КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

До захисту допущено:
Завідувач кафедри ІСТ
Олександр КУЧАНСЬКИЙ
«__» _____ 2022 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

спеціальності 126 «Інформаційні системи та технології» 126 Інтернет речей
освітньої програми «Програмні технології інтернет речей»

(код і назва спеціальності)

на тему: IoT система моніторингу наявності товарів для вендингових апаратів

Виконала студентка IV курсу, групи ІР-41
Анастасія ДЕМЧУК
(прізвище ім'я по-батькові) (підпис)

Керівник к.ф.-м.н., асистент Роман ПОНОМАРЕНКО
(посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Консультант нормо контроль к.т.н., доцент Ростислав ЛІСНЕВСЬКИЙ
(посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент к.т.н., Анна КАРАПЕТЯН
(посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Засвідчую, що у поянювальна записка немає запозичень з праць інших авторів без відповідних посилань.
Здобувач освіти

Київ – 2022 року

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Освітньо-кваліфікаційний рівень **бакалавр**
спеціальності 126 «Інформаційні системи та технології»
освітньої програми «Програмні технології інтернет речей»

ЗАТВЕРДЖУЮ
Завідувач кафедри
інформаційних систем та технологій
Олекасндр КУЧАНСЬКИЙ
(прізвище та ініціали)
«__» _____ 2022 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

Анастасії ДЕМЧУК

1. Тема проекту (роботи) “IoT система моніторингу наявності товарів для вендингових апаратів”

   керівник проекту (роботи) Роман ПОНОМАРЕНКО, к.ф.-м.н., асистент
   (прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджена протоколом засідання кафедри ІСТ №05/21_22 від 03.12.2021 року

2. Термін здачі студентом закінченого проекту (роботи) 22 червня 2022р.
3. Вихідні дані до проекту (роботи)
   Дані про користувачів IoT системи моніторингу наявності товарів для

вендингових апаратів, дані про модулі та датчики телеметрії для вендингових апаратів, дані про існуючі системи моніторингу.

4. Зміст розрахунково-пояснювальної записки

   Належить оцінити існуючі системи моніторингу наявності товарів для вендингових апаратів, проаналізувати модулі та дачики телеметрії, які використовуються в цих системах і розробити прототип IoT системи, що може моніторити наявність товарів в режимі реального часу.

5. Перелік графічного матеріалу: схема роботи вендингової системи з застосуванням IoT рішення, схема бази даних для IoT системи моніторингу наявності товарів, інтерфейс програмного продукту для моніторингу наявності товарів та аналізу отриманих даних, інтерфейс мобільного застосунку для купівлі товарів.

6. Календарний план виконання роботи:

№ п/п	**Найменування етапів дипломного проекту**	**Строк виконання етапів проекту (початок-кінець)**	**Примітки**
1	*Затвердження теми роботи*	20.11.2021 – 28.12.2021	*виконано*
2	*Вивчення та аналіз завдання*	01.01.2022 – 11.02.2022	*виконано*
3	*Розробка архітектури та загальної структури систем*	12.02.2022 - 17.02.2022	*виконано*
4	*Розробка структур окремих підсистем*	18.02.2022 – 02.04.2022	*виконано*
5	*Технічна реалізація програмного забезпечення*	03.04.2022 – 20.05.2022	*виконано*
6	*Оформлення пояснювальної записки*	21.05.2022 – 05.06.2022	*виконано*
7	*Попередній захист кваліфікаційної роботи*	07.06.2022	*виконано*
8	*Перевірка на плагіат*	14.06.2022	*виконано*
9	*Нормоконтроль*	15.06.2022 – 20.06.2022	*виконано*
10	*Рецензування кваліфікаційної роботи бакалавра і представлення роботи на кафедру в друкованому вигляді*	15.06.2022 – 20.06.2022	*виконано*

№ п/п	Найменування етапів дипломного проекту	Строк виконання етапів проекту (початок-кінець)	Примітки
11	*Захист кваліфікаційної роботи бакалавра*	23.06.2022	*виконано*

Дата видачі завдання «__» _____ 2022 р.

Здобувач освіти на освітньому рівні «бакалавр» 4-го курсу групи ІР-41 _____ (підпис) А.Б. ДЕМЧУК (ініціали, прізвище)

Керівник роботи _____ (підпис) Р.М. ПОНОМАРЕНКО (ініціали, прізвище)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра «IoT система моніторингу наявності товарів для вендингових апаратів» складається зі вступу, основної частини, що містить 3 розділи, висновків і списку літератури та джерел. Загальний обсяг роботи – 84 сторінки. Робота містить 63 рисунки, 1 таблицю. Список використаних джерел включає 34 джерела.

Об'єкт дослідження – IoT система моніторингу наявності товарів для вендингових апаратів.

Мета роботи –дослідження, проектування та програмна реалізації частини IoT системи моніторингу наявності товарів для вендингової системи.

Предмет дослідження – процес проектування IoT системи, керування програмними інтерфейсами та функціонування ІС моніторингу наявності та купівлі товарів в вендингових апаратах.

Методи дослідження: пошук інформації в мережі Інтернет, прослуховування лекцій, аналіз спеціалізованих матеріалів і літератури, практична робота зі створення бази даних, проектування програмного інтерфейсу, розробки мобільного додатку, вивчення наявних технологій та роботи досліджуваної системи в цілому.

Практичне значення роботи полягає в розробці IoT системи, яка включає програмний продукт для моніторингу наявності товарів та аналізу отриманих даних, а також мобільний застосунок для купівлі товарів. Результати здійснених у дипломній роботі досліджень можуть бути використанні при подальшому проведенні науково-дослідницьких робіт.

*Напрямки подальших досліджень***:** доопрацювання алгоритму обміну даними для великої кількості автоматів в різних регіонах, покращення інтерфейсу і продуктивності мобільних додатків, імплементація в реальний вендинговий апарат.

*Ключові слова***:** вендинг, база даних, фізична модель, телеметрія, роздрібна торгівля, моніторинг, GUI, протокол MQTT, dex, безконтактний зчитувач, хмарні сервери, мережа, IoT, автоматизована система, датчики телеметрії.

ABSTRACT

The qualification work "IoT system for monitoring the availability of goods in vending machines" of bachelor consists of an introduction, the main part, which contains 3 sections, conclusions and a list of references and sources. The total volume of work is 84 pages. The work contains 63 figures, 1 table. The list of sources used includes 34 sources.

The object of research is the IoT system for monitoring the availability of goods for vending machines.

The purpose of the work is research, design and software implementation of the IoT system for monitoring the availability of goods for the vending system.

The subject of research is the process of IoT system design, software interface management and operation of IS monitoring of availability and purchase of goods in vending machines.

Research methods: searching for information on the Internet, listening to lectures, analysis of specialized materials and literature, practical work on creating a database, designing a software interface, developing a mobile application, studying existing technologies and the system as a whole.

The practical significance of the work is to develop an IoT system, which includes a software product for monitoring the availability of goods and analysis of the data obtained, as well as a mobile application for the purchase of goods. The results of research conducted in the thesis can be used in further research.

Areas of further research: refinement of the data exchange algorithm for a large number of machines in different regions, improving the interface and performance of mobile applications, implementation in a real vending machine.

Keywords: vending, database, physical model, telemetry, retail, monitoring, GUI, MQTT protocol, dex, contactless reader, cloud servers, network, IoT, automated system, telemetry sensors.

ЗМІСТ

ВСТУП

Мережу Інтернет можна розглядати, як складну екосистему, яка стрімко розширює свої кордони і знаходить застосування щоразу в нових напрямках. А такою технологією, як IoT, та всім, що пов'язано з Інтернет речами, на сьогодні вже нікого не здивуєш. Цей, ще донедавна фантастичний винахід, став новою реальністю, і з плином часу ще й спробує зазіхнути на статус тривіальної буденності для всього людства.

Доцільність даного дослідження полягає в тому, що застосування технології IoT забезпечить передачу даних про кількість товарів в режимі реального часу через мережу Інтернет, що дозволить безперервно моніторити вендингову систему без втручання людини в даний процес. Дана технологія може бути корисною для вендингових машин, що знаходяться у важкодоступних місцях або потребують ретельнішого контролю над процесом продажу продукції. Крім цього, додаткове використання однієї з Cloud Platform дозволить проводити якісну автоматизовану аналітику, а встановлена система телеметрії надасть можливість безконтактної та безготівкової оплати послуги.

Завдання роботи: 1) спроектувати і змоделювати базу даних для ІС моніторингу вендингових апаратів з застосуванням технології IoT для передачі даних про кількість товарів в режимі реального часу через мережу Інтернет та без втручання людини у даний процес; 2) розробка програмного інтерфейсу для зручного моніторингу наявності товарів кінцевими користувачами системи; 3) розробка мобільного додатку для підключення до вендингового апарату і купівлі товарів; 4) налаштування MQTT брокера та розробка скрипту для передачі даних з вендингового автомату в БД.

Результати: спроектовано прототип бази даних для інформаційної системи моніторингу наявності товарів для вендингових апаратів, розроблено програмний продукт, який представляє собою графічний інтерфейс для моніторингу наявності

товарів та аналізу даних, створено мобільний застосунок для прямої роботи з вендинговим апаратом, а саме купівлі товарів.

Головна ціль цієї дипломної роботи – це створення прототипу IoT системи для моніторингу наявності товарів у вендингових автоматах в режимі реального часу. Застосування технології IoT забезпечить передачу даних про кількість товарів в режимі реального часу через мережу Інтернет, що дозволить безперервно моніторити вендингову систему без втручання людини в даний процес. Дана технологія може бути корисною для вендингових машин, що знаходяться у важкодоступних місцях, або потребують ретельнішого контролю над продажами продукції. Крім цього, додаткове використання однієї з Cloud Platform дозволить проводити якісну автоматизовану аналітику, а встановлена система телеметрії надасть можливість проведення безконтактної та безготівкової оплати послуги.

РОЗДІЛ 1 ОГЛЯД ВИМОГ ДО ВЕНДИНГОВИХ СИСТЕМ. ПОСТАНОВКА ЗАДАЧІ

Даний розділ містить інформацію про сферу застосування технології IoT у вендингових апаратах, визначення необхідності створення зручної системи для покупців та обслуговуючого персоналу, піднімає питання про моніторинг наявності товарів в режимі реального часу, актуальність розробки індивідуального програмного інтерфейсу з метою взаємодії системи для вендингової компанії з її клієнтами.

1.1 Вендингові системи, їх аналіз та опис сфери застосування

IoT технології – це не лише про розумні будинки та теплиці, IoT – це будь-який об’єкт, що містить сенсори, програмне забезпечення або інші технології для підключення та обміну даними з іншими об’єктами за допомогою мережі Інтернет [1]. Діапазон таких об’єктів величезний – від звичайних побутових предметів до складних промислових інструментів. За даними компанії Oracle, сьогодні вже існує понад 7 мільярдів пристроїв, підключених до IoT, і ця цифра продовжує зростати. Так, до 2025 року експерти прогнозують, що кількість IoT-пристроїв перетне позначку в 22 мільярди [2].

Серед пристроїв, підключених до IoT, є торгові автомати та вендингові системи в цілому. Останні роки вони стрімко зростають у популярності, пропонуючи споживачам зручність та даючи можливість продавцям роздрібної продукції продавати свої товари цілодобово і невеликими обсягами, при цьому мінімізуючи витрати на, власне, процес продажу. А такі інноваційні рішення як IoT полегшують і забезпечують постачальникам централізоване управління різними типами машин з будь-якої локації та покращують роздрібну торгівлю завдяки комфортному і впевненому моніторингу. Вендингові апарати, під'єднані до мережі Інтернет з можливістю моніторингу та управління, носять назву «розумний (смарт) вендинг»

(англ. Smart Vending) [3]. Таке словосполучення часто можна зустріти в літературі, пов’язаній із специфікою даної теми.

1.2 Застосування технології IoT для моніторингу наявності товарів у вендингових апаратів

Розумні торгові автомати, як правило, підключені до Інтернету та пропонують оновлення та моніторинг у режимі реального часу.

Пристрої, що дозволяють підключити вендинговий апарат до мережі Інтернет, носять назву датчиків телеметрії. Телеметрія, в свою чергу, представляє собою збір даних, отриманих у ході вимірювань або іншим способом у віддалених точках (локаціях), та їх автоматична передача до приймального обладнання (телекомунікації) для моніторингу, аналізу тощо [4].

Отже, застосування IoT передбачає ідентифікацію вендингового апарату в мережі Інтернет, що дає можливість приймати різні типи платежів, включаючи біометричну ідентифікацію: споживачі можуть здійснювати платежі, скануючи QR-код своїм мобільним пристроєм, а застосування більш комплексних телеметричних систем дозволяє також проводити безконтактну оплату за допомогою технології PayPass. Крім того, оператори можуть отримувати переваги від аналізу даних користувачів та транзакцій, що дозволяє їм постійно вдосконалювати свої операції, асортимент продуктів та відображати вміст вендингових апаратів в режимі реального часу.

Нижче зображена базова мережева архітектура розумних торгових автоматів (рис. 1.1) [5].

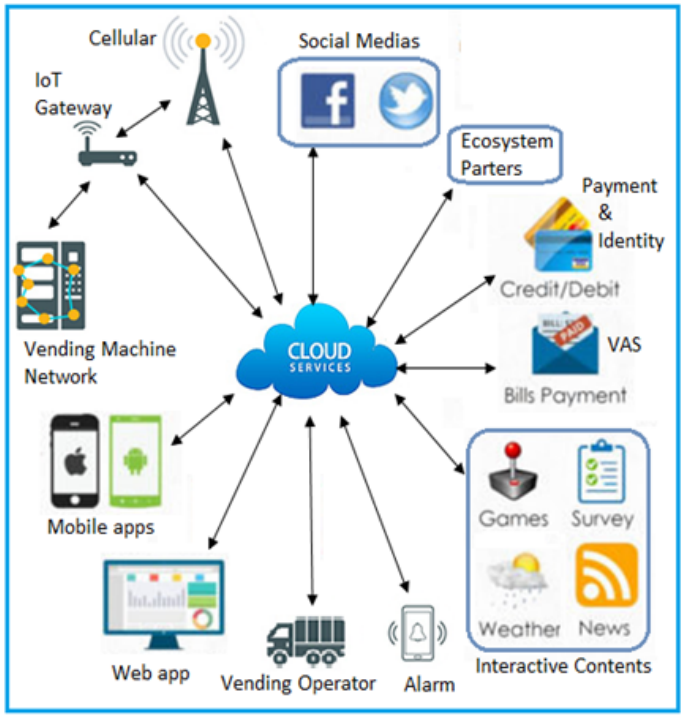


Рисунок 1.1 Архітектура розумних торгових автоматів

На основі вищенаведеного, можна виділити такі переваги застосування технології IoT для вендингу:

- Звітування про кількість проданих товарів у режимі реального часу, здійснення контролю запасів продукції.
- Надання послуг покупцям і отримання переваг для продавців цілодобово та без вихідних.
- Відстежування витрат автоматично, використовуючи мобільний додаток або веб-програму для веб-звітності на основі використання хмарних технологій.
- Можливість приймати різні типи платежів, один з яких – безконтактна оплата.
- Зниження часу простою торгових автоматів, завдяки доступу до всієї інформації про працюючі та непрацюючі машини, генерації попереджень та сповіщень при несправності машини в реальному часі.

Впроваджуючи IoT-систему, також варто взяти до уваги проблеми, з якими можна стикнутись в процесі та після реалізації даного рішення:

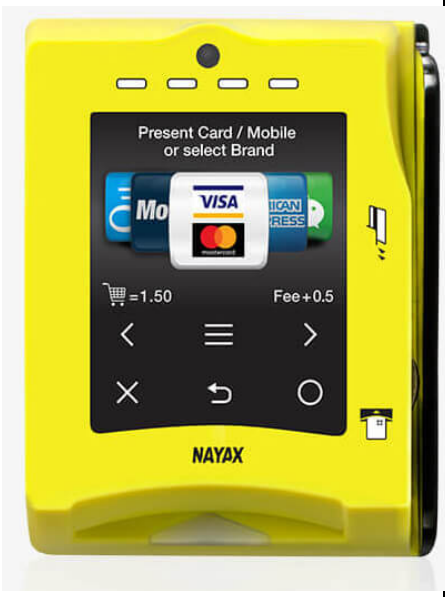
- Необхідна додаткова реалізація захисту смарт вендингу, оскільки ідентифікація торгового апарату в мережі Інтернет дає можливість зловмисникам отримати доступ до банківських операцій споживачів та може призвести до крадіжки товарів.
- Розумні торгові автомати вимагають більших початкових інвестицій. Крім того, вони можуть мати більші експлуатаційні витрати та витрати на обслуговування.
- Потреба в наявності кваліфікованого персоналу для ремонту розумних торгових автоматів.
- Управління декількома рішеннями різних типів машин в різних географічних регіонах може бути доволі складним завданням за відсутності відповідних спеціалістів або без використання хмарних платформ.

1.3 Огляд обладнання вендингового апарату

1.3.1 Датчики телеметрії та сенсори всередині автомату

Датчики телеметрії являють собою повноцінні контролери з можливістю підключення до мережі Інтернет та з підтримкою таких протоколів канального рівня для вендингу, як DEX, DDCMP та інтерфейс Rs232. Оскільки існуюча система базується на протоколі DEX, то оптимально буде обрати датчики з підтримкою саме цього протоколу. Крім цього, варто зазначити, що датчик телеметрії дає можливість приймати різні типи платежів, включаючи біометричну ідентифікацію: споживачі можуть здійснювати платежі, скануючи QR-код своїм мобільним пристроєм, а більш комплексні телеметричні системи дозволять безконтактну оплату за допомогою технології PayPass. Для реалізації простого моніторингу, на вибір телеметра це не впливає, однак клієнтам варто враховувати дану різницю як ще одну можливість зробити свої автомати більш комфортними в користуванні.

Таблиця 1.1- Приклади датчиків телеметрії

Зображення	Назва пристрою	Характеристики
1	2	3
	Зчитувач безготівкових платіжних карток – VPOS TOUCH [6]	Захист сертифікаціями: EMVCo (рівні 1 та 2), CE, FCC, RoHS, MasterCard Робоча температура: від 20 ° C до + 55 ° C Вологість: від 25% до 95% (без конденсації) Стійкість до пилу та води: IP 55 Горючість: UL-94 V0 Дисплей: кольоровий сенсорний РК-екран 2,4 дюйма, 320 x 240 пікселів + 4 світлодіоди + 6 сенсорних кнопок Технологія ARM – 2xCortex, 32 біт, 180/216 МГц Багатомовна підтримка – текстові та голосові звукові інструкції на РК-дисплеї Безпека: підтримка DES, 3DES, AES, RSA, ECC, SHA1 алгоритми, генератор випадкових чисел

Таблиця 1.1- Приклади датчиків телеметрії (продовження)

Зображення	Назва пристрою	Характеристики
1	2	3
		Блок живлення: 12-42 В постійного струму, 12-24 В змінного струму; 5 Вт, рекомендований зовнішній блок живлення: 24 В постійного струму / 2 ампер. Внутрішній датчик удару / нахилу, антивандальне скло Dragontrail Зовнішній інтерфейс – Ethernet, 2 X RS232 (рівень RS232 або TTL через SW), Зовнішній зчитувач, 1 X датчик відкритих дверей, 1 X датчик температури Імпульсний вхід / вихід 6 імпульсних інтерфейсів вводу / виводу + 1 заборона введення Вихід в мережу: 3G / 4G / Ethernet / NFC Машинні протоколи: MDB / DEX / DDCMP / Pulse / VCCS / ccTalk / JVMA

Таблиця 1.1- Приклади датчиків телеметрії (продовження)

Зображення	Назва пристрою	Характеристики
1	2	3
	AMIT 3.0 Nayax – інтелектуальний телеметричний пристрій, який забезпечує можливість моніторингу машин у режимі реального часу за допомогою постійно оновлюваної інформації про всі аспекти роботи [7]	Робоча температура: 30°C - 50°C Вологість: - 5% - 95% (без конденсації) Зовнішній зв'язок – GSM / GPRS / CDMA / Ethernet / Wi-Fi/ MDB / Pulse / VCCS / ccTalk Інтерфейс DEX / DDCMP / RS232 Технологія ARM – Cortex, 32-розрядна, 168 МГц Система шифрування з високим рівнем безпеки: AES / DES / RSA / SSL Джерело живлення: вхід 10-42 вольт змінного струму, 10-30 вольт постійного струму, зовнішній вихід 8 вольт постійного струму Міжнародний рівень захисту: IP50 Споживана потужність 0,3 Вт (в режимі очікування) – 1 Вт (в Інтернеті) 1,9 ”/ 48,3 ммС

Таблиця 1.1- Приклади датчиків телеметрії (продовження)

Зображення	Назва пристрою	Характеристики
1	2	3
	ePort G10 Pulse Kit – кард-рідер, телеметр, імпульсний інтерфейс [8]	Виробник Cantaloupe, Inc. Доступність – США Платіжний процесор Cantaloupe, Inc. Методи оплати: EMV безконтактні, NFC / RFID, Magstripe, мобільний гаманець Розмір висоти телеметра: 5,40 ″ Ширина: 4,21 ″ Глибина: 1,54 ″ Доступ до мережі: Verizon 4G / LTE або AT&T 4G / LTE з антеною Підтримувані апаратним забезпеченням з'єднання: Serial, DEX, MDB, Pulse

Таблиця 1.1- Приклади датчиків телеметрії (продовження)

Зображення	Назва пристрою	Характеристики
1	2	3
	Безконтактний зчитувач ePort Engage [9]	Виробник Cantaloupe, Inc. Доступність – США Платіжний процесор Cantaloupe, Inc. Методи оплати: EMV contactless and chip-card, NFC/RFID, Magstripe, Mobile Wallet Доступ до мережі: Verizon 4G / LTE або AT&T 4G / LTE з антеною Підтримувані апаратним забезпеченням з'єднання: Serial, DEX, MDB, Pulse

Таблиця 1.1- Приклади датчиків телеметрії (продовження)

Зображення	Назва пристрою	Характеристики
1	2	3
	Seed Telemeter дозволяє без нагляду операторам керувати віддаленими пристроями, відстежуючи операційні дані, ефективність продажів та повідомлення машин, використовуючи бездротове підключення через мережу Інтернет [10].	Виробник Cantaloupe, Inc. Доступність – США, Канада, Австралія, Нова Зеландія, Мексика Платіжний процесор Cantaloupe, Inc. Доступ до мережі: AT&T 4G/LTE with 3G fallback Підтримувані апаратним забезпеченням з'єднання: Serial, DEX, MDB, Pulse

Проаналізувавши ринок телеметричних датчиків та оцінивши використання різного виду протоколів передачі даних між кінцевими пристроями IoT, можна прийти до висновку, що вендинговий автомат може бути ідентифіковано в мережі, як за допомогою датчика Seed Telemeter, так і з використанням будь-якого з безконтактних зчитувачів, що підтримують технологію NFC та/або PayPass.

Seed Telemeter – найдешевший з можливих варіантів, що найкраще підходить для малого бізнесу. А для більш вимогливих клієнтів можна встановити зчитувач

безготівкових платіжних карток – VPOS TOUCH, який відповідатиме усім можливим вимогам. Порівняльна характеристика аналогових систем для моніторингу вендингових автоматів представлена нижче (рис 1.2).

Порівняльна характеристика аналогових систем

Назва системи	Критерії порівняння				
	Призначення	Можливості	Характеристики	Недоліки	Витрати
IoT система з використанням зчитувача безготівкових платіжних карток	Контроль вендингових автоматів в режимі реального часу, безконтактна оплата	• Забезпечення постійної технічної підтримки вендингових автоматів • Моніторинг запасів товарів в режимі реального часу • Безконтактна оплата за допомогою технології PayPass • Аналіз транзакцій, активних годин користування і тд.	• Методи оплати: EMV contactless and chip-card, NFC/RFID, Mobile Wallet • Доступ до мережі: 4G / LTE або AT&T 4G / LTE з антеною, 3G / Ethernet • Робоча температура: -30° C - 50° C • Система шифрування з високим рівнем безпеки: AES / DES / RSA / SSL • Споживана потужність: 0,3 Вт (в режимі очікування) - 1 Вт (в Інтернеті) • Машинні протоколи: MDB / DEX / DDCMP / Pulse / VCCS / ccTalk / JVMA	• Більші експлуатаційні витрати та витрати на обслуговування. • Потреба в наявності кваліфікованого персоналу для ремонту розумних торгових автоматів • Управління декількома рішеннями різних типів машин в різних географічних регіонах може бути доволі складним завданням за відсутності відповідних спеціалістів	400 - 300$ / за один пристрій на один вендинговий автомат
IoT система з використанням Seed Telemeter	Контроль вендингових автоматів в режимі реального часу	• Моніторинг запасів продукції в режимі реального часу • Цілодобова технічна підтримка • Можливість оплати за допомогою сканування QR або штрих-коду	• Методи оплати: QR-code, bar code • Доступ до мережі: AT&T 4G / LTE, 3G • Робоча температура: -20° C - 40° C • Споживана потужність: 0,1 Вт (в режимі очікування) - 0,5 Вт (в Інтернеті) • Машинні протоколи: MDB / DEX / DDCMP / Pulse	• Необхідна додаткова реалізація захисту смарт вендингу, оскільки ідентифікація торгового апарату в мережі Інтернет дає можливість зловмисникам отримати доступ до банківських операцій споживачів та може призвести до крадіжки предметів • Відсутність оплати за допомогою PayPass	150 - 100$ / за один пристрій на один вендинговий автомат
Існуюча система без IoT	Періодичне оновлення даних про запаси продукції	• Моніторинг запасів продукції з періодичним оновленням інформації	• Методи оплати: готівка • Робоча температура: 0° C - 50° C • термін служби літієвої батареї на 1000 мА (залежить від використання) 2-7 років • Машинні протоколи: MDB / DEX / DDCMP / Pulse • ARM Cortex-M4F processor	• Відсутність можливості безготівкової оплати • Моніторинг і оноволення даних вимагає людських ресурсів	-

Рисунок 1.2 Порівняльна характеристика аналогових систем для моніторингу вендингових автоматів

В результаті дослідження було визначено IoT систему з використанням датчику телеметрії Seed Telemeter найоптимальнішою для проектування інформаційної

системи «Моніторингу наявності товарів для вендингових апаратів» в рамках даної дипломної роботи.

1.3.2 Протоколи для зв'язку між кінцевими IoT пристроями та користувачами системи

Для обраного контролеру потрібно завантажити програму, яка міститиме необхідні команди отримання та відправки даних. Згідно багаторівневої мережевої моделі OSI (Open Systems Interconnection model) (рис. 1.3) [11], для передачі даних на прикладному рівні можна використовувати загальновідомий протокол передачі гіпертексту HTTP, з яким легко взаємодіяти за допомогою принципу «запит-відповідь». В момент, коли клієнту потрібно отримати дані із сервера, він надсилає повідомлення запиту на сервер, а сервер, в свою чергу, повертає клієнтові відповідь. Тобто клієнт ініціює запит даних, а сервер відповідає на нього.

Рисунок 1.3 Модель OSI

Однак програми IOT з обмеженими ресурсами можуть стикнутись з особливостями функціонування протоколу HTTP. Перш за все, як уже зазначалося, HTTP призначений для одностороннього зв'язку, тобто передача даних може бути ініційована лише з одного боку. Сервер не може самостійно ініціювати зв'язок з клієнтським пристроєм. По-друге, більшості додатків IoT потрібно працювати з

обмеженою пропускною здатністю мережі. Самі пристрої IoT мають обмежену кількість вбудованої оперативної пам'яті та ПЗУ, обмежені обчислювальні ресурси. Вони також повинні передавати дані в режимі реального часу, що можливо лише з невеликими фрагментами коду, який дозволить забезпечити надійність, незважаючи на обмежену пропускну здатність та вбудовані ресурси. Такі специфічні вимоги середовища IoT спонукають спеціалістів до пошуку кращих рішень, ніж використання протоколів HTTP та інших розповсюджених протоколів комп'ютерних мереж.

Одним з таких рішень є використання протоколу прикладного рівня – MQTT (Message Queue Telemetry Transport). MQTT – це протокол або набір правил, який використовується для зв'язку пристрою до пристрою (D2D) та для віддаленого обміну даними від одного клієнта до іншого [12]. Він підключає пристрій до пристрою для обміну даними через глобальну мережу з допомогою проміжної компоненти, яка має назву MQTT брокер (сервер). MQTT розроблений таким чином, що і клієнт, і сервер можуть спілкуватися між собою. З протоколом MQTT будь-який з кінцевих пристроїв може ініціювати передачу даних. Пристрій, який запитує дані, називається абонентом у протоколі MQTT, а пристрій, що надає дані, називається видавцем. Це полегшений протокол публікації / підписки, в якому кожен окремий клієнт MQTT підключений до посередника MQTT для спілкування з іншими клієнтами MQTT. Пристрій IOT можна налаштувати як видавця, абонента або як обох з них за допомогою брокера MQTT (рис. 1.4).

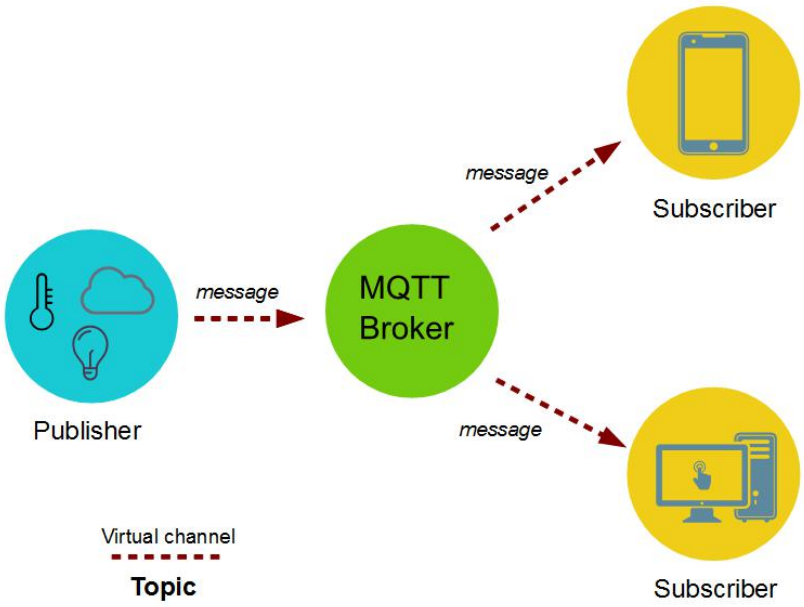


Рисунок 1.4 Архітектура протоколу MQTT

Загалом вищенаведений протокол орієнтований на такі повідомлення, в яких клієнти надсилають та отримують дані у формі повідомлення з певною темою. Будь-хто – видавець або підписник – спілкується через мережу лише за допомогою брокера. Отже, видавець може надавати дані брокеру, коли це потрібно, а підписник так само може запитувати від нього дані. Передача даних, хоча і ініціюється будь-яким з пристроїв, передається в режимі реального часу. Отже, за допомогою брокера може існувати двосторонній зв’язок через мережу [13].

Крім того, MQTT можна успішно застосовувати також для пристроїв з обмеженими ресурсами та пристроїв з обмеженою потужністю. Протокол призначений для надійного передавання невеликих розмірів коду в режимі реального часу, незважаючи на обмежену пропускну здатність мережі. Програми IoT можливі без MQTT, але MQTT забезпечує архітектурні переваги перед типовими протоколами рівня прикладних комп'ютерних мереж, коли пристрої IoT з обмеженими ресурсами повинні взаємодіяти двонаправлено через обмежену пропускну здатність.

Причиною того, що MQTT називається легким протоколом, є те, що накладні витрати на пакет цього протоколу надзвичайно малі. Найменший пакет має лише 2

байти накладних витрат. Ці невеликі транспортні накладні витрати допомагають зменшити мережевий трафік [13].

Враховуючи вище описані переваги, можна стверджувати, що використання протоколу MQTT є найбільш доцільним для функціонування системи з IoT пристроями.

1.4 Огляд готових IoT-рішень

1.4.1 Vagabond

Vagabond – міжнародна компанія, яка займається виготовленням вендингових машин для Північної і Центральної Америки [14]. IoT-мережа Vagabond дозволяє операторам торгових автоматів віддалено побачити, коли запасів продукції мало, а також те, як швидко продаються товари, з подальшою організацією послуги їх поповнення при необхідності. Датчики забезпечують передачу даних в режимі реального часу, а також можуть виявляти несправності, що дозволяє відремонтувати зламані машини якомога швидше. Vagabond використовує API Google Maps Javascript для веб-додатку (рис. 1.5), який відображає свої торгові автомати у вигляді шпильок на карті, для появи інформації про машину потрібно просто натиснути на неї, отримані дані також включають запаси і рівень готівки в автоматі [15].

