НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.9

«До захисту допущено»

Завідувач кафедри

_____ Едуард ЖАРІКОВ

«___»_____2022 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Інженерія програмного забезпечення комп'ютерних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Методи та програмні засоби для управління кластером сонячних електростанцій»

Виконав:
студент ІІ курсу, групи ІТ-01мн
Мокрий Андрій Вікторович _____

Керівник:
Доцент кафедри ІПІ, к. т. н., доцент
Баклан Ігор Всеволодович _____

Рецензент:
доцент кафедри ІСТ, к. т. н., доцент
Резніков Сергій Анатолійович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-наукова програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Едуард ЖАРІКОВ

«___»_____2022р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Мокрому Андрію Вікторовичу

1. Тема дисертації «Методи та програмні засоби для управління кластером сонячних електростанцій», науковий керівник дисертації Баклан Ігор Всеволодович, к. т. н., доцент, затверджені наказом по університету від «24» квітня 2022 р. № 88

2. Термін подання студентом дисертації «6» червня 2022 р.

3. Об'єкт дослідження – програмне забезпечення для управління сонячними електростанціями.

4. Предмет дослідження – методи та програмні засоби для забезпечення віддаленого управління кластером сонячних електростанцій.

5. Перелік завдань, які потрібно розробити: провести аналіз існуючих рішень для управління сонячними електростанціями; провести аналіз проблем прототипу та можливих рішень; дослідити шляхи підвищення швидкості роботи системи; розробити алгоритм паралельного обчислення оптимального положення сонячних панелей; спроєктувати архітектуру для більш ефективного управління кластером сонячних електростанцій; провести аналіз результатів роботи спроєктованої системи управління; порівняти результати з прототипом.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: Архітектура розробленої системи; Схема інформаційних потоків системи; Алгоритм q-

навчання; Діаграма класів сервісу станції; Схема розробленої бібліотеки; Алгоритм роботи сервісу станції.

7. Орієнтовний перелік публікацій: Мокрий А.В. Методи та програмні засоби для управління кластером сонячних електростанцій / А.В. Мокрий, І.В. Баклан // Науковий журнал «Адаптивні системи автоматичного управління» (АСАУ-2022) – м. Київ: НТУУ «КПІ ім. Ігоря Сікорського», 2022 р.

8. Дата видачі завдання «30» вересня 2020 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Ознайомлення з завданням	01.10.2020 – 31.10.2020	
2	Аналіз предметної області	01.11.2020 – 31.03.2021	
3	Аналіз існуючих рішень	01.04.2021 – 18.09.2021	
4	Тестування прототипу	19.09.2021 – 30.09.2021	
5	Проєктування архітектури системи	01.10.2021 – 31.10.2021	
6	Розробка ПЗ електростанцій	01.11.2021 – 30.11.2021	
7	Розробка клієнтського додатку	01.12.2021 – 31.12.2021	
8	Тестування системи	01.01.2022 – 15.01.2022	
9	Виконання експериментальних досліджень	16.01.2022 – 31.01.2022	
10	Оформлення пояснювальної записки	01.02.2022 – 21.04.2022	
11	Подання дисертації на попередній захист	22.04.2022	
12	Подання дисертації на захист	10.05.2022	

Студент Андрій МОКРИЙ

Науковий керівник Ігор БАКЛАН

АНОТАЦІЯ

Розмір пояснювальної записки – 75 аркушів, містить 38 ілюстрацій, 5 таблиць, 8 додатків.

Актуальність теми. На сьогоднішній день технології виробництва сонячних панелей бурхливо розвиваються, інвестиції в сонячну енергетику зростають, тож користувачі зацікавлені у збільшенні виробітку енергії для швидшої окупності вкладень. Для підвищення ефективності сонячних панелей використовують спеціальні поворотні механізми, які повертають панель перпендикулярно до сонячних променів. Це дозволяє значно збільшити кількість енергії, яку виробляє панель за проміжок часу. У даній роботі розглядаються шляхи підвищення ефективності генерації енергії сонячними панелями з використанням мінімальної кількості додаткових датчиків. Концепції віддаленого управління та паралельних обчислень, які розглядаються в даній роботі, можуть бути використані для вирішення багатьох типів прикладних задач.

Мета дослідження. Основною метою є розширення функціоналу існуючих систем управління сонячних електростанцій та забезпечення їх швидкодії, якості та надійності.

Об’єкт дослідження: програмне забезпечення для управління сонячними електростанціями.

Предмет дослідження: методи та програмні засоби для забезпечення віддаленого управління кластером сонячних електростанцій.

Для реалізації поставленої мети **сформульовані наступні завдання:**

- провести аналіз існуючих рішень для управління сонячними електростанціями;
- провести аналіз проблем прототипу та можливих рішень;
- дослідити шляхи підвищення швидкості роботи системи;
- розробити алгоритм паралельного обчислення оптимального положення сонячних панелей;

- спроєктувати архітектуру для більш ефективного управління кластером сонячних електростанцій;
- провести аналіз результатів роботи спроєктованої системи управління порівняно з прототипом.

Наукова новизна результатів магістерської дисертації полягає в тому, що вперше створено бібліотеку для паралельного обчислення оптимального положення сонячних панелей з використанням машинного навчання. Також набуло подальшого розвитку використання мікросервісної архітектури для вирішення прикладних задач.

Практичне значення отриманих результатів полягає в тому, що розроблене рішення може самостійно коригувати положення сонячних панелей з мінімальною кількістю датчиків. Процес корекції не залежить від backend-частини. Розробка даної системи відбувалася в мирний час, але концепцію паралельного обчислення оптимального положення сонячних панелей можна використати для реалізації одночасного наведення артилерійських батарей або засобів протиповітряної оборони по координатах.

Зв’язок з науковими програмами, планами, темами. Робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського".

Публікації. Наукові положення дисертації опубліковані в:

1) Мокрий А.В. Методи та програмні засоби для управління кластером сонячних електростанцій / А.В. Мокрий, І.В. Баклан // Науковий журнал «Адаптивні системи автоматичного управління» (АСАУ-2022) – м. Київ: НТУУ «КПІ ім. Ігоря Сікорського», 2022 р.

Ключові слова: СОНЯЧНА ЕЛЕКТРОСТАНЦІЯ, МОНІТОРИНГ СОНЯЧНОЇ ЕЛЕКТРОСТАНЦІЇ, МАШИННЕ НАВЧАННЯ З ПІДКРІПЛЕННЯМ, Q-НАВЧАННЯ, ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ, ПОТОКИ, МІКРОСЕРВІСИ.

ABSTRACT

Explanatory note size – 75 pages, contains 38 illustrations, 5 tables, 8 appendices.

Topicality: Nowadays, solar panel production technologies are developing rapidly, investments in solar energy are growing, so users are interested in increasing energy production for faster return on investment. To increase the efficiency of solar panels, special rotating mechanisms are used to rotate the panel perpendicularly to the sun rays. This allows users to significantly increase the amount of energy produced by the panel over time. This paper considers ways to increase the efficiency of solar panels' energy generation by using a minimum number of additional sensors. The concepts of remote control and parallel computing, which are considered in this paper, can be used to solve wide range of problems.

The aim of the study. The main goal is to extend the functionality of existing solar power plant management systems and ensure their speed, quality and reliability.

Object of research: software for solar power plant management.

Subject of research: methods and software for solar power plant cluster management.

To achieve this goal, the **following tasks** are formulated:

- perform an analysis of existing solar power plant management systems;
- perform an analysis of the prototype's problems and possible solutions;
- explore ways to increase the speed of the system;
- develop an algorithm for parallel calculation of the optimal position of solar panels;
- design an architecture for more efficient cluster management of solar power plants;
- analyze the work results of the designed control system in comparison with the prototype.

The scientific novelty of the results of the master's dissertation is that for the first time, there has been created a library for parallel calculation of the optimal position of

solar panels using machine learning. The use of microservice architecture to solve applied problems has been further developed.

The practical value of the obtained results is that the developed solution can independently adjust the position of solar panels with a minimum number of sensors. The correction process does not depend on the backend part. The development of this system took place in peacetime, but the concept of parallel calculation of the solar panels' optimal position can be used to implement simultaneous aiming at coordinates for artillery batteries or air defense systems.

Connection with scientific programs, plans, topics. The work was performed at the Department of Informatics and Software Engineering of the National Technical University of Ukraine "Kyiv Polytechnic Institute named after Igor Sikorsky".

Publications. The scientific provisions of the dissertation published in:

1) Mokryi A.V. Methods and software for solar power plant cluster management / A.V. Mokryi, I.V. Baklan // Scientific Journal "Adaptive Automatic Control Systems" (AACS-2022) - Kyiv: NTUU "KPI them. Igor Sikorsky ", 2022.

Keywords: SOLAR POWER PLANT, SOLAR POWER PLANT MONITORING, REINFORCEMENT MACHINE LEARNING, Q-LEARNING, PARALLEL PROGRAMING, THREADS, MICROSERVICES.

ЗМІСТ

ПЕРЕЛІК ВАЖЛИВИХ СКОРОЧЕНЬ І ТЕРМІНІВ

API – програмний інтерфейс, який дозволяє іншим додаткам взаємодіяти з даним.

GPS – система глобального позиціонування.

IP-адреса – ідентифікатор мережевого рівня, який використовується для адресації комп'ютерів чи пристроїв у мережах.

БД – база даних.

Ідентифікатор, ID – унікальний числовий номер.

Інвертор – пристрій, який перетворює постійний струм у змінний з вказаною частотою.

Мікросервіс, сервіс – модуль ПЗ, який працює у своєму власному процесі і взаємодіє з рештою сервісів.

ОС – операційна система.

ПЗ – програмне забезпечення.

Потік (thread) – набір команд, що виконуються послідовно.

Сервер (ПЗ) – програма, яка надає деякі послуги іншим програмам (клієнтам).

СЕС, ФЕС – сонячна електростанція, фотоелектрична станція.

СКВ – система керування версіями.

СУБД – система управління базами даних.

ВСТУП

Сучасний світ неможливо уявити без електричної енергії. На сьогодні електроенергія займає одне з провідних місць у житті та життєдіяльності людини. Майже вся наявна інфраструктура всіх країн світу для своєї роботи використовує даний вид енергії. Мешканці як великих міст, так і невеличких поселень залежні від неї тому, що без неї не буде освітлення, центрального водопостачання та водовідведення, опалення. Робота майже всіх пристроїв – як побутових, так і промислових – живиться від електроенергії.

У період зростання виробництва збільшується попит на електроенергію, тому будівництво нових електростанцій є необхідністю. До традиційних видів електростанцій відносять теплові, атомні та гідроелектростанції. Кожен з цих видів має суттєві недоліки: ТЕС та АЕС працюють на викопному паливі, яке рано чи пізно закінчиться; ГЕС виробляють до 18,1% електроенергії світу, але для їх роботи необхідний потужний гідроенергопотенціал.

З 1980-х років країни світу активізували пошук альтернативних джерел отримання електроенергії. На морських узбережжях, у відкритих районах, для отримання електроенергії почали використовувати енергію вітру. Сонячна енергія є найдешевшим видом енергії, але на даний час використовується недостатньо. На відміну від ТЕС та АЕС, такі станції не потребують палива для роботи і вироблятимуть енергію, доки світитиме Сонце. Також при їх спорудженні не потрібно проводити масштабні роботи, такі як створення водосховищ при будівництві ГЕС. Крім того, наслідки аварій на сонячних електростанціях (СЕС) будуть менш руйнівними.

Наведені вище аргументи зумовлюють зростання популярності даного виду електростанцій. На сьогоднішній день технології виробництва сонячних панелей бурхливо розвиваються, інвестиції в сонячну енергетику зростають, тож користувачі зацікавлені у збільшенні виробітку енергії для швидшої окупності вкладень. Використання поворотних механізмів, які повертають панель

перпендикулярно до сонячних променів дозволяє збільшити кількість енергії, яку виробляє панель за проміжок часу.

Існуючі рішення для управління сонячними електростанціями вимагають використання додаткових датчиків для реалізації стеження за Сонцем. Вони також накладають суворі обмеження на обладнання, на яке вони можуть бути встановлені. Серед існуючих рішень відсутні кросплатформні, вони працюють тільки на обладнанні конкретного виробника. Розроблене рішення передбачає використання машинного навчання для вирішення проблеми стеження за Сонцем з мінімальною кількістю датчиків. У даній роботі розглянуто прототип системи управління сонячними електростанціями з використанням машинного навчання, його переваги та недоліки. На прикладі прототипу досліджуються способи підвищення швидкості роботи, якості та надійності подібних систем.

Об’єктом дослідження даної роботи є програмне забезпечення для управління сонячними електростанціями.

Предметом дослідження даної роботи є методи та програмні засоби для забезпечення віддаленого управління кластером сонячних електростанцій.

Метою даної роботи є розширення функціоналу існуючих систем управління сонячними електростанціями, а також підвищення їх швидкості роботи, якості та надійності за рахунок використання паралельного програмування та машинного навчання.

Для досягнення мети необхідно провести аналіз існуючих рішень для управління сонячними електростанціями, провести аналіз проблем прототипу та можливих рішень, дослідити шляхи підвищення швидкості роботи системи, розробити алгоритм паралельного обчислення оптимального положення сонячних панелей, спроєктувати архітектуру для більш ефективного управління кластером сонячних електростанцій, провести аналіз результатів роботи спроєктованої системи управління, порівняти результати з прототипом.

1 ЗАГАЛЬНИЙ ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ДЛЯ УПРАВЛІННЯ СОНЯЧНИМИ ЕЛЕКТРОСТАНЦІЯМИ

1.1 Опис предметної області

Сонячне випромінювання є одним з найбільш перспективних джерел екологічно чистої енергії. Станом на 2021 рік частка сонячної енергії у світовій енергетиці перевищила 5%. Сонячна енергетика найбільш динамічно розвивається серед усіх галузей альтернативної енергетики. За останні 15 років щорічний приріст потужностей, що вводяться в експлуатацію, складає 40-50%, водночас інвестиції в удосконалення технологій виготовлення фотоелектричних модулів склали близько 300 мільярдів доларів США. У зв’язку з цим з кожним роком значно знижується собівартість даного виду електроенергії. Так, у деяких країнах сонячна енергія стала дешевшою за одержувану з газу, нафти або вугілля. Серед них Австралія, Мексика, Німеччина.

