МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри

_____ Литвиненко О.Є.

«_____»_____20__ р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО–КВАЛІФІКАЦІЙНОГО РІВНЯ «БАКАЛАВР»

Тема: Система «Розумний дім». Підсистема енергозабезпечення

Виконавець: студент групи СП-501Бз Костенко Б.В.

Керівник: к.т.н., доц. Масловський Б. Г.

Нормоконтролер: _____ Тупота Є.В.
(підпис) (П.І.Б.)

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Спеціальність 123 «Комп'ютерна інженерія»

Спеціалізація «Системне програмування»

ЗАТВЕРДЖУЮ

Завідувач кафедри КСУ

Литвиненко О. Є.

«_____» _____ 20___р.

ЗАВДАННЯ

на виконання дипломної роботи (проекту)

Костенко Богдан Валерійович

1. Тема проекту: Система «Розумний дім». Підсистема енергозабезпечення.

затверджена наказом ректора від «21» грудня 2020 р. № 2523 /ст

2. Термін виконання проекту (роботи): з 11.01.2021 р. по 28.02.2021 р.

3. Вихідні дані до проекту (роботи): Сучасні дані про побудову системи "Розумний дім". Інформація про засоби збирання інформації, управління підсистеми та алгоритми роботи підсистеми. Вимоги до оформлення дипломного проекту.

4. Зміст пояснювальної записки: Основні поняття та існуючі варіанти створення системи «Розумний дім». Побудова підсистеми енергозабезпечення. Алгоритми управління підсистемою енергозабезпечення. Програмна реалізація алгоритму управління. Тестування розроблених модулів програми.

5. Перелік обов'язкового ілюстративного матеріалу:

1) структурна схема підсистеми енергозабезпечення;
2) функціональна схема підсистеми енергозабезпечення;
3) алгоритм роботи підсистеми енергозабезпечення;
4) алгоритм збору даних;
5) екранна форма інтерфейсу підсистеми енергозабезпечення.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.			
2.			
3.			
4.			
5.			
6.			
7.			
8.			
9.	Оформлення пояснювальної записки та обов'язкового графічного матеріалу		
10.	Підготовка доповіді та презентації		

7. Дата видачі завдання: «22» грудня 2020 р.

Керівник дипломної роботи (проекту) _____ Масловський Б.Г.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Костенко Б.В.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Дипломний проект присвячений розробці підсистеми енергозабезпечення системи «розумний дім» на основі програмних додатків та системних плат, дана розробка є актуальним завданням.

Пояснювальна записка до дипломного проекту «Система «Розумний дім». Підсистема енергозабезпечення» містить 55 с., 24 рис., 12 літературних джерел, 1 додаток.

РОЗУМНИЙ ДІМ, ЕНЕРГОЗАБЕЗПЕЧЕННЯ, ПЛАГІН, ARDUINO, C#, *DHT22*

Об’єкт дослідження – підсистема енергозабезпечення для обслуговування мешканців будинку.

Предмет дослідження – прототип підсистеми енергозабезпечення в розумному домі, яка включає: базу даних, систему керування та системні плати.

Мета дипломного проекту – побудова підсистеми енергозабезпечення, розробка програмного забезпечення.

Метод дослідження – аналіз існуючих варіантів побудови підсистеми, розробка алгоритмів збору інформації та управління роботою підсистеми.

Новизна – використано не стандартний стек технологій, розроблено унікальний інтерфейс.

Актуальність – розвиток нових технологій та популярність автоматизованих систем в наш час.

Матеріали дипломного проекту рекомендуються використовувати в будинках та квартирах при обслуговувані жителів та у навчальному процесі.

Прогнозні припущення щодо розвитку об’єкта дослідження – створення підсистеми енергозабезпечення для бажаного економічного ефекту.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Wi-Fi – Wireless Fidelity (бездротова передача)

SH – Smart House (розумний дім)

PID – Proportional Integral Differential (ідентифікатор процесу)

SQL – Structured Query Language (мова структурованих засобів)

IDE – Integrated Development Environment (вбудоване середовище розробки)

GPIO - General-Purpose Input/Output (інтерфейс зв’язку компонентів)

LDR – Light Dependent Resistor (світовловий резистор)

ORM – Object-Relational Mapping (об'єктно-реляційне відображення)

WEB – World Wide Web (всесвітня мережа)

IP – Internet Protocol (інтернет протокол)

РД – Розумний дім

ВСТУП

Розроблена система є актуальною, адже вона надає кожній українській квартирі, будинку, приватним установам не відставати сучасних побудов, які можуть використовувати свої функції за допомогою системи розумний дім:

1) енергозабезпечення;

2) збереження та обробка інформації стосовно результатів оглядів;

3) інформування користувача про проблеми з використання енергії в будинку.

Саме представлена система дасть нам змогу добитися кроссплатформеності та адаптованості системи обслуговування до середовища. Цей сервіс зручний у використанні як для клієнтів, так і для адміністратора, оскільки для користувачів це, по суті, лише інтерфейс, а всі складні програмні модулі залишаються створенні програмістом і невидимі навіть адміністратору сервісу.

Для реалізації сервісу програмними засобами необхідно розробити комплекс взаємопов'язаних алгоритмів та модулів, які зможуть забезпечити збереження інформації, просте та зрозуміле оброблення і функціональні можливості системи.

Проектовану систему запропоновано розробити за допомогою *Arduino Mega*. Цей продукт дозволяє поєднати різноманітні програмні технології так як *C#*. Саме цю мову програмування було обрано для створення проекту.

Мета дипломного проекту – збереження та обробка інформації в єдиному просторі, комфортна робота з компонентами будинку, що відповідають за використання енергії, знаходження оптимальної економічної стратегії в будинку. Для досягнення визначеної мети ставляться такі завдання:

- спростити процес доступу до інформації;
- налагодити коректну передачу даних з датчиків;
- реалізувати прийнятний інтерфейс для підсистеми.

Новизна отриманих результатів – оптимізовано витрату енергії в жилих будинках за рахунок аналізу класичної вибору оптимальних датчиків та розробленого програмного засобу.

Матеріали дипломного проекту рекомендуються використовувати в будинках та квартирах при обслуговуванні мешканців та у навчальному процесі.

Розроблено особисто. Алгоритми роботи підсистеми, алгоритм передачі даних, структурну та функціональну схеми. Створено інтерфейс та логіку програмного засобу за допомогою *C#*. Налаштовано процес збору та передачі інформації за допомогою *Arduino*.

Практичне значення результатів дає змогу користувачам розробленої підсистеми, регулювати енерговитрати в своїх будинках.

Потенціал систем «інтелектуальних будинків» для українського ринку важко переоцінити. Використання концепції дозволяє досягти відчутного полегшення в енергозбереженні ресурсів (від 20% до 30%). Цей фактор готові брати до уваги і будівельники, які передбачають застосування даної технології в 30% проектованих та збудованих будинків.

При виборі теми дипломного проекту, я зупинився саме на цій. Оскільки розроблена система може не тільки показати всі навички, набуті за період навчання, а й бути корисною в майбутньому.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ПОБУДОВИ ПІДСИСТЕМИ ЕНЕРГОЗАБЕЗПЕЧЕННЯ

1.1 Огляд задач, які вирішуються за допомогою розумного дому.

Системи управління будинком – не новина. Вони з'явилися вже давно, але всі існуючі до останнього часу розробки припускали найскладніші монтажні роботи. Кожен компонент системи вимагав підводки індивідуального кабелю, що майже вдвічі збільшувало систему, отже і прозводило до вкрай високої вартості системи. Автоматизовану систему може дозволити собі практично кожен. І користуватия нею має можливість пересічний громадянин.

Бездротова система, розроблена в США і отримавша назву *Smart house*, нею користується весь світ. Своєю популярністю вона зобов'язана, перш за все, швидкою і безболісною інтеграцією, що зводиться до простої заміни вимикачів і розеток, так як все управління технікою всередині будинку здійснюється через звичайну електромережу (рис. 1.1).

Крім компонентів, що відповідають за управління аудіо, відео, побутовою технікою і світлом, в систему може входити велика кількість додаткових модулів, підбір яких залишається за користувачем і обмежується лише фантазією. Немов конструктор "Лего", систему "Слухняний дім" можна збирати поступово, раз від разу набуваючи все нові опції.

Рис. 1.1 – Приклад роботи з розумним домом.

Почати можна з мінімального комплекту, в який входить універсальний учень пульт на вісім пристроїв і приймач, що дозволяє управляти технікою, перебуваючи в іншій кімнаті. У якій би частині квартири ви не знаходилися, вам будуть підвладні всі електронні пристрої. Пульт управління посилає сигнал на спеціальний приймач у вигляді симпатичної пірамідки, яка в свою чергу віддає команди апаратурі. Однак всю радість володіння "інтелектуальним домом" повною мірою можна відчути, користуючись додатковими компонентами.

Вперше поняття «розумний дім» з`явилося в 50-х роках минулого століття. Прародителькою системи, здатної контролювати обстановку в цілому будинку, є технологія *Java*. Розробники цієї технології намагалися впровадити її в побутові прилади, тим самим зробивши їх більш «інтелектуальними». Наприклад, вже в той час почали з`являтися перші Мікрохвильові печі з функцією автоматичного вимкнення, кондиціонери, здатні регулювати мікроклімат приміщення в залежності від погоди за вікном і т.д.

Однак поняття «розумний дім» включає в себе не тільки інтелектуальну побутову техніку. Якщо пояснювати простими словами, то ця система координує роботу всіх технічних пристосувань, що знаходяться в будинку. Причому управління системою може здійснюватися як за допомогою пульта, так і дистанційно, за допомогою сучасних девайсів - айфона, смартфона, планшета (рис 1.2).

Рис. 1.2 – Керування розумним домом за допомогою девайсів.

Можливості системи «розумний дім» багатогранні. Наприклад, щоб запобігти ймовірність пограбування, коли в будинку нікого немає, система імітує присутність господаря шляхом розсовування жалюзі, включення, або вимикання світла. Якщо ж зловмисники все ж проникають всередину приміщення або відбувається інша екстраординарна ситуація, система блискавично сповіщає про це господаря. Крім того, технологія «розумний дім» дозволяє структурувати роботу всього технічного та інженерного обладнання, задавши йому певний сценарій. Наприклад, перед вашим пробудженням система нагріє підлоги у ванній кімнаті, включить музичний центр, налаштує роботу кондиціонера на задану температуру, відрегулює оптимальну вологість в приміщенні і вирішить безліч інших побутових завдань все це можливо завдяки контролерам передачі даних приклад яких зображено на рис. 1.3.

Рис. 1.3 – Контролери системи інтелектуального дому.

Система «розумний дім» умовно ділиться на кілька самостійних підсистем:

- безпеки;
- *Multiroom;*
- клімат-контролю;
- енергозабезпечення, та ін.

Підсистема безпеки розумного дому включає в себе такі елементи як сигналізація, протипожежні датчики, а також датчики, що реагують на несправність комунікативних ліній.

Підсистема *multiroom* відповідає за розподіл аудіо та відеосигналу по всьому дому.

Пистема клімат-контролю в «розумному домі» управляє пристроями, що відповідають за опалення, кондиціонування, очищення і зволоження повітря.

Підсистема енергозабезпечення розумного дому управляє всіма засобами елктро споживання, перш за все, освітлювальними елементами об`єкта, (рис. 1.4).

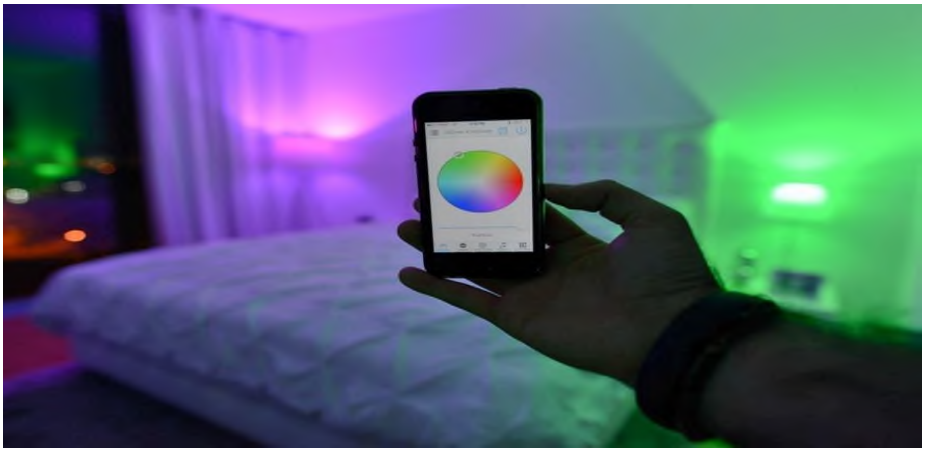

Рис. 1.4. – Приклад роботи системи освітлення.

При необхідності вона відключає невикористовувані в даний момент джерела світла, автоматично регулює ступінь освітленості в залежності від часу доби і навіть пори року, а також вмикає світло, як тільки ви з`являєтеся вдома. Крім того, щоб відвернути увагу зловмисників, в системі можна встановити програму періодичного включення, або вимикання світла в будь-якому приміщенні під час вашої відсутності в будинку.