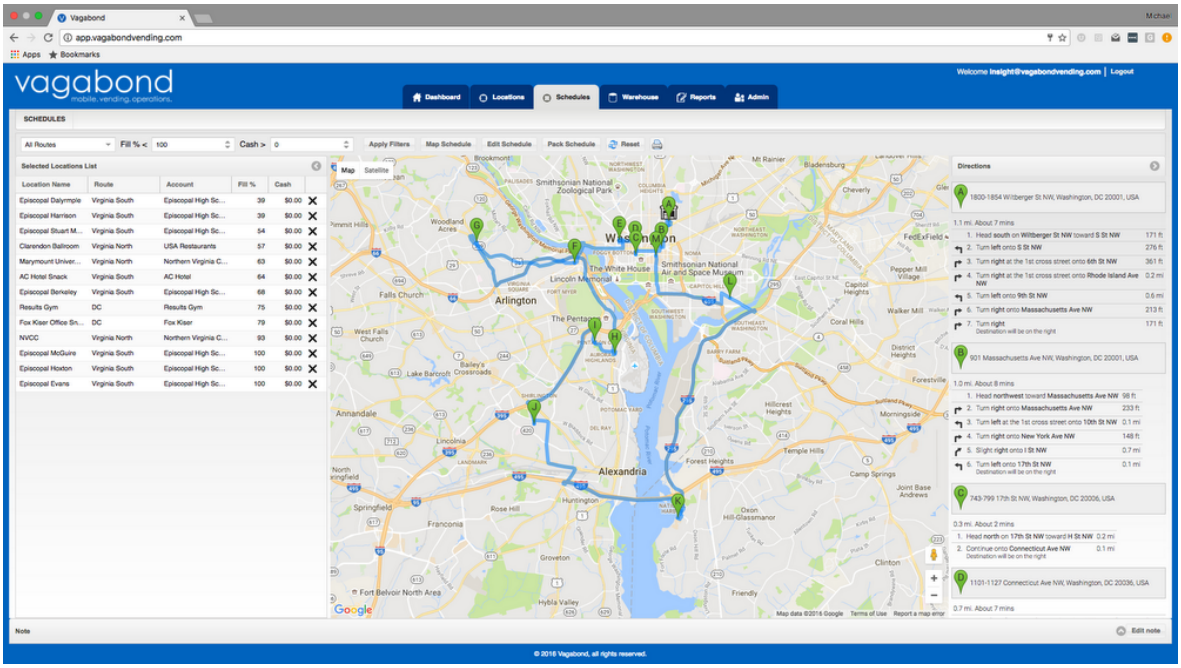


Рисунок 1.5 Зображення веб-додатку для вендингу компанії Vagabond

Картуючи дані автоматів, Vagabond допомагає операторам будувати більш ефективні маршрути для обслуговування машин. Таким чином вони максимізують прибуток. Оскільки оператори знають, яка саме продукція найкраще продається та на яких машинах, вони мають можливість адаптувати свої запаси відповідно до попиту споживачів, що дозволяє збільшити дохід на машину в середньому на 15 відсотків, про що зазначив Майкл Ловетт, співзасновник та генеральний директор Vagabond [15]. Створення ефективних маршрутів зменшує кількість надурочних робіт, що виплачуються водіям, а також витрати на обслуговування бензину та вантажних автомобілів. Клієнти Vagabond також спостерігають зменшення витрат на 15 відсотків. За словами Ловетта [15], оператори можуть збільшити норму прибутку з 2,5 до 20 відсотків.

Основа системи вендингу Vagabond для управління апаратами – хмарна платформа Google Cloud IoT Core [16].

1.4.2 Gimme Vending

Система Gimme Vending – це готове рішення для автоматизованого управління замовленнями, інвентаризацією, рахунками-фактурами на доставку, прокладання

маршрутів для водіїв і т.п., тобто для усього, що раніше розраховувалось вручну, за допомогою «олівця і ручки». Саме тому продукція Gimme Vending Inc широко розповсюджена в США, а її клієнти – невеликі компанії, що займаються роздрібною торгівлею з використанням вендингових машин, та потребують автоматизації вище наведених процесів.

Вендингова система компанії Gimme Inc є повноцінним програмно-апаратним комплексом, який, за допомогою трьох окремих програмних застосунків та одного апаратного ключа, регулює роботу торгових автоматів та забезпечує моніторинг і контроль запасів продукції для отримання можливості вчасного поновлення асортименту.

Принцип роботи програмно-апаратного комплексу Gimme Vending [17]:

1. На вендинговий автомат встановлюється апаратний ключ Gimme Key Pro, який відіграє роль контролера та містить всередині мікропрограму з підтримкою технології Bluetooth Low Energy для предачі DEX файлів. Розширення .dex використовується для формату робочої книги або електронної таблиці, розробленої корпорацією Майкрософт [18]. Збір даних з машини передбачає або підключення мобільного пристрою до порту DEX машини, або використання оптичної альтернативи, наприклад, інфрачервоної технології [19].
2. Клієнт Gimme Vending встановлює програмне забезпечення Gimme VMS (рис. 1.6), реєструє свої автомати з ключем Gimme Key Pro в системі, розміщує всі заплановані точки продажу в єдиному режимі перегляду. Формує базу продукції і т. д.

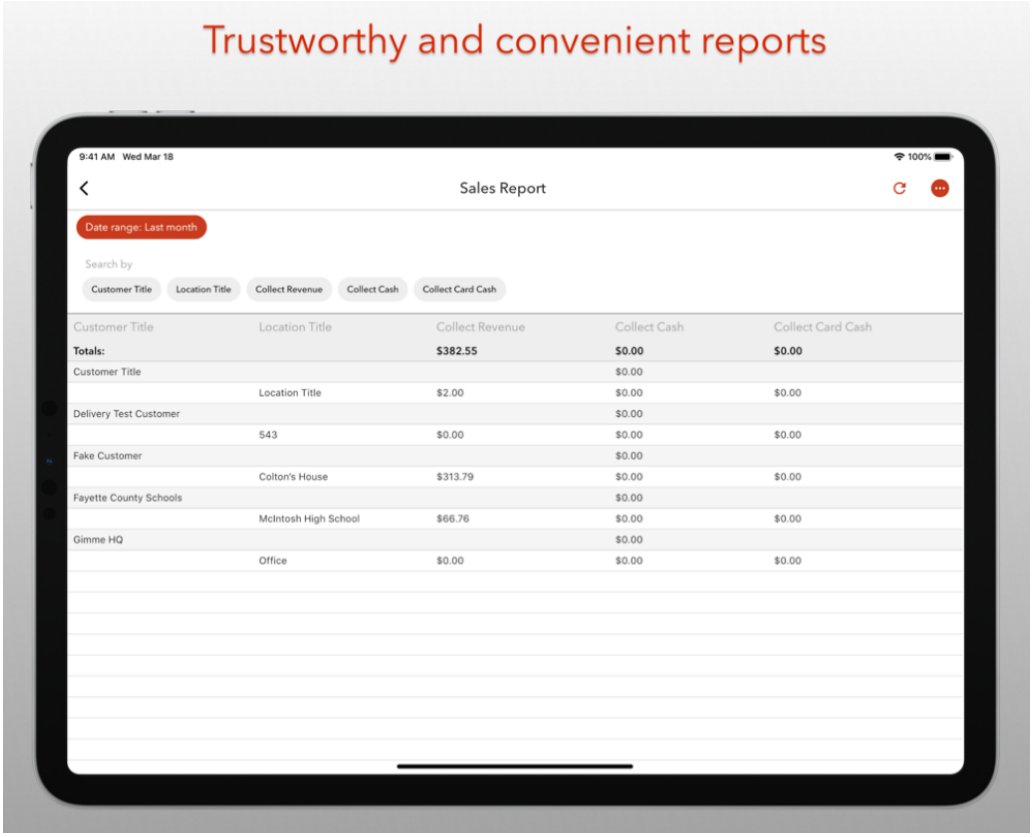


Рисунок 1.6 Надійні та зручні звіти в додатку Gimme VMS

3. Кадри, що працюють на складі і відповідають за наповнення вендингових автоматів товарами, а також водії, що займаються розвезенням продукції, встановлюють програмний додаток Gimme Field, де відображаються усі машини, зареєстровані клієнтом-компанією в Gimme VMS [20].
4. Водії, відповідальні за доставку, також встановлюють програмний додаток Gimme DEX, для взаємодії з автоматами.
5. При виконанні чергового поновлення запасів продукції, водій під’єднується до вендингового автомату за допомогою Gimme DEX, проходить ідентифікацію та отримує можливість завантажити DEX файл на свій мобільний пристрій. Після цього, за допомогою камери власного мобільного пристрою, він сканує вигляд та асортимент товарів. Далі ця інформація відправляється на сервер і буде відображена в додатку Gimme Field і Gimme VMS.

6. Персонал, що відповідальний за управління товарами та грошовий облік, аналізують отримані дані в додатку Gimme VMS і, згідно цієї інформації, формують замовлення на поновлення продукції та маршрут водіїв.
7. Персонал на складі отримує точні цифри щодо кількості необхідних товарів та приїзду водіїв для їх забору в додатку Gimme Field.

1.4.3 ProstoPay

ProstoPay – українська компанія, що обслуговує вендингові апарати та кавомашини [23]. Організація володіє нативним програмним забезпеченням під Android та iOS, які дозволяють проводити безготівкові розрахунки у торгових автоматах. Реалізація IoT-системи існує за рахунок датчиків телеметрії та власної розробки API, що отримує дані від встановлених телеметрів за допомогою протоколу MQTT. MQTT – це стандартний протокол обміну повідомленнями OASIS для Інтернету речей (IoT). Він розроблений як надзвичайно легкий протокол обміну повідомленнями для публікації / підписки, що ідеально підходить для підключення віддалених пристроїв з невеликим розміром коду та мінімальною пропускною здатністю мережі. Дане рішення є дешевим і може слугувати найкращим вибором для малого бізнесу і невеликих вендингових компаній, що мають на меті реалізувати технологію «смарт вендингу».

1.5 Аналіз актуальності теми та постановка задачі

Дослідивши та проаналізувавши ринок систем, які пропонують готові рішення, постало питання розробки IoT системи, яка дозволить проводити безперервний моніторинг вендингових апаратів без втручання людини в цей процес, за допомогою зручного програмного забезпечення для подальшого використання персоналом обслуговування даних машин.

Для визначення потреб підприємств, які займаються вендинговим бізнесом, проаналізовано найбільш поширені види робіт, що є необхідними до виконання:

- моніторинг наявності товарів, їх своєчасне поповнення необхідною продукцією;
- логістика водіїв;
- поповнення асортименту товарів, залучення ресурсів;
- безконтактна оплата для зручності покупців;
- підтримка апаратних вимог для комфортного користування автоматом (оновлення датчиків телеметрії на апаратах, тощо);
- ведення аудиту, контролю над операційною діяльністю обслуговуючого персоналу;
- просування та рекламування продукту;
- аналіз отриманої інформації про продажі та маркетинг;
- клієнтська підтримка;
- підтримка віддаленого управління кожним автоматом, який компанія надає в користування замовникам.

На основі зібраної інформації було сформовано список задач, що потребують автоматизації:

- клієнтська підтримка;
- звітування про кількість проданих товарів у режимі реального часу, контроль запасів продукції;

- автоматичне відстежування витрат, використовуючи мобільний додаток або веб-програму з веб-звітності за допомогою хмари;
- можливість приймати різні типи платежів, один з яких – безконтактна оплата;
- зниження часу простою торгових автоматів, завдяки можливості доступу до всієї інформації про працюючі та непрацюючі машини, генерації попереджень та сповіщень при несправності машини в реальному часі;
- формування списку необхідних матеріалів;
- передача документів у електронному вигляді.

Можна зробити висновок, що зручне програмне забезпечення для покупців та обслуговуючого персоналу є одним з найважливіших аспектів для ведення вендингового бізнесу, оскільки забезпечує контроль і аналіз продажів, допомагає в створенні логістики та надає можливість проведення ефективної маркетингової кампанії і залучення покупців, за допомогою мобільного застосунку.

Постановка задачі:

- Проєктування фізичної моделі бази даних «Моніторинг наявності товарів для вендингових апаратів».
- Розробка бази даних з використанням системи управління нереляційною базою даних Firebase, створення та наповнення колекцій даними.
- Конфігурація та налаштування MQTT брокера.
- Розробка скрипту для передачі даних в Firebase Realtime Database.
- Розробка та дизайн мобільних додатків для моніторингу та купівлі товарів засобами фреймворку Flutter.
- Створення інструкції користувача для даних GUI, аналіз БД та генерація звітів.

Як результат, застосування технології IoT забезпечить передачу даних про кількість товарів в режимі реального часу, що дозволить безперервно моніторити вендингову систему без втручання людини в даний процес.

1.6 Висновки до розділу

У даному розділі обґрунтовано актуальність обраної тематики, проаналізовано існуючі готові рішення та впроваджені аналоги, а також проведений аналіз ринку вендингових апаратів. У розділі поставлено задачі, які необхідно виконати задля успішної побудови нової, ефективної і зручної в користуванні IoT-системи моніторингу наявності товарів для вендингових апаратів.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ІОТ СИСТЕМИ МОНІТОРИНГУ НАЯВНОСТІ ТОВАРІВ ДЛЯ ВЕНДИНГОВИХ АПАРАТІВ

Даний розділ містить інформацію про проектування IoT системи моніторингу наявності товарів для вендингових апаратів, а також зображення та опис цієї системи, з метою подальшої розробки бази даних, скрипту і програмних інтерфейсів для керування обраною системою в цілому.

2.1 Алгоритм роботи системи

Вендингову сферу можна умовно розділити на дві частини – надавачі послуг (компанія, вендингові автомати, обслуговуючий персонал, постачальники) та клієнти (покупці або користувачі автоматів). Інтернет речей створює віртуальний міст для комунікації цих частин без участі посередників.

До надавачів послуг можна віднести наступні компоненти: вендинговий автомат з певною категорією товарів, що містить датчик телеметрії, який призначений для виконання скрипту з прийому і передачі даних про оплату в БД, та програмне забезпечення для моніторингу продукції обслуговуючим персоналом – водіями, логістами, маркетологами тощо. До клієнтської частини відноситься мобільний застосунок, що надає можливість обирати товари, здійснювати безконтактну оплату. IoT частина – це MQTT брокер, який керує передачею даних про оплату і надсилає їх від покупця до вендингового автомату. Незалежний компонент – це база даних, яка містить інформацію про вендингові апарати, продукцію, оплати, користувачів тощо.

Принцип роботи прототипу програмно-апаратного комплексу системи, розробленого в дипломній роботі, такий:

1. На вендинговий автомат встановлюється датчик телеметрії, який відіграє роль контролера з підтримкою технології Wi-Fi та містить всередині мікропрограму для передачі даних. Збір і відправка даних з машини здійснюється за допомогою Python скрипту, який слухає конкретний топік, а саме унікальний ID даного автомату, і при отриманні, відправляє їх у хмарне сховище.
2. Cloud Platform зберігає інформацію про продукцію, постачальників тощо, а Cloud Realtime Database – зберігає дані про кількість продукції в режимі реального часу. Компанія реєструє свої автомати на хмарній платформі, розміщує всі заплановані точки продажу в єдиному режимі перегляду, формує базу продукції і тому подібне.
3. Для купівлі товару, користувач, за допомогою мобільного застосунку Pay&Drink, сканує QR-код, нанесений на торговий автомат, що містить інформацію необхідну для підключення до нього (в даному випадку це унікальний ID автомату). Після сканування QR-коду, додаток відправляє запит на хмарну платформу, з метою перевірки на існування такого автомату в даній вендинговій мережі. При отриманні підтвердження, користувач перенаправляється на сторінку з інформацією про продукцію для подальшого вибору та оплати. Після здійснення оплати, додаток публікує повідомлення про купівлю товару, з використанням MQTT протоколу, на топік унікального ID автомату.
4. MQTT брокер приймає повідомлення, відправлені з мобільних застосунків та зберігає їх, в той час як автомати-підписники слухають свої унікальні ідентифікатори, і при появі повідомлення відправляють їх у хмарне сховище.
5. Кадри, що працюють на складі і відповідають за наповнення вендингових автоматів товарами, а також водії, що займаються розвезенням продукції, встановлюють програмний додаток Monitor drinkers, де відображаються усі

машини, зареєстровані клієнтом-компанією, а також наявна там продукція та її кількість в режимі реального часу.

6. При виконанні чергового поновлення запасів продукції, водій проходить ідентифікацію та отримує можливість оновити інформацію про товари, після його поповнення. Далі ця інформація відправляється в хмарне середовище і буде відображена в додатку Monitor drinkers.
7. Персонал, що відповідальний за управління товарами та грошовий облік, аналізує отримані дані в додатку Monitor drinkers і, згідно цієї інформації, формує замовлення на поновлення продукції та маршрут водіїв.

Отже, на основі описаного вище принципу роботи, існуюча система реалізує проведення моніторингу, а також технічну підтримку, в режимі реального часу.

2.2 Проєктування програмної архітектури IoT системи

Для створення прототипу описаної вище системи важливо і необхідно провести декомпозицію проєктування, щоб послідовно реалізувати кожен з компонентів. Отож, систему можна розділити на такі складові:

- Cloud Platform, або хмарна платформа – реалізує необхідні обчислення та збереження даних.
- MQTT брокер – керує підписниками та слухачами, зберігає повідомлення, відправлені на топіки.
- Мікропрограма, що встановлюється на датчик телеметрії – реалізує передачу, отриманих на певний топік, даних до Cloud Platform.
- Мобільний застосунок для моніторингу наявності товарів у вендингових автоматах – використовується надавачем послуг для аналізу, логістики і контролю продукції.
- Мобільний застосунок для купівлі товарів у вендингових автоматах – створений для клієнтів вендингової компанії, вибору продукції, здійснення оплати.

Прототип вендингової системи, з використанням технології IoT та описаних в розділі 1 датчиків і протоколу MQTT (рис. 2.1).

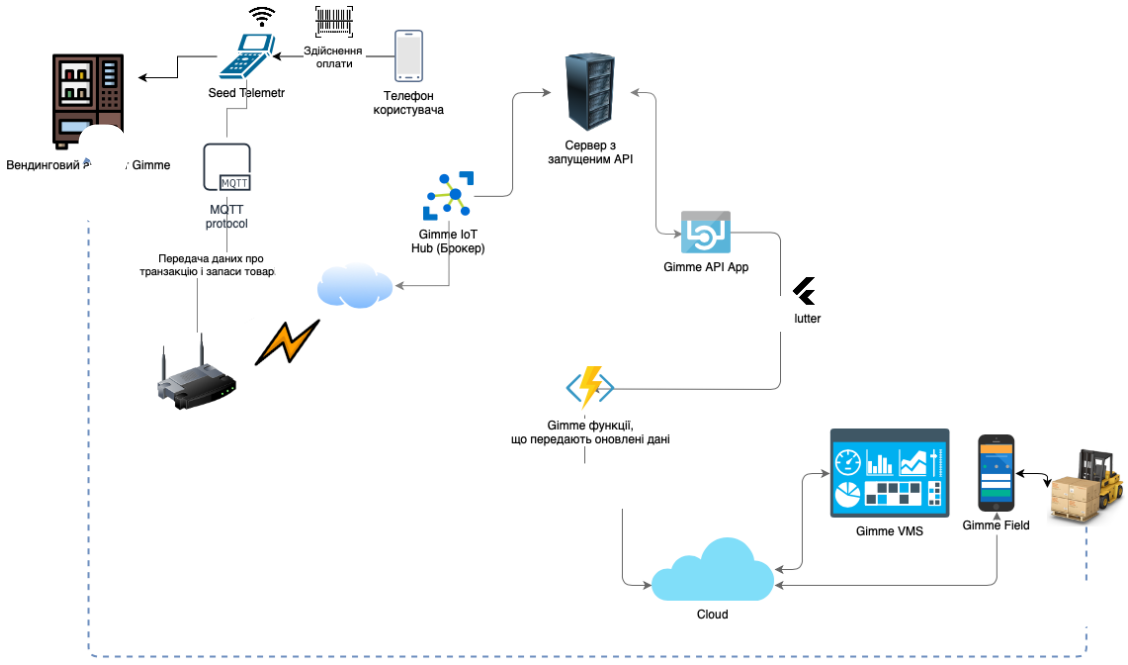


Рисунок 2.1 Схема роботи вендингової системи з застосуванням IoT рішення

Отже, спроектована схема ілюструє роботу IoT системи для вендингового автомату та показує можливість оптимізації моніторингу, технічної підтримки, а також додаткову функцію безготівкової оплати послуг і товарів.

2.3 Проектування дизайну схеми БД

Невід'ємною частиною розробки ІС є етап проектування, що містить логіку бази даних та таблиці, які потрібно створити. Даний етап включає створення схеми бази даних, що передує інтеграції даної схеми в Firebase.

Бази даних Firebase, будь то база даних реального часу або хмара Firestore, є документно-орієнтованими базами даних NoSQL. Це означає, що там немає таблиць, лише колекції документів, де кожен документ є структурою, подібною до JSON, а документи можуть містити інші вбудовані колекції або документи. Теоретично кожен документ у колекції може повністю відрізнятися від усіх інших. Нормалізація даних

не є пріоритетом у NoSQL. Його перевага – це гнучкість. NoSQL є найкращим варіантом для неструктурованих даних, його акцент зосереджений на тому, щоби швидко реагувати на будь-які зміни.

Таким чином, схема бази даних буде використовуватися не тільки для бази даних безпосередньо, але вона буде також використовуватися для проектування об'єктної моделі застосунку (даних і поведінки), незалежно від бази даних. В такому випадку модель може допомогти спроектувати базу даних на основі типу об'єктів, необхідних для задоволення потреб користувачів.

Отже, в даній роботі була розроблена схема бази даних (рис. 2.2) для IoT системи моніторингу наявності товарів у вендингових апаратах.

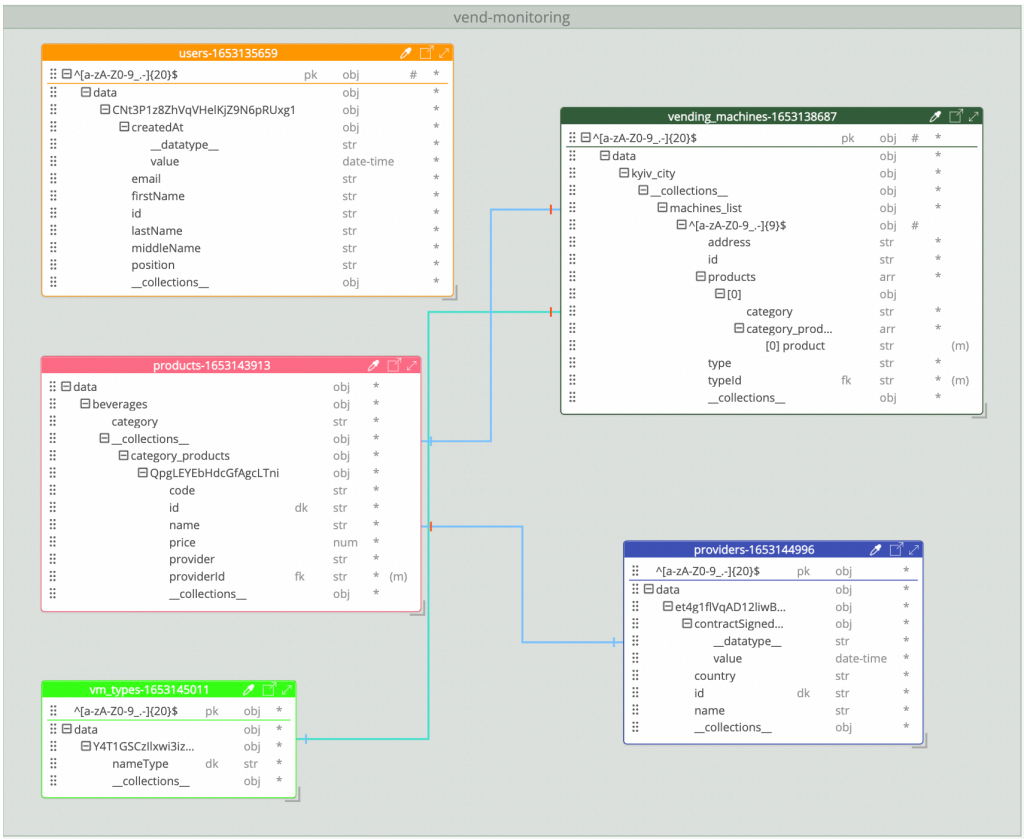


Рисунок 2.2 Схема бази даних для IoT системи моніторингу наявності товарів для вендингових апаратів

Фізична модель бази даних була побудована за допомогою сервісу Hackolade [24], а також була згенерована документація для проектування бази даних в Firestore, яка міститься в додатку А.

2.4 Проєктування програмної архітектури мобільних застосунків

Для проєктування програмного інтерфейсу мобільних застосунків було вирішено використати за основу відомий архітектурний патерн «Чиста архітектура», засновником якої вважається Роберт С. Мартін [25]. Дана архітектура являється багаторівневою та, як правило, складається з 3 основних рівнів (у деяких випадках, кількість рівнів може бути зменшена до двох):

- Рівень даних: включає логіку зв’язку з базою даних, мережевим API тощо.
- Рівень домену: рівень бізнес-логіки.
- Рівень презентації: UI.

Чиста архітектура — це філософія розробки програмного забезпечення, яка розділяє елементи дизайну на рівні кільцевих дій. Одним з головних завдань чистої архітектури є організація програмного коду таким чином, щоб у ньому було інкапсульовано бізнес-логіку, але окремо від логіки графічного інтерфейсу. Основне правило — залежності від коду можуть рухатися лише з зовнішніх рівнів усередину. Код на внутрішніх шарах може не знати про функції на зовнішніх шарах. Змінні, функції та класи (будь-які сутності), які існують у зовнішніх шарах, не можуть бути згадані на більш внутрішніх рівнях. Рекомендується, щоб формати даних також залишалися окремими між рівнями.

Підхід чистої архітектури зображено нижче (рис. 2.3) [26].

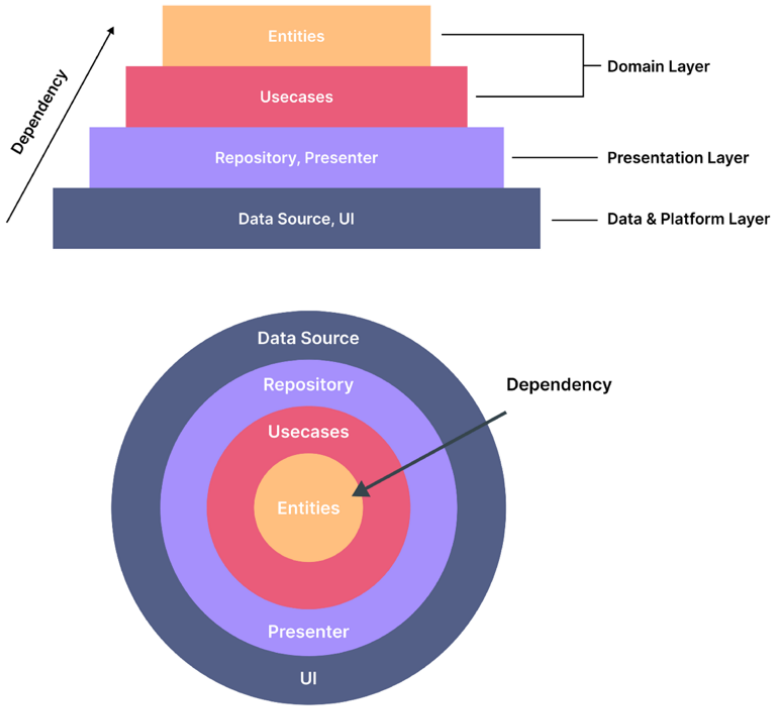


Рисунок 2.3 Clean architecture (чиста архітектура)

Отже, перевагами такого підходу є:

1. Надійність – бізнес-логіка відокремлена від інтерфейсу, кожен рівень працює незалежно і може бути перевірено та досліджено окремо від інших рівнів.

2. Обслуговування та підтримка програми – дозволяє одночасну паралельну розробку між програмістами. Кожен розробник може працювати над окремим шаром самостійно. Наприклад: перший з них може працювати на рівні даних, другий – на домені, а інший може бути залучений до розробки частини інтерфейсу користувача, яка є рівнем презентації.

3. Майбутня масштабованість та логічні зміни – кожну частину програми можна «безболісно» змінити на іншу. Наприклад: якщо будуть отримані деякі зміни в БД, які не передбачають зміни інтерфейсу користувача, то модифікації підлягатиме лише рівень даних.

2.5 Висновки до розділу

Було описано принцип роботи вендингової системи з застосуванням IoT рішення, розроблена архітектура програмного застосунку. Також була проведена декомпозиція задачі проєктування для подальшої програмної реалізації.

У розділі розроблено схему бази даних для IoT системи моніторингу наявності товарів для вендингових апаратів.

РОЗДІЛ 3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Даний розділ містить інформацію про програмну реалізацію компонентів IoT системи моніторингу товарів для вендингових апаратів, опис їхньої реалізації та огляд функціональності.

3.1 Вибір програмного забезпечення для розробки прототипу IoT системи моніторингу

Для збереження та маніпуляції з даними було обрано Cloud Platform – Firebase [27]. Дана платформа використовується для розробки додатків і веб-застосунків, та містить в собі різноманітні сервіси – Firebase Analytics, Firebase Cloud Messaging , Firebase Auth, Realtime Database, Firebase Storage, Firebase Hosting та Functions, – які вирішують низку найважливіших аспектів розробки, таких як аналітика, аутентифікація, з вбудованою підтримкою соціальних логін-провайдерів – Facebook, GitHub, Twitter і Google, збереження даних в режимі реального часу тощо.

Платформа Firebase розроблена компанією Google, та є надійним BaaS-рішенням (Backend as a Service), яке виступає для розробника в якості нереляційної бази даних, сервера, аутентифікації та хостингу, зібраних на одній платформі. А Firebase Realtime Database надає розробникам API, який дає можливість синхронізації додатків і даних між клієнтами та зберігає їх в хмарному сховищі. Наприклад, застосунок підключається до бази даних через WebSocket, що, в свою чергу, дозволяє відображати дані синхронізовано протягом усього сеансу. Переваги платформи Firebase: у високій швидкості роботи; надійній інфраструктурі – відсутність збоїв в роботі, наявності зручної статистики, яка дозволяє отримувати дані про дії користувачів та підтримувати зворотний зв’язок, а також у простоті масштабованості – зростання кількості користувачів не потребують змін у серверному коді.

На основі вище описаних переваг, було обрано Firebase Auth для автентифікації користувачів у спроєктованій IoT системі, Firebase Storage для збереження даних про

користувачів, вендингові автомати, продукцію і постачальників, а Realtime Database – для збереження інформації про наявність продукції у торгових апаратах в режимі реального часу.

Для хостингу MQTT брокера було обрано shiftr.io IoT Cloud Service. Хмарний сервіс shiftr.io дозволяє запускати безкоштовні або платні ізольовані екземпляри shiftr.io у хмарі. Ці екземпляри надають досить велику пропускну здатність як для малих, так і великих проектів. Перевагами даного сервісу є: ізольованість процесу – кожен екземпляр shiftr.io Cloud планується як окремий процес у їхній інфраструктурі, щоб забезпечити низьку затримку та високу пропускну здатність; безпека – усі екземпляри shiftr.io Cloud безпечно використовують протоколи MQTT і HTTP за допомогою TLS; миттєва масштабованість (рис. 3.1).

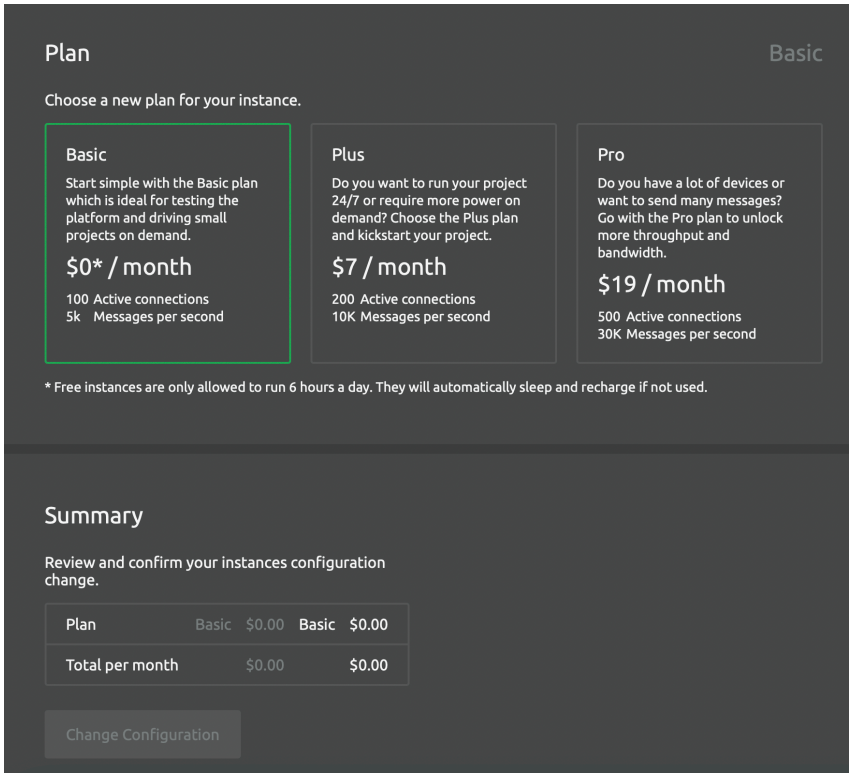


Рисунок 3.1 План користування shiftr.io IoT Cloud Service

За основу мікропрограми або скрипту, для запуску на датчиках телеметрії, а в даній дипломній роботі для створення прототипу IoT системи – локально, було використано існуючу Python програму mqtt2firebase.py [28] та в подальшому

проведено її модифікацію для взаємодії з shiftr.io IoT Cloud Service та відправкою необхідних повідомлень в Firebase Realtime Database.

