Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

факультет Електронної та біомедичної інженерії
(повна назва)

Кафедра Мікроелектроніки електронних приладів та пристроїв
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерській)
(рівень вищої освіти)
«Обробка інформаційних сигналів витратомірів на основі взаємодії ультразвукових і електромагнітних коливань з рухомим середовищем»
(тема)

Виконав: студент 2 курсу II рівня вищої освіти
групи ЕППм -18 -1
спеціальності 171 – «Електроніка»
(код і повна назва спеціальності)
Золотухін А.А.
(прізвище, ініціали)
Тип програми освітньо-професійна
Освітня програма «Електронні прилади та пристрої»
(повна назва освітньої програми)
Керівник зав. каф. МЕЕПП Бондаренко І.М.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Бондаренко І.М.
(підпис) (прізвище, ініціали)

2019 р.

Харківський національний університет радіоелектроніки

факультет Електронної та біомедичної інженерії
Кафедра Мікроелектроніки електронних приладів та пристроїв
Рівень вищої освіти другий (магістерській)
спеціальність 171 «Електроніка»
(шифр и назва)
Тип програми освітньо-професійна

Освітня програма «Електронні прилади та пристрої »
(назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____» _____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Золотухіну Антону Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Обробка інформаційних сигналів витратомірів на основі взаємодії ультразвукових і електромагнітних коливань з рухомим середовищем»

ЗАТВЕРДЖЕНА наказом по університету від " 04 " листопаду 2019р. № 1635 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 10.12.2019 р.
3. Вихідні дані до роботи: Визначити особливості побудови систем обробки інформаційних сигналів витратомірів на основі взаємодії ультразвукових і електромагнітних коливань з рухомим середовищем.

4. Перелік запитань, що потрібно опрацювати в роботі Вступ. 1.Засоби вимірювання втрат в системах з рухомим середовищем. 2.Види інформаційних сигналів потокових витратомірів.3.Алгоритми обробки інформаційних сигналів витратомірів. 4. Особливості обробки інформаційних сигналів витратомірів на основі взаємодії ультразвукових і електромагнітних коливань з рухомим середовищем. 5. Функціональна схема обробки інформаційних сигналів потокових витратомірів, що розглядаються.

5. Перелік графічного матеріалу (із зазначеним креслеників, схем, плакатів, комп'ютерних ілюстрацій, слайдів)
Демонстраційний матеріал – 10 шт

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	термін Виконання етапів роботи	Примітка
1	Інформаційно-тематичний пошук та огляд літературних джерел про реалізацію системи відображення для потокових витратомірів	15.05.19 – 20.06.19	Виконано
2	Інформаційно-тематичний пошук та огляд літературних джерел щодо використовуваних технологій розробки системи відображення	21.06.19 – 24.07.19	Виконано
3	Інформаційно-тематичний пошук та огляд літературних джерел щодо клієнтської та серверної частини реалізації системи	25.07.19 – 28.09.19	Виконано
4	Оформлення пояснювальної записки	02.10.19 – 14.11.19	Виконано
5	Оформлення графічних та демонстраційних матеріалів	15.11.19 – 29.11.19	Виконано
6	Проходження нормоконтролю та отримання рецензії	01.12.19 – 11.12.19	Виконано
7	Підготовка та захист атестаційної роботи	12.12.19 – 16.12.19	Виконано

Дата видачі завдання ____ _____ _____р

студент _____
(підпис)

Керівник роботи _____ зав. каф. Бондаренко І.М.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 57 сторінок, 17 рисунків, 3 додатки, 6 джерел.

ВИТРАТОМІР, ДОДАТОК, БАЗА ДАНИХ, СЕРВЕР, ФРЕЙМВОРК, ПРОГРАМУВАННЯ, СХЕМА.

Об’єкт дослідження – процес взаємодії даних витратомірів на основі ультразвукових коливань у клієнтському додатку.

Мета роботи – проектування та конструювання розумної системи обробки та виводу даних витратомірів на основі серверних та клієнтських web технологій.

Метод дослідження – теоретичний: аналіз існуючих систем обробки даних витратомірів та розгляд принципів їх роботи.

У першому розділі атестаційної роботи розглянуті основні принципи та особливості реалізації системи моніторингу даних витратомірів. Досліджені відмінності та принципові аргументи щодо застосування технології на теоретичній основі. Детально розглянута робота витратомірів та їх інтерфейс взаємодії на фізичному рівні.

У другому розділі інформація розподілена у вигляді тезисного описання побудованого об’єкта дослідження. Розкриті основні сфери технологічного застосування використаних технологій, а також виділені основні відмінності розробленої системи.

У третьому розділі наведені дослідні дані щодо розглянутих витратомірів, їх інтерфейс взаємодії с точки зору програмного коду, а також приведені різноманітні діаграми та формули щодо дослідження. Вставки вихідного коду представлені у вигляді окремих функціональних блоків, котрі розкривають окремі сутності та особливості.

ABSTRACT

Certificate of attestation: 57 pages, 17 drawings, 3 applications, 6 sources.

WINDOWS, APPLICATION, DATABASE, SERVER, FRAMEWORK, PROGRAMMING, DIAGRAM.

The object of study is the process of interaction of the flowmeter data based on ultrasonic oscillations in the client application.

The purpose of the work is to design and construct a smart system for processing and output of flowmeter data based on server and client web technologies.

The research method is theoretical: analysis of existing data processing systems of flowmeters and consideration of principles of their operation.

The first section of the performance appraisal discusses the basic principles and features of implementing a flowmeter monitoring system. The differences and principal arguments regarding the theoretical application of technology are investigated. The work of flowmeters and their interface of interaction at the physical level is considered in detail.

In the second section, the information is distributed as a summary description of the constructed object of study. The main areas of technological application of the used technologies are revealed, as well as the main differences of the developed system are highlighted.

The third section presents the research data on the flowmeters considered, their interface of interaction in terms of program code, as well as various diagrams and formulas for the study. Source code inserts are presented as separate functional blocks that reveal individual entities and features.

ЗМІСТ

ВСТУП 7

1 СУТНІСТЬ РЕАЛІЗАЦІЇ ПОБУДОВИ СИСТЕМ МОНІТОРИНГУ ДАНИХ ТА ВИКОРИСТОВУВНІ ТЕХНОЛОГІЇ…….. 8

1.1 Бази даних. Основні особливості та відмінності…….. 8

1.2 Серверні технології реалізації……... 10

1.3 Серверна програмна платформа Node.js. Принцип роботи основних механізмів……... 11

1.4 Використання клієнтських технологій. Сутність проектування та основні інструменти……... 13

2 ПРОЕКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ ТА ВЗАЄМОДІЇ ДАНИХ ВИМІРЮВАЧІВ РІДИНИ, ГАЗУ ТА ЕЛЕКТРОЕНЕРГІЇ…….. 16

2.1 Представлення інтерфейсу взаємодії користувача з системою….. 16

2.2 Особливості побудови витратомірів щодо системи відображення 16

2.3 Реалізація бази даних розроблюваної системи……. 19

2.4 Архітектурні особливості. Представлення системи відносно користувача……. 21

2.5 Серверна реалізація системи та її особливості……. 23

3 ДОСЛІДНІ ВИХІДНІ ДАНІ ТА ВЗАЄМОДІЯ ІНТЕРФЕЙСУ ПЕРЕДАЧІ ДАНИХ ВІДПОВІДНО СТРУКТУРИ ВИТРАТОМІРУ…… 29

3.1 Способи побудови дискретних лінійних динамічних систем……. 29

3.2 Особливості імпульсної характеристики…….. 30

3.3 Сутність використання рядів Фур’є…….. 32

ВИСНОВКИ……..……..…… 36

ПЕРЕЛІК ДЖЕРЕЛ……..……..…… 37

ДОДАТОК А……..……..……... 38

ДОДАТОК Б……..……..…… 39

ДОДАТОК В……..……..……... 48

ВСТУП

В сучасному світі, будь то товар, компанія або якась послуга, часто мають у своєму арсеналі різноманітні інструменти розповсюдження або користування. Системи розумного дому, охоронні, господарські системи автоматизації – кожна із перелічених зазвичай існує в сукупності з необхідним програмним забезпеченням. Майже нікого не здивуєш девайсом чи системою з наявністю web-сторінки, мобільного додатку тощо.

Системи управління чи контролю над різноманітними пристроями до відповідного часу не мали зручного UI інтерфейсу, на якому можливо б було контролювати та отримувати необхідні звіти робочого процесу. Це призводило до того, що точність вимірювань, час, а також якість отримуваних результатів співвідносилися лише з поверхневими даними, які не сприймалися серйозно. Для вирішення проблеми написання додатків перш за все постає питання платформи, котра буде основною серед користувачів. Це в свою чергу дозволяє зрозуміти, які технології слід використовувати.

Найпопулярніші рішення, котрі не мають програшних варіантів – смартфони, - пристрої, які можуть рівнятися по потужності уже з ПК, адже кожний при користуванні мобільним телефоном уже не відчуває відповідних проблем, що дозволяє розглядати цей пристрій, як основний серед використовуваних. Сфера web технологій має дуже високий потенціал як на етапі початку розробки, так і на етапі використання користувачами, так як нікого не бентежить встановлення лише браузеру та наявності мережі, у відмінності з прикладом сторонніх додатків, котрі невідомо як працюють і не викликають довіри.

