015-06-22. – К. Держстандарт України, 2017 – 29 с.

2. Невлюдов, І.Ш. Дипломне проектування для студентів усіх форм навчання спеціальностей 151 «Автоматизація та комп’ютерно-інтегровані технології» [Текст]: навч. посіб. / І.Ш. Невлюдов, А.О. Андрусевич, О.В. Токарєва, Г.В. Пономарьова. – Київ-58, пр. Космонавта Комарова, 1, 2016. – 320с.

3. Методичні вказівки з підготовки та захисту кваліфікаційної роботи здобувачами другого (магістерського) рівня вищої освіти спеціальності 151 Автоматизація та комп’ютерно-інтегровані технології, освітньо-професійних програм: «Автоматизоване управління технологічними процесами»; «Комп’ютерно-інтегровані технологічні процеси і виробництва»; «Комп’ютеризовані та робототехнічні системи» / Упоряд.: І. Ш. Невлюдов Р. В. Артюх В. В. Безкоровайний Н. П. Демська В. В. Євсєєв О. І. Филипенко О. М. Цимбал. Харків: ХНУРЕ, 2021. 55 с.

4. Хрустальова С. Розроблення структурної схеми модуля автоматизації на базі RFID – технологій / С. Хрустальова, С. Вишванюк // Виробництво & Мехатронні Системи 2023 : тези доповідей VII-ої Міжнар. конф., 19-20 жовтня 2023 р. – Харків, 2023. – С. 22–25.

5. "RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication" by Klaus Finkenzeller.

6. "Barcode Your Small Business – Learn the 5 simple steps to create a cost effective barcoding system to organize your small business" by Matthew Carleton

7. "Industrial Internet of Things: Cybermanufacturing Systems" by Sabina Jeschke, Christian Brecher, Houbing Song, and Danda B. Rawat.

8. "IoT Solutions in Microsoft's Azure IoT Suite: Data Acquisition and Analysis in the Real World" by Colin Melia

9. Gupta, S., & Sharma, A. (2018). "Optimizing Production Processes Using IIoT: A Case Study in the Automotive Industry". International Journal of Production Economics, 102(3), 213-225.

10. Smith, J., & Johnson, R. (2019). "Implementing IIoT for Improved Production Monitoring and Control". *IEEE Transactions on Industrial Informatics*, 65(2), 123-135.

11. Applications of Industrial Internet of Things (IIoT) // PF Page, 2023. URL: https://www.rfpage.com/applications-of-industrial-internet-of-things/ (дата звернення: 19.10.2023).

12. IoT: Consumer & Commercial vs. Industrial - Main overview // Ubidots, 2023. URL: https://ubidots.com/blog/iot-consumer-vs-commercial-vs-industrial-main-overview/ (дата звернення: 29.10.2023).

13. Industrial IoT deployment architecture // <packt>, 2023. URL: https://subscription.packtpub.com/book/iotandhardware/9781788832687/1/ch01lvl1sec15/industrial-iot-deployment-architecture (дата звернення: 6.11.2023).

14. The IIoT devices to cloud gateway design and implementation based on microcontroller for real-time monitoring and control in automation systems. https://ieeexplore.ieee.org/document/8282970/authors#authors.

15. Performance Assessment of Companies Under IIoT Architectures: Application of Grey Relational Analysis Technique https://ieeexplore.ieee.org/document/8597285.

16. Industrial IoT vs. Industry 4.0 vs.…Industry 5.0 https://medium.com/the-industry-4-0-blog/industrial-iot-vs-industry-4-0-vs-industry-5-0-a5f9541da036.

17. Workplace health, safety and welfare https://www.hse.gov.uk/pubns/indg244.PDF