Для країн, які повністю залежать від поставок викопного палива, впровадження сонячних технологій дає можливість забезпечити енергетичну незалежність. Насамперед це стосується острівних країн. Яскравим прикладом є острів Тау (Американське Самоа), мешканці якого стали незалежними від поставок дизельного палива після запуску сонячної електростанції.

В Україні, як і в більшості європейських країн, сонячна енергетика стрімко розвивається. Станом на перший квартал 2021 року частка сонячної енергії склала близько 6% від усієї виробленої енергії. Якщо й в подальшому темпи приросту зростатимуть, то до 2030 року удесятеро збільшиться використання даної енергії, і це позитивно вплине на зниження залежності від традиційних видів палива, зокрема можна буде скоротити споживання природного газу на 15%.

Електрика та інші види енергії, навіть у хмарну погоду, отримуються від сонця. Уся територія України, її клімат і географічне положення сприятливі для розвитку сонячної енергетики. Навіть у північних областях даний вид енергетики активно розвивається. З 2008 року в Україні діє «Зелений тариф» – держава

закуповує електроенергію, отриману з відновлюваних джерел, у приватних осіб та організацій за підвищеною ціною.

У березні 2022 року оператором ГТС України прийнято рішення встановити сонячні панелі на газорозподільних станціях для забезпечення резервного живлення. Реалізація цього проєкту планувалася ще до війни для оптимізації споживання енергетичних ресурсів. Забезпечення додатковими джерелами електроенергії дозволить безперебійно забезпечувати газом населення навіть у регіонах, які найбільше постраждали під час бойових дій.

«*Тепер, коли електромережі України потерпають від постійних обстрілів ворога, це необхідність, щоб газорозподільні станції мали автономні варіанти живлення»*, – повідомив гендиректор ТОВ «Оператор ГТС України» Сергій Макогон [1].

Також сонячна енергетика допомогла мешканцям кількох окупованих сіл на Київщині підтримувати роботу життєво необхідного обладнання і побутових приладів. Місцевий житель зміг самостійно змонтувати конструкцію, яку він називав «сумішшю автономки з AC-coupling». Такий спосіб підключення сонячних панелей відрізняється від більш «традиційного» відсутністю накопичувача електричної енергії (акумуляторних батарей) і, відповідно, перетворювачів постійного струму, які необхідні для роботи накопичувачів енергії. За допомогою цієї системи вдалося забезпечити живленням одночасно як потужну помпу свердловини, так і деякі електроприлади, а також сусідські будинки. За допомогою даної системи мешканці змогли дочекатись відновлення мережі [2].

Сонячна енергія використовується у всьому світі і стає все більш популярною. ЇЇ використовують для вироблення електроенергії, опріснення води, опалення та інших цілей.

Є два основні способи для генерації сонячної енергії: фотогальванічні елементи та системи концентрованої сонячної енергії.

Фотогальванічні (PV) або сонячні елементи – це напівпровідникові пристрої, за допомогою яких сонячне світло перетворюється в електрику. З такими

сонячними елементами так чи інакше стикалася переважна кількість людей в повсякденному житті. Прикладом таких елементів є панелі, встановлені на калькуляторах. Також сонячні елементи даного типу встановлюються на полях, дахах або стінах будинків. Вони були винайдені в 1954 році в Bell Telephone Laboratories в США. На сьогодні такі системи розвиваються з вражаючою швидкістю. Використання поновлюваних джерел енергії зіграє важливу роль в майбутньому глобальному виробництві електроенергії. Сонячні фотоелектричні установки використовують як для забезпечення електроенергією в промислових масштабах, так і для забезпечення невеликих виробництв або приватних господарств За останні роки завдяки здешевленню виробництва, та досить тривалому терміну служби (близько 30 років) даний вид енергії став не тільки доступним, а й досить дешевим видом електроенергії.

Іншим способом вироблення сонячної енергії є системи концентрованої сонячної енергії (CSP). Для даних систем використовуються дзеркала для концентрації сонячних променів. За допомогою цих променів нагрівається рідина і за допомогою пари запускаються турбіни для вироблення електроенергії. Для роботи зазвичай використовується значна кількість дзеркал, які перенаправляють промені на високу тонку вежу. Така технологія використовується для вироблення електроенергії на великих електростанціях. Перевагою перед сонячною фотоелектростанцією є те, що вона може бути доповнена ємностями з розплавленими солями для накопичення теплової енергії. Даний підхід дозволяє генерувати електрику навіть після заходу сонця [3].

Як і інші виробництва, сонячні електростанції мають свої переваги та недоліки. До переваг можна віднести:

- Енергонезалежність виробництва. Сонячне світло невичерпне, на відміну від корисних копалин. Використання сонячних систем суттєво зменшує використання традиційної електроенергії.

- Добування сонячної енергії в абсолютно безпечне для навколишнього середовища. В подальшому людство здатне забезпечити свої потреби в енергії повністю за рахунок сонця;
- Даний вид енергії доступний кожній країні;
- Сонячні електростанції не потребують великої кількості обслуговуючого персоналу. Це пов'язано з тим, що технології постійно розвиваються і дані системи доволі прості в експлуатації. Загалом за рахунок високої автоматизації технічне обслуговування не потребує значних витрат. Термін окупності інвестицій близько 10 років;

Серед недоліків сонячних електростанцій можна виділити наступні:

- Висока вартість. Ці інвестиції окупляться через досить тривалий час;
- Залежність від сезону. Адже в літні місяці сонячна активність набагато більша, ніж в зимові, світловий день влітку та взимку суттєво відрізняється. Тому влітку електростанції працюють у посиленому режимі. Використання акумуляторів для накопичення енергії значно збільшує собівартість.
- Також для встановлення сонячних електростанцій необхідна значна вільна площа: земля, дах або стіна будинку.

Підсумовуючи вище сказане, можна сказати, що сонячні електростанції – це інвестиції в екологічно чисте майбутнє не тільки України, а й планети в цілому.

1.2 Аналіз існуючих рішень

Існує багато реалізацій систем управління сонячними станціями. Виробники сонячних електростанцій пропонують своє обладнання для систем керування, яке використовує унікальні технології для підвищення якості та потужності. Більшість таких систем керує положенням панелей; може відображати інформацію про виробіток кожної панелі, стан перемикачів та з'єднань станції; статистику генерації та споживання електроенергії. Можливість віддаленого онлайн-моніторингу з будь-якої точки Землі теж входить у функціонал сучасного обладнання.

Для користувача створюється обліковий запис (акаунт) – робочий кабінет, через який можна стежити за процесом генерації енергії в реальному часі. Таким чином з’являється можливість спостерігати за роботою електростанції з комп’ютера або смартфона, та за необхідності оперативно реагувати на зміни.

1.2.1 Solar.web

Одним з рішень для моніторингу сонячних електростанцій є система “Solar.web” [4] від компанії “Fronius” (рис. 1.1).

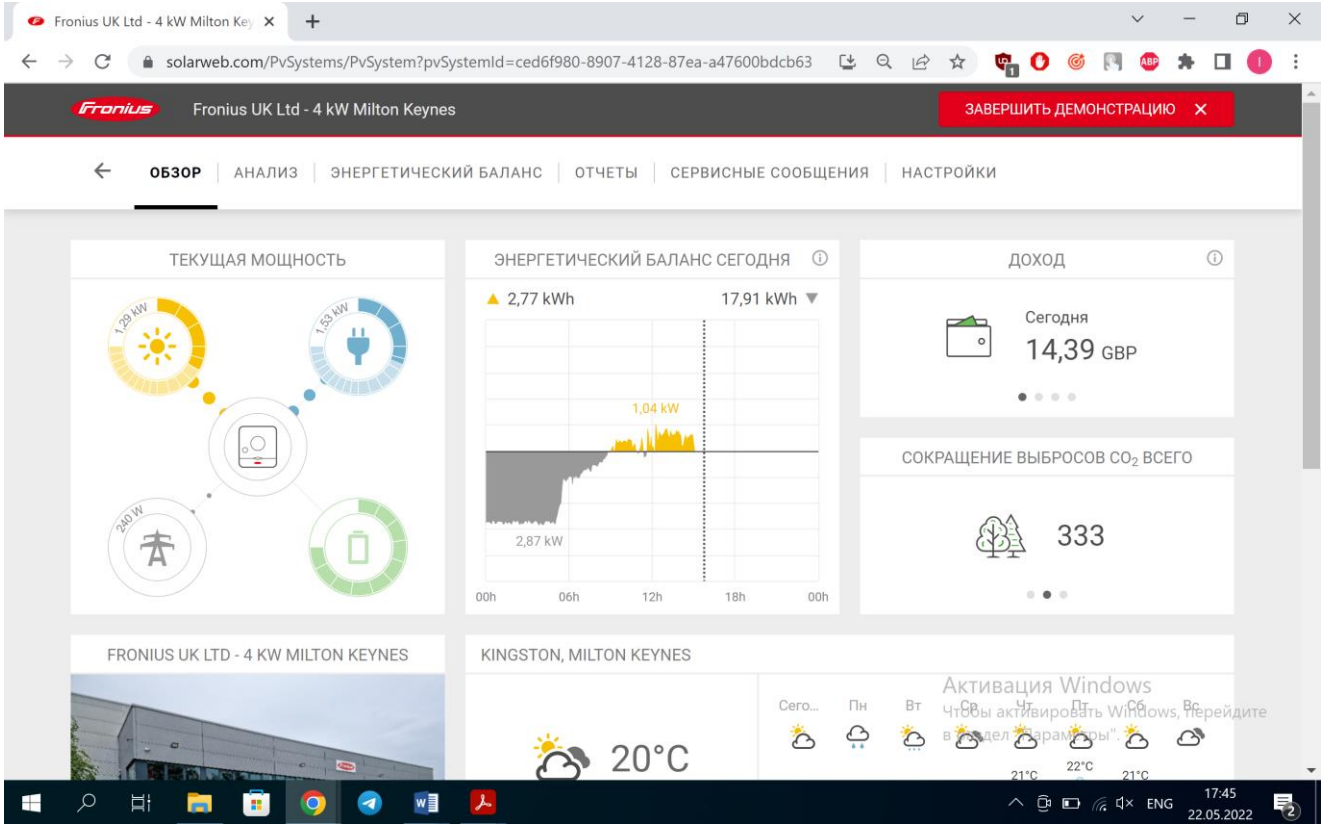


Рисунок 1.1 – Демонстрація роботи системи “Solar.web”

Дана система має наступні переваги:

- Дозволяє легко керувати великою кількістю сонячних електростанцій.
- Відображає енергетичні потоки в режимі реального часу.
- Відображає графіки щоденного виробітку та споживання електроенергії за останні три дні.

- Показує історію виробництва та споживання на денному, місячному та річному рівнях протягом усього терміну служби фотоелектричної системи.
- Забезпечує детальний аналіз параметрів фотоелектричної системи протягом усього терміну експлуатації системи.
- Можливість симуляції підключення додаткового обладнання до станції.
- Доступні версії для iOS, Android та веб-версія.
- Показує прогноз погоди.

Серед недоліків даної системи можна виділити наступні:

- Прив’язка до виробника. Дане рішення може встановлюватися лише на інвертори компанії “Fronius”.
- Відсутність інформації по окремим панелям, доступна інформація лише про блоки панелей, підключених до одного інвертора.
- Відсутність можливості керувати з’єднаннями станції та підключенням окремих блоків.

1.2.2 SolarEdge Monitoring System

Іншим рішенням для моніторингу сонячних електростанцій є “SolarEdge Monitoring System” [5] від компанії “SolarEdge” (рис.1.2).

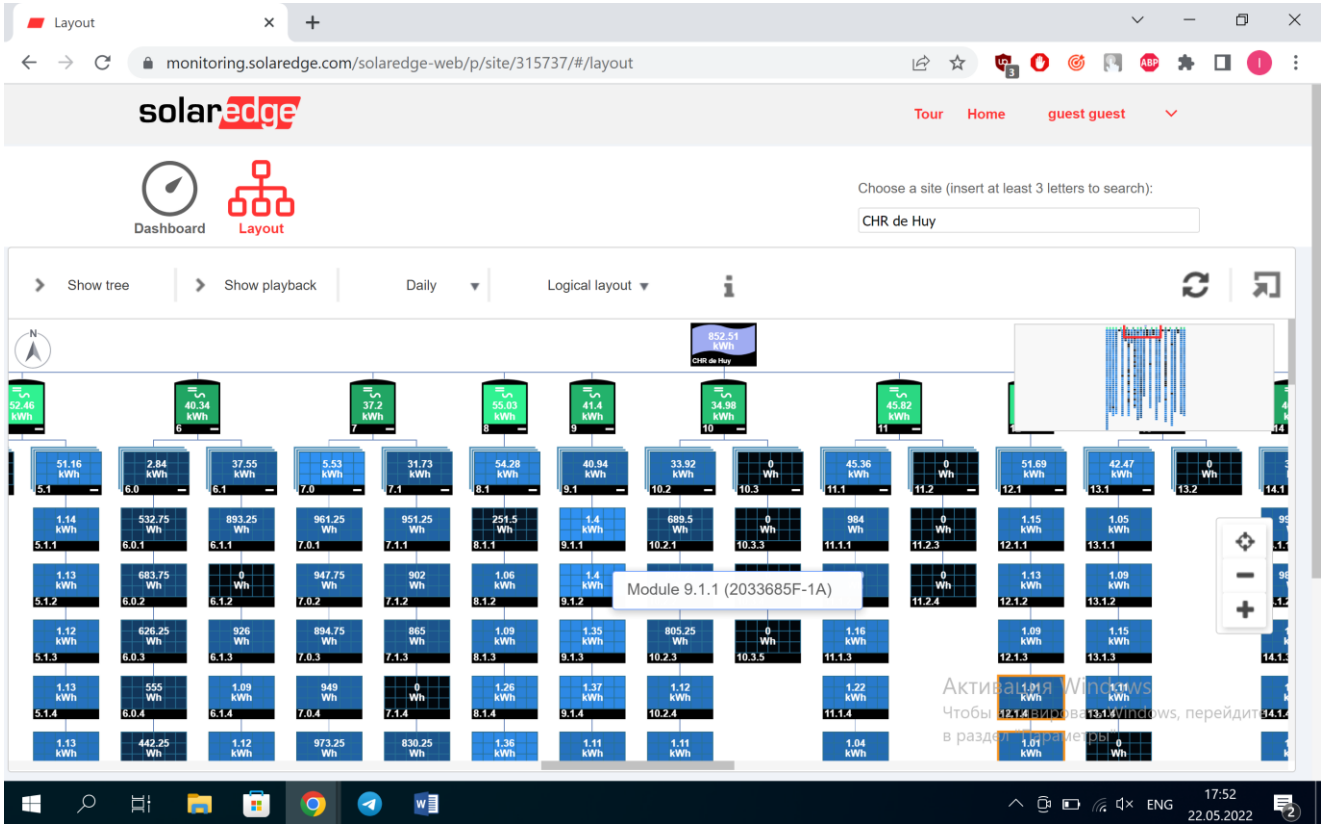


Рисунок 1.2 – Демонстрація роботи системи “SolarEdge”

Дана система має наступні переваги:

- Дозволяє легко керувати великою кількістю сонячних електростанцій.
- Показує схему фізичного розміщення сонячних панелей.
- Відображає поточну потужність та положення для кожної панелі.
- Відображає графіки щоденного виробітку електроенергії за день, тиждень, місяць.
- Показує порівняльний графік виробітку енергії станцією за місяць, квартал, рік протягом останніх п’яти років.
- Доступні версії для iOS, Android та веб-версія.
- Показує прогноз погоди.