1 СУТНІСТЬ РЕАЛІЗАЦІЇ ПОБУДОВИ СИСТЕМ МОНІТОРИНГУ ДАНИХ ТА ВИКОРИСТОВУВНІ ТЕХНОЛОГІЇ

1.1 Бази даних. Основні особливості та відмінності

В загальному випадку базою даних можна вважати будь-який впорядкований набір даних. На даний час рішення для роботи з базами даних є одними з найпоширеніших прикладних програм.

В сучасних інформаційних системах для забезпечення та створення роботи з базами даних використовують спеціально створені системи керування.

Система керування базами даних — це комплекс, який має свою основу на програмних засобах та забезпечує контроль, маніпулювання, створення, визначення, керування та використання баз даних (за стандартом ISO/IEC 2382:2015). Найпопулярнішими СКБД є My, Microsoft SQL, PostgreSQL, Server, Oracle, Sybase, Interbase, та IBM DB2. СКБД створюють комфорт при роботі з базами даних, обсяг яких робить неможливим їх ручне опрацювання [1].

Через тісний зв'язок баз даних з СКБД під терміном «база даних» інколи необґрунтовано та неточно мають на увазі систему керування базами даних. Але варто розрізняти базу даних — сховище даних, та СКБД — засоби для роботи з базою даних. СКБД з інформаційної системи може бути видалена, але база даних продовжить існувати. І навпаки: СКБД може функціонувати без жодної бази даних.

В загальному випадку БД неможливо просто перемістити з однієї СКБД до іншої. Але СКБД використовують стандарти (SQL, ODBC, JDBC), що уніфікують та узагальнюють ряд операцій по роботі з даними і дозволяють по різному працювати з базами даних СКБД. Системи керування базами даних часто класифікують за моделлю організації даних. Найпопулярніші СКБД використовують реляційну модель, у якій дані подають у виді таблиць. Для кінцевого користувача (та прикладних програм) робота з базою даних напряму неможлива. Всі маніпуляції над даними здійснюють через спеціальні запити, які

надсилають до СКБД. СКБД опрацьовує їх і повертає результат. Безпосередньо з базою даних працює виключно СКБД.

Сучасні СКБД забезпечують функції щодо керування даними, які можна поділити на такі групи:

- оголошення даних — створення, зміна та видалення визначень, які описують організацію даних;
- модифікація даних — додавання даних, їх редагування та видалення;
- отримання даних — надання даних за запитом застосунку у формі, яка дозволяє їх безпосереднє використання. Дані можуть надаватись або у формі, в якій вони зберігаються у базі даних, або в іншій формі (наприклад, через поєднання різних даних);
- адміністрування даних — реєстрування та відслідковування дій користувачів, дотримання безпеки роботи з даними, забезпечення надійності та цілісності даних, моніторинг продуктивності, резервне копіювання та відновлення даних тощо;

Бази даних класифікують за різними критеріями. За моделлю організації даних розрізняють такі бази даних:

- адміністрування даних — реєстрування та відслідковування дій користувачів, дотримання безпеки роботи з даними, забезпечення надійності та цілісності даних, моніторинг продуктивності, резервне копіювання та відновлення даних тощо;
- ієрархічна. Ієрархічна база даних може бути представлена як дерево, що складається з об'єктів різних рівнів. Між об'єктами існують зв'язки типу «предок-нащадок». При цьому можлива ситуація, коли об'єкт не має нащадків або має їх декілька, тоді як у об'єкта-нащадка обов'язково тільки один предок;
- мережна. Така база даних подібна до ієрархічної, за винятком того, що кожен об'єкт може мати більше одного предка;
- реляційна. Реляційна база даних зберігає дані у вигляді таблиць. Найвживаніші СКБД використовують реляційну модель даних;

− об'єктно-орієнтована. У базі даних цього виду дані оформляють у вигляді моделей об'єктів;

За розміщенням даних виділяють такі види баз:

− локальна, або централізована. Така база даних підтримується на одному комп'ютері;

− розподілена. Частини такої бази даних розміщують на різних комп'ютерах мережі;

Структуровані БД використовують структури даних, тобто структурований опис типу фактів за допомогою схеми даних, більш відомої як модель даних. Модель даних описує об'єкти та взаємовідношення між ними. Існує декілька моделей (чи типів) баз даних, основні: ієрархічна, мережна та реляційна [2].

До неструктурованих БД належать повнотекстові бази даних, які містять неструктуровані тексти статей чи книг у формі, що дозволяє здійснювати швидкий пошук (наприклад, як Вікіпедія).

1.2 Серверні технології реалізації

Різні системи структуризації і відображення інформаційних потоків мають в своїй основі реалізацію серверної та клієнтської частини. Кожна з них має свої особливості в залежності від вимог, але в той же час має автономність один від одного, що дозволяє будувати реалізацію більш гнучким чином. У той же час кількість інструментів, за допомогою яких існує можливість реалізації, перевищує можливі рамки залежностей та проблематики.

Існує напрямок, який відповідає за "тіньову" частину, певний чорний ящик для користувача, в якому відбувається основний процес взаємодії і формування даних, які в свою чергу подаються користувачу у вигляді певних таблиць, діаграм, графіків тощо [3].

Back-end - це все, що працює на сервері, тобто «не в браузері» або «на комп'ютері, підключеному до мережі (зазвичай до Інтернету), який відповідає на повідомлення від інших комп'ютерів». Для back-end частини можливе

використання будь-яких інструментів, що доступні на сервері. Дана особливість дозволяє використовувати будь-яку з перелічених мов програмування: Ruby, PHP, Python, Java, JavaScript / Node, bash. Це також дозволяє використовувати системи управління базами даних, такі як MySQL, PostgreSQL, MongoDB, Cassandra, Redis, Memcached та ін.

Різні способи реалізації диктуються на підставі необхідних рішень, які вимагає очікуваний продукт, або ж його потенціал. Потенційний додаток має у себе на борту реалізацію back-end частини на підставі мови програмування js, а саме node.js [2].

1.3 Серверна програмна платформа Node.js. Принцип роботи основних механізмів

Особливості використання клієнтської мови програмування як серверної ведуть за собою певні нюанси. Виконання JavaScript в Node.js є однопотоковою, тому паралельність відноситься до здатності циклу подій виконувати функції зворотного виклику JavaScript після завершення інших робіт. Будь-який код, який, як очікується, працює одночасно, повинен дозволяти циклу подій продовжувати працювати, оскільки відбуваються операції, що не належать JavaScript, як-от введення-виведення. Як асинхронний сценарій виконання JavaScript, Node.js призначений для створення масштабованих мережевих додатків. Після кожного з'єднання зворотний виклик спрацьовує, але якщо роботи не буде виконано, Node.js перейде у режим "сну" [1].

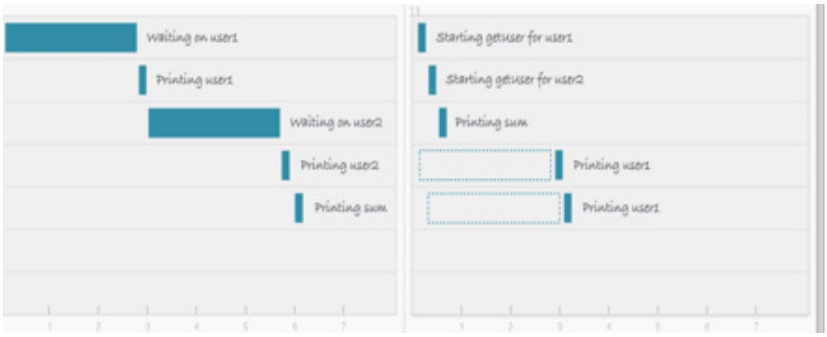

Рисунок 1.1 – Діаграма прикладу роботи node.js

Блокування - це коли виконання додаткового JavaScript у процесі Node.js має очікувати, поки не завершиться операція, що не стосується JavaScript. Це трапляється тому, що цикл подій не в змозі продовжити роботу JavaScript під час операції блокування. У Node.js JavaScript, який демонструє низьку ефективність через інтенсивність процесора, а не очікування операцій, що не належать JavaScript, наприклад, вводу / виводу, не називається блокуванням. Синхронні методи в стандартній бібліотеці Node.js, які використовують libuv, є найбільш часто використовуваними операціями блокування. Вбудовані модулі також можуть мати методи блокування [2].

Усі методи вводу-виводу в стандартній бібліотеці Node.js забезпечують асинхронні версії, які не блокують, і приймають функції зворотного виклику. Деякі методи також мають аналоги блокування, які мають імена, що закінчуються синхронізацією.

Мережа на основі “ниток” порівняно неефективна і дуже складна у використанні. Крім того, користувачі Node.js звільняються від турботи про мертве блокування процесу, оскільки блокувань немає. Майже жодна функція в Node.js безпосередньо не виконує введення-виведення, тому процес ніколи не блокується. Оскільки нічого не блокує, масштабовані системи дуже розумно розробляти в Node.js.

