# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

APS – Advancing Planning & Scheduling

ASP.NET – Active Server Pages

CLR – Common Language Runtime

CRM – Customer Relationship Management

ERP – Enterprise Resource Planning

GUI – Graphic User Interface

IIoT – Industrial Internet of Things

IoT – Internet of Things

LAS – Application Server

LPWAN – low-power wide-area network

MES – Manufacturing Execution System

MVVM – Model-View-ViewModel

PDM – Product Data Management

PLM – Product Lifecycle Management

SCADA – Supervisory Control Data Acquistition

SCM – Supply Chain Management

WLAN – Wireless Local Area Network

АСУТП – Автоматизована Система Управління Технологічним Процесом

САПР – Система Автоматизованого Проектування

# ВСТУП

Через сучасну ситуацію у світі, яка пов'язана з вірусом і карантином, величезна кількість заводів і підприємств втратила робочу силу. Виробництво сповільнилося, попит збільшився, а пропозиції немає. Отже, відмінним варіантом може бути віддалене управління виробництвом або зовсім його автоматизація. За допомогою сучасних технологій можна запрограмувати машину на безперервну роботу в автоматичному режимі. А людина може керувати всіма механізмами, перебуваючи у себе вдома, використовуючи при цьому комп'ютер або смартфон.

На сучасному етапі розвитку людства системи автоматизації виробничих процесів ґрунтуються на використанні комп'ютерів і різного програмного забезпечення. Вони сприяють скороченню ступеня участі людей в діяльності або повністю виключають його. До завдань автоматизації виробничих процесів входить підвищення якості виконання операцій, скорочення часу, якого на них потрібно, зниження вартості, збільшення точності і стабільності дій.

У наш час популярними є мікроконтролери та одноплатні комп’ютери, що мають достатньо обчислювальної потужності для реалізації функцій промислових систем автоматизації.

**Мета:** створення мобільного застосунку по віддаленому керуванню виробництвом.

**Об’єкт:** методи та засоби керування виробництвом на основі використання інтелектуальних систем.

**Предметом дослідження є** архітектура, принципи та апаратне забезпечення системи керування виробництвом.

Для досягнення поставленої мети необхідно виконати наступні **завдання**:

- дослідження керуючих систем для підприємств;
- визначення функціональності та режимів роботи керуючих систем;
- дослідження засобів керування системами виробництва;

- проведення порівняльного аналізу популярних моделей керуючих систем з метою визначення їх основних переваг та недоліків;
- аналіз апаратного забезпечення, що необхідне для побудови керуючої системи;
- розробка схеми підключення обраного апаратного забезпечення;
- розробки програмного забезпечення для керування;
- розробка застосунку для керуванням програмним забезпеченням системи;
- розробка застосунку для керування апаратним забезпеченням.

**Наукова новизна:** розробка програми для дистанційного керування підприємством за допомогою смартфону або планшету.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ, ДОСЛІДЖЕННЯ ОБ’ЄКТУ ТА ОГЛЯД ДІЮЧИХ СИСТЕМ.

## 1.1 Поняття IIoT

Промисловий Інтернет речей (IIoT) об'єднує машини, хмарні обчислення, аналітику та співробітників, щоб підвищити ефективність промислових процесів. Завдяки IIoT промислові компанії можуть оцифровувати процеси, модифікувати бізнес-моделі, а також підвищувати продуктивність та ефективність, водночас скорочуючи витрати. Ці ресурсомісткі компанії, що працюють у різних галузях промисловості, таких як виробництво, енергетика, сільське господарство, транспорт та комунальні послуги, розробляють проекти Інтернету речей, покликані пов'язати мільярди пристроїв та приносити користь у різних сценаріях використання, включаючи аналітику профілактичного обслуговування та контролю якості, моніторинг стану ресурсів та оптимізацію процесів.

Зазвичай на промисловому об'єкті розміщено тисячі датчиків, що генерують дані. За допомогою IIoT виробники, наприклад, можуть об'єднати технічні дані з однієї виробничої лінії, заводу або мережі об'єктів, таких як промислові підприємства, а також складальні та нафтопереробні заводи, для активного підвищення продуктивності шляхом виявлення та попередження потенційних факторів, що гальмують виробничий процес, збоїв, прогалин у виробництві та проблем якості. Об'єднання даних мережі об'єктів також може підвищити ефективність контролю потоку матеріалів та сприяти виявленню, виявленню та усуненню факторів, що гальмують виробництво або постачання на ранніх етапах, а також оптимізувати роботу техніки та обладнання на об'єктах[1].

Існує три основних сценарія використання промислового Інтернету речей:

1. Прогнозування якості. Прогнозна аналітика якості (рис. 1.1) дозволяє отримувати практичну інформацію з промислових джерел даних, таких як виробниче обладнання, умови довкілля та спостереження працівників, щоб оптимізувати якість продукції, що виробляється. Завдяки AWS IoT промислові виробники можуть створювати прогнозні моделі для контролю якості, які допомагають їм підвищити якість продукції. Чим вище якість продукції, тим більша задоволеність клієнтів і тим менше буде повернень.

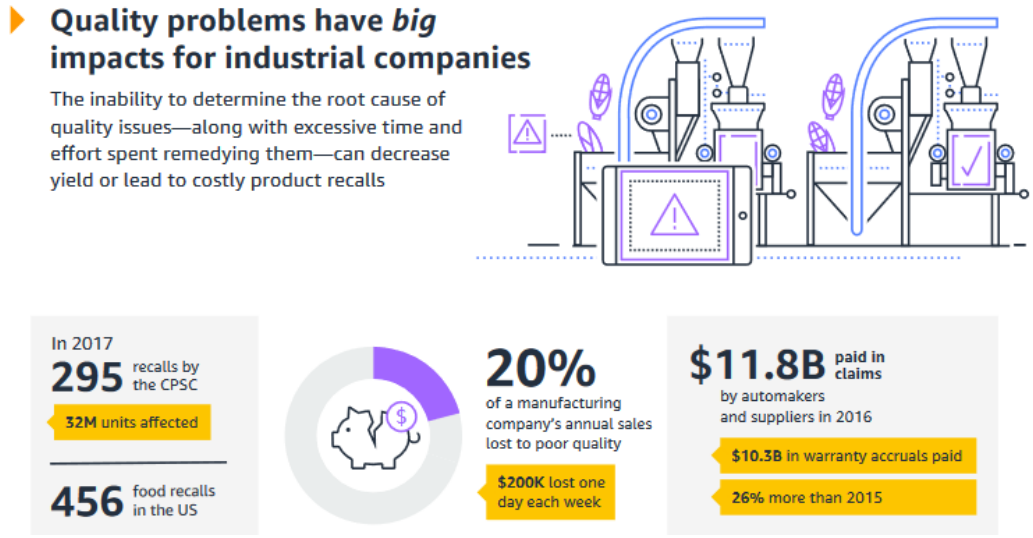


Рисунок 1.1 – Інфографіка прогнозування якості

2. Моніторинг стану ресурсів. Моніторинг стану ресурсів (рис.1.2) дозволяє збирати дані про стан машин та обладнання, щоб визначати ефективність їхньої роботи. Завдяки AWS IoT ви можете збирати всі дані Інтернету речей, наприклад, показники температури та вібрацій, а також коди помилок, що вказують, чи працює обладнання оптимальним чином. Завдяки підвищеній видимості, ви можете гарантувати, що можливості ресурсу використовуються повною мірою.

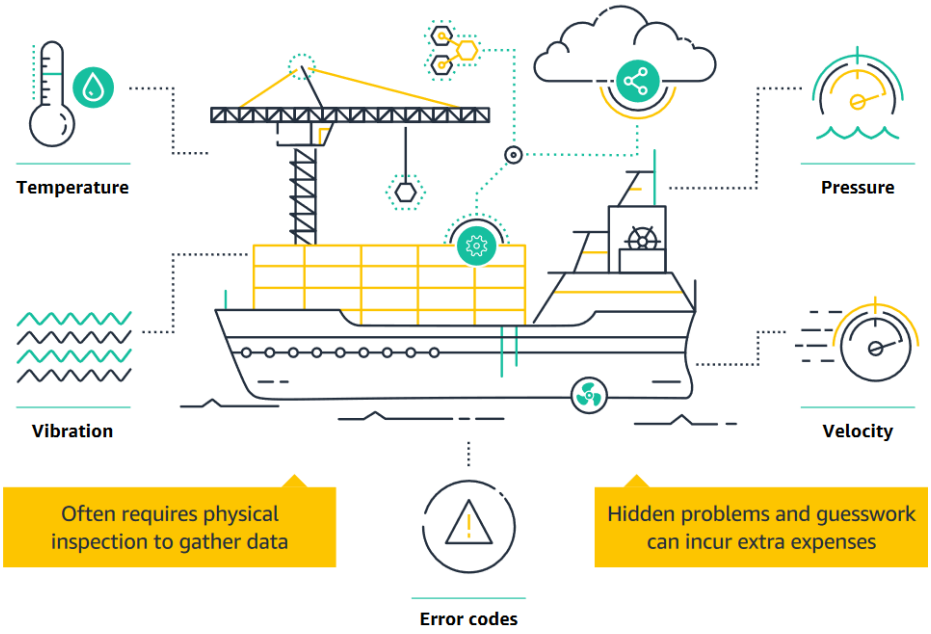


Рисунок 1.2 – Інфографіка стану ресурів

3. Профілактичне обслуговування. Аналітика профілактичного обслуговування (рис. 1.3) дозволяє збирати дані про стан промислового обладнання, щоб виявляти потенційні несправності перед тим, як вони вплинуть на виробництво. Це допомагає продовжити термін служби обладнання, забезпечити більш безпечні умови роботи для працівників та оптимізувати ланцюжок постачання. Завдяки AWS IoT є можливість безперервно відстежувати та аналізувати стан, працездатність та продуктивність обладнання, щоб виявляти проблеми в реальному часі.

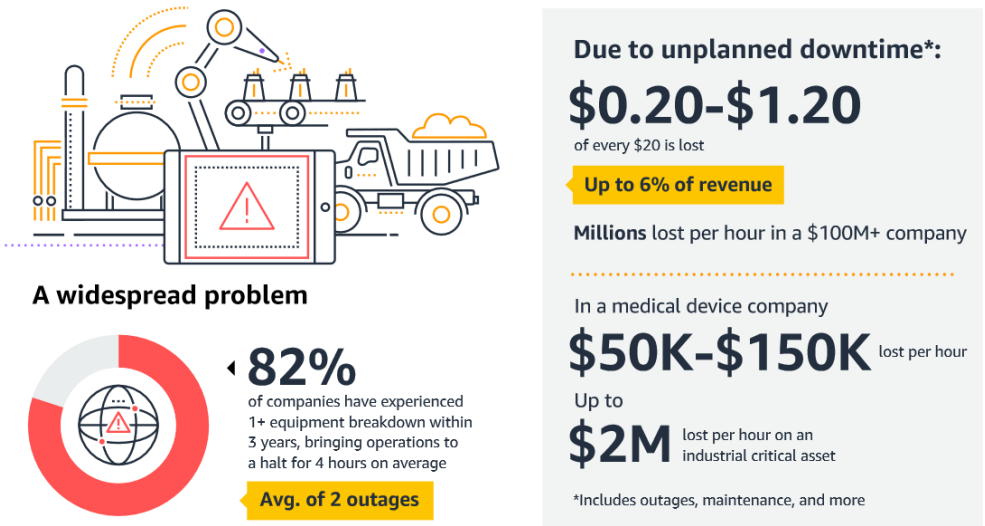


Рисунок 1.3 – Інфографіка профілактики

## 1.2 Функціонування IIoT

Принцип роботи технології полягає в наступному: спочатку встановлюються датчики, виконавчі механізми, контролери та людино-машинні інтерфейси на ключові частини обладнання, після чого здійснюється збір інформації, яка згодом дозволяє компанії придбати об'єктивні та точні дані про стан підприємства. Оброблені дані доставляють у всі відділи підприємства, що допомагає налагодити взаємодію між співробітниками різних підрозділів та приймати обґрунтовані рішення.

Крім цього, компанії можуть замінити паперову документацію, що швидко застаріває, а також акумулювати експертні знання фахівців.

Отримана інформація може бути використана для запобігання позаплановим простоям, поломкам обладнання, скорочення позапланового технічного обслуговування та збоїв в управлінні ланцюжками поставок, тим самим дозволяючи підприємству функціонувати більш ефективно[2].

При обробці величезного масиву неструктурованих даних їх фільтрація та інтерпретація є пріоритетним завданням для підприємств. У даному контексті особливу значущість набуває коректне представлення інформації у зрозумілому користувачеві вигляді, для чого сьогодні на ринку представлені передові аналітичні платформи, призначені для збирання, зберігання та аналізу даних про технологічні процеси та події у реальному часі.

У 2011 році людством було згенеровано 1,8 зеттабайт інформації. У 2012 році обсяг цінних даних збільшився майже вдвічі та склав 2,8 зеттабайт. До 2022 року ця цифра досягне 40 зеттабайт. Такі великі обсяги даних вимагають обробки, щоб бути використаними в процесі прийняття рішень.

Щоб уникнути простоїв і задля збереження безпеки для підприємства необхідно впровадження технологій, дозволяють виявляти і прогнозувати ризики. Безперервний проактивний моніторинг ключових показників дає можливість визначити проблему та вжити необхідних заходів для її вирішення. Для зручності операторів сучасні системи дозволяють візуалізувати умови

протікання технологічних процесів і виявляти фактори, що впливають на них, за допомогою будь-якого веб-браузера. Оперативний аналіз допомагає користувачам швидше знаходити причини неполадок.

Завдяки таким рішенням виробничі дані перетворюються на корисну інформацію, яка необхідна для безпечного та раціонального управління підприємством.

Впровадження таких технологій дає змогу підприємствам із різних галузей економіки отримати певні переваги: збільшити ефективність використання виробничих активів на 10% за рахунок скорочення кількості незапланованих простоїв; знизити витрати на технічне обслуговування на 10%, удосконаливши процедури прогнозування та запобігання катастрофічним відмовам обладнання та виявляючи неефективні операції; підвищити продуктивність на 10%, збільшити рівень енергоефективності та скоротити експлуатаційні витрати на 10% за рахунок більш ефективного використання енергії[3].

Таким чином, нові технології дозволяють підприємствам різних галузей промисловості досягти суттєвих конкурентних переваг.

### 1.3 Переваги промислового інтернету речей для економіки

За кількісним зростанням інтернету речей та організаційно-технологічною трансформацією виробництва стоять важливі якісні зміни в економіці:

- дані, які раніше були не доступні, зі зростанням проникнення вбудованих пристроїв є цінною інформацією про характер використання продукту та обладнання для всіх учасників виробничого циклу, є основним формуванням нових бізнес-моделей і забезпечують додатковий дохід від пропозиції нових послуг, таких як, наприклад: договір життєвого циклу на промислове обладнання, контрактне виробництво як обслуговування, транспорт як обслуговування, безпеку як обслуговування та інші;

- віртуалізація виробничих функцій супроводжується формуванням «економіки спільного використання», що характеризується значно більш високою ефективністю та продуктивністю за рахунок підвищення використання наявних ресурсів, зміни функціоналу пристроїв без внесення змін до фізичних об'єктів шляхом зміни технологій управління ними;
- моделювання технологічних процесів, наскрізне проектування і, як результат, оптимізація ланцюжка створення вартості на всіх етапах життєвого циклу продукту в режимі реального часу, дозволяють виробляти штучний або дрібносерійний продукт за мінімальною ціною для Замовника та з прибутком для виробника, що у традиційному виробництві можливе лише за масовому виробництві;
- еталонна архітектура, стандартизовані мережі і модель оренди замість оплати повної вартості володіння, роблять спільну виробничу інфраструктуру доступною для середнього та малого бізнесу, що полегшує їх зусилля з управління виробництвом, дозволяє прискорити реагування на вимоги ринку, що змінюються, і скорочення життєвого циклу продукції, і тягне за собою розробку та появу нових додатків та сервісів;
- аналіз даних про користувача, його виробничі об'єкти (машини, будівлі, обладнання) та характер споживання відкривають можливості для постачальника послуги з покращення клієнтського досвіду, створення більшої зручності користування, кращого рішення та скорочення витрат клієнта, що веде до підвищення задоволеності та лояльності від роботи з даним постачальником;
- функціонування різних галузей економіки безперервно становиться більш складним під впливом розвитку технологій і все більше здійснюватиметься за рахунок автоматичного прийняття рішень самими машинами на основі аналізу великого обсягу даних із підключених пристроїв, що призведе до поступового зниження ролі виробничого персоналу, у тому числі кваліфікованого[4].

## 1.4 Поняття «Розумна фабрика» (Smart Factory)

Розумне виробництво визначається як «повністю інтегровані виробничі системи підприємства, які можуть реагувати в режимі реального часу на зміну умов виробництва, вимог ланцюга поставок і задовольняти потреби клієнтів». У цьому визначенні головне: «в реальному часі», тобто максимально швидко. Ці цілі досягаються за рахунок інтенсивного та комплексного використання інформаційних технологій та кібер-фізичних систем на всіх етапах виробництва та доставки.

Розумне виробництво разом із промисловим Інтернетом речей є основою Індустрії 4.0.

**Industry 4.0.**

Характерними особливостями Індустрії 4.0 є повністю автоматизоване виробництво, де всі процеси контролюються в режимі реального часу та з урахуванням змін зовнішніх умов. Кіберфізичні системи створюють віртуальні копії об’єктів у фізичному світі, контролюють фізичні процеси та приймають децентралізовані рішення. Вони здатні об'єднуватися в одну мережу, взаємодіяти в режимі реального часу, самонавчатися. Важливу роль відіграють Інтернет-технології, що забезпечують зв'язок між персоналом і машинами. Підприємства випускають продукцію відповідно до вимог окремого замовника, оптимізуючи собівартість продукції. Серед багатьох технологій найпоширенішими є:

**IoT.** У цій технології Інтернет використовується для обміну інформацією як між людьми, а й між всілякими «мовами», тобто машинами, пристроями, датчиками тощо. З одного боку, речі, забезпечені датчиками, можуть обмінюватися даними та обробляти їх без участі людини. З іншого боку, людина може брати активну участь у цьому процесі, наприклад, коли йдеться про «розумний будинок»[5].

**Цифрові екосистеми.** Це системи, що складаються з різних фізичних об'єктів, програмних систем та контролерів, що дозволяють уявити таку освіту

як єдине ціле. Фізичні та обчислювальні ресурси у такій екосистемі тісно пов'язані, моніторинг та управління фізичними процесами використання з використанням технологій IIoT.

**Аналітика великих даних** або просто Великі дані. Величезні обсяги інформації, що накопичуються в результаті «оцифрування» фізичного світу, можуть бути ефективно оброблені лише комп'ютерами (в майбутньому, можливо, квантовими) із застосуванням хмарних обчислень та технологій штучного інтелекту. У результаті людина, яка контролює той чи інший процес, ситуацію, обстановку повинна отримувати оброблені дані, максимально зручні для сприйняття, аналізу та прийняття рішення.

**Складні інформаційні системи**, відкриті для використання клієнтами та партнерами. Це можуть бути цифрові платформи та системи для управління бізнес-процесами, для інтеграції інтернету речей у фізичні бізнес-процеси, для аналізу та прогнозування стану обладнання.

