Міністерство освіти і науки України
ДВНЗ «Прикарпатський національний університет імені Василя Стефаника»
Кафедра комп'ютерної інженерії та електроніки
(повна назва кафедри)

Хромишин Назарій Іванович
Khromyshyn Nazarii

УДК 004:681.5

Спеціальність 6.050102 «комп'ютерна інженерія»
(шифр та назва спеціальності)

Кваліфікаційна робота
на здобуття освітньо-кваліфікаційного рівня бакалавр
(бакалавр, спеціаліст, магістр)

Проектування голосової системи керування елементами "розумного будинку"
Designing of Voice Operation System of the Smart House Elements

Науковий керівник:
кандидат технічних наук,
доцент Грига В.М.

Рецензент:
Кандидат фіз.-мат. наук,
професор кафедри фізики і
хімії твердого тіла
Никируй Л.І.

Івано-Франківськ
2020

Форм.	Зона	Поз.	Позначення	Найменування	К-ть	Прим.
			6.050102. УДК 004:681.5	Пояснювальна записка	68	

					6.050102. УДК 004:681.5		
Змн.	*Арк.*	*№ докум.*	*Підп.*	*Дата*			
Разробив		Хромишин Н.І.			Специфікація	*Літ.*	*Арк.*	*Аркушів*
Перевірив		Грига В.М.					2	68
Н. Конт.		Грига В.М.						
Затвердив		Когут І.Т.						

АНОТАЦІЯ

В бакалаврській кваліфікаційній роботі розроблено спеціалізовану програмно-апаратну систему голосового керування елементами ”розумного будинку” на базі реалізованого веб-сервера Amazon Alexa та мікропроцесорного модуля Node MCU ESP8266.

У пояснювальній записці проведено аналіз технологій ”Інтернет речей”, описано принципи роботи та способи керування системами “розумного будинку”, розглянуто існуючі голосові технології та визначено їхні позитивні сторони і недоліки, обґрунтовано вибір засобів розробки програмного і апаратного забезпечення, спроектовано узагальнену структурну схему системи керування, розроблено програмне забезпечення для веб-сервера і апаратного модуля Node MCU ESP8266, проведено його тестування та розраховано економічну ціну реалізації системи.

Загальний обсяг роботи – 68 сторінок, 29 рисунків, 3 таблиці, 17 посилань.

Ключові слова: ”Інтернет речей”, ”розумний будинок”, голосові асистенти, веб-сервер, мікропроцесорний модуль Node MCU ESP8266, блок-схема алгоритму.

ABSTRACT

In the bachelor's works, a specialized software and hardware system for voice control of "smart home" elements was developed based on the implemented Amazon Alexa web server and the Node MCU ESP8266 microprocessor module.

The explanatory note analyzes the technologies of the Internet of Things, describes the operating principles and methods of controlling the "smart home" systems, examines existing voice technologies and identifies their positives and disadvantages, justified the choice of software and hardware development tools, designed a generalized structural diagram of the system , software was developed for the web server and the Node MCU ESP8266 hardware module, tested, and cost-effective to implement the system.

Total volume of work - 68 pages, 29 drawings, 3 tables, 17 links.

Key words: ”Internet of Things”, ”Smart house”, voice assistants, web-server, microprocessor module Node MCU ESP8266.

Міністерство освіти і науки України
Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Фізико-технічний факультет
Кафедра «Комп'ютерної інженерії та електроніки»

Пояснювальна записка

до кваліфікаційної роботи на тему:

Проектування голосової системи керування елементами "розумного будинку"

					6.050102.КІ-41.27		
Змін.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		*Хромишин Н.І.*			*Пояснювальна записка*	*Арк.*	*Аркуш*	*Аркушів*
Перевірив		*Грига В. М.*					*5*	*68*
Н. Контр.							
Затверд.							

ЗМІСТ

ВСТУП

Досить швидкі темпи розвитку сучасної цивілізації потребують забезпечення інноваційними розробками у всіх сферах життєдіяльності людини. Відколи в наших домівках появилася електроенергія, людство намагалося різними способами покращити своє житло та перебування в ньому, зробити свою звичну домашню роботу комфортнішою за допомогою різних електронних приладів. Розумний будинок – це нова концепція використання звичних нам пристроїв, які є в будинку.

З кожним роком зростає число замовників на «інтелектуалізацію» свого будинку. Збільшується число компаній, що займаються проектуванням і впровадженням системи, але, не дивлячись на це, даний процес в Україні йде повільно. Проблемою цього є недостатня інформованість споживача про всі можливості автоматизованого житла, а також висока вартість системи. Велику роль відіграє і рівень розвитку підприємств, які потенційно готові впроваджувати дані системи [1]. Окреслені проблеми і зумовлюють актуальність обраної теми.

Разом із технологіями «Інтернету речей» невпинно розвиваються й системи штучного інтелекту та системи обробки природної мови. Ці інновації дозволяють вивести комунікацію між людиною і роботом на зовсім інший рівень. В основі систем природної обробки мови лежать нейронні мережі, які невпинно розвиваються та з кожним днем з’являються все нові і кращі рішення.

Метою бакалаврської кваліфікаційної роботи є розроблення голосової системи керування елементами «Розумного будинку» на базі апаратної платформи Node Mcu ESP8266 з використанням відповідних давачів та модулів, веб-сервера та голосового асистента. Така комбінація технологій дозволяє керувати розумним будинком за допомогою голосових команд. Користувач зможе спілкуватися з електронікою свого будинку ніби з людиною. Розроблене технічне рішення може широко застосовуватися для підвищення комфорту користувачів та суттєво спростити складні в конфігурації та використанні наявні на ринку системи.

1. ОГЛЯД ТА АНАЛІЗ ОСНОВНИХ ТЕХНОЛОГІЙ ”ІНТЕРНЕТ РЕЧЕЙ”

1.1.Огляд технологій “Інтернет речей”

Розумний будинок є невід’ємною частиною “Інтернет речей”, це термін, який використовується для позначення сучасних будинків чи різного роду приміщень, в яких інженерні, інформаційні й охоронні системи об’єднані в єдину узгоджену інтелектуальну систему. Система управління будинком передбачає керування централізовано – користувачем з пульта-дисплею, автоматично за допомогою певних алгоритмів, а також дистанційні варіанти керування за допомогою мобільного телефону чи комп’ютера.

“Інтернет речей” на сьогодні є однією з найперспективніших технологій останніх років. Наприклад, сміттєві баки зможуть сповіщати міські служби про необхідність їх очищення від сміття. Розумний кондиціонер, зможе відстежувати місце перебування свого власника і включатися, коли той прямує додому, а домашній холодильник - стежити за кількістю та асортиментом продуктів. Переваги “Інтернету речей”, які вже доступні і які ще в процесі розробки можна краще продемонструвати на прикладах, оскільки сфер використання цієї технології чимало.

Рис. 1.1. Розумна зубна щітка.

На рисунку 1.1. зображений приклад використання елементів інтернету речей. Навіть зубна щітка може під’єднуватися до Wi-Fi та збирати різну аналітику про звички користувача та надавати інформацію про правильність чищення зубів.

Рис. 1.2. Розумна скарбничка.

На рисунку 1.2. зображена розумна скарбничка. Вона дозволяє користувачу слідкувати за своїми збереженнями в режимі он-лайн. Також для неї у користувача буде зручна аплікація на смартфоні, щоб можна було зручно керувати різними конфігураціями, наприклад, вона сповіщатиме про внесену суму, так само можна налаштувати різні сигнали, які спрацьовуватимуть при внесенні коштів.

Рис. 1.3. Розумний лоток для яєць.

На рисунку 1.3. зображений ще один яскравий приклад того, що скоро багато людей не зможуть уявити свого життя без інтернету речей. Даний пристрій дозволяє слідкувати за кількістю та станом яєць вдома за допомогою зручної аплікації на смартфоні користувача.

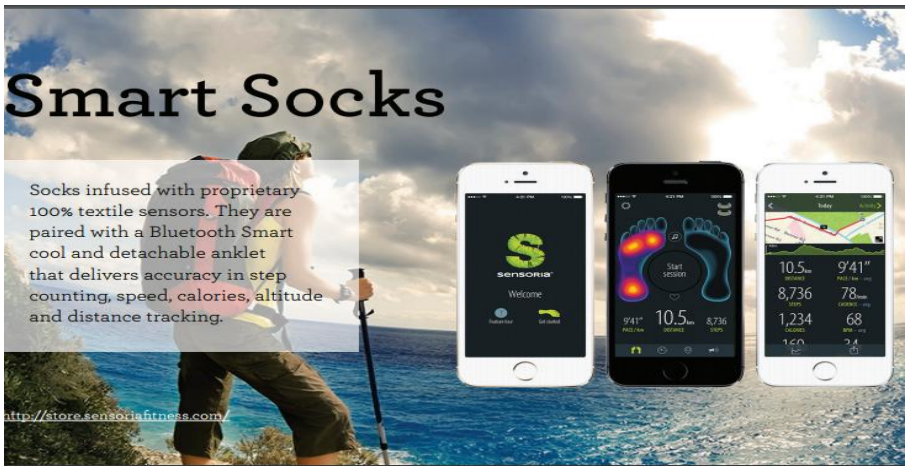


Рис. 1.4. Розумні шкарпетки.

На рисунку 1.4 зображені розумні шкарпетки. Ця передова технологія дозволяє слідкувати за правильністю хотьби, кількістю зроблених кроків, швидкістю, калоріями та навіть за пройденою дистанцією.

Індустрія інтернету речей постійно збільшувалася в останні роки, і її ріст лише прискорюватиметься. Дослідницька і консалтингова компанія Gartner у своїй доповіді зробила прогноз, що у 2016 році у світі налічується 6,4 мільярди "речей" – 30-відсоткове зростання порівняно із 2015 роком (4,9 мільярдів пристроїв). У грошовому вимірі, згідно доповіді, розмір індустрії зросте із 1183 мільярдів доларів у 2015 році до 1414 мільярдів доларів у 2016-му. У 2020-му році підключених пристроїв буде вже 20,7 мільярдів, а загальні витрати на них становитимуть 3 трильйони доларів (з них близько половини –споживачі, інша половина – бізнес).

Інтернет речей проникає у всі сфери життя і забезпечує роботу багатьох систем: від суто споживацьких (таких як різноманітні побутові сенсори, носима електроніка, розумні будинки тощо) до індустріальних (керування і моніторинг виробничих процесів, розумні енергосистеми, розумні міста, автономні автомобілі тощо). За допомогою "речей" додана вартість збільшуватиметься за рахунок:

- покращення клієнтського досвіду (customer experience);
- зменшення часу, що проходить від задуму продукту до його появи у продажу (time-to-market);

- поліпшення ланцюжків постачання та логістики;
- збільшення продуктивності працівників;
- більш ефективного використання активів (зменшення витрат).

Розвиток інтернету речей стимулює прогрес у багатьох галузях інженерних наук. Очевидно, що зростання попиту на апаратне забезпечення для

IoT сприяє інноваціям в галузі електроніки, яка забезпечує індустрію інтернету речей електронними платами, сенсорами, акумуляторами тощо. Специфіка портативних пристроїв накладає певні обмеження на програмне забезпечення, яке керує ними. Обмеженість обчислювальних можливостей компактних плат, а також вимоги щодо використання енергії (особливо для "речей", які живляться від акумулятора), вимагає економного ставлення до ресурсів.

Серверні платформи, які обробляють покази сенсорів, також повинні обов'язково враховувати те, що дані неминуче містять похибки, а пристрої можуть виходити з ладу чи функціонувати зовсім неправильно.

Питання безпеки також набуває все більшого значення в світі, де велика кількість пристроїв збирає, обробляє та передає різноманітні дані, які мають певну індустріальну чи бізнесову цінність, або містять персональну інформацію про користувачів. Також необхідно реагувати на виклики, пов'язані з постійно зростаючою кількістю інформаційних загроз.

Розробка програмних рішень, що керують великою кількістю пристроїв, вимагає застосування новітніх підходів у галузі хмарних обчислень, обробки і зберігання великих даних (Big Data). Такі серверні рішення повинні горизонтально масштабуватися, мати велику пропускну спроможність, відмовостійкість, короткий час відповіді на запити.

1.2. Голосові асистенти

На сьогоднішній день дуже багато користувачів різних країн світу спробували послуги голосових технологій, таких компаній виробників як, Apple Siri, Samsung S Voice та Google Assistant, і багато користувачів використовують такі послуги щомісяця. Зі вступом на ринок Amazon потенціал для подальшого

поглинання став ще вищим, оскільки на ринку з'являються більш досконаліші та багатофункціональні продукти та додатки [3].

1.2.1. Голосові технології

Голосові технології в останні роки стали доволі часто появлятися на ринку технологій - від автоматичних систем телефонного розпізнавання, які не розуміли акценти, до простих голосових текстових диктофонів, що виробляли неточні копії, але недоліки цих систем запобігали їх широке поширення.

Проте, досягнення в області штучного інтелекту означають, що можливості глобального активації голосу, стають доволі все реальнішими, згідно з дослідженням компанії Speak Easy, проведеного Дж. Вальтером Томпсоном, Кантаром та Міндшаном.

На сьогодні найбільш популярні способи використання голосових асистентів серед населення відносно очікувані - цікаві запитання (32%), пошук в мережі Інтернет (30%) та відтворення музики (27%). Проте, швидше за все функціональність збільшуватиметься швидше, коли приєднані пристрої почнуть з'являтися частіше.

Станом на 2017 рік можливості та використання віртуальних помічників швидко розширюються, коли нові продукти виходять на ринок. Найбільш широко використовуваними у США згідно з інтернет-опитуванням у травні 2018 року є Apple Siri (34%), Google Assistant (19%), Amazon Alexa (6%) і Microsoft Cortana (4%) та Facebook Messenger (3,5%). Компанії Apple і Google мають великі встановлені бази даних користувачів на смартфонах, а компанія Microsoft має велику встановлену базу персональних комп'ютерів на базі операційної системи Windows (де Cortana працює на додаток до телефонів і розумні динаміки). Тоді як, Alexa був першим, хто отримав можливість розмістити онлайн-замовлення електронної комерції від Amazon.

1.2.2. Історія розвитку

Першим відомим інструментом, який виконував розпізнавання цифрового мовлення, був IBM Shoebox, який був представлений широкій публіці під час світової виставкової кампанії у Сіетлі 1962 року після її первинного запуску в

1961 році. Цей ранній комп'ютер, розроблений майже за 20 років до введення першого IBM Personal Computer в 1981 році, вдалося розпізнати 16 розмовних слів та цифри від 0 до 9.