Серед недоліків даної системи можна виділити наступні:

- Прив’язка до виробника. Дане рішення може встановлюватися лише на інвертори компанії “SolarEdge”.

- Відсутність можливості керувати з’єднаннями станції та підключенням окремих блоків.

1.3 Постановка задачі

Метою роботи є розширення функціоналу існуючих систем моніторингу сонячних електростанцій та забезпечення їх швидкодії, якості та надійності. Проаналізувавши функціональні можливості існуючих рішень, робимо висновок про необхідність розробки програмного забезпечення, яке буде поєднувати в собі функції контролю стану станції, віддаленого управління сонячною електростанцією та онлайн-моніторингу. Для досягнення мети необхідно вирішити наступні задачі:

- Провести комплексний аналіз проблем прототипу та можливих рішень.
- Дослідити шляхи підвищення швидкості роботи системи.
- Удосконалити алгоритм періодичного коригування положення сонячних панелей.
- Спроєктувати архітектуру для управління кластером сонячних електростанцій.
- Провести аналіз результатів роботи спроєктованої системи управління.

Створена система повинна відповідати наступним вимогам:

- корекція положення сонячних панелей на станціях має виконуватися автоматично із заданою періодичністю;
- інформація про стан станції, положення та потужність панелей має бути доступна в режимі реального часу;
- користувач повинен мати можливість керувати підключенням окремих панелей, накопичувача енергії та загальної електромережі;
- станції повинні мати захист від несанкціонованого доступу;
- система має бути простою у налагодженні та використанні;
- клієнтська частина повинна мати зручний інтерфейс користувача.

1.4 Загальні відомості про прототип

1.4.1 Архітектура прототипу

Прототипом є система контролю роботи сонячної електростанції, яка була розроблена мною в ході виконання дипломного проєкту бакалавра. Дана система складається з наступних компонентів (рис. 1.3):

a) Рівень центрального сервера.
b) Рівень системи контролю.
c) Рівень веб-додатку.

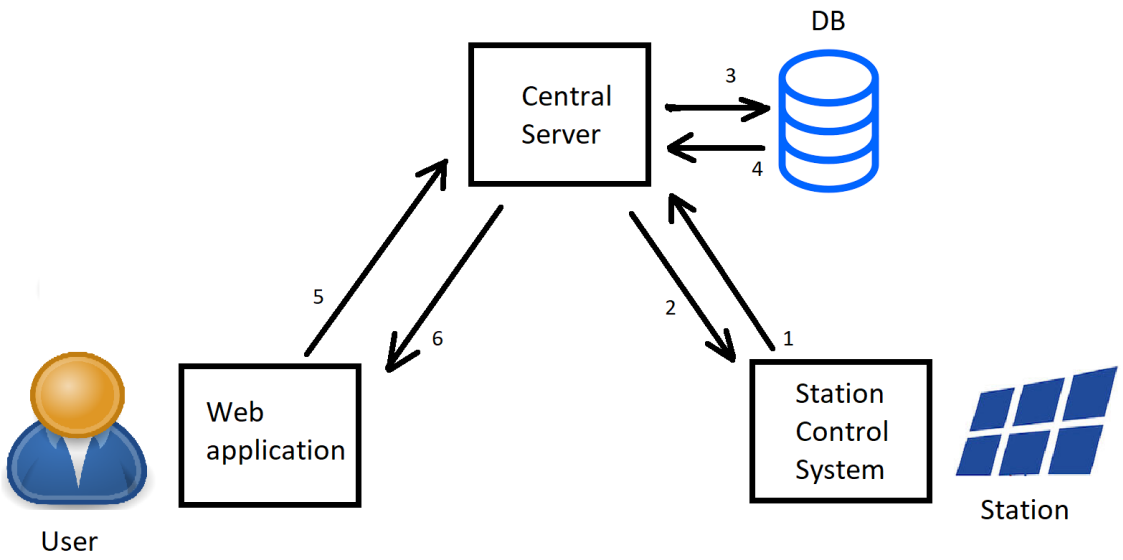


Рисунок 1.3 – Схема взаємодії модулів прототипу

Центральний сервер має доступ до бази даних, в якій зберігається інформація про користувачів системи, їхні панелі, вироблену та витрачену електроенергію. З рисунку ми бачимо, що всі запити користувачів та станцій проходять через центральний сервер, який є точкою входу для системи.

Рівень системи контролю являє собою програму, яка встановлюється безпосередньо на контролері електростанції. Ця програма з заданою періодичністю зчитує дані про кожну панель, робить запити на центральний сервер, передає

інформацію про стан станції та отримує команди управління, такі як «повернути панель X на 1 градус вгору». При цьому запит передається шляхом 1-3-4-2.

За допомогою рівня веб-додатку користувач отримує доступ до актуальної інформації про роботу своєї станції на сайті компанії. Даний рівень також дозволяє керувати підключенням окремих блоків. Для отримання доступу до даних своєї електростанції користувач має авторизуватися, після чого центральний сервер передає команди управління відповідній станції.

1.4.2 Пошук оптимального положення сонячної панелі

Для пошуку оптимального положення панелі використовується алгоритм навчання з підкріпленням [6]. Кожна панель в кожний момент часу знаходиться в певному стані, який визначається координатами азимуту (поворот за годинниковою стрілкою від напрямку на Північ) та висоти. Кожний стан використовує чотири варіанти дії (вгору, вниз, лівіше, правіше). При виборі виконується поворот на 1 градус у відповідну сторону.

Кожний стан має для кожної дії значення корисності. Нехай стан X має для дій вгору, вниз, лівіше, правіше відповідні значення: a, b, c, d. Прототип використовує наступний алгоритм:

a) Встановлюємо первісні значення корисності дій для даного стану панелі. Відомо, що Сонце впродовж дня рухається за годинниковою стрілкою, встановимо корисність дії “поворот правіше” значенню 0.1, іншим діям встановимо корисність що буде дорівнювати 0. Для стану “на 1 правіше” корисність дій “лівіше” та “правіше” встановимо на 0.

b) Обчислюємо поточну потужність Px.

c) Визначаємо рекомендовану дію

$$i := \text{indexOf}(\max(a, b, c, d)) \tag{1.1}$$

d) Виконуємо дію, що переводить панель у стан X+1.

e) Обчислюємо потужність в результаті виконання дії P(x+1).

f) Обчислення вигоди від дії i.

$$diff := P(x+1) – Px \quad (1.2)$$

g) Оновлюємо значення корисності від дії i.

Припускаємо, що корисність дії i в стані X збільшилася на величину diff:

$$X[i] := X[i] + diff \quad (1.3)$$

Оскільки Сонце знаходиться по прямій, вважаємо, що панель у разі “успіху” дії i повинна продовжити рух у тому ж напрямку. Для цього:

$$(X+1)[i] := (X+1)[i] + diff / 10 \quad (1.4)$$

У результаті, якщо хід збільшив отримувану потужність панелі, вона отримає стимул рухатися вперед. Якщо ж хід зменшив отримувану потужність панелі, вона отримає стимул повернутися назад і обрати інший шлях.

h) Виконується перевірка оптимальності встановленого положення. Якщо корисність для усіх дій від’ємна, то завершуємо роботу, інакше переходимо до кроку b).

i) Кінець.

Система повторює даний алгоритм кожні 10 хвилин.

1.4.3 Можливості прототипу

Прототип здатний виконувати наступні функції:

- Самостійний, без участі користувача пошук оптимального положення для кожної панелі. З часом пошук відбувається швидше за рахунок зберігання максимально корисних дій. Таким чином система «навчається».
- Забезпечує користувачу можливість віддалено спостерігати за роботою своєї станції, а також керувати підключенням панелей, виходу до мережі.
- Створюється можливість накопичити певну кількість енергії та продати її в момент найбільшого попиту і, відповідно, ціни на неї.
- Дозволяє зекономити на обладнанні, оскільки для пошуку оптимального положення панелі використовуються тільки датчики положення та потужності панелі. Системі не потрібно знати положення Сонця в кожний

момент часу для даної широти, отже, не потрібно встановлювати GPS-модуль. Також не потрібно встановлювати панелі на ідеально вирівняну поверхню та направляти їх на ідеально південний напрямок. Це дозволяє зекономити при підготовці поверхні до монтажу станції без втрати продуктивності.

1.4.4 Обмеження прототипу

Машинне навчання прототипу передбачає роботу з корисністю визначених дій у певних станах. У випадку з системою контролю роботи сонячної електростанції стан кожної сонячної панелі визначається координатами її положення. Відповідно до алгоритму, на кожному кроці потрібно дізнатися найкращу дію для поточного стану, а потім оновити значення корисності дій для даного стану. Таким чином, програмне забезпечення станції здійснює велику кількість запитів до бази даних. Оскільки всі такі запити проходять через центральний сервер, підключення багатьох сонячних панелей буде викликати його перевантаження. Дану проблему можна вирішити, якщо дозволити станціям взаємодіяти з базою даних напряму, що і розглядається у моїй роботі.

Іншим недоліком прототипу є те, що пошук оптимального положення виконується послідовно для кожної панелі. Оскільки в реальності зміна положення панелі може займати десятки секунд, то при збільшенні кількості сонячних панелей алгоритм може не встигати оновити їх положення за визначений час. У даній роботі розглядається використання засобів паралельного програмування для пришвидшення процесу знаходження оптимального положення сонячних панелей у декілька разів.

Також вузьким місцем прототипу є центральний сервер, оскільки при коригуванні положення сонячних панелей через нього проходить велика кількість запитів до бази даних. Виведення його з ладу не лише унеможливлює перегляд інформації користувачем, а й блокує процес стеження панелей за Сонцем. Таким чином, навіть виконання планових технічних робіт на сервері буде заважати

користувачам отримувати максимальний прибуток від своїх електростанцій. У даній роботі розглядається модифікація архітектури системи для вирішення цієї проблеми.

1.5 Деталізація постановки задачі

Після аналізу проблем прототипу можна конкретизувати вимоги до системи, що створюється:

1. Необхідно забезпечити автономність електростанцій. Процес стеження панелей за Сонцем повинен відбуватися незалежно від інших компонентів системи.
2. Пошук оптимального положення має відбуватися паралельно для декількох панелей для підвищення швидкодії.
3. Необхідно знизити навантаження на окремі компоненти системи.
4. Забезпечити простоту масштабування системи.

Реалізація даних вимог забезпечить підвищення швидкості роботи, якості та надійності системи управління кластером сонячних електростанцій.

Висновки до розділу

У даному розділі надано прогноз розвитку сонячної енергетики базуючись на сучасному стані та статистичних даних за минулі роки. Наведено приклади сучасних систем управління сонячними станціями з оглядом їх можливостей і обмежень. Акцентована прив’язка таких систем лише до певного обладнання. Зроблено висновок про необхідність розробки нової системи, вказано основні вимоги, яким вона має відповідати. Наведено відомості про прототип, його сильні та слабкі сторони. Розглянуто алгоритм навчання з підкріпленням, який використовується для розрахунку оптимального положення сонячної панелі. Також вказано шляхи підвищення швидкодії та надійності програмного комплексу.

2 МЕТОДИ ТА ПРОГРАМНІ ЗАСОБИ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ УПРАВЛІННЯ СОНЯЧНИМИ ЕЛЕКТРОСТАНЦІЯМИ

2.1 Підходи до реалізації взаємодії компонентів програмних систем

Однією з проблем, яку необхідно вирішити при створенні програмних систем, є забезпечення взаємодії компонентів даної системи. На сьогоднішній день найпоширенішими є наступні підходи до розробки додатків (рис. 2.1):

- Монолітна багатошарова архітектура.
- Сервіс-орієнтована архітектура.
- Мікросервісна архітектура.

Рисунок 2.1 – Основні види архітектури додатків

Кожний спосіб має свої переваги та недоліки та підходить для певних задач. Нижче детальніше розглянуто кожний підхід.

2.1.1 Монолітна багатошарова архітектура

Монолітна архітектура – найпростіший з точки зору реалізації спосіб розробки додатків. Даний підхід передбачає створення єдиної програми, яка працює за принципом поділу відповідальності. Програма розділена на шари [7], кожен з яких виконує певну функцію (рис. 2.2):

a) Шар представлення.

b) Шар бізнес-логіки.

c) Шар доступу до даних.

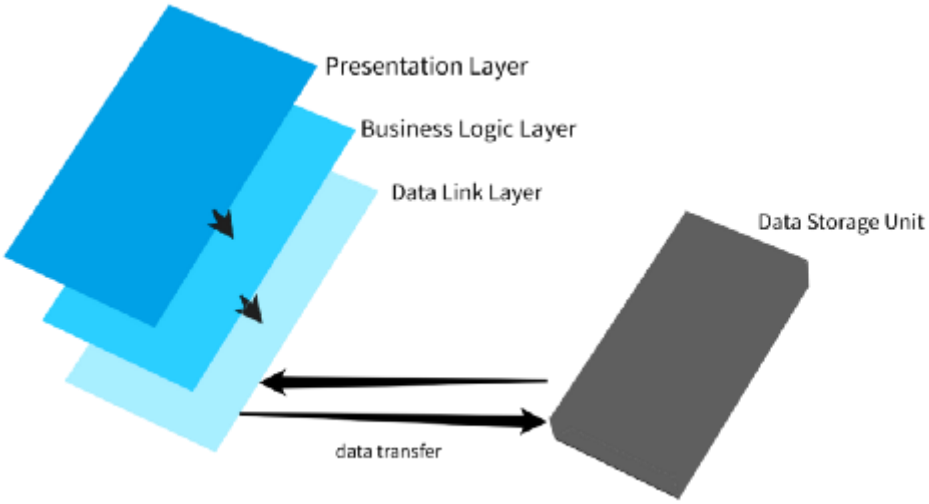


Рисунок 2.2 – Багатошарова архітектура

Шар представлення (Presentation Layer) містить інтерфейс користувача і відповідає за кодування/декодування даних, забезпечення хорошого користувальницького досвіду.