Node.js схожий за конструкцією та під впливом таких систем, як Ruby's Event Machine і Python's Twisted. Node.js трохи більше забирає модель подій. Він представляє цикл подій як конструкцію виконання, а не як бібліотеку. В інших системах завжди є виклик блокування для запуску циклу подій. Як правило, поведінка визначається через зворотні виклики на початку скрипту, а наприкінці сервер запускається через блокуючий виклик, наприклад EventMachine :: run (). У Node.js немає такого виклику циклу запуску події. Node.js просто входить у цикл подій після виконання сценарію введення. Node.js виходить з циклу подій, коли більше немає зворотних викликів для виконання. Така поведінка нагадує JavaScript браузера - цикл подій прихований від користувача [3].

HTTP - це першокласний представник у Node.js, розроблений з урахуванням потокової та низької затримки. Це робить Node.js добре підходящим для створення веб-бібліотеки чи фреймворку.

То, що Node.js розроблений без потоків не означає, що не має можливості скористатися декількома ядрами в оточенні. Дочірні процеси можна породжувати за допомогою API child_process.fork(), і вони розроблені так, щоб їх було легко з'єднати. На цьому ж інтерфейсі побудований кластерний модуль, який дозволяє обмінюватися сокетами між процесами, щоб забезпечити балансування навантаження над ядрами.

Вибір Node.js в якості серверної мови програмування був визначений на підставі універсальності мови JavaScript, а саме можливості використання її як на стороні клієнта, так і на стороні серверу. Так само популярність даної серверної технології має величезні показники за останній час в глобальному сенсі, що тільки підтверджує рішення використовувати саме node.js [2].

1.4 Використання клієнтських технологій. Сутність проектування та основні інструменти

Клієнтська частина має на увазі факт процесів, які відбуваються безпосередньо на стороні користувача. Іншими словами обробка всіляких даних, реалізація інтерактиву в інтерфейсі, диктуються на локальній «машині» клієнта.

Найчастіше побудова клієнтської частини програми відбувається на підставі необхідних рішень і потенційних можливостей підсумкового продукту. Одним з найпопулярніших реалізацій на даний момент є мова програмування - JavaScript.

JavaScript - клієнтська мова програмування, компіляція вихідного коду якого відбувається на рівні двигуна браузера. З 2009 року, як уже було сказано раніше, його використання вдалося розширити і на рівні створення десктопних додатків за допомогою вилучення браузерного движка V8, на якому працює Google Chrome.

Можливості мови програмування в плані побудови веб інтерфейсів мають величезний пріоритет в діапазоні всіляких рішень - інструментарію, що

представляється фреймворками. Подібні платформи дозволяють розробляти програми високої складності на досить комфортних умовах. Найчастіше на борту таких допоміжних інструментів реалізовано величезна кількість всіляких методів, які працюють з численними модулями, що дозволяє в свою чергу домагатися дуже успішних результатів в області оптимізації [2].

Апріорі, починаючи працювати з певною мовою програмування, кожен розробник програмного забезпечення починає стикатися з питаннями установки оточення розробки і, звичайно ж, вибірки певних готових рішень - всіляких бібліотек, які дозволяють істотно економити час і виробляти за замовчуванням вищу оптимізацію коду за допомогою підтримки розробниками сторонніх бібліотек.

На етапі вирішення вибору інструментарію розробки внаслідок певної величини складності остаточне рішення може схилятися до схильного фреймворку як з боку back-end, так і з боку front-end.

Одним з основних і найбільш популярних подібних рішень з боку клієнтської частини є фреймворк Angular внаслідок його масштабності і величезного функціоналу «під капотом».

src/app/product-list/product-list.component.html

```
<h2>Products</h2>

<div *ngFor="let product of products">

  <h3>
    <a [title]="product.name + ' details'">
      {{ product.name }}
    </a>
  </h3>

  <p *ngIf="product.description">
    Description: {{ product.description }}
  </p>

</div>
```

Рисунок 2.2 – Приклад коду на Angular

Angular (зазвичай так називають фреймворк Angular 2 або Angular 2+, тобто вищі версії) — написаний на TypeScript front-end фреймворк з відкритим кодом,

який розробляється під керівництвом Angular Team у компанії Google, а також спільнотою приватних розробників та корпорацій. Angular — це AngularJS, який переосмислили та який був повністю переписаний тією ж командою розробників.

Angular надає таку функціональність, як двостороннє зв'язування, що дозволяє динамічно змінювати дані в одному місці інтерфейсу при зміні даних моделі в іншому, шаблони, маршрутизація і так далі.

Однією з ключових особливостей Angular є те, що він використовує в якості мови програмування TypeScript. Тому перед початком роботи рекомендується ознайомитися з основами цієї мови [3].

2 ПРОЕКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ ТА ВЗАЄМОДІЇ ДАНИХ ВИМІРЮВАЧІВ РІДИНИ, ГАЗУ ТА ЕЛЕКТРОЕНЕРГІЇ

2.1 Представлення інтерфейсу взаємодії користувача з системою

На сьогодні кожна система контролю має в своєму арсеналі оптимізований ui інтерфейс, який в свою чергу дозволяє зручно моніторити дані і своєчасно відстежувати різні зміни в роботі. Подібна практика написання софта має відмінні підсумкові результати, так як з ростом технологічного прогресу виконувати адміністраторські функції стає все складніше і складніше, а багато процесів стає тільки в спад часового ресурсу щодо людини. Виходячи з цього, процеси, що відбуваються в апаратній частині, такі як інформаційні сигнали і їх обробка, зберігають свої значення в підсумковому варіанті в БД.

Різні взаємодії електромагнітних та ультразвукових сигналів в кінцевому підсумку перетворюються в цифрові сигнали, які генерують підсумкове значення, зрозуміле для подальшої взаємодії в межах бази даних і її взаємодії з back-end частиною. Таке рішення дозволяє гнучко розробляти цілі комплекси, які будуть в зручному вигляді надавати загальні інформаційні дані, що в свою чергу позбавляє проблеми в людських ресурсах по типу додаткового освоєння певних умінь та знань [4].

2.2 Особливості побудови витратомірів щодо системи відображення

Різноманітні витратоміри мають в своєму складі різноманітні особливості щодо реалізації та кінцевого виводу даних. В якості реалізації відображення вихідних даних витратомір перш за все взаємодіє з системою, котра контролює фізичні дії щодо моніторингу пропускних об’ємів даних, таких як рідина, газ тощо. Завдяки цьому незалежно від типу дії витратоміру, будь то манометричний

витратомір, або витратомір на основі гідравлічних опорів, вихідні дані повинні відображатись в комфортному вигляді відносно користувача.

Ультразвуковий витратомір, що використовується в даному проекті, має в своєму складі відрізок труби, на якому встановлені пьезоелементи. Діаметр пьезоелемента знаходиться в межах 5-20 міліметрів, а його товщина вибирається залежно від частоти. У частотних і час-імпульсних витратомірах для повішення точності вимірювань використовують частоти 5-20 Мгц [4].

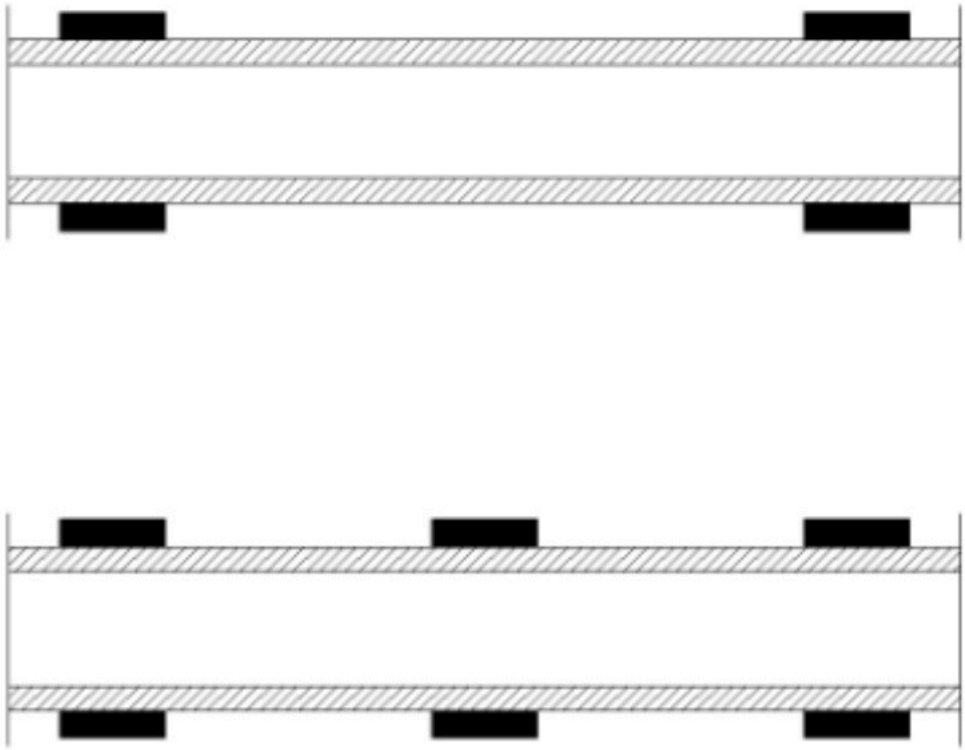

Рисунок 2.1 – Розміщення сенсорів витратоміра

Подібна конструкція сенсорів, а також форма та діаметр забезпечать якісне стеження за об’ємами об’єкта, для якого призначений витратомір. Завдяки цьому дані будуть формуватися в коректній формі в цифровий сигнал, котрий після перетворення в двійковий код без дефектів в послідовній обробці буде формувати дані в базі даних.