Оскільки поняття «розумне виробництво» дуже розпливчасте, а перехід до нього відбувається в кілька етапів, робляться спроби розділити це поняття на три. Таким чином фабрики майбутнього поділяються на три основні типи — цифрові, «розумні» та віртуальні[9].

**Цифрові.** Основною задачею цифрової фабрики є розробка моделей виробів, що випускаються з використанням інструментів цифрового проектування та моделювання.

Основні системи та технології:

- САПР – система автоматизованого проектування;
- PDM – система керування даними виробу;
- PLM – прикладне програмне забезпечення для управління життєвим циклом продукції;
- верстаки з ЧПУ;
- 3D-принтери та інші адитивні технології.

**Розумні.** "Розумні" заводи націлені на масове виробництво, але при зберіганні максимальної гнучкості виробництва. Це забезпечується високим рівнем автоматизації і робототехніки підприємства. Широко використовуються автоматизовані системи управління технологічними і виробничими процесами. Технології IIoT забезпечують відповідність обладнання. Виробничі активи компанії з датчиками та засобами зв'язку, що працюють за протоколом IPv6, здатні випускати продукцію практично без вміщення людини. Технології великих даних дозволяють справлятися з швидкісними потоками інформації, що працюють від датчиків і автоматизованих систем управління. Основні системи та технології:

- АСУТП - автоматизована система управління технологічними процесами;
- APS – синхронне (удосконалене) планування виробництва;
- MES – система управління виробничими процесами;
- IIoT - промисловий інтернет речей;
- Big Data – великі дані.

**Віртуальні.** Віртуальна фабрика – це мережа цифрових та «розумних» фабрик, до якої також входять постачальники матеріалів, комплектуючих та послуг. На такому заводі використовується ряд автоматизованих систем управління підприємством для управління глобальними ланцюжками постачання та розподіленими виробничими активами. За належного ступеня інтеграції вони дозволяють розробляти і використовувати віртуальну модель всіх організаційних, технологічних, логістичних та інших процесів, що протікають не тільки на підприємстві, а й на рівні розподілених виробничих активів та глобальних ланцюжків постачання, аж до післяпродажного обслуговування. Основні системи та технології:

- ERP – планування ресурсів підприємства;
- CRM – система управління взаємовідносинами з клієнтами;
- SCM – управління ланцюжками постачання.

Компанія «Контрон-Україна» є офіційним представником холдингу Kontron AG, вже більше 12 років постачає вбудоване комп'ютерне обладнання для подальшого застосування в сфері телекомунікаційного призначення, промислової автоматизації, систем військового призначення, авіації, енергетики, систем наземного транспорту, контрольно-вимірювального та медичного обладнання на українському ринку.

## 1.5 Industry 4.0 в Україні

Компанія Kontron є одним із основних ініціаторів впровадження рішень Індустрії 4.0 у сегменті промислової автоматизації та Smart Factory . Крім того, компанія створює як власні комплексні рішення, так і виробляє обладнання для провідних світових OEM-виробників, які мають вбудовувані комп’ютерні технології, відкриту архітектуру для задоволення потреб пристроїв IoT і Big Data.

Таким чином, випуск широкого асортименту системних та легко кастомізованих продуктів є частиною філософії компанії та дає можливість гарантувати клієнтам максимально швидкий вихід рішення на ринок.

Управління виробничим підприємством, заснованого на постійному прагненні усунення всіх видів втрат та максимальної орієнтації на споживача.

Це призвело до створення абсолютно нового цеху, який включає інфраструктуру та робочі місця і тепер дозволяє компанії нарощувати або скорочувати виробництво протягом кількох годин. Вона також може реагувати на дуже специфічні виробничі вимоги за короткий термін і з безпрецедентною гнучкістю.

Необхідно впровадити виробничу IT-інфраструктуру, яка підтримує:

- цілком безпаперовий «розумний» виробничий процес;
- виробництво кількох продуктів та конфігурацій;
- різноманітні виробничі лінії з можливістю швидкого запуску;
- підвищена продуктивність та вимоги до точності;

− оптимізацію виробничих витрат Підвищена швидкість виходу ринку.

Для досягнення цілей Smart Factory основною вимогою було успішне впровадження повністю безпаперового виробничого процесу для забезпечення оптимальної продуктивності, ефективності та якості[6].

За наявності великого об’єму кількостей у специфікації матеріалів паперовий виробничий процес є вкрай неефективним і може значно збільшити ймовірність людської помилки за рахунок використання старої або неправильної документації. Тому безпаперовий онлайн-підхід був необхідний, особливо з урахуванням великого портфоліо продуктів та різних варіантів конфігурації. Це призвело до початкового розгортання стандартних ПК на робочих станціях для забезпечення доступу до внутрішньої мережі з усіма інструкціями зі збирання продукту, а також до центральної ERP-системи (система планування ресурсів підприємства) та системи управління виробництвом. Однак незабаром стало очевидно, що ПК займають занадто багато цінного місця і не мають обчислювальної потужності, довговічності, сполучності або масштабованості, необхідних для задоволення зростаючих і майбутніх потреб розумного виробництва Kontron.

Kontron досить швидко знайшов ідеальне рішення – промислову комп'ютерну платформу KBox-A101 (рис. 1.4).

У пристрої використаний двоядерний процесор Intel Atom CPU D2550D, завдяки чому він має високу продуктивність і відмінно підходить для промислових обчислювальних завдань. Крім того, компактний KBox A-101 приваблює своєю конструкцією, яка не потребує догляду. Він оснащений вбудованим твердотілим накопичувачем (SSD) ємністю до 64 гігабайт для надійного зберігання та швидкого доступу до даних.

Крім того, безвентиляторні KBox-A101 готові до використання в IoT та підтримують широкий спектр промислових інтерфейсів, включаючи DisplayPort, два Gbit Ethernet, два USB 2.0 та два порти USB 3.0, а також додаткову шину CAN та Profibus інтерфейси. Для застарілого обладнання також

може використовуватися послідовний інтерфейс RS232, а для бездротового підключення до хмари або локальної мережі системи K-Box можуть бути оснащені системами LTE (4G) та GSM (2G) або Wi-Fi з двома зовнішніми антеними роз'ємами, що забезпечують високу якість сигналу.

Для легкого впровадження будь-якого робочого місця доступні різноманітні варіанти монтажу, включаючи настільний, горизонтальний настінний, книжковий формат. Також за складних умов зовнішнього середовища до речі виявиться розширений температурний діапазон (від -10 °C до 60 °C) та ударомiцність до 15G.


Рисунок 1.4 – Kbox-A101

**Переваги цього рішення**. Безліч блоків KBox було встановлено на кожному мобільному робочому столі та тестовій стійці, щоб оператори могли легко отримати онлайн-доступ до всіх відповідних та новітніх версій документів, що охоплюють весь виробничий процес, включаючи інструкції зі збирання та випробувань.

Крім того, система KBox діє як блок управління обладнанням для тестування складальної лінії заводу, здійснюючи онлайн-моніторинг виробництва та збір даних для управління контролем якості, робочим процесом та статусом виконання замовлень. Система акумулює дані та надає персоналу заводу актуальну інформацію в режимі реального часу про точний статус та вимоги для кожного продукту, що проходить складання.

Прийняття компанією Kontron рішення про відмову від паперового виробничого процесу дало можливість виконувати різні вимоги клієнтів одночасно, найкраще підтримуючи баланс точності та швидкості. За допомогою промислової платформи KBox Kontron може швидше реагувати на вимоги клієнтів, дозволяючи підвищити продуктивність, скоротити витрати виробництва та оптимізувати час виконання замовлень, зберігаючи при цьому високий рівень якості, яким компанія відома у всьому світі[7].

Програма безперервного вдосконалення призвела до заміни комп'ютерів стандартного розміру, розміщених у зоні виробничого цеху на рішення Kontron KBox. Це дозволило значно розширити простір на робочих станціях, а також підвищити загальну доступність та надійність системи.

Можливості Kbox:

- Kbox успішно підтримує безпаперове діловодство;
- висока продуктивність обробки;
- комплексним та різноманітним можливостям підключення;
- всі необхідні порти вводу-виводу;
- малому форм-фактору;
- безвентиляторному блоку, без рухомих частин та необхідності догляду;
- легкість установки;
- легкість управління пристроями у існуючих IT ERP-системах.

### 1.6 Фінансові перспективи та етапи впровадження

Потенціал зростання світового ринку «фабрик майбутнього» величезний. Обсяг ринку цифрових фабрик досягне, за різними оцінками, 260 млрд. доларів до 2020 року та 740 млрд. доларів до 2035 року. Обсяг ринку «розумних фабрик» – відповідно 490 млрд. доларів та 1,35 трлн. доларів. За віртуальними фабриками експерти очікують зростання у 690 млрд. доларів до 2020 року та майже 1,5 трлн. доларів за 20 років. Можливо, вже реалізуються проекти побудови нових підприємств, максимально наближених до реалізації концепції

«розумної» фабрики і навіть віртуальне виробництво , однак переведення підприємств, що вже працюють, на нові принципи планування, виробництва, поставок та післяпродажного обслуговування продукції буде здійснюватися поступово і з максимальним використанням вже наявних виробничих активів. Послідовність переходу істотно залежить від специфіки роботи підприємства та доступності нових технологій.

У компанії виділяють такі етапи, які потрібно пройти для того, щоб реалізувати концепцію «розумних» фабрик та закласти основи для подальшого переходу до віртуальних фабрик.

Цифровізація виробництва. Забезпечення персоналу мобільними платформами, встановлення на обладнання датчиків та промислових контролерів. Установка нового обладнання, яке вже спочатку обладнане цифровими інтерфейсами. Ідентифікація фізичних об'єктів підприємства

Забезпечення мережевої взаємодії. Завдання збору даних із датчиків у реальному масштабі часу можна вирішити за рахунок підключення всіх пристроїв та датчиків до платформи IIoT. Оперативний обмін інформацією між співробітниками.

Побудова цифрового двійника підприємства. Вирішення задачі візуалізації реального стану справ на підприємстві. Вироблення чітких правил, якими можна виявити відхилення від норми, що відбулися і під час виробничих і бізнес-процесів. ERP-система дозволяє дуже детально та оперативно візуалізувати та відслідковувати стан виробництва по всьому холдингу, на підприємстві, показники роботи підрозділів та конкретного обладнання.

Забезпечення за допомогою мобільних платформ, синхронізації даних автоматизованої системи планування та даних, отриманих від обладнання, оперативне коригування планів. Забезпечення достовірності та корисності оперативної інформації.

Перехід до завдань планування у реальному масштабі часу з урахуванням достовірної інформації про перебіг виробничих процесів[8].

Забезпечення автоматичної реакції системи управління більшість виробничих ситуацій. Тобто це рішення, яке вироблено індивідуально для конкретного обладнання, яке індивідуально налаштовується, і завдяки цьому система зможе запускати автоматичні реакції на виробничі події з виробництва.

## 1.7 Аналіз діючих систем IIoT

**Bystronic System.**

Bystronic є провідною світовою компанією, що спеціалізується на розробці ефективних технологій обробки листового металу. Практичні рішення цієї компанії забезпечують ефективне та екологічне виробництво. Основний акцент робиться на створенні єдиної виробничої лінії з різання та згинання, а також на автоматизованій обробці матеріалів та даних. Ефективне впровадження мережевих технологій, яке реалізується на установках лазерного різання та листозгинальних пресів із застосуванням інноваційних рішень у галузі автоматизації, програмного забезпечення та сервісного обслуговування, є ключем до глобальної цифровізації усієї галузі[10].

Компанія розробляє наступне автоматизоване обладнання: автоматизоване лазерне різання металу, а також автоматизоване сгибання металу для створення потрібної форми.

**Автоматизація Лазер.** Автоматизація виробничого процесу дозволяє оптимізувати проведення матеріалу, покращити завантаження обладнання та підвищити експлуатаційну та технологічну безпеку. Всі компоненти є окремими модулями, що дозволяє надалі регулювати ступінь автоматизації. Ступінь автоматизації може бути різним – від простого ручного до повністю автоматичного управління.

Пристрій Byloader автоматично подає металеві листи на змінний стіл установки лазерної різки (рис. 1.5). Номінальний розмір листів, що обробляються 3000 х 1500 мм. Товщина металевих листів, завантаження 0,8-25 мм. Максимальна вага металевих листів 890 кг.

Рисунок 1.5 – Byloader

Переваги:

- швидка та надійна автоматична загрузка металевих листів на змінний стіл;
- оптимальна загрузка установки для лазерної різки;
- легке керування за допомогою системи керування установкою.

Пристрій ByTrans Extended (рис. 1.6) який приймає на себе функцію завантаження та розвантаження споруд лазерної різки.

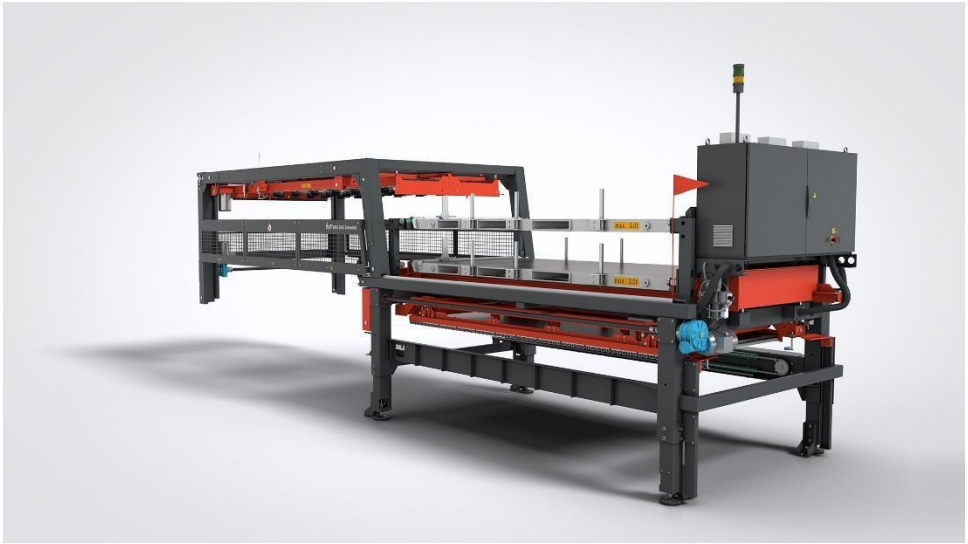

Рисунок 1.6 – ByTrans Extended

Ефективне рішення для завантаження та розвантаження установок для лазерного різання. Переваги:

- швидка обробка замовлень за рахунок того, що автоматичне завантаження та розвантаження скорочує час на переналагодження;
- значне збільшення продуктивності установки за мінімальних вкладень;
- варіант ByTrans Extended оснащений не однією, а двома касетами, що робить всю систему ще більш автономною;
- універсальність використання. Підходить не тільки для зберігання/складування, але й для відбору крупноформатних деталей, а також для підготовки пластикових захисних плит, які укладають між листами металу (ByTrans Extended);
- можливість обробки з мінімальним втручанням оператора;
- ByTrans Extended пропонується у форматах 3×1,5 м та 4×2 м.

Пристрій ByTrans Cross (рис. 1.7) дозволяє повністю автоматизувати проведення матеріалу в ході процесу лазерної різки.

Рисунок 1.7 – ByTrans Cross

**Автоматизація згинання.** Рішення в галузі автоматизації процесу згинання прискорюють процедури обробки, підвищують якість виробництва і дозволяють повною мірою використовувати потенціал інтегрованих систем

згинання. Залежно від потреби рішення з автоматизації від Bystronic, спрямовані на підтримку процесів згинання, можуть реалізовуватися шляхом впровадження повністю автоматизованого обладнання через продумане поводження з матеріалом, а також за допомогою автоматизованої заміни інструментів. За запитом повністю автоматизовані рішення компанія Bystronic оптимально інтегрує у перед- або післявключені процеси.

Переваги:

- автоматизація, що відповідає будь-яким запитам. Дооснащення модульної системи для завантаження та розвантаження ByTrans Cross можливе в будь-який момент;
- висока гнучкість та доступність. Короткі цикли завантаження та розвантаження забезпечують високу ефективність та підвищують комфорт роботи для оператора;
- повна доступність установки для лазерного різання, тому що ByTrans Cross забезпечує автоматизоване переміщення матеріалу навіть за умов дуже обмеженого простору;
- все на одному сенсорному екрані: вбудована система керування автоматизацією за допомогою пульта керування установки для лазерного різання;
- захист матеріалу та правильне сортування. Автоматичний відбір габаритних деталей та облік обрізків.

Пристрій ByCell Bend Smart (рис. 1.8), який дозволяє автоматизувати згинання металу будь-якої потрібної форми.

Рисунок 1.8 – ByCell Bend Smart

Переваги:

- недорогий згинальний модуль для освоєння технології автоматизованого згинання;
- стабільне та надійне рішення для автоматизації згинання з метою максимального збільшення вашої продуктивності;
- оптимізована компоновка всіх згинальних модулів;
- безліч конфігурацій та захоплення, виготовлені за специфікацією замовника, на роботизованому згинальному модулі Smart;
- збільшена робоча зона робота за рахунок системи заміни, завдяки чому підвищується ефективність роботи усієї виробничої лінії;
- система захватів, адаптована до реальних умов застосування;
- спеціально розроблене програмне забезпечення для управління роботою роботизованих згинальних модулів;
- ефективне та просте в освоєнні програмування завдяки параметрам та командам, що задаються для роботизованого модуля.

**Eton(Opta) System.**

Виробництво одягу на ринку споживачів, що швидко розвивається, вимагає коротких термінів пропускної здатності та низької вартості якості для підтримки прибутковості.

Eton Systems дозволяє виробникам одягу перейти від традиційної та неефективної пакетної системи до системи виробництва одиниць. Завдяки системі Opta усі швейні виробництва можуть скористатися перевагами UPS.

Найважливішим фактором для підтримки високої ефективності у виробництві одягу є те, щоб виробництво не зупинялося. За допомогою Opta всі компоненти автоматично послідовно розподіляються на кожну робочу станцію, яка надається безпосередньо оператору.

Програмне забезпечення містить усі ключові функції, які можуть знадобитися виробнику одягу. Мета полягає в тому, щоб була можливість повністю контролювати весь ланцюжок створення вартості. Тому прозорість операцій є ключовою. Opta допомогає забезпечити безперебійну роботу усіх операцій.

Промисловість домашнього текстилю загалом пройшла довгий шлях з точки зору автоматизації, наприклад, у виробництві ковдри. Машини для наповнення волокна часто автоматизовані та інтегровані з іншими машинами. Eton Systems має гнучкість, необхідну для повної інтеграції системи обробки матеріалів як із наявними машинами, так і з наявним програмним забезпеченням.

Автоматизація замість ручного транспортування ковдр, простирадла та інших виробів домашнього текстилю може творити чудеса для виробничого процесу. Головне – усунути ручне переміщення матеріалів з одного робочого місця на інше. Після завершення цих завдань оператори можуть витрачати більше часу на додавання цінності до кінцевого продукту[11].

Останнє, але не менш важливе. Автоматизоване транспортування матеріалів замість повторюваного ручного підйому заощадить персонал і

зменшить кількість травм на виробництві. Конвеєрні системи Eton Systems безпечно піднімають окремі шматки вагою до 12 кг.

Система Eton допомогає з внутрішньою логістикою між обладнанням протягом усього виробництва. Від розрізання до пакування. Автоматизоване транспортування продукції перенесе ваші виробничі лінії від напівавтоматичних до автоматичних, включаючи швейні станції.