Мовою розробки програмних застосунків було обрано мову Dart – клієнто-оптимізована мова для створення швидких додатків на будь-якій платформі [29] з використанням фреймворку Flutter – набір програмного забезпечення для інтерфейсу з відкритим кодом, створений компанією Google, що використовується для розробки додатків для Android, iOS, Linux, Mac, Windows, Google Fuchsia та веб-додатків з єдиної кодової бази [29], який повністю підтримує протокол MQTT та має власну бібліотеку – mqtt_client 9.3.1 для роботи з ним [30]. Отже, для централізованого керування та контролю своїх машин за допомогою однієї єдиної платформи, компанії буде достатньо мати спеціалістів, які зможуть реалізувати вище описану технологію. Суттєва перевага фреймворку Flutter – його абсолютна інтеграція з платформою Firebase [31].

Генерацію QR-коду для симуляції з'єднання з вендинговими автоматами було використано сервіс QRCode Monkey, який є одним з найпопулярніших безкоштовних онлайн-генераторів QR-коду. Його переваги – висока роздільна здатність QR-кодів, різноманітні варіанти дизайну, які можна використовувати для комерційних і друкованих цілей [32].

3.2 Розробка бази даних в Firebase Firestore і Realtime Database для ІС моніторингу наявності товарів

Створення проєкту в Firebase відбувається у Firebase Console [33]. Документація для роботи з новим проектом ілюструє кожен крок, і тому початкове налаштування є доволі простим [34]. Основні налаштування зображені нижче (рис 3.2). Щоб використовувати Firebase у мобільних застосунках Apple та Android, потрібно зареєструвати програми у створеному проекті Firebase. Реєстрація програм Monitor drinkers для моніторингу товарів та Pay&Drink для покупців, розроблених для iOS і Android платформ (рис 3.3).

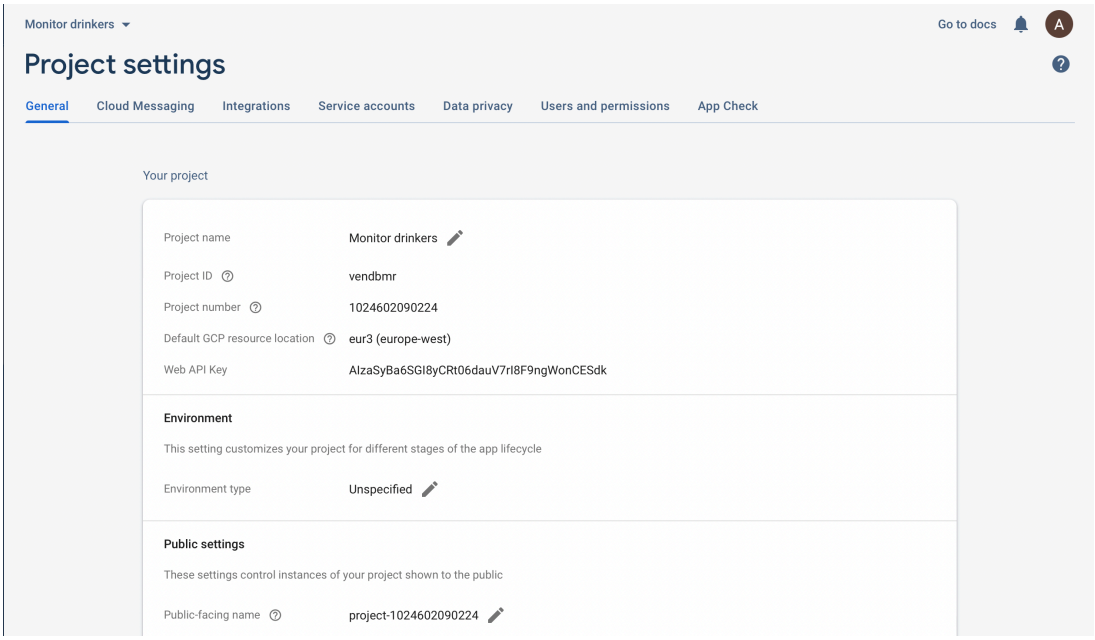


Рисунок 3.2 Основні налаштування проєкту

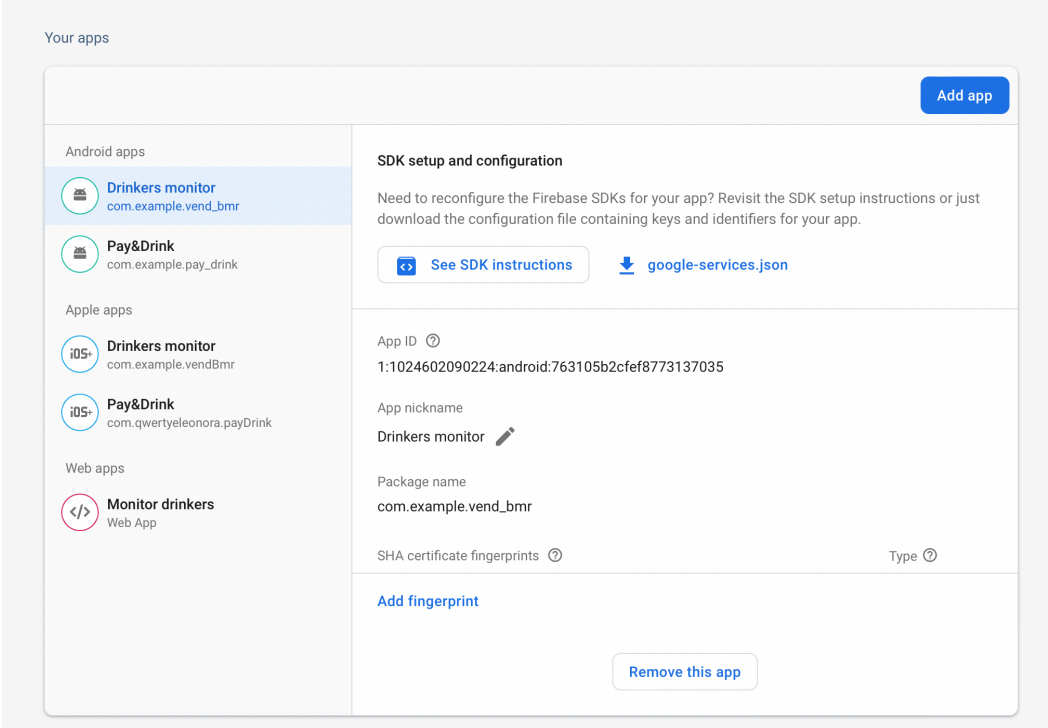


Рисунок 3.3 Список зареєстрованих мобільних застосунків спроєктованої IoT системи

Для взаємодії Python скрипту з Firebase Realtime Database необхідно ініціалізувати пакет SDK за допомогою авторизації, яка поєднує файл облікового

запису служби разом із обліковими даними програми Google за замовчуванням. Проекти Firebase підтримують облікові записи служби Google, які можна використовувати для виклику API сервера Firebase із сервера застосунків або довіреного середовища. Для аутентифікації облікового запису служби та авторизуваці його для доступу до служб Firebase, розробник повинен створити файл приватного ключа у форматі JSON. Щоб створити файл приватного ключа для облікового запису служби: на консолі Firebase необхідно відкрити «Налаштування» > «Облікові записи служб» та згенерувати новий Private Key, а потім варто надійно зберегти файл JSON, що цей ключ містить (рис. 3.4)

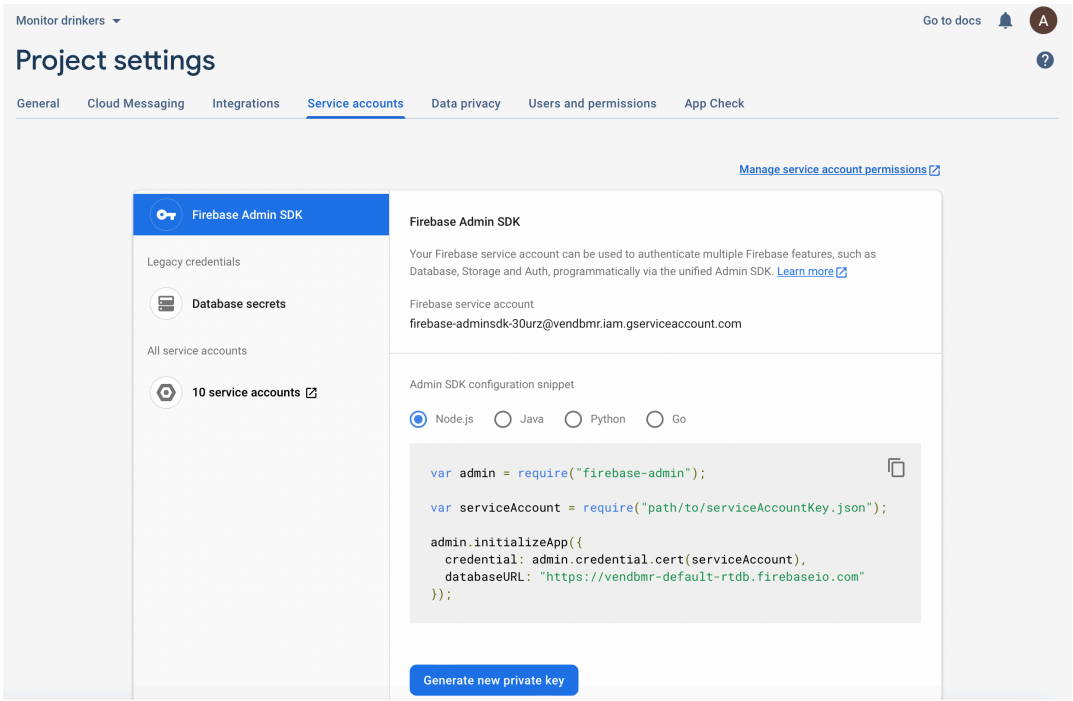


Рисунок 3.4. Сервіс генерації приватного ключа для взаємодії Python скрипту з Firebase Database

Авторизовані користувачі та можливі методи авторизації і реєстрації в програмних застосунках (рис. 3.5 – 3.6).

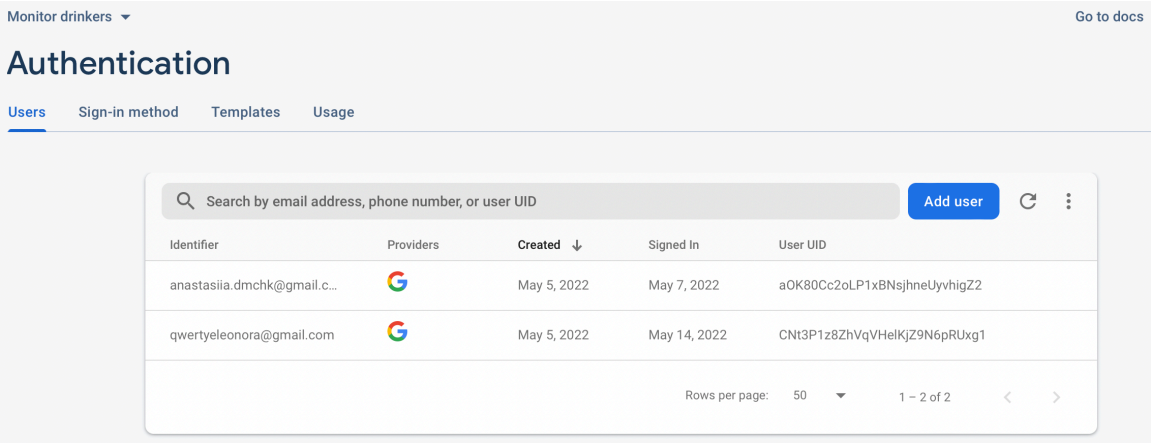


Рисунок 3.5 Авторизовані користувачі

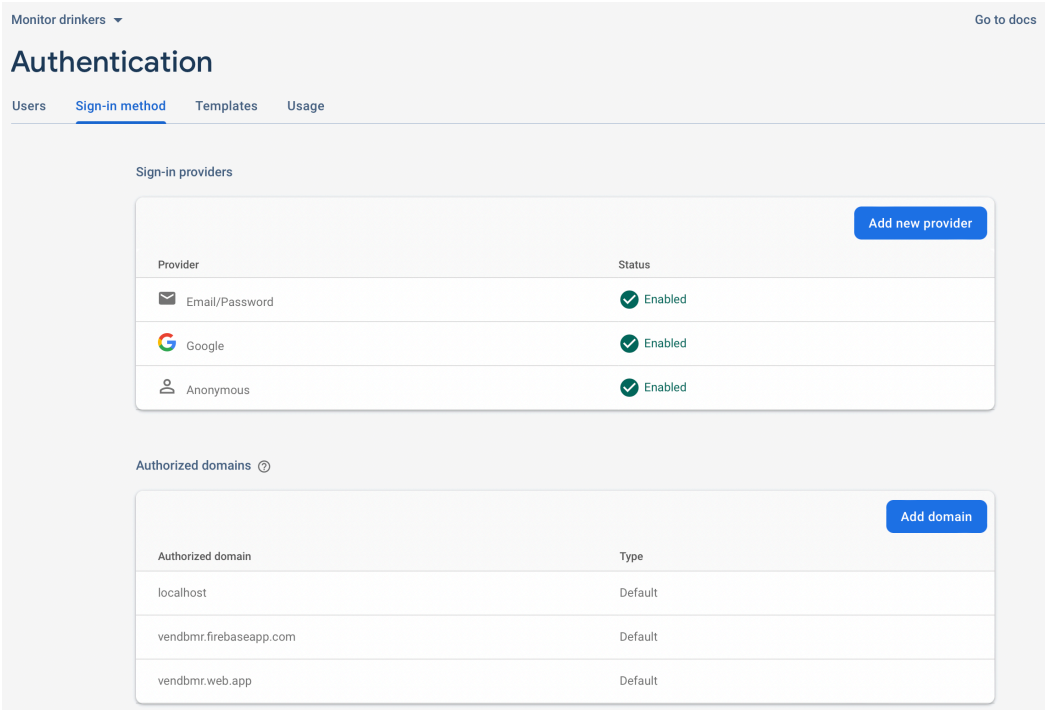


Рисунок 3.6 Дозволені методи реєстрації і авторизації в Firebase Database

Налаштування доступу та шаблону попереджуючого повідомлення при вході (рис. 3.7 – 3.8).

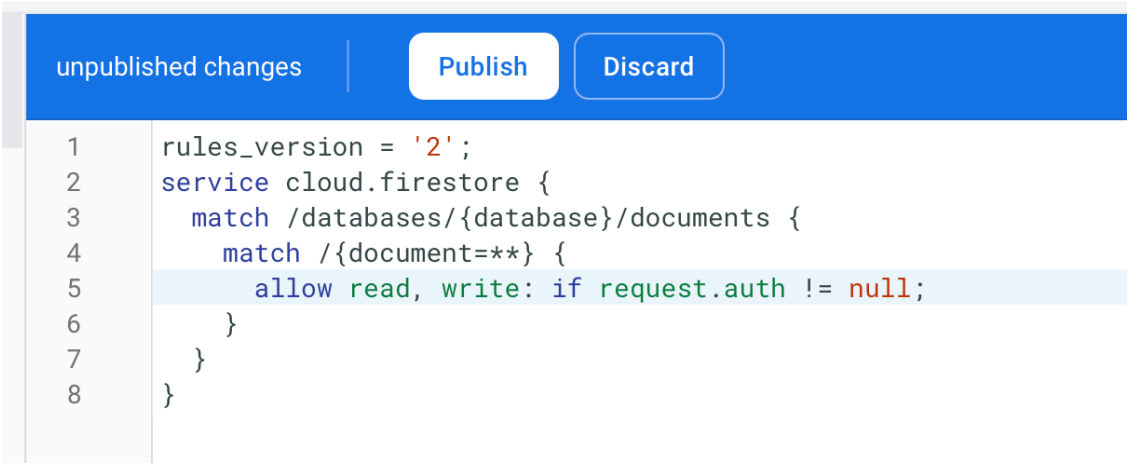


Рисунок 3.7 Дозволені методи запису і читання в Firebase Database

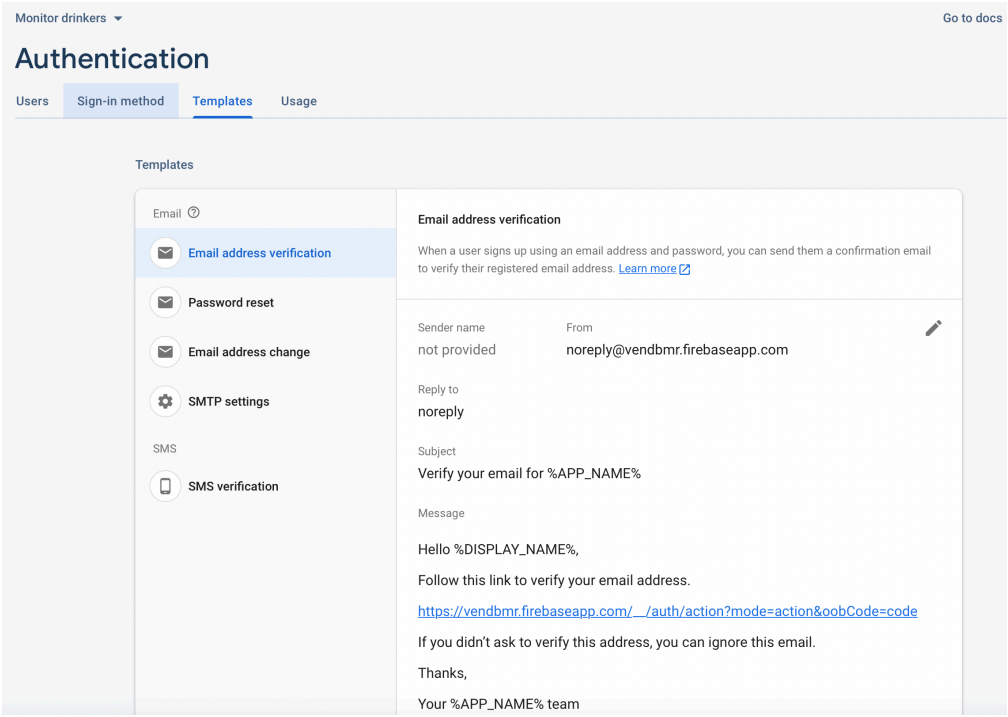


Рисунок 3.8 Шаблон попереджуючого повідомлення при вході в систему через сервіс Google

Заповнені колекції та документи в Cloud Firestore з даними (рис. 3.9 – 3.15).

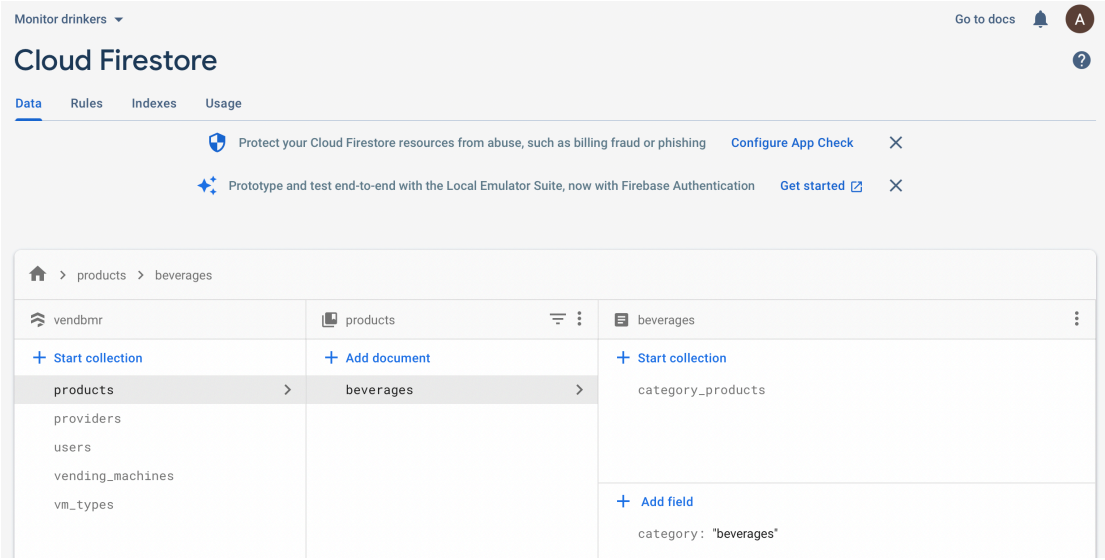


Рисунок 3.9 Колекція продуктів в Firebase Firestore

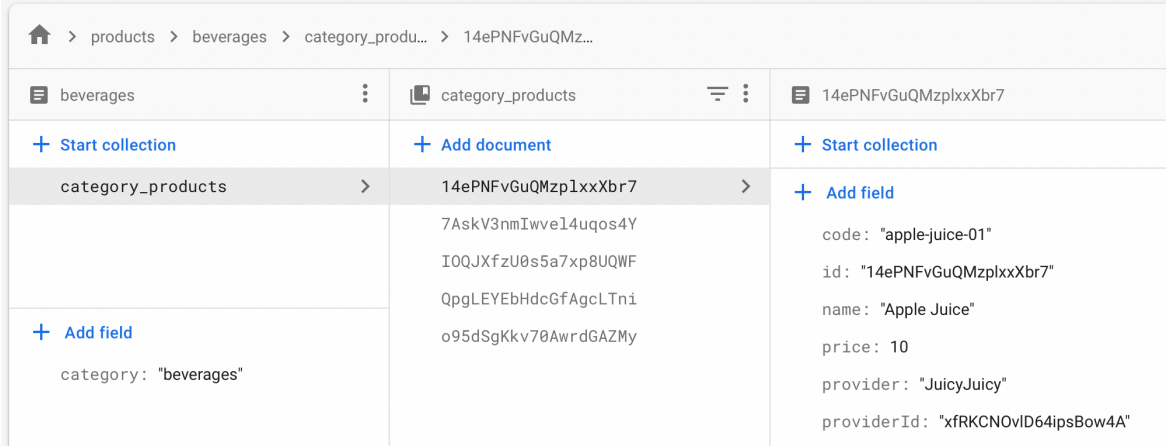


Рисунок 3.10 Колекція категорій продуктів у Firebase Firestore

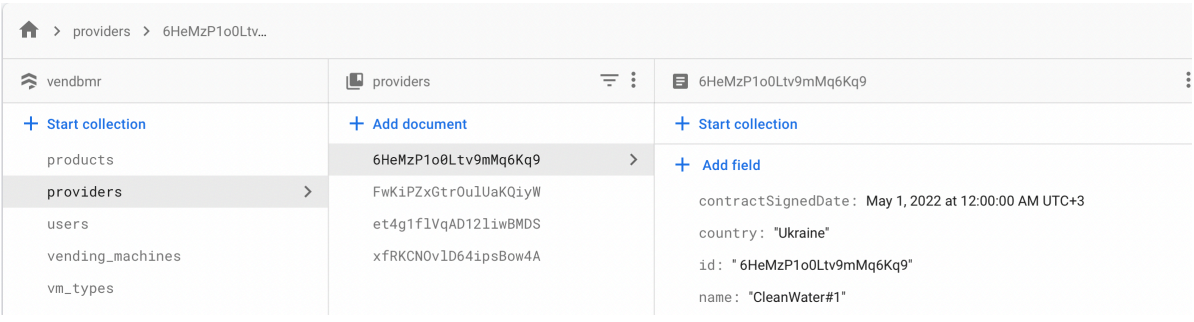


Рисунок 3.11 Колекція постачальників продуктів у Firebase Firestore

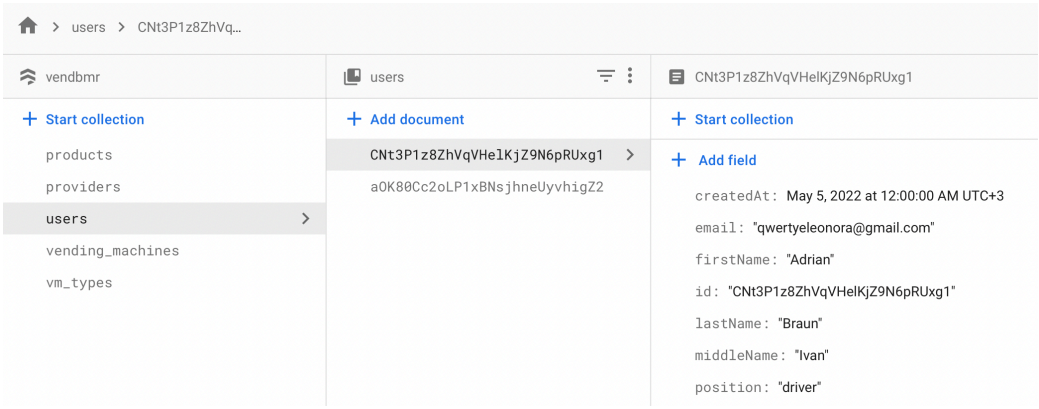


Рисунок 3.12 Колекція користувачів у Firebase Firestore

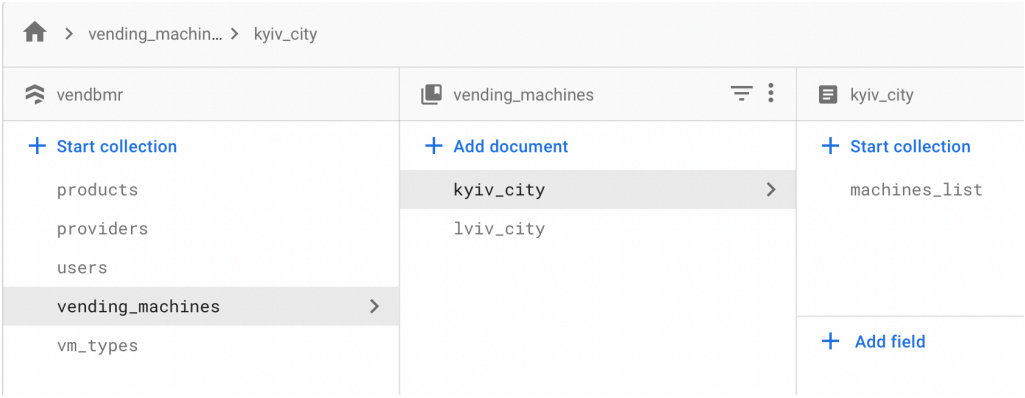


Рисунок 3.13 Колекція вендингових машин у Firebase Firestore

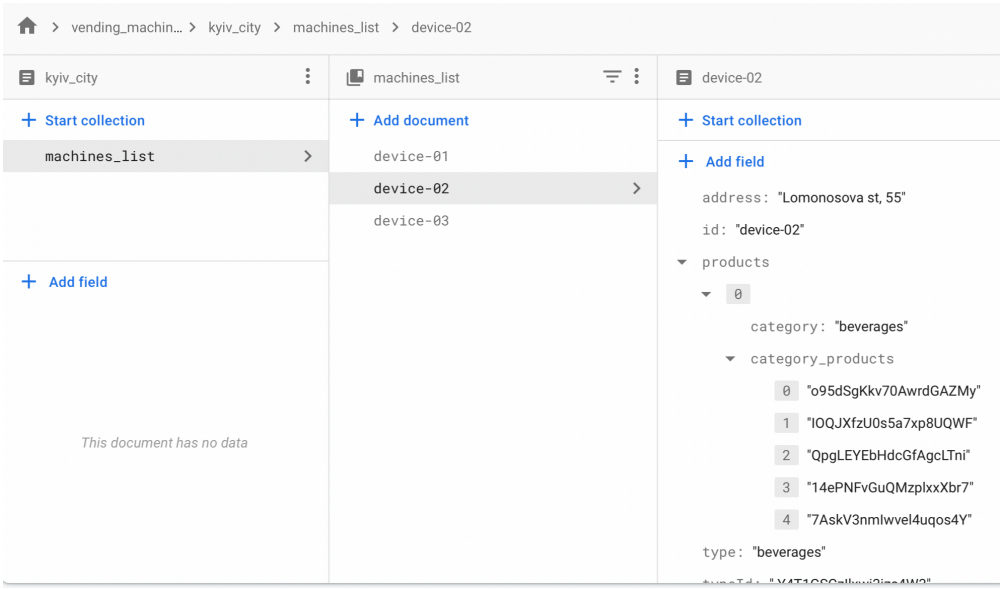


Рисунок 3.14 Колекція вендингових автоматів в одній з локацій у Firebase Firestore

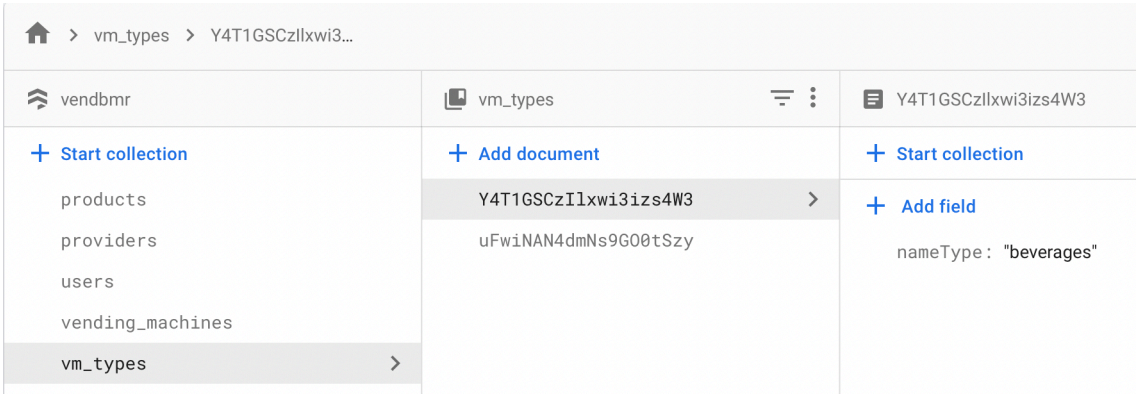


Рисунок 3.15 Колекція типів вендингових машин у Firebase Firestore

Повідомлення, отримані від мобільних застосунків Pay&Drink, з використанням скрипту для передачі їх у Realtime Dabase, та збережених в форматі JSON (рис. 3.16 – 3.18). Аналітика записів (рис. 3.19).

Рисунок 3.16 Список транзакцій для device-01 в Firebase Database

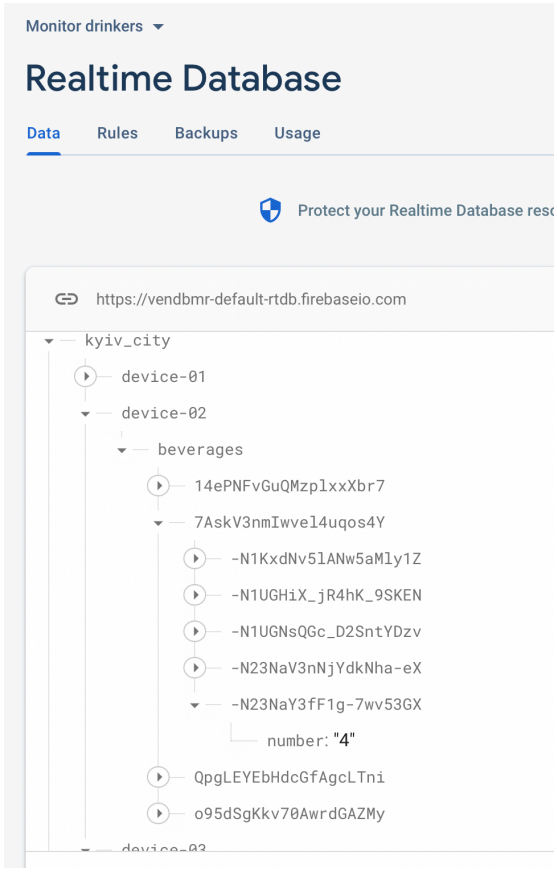


Рисунок 3.17 *Список транзакцій для device-02 в Firebase Database*

Рисунок 3.18 *Список транзакцій для device-03 в Firebase Database*

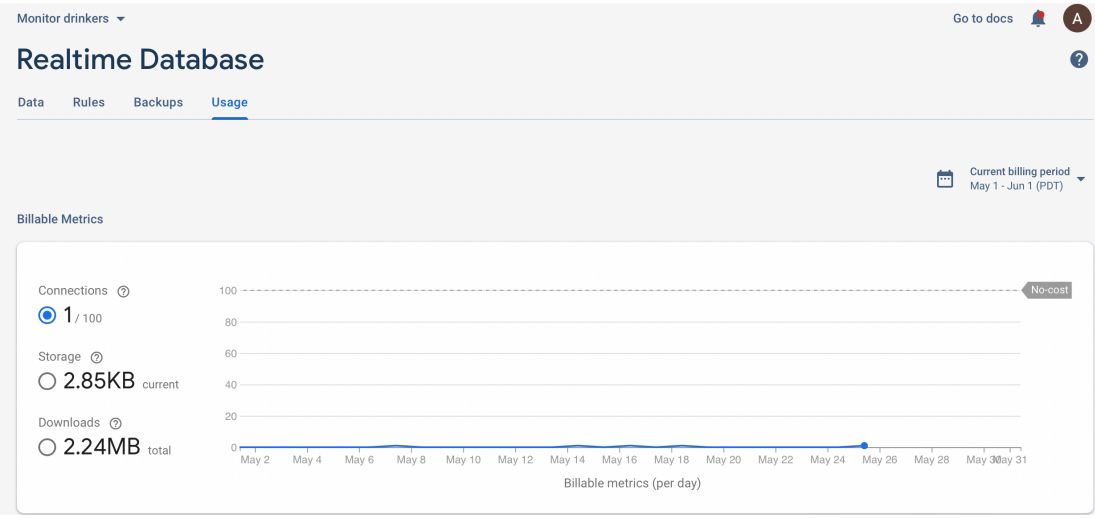


Рисунок 3.19 Аналітика записів у Firebase Database

Отже, розроблена система відповідає вимогам, описаним в розділі 2, і дозволяє перейти до наступного кроку – налаштування MQTT брокера.