На поточний момент ринок пристроїв для керування розумним домом вражає своєю різноманітністю і ціновими рамками.

Після вивчення ринку пристроїв і контролерів для управління розумним домом можна зупинитися на доступних і недорогих типах.

У підсумку, для представлення способу реалізації було обрано сімейство мікроконтролерів *Arduino*, одноплатних міні-комп'ютерах *Raspberry PI* і на сучасних *Wi-Fi* мікроконтролерах *ESP8266* в різних модифікаціях.

Наступним кроком є вибір протоколу спілкування пристроїв з системою. Для першого етапу проекту запропоновано використовувати поширений в світі *Internet of Things* протокол *MQTT* (*Message Queue Telemetry Transport*). Це легкий мережевий протокол, який працює поверх *TCP / IP*. Використовується для обміну повідомленнями між пристроями за принципом видавець-передплатник.

Оскільки функціонал інтелектуального будинку може відрізнятися один від одного. Слід розглянути як саме може поводити себе система с точки зору користувача. Поріг входження в навчання роботі з такими пристроями повинен бути мінімальний для представників IT-професії. Як елементи взаємодіють один з одним та який цілісний комплекс вони утворюють.

Центральною деталлю в комплексі є контролер. Він встановлюється десь біля вхідних дверей, як правило, має масу кнопок і екран, на якому відображаються всі дані. Контролер може існувати як самостійно, так і підключатися до комп'ютера.

Він має взаємодіяти з виконавчими елементи які зображено на рис. 1.5. Це різні датчики і вимикачі, які встановлюються в потрібних місцях і мають зв'язок з контролером. Наприклад, для управління світлом без вимикача використовується адаптер - патрон, який при подачі сигналу від контролера включає і вимикає лампу. Сигнал від контролера надходить, якщо з пульта дистанційного керування чоловік подав команду або від датчика руху (аудіо-датчика або фотоелемента).

Дана система може бути 2 видів. У першому випадку управління здійснюється за допомогою дротового зв'язку, тоді найкраще встановлювати «розумний дім» одночасно з електропроводкою.

У другому зв'язок здійснюється за допомогою мережі, тоді монтувати систему можна в будь-який час.

Рис. 1.5 – Центральний контролер та виконавчі елементи.

Монтаж системи виконується шляхом розміщення різних датчиків і заміни звичайних вимикачів і розеток на пристрої з комплекту. Потім йде довгий період підгонки всієї системи, коли встановлені елементи програмуються і з'єднуються з контролером.

1.2. Підсистема енергозабезпечення.

Енергозабезпечення стосується зменшення споживання енергії за рахунок використання меншої кількості енергетичних послуг. Енергозабезпечення відрізняється від енергоефективності, яке стосується використання меншої кількості енергії за тої самої послуги. Наприклад, менше користуватись авто - заощадження енергії, а пересісти на авто з меншою витратою палива, або на електромобіль буде означати саме енергоефективність. Але і енергозабезпечення, і енергоефективність є техніками зменшення використання енергії.

Енергозберігаюча система управління освітленням в багатоповерхових будинках (під'їзди, автостоянки, прибудинкові території, підвали, горища) дозволить знизити кількість споживаної електроенергії в декілька разів.

У цих системах застосовується пристрій управління освітленням з роздільними силовими компонентами, що дозволяє використовувати існуючі лінії електропередач.

Великий ефект надає використання енергозберігаючих освітлювальних приладів. Однак навіть сама «економна» лампа, якщо вона горить в порожньому приміщенні, стане безглуздим джерелом енерговитрат. Найкраще енергозабезпечення забезпечують автоматичні вимикачі світла з використанням інфрачервоних та електронних датчиків. Електронні датчики вимірюють рівень освітленості приміщення і, при досягненні заданого 20 значення, видають команду на включення або виключення освітлення (датчики освітленості), або безпосередньо «бачать», що до приміщення увійшов чоловік, і вмикають світло (датчики руху). Світлочутливий елемент блокує ввімкнення освітлення при достатньому природному освітленні. Оскільки на відміну від реле-датчиків часу датчики руху вмикають світло тільки на час фактичної присутності людини в приміщенні, а витрати електроенергії на освітлення можуть бути знижені в кілька разів.

1.3. Сучасна практика використання енергозаощадження.

Питання досить важливе, так як енергоспоживання в житлових і багатоквартирних будинках дорівнює 30% загального споживання енергії в нашій країні. Потенціал енергозабезпечення в цьому секторі величезний.

Перед тим як говорити про енергозабезпечення, необхідно провести аналіз використання електроенергія в будинку або квартирі, а на його основі можна шляхи покращення ситуації з енергоспоживанням.

Спостереження, що публікуються багатьма дослідниками надають наступні результати.

Енергозберігаючі лампи приблизно в 4-5 разів ефективніше звичайних ламп розжарювання.

Мікрохвильовка споживає на 50% менше енергії, ніж звичайна духовка.

Телевізор, аудіо система, магнітофон, в режимі очікування, споживає в середньому 10 ват на годину.

Зарядки мобілок, лептопи увіткнені в розетку споживають електрику, навіть якщо ви нічого не заряджаєте.

Кондиціонер повинен бути з термостатомо ремонту.

Комп'ютер краще відключати, коли він не використовується. Деякі комп'ютери споживають стільки ж електроенергії в режимі очікування, як маленький холодильник.

Ці методи заощадження відносяться до пасивних методів і надають певні результати. Але, наряду з ними застосування активних методів надає найбільший ефект енергозаощадження.

1.4. Варіанти реалізації системи енергозаощадження.

Починаючи з аналізу структурних рішень, за допомогою бібліотек і *IDE* спробуємо пояснити вихідний код програми.

Як можна бачити на рис. 1.14 Центральний блок складається з двох апаратної платформи: *The Raspberry Pi* з метою управління логічною частиною та Інтернетом, зв'язком та приймачем, який використовується для зв'язку з іншим пакетом.

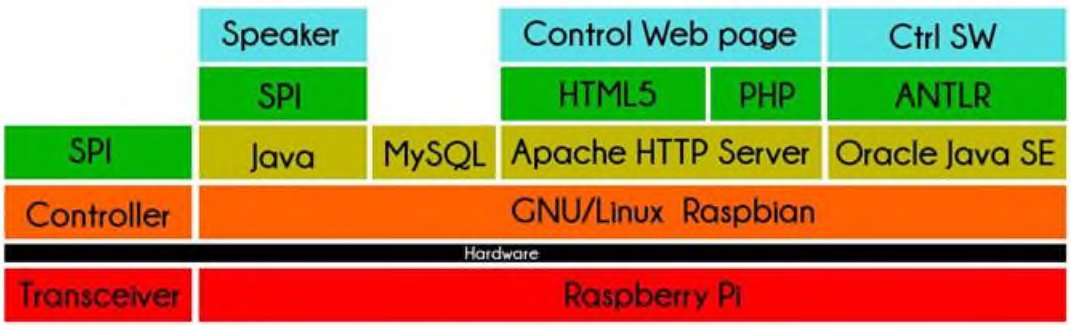

Рис. 1.14 – центральний блок системи розумного дому.

Звичайно, перше, що потрбіно зробити встановити навколишнє середовище, потрібно шукати *ОС*, на якій можна було б встановити всі

програмні додатки, що важливі для створення, тобто всі програмні частини які представлені на рис. 1.14.

Потім, все ще в *Raspberry Pi*, було обрано встановити *Apache,* як сервер, *MySQL*, як СУБД і *PHP* як мову сценаріїв, взагалі програмне забезпечення цієї форму має форму так званної *LAMP* архітектури, за рахунок такого підходу не потрібно активно використовувати принципи обьєктно-орієнтованого програмування описаного в книжці «Власні веб-інтерфейси и мобільні додатки для з домашньої систем автоматизації» Давида Анджеліса. Що стосується контролю та управління частіше в таких системах використовується програмне забезпечення на мові *Java*, щоб полегшити переносимість та підвищити швидкість, виникла потреба у формуванні віртуальної машини *Java* и бібліотек *Java*. Для Досягнення цієї мети *Oracle* пропонує *Java SE* яку можна встановити в *ARM* платформі, представленій в наступних розділах. Програмне забезпечення управління є ядром системи.

В інтелектуальному будинку головна мета це комфорт для користувача, даний стек технологій може забезпечити це. С точки зору розробника, дана архітектура має доволі потужний функціонал та прекрасно підходить для представленого дипломного проекту.

1.5. Висновки до розділу.

Виходячи з аналізу підсистем енергозабезпечення було обрано наступний варіант створення: інтеграція системних плат та датчиків в існуючу концепцію житлового будинку, налагодження комунікації апаратного рівня з програмним, в програмний рівень входить плагін написаний на мові програмування *C#,* база даних та модулі передачі даних. Основна задача підсистеми енергозабезпечення управління використанням електроенергії в різних обставинах.

Для реалізації поставленої задачі, управління використанням електроенергії, необхідно наступне:

1) Розробити структурну схему управління енергоспоживанням.
2) Розробити функціональну схему управління енергоспоживанням.
3) Розробити алгоритм збору даних.
4) Розробити алгоритм роботи підсистеми.
5) Провести програмну реалізацію розроблених алгоритмів.
6) Провести тестування розробленого програмного засобу.

Проаналізований досвід розробки аналогічних додатків було використанно при розробці власного продукту. Існуючі архітектури цілісних систем клімат контролю враховані при розробці додатку для майбутніх оптимізацій.

РОЗДІЛ 2
ПІДСИСТЕМА ЕНЕРГОЗАБЕЗПЕЧЕННЯ СИСТЕМИ «РОЗУМНИЙ ДІМ»

2.1. Вибір датчиків та виконуючих пристроїв.

Надійний контроль безпеки «розумного будинку» і стану його інженерних систем просто неможливий без цих невеликих, але вкрай важливих електронних пристроїв - датчиків. Набір датчиків різного призначення, підключених до контролера «розумного будинку», забезпечує захист людей і майна, допомагає підтримувати комфортні умови проживання. Датчики використовують для найрізноманітніших завдань.

Завдяки датчикам «розумний дім» зуміє ліквідувати протікання води або витік газу в аварійній ситуації, поллє засихаючий газон в жаркий день. І всю цю складну роботу «інтелектальний будинок», завдяки датчикам, зробить без втручання з боку людини.

Датчики розумного будинку умовно можна розділити на три групи: датчики охорони, датчики відслідковують стан навколишнього середовища і датчики, що стежать за станом інженерних комунікацій. Нас цікавлять останні два типи.

Датчики, які відстежують стан навколишнього середовища, допомагають регулювати мікроклімат в будинку, щоб домогтися умов, що забезпечують максимальний комфорт і затишок для мешканців. Наприклад, датчики температури через контролер «розумного будинку» пов'язані з кондиціонерами,

нагрівальними приладами та системами теплої підлоги, що сприяють безперервній підтримці найбільш оптимальної температури в приміщеннях. А свідчення зовнішнього датчика температури і вологості можуть використовуватися для роботи автоматизованої системи поливу.

Завдання дипломного проекту полягає в створені підсистеми енергозабезпечення. Для таких цілей було обрано наступний набір мікроконтролерів та датчиків:

1) датчики температури і вологості типу *DHT22*, *DHT11*;
2) датчики барометричного тиску типу *BMP180*;
3) *WiFi* модуль *ESP8266*;
4) радіомодуль типу *nRF24* 2,4 Ггц;
5) сімейство *Arduino Pro Mini*, *Arduino Mega*;
6) сонячною батареєю і акумуляторами.

Розглянемо кожен з пунктів більш детально. Почнемо з датчиків температури та вологості, оскільки основним модулем енергозберігання є саме збереження енергії за рахунок регуляції клімату, потрібно підійти до вибору даних датчиків дуже відповідально.

Розглянемо основи роботи з недорогими датчиками температури і вологості серії *DHT*. Ці сенсори прості і повільні, але при цьому відмінно підходять для не великих проектів на *Arduino*. Датчики *DHT* складаються з двох основних частин: ємнісний датчик вологості і термістор. Також в корпусі встановлений простенький чіп для перетворення аналогового сигналу в цифровий. Зчитувати цифровий сигнал на виході досить просто, можна використовувати будь-який контролер, не обов'язково *Arduino*. На вигляд дані датчик можна побачити на рис. 2.1.

Існують дві версії сенсорів *DHT*. Виглядають вони майже однаково. Терморегулятори теж однакова. Основні відмінності - в технічних характеристиках. *DHT*22 більш точний і має більший діапазон вимірюваних

значень. Обидва датчика мають по одному цифровому виходу. Запити до них можна відправляти не частіше ніж один в секунду або дві.