Наступний етап у розвитку технології розпізнавання голосу був досягнутий в 1970-х роках в Університеті Карнегі-Меллона в Пітсбурзі, штат Пенсільванія, з суттєвою підтримкою Департаменту США Оборони та її агентства DARPA. Їхнім інструментом "Гарпія" освоєно приблизно 1000 слів словником трьох-річного віку.

Близько 12 років тому одна і та ж група вчених розробила систему, яка могла не тільки аналізувати окремі слова, але й цілі послідовності слів, включені за допомогою моделі прихованих маркерів. Таким чином, перші віртуальні помічники, які застосовували програмне забезпечення для розпізнавання мови, були автоматизованими програмами супроводу та медичного цифрового диктування.

У 90-х роках методологія розпізнавання мовних технологій стала обов’язковою особливістю персонального комп'ютера, у зв’язку із агітацією за клієнтів таких корпорацій, як Microsoft, IBM, Philips та Lernout & Hauspie. Значно пізніше запуск ринку першого смартфона IBM Simon в 1994 році заклав основу для інтелектуальних віртуальних помічників.

Першим сучасним цифровим віртуальним помічником, встановленим на смартфоні, був Siri, який був представлений як особливість iPhone 4S 4 жовтня 2011 року. Компанія Apple Inc. розробила Siri після придбання в 2010 році Siri Inc., відділення SRI International, який є дослідницьким інститутом, який фінансується DARPA та Міністерством оборони США.

1.2.3. Методи інтеракції

Віртуальні помічники виконують свої функції за допомогою:

- тексту (чат онлайн), особливо в додатку для обміну миттєвими повідомленнями чи іншим додатком;
- голосу, наприклад, в Amazon Alexa на пристрої Amazon Echo або Siri на iPhone;

- обміну зображеннями, як у випадку Samsung Bixby на Samsung Galaxy S8;

Деякі віртуальні помічники доступні за допомогою кількох методів, таких як Google Assistant за допомогою чату в додатку Google Allo та голосом у смарт-гучномовцях Google Home.

Віртуальні помічники використовують обробку природної мови (NLP), щоб відповідати текстовій або голосовій вхідній інформації для виконуваних команд. Багато хто постійно навчається за допомогою методів штучного інтелекту, включаючи машинне навчання.

Щоб активувати віртуальний помічник за допомогою голосу, може бути використано слово “пробудження”. Це слово або групи слів, такі як "Alexa" або "OK Google".

1.2.4. Економічна доцільність

Цифрові можливості, надані віртуальними помічниками, вважаються одним з основних останніх технологічних досягнень та найбільш перспективних споживчих трендів. Експерти стверджують, що цифровий досвід досягне рівня статусу, який буде співмірний з "реальним" досвідом, якщо не стане більш популярним і дороговартісним.

Дана тенденція підтверджена великою кількістю постійних користувачів та суттєвим зростанням світових користувачів цифрових віртуальних помічників. У середині 2018р. кількість часто використовуваних цифрових віртуальних помічників оцінювалася приблизно в 1,2 млрд. у всьому світі.

Крім того, можна бачити, що технологія віртуальних цифрових асистентів більше не обмежується застосуванням смартфонів, але представлена багатьма галузевими секторами (включаючи автомобільне, телекомунікаційне, роздрібне, медичне та освітнє обслуговування). У відповідь на значні витрати на проведені дослідження та розробки фірм-розробників у всіх секторах та на збільшення імплементації мобільних пристроїв, ринок технології розпізнавання мовлення прогнозується на період 2016-2024 рр. Він зросте на 34,9% у цілому і,

таким чином, перевершить світовий ринок розміром 7,5 млрд. дол. США до 2024 року.

Очікується, що, беручи до уваги регіональний розподіл лідерів ринку, компанії в Північній Америці (наприклад, Nuance Communications, IBM, eGain), будуть домінувати в галузі протягом наступних років через значний вплив BYOD (Bring Your Own Device) та бізнесу з мобільності підприємств.

Крім того, очікується, що зростаючий попит на платформах, що підтримують смартфони, ще більше посилить зростання промисловості в галузі інтелектуального віртуального помічника (IVA) в Північній Америці. Незважаючи на менший розмір порівняно з північноамериканським, інтелектуальна віртуальна асистентська індустрія з Азіатсько-Тихоокеанського регіону з головними гравцями, розташованими в країнах Індії та Китаї, прогнозується, зростатиме при річному зростанні 40% (вище середнього значення) протягом періоду 2016-2024 років.

1.3. Принципи роботи системи «розумний будинок»

Термін «розумний будинок» означено будь-яку систему з автоматизованим керуванням приладами, яка спрощує життя людини та підвищує рівень її комфорту. Через нечіткі рамки виникло багато реалізацій з різним рівнем інтеграції та принципом роботи [4].

В загальному "розумний будинок" - це термін, який звичайно використовується для визначення місця проживання, яке має автоматизовані прилади, освітлення, опалення, кондиціювання повітря, телевізори, комп'ютери, аудіо- та відеосистеми, системи безпеки та спеціалізовані камери, здатні спілкуватися один з одним і можуть бути дистанційно керовані за наперед заданим алгоритмом, з будь-якої кімнати в будинку, а також віддалено з будь-якого місця у світі за допомогою смартфона або через Інтернет (рис. 1.5).

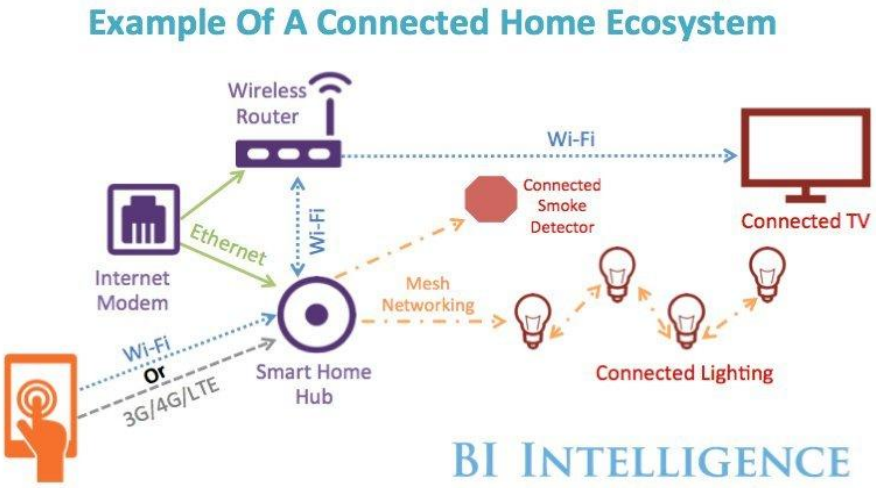


Рис. 1.5. Приклад екосистеми розумного будинку.

Встановлення смарт-продуктів дає будинку та його мешканцям різні переваги – зручність, комфорт, економію часу, грошей та енергії.

Більшість будинків не мають цих пристроїв та систем, які в них вбудовані, тому найпоширеніший і доступний підхід для домашнього власника - встановити пристрої інтелектуальної власності до власного будинку.

Визначальною особливістю розумного будинку є збереження обмежених ресурсів планети. Все більше і більше людей усвідомлюють здатність робити свої будинки дійсно розумними та економними за допомогою домашніх контролерів, інтегрованих з усіма домашніми підсистемами, щоб збільшити заощадження, керуючи освітленням, покриттям вікон, кондиціонуванням повітря та моніторингом використання. Багато домашніх контролерів мають вбудовані системи моніторингу, за допомогою яких вони обчислюють та використовують журнал всіх під'єднаних пристроїв, підвищуючи обізнаність та знання власника будинку, щоб вносити необхідні зміни. Ці системи можна навіть отримати через Інтернет з будь-якої точки світу, тому власник житла може регулювати споживання в будь-який час, де завгодно.

У кожному сучасному будинку в тій чи й іншій мірі функціонує значна кількість обладнання, яка забезпечує комфорт, затишок, зв'язок і безпеку, що допомагає відпочити і створює повноцінне робоче середовище. Зручність управління цими системами, їх інтеграція один з одним, можливість

злагоджено працювати разом, збільшуючи тим самим функціональність кожної з них окремо і дає можливість назвати такий будинок розумним.

У випадках відсутності людини “розумний будинок” буде підтримувати оптимальним чином постійний мікроклімат, зберігаючи тим самим затишок, вологість і полив кімнатних рослини та захист меблів. Система вимкне не потрібне світло або навпаки буде створювати видимість присутності господаря, включаючи і вимикаючи освітлення в тій або іншій кімнаті час від часу.

Розумний будинок виконує стеження за всіма інженерними системами в будинку і не допустить спалаху або вибуху пов'язаного з витоком газу або зіпсуванню меблів через витік води.

Власник має можливість постійно керувати розумним будинком і отримувати інформацію про стан всіх систем у будинку, перебуваючи на далекій відстані.

Системи керування ”розумним будинком” можна умовно поділити на три групи:

• вбудовані системи з центральним контролером;

• вбудовані системи без центрального контролера;

• системи з інтеграцією, що налаштовується.

Перша група представляє з себе повністю налаштовану і встановлену виробником систему, яка керується центральним обчислювальним пристроєм і не передбачає прямої взаємодії своїх модулів між собою.

Друга група є системою з напівавтономними пристроями. Алгоритми взаємодії прописуються з програми контролера безпосередньо в пам’ять кожного пристрою.

Третя група – це зовнішні контролери, які під’єднують до звичайних приладів і залежно від показання своїх сенсорів і вбудованого алгоритму регулюють їх роботу. Можуть мати центральний контролер, але часто керуються і налаштовуються з Інтернет або хмарного сервісу.

Більшість електричних систем які знаходяться в будинку вже давно працюють в автоматичному режимі і без розумного дому, наприклад керування

опаленням чи освітленням. Але основною перевагою розумного дому є те, що всі ці автоматизовані системи можна поєднати в тісно взаємодіючу систему.

Для прикладу лампочкою в коридорі можна керувати: з вимикача, комп'ютера, телефона, давачем руху, давачем відкриття дверей або будь-яким іншим доступним способом. Крім того способи керування можна перепрограмувати в будь-який необхідний момент.

В основному всі елементи розумного будинку можна розділити на три групи: елементи отримання інформації, елементи отримання команд, елементи виконання команд, а також центральний виконавчий модуль, який ці всі елементи поєднує (рис. 1.6).

Рис. 1.6. Схема розумного будинку.

До елементів отримання інформації відносяться всі прилади розумного дому, які можуть діставати інформацію з навколишнього світу – давачі температури, руху, газу, диму, води, вологості, світла, відкриття дверей, та багато інших.

До елементів отримання команд відносяться пристрої через які користувач розумного дому передає свої команди розумного дому. Це різноманітні кнопки, вимикачі, сенсорні панелі, пристрої отримання голосових команд, персональні комп'ютери, смартфони та інше.

Виконавчі модулі – це прилади, які виконують команди розумного дому – релейні блоки, димери, пристрої керування жалюзями, аудіо- та відеомодулі, GSM модулі, модулі передачі інфрачервоних команд.

Завдання центрального модуля – отримувати команди з давачів та приладів керування і передавати їх виконавчим модулям. Також центральний модуль може задавати логічні та часові функції.

1.3.1. Елементи та способи керування системою

Управління системою інтелектуального (розумного) будинку може здійснюватися трьома способами: бездротовим локальним, віддаленим бездротовим, проводовим локальним, віддаленим провідним (рис. 1.7). Прилади керування, що йдуть в комплекті з системою – це графічні панелі керування з сенсорним або кнопковим введенням, пульти з приймачами, налаштованими на певну частоту. Прилади, що не йдуть в комплекті, як мобільні компактні комп'ютери (смартфони, планшети), настроюються за допомогою спеціального програмного забезпечення для управління віддаленим способом, через всесвітні інформаційні мережі [5,6].

Рис. 1.7. Системою розумний будинок можна керувати пультом, панеллю керування і мобільними пристроями.

Бездротовий локальний спосіб управління з обмеженим радіусом проводиться із застосуванням приладів керування через локальний (місцевий) радіосигнал, Wi-Fi або Bluetooth бездротові радіомережі. В принципі, таким

способом можна керувати системою з будь-якої точки будинку і навіть перебуваючи на присадибній ділянці неподалік. Проте, у великих будинках, можливо, знадобляться додаткові радіоточки, підсилювачі бездротового сигналу. Управління здійснюється пультами, сенсорними панелями, мобільними приладами (смартфонами, планшетами, ноутбуками), які мають вбудований або підключений ззовні передавач тієї чи іншої мережі Wi-Fi, Bluetooth або фірмового радіочастотного сигналу.

Бездротовий віддалений спосіб керування доступний, якщо здійснений монтаж системи розумний дім до глобальних мереж або модулів розширення зв'язку, забезпечує доступ до них. До таких мереж можна віднести GSM/GPRS (управління за рахунок мобільного зв'язку), мобільний інтернет, спеціальний виділений радіосигнал. Наявність виходу в GSM/GPRS мережа дає можливість системі відсилати СМС, ММС і звукові повідомлення на номер телефону власника будинку. Також теоретично можливо управління через голосове меню. Засоби для управління системами розумного будинку через глобальні мережі – це, як правило, смартфони, а також планшети та ноутбуки з вбудованими модулями-передавачами для мобільного інтернету.

Дротовий локальний спосіб управління в залежності від застосовуваних протоколів передачі даних системи розумний будинок, може мати середовища передачі даних по витій парі (кабелю комп'ютерної мережі), електричній проводці (протокол системи X10) або яким-небудь іншим кабелем. У кожному регіональному та центральному контролері система управління розумний будинок передбачає вихід для дротового зв'язку з керуючими приладами, а також обміну інформацією з іншими інтелектуальними приладами, які знаходяться у нього в «підпорядкуванні». Якщо кілька приладів, то інтерфейс розширюють за допомогою комутатора (див. вище «модулі розширення зв'язку»), створюючи кілька відгалужень. Для дротового локального способу управління застосовуються панелі управління і вимикачі (кнопкові і сенсорні), що йдуть в комплекті і окремо, а також можна управляти через підключений до кабелю комп'ютер або ноутбук.

Провідний дистанційний спосіб управління системами розумного будинку – це мережа, за розмірами більше локальної, тобто, управління будинком відбувається ззовні по прокладеному кабелю, підключеному до внутрішньої мережі управління. Як правило, такий спосіб управління застосовується в управлінні автоматизованими процесами корпусів будівель.

1.4. Огляд існуючих рішень

На сьогоднішній день існує декілька платформ, що застосовуються в галузі розумних ситем. Найбільш відомі з них: Amazon Web Services IoT Platform, ThingWorx, Google Cloud IoT Platform, Oracle IoT, IBM IoT Platform [7].