3.3 Налаштування і конфігурація MQTT брокера

Було створено акаунт в shiftr.io IoT Cloud Service та зареєстровано екземпляр pay-drink для віддаленого хостингу MQTT брокера. Екземпляр, генерація приватних ключів та програмний інтерфейс shiftr.io IoT Cloud Service (рис. 3.20 – 3.23).

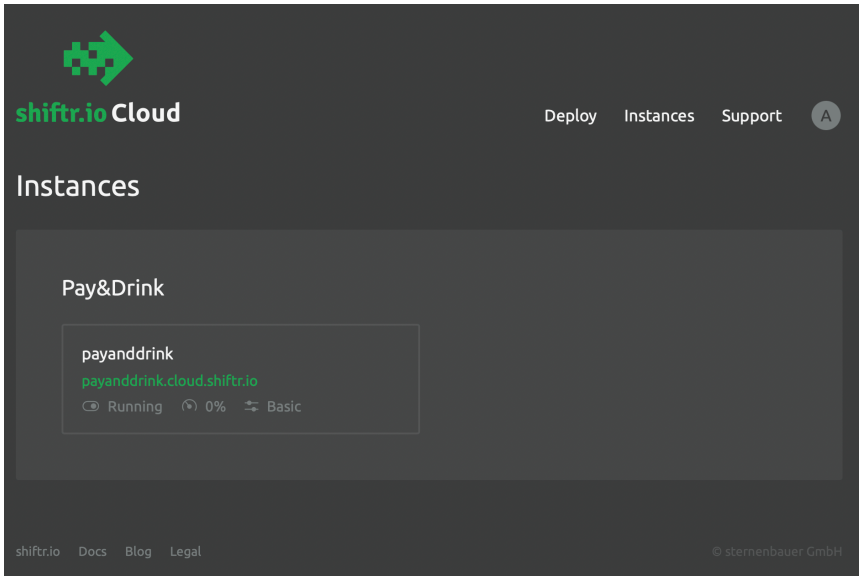


Рисунок 3.20 Створення проекту в shiftr.io IoT Cloud Service

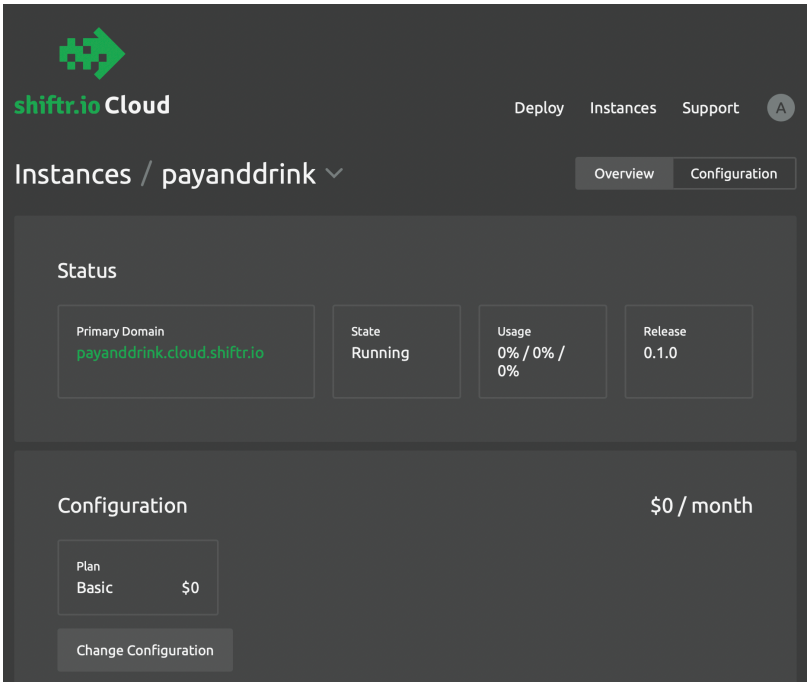


Рисунок 3.21 Створення екземпляру в shiftr.io IoT Cloud Service

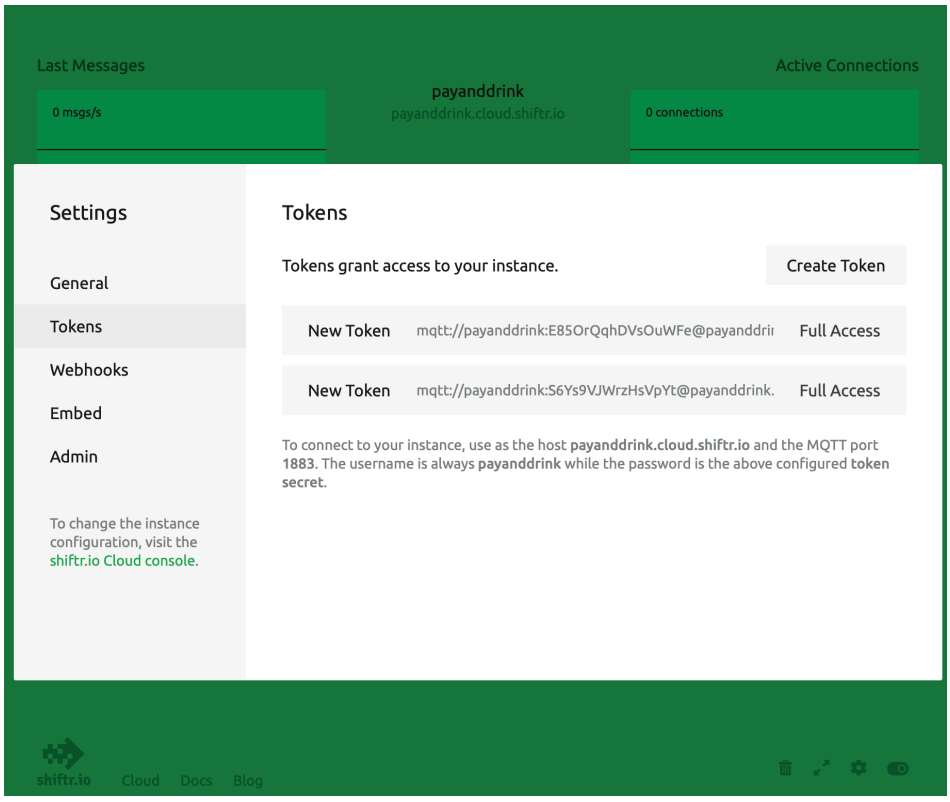


Рисунок 3.22 Генерація токенів для підключення пристроїв до shiftr.io IoT Cloud Service

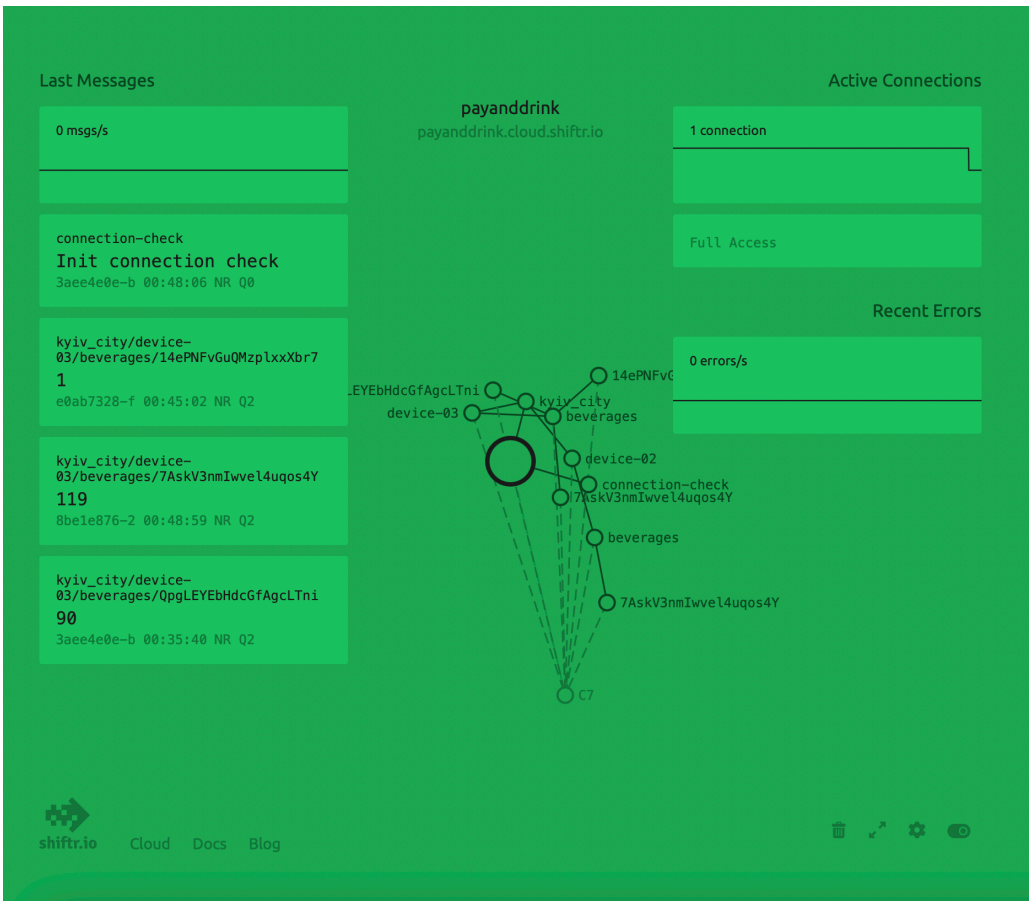


Рисунок 3.23 Ілюстрація зв'язків між підключеними пристроями в shiftr.io IoT Cloud Service

Отже, налаштування MQTT брокера можна вважати завершеним, що дозволяє перейти до задачі розробки скрипту для передачі даних до Firebase Realtime Database.

3.4 Розробка скрипту для передачі даних з MQTT брокера до Firebase Realtime Database

За основу мікропрограми або скрипту, для створення прототипу IoT системи, з локальним виконанням, було використано існуючу Python програму mqtt2firebase.py [28] та покращено для взаємодії з shiftr.io IoT Cloud Service та відправкою необхідних повідомлень в Firebase Realtime Database. Для її налаштування було використано приватний ключ з обліковим записом в Firebase та приватний токен для взаємодії з shiftr.io IoT Cloud Service.

Розроблений скрипт міститься в додатку Б.

3.5 Розробка програмного інтерфейсу для моніторингу даних, отриманих на основі MQTT протоколу та збережених в Firebase Realtime Database

Даний розділ описує засоби, що були використані для розробки програмного інтерфейсу для ІС «Моніторинг наявності товарів для вендингових апаратів», містить вхідну і вихідну інформацію, а також опис основного коду програми, що є частиною розробки та опис інтерфейсу розрозробленої програми для моніторингу.

3.5.1 Опис програмної реалізації мобільного застосунку

Для проектування програмного інтерфейсу моніторингу наявності товарів для вендингових апаратів було використано «чисту архітектуру», описану в розділі 2. У контексті Flutter чиста архітектура допомагає нам розділяти код для бізнес-логіки від коду, пов’язаного з такими платформами, як інтерфейс користувача, управління станом і зовнішні джерела даних. Крім того, для, написаного за чистою архітектурою, коду, зручно писати Unit і Widget тести, оскільки можна протестувати кожен з рівнів незалежно один від одного і, в такому випадку, легко усунути будь-яку несправність.

Згідно наведеної архітектури, структура проекту моніторингу товарів (рис. 3.24).

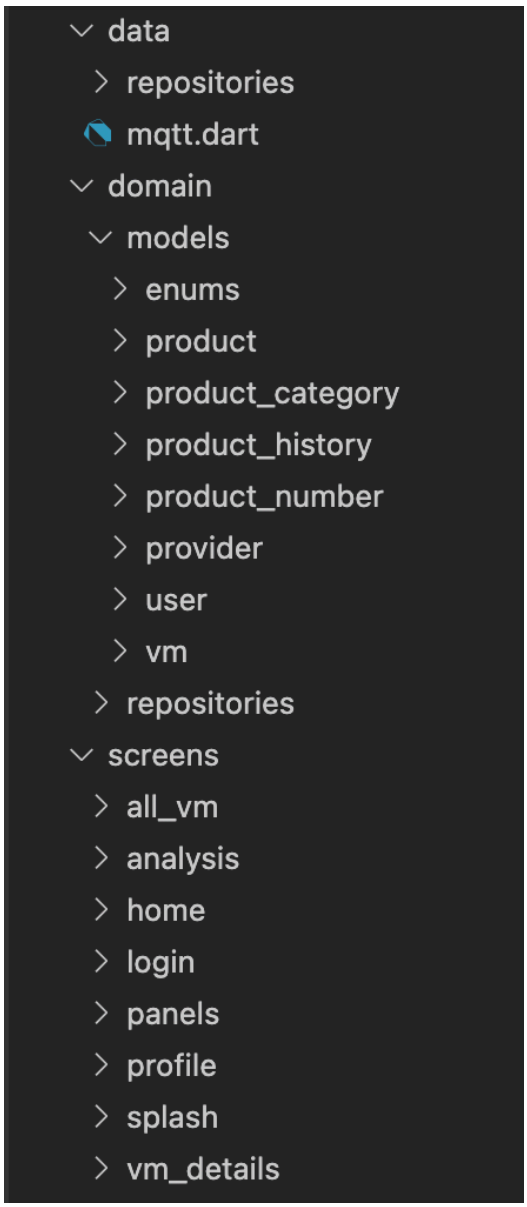


Рисунок 3.24 Структура проекту відповідно до Clean architecture

Для передачі станів між об’єктами було використано архітектурний підхід BloC (Business Logic of Components), який реалізує бізнес логіку мобільного застосунку та відповідає за збереження станів користувача, передачу даних між екранами, відслідковування станів рівня презентації [23]. Наявні блоки зображено нижче (рис. 3.25).

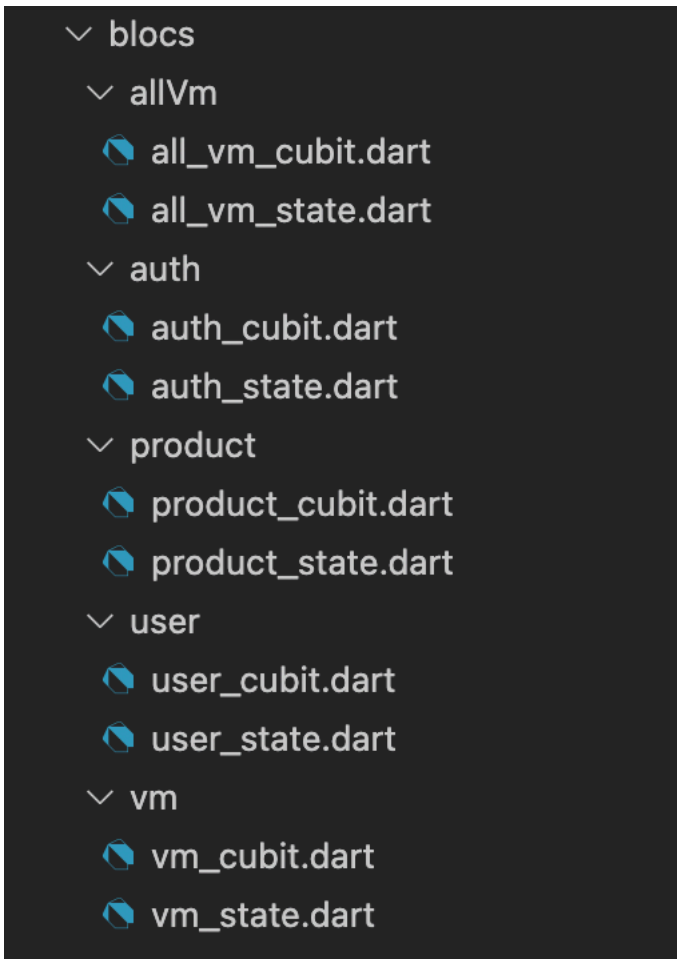


Рисунок 3.25 BloC компоненти проекту

Фреймворк Flutter, повністю підтримує інтеграцію з Firebase Realtime Database [30], за допомогою бібліотеки firebase_database, яка була використана в даному ПЗ.

Згенерований програмний код Flutter міститься в додатку В.

Вхідна інформація для проектування програмного застосунку – фізична модель бази даних, яка спроєктована в розділі 2.3.

3.5.2 Огляд функціональності мобільного застосунку

Графічний інтерфейс застосунку досліджуваної інформаційної системи «Моніторинг наявності товарів для вендингових апаратів» містить такі компоненти: авторизацію (рис. 3.26), головний екран, що відображає список працюючих вендингових автоматів для даної області, екран аналітики товарів та екран профілю користувача. Для отримання детальної інформації про кількість товарів створено

екран зі списком продукції та їхньої кількості. При відсутності або помилці завантаження даних буде відображено екран з відповідним повідомленням (рис. 3.27)

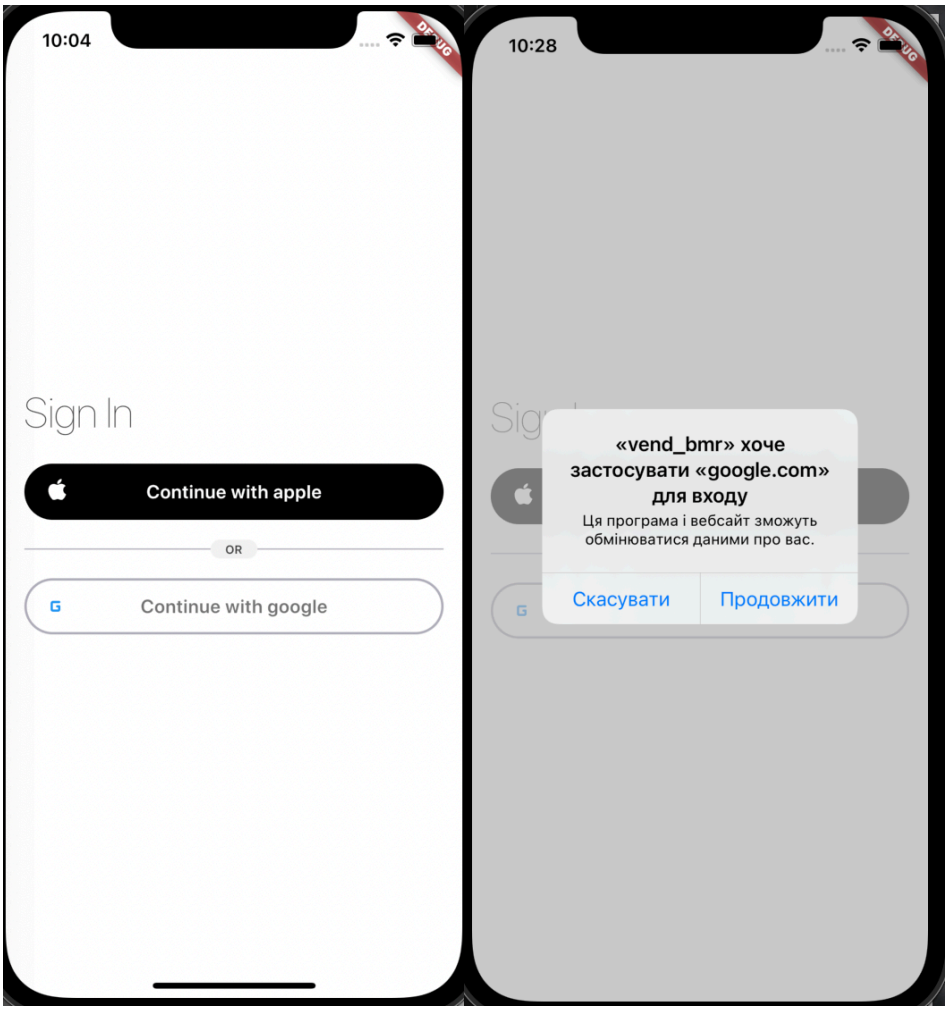


Рисунок 3.26 Екран авторизації та запит входу за допомогою сервісу Google

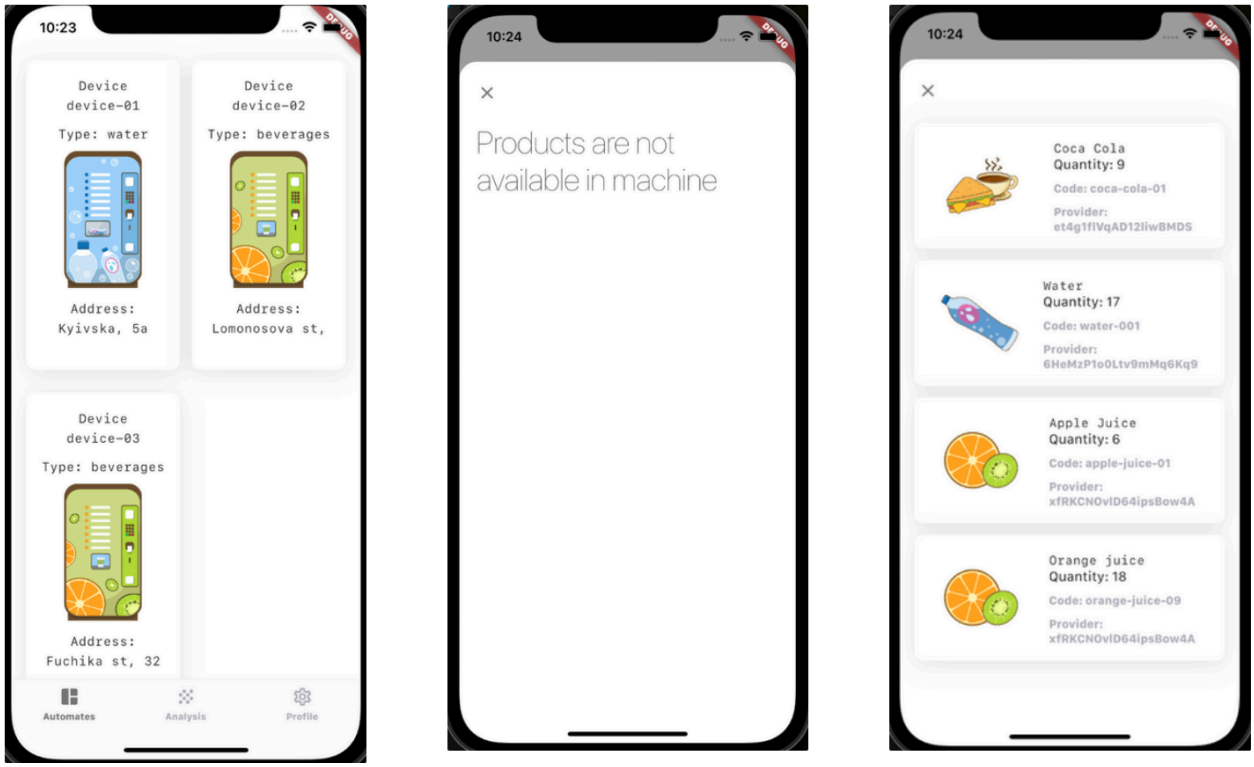


Рисунок 3.27 Екран з списком автоматів та листок (bottom sheet) з інформацією про продукцію

Для аналізу наявності і покупок продукції була розроблена вкладка Analysis з графіками статистики для всіх машин та окремо для кожного з автоматів. При недостатній кількості даних для аналітики відображається відповідне повідомлення. Вкладка статистики є 2-ю вкладкою (рис. 3.28).

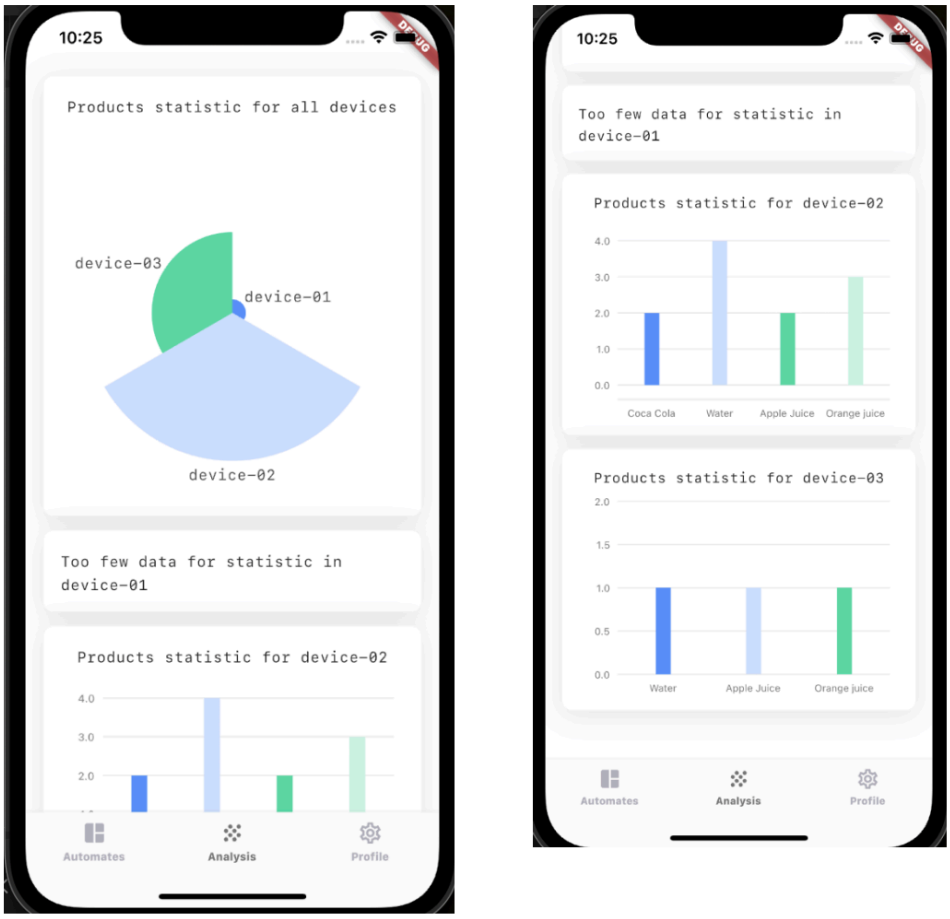


Рисунок 3.28 Вкладка статистики вендингових автоматів

Основна інформація про користувача та кнопка виходу з системи знаходяться на вкладці профілю (рис. 3.29).

Рисунок 3.29 Вкладка профілю користувача

Інструкція користувачеві для даного застосунку міститься в додатку Г.

3.6 Розробка прототипу мобільного застосунку для купівлі товару у вендинговому автоматі

Даний підрозділ описує засоби, що були використані для розробки програмного інтерфейсу для купівлі товарів у вендингових апаратах розробленої IoT системи, містить вхідну і вихідну інформацію, а також опис основного коду програми, що є частиною розробки та опис інтерфейсу користувача програми.

3.6.1 Опис програмної реалізації

Розробка прототипу мобільного застосунку для купівлі товару схожа за архітектурою на застосунок для моніторингу, описаного в розділі 3.5, з різницею в екранах та підключенням додаткових бібліотек для сканування QR-коду та публікування повідомлень з використанням протоколу MQTT. Структура проекту відповідно до Clean architecture представлено нижче (рис. 3.30).

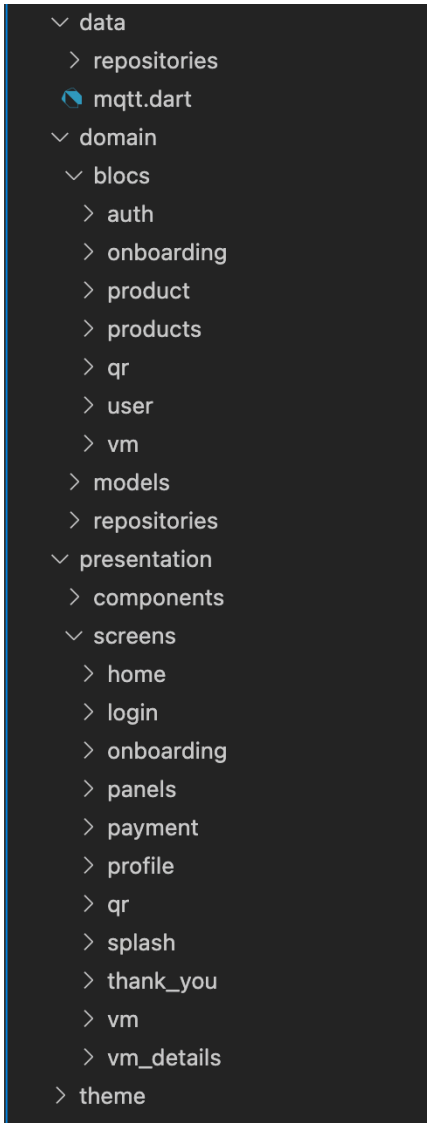


Рисунок 3.30 Структура проекту відповідно до Clean architecture

Структура рівня domain для даного проекту (рис. 3.31).

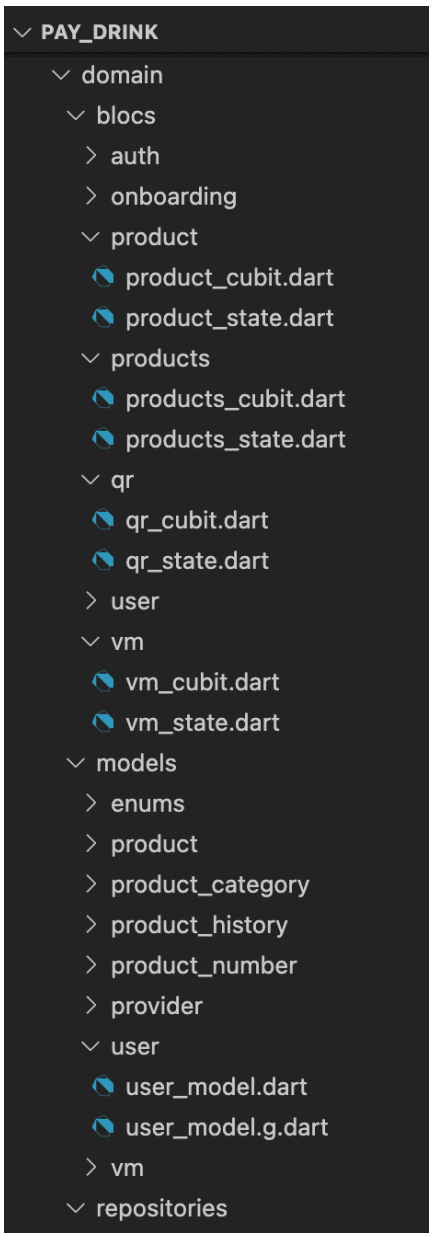


Рисунок 3.31 BloC компоненти проекту

Згенерований програмний код Flutter міститься в додатку Д.

3.6.2 Огляд функціональності застосунку

Для сканування QR-коду був розроблений екран з відкритою активною камерою (рис. 3.32), який при наведенні на QR-код та проведенні верифікації змінюється на екран з відповідним автоматом (рис. 3.33).

Інформація про продукти міститься на окремому листі, який з’явиться після натискання на зображення торгового апарата (рис. 3.34).

Екрани для вибору кількості товару та проведення оплати (рис. 3.35). Також текстове поле, що дозволяє введення кількості бажаної продукції та містить валідацію, проілюстровано на цьому ж зображенні.