Шар бізнес-логіки (Business Logic Layer) містить систему зв’язків, набір алгоритмів та бізнес-функцій, які задають та описують бізнес-правила реального світу. Він відокремлює UI/UX від обчислень, пов’язаних із основним функціоналом. Це дозволяє легко змінювати логіку залежно від постійно змінюваних бізнес-вимог, не впливаючи на інші шари.

Шар доступу до даних (Data Link Layer) відповідає за взаємодію з базами даних. Він надає зручний інтерфейс для обробки звернень із шару бізнес-логіки.

Серед переваг даного підходу можна виділити наступні:

- Простіша реалізація порівняно з іншими підходами.
- Розподіл відповідальності забезпечує ізолювання та захист одних шарів від зміни інших.

Основні недоліки даного підходу пов’язані з сильною зв’язаністю компонентів системи:

- Ускладнене розуміння коду та внесення модифікацій.
- Ускладнене масштабування, оскільки слід масштабувати всю програму.

Незважаючи на те, що монолітна архітектура має логічну багатошарову структуру, такі програми розгортаються як єдиний моноліт. Вони не мають належної модульності та мають єдину кодову базу. Такий підхід застосовується лише для невеликих однокористувацьких додатків.

2.1.2 Сервіс-орієнтована архітектура

Сервіс-орієнтована архітектура (СОА) — спосіб проєктування, при якому принцип написання програмного забезпечення базується на використанні розподілених по різних вузлах мережі слабко зв’язаних компонентів, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами. Компоненти системи через те що розподілені по різних вузлах можуть пропонуватися як незалежні, слабко зв’язані, замінювані сервіси-застосування [8]. Інтерфейс кожного компонента забезпечує інкапсуляцію деталей реалізації (операційну систему, платформу, мову програмування) даного компонента від інших. Як результат, для побудови складних розподілених програмних систем використання сервісно-орієнтованої архітектури надає гнучкий спосіб комбінування і повторного використання компонентів.

В загальному випадку СОА має три основні елементи: постачальник сервісу, споживач сервісу та реєстр сервісів (рис. 2.3). Взаємодія між цими елементами відбувається наступним чином: реєстр сервісів реєструє всі сервіси, які пропонують постачальники; а споживач звертається до реєстру із запитом.

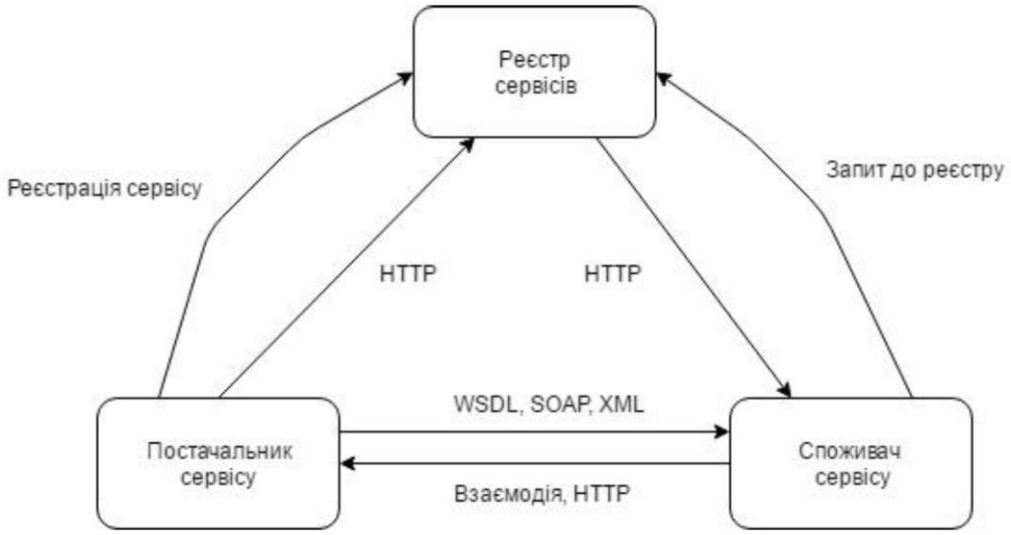


Рисунок 2.3 – Базова схема СОА

При цьому сервіси нічого не знають про внутрішню структуру один одного, вся взаємодія відбувається через інтерфейси за моделлю чорного ящика. Таким чином забезпечується незалежність від технологій розробки та платформ. Наприклад, один і той самий додаток може викликати сервіси, написані на C#, що працюють на платформі .Net і сервіси на Java, що працюють на платформі Java EE. Дана властивість значно полегшує повторне використання компонентів.

Отже, можна виділити основні переваги сервіс-орієнтованої архітектури порівняно з монолітом:

- Можливість часткового розгортання. Оскільки сервіси є незалежними, при зміні одного з них достатньо перерозгорнути лише цей сервіс. У випадку з монолітом необхідно заново збирати та розгортати всю систему.
- Підвищення відмовостійкості. У разі виходу з ладу окремого сервіса зламається лише функціонал, який забезпечувався цим сервісом. Решта сервісів продовжить працювати.
- Відсутність стану. Вся необхідна для обробки інформація передається з поточним запитом, а результат не залежить від минулих запитів.

- Гетерогенність. Оскільки взаємодія відбувається через стандартизовані інтерфейси, різні сервіси можуть використовувати різні мови програмування та різні фреймворки.

Подальшим розвитком даного підходу є мікросервісна архітектура, про яку мова буде йти далі.

2.1.3 Мікросервісна архітектура

Мікросервісна архітектура (МСА) – архітектурний стиль, який передбачає реалізацію програмної системи у вигляді набору невеликих мікросервісів, кожен з яких розгортається та працює в окремому процесі та взаємодіє з іншими мікросервісами, використовуючи легкі механізми, як правило, HTTP (рис.2.4). На відміну від сервісів у СОА, мікросервіс має вузьку спеціалізацію та працює за принципом “Роби одну задачу і роби її якісно” [9].

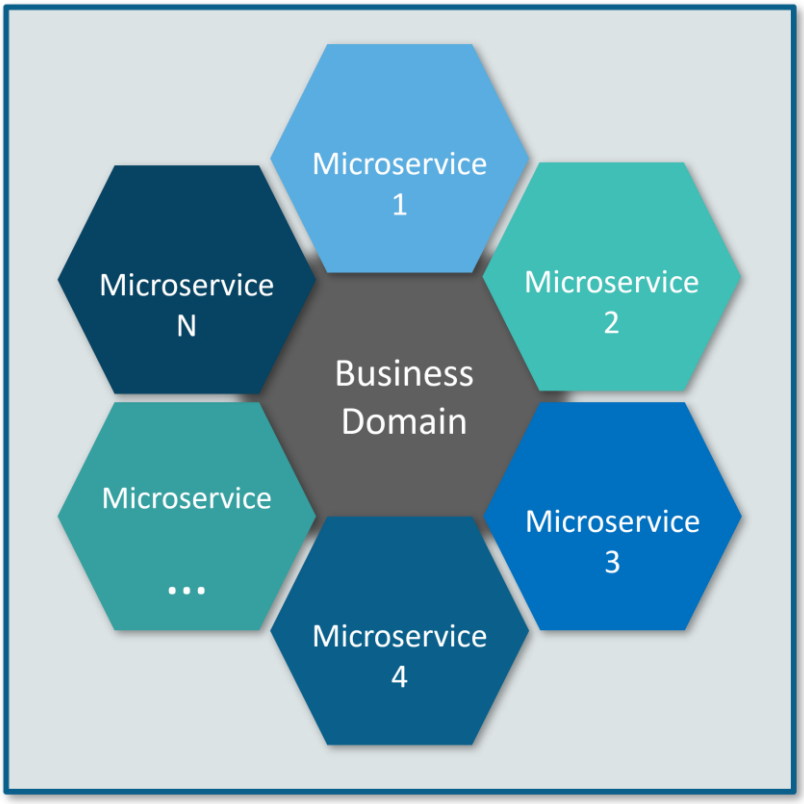


Рисунок 2.4 – Мікросервісна архітектура

Як і SOA, мікросервісні додатки складаються зі слабко зв'язаних, багаторазових і спеціалізованих компонентів, які часто працюють незалежно один від одного. Мікросервіси також використовують високий ступінь згуртованості, інакше відомий як обмежений контекст. Обмежений контекст означає, що компонент і його дані представляються як окрема сутність або одиниця з дуже малою кількістю залежностей.

Основною відмінністю МСА від сервіс-орієнтованої архітектури є сфера застосування (рис. 2.5). СОА зазвичай використовується для створення множини додатків, які взаємодіють через спеціальні протоколи в масштабах підприємства. У свою чергу, мікросервіси спілкуються через інтерфейси прикладних програм (API) і використовуються для створення єдиного додатку, який виконує певну бізнес-функціональність (або функціональність для певних сфер бізнесу). В результаті підвищується його гнучкість, масштабованість та стійкість. Найпопулярнішою мовою програмування для розробки мікросервісів є Java. Також можуть використовуватися інші мови, такі як Golang та Python [10].

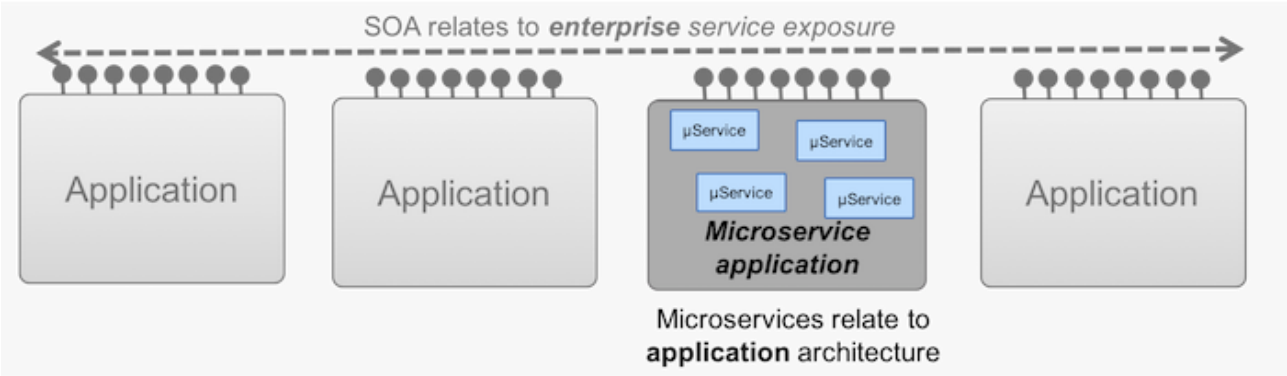


Рисунок 2.5 – Порівняння концепцій мікросервісів та СОА

Мікросервісна архітектура має наступні особливості:

- Слабка зв'язність – мікросервіси всередині системи значною мірою відокремлені. Таким чином, систему в цілому можна легко створювати, змінювати та масштабувати.

- Компонентизація – мікросервіси розглядаються як незалежні компоненти, які можна легко замінити та оновити.
- Бізнес-можливості – мікросервіси дуже прості та зосереджені на одній задачі.
- Автономність – команди розробників можуть працювати незалежно одна від одної, що збільшує швидкість розробки системи.
- Безперервна доставка – дозволяє часті оновлення програмного забезпечення за рахунок систематичної автоматизації створення, тестування та затвердження програмного забезпечення
- Відповідальність – мікросервіси не зосереджуються на додатках як проєктах. Натомість вони розглядають програми як продукти, за які вони відповідають.
- Логіка на кінцевих точках, а не на шляхах – запити не опрацьовуються в процесі передачі. Мікросервіси звертаються напряму один до одного, використовуючи легкі протоколи передачі даних, найчастіше HTTP.
- Децентралізоване управління – ідея полягає у використанні оптимальних інструментів для вирішення задачі. Це означає, що не існує стандартизованого шаблону чи будь-якого технологічного шаблону. Розробники мають свободу обирати найкращі технології для вирішення своїх задач.
- Гнучкість – будь-яку нову функцію можна швидко розробити або відключити.

У результаті додаток являє собою сукупність незалежних компонентів, кожний з яких по суті є монолітним додатком, який виконує єдину функцію. Розглянуті особливості визначають основні переваги мікросервісної архітектури:

- Незалежна розробка – окремі мікросервіси можна легко розробити на основі їх індивідуальної функціональності.

- Незалежне розгортання – будь-який мікросервіс можна розгорнути незалежно від інших. Тому при зміні одного з них не потрібно перезапускати всю систему, інші мікросервіси можуть продовжувати працювати.
- Ізоляція несправностей – навіть якщо один мікросервіс не працює, система продовжує функціонувати. Ламається лише функціонал, який забезпечує даний мікросервіс.
- Незалежність від технологій – при розробці нового мікросервіса можна обрати новий стек технологій. Також будь-який мікросервіс можна переписати з нуля в межах прийнятного часу та бюджету без необхідності перебудовувати всю систему.
- Індивідуальне масштабування – окремі компоненти можуть масштабуватися відповідно до потреб, немає необхідності масштабувати всі компоненти разом. Для індивідуального масштабування використовується балансувач навантаження, який розподіляє запити між вільними екземплярами потрібного мікросервіса (рис. 2.6).