Підвісні конвеєри (рис. 1.9) з індивідуально адресованими носіями продукту автоматично спрямовують свій шлях через запрограмовані послідовності операцій.

Рисунок 1.9 – Підвісні конвеєри

Концепція Eton’s Flexible Productivity заснована на двох системних платформах з широким набором функцій, включаючи:

- кілька рейок для сортування на робочих станціях;
- буферизація;
- автоматичні завантажувальні пристрої;
- мости та ліфти, які з’єднують системи та поверхи;
- програмне забезпечення на основі модулів.

**IoTJi by deps.**

У зв'язку з великою площею приміщень та необхідністю дотримання техніки безпеки при організації робочого процесу необхідно підтримувати освітлення з високим ступенем освітленості та яскравості на робочому місці, що нерідко досягає збільшення освітленості. лампи з великим навантаженням та споживаною потужністю. У зв'язку з цим постає питання зниження собівартості продукції - реалізації енергозберігаючих заходів на підприємствах.

Крім того, виробничі та характерні для виробничої сфери нюанси освітлення виробничих приміщень. Наприклад, доступ до щита управління електрообладнанням повинен мати обмежену кількість людей, яких приваблює необхідна постійна фізична присутність працівників, які відповідають за включення та вимкнення світла. Це, у свою чергу, означає додаткові адміністративні витрати та неефективну організацію робочих процесів.

Системою на основі бездротової передачі даних LoRaWAN, до якої підключаються димові світильники, що забезпечує моніторинг параметрів освітлення та дистанційну функцію. На території підприємства встановлена базова станція LoRa, яка отримує інформацію від ламп і передає команду на підключення до найкращого часу доби, рівня природного освітлення з вікон або від бригади датчиків руху. Інформація про роботу ламп і щитів від радіомодулів LoRa в датчиках вбудована в єдиний пункт управління виробничих приміщень. В результаті кількість поїздів закривається, система автоматично включає промислове освітлення або переводить його в режим максимальної енергоефективності, зменшуючи споживання електроенергії на 50%[12].

**Як побудована мережа LoRaWAN.**

LoRaWAN використовує неліцензовану частину радіочастотного спектру в діапазоні 868,0 – 868,6 МГц та енергозберігаючі технології. Стандарт має базові станції та абонентські пристрої, які за умови автономного живлення, більшу частину часу перебувають у режимі збереження енергії. Датчики

LoRaWAN «прокидаються» лише для обміну даними із сервером. Це дозволяє виробникам гарантувати експлуатацію IoT-пристрою протягом довгого часу без необхідності заміни батарей. Схема типової мережі LoRaWAN наведена у (рис. 1.10).

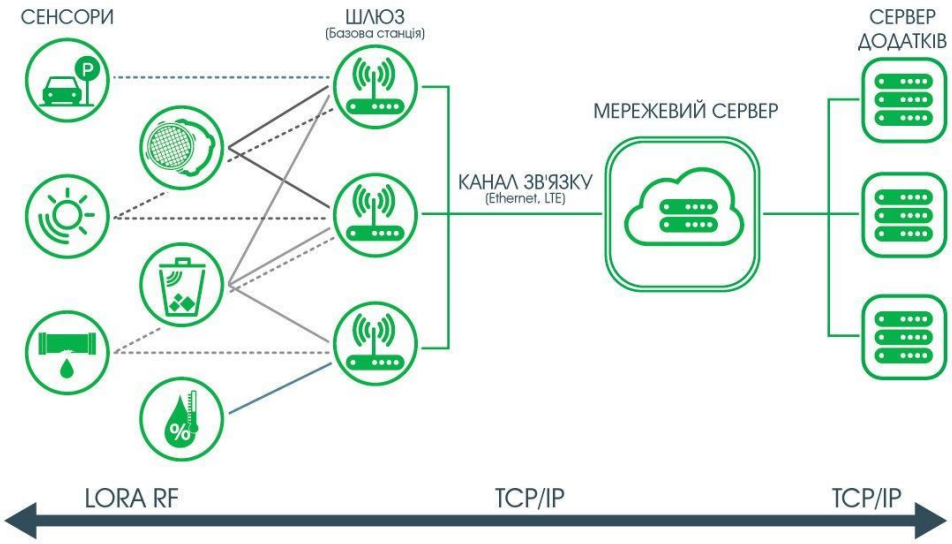


Рисунок 1.10 – Типова мережа LoRaWAN

У мережі LoRaWAN радіомодулі направляють на концентратор. Від базової станції за швидкісним каналом пакети даних передаються на свою чергу, дає конкретний тип даних відповідно до серверних додатків.

Крім того, технологія LoRaWAN дозволяє розширити систему та встановити моніторинг енергоспоживання. Таким чином досягається 100% контроль справності всього освітлювального обладнання, а також доступності до нього живлення. Датчики через IoT-модеми можуть сигналізувати про стан відкриття дверей, підвищуючи систему безпеки приміщення. А гнучка система керування живленням може автоматизувати подачу живлення для будь-якого типу обладнання[13].

Індустріальні LoRaWAN температурні датчики (рис. 1.11) дозволяють контролювати ступінь нагріву обмотки електродвигунів та температуру підшипників. Якщо говорити про те, які параметри вимірює датчик вібрації, то це як віброшвидкість, так і віброприскорення по одній осі або відразу за трьома.

Розумні датчики тиску здійснюють виміри в агресивному промисловому середовищі, забезпечуючи надійний моніторинг технологічних процесів на фабриці.

Перевагами систем LoRaWAN вважаються зручністю установки датчиків а також великим масштабом покриття:

1. Одна базова станція покриває всю територію цеху надійним зв’язком.
2. Майже всі показники роботи обладнання можна знімати через датчики.
3. Зручність монтажу датчиків забезпечується магнітною підставкою для фіксації на промисловому об’єднанні фабрики.
4. Енергоефективний принцип роботи дозволяє працювати датчикам кількох років без замінних елементів живлення. В той час, що більше часу пристрій є у стані сну, а прокидається лише для передачі зібраних даних на кілька секунд.


Рисунок 1.11 – Датчик вібрації і температури Netvox R718E

Основною частиною будь-якого проекту Інтернету Речей є його софтверна складова. Для проекту з управління вібраціями – це LoRaWAN Network Server, що забезпечує зв’язок між собою датчиків, актуаторів та

базових станцій. LNS також надає інформацію про LoRaWAN Application Server, який відповідає бізнес-логіці.

Використання технологій IIoT дозволяє перейти від прогнозованих характеристик до постійного контролю, значно збільшуючи горизонтування відмовлення обладнання. Підключення штучного інтелекту до обробки масиву даних робить всю ще ефективну, надає можливість відслідковувати довгострокові тренди.

Маючи оптимальні та критичні значення температури та частоти вібрації промислових машин, Iotji спроектували бездротову систему контролю вібрації обладнання. Усі зібрані з контрольних точок верстатів через протокол LoRaWAN виробляються та візуалізуються на платформі IoT Thingsboard PE, а також інтегруються в SCADA, що давно використовується на фабриках. Інтеграція IoT LoRaWAN в SCADA особливо підходить під час переналаштування виробничих процесів або перенесення обладнання.

Економічний принцип роботи датчиків дозволяє експлуатувати без технічного обслуговування збору протягом тривалого часу, оптимізуючи витрати виробництва.

Після встановлення мережі фабрика LoRaWAN вже розширюється функціональні датчики. необхідно планується підключення датчиків струму для ведення технічного обліку всіх енергоресурсів з деталізації по кожній одиниці обладнання. Так фабрика може точно планувати свої витрати та формувати собівартість продукції. Мережа LoRaWAN дозволяє підключити й інші рішення (рис. 1.12, рис. 1.13), необхідних у процесі роботи промислового підприємства: датчики руху, контролю відчинення/зачинення вікон і дверей стануть незамінними для відстеження доступу до об’єктів. А димова сирена 868 МГц стане частиною системи пожежної безпеки.


Рисунок 1.12 – Герконовий датчик Netvox R718F


Рисунок 1.13 – Netvox RA02A датчик диму на батарейках

**Висновок до розділу 1**

Промисловий Інтернет речей об'єднує машини, хмарні обчислення, аналітику та співробітників, щоб підвищити ефективність промислових процесів. Завдяки IIoT промислові компанії можуть оцифровувати процеси, модифікувати бізнес-моделі, а також підвищувати продуктивність та ефективність, водночас скорочуючи витрати.

Поняття «розумна фабрика» визначається так: це повністю інтегровані корпоративні виробничі системи, які здатні в реальному масштабі часу реагувати на умови виробництва, що змінюються, вимоги мереж поставок і задовольняти потреби клієнтів. Досягаються названі цілі за допомогою інтенсивного і всеосяжного використання інформаційних технологій усім етапах виробництва та її поставки.

Після проведення аналізу ринку систем типу «Smart Factory» було виділено чотири найпопулярніших брендів та їх представників. Це Bystronic System, Eton (Opta) Systems, Iotji by deps. Кожна з них профілюється на своїх окремих задачах.

Була розглянута кожна з систем, їх можливості та засоби використання у виробничих сферах.

Окремо хочеться виділити мережу LoRaWAN, який дозволяє автоматизувати процеси віддалено. Створити мережевий сервер підприємства, зав’язати на нього обладнання, датчики та інші засоби по керуванню, та вести керування віддалено за допомогою мобільного пристрою, або комп’ютера.

На основі проведеного аналізу існуючих систем було прийняте рішення розробити унікальну систему дистанційного керування підприємством використовуючи мобільний телефон або планшет за допомогою внутрішнього Wi-Fi середовища, на якому будуть зав’язані всі процеси та функції підприємства. На сьогоднішній день аналогів такій програмі на телефон не існує.

# 2 ТЕХНОЛОГІЇ СТВОРЕННЯ ДИСТАНІЙНОГО АВТОМАТИЗОВАНОГО КЕРУВАННЯ

У сучасному світі існує велика кількість технологій, які дозволяють створити автоматизоване дистанційне керування підприємством. Існують мікро комп’ютери, які підключаються до певного вузлу фабрики та дозволяє отримати керування над ним через мережу, також це відноситься до всіляких датчиків, перемикачів, то що.

## 2.1 Промислові комп’ютери для автоматизації процесів

Вбудовувані комп’ютери – це готові високовиробничі рішення, які можна використовувати під конкретні задачі або розширити їх під окрему індивідуальну задачу. Вони підходять для використання в обмеженому просторі для сприйняття свого невеликого форм-фактора і пасивної системи охолодження.

Існує велика кількість промислових комп’ютерів для вирішення будь-якої потреби. Найпоширенішою компанією по розробці промислових комп’ютерів є Advantech UNO. Розроблені за всіма вимогами «Індустрії 4.0» та представляють з себе компактні вбудовані промислові ПК, які можуть виконувати периферійні обчислення, скорочуючи розрив між IT та OT. Розглянемо декілька систем цієї компанії, виявимо їх здібності та переваги[14].

Промисловий комп’ютер Intel Celeron J1900 (рис. 2.1).


Рисунок 2.1 – Intel Celeron J1900

Модульний дизайн та проста адаптація під конкретне виробниче завдання дозволяє швидко побудувати необхідне рішення з мінімальними витратами.

− Економічні 4-х ядерні процесори Intel Celeron J1900 із попередньо встановленою оперативною пам'яттю 4G DDR3L.

− Міцна конструкція без використання внутрішньої проводки для високої надійності.

− Компактний корпус із пасивним охолодженням та широким вибором варіантів встановлення – на монтажну панель, DIN-рейку або на VESA кріплення.

− Модульний дизайн із використанням додаткового стека для встановлення 2-х модулів iDoor.

− Великий вибір опціональних бездротових модулів зв'язку – 3G/GPS/GPRS/Wi-Fi.

− Розширення комунікацій з периферійними пристроями за допомогою польових інтерфейсів.

На процесорній платі відразу встановлено 4Гб оперативної пам'яті DDR3L, яку можна розширити до 8Гб.

Міцний корпус із пасивним охолодженням, відсутність внутрішньої проводки дозволяють встановлювати комп'ютери UNO у виробничих умовах.

Комп'ютери можуть працювати всередині гарячих цехів або зовні, цілий рік, без додаткового обігріву або охолодження.

Якщо потрібна підтримка безпроводових або стільникових мереж, з'єднання з ПЛК або цифровими датчиками, в комп'ютері встановлюються модулі розширення miniPCIe. Спеціальний стіковий модуль розширення дозволяє встановити до комп'ютера до 2-х інтерфейсних модулів iDoor. Промислові комп'ютери сумісні з Windows7, Windows 10 та Linux.

Одне з традиційних та найпоширеніших застосувань компактних комп'ютерів – це створення шлюзів промислового Інтернету Речей (IIoT gateway). Шлюзи промислового інтернету речей потрібні для отримання, обробки та подальшої передачі на верхній рівень інформації від датчиків та виконавчих пристроїв. Сучасні датчики здатні генерувати великий обсяг інформації, яку можна обробляти та використовувати безпосередньо на об'єкті, знижуючи вимоги до продуктивності каналів зв'язку та обсягу накопичувачів для їх віддаленого зберігання.

Промисловий безвентиляторний панельний ПК W24IF7T-GCA2 (рис. 2.2).

Рисунок 2.2 – Промисловий комп’ютер W24IF7T-GCA2

Ключовими особливостями цього комп’ютера полягають у наступних параметрах:

- 23,8 дюйми, 1920 x 1080, сенсорний екран P-CAP;
- процесор Intel® Core™ i3/i5/i7 9-го покоління (Coffee Lake Refresh);
- передня панель IP65 для захисту від води та пилу;
- плоска передня поверхня, що легко очищається, з дизайном від краю до краю;
- гнучкі пристрої зберігання, один 2,5-дюймовий жорсткий диск та одне гніздо для SSD.

Фактично плоска лицьова поверхня, що легко очищається, з дизайном від краю до краю. Плоскі алюмінієві панельні ПК та дисплей Winmate P-Cap виходять за рамки стандартних промислових панельних комп'ютерів завдяки елегантному дизайну від краю до краю. Легко обслуговувати та чистити, щоб вода або краплі рідини не проникали у кожен отвір у пристрої.

Передня панель IP65 Захист від води та пилу за допомогою роз'ємів M12 Багатоцільовий водонепроникний повністю герметичний панельний ПК зі ступенем захисту IP65 від Winmate захищений від пилу та водяних бризок та ідеально підходить для використання в суворих умовах.

Безвентиляторна система охолодження та наднизьке енергоспоживання. Лінія G-WIN пропонує безвентиляторний дизайн платформи з низьким енергоспоживанням, панелі, що читаються при сонячному світлі, інтеграцію з WLAN, захист від ударів та вібрації, а також антикорозійне покриття корпусів із алюмінієвого сплаву.

## 2.2 IoT у розробці мобільних додатків

Сьогодні майже кожна людина, включаючи промислових робітників, має смартфон, що забезпечує цілодобовий доступ до підключеної інфраструктури.

Програми з підтримкою Інтернет речей увійшли майже до всіх сфер виробництва і кардинально змінили спосіб функціонування підприємств. Звичайно, мобільний додаток – це лише частина інфраструктури Інтернет речей. Однак у поєднанні з новими технічними досягненнями та технологіями на даний момент це може підвищити ваші продажі та рівень продуктивності, а також повністю змінити спосіб ведення бізнесу.

- Доступність: мобільність телефону гарантує, що користувач має цілодобовий доступ до управління системою IoT.
- Більше даних для аналізу: системи Інтернет речей збирають та обробляють дані з датчиків, підключених пристроїв та мобільних телефонів. Коли смартфон стає ще одним елементом підключеної інфраструктури, він надає масу даних про звички, уподобання та спосіб життя своїх користувачів, що відкриває двері для більш персоналізованих рекламних акцій, домашніх налаштувань та інших варіантів використання. Ці елементи надалі використовуються службами розробки мобільних програм для персоналізації програм.

Проте, прагнучи зробити мобільні програми зручними для кінцевих користувачів, компанії-розробники додатків Інтернет речей більше не зосереджуються виключно на наданні зручних інтерфейсів. Сьогодні мобільні програми, що входять до мережі IoT, створюються з урахуванням кращої інтеграції з іншими підключеними пристроями. Вони мають, з одного боку, взаємодіяти з іншими частинами системи; з іншого боку, очікується, що вони усвідомлюють переваги додатків, а саме портативність, інтуїтивно зрозуміле використання та безшовну інтеграцію у повсякденну рутинну діяльність.

За допомогою мобільного телефону будь-який підключений продукт може бути доступний з будь-якого місця будь-коли.

Нафтогазова промисловість, виробничі підприємства, металургійні заводи — будь-яка галузь може покращити свою політику профілактичного обслуговування, встановивши ефективне корпоративне рішення IoT та надавши своїм співробітникам додаток для моніторингу в реальному часі поверх нього. Такі програми є інформаційною панеллю, що дозволяє користувачам переглядати дані про продуктивність обладнання у вигляді діаграм, діаграм і графіків, а також отримувати попередження, якщо щось піде не так.

Мобільний застосунок може бути додано до маси датчиків, об'єднаних в екосистему IoT, і може бути як модемом, так і периферійним пристроєм.

Мобільні програми як частина інфраструктури Інтернет речей можуть покращити багато операцій, у тому числі:

- керування персональними даними у будь-якому проекті IoT;
- платежі та перекази у фінансових операціях;
- підписання контрактів за допомогою смарт-контрактів;
- моніторинг здоров'я в Smart Hospital та підключених системах охорони здоров'я.

Системи, що включають транзакції, пов'язані з грошима, документообігом або контрактами, виграють лише в тому випадку, якщо пов'язані з ними мобільні програми будуть такими ж безпечними та стабільними, як функціонування блокчейна.

Вдалий приклад технології блокчейн, інтегрованої в IoT, зокрема розробки мобільних додатків — Chronicled. Програма, підтримувана технологією блокчейн, використовується для безпечного відстеження документів та змін до них, таких як нові версії, виправлення та оновлення статусу.

## 2.3 Загальні відомості NET. Framework

Платформа .NET Framework — це технологія, яка підтримує створення та виконання веб-служб і додатків Windows. При розробці платформи .NET Framework враховувалися наступні цілі.

Забезпечення узгодженої об'єктно-орієнтованої середи програмування для локального збереження та виконання об'єктного коду, для локального виконання коду, розподіленого в Інтернеті, або для віддаленого виконання.

Надання середовища виконання коду, в якому:

- зведено до мінімуму ймовірність конфліктів у процесі розгортання програмного забезпечення та управління його версіями;
- гарантується безпечне виконання коду, включаючи код, створений невідомим або повністю довіреним стороннім виробником;
- виключаються проблеми з продуктивністю серед виконання скриптів або інтерпретованого коду;
- забезпечуються єдині принципи розробки для різних типів програм, таких як програми Windows та веб-програми;
- забезпечується взаємодія на основі промислових стандартів, що гарантує інтеграцію коду платформи .NET Framework із будь-яким іншим кодом.