Система відображення в свою чергу виконує функції коректного відбору даних та їх відображення в зручному вигляді. По принципу різноманітних індикаторів, котрі використовувались раніше, на кожній сторінці додатку система

в реактивному форматі відстежує кожні зміни щодо даних та виконує відображення нових відносно обновлення. В свою чергу це дозволяє повністю контролювати стан всієї системи та зручно стежити за кожними змінами, навіть будучи в поїздці з єдиним смартфоном та доступом до мережі. Наявність доцільності розробленого додатку відносно витратомірів пояснюється тим, що кожна система повинна мати зручні умови для використання, а також максимальну мобільність в плані можливостей. В даному випадку витратоміри надають повну інформацію щодо споживання тих чи інших ресурсів [5].

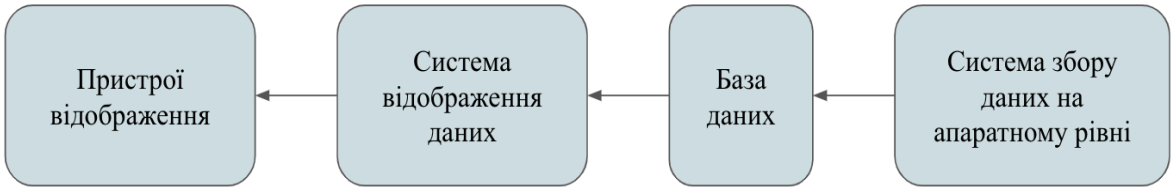

Рисунок 2.2 – Схема доцільності застосування системи відображення

Сутність розкривається також в діючих можливостях системи в цілому. На зараз навіть в спеціалізованих лабораторіях використовуються по замовчуванню різноманітні стаціонарні комп’ютери з відповідною операційною системою, тоді як з досвіду відомо, що раніше практика зазвичай мала в своєму складі системні блоки надвеликого розміру для зберігання та обчислення інформації. Це в свою чергу дозволяє будувати архітектурні рішення без обмеження в певній потужності, а також використовуваних платформах.

На разі представлені витратоміри створюють систему автоматизованою взаємодії, котра в сукупності з інтерфейсом користувача в виді системи відображення створює комплексне рішення для використання. Без складової виводу даних апаратна частина комплексу не має сенсу, адже взаємодія остаточна взаємодія відбувається на рівні користувача, що в кінцевій дії виконує різноманітні маніпулювання та стеження за даними [6].

2.3 Реалізація бази даних розроблюваної системи

Представлена база даних в проекті має в своїй архітектурі досить просту конфігурацію, яку можливо легко очистити без проблеми зв'язування даних між колекціями.

Перш за все варто відзначити момент того, що колекціями є згруповані суті у вигляді об'єктів, в якому описана однотипна інформація для кожного з об'єктів з унікальними значеннями.

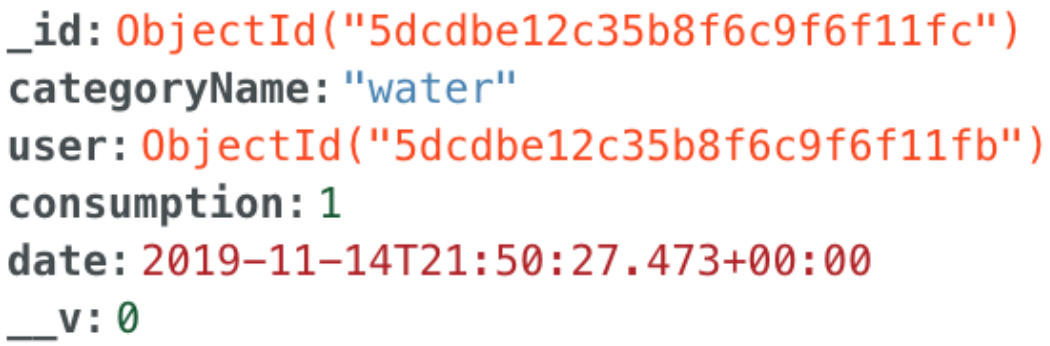

Рисунок 2.1 – Представлення об'єкта колекції в БД

У проектованої системі кількість таких колекцій прирівнюється до двох, які в свою чергу називаються users і positions. Логіка заповнення бази даних така, що в колекції users зберігаються всі зареєстровані користувачі, інформація яких складається всього лише з email і зашифрованного паролю. Подібна реалізація зроблена для того, щоб мінімізувати ризики проникнення в систему зловмисниками. Друга колекція представляє собою набір позицій і кожен об'єкт несе в собі інформацію про деяких звітностях системи, прив'язуючи відповідні дані до певного користувача. Таким чином, логи роботи системи для кожного користувача будуть заповнюватися в єдиному місці, а в той же час угруповання на категорії за типом вода, газ, електрика буде відбуватися на стороні клієнта [2].

Рисунок 2.2 – Подання об'єктів колекції positions

Представлені дані мають підсумкові значення на підставі попередньої фільтрації з боку програмного забезпечення апаратної частини.

Подібний підхід має ряд переваг і деякі недоліки. Слід відзначити той факт, що подібне зберігання буде вкрай складно простежити з точки зору конкретного

користувача, що в свою чергу безпосередньо створює захист від зловмисників. Також подібна архітектура має пріоритет модифікації в майбутньому [3].

2.4 Архітектурні особливості. Представлення системи відносно користувача

Раніше згадана система моніторингу та управління даними комунальних послуг має в своєму базовому прототипі реалізацію моніторингу витрати і споживання в залежності від категорії - вода, газ, електрика. Вся інформація представлена в зручному вигляді з приємним web-інтерфейсом, з яким в свою чергу можливо успішно працювати і на мобільних пристроях, що дозволяє полегшити роботу поза межами дому, офісу і т.п.

Також слід зазначити, що архітектура додатка побудована таким чином, щоб можливість розширення і впровадження нової функціональності була мінімальна в плані трудовитрат розробника. Деякі з потенційних розширень будуть проводитися автоматично при відповідній конфігурації з боку апаратної частини.

Важливим фактором, який слід відзначити при розробці даної системи, є тип програми - Single Page Application (SPA). У більш простому розумінні web розробки подібний термін трактується як додаток, яке має всього лише один файл розмітки. Фреймворк Angular в даному випадку забезпечує максимально комфортну розробку front-end частини, дозволяючи взаємодіяти з додатком без перезавантаження сторінки з максимальною швидкістю роботи [2].

У свою чергу, для кожного користувача буде в зручному вигляді реалізована система авторизації, яка вимагає лише базові дані у вигляді призначеного для користувача email і пароля, що в свою чергу якісно валідується в режимі real time без перезавантаження сторінки. Однофакторна авторизація може здатися на перший погляд суттєвої вразливістю в масштабі всієї системи, проте ж процедура шифрування, система сесій, а також зберігання даних в модифікованому вигляді на рівні бази даних забезпечує максимальний захист від зловмисників. Однак ж реалізація розширення не має ніякої проблеми в потенційному розвитку проекту,

так як архітектура максимально заточена під зростання базової конфігурації і модернізації прототипу.