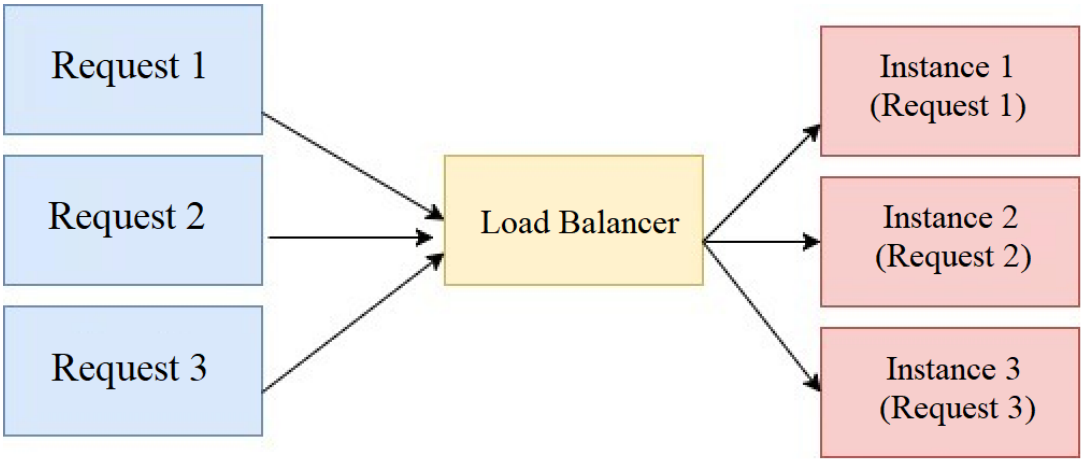


Рисунок 2.6 – Балансування навантаження

Незважаючи на переваги, даний підхід також має свої недоліки. Вони насамперед пов’язані з розподіленим характером мікросервісної архітектури. Основні недоліки даного підходу:

- Складність створення розподіленої системи. Як наслідок, на реалізацію потрібно значно більше часу.
- Великою кількістю сервісів важче керувати.
- Збільшується ризик збоїв при обміні даними між сервісами.
- Необхідно вирішувати проблеми затримок у мережі, балансування навантаження та ін.
- Такі системи потребують комплексного тестування в розподіленому середовищі.

Підсумовуючи вище сказане, можна констатувати, що мікросервісна архітектура найкраще підходить для систем, що динамічно розвиваються. Оскільки у системі, що створюється, однією з вимог є забезпечення масштабованості, для реалізації взаємодії компонентів прийнято рішення використовувати мікросервісну архітектуру.

2.2 Методи зберігання даних

2.2.1 Реляційні СУБД

Реляційна (від англ. relation) база даних — це база даних, інформація в якій зберігається в сукупностях електронних таблиць, зв’язок між якими встановлюється за однаковими значеннями ключових полів.

Реляційна модель даних була запропонована в 1969 році Едгардом Коддом. Це було так званим спрощенням баз даних, у яких дані зберігалися в ієрархічних структурах. Порівняно з ієрархічною системою в реляційних системах завдяки простій структурі робота з даними має більше можливостей. Реляційні системи дещо схожі до файлових систем, але на принципово іншому рівні. Дані в них зберігаються в таблицях, що мають рядки і стовпчики. Робота реляційних систем побудована з використанням математичних механізмів, таких як теорія множин, реляційна алгебра й реляційні обчислення.

Таблиці реляційної бази даних називаються відношеннями. Вони складаються з полів (стовпчиків) та записів (рядків). Кожна таблиця містить один або ряд категорій даних у стовпчиках, та описує деякий об’єкт, або взаємозв'язок між об'єктами. В рядках міститься ключ (унікальний ідентифікатор) для категорій. Таблиці мають ключ для ідентифікації інформації. Зв’язок між ними відбувається за допомогою ключів, що посилаються на ключі інших таблиць.

Дані в таблицях повинні задовольняти наступним вимогам:

- в таблиці не може бути однакових полів;
- кожна таблиця повинна мати унікальне ім’я (первинний ідентифікатор);
- таблиця може мати довільну послідовність записів.

При створенні реляційних баз даних важливо приділити увагу множині відношень, що дозволяють описати об'єкти предметної області та зв'язки між ними. Розглянемо основні елементи реляційної моделі [11]:

a) Таблиця – таблиця з певним набором даних.

b) Сутність – опис властивостей об’єкту, про який зберігаються дані в базі даних.

c) Атрибут – заголовок стовпця таблиці. Це властивості, які визначають відношення або сутність.

d) Кортеж – рядок таблиці, що містить один запис.

e) Домен – можливі значення певного атрибута відношення. Тобто коли кожен атрибут має певне значення та область дії.

f) Схема відношень – назва відношень з іменами атрибутів.

g) Ступінь відношення – загальна кількість атрибутів.

h) Стовпець – набір значень певного атрибуту.

i) Екземпляр відношення – набір кортежів у системі.

j) Ключ відношення – один або декілька атрибутів кожного рядка.

Для коректної роботи необхідно забезпечити таку організацію таблиць, що унеможливить суперечливе або некоректне введення даних. Для цього таблиці необхідно нормалізувати. Основними є три нормальні форми (табл. 2.1). Перша

нормальна форма (1НФ) вимагає, щоб всі атрибути таблиці мають бути простими, а поля не повинні повторюватись. Друга нормальна форма (2НФ) вимагає, щоб неключові поля залежали від усього первинного ключа, а не від його частини. Третя нормальна форма (3НФ) вимагає, щоб неключові поля не залежали від інших неключових полів даної таблиці.

Таблиця 2.1. Критерії нормальних форм

Нормальна форма	Критерій відповідності
1НФ	Кожна комірка містить єдине значення
2НФ	Немає полів, які залежать від частини ключа
3НФ	Кожне поле нетранзитивно залежить від ключових

Існує три типи зв’язків між таблицями:

a) Один до одного – одному рядку з таблиці А відповідає тільки один рядок у таблиці В. Таке відношення створюється, коли обидва поля є ключовими, або мають унікальні індекси.

b) Один до багатьох – одному рядку з таблиці А відповідає кілька рядків у таблиці В. Відношення створюється, коли тільки одне з полів є полем первинного ключа або унікального індексу.

c) Багато до багатьох – пов’язує декілька рядків з таблиці А з декількома рядками таблиці В. В реляційних СУБД таке відношення реалізується через два відношення “один до багатьох” з проміжною таблицею С. Первинний ключ такої таблиці складається з полів зовнішнього ключа таблиць А та В.

Для взаємозв’язків між таблицями використовуються ключі. Ключ представляє собою стовпець чи декілька стовпців, за допомогою яких встановлюється зв’язок з іншою таблицею. Ключі можуть бути первинними і вторинними.

Первинні ключі завжди мають унікальний індекс і не можуть бути порожніми. Це поля, значення яких визначає кожний запис в таблиці. Первинний

ключ використовується для зв’язування однієї таблиці з вторинними ключами інших таблиць. Існує три типи таких ключів: лічильник, простий ключ і складений ключ. Поле лічильника автоматично заповнюється унікальним числовим значенням. Якщо поле не містить повторювані значення, тобто всі поля унікальні і мають значення кодів або індивідуальних номерів, такі поля є простими ключами. Якщо можливо створити ключ з декількох полів або такі поля можуть бути не унікальними, то такий ключ вважаємо складеним. Складені ключі найчастіше виникають в таблиці зі зв’язками “багато до багатьох” [12].

Вторинний ключ також іноді називають зовнішнім ключем. Він являє собою стовпчик або декілька з посиланням на поле первинного ключа іншої таблиці. За допомогою зовнішніх ключів визначається спосіб об’єднання таблиць. Якщо розглядати дві зв’язані таблиці, то одна буде головною таблицею або таблицею первинного ключа, а інша – вторинною або таблицею вторинного ключа. Вторинна таблиця є підпорядкованою до головної таблиці.

Реляційні бази даних досить популярні в усьому світі. На сьогодні найпоширенішими реляційними СУБД є:

- SQLite – система управління базами даних з відкритим кодом. У цій СУБД всі дані зберігаються в одному файлі, завдяки чому вони можуть зберігатися локально без прив’язки до сервера. Прикладом застосування є використання SQLite в телевізійних приставках, смартфонах та інших пристроях [13].
- PostgreSQL також має відкритий код, досить бюджетна, надійна та проста. СУБД досить популярна серед багатьох розробників завдяки тому, що не вимагає складних налаштувань. Використовується для розробки веб-програм. На відміну від інших подібних СУБД, вона працює дещо повільніше [14].
- MySQL – також популярна реляційна СУБД, як і інші має відкритий вихідний код. До переваг насамперед слід віднести бюджетність, надійність, простоту використання. Використовується у веб-додатках, може бути

доступною за допомогою PHP. Також вона оптимізована для хмарних обчислень [15].

У даній роботі для зберігання даних використовується СУБД MySQL.

2.2.2 Документо-орієнтовані СУБД

Документо-орієнтована система керування базами даних (англ document-oriented database) – це система зберігання ієрархічних структур даних. Документо-орієнтовані бази добре зарекомендували себе при обробці та аналізі великих об’ємів даних. Основна перевага таких систем над традиційними реляційними системами керування базами даних полягає у можливості горизонтального масштабування без додаткових витрат, що дозволяє використовувати паралельну обробку даних. Недоліками можна назвати складність адміністрування та відсутність універсальної мови запитів (як наприклад SQL для реляційних баз даних), унікальність кожної системи, що підвищує вартість розробки прикладних програмних продуктів та ускладнює міграцію з одних документо-орієнтованих баз даних на інші. Досить часто документо-орієнтовані бази даних застосовуються разом із реляційними базами даних для використання переваг обох архітектур і нівелювання їх недоліків [16].

Основною відмінністю ДОБД від реляційних баз даних полягає в тому, що вони оперують документами. Рядки в реляційних базах і документи всередині подібні, але від записів в ДОБД не вимагається мати однакові поля або ключі. Також відсутня схема організації даних. Це означає, що:

- структура сутностей неоднорідна, в різних стовпцях містяться різні записи;
- кожний запис може мати різні типи значень
- колонки можуть бути представлені у вигляді масивів, тобто мати більше одного запису;
- у будь якому записі може бути вкладена структура.

Поняття «ключ» також використовується в ДОБД. Ключем може бути як рядок, так і шлях. Ключ використовується для отримання документу з бази даних. Пошук в базі даних відбувається як за допомогою ключа, так і за допомогою програмного інтерфейсу (API) або мови запитів. Це дозволяє користувачеві шукати той чи інший документ.

Документо-орієнтовані БД часто використовують внутрішні представлення даних, найчастіше JSON, які можуть бути опрацьовані безпосередньо в додатках. Документи зберігаються як одно-, дво- або багаторівневі структури типу ключ-значення з необмеженою складністю таких структур. JSON-документи можна зберігати і в колонках реляційних баз даних у вигляді текстового представлення, у вигляді ключ-значення або інших внутрішніх структур даних. Незважаючи на простоту такого використання, подібний варіант зберігання JSON-документів в реляційних системах управління базами даних має певні недоліки: горизонтальне масштабування значно ускладнене або відсутнє, найчастіше відсутні можливості, які надаються сховищами документів (вторинні індекси та посилання на інші документи), вся обробка документів виконується на клієнтській стороні. Сховища документів в свою чергу також мають певні недоліки, такі як ускладненість транзакцій, неможливість отримання окремих полів документу (клієнт отримає весь документ, а це може вплинути на об’єм переданих даних та швидкість обробки), ускладненість оновлення даних. Натомість пропонується висока продуктивність сховищ документів, швидкість читання документів, можливість обробки даних на сервері та вбудовані можливості горизонтального масштабування.

Документо-орієнтовані бази даних зберігають інформацію як набір непов’язаних між собою документів. При цьому можна знайти аналогії до табличного зберігання даних у реляційних базах даних.

Найпоширенішими сферами застосування документо-орієнтованих баз даних є:

- ущільнення інформації – документи в сховищі даних дозволяють працювати зі складними структурами з багаторівневими вкладеннями.
- інтеграція з системами на основі JavaScript – сховища документів легко інтегруються з системам, які використовують представлення даних JSON.

Існує декілька найвідоміших ДОБД, таких як:

- CouchDB – працює з напівструктурованою інформацією, відсутній опис схеми даних. Дані зберігаються у вигляді JSON – подібних документів. Завдяки тому, що програма вільна і відкрита, її використовують на великій кількості веб-сайтів та в більшості програмних продуктів [17].
- MongoDB – лідер на ринку серед NoSQL рішень. Дана документо-орієнтована система отримала значне поширення також завдяки тому, що дані всередині документів зберігаються у схожому на JSON форматі, система має відкритий код, написаний на мові C++. При використанні не потрібно описувати схеми таблиць. На етапі розробки є можливість створювати як різноманітні зв'язки між документами, так і вбудовані документи. Для звернень до бази даних використовуються досить гнучкі ad-hoc-запити, які повертають поля документів та користувацькі JavaScript-функції. За допомогою техніки сегментування об'єктів та розподілу їх частин на різних вузлах система горизонтально масштабується. Систему можна використовувати також як файлове сховище. Дана функція надається разом з драйверами [18].
- CouchBase – гнучка розподілена документо-орієнтована база даних, націлена на використання оперативної пам'яті. В ОЗП переміщуються JSON-документи, метадані та індекси, що найчастіше використовуються в запитах. Подібна гібридна система зберігання та обробки даних є своєрідним поєднанням двох різних концепцій зберігання документів: MemBase, де всі документи зберігаються в ОЗП, та CouchDb, де документи зберігаються на жорстких дисках. Це дозволяє отримати високу швидкість доступу за малою вартістю системи [19].

Розглянемо шість основних концепцій документо-орієнтованих баз даних на прикладі MongoDB.

a) MongoDB як сховище даних подібна до реляційних баз даних. Всередині MongoDB може бути нуль або декілька баз даних, кожна з яких є контейнером для інших об'єктів.

b) База даних може мати нуль або більше колекцій (рис. 2.7). Колекції можна розглядати як таблиці реляційних баз даних через їхню схожість.

c) В свою чергу, колекція містить нуль або більше документів, що є прямою аналогією до рядків SQL-таблиць.

d) Відповідно, складовими документу є одне або декількох полів, які відповідають стовпчикам таблиці в SQL.

e) Індекси MongoDB виконують ту саму роль, що і індекси реляційних базах даних.

f) Курсори MongoDB виконують роль не лише результату пошуку в реляційних базах даний, вони також дозволяють керувати спеціальними можливостями виконання запиту (наприклад, використовувати чи не використовувати файлову систему для зберігання результату запиту).