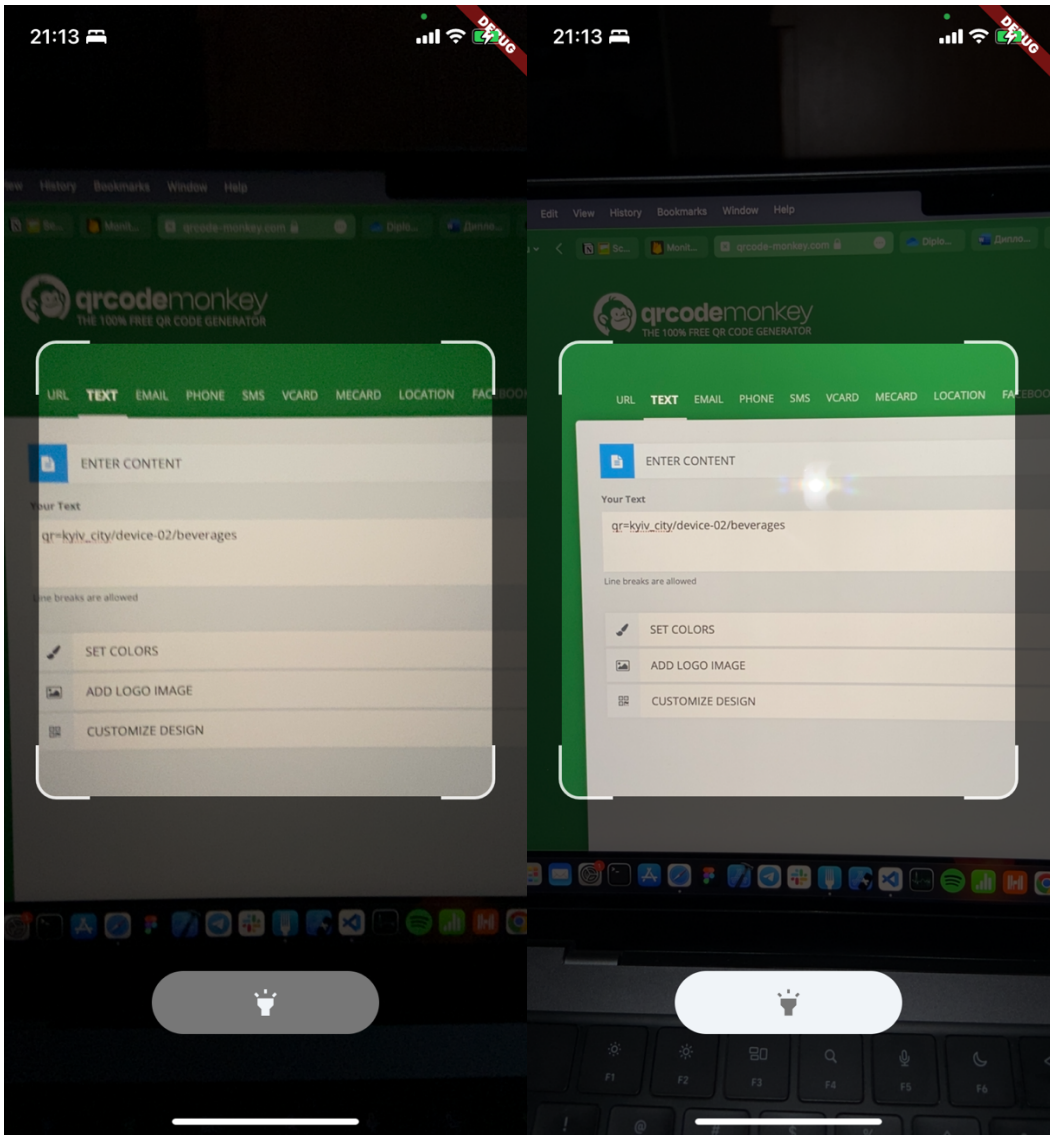


Рисунок 3.32 Екран сканування QR-коду

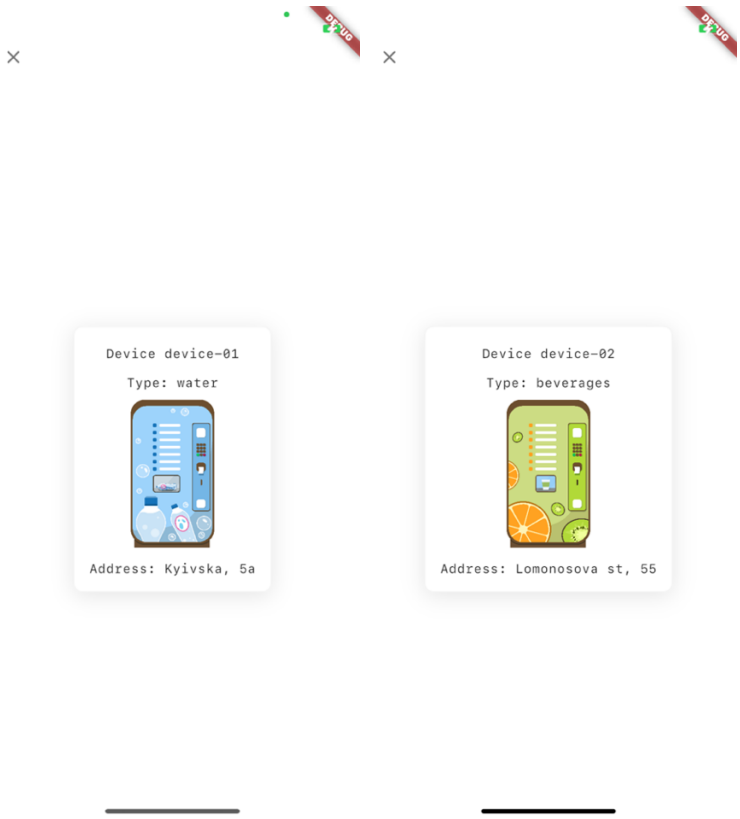


Рисунок 3.33 Екрани з даними про відсканові торгові автомати

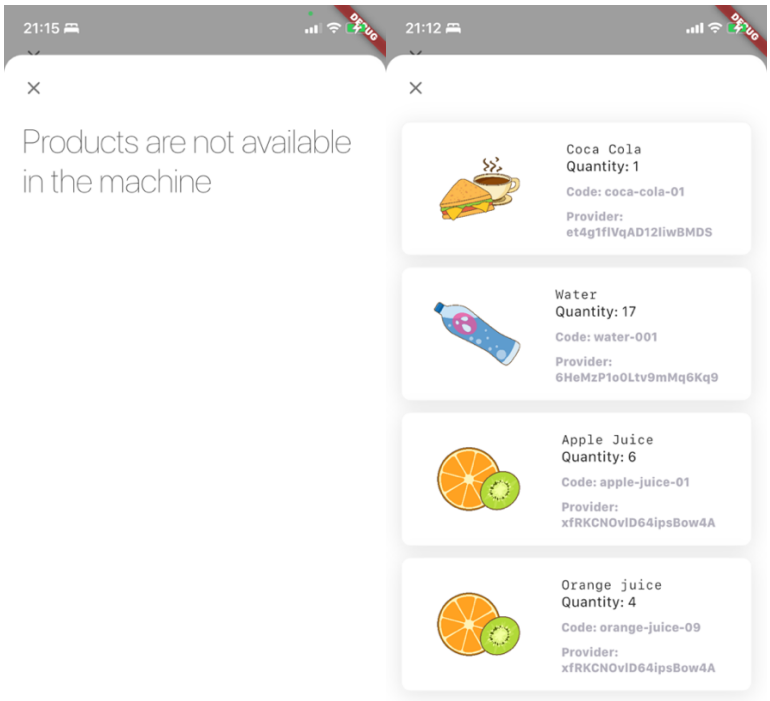


Рисунок 3.34 Екрани з інформацією про продукцію

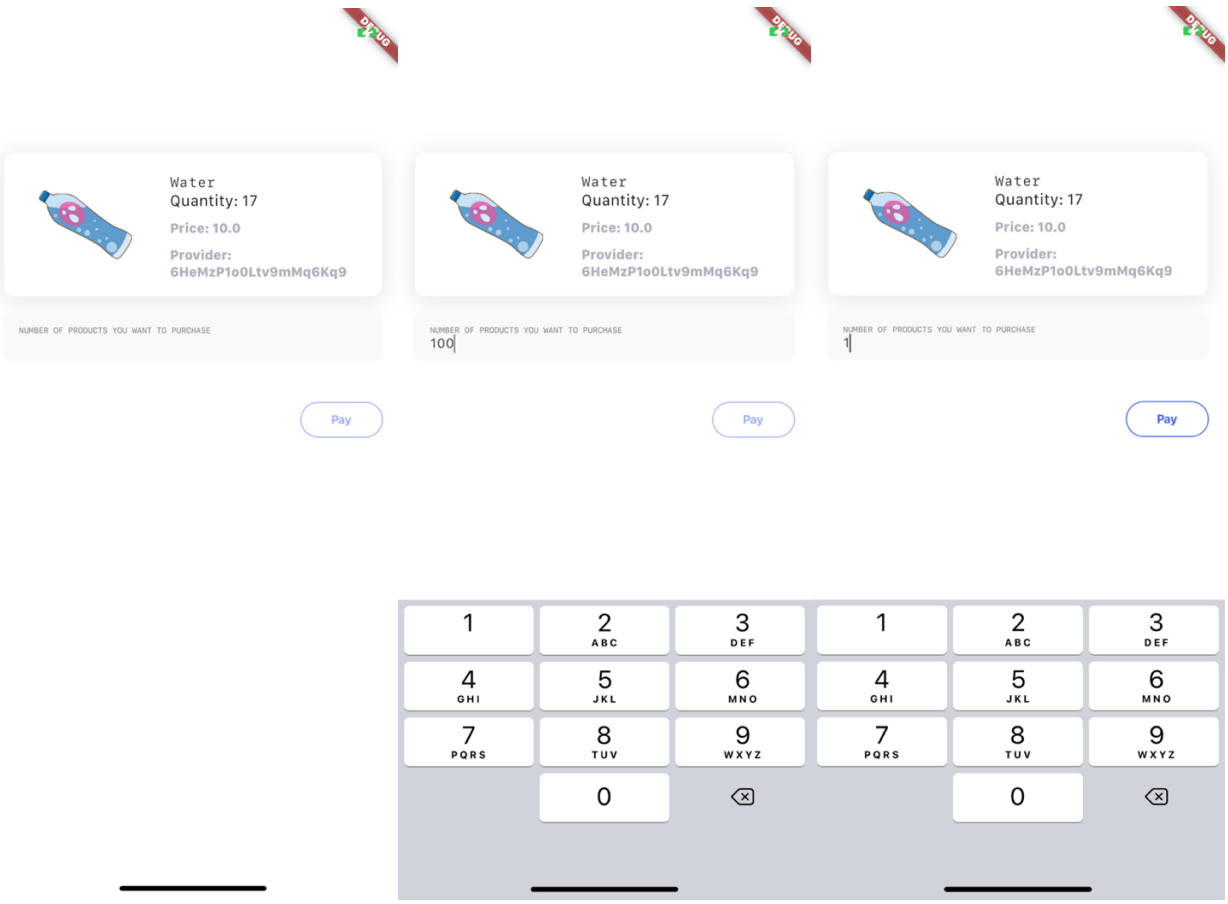


Рисунок 3.35 Екрани для вибору кількості продукції та оплату

Екран після завершення оплати та публікування повідомлень з використанням протоколу MQTT, який в подальшому може містити оцінку сервісу або відгук користувача (рис. 3.36).

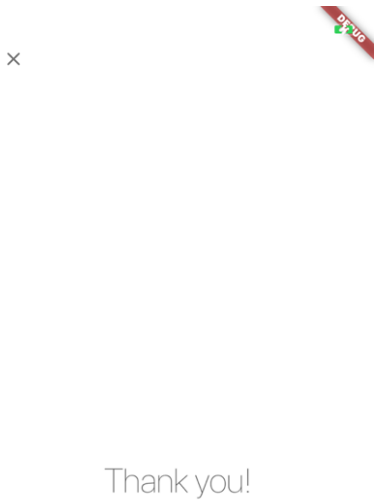


Рисунок 3.36 Екран після завершення оплати

Інструкція користувачеві для даного застосунку міститься в додатку Е.

3.7 Тестування розробленої IoT-системи

Тестування системи варто розділити на 3 частини: виконання скрипту з передачею даних до Firebase Realtime Database та запуск мобільного застосунку для купівлі і застосунку для моніторингу.

3.7.1 Виконання скрипту для передачі даних до Firebase Realtime Database

Для коректного виконання скрипту, перш за все, потрібно запустити shiftr.io IoT Cloud Service, який містить екземпляр pay-drink для віддаленого хостингу MQTT брокера.

Щоб виконати Python мікропрограму для передачі даних до Firebase Realtime Database, вводимо в терміналі команду – python3 mqtt2firebase.py -a 'vendbmr-firebase-adminsdk-30urz-ba6f5524e1.json' -N 'vendbmr-default-rtdb' -t 'kyiv_city' -v. Дана

команда містить шлях до файлу з конфігурацією доступу до хмарної платформи Firebase – vendbmr-firebase-adminsdk-30urz-ba6f5524e1.json і назву теми, яку буде слухати автомат, в даному випадку це 'kyiv_city', оскільки скрипт виконується локально та підписується на тему для усієї локації.

Після виконання скрипту, на графічному інтерфейсі хмарного сервісу shiftr.io з'явиться інформація про підключення. Виконання скрипту в IDE Visual Studio та графічний інтерфейс shiftr.io з активним зв'язком зображено нижче (рис. 3.37– 3.38).

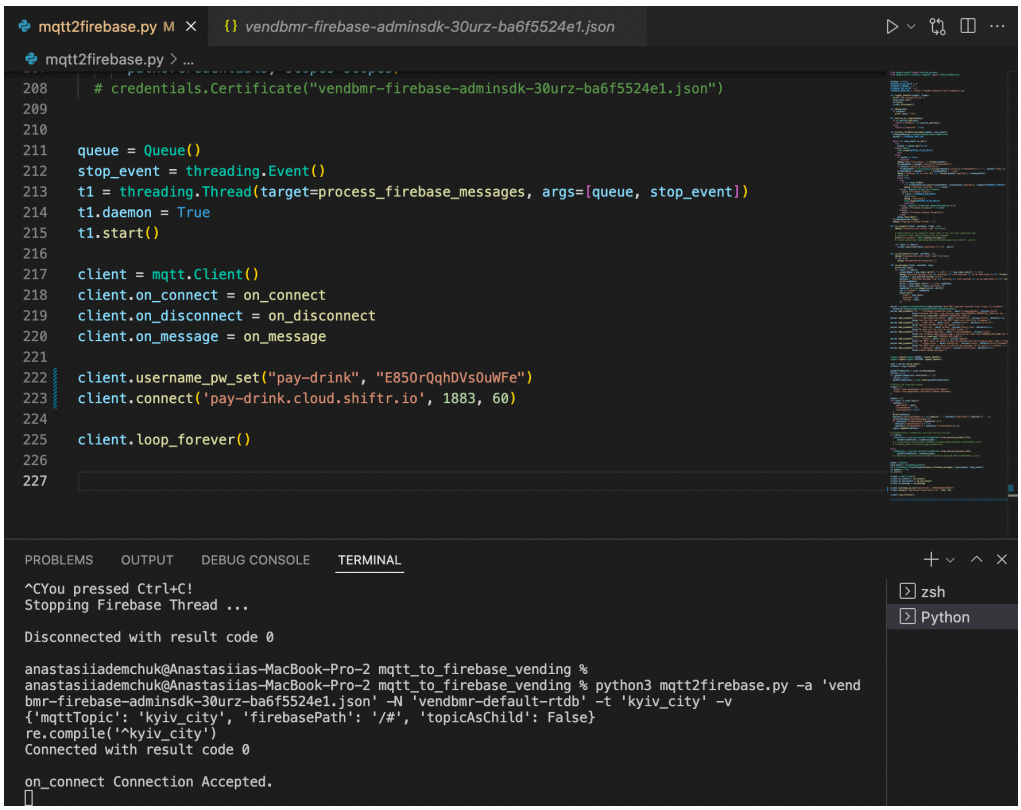


Рисунок 3.37 Виконання скрипту в IDE Visual Studio

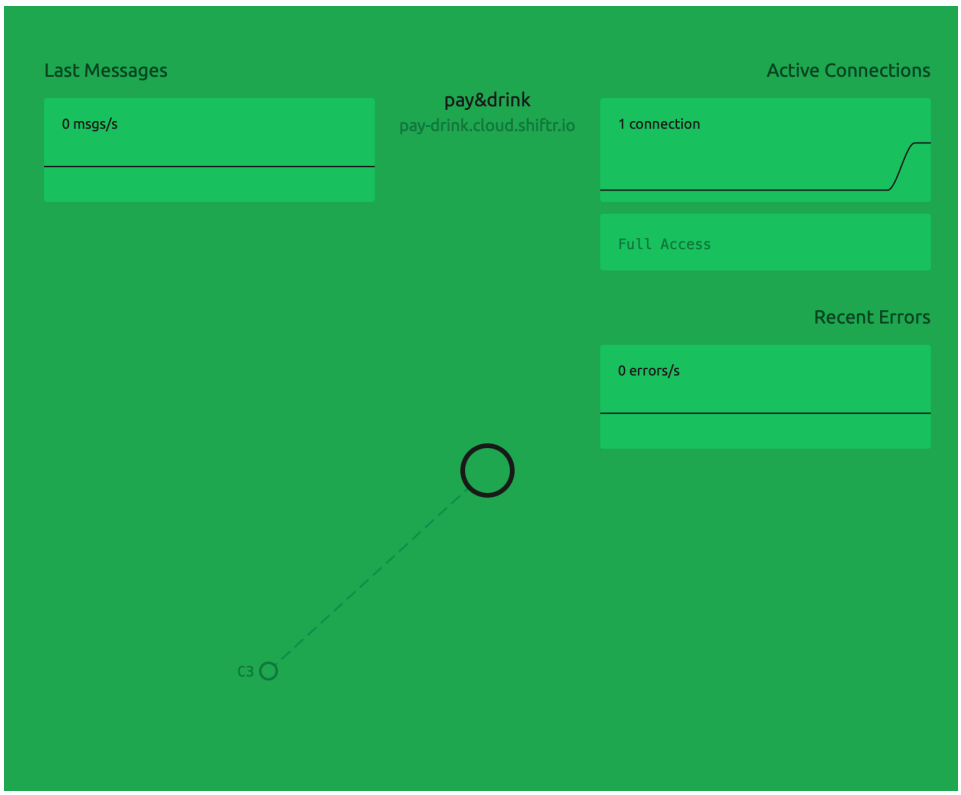


Рисунок 3.38 Графічний інтерфейс shiftr.io з активним зв'язком

Отже, умовна ідентифікація вендингового автомату, за допомогою розробленого скрипту, була успішно проведена в мережі shiftr.io IoT Cloud Service.

3.7.2 Запуск мобільного застосунку для купівлі

Тестування мобільного застосунку для купівлі передбачає сканування QR-коду, який повинен містити унікальний ID вендингового апарату, генерацію якого було зроблено з використанням сервісу QRcodemonkey (рис.3.39).

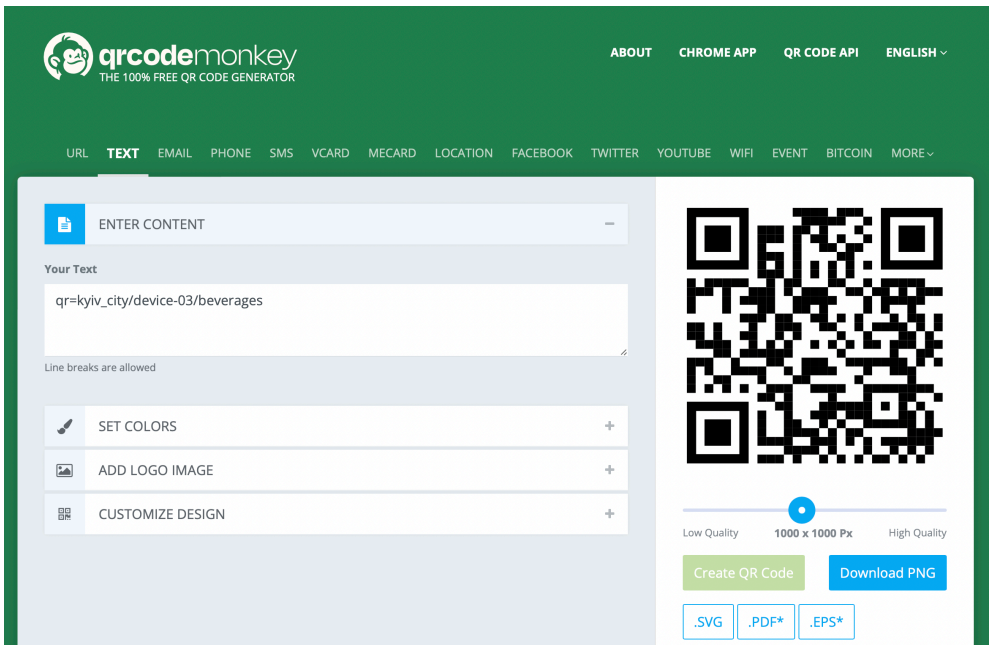


Рисунок 3.39 Згенерований QR-код для тестування мобільного застосунку

Для запуску програмного застосунку потрібно під’єднати мобільний пристрій (для даного тестування був обраний телефон моделі iPhone 11) до комп’ютера з операційною системою macOS, на якому, за допомогою програми Xcode, буде завантажено і виконано програму (рис. 3.40).

Рисунок 3.40 Іконка завантаженої програми Pay&Drink

Головний екран програми – сканер QR-коду, наводимо на згенероване зображення. Після верифікації автомату користувач буде спрямований на екран з відповідним автоматом. Далі потрібно вибрати кількість товару та провести симуляцію оплати, після чого застосунок надсилає повідомлення на отриману з QR-коду тему MQTT брокеру. Інформація про публікацію повідомлення з’явиться на графічному інтерфейсі shiftr.io IoT Cloud Service, а скрипт mqtt2firebase відправить повідомлення на хмарну платформу в Firebase Realtime Database. Дії, описані вище, послідовно зображені нижче (рис. 3.41 – 3.46).

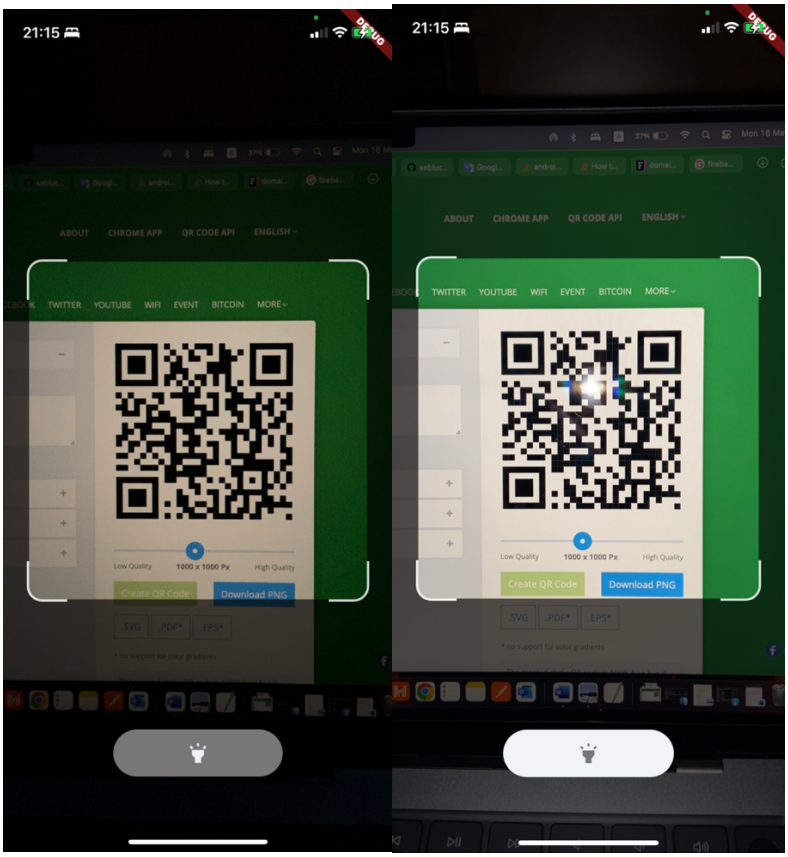


Рисунок 3.41 Сканування QR-коду програми Pay&Drink

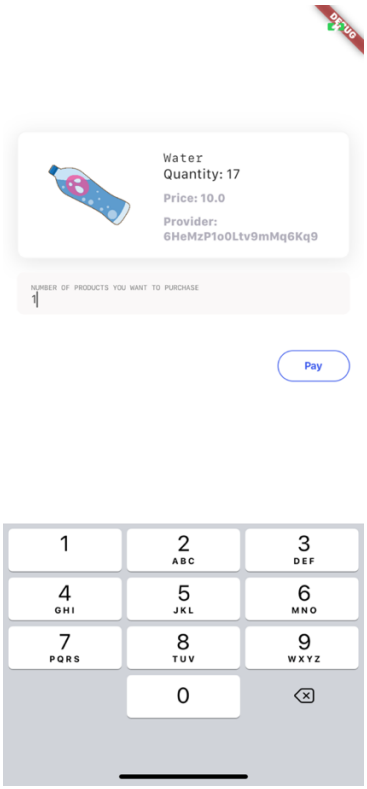


Рисунок 3.42 Вибір кількості товару та проведення симуляції оплати

```
flutter: connecting....
flutter: TOPIC
flutter: kyiv_city/device-02/beverages/7AskV3nmIwvel4uqos4Y
flutter: EXAMPLE::Mosquitto client connecting....
flutter: EXAMPLE::OnConnected client callback - Client connection was successful
flutter: EXAMPLE::Mosquitto client connected
2 flutter: EXAMPLE::Publishing our topic
flutter: device_widget_conection
flutter: EXAMPLE::Subscription confirmed for topic kyiv_city/device-02/beverages/7AskV3nmIwvel4uqos4Y
flutter: EXAMPLE::Change notification:: topic is <kyiv_city/device-02/beverages/7AskV3nmIwvel4uqos4Y>, payload is <-- 3 -->
flutter:
```

Рисунок 3.43 Логування відправки повідомлення на отриману з QR-коду тему MQTT брокеру

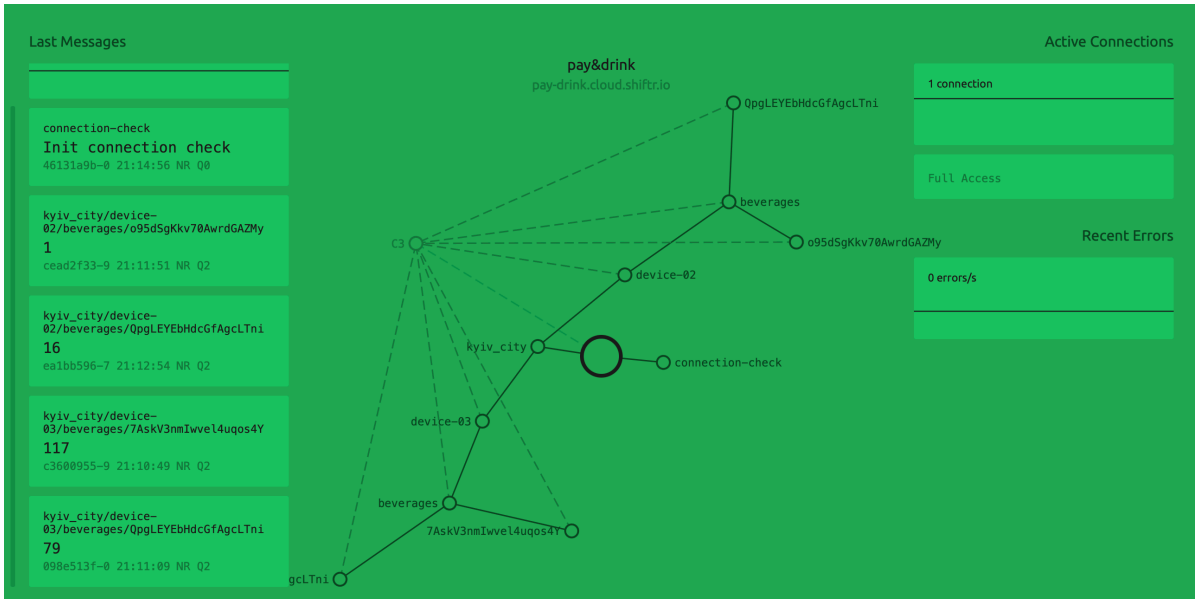


Рисунок 3.44 Отримані повідомлення на відповідну тему MQTT брокером

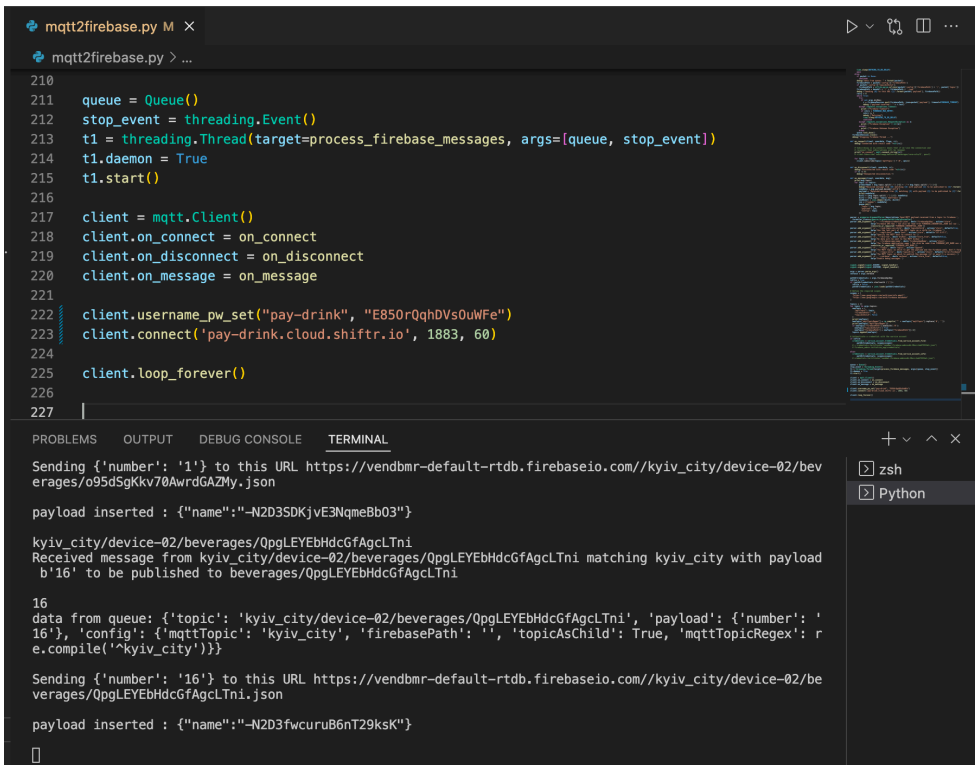


Рисунок 3.45 Логування відправки повідомлення, підслуханого, на вибрану при запуску тему, з MQTT брокеру, до Firebase Realtime Database

Рисунок 3.46 Поява нових повідомлень в Firebase Realtime Database

Отже, застосунок було успішно протестовано та здійснено симуляцію оплати, при проведенні якої відбулося оновлення інформації про стан продуктів в Firebase Realtime Database.

3.7.3 Запуск мобільного застосунку для моніторингу

Для тестування мобільного застосунку моніторингу товарів було використано симулятор мобільного пристрою від програми Xcode (рис. 3.47).

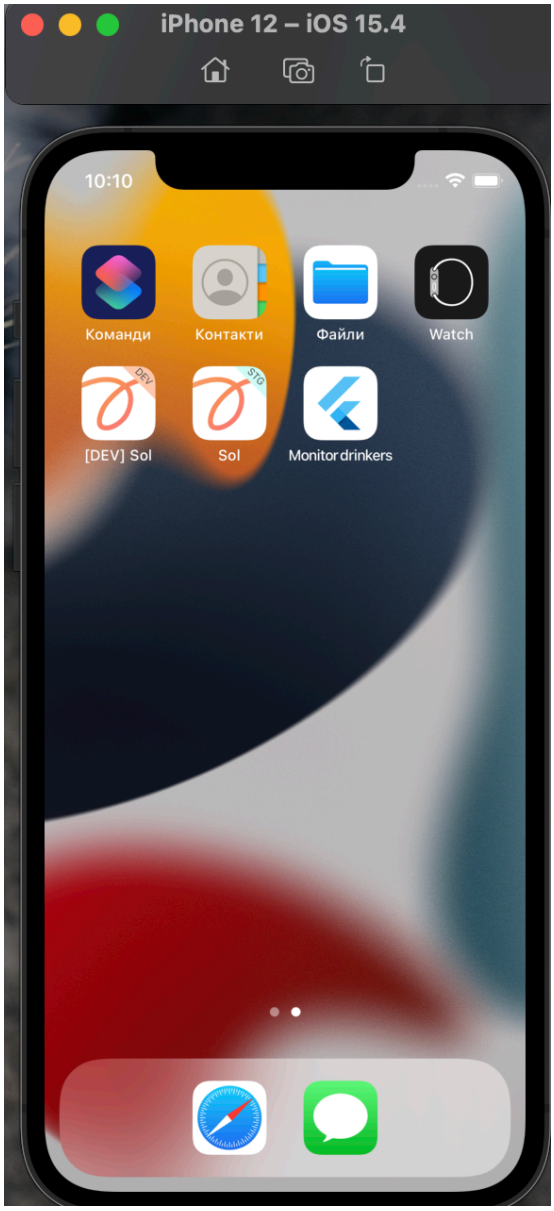


Рисунок 3.47 Іконка завантаженої програми на симулятор мобільного пристрою

Оновлені дані про кількість продукції, синхронізовані з Firebase Realtime Database (рис. 3.48), а також представлено оновлену статистику (рис. 3.49).