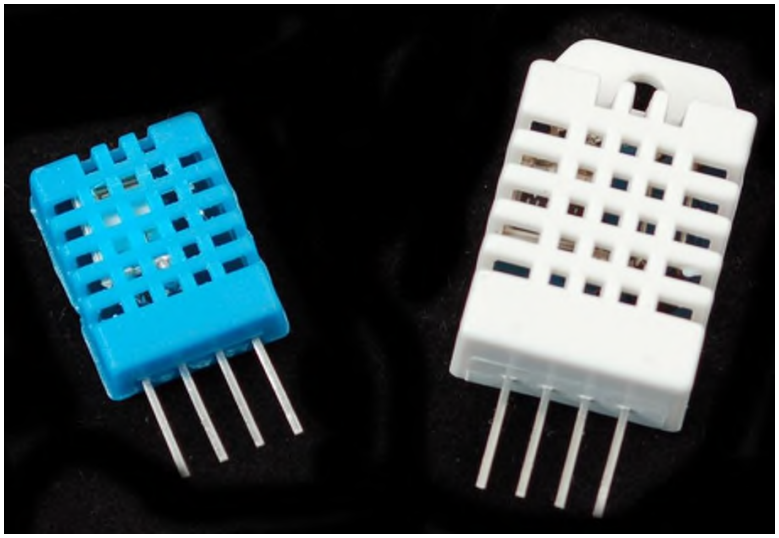

Рис. 2.1 – Датчики *DHT22* та *DHT11*.

Розглянемо характеристики запропонованих датчиків в табл. 2.1.

Таблиця 2.1.

Характеристики датчиків *DHT*

Назва датчику	*DHT*22	*DHT*11
Приблизна ціна	100 грн.	70 грн.
Потужність	Від 3 до 5В	Від 3 до 5В
Спож. струму	2А	2А
Рівень вологості	Від 0 до 100%	Від 20 до 80%
Температура	Від -40 до 125С	Від 0 до 50С
Частота вимірів	0.5Гц	1Гц
Кільк. коннекторів	4	4

Датчик *DHT11* буде використано в домашніх умовах, через те, що температура не буде досягати відмітки меньше нуля. Датчик *DHT22* обраний для для вимірювання температури на балконі, оскільки рівень коливання температури та вологості там набагато більша.

Датчик барометричного тиску потрібен в даній підсистемі для того, щоб виміряти тиск, саме це дасть змогу нам передбачувати кліматичні умови в будинку, та за рахунок цього корегувати дані для енергозабезпечення.

Барометричний датчик тиску *BMP180* призначений для вимірювання барометричного, абсолютного, диференціального, надлишкового тиску, а також значення температури навколишнього середовища.

Для використання датчика тиску спочатку потрібно зібрати на його основі макет, попередньо підключивши. Потім датчик тиску потрібно відкалібрувати за допомогою спеціальних програм. Також для нормальної роботи датчика з Arduino контролерами потрібна спеціальна бібліотека. Сам датчик представлено на рис. 2.2.

Рис. 2.2 – Датчик *BMP180.*

Розглянемо характеристики запропонованих датчиків в табл. 2.1.

Таблиця 2.2

Характеристика датчику *BMP*

Назва датчику	*BMP180*
Приблизна ціна	85 грн.
Потужність	Від 3.3 до 5В
Спож. струму	3мкА
Рівень тиску	300 – 1100hPa
Температура	Від -40 до 85С
Час відклику	0.5мс
Кільк. коннекторів	5

Модуль можна використовувати в домашніх метеостанціях, літальних апаратах. В дипломному проекті цей датчик є блоком передбачення клімату.

В центрі квартири чи будинку повинен знаходитися датчик який буде збирати інформацію: *ESP8266.* Оскільки підсистема енергозабезпечення повинна мати чітку комунікацію з сервером, датчиками та користувачем. Було обрано цей надійний та доволі дорогий пристрій.

Типове застосування *ESP8266* як апаратної основи *IoT* (*Internet of Things)* найчастіше має на увазі установку в будинках або офісах. При цьому підключення до мережі здійснюється до домашньої або офісної локальної мережі з виходом в інтернет через роутер. Власник пристрою може контролювати його за допомогою планшета або комп'ютера через свою локальну мережу або віддалено, через Інтернет.

ESP8266 може працювати як в ролі точки доступу так і кінцевої станції. При нормальній роботі в локальній мережі *ESP8266* конфигурується в режимі кінцевої станції. Для цього пристрою необхідно задати *SSID Wi-Fi* мережу і, в закритих мережах, пароль доступу. Для початкового конфігурування цих параметрів зручний режим точки доступу. У режимі точки доступу пристрій видно при стандартному пошуку мереж в планшетах і комп'ютерах. Залишається підключитися до пристрою, відкрити HTML сторінку конфігурації і задати параметри мережі. На вигляд представлений датчик можна побачити на рис. 2.3.

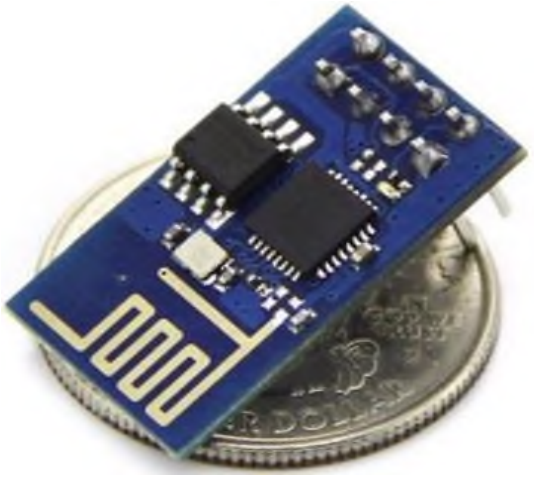

Рис. 2.3 – *WiFi* модуль *ESP8266.*

При використанні *WiFi* модулю виникають проблеми з комунікацією між датчиками та основним домашнім блоком. Оскільки блок повинен отримувати сигнал чітко і швидко, представлений датчик не дає нам таких опцій.

Не зважаючи навіть на цей фактор можна розглянути ще більше негативних сторін даного підходу:

- необхідний стійкий *WiFi* за межами будинку;
- дальність зв'язку не буде великою;
- постраждає надійність, при пропажі інтернету ми не побачимо свої віддалені датчики;
- більше енергоспоживання.

Зрештою, було обрано наступне рішеня проблеми, для зв'язку віддалених датчиків з основним домашнім блоком був обраний чіп *nRF24L01* з 2,4 Ггц передавачем і приймачем в одному флаконі, з додатковою зовнішньою антеною, щоб уже напевно коректно та швидко передвати сигнал через стіни. Чіп *nRF24L01* позакано на малюнку 2.4.

Рис. 2.4 - Чіп *nRF24L01*.

Відстань залежить від зовнішніх умов. Тобто: перебуваєте ви в приміщенні або на вулиці, чи є перешкоди на шляху сигналу від радіопередавача. Більшість виробників модулів *nRF24L01* з маленькою потужністю передавача вказують відстань його дії 200 футів або 100 метрів. Це характеристика передавачів, які використовуються на відкритому повітрі, без перешкод на шляху сигналу.

Частота передавача налаштована на 2500 кГц. Усередині приміщень, через наявність стін і інших перешкод відстань спрацьовування буде менше.

Рекомендовано протестувати радіопередавач в конкретних умовах перед його використанням. Крім того, є деякі модифікації радіопередавачів *nRF24L01* із зовнішньою антеною, яка підсилює сигнал. Кожна конкретна ситуація накладає свої обмеження, так що встановити точну характеристику відстані їх дії доволі складно, з використанням антени передача даних в квартирі була досить прийнятною.

Я тестував окремі частини системи на *Arduino UNO*. Тобто підключав до уно *ESP* модуль і вивчав його, відключав, потім підключав *nRF24* і т.д. Для фінальної реалізації заоконного датчика вибрав *Arduino Pro Mini* як найбільш близьку до *Uno* з мініатюрних плат, показано на рис 2.5.

Рис. 2.5 – Плата *Arduino Pro Mini.*

Arduino Pro Mini призначений для напівстаціонарного монтажу в різне устаткування або установки. Плата спеціально поставляється без впаяних роз'ємів, що дозволяє користувачеві прикріпляти дроти або використовувати необхідні типи роз'ємів на свій розсуд.

Оскільки головною метою дипломного проекту є побудова підсистеми енергопостачання із функцією збеігання енергії, то потрібно вибирати всю аппаратну частину системи також не енергозатратною.

Для центрального блоку, оскільки до нього планувалося підключити численну периферію, була обрана плата *Arduino Mega*. До того ж вона повністю сумісна з *UNO* і має більше пам'яті.

Arduino Mega - це пристрій на основі мікроконтролера *ATmega2560* . У його склад входить все необхідне для зручної роботи з мікро контролером: 54 цифрових входу та виходу, 16 аналогових входів, 4 *UART* (апаратних приймача для реалізації послідовних інтерфейсів), кварцовий резонатор на 16 МГц , роз'єм *USB*, роз'єм живлення, роз'єм *ICSP* для внутрисхемного програмування і кнопка скидання. Для початку роботи з пристроєм досить просто подати живлення від *AC/DC*-адаптера або батарейки, або підключити його до комп'ютера за допомогою *USB*-кабелю. *Arduino Mega* сумісний з більшістю плат розширення, розроблених для *Arduino Duemilanove* і *Diecimila*. На рис. 2.6. показана представлена плата.

Рис. 2.6 – Плата *Arduino Mega.*

Оскільки головною метою дипломного проекту є енергозабезпечення потрібно обрати як саме підзаряжати всю систему інтелектуального дому. Я зупинився на приладі *Power Supply Module AC 110v ~ 240V to 12V DC*. Показаному на рис. 2.7.

А якщо цього не вистачить, то можна і потужніший поставити. Іншими словами економити електроживлення для центрального блоку немає особливого сенсу. А ось для віддаленого датчика енергозабезпечення є найважливішою частиною. Але і функціональність при цьому не хотілося б втрачати. Тому *Arduino Pro Mini* і радіомодуль *nRF24* будуть живитися від зв'язки 4-х *Ni-Mh* акумуляторів, показаних на рис. 2.8.

Рис. 2.7 – Накопичувач енергії для інтелектуального будинку.

Рис. 2.8 – *Ni-Mh* акамулятори для живлення модулів та плат.

Якщо поглянути на запропоновану систему датчиків, плат, акамуляторів та радіомодулю можна зробити висновок, що був обраний доволі дешевий набір технологій, які не дають бажаної надійності та результату. Але для досягнення цілей поставлених в дипломному проекті таких інструментів більше ніж достатньо.

2.2. Загальна структура підсистеми енергозабезпечення.

Для глибокого розуміння створеної системи користувачу та розробнику потрібно розумти як саме продукут працює. Даний підрозділ присвячений аналізу даного питання. Оскільки розробка інтелектуального дому, чи підсистеми енергозабезпечення не є звичайним проектів для програміста. Слід підійти до створення структурної схеми дуже відповідально та обачно. Адже в даному випдаку ми оперуємо не тільки класами, модулями и пакетами, а й датчиками, серверами, платами та блоками управління. Структурну схему можна представити наступним чином (рис. 2.9).

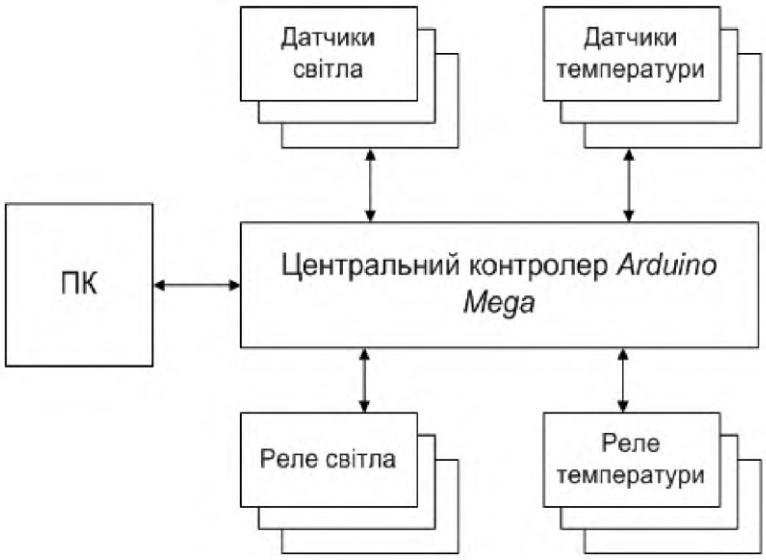

Рис. 2.9 – Структурна схема підсистеми енергозабезпечення.

Представлену структурну схему можна умовно поділити на 5 блоків:

1) блок управління за допомогою реле;
2) блок центрального процесору *Arduino Mega*;
3) набір датчиків;
4) ПК;
5) програмні засоби реалізовані на ПК.

Кожен з цих блоків, має свою задачу в запропонованому додатку.

Блок управління за допомогою реле дає змогу додатку впливати на існуючі опції в квартирі: вимкнути світло, зменшити витрату електроенергії, послабити отоплення квартири. Реле кріпляться на домашні пристрої, такі як перемикачі світла, ричаги в батареях та центральні блоки управління електроенергією. Приймаючи сигнал від *Arduino Mega.* Реле генерує певний електричний, або механічний імпульс, що і змінює стан пристроїв в квартирі. За рахунок цього у нас створена комунікація між командами та пристроями в будинку.