Здебільшого існуючі рішення пропонують загальний набір функціональності, який включає зберігання даних, а також досить широкі можливості з їх відображення та аналізу.

Усі перераховані платформи є комерційними із закритим вихідним кодом, що відповідно і є їх значним недоліком. Така закритість ускладнює їх розширення, адаптацію й оптимізацію під конкретний сценарій використання. Корпорації, які використовують платформи, що працюють за такою моделлю, часто прив’язані до постачальника (vendor lock-in) і не можуть переорієнтуватися на застосування іншої платформи без значних витрат часу та коштів.

Платформи з відкритим вихідним кодом майже відсутні. Проте цілком можливо розробити функціональну платформу, що використовуватиме opensource компоненти та бібліотеки, оскільки наразі доступно багато рішень із відкритою чи вільною ліцензією, які можуть стати частинами системи і які вже є стабільними та мають широкі функціональні можливості. Отже проведемо певне дослідження, порівняння та використання відповідних компонентів.

Існуючі платформи іноді тільки в деяких випадках мають готові рішення по очищенню помилкових даних. Проте загалом подібні платформи можуть надавати базові інструменти зі статистики чи машинного навчання, які застосовні майже для всіх сценаріїв використання. Слід зауважити, що

очищення даних є великою і складною частиною науки про дані (data science). Особливо перспективною галуззю є виявлення аномалій (anomaly detection). Online anomaly detection наразі є однією з найменш досліджених розділів машинного навчання.

Розглянемо для прикладу одну з найбільш функціональних і популярних із існуючих платформ – AWS IoT Platform. Її архітектуру зображено в додатку А. На діаграмі показано основні компоненти системи. Розглянемо їх детальніше:

- SDK для пристрою, за допомогою якого надсилаються повідомлення;
- вузол автентифікації та авторизації, який використовує SigV4 або сертифікати X.509;
- шлюз пристроїв, який забезпечує комунікацію за принципом pub/sub по протоколах MQTT, WebSocket, HTTP 1.1;
- регістр для зберігання інформації про пристрій;
- «Тіні» пристроїв – механізм збереження останнього отриманого стану пристрою для його подальшого запиту через REST API;
- механізм задання правил, який дозволяє описати інтеграцію з іншими сервісами AWS;

Як бачимо, AWS IoT пропонує вельми функціонально завершеній інструменти безпеки, але є повністю зав'язаною на сервіси AWS, використовує мало непоширений протокол MQTT та не надає API для визначення власних ланцюжків обробки даних.

Наступною розглянутою платформою для "Інтернету речей" є IBM IoT Platform [8]. Розробники розділили її функціональність на чотири основних блоки:

1. Підключення. Платформа пропонує різноманітні SDK для пристроїв і шлюзів, а також API для взаємодії із клієнтським програмним забезпеченням. Ці можливості доступні для різних мов програмування, включно з оптимізованими низькорівневими рішеннями на Embedded C. Цей блок

функціональності призначений для безпечного підключення пристроїв до платформи для передачі даних. Основними протоколами передачі є MQTT і HTTPS.

2. Керування ризиками. Ця група функцій відповідає за керування великими групами пристроїв, дозволяючи користувачам переглядати інформацію про їхнє функціонування на панелях управління. Також наявні повідомлення адміністраторів про нештатні ситуації, які дозволяють оперативно ізолювати різноманітні інциденти.

3. Керування інформацією. Блок функціональності, що дозволяє управляти збереженням і архівуванням операційної інформації та метаданих. Наявні можливості по парсингу та трансформації даних, генерування звітів. Також можливо вводити дані з різноманітних джерел і платформ.

4. Аналітика. Інтеграція з IBM Watson дозволяє користувачам виконувати складний аналіз даних, застосовуючи наявні підсистеми Watson. Наприклад, блок аналізу природньої мови, різноманітні алгоритми машинного навчання, інструменти аналітики відео та зображень, аналіз текстуальних даних.

Отже, основною перевагою IBM IoT Platform є широкі можливості по аналізу даних, які реалізуються системою IBM Watson. Інша функціональність є досить типовою для подібних платформ. Основним недоліком є закритість платформи, а відповідно, неможливість її розширення.

2. ВИБІР ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ ГОЛОСОВОГО КЕРУВАННЯ ”РОЗУМНИМ БУДИНКОМ”

2.1.Вибір мови програмування для реалізації веб-сервера

Для написання back end частини даної роботи було вибрано мову програмування Java [9].

Java - об'єктно-орієнтована мова програмування, яка була розроблена у 1995 року компанією Sun Microsystems. Він має на меті дозволити розробникам програм «писати один раз, працювати в будь-якому місці» (WORA), що означає, що скомпільований код Java може працювати на всіх платформах, що підтримують Java, без необхідності його перекомпіляції. Програми Java, як правило, збираються до байт-коду, який може працювати на будь-якій віртуальній машині Java (JVM) незалежно від архітектури комп'ютера. Починаючи з 2016 року, Java є однією з найпопулярніших мов програмування, зокрема веб-додатків клієнт-сервер, з 9 мільйонами розробників. Java була спочатку розроблена Джеймсом Гослінгом у Sun Microsystems (який з тих пір був придбаний корпорацією Oracle) і випущений в 1995 році як основний компонент платформи Java Sun Microsystems. Мова запозичує значну частину синтаксису з мов високого рівня C і C ++, але в ній менше об'єктів низького рівня, ніж будь-який з них.

Оригінальна та еталонна реалізація Java-компіляторів, віртуальних машин і бібліотек класу були випущені Sun за власними ліцензіями. Станом на травень 2007 року, відповідно до специфікацій процесу спільноти Java, корпорація Sun повторно ліцензувала більшість своїх технологій Java відповідно до загальної публічної ліцензії GNU. Інші також розробили альтернативні реалізації цих технологій Sun, таких як компілятор GNU для Java (компілятор байт-коду), GNU Classpath (стандартні бібліотеки) та IcedTea-Web (плагін для браузера для аплетів).

Остання широко застосована версія - це Java 9, випущена у 2019 р., І є однією з двох версій, які в даний час підтримуються Oracle безкоштовно. Версії раніше, ніж Java 8, підтримуються компаніями на комерційній основі.

2.1.2. Платформа Java

На сьогоднішній момент мова Java є одним з найбільш поширених і популярних мов програмування. Мова Java була задумана в 1991 році співробітниками компанії Sun Microsystems Джеймсом Гослінгом, Патріком Нотона, Крісом Уорт, Едом Френком і Майком Шериданом.

Завдання створення крос-платформних програм виникла чи не разом з появою перших комп'ютерів, але взятися за її рішення так і не вдавалося через необхідність вирішувати інші, більш важливі і невідкладні завдання. Проте з появою Інтернету проблема переносимості програм перейшла в розряд абсолютно невідкладних. Адже Інтернет складається з безлічі різнотипних комп'ютерів з різною архітектурою процесорів і різними операційними системами. У 1993 році розробникам Java стало ясно, що завдання переносимості потрібно вирішувати не тільки при програмуванні мікропроцесорних пристроїв, але і при створенні коду для Інтернет - додатків. Сфера застосування мови розширилася. І якщо програмування мікроконтролерів стало спонукальною причиною для створення Java, то Інтернет сприяв широкому поширенню цієї мови.

Мова Java дуже схожа на мови C і C ++. Від C мова Java успадкувала синтаксис, а від C ++ - об'єктну модель. Подібність Java з мовами C і C ++ грає важливу роль.

Ключовою особливістю мови є те, що код спочатку транслюється в спеціальний байт-код, сумісний із різними платформами. А потім цей байт-код виконується віртуальною машиною JVM. Віртуальна машина Java, по суті, являє собою інтерпретатор байт-коду. В цьому плані Java відрізняється від стандартних різних мов як PHP або Perl, код яких відразу ж виконується інтерпретатором. У той же час Java не є і чисто компільовані мовою, як C або C++.

Виконання програми під управлінням віртуальної машини дозволяє вирішити багато труднощів, що виникають в роботі. Трансляція вихідного коду Java в байт-код спрощує перенесення програм з одного середовища в інше,

оскільки для забезпечення працездатності коду досить реалізувати на кожній платформі віртуальну машину. Якщо на комп'ютері присутній пакет виконує системи, то на ньому може працювати будь-яка програма, написана на Java. Незважаючи на те що віртуальні машини на різних платформах можуть бути реалізовані по-різному, вони повинні однаково інтерпретувати байт-код. Виконання програми під управлінням віртуальної машини допомагає також забезпечити безпеку. Віртуальна машина може заборонити програмі виконувати операції, побічні ефекти яких здатні вплинути на ресурси за межами виконуючої системи [6].

Подібна архітектура забезпечує кросплатформність і апаратну переносимість програм на Java, завдяки чому подібні програми без перекомпіляції можуть виконуватися на різних платформах - Windows, Linux, Mac OS і т.д. Для кожної з платформ може бути своя реалізація віртуальної машини JVM, але кожна з них може виконувати один і той же код.

Java є об'єктно-орієнтованою мовою. Він підтримує поліморфізм, успадкування, статичну типізацію. Об'єктно-орієнтований підхід дозволяє вирішити завдання з побудови великих, але в той же час гнучких, масштабованих і розширюваних додатків.

2.1.3. Технологія Spring Framework

Мова програмування Java є однією з найбільш розвинених та містить чимало фреймворків, які суттєво спрощують розробку програм та роблять її ефективнішою. Для реалізіції поставленого технічного завдання в бакалаврській роботі був вибраний Spring Framework, який є одним із найбільш розвинених фреймворків Java [11].

Spring Framework (або коротко Spring) - універсальний фреймворк із відкритим вихідним кодом для Java-платформи.

Перша версія була написана Родом Джонсоном, який вперше опублікував її разом з виданням своєї книги.

Незважаючи на те, що Spring не забезпечив будь-яку конкретну модель програмування, він став широко розповсюдженим у Java, головним чином як

альтернатива і заміна моделі Enterprise JavaBeans. Spring надає велику свободу Java-розробникам в проектування; крім того, він надає добре документовані та прості у використанні засоби вирішення проблем, що виникають при створенні прикладних програм корпоративного масштабу.

Крім того, особливості ядра Spring застосовуються в будь-якому Java-додатку, та є велике число його доповнень і вдосконалень для побудови веб-застосунків на Java Enterprise платформі. По цих причинах Spring здобув більшу популярність і визнання розробників як стратегічно важливий фреймворк.

Центральною частиною Spring є контейнер Inversion of Control, який надає засоби конфігурування та керування об'єктами Java за допомогою рефлексії. Контейнер відповідає за управління життєвим циклом об'єкта: створення об'єктів, виклик методів ініціалізації та конфігурації об'єктів шляхом їх зв'язування між собою.

Об'єкти, створені контейнером, також називаються керованими об'єктами (beans). Зазвичай конфігурація контейнера здійснюється шляхом завантаження XML-файлів, що містять цільові визначення і надають інформацію, необхідну для створення файлів.

Об'єкти можуть бути отримані одним з двох способів:

- Пошук залежностей - шаблон проектування, в якому виклик об'єкта запитує в об'єкті-контейнера екземпляр об'єкту з певним ім'ям або певним типом.
- Впровадження залежності - шаблон проектування, в якому контейнер передає екземпляри об'єктів за їх іменем іншими об'єктами за допомогою конструктора, властивостей або фабричного методу.

Spring має власну MVC-платформу веб-додатків, яка не була спочатку запланована. Розробники Spring вирішили написати її як реакцію на те, що вони сприйняли як невдалість конструкції (тоді) популярного Apache Struts, а також інших доступних веб-фреймворків. Зокрема, на їхню думку, не вистачало

поділу між шарами подання та обробки запитів, а також між шаром обробки запитів і моделлю.

Клас *DispatcherServlet* є основним контролером фреймворка і відповідає за делегування управління в різних інтерфейсах, на всіх етапах виконання HTTP-запиту.

Як і *Struts, Spring MVC* є фреймворком, орієнтованим на запити. У ньому визначені стратегічні інтерфейси для всіх функцій сучасної орієнтованої системи. Мета кожного інтерфейсу - бути простим і зрозумілим, щоб користувачам було легко його заново імплементувати, якщо вони того побажають. MVC прокладає шлях до більш чистого front-end-коду. Всі інтерфейси тісно пов'язані з Servlet API.

Цей зв'язок розглядається деякими як нездатність розробників Spring запропонувати для веб-додатків абстракцію більш високого рівня. Однак цей зв'язок залишає особливості *Servlet API* доступними для розробників, полегшуючи роботу з ним. Найбільш важливі інтерфейси, притаманні Spring MVC, перераховані нижче:

- *HandlerMapping:* вибір класу і його методу, які повинні обробити даний вхідний запит на основі будь-якого внутрішнього або зовнішнього для цього запиту атрибута або стану.
- *HandlerAdapter:* виклик і виконання обраного методу обробки вхідного запиту.
- *Controller:* включений між Моделлю (Model) і Оглядом (View). Управляє процесом перетворення вхідних запитів в адекватні відповіді. Діє як потік команд, направляючі всю інформацію, що надходить. Перемикає потік інформації з моделі в об’єкт і назад.
- View: відповідає за повернення відповіді клієнту у вигляді текстів і зображень. Деякі запити можуть проходити прямо під View, не заходячи в Model; інші проходять через всі три шари.
- ViewResolver: вибір, яке саме View має бути показано клієнту.

- HandlerInterceptor: перехоплення вхідних запитів. Порівняємо, але не еквівалентний сервлет-фільтрам (використання не є обов'язковим і не контролюється DispatcherServlet-му).
- LocaleResolver: отримання і можливо, збереження локальних налаштувань (мова, країна, часовий пояс) користувача.
- MultipartResolver: забезпечує Upload - завантаження на сервер локальних файлів клієнта.

Spring MVC надає розробнику наступні можливості:

- Прозорий поділ між шарами в MVC і запитах.
- Стратегія інтерфейсів - кожен інтерфейс робить тільки свою частину роботи.
- Інтерфейс завжди може бути замінений альтернативною реалізацією.
- Інтерфейси тісно пов'язані з Servlet API.
- Високий рівень абстракції для веб-додатків.

У веб-додатках можна використовувати різні частини Spring, а не тільки Spring MVC.

2.2.Вибір середовища програмування апаратури Arduino IDE

Для написання програми для вибраного мікроконтролерного макету Arduino Uno вибрано середовище розробки – Arduino IDE. Дане програмне середовище має велику кількість переваг: використовує мало пам’яті, невибаглива до ресурсів комп’ютера, проста і зручна у користуванні, легка у підключенні до плати та багато інші.

На рис. 2.1 зображено головне вікно вибраного програмного середовища Arduino IDE [10].