Платформа .NET Framework складається із загальномовного середовища виконання та бібліотеки класів .NET Framework. Основою платформи .NET Framework є середовище CLR. Середовище виконання можна вважати агентом, який керує кодом під час виконання та надає основні служби, такі як управління пам'яттю, управління потоками та віддалену взаємодію. При цьому середовищем накладаються умови суворої типізації та інші види перевірки точності коду, що забезпечують безпеку та надійність. Фактично основним завданням середовища виконання є керування кодом. Код, який звертається до середовища виконання, називають керованим кодом, а код, який не звертається до середовища виконання, називають некерованим кодом. Бібліотека класів є комплексною об'єктно-орієнтованою колекцією повторно використовуваних

типів, які застосовуються для розробки додатків - починаючи зі звичайних додатків, що запускаються з командного рядка, та додатків з графічним інтерфейсом (GUI) і закінчуючи програмами, що використовують останні технологічні можливості ASP.NET, такі як веб-форми та веб-служби XML.

Платформа .NET Framework може розміщуватися некерованими компонентами, які завантажують середовище CLR у власні процеси та запускають виконання керованого коду, створюючи таким чином програмне середовище, що дозволяє використовувати засоби як керованого, так і некерованого виконання. Платформа .NET Framework не лише надає кілька базових середовищ виконання, але й підтримує розробку базових середовищ виконання незалежними виробниками.

Наприклад, ASP.NET розміщує середовище виконання та забезпечує масштабоване середовище для керованого коду на стороні сервера. ASP.NET працює безпосередньо з середовищем виконання, щоб забезпечити виконання програм ASP.NET та веб-служб XML, що обговорюються нижче в цій статті.

Оглядач Internet Explorer може служити прикладом некерованої програми, яка розміщує середовище виконання. Розміщення середовища виконання в браузері Internet Explorer дозволяє впроваджувати керовані компоненти або елементи керування Windows Forms HTML-документи. Таке розміщення середовища дозволяє виконувати керований мобільний код та користуватися його суттєвими перевагами, зокрема виконанням в умовах неповної довіри та ізольованим зберіганням файлів.

На наступному рисунку (рис 2.3) демонструється взаємозв'язок середовища CLR і бібліотеки класів з додатками користувача і всією системою. На рисунку також показано, як керований код працює у межах ширшої архітектури.

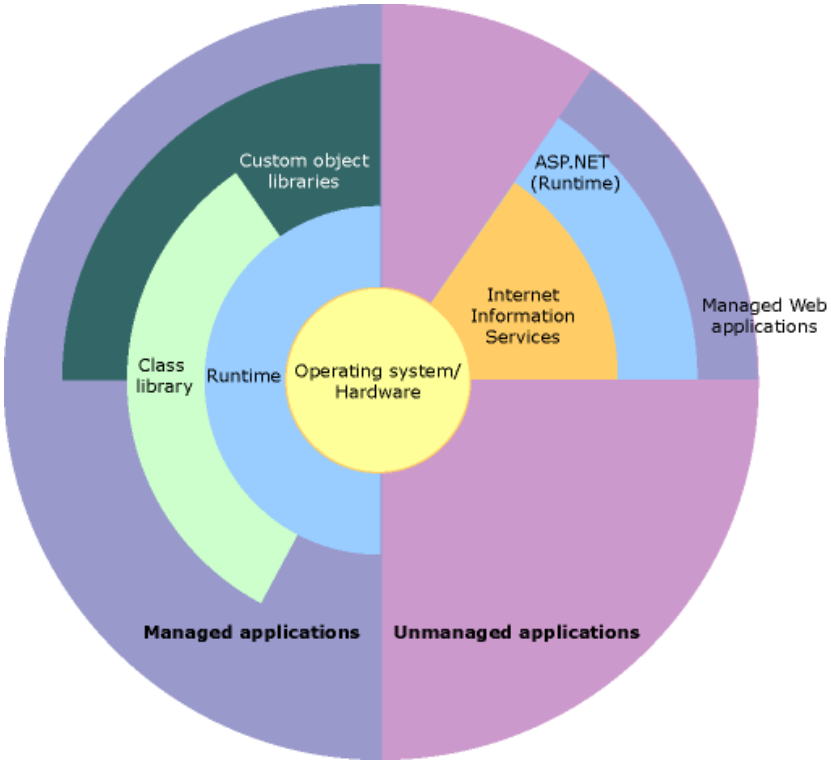


Рисунок 2.3 – Взаємозв’язок середовища CLR

**Можливості середовища CLR.** Середовище CLR управляє пам'яттю, виконанням потоків, виконанням коду, перевіркою безпеки коду, компіляцією та іншими системними службами. Ці засоби є внутрішніми для керованого коду, який виконується серед CLR.

З міркувань безпеки керованим компонентам присвоюються різні ступені довіри, що залежать від низки факторів, до яких входить їхнє походження (наприклад, Інтернет, мережа підприємства або локальний комп'ютер). Це означає, що керований компонент може або не може виконувати операції доступу до файлів, операції доступу до реєстру або інші важливі функції, навіть якщо він використовується в тому самому активному додатку.

Середовище виконання також забезпечує надійність коду, реалізуючи інфраструктуру суворої типізації та перевірки коду, яку називають системою загальних типів. Система загальних типів забезпечує самоопис всього керованого коду. Різні мовні компілятори корпорації Microsoft і незалежних виробників створюють керований код, який відповідає системі загальних типів.

Це означає, що керований код може приймати інші керовані типи та екземпляри, забезпечуючи правильність типів і строгу типізацію.

Крім того, кероване середовище виконання виключає багато проблем, що часто виникають з програмним забезпеченням. Наприклад, середовище виконання автоматично керує розміщенням об'єктів та посиланнями на об'єкти, звільняючи їх, коли вони більше не використовуються. Автоматичне керування пам'яттю виключає дві найпоширеніші помилки додатків: витоку пам'яті та недійсні посилання на пам'ять.

Середовище виконання також підвищує продуктивність розробників. Наприклад, програмісти можуть писати програми звичною мовою розробки, при цьому використовуючи всі переваги середовища виконання, бібліотеки класів і компонентів, написаних іншими розробниками іншими мовами. Це доступно будь-якому виробнику компіляторів, які звертаються до середовища виконання. Мовні компілятори, призначені для платформи .NET Framework, роблять засоби .NET Framework доступними для існуючого коду, написаного відповідними мовами, істотно полегшуючи процес перенесення існуючих програм.

Хоча середовище виконання розроблялося для майбутнього програмного забезпечення, воно також підтримує сьогоднішнє та вчорашнє програмне забезпечення. Взаємодія керованого та некерованого кодів дозволяє розробникам використовувати необхідні компоненти COM та бібліотеки DLL.

Середовище виконання розроблено підвищення продуктивності. Хоча загальномовне середовище виконання надає багато стандартних служб часу виконання, керований код ніколи не інтерпретується. Засіб компіляції на вимогу (JIT) дозволяє виконувати весь керований код машинною мовою комп'ютера, де він запускається. Тим часом диспетчер пам'яті усуває можливість фрагментації пам'яті і збільшує обсяг пам'яті, що адресується для додаткового підвищення продуктивності.

Нарешті, середовище виконання може розміщуватися у високопродуктивних серверних програмах, таких як Microsoft SQL Server та служби IIS. Така інфраструктура дозволяє використовувати керований код для написання своєї логіки програм, користуючись при цьому високою продуктивністю найкращих виробничих серверів, які підтримують розміщення середовища виконання.

**Бібліотека класів .NET Framework.** Бібліотека класів платформи .NET Framework є колекцією типів, які тісно інтегруються з середовищем CLR. Бібліотека класів є об'єктно-орієнтованою. Вона надає типи, від яких керований код користувача може успадковувати функції. Це не тільки полегшує роботу з типами .NET Framework, але й скорочує час вивчення нових засобів платформи .NET Framework. Крім того, компоненти незалежних виробників можна легко поєднувати з класами платформи .NET Framework.

Наприклад, у класах колекцій .NET Framework реалізується набір інтерфейсів для розробки власних класів колекцій. Користувацькі класи колекцій легко поєднуються з класами .NET Framework.

Як і очікується від об'єктно-орієнтованої бібліотеки класів, типи .NET Framework дозволяють вирішувати типові завдання програмування, включаючи роботу з рядками, збирання даних, підключення до баз даних та доступ до файлів. На додаток до цих звичайних завдань бібліотека класів містить типи, що підтримують багато спеціалізованих сценаріїв розробки. Ви можете використовувати платформу .NET Framework для розробки наступних типів програм та служб:

- консольні програми;
- програми с графічним інтерфейсом;
- програми Windows Presentation Foundation;
- програми ASP.NET.

Класи Windows Forms є повний набір типів, які спрощують розробку графічних інтерфейсів користувача Windows. Під час написання веб-форм ASP.NET можна використовувати класи веб-форм[15].

## 2.4 Xamarin Native Projects

Xamarin - це платформа з відкритим вихідним кодом, призначена для побудови сучасних продуктивних програм для iOS, Android та Windows з .NET. Платформа Xamarin є рівень абстракції, який забезпечує управління взаємодією між загальним кодом та кодом базової платформи. Xamarin виконується в керованому середовищі, яке реалізує такі можливості, як виділення пам'яті та складання сміття.

Завдяки Xamarin в середньому 90% коду програми можна використовувати без змін на різних платформах. За допомогою цього шаблону розробник може написати всю бізнес-логіку однією мовою (або використовувати існуючий код програми), але при цьому отримати характеристики продуктивності, оформлення та поведінку, характерні для кожної відповідної платформи.

Програми Xamarin можна писати на ПК або Mac і компілювати у власні пакети програм, наприклад файли з розширенням .apk для Android або .ipa для iOS.

Платформа Xamarin орієнтована на розробників, перед якими стоять такі завдання:

- спільне використання коду, тестів та бізнес-логіки на різних платформах;
- написання кросплатформових додатків мовою C# у Visual Studio.

**Принцип роботи платформи Xamarin.**

На схемі (рис 2.4) показана загальна архітектура кроссплатформенного додатка Xamarin. За допомогою Xamarin ви можете створювати власний

інтерфейс для кожної платформи і писати мовою C# загальну бізнес-логіку, яка буде використовуватися на різних платформах. У більшості випадків Xamarin дозволяє використовувати на різних платформах 80% коду програми.

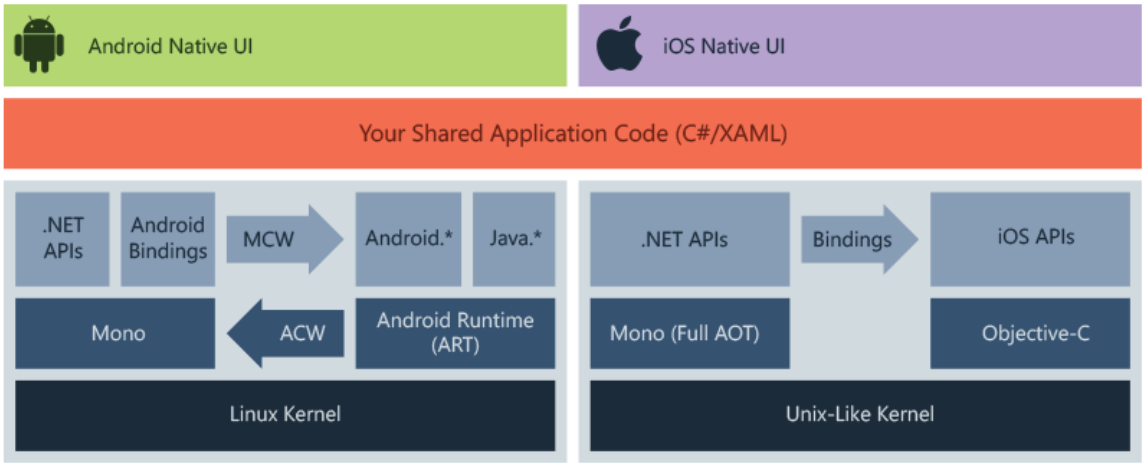


Рисунок 2.4 – Загальна архітектура Xamarin

Xamarin поєднує можливості власних платформ з додаванням можливостей, до яких відносяться:

1. Повна прив'язка до базових пакетів SDK. Xamarin містить прив'язки практично для всіх базових пакетів SDK в iOS та Android. Крім того, ці прив'язки є строго типізованими, що означає, що вони зручні у навігації та використанні, а також дозволяють здійснювати якісну перевірку типів під час компіляції та розробки. Строго типізовані прив'язки, що дозволяють скоротити кількість помилок часу виконання та підвищити якість додатків.
2. Взаємодія Objective-C, Java, C та C++. Xamarin дозволяє безпосередньо викликати бібліотеки Objective-C, Java, C і C++ для ефективнішого використання різноманітного стороннього коду. Ця можливість дозволяє використовувати існуючі бібліотеки iOS та Android, написані на Objective-C, Java або C/C++. Крім того, Xamarin пропонує проекти прив'язки для власних бібліотек Objective-C і Java за допомогою декларативного синтаксису.
3. Сучасні конструкції мови. Програми Xamarin написані сучасною мовою C#, яка характеризується значними поліпшеннями порівняно з Objective-C та Java. Сюди входять динамічні функції мови, функціональні конструкції,

наприклад, лямбда-вираження, LINQ, функції паралельного програмування, універсальні шаблони тощо.

4. Надійна бібліотека базових класів (BCL). Програми Xamarin використовують бібліотеку BCL. забезпечуючи доступ до тисяч бібліотек, які містять додаткові функції, що виходять за рамки BCL.

5. Сучасне інтегроване середовище розробки (IDE). Xamarin використовує сучасне середовище Visual Studio, в якому реалізовані такі можливості, як автозавершення коду, досконаліша система управління проектами та рішеннями, вичерпна бібліотека шаблонів проектів, інтегрована система управління версіями та багато іншого.

6. Підтримка кросплатформових мобільних додатків. Xamarin пропонує вдосконалену кросплатформну підтримку для трьох основних платформ - iOS, Android та Windows. Обсяг загального коду у створених програмах може досягати 90%, а бібліотека Xamarin.Essentials пропонує універсальний API-інтерфейс для доступу до загальних ресурсів на всіх трьох платформах. Це дозволяє значно скоротити витрати на розробку та час випуску продуктів на ринок для розробників, що створюють мобільні програми.

Програми Xamarin.Android компілюються з мови C# в проміжну мову (IL), який при запуску програми зазнає Just-in-Time-компіляції (JIT) в машинне складання. Програми Xamarin.Android працюють у середовищі виконання Mono паралельно з віртуальною машиною середовища виконання Android (ART). Xamarin надає прив'язки .NET до просторів імен Android.* та Java.*. Середовище виконання Mono звертається до цих просторів імен з використанням керованих оболонок (MCW) і надає середовищу виконання ART викликані програми-оболонки Android (ACW), завдяки чому обидва середовища можуть викликати код один одного[16].

## 2.5 Функції та можливості EWeLink

Для розробки застосунку по віддаленому керуванню фабрикою була використана програма EWeLink, яка має наступні функції:

1. Функція дистанційного керування Можна керувати включенням/вимкненням пристрою зі смартфона з будь-якої точки світу
2. Функція роздачі права керування пристроями іншим. Головний обліковий запис (перший обліковий запис для додавання пристрою) може поділитися пристроями з іншими. Після цього інші теж можуть керувати пристроями.
3. Функція режимів для таймера та роботи без мережі. Пристрій працює online або за розкладом, у тому числі таймер одноразовий/багаторазовий та таймер зворотного відліку. Якщо Ваш пристрій відпадає від мережі, вказані таймери будуть працювати як завжди, тільки пристрій потрібно підключити до живлення.
4. Сценарій. Є можливість створити звичайні сценарії чи розумні сценарії. Натиснувши піктограму увімкнення/вимкнення, можна увімкнути/вимкнути всі додані пристрої в режимі звичайного сценарію.
5. Безпека. Кожен пристрій може бути доданий в один обліковий запис (головний обліковий запис), після додавання, інші не можна додати його. Але головний обліковий запис може поділитися пристроєм з іншими[17].

На рисунку 2.5 наведена демонстрація інтерфейсу програми, до якої підключається розумна розетка.

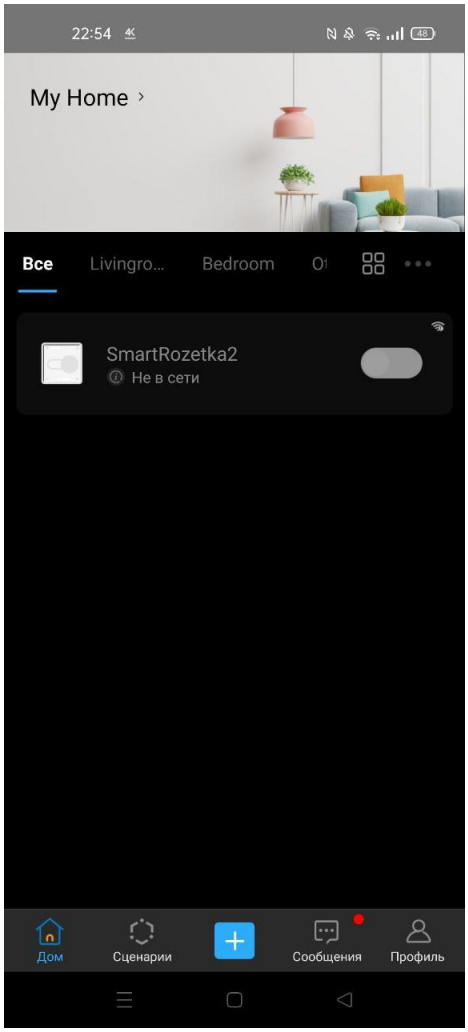


Рисунок 2.5 – EweLink Application

Ця програма виконує функцію Wi-Fi середовища, до якого і буде звертатися мобільний застосунок Factory IoT.

## 2.6 Паттерн MVVM

MVVM - це шаблон, який з'явився для обходу обмежень патернів MVC і MVP, і який об'єднує деякі з сильних сторін. Ця модель вперше з'явилася у складі фреймворку Small Talk, і була пізніше покращена з урахуванням оновленої моделі презентацій (MVP)[18].

Шаблон MVVM має три основні компоненти:

- модель, яка представляє бізнес-логіку програми;
- представлення інтерфейсу користувача XAML;

– уявлення-модель, в якому міститься вся логіка побудови графічного інтерфейсу і посилання на модель, тому він виступає як модель для представлення.

У наступному рисунку (рис 2.6) наведена діаграма, яка показує, як реалізувати шаблон MVVM.

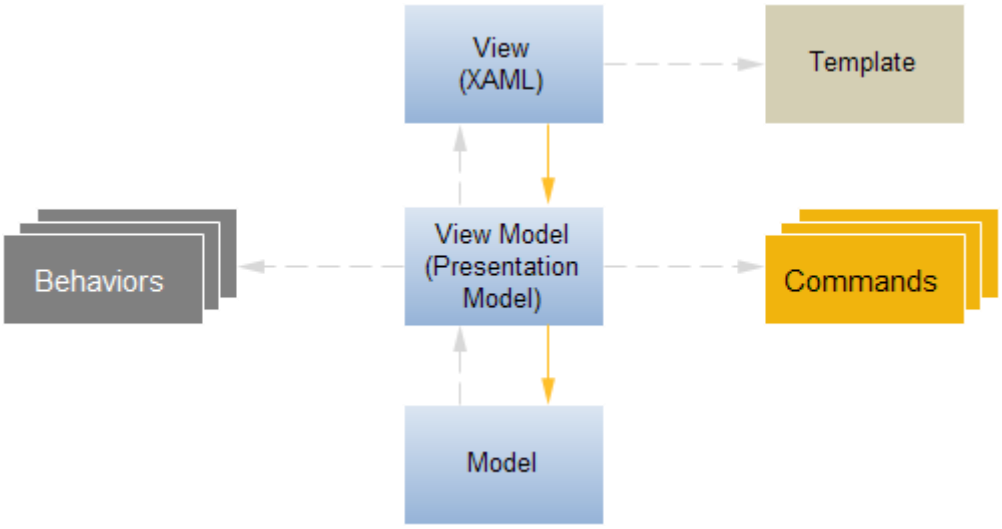


Рисунок 2.6 – MVVM реалізація

**Висновок до розділу 2**

У цьому розділу були розглянуті та описані технології створення дистанційного автоматизованого керування. Розглянуті вбудовані промислові комп’ютери, такі як Intel Celeron J1900, а також безвентиляторний панельний ПК W24IF7T-GCA2, які можуть виконувати периферійні обчислення, скорочуючи розрив між IT та OT.