Arduino Mega – центальна плата розробленоно проектку. Саме через неї відбувається комунікація між датчиками, між реле та персональним компьютором. Включаючи в себе певний набір функцій і опцій ардуіно дає змогу нам коректно та швидко передавати сигнали між іншими пристроями. Центральний блок управління збирає дані з датчиків потів відправляє їх до блоку

логіки, а саме, до комп'ютера. Там в свою чергу, залежно від поставленого сигналу відбувається обробка інформації та вирішується яку саме команду відіслати до центральної плати управління. Потім *Arduino Mega* відправляє сигнал до реле які в свою чергу генерують певний електричний, або механічний імпульс, що і змінює стан пристроїв в квартирі.

В реалізації бажаного функціоналу підсистеми енергозабезпечення необхідно використати:

1) 3 датички *DHT22* для збору даних по температурі та вологості в коридорі, вітальні та балконі.

2) 2 датчики *DHT11* для збору даних по температурі та вологості в спальні та кухні.

3) 2 датчики *ESP826* буде розміщено в вітальні та спальні для коректної передачі сигналу *Wi-Fi*.

4) Датчик *nRF24* для посилення потужності та точності передваємого сигналу з датчиків до *Arduino Mega*.

Саме ця інформація буде надходити до центральної плати та зрештую оброблятися в комп'ютері для вирішення задач як саме потрібно економити енергію в будинку. Представлені датчики не дають ідеальну точність та потрібні потужності. Саме тому було вирішено використати додатковий датичк *ESP826* для збільшення діапазону передачі сигналу та *nRF24* успішного подолання перешков в вигляді стін та дверей.

ПК в даному проекті є центральним логічним модулем, оскільки саме ПК може обробляти надану інформацію з бажаною швидкість та в той же час зберігати її і надавати інтерфейс для роботи користувача. ПК напряму з'єднане з *Arduino Mega* і працює як на прийом так і на передачу даних.

Представлений модуль ПК складається з трьох части: *web*-серверу, інтерфейсу користувача та бази даних.

Web-сервер потрбібен для того, щоб налагодити вітдалену комунікацію між додатком та користувачем. Також, дана сруктура дозволить викоистовувати базу даних та інтерфейс, без додаткових модулів та програмних засобів.

База даних включає в себе подану інформацію та інформацію яку потрібно буде надавати для вихідних даних. Саме в БД і зберігаються дані зняті з датчиків та дані які оброблені за рахунок логічного блоку в *web*-сервері.

UI (user interface) виконано для прийнятної роботи користувача та системи, саме в цьому модулі і показана вся комунікація системи, що дозволяє впевнено та коректно працювати з системою мешканцям будинку.

2.3. Функціональна схема підсистеми енергозабезпечення

Функціональна схема в підсистемі енергозабезпечення це документ, який показує як процеси, що протікають в окремих функціональних колах взаємодіють між собою. Функціональна схема є поясняющим матеріалом окремих видів процесів, що протікають в цілісних функціональних блоках і ланцюгах пристрою або програми.

Функціональна схема показує чітку картину того, як саме працює підсистема та які опції вона може виконувати для розробленого програмного засобу вона показана на рис. 2.10.

Розглянемо можливі операції програмного засобу проілюстраваного вище, більш детально. Пункт який знаходиться першим є збереження даних. Цей блок дає змогу системі зберігати дані які ми приймаємо з датчиків та відправляти потрібну інформацію до центру обробки даних.

Два модулі центру обробки даних розташовані нижче ніж модуль збереження інформації, вони слугують логічними блоками, що приймають дані датчиків, передають їх до БД та оброблюють залежно від поставлених задач, потів на вивід подають інфармацію для реле, щоб змінити функції опалення або

змінити постачання еклектроенергії. Центральний блок обробки знаходиться на веб сервері та оперує логікою, що прописана в програмному коді.

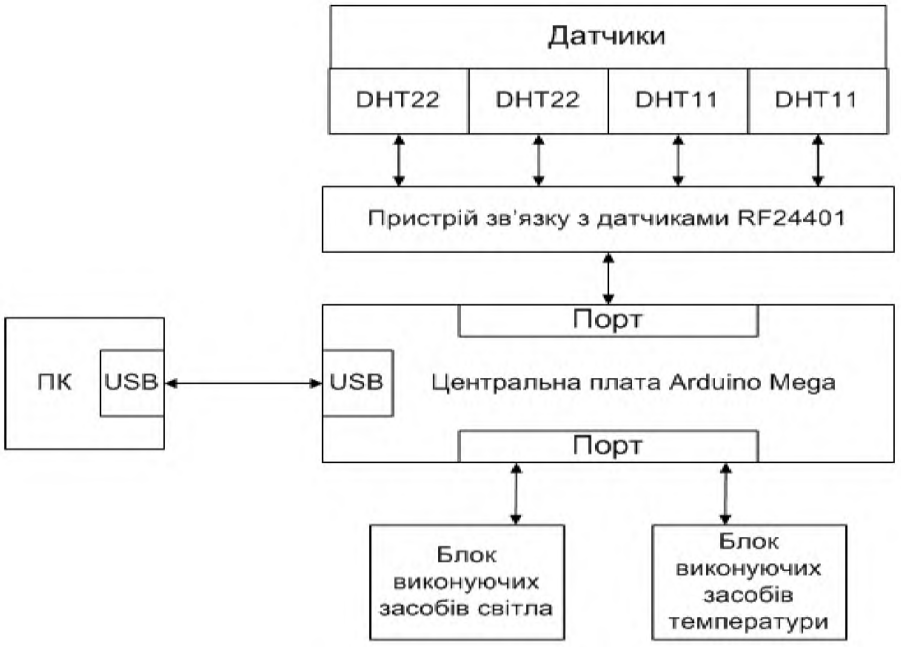

Рис. 2.10 – Функціональна схема енергозабезпечення.

Не можна не згадати, що функціонал повинен включати не тільки логічний рівень, що прописаний в коді, а й блок керування користувача. Оскільки користувач може внести дані в ручну та показати системі, що саме потрбіно робити. Ми повині розробити, ще один модуль для обробки інформації і передачі даних в базу. Як показано на функціональній схемі так операції робляться так само як вище згаданий структурах з датчиками. Просто замість датчика інформацію вводить користувач. А блок обробки і виводі даних на реле залишається тим самим яким і був.

2.4. Алгоритм роботи підсистеми

Розглянувши структурна схему та функціональну схему можна зразумити між якими елеемнтами відбувається комунікація та як саме це відбувається також можна зрозуміти як підсистема функціонує, та які опції виконує.

Для перенесення цих схем в програмний код та прогрмний додаток потрібно створити алгоритм роботи підстистеми, що дасть змогу розуміти як

підсистема працює, як її створювати, та на які питання треба відповісти перед тим як приступати до експлуатації розробленого додатку.

Алгоритм роботи підсистеми наведено на рис. 2.11, він складається з наступних процесів:

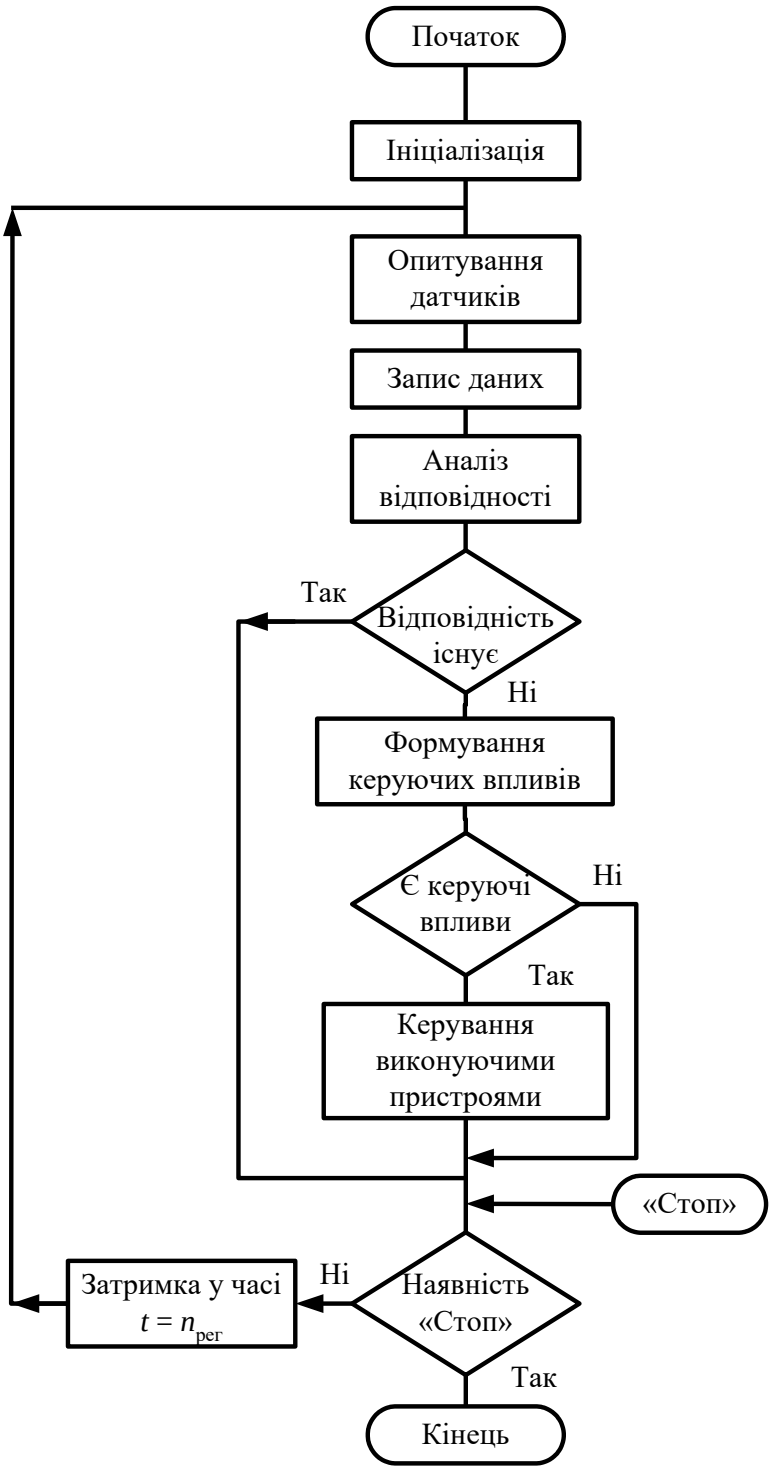

Рис. 2.11 – Алгоритм роботи підсистеми

1) Процес ініціалізації призначений для перевірки працездатності всіх складових технічних засобів та встановлення їх у робочий стан (запис управляючих слів, даних, тощо).

2) Далі відбувається циркулярне опитування датчиків для перевірки їх працеспроможності та знімання з них даних з подальшим записом до блоку пам’яті.

3) Наступний процес призначений для аналізу відповідності отриманих даних до встановлених значень на поточний час і стан системи в цілому.

4) За умови існування описаної відповідності відбувається перехід до перевірки керуючого сигналу завершення роботи підсистеми «Стоп». Прийого наявності підсистема закінчує свою роботу. За його відсутності управління переходить до процесу формування часової затримки на час $t = n_{рег}$, де $n_{рег}$ – регульований проміжок часу.

5) По закінченню затримки управління знову переходить до процесу опитування датчиків.

Одже алгоритм циклічний і працює до приходу сигналу завершення роботи підсистеми «Стоп».

2.5. Алгоритми опитування датчиків

Одною з основних частин розробленої підсистеми є збір інформації з встановлених датчиків. Датчики використовуються двох видів: пасивні і активні.

Перш за все відбувається ініціалізація датчиків. Якщо датчик аналоговий то як правило його ініціалізація зводиться до подачі напруги і сняття сигналу, якщо цифровий - то все залежить від архітектури пристрою. Для датчиків з автоматичним настроюванням може знадобитися ініціалізація внутрішніх регістрів із записом в них стартових значень.

Для прикладу розлянуто алгоритм опитування пасивних датчиків, схему якого надано на рис. 2.12.