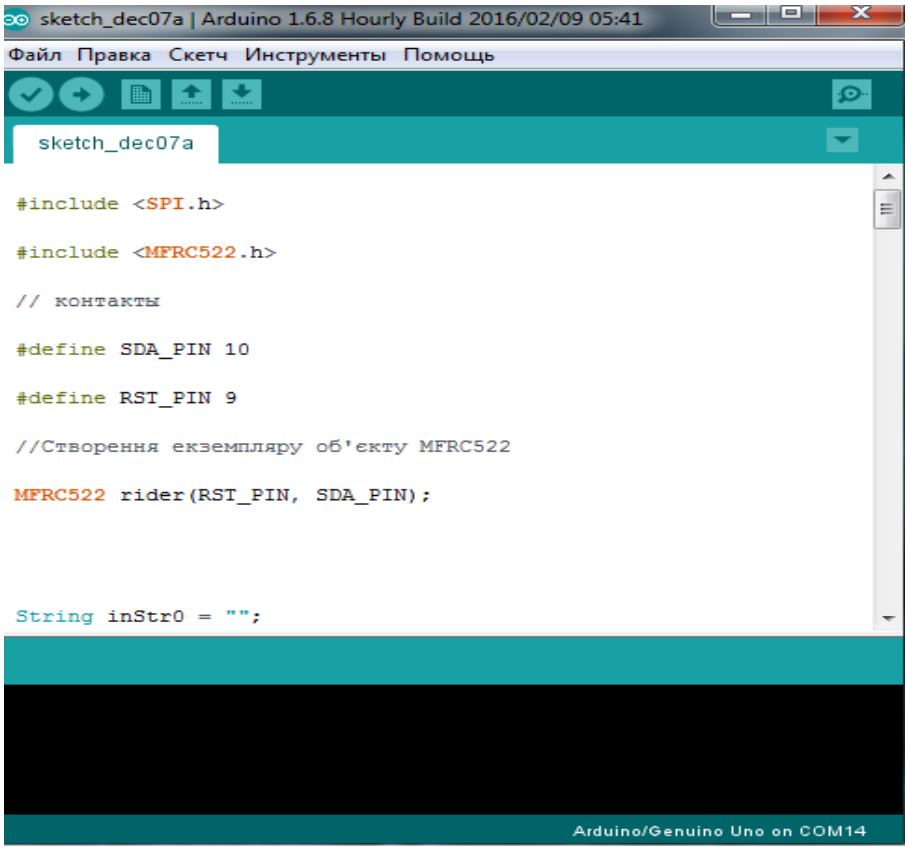


Рис. 2.1 Інтерфейс програми Arduino IDE.

Як видно із рисунку, середовище складається з вбудованого текстового редактора коду, вікна виводу тексту (консоль), панелі інструментів з кнопками команд. Що використовуються найчастіше і декількох меню. Програма, що була написана в Arduino IDE називається скетч. Програма (скетч) пишеться у текстовому редакторі на мові програмування C/C++. Скетч складається з 2 блоків, це блок *setup* та блок вічного циклу - *loop*. У першому блоці приписуються налаштування пінів, ініціалізація певних допоміжних блоків. Другий блок є основним тілом програми. Тут записується основний код програми, тобто певні зчитування з давачів, обробка даних, виведення інформації. Код, який знаходиться у функції *loop* виконується циклічно. Вікно виводу тексту, або ж консоль, відображає повідомлення про хід завантаження повідомлення помилок, що виникли у ході компіляції або завантаження скетчу.

2.3. Вибір модуля передачі інформації

В якості Wi-Fi модуля було обрано Wi-Fi модуль NodeMcu ESP8266 CP2102 [13]. Його роль в даній роботі полягає в отриманні команд від веб-сервера про ввімкнення/вимкнення реле.

NodeMCU - open source платформа для IoT. Вона включає в себе прошивку, яка працює на ESP8266 (рис. 2.2), Wi-Fi SoC від Espressif Systems з інтерфейсом Wi-Fi, а також обладнання, яке базується на модулі ESP-12. Прошивка використовує мову сценаріїв Lua. Вона заснована на проекті eLua і побудована на Espressif Non-OS SDK для ESP8266. В даному модулі використовується багато проектів з відкритим кодом, таких як lua-cjson та spiffs.

Рис. 2.2. Зовнішній вигляд модуля NODEMCU ESP8266.

NodeMCU був створений за короткий час після виходу ESP8266. 30 грудня 2013 року Espressif Systems розпочала виробництво ESP8266. ESP8266 являє собою Wi-Fi SoC, інтегрований з ядром Tensilica Xtensa LX106, широко використовується в програмах IoT. NodeMCU розпочався 13 жовтня 2014 р., Коли Хонг здійснив перший файл nodemcu-прошивки GitHub. Через два місяці проект розширився, щоб включити відкриту апаратну платформу, коли розробник Huang R зробив файл Gerber плати ESP8266 під назвою devkit v0.9. Пізніше того ж місяця, Tuan PM перенесла клієнтську бібліотеку MQTT від Contiki до платформи ESP8266 SoC і виконувала проект NodeMCU, а потім

NodeMCU змогла підтримувати протокол IoT MQTT, використовуючи Lua для доступу до брокера MQTT. В NodeMCU легко керувати РК-дисплеєм, екраном, OLED, навіть VGA-дисплеями.

Влітку 2015 року творці відмовилися від програми прошивки та взяли на себе групу незалежних, але відданих авторів. NodeMCU включає в себе понад 40 різних модулів. Через обмеження ресурсів користувачам необхідно вибрати модулі, що мають відношення до свого проекту, і побудувати прошивку, адаптовану до їхніх потреб.

NodeMcu має виводи для всіх доступних варіантів ESP8266, до них відносяться, 11 портів введення-виведення загального призначення, деякі з яких мають додаткові функції (рис. 2.3).

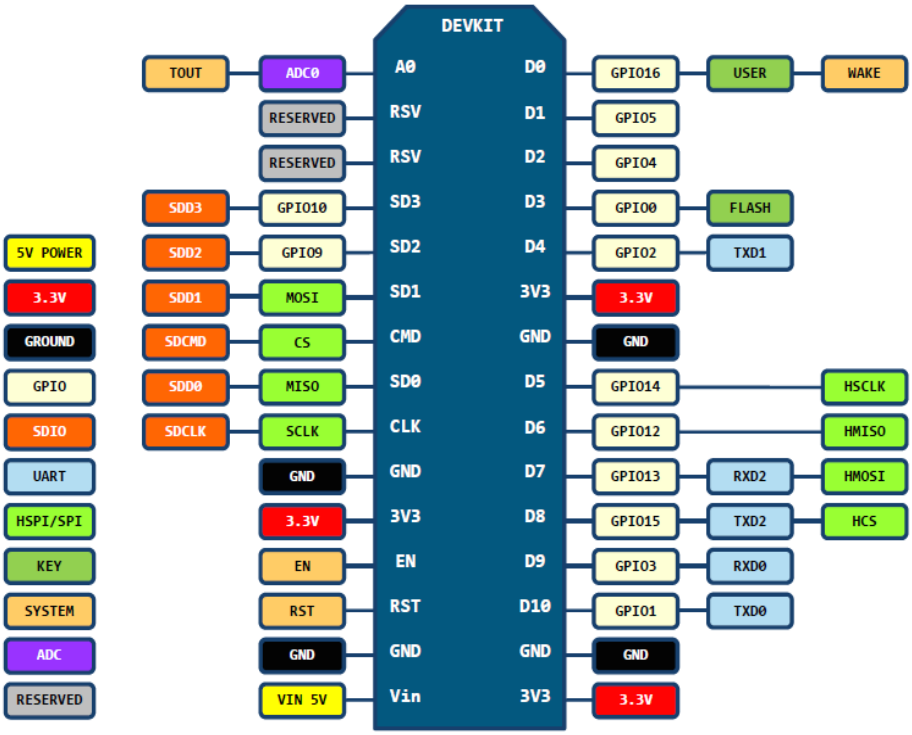


Рис. 2.3. Призначення виводів NodeMcu ESP8266.

В таблиці 2.1 наведені основні технічні характеристики компонентів модуля NODE MCU ESP8266.

Таблиця 2.1

Основні технічні хараетристики NodeMcu ESP8266

1.	Мікроконтролер	ESP8266
2.	Wi-fi Стандарт	802.11 b / g / n
3.	Підтримка режимів	STA / AP / STA + AP
4.	Швидкість передачі	110-460800 б / сек
5.	Підтримка інтерфейсів	UART / GPIO
6.	Робоча напруга	4.5 – 9 В
7.	Напруга живлення (рекомендована)	5 В
8.	Напруга живлення (максимальна)	10В
9.	Живлення:	USB
10.	Цифрові входи / виходи	D0 ~ D8, SD1 ~ SD3
11.	Аналогові входи	1
12.	Струм одного контакту	15 мА
13.	Споживання струму в середньому	50 мА
14.	Споживання при обміну даними	70 мА
15.	Споживання при обміну даними	200 мА
16.	Максимальне споживання	<200 мкА
17.	Діапазон робочих температур:	-40 ~ +125 ° C
18.	Маса	18 г

2.4. Вибір модуля реле

Модуль одно-канального реле ідеально підходить для робототехнічних проектів з сильним навантаженням. Реле може управляється за допомогою

мікроконтролерів Arduino, AVR, PIC, ARM, STM і MSP430 за допомогою цифрового порту вводу-виводу.

Характеристики реле, паданого на рис. 2.4, наступні:

- Напруга живлення: 5В DC.
- Керуюча напруга TTL 5 Вольт DC.
- 2 реле на 10А / 220В кожне.
- Швидкодія 10 мс.
- Робочий струм логіки 20мА.
- Індикація стану реле.
- Розмір: 53х28х19 мм.

Рис. 2.4. Модуль одноканального реле.

На рис. 2.5 зображена електрична схема одно-канального реле, яка містить один роз'єм, 2 транзистори, резистор, ключ та діоди.

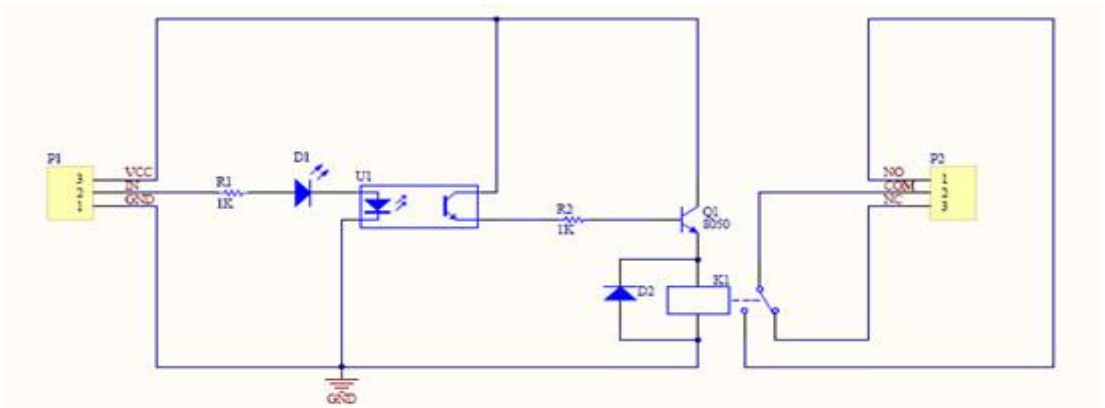


Рис. 2.5. Електрична схема одноканального реле.

Оскільки необхідна напруга живлення для реле рівна 5В, а NodeMCU підтримує тільки 3.3В, постала необхідність в стабілізаторі DC/DC, на вхід якого потрібно подати 3.3В, а на виході отримуємо 5В. В якості такого стабілізатора був вибраний DC-DC step-up MT3608 (рис. 2.6). Його переваги – неймовірна дешевизна та простота у використанні.

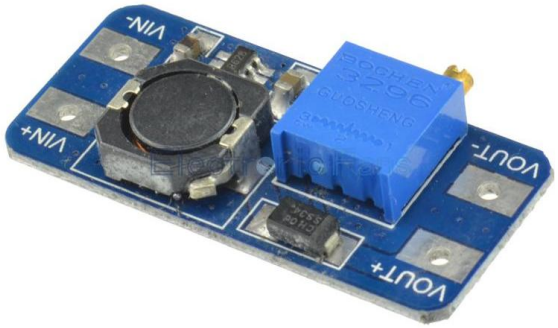


Рис. 2.6. Стабілізатор напруги DC-DC step-up MT3608.

На рис. 2.7 зображена електрична схема модуля стабілізатора напруги DC-DC step-up MT3608.

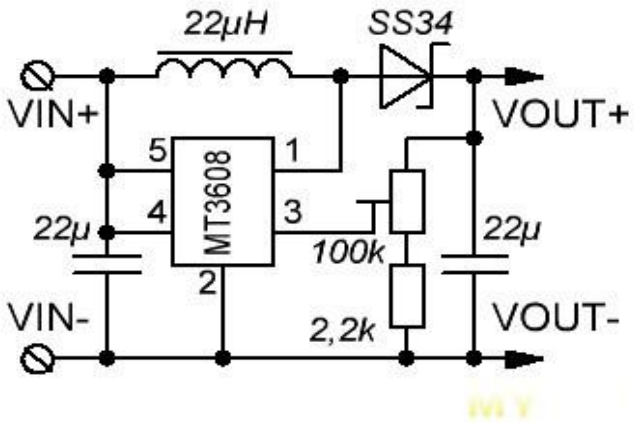


Рис. 2.7. Схема електрична модуля DC-DC step-up MT3608.

2.5. Вибір голосового асистента

В якості голосового асистента в даній роботі обрано “розумну” колонку Amazon Echo і голосовий асистент Alexa [14] (рис. 2.8).

Alexa - це розумний персональний помічник, розроблений компанією Amazon, вперше використаний в Amazon Echo та амазонських Echo Dot пристроях, розроблених Amazon Lab126. Він здатний здійснювати голосові взаємодії, відтворювати музику, створювати списки справ, встановлювати нагадування, транслювати підкасти, відтворювати аудіокниги та надавати інформацію про погоду, трафік та іншу інформацію в реальному часі, наприклад, новини. Alexa також може керувати кількома розумними пристроями, використовуючи себе як систему домашньої автоматизації, що дозволяє змінювати ім'я "Alexa" та багато іншого.

Більшість пристроїв з Alexa дозволяють користувачам активувати пристрій за допомогою пробудження (наприклад, Echo); інші пристрої (наприклад, додаток Amazon на iOS або Android) вимагають від користувача натискання кнопки, щоб активувати режим прослуховування Alexa. В даний час взаємодія та спілкування з Alexa доступні лише англійською та німецькою мовами.