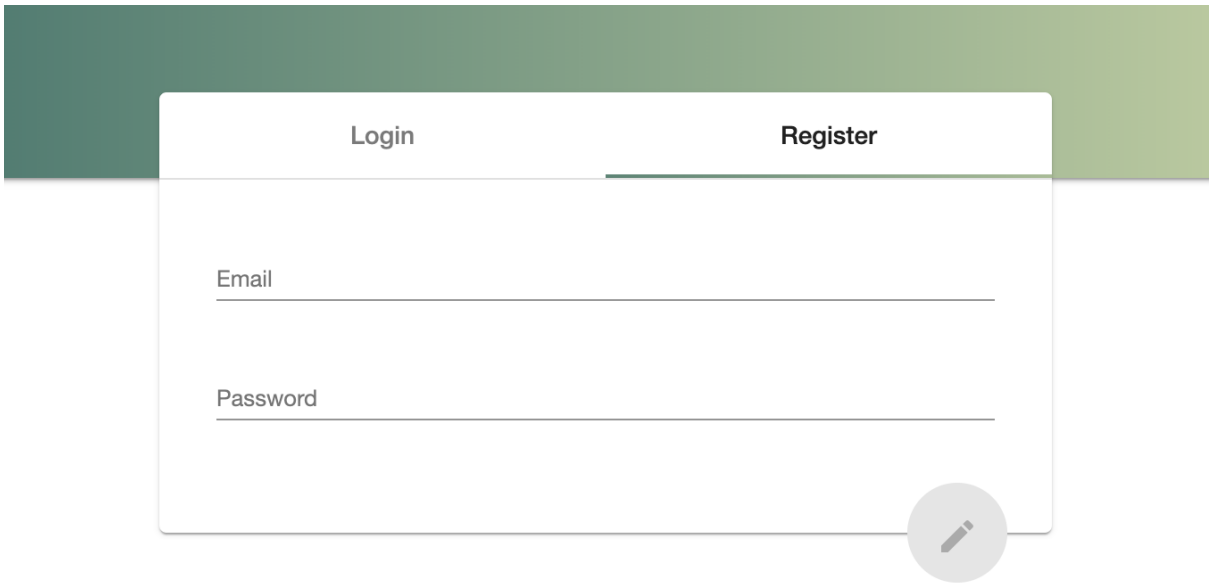

Рисунок 2.3 - WEB-інтерфейс авторизації системи

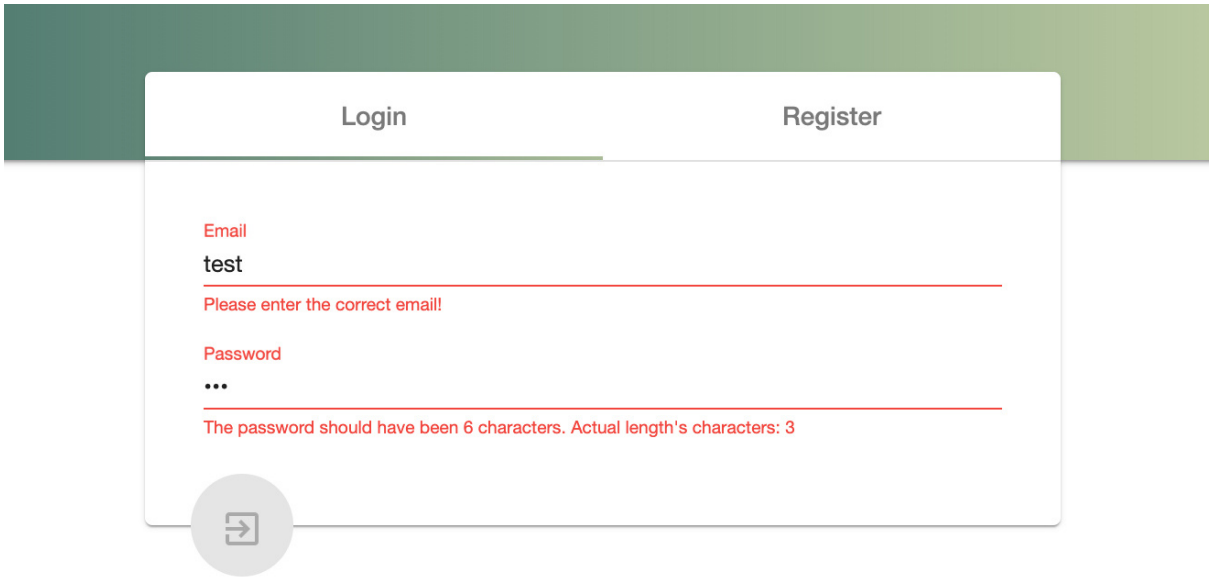

Рисунок 2.4 - WEB-інтерфейс прикладу валідації даних при вході в систему

У свою чергу, з точки зору архітектури коду front-end частини, реалізація винесена в окремий модуль і може бути повністю автономна. Компоненти створені

за стандартами розробки в оточенні фреймворка Angular і дотримуються всіх основ неймінга класів, методів, властивостей, змінних і т.д. В межах прототипу системи реалізації e2e тестування не була проведена, так як вимагає більш глибокого впровадження в плані часових ресурсів і не представляє доцільності на етапі старту демо версії. Подібна практика може призвести до нестабільності роботи системи, але в перспективі розвитку кожна сутність має всі можливості і доступи для покриття тестами для забезпечення непередбачених збоїв системи [2].

```
<form class="default-form inner-offset-2r" action="" method="post" [formGroup]="form" (ngSubmit)="onSubmit()">
  <mat-form-field class="full-width form-field">
    <input matInput type="email" placeholder="Email" formControlName="email" autocomplete="current-email">
    <mat-error *ngIf="form.get('email').touched && form.get('email').invalid">
      <span *ngIf="form.get('email').errors['required']">Email field should not be empty!</span>
      <span *ngIf="form.get('email').errors['email']">Please enter the correct email!</span>
    </mat-error>
  </mat-form-field>
  <mat-form-field class="full-width form-field">
    <input matInput type="password" placeholder="Password" formControlName="password" autocomplete="current-password">
    <mat-error *ngIf="form.get('password').touched && form.get('password').invalid">
      <span *ngIf="form.get('password').errors['required']">Password field should not be empty!</span>
      <span *ngIf="form.get('password').errors.minlength && form.get('password').errors.minlength.requiredLength">
        The password should have been
        {{ form.get('password').errors.minlength.requiredLength }}
        characters. Actual length's characters:
        {{ form.get('password').errors.minlength.actualLength }}
      </span>
    </mat-error>
  </mat-form-field>
  <div class="form-action form-action-login">
    <button class="btn btn-auth" mat-button mat-fab matTooltip="Login to the system" type="submit" [disabled]="form.invalid
  </div>
</form>
```

Рисунок 2.5 - Приклад побудови компонента логіна на рівні архітектури коду

2.5 Серверна реалізація системи та її особливості

Відносно роботи авторизації на рівні back-end слід зазначити, що кожен запит до бази даних, буде то створення користувача або логін в систему, супроводжується чітким поділом логіки на рівні контролерів, які в свою чергу зручним чином декомпозовані і винесені, як окремі методи. Таким чином, при вирішенні потенційної модернізації та вдосконалення функціоналу на стороні серверної частини не призведе до довгострокової перспективи даного процесу, так як зручність використання і розподілу смислового навантаження в плані коду дозволяє впровадити будь-яку сутність без дефектів на будь-якому іншому рівні.

```
1   const mongoose = require('mongoose');
2
3   // vendor packages
4   const bcrypt = require('bcryptjs');
5   const jwt = require('jsonwebtoken');
6
7   // config
8   const keys = require('../config/keys');
9
10  // utils
11  const errorHandler = require('../utils/errorHandler');
12
13  // models
14  const User = require('../models/user');
15  const Category = require('../models/category');
16  const Position = require('../models/position');
17
18  module.exports.login = async (req, res) => {...};
52
53  module.exports.register = async (req, res) => {...};
100
```

Рисунок 2.6 - Структура реалізації контролерів блоку авторизації

Система роутів, яка впроваджена в додаток, має в своєму арсеналі відмінні риси реалізації в першу чергу щодо розуміння. На перший погляд аргументація може здатися необгрунтованою, але часто час, витрачений на розбір коду, має чималі обсяги, що призводить в будь-якому випадку до втрати оптимальності оплати годин на розробку в бізнес процесі. Концепція роутів розглядається як очевидна система поділу бізнес-логіки на окремі функціональні блоки. Вони ж в свою чергу за певним uri здійснюють запит за правилами REST API, після чого за допомогою контролерів відбуваються певні маніпуляції, дії і т.д [3].

```
const router = require('express').Router();

// middleware
const upload = require('../middleware/upload');

// guard for the route
const passport = require('passport');

// controller of the auth block
const controller = require('../controllers/category');

router.get('/', passport.authenticate( strategy: 'jwt',  options: {session: false}), controller.getAll);
router.get('/:id', passport.authenticate( strategy: 'jwt',  options: {session: false}), controller.getById);
router.delete('/:id', passport.authenticate( strategy: 'jwt',  options: {session: false}), controller.remove);
router.post('/', passport.authenticate( strategy: 'jwt',  options: {session: false}), upload.single( name: 'image'), controller.create);
router.patch('/:id', passport.authenticate( strategy: 'jwt',  options: {session: false}), upload.single( name: 'image'), controller.update);

module.exports = router;
```

Рисунок 2.7 - Приклад реалізації роутів

Поняття REST при побудові і розробці програми взяло фундаментальну роль. REST API - Representational State Transfer (передача стан-подання). Сервер, який використовується при розробці, може бути написаний будь-якою мовою програмування. Якщо ж звичайний веб-сервер відправляє в якості відповіді html сторінку, то його приналежність може бути використана тільки для певних клієнтів, в даному випадку браузера, який за замовчуванням розуміє подібний формат. Суть полягає в тому, що REST сервер відправляє в якості відповіді тільки дані і як правило вони представлені в універсальному форматі json (цей формат є необов'язковим і може використовуватися будь-який інший). Від сервера в такому випадку клієнт отримує лише стан, необмежену зайвим форматом і т.д.

Самим позитивним моментом при такому підході є істотне зменшення споживчого трафіку. Іншими словами, замість величезного обсягу специфічних даних за типом розмітки, стилів, скриптів, відправляються лише ті дані, які можливо використовувати як в браузері, так і наприклад на мобільних платформах. Таким чином варіант розширити програми під потреби окремого мобільного додатка не складе особливих труднощів, так як всі ці дані будуть без проблем парситись на будь-який клієнтській стороні, яка вже в залежності від цього буде будувати необхідний інтерфейс.

При успішній авторизації користувач отримує сторінку з інформацією у вигляді діаграми про те, які на поточний момент існують категорії і яке загальне споживання по кожній з них. Зручний інтерфейс і сучасний дизайн дозволяють легко розібратися з загальним уявленням про ситуацію вимірювальних систем, тим

самим максимально швидко визначити звітність по кожній з них. У разі відсутності будь-яких даних в БД, або отримання певної помилки сервера, користувачеві буде відображатися відповідна інформація.