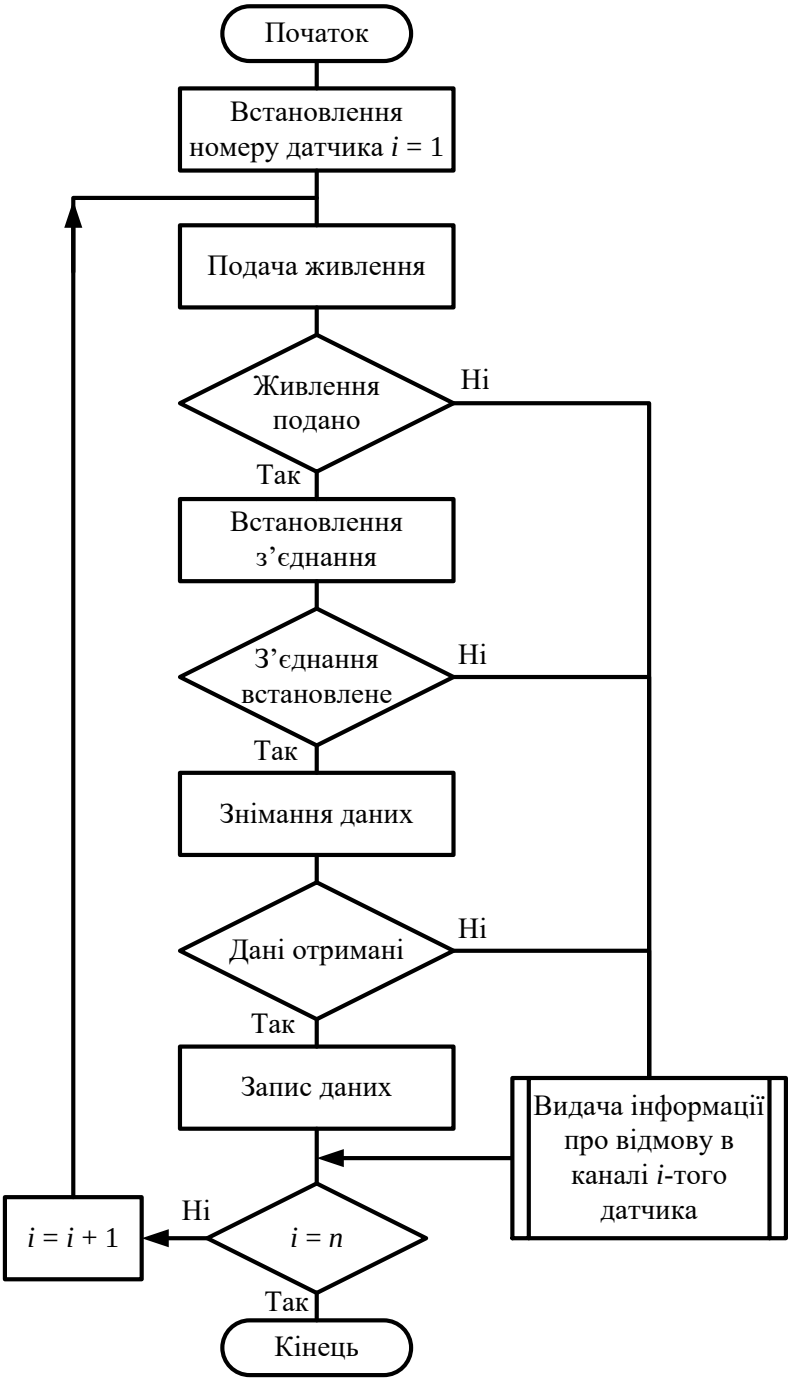

Рис. 2.12 – Алгоритм опитування пасивних датчиків

1) При опитуванні датчиків перш за все встановлюється значення лічильника номеру датчика.

2) Подається живлення і перевіряється сигнал отримання живлення. У випадку його відсутності видається інформація про відмову в каналі *i*-того датчика, за присутності сигналу перехід далі.

3) Аналогічним чином перевіряються встанолення з’єднання та отримання даних з датчика.

4) Далі відбувається процес запису даних. Запис даних в базу даних після проходження через *Arduino Mega* відбувається за рахунок додавання нової інформації в таблиці бази даних.

5) Після цього відбувається перевірка номеру датчика. Якщо $i \neq n$ номер лічильника збільшується на одиницю ($i = i+1$), а якщо оброблено останній датчик ($i = n$), алгоритм закінчує роботу.

2.6. Висновки до розділу

В розділі подано основні структури підсистеми енергозабезпечення: функціональну та структурну схему. Розкрито основні методи розробки підсистеми на основі існуючих технологій.

В розділі проаналізовано основні переваги та недоліки системних плат та датчиків передачі даних:

1) *DHT22, DHT11*;

2) *Arduino Mega*;

3) *RF24401*;

Розроблено алгоритм роботи підсистеми енергозабезпечення з боку апаратної частини системи. Описано встановлення датчиків для сбору даних.

Обрано набір технологій для програмної частини сервісу та інтерфейсу для користувачів.

- структурна схема роботи підсистеми енергозабезпечення;
- функціональна схема роботи підсистеми енергозабезпечення;
- алгоритм роботи підсистеми;
- алгоритм опитування пасивних датчиків.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ ПІДСИСТЕМИ ЕНЕРГОЗАБЕЗПЕЧЕННЯ

3.1. Вибір мови програмування

Вибір мови програмування є дуже важливою частиною реалізації будь якого програмного засобу. Якщо цей програмний засіб це розумний дім, то різноматність програмних інструментів та технологій може призвести до досить великих суперечок. В даному підрозділі я надам свою властну думку, що до доцільності використання тих чи інших мов програмування.

Для аналізу було обрано наступні мови програмування:

- *C++*;
- *C#*;
- *C;*
- *Python*.

Розглянуто переваги та недоліки кожного з них. *C++* в основному сумісна з *C*, будь-який компілятор *C* повинен бути в змозі інтерпретувати або компілювати машиний код для датчиків. Що означає, що можна будет використовувати вбудований асемблер для програмних додатків більш низького рівня.

C++ також поставляється з великою кількістю корисних інструментів, хоча, коли використовується неправильно це дійсно може мати неприємні наслідки. Дана мова належить до більшісті мов програмування, які не тільки дозволяють багато зробити, але і дають можливість виконувати великий набір функцій.

Розглянемо недоліки даної мови:

1) Наявність множини можливостей, які порушують принципи безпеки параметрів призводять до того, що в *C++* програму може легко закрастися важковловима помилка.

2) Погана підтримка модульності. Підключення інтерфейсу зовнішнього модуля через препроцесорну вставку заголовки (*#include*) серйозно уповільнює компіляцію, при підключенні великої кількості модулів.

3) Мова C ++ є складною для вивчення і для компіляції.

Це лише невеличка частина всіх недоліків *C++*, але їх достатньо щоб зрозуміти, для створення підсистеми енергозабезпечення дана мова не підходить.

Мова *C* унікальна з тієї точки зору, що саме вона стала першою мовою високого рівня, всерйоз потіснившим асемблер в розробці системного програмного забезпечення. Вона залишається мовою, реалізованою на максимальній кількості апаратних платформ, і одною з найпопулярніших мов програмування, особливо в світі вільного програмного забезпечення.

Однак не позбавлена ця мова і недоліків: так у ній досить високий поріг входження, що ускладнює її використання в навчанні в якості першої мови програмування.

Те що *C* не є сучасним засобом реалізації програмних додатків досить сильно відлякує від її використання. Оскільки в майбутньому я хочу створити певну систему яка буде аналізувати та приймати рішення самостійно, без втручання користувача в роботу підсистеми енергозабезпечення. *C* не підходить для представленого програмного засобу.

Розглянемо *Python* як мову програмування для інтелектуального будинку. Безсумнівним перевагою є те, що інтерпретатор *Python* реалізований практично на всіх платформах і операційних системах. Першою такою мовою була C, проте її типи даних на різних машинах могли займати різну кількість пам'яті і це

служило деякою перешкодою при написанні коду, дійсно, яку переносять програми. Мові *Python* такий недолік не притаманний.

Наступна важлива риса - розширюваність мови, цьому надається велике значення і, як пише сам автор, мова була задумана саме як розширюваний додаток. Це означає, що є можливість вдосконалення мови всіма усіма зацікавленими програмістами. Інтерпретатор написаний на *C* і вихідний код доступний для будь-яких маніпуляцій. У разі необхідності, можна вставити його в свою програму і використовувати як вбудовану оболонку. Або ж, написавши на *C* свої доповнення до *Python* і скомпілювавши програму, отримати "розширений" інтерпретатор з новими можливостями.

Але *Python* також має низку своїх недоліків. Одним з головних недоліків є його відносно низька швидкість виконання. *Python* є мовою з повною динамічною типізацією, автоматичним управлінням пам'яттю. Якщо на перший погляд це може здаватися перевагою, то при розробці програм з підвищеним вимогою до ефективності, Python може значно програвати за швидкістю своїм статичним братам (*C / C ++, Java, Go*).

З ростом кодової бази (а це часто неминучий процес в успішних проектах), стежити за типом передаються одне одному даних буває дуже складно (а при відсутності виразних доків і тестів практично неможливо), звідси з'являються проблеми, коли, наприклад, у *None* намагаються викликати метод або звернутися до атрибуту в процесі виконання коду.

Через його повільність і проблеми при роботі з даними ми не можемо обрати *Python* для реалізації системи розумного дому.

Виходячи з особливостей мови програмування *C#,* сформулюємо основні переваги даної мови.

- Мова програмування C# претендує на справжню об'єктно-орієнтованість

- Компонентно-орієнтований підхід до програмування, сприяє меншій машинно-архітектурній залежності результуючого програмного коду, гнучкості, переносимості і легкості повторного використання (фрагментів) програм;
- Орієнтація на безпеку коду (в порівнянні з *C* і *C* ++);
- Уніфікована система типізації;
- Розширена підтримка подієво-орієнтованого програмування.
- Незважаючи на переваги, мова *C#* має деякі недоліки, такі як:
- Досить складний синтаксис (75% з *Java*, 10% з *C++*, 5% з *Visual Basic*);
- Мало свіжих концептуальних ідей (приблизно менш ніж 10% конструкцій мови);
- Відносно невисока продуктивність, набагато повільніше,ніж мова *C*;
- Так як *C#* розроблений компанією *Microsoft*, то і працює він тільки під операційною системою *Windows*, хоча в даний момент вже розробляється крос-платформна версія даної мови.

Саме *C#* було обрано для реалізації підсистеми енергозабезпечення. Дивлячись на переваги та недоліки запропонованих мов програмування. *C#* для створення такої підсистеми є, як кажуть, меншим злом серед усіх.

3.2. Опис програмних модулів

Весь функціонал системи знаходиться в плагінах. Якщо вам чогось не вистачає, ви можете легко написати власний плагін, який буде взаємодіяти з потрібним залізом, інтернет-сервісами або робити що-небудь ще. Щоб полегшити написання власних плагінів, я створив невеликий приклад, який можна використовувати.

Цей демо-плагін збирає інформацію з датчиків температури та вологості *DHP22* і відображає отриману інформацію в веб-інтерфейсі. Результат виглядає приблизно так, як показано на рис. 3.1.

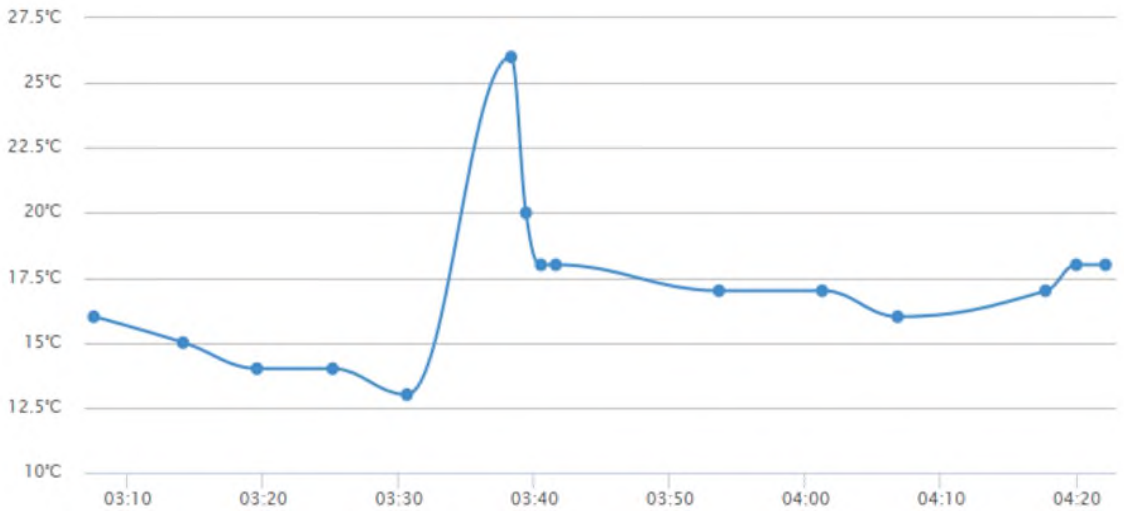

Рис. 3.1. – результат роботи датчика *BHP22*.

Опис процесу розробки плагіна для розумного дому складається з наступних дій.

1. Налаштування середовища і створення заготовки проекту
2. Створення таблиці в БД для зберігання даних.
3. Створення потрібної структури
4. Міграція для таблиці даних.
5. Перевірити додавання записів про датчики і отримання інформації з БД

Налаштування середовища і створення заготовки проекту

Отже, підсистема, яку ми пишемо - це плагін для програми – *windows-сервісу*. Відповідно, спочатку потрібно розгорнути на комп'ютері сервіс, до якого буде підключатися наш плагін. Зробити це дуже просто: потрібно завантажити інсталятор, запустити його і кілька разів натиснути «Далі». Під час настроювання не запитуються ніякі параметри. Сервіс встановлюється в папку *C:\Program Files(x86)\ThinkingHome\service*.

Тепер створимо в *Visual Studio* порожній *C#* проект *Class Library*, при створенні вибираємо *.NET Framework 4.5*. В принципі, підійде не тільки *VS*, але і будь-яка інша *IDE*, наприклад, безкоштовна *Xamarin Studio* (ну і, звичайно ж можна використовувати безкоштовну *Visual Studio Express*), рис.3.2.