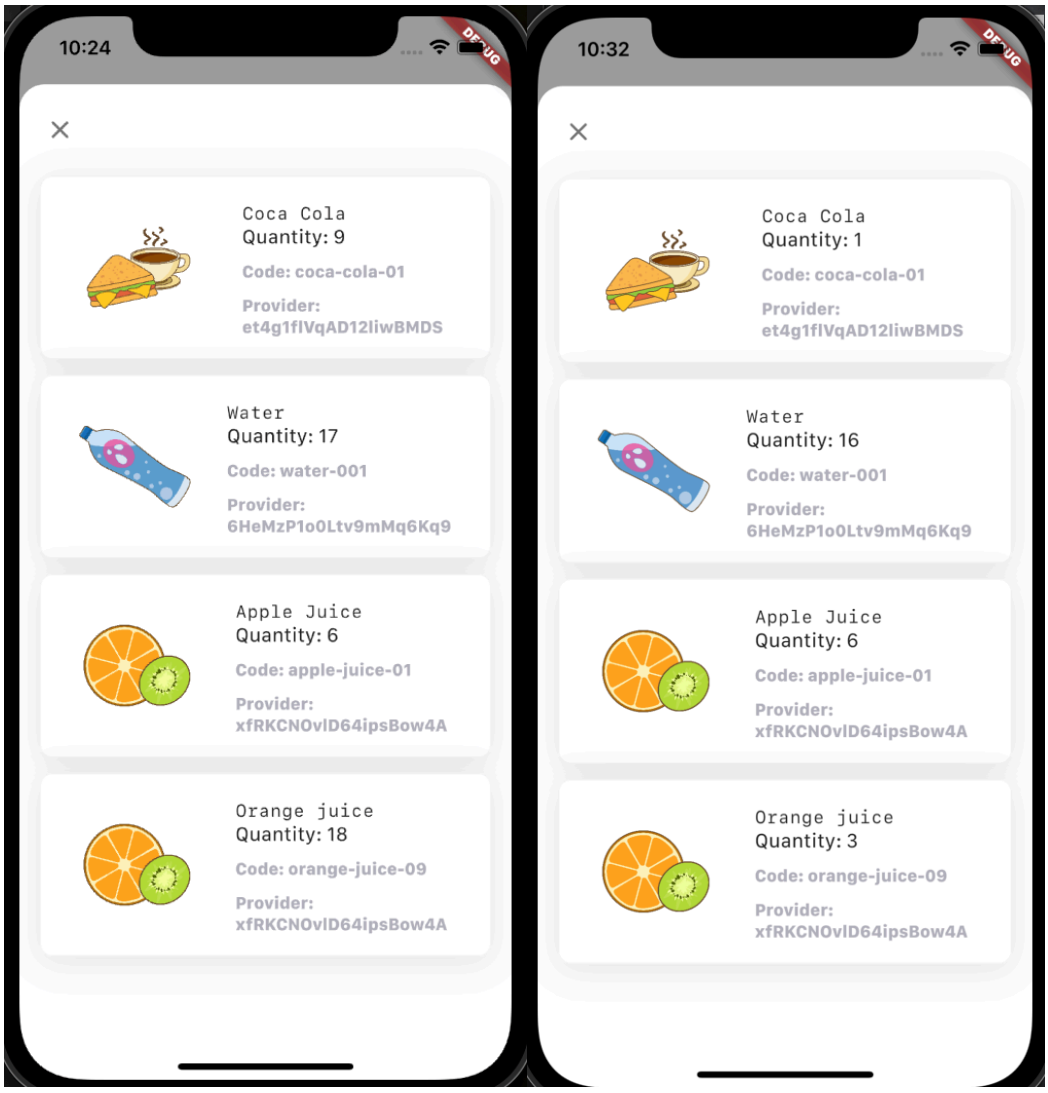


Рисунок 3.48 Оновлені дані, синхронізовані з Firebase Realtime Database

Рисунок 3.49 Оновлена статистика на основі даних, синхронізованих з Firebase Realtime Database

Отже, була проілюстрована синхронізація даних системи моніторингу, відповідно з діями клієнта, тому проведене тестування можна визнати успішним і закінченим.

3.8 Аналіз отриманих даних

Однією з переваг хмарної платформи Firebase є наявність служби аналітики, яка дозволяє відстежувати користування застосунком, кількість активних користувачів, записів в БД тощо. В моїй дипломній роботі було розглянуто аналітику Realtime Database (рис. 3.50) та проаналізовано користування застосунком за допомогою сервісу Firebase Analytics (рис. 3.51 – 3.55).

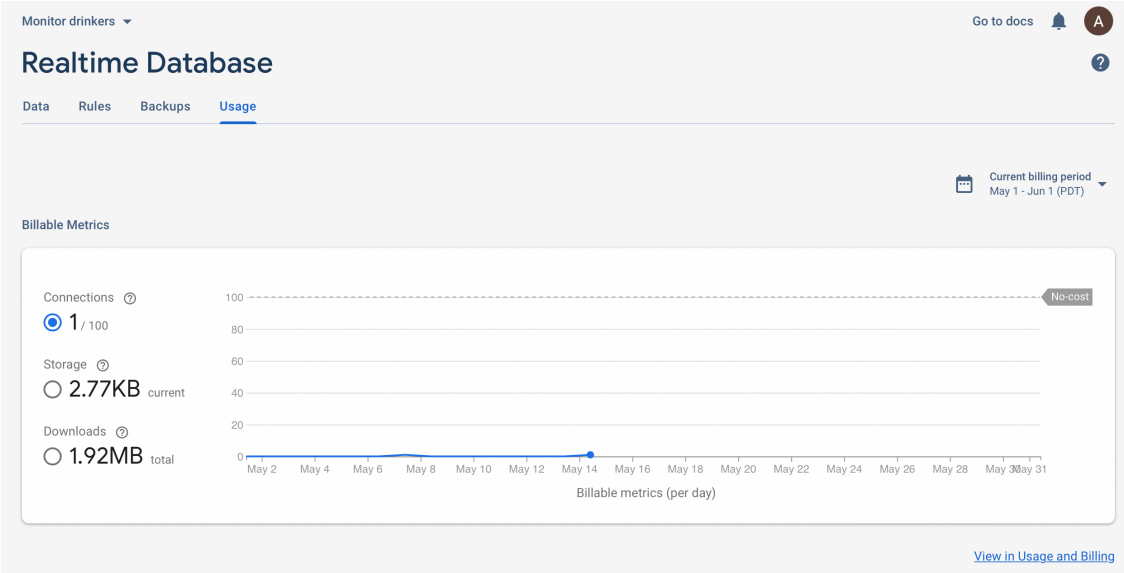


Рисунок 3.51 Аналітика записів в Firebase Database

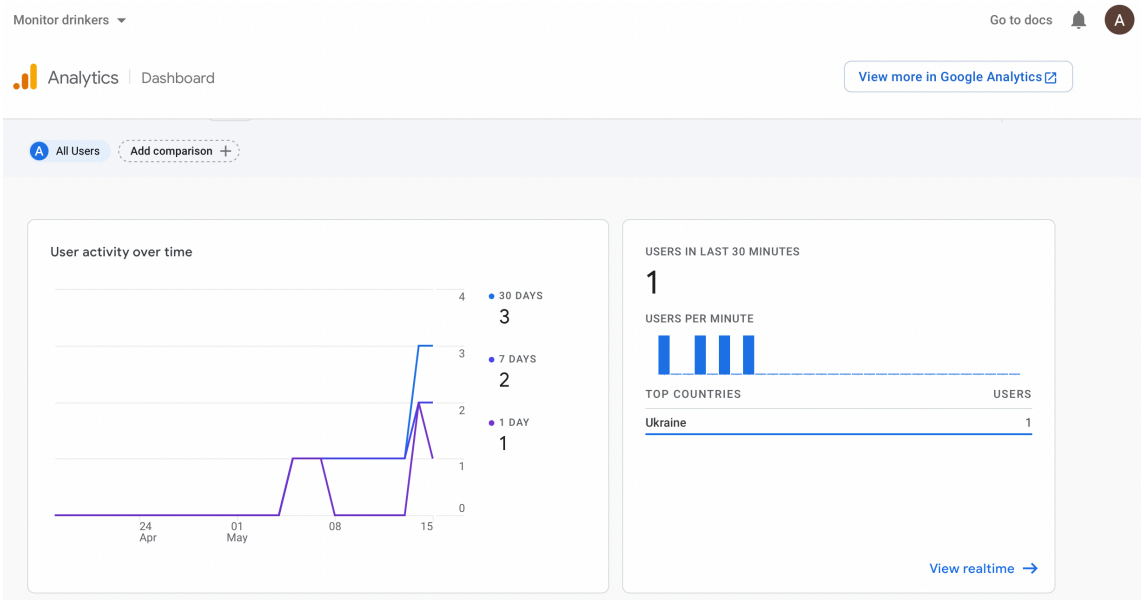


Рисунок 3.52 Аналітика активності користувачів застосунку

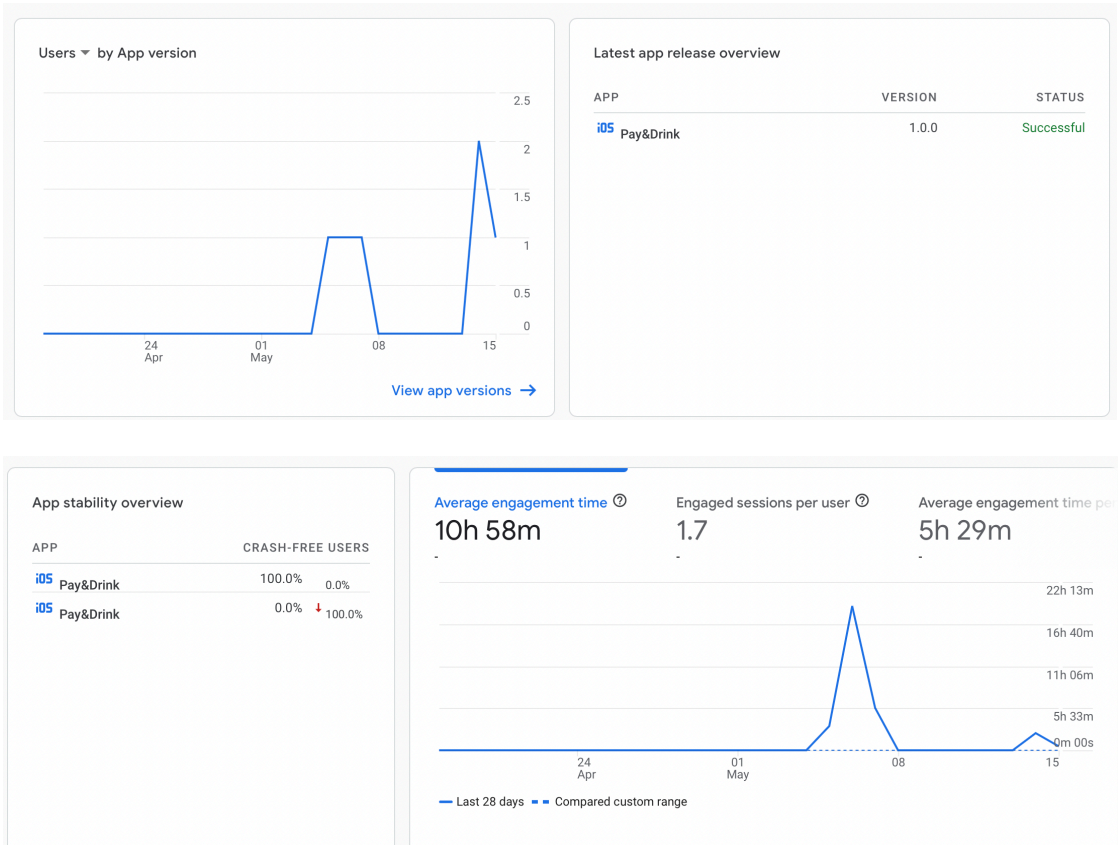


Рисунок 3.53 *Аналітика кількості активних користувачів* застосунку

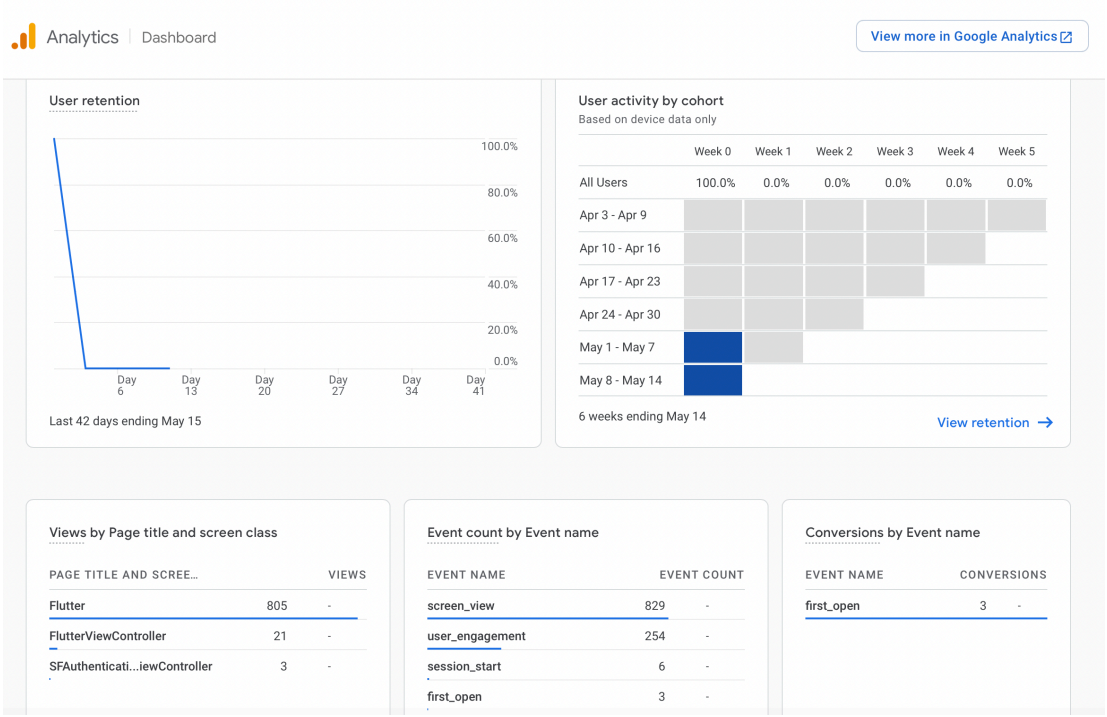


Рисунок 3.54 *Аналітика перегляду екранів у* застосунку

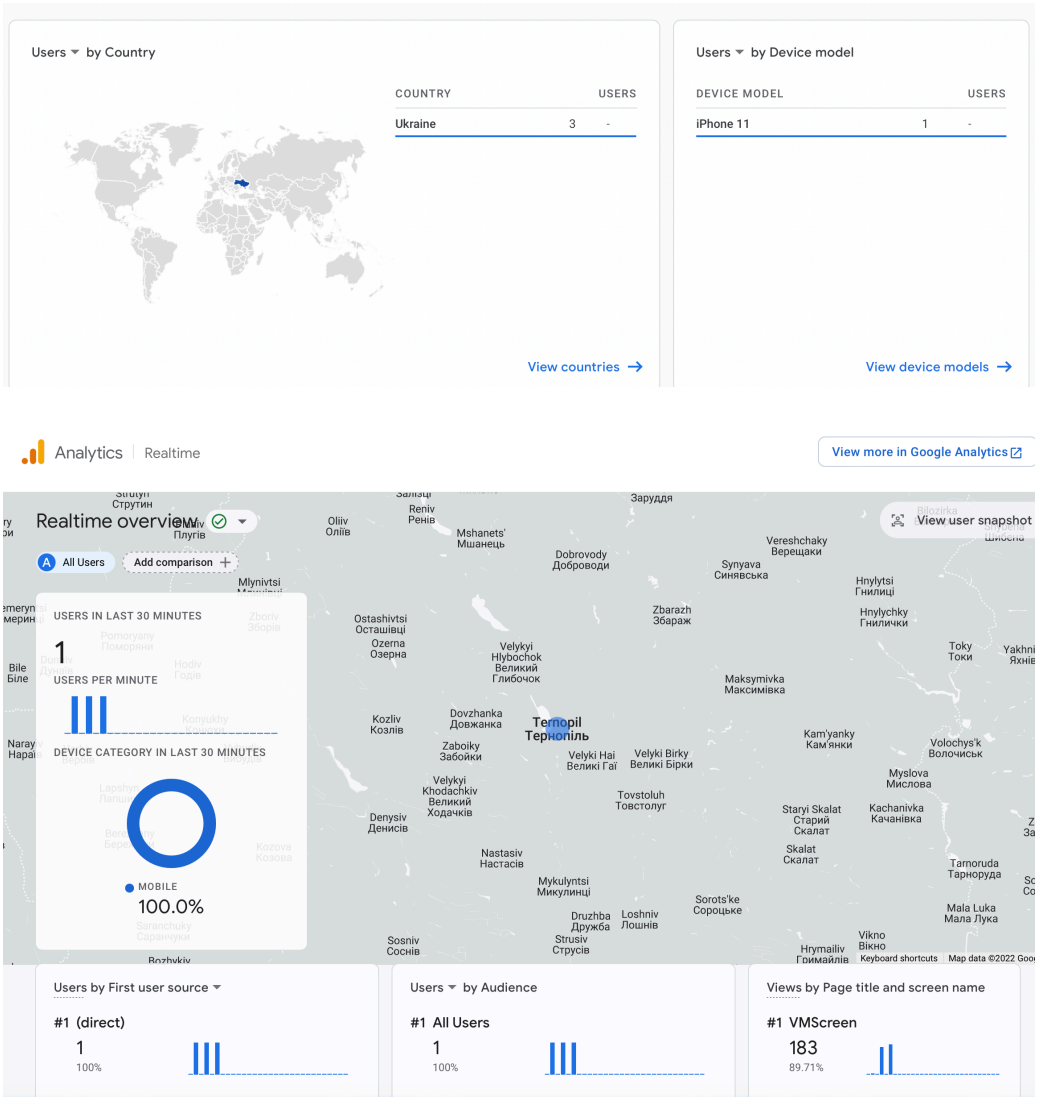


Рисунок 3.55 Аналітика переважного місцезнаходження користувачів застосунку

Згідно приведеної аналітики, користувач знаходився переважно в місті Тернопіль та найбільше користування застосунком припадає на період 1-10 травня, з відкриттям головного екрану сумарно 829 разів.

3.9 Висновки до розділу

В даному розділі було проаналізовано та обрано програмне забезпечення для розробки IoT системи моніторингу, спроектовано базу даних, налаштовано хмарну платформу та IoT сервіс для брокера для роботи з системою, а також розроблено два

прототипи мобільних застосунків для моніторингу та купівлі товарів у вендинговій системі.

Даний розділ містить інформацію про програмне тестування компонентів IoT-системи моніторингу товарів для вендингових апаратів, опис їхньої роботи та аналітику даних, отриманих на основі проведених досліджень.

Було виконано програмне тестування компонентів IoT-системи моніторингу товарів для вендингових апаратів, описано їхню роботу та розглянуто аналітику в хмарному сервісі Firebase Analytics на основі даних, одержаних у ході практичних досліджень розроблених програмних засобів.

ВИСНОВКИ

Нині системи забезпечення вендингу є одним з провідних напрямків розвитку малого бізнесу в цілому світі. Як результат, ця сфера потребує автоматизації та сучасних засобів управління процесами, що можна забезпечити лише за наявності якісно спроектованої інформаційної системи. Актуальність розробки такої IoT-системи, яка забезпечить передачу даних про кількість товарів в режимі реального часу та програмних інтерфейсів для її взаємодії і керування було розглянуто у вступі. Згідно постановки задачі було виконано дослідження на тему «Моніторинг наявності товарів для вендингових апаратів», де було розглянуто застосування та вплив технології IoT у галузі вендингу, неведені переваги та недоліки використання Інтернету речей для автоматизації процесу моніторингу, а також проаналізована актуальність створення мобільних застосунків для вендингових інформаційних систем.

Розглянуто і запропоновано рішення у вигляді розробки IoT системи моніторингу товарів в режимі реального часу через мережу Інтернет без втручання людини в цей процес. Дане рішення включає спроектований прототип бази даних для системи моніторингу наявності товарів, мобільний застосунок для моніторингу наявності товарів та аналізу даних, а також застосунок для прямої роботи з вендинговим апаратом, а саме купівлі товарів у ньому.

Моя розробка полегшує доступ до інформації про кількість продуктів у вендингових автоматах, яка необхідна для аналізу продажів та щодо наявності товарів у режимі реального часу в інформаційній системі «Моніторинг наявності товарів для вендингових апаратів».

Згідно поставлених завдань було:

1) спроектовано базу даних для ІС моніторингу вендингових апаратів із застосуванням технології IoT для передачі даних про кількість товарів у режимі реального часу без втручання людини в даний процес;

2) розроблено мобільний застосунок для зручного моніторингу наявності товарів кінцевими користувачами системи;

3) створено мобільний застосунок для підключення до вендингового апарату і купівлі товарів;

4) налаштовано MQTT брокер та розроблено скрипт для передачі даних з вендингового автомату до БД.

Мета дипломної роботи, що передбачає дослідження, проектування та програмну реалізацію частини IoT системи моніторингу наявності товарів для вендингової системи – досягнута.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Ponomarenko, R. and Demchuk, A. IoT Control Systems base on Fuzzy PWM-controller. CEUR Workshop Proceedingsthis. – vol. 3106. – 2021, pp. 55–64.
2. Oracle - What Is the Internet of Things (IoT)? [Електронний ресурс] – Режим доступу: https://www.oracle.com/internet-of-things/what-is-iot/ (дата звертання - 14.05.2022)
3. 24×7 Revenue with Smart Vending and the IoT [Електронний ресурс] – Режим доступу: https://www.insight.tech/content/24-7-revenue-with-smart-vending-and-the-iot#main-content (дата звертання - 14.05.2022)
4. Telemetry: Summary of concept and rationale [Електронний ресурс] – Режим доступу: https://ui.adsabs.harvard.edu/abs/1987STIN...8913455./abstract (дата звертання - 15.05.2022)
5. Advantages of Smart Vending Machine | disadvantages of Smart Vending Machine [Електронний ресурс] – Режим доступу: https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-IoT-based-smart-vending-machine.html (дата звертання - 19.05.2022)
6. VPOS Touch [Електронний ресурс] – Режим доступу: https://www.nayax.com/cashless_payments/vpos-touch/ (дата звертання - 19.05.2022)
7. AMIT 3.0 [Електронний ресурс] – Режим доступу: https://www.nayax.com/telemetry_solutions/amit3/ (дата звертання - 19.05.2022)
8. ePort G10 Pulse Kit [Електронний ресурс] – Режим доступу: https://www.cantaloupe.com/eport-g10-pulse-kit/ (дата звертання - 19.05.2022)
9. ePort Engage [Електронний ресурс] – Режим доступу: https://www.cantaloupe.com/hardware-eport-engage/ (дата звертання - 19.05.2022)
10. Seed Telemeter [Електронний ресурс] – Режим доступу: https://www.cantaloupe.com/seed-telemeter/ (дата звертання - 19.05.2022)

11. Модель OSI і TCP/IP: Офіційний сайт компанії Cisco [Електронний ресурс] – Режим доступу: Cisco - Global Home Page (дата звертання - 19.05.2022)
12. Protocol MQTT for IoT [Електронний ресурс] – Режим доступу: https://www.engineersgarage.com/featured-contributions/understanding-mqtt-protocol-iot-part-14/ (дата звертання - 19.05.2022)
13. MQTT: The Standard for IoT Messaging [Електронний ресурс] – Режим доступу: https://mqtt.org (дата звертання - 20.05.2022)
14. Mars gains in-store display insights with Azure IoT Central [Електронний ресурс] – Режим доступу: https://customers.microsoft.com/en-us/story/776730-mars-inc-consumer-goods-azure (дата звертання - 19.05.2022)
15. AMS and Vagabond have partnered to deliver the AMS Touchless, the world's first truly touchless vending machine. [Електронний ресурс] – Режим доступу: https://vgbnd.co (дата звертання - 22.10.2021)
16. Keep track — from vending machines to pets with Google Maps APIs plus IoT [Електронний ресурс] – Режим доступу: https://www.blog.google/products/google-cloud/keep-track-vending-machines-pets-google-maps-apis-plus-iot/ (дата звертання - 23.10.2021)
17. Gimme VMS [Електронний ресурс] – Режим доступу: https://apps.apple.com/us/app/gimme-vms/id1444573845 (дата звертання - 24.10.2021)
18. Gimme Field [Електронний ресурс] – Режим доступу: https://apps.apple.com/us/app/gimme-field/id1453434950 (дата звертання - 22.10.2021)
19. Gimme DEX [Електронний ресурс] – Режим доступу: https://apps.apple.com/us/app/gimme-dex/id1104995503 (дата звертання - 22.10.2021)
20. Gimme Key Pro [Електронний ресурс] – Режим доступу: https://www.gimmedsd.com (дата звертання - 22.10.2021)

21. The emerging methods of dex-data collection [Електронний ресурс] – Режим доступу: https://www.vendingmarketwatch.com/home/article/10273789/the-emerging-methods-of-dex-data-collection (дата звертання - 19.05.2022)
22. Опис реалізації технології Gimme Vending [Електронний ресурс] – Режим доступу: https://www.gimmevending.com/gimmevms (дата звертання - 19.05.2022)
23. ProstoPay – IoT рішення для вендингу та кавомашин [Електронний ресурс] – Режим доступу: https://prostopay.net (дата звертання - 19.05.2022)
24. Hackolade [Електронний ресурс] – Режим доступу https://hackolade.com/ (дата звертання - 20.05.2022)
25. Clean Architecture: A Craftsman's Guide to Software Structure and Design by Robert C. Martin — Pearson, 2017. — 445c.
26. Flutter Clean Architecture [Електронний ресурс] – Режим доступу: https://betterprogramming.pub/flutter-clean-architecture-test-driven-development-practical-guide-445f388e8604 (дата звертання - 19.05.2022)
27. Firebase [Електронний ресурс] – Режим доступу: https://firebase.google.com (дата звертання - 22.05.2022)
28. Mqtt to Firebase library (Copyright 2016 Sébastien Lucas sebastien@slucas.fr) – [Електронний ресурс] – Режим доступу: https://github.com/seblucas/mqtt2firebase (дата звертання - 15.06.2022)
29. Flutter technology [Електронний ресурс] – Режим доступу: https://flutter.dev (дата звертання - 19.05.2022)
30. MQTT client for Dart [Електронний ресурс] – Режим доступу: https://pub.dev/packages/mqtt_client (дата звертання - 22.05.2022)
31. Firebase integration in Flutter [Електронний ресурс] – Режим доступу: https://docs.flutter.dev/development/data-and-backend/firebase (дата звертання - 22.05.2022)
32. QRCodeMonkey [Електронний ресурс] – Режим доступу: https://www.qrcode-monkey.com (дата звертання - 23.05.2022)

33. Firebase Console [Електронний ресурс] – Режим доступу: https://console.firebase.google.com/?authuser=0 (дата звертання - 23.05.2022)
34. Firebase Setup Documentation [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/ios/setup?authuser=0&hl=en (дата звертання - 23.05.2022)

ДОДАТОК А

ДОКУМЕНТАЦЯ ДЛЯ ПОБУДОВИ БАЗИ ДАНИХ В FIRESTORE

482.IoT.КНУШ.211297-01

Листів 7

Розробник Демчук А. Б.

Керівник Пономаренко Р. М.