Також були розглянуті технології для створення мобільних програм, які дозволяють вести керування дистанційно, такі як:

– NET Framework;
– Xamarin native projects;
– EWeLink;
– патерн MVVM.

Наведені їх властивості та переваги.

# 3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МОБІЛЬНОГО ДОДАТКУ ВІДДАЛЕНОГО КЕРУВАННЯ ФАБРИКОЮ

На сьогоднішній день людина не може уявити собі життя без смартфонів або планшетів. Ці гаджети є у кожної людини, а їх потужність вже можна порівнювати з деякими стаціонарними комп’ютерами.

Тому було прийнято рішення розробити саме мобільний застосунок, який буде працювати на будь-якому сучасному смартфоні або планшеті, що буде значно спрощувати керування виробництвом.

Для розробки мобільного додатку була використана сучасна версія середовища Visual Studio Community 2022, з застосуванням найновішого Android SKD версії Android 11 (API level 30).

Код програми буде наведено у Додатку А.

## 3.1 Створення діаграми Use Case

Перед тим як розпочати розробку мобільного додатку, потрібно мати уяву, як буде виглядати фабрика, а саме як, що и до чого відноситься у системі по віддаленому керуванню. Для цього була розроблена Use Case діаграма, яка демонструє взаємозв’язок між всіма компонентами програми (рис. 3.1).

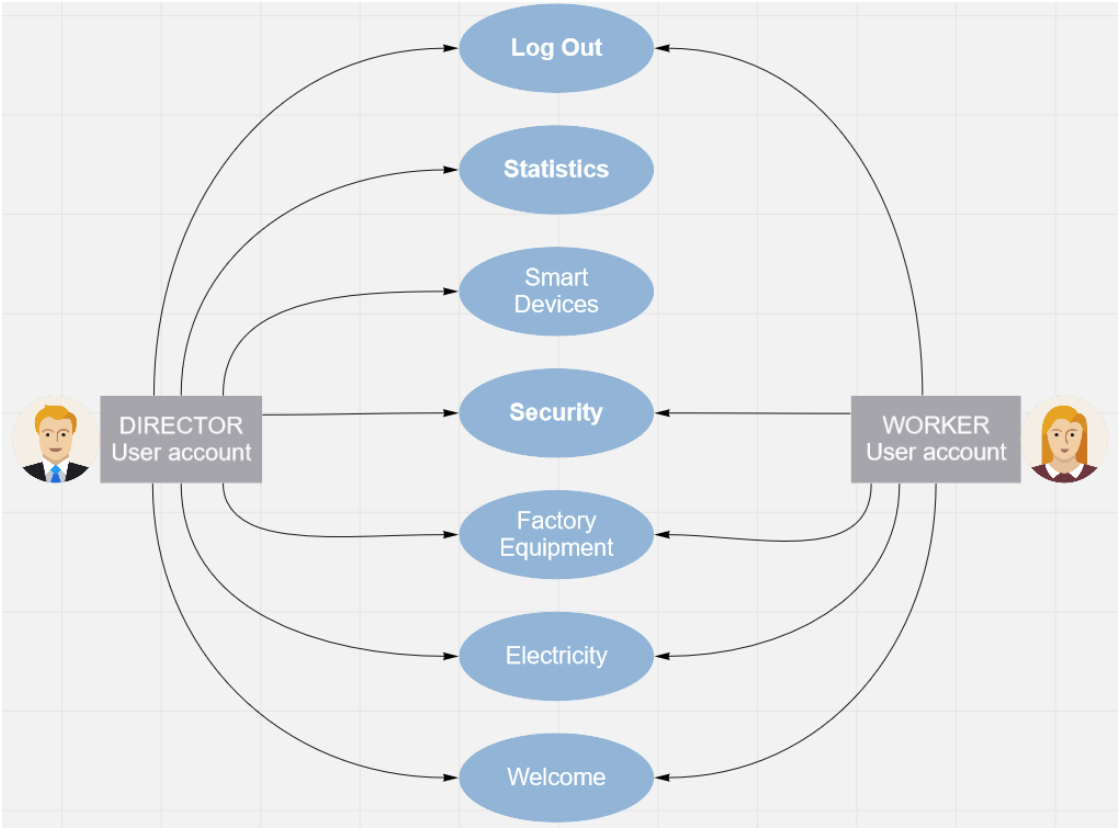


Рисунок 3.1 – Use Case діаграма

Розглянемо цю діаграму більш детальніше. Існує два типи облікових записів, а саме: Director та Worker. Вони відрізняються один від одного кількістю повноважень. Director має повний доступ до всіх функцій керування, а також може дивитися статистику по роботі фабрики, це потрібно йому для проведення розрахунків по прибутку, а також це допомагає підвищити продуктивність виробництва. Worker, тобто звичайний працівник немає доступу до статистики, а також до «розумних» девайсів, але він має доступ до усіх інших систем на одному рівні с директором фабрики.

Сам функціонал поділяється на наступні пункти:

– Welcome – це базова сторінка, у якій описаний функціонал мобільного додатку, для того щоб новий користувач мав можливість швидко розібратися у роботі програми.

– Electricity (Електрика) – основна панель по керуванню струмом, температурою та вологістю приміщення. Має можливість гнучкого налаштування за допомогою зручного функціоналу.

− Factory Equipment (Обладнання фабрики) – панель по керуванню обладнанням, станки та конвеєри.

− Security (Безпека) – панель з екстреною зупинкою роботи фабрики, при виникненні надзвичайної ситуації, наприклад пожежі, затоплення, землетрусу то що. Функція дозволяє у максимально короткий час зупинити та вимкнути всі прилади та обладнання.

− Smart Devices – експериментальна панель у якій проводиться тестування нових технологій, які ще не пройшли остаточної обкатки. На даний момент проводиться тестування розумної розетки.

− Statistics (Статистика) – панель керування, на якій можна відслідковувати результати роботи фабрики, кількість поломок, зупинок і тому подібне.

## 3.2 Створення системи підключення мобільної програми до віддаленого управління фабрикою

Для більш зручного розуміння яким чином звичайний смартфон зможе керувати виробництвом була створена блок-схема (рис. 3.2), на якій демонструється підключення всіх пристроїв до єдиної Wi-Fi мережі, назвемо її «Smart Factory IoT».

Рисунок 3.2 – Блок схема підключення пристроїв до Wi-Fi мережі

На вигляд здається заплутаною, тому буде зроблено пояснення щодо цієї схеми.

Є фабрика, вона містить в собі певне обладнання, наприклад Wi-Fi датчик температури та вологості фірми AVATTO, або промислові комп’ютери, які підключаються до станків, конвеєрів, верстаків тощо. Все це можно зав’язати на одній Wi-Fi мережі, до якої і має можливість підключитися мобільний застосунок Smart Factory IoT. Всі дані щодо роботи пристроїв лежать на EWeLink сервері, до якої мобільний застосунок звертається, коли проводяться будь-які дії по керуванню, будь то зміна температури, зупинка або запуск конвеєру і т.д.

Для того, щоб наглядно продемонструвати принцип цієї системи була використана «розумна» розетка (рис. 3.3) від фірми Sonoff.


Рисунок 3.3 – WiFi Smart Socket S20

## 3.3 Діаграма класів мобільного додатку Smart Factory IoT

Мобільний застосунок побудований з використанням шаблону проектування MVVM через інтеграцію бібліотеки Mvvm.Cross. Тому має в наявності такі класи як **App**, **Setup**, які є основою цієї інтеграції.

Для розуміння структури мобільного додатку була зроблена діаграма класів (рис. 3.4 та рис. 3.5) за допомогою програми Visual Paradigm[19].

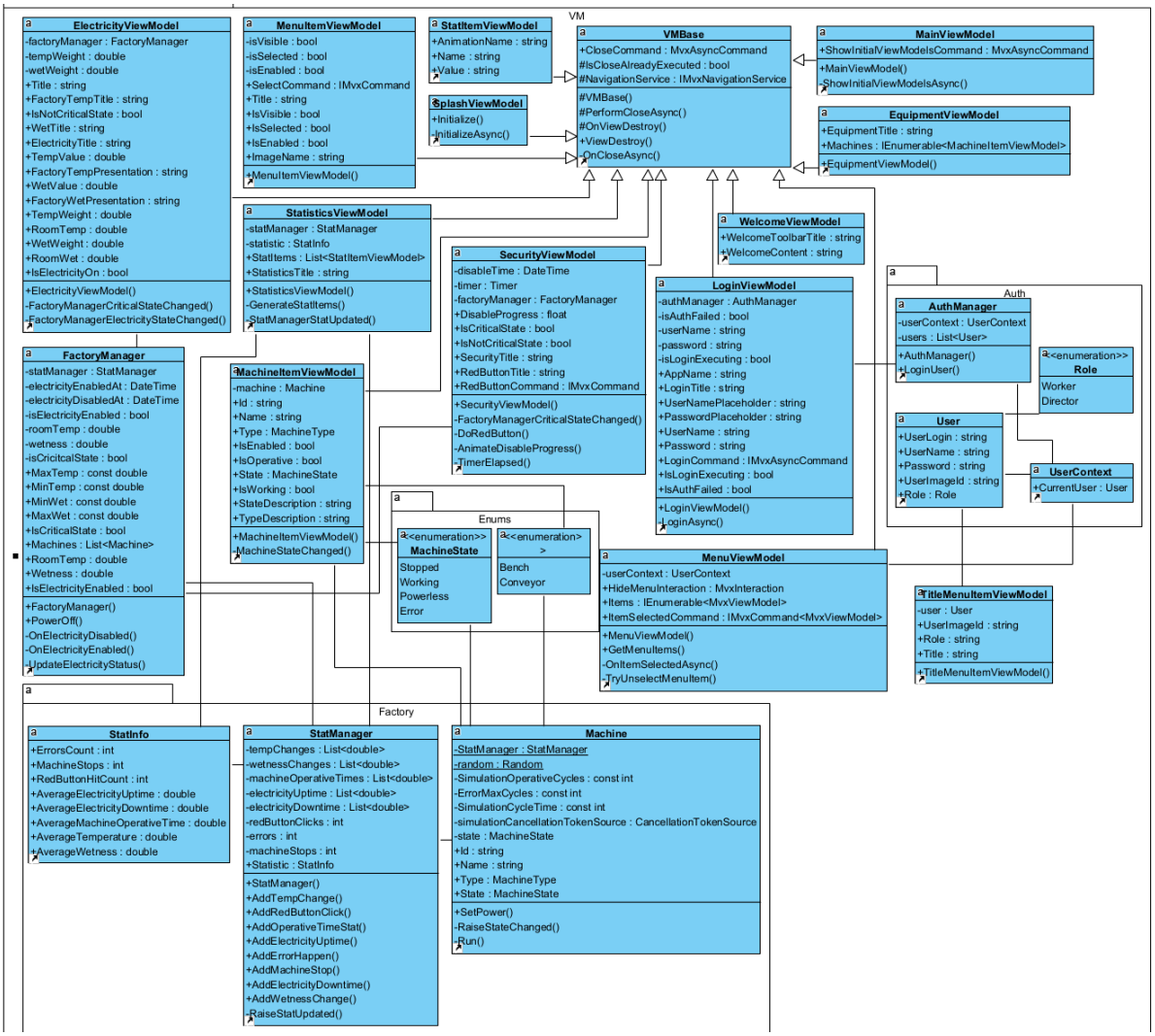


Рисунок 3.4 – Діаграма класів

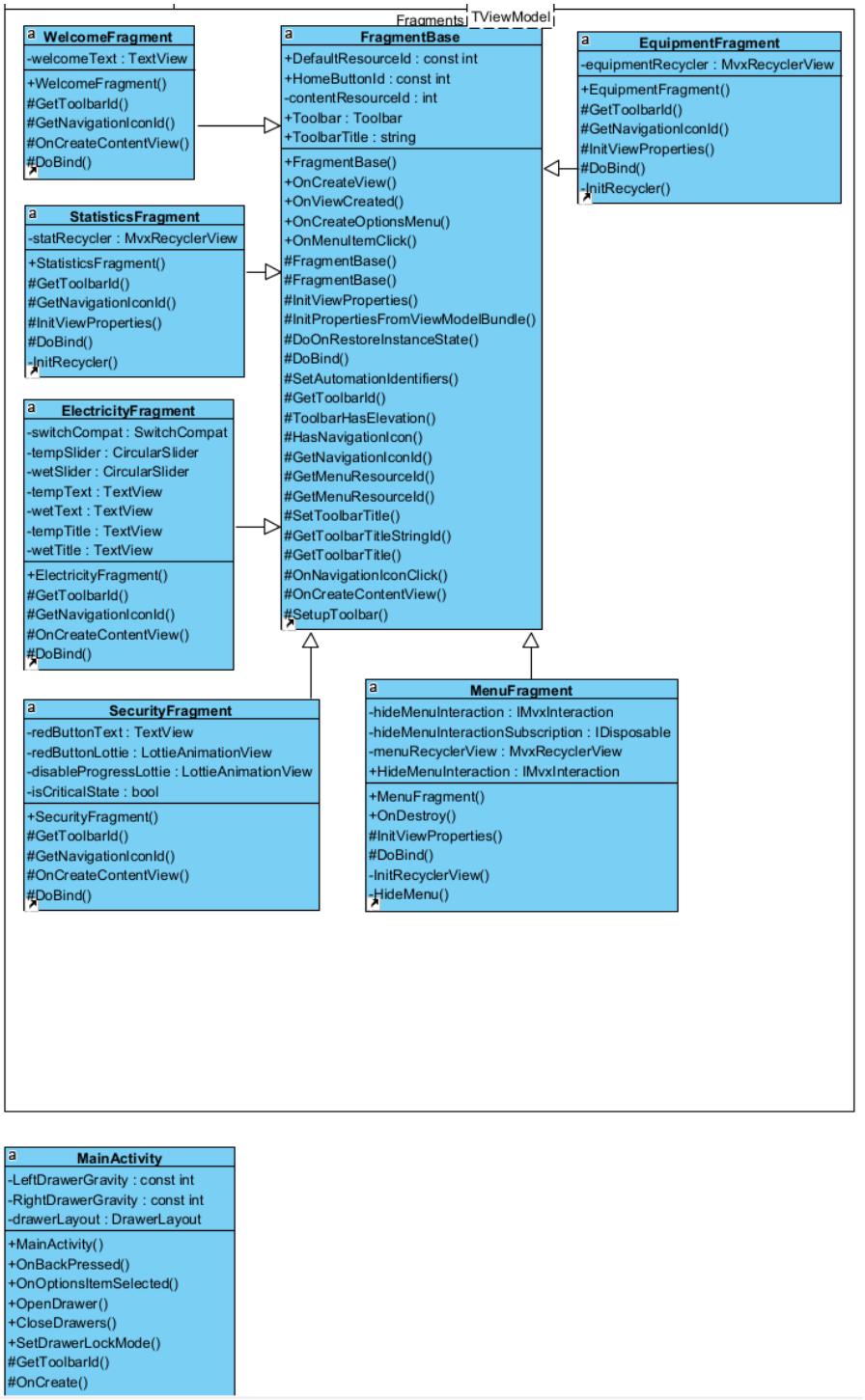


Рисунок 3.5 – Діаграма класів

Клас **App** - представляє собою кросплатформову модель(сутність) даного додатку є своєрідною вхідною точкою для запуску.

Класс **MainApplication** представляє собою вхідну точку саме для додатку Android і пов’язує клас **App** і **Setup** через шаблонний базовий клас **MvxAndroidApplication**, що є частиною бібліотеки **MvvmCross.**

Далі розглянемо шаблон Mvvm(Model-View-ViewModel), що представлено класами даного додатку.

В основу **Model** класів, що є "бізнес-логікою", розроблено класи як **FactoryManager**, **Machine**, **StatInfo**, **StatManager**.

Також для авторизації використовується класс **AuthManager** який маніпулює экземплярами класу **User**, що є моделлю, що описує користувача. Логін та пароль, що використовуються в даному класі зберігаються в класс **UserContext** який надалі використовується в додатку для швидкого доступу до даних користувача. Авторизація є локальною у сеансі роботи з додатком, тому не підключається до реальних сервісів авторизації і не зберігається на пристрої.

Клас **User** використаний як модель користувача додатком. Характерною ознакою будь-якого користувача є роль, (директор або робітник) для доступу до різного функціоналу додатку. В даному випадку в класі **User** реалізовано подібне обмеження за допомогою перелічення **Role**, як ознака певних прав на використання функціоналу. Роль Директор - має доступ до усіх функції додатку, в той час як Робітник має обмежений доступ. Дані обмеження будуть описані нижче в описані реалізації меню додатку.

Клас **FactoryManager** використовується для безпосереднього управління станками та електроенергією, а також регулювання температури та вологості приміщень. Даний клас представляє собою симуляцію моделі фабрики, тому не має жодного зв'язку з реальним пристроями.

Клас **FactoryManager** маніпулює моделями пристроїв, що представлені класом **Machine** - який є симуляцією моделі фабричних станків та конвеєрів. Розділення на типи «Станок» та «Конвейер» реалізовано за допомогою перелічувального типу даних c# Enum.

Симуляція станів фабричних станків також реалізована за допомогою перелічувального типу даних і описана такими станами як «Зупинено»(**Stopped**), «Працює»(**Working**), «Знеструмлений»(**Powerless**) та «Помилка»(**Error**).

Клас **StatManager** використовується для ведення статистики використання функціоналу додатку, зокрема: кількість натискання червоної кнопки, середній час безперебійної роботи пристроїв, середня вологість приміщень, середня температура тощо. Всі дані зберігаються в оперативній пам’яті пристрою і не можуть бути розглянуті в іншому сеансу роботи з додатком.

Модель взаємодії з реальними пристроями реалізована класом **EwelinkManager**. Взаємодія відбувається через використання портованої бібліотеки Ewelink.Api. Підключення до реальних пристроїв відбувається у декілька етапів. Початковим етапом є авторизація з використанням даних користувача використаних для авторизації у додатку. Після успішної авторизації методи даного класу використовуються для отримання інформації про реальний стан приладів типу «розумна розетка» які закріпленні за авторизованим користувачем. Бібліотека Ewelink.Api забезпечує повний контроль за станом приладів, які представлені класом **Device**. Зокрема стан «Online» чи підключений пристрій до мережі інтернет та чи підключений пристрії до екектромережі та стан тумблеру «On/Off».

**ViewModel** класи описують взаємодію з описаними вище классами та представляють собою кросплатформені моделі візуальних представлень сторінок та елементів списку у мобільному застосунку.