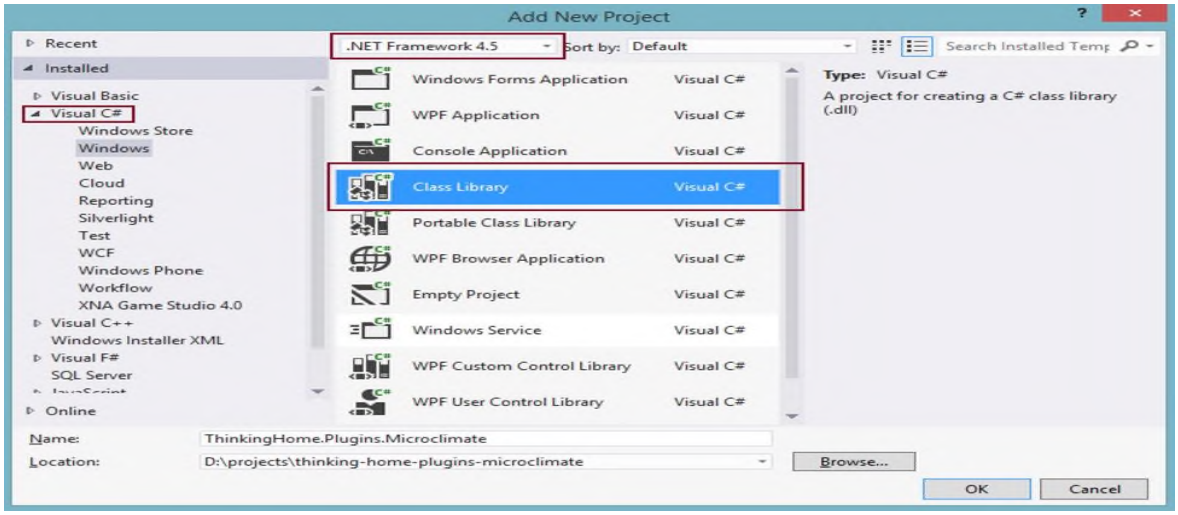

Рис. 3.2 – Приклад створення проекту для ІД.

Тепер потрібно додати в проект посилання на бібліотеку *ThinkingHome.Core.Plugins*, в якій містяться базові класи для плагінів. Найпростіший спосіб зробити це – підключити її через менеджер пакетів *NuGet*. Просто потрібно набрати в консолі менеджера пакетів:

Install-Package ThinkingHome.Core.Plugins

Створюємо клас *MicroclimatePlugin*, успадковуємо його від базового класу *ThinkingHome.Core.Plugins.PluginBase* і помічаємо атрибутом *[ThinkingHome.Core.Plugins.PluginAttribute]*. Клас *PluginBase* реалізує базовий функціонал плагіна, а атрибут потрібен для підключення плагіна до сервісу через *MEF*.

```
[Plugin]
public class MicroclimatePlugin: PluginBase
{
// цей метод викликається, коли сервіс завантажує плагіни
public override void InitPlugin ()
{
Logger.Debug ("init");
base.InitPlugin ();
}
// викликається після того, як всі плагіни ініціалізовані
```

```
public override void StartPlugin ()
{
Logger.Debug ("start");
base.StartPlugin ();
}
// викликається при зупинці сервісу
public override void StopPlugin ()
{
Logger.Debug ("stop");
base.StopPlugin ();
}
}
```

Тепер інтеграція методів базового класу, щоб додати власну логіку в плагін. Як бачите, ми поки просто додали запис повідомлень в лог.

Підключення плагіну до сервісу. У папці, куди був встановлений сервіс (*C:\Program Files(x86)\ThinkingHome\service*), є папка *Plugins*, з якої завантажуються плагіни при старті сервісу. Файли кожного з полігонів повинні лежати в окремій папці. Створюємо там папку для нашого плагіна (наприклад, назвемо її «*ThinkingHome.Plugins.Microclimate*») і у властивостях проекту встановимо для параметра *Output Path* значення «*C:\ ProgramFiles(x86)\ThinkingHome\service\Plugins\ThinkingHome.Plugins.Microclimate*»

Тепер компілюємо проекти і бачимо, що в зазначеній нами папці з'явилася *DLL* з нашим плагіном, показано на рис. 3.3.

Оскільки в процесі розробки і налагодження ми будемо часто запускати і зупиняти сервіс, має сенс відключити його автоматичний запуск і запускати його як консольний додаток. В папці з сервісом є файл *ThinkingHome.TestConsole.exe*. Запускаємо його (з правами адміністратора), результат показано на рис. 3.4.

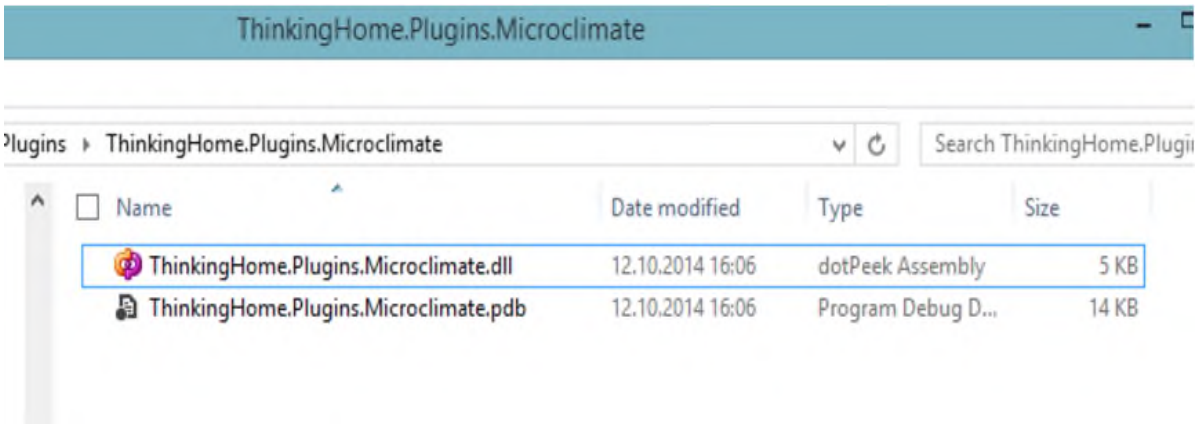

Рис. 3.3 – *Dll* з створении плагіном.

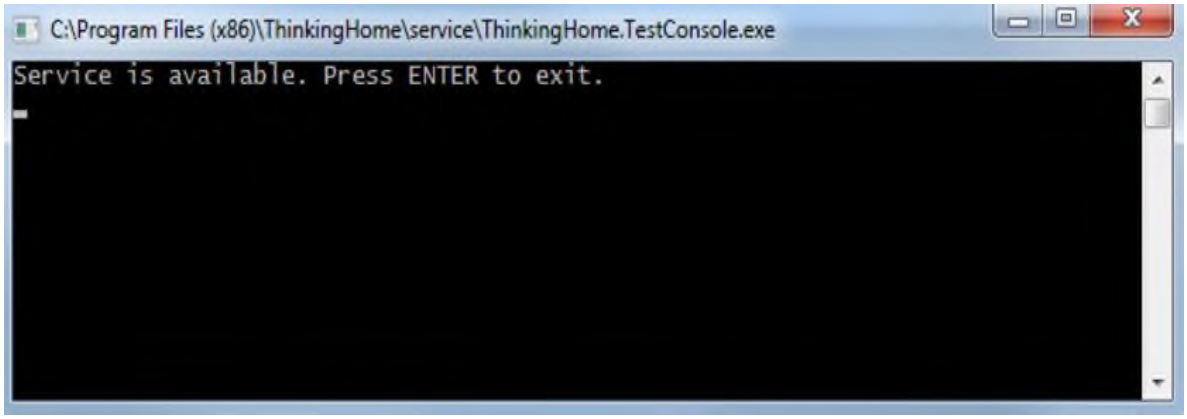

Рис. 3.4 – Запуск сервісу для перевірки роботи.

Закриваємо консоль, переходим в папку з модулями (*C:\Program Files(x86)\ ThinkingHome\service\Logs*) і дивимося файл *ThinkingHome.Plugins. Microclimate.MicroclimatePlugin.log*.

Там буде показано такі рядки:

2017-05-12 16: 40: 07.0981, *Info*, *init*

2017-05-12 16: 40: 07.1132, *Info*, *start*

2017-05-12 16: 40: 52.1292, *Info*, *stop*

Таким чином, ми встановили сервіс, створили заготовку плагіна і перевірили, що він успішно підключається до сервісу.

Таблиці в БД для зберігання даних.

Якщо плагін, підключений до системи, можна зберігати свої дані в системній БД (*MS SQL Server CE 4*). Засоби для роботи з БД надає базовий клас *PluginBase*.

Структура БД показана на рик. 3.5. Саме цю БД ми будемо використовувати для реалізації поставлених програмних модулів.

Скаладається БД з двух таблиць: *TempetatureSensor* та *TemperatureData*. В кожній таблиці є пункт *Id* для знаходження потрібних даних та датчиків. Саме в ньому відбувається ініціалізація запропонованих мікроконтролерів. Все це утворює з апаратною частиною чітку та зрозумілу комунікацію.

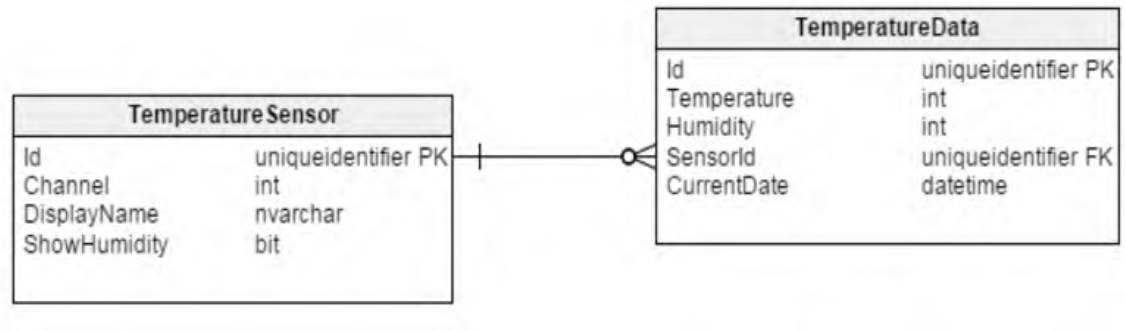

Рис. 3.5 – структура бази даниз запропонованого проекту.

Отже, ми плануємо отримувати інформацію з датчиків температури та вологості *DHP22*. Кожен датчик буде прив'язаний до якогось каналу *USB*-адаптера *Arduino Mega* і буде періодично відправляти йому дані про поточну температуру та вологість.

Таким чином, у нас в системі повинен бути список датчиків, для кожного з яких повинен бути вказаний канал адаптера, на який відправляються дані і, як мінімум, назва датчика для відображення його в веб-інтерфейсі.

Також нам потрібна ще одна сутність для зберігання інформації з датчика в заданий момент часу. Відповідно, вона буде мати поля: «значення температури», «значення вологості», «поточний час», «*ID* датчика»

Створення таблиць.

Для автоматичного створення потрібної структури БД в проекті використовується інструмент *ECM7.Migrator*. Модулі можуть містити міграції – невеликі класи на *C#* описують порції змін БД. Кожна міграція має номер версії БД і мігратор може оновити БД до останньої версії з будь-якого попереднього

стану. Просто додайте міграції в проект з плагіном і вони будуть автоматично виконані при старті сервісу.

Для початку потрібно додати в проект посилання на бібліотеку *ECM7.Migrator.Framework*. Знову ж, найпростіший спосіб це зробити - через *NuGet*. Наберіть в консолі менеджера пакетів:

Install-Package ECM7Migrator

Не забуваємо встановлювати для всіх збірок, що додаються до проекту, параметр *Copy Local = False*.

Далі необхідно позначити всю збірку атрибутом *MigrationAssembly* (наприклад, у файлі *AssemblyInfo.cs*):

[*Assembly: MigrationAssembly ("ThinkingHome.Plugins.Microclimate*")]

Цей атрибут потрібен для того, щоб облік версій БД для нашого плагіна виконувався незалежно від інших плагінів. Як параметр необхідно передати деякий рядок – унікальний ключ, який не повторюватиметься в інших плагінах. Рекомендується використовувати для цього повну назву плагіна.