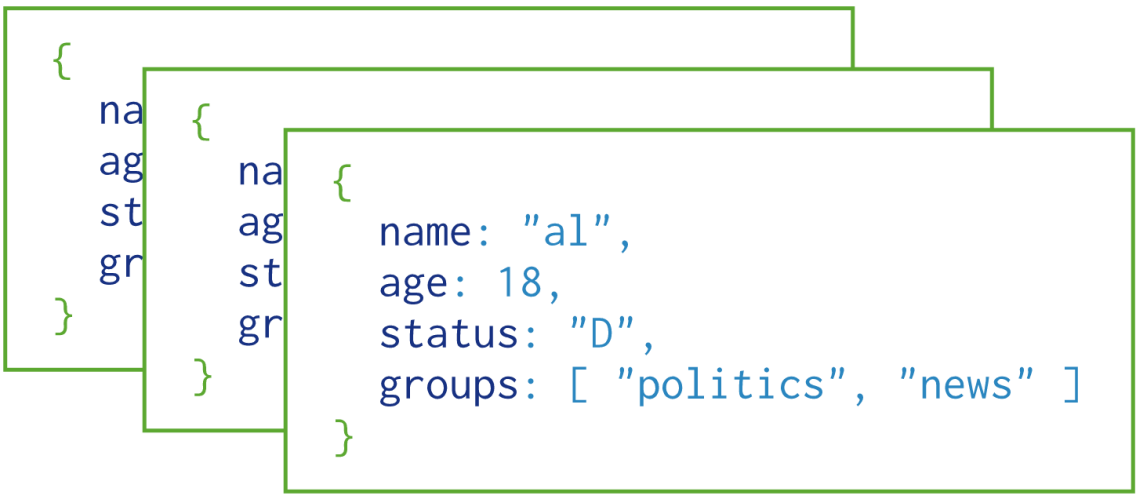


Рисунок 2.7 – Колекція MongoDB

Важливою є відмінність, що результатом запиту є покажчик, за допомогою якого можна виконувати розрахунки або пропускати певну кількість попередніх записів без завантаження даних на клієнт.

Документо-орієнтовані СКБД базуються на документних сховищах (англ. document store), зі структурою дерева (іноді лісу). Структура дерева містить кореневий вузол як початковий елемент та може містити кілька внутрішніх і листових вузлів. Листові вузли зберігають дані, які при додаванні документа додаються в індекси. Це дозволяє знаходити потрібні документи за їх шляхом в дереві незалежно від складності самого дерева. API пошуку документів надає можливість отримувати за запитом як повні, оригінальні документи, так і їх частини. Так, результат запиту до документного сховища може містити оброблені документи без необхідності завантаження оригіналів цих документів клієнтською програмою.

Колекції можуть містити документи (що споріднює колекції та таблиці реляційних баз даних) та інші колекції. Незважаючи на те, що колекції можуть містити документи будь-якої внутрішньої структури, для ефективнішого індексування бажано об’єднувати в колекції документи з однаковою або схожою внутрішньою структурою.

ДОБД можна сказати є альтернативою реляційних баз даних. Існують деякі відмінності в роботі, але, загалом, дана система дозволяє робити все те, що роблять інші бази даних.

До переваг ДОБД можна віднести:

- При роботі з великим об’ємом даних продуктивність набагато вища порівняно з реляційними базами даних, в яких використовуються SQL.
- Масштабування відбувається легше, ніж в SQL рішеннях.
- Децентралізовані.
- Завдяки тому, що схеми організації даних не потребують певних обмежень, легко зберігаються неструктуровані дані, а також при зміні схеми

даних не потрібно додавати нові поля або додатково оновлювати всю систему.

- Вся інформація для певного об’єкту зберігається в одному місці.
- Інтерфейс запитів до бази даних порівняно простий.

Поряд з перевагами існують і деякі недоліки ДОБД. Перш за все необхідно зазначити відсутність логіки транзакцій. Через те, що записи можуть не мати ключі, існує проблема контролю цілісності і це необхідно реалізовувати в додатковій логіці. Проте спеціалізована логіка в деяких випадках буває ефективнішою за алгоритми реляційних баз даних.

Також для побудови запитів до бази даних необхідні знання додаткової мови програмування. Але враховуючи, що запити не складні, більшість програмістів зможе опанувати цю мову.

2.3 Методи паралельного програмування

2.3.1 Основні поняття паралельного програмування

Процес у програмуванні – контейнер системних ресурсів, які виділяються для забезпечення виконання програми. Кожний процес має власний захищений віртуальний адресний простір (ВАП) – діапазон адрес віртуальної пам’яті, до яких він може звертатися. ВАП використовується для ізоляції процесів один від одного. Таким чином, інші процеси не мають доступу до пам’яті, яка виділена для даного. Існує два типи ресурсів, необхідних для успішного виконання програми:

a) Процесорний час – основний ресурс, необхідний для послідовного виконання програмного коду.

b) Ресурси пам’яті – для збереження інформації, необхідної для виконання програмного коду. До них належать регістри процесора, оперативна пам'ять та ін. Для забезпечення цих ресурсів використовується віртуальний адресний простір процесу.

Основним елементом процесу є потік – набір команд процесора, які виконуються послідовно і використовують адресний простір процесу. Потік є базовим об'єктом, якому операційна система виділяє процесорний час. Сам по собі процес не виконує ніяких дій, для цього призначені потоки. Усі потоки одного процесу спільно використовують його віртуальний адресний простір для збереження контексту виконання в той час, коли процесор перемикається на інший потік. У контекст входять регістри процесора, змінні оточення, стеки ядра і користувача.

Процес обов’язково має хоча б один потік. Виконання процесу починається зі стартового потоку, далі за необхідності він може створювати інші потоки. У багатопроцесорних системах окремі потоки можуть виконуватися на окремих процесорах. Таким чином, процес являє собою сукупність одного або декількох потоків і захищеного простору адрес, у якому ці потоки виконуються [20].

2.3.2 Багатопотоковість

Багатопотоковість – можливість одночасно запускати та виконувати декілька потоків всередині одного процесу. У цьому випадку потоки мають доступ до спільного простору адрес свого процесу, і таким чином заощаджують пам’ять комп’ютера (рис. 2.8). Крім того, переключення контексту між ними відбувається в найкоротші терміни. Багатопотоковість підтримується багатьма мовами програмування, серед яких C++, Java, Python та інші.

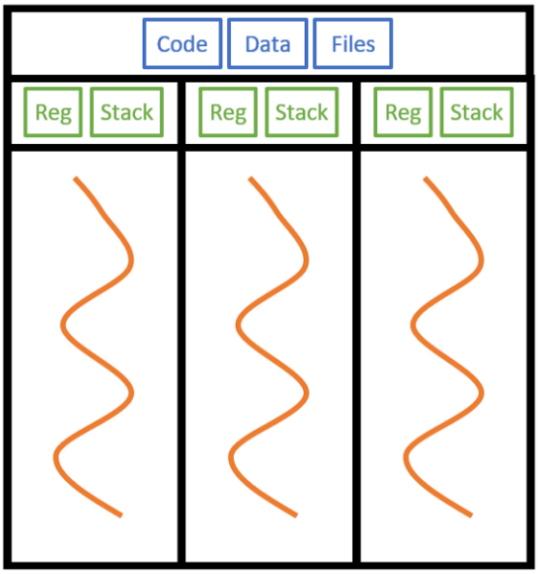


Рисунок 2.8 – Потоки всередині процесу

На рисунку бачимо, що кілька потоків спільно використовують код, дані та файли, але виконуються в своєму регістрі та стеку. Даний підхід робить можливим одночасне виконання двох або більше частин програми. Завдяки цьому додаток максимально використовує процесорний час, а час простою зводиться до мінімуму. Багатопотоковість має наступні переваги, які підвищують ефективність роботи програм:

a) Більш ефективне використання процесора за рахунок розпаралелювання обчислень та операцій введення/виведення.

b) Час створення потоку та час переключення між потоками менші, ніж у процесу.

c) Простота реалізації за рахунок використання спільного адресного простору, оскільки не потрібно організовувати міжпроцесну взаємодію.

Багатопотоковість ідеально працює, коли кожний потік використовує лише певні змінні. Наприклад, при множенні матриць потоки не змінюють початкові дані, і при цьому кожен потік записує результат обчислень у чітко визначений для нього діапазон комірок. Це дозволяє оптимізувати час виконання операції множення за рахунок збільшення потоків і уникнути конфліктів доступу до розподіленого ресурсу – у даному прикладі комірок пам’яті. Збільшення

продуктивності (швидкості обчислення) при збільшенні кількості потоків є нелінійним, і завжди існує межа, за якою додавання ще одного потоку призведе до гальмування процесу обчислення через фізичні обмеження розподіленого ресурсу. Такими обмеженнями можуть стати швидкість та кількість каналів доступу до оперативної пам’яті; обмеження файлової систем; обмеження під’єднаних фізичних пристроїв, таких як адаптери жорстких дисків або мережеві адаптери; обмеження ресурсів, які накладає менеджер віртуальних машин у системах віртуалізації тощо.

У даній роботі багатопотоковість використовується для одночасного обчислення оптимального положення декількох сонячних панелей.

2.3.3 Багатопроцесність

Багатопроцесність – це можливість одночасного виконання декількох процесів на одному фізичному комп’ютері. На відміну від потоків усередині процесу, які порівняно легко можуть комунікувати між собою, процеси є ізольованими середовищами і комунікація між окремими процесами зазвичай обмежена використанням черг (queue), тунелів (pipe), семафорів. За рахунок ускладнення реалізації паралельного програмування і ізолювання процесів один від одного, багатопроцесність дозволяє створювати системи, стійкі до збоїв – в яких збій в роботі одного з процесів не вплине на роботу решти (рис. 2.9). Наприклад, реалізація процесів моніторингу може включати в себе зупинку/припинення користувацьких процесів в залежності від навантаження на процесори, об’єм переданого мережевого трафіку, іншу підозрілу діяльність, пов’язану з зараженням вірусами. Багатопроцесність дозволяє безпечно працювати сотням і тисячам контейнерів користувачів у системах віртуалізації Kubernetes / Docker за рахунок ізольованості контейнерів один від одного.

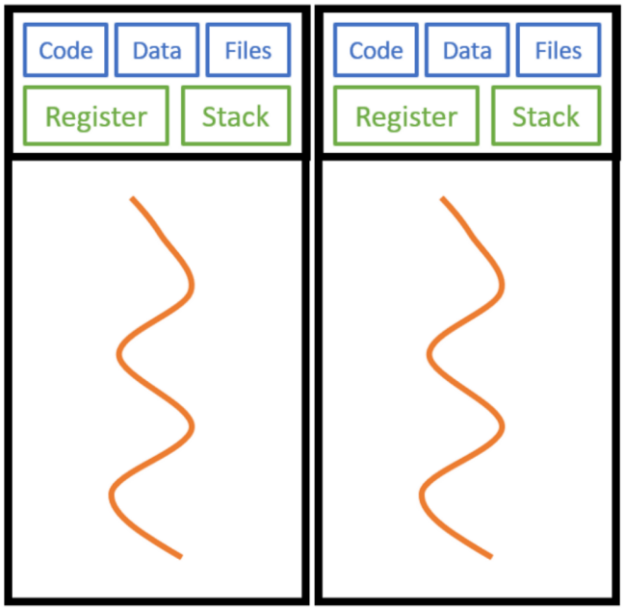


Рисунок 2.9 – Багатопроцесність

Через те, що процес є повністю ізольованим середовищем, це дозволяє не лише використовувати декілька процесорів на одному фізичному комп'ютері, а і організувати розподілені обчислення між декількома комп'ютерами або обчислювальними блоками. Прикладами систем з десятками тисяч комп'ютерів можуть бути розподілені обчислення, пов'язані з обробкою великих об'ємів даних, таких як дослідження ДНК, пошук ліків проти раку та СНІДу, обробка відкритих джерел астрономічних спостережень для пошуку потенційно небезпечних для нашої планети астероїдів. У таких системах основною метою багатопроцесності та розподіленої багатопроцесності є пришвидшення обчислень. Іншим прикладом є системи, в яких ізоляція та розподіл процесів між декількома процесорами або обчислювальними блоками викликана вимогами безпеки – так, обчислювальні блоки сучасного автомобіля пов'язані в єдину систему, проте збій у процесі, відповідальному за керування мультимедійною системою не має вплинути на критично важливі процеси відповідальні за керування двигуном, кермування або гальмування.

Багатопроцесність також має нелінійну криву швидкості роботи від кількості запущених процесів. Головним чинником, як і для потоків, буде конкуренція між

процесами за критично важливий розподілений ресурс. Для задач з інтенсивним використанням процесорного часу критичним буде відношення кількості процесорів до кількості запущених процесів – коли кількість процесів перевищить кількість процесорів, відбудеться зменшення швидкості обчислень. Для задач, пов'язаних з обробкою візуального контенту важливим може стати швидкість випадкового доступу до файлової системи і т. ін.

Дана робота безпосередньо не використовує багатопроцесність для пришвидшення обчислень, проте застосування багатопроцесності тут може бути продиктоване питаннями надійності та безпеки – так, додавання системи моніторингу/перезапуску основного процесу або системи автоматичного оновлення програмного продукту потрібно буде реалізовувати з урахуванням багатопроцесності з огляду на те, що збій основної програми не має викликати збій систем оновлення та моніторингу.

2.4 Засіб візуалізації даних Flutter

Flutter — це фреймворк для розробки мобільних і веб додатків, створений Google. Цей інструмент дозволяє створювати як веб-додатки, так і мобільні додатки для пристроїв на операційних системах Android та iOS. Таке використання єдиного коду має глибокий вплив на розробку мобільних додатків. Крім того, кросплатформний додаток допомагає заощадити бізнес-ресурси, а також дозволяє додаткам підтримувати однакову функціональність. Flutter, певним чином, зробив революцію на ринку мобільної розробки. В той час, як головне рішення конкурента — React Native — є бюрократичним і складним, Flutter пропонує простіший підхід, що надає значні переваги для розробників [21]. Отже, основними перевагами даного фреймворку є:

a) Спрощення та пришвидшення процесу розробки.
b) Єдина мова для розробки.
c) Спрощене початкове налаштування.
d) Естетична та функціональна сумісність.

e) Швидкість виконання програми.

f) Служба підтримки Google.

Детальніше розглянемо кожний пункт.

Першою особливістю Flutter є швидкість, яку він надає розробнику. Забезпечуючи систему віджетів для розробки, набагато легше створювати інтерактивні елементи під час створення програмного забезпечення. Таким чином забезпечується гнучкість і висока якість кінцевого продукту у версіях для Android та iOS. Крім того, опція «Hot reload» дозволяє легко переглядати зміни, внесені в код, забезпечуючи миттєвий зворотний зв'язок з розробником і, як наслідок, підвищуючи продуктивність.

Flutter робить ставку на мову Dart як єдину мову для створення своїх програм. Таким чином, це значно спрощує роботу розробників, яким для виконання своїх завдань достатньо знати Dart. Ця мова також розроблена Google у 2011 році і за своїми характеристиками коду дуже схожа на C# та Java. Іншими словами, для розробників, які звикли до вищезгаданих мов, адаптація до Dart не є складним завданням.