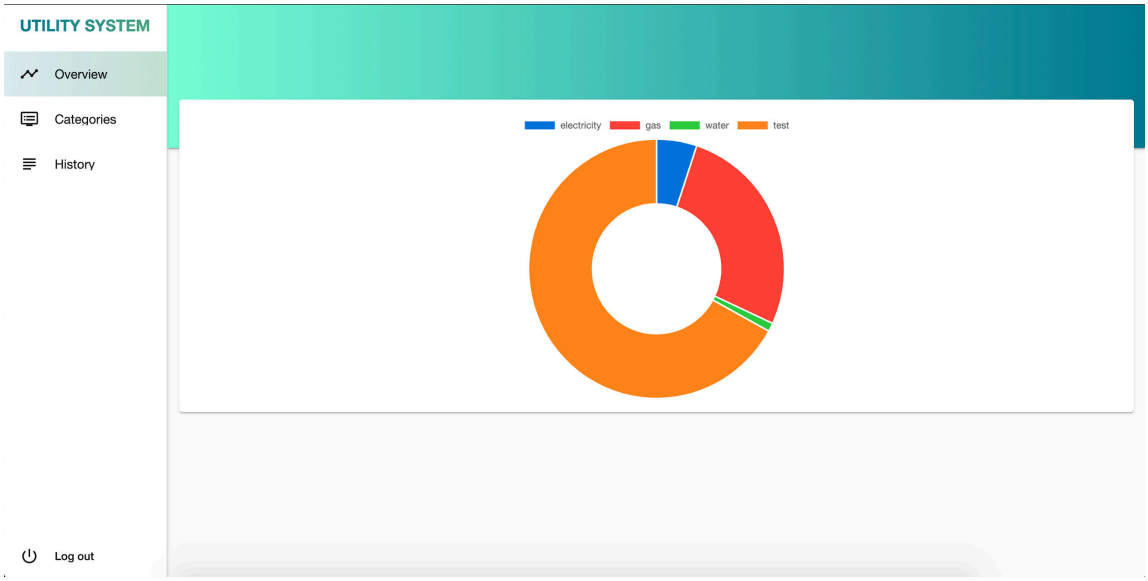

Рисунок 2.8 - Огляд сторінки overview додатка

Головна сторінка додатка має в своїй основі найостаннішу оновлену інформацію про споживання в залежності від категорії. Кожна категорія ініціалізується миттєво при проходженні системи авторизації. У той же час при відсутності будь-яких початкових даних, які заповнюються апаратною частиною в БД, користувач отримує відповідну інформацію.

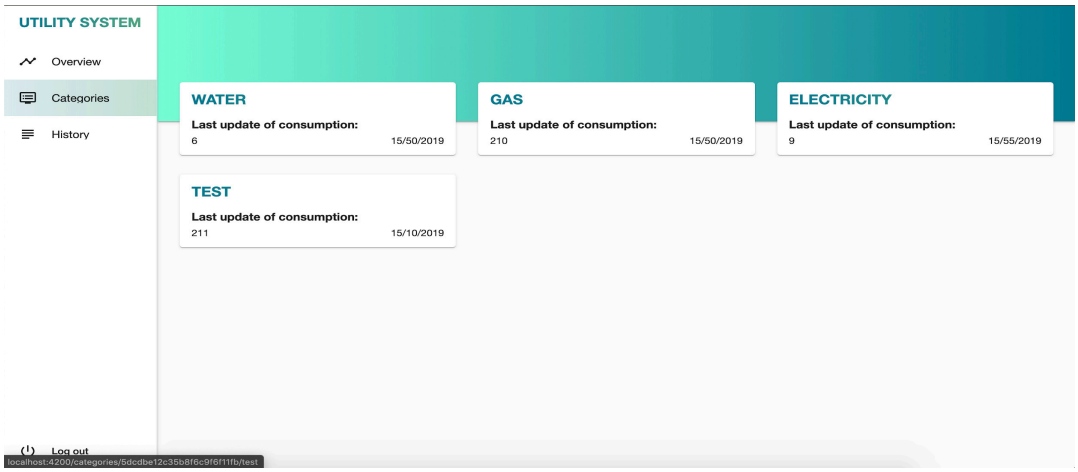

Рисунок 2.9 - Головна сторінка додатку

При створенні нового облікового запису і прив'язці апаратної частини до системи в додатку будуть відповідні інформаційні блоки про те, що на даний момент немає ніякої інформації про роботу системи і при найближчій звітності в базу даних інформація з'явиться в системі. У первісному підході для реалізації парсинга інформації з БД було прийнято рішення вручну оновлювати інформаційну сторінку, так як можливість впровадження технології на підставі сокетів, які без особливої втрати продуктивності в режимі реального часу змогли б оновлювати інформацію в залежності від подій, не має можливості.

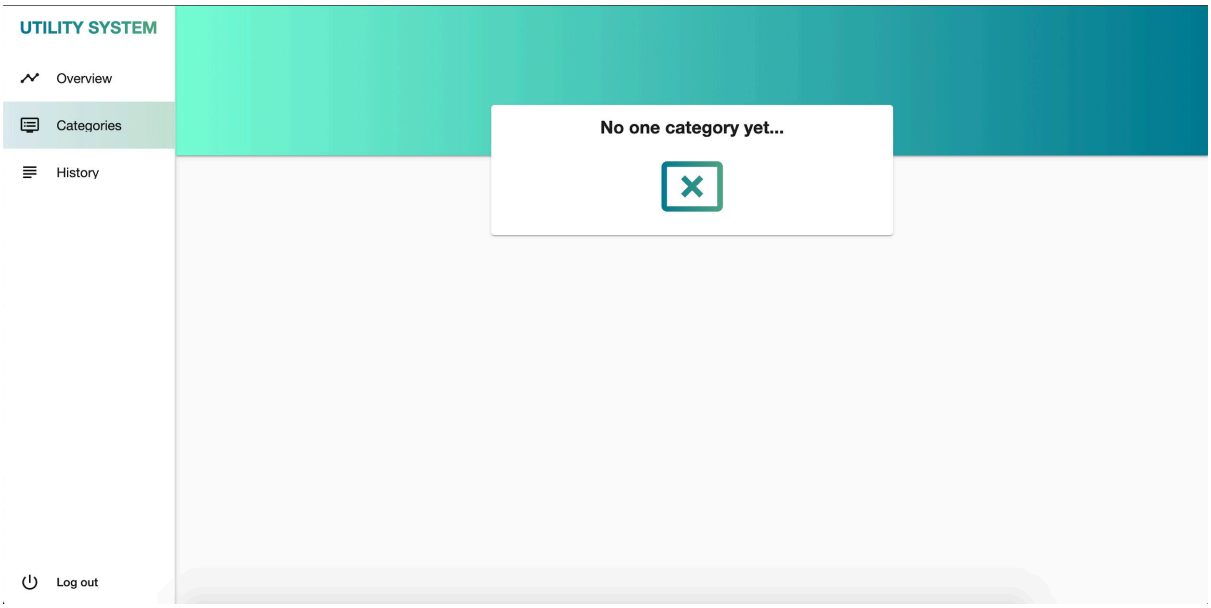

Рисунок 2.10 - Представлення сторінки без даних

Кожна категорія має в своїй суті певний набір позицій, які формується на підставі даних з БД. При переході на окрему категорію користувач матиме можливість переглянути графік споживання за часом залежно від значення, тим самим отримавши можливість в зручному і сучасному вигляді переглядати звітності.

Також буде виведена історія позицій - список всіх оновлень з бази даних щодо споживання та часу, коли це сталося. Тим самим користувач також наочно зможе спостерігати за значеннями споживання кожної категорії у вигляді зручного відсортованого списку.

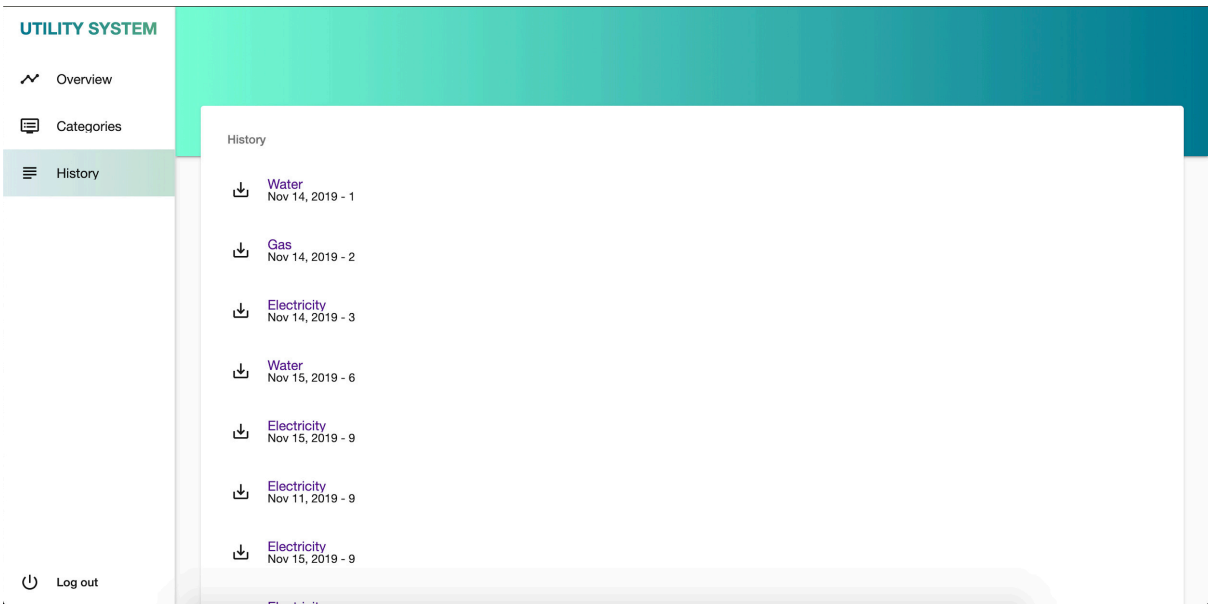

Рисунок 2.11 - Представлення сторінки історії звітів додатка

Кнопка оплати, яка буде виводитися на сторінці кожної з категорії, буде нести функціональність оплати комунальних послуг за різними методами оплати в залежності від зручності для кожного. Дана кнопка буде представлена як потенційно впроваджувана, так як процедура імплементації системи оплати в залежності від споживання вимагає чималої кількості часу.