Після цього додаємо міграції, що описують зміни БД. Як вже було сказано, кожна міграція - це окремий клас. Він повинен бути успадкований від спеціального базового класу *Migration* та він повинен бути позначений спеціальним атрибутом *MigrationAttribute*, в якому в якості параметра переданий унікальний номер версії БД.

```
using ECM7.Migrator.Framework;
    . . .
    namespace ThinkingHome.Plugins.Microclimate.Migrations
{
    [Migration (1)]
    public class Migration01_TemperatureSensor: Migration
{
    public override void Apply ()
{
```

```
Database.AddTable ("Microclimate_TemperatureSensor",
    new Column ("Id", DbType.Guid, ColumnProperty.PrimaryKey, "newid ()"),
    new Column ("Channel", DbType.Int32, ColumnProperty.NotNull),
    new Column ("DisplayName", DbType.String.WithSize (255), ColumnProperty.NotNull),
    new Column ("ShowHumidity", DbType.Boolean, ColumnProperty.NotNull, false)
    );
}
    public override void Revert ()
{
    Database.RemoveTable ("Microclimate_TemperatureSensor");
    }
    }
}
```

Тут перевизначаються методи *Apply* і *Revert* для базового класу. *Apply* - оновлення БД до версії, зазначеної в параметрі атрибута [*Migration*]. *Revert* - відкат змін. Властивість *Database* базового класу містить спеціальний об'єкт, що надає *API* для виконання різних операцій над БД. *API* має функціонал для виконання всіх основних операцій з БД та на крайній випадок там є метод *ExecuteNonQuery*, за допомогою якого можна виконати довільний *SQL* запит.

Міграція для таблиці даних.

Ця міграція, крім таблиці, додає ще й зовнішній ключ. При відкаті видаляти його не обов'язково – мігратор при видаленні таблиці сам видалить її зовнішні ключі.

Для перевірки скомпілюємо збірку і запустимо тестову консоль.

В папці з логами дивимося вміст файлу *ecm7-migrator-logger.log* і бачимо результат роботи сервісу. Також зверніть увагу, що назви властивостей моделі

збігаються з назвами полів таблиць, але для поля *SensorId*, що є посиланням на таблицю датчиків, описано властивість *Sensor* (без закінчення «*Id*»), тип якого відповідає типу моделі для пов'язаної таблиці.

Меппінг моделі на таблиці БД.

Для визначення відповідності моделі таблиці БД треба перевизначити в плагіні метод *InitDbModel* з базового класу.

```
public override void InitDbModel (ModelMapper mapper)
{
    mapper.Class <TemperatureSensor> (cfg => cfg.Table ( "Microclimate_TemperatureSensor"));
    mapper.Class <TemperatureData> (cfg => cfg.Table ( "Microclimate_TemperatureData"));
}
```

В якості вхідного параметра сюди передається екземпляр класу *NHibernate.Mapping.ByCode.ModelMapper*. За замовчуванням, меппер вважає, що назви полів таблиць співпадають з назвами властивостей класів, а назви полів-посилань на інші таблиці відповідають однойменним властивостями, але без закінчення *Id*. Таким чином, в нашому випадку досить задати лише відповідність класів таблиць в БД - інше *NHibernate* налаштує самостійно. Природно, *ModelMapper* надає функціонал, за допомогою яких можна налаштувати будь-який інший, більш складний меппінг моделі на таблиці БД.

Додавання записів про датчики і отримання інформації з БД

Щоб додати в БД запис про датчику або отримати з БД його дані необхідно створити спеціальинй об'єкт - сесію *NHibernate* (це щось схоже на *DbConnection* в *ADO.NET*, які було вивчено за університетською програмою). Базовий клас плагіна має властивість *Context*, що містить контекст програми - об'єкт, який реалізує інтерфейс *IServiceContext*. Створити сесію *NHibernate* можна за допомогою його методу OpenSession.

3.3. Тестування підсистеми

Тестування підсистеми важлива частина її створення, саме тут буде показано працездатність розробленого додатку, та його інтерфейс. Для тестування в даному дипломному проекті вирішено розглянути головні вікна програмного додатку та описати їх функціонал.

За рахунок того, що програмний засіб створювався мовою програмування *C#,* створення інтерфейсу проведено в *Microsoft Visual Studio*

На рис. 3.6. зображено головне меню користувача, який може ввімкнути світло, або вимкнути його залежно від умов середовища, звичайно, логіка роботи підсистеми енергозабезпечення дає змогу виконувати такі дії без втручання користувача, але мешканець може мати певні цілі які будуть потребувати швидкого переключення світла чи інших умовних параметрів. Кнопка *Add script* дає нам змогу додавати сценарії роботи підсистеми через інтерфейс. На жаль реалізувати конструкторний варіант додавання не вдалося, саме тому після натискання кнопки *Add Script* відкривається вікно з вихідним кодом, що не є досить прийнятним для користувача системи інтелектуального дому в цілому.

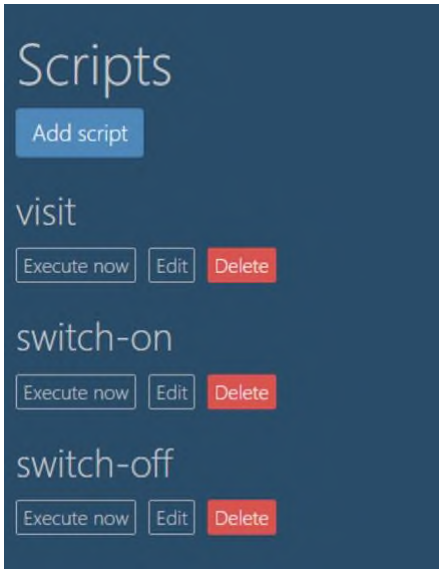

Рис. 3.6 – Сторінка опцій користувача.

Розроблена підсистема є невдалою с точки зору комунікації прогорамного засобу та користувача. Для користування все ж треба знати програмування.

Так в розділі *switch-on* при натисканні вкладки *Edit* показані не конструктивнні опції, а вихідних програмний код. В якому при збережені даних пишеться тивіальна фраза *hello, world*! Та вмикається світло в певній кімнаті. На рис. 3.7. показана робота кнопки *Edit* та її вихідний код для корегування інформації перемикача світла.

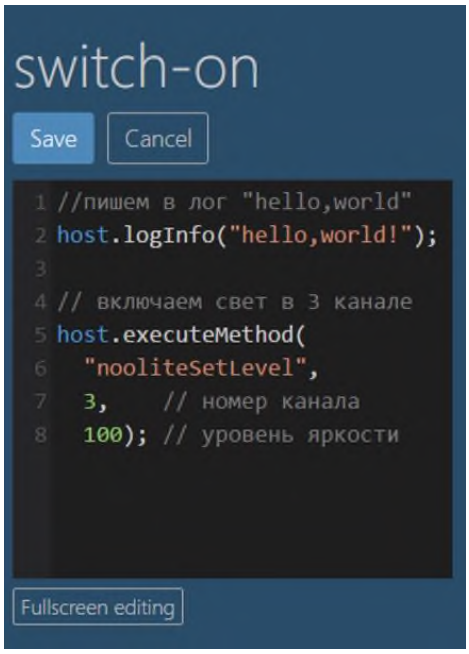

Рис. 3.7 – Вкладка *Edit* в пункті *Switch-on*.

При натисканні кнопки *Execute now.* на екрані комп'ютера буде показано меню (рис 3.8).

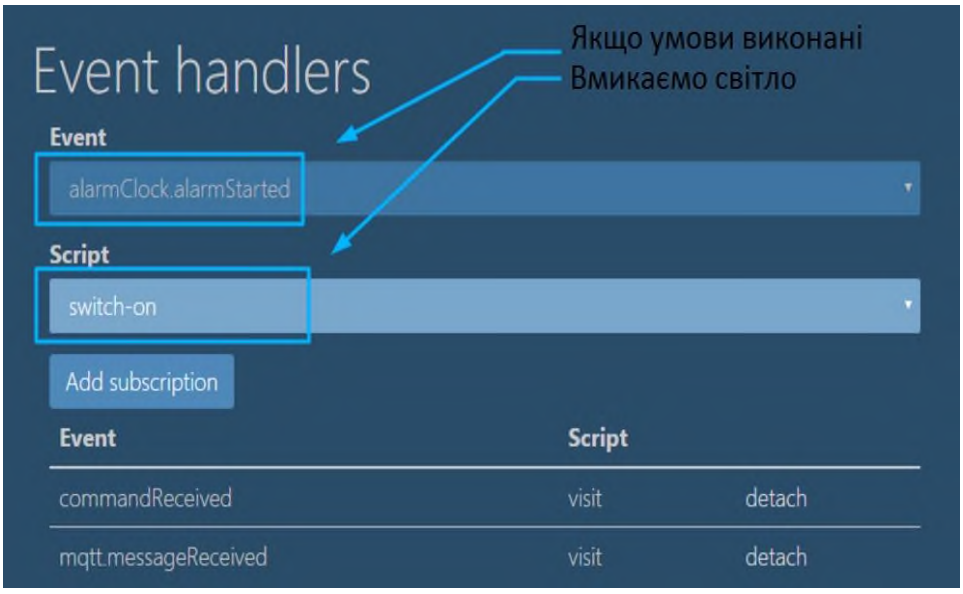

Рис. 3.8 – вкладки *Execute Now.*

Виконуючі певні умови записані в сценарії ми отримуємо зміну параметрів середовища, залежно від поставлених цілей підсистеми енергозабезпечення.

Тобто, якщо нам потрібно вимикати все світло в будинку коли на годинику 00.00. Ми можемо просто прописати це в модулі *Event*.

За рахунок цього і реалізована система енергозабезпечення. При зміні навколишніх параметрів, на певні – не прийнятні, підсистема робить дії, що допомогають нам в економії витрат. Такий підхід є досить тривіальний, та найдійний. Але проблема в тому, що базуючись на такій логіці буде дуже важко розвинути систему до повністью автоматичного рівня. Також буде практично не можливо застосовувати підходи оптимізації.

При натисканні клавіші *Add script* ми отримуєсо наступну картину роботи. Додаток запропононує ввести назву сценарію та прив'язати до нього скрипт заздалегіть написаний у програмному коді, що розглядайться в підрозділі 3.2. Після цих дій можна зберегти, або відмінити виконані дії.

Проблема такого підходи полягає в тому, що якщо ми не пропишемо скрипт для реалізації події наша підсистема працювати не буде. Це все потрібно реалізовувати та моделювати в програмному коді, до якого, повторюсь ще раз не має доступу користувач. Через обмежені можливості в програмуванні. Конструкторна реалізація додатку не була виконана до кінця. Через це, виникають певні не закриті питання:

1) Як саме користувач буде додавати бажані події;
2) Як саме додані події вплинуть на вже ралізовані;
3) Чи коректно були підібрані датчики для данної підсистеми;
4) Як даний програмний засіб буде працювати в різних умовах експлуатації.

Логіку додавання програмних сценаріїв показано на рис. 3.9. Можна побачити, що запропонований скрипт світиться сірим кольором, це свідчить про те, що він не активний та не може буди доданий до функціоналу інтелектуального будинку. *Display name* можна прописувати на будь якій

доступній мові, за це треба відзначити *Visual Studio* та *C#.* Такої можливості я ще не бачив в жодній платформі для програмування та мові.

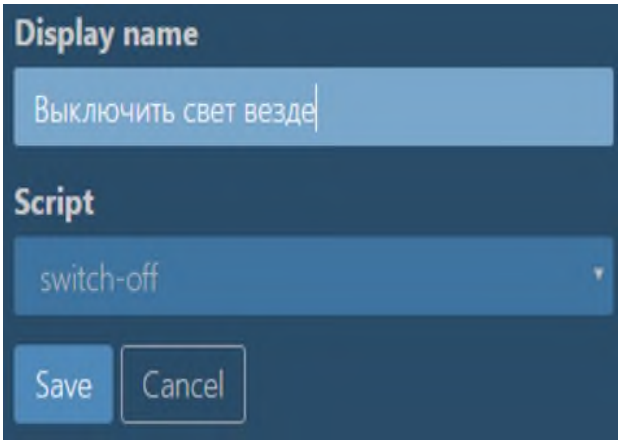

Рис. 3.9 – вкладка з назвою *Add Script.*

Проблема такого підходи полягає в тому, що якщо ми не пропишемо скрипт для реалізації події наша підсистема працювати не буде. Припустимо, що функцію запрограмовано та реалізовано, після натискання клавіші *Save* отримаємо наступний розділ, показаний та малюнку 3.10. Слід зазначити, що збереження відбувається відразу в базі даних, навіти якщо функції не запрограмовані в логічному модулі.

Рис. 3.10 – вкладка клавіші *Save.*

Додаток запропонує обрати розділ дій для зберігання енергії а саме пункт *Alarm* для зберігання електронергії та *Weather* для запобігання надмірного використання електроенергії. Пункт *My plugin* це кнопка за рахунок якої можна додати такий сценарій наприклад енергозабезпечення за рахунок освітлення квартири.

На жаль, пункт який ми захочено додати при інтегрування нового збереження також повинен міститися у програмному коді. Це дуже важливо оскільки ми зможемо додати пункт, але не зможемо виконати його. Наприклад, якщо залишито домівку та додави пункт про вимкнення світла, можна залишити будинок з вімкненим світлом, бо центральний модуль просто не буде знати, що саме робити з запропонованим сценарієм. На рис. 3.11 показано сторінку на яку буде направлено користувача після натискнення певного пункту.