Рис. 2.8. Модуль Amazon Alexa.

Основні можливості Amazon Alexa [14]:

- Голосові дзвінки та обмін повідомленнями.
- Багатофункціональна аудіосистема.
- Spotify Premium.

- Exo Навички в Alexa App. Можна оформити замовлення, не піднімаючи пальця.
- Pandora і Prime Music. Потокове передавання музики Pandora.
- Аудіокниги з Audible
- Кілька таймерів та будильників.
- Контроль розумного будинку.

3. РЕАЛІЗАЦІЯ СИСТЕМИ ГОЛОСОВОГО КЕРУВАННЯ ”РОЗУМНИМ БУДИНКОМ”

3.1. Розроблення структури системи керування

Система керування представляє собою сукупність апаратних та програмних засобів, які призначені для керування сенсорами “розумних систем” за допомогою голосового інтерфейсу, що також надає додаткові можливості, наприклад, контроль присутності.

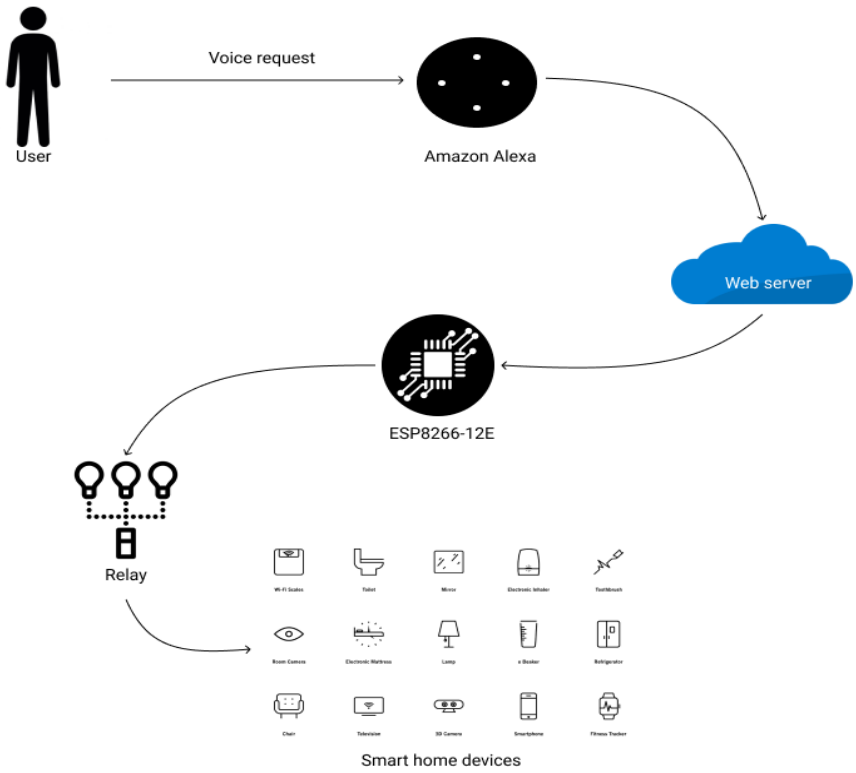


Рис. 3.1.Узагальнена структурна схема системи керування.

Згідно рис. 3.1, роботу системи можна описати наступним чином:

1. користувач робить голосовий запит до модуля Amazon Alexa;

2. Alexa розпізнає намір користувача та надсилає його в JSON форматі в веб-хмару, де розгорнутий веб-сервер;

3. тоді в залежності від того, який це намір(intent), веб-сервер робить відповідний запит на контролер ESP8266, який під’єднаний до Wi-Fi;

4. контролер замикає або розмикає реле.

3.2. Реалізація веб-сервера системи керування

Програма, яка отримує інформацію від Amazon Alexa, обробляє її та висилає запит на веб-клієнт NodeMCU написана з використанням мови програмування Java, а саме, використовуючи Spring Boot та Spring Web MVC фреймворки [15].

Java – це об'єктно-орієнтована мова програмування, виконання коду в якій відбувається з головного класу. Приклад реалізації головного класу:

```
@SpringBootApplication
public class IoTApplication {
    public static void main(String[] args) {
        SpringApplication.run(IoTApplication.class, args);
    }
}
```

По суті, даний клас запускає Spring Boot Framework, який створює всі необхідні біни (об'єкти) та залежності. Також запускається вбудований веб-сервер, який входить в Spring Framework. Далі спілкування між клієнтом і сервером відбувається по REST-методології за допомогою JSON представлення даних (рис. 3.2).

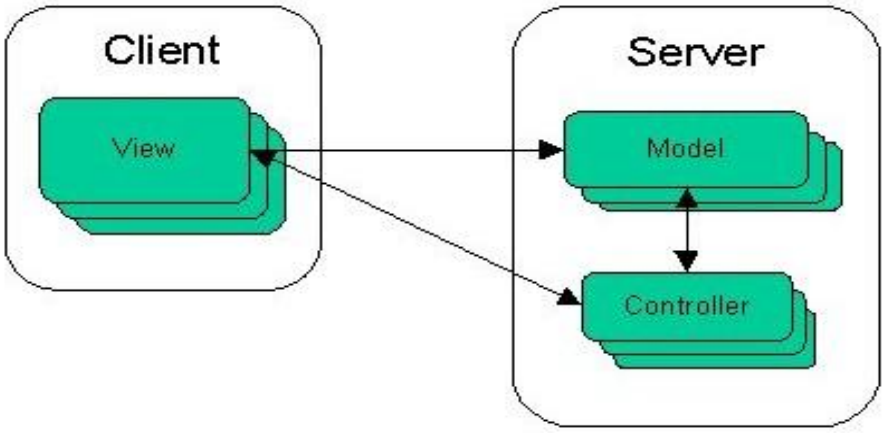


Рис. 3.2. Модель REST-архітектури.

Дана модель поділяє систему на три частини: модель даних, вигляд даних та керування. Застосовується для відокремлення даних (модель) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально

впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Архітектурний шаблон MVC поділяє програму на три частини. У тріаді до обов'язків компоненти Модель (Model) входить зберігання даних і забезпечення інтерфейсу до них. Вигляд (View) відповідальний за представлення цих даних користувачеві. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача, і повідомляє про зміни компоненту Модель. Така внутрішня структура в цілому поділяє систему на самостійні частини і розподіляє відповідальність між різними компонентами.

Для представлення різних моделей даних в Java потрібно створити відповідні класи. Приклади класів для представленнях даних:

Request.class

```
@Data
public class Request {
        private String type;
        private String requestId;
        private String locale;
        private String timestamp;
        private Intent intent;
        private String reason;
 }
@Data
public class Session {
        private String sessionId;
        private Application application;
        private User user;
        @JsonProperty("new")
        private Boolean isNew;
}

public class Intent {
        private String name;
        private Slots slots;
        public String getName() {
```

```
        return name;
            }
            public void setName(String name) {
              this.name = name;
            }
            public Slots getSlots() {
              return slots;
            }
            public void setSlots(Slots slots) {
              this.slots = slots;
            }
    }
    @Data
    public class OutputSpeech {
    private String type;
    private String text;
    private String ssml;
    }
```

За допомогою цих представлень даних проходить спілкування між клієнтом та сервером. Тепер потрібно створити контролери, які будуть отримувати дані та давати відповідь на них. Приклад реалізації класу MainController.java:

```
    @RestController
    public class MainController {
        @Autowired
        private AnswersService answerService;

        @RequestMapping(value = "/alexa/iot", method = RequestMethod.POST)
        public ResponseEntity<ResponseWrapper> catchFilmLink(@RequestBody
RequestWrapper requestWrapper) {
                if (requestWrapper.getSession() == null
||requestWrapper.getRequest()== null || requestWrapper.getVersion() == null) {
                    return new
ResponseEntity<ResponseWrapper>(HttpStatus.BAD_REQUEST);
                }
```

```
            return answerService.handleQuestion(requestWrapper);
        }
    }
```

Даний клас отримує POST запит на відповідний хост, обробляє його, та повертає у відповідь дані у JSON представленні. Параметри цих даних описані в класі ResponseWrapper.java

```
        public class ResponseWrapper {
        private String version;
        private Response response;

        public String getVersion() {
            return version;
        }

        public void setVersion(String version) {
            this.version = version;
        }

        public Response getResponse() {
            return response;
        }
        public void setResponse(Response response) {
            this.response = response;
        }
        public static ResponseWrapper init(Response response) {
            final ResponseWrapper responseWrapper = new
ResponseWrapper();
            responseWrapper.setVersion("1.0");
            responseWrapper.setResponse(response);
            return responseWrapper;
        }
    }
```

Обробка отриманих даних, тобто бізнес-логіка відбувається в класі AnswersService.java:

```
@Service
public class AnswersServiceImpl implements AnswersService {
    @Autowired
    private RestTemplate restTemplate;

    @Override
    public ResponseEntity<ResponseWrapper>
handleQuestion(RequestWrapper requestWrapper) {
        if
(requestWrapper.getRequest().getIntent().getName().equals("TURN_ON_LIGHTS"))
{
            result =
restTemplate.getForObject("http://172.20.10.4/gpio/first_relay/on", String.class);
            speech = "Ok, lights on";
        }

        if
(requestWrapper.getRequest().getIntent().getName().equals("TURN_OFF_LIGHTS"
)) {
            result =
restTemplate.getForObject("http://172.20.10.4/gpio/first_relay/off", String.class);
            speech = "Ok, lights off";
        }
    }
}
```

Даний клас отримує запит від клієнта, в даному випадку це Amazon Alexa, перевіряє намір користувача, і якщо він рівний, наприклад, TURN_ON_LIGHTS, надсилає відповідний запит на NodeMCU. Це дуже гнучка та легкомасштабована модель спілкування між клієнтом та сервером, а Java дозволяє ефективно обробляти велику кількість даних завдяки своїй високій продуктивності та багатопотоковості.

3.3. Реалізація Wi-Fi Web Server на NodeMCU

Модуль мікроконтролера NodeMCU прошивається з використанням мови C та відповідних готових бібліотек. Перш за все потрібно підключити відповідні бібліотеки та налаштувати параметри Wi-Fi точки доступу, до якої ми будемо приєднюватися.

Робиться це наступним чином:

```
#include <ESP8266WiFi.h>
const char* ssid = "iPhone";
const char* password = "604kbyamfq3fo";
```

Наступним кроком потрібно ініціалізувати потрібний GPIO:

```
 pinMode(4, OUTPUT);
 digitalWrite(4, 1);
```

Далі намагаємося підключитися до Wi-Fi точки доступу:

```
WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  }
```

Якщо все відбулося успішно, запускаємо веб-сервер:

```
server.begin();
```

Далі організовуємо вічний цикл, щоб сервер був завжди готовий отримати запит, та перевіряємо в ньому чи сервер є доступний на поточний час:

```
WiFiClient client = server.available();
 if (!client) {
  return;
 }
```

Після цього можна отримати запит від клієнта та, відповідно до цього запиту, подати запит на реле:

```
 if (req.indexOf("/gpio/first_relay/on") != -1){
    digitalWrite(4, 0);
    s += " on";
 }else if(req.indexOf("/gpio/first_relay/off") != -1){
    digitalWrite(4, 1);
    s += " off";
 }
```

Тепер, за допомогою Arduino IDE (рис. 3.3) можна завантажити скетч (прошивку) в мікроконтролер модуля NodeMCU, і кожного разу, коли він буде підключатися до живлення, запускатиметься веб-сервер.

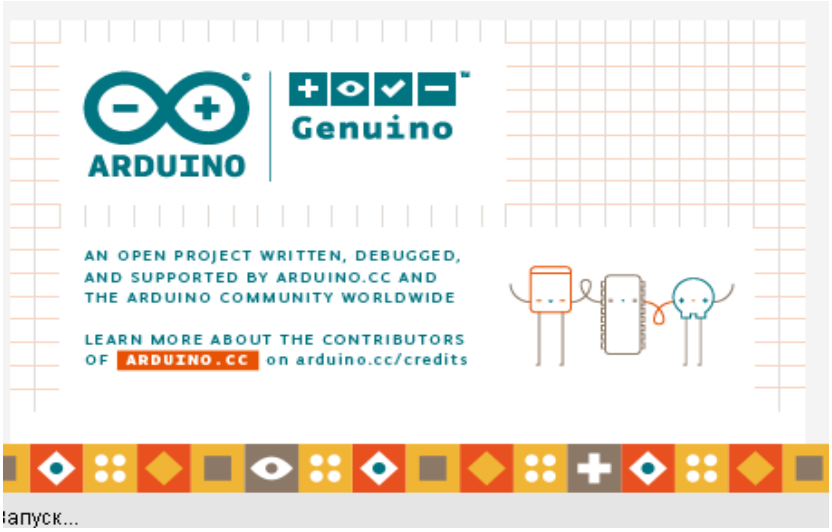


Рис. 3.3. Запуск середовища програмування мікроконтролерів Arduino IDE.

3.4. Реалізація Alexa Skill

Створення навичок для Amazon Alexa відбувається в спеціальній адміністративній панелі, в якій потрібно заповнити необхідні дані про тренування NLP та про власний сервер (рис. 3.4).

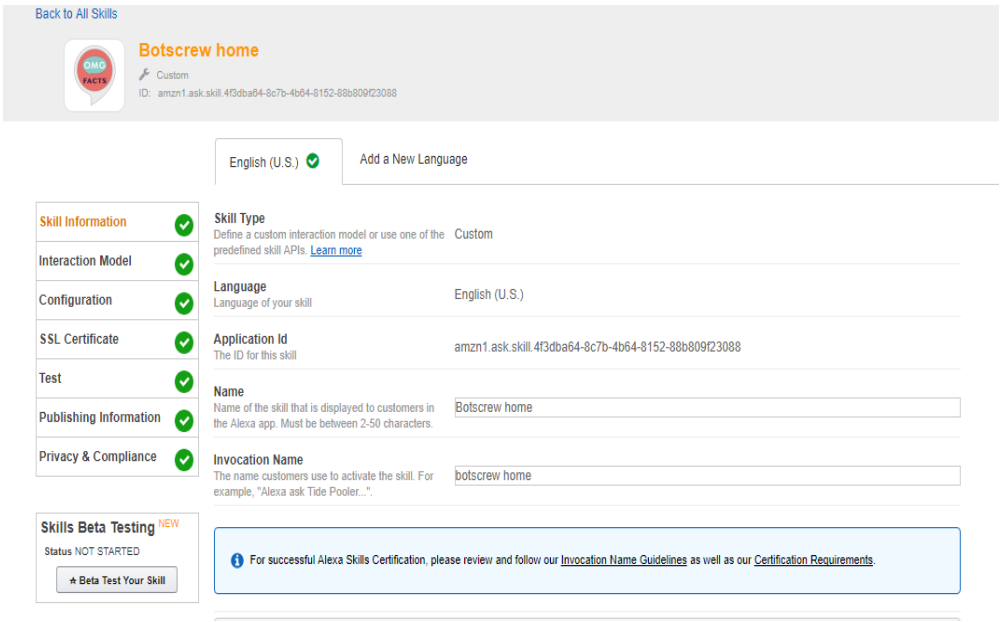


Рис. 3.4. Вигляд панелі налаштувань навичок для Amazon Alexa.