Установка Flutter дуже проста і практична. Щоб почати роботу, достатньо лише встановити IDE для розробки — доступну для всіх операційних систем — і створити проєкт з нуля. Немає складних залежностей чи конфігурацій, щоб побачити, як працюють перші коди. Ця функція важлива, щоб розробники могли швидше почати роботу з цим фреймворком.

Flutter пропонує власні віджети, які забезпечують функціональну та естетичну сумісність. Під час використання програми, створеної за допомогою Flutter, одна і та ж версія поширюється на будь-який мобільний пристрій, незалежно від того, встановлено останню версію операційної системи чи ні. Тим не менш, зовнішній вигляд елементів буде однаковим на усіх згаданих системах. У випадку ж з нативними додатками, якщо операційна система пристрою має версії з візуальними змінами елементів (наприклад, кнопок або меню), вони будуть виглядати інакше.

Велика проблема з іншими інструментами, які створюють версії програми для різних операційних систем, — це продуктивність виконання. Однак фреймворк Flutter вирішив цю проблему. Оскільки він використовує власні компоненти, час завантаження та виконання додатків не є проблемою для користувача. Іншими словами, для користувача програми, створені за допомогою Flutter, майже не відрізняються від нативних програм.

Важливим моментом для розробників є повна підтримка Google. У цьому сенсі мається на увазі енергійна та активна спільнота, яка постійно працює над удосконаленням інструменту. Крім того, документація Flutter досить повна як для новачків, так і для досвідчених програмістів.

Виходячи з наведених переваг, ми бачимо, що Flutter є ідеальним інструментом для компаній, які хочуть бути більш продуктивними та рентабельними у створенні програм. Оскільки компанія Google зацікавлена в тому, щоб підтримувати мобільну спільноту, яка створює та розвиває програмні рішення, цілком природно запропонувати найкращі ресурси для цього.

2.5 Засіб безперервної доставки Git

2.5.1 Загальний огляд систем контролю версій

Система керування версіями (система контролю версій, СКВ) – програмний засіб для збереження різних версій певного проєкту. Контроль версій може застосовуватися до будь-якого типу файлів, не лише до програмного коду. СКВ дозволяє команді розробників одночасно працювати над груповими проєктами, не заважаючи один одному: кожний працює на своїй гілці над своїм функціоналом, потім зміни додаються в загальну кодову базу (рис. 2.10). Завдяки цьому забезпечується безперервна доставка програмного продукту.

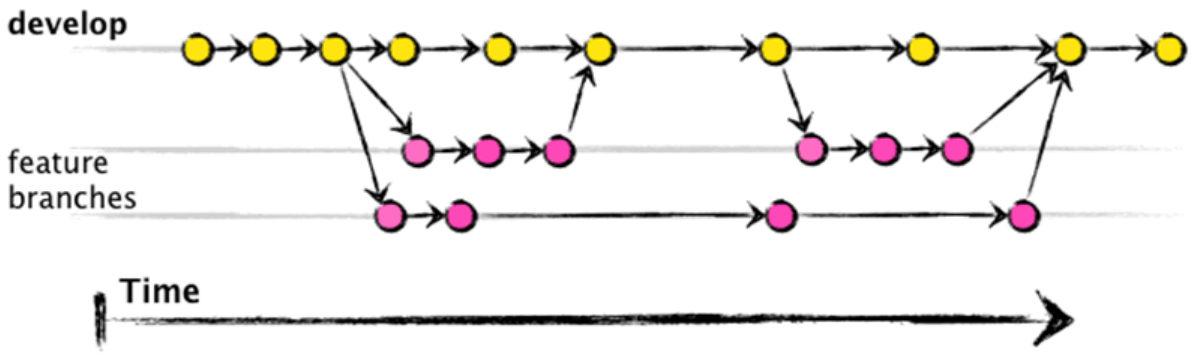


Рисунок 2.10 – Схема роботи СКВ

Системи контролю версій надають наступні можливості:

- дозволяють повернути весь проєкт або вибрані файли до деякого попереднього стану;
- дозволяють порівняти зміни з деяким станом;
- дозволяють визначити, які зміни спровокували проблему, хто автор цих змін та ін.

Таким чином, при поломці деякого функціоналу або втраті файлів є можливість легко це виправити. Крім того, використання систем контролю версій не потребує накладних витрат. На сьогодні СКВ широко використовуються не лише для командних, а й для індивідуальних проєктів. За типом організації контролю версій існує три типи СКВ (рис. 2.11): локальні, централізовані та децентралізовані [22].

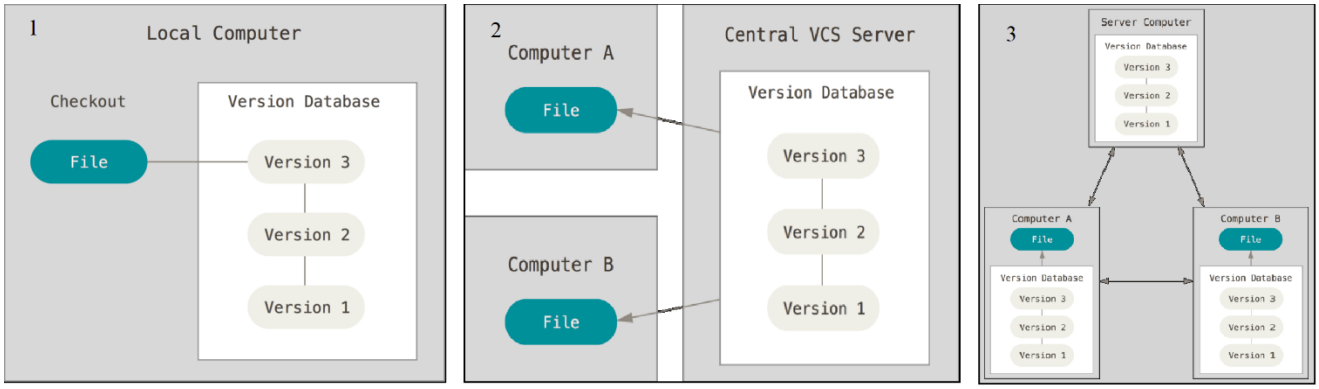


Рисунок 2.11 – Типи систем контролю версій

Локальні СКВ (1) мають просту базу даних, в якій зберігаються всі версії файлів. Замість того, щоб зберігати повну копію документу (він може бути громіздким), такі системи зберігають лише внесені зміни, які називають “ латки”. Стан файлу на конкретний збережений момент часу можна повністю відтворити шляхом додавання латок.

Наступним етапом розвитку стали централізовані СКВ (2). Такі системи мають єдиний сервер, на якому зберігаються всі версії файлів, та деяке число клієнтів, які отримують файли з центрального сховища. У цьому випадку кожному учаснику відомо, певною мірою, чим займаються партнери. Також централізовані СКВ легше адмініструвати порівняно з локальними базами даних кожного учасника. Проте вони також мають недоліки: якщо центральний сервер виходить з ладу, то протягом цього часу учасники не можуть зберігати зміни в системі та отримувати оновлення коду. Також у випадку пошкодження центральної бази даних уся історія змін проєкту може бути втрачена. Загалом ця проблема актуальна для локальних та централізованих систем контролю версій: якщо вся історія проєкту зберігається в одному місці, ви ризикуєте втратити все.

Для подолання цього недоліку вирішили об’єднати два розглянуті підходи: клієнти не просто отримують останній знімок файлів репозиторію, натомість вони отримують повну копію сховища з усією історією змін (3). У цьому разі, навіть якщо дані на сервері втрачено, будь-який клієнт може скопіювати свій репозиторій на сервер і відновити історію змін. На сьогоднішній день даний підхід є стандартом для систем контролю версій. До цього типу відноситься СКВ Git.

2.5.2 Система контролю версій Git

Система контролю версій Git — це програма для відслідковування змін в будь-яких файлах, яка широко використовується для кооперації розробників програмного забезпечення, що працюють над спільним програмним кодом. Git є безкоштовним ПЗ з відкритим вихідним кодом, і призначений для швидкої та ефективної роботи з проєктами будь-якого масштабу, від малих до дуже великих.

Початковим автором даної СКВ є Лінус Торвальдс – творець ядра операційної системи Linux. Git застосовується для керування версіями в рамках колосальної кількості проєктів. Ліцензія GPL-2.0-only яка використовується для розповсюдження даної СКВ не обмежує використання даного ПЗ лише проектами з відкритим вихідним кодом, але й дозволяє використання в закритих та комерційних проєктах. Портована на всі сучасні операційні системи та підтримувана безліччю інтегрованих середовищ розробки (IDE) СКВ Git де-факто є основною системою контролю версій для професійних розробників програмного забезпечення.

Головною відмінністю від популярних у минулому централізованих систем на кшталт CVS і Subversion (SVN), в яких файлове сховище з повною історією змін розташоване в єдиному центрі, Git має розподілену архітектуру, в якій кожен клон сховища є повноцінним і автономним репозиторієм. Завдяки цьому всі розробники зберігають історію змін у повному обсязі [23]. Дана СКВ перевершує інші інструменти завдяки наступним особливостям:

a) Висока продуктивність.
b) Безпека.
c) Гнучкість.

Детальніше розглянемо кожний пункт.

По-перше, в Git оптимізовані процедури фіксації комітів, операції відгалуження від основної лінії розробки, інтеграції змін в основну лінію розробки, перегляд історії версій файлів. Алгоритми визначення змін у файлах побудовані з урахуванням реальних дерев файлових систем вихідного коду та динаміки змін вмісту файлів.

Деякі СКВ у якості ідентифікатора, до якого прив’язується вся історія версій файлу, використовують ім’я або шлях до файлу. Це унеможливлює відслідковування точної історії змін переміщених або перейменованих файлів. На відміну від них Git аналізує вміст, що підвищує продуктивність роботи і дозволяє не втрачати історію змін у випадку інтенсивного перейменування або переміщення

файлів. Усі внутрішні об’єкти репозиторію Git стиснуті дельта-кодуванням. Ці об’єкти зберігають зліпки каталогів та метадані версій, що забезпечує високу продуктивність і високу компресію даних.

По-друге, Git гарантує цілісність вихідних файлів. Так вміст кожного файлу, каталогу, внутрішніх об’єктів репозиторію, історія версій, теги, коміти захищені за допомогою підпису з використанням стійкого криптографічного алгоритму хешування SHA-1. Це дозволяє захистити дані від випадкового або навмисного пошкодження, а також дозволяє відслідкувати історію змін у повному обсязі. Таким чином гарантується автентичність історії змін коду.

По-третє, система Git розрахована насамперед на створення гілок і використання тегів. Тому процедури за участю гілок та тегів (наприклад, об'єднання та повернення до попередньої версії) зберігаються в історії змін. Даний підхід забезпечує підтримку нелінійних циклів розробки, ефективність використання з малими та великими проєктами, а також сумісність з багатьма системами та протоколами.

2.6 Структура бібліотеки для паралельного обчислення оптимального положення панелей

2.6.1 Опис моделі

Програмне забезпечення станції використовує наступні поняття (рис. 2.12):

а) Станція – сонячна електростанція зі стандартним набором готового обладнання, яка належить конкретному власнику. На кожній станції встановлюється програмне забезпечення, яке має унікальний ідентифікатор. У базі даних зберігається інформація про належність станцій користувачам. На відміну від прототипу, у одного користувача може бути багато станцій. Станція містить інформацію про панелі, кількість накопиченої енергії та стан перемикачів.

Енергія з сонячних панелей потрапляє на акумулятор, потім з нього передається в мережу. Перший перемикач відповідає за подачу енергії в мережу. Якщо його розімкнути, можна накопичувати енергію. Другий перемикач відповідає за живлення панелей. Якщо панелі централізовано знеструмити, то вони перестають виробляти енергію, але зберігається можливість продавати в мережу раніше накопичену енергію. Кожна панель при цьому залишається підключеною або відключеною, як була до знеструмлення.

b) Панель – сонячна панель, яка прив'язана до конкретної станції. На кожній станції може бути встановлено декілька панелей. Панелі, що встановлені на станціях можуть мати стан «під'єднана» або «не під'єднана». У стані «під'єднана» панель генерує енергію з певною потужністю та може коригувати своє положення відносно Сонця з встановленою періодичністю. Якщо стан панелі «не під'єднана», тобто знеструмлена, то панель нічого не генерує, але зберігає своє положення.

c) Стан – описує можливі положення конкретної панелі, визначає рекомендовану дію, наприклад: повернути на один градус правіше, лівіше, вниз, вгору; не змінювати положення (при цьому вважаємо, що положення відносно Сонця оптимальне). Визначається ідентифікатором панелі, її азимутом та висотою. Знаходження в певному положенні у кожний момент часу – це і є знаходження у деякому стані.

Наприклад, станція має 10 панелей, при цьому панель номер 3 знаходиться на тривалому ремонті. Власник вирішив провести плановий огляд станції. Для цього він на короткий час централізовано знеструмив усі панелі. Огляд виявив відхилення у панелі номер 7, її від'єднали. Після огляду централізовано увімкнули панелі. Панелі 3 та 7 залишилися відключеними.