Класи **SplashViewModel**, **LoginViewModel**, **MainViewModel** призначені для основних сторінок. **SplashViewModel** - використовується для представлення початкової сторінки, яка показується при запуску і представляє собою екран за анімацією. Подібні екрани використовуються для довгих процесів початкової ініціалізації додатку з використанням первинного завантаження або автоматичної повторної авторизації.

**LoginViewModel** використовується для представлення екрану авторизації в застосунок. Як ядро використовується класс **AuthManager**.

Після успішної авторизації відбувається навігація на **MainViewModel**, що відповідає за головний екран мобільного додатку. Клас **MenuViewModel** використовується для представлення меню керування фабрикою. Для моделі візуального представлення елементів меню використовується клас **MenuItemViewModel**. Пункти меню додатку виконують функції навігації між різними внутрішніми сторінками додатку які представлені класами, що зазначені нижче:

1. **WelcomeViewModel** - використовується для початкової сторінки-вітання з мінімальною інформацією про застосунок.
2. **ElectricityViewModel** - містить у собі елементи взаємодії з класом **FactoryManager**, регулювання температури, вологості, тумблер подачі електроенергії до приміщень фабрики. При вимкненні електроенергії всі елементи стають недоступні користувачу поки тумблер електроенергії не буде приведений в активний стан.
3. **EquipmentViewModel** - використовується для відображення обладнання фабрики станків та конвеєрів та регулювання їх стану. Для елементів списку на даній сторінці, використовується класс **MachineItemViewModel.**
4. **SecurityViewModel** - використовується для окремої великої кнопки аварійної зупинки роботи фабрики, функціонал котрої полягає в виключенні електроенергії і встановлення неможливості її швидкого відновлення на протязі 10 секунд.
5. **EwelinkViewModel** - використовується для взаємодії з класом **EwelinkManager**, представляє собою сторінку на якій відображається стан реальних під’єднаних пристроїв. Для елементів списку на даній сторінці використвується клас **DeviceViewModel**. Він містить у собі функції взаємодії с моделями ewelink.Api - зокрема інформацією про пристрій та доступом до тумблера приладу.

6. **StatisticsViewModel** - сторінка, що недоступна для користувачів з роллю Worker(робітник). Як можна зрозуміти з назви описує візуальну модель для класу StatManager. Всі дані статистики відображаються на даній сторінці.

Всі інші класи типу -Fragment, -Activity, -ViewHolder використовуються для реалізації рівню **View**. Кожен з подібних класів взаємодіє з класами **ViewModel** для відображення та керування інформацією.

**Висновок до розділу 3**

У цьому розділі наведено детальний опис програмної реалізації мобільного додатку, а саме:

Use Case діаграма – простий та зрозумілий кожному опис фабрики, відношення компонентів програми один до одного.

Блок-схема підключення пристроїв до Wi-Fi мережі – схема, яка пояснює яким чином буде проводитися керування, що до чого підключається, та як налаштовується.

Діаграма класів – спеціальна діаграма класів мобільного додатку, яка демонструє структуру програми.

# 4 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЕКТУ

Першим етапом розробки мобільного додатку по керуванню фабрикою буде авторизація користувача (рис. 4.1). Це потрібно для того, щоб тільки ті люди, які працюють на підприємстві мали доступ до управління процесами.

У якості прикладу було розроблено дві облікові записи. Одна має повноваження директору фабрики, друга має повноваження звичайного працівника.


Рисунок 4.1 – Панель авторизації користувача

Після авторизації ми потрапляємо у меню з описом того, які функції може виконувати мобільний застосунок(рис 3.3).

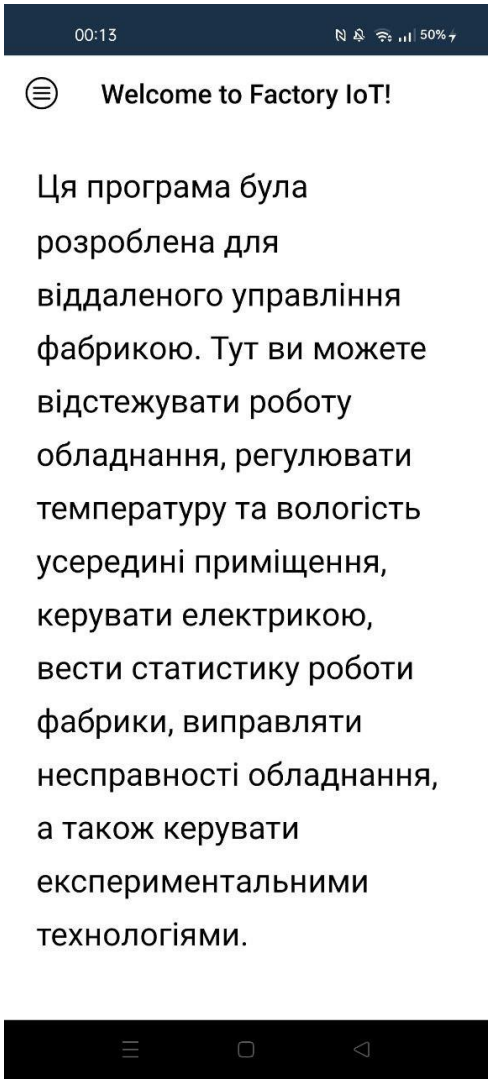


Рисунок 4.2 – Головна сторінка застосунку

Зліва зверху присутня іконка «Меню» застосунка. Натиснувши на нього ми потрапляємо у меню навігації по керуванню фабрикою(рис. 4.3).

У меню є такі основні пункти по керуванню, як:

− Electricity (Електрика) – панель по керуванню струмом, температурою та вологістю приміщення.

− Factory Equipment (Обладнання фабрики) – панель по керуванню обладнанням, станки та конвеєри.

− Security (Безпека) – панель з екстреною зупинкою роботи фабрики, при виникненні надзвичайної ситуації.

− Statistics (Статистика) – панель керування, на якій можна відслідковувати роботу фабрики.

– Smart Devices (Розумні пристрої) – експериментальна панель у якій проводиться тестування нових технологій, які ще не пройшли остаточної обкатки. На даний момент проводиться тестування розумної розетки.

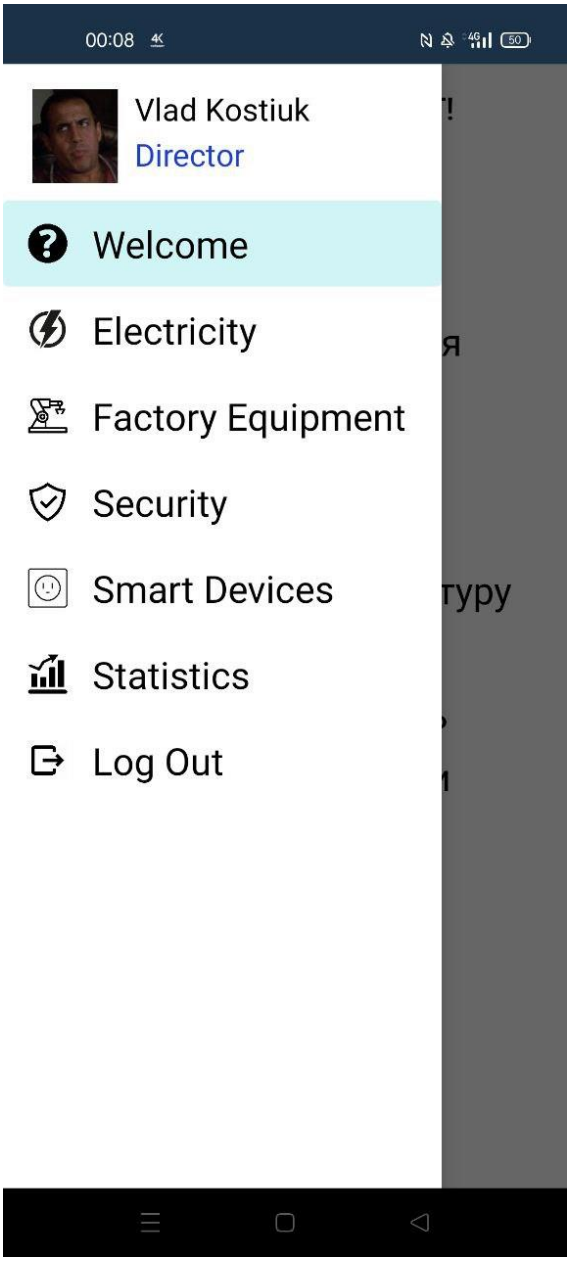


Рисунок 4.3 – Меню мобільного застосунку з посади Director

Також зверху відображається ім’я та посада людини, яка авторизувалася. Надпис Director позначає те, що авторизований голова фабрики, який має доступ до пункту «Статистики» підприємства.

У якості прикладу на (рис. 4.4) буде авторизований звичайний працівник, який має посаду «Worker», тобто працівник. У нього немає доступу до пункту

«Статистики», а також до «Smart Devices» тому що він має недостатньо повноважень для цього.

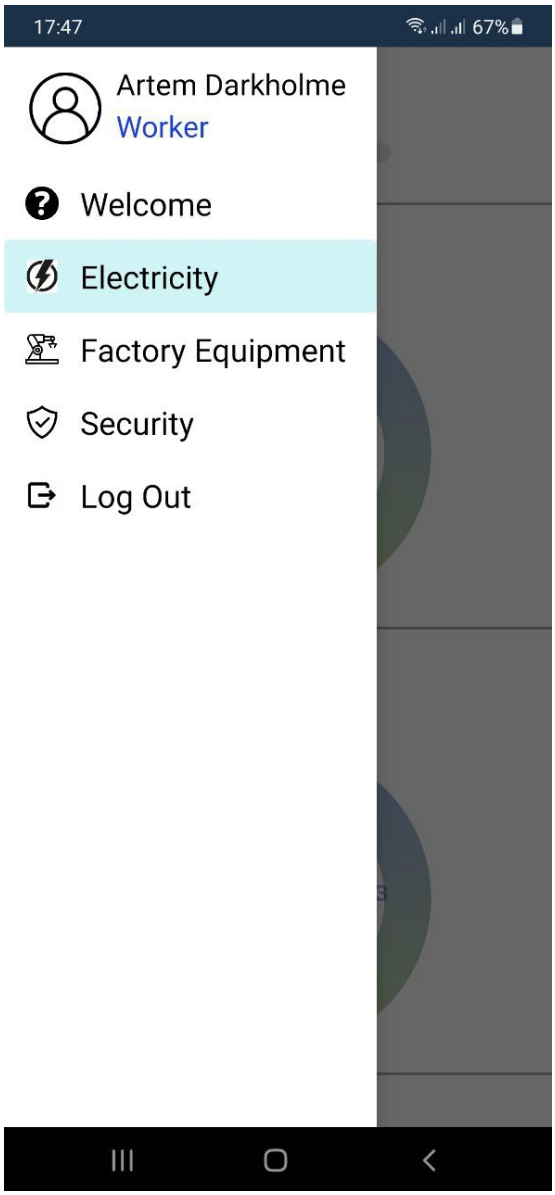


Рисунок 4.4 – Меню мобільного застосунку з посади Worker

Далі переходимо безпосередньо до кожного з пунктів меню для більш детального опису можливого функціоналу програми.

**Пункт меню «Electricity»** (рис. 4.5). Цей пункт надає можливість керувати електрикою на фабриці. Функціонал представляє з себе:

- подача струму;
- температура у приміщенні (Factory Temp);
- вологість у приміщенні (Humidity).

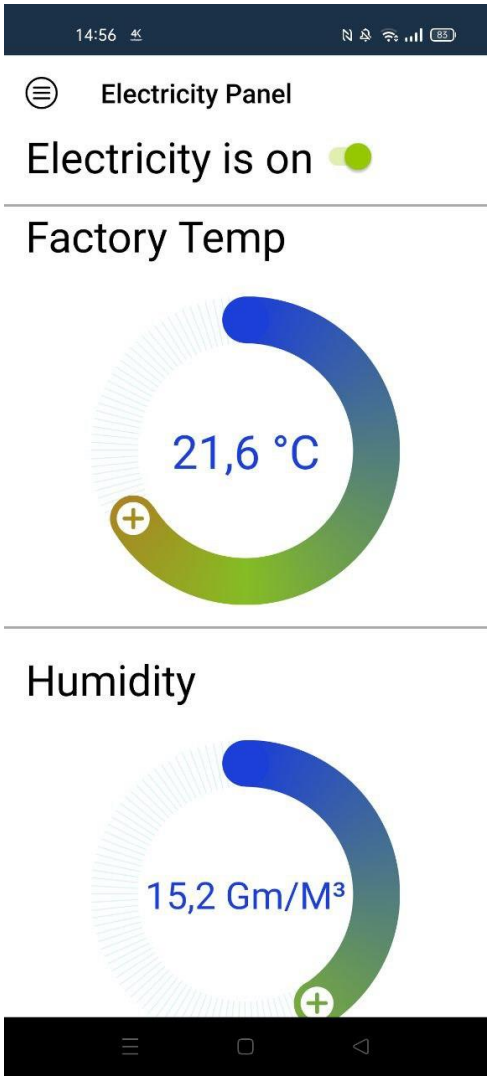


Рисунок 4.5 – Меню Electricity

Датчики по керуванню температурою та вологістю були розроблені у вигляді регуляторів. Це дозволяє дуже легко налаштовувати потрібні параметри.

**Пункт меню «Factory Equipment»** (рис. 4.6). Цей пункт надає можливість безпосередньо керувати обладнанням, а саме конвеєром та верстаками по обробці матеріалів.

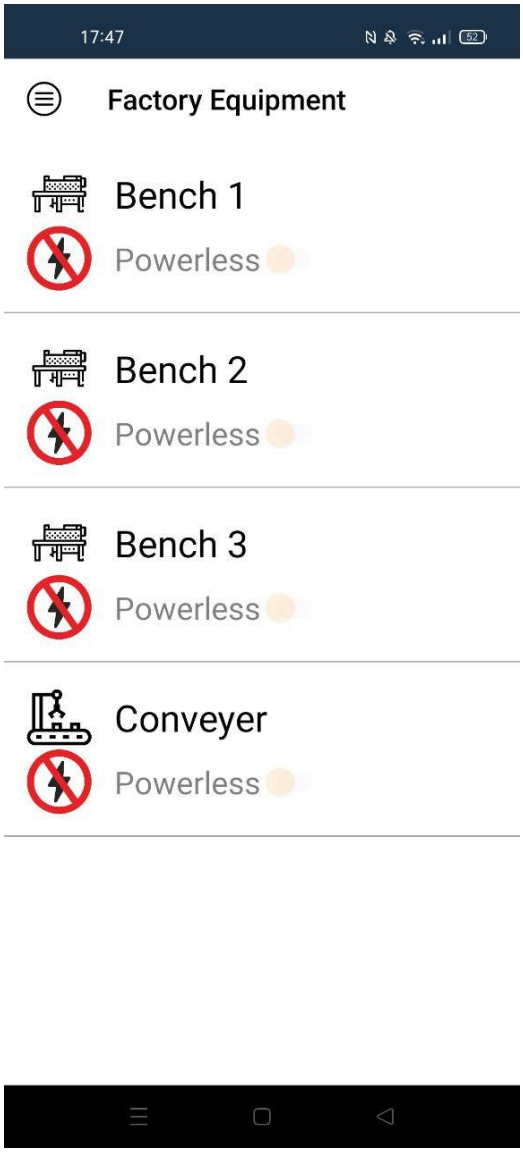


Рисунок 4.6 – Меню Factory Equipment

Якщо одразу перейти до цього пункту, програма покаже, що все обладнання знеструмлено, тому потрібно перш за все його ввімкнути. Але коли струм є, з’явиться можливість увімкнути обладнання (рис. 4.7).

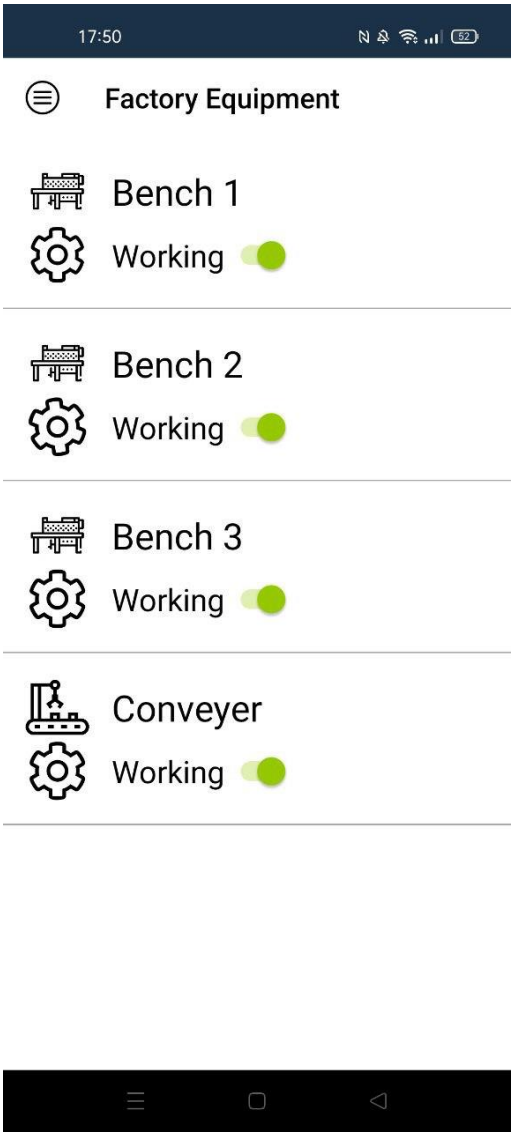


Рисунок 4.7 – Меню Factory Equipment

Також була розроблена генерація поломки або несправності обладнання (рис 4.8). Під кожним верстаком відображається його стан, працює він або ні. Є можливість вмикати лише один верстак, або тільки конвеєр.

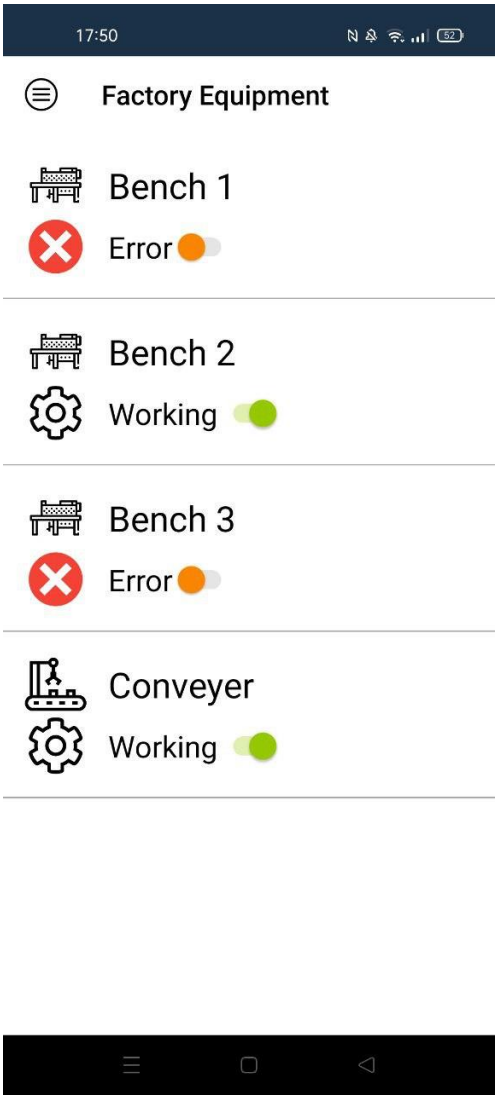


Рисунок 4.8 – Меню Factory Equipment

Під кожним верстаком відображається його стан, працює він або ні. Є можливість вмикати лише один верстак, або тільки конвеєр.