Перш за все необхідно вказати ім’я виклику навички користувача та назву, яка потім відображатиметься в marketplace.

Одним із найважливіших пунктів є **Interaction Model,** оскільки тут відбувається вся конфігурація NLP. Для цього потрібно створити список намірів (intent) в JSON форматі та додати до кожного з них приклади тексту.

Виглядає даний JSON файл таким чином:

```
{
  "intents": [
    {
      "intent": "AMAZON.StopIntent"
    },
    {
      "intent": "TEMPERATURE"
    },
    {
      "intent": "TURN_ON_LIGHTS"
    },
    {
      "intent": "TURN_OFF_LIGHTS"
    },
    {
      "intent": "KETTLE_ON"
    },
    {
      "intent": "KETTLE_OFF"
    }
  ]
}
```

Дані наміри будуть відправлятися на наш бекенд, символізуючи те, що Alexa зрозуміла бажання користувача.

Зразки висловлювань, які потрібно додати до кожного наміру:

AMAZON.StopIntent good bye

TEMPERATURE what is the temperature

TURN_ON_LIGHTS turn on light

TURN_ON_LIGHTS turn on lights

TURN_OFF_LIGHTS turn off lights

TURN_OFF_LIGHTS turn off light

KETTLE_ON turn kettle on

KETTLE_ON turn on kettle

KETTLE_OFF turn kettle off

KETTLE_OFF turn off kettle

Тут все досить просто, потрібно тільки написати назву наміру та додати приклад висловлювання. Коли це зроблено, можна запускати тренування NLP.

Не менш важливою є і вітка Configuration, яка подана на рис. 3.5.

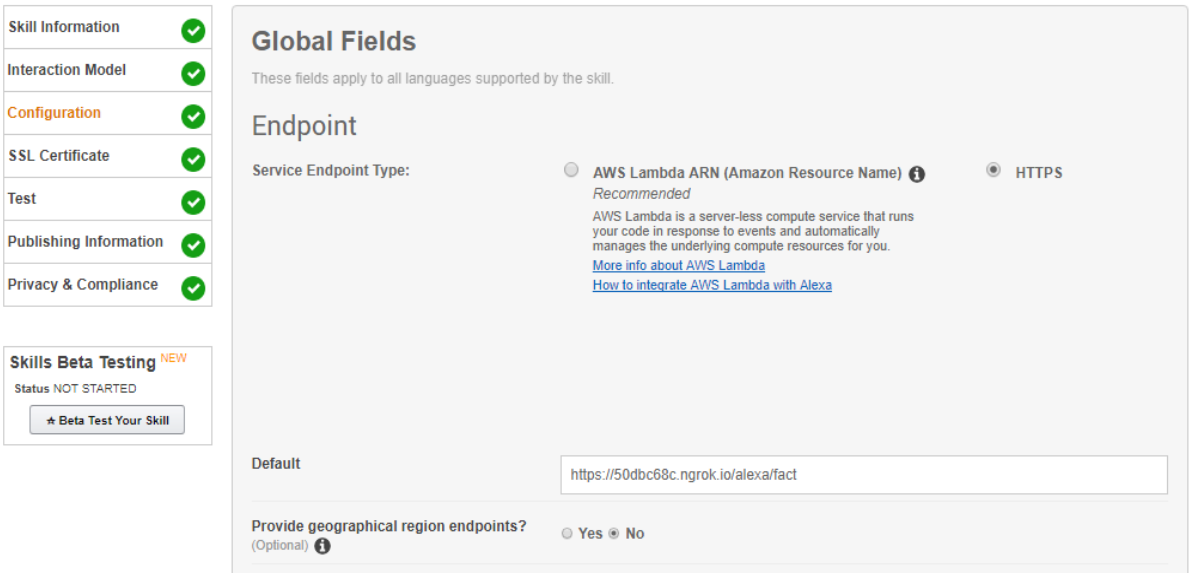


Рис. 3.5. Вигляд вітки Configuration в Alexa Dashboard.

Тут у нас є вибір, який тип сервера вибрати – AWS Lambda чи HTTPS.

В даній роботі використовується HTTPS, а в якості проксі-сервера - ngrok.

Також можна протестувати зроблену роботу за допомогою вітки Test (рис. 3.6):

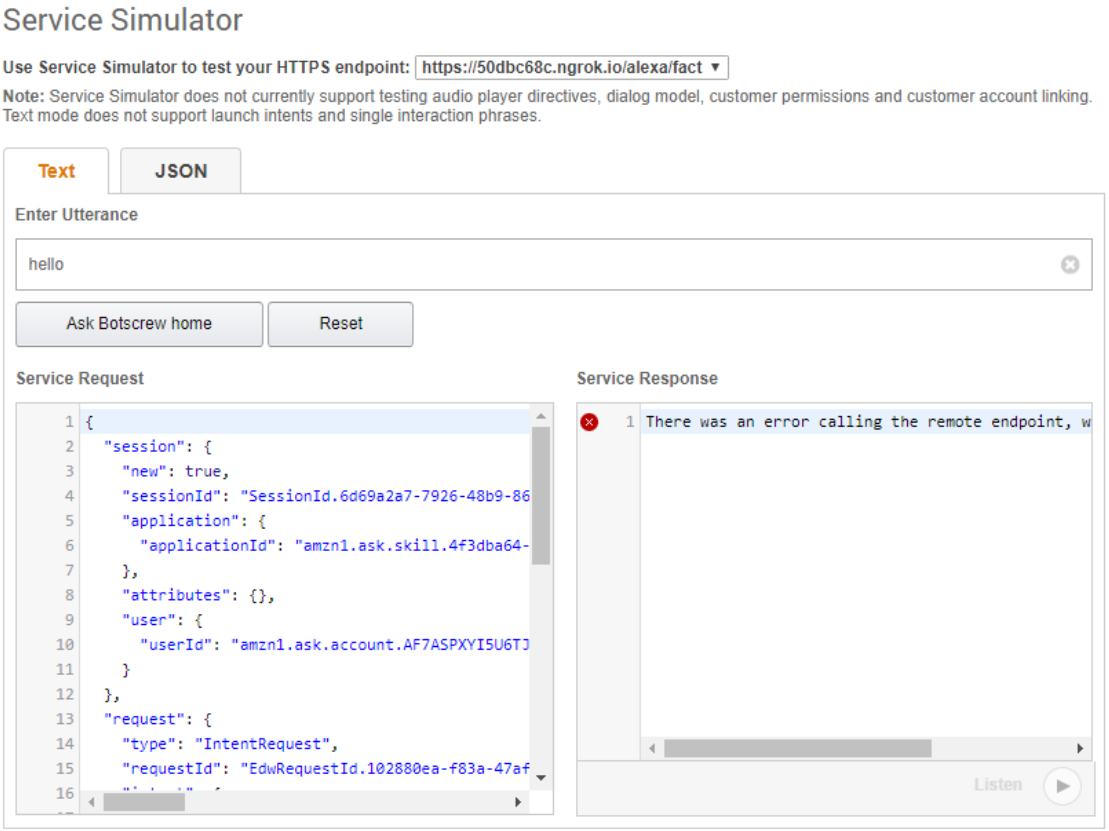


Рис. 3.6. Тестування Amazon Alexa skill.

3.5. Тестування веб-сервера

Для створення навантаження на сервер та тестування його, використано програму із відкритим кодом – JMeter.

Додаток Apache JMeter - це програмне забезпечення з відкритим кодом, що на повністю написане на мові Java, призначене для завантаження функціональної поведінки тестування та вимірювання продуктивності. Спочатку він був призначений тільки для тестування веб-додатків, але в подальшому він розширився до інших тестових функцій.

Перш за все було створено групу потоків. Для початку встановлено наступні параметри:

- Число потоків (умовних користувачів) – 10
- Період тестування (в секундах) – 10
- Число повторювань – 10

Налаштування HTTP запиту виглядає ось так:

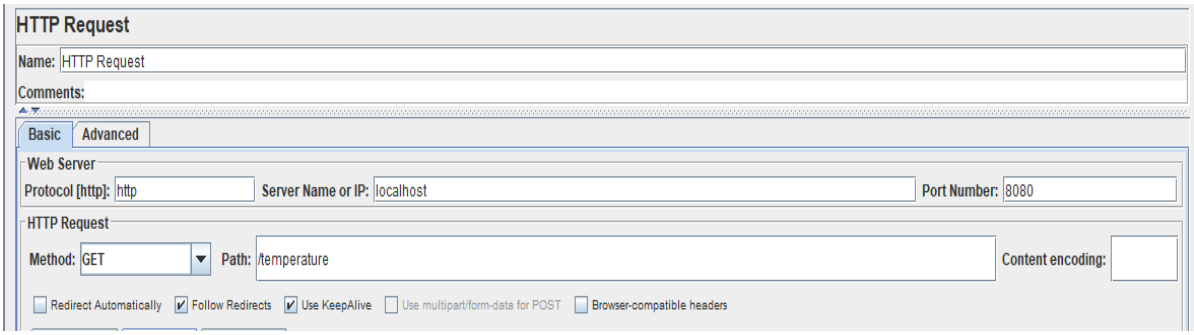


Рис. 3.7. Налаштування програми JMeter.

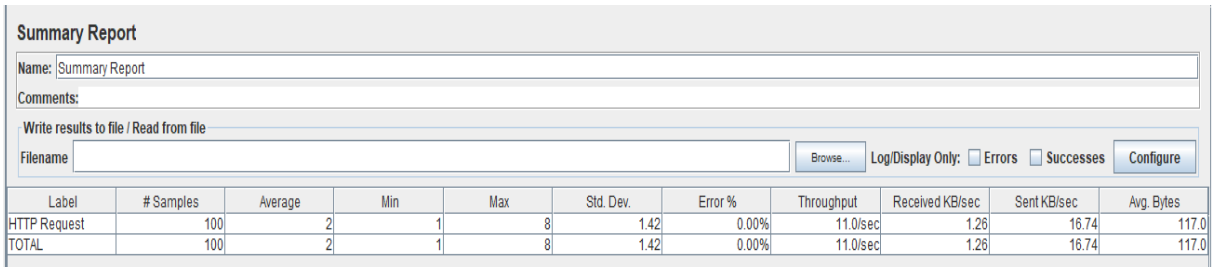
Отже, при проведенні даного тесту було симульовано ситуацію із 10 активними користувачами, які встигають робити по 10 запитів в секунду. Разом виходить 100 запитів на сервер.

Результати виконання тесту:

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
64	22:50:44.543	Thread Group 1-7	HTTP Request	3		117	1553	2	0
65	22:50:44.546	Thread Group 1-7	HTTP Request	3		117	1553	3	0
66	22:50:44.550	Thread Group 1-7	HTTP Request	3		117	1553	3	0
67	22:50:44.554	Thread Group 1-7	HTTP Request	3		117	1553	3	0
68	22:50:44.557	Thread Group 1-7	HTTP Request	3		117	1553	3	0
69	22:50:44.561	Thread Group 1-7	HTTP Request	2		117	1553	2	0
70	22:50:44.563	Thread Group 1-7	HTTP Request	3		117	1553	2	0
71	22:50:45.525	Thread Group 1-8	HTTP Request	6		117	1553	5	2
72	22:50:45.532	Thread Group 1-8	HTTP Request	4		117	1553	4	0
73	22:50:45.537	Thread Group 1-8	HTTP Request	5		117	1553	5	0
74	22:50:45.543	Thread Group 1-8	HTTP Request	5		117	1553	5	0
75	22:50:45.549	Thread Group 1-8	HTTP Request	3		117	1553	3	0
76	22:50:45.553	Thread Group 1-8	HTTP Request	2		117	1553	2	0
77	22:50:45.556	Thread Group 1-8	HTTP Request	2		117	1553	2	0
78	22:50:45.558	Thread Group 1-8	HTTP Request	2		117	1553	2	0
79	22:50:45.561	Thread Group 1-8	HTTP Request	2		117	1553	2	0
80	22:50:45.563	Thread Group 1-8	HTTP Request	3		117	1553	3	0
81	22:50:46.524	Thread Group 1-9	HTTP Request	7		117	1553	7	3
82	22:50:46.532	Thread Group 1-9	HTTP Request	5		117	1553	5	0
83	22:50:46.538	Thread Group 1-9	HTTP Request	4		117	1553	4	0
84	22:50:46.544	Thread Group 1-9	HTTP Request	6		117	1553	6	0
85	22:50:46.551	Thread Group 1-9	HTTP Request	5		117	1553	5	0
86	22:50:46.557	Thread Group 1-9	HTTP Request	5		117	1553	5	0
87	22:50:46.563	Thread Group 1-9	HTTP Request	4		117	1553	4	0
88	22:50:46.568	Thread Group 1-9	HTTP Request	3		117	1553	3	0
89	22:50:46.572	Thread Group 1-9	HTTP Request	2		117	1553	2	0
90	22:50:46.574	Thread Group 1-9	HTTP Request	3		117	1553	3	0
91	22:50:47.525	Thread Group 1-10	HTTP Request	6		117	1553	6	2
92	22:50:47.532	Thread Group 1-10	HTTP Request	5		117	1553	5	0
93	22:50:47.539	Thread Group 1-10	HTTP Request	4		117	1553	4	0
94	22:50:47.544	Thread Group 1-10	HTTP Request	4		117	1553	4	0
95	22:50:47.549	Thread Group 1-10	HTTP Request	5		117	1553	5	0
96	22:50:47.555	Thread Group 1-10	HTTP Request	4		117	1553	4	0
97	22:50:47.560	Thread Group 1-10	HTTP Request	3		117	1553	3	0
98	22:50:47.563	Thread Group 1-10	HTTP Request	3		117	1553	3	0
99	22:50:47.567	Thread Group 1-10	HTTP Request	2		117	1553	2	0
100	22:50:47.570	Thread Group 1-10	HTTP Request	2		117	1553	2	0

Scroll automatically? Child samples? No of Samples 100 Latest Sample 2 Average 2 Deviation 1

Рис. 3.8. Таблиця із результатом виконання тесту 10 користувачів.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	100	2	1	8	1.42	0.00%	11.0/sec	1.26	16.74	117.0
TOTAL	100	2	1	8	1.42	0.00%	11.0/sec	1.26	16.74	117.0

Рис. 3.9. Загальний звіт виконання тесту із 10 користувачами.