Київ – 2022

1.1.3 New model Model Definitions

1.1.3.1 Field vending_machines-1653138687

1.1.3.1.1 vending_machines-1653138687 Tree Diagram

1.1.3.1.2 vending_machines-1653138687 Hierarchy

Parent field: Definitions
Child field(s):

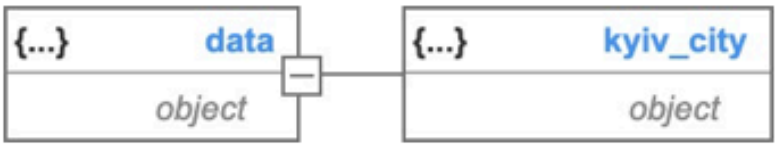
FIELD	TYPE	REQ	KEY	DESCRIPTION	COMMENTS
data	object	true			

1.1.3.1.3 vending_machines-1653138687 properties

PROPERTY	VALUE
Name	vending_machines-1653138687
Technical name	
Id	
Type	object
Description	
Dependencies	
Required	
Primary key	false
Min Properties	
Max Properties	
Additional properties	true
Comments	

1.1.3.2 Field data

1.1.3.2.1 data Tree Diagram



1.1.3.2.2 data Hierarchy

Parent field: vending_machines-1653138687
Child field(s):

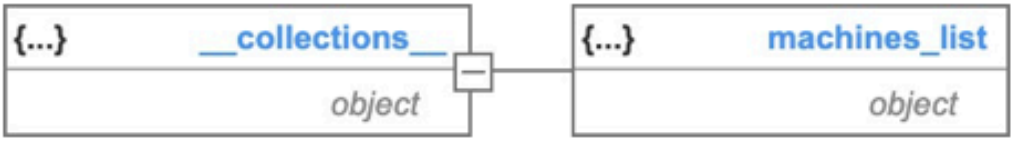
FIELD	TYPE	REQ	KEY	DESCRIPTION	COMMENTS
kyiv_city	object	true			

1.1.3.3.3 kyiv_city properties

PROPERTY	VALUE
Name	kyiv_city
Technical name	
Activated	true
Id	
Type	object
Description	
Dependencies	
Required	true
Primary key	false
Min Properties	
Max Properties	
Additional properties	false
Comments	

1.1.3.4 Field __collections__

1.1.3.4.1 __collections__ Tree Diagram



1.1.3.4.2 __collections__ Hierarchy

Parent field: kyiv_city
Child field(s):

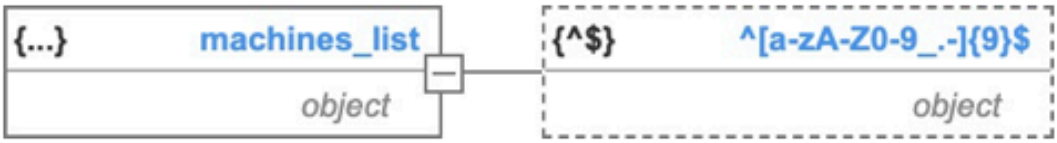
FIELD	TYPE	REQ	KEY	DESCRIPTION	COMMENTS
machines_list	object	true			

1.1.3.4.3 __collections__ properties

PROPERTY	VALUE
Name	__collections__
Technical name	
Activated	true
Id	
Type	object
Description	
Dependencies	
Required	true
Primary key	false
Min Properties	
Max Properties	
Additional properties	false
Comments	

1.1.3.5 Field machines_list

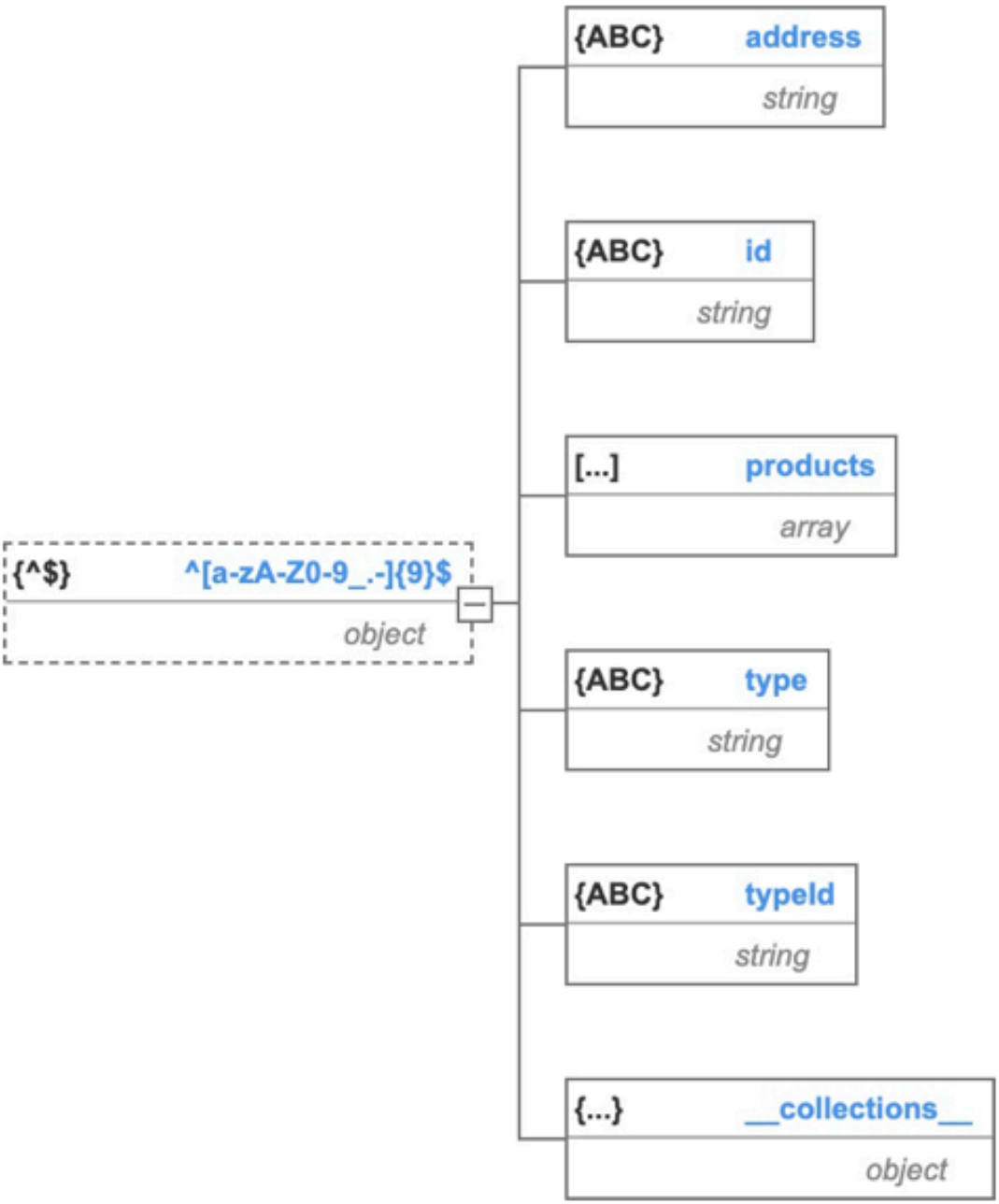
1.1.3.5.1 machines_list Tree Diagram



1.1.3.5.2 machines_list Hierarchy

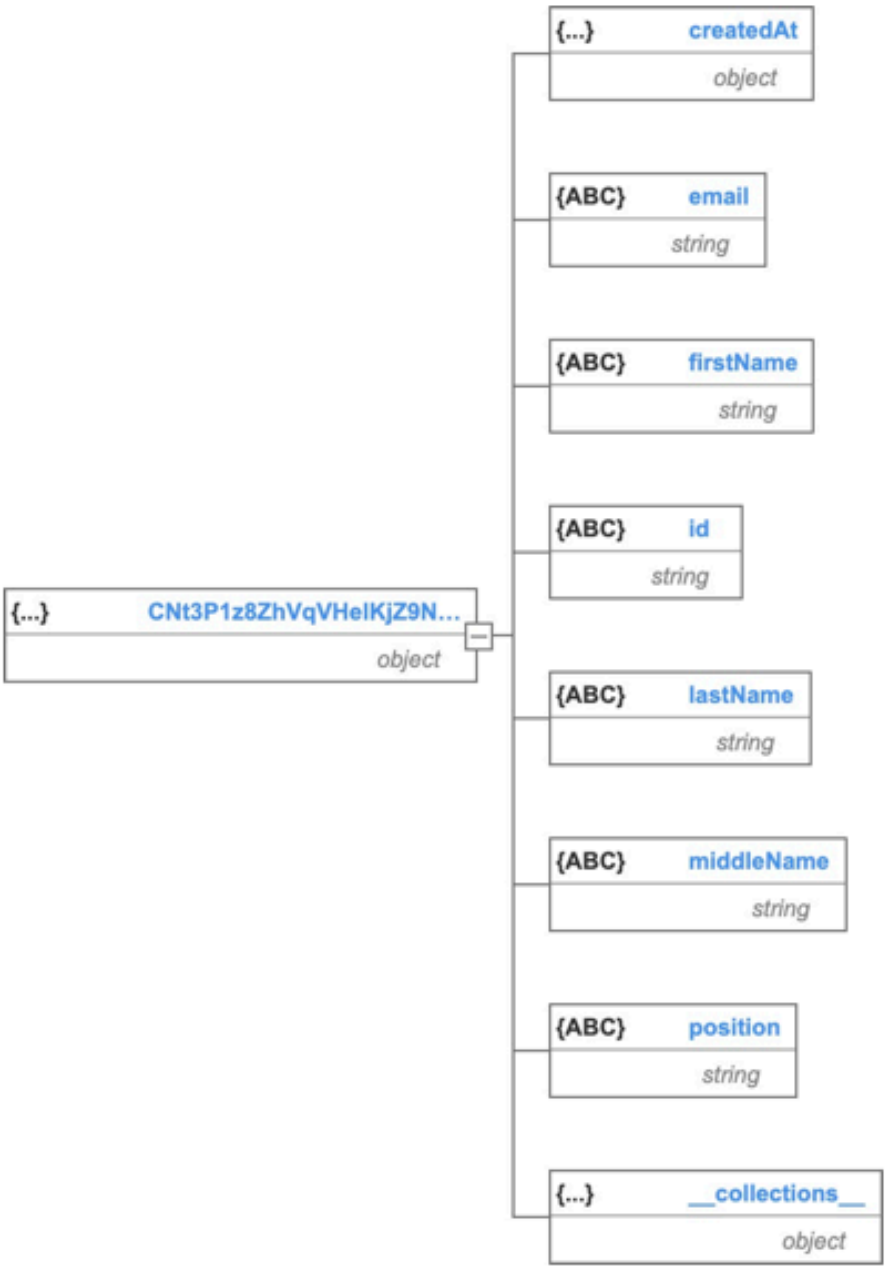
Parent field: __collections__
Child field(s):

FIELD	TYPE	REQ	KEY	DESCRIPTION	COMMENTS
^[a-zA-Z0-9_.-]{9}$	object	false			



1.1.3.6.2 ^[a-zA-Z0-9_.-]{9}$ Hierarchy

Parent field: machines_list
Child field(s):



1.1.3.20.2 CNt3P1z8ZhVqVHelKjZ9N6pRUxg1 Hierarchy

Parent field: data
Child field(s):

3.1.2.4.4.1.2 typeId properties

PROPERTY	VALUE
Name	typeId
Technical name	
Id	
Type	string
Description	
Dependencies	
Required	true
Primary key	false
Foreign collection	
Foreign field	
Relationship type	
Relationship name	
Cardinality	
Default	
Min length	
Max length	
Pattern	
Format	
Enum	
Faker function	
Sample	
Comments	
Other Attribute	

3.1.2.4.5 vending_machines-1653138687 JSON Schema


```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "title": "vending_machines-1653138687",
  "definitions": {
    "typeId": {
      "type": "string"
    }
  },
  "additionalProperties": true,
  "required": [
    "^[a-zA-Z0-9_.-]{20}$"
  ],
  "patternProperties": {
    "^[a-zA-Z0-9_.-]{20}$": {
      "type": "object",
      "properties": {
        "data": {
          "type": "object",
          "properties": {
```

```
{
  "^[a-zA-Z0-9_.-]{20}$": {},
  "data": {
    "Y4T1GSCzlbxwi3izs4W3": {
      "nameType": "beverages",
      "__collections__": {}
    }
  }
}
```


4. RELATIONSHIPS

4.1 Relationship fk products-1653143913. to vending_machines-1653138687.

4.1.1 fk products-1653143913. to vending_machines-1653138687. Diagram

PARENT TABLE	PARENT FIELD
products-1653143913	data.beverages.__collections__.category_products.QpgLEYEbHdcGfAgcLTni.id

CHILD TABLE	CHILD FIELD
vending_machines-1653138687	^[a-zA-Z0-9_.-]{20}$.[-1].[-1].[-1].[-1].[-1].product.[-1].[-1].product.[-1], ^[a-zA-Z0-9_.-]{20}$.[-1].[-1].[-1].[-1].[-1].product.[-1].[-1].product

4.1.2 fk products-1653143913. to vending_machines-1653138687. Properties

PROPERTY	VALUE
Name	fk products-1653143913. to vending_machines-1653138687.
Description	
Parent Collection	products-1653143913
Parent field	id
Parent Cardinality	1
Child Collection	vending_machines-1653138687
Child field	, product
Child Cardinality	1
Comments	

4.2 Relationship fk providers-1653144996. to products-1653143913.

ДОДАТОК Б

ТЕКСТ ПРОГРАМИ СКРИПТУ ДЛЯ ПЕРЕДАЧІ ДАНИХ ДО FIREBASE REALTIME DATABASE

482.IoT.КНУШ.211297-01

Листів 5

Розробник Демчук А. Б.

Керівник Пономаренко Р. М.

Київ – 2022

```
from firebase_admin import credentials
import os, re, time, json, argparse, signal, threading
import urllib.parse
from queue import Queue, Empty
import requests
import paho.mqtt.client as mqtt # pip install paho-mqtt
from google.oauth2 import service_account
from google.auth.transport.requests import AuthorizedSession

verbose = False
NOTHING_TO_DO_DELAY = 5
FIREBASE_TIMEOUT  = 7
FIREBASE_MAX_RETRY  = 2
FIREBASE_BASE_URL = 'https://vendbmr-default-rtdb.firebaseio.com'
def signal_handler(signal, frame):
  print('You pressed Ctrl+C!')
  stop_event.set()
  t1.join()
  client.disconnect()
def debug(msg):
  if verbose:
    print (msg + "\n")

def environ_or_required(key):
  if os.environ.get(key):
    return {'default': os.environ.get(key)}
  else:
    return {'required': True}

def process_firebase_messages(lqueue, stop_event):
  firebaseSession = AuthorizedSession(credentials)
```

```
baseUrl = FIREBASE_BASE_URL


while not stop_event.is_set():
 try:
  packet = lqueue.get(False)
 except Empty:
  time.sleep(NOTHING_TO_DO_DELAY)
  pass
 else:
  if packet is None:
   continue
  debug("data from queue: " + format(packet))
  firebasePath = packet['config']['firebasePath']
  if packet['config']['topicAsChild']:
   firebasePath = urllib.parse.urljoin(packet['config']['firebasePath'] + '/', packet['topic'])
  firebasePath = baseUrl + '/' + firebasePath + '.json'
  debug ("Sending {0} to this URL {1}".format(packet['payload'], firebasePath))
  retry = 0
  while True:
   try:
    if not args.dryRun:
     r = firebaseSession.post(firebasePath, json=packet['payload'], timeout=FIREBASE_TIMEOUT)
     debug ("payload inserted : " + r.text)
    except requests.exceptions.Timeout:
     print ("Firebase Timeout")
     if retry < FIREBASE_MAX_RETRY:
      retry += 1
      debug ("Retrying")
      time.sleep(NOTHING_TO_DO_DELAY)
      continue
    except requests.exceptions.RequestException as e:
```

```
            print ("Firebase Exception" + str(e))
          except:
            print ("Firebase Unknown Exception")
          break
        queue.task_done()
      firebaseSession.close()
      debug("Stopping Firebase Thread ...")

    def on_connect(client, userdata, flags, rc):
      for topic in topics:
        client.subscribe(topic['mqttTopic'] +'/#', qos=1)

    def on_disconnect(client, userdata, rc):
      debug("Disconnected with result code "+str(rc))
      if rc != 0:
        debug("Unexpected disconnection.")

    def on_message(client, userdata, msg):
      print(msg.topic)
      for topic in topics:
        productName = msg.topic.split('/') [-2] + '/'+ msg.topic.split('/') [-1]
        debug("Received message from {0} matching {3} with payload {1} to be published to {2}".format(msg.topic, str(msg.payload), productName, topic['mqttTopic']))
        nodeData = msg.payload.decode('utf-8')
        payload = "Received message from {0} matching {3} with payload {1} to be published to {2}".format(msg.topic, str(msg.payload), productName, topic['mqttTopic'])
        print(nodeData)
        dict1 = {msg.topic.split('/') [-1]: nodeData}
        dict2 = {msg.topic: topic['mqttTopic']}
        newObject = json.dumps([dict1, dict2])
        jsn = {"number": nodeData}
```

```
        queue.put({
          "topic": msg.topic,
          "payload": jsn,
          "config": topic
        })
parser = argparse.ArgumentParser(description='Send MQTT payload received from a topic to firebase.',
  formatter_class=argparse.ArgumentDefaultsHelpFormatter)
signal.signal(signal.SIGINT, signal_handler)
signal.signal(signal.SIGTERM, signal_handler)
args = parser.parse_args()
verbose = args.verbose
pathOrCredentials = args.firebaseApiKey
isFile = True
if (pathOrCredentials.startswith ('{')):
  isFile = False
  pathOrCredentials = json.loads(pathOrCredentials)
# Define the required scopes
scopes = [
  "https://www.googleapis.com/auth/userinfo.email",
  "https://www.googleapis.com/auth/firebase.database"
]

topics = []
for topic in args.topics:
  newTopic = {
    "mqttTopic": topic,
    "firebasePath": '/#',
    "topicAsChild": False
  }
  print(newTopic)
  newTopic["mqttTopicRegex"] = re.compile('^' + newTopic["mqttTopic"].replace('#', '))
```

```
  print(newTopic["mqttTopicRegex"])
  if newTopic["firebasePath"].endswith('/#'):
    newTopic["topicAsChild"] = True
    newTopic["firebasePath"] = newTopic["firebasePath"][:-2]
  topics.append(newTopic)
# Authenticate a credential with the service account
if isFile:
  credentials = service_account.Credentials.from_service_account_file(
      pathOrCredentials, scopes=scopes)
  # = credentials.Certificate("vendbmr-firebase-adminsdk-30urz-ba6f5524e1.json")
  # firebase_admin.initialize_app(credentials)
else:
  credentials = service_account.Credentials.from_service_account_info(
      pathOrCredentials, scopes=scopes)
  # credentials.Certificate("vendbmr-firebase-adminsdk-30urz-ba6f5524e1.json")
queue = Queue()
stop_event = threading.Event()
t1 = threading.Thread(target=process_firebase_messages, args=[queue, stop_event])
t1.daemon = True
t1.start()
client = mqtt.Client()
client.on_connect = on_connect
client.on_disconnect = on_disconnect
client.on_message = on_message
client.username_pw_set("pay-drink", "E85OrQqhDVsOuWFe")
client.connect('pay-drink.cloud.shiftr.io', 1883, 60)
client.loop_forever()
```

ДОДАТОК В

ТЕКСТ ПРОГРАМИ МОБІЛЬНОГО ДОДАТКУ ДЛЯ МОНІТОРИНГУ ТОВАРІВ

482.IoT.КНУШ.211297-01

Листів 5

Розробник Демчук А. Б.

Керівник Пономаренко Р. М.

Київ – 2022

```
class App extends StatefulWidget {
  const App({Key? key}) : super(key: key);
  @override
  State<App> createState() => _AppState();
}
class _AppState extends State<App> {
  @override
  void initState() {
    super.initState();
    _setPreferredOrientations();
    _setSystemUIOverlayStyle();
  }
  @override
  Widget build(BuildContext context) {
    return MultiBlocProvider(
      providers: [
        BlocProvider(
          create: (context) => UserCubit(
            userRepo: locator<UserRepo>(),
          ),
        ),
        BlocProvider(
          create: (context) => AuthCubit(
            authRepo: locator<AuthRepo>(),
          ),
        ),
        BlocProvider(
          create: (context) => VmCubit(
            realDbRepo: locator<RealDbRepo>(),
          ),
        ),
```

```
      BlocProvider(
        create: (context) => ProductCubit(
          productRepo: locator<VmStatisticRepo>(),
        ),
      ),
      BlocProvider(
        create: (context) => AllVmCubit(
          vmRepo: locator<VmRepo>(),
        ),
      ),
    ],
    child: MaterialApp(
      navigatorKey: Catcher.navigatorKey,
      theme: ThemeData(
        brightness: Brightness.light,
        backgroundColor: Colors.white,
        scaffoldBackgroundColor: Colors.white,
      ),
      builder: (context, widget) {
        final scaleFactor =
            MediaQuery.of(context).copyWith(textScaleFactor: 1.0);
        return MediaQuery(
          data: scaleFactor,
          child: widget!,
        );
      },
      home: const SplashScreen(),
    ),
  );
}
void _setPreferredOrientations() {
```

```
    SystemChrome.setPreferredOrientations([
      DeviceOrientation.portraitUp,
      DeviceOrientation.portraitDown,
    ]);
  }
  void _setSystemUIOverlayStyle() {
    SystemChrome.setSystemUIOverlayStyle(
      const SystemUiOverlayStyle(
        statusBarIconBrightness: Brightness.dark,
        statusBarBrightness: Brightness.light,
      ),
    );
  }
}
class RealDbRepoImpl implements RealDbRepo {
  final _database = FirebaseDatabase.instance
    ..setPersistenceCacheSizeBytes(10000000)
    ..setPersistenceEnabled(true);
  @override
  Future<List<Stream<ProductNumber?>>?> getVmProductsInfo(
      {required VmModel vmModel, String? location}) async {
    location = 'kyiv_city/';
    final category = (vmModel.products?.first.category != null ? '/' : '') +  (vmModel.products?.first.category ?? '');
    final beveragesList = vmModel.products?.first.categoryProducts;
    List<Stream<ProductNumber?>>? productsList;
    try {
      if (beveragesList?.isNotEmpty ?? false) {
        productsList = beveragesList!.map((e) {
          if (e.id != null) {
            final Stream record = _database
                .ref(location! + vmModel.id + category + '/' + e.id!).onValue.asBroadcastStream();
```

```
          return record.map<ProductNumber?>((event) {
            print(event.snapshot.value);
            final List? numbers = event.snapshot.children
                .map((number) => ProductNumber(
                    e, int.tryParse((number.value)['number'] ?? '0')))
                .toList();
            print(numbers);
            if (numbers?.isNotEmpty ?? false) {
              return numbers?.last;
            } else {
              return null;
            }
          });
        } else {
          return Stream.value(ProductNumber(e, 0));
        }
      }).toList();
      return productsList;
    }
  } catch (e, s) {
    logError('URealDbRepoImpl::getVmModelInfo:', error: e, stackTrace: s);
    return null;
  }
  return null;
 }
}
class VmStatisticRepoImpl implements VmStatisticRepo {
 final _database = FirebaseDatabase.instance
  ..setPersistenceCacheSizeBytes(10000000)
  ..setPersistenceEnabled(true);
 @override
```

```
Future<List<ProductHistory?>?> getProductsHistory(
    {required VmModel vmModel, String? location}) async {
  location = 'kyiv_city/';
  final category = (vmModel.products?.first.category != null ? '/' : '') +
      (vmModel.products?.first.category ?? '');
  final beveragesList = vmModel.products?.first.categoryProducts;
  List<ProductHistory?>? productsList = [];
  try {
    if (beveragesList?.isNotEmpty ?? false) {
      for (final e in beveragesList!) {
        if (e.id != null) {
          print(e.id);
          final DatabaseEvent record = await _database
              .ref(location + vmModel.id + category + '/' + e.id!)
              .once();
          final List<int> numbers = record.snapshot.children
              .map((number) =>
                  int.tryParse((number.value as Map?)?['number'] ?? '0') ?? 0)
              .toList();
          productsList.add(ProductHistory(e, numbers));
        }
      }
      return productsList;
    }
  } catch (e, s) {
    logError('VmStatisticRepoImpl::getVmModelInfo:', error: e, stackTrace: s);
    return null;
  }
  return null;
 }
}
```

ДОДАТОК Г

ІНСТРУКЦІЯ КОРИСТУВАЧЕВІ MONITOR DRINKERS ІНСТРУКЦІЯ

482.IoT.КНУШ.211297-01

Листів 9

Розробник Демчук А. Б.

Керівник Пономаренко Р. М.

Київ – 2022

АНОТАЦІЯ

Дана інструкція містить відомості про користування мобільним застосунком інформаційної системи «Моніторинг наявності товарів для вендингових апаратів», його налаштування, а також інформацію, необхідну для розуміння функцій розробленої програми і її експлуатації. Керування інформацією про вендингові апарати, кількість товарів, отриману з Realtime Database, кінцевими користувачами системи здійснюється за допомогою натискання необхідних кнопок у вікнах програмних форм, введенням відповідної інформації в поля вводу під час входу або реєстрації. Докладному описові перерахованих питань і присвячене дане керівництво.

ЗМІСТ

РОЗДІЛ 1 СПОСІБ НАЛАШТУВАННЯ ПРОГРАМИ

Даний розділ охоплює кроки запуску мобільного застосунку та входу в систему.

Розроблений мобільний застосунок потрібно встановити на телефон, відповідно до операційної системи. Для Android ОС файл з програмою буде з розширенням apk, а для iOS – ipa. Проте згідно політики Apple, мобільні додатки не можуть бути встановлені на телефон без використання магазину додатків AppStore або програми Xcode, яка дозволяє завантажувати тестову версію скомпільованого застосунку на телефон шляхом його підключення до комп'ютера з операційною системою macOS. Найчастіше розробники використовують програму TestFlight, яка була створена спеціально для тестування iOS додатків, для даного тестування потрібно створити проект на платформі Apple, яка дозволить згенерувати унікальний ідентифікатор даного додатку і відправити ipa файл в TestFlight.

Після встановлення додатку, за допомогою одного з обраних шляхів, потрібно натиснути на іконку, щоб відкрити програму (рис. Г.1.1) .

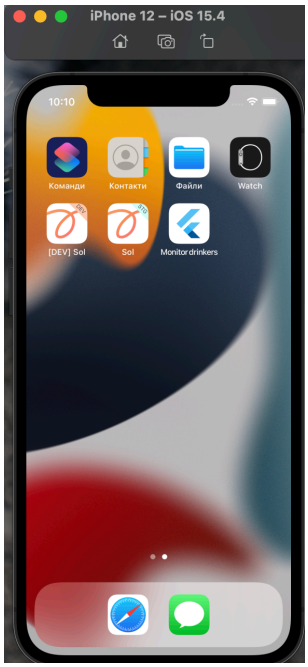


Рисунок Г.1.1 Іконка завантаженої програми на симулятор мобільного пристрою

Щоб виконати вхід в систему потрібно натиснути на кнопку «Continue with google» та ввести відповідний логін і пароль для поштового сервісу Google, який створено для користувачів всередині компанії. Вхід показано нижче (рис. Г.1.2).

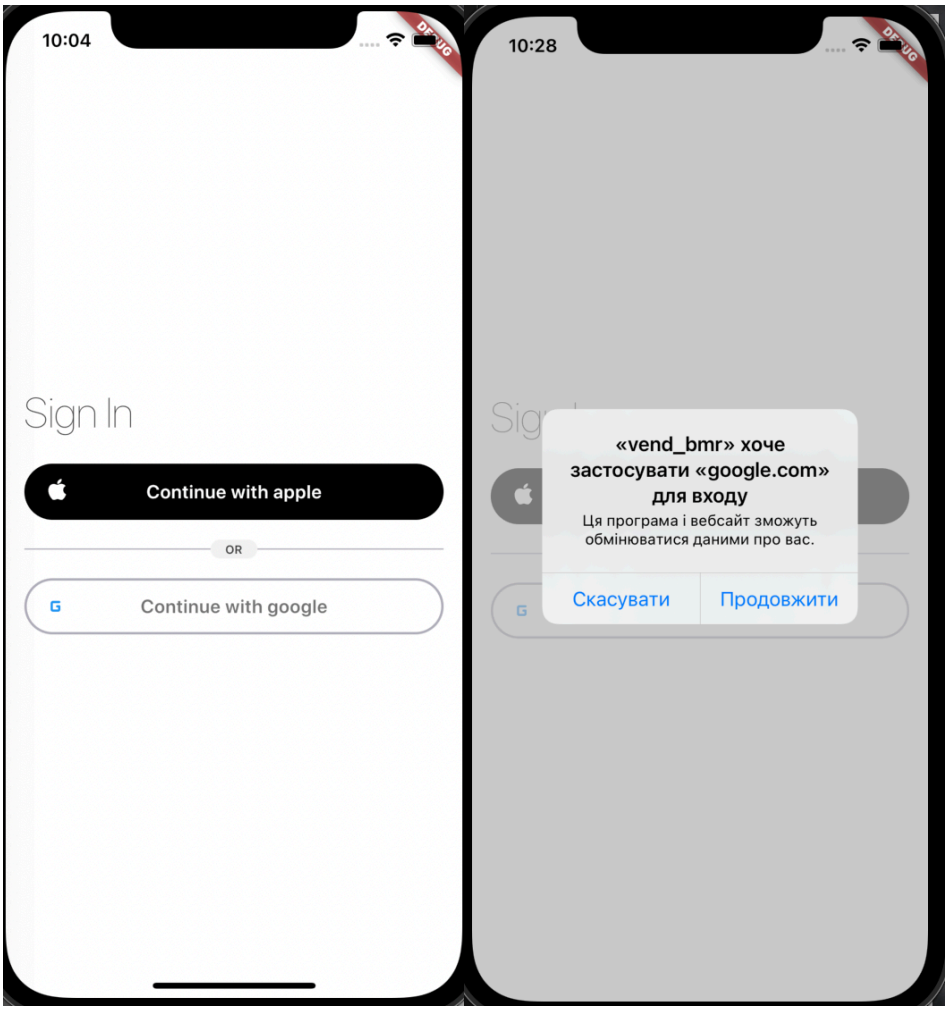


Рисунок Г.1.2 Екран авторизації та запит входу за допомогою сервісу Google

Після введення правильної інформації в дані поля, користувач потрапить на головну сторінку системи (рис. Г.1.3).

Рисунок Г.1.3 Екран зі списком автоматів

Далі усі маніпуляції з вендинговими автоматами проводяться залогіненим користувачем і будуть описані в розділі 2 ІНТЕРФЕЙС ПРОГРАМИ.

РОЗДІЛ 2 ІНТЕРФЕЙС ПРОГРАМИ

Даний розділ містить опис інтерфейсу програми інформаційної системи «Моніторинг наявності товарів для вендингових апаратів».

В залежності від потреб, працівник компанії може обрати будь-який з вендингових автоматів з бази, а також переглянути загальну інформацію та аналітику у вкладці Analytics. Остання вкладка на нижній панелі управління здійснює перехід на екран профілю користувача. Панель управління ІС зображено нижче (рис. Г.2.1).

Рисунок Г.2.1 Ілюстрація панелі керування ІС «Моніторинг наявності товарів для вендингових апаратів»

Для перегляду товарів, користувачу необхідно натиснути на один з автоматів, після цього відкриється листок з інформацією про продукцію. При відсутності товарів буде відображений відповідний напис (рис. Г.2.2 – Г.2.3).

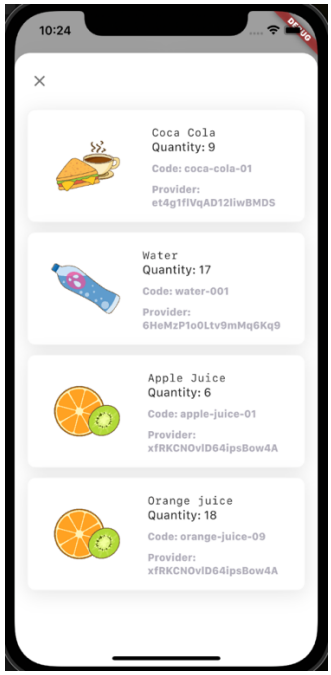


Рисунок Г.2.2 Ілюстрація нижнього листка (bottom sheet) з інформацією про продукцію

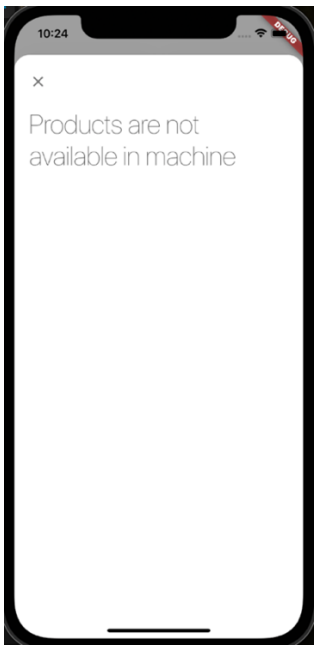


Рисунок Г.2.3 Ілюстрація нижнього листка (bottom sheet) з інформацією про відсутність продукції

Для аналізу наявності і покупок продукції є вкладка Analysis з графіками статистики для всіх машин та окремо для кожного з автоматів. При недостатній кількості даних для аналітики відображається відповідне повідомлення. Вкладка статистики є 2 вкладкою і проілюстрована на рисунку Г.2.4.

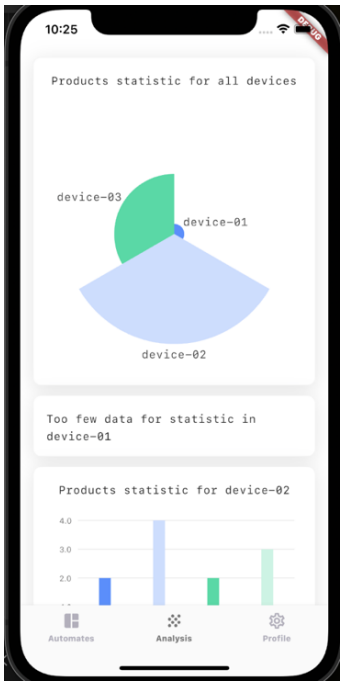


Рисунок Г.2.4 Вкладка статистики вендингових автоматів

Основна інформація про користувача та кнопка виходу з системи знаходяться на вкладці профілю (рис. Г.2.5).

Рисунок Г.2.5 Вкладка профілю користувача

Для виходу з програми необхідно натиснути кнопку Sign out.

ВИСНОВКИ

Даний спроектований інтерфейс для інформаційної системи «Моніторинг наявності товарів для вендингових апаратів» включає всі необхідні компоненти для користування мобільним застосунком, а саме його налаштування, керування інформацією про вендингові апарати, кількість товарів, отриману з Realtime Database, кінцевими користувачами системи, а також генерацію і аналіз сформованої аналітики.

Створений програмний продукт спрямований на те, щоб полегшити адміністраторам, водіям, обслуговуючому персоналу, що працюють в сфері вендингового бізнесу, проєктування маршрутів, логістику та ефективне постачання товарів.

ДОДАТОК Д

ТЕКСТ ПРОГРАМИ МОБІЛЬНОГО ДОДАТКУ ДЛЯ КУПІВЛІ ТОВАРІВ В ВЕНДИНГОВОМУ АПАРАТІ

482.IoT.КНУШ.211297-01

Листів 18

Розробник Демчук А. Б.

Керівник Пономаренко Р. М.