**Пункт меню «Security»** (рис 4.9). Цей пункт був розроблений для того, щоб при виникненні небезпечної ситуації була можливість екстрено зупинити всі можливі процеси. В якості інтерфейсу була зроблена велика анімована червона кнопка. При натискані показується ще одна анимація (рис. 4.10).

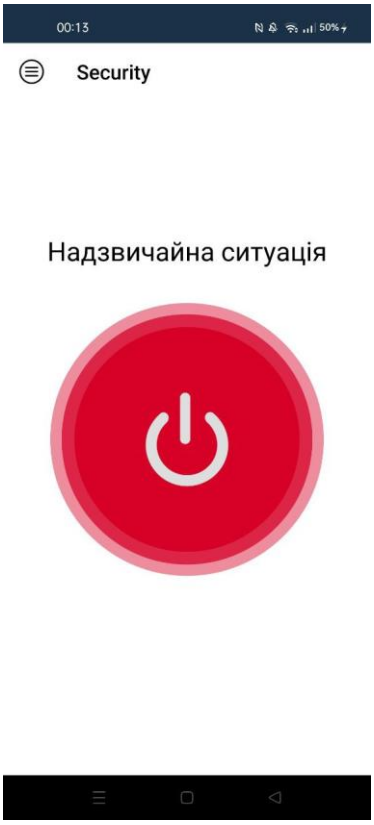


Рисунок 4.9 – Меню Security


Рисунок 4.10 – Натискання червоної кнопки

При натисканні на червону кнопку всі процеси вимикаються, та деякий час після натискання кнопки неможливо нічого ввімкнути.

**Пункт меню «Statistics»** (рис 4.11). Найважливіший пункт для будь-якого керівники підприємством. У цьому меню можна відстежувати то розраховувати якість роботи фабрики, дивитись скільки часу працює обладнання, середню температуру повітря та вологості, кількість зупинок машин через несправність та кількість натискань на червону кнопку.

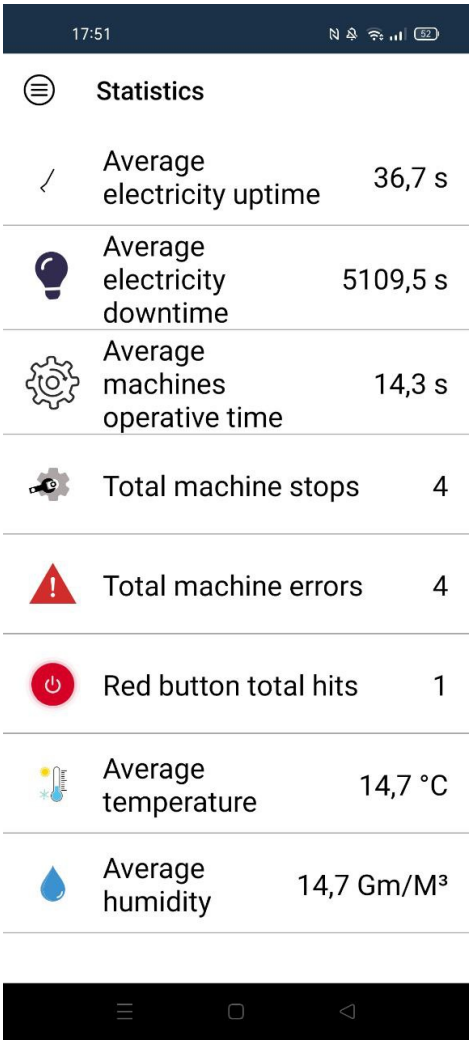


Рисунок 4.11 – Меню Statistics

Важливе зауваження, до цього пункту меню має доступ тільки директор фабрики, інші робітники до статистики доступу не мають.

**Пункт меню «Smart Devices»** (рис 4.12 – 4.13). Пункт меню, який був розроблений додатково для реальної демонстрації роботи мобільного додатку.

В даному випадку використовується «розумна» розетка, якою можна вести керування дистанційно за допомогою смартфону та Wi-Fi мережі.

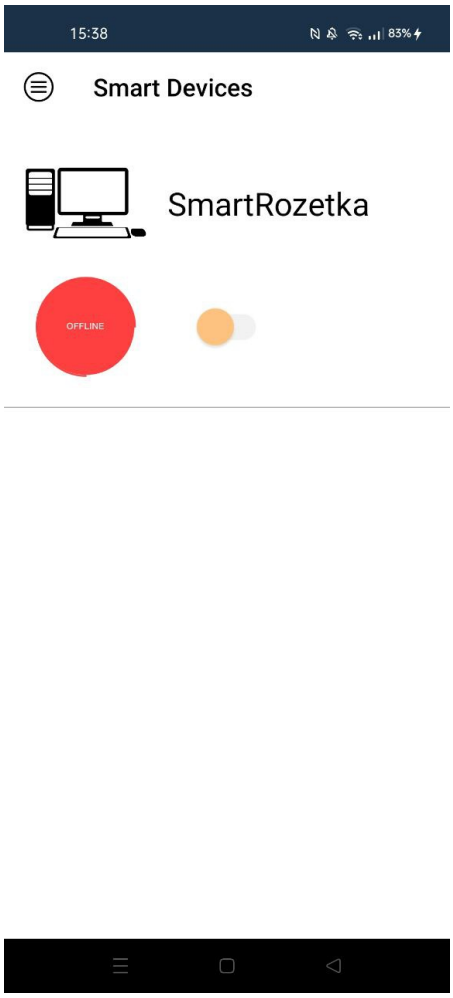


Рисунок 4.12 – Меню Smart Devices (статус Offline)

Так виглядає меню коли девайса немає у мережі.

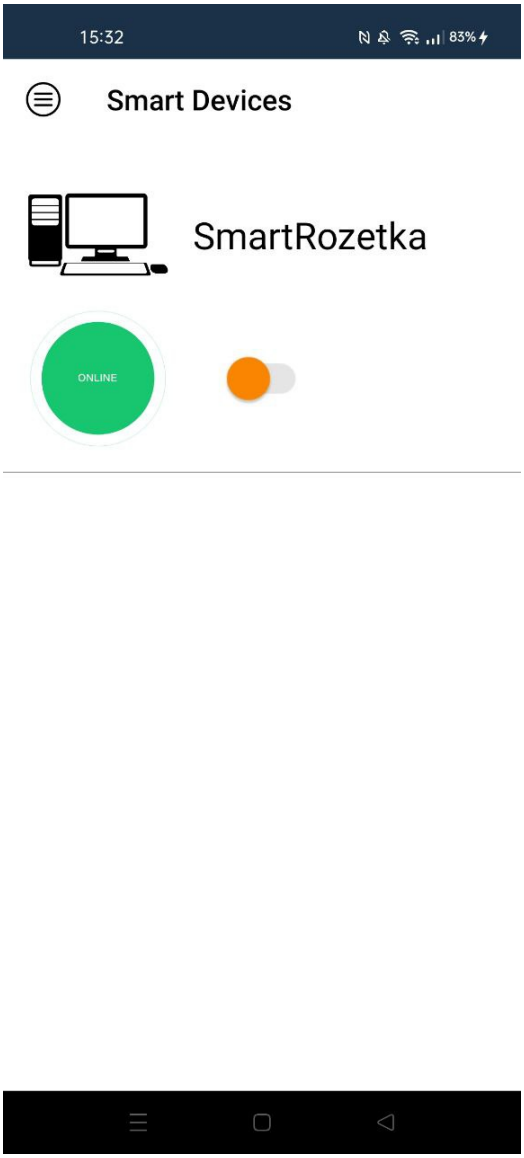


Рисунок 4.13 – Меню Smart Devices (статус Online)

Так виглядає меню коли девайс у мережі та готовий до роботи.

У наступних картинках (рис. 4.14 – 4.15) можна побачити девайс у реальному житті. Якщо кнопка живлення горить зеленим девайс у мережі, але не включений. А якщо кнопка живлення змінює колір с зеленого на жовто-зелений це позначає, що девайс включений.


Рисунок 4.14 – Девайс у мережі, але не включений

На цьому рисунку можна подивитися на індикацію девайсу. Зелений колір розетки говорить о тому, що вона знаходиться у мережі, але не включена.


Рисунок 4.15 – Девайс у мережі та включений

На цьому рисунку можна помітити жовтий колір індикатора. Це позначає, що розетка знаходиться у мережі та включена, нею можна користуватися.

**Висновок до розділу 4**

В останньому розділі дипломної роботи було продемонстровано роботу мобільного застосунку, наведено інтерфейс програми, а також усі її функції. У якості прикладу використовувався девайс «розумна» розетка з можливістю підключення до мобільного телефону.

Таким чином, була продемонстрована технічна можливість керування підприємством.

# ВИСНОВКИ

В ході виконання магістерської дипломної роботи було проведено дослідження керуючих систем та підприємств з використанням IIoT технологій.

Дослідивши засоби керування системами виробництва було виявлено, що у сучасному світі якість та швидкість виробництва зробила великий скачок, тому що автоматизовані комп’ютерні системи можуть виконувати свої функції без втручання людини. Розглянуто промислові комп’ютери, їх технічні характеристики та засоби використання.

Було проведено аналіз діючих IIoT систем, а також виявлені переваги та недоліки кожної з систем, наведені приклади їх продукції та в якій сфері виробництва вони використовуються.

Розглянувши існуючи системи по керуванню, за допомогою програмного середовища Visual Studio 2022, було розроблено унікальний мобільний застосунок на платформі Andorid OC, який базується на керуванні пристроями, підключених до Wi-Fi мережі фабрики.

Для простого розуміння структурної складової програми було створено систему підключення, за допомогою Use Case діаграми, а також наведено блок-схему того, яким чином мобільний застосунок проводить віддалене керування. Наведені пояснення щодо процесу створення мобільного застосунку. Для наглядного відображення було створено діаграму класів за допомогою програми Visual Paradigm.

Було протестовано мобільний застосунок та виявлено, що він повністю виконує свої функції та працює без нарікань. Для наглядного прикладу була використана «розумна» розетка фірми Sonoff.

Якщо використати більш складне та дороге обладнання та підключити його до застосунку, можна значно розширити його функціонал, який буде дозволяти виконувати складні технологічні операції на підприємстві,

наприклад, додати машини-роботів, які виконують збирання або сортування продукції.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Поняття IIoT. URL: https://www.hpe.com/ru/ru/what-is/industrial-iot.html
2. Цифрова промисловість. URL: https://www.intel.ru/content/www/ru/ru/internet-of-things/industrial-iot/overview.html
3. AWS IoT. URL: https://aws.amazon.com/ru/iot/solutions/industrial-iot/
4. Функції IIoT. URL: https://inlnk.ru/1PLDgY
5. Industry 4.0. URL: https://inlnk.ru/0Q5Jgd
6. Моделювання виробничих пристроїв. URL: http://economyandsociety.in.ua/journals/15_ukr/49.pdf
7. Автоматизація виробництва і виробничих процесів. URL: https://peskiadmin.ru/uk/avtomatizaciya-proizvodstva-i-proizvodstvennyh-processov.html
8. IoT та бізнес. URL: https://woxapp.com/ru/our-blog/internet-of-things-iot/
9. IT-рішення IoT. URL: https://www.it.ua/ru
10. Bystronic System. URL: https://www.bystronic.com/ru/
11. Eton System. URL: https://www.etonsystems.com/en
12. IoTJi by deps. URL: https://iotji.io/ru/
13. LoRaWAN system. URL: https://lora-alliance.org/
14. Промислові комп'ютери. URL: https://www.svaltera.ua/catalog/736/
15. .NET Framework URL: https://docs.microsoft.com/ru-ru/dotnet/framework/
16. Xamarin. URL: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin
17. EWeLink. URL: https://www.proline-rus.ru/articles/instrukcija-dlja-prilozhenija-ewelink-74/

18. MVVM паттерн. URL:
https://professorweb.ru/my/WPF/documents_WPF/level36/36_5.php
19. Visual Paradigm. URL: https://www.visual-paradigm.com/

# Додаток А

## MenuViewModel.cs

```
using System.Collections.Generic;
using System.Linq;
using FactoryIoT.Auth;
using FactoryIoT.Res;
using MvvmCross.Commands;
using MvvmCross.ViewModels;

namespace FactoryIoT.VM
{
public class MenuViewModel : VMBase
{
private readonly UserContext;

public MenuViewModel(UserContext userContext)
{
this.userContext = userContext;
Items = GetMenuItems().ToList();
ItemSelectedCommand = new MvxCommand<MvxViewModel>(OnItemSelectedAsync);
HideMenuInteraction = new MvxInteraction();
}

public MvxInteraction HideMenuInteraction { get; }

public IEnumerable<MvxViewModel> Items { get; }

public IMvxCommand<MvxViewModel> ItemSelectedCommand { get; }

public IEnumerable<MvxViewModel> GetMenuItems()
{
yield return new TitleMenuItemViewModel(userContext.CurrentUser);
yield return new MenuItemViewModel(AppStrings.WelcomeTitle, "faq",
() => NavigationService.Navigate<WelcomeViewModel>())
{
IsSelected = true
};

yield return new MenuItemViewModel(AppStrings.ElectricityTitle, "light",
() => NavigationService.Navigate<ElectricityViewModel>());

yield return new MenuItemViewModel(AppStrings.EquipmentTitle, "equipment",
() => NavigationService.Navigate<EquipmentViewModel>());

yield return new MenuItemViewModel(AppStrings.SecurityTitle, "shield",
() => NavigationService.Navigate<SecurityViewModel>());

if (userContext.CurrentUser.Role == Role.Director)
{
yield return new MenuItemViewModel(AppStrings.StatisticsTitle, "stat",
() => NavigationService.Navigate<StatisticsViewModel>());
}
yield return new MenuItemViewModel("Log Out", "logout", () => NavigationService.Navigate<LoginViewModel>());
}

private void OnItemSelectedAsync(MvxViewModel item)
{
if (item is MenuItemViewModel menuItemViewModel)
{
var unselected = TryUnselectMenuItem(menuItemViewModel);

HideMenuInteraction.Raise();
```

```
if (unselected)
{
menuItemViewModel.IsSelected = true;
menuItemViewModel.SelectCommand.Execute();
}
}
}

private bool TryUnselectMenuItem(MenuItemViewModel menuItemViewModel)
{
var previouslySelectedItem = Items
.OfType<MenuItemViewModel>()
.FirstOrDefault(item => item.IsSelected);

if (previouslySelectedItem == null || menuItemViewModel == previouslySelectedItem)
{
return false;
}
previouslySelectedItem.IsSelected = false;
return true;
}
}
}
```

## ElectricityViewModel.cs

```
using System;
using FactoryIoT.Common;
using FactoryIoT.Res;
namespace FactoryIoT.VM
{
public class ElectricityViewModel : VMBase
{
private readonly FactoryManager;
private double tempWeight;
private double wetWeight;
public ElectricityViewModel(FactoryManager factoryManager)
{
this.factoryManager = factoryManager;
factoryManager.ElectricityStateChanged += FactoryManagerElectricityStateChanged;
factoryManager.CriticalStateChanged += FactoryManagerCriticalStateChanged;

tempWeight =
(RoomTemp - FactoryManager.MinTemp) /
(FactoryManager.MaxTemp - FactoryManager.MinTemp);

wetWeight =
(RoomWet - FactoryManager.MinTemp) /
(FactoryManager.MaxWet - FactoryManager.MinWet);
}
private void FactoryManagerCriticalStateChanged(object sender, EventArgs e)
{
RaisePropertyChanged(nameof(IsNotCriticalState));
}
private void FactoryManagerElectricityStateChanged(object sender, EventArgs e)
{
RaisePropertyChanged(nameof(IsElectricityOn));
RaisePropertyChanged(nameof(ElectricityTitle));
}
public string Title => AppStrings.ElectricityPanelTitle;
public string FactoryTempTitle => AppStrings.FactoryTempTitle;
public bool IsNotCriticalState => !factoryManager.IsCriticalState;
public string WetTitle => AppStrings.HumidityTitle;
public string ElectricityTitle =>
IsElectricityOn ?
AppStrings.ElectricityOnTitle :
AppStrings.ElectricityOffTitle;

public double TempValue => FactoryManager.MinTemp +
(FactoryManager.MaxTemp - FactoryManager.MinTemp) * TempWeight;

public double TempWeight
{
get => tempWeight;
set => SetProperty(ref tempWeight, value, () =>
```

```
{
RaisePropertyChanged(nameof(TempValue));
RaisePropertyChanged(nameof(FactoryTempPresentation));
RoomTemp = TempValue;
});
}
public double RoomTemp
{
get => factoryManager.RoomTemp;
set
{
factoryManager.RoomTemp = value;
RaisePropertyChanged();
}
}
public string FactoryTempPresentation => TempValue.FormatAsTemp();

public double WetValue => FactoryManager.MinWet +
(FactoryManager.MaxWet - FactoryManager.MinWet) * WetWeight;

public double WetWeight
{
get => wetWeight;
set => SetProperty(ref wetWeight, value, () =>
{
RaisePropertyChanged(nameof(WetValue));
RaisePropertyChanged(nameof(FactoryWetPresentation));
RoomWet = WetValue;
});
}

public string FactoryWetPresentation => WetValue.FormatAsWetness();

public double RoomWet
{
get => factoryManager.Wetness;
set
{
factoryManager.Wetness = value;
RaisePropertyChanged();
}
}
public bool IsElectricityOn
{
get => factoryManager.IsElectricityEnabled;
set => factoryManager.IsElectricityEnabled = value;
}
}
}
```

## EquipmentViewModel.cs

```
using System.Collections.Generic;
using System.Linq;
using FactoryIoT.Res;

namespace FactoryIoT.VM
{
public class EquipmentViewModel : VMBase
{
public EquipmentViewModel(FactoryManager factoryManager)
{
Machines = factoryManager.Machines.Select(m => new MachineItemViewModel(m)).ToList();
}

public string EquipmentTitle => AppStrings.EquipmentTitle;

public IEnumerable<MachineItemViewModel> Machines { get; }
}
}
```

## LoginViewModel.cs

```
using System;
```

```
using System.Diagnostics;
using System.Threading.Tasks;
using FactoryIoT.Auth;
using FactoryIoT.Common;
using FactoryIoT.Ewelink;
using FactoryIoT.Res;
using MvvmCross.Commands;

namespace FactoryIoT.VM
{
public class LoginViewModel : VMBase
{
private readonly AuthManager;
private readonly EwelinkManager;

private bool isAuthFailed;

private string userName;
private string password;
private bool isLoginExecuting;

public LoginViewModel(AuthManager, EwelinkManager ewelinkManager)
{
this.authManager = authManager;
this.ewelinkManager = ewelinkManager;
LoginCommand = new MvxAsyncCommand(LoginAsync);
UserName = "noyneim3@gmail.com";
Password = "NoyNeim3/11";
}

public string AppName => Constants.AppName;

public string LoginTitle => AppStrings.LoginTitle;

public string LoginErrorTitle => AppStrings.LoginErrorTitle;

public string UserName
{
get => userName;
set => SetProperty(ref userName, value);
}

public string UserNamePlaceholder => AppStrings.UsernamePlaceholder;

public string Password
{
get => password;
set => SetProperty(ref password, value);
}

public string PasswordPlaceholder => AppStrings.PasswordPlaceholder;

public IMvxAsyncCommand LoginCommand { get; }

public bool IsLoginExecuting
{
get => isLoginExecuting;
set => SetProperty(ref isLoginExecuting, value);
}

public bool IsAuthFailed
{
get => isAuthFailed;
set => SetProperty(ref isAuthFailed, value);
}

private async Task LoginAsync()
{
IsAuthFailed = false;
IsLoginExecuting = true;
//NOTE: simulation of long process
await Task.Delay(2000);

var user = authManager.LoginUser(UserName, Password);
```

```
if (user == null)
{
IsAuthFailed = true;
}
else
{
try
{
await ewelinkManager.ConnectToEwelinkAsync();
}
catch (Exception ex)
{
Debug.WriteLine(ex);
}

await NavigationService.Navigate<MainViewModel>();

IsAuthFailed = false;
}

IsLoginExecuting = false;
}
}
}
```