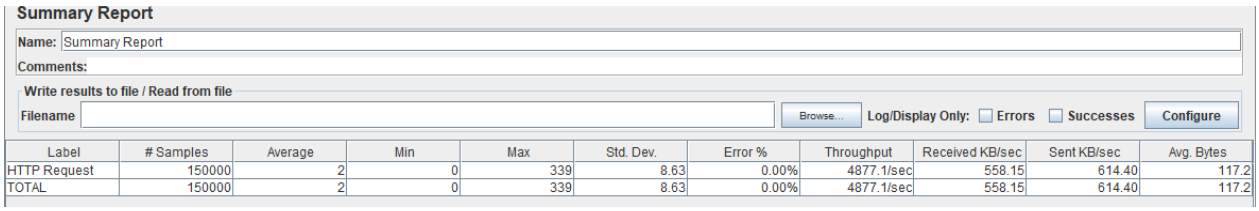
Як можна побачити із наведених в рис. 3.7. та рис. 3.8. результатів, середній час відповіді сервера складає 2 мс., а максимальний час відповіді – 8 мс. По цих даних можна зробити висновок, що 10 дуже активних користувачів

для нашого сервера взагалі не є проблемою, а в межах одного будинку, це досить багато. Наступне збільшення показників до 100 активних користувачів показало майже той самий результат, що і при 10. В зв'язку з цим, довелося суттєво підняти тестові показники, щоб хоч якось урізноманітнити дані.

Наступними параметрами для проведення тестів було обрано:

- Число потоків (умовних користувачів) – 300
- Період тестування (в секундах) – 30
- Число повторювань – 500

В результаті даного тесту, сервер успішно обробив 150 000 запитів за 30 секунд. Сумарні результати тестування показані на рис. 3.9., а графік на рис.3.10.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	150000	2	0	339	8.63	0.00%	4877.1/sec	558.15	614.40	117.2
TOTAL	150000	2	0	339	8.63	0.00%	4877.1/sec	558.15	614.40	117.2

Рис. 3.10. Сумарний звіт проведеного тестування

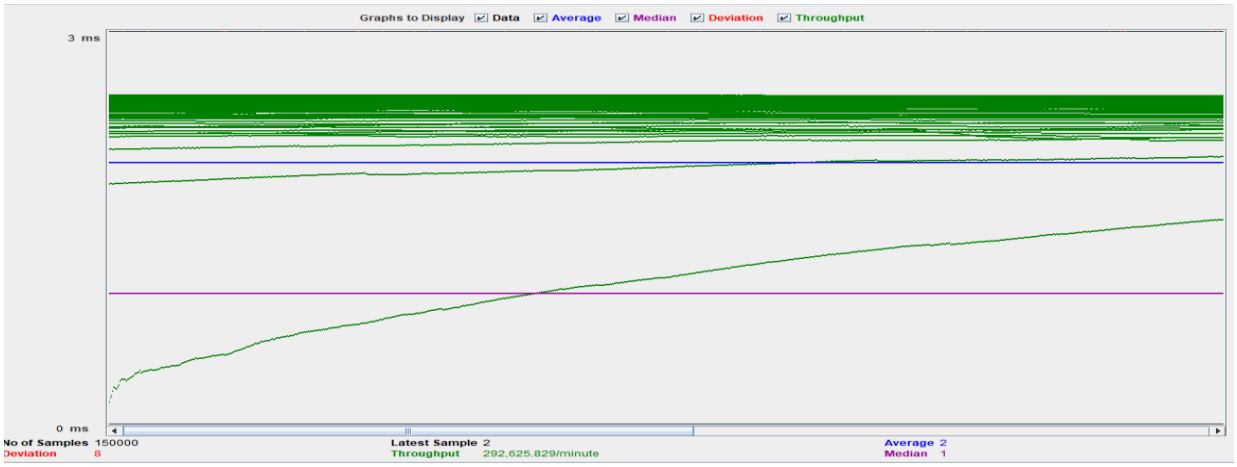


Рис. 3.11. Графік проведеного тесту.

Отже, згідно результатів проведеного тестування, веб-сервер може витримувати дуже значні навантаження та час відповіді на запити залишатиметься майже незмінним.

3.6. Реалізація голосового керування системою протипожежної безпеки ”розумного будинку”

Розглянемо реалізацію голосового керування однією із ситем керування “розумним будинком”. На рис. 3.12 подано приклад системи голосового керування системою протипожежної безпеки ”розумного будинку”.

Спочатку користувач надсилає голосовий запит про стан сенсорів протипожежної системи безпеки ”розумного будинку”. Наступним кроком молуль Amazon Alexa в якому є функція голосового асистента розпізнає надіслану голосову команду користувача та надсилає її в JSON форматі в веб-хмару, де розгорнутий веб-сервер. Далі веб-сервер робить відповідний запит на мікроконтролер Node MCU ESP8266, який під’єднаний до безпровідного інтерфейсу Wi-Fi.

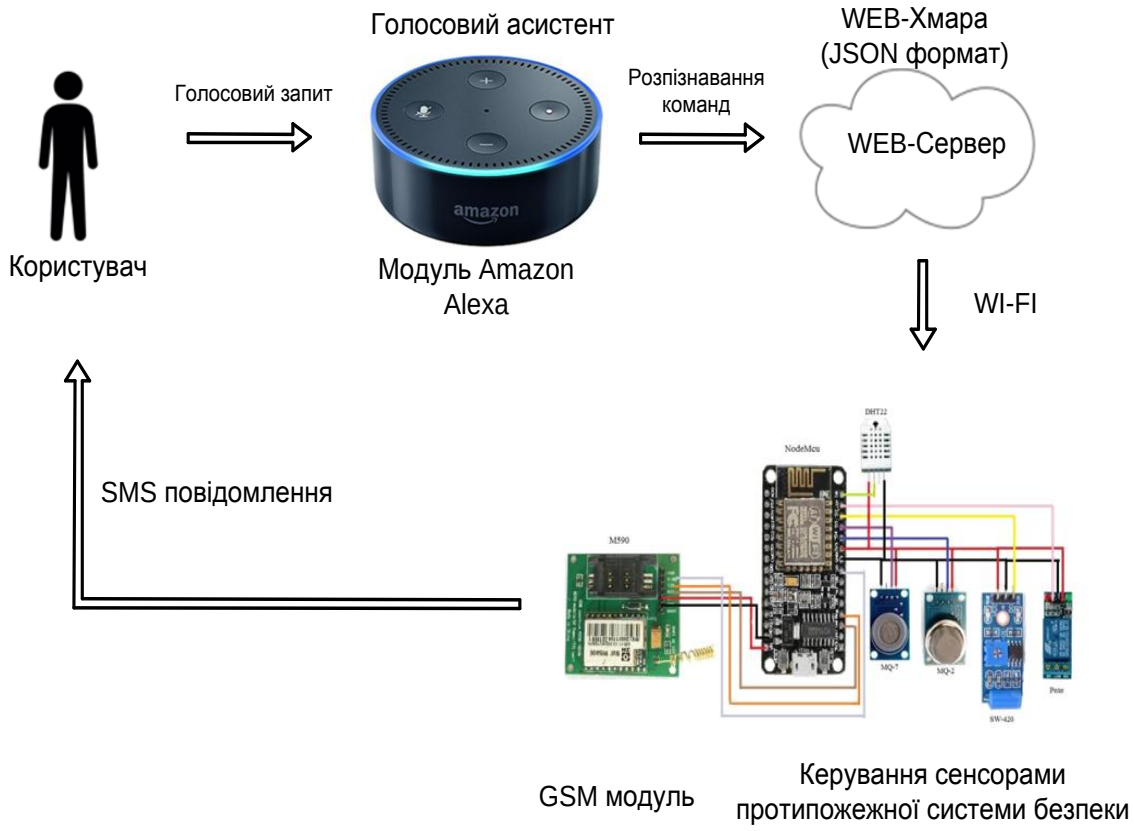


Рис. 3.12. Голосове керування системою протипожежної безпеки ”розумного будинку”.

Мікроконтролер Node MCU ESP8266 опитує стан давача температури та вологості, давача визначення чадного газу та давача фіксації диму в приміщення та за допомогою додаткового GSM модуля користувачу приходять дані про стан давачів та інформація про інші запити у вигляді SMS-повідомлення.

3.7. Алгоритм функіцонування системи голосового керування

На рис. 3.13 подано узагальнену блок-схему алгоритму функціонування системи голосового керування елементами ”розумнгого будинку”.

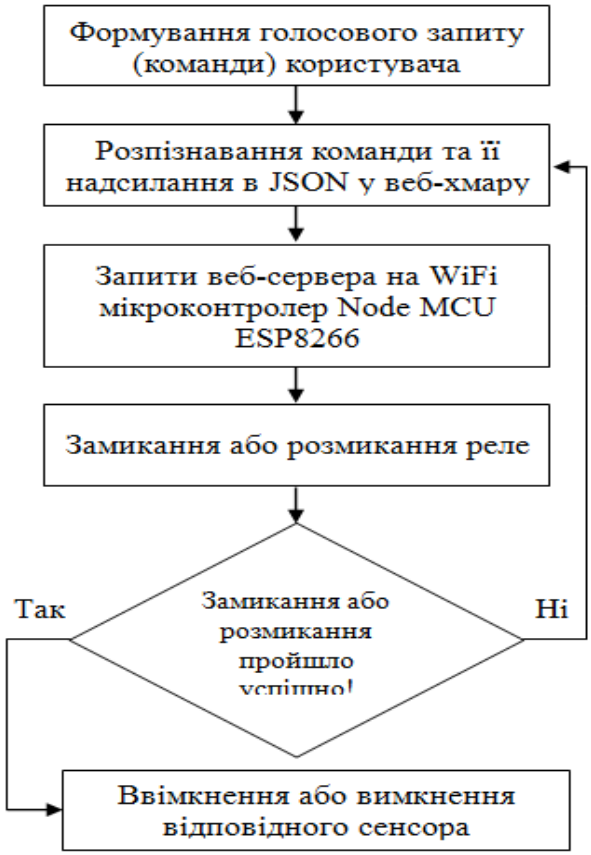


Рис. 3.13. Блок-схема алгоритму функціонування системи голосовогоо керування.

Узагальнений алгоритм роботи системи керування розпочинається із формування голосових запитів або команд користувача, наступним кроком модуль Alexa розпізнає дані команди і надсилає їх в JSON форматі у веб-хмару де реалізований веб-сервер, оскільки реалізований веб-сервер зв’язаний із апаратним мікропроцесорним модулем Node MCU ESP8266 через інтерфейс

WiFi, то зі сторони веб-сервера формуються відповідні запити і відправляються на апартний модуль в результаті чого модуль починає керувати різними сенсорами ”розумного будинку” вмикаючи їх або вимикаючи за допомогою модуля реле.

4. ЕКОНОМІЧНА ЧАСТИНА

4.1. Визначення собівартості та ціни спроектованого приладу

Визначення виробничої собівартості спроектованого приладу здійснюється за питомою вагою у ньому окремих елементів витрат. В таблиці 4.1 наведено вартість комплектуючих виробів.

Таблиця 4.1

Вартість комплектуючих виробів системи

Комплектуючі вироби	Кількість, шт.	Вартість за одиницю, грн.	Сума, грн.
NODEMCU	2	214	428
Одноканальний модуль реле	2	68	136
Amazon Alexa	1	1777	1777
DC-DCSTEP-UP MT3608	2	50	100
Електричний чайник	1	909	909
Настільна лампа	1	100	100
Всього			3450

Для визначення економічного ефекту в умовах виробництва знаходимо ціну спроектованого пристрою:

$$Ц_2 = 5573,07 \cdot (1 + 30/100) = 7244,99 \text{ (грн.)}$$

Отже, економічний ефект у сфері виробництва становить:

$$Ев = 8000 – 7244,99 = 755,01 \text{ (грн.)}$$

4.2. Визначення економічного ефекту в сфері експлуатації

Річний економічний ефект на витратах на енергію визначається:

$$E_{ен} = (M_1 – M_2) * T * a \qquad (4.1)$$

де, M_1, M_2 – споживані потужності відповідно аналога та спроектованого приладу.

M_1 = 15,13 кВт/рік – споживана потужність аналога в рік;

M_2 = 15,09 кВт/рік - споживана потужність спроектованого приладу в рік;

а = 1,68 грн – тариф за 1 кВт/год.;

Т = 264·8 – кількість робочих годин на рік (згідно з прийнятим бюджетом робочого часу на відповідний період);

$$Е_{ен} = (15,13 – 15,09) \cdot 2112 \cdot 1,68 = 141,92 \text{ (грн/рік)}$$

Річний економічний ефект по заробітній платі у нашому випадку :

$Е_{зпл.} = 0$, так як зарплата фахівця з обслуговування залишилася такою ж.

Оскільки конструкцією контролера ремонт не передбачений, то Ер=0.

Сумарний річний економічний ефект визначається за формулою:

$$Е_{ер} = 75,65 + 141,92 + 0 + 0 = 217,57 \text{ (грн/рік)}$$

Термін експлуатації пристрою становить 8 років. Річний

$Е_{ер}$ = 217,57 грн/рік. Тоді сумарний економічний ефект за термін експлуатації становить:

$$Е_е = 217,57 \cdot 3,19 + 217,57 \cdot 2,7 + 217,57 \cdot 2,29 + 217,57 \cdot 1,94 + 217,57 \cdot 1,64 + 217,57 \cdot 1,39 + 217,57 \cdot 1,18 + 217,57 \cdot 1 = 3335,34 \text{ (грн.)}$$

Загальний економічний ефект:

$$Е_з = Е_в + Е_е = 755,01 + 3335,34 = 4090,35 \text{ (грн.)}$$

Отже, в даному розділі було проведено економічне обґрунтування доцільності розробки та впровадження у виробництво розробленої в дипломній роботі системи.

Проведений аналіз показав, що проектна розробка є кращою за своїми параметрами у порівнянні з аналогом, за рахунок нижчої собівартості та вищих показників продуктивності, а її комплексний показник якості складає 1,265. Умова, за якої розрахункова оптова ціна розробленого виробу має бути меншою від лімітної ціни спроектованого виробу – виконується. Для спроектованої системи є характерним загальне додатне значення сумарного економічного ефекту ($Е_з$ = 4090,35 грн. > 0). Значення ефекту у сфері виробництва є додатнім ($Е_в$ = 755,01 > 0). Також додатне є значення ефекту у сфері використання спроектованої системи ($Е_е$ = 3335,34 грн. > 0).