3 ДОСЛІДНІ ВИХІДНІ ДАНІ ТА ВЗАЄМОДІЯ ІНТЕРФЕЙСУ ПЕРЕДАЧІ ДАНИХ ВІДПОВІДНО СТРУКТУРИ ВИТРАТОМІРУ

3.1 Способи побудови дискретних лінійних динамічних систем

Етап зчитування інформації в першу чергу забезпечується початковими етапами проходження і фільтрації сигналів щодо тих чи інших категорій приборів. В даному прототипі існує три основних джерела інформації - вода, газ і електрика. У процесі споживання розробляється система на рівні апаратної частини, що вимірює значення споживання за принципом дискретних лінійних динамічних систем.

Існують два основних способи побудови дискретних лінійних динамічних систем. У літературі такі системи прийнято називати цифровими фільтрами, які поділяються на два основних типи: фільтри з кінцевою імпульсною характеристикою (КІХ) і фільтри з нескінченною імпульсною характеристикою (НІХ) [4].

Алгоритмічна сутність фільтра з КІХ полягає в дискретному обчисленні інтеграла згортки:

$$y(t) = \int_{o}^{t} x(\tau) \cdot h(t - \tau) dt \qquad (3.1)$$

де x () - вхідний сигнал;

y (t) - вихідний сигнал;

h (t) - імпульсна характеристика фільтра або реакція фільтра на дельта функцію.

3.2 Особливості імпульсної характеристики

Імпульсна характеристика є зворотним перетворенням Фур'є комплексної частотної характеристики фільтра K(f). Для формування більш ясної картини наведено приклад дискретного обчислення інтеграла згортки на мові програмування С в реальному часі, який включений в загальний білд даної системи.

```
#define L (4) //довжина фільтру
int FIR(int a)
{
static int i=0; //поточна позиція
static int reg[L]; //масив вхідних значень
static const int h[L]={1,1,1,1};//імпульсна характеристика
int b=0;//вихідне значення
reg[i]=a; //копіювання вхідного значення в масив вхідних значень
for(int j=0;j<L;j++)//згортка
  {
  b=b+h[j]*reg[i];
  i=(i-1)&(L-1);
  }
i=(i+1)&(L-1);//інкрементування указателю на наступну позицію
return b;
}
```

Викликаючи цю функцію через певні інтервали часу T і передаючи їй як аргумент вхідний сигнал, на виході виходить вихідний сигнал, відповідний реакції фільтра з імпульсною характеристикою виду:

$$h(t)=1 \text{ при } 0<t<4T,$$
$$h(t)=0 \text{ в інших випадках.} \qquad (3.2)$$

Фільтр з такою імпульсною характеристикою більш відомий під назвою «фільтр змінного середнього», і, відповідно, реалізується він набагато простіше.

Використовуючи фільтр з кінцевою імпульсною характеристикою, слід враховувати момент того, що подібна імітація є вигаданою, так як в природі не існує динамічних систем, що мають кінцеву імпульсну характеристику. Процес фільтрації з КІХ можна представити у вигляді спроби ісказити частотно-часову

область, тому частотні характеристики цих фільтрів найчастіше забезпечуються несподіваними властивостями [3].

Алгоритмічна сутність фільтрів з нескінченною імпульсною характеристикою зводиться до рекурентного рішення диференціального рівняння, що описує фільтр. Тобто, кожне наступне значення вихідного сигналу фільтра обчислюється на підставі попереднього значення. Визначення поняття «порядок фільтра» є кількість змінних, які піддаються рекурентним операціями.

```
#define alfa (2) //параметр сглажування
int filter(int a)
  {
  static int out_alfa=0;
  out_alfa=out_alfa - (out_alfa >>alfa) + a;
  return (out_alfa >> alfa);
  }
```

Викликаючи цю функцію з частотою F і передаючи їй як аргумент вхідний сигнал, на виході виходить вихідний сигнал, відповідний реакції фільтра низьких частот першого порядку з частотою зрізу:

$$F_{077} = \frac{acos\left[\frac{1}{2}\cdot\frac{\sqrt{2^{\alpha}\cdot(2^{\alpha}-1)\cdot\left[-1+4\cdot(2^{\alpha})^{2}-4\cdot 2^{\alpha}\right]}}{\left[(2^{\alpha})\cdot(2^{\alpha}-1)\right]}\right]}{\pi}\cdot F \qquad (3.3)$$

Рекурентний вираз фільтра з прикладу коду побудовано таким чином, щоб уникнути втрати значущих розрядів. Саме кінцева точність обчислень може зіпсувати всю красу цифрового фільтра з нескінченною імпульсною характеристикою. Особливо це помітно на фільтрах високих порядків, що відрізняються високою добротністю. У реальних динамічних системах така проблема не виникає [6].

3.3 Сутність використання рядів Фур’є

Слід зазначити, що існує два основних підходу до аналізу лінійних динамічних систем: аналіз у часовій області та аналіз в частотній області. Друге більше відноситься до диференціальних рівнянь, интегралів згортки та Дюамеля. Ці методи аналізу дискретно втілилися в цифрових фільтрах НІХ і КІХ.

Але також існує частотний підхід до аналізу лінійних динамічних систем, який іноді називають операційним. В якості операторів використовуються перетворення Фур'є, Лапласа і т.п. Даний метод аналізу не отримав широкого поширення при побудові цифрових фільтрів.

Реакція лінійної динамічної системи є зворотне перетворення Фур'є від похідної зображення по Фур'є вхідного сигналу x(t) на комплексний коефіцієнт передачі K(f):

$$y(t) = \int_{-\infty}^{\infty} \left[\left(\int_{-\infty}^{\infty} x(t) \cdot e^{-i \cdot 2 \cdot \pi \cdot \varphi \cdot \tau} dt \right) \cdot K(f) \right] \cdot e^{i \cdot 2 \cdot \pi \cdot \varphi \cdot \tau} df \qquad (3.4)$$

У практичному плані даний аналітичний вираз передбачає наступний порядок дій: існує перетворення Фур'є від вхідного сигналу, яке множиться на результат і на комплексний коефіцієнт передачі, після чого виконується зворотне перетворення Фур'є, результатом якого є вихідний сигнал. У реальному дискретному часі такий порядок дій виконати неможливо, так як інтеграл за часом від мінус до плюс нескінченності є не більше ніж абстракцією.

У дискретному оточенні для виконання перетворення Фур'є існує інструмент - певний алгоритм швидкого перетворення Фур'є (ШПФ). Саме він і став рухомим механізмом при реалізації Фур'є-фільтра. Аргументом функції БПФ є масив тимчасових відліків з 2^n елементів, результатом - два масиви довжиною 2^n елементів, що відповідні дійсній та уявній частині перетворення Фур'є. Дискретною особливістю алгоритму БІФ є те, що вхідний сигнал вважається періодичним з інтервалом 2^n. Це накладає певні обмеження на алгоритм Фур'є-

фільтра, так як якщо взяти послідовність вибірок вхідного сигналу, провести від них БІФ, помножити результат БІФ на комплексний коефіцієнт передачі фільтра і виконати зворотне перетворення, - результату як такого не буде. Вихідний сигнал буде мати величезні нелінійні спотворення в околиці стиків вибірок [5].

При дослідженні вихідних проблем були виявлені два концептуальних рішення, які в кінцевому підсумку реалізувалися на рівні написання кодової частини обробки:

– вибірки необхідно обробляти перетворенням Фур'є з перекриттям. Іншими словами кожна наступна вибірка повинна містити частину попередньої. В ідеальному випадку вибірки повинні перекриватися на (2^n-1) відліків, але це вимагає величезних обчислювальних витрат. На практиці досить тричетвертними (2^n-$2^{(n-2)}$), половинного ($2^{(n-1)}$) і навіть четвертного перекриття ($2^{(n-2)}$);

– результати зворотного перетворення Фур'є для отримання вихідного сигналу необхідно перед накладенням один на одного помножити на вагову функцію (масив вагових коефіцієнтів). Вагова функція в свою чергу повинна відповідати таким вимогам: Дорівнює нулю всюди, крім інтервалу 2^n, На краях інтервал прагне до нуля, Сума вагових функцій Fv(t), зсунутих на інтервал перекриття k, повинна бути постійна;

$$\int_{e=-\infty}^{\infty} Fv(t - k \cdot n) = const \tag{3.5}$$

Такі функції широко застосовуються в техніці цифрової обробки сигналів і часто називаються "вікнами".