## MachineViewModel.cs

```
using System;
using FactoryIoT.Enums;
using FactoryIoT.Factory;
using FactoryIoT.Res;

namespace FactoryIoT.VM
{
public class MachineItemViewModel : VMBase
{
private readonly Machine;

public MachineItemViewModel(Machine machine)
{
this.machine = machine;
machine.StateChanged += MachineStateChanged;
}

public string Id => machine.Id;

public string Name => machine.Name;

public MachineType Type => machine.Type;

public MachineState State
{
get => machine.State;
set
{
machine.State = value;
RaisePropertyChanged(nameof(State));
RaisePropertyChanged(nameof(StateDescription));
RaisePropertyChanged(nameof(IsWorking));
RaisePropertyChanged(nameof(IsOperative));
}
}

public bool IsEnabled => machine.State != MachineState.Powerless;

public bool IsOperative => IsEnabled && machine.State != MachineState.Error;

public bool IsWorking
{
get => State == MachineState.Working;
set => State = value ? MachineState.Working : MachineState.Stopped;
}

public string StateDescription
{
```

```
get => State switch
{
MachineState.Error => AppStrings.MachineErrorStateTitle,
MachineState.Stopped => AppStrings.MachineStoppedStateTitle,
MachineState.Powerless => AppStrings.MachinePowerlessStateTitle,
MachineState.Working => AppStrings.MachineWorkingStateTitle,
_ => throw new NotImplementedException()
};
}

public string TypeDescription
{
get => Type switch
{
MachineType.Bench => AppStrings.BenchTypeTitle,
MachineType.Conveyor => AppStrings.BenchTypeTitle,
_ => throw new NotImplementedException()
};
}

private void MachineStateChanged(object sender, EventArgs e)
{
RaisePropertyChanged(nameof(State));
RaisePropertyChanged(nameof(StateDescription));
RaisePropertyChanged(nameof(IsEnabled));
RaisePropertyChanged(nameof(IsWorking));
RaisePropertyChanged(nameof(IsOperative));
}
}
}
```

## SecurityViewModel.cs

```
using System;
using System.Timers;
using FactoryIoT.Res;
using MvvmCross.Commands;

namespace FactoryIoT.VM
{
public class SecurityViewModel : VMBase
{
private DateTime disableTime;
private Timer;
private FactoryManager;

public SecurityViewModel(FactoryManager factoryManager)
{
timer = new Timer();
timer.Interval = 50;
this.factoryManager = factoryManager;
RedButtonCommand = new MvxCommand(DoRedButton);
factoryManager.CriticalStateChanged += FactoryManagerCriticalStateChanged;
}

public float DisableProgress => (float)(DateTime.Now - disableTime).TotalMilliseconds / 10000;

public bool IsCriticalState => factoryManager.IsCriticalState;

public bool IsNotCriticalState => !factoryManager.IsCriticalState;

private void FactoryManagerCriticalStateChanged(object sender, EventArgs e)
{
RaisePropertyChanged(nameof(IsCriticalState));
RaisePropertyChanged(nameof(IsNotCriticalState));
RaisePropertyChanged(nameof(RedButtonTitle));

if (!IsCriticalState)
{
timer.Stop();
}
}

private void DoRedButton()
{
```

```
factoryManager.PowerOff();
AnimateDisableProgress();
}

private void AnimateDisableProgress()
{
disableTime = DateTime.Now;
timer.Start();
timer.Elapsed += TimerElapsed;
}

private void TimerElapsed(object sender, ElapsedEventArgs e)
{
RaisePropertyChanged(nameof(DisableProgress));
}

public string SecurityTitle => AppStrings.SecurityTitle;

public string RedButtonTitle =>
IsCriticalState ?
AppStrings.FactoryIsInCriticalStateTitle :
AppStrings.RedButtonTitle;

public IMvxCommand RedButtonCommand { get; }
}
}
```

## StatitsticsViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using FactoryIoT.Common;
using FactoryIoT.Factory;
using FactoryIoT.Res;

namespace FactoryIoT.VM
{
public class StatisticsViewModel : VMBase
{
private readonly StatManager;

public StatisticsViewModel(StatManager statManager)
{
this.statManager = statManager;
statManager.StatUpdated += StatManagerStatUpdated;
}

private StatInfo statistic => statManager.Statistic;

public List<StatItemViewModel> StatItems => GenerateStatItems().ToList();

public string StatisticsTitle => AppStrings.StatisticsTitle;

private IEnumerable<StatItemViewModel> GenerateStatItems()
{
yield return new StatItemViewModel
{
Name = AppStrings.ElectricityUptimeTitle,
Value = statistic.AverageElectricityUptime.ToString("F1") + " s",
AnimationName = "lightning"
};

yield return new StatItemViewModel
{
Name = AppStrings.ElectricityDowntimeTitle,
Value = statistic.AverageElectricityDowntime.ToString("F1") + " s",
AnimationName = "no_light"
};
```

```
yield return new StatItemViewModel
{
Name = AppStrings.MachineOperativeTimeTitle,
Value = statistic.AverageMachineOperativeTime.ToString("F1") + " s",
AnimationName = "gear_working"
};

yield return new StatItemViewModel
{
Name = AppStrings.MachineStopsTitle,
Value = statistic.MachineStops.ToString(),
AnimationName = "gear_operative"
};

yield return new StatItemViewModel
{
Name = AppStrings.ErrorsTitle,
Value = statistic.ErrorsCount.ToString(),
AnimationName = "error"
};

yield return new StatItemViewModel
{
Name = AppStrings.RedButtonHitCountTitle,
Value = statistic.RedButtonHitCount.ToString(),
AnimationName = "redbutton"
};

yield return new StatItemViewModel
{
Name = AppStrings.AverageTempTitle,
Value = statistic.AverageTemperature.FormatAsTemp(),
AnimationName = "temperature"
};

yield return new StatItemViewModel
{
Name = AppStrings.AverageWetnessTitle,
Value = statistic.AverageTemperature.FormatAsWetness(),
AnimationName = "wetness"
};
}

private void StatManagerStatUpdated(object sender, EventArgs e)
{
RaisePropertyChanged(nameof(StatItems));
}
}
}
```

## AuthManager.cs

```
using System.Collections.Generic;
using System.Linq;

namespace FactoryIoT.Auth
{
public class AuthManager
{
private readonly UserContext;
private readonly List<User> users;

public AuthManager(UserContext userContext)
{
this.userContext = userContext;

users = new List<User>
{
new User
{
UserLogin = "vlad@gmail.com",
Password = "0000",
UserImageId = "director",
UserName = "Vlad Kostiuk",
```

```
Role = Role.Director,
},

new User
{
UserLogin = "artem@gmail.com",
Password = "1111",
UserImageId = "user",
UserName = "Artem Darkholme",
Role = Role.Worker
},
};
}

public User LoginUser(string userName, string password)
{
var user = users.FirstOrDefault(u => u.UserLogin == userName && u.Password == password);

if (user != null)
{
userContext.CurrentUser = user;
}

return user;
}
}
}
```

## FactoryManager.cs

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using FactoryIoT.Enums;
using FactoryIoT.Factory;
namespace FactoryIoT.VM
{
public class FactoryManager
{
private readonly StatManager;
private DateTime electricityEnabledAt;
private DateTime electricityDisabledAt;
private bool isElectricityEnabled;
private double roomTemp = MinTemp;
private double wetness = MinWet;
private bool isCricitcalState;
public const double MaxTemp = 30d;
public const double MinTemp = 5d;
public const double MinWet = 5d;
public const double MaxWet = 30d;
public event EventHandler ElectricityStateChanged;
public event EventHandler CriticalStateChanged;
public FactoryManager(StatManager statManager)
{
this.statManager = statManager;
electricityDisabledAt = DateTime.Now;
statManager.AddTempChange(MinWet);
statManager.AddWetnessChange(MinWet);
Machines = new List<Machine>
{
new Machine
{
Id = "100",
Name = "Bench 1",
Type = MachineType.Bench,
},

new Machine
{
Id = "200",
Name = "Bench 2",
Type = MachineType.Bench,
},

new Machine
```

```
{
Id = "300",
Name = "Bench 3",
Type = MachineType.Bench,
},

new Machine
{
Id = "400",
Name = "Conveyer",
Type = MachineType.Conveyor
}
};
}
public List<Machine> Machines { get; }

public double RoomTemp
{
get => roomTemp;

set
{
roomTemp = value;
statManager.AddTempChange(value);
}
}

public double Wetness
{
get => wetness;
set
{
wetness = value;
statManager.AddWetnessChange(value);
}
}

public bool IsCriticalState => isCricitcalState;

public bool IsElectricityEnabled
{
get => isElectricityEnabled;
set
{
if (isElectricityEnabled && !value)
{
OnElectricityDisabled();
}

if (!isElectricityEnabled && value)
{
OnElectricityEnabled();
}

isElectricityEnabled = value;
UpdateElectricityStatus();
ElectricityStateChanged?.Invoke(this, EventArgs.Empty);
}
}

private void OnElectricityDisabled()
{
var uptime = DateTime.Now - electricityEnabledAt;
statManager.AddElectricityUptime(uptime.TotalSeconds);

electricityDisabledAt = DateTime.Now;
}

private void OnElectricityEnabled()
{
electricityEnabledAt = DateTime.Now;

var uptime = DateTime.Now - electricityDisabledAt;
statManager.AddElectricityDowntime(uptime.TotalSeconds);
}
```

```
private void UpdateElectricityStatus()
{
Machines.ForEach(m => m.SetPower(IsElectricityEnabled));
}

public async void PowerOff()
{
statManager.AddRedButtonClick();

isCricitcalState = true;
CriticalStateChanged?.Invoke(this, EventArgs.Empty);
IsElectricityEnabled = false;

await Task.Delay(10000);

isCricitcalState = false;
CriticalStateChanged?.Invoke(this, EventArgs.Empty);
}
}
}
```

## Machine.cs

```
using System;
using System.Threading;
using System.Threading.Tasks;
using FactoryIoT.Enums;
using MvvmCross;
namespace FactoryIoT.Factory
{
public class Machine
{
private static StatManager => Mvx.IoCProvider.Resolve<StatManager>();
private static readonly Random = new Random();
private const int SimulationOperativeCycles = 10;
private const int ErrorMaxCycles = 20;
private const int SimulationCycleTime = 1000;
private CancellationTokenSource simulationCancellationTokenSource;
private MachineState state = MachineState.Powerless;

public string Id { get; set; }
public string Name { get; set; }
public MachineType Type { get; set; }
public MachineState State
{
get => state;
set
{
state = value;

if (value == MachineState.Working)
{
simulationCancellationTokenSource?.Cancel();
simulationCancellationTokenSource = new CancellationTokenSource();
Run(simulationCancellationTokenSource.Token);
}
else
{
simulationCancellationTokenSource?.Cancel();
}
}
}
public event EventHandler StateChanged;

public void SetPower(bool on)
{
State = on ? MachineState.Stopped : MachineState.Powerless;
RaiseStateChanged();
}

private void RaiseStateChanged()
{
StateChanged?.Invoke(this, EventArgs.Empty);
}
```

```
private async Task Run(CancellationToken cancellationToken)
{
var timeStarted = DateTime.Now;

try
{
var errorIn = random.Next(SimulationOperativeCycles, ErrorMaxCycles);
await Task.Delay(errorIn * SimulationCycleTime, cancellationToken);
State = MachineState.Error;
StatManager.AddOperativeTimeStat(errorIn);
StatManager.AddErrorHappen();
RaiseStateChanged();
}
catch
{
}
finally
{
var actualOperativeTime = (DateTime.Now - timeStarted).TotalSeconds;

StatManager.AddMachineStop();
StatManager.AddOperativeTimeStat(actualOperativeTime);
}
}
}
}
```

## StatManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
namespace FactoryIoT.Factory
{
public class StatManager
{
private readonly List<double> tempChanges;
private readonly List<double> wetnessChanges;
private readonly List<double> machineOperativeTimes;
private readonly List<double> electricityUptime;
private readonly List<double> electricityDowntime;
private int redButtonClicks;
private int errors;
private int machineStops;
public event EventHandler StatUpdated;
public StatManager()
{
tempChanges = new List<double>();
machineOperativeTimes = new List<double>();
electricityUptime = new List<double>();
electricityDowntime = new List<double>();
wetnessChanges = new List<double>();
}
public StatInfo Statistic => new StatInfo
{
AverageTemperature = tempChanges.Average(),
AverageWetness = wetnessChanges.Average(),
AverageMachineOperativeTime = machineOperativeTimes.Count == 0 ? 0 : machineOperativeTimes.Average(),
AverageElectricityUptime = electricityUptime.Count == 0 ? 0 : electricityUptime.Average(),
AverageElectricityDowntime = electricityDowntime.Count == 0 ? 0 : electricityDowntime.Average(),
MachineStops = machineStops,
RedButtonHitCount = redButtonClicks,
ErrorsCount = errors
};
public void AddTempChange(double temp)
{
tempChanges.Add(temp);
RaiseStatUpdated();
}
public void AddRedButtonClick()
{
redButtonClicks++;
RaiseStatUpdated();
}
```

```
public void AddOperativeTimeStat(double seconds)
{
machineOperativeTimes.Add(seconds);
RaiseStatUpdated();
}
public void AddElectricityUptime(double seconds)
{
electricityUptime.Add(seconds);
RaiseStatUpdated();
}
public void AddErrorHappen()
{
errors++;
RaiseStatUpdated();
}
public void AddMachineStop()
{
machineStops++;
RaiseStatUpdated();
}
public void AddElectricityDowntime(double seconds)
{
electricityDowntime.Add(seconds);
RaiseStatUpdated();
}
public void AddWetnessChange(double wetness)
{
wetnessChanges.Add(wetness);
RaiseStatUpdated();
}

private void RaiseStatUpdated()
{
StatUpdated?.Invoke(this, EventArgs.Empty);
}
}
}
```

## EwelinkViewModel.cs

```
using System.Linq;
using System.Threading.Tasks;
using FactoryIoT.Ewelink;
using FactoryIoT.Res;
using MvvmCross.Commands;
using MvvmCross.ViewModels;

namespace FactoryIoT.VM
{
public class EwelinkViewModel : VMBase
{
private readonly EwelinkManager;
private bool isLoading;
private bool isRefreshing;

public EwelinkViewModel(EwelinkManager ewelinkManager)
{
Devices = new MvxObservableCollection<DeviceViewModel>();

this.ewelinkManager = ewelinkManager;
ewelinkManager.DeviceStatusChanged += EwelinkManagerDeviceStatusChanged;
RefreshCommand = new MvxAsyncCommand(RefreshAsync);
}

public string Title => AppStrings.EwelinkDevicesTitle;

public MvxObservableCollection<DeviceViewModel> Devices { get; }

public IMvxCommand RefreshCommand { get; }

public bool IsLoading
{
get => isLoading;
set => SetProperty(ref isLoading, value);
}
```

```
public bool IsRefreshing
{
get => isRefreshing;
set => SetProperty(ref isRefreshing, value);
}

public override async Task Initialize()
{
IsLoading = true;

var devices = await ewelinkManager.GetDevicesAsync();

IsLoading = false;

var deviceVmList = devices.Select(d => new DeviceViewModel(d, OnDevicePowerSwitch)).ToArray();
Devices.ReplaceWith(deviceVmList);
}

protected override void OnViewDestroy(bool viewFinishing)
{
base.OnViewDestroy(viewFinishing);

ewelinkManager.DeviceStatusChanged -= EwelinkManagerDeviceStatusChanged;
}

private async void OnDevicePowerSwitch(DeviceViewModel deviceVm)
{
await ewelinkManager.SetDevicePower(deviceVm.EwelinkDevice, deviceVm.IsDevicePowerOn);
}

private async void EwelinkManagerDeviceStatusChanged(object sender, DeviceChangedEventArgs e)
{
var deviceToUpdate = Devices.FirstOrDefault(v => v.EwelinkDevice.Deviceid == e.DeviceId);
var device = await ewelinkManager.GetDeviceAsync(e.DeviceId);
if (device == null)
{
return;
}

if (e.SwitchChange.HasValue)
{
device.Paramaters.Switch = e.SwitchChange;
}

if (e.IsOnline.HasValue)
{
device.Online = e.IsOnline;
}

deviceToUpdate.EwelinkDevice = device;
}

private async Task RefreshAsync()
{
IsRefreshing = true;

await Initialize();

IsRefreshing = false;
}
}
}
```

DeviceViewModel.cs

```
using System;
using EWeLink.Api.Models;
using MvvmCross.Commands;

namespace FactoryIoT.VM
{
public class DeviceViewModel : VMBase
{
private Device ewelinkDevice;
```

```
public DeviceViewModel(Device ewelinkDevice, Action<DeviceViewModel> onPowerSwitch)
{
EwelinkDevice = ewelinkDevice;

SwitchDevicePower = new MvxCommand(() => onPowerSwitch(this), () => IsDeviceOnline);
}

public Device EwelinkDevice
{
get => ewelinkDevice;
set
{
SetProperty(ref ewelinkDevice, value);
RaiseAllPropertiesChanged();
}
}

public bool IsDeviceOnline => EwelinkDevice.Online ?? false;

public bool IsDevicePowerOn => EwelinkDevice.Paramaters.Switch == SwitchState.On;

public IMvxCommand SwitchDevicePower { get; }
}
}
```

VMBase.cs

```
using System.Threading.Tasks;
using MvvmCross;
using MvvmCross.Commands;
using MvvmCross.Navigation;
using MvvmCross.ViewModels;

namespace FactoryIoT.VM
{
public class VMBase : MvxViewModel
{
protected VMBase()
{
CloseCommand = new MvxAsyncCommand(OnCloseAsync);
}

public MvxAsyncCommand CloseCommand { get; }

protected IMvxNavigationService NavigationService => Mvx.IoCProvider.Resolve<IMvxNavigationService>();

protected bool IsCloseAlreadyExecuted { get; set; }


public sealed override void ViewDestroy(bool viewFinishing = true)
{
base.ViewDestroy(viewFinishing);
OnViewDestroy(viewFinishing);
}

protected virtual async Task PerformCloseAsync()
{
await NavigationService.Close(this);
}

protected virtual void OnViewDestroy(bool viewFinishing)
{
}

private async Task OnCloseAsync()
{
if (IsCloseAlreadyExecuted)
{
return;
}

await PerformCloseAsync();

IsCloseAlreadyExecuted = true;
```

```
}
}
}
```