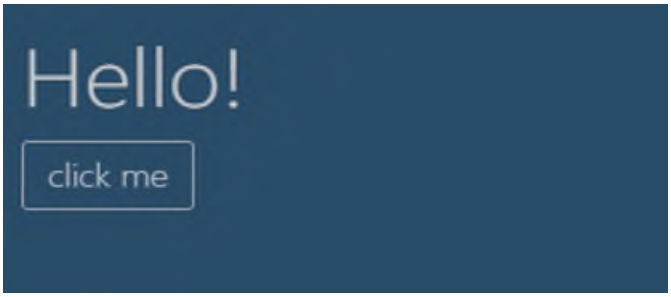

Рис. 3.11 – сторінка після інтеграції функціоналу.

Як висновок тестуванню роботи підсистеми можна сказати, що розроблений додаток містить ряд неточностей:

1) Додавання клавіш, що не виконуються;
2) Потрібно прописувати практично всі дії в програмному коді;
3) Не зрозумілий порядок пунктів в меня *Execute now*.

Саме тому, запропонована реалізація потребує покращення та не може бути використаною користувачем, що не розуміється в програмуванні. Саме тому, розроблений додаток не є коректним для взаємозв’язку користувача та центрального програмного модулю.

Слід зазначити, що для розробника створений підхід доволі простой, оскільки весь функціонал, що буде додаватися можна перевірити в програмному коді та додати такі опції, що не були би передбачені конструктором. Це не є явно позитивна частина розробленого програмного засобу підсистеми енергозбережння, але все ж для майбутньої оптимізації даного дипломного проекту було вирішено працювати з додатком саме в такому вигляді. Проблема такого підходи полягає в тому, що якщо ми не пропишемо скрипт для реалізації

події наша підсистема працювати не буде. А перейти до створення конструктора, зрозумілому для пересічних користувачів, можна буде вже після остаточної розробки підсистеми орієнтованої на різні локації та різних користувачів. Найголовніше, це те що функціонал можна додавати, можна отпримати комунікацію з центральним модулем, та можна видалята вже існуючі опції.

3.4. Висновки до розділу.

Дипломний проект присвячений актуальній тематиці обслуговування мешканців будинків за допомогою підсистеми енергозабезпечення.

Під час виконання програмної ралізації дипломного проекту було розроблено:

1) обрану структуру роботи підсистеми;
2) логіку роботи підсистеми;
3) модуль сбору даних;
4) модуль системи зв’язку;

Створено:

1) інтерфейс системи;
2) базу даних для збереження інформації.
3) оформлені сторінки для користувачів.

При реалізації підсистеми клімат контролю було використано стек технологій, який дав змогу нам створити сервіс. Дану систему розроблено за допомогою *Arduino Mega*.

Впровадження розробленого сервісу дозволяє добитися економічного ефекту та зберегти час мешканців будинку за рахунок:

1) спрощення процесу передачі та доступу до інформації;
2) зниження витрат за рахунок модуля роботи енергозабезпечення;
3) дистанційну комунікацію користувача і підсистеми.

Створену систему можуть використовувати як мешканці однокімнатних квартир багатоповерхівок, так і житель великих дачних будків.

ВИСНОВКИ

Дипломний проект присвячений актуальній тематиці створення підсистеми енергозабезпечення в системі «Розумний дім».

Під час виконання дипломного проекту виконано:

1) існуючі варіанти створення підсистеми;
2) датчики для апаратного рівня;
3) мови програмування для створення логіки системи;

У дипломному проекті розроблено наступне.

1) Структурну та функціональну схеми;
2) Алгоритм передачі даних та роботи підсистеми.
3) Програмний плагін та інтерфейс.

При реалізації підсистеми енергозабезпечення було використано стек технологій, який дав змогу нам створити сервіс. Дану систему розроблено за допомогою *Arduino Mega.*

Впровадження розробленого сервісу дозволяє добитися економічного ефекту та зберегти час мешканців будинку за рахунок:

1) спрощення процесу передачі та доступу до інформації;
2) зниження витрат за рахунок модуля роботи енергозабезпечення;
3) дистанційну комунікацію користувача і підсистеми.

Матеріали дипломного проекту рекомендуються використовувати в будинках та квартирах при обслуговуванні жителів та у навчальному процесі.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1) *Dover D., Dafforn E. Search Engine Optimization Secrets. Indianapolis: Wiley Publishing, Inc*., 2011. 456 p.

2) Томас Коннолли. Базы данных. Проектирование, реализация и сопровождение. Теория и практика.

3) *J. Gosling, B. Joy, G. Steele, G. Brachda.* The Java Language Specification, 2nd Edition2006. - 256 с.

4) Стивен Хольцнер *HTML5* за 10 минут [пятого издание] [Пер. с англ.] / М .: Издательский дом «Вильямс», 2011. - 240 с.

5) *Best Home Automation System – Consumer Reports". www.consumerreports.org. Retrieved* 2016-02-14- 992 с. - *ISBN* 978-5-93286-210-0.

6) *What is smart home - Basic Idea". cctvinstitute.co.uk. Retrieved* 2010. - 864 с - *ISBN* 978-5-93286-157-8

7) *Brush, A. J.; Lee, Bongshin; Mahajan, Ratul; Agarwal, Sharad; Saroiu, Stefan; Dixon, Colin (2011-05-01). "Home Automation in the Wild: Challenges and Opportunities". Microsoft Researc*.

8) Интелектуальний дом и разработка системных решений по енергозбережению температуры (23 ноября 2012 г. № 143-ПЗ) 2015. - 720 с.

9) *Smart home and some function for his realization. smartinhouse.co.ua. Retrieved* 2014. - 512 с - *ISBN* 123-1-12342-667-9

10) Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного Авіаційного Університету. НАУ, 2017 р.

11) ГОСТ .19.701-90 ЕСКД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения

12) ГОСТ 19.003-80 ЕСКД. Схемы алгоритмов и программ. Обозначения условные графические

ДОДАТОК

Лістинг вихідних кодів

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SmartHouse.Eqipment;

namespace SmartHouse
{
  internal static class Home
  {
    public static List<BaseEquipment> equipments = new List<BaseEquipment>();

    private static int index = 0;

    public static void AddToList(BaseEquipment equip)
    {
      equip.Id = index;
      equipments.Add(equip);
      SetName(index);
      index++;
    }

    public static void GetList()
    {
      Console.Clear();
      Console.WriteLine("Objects list.");
```

```
  //Console.WriteLine("Amount of objects is - {0}", equipments.Count);

  foreach (BaseEquipment equipment in equipments)
  {
    if (equipment != null)
    {
      equipment.Status();
    }
  }
}

public static void SetName(int id)
{
  Console.Clear();
  Console.Write("Enter the name - ");
  string name = Console.ReadLine();

  while (String.IsNullOrEmpty(name))
  {
    Console.WriteLine("You do not enter the title!\nTry again");
    name = Console.ReadLine();
  }

  equipments[id].Name = name;
}

public static void SetState(int id)
{
  Console.Clear();
  Console.WriteLine("Press 1 to turn on or 0 to turn off");
  Console.Write("\n: ");
```

```
  int state = 0;
  string input = Console.ReadLine();

  if (Int32.TryParse(input, out state))
  {
    if (state == 0)
    {
      equipments[id].State = false;
    }
    else if(state == 1)
    {
      equipments[id].State = true;
      if (equipments[id].GetType() == typeof(Lamp))
      {
        ((Lamp)equipments[id]).Brightness = 100;
      }
    }
    else
    {
      Console.WriteLine("Incorrect value! \nTurning off");
    }
  }
}

public static bool CheckId(int id)
{
  bool check = false;

  try
  {
```

```
        if (equipments[id] != null)
        {
        check = true;
        }
      }
      catch
      {
        Console.WriteLine("Incorrect ID.");
        check = false;
        return check;
      }
      return check;
    }

    public static void Delete(int id)
    {
      equipments[id] = null;
      //equipments.RemoveAt(id);
    }
  }
}
```

Модуль *runnertip*

```
using System;
using SmartHouse.Eqipment;

namespace SmartHouse
{
  internal class CreateMenu
  {
    public static void SelectionObjectType()
```

```
{
  Console.Clear();
  Console.WriteLine("Select object type");
  Console.WriteLine("1. Lamp.");
  Console.WriteLine("2. Radio.");
  Console.WriteLine("3. Conditioner.");
  Console.WriteLine("4. Boiler.");
  Console.WriteLine("5. Main Menu.");
  Console.Write("\n: ");

  int choice = 0;
  string input = Console.ReadLine();

  if (Int32.TryParse(input, out choice))
  {
   switch (choice)
   {
     case 1:
       Home.AddToList(new Lamp());
       Console.WriteLine("lamp successfully created\nPress Enter to continue");
       break;
     case 2:
       Home.AddToList(new Radio());
       Console.WriteLine("Radio successfully created\nPress Enter to continue");
       break;
     case 3:
       Home.AddToList(new Conditioner());
       Console.WriteLine("Conditioner successfully created\nPress Enter to continue");
       break;
     case 4:
       Home.AddToList(new Boiler());
```

```
            Console.WriteLine("Boiler successfully created\nPress Enter to continue");
            break;
          case 5:
            MainMenu.ChoiceMenu();
            break;
          default:
            Console.WriteLine("Incorrect input \nPress any key to continue");
            Console.ReadLine();
            MainMenu.ChoiceMenu();
            break;
        }

        Console.ReadKey();
        MainMenu.ChoiceMenu();
      }
      else
      {
        Console.WriteLine("Incorrect input \nPress any key to continue");
        Console.ReadLine();
        MainMenu.ChoiceMenu();
      }
    }
  }
}

using System;
using SmartHouse.Eqipment;

namespace SmartHouse
{
  internal class MainMenu
```

```
{
  public static void EntryPoint()
  {
    Console.Clear();
    Console.WriteLine("Welcome to the SmartHouse!");
    Console.WriteLine("1. Settings.");
    Console.WriteLine("2. Exit.");
    Console.Write("\n: ");

    int choice = 0;

    while (choice != 2)
    {
      string input = Console.ReadLine();
      if (Int32.TryParse(input, out choice))
      {
        switch (choice)
        {
          case 1:
            ChoiceMenu();
            break;
          case 2:
            Console.WriteLine("Press any key to exit");
            break;
          default:
            Console.WriteLine("You enter invalid value\nTry again");
            break;
        }
      }

      Console.ReadKey();
```

```
  }
}

public static void ChoiceMenu()
{
  Console.Clear();
  Console.WriteLine("Main Menu");
  Console.WriteLine("1. Create object.");
  Console.WriteLine("2. Object Preview.");
  Console.WriteLine("3. Modify objecst.");
  Console.WriteLine("4. Remove objects.");
  Console.WriteLine("5. Quit programm.");
  Console.Write("\n: ");

  int choice = 0;
  string input = Console.ReadLine();
  if (Int32.TryParse(input, out choice))
  {
    switch (choice)
    {
      case 1:
        CreateMenu.SelectionObjectType();
        break;
      case 2:
        Home.GetList();

        Console.WriteLine("\nPress Enter to continue");
        Console.ReadLine();
        MainMenu.ChoiceMenu();
        break;
      case 3:
```

```
            PropertyMenu.SelectionId();
            break;
          case 4:
            DeleteMenu.SelectionId();
            break;
          case 5:
            EntryPoint();
            break;
          default:
            break;
        }
      }
      else
      {
        Console.WriteLine("Incorrect input");
      }
    }
  }
}
```

Модуль *Energysolv*

```
using System;
using SmartHouse.Eqipment;

namespace SmartHouse
{
  internal class PropertyMenu
  {
    public static void SelectionId()
    {
```

```
    Home.GetList();

    Console.WriteLine("\nChoice objects ID, to change property.\nPress M to go to Main Menu .");
    Console.Write("\n: ");

    int id = 0;
    string input = Console.ReadLine();

    if (Int32.TryParse(input, out id))
    {
      if (Home.CheckId(id))
      {
        PropertySelection(id);
      }
      else
      {
        Console.WriteLine("Incorrect input \nPress any key to continue");
        Console.ReadLine();
        MainMenu.ChoiceMenu();

      }
    }
    else
    {
      MainMenu.ChoiceMenu();
    }
  }

  public static void PropertySelection(int id)
  {
```

```
int choice = 0;
while (choice != 9)
{
  Console.Clear();
  Console.WriteLine("Property list.");
  Console.WriteLine("1. Turn on or turn off.");
  Console.WriteLine("2. Change name.");

  if (Home.equipments[id].GetType() == typeof(Lamp))
  {
    Console.WriteLine("3. Change brightness.");
  }

  Console.WriteLine("9. Back to previus menu.");
  Console.Write("\n: ");

  string input = Console.ReadLine();

  if (Int32.TryParse(input, out choice))
  {
    switch (choice)
    {
      case 1:
        Home.SetState(id);
        break;
      case 2:
        Home.SetName(id);
        break;
      case 3:
        if (Home.equipments[id].GetType() == typeof(Lamp))
        {
```

```
            ((Lamp)Home.equipments[id]).SetBrightness();
          }
          else
          {
            Console.WriteLine("Incorrect input \nPlease try again");
            Console.Write("\n: ");
          }
          break;
        case 9:
          SelectionId();
          break;
      }
    }
    else
    {
      Console.WriteLine("Incorrect input \nPlease try again");
      Console.Write("\n: ");
    }
  }
 }
}
```