Розроблений пристрій є досить інноваційним рішенням в галузі “Інтернет речей”. Він дозволяє керувати розумним будинком за допомогою голосу, який

аналізується штучним інтелектом. На відміну від існуючих платформ, розроблене рішення є позитивним тому сенсі, що його можливо легко адаптувати під різноманітні пристрої і сенсорів, задаючи ланцюжки обробки даних. Розроблена система орієнтована на широке коло користувачів.

ВИСНОВКИ

В результаті виконання даної бакалаврської кваліфікаційної роботи виконано наступні завдання:

1. Проведено огляд та аналіз голосових технологій ”Інтернет речей”. Розглянуто принципи роботи та способи керування системами “розумного будинку”.

2. Розглянуто відомі існуючі рішення голосових технологій провідних фірм виробників та визначено їхні основні переваги та недоліки.

3. Виконано вибір та обґрунтування програмних та апаратних засобів реалізації системи голосового керування елементами ”розумного будинку”. Для реалізації програмної частини веб-сервера вибрано мову програмування Java, технологію Spring Framework та середовище програмування Eclipse. Для реалізації апаратної частини вибрано мікропроцесорний модуль NodeMCU ESP8266 з вбудованим інтерфейсом WiFi, модуль реле, модуль Amazon Alexa та середовище програмування мікропроцесорних пристроїв Adruino IDE.

4. Розроблено структурну схему голосової системи керування, яка складається з прогамної та апаратної частини. Реалізовано веб-сервер Amazon Alexa та веб-сервер на модулі WiFi NodeMCU ESP8266.

5. Розроблено програмне забезпечення системи голосового керування та описано алгоритм її функціонування. Виконано тестування веб-сервера.

6. Розроблено економічну оцінку готового пристрою, яка у порівнянні з аналогами має нижчу собівартість.

Розроблений пристрій є досить інноваційним рішенням в галузі “Інтернет речей” та дозволяє керувати елементами “розумного будинку” за допомогою голосу, який аналізується штучним інтелектом. На відміну від існуючих платформ, розроблене рішення є легко адаптованим для різноманітних пристроїв і сенсорів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бучма І.М. Мікропроцесорні пристрої Навчальний посібник. Львів: Видавництво Львівської політехніки, 2005. – 236 с.
2. Гололобов В. Н. «Умный дом» своими руками / В. Н. Гололобов. – М.: НТ Пресс, 2007. – 216 с.
3. Петин В. А. Создание умного дома на базе Arduino. – М.: ДМК Пресс, 2018. – 180 с.
4. Елементи системи розумний будинок, їх призначення та принцип роботи – Режим доступу: http://mastery-of-building.org/uk/sostavlyayushhie-elementy-sistemy-umnyj-dom-ix-naznachenie-i-princip-raboty/#i. – Дата доступу: 25.03.2020 р
5. Особливості системи «Розумний будинок» – Режим доступу: http://nauka.zinet.info/26/shulgan.php – Дата доступу: 19.03.2020 р.
6. Розумний будинок – що можуть сучасні системи автоматизації – Режим доступу: http://elektruk.info/main/automation/1187-umnyy-dom-sistemy-avtomatizacii.html – Дата доступу: 25.03.2020 р.
7. Інформація про платформу AWS IoT. – https://aws.amazon.com/iot/how-it-works/ - Дата доступу: 19.04.2020 р
8. Інформація про платформу IBM IoT Platform.: http://www.ibm.com/internet-of-things/iot-platform.html - Дата доступу: 19.03.20 р
9. Офіційна документація Oracle по Java - https://docs.oracle.com/javase/7/docs/api - Дата доступу: 19.04.2020 р
10. Інформація про платформу Oracle IoT - https://cloud.oracle.com/iot - Дата доступу: 19.04.2020 р
11. Офіційна документація по Spring Framework - https://spring.io/docs - Дата доступу: 19.03.2020 р
12. Arduino IDE- Режим доступу: https://www.arduino.cc/en/main/software - Дата доступу: 30.04.2020 р.

13. Інформація про NodeMCU ESP8266 https://diylab.com.ua/p280592656-modul-nodemcu-esp8266.html - Дата доступу: 19.04.2020 р

14. Інформація про Amazon Alexa - https://en.wikipedia.org/wiki/Amazon_Alexa - Дата доступу: 19.03.2020 р

15. Документація по налаштуванню веб-сервера для створення Alexa skill - https://developer.amazon.com/docs/custom-skills/understanding-custom-skills.html - Дата доступу: 19.03.2020 р

16. Економіка підприємства: Підручник / за заг. редакцією Й.М. Петровича.- Львів: “Магнолія плюс”, Видавець В.М. Піча– 2004.– 680 с.

17. О.О. Гетьман, В.М. Шаповал Економіка Підприємства : Навчальний посібник для студентів вищих навчальних закладів. – К: Центр навчальної літератури) 2006. – 488 с.

ДОДАТКИ

Додаток А.

Лістинг програми налаштування веб-сервера

AlexaIoTApplication.java

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class AlexaIoTApplication {
    public static void main(String[] args) {
    SpringApplication.run(AlexaIoTApplication.class, args);
    }
}
```

ServletInitializer.java

```
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.support.SpringBootServletInitializer;
public class ServletInitializer extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
    return application.sources(AlexaIoTApplication.class);
    }
}
```

ApplicationConfiguration.java

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.converter.ByteArrayHttpMessageConverter;
import org.springframework.http.converter.ResourceHttpMessageConverter;
import org.springframework.http.converter.StringHttpMessageConverter;
import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
import org.springframework.http.converter.support.AllEncompassingFormHttpMessageConverter;
import org.springframework.http.converter.xml.SourceHttpMessageConverter;
import org.springframework.web.client.RestTemplate;
import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;
@Configuration
public class ApplicationConfiguration {
    @Bean
    public RestTemplate restTemplate() {
    final RestTemplate restTemplate = new RestTemplate();
    restTemplate.setMessageConverters(Arrays.asList(new ByteArrayHttpMessageConverter(),
new StringHttpMessageConverter(),
                new ResourceHttpMessageConverter(), new SourceHttpMessageConverter<>(),
new AllEncompassingFormHttpMessageConverter(),
                new MappingJackson2HttpMessageConverter(jacksonObjectMapper())));
    return restTemplate;
    }
    @Bean
    public ObjectMapper jacksonObjectMapper() {
    final ObjectMapper objectMapper = new ObjectMapper();
    objectMapper.configure(DeserializationFeature.ACCEPT_SINGLE_VALUE_AS_ARRAY,
true);
```

```
    objectMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,
false);
    return objectMapper;
    }
}
```

MainController.java

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.botscrew.funFacts.model.RequestWrapper;
import com.botscrew.funFacts.model.ResponseWrapper;
import com.botscrew.funFacts.service.AnswersService;

@RestController
public class MainController {

    @Autowired
    private AnswersService answerService;

    @RequestMapping(value = "/alexa/iot", method = RequestMethod.POST)
    public ResponseEntity<ResponseWrapper> catchFilmLink(@RequestBody RequestWrapper
requestWrapper) {
    if (requestWrapper.getSession() == null || requestWrapper.getRequest() == null ||
requestWrapper.getVersion() == null) {
            return new ResponseEntity<ResponseWrapper>(HttpStatus.BAD_REQUEST);
    }
    return answerService.handleQuestion(requestWrapper);
    }

    @RequestMapping(value = "/temperature", method = RequestMethod.GET)
    public String catchtemp() {
    return "here";
    // return answerService.getTemp().replaceAll("\r\n",
    // "").split("<html>")[1].split("</html>")[0].replaceAll("\r\n", "");
    }
}
```

AnswersService.java

```
import com.botscrew.funFacts.model.RequestWrapper;
import com.botscrew.funFacts.model.ResponseWrapper;
public interface AnswersService {
    ResponseEntity<ResponseWrapper> handleQuestion(RequestWrapper requestWrapper);
    String getTemp();
}
```

AnswersServiceImpl.java

```
import java.util.ArrayList;
import java.util.List;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;
import com.botscrew.funFacts.model.OutputSpeech;
import com.botscrew.funFacts.model.Reprompt;
import com.botscrew.funFacts.model.RequestWrapper;
import com.botscrew.funFacts.model.Response;
import com.botscrew.funFacts.model.ResponseWrapper;
import com.botscrew.funFacts.service.AnswersService;

@Service
public class AnswersServiceImpl implements AnswersService {
    @Autowired
    private RestTemplate restTemplate;
    private static final List<String> facts = new ArrayList<>();

    @Override
    public ResponseEntity<ResponseWrapper> handleQuestion(RequestWrapper requestWrapper)
{
    String result = "";
    String speech = "I didn't get it";
    System.out.println(requestWrapper.getRequest().getIntent().getName());
    if (requestWrapper.getRequest().getIntent().getName().equals("TEMPERATURE")) {
            result                                =                                getTemp().replaceAll("\r\n",
"").split("<html>")[1].split("</html>")[0].replaceAll("\r\n", "");

            System.out.println(result);
            // speech = "Ok, lights on";
    }

    if (requestWrapper.getRequest().getIntent().getName().equals("TURN_ON_LIGHTS")) {
            result        =        restTemplate.getForObject("http://172.20.10.4/gpio/first_relay/on",
String.class);
            speech = "Ok, lights on";
    }

    if (requestWrapper.getRequest().getIntent().getName().equals("TURN_OFF_LIGHTS")) {
            result        =        restTemplate.getForObject("http://172.20.10.4/gpio/first_relay/off",
String.class);
            speech = "Ok, lights off";
    }

    if (requestWrapper.getRequest().getIntent().getName().equals("FIRST_RELAY_ON")) {
            result        =        restTemplate.getForObject("http://192.168.0.109/gpio/first_relay_on",
String.class);
            speech = "Ok, first relay on";
    }

    if (requestWrapper.getRequest().getIntent().getName().equals("FIRST_RELAY_OFF")) {
```

```
            result = restTemplate.getForObject("http://192.168.0.109/gpio/first_relay_off",
String.class);
            speech = "Ok, first relay off";
        }

        if (requestWrapper.getRequest().getIntent().getName().equals("KETTLE_ON")) {
            result = restTemplate.getForObject("http://172.20.10.3/gpio/first_relay_on",
String.class);
            speech = "Ok, kettle on";
        }

        if (requestWrapper.getRequest().getIntent().getName().equals("KETTLE_OFF")) {
            result = restTemplate.getForObject("http://172.20.10.3/gpio/first_relay_off",
String.class);
            speech = "Ok, kettle off";
        }

        if (requestWrapper.getRequest().getIntent().getName().equals("OFF_ALL_DEVICES")) {
            result = restTemplate.getForObject("http://192.168.0.134/gpio/first_relay_off",
String.class);
            result = restTemplate.getForObject("http://192.168.0.111/gpio/first_relay_off",
String.class);
            speech = "Ok, all devices are off";
        }

        final OutputSpeech outputSpeech = OutputSpeech.buildPlainSpeech(speech);
        final Reprompt reprompt = Reprompt.buildReprompt("I am waiting");
        final Response response = Response.buildResponse(reprompt, outputSpeech, true);
        return new ResponseEntity<>(ResponseWrapper.init(response), HttpStatus.OK);
        }

        @Override
        public String getTemp() {
        return restTemplate.getForObject("http://192.168.2.83/temperature", String.class);
        }
```

Application.java

```
public class Application {
    private String applicationId;
    public String getApplicationId() {
    return applicationId;
    }
    public void setApplicationId(String applicationId) {
    this.applicationId = applicationId;
    }
}
```

OutputSpeech.java

```
public class OutputSpeech {
    private String type;
    private String text;
    private String ssml;

    public String getType() {
```

```
        return type;
        }

        public void setType(String type) {
        this.type = type;
        }

        public String getText() {
        return text;
        }

        public void setText(String text) {
        this.text = text;
        }

        public String getSsml() {
        return ssml;
        }

        public void setSsml(String ssml) {
        this.ssml = ssml;
        }

        public static OutputSpeech buildPlainSpeech(String text) {
        final OutputSpeech speech = new OutputSpeech();
        speech.setType("PlainText");
        speech.setText(text);
        return speech;
        }
}
```

Request.java

```
public class Request {
        private String type;
        private String requestId;
        private String locale;
        private String timestamp;
        private Intent intent;
        private String reason;

        public String getType() {
        return type;
        }

        public void setType(String type) {
        this.type = type;
        }

        public String getRequestId() {
        return requestId;
        }

        public void setRequestId(String requestId) {
```

```
this.requestId = requestId;
}

public String getLocale() {
return locale;
}

public void setLocale(String locale) {
this.locale = locale;
}

public String getTimestamp() {
return timestamp;
}

public void setTimestamp(String timestamp) {
this.timestamp = timestamp;
}
public Intent getIntent() {

return intent;

}

public void setIntent(Intent intent) {

this.intent = intent;

}

public String getReason() {

return reason;

}

public void setReason(String reason) {

this.reason = reason;

}
}
```

Додаток А.

Лістинг програми для реалізації веб-сервера на Node MCU ESP8266

```
#include <ESP8266WiFi.h>
const char* ssid = "iPhone";
const char* password = "604kbyamfq3fo";

// Create an instance of the server
// specify the port to listen on as an argument
WiFiServer server(80);

void setup() {
  Serial.begin(115200);
  delay(10);

  // prepare GPIO2
  pinMode(4, OUTPUT);
  digitalWrite(4, 1);

  // Connect to WiFi network
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  // Start the server
  server.begin();
  Serial.println("Server started");

  // Print the IP address
  Serial.println(WiFi.localIP());
}

void loop() {
  // Check if a client has connected
  WiFiClient client = server.available();
  if (!client) {
    return;
  }

  // Wait until the client sends some data
  Serial.println("new client");
```

```
  while(!client.available()){
    delay(1);
  }

  // Read the first line of the request
  String req = client.readStringUntil('\r');
  Serial.println(req);
  client.flush();

  // Prepare the response
  String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE
HTML>\r\n<html>\r\nRelay";

  // Match the request
  if (req.indexOf("/gpio/first_relay/on") != -1){
    digitalWrite(4, 0);
    s += " on";
  }else if(req.indexOf("/gpio/first_relay/off") != -1){
    digitalWrite(4, 1);
    s += " off";
  }

  client.flush();
  s += "</html>\n";

  // Send the response to the client
  client.print(s);
  delay(1);
  Serial.println("Client disonnected");

  // The client will actually be disconnected
  // when the function returns and 'client' object is detroyed
```