За результатами досліджень, найкращим з практичної точки зору є вікно імені Хана:

$$Han(t) = 1 - cos(\frac{2 \cdot \pi \cdot \tau}{2^n}) \tag{3.6}$$

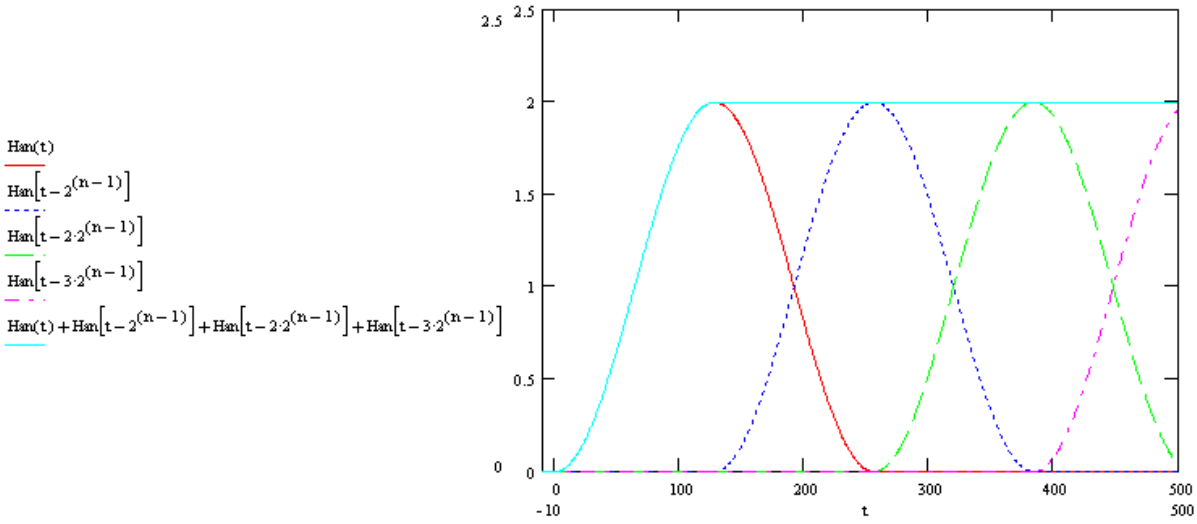

Рисунок 3.1 - Властивості вікна Хана довжини 2^n= 256

На наступному малюнку наведена схема обчислень Фур'є-фільтра при довжині вибірки 2^n= 8. На подібних малюнках дуже складно відобразити процес обчислень, особливо важко показати його циклічність, тому представлене уявлення обмежене кількістю вибірок рівним трьом [4].

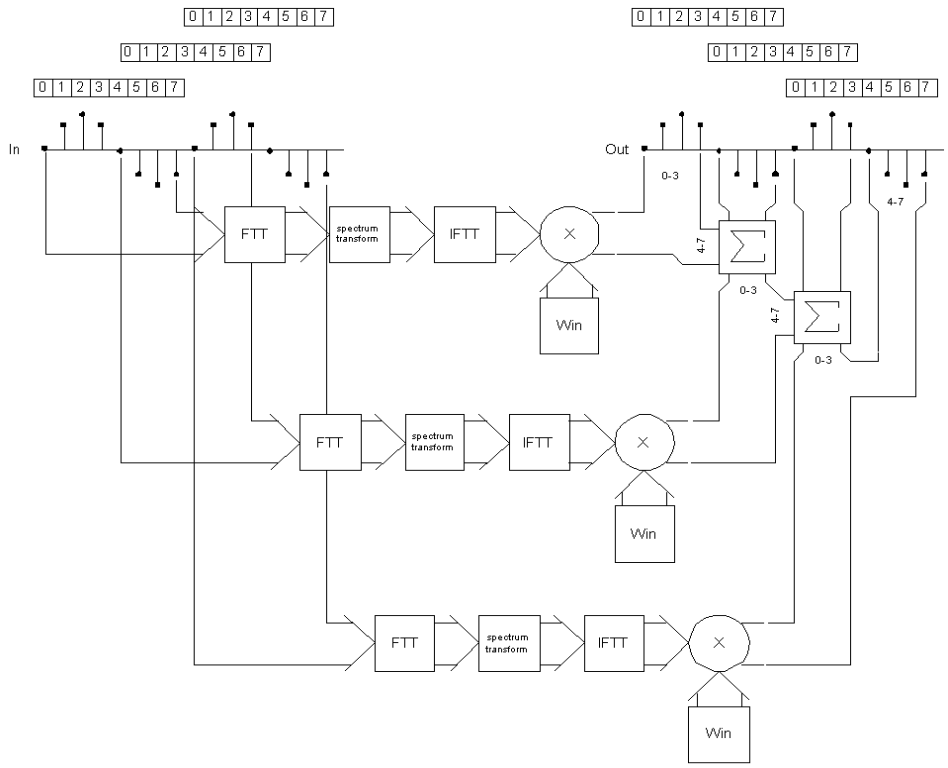

Рисунок 3.2 - Схема обчислень Фур'є-фільтра

Вхідний сигнал розбивається на блоки довжиною $2^n = 8$ з перекриттям 50%. Від кожного блоку береться БІФ, результати якого піддаються потрібній трансформації. Надалі береться зворотне БІФ, в результаті чого значення зворотного ШПФ скалярно множиться на вікно, де після множення блоки складаються з перекриттям.

При виконання трансформацій спектра не варто забувати про головну властивість масиву БПФ дійсних сигналів. Перша половина масиву БІФ комплексно пов'язана з другою половиною, тобто Re [i] = Re [(1 << n) -i]); Im [i] = -Im [(1 << n) -i]) ;. Якщо ці вимоги не виконати - вихідний сигнал не збереться. Представлений алгоритм згодом переводиться в кодове подання, яке найпростіше було реалізувати на рівні мови С [2].

ВИСНОВКИ

Використання різноманітних технологій та процесів при створенні комплексної системи на основі витратомірів потребує суттєвих знань щодо представлення підсумкових даних в комфортному вигляді.

В ході роботи були досліджені відповіді варіанти реалізації взаємодії даних, їх створення та запис, а також відображення в прийнятному вигляді щодо користувача. Для побудови системи використовувалися клієнтські та серверні технології реалізації, в основі яких базувалась єдина мова програмування – JavaScript. В якості використовуваної бази даних була обрана MongoDB, котра дозволяла зручно працювати з будь-якими даними та відтворювати надійну архітектуру. Реалізація WEB-інтерфейсу дозволила також зручно працювати як на персональних комп’ютерах, так і на мобільних пристроях.

Відносність побудованої системи в першу чергу виражається в тому, що її основна функція – відображати надходженні дані з бази даних та дозволяти користувачу здійснювати повний контроль та маніпуляції щодо кінцевого відображення. Слід зазначити, що дана система являється невід’ємною складовою всього комплексу представленої роботи в силу того, що праця кожної частини має кінцеву корисну дію тільки в сукупності.

В першій частині дипломного проекту розкриваються основні концепти побудови WEB додатків та їх особливості. Описані концептуальні відмінності та аргументи щодо вибору остаточних рішень реалізації.

В другій частині представлений детальний опис роботи та функціонування системи, а також її особливості та відмінності відповідно до аналогів. Також фокус уваги приділяється відносності саме щодо витратомірів та їх роботи в зв’язку з роботою системи додатку.

В третій частині приведені експериментальні дослідження щодо побудови дискретних лінійних динамічних систем, а також особливості імпульсної характеристики та сутності використання рядів Фур’є.

ПЕРЕЛІК ДЖЕРЕЛ

1. Бондаренко М. Ф., Биков М. М., Дзюбенко М. І., Пащенко О. Г., Пащенко Ж. Ф. Прилади та пристрої квантової електроніки з використанням ЕОМ для дослідження і аналізу – Харків, ХНУРЕ, 2002 – 388 с.
2. Фрісбі М. Angular. Сборник рецептов – Москва, Діалектика-Вільямс, 2018 – 205с.
3. Кантелон М., Янг А., Мек М. – Node.js в действии. 2-е издание ,– Петербург, 2017 – 380с.
4. Кузьмін С.Т., Ліпавскій В.Н., Смирнов П.Ф. Промислові прилади і засоби автоматизації в нафтопереробній та нафтохімічній промисловості – М .: Хімія, 1987.-272с
5. Гордюхін А.І., Гордюхін Ю.А. Вимірювання витрати та кількості газу і його облік – Л .: Недра, 1987.-213с.
6. Levchenko.Y.L, Zolotuhin A.A. Bondarenko I.M. COMPUTER MODELING OF THE ULTRASONIC FLOW METER SYSTEM BASED ON THE WAVE DEFLECTION PRINCEPLE/ Y.L. Levchenko, A.A. Zolotuhin, I.M. Bondarenko – K SPO-2019 26-29, 2019