Київ – 2022

```
class ScannerPage extends StatefulWidget {
  const ScannerPage({
    Key? key,
    this.callback,
  }) : super(key: key);
  final Function(String)? callback;
  @override
  ScannerPageState createState() => ScannerPageState();
}
class ScannerPageState extends State<ScannerPage> {
  late final QrCubit qrCubit;
  QRViewController? controller;
  String? phone;
  final BehaviorSubject<bool?> _flashLightState = BehaviorSubject<bool?>();
  final TextEditingController _searchTextController = TextEditingController();
  final GlobalKey qrKey = GlobalKey(debugLabel: 'QR');
  final GlobalKey<ScaffoldState> _scaffoldKey = GlobalKey<ScaffoldState>();
  FocusNode focusNode = FocusNode();
  bool _isManualEntering = false;
  @override
  void initState() {
    Wakelock.toggle(enable: false);
    qrCubit = context.read();
    super.initState();
  }
  @override
  void dispose() {
    Wakelock.toggle(enable: true);
    controller?.dispose();
    qrCubit.dispose();
    super.dispose();
```

```
}
@override
Widget build(BuildContext context) {
 return AnnotatedRegion<SystemUiOverlayStyle>(
  value: const SystemUiOverlayStyle(
   // For Android.
   // Use [light] for white status bar and [dark] for black status bar.
   statusBarIconBrightness: Brightness.dark,
   // For iOS.
   // Use [dark] for white status bar and [light] for black status bar.
   statusBarBrightness: Brightness.dark,
  ),
  child: Scaffold(
   key: _scaffoldKey,
   floatingActionButtonLocation: FloatingActionButtonLocation.startTop,
   resizeToAvoidBottomInset: false,
   backgroundColor: Colors.black,
   body: BlocConsumer<QrCubit, QrState>(
    bloc: qrCubit,
    listener: (BuildContext context, QrState qrState) {
     if (qrState.deviceInfo != null &&
       !qrState.isLoadingDeviceInfoString) {
      // if (qrState is StartFetchingVmModel) {
      controller?.dispose();
      return NavigationUtil.toScreenAndCleanBackstack(
        context: context,
        screen: VMScreen(deviceInfo: qrState.deviceInfo!));
     }
    },
    builder: (BuildContext context, QrState qrState) {
     return Stack(
```

```
fit: StackFit.expand,
children: [
  _buildQrView(qrState),
  _colorFilter(),
  SizedBox(
    width: MediaQuery.of(context).size.width,
    height: MediaQuery.of(context).size.height,
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.center,
      mainAxisAlignment: MainAxisAlignment.end,
      children: [
        Padding(
          padding: EdgeInsets.only(
              bottom: _isManualEntering
                  ? MediaQuery.of(context).viewInsets.bottom + 20
                  : 80.0),
          child: Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Stack(children: [
                AnimatedOpacity(
                    curve: Curves.easeInOut,
                    duration: const Duration(milliseconds: 190),
                    opacity: _isManualEntering ? 1.0 : 0.0,
                    child: AnimatedContainer(
                        curve: Curves.easeInOut,
                        duration:
                            const Duration(milliseconds: 200),
                        height: 50.0,
                        width: _isManualEntering
                            ? MediaQuery.of(context).size.width <
```

```
          400
        ? 200
        : 250.0
    : 180.0,
child: TextField(
  maxLines: 1,
  textAlign: TextAlign.center,
  keyboardType:
      /* ? TextInputType.number
                : */
      const TextInputType
            .numberWithOptions(
          decimal: true),
  focusNode: focusNode,
  onChanged: (string) {},
  onEditingComplete: () {
   FocusScope.of(context)
       .requestFocus(FocusNode());
  },
  style: const TextStyle(
      color: Colors.black87),
  decoration: const InputDecoration(
      contentPadding:
          EdgeInsets.symmetric(
              vertical: 1.0,
              horizontal: 12.0),
      filled: true,
      fillColor: Colors.white,
      border: OutlineInputBorder(
          borderSide: BorderSide.none,
          borderRadius: BorderRadius.all(
```

```
                    Radius.circular(25.0)))),
          controller: _searchTextController,
        ))),
AnimatedOpacity(
  curve: Curves.easeInOut,
  duration: const Duration(milliseconds: 200),
  opacity: _isManualEntering ? 0.0 : 1.0,
  child: AnimatedContainer(
    curve: Curves.easeInOut,
    duration: const Duration(milliseconds: 200),
    height: 50.0,
    width: _isManualEntering
        ? MediaQuery.of(context).size.width < 400
            ? 200
            : 250.0
        : 180.0,
    child: GestureDetector(
      onTap: () {
        try {
          controller!
              .getFlashStatus()
              .then((value) {
            if (value!) {
              controller!.toggleFlash();
              controller!.getFlashStatus().then(
                  (value) => _flashLightState
                      .add(value));
            }
          });
        } catch (e) {
          print(e.toString());
```

```
  }
  if (!_isManualEntering) {
    _isManualEntering = !_isManualEntering;
    focusNode.requestFocus();
  }
},
child: StreamBuilder(
  stream: _flashLightState.stream,
  builder: (BuildContext context,
      AsyncSnapshot<bool?> snapshot) {
    if (snapshot.hasData) {
      return GestureDetector(
          onTap: () {
            if (!_isManualEntering) {
              try {
                controller!.toggleFlash();
                controller!
                    .getFlashStatus()
                    .then((value) =>
                        _flashLightState
                            .add(value));
              } catch (e) {
                print(e.toString());
              }
            } else {
              if (_isManualEntering) {
                _isManualEntering =
                    !_isManualEntering;
                FocusScope.of(context)
                    .requestFocus(
                        FocusNode());
```

```
    }
    final trimmedString =
        _searchTextController
            .text;
  }
},
child: Container(
  child: Center(
     child: Icon(
    _isManualEntering
        ? Icons.done
        : Icons.highlight,
    color: _isManualEntering
        ? AppColors.milkWhite
        : (snapshot.data !=
                null &&
            snapshot.data!
              ? AppColors.uiDarkGrey
              : AppColors
                .milkWhite),
    size: _isManualEntering
        ? 30.0
        : 25.0,
  )),
  decoration: BoxDecoration(
    borderRadius:
        const BorderRadius.all(
          Radius.circular(
              30.0)),
    color: _isManualEntering
        ? AppColors.uiDarkGrey
```

```
                : (snapshot.data !=
                        null &&
                    snapshot.data!
                ? AppColors.milkWhite
                : AppColors
                    .uiDarkGrey),
          ),
          height: _isManualEntering
              ? 60.0
              : 50.0,
          width: _isManualEntering
              ? 60.0
              : 50.0,
        ));
  } else {
    return GestureDetector(
      onTap: () async {
        if (!_isManualEntering) {
          if (controller != null) {
            try {
              controller!.toggleFlash();
              controller!
                  .getFlashStatus()
                  .then((value) =>
                      _flashLightState
                          .add(value));
            } catch (e) {
              print(e.toString());
            }
          }
        } else {
```

```
    if (_isManualEntering) {
      _isManualEntering =
          !_isManualEntering;
      FocusScope.of(context)
          .requestFocus(
              FocusNode());
    }
    final trimmedString =
        _searchTextController.text;
  }
},
child: Container(
  child: Center(
      child: Icon(
    _isManualEntering
        ? Icons.done
        : Icons.highlight,
    color: AppColors.milkWhite,
    size: _isManualEntering
        ? 30.0
        : 25.0,
  )),
  decoration: BoxDecoration(
      borderRadius:
          BorderRadius.all(
              Radius.circular(
                  _isManualEntering
                      ? 30.0
                      : 25.0)),
      color: AppColors.uiDarkGrey),
  height: _isManualEntering
```

```
                        ? 60.0
                        : 50.0,
                    width: _isManualEntering
                        ? 60.0
                        : 50.0,
                  ),
                );
              }
            },
          ),
        ),
      ),
    ),
  ],
 ),
),
AnimatedPositioned(
 curve: Curves.easeInOut,
 left: (MediaQuery.of(context).size.width / 2.0) - 100.0,
 duration: const Duration(milliseconds: 250),
 top: _isManualEntering
    ? (MediaQuery.of(context).size.height / 2.0 - 200.0)
    : -100.0,
 child: AnimatedOpacity(
  opacity: _isManualEntering ? 1.0 : 0.0,
  duration: const Duration(milliseconds: 200),
  curve: Curves.easeIn,
```

```
child: GestureDetector(
  onTap: () {
    if (_isManualEntering) {
      _isManualEntering = !_isManualEntering;
      FocusScope.of(context).requestFocus(FocusNode());
    }
  },
  child: Stack(
    children: [
      Container(
          width: 200.0,
          height: 200.0,
          decoration: const ShapeDecoration(
            shape: OverlayShape(
                borderColor: Colors.white,
                borderRadius: 14.0,
                borderLength: 20.0,
                borderWidth: 5.0,
                cutOutSize: 80.0),
          )),
      Positioned(
        top: 90.0,
        left: intl.Intl.getCurrentLocale() == 'ua'
            ? 34.0
            : 46,
        child: Text(
          'scannerPageContinueScanningMessage'
              .toUpperCase(),
          style: TextStyle(
              fontSize: 16.0,
              color: Colors.white.withOpacity(0.6),
```

```
                        fontWeight: FontWeight.w600),
                  ),
                ),
              ],
            ),
          ),
        ),
      ),
    ],
  );
 },
 ),
 ),
 );
}
Widget _buildQrView(QrState state) {
 var scanArea = (MediaQuery.of(context).size.width < 400 ||
     MediaQuery.of(context).size.height < 400)
   ? MediaQuery.of(context).size.width - 50
   : MediaQuery.of(context).size.width - 50;
 return QRView(
  key: qrKey,
  onQRViewCreated: (qrController) {
   controller = qrController;
   controller?.scannedDataStream.listen((scanData) {
    final currentScan = DateTime.now();
    if (state.canScan &&
      (state.lastScan == null ||
        currentScan.difference(state.lastScan!) >
          const Duration(seconds: 3))) {
     qrCubit.readScan(scanData, currentScan);
```

```
        }
      });
    },
    overlay: QrScannerOverlayShape(
        overlayColor: Colors.black.withOpacity(_isManualEntering ? 0 : 0.8),
        borderColor: Colors.white,
        borderRadius: 14.0,
        borderLength: 38.0,
        borderWidth: 5.0,
        cutOutSize: scanArea),
  );
}
Widget _menu() {
  return FloatingActionButton(
    backgroundColor: AppColors.uiDarkGrey,
    child: Icon(
      ModalRoute.of(context)!.isFirst ? Icons.person : Icons.close,
      color: AppColors.uiPaleWarmGrey,
    ),
    elevation: 8,
    onPressed: () {
      _isManualEntering = false;
      _searchTextController.text = '';
      FocusScope.of(context).requestFocus(FocusNode());
      NavigationUtil.toScreen(context: context, screen: ProfileScreen());
    },
  );
}
void showDemoDialog<T>({required BuildContext context, Widget? child}) {
  showDialog<T>(
    context: context,
```

```
      builder: (BuildContext context) => child!,
    );
  }
 ColorFiltered _colorFilter() {
   return ColorFiltered(
     colorFilter: ColorFilter.mode(
       Colors.black.withOpacity(_isManualEntering ? 0.8 : 0),
       BlendMode.srcOut,
     ), // This one will create the magic
     child: Stack(
       fit: StackFit.expand,
       children: [
         GestureDetector(
           onTap: () {
             if (_isManualEntering) {
               _isManualEntering = !_isManualEntering;
               FocusScope.of(context).requestFocus(FocusNode());
             }
           },
           child: Container(
             decoration: const BoxDecoration(
               color: Colors.white,
               backgroundBlendMode: BlendMode.dstOut,
             ),
           ),
         ),
       ],
     ),
   );
  }
}
```

```
class OverlayShape extends ShapeBorder {
  const OverlayShape({
    this.borderColor = Colors.red,
    this.borderWidth = 3.0,
    this.overlayColor = const Color.fromRGBO(0, 0, 0, 60),
    this.borderRadius = 0,
    this.borderLength = 40,
    this.cutOutSize = 250,
    this.cutOutBottomOffset = 0,
  }) : assert(borderLength <= cutOutSize / 2 + borderWidth * 2,
          "Border can't be larger than ${cutOutSize / 2 + borderWidth * 2}");
  final Color borderColor;
  final double borderWidth;
  final Color overlayColor;
  final double borderRadius;
  final double borderLength;
  final double cutOutSize;
  final double cutOutBottomOffset;

  @override
  EdgeInsetsGeometry get dimensions => const EdgeInsets.all(10);

  @override
  Path getInnerPath(Rect rect, {TextDirection? textDirection}) {
    return Path()
      ..fillType = PathFillType.evenOdd
      ..addPath(getOuterPath(rect), Offset.zero);
  }

  @override
  Path getOuterPath(Rect rect, {TextDirection? textDirection}) {
```

```
  Path _getLeftTopPath(Rect rect) {
    return Path()
      ..moveTo(rect.left, rect.bottom)
      ..lineTo(rect.left, rect.top)
      ..lineTo(rect.right, rect.top);
  }
  return _getLeftTopPath(rect)
    ..lineTo(
      rect.right,
      rect.bottom,
    )
    ..lineTo(
      rect.left,
      rect.bottom,
    )
    ..lineTo(
      rect.left,
      rect.top,
    );
}
@override
void paint(Canvas canvas, Rect rect, {TextDirection? textDirection}) {
  final width = rect.width;
  final borderWidthSize = width / 2;
  final height = rect.height;
  final borderOffset = borderWidth / 2;
  final _borderLength = borderLength > cutOutSize / 2 + borderWidth * 2
      ? borderWidthSize / 2
      : borderLength;
  final _cutOutSize = cutOutSize < width ? cutOutSize : width - borderOffset;
```

```
final backgroundPaint = Paint()
  ..color = overlayColor
  ..style = PaintingStyle.fill;
final borderPaint = Paint()
  ..color = borderColor.withOpacity(0.6)
  ..style = PaintingStyle.stroke
  ..strokeWidth = borderWidth;
final boxPaint = Paint()
  ..color = borderColor
  ..style = PaintingStyle.fill
  ..blendMode = BlendMode.dstOut;
final cutOutRect = Rect.fromLTWH(
  rect.left + width / 2 - _cutOutSize / 2 + borderOffset,
  -cutOutBottomOffset +
      rect.top +
      height / 2 -
      _cutOutSize / 2 +
      borderOffset,
  _cutOutSize - borderOffset * 2,
  _cutOutSize - borderOffset * 2,
);

canvas
  ..saveLayer(
    rect,
    backgroundPaint,
  )
  ..drawRRect(
    RRect.fromLTRBAndCorners(
      cutOutRect.right - _borderLength,
      cutOutRect.top,
```

```
    cutOutRect.right,
    cutOutRect.top + _borderLength,
    topRight: Radius.circular(borderRadius),
  ),
  borderPaint,
)
// Draw top left corner
..drawRRect(
  RRect.fromLTRBAndCorners(
    cutOutRect.left,
    cutOutRect.top,
    cutOutRect.left + _borderLength,
    cutOutRect.top + _borderLength,
    topLeft: Radius.circular(borderRadius),
  ),
  borderPaint,
)
// Draw bottom right corner
..drawRRect(
  RRect.fromLTRBAndCorners(
    cutOutRect.right - _borderLength,
    cutOutRect.bottom - _borderLength,
    cutOutRect.right,
    cutOutRect.bottom,
    bottomRight: Radius.circular(borderRadius),
  ),
  borderPaint,
)
// Draw bottom left corner
..drawRRect(
  RRect.fromLTRBAndCorners(
```

```
          cutOutRect.left,
          cutOutRect.bottom - _borderLength,
          cutOutRect.left + _borderLength,
          cutOutRect.bottom,
          bottomLeft: Radius.circular(borderRadius),
        ),
        borderPaint,
      )
      ..drawRRect(
        RRect.fromRectAndRadius(
          cutOutRect,
          Radius.circular(borderRadius),
        ),
        boxPaint,
      )
      ..restore();
  }
  @override
  ShapeBorder scale(double t) {
    return QrScannerOverlayShape(
      borderColor: borderColor,
      borderWidth: borderWidth,
      overlayColor: overlayColor,
    );
  }
}
```

ДОДАТОК Е

КОРИСТУВАЧЕВІ PAY&DRINK

482.IoT.КНУШ.211297-01

Листів 8

Розробник Демчук А. Б.

Керівник Пономаренко Р. М.

Київ – 2022

АНОТАЦІЯ

Дана інструкція містить відомості про користування мобільним застосунком для купівлі товарів в розробленій IoT системі, взаємодії з вендинговим апаратом, а також інформацію, необхідну для розуміння функцій розробленої програми і її експлуатації. Зчитування ID вендингової машини, вибір товару, його кількості та оплата продукту кінцевими користувачами системи здійснюється за допомогою натискання необхідних кнопок у вікнах програмних форм, введенням відповідної інформації в поля вводу. Докладному описові перерахованих питань і присвячене дане керівництво.

ЗМІСТ

РОЗДІЛ 1 СПОСІБ ЗАПУСКУ ПРОГРАМИ

Даний розділ охоплює кроки запуску мобільного застосунку та початку роботи в системі.

Розроблений мобільний застосунок потрібно встановити на телефон, відповідно до операційної системи. Для Android ОС файл з програмою буде з розширенням apk. Згідно політики Apple, мобільні додатки не можуть бути встановлені на телефон без використання магазину додатків AppStore або програми Xcode, яка дозволяє завантажувати тестову версію скомпільованого застосунку на телефон шляхом його підключення до комп'ютера з операційною системою macOS.

Після встановлення додатку, за допомогою одного з обраних шляхів, потрібно натиснути на іконку, щоб відкрити програму (рис. Е.1.1) .

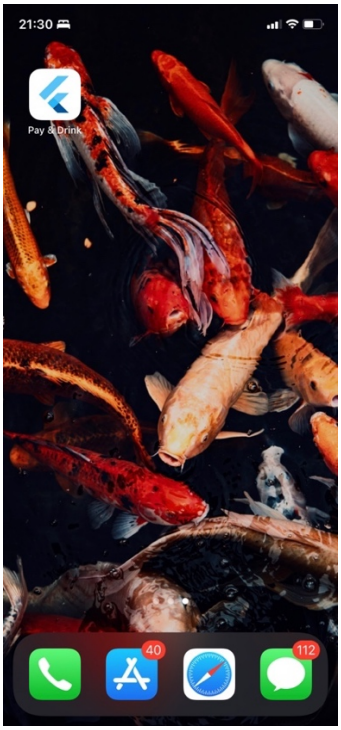


Рисунок Е.1.1 Іконка завантаженої програми Pay&Drink

Після цього користувач потрапить на головну сторінку системи, яка буде екраном сканування QR-коду. Далі взаємодія користувача з вендинговим автоматом буде описана в розділі 2 ІНТЕРФЕЙС ПРОГРАМИ.

РОЗДІЛ 2 ІНТЕРФЕЙС ПРОГРАМИ

Даний розділ містить опис інтерфейсу програми для вибору і купівлі товарів у вендинговому апараті розробленої IoT системи.

Головний екран програми – сканер QR-коду, наводимо на згенероване зображення (рис. Е.2.1). Після верифікації автомату користувач буде спрямований на екран з відповідним автоматом. Приклади автоматів, які користувач може побачити зображено нижче (рис. Е.2.2).

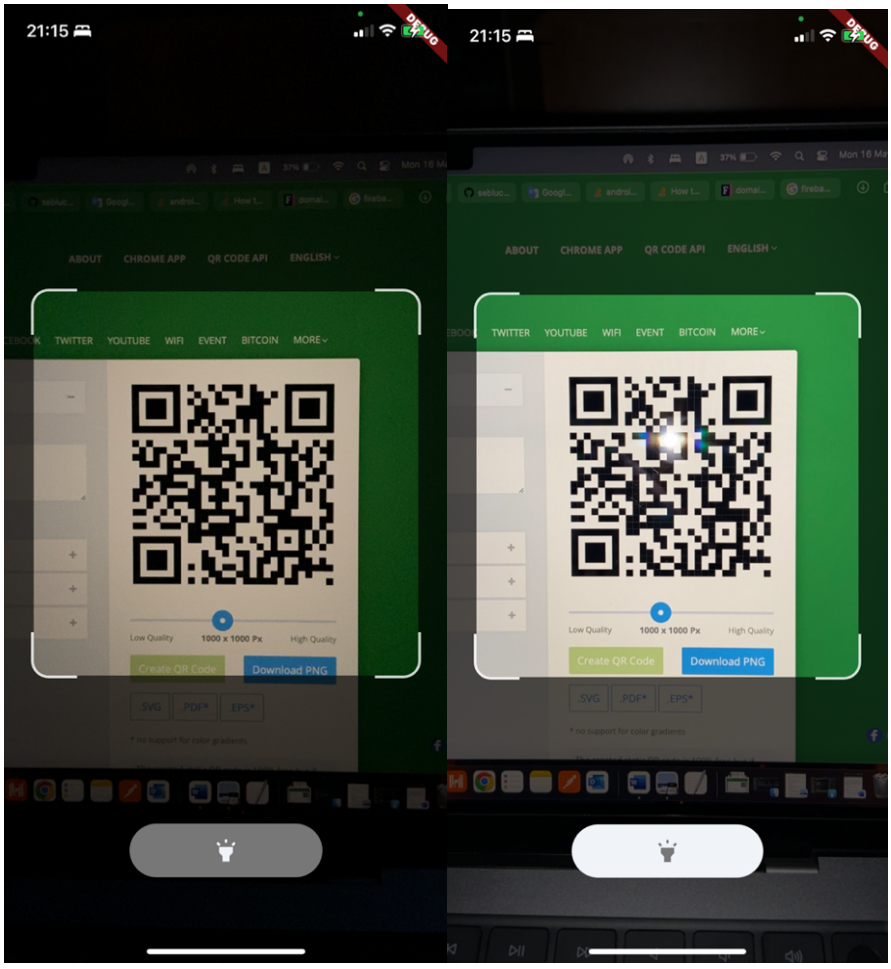


Рисунок Е.2.1 Сканування QR-коду програми Pay&Drink

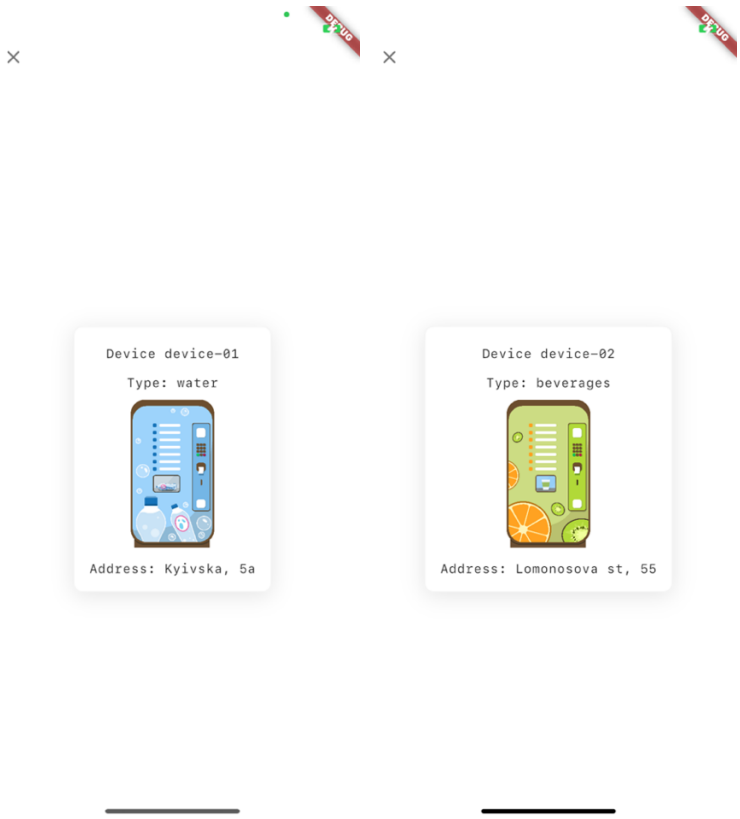


Рисунок Е.2.2 Екрани з даними про відсканові торгові автомати

Щоб обрати товар, потрібно натиснути на зображення автомату, після чого користувач потрапить на сторінку з інформацією про продукцію (рис.Е.2.3).

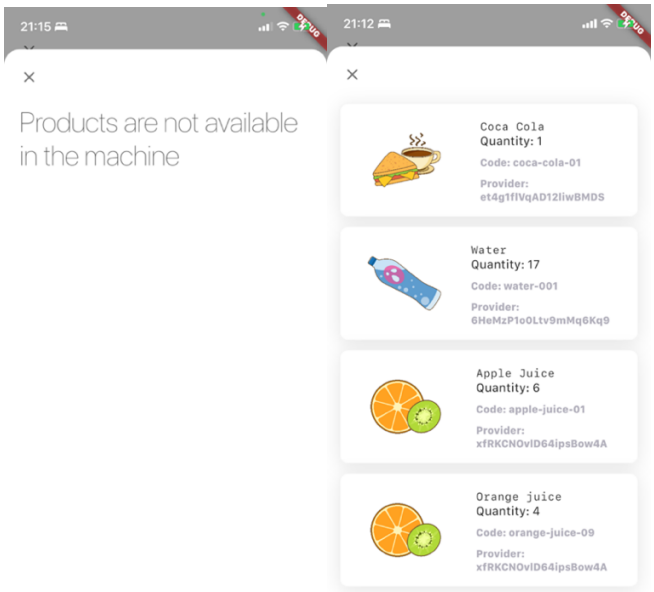


Рисунок Е.2.3 Можливі екрани з інформацією про продукцію

Далі потрібно вибрати кількість товару та провести симуляцію оплати (рис. Е.2.4).

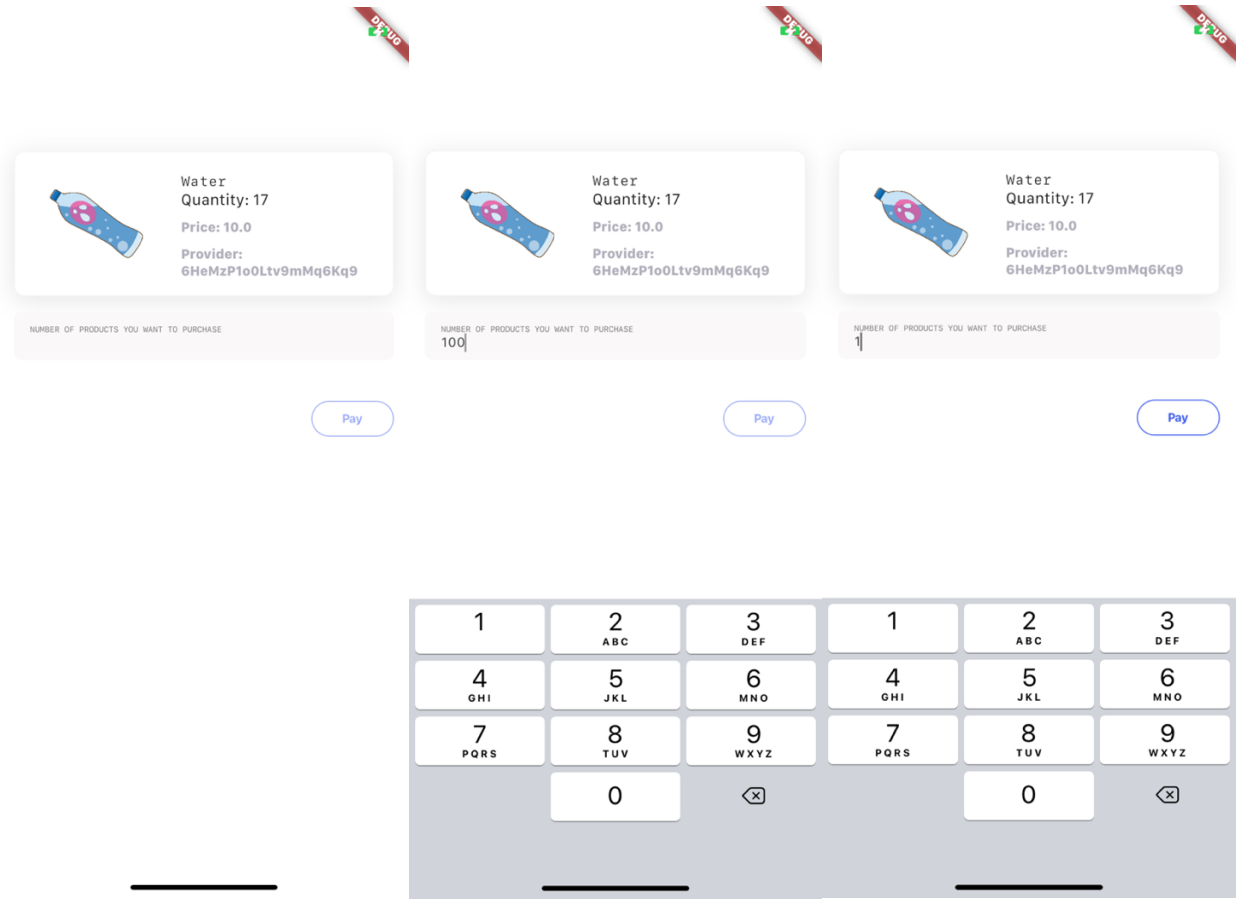


Рисунок Е.2.4 Екрани для вибору кількості продукції та оплату

Після завершення симуляції оплати та відправки повідомлень до БД, користувач буде спрямований на екран подяки, який в подальшому може містити оцінку сервісу або відгук користувача (рис. Е.2.5).

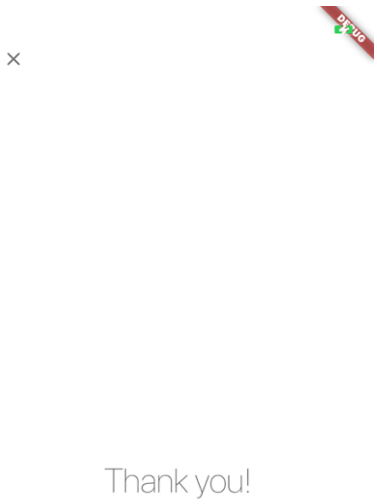


Рисунок Е.2.5 Екран після завершення оплати

Для повернення на екран сканування QR-коду необхідно натиснути кнопку закрити – «Х» в лівому верхньому кутку.

ВИСНОВКИ

Даний спроектований інтерфейс для купівлі товарів в розробленій IoT системі включає всі необхідні компоненти для користування мобільним застосунком, а саме зчитування ID вендингової машини, вибір товару, його кількості та оплата продукту кінцевими користувачами системи.

Створений програмний продукт спрямований на те, щоб забезпечити якісну взаємодію клієнтів з автоматами компанії та полегшити процес відслідковування купівлі товарів, створення маркетингової кампанії, завдяки чому можливе створення поля лояльних користувачів в майбутньому.

ДОДАТОК Є

ОСНОВНІ СЛАЙДИ ПРЕЗЕНТАЦІЇ

482.IoT.КНУШ.211297-01

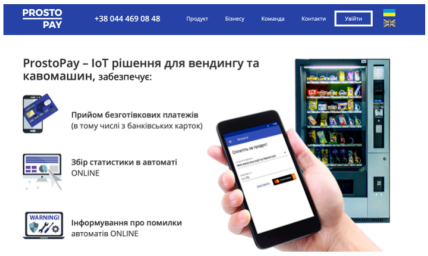
Листів 1

Розробник Демчук А. Б.

Керівник Пономаренко Р. М.

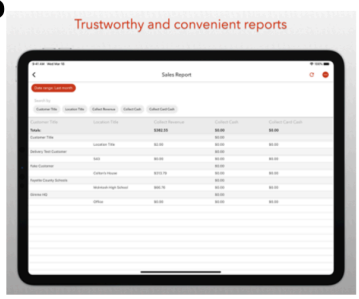
Київ – 2022

Огляд готових рішень



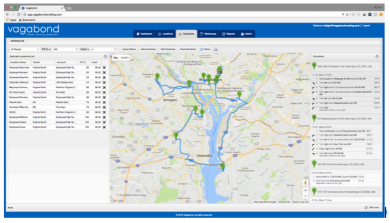
1.4.2 Gimme Vending

Система Gimme Vending – це готове рішення для автоматизованого управління замовленнями, інвентаризацією, рахунками-фактурами на доставку, прокладання маршрутів для водіїв і т.п., тобто для усього, що раніше розраховувалось вручну, за допомогою «олівця і ручки». Тому продукція Gimme Vending Inc широко розповсюджена в США, а її клієнти – невеликі компанії, що займаються роздрібною торгівлею з використанням вендингових машин та потребують автоматизації вище описаних процесів.



1.4.3 ProstoPay

ProstoPay – українська компанія, що обслуговує вендингові апарати та кавомашини [13]. Організація володіє нативним програмним забезпеченням під Android та iOS, які дозволяють проводити безготівкові розрахунки у торгових автоматах. Реалізація IoT-системи існує за рахунок датчиків телеметрії та власної розробки API, яке отримує дані від встановлених телеметрів за допомогою протоколу MQTT. MQTT - це стандартний протокол обміну повідомленнями OASIS для Інтернету речей (IoT). Він розроблений як надзвичайно легкий протокол обміну повідомленнями для публікації / підписки, який ідеально підходить для підключення віддалених пристроїв з невеликим розміром коду та мінімальною пропускною здатністю мережі [14]. Дане рішення є дешевим і найкраще підходить для малого бізнесу і невеликих вендингових компаній, що мають на меті реалізувати технологію «смарт вендингу».



1.4.1 Vagabond

Vagabond – міжнародна компанія, яка займається виготовленням вендингових машин для Північної і Центральної Америки [13]. IoT-мережа Vagabond дозволяє операторам торгових автоматів віддалено побачити, коли запасів продукції мало, а також те, як швидко продаються товари, з подальшою організацією послуги їх поповнення при необхідності. Датчики забезпечують передачу даних в режимі реального часу, а також можуть виявляти несправності, що дозволяє відремонтувати зламані машини якомога швидше. Vagabond використовує API Google Maps Javascript для веб-додатку (рис. 1.5), який відображає свої торгові автомати у вигляді шпильок на карті, для появи інформації про машину потрібно просто натиснути на неї, отримані дані також включають запаси і рівень готівки в автоматі [14].

Тестування

Запуск мобільного додатку для моніторингу

Для тестування мобільного застосунку моніторингу товарів було використано симулятор мобільного пристрою від програми Xcode (рис. 4.11).

Рис. 4.11 Іконка завантаженої програми на симулятор мобільного пристрою

Оновлені дані про кількість продукції, синхронізовані з Firebase Realtime Database зображено на рис. 4.12

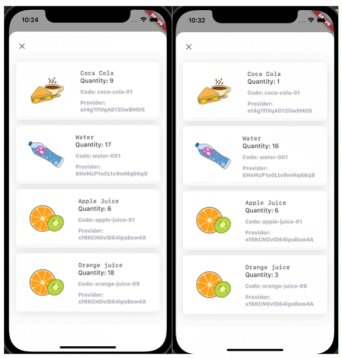


Рис. 4.12 Оновлені дані, синхронізовані з Firebase Realtime Database

Рис. 4.13 Оновлені статистика на основі даних, синхронізованих з Firebase Realtime Database