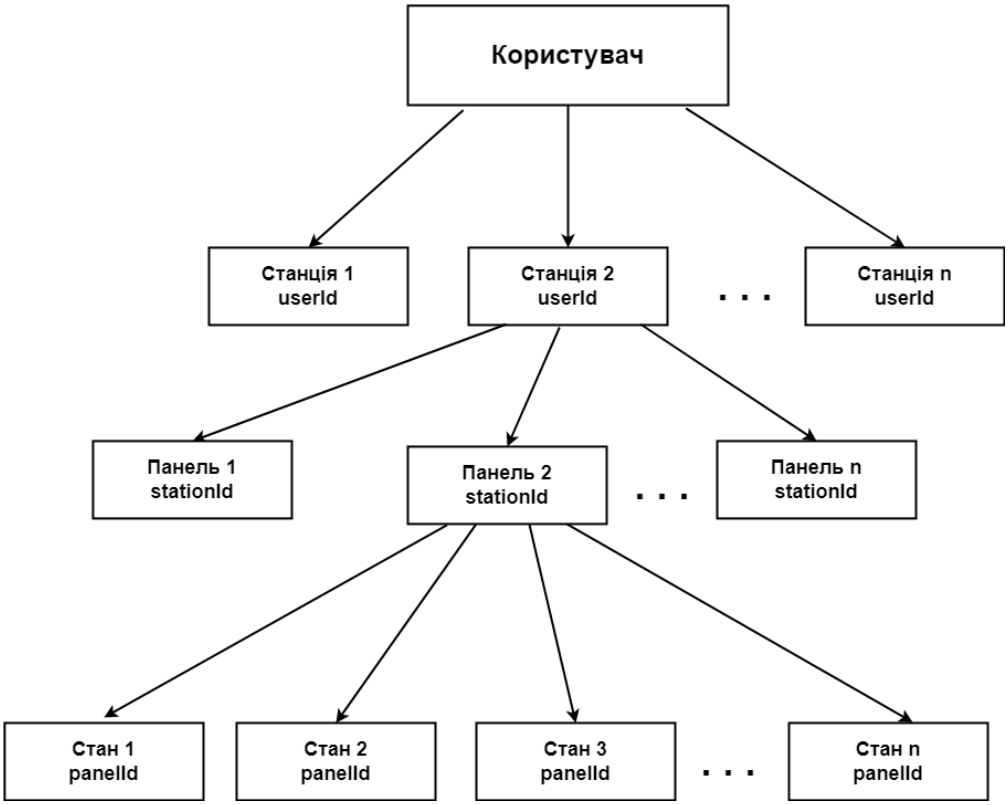


Рисунок 2.12 – Модель сонячної електростанції

Під час роботи станція із заданою періодичністю коригує положення увімкнених панелей так, щоб потужність панелі була максимальною в кожний момент часу. Для цього станція фіксує стан, у якому знаходиться дана панель. Для кожного стану панелі існують допустимі дії та значення корисності для них. Алгоритм у 95 % випадків обирає дію за найбільшою корисністю, у решті випадків – обирає дію випадковим чином.

Корисність дії – суб’єктивне поняття. Під час роботи алгоритм сподівається отримати винагороду, яка визначається збільшенням потужності панелі. При цьому вдалий досвіт, тобто збільшення потужності панелі, запам’ятовуємо і збільшуємо корисність виконаної дії. Якщо ж у результаті отримуємо зменшення потужності панелі, користь повинна бути зменшена. Панелі виконують поворот набагато швидше, ніж Сонце, і в деякий момент часу виникає ситуація, коли виконання будь-якої з дій призводить до зменшення потужності. У такому разі виконання алгоритму припиняється.

2.6.2 Схема бібліотеки

Бібліотека у програмуванні – це набір готових рішень для деяких задач. Вона містить протестований код для типових завдань, який можна підключати і використовувати у своїй роботі. Використання бібліотек значно економить час програмістів, оскільки не потрібно писати весь код з нуля. Для зручності інформація у віртуальній бібліотеці розбивається за пакетами ("packages"). Кожний пакет містить протестований код за якимось окремим напрямком. У даній роботі було вирішено створити бібліотеку, яка буде містити готовий код для реалізації паралельного обчислення оптимального положення сонячних панелей. Це дасть можливість використовувати готове рішення одразу в багатьох додатках без дублювання коду. Крім того, створену бібліотеку буде легше модифікувати в майбутньому. Після аналізу функціональних вимог та можливостей мови програмування Java було прийнято рішення виділити наступні пакети (рис. 2.13):

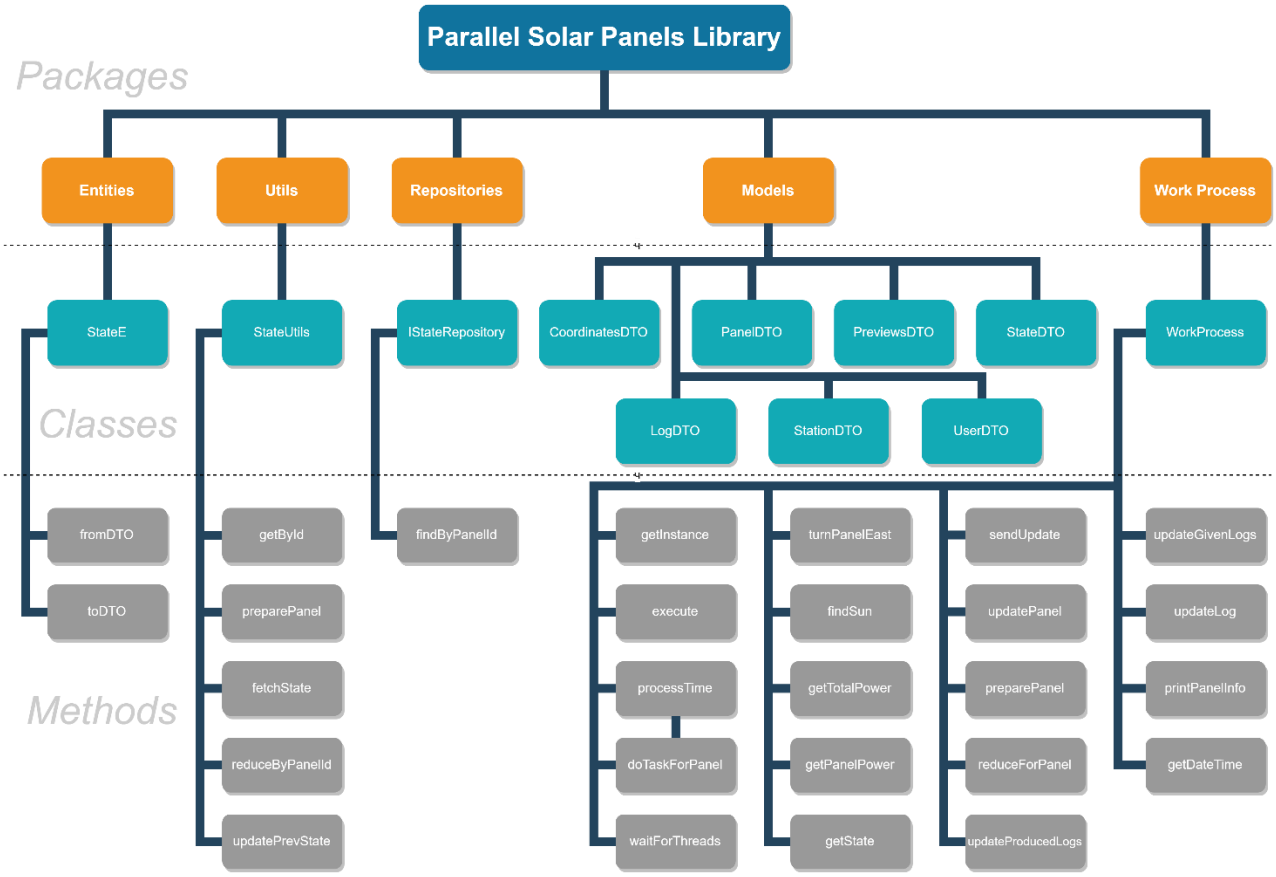


Рисунок 2.13 – Схема бібліотеки

a) Entities – для класів, які є сутностями бази даних. У бібліотеці присутній клас `StateE`. Містить наступні методи (табл. 2.2)

Таблиця 2.2. Методи класу StateE

Метод	Призначення
toDTO()	Перетворити в DTO
fromDTO(StateDTO stateDTO)	Отримати об’єкт сутності з DTO

b) Model – містить класи моделі, які передаються у запитах (DTO – Data Transfer Object). Детальна діаграма класів наведена у додатку Д.

c) Repositories – містить інтерфейс `IStateRepository`, який дозволяє отримувати стани з бази даних та зберігати в неї. Крім стандартних методів CRUD, містить метод `findByPanelId(String panelId)`, що дозволяє отримати всі стани даної панелі.

d) Utils – містить клас `StateUtils`, який має методи для обробки станів сонячних панелей (табл. 2.3):

Таблиця 2.3. Методи класу StateUtils

Метод	Призначення
getById(String id)	Отримати з БД стан за його ідентифікатором.
fetchState(StateDTO stateDTO)	Отримати з БД стан, який відповідає даному за азимутом, висотою та ID панелі. Якщо такого стану не існує, створити його.
updatePrevState(PreviousDTO previousDTO)	Оновити існуючий стан за ID стану, що передається.
reduceByPanelId(String panelId)	Зменшити корисність для всіх станів за заданим ID панелі.
preparePanel(PanelDTO panel)	Підготувати панель до пошуку оптимального положення – задати стимул рухатися правіше.

е) WorkProcess – містить однойменний клас, який виконує основну задачу, а саме корекцію сонячних панелей відносно Сонця. Даний клас містить наступні методи (табл. 2.4):

Таблиця 2.4. Методи класу WorkProcess

Метод	Призначення
getInstance()	Повертає екземпляр класу (singleton)
execute()	Виконати алгоритм пошуку оптимального положення для всіх наявних панелей
processTime(long time)	Вивести статистику по часу, що знадобився для встановлення всіх панелей в оптимальне положення
doTaskForPanel(PanelDTO panel)	Виконати алгоритм пошуку оптимального положення для даної панелі
waitForThreads(List<Thread> threads)	Дочекатися завершення виконання наявних потоків
turnPanelEast(PanelDTO panel)	Повернути панель напрямком на схід
findSun(PanelDTO panel)	Знайти Сонце для даної панелі. Виконується, якщо її потужність більше критичного рівня
getTotalPower()	Знайти поточну потужність всіх панелей
getPanelPower(PanelDTO panel)	Знайти поточну потужність даної панелі
getState(PanelDTO panel)	Знайти стан для поточного положення даної панелі
sendUpdate(PreviousDTO previousDTO)	Запит на оновлення інформації про попередній стан
updatePanel(PanelDTO panelDTO)	Запит на оновлення інформації про дану панель
preparePanel(PanelDTO panelDTO)	Запит на підготовку панелі до пошуку оптимального положення.
reduceForPanel(String panelId)	Запит на зменшення корисності для всіх станів за заданим ID панелі.
updateProducedLogs(PanelDTO panelDTO)	Запит на оновлення даних про виробіток енергії
updateGivenLogs()	Запит на оновлення даних про продаж енергії
updateLog(LogDTO logDTO)	Запит на оновлення статистичних даних
printPanelInfo(PanelDTO panel, int iterator)	Вивести статистику по кількості ітерацій, що знадобилася для встановлення кожної панелі в оптимальне положення
getDateTime()	Повертає поточну дату і час доби

Таким чином, бібліотека буде надавати готові методи для паралельного обчислення оптимального положення сонячних панелей з використанням машинного навчання.

Висновки до розділу

У даному розділі розглянуто основні проблеми, які необхідно вирішити при реалізації віддаленого управління кластером сонячних електростанцій, та способи їх вирішення. Здійснено аналіз різних підходів для реалізації взаємодії компонентів програмних систем, їх переваг та недоліків. Розглянуто моделі зберігання даних та доступу до них. Досліджено методи паралельного програмування для підвищення швидкодії програмних систем. Описано переваги засобу візуалізації Flutter. Розглянуто переваги використання систем керування версіями (СКВ), описано можливості СКВ Git для забезпечення безперервної доставки програмних продуктів. Описано структуру бібліотеки для паралельного обчислення оптимального положення сонячних панелей та модель, якою оперує дана бібліотека.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Паралельне обчислення оптимального положення сонячних панелей

Одним з ключових факторів ефективності системи управління сонячними електростанціями є швидкість реагування сонячних панелей на зміну положення Сонця. Оскільки максимальна генерація електричної енергії досягається, коли сонячні промені падають перпендикулярно до площини панелей, їх необхідно періодично коригувати протягом дня. Для цього програмне забезпечення станції кожні 10 хвилин виконує пошук оптимального положення для кожної панелі. У ході тестування прототипу було визначено, що для коригування положення однієї панелі витрачається 8.04 секунд у перший день та 6.63 секунд у другий день роботи. Для реальних сонячних панелей цей час буде більшим, оскільки треба повернути панель та засікти зміну потужності. Виходячи з ситуації, було прийнято рішення застосовувати алгоритм пошуку оптимального положення паралельно для кожної панелі.

Для реалізації паралельного обчислення на локальному комп’ютері існує 2 основні підходи:

a) Багатопроцесність – використовується для обчислень з розподіленою пам’яттю. Кожний процес незалежний, дублює значну частину інформації про стан та має окремий адресний простір. Процеси виконують кожен свою задачу та взаємодіють один з одним через системні міжпроцесні механізми комунікацій.

b) Багатопотоковість – використовується для обчислень зі спільною пам’яттю. Потоки всередині процесу розподіляють інформацію про стан процесу, і мають прямий доступ до спільної пам'яті та інших ресурсів. Переключення контексту між потоками процесу відбувається швидше, ніж переключення контексту між процесами.

Оскільки у системі, що створюється, паралельно вирішується одна й та сама задача, було прийнято рішення використовувати модель потоків. Алгоритм паралельного обчислення на основі потоків має наступний вигляд (табл. 3.1):

a) Для кожної сонячної панелі створюється потік, який виконує певну задачу.

b) Потік запускається на виконання та додається до масиву потоків.

c) Якщо кількість потоків у масиві досягла деякого максимального значення, очікуємо завершення виконання наявних потоків.

Таблиця 3.1 Паралельні обчислення з використанням потоків (Java)

```
List<Thread> threads = new ArrayList<>();
for (PanelDTO panel : panels) {
    Thread thread = new Thread(() -> doTaskForPanel(panel));
    thread.start();
    threads.add(thread);
    if (threads.size() >= maxSize) {
        waitForThreads(threads);
    }
}

...

private void waitForThreads(List<Thread> threads) {
    for (Thread thread : threads) {
        try {
            thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    threads.clear();
}
```

Для виконання потоку виділяються ресурси ядра комп’ютера. Відповідно, максимальна кількість потоків, які можуть працювати паралельно, дорівнює кількості ядер в системі. Таким чином, використання даного підходу гарантує пришвидшення виконання комплексної задачі в N разів, де N – кількість ядер

процесора. Після вдосконалення алгоритм роботи програмного забезпечення станції має наступний вигляд (рис. 3.1):

a) Виконується зчитування даних стосовно стану перемикачів станцій, запасу енергії акумулятора, сонячних панелей.

b) Якщо панелі знеструмили централізовано, користувачу виводиться відповідне повідомлення і виконується перехід до кроку j).

c) Якщо кількість потоків більша за максимальну, очікуємо їх завершення.

d) Береться наступна панель зі списку, створюється окремий потік для неї. Для цього потоку виконуються кроки e) – i), а основний алгоритм переходить до кроку c).

e) При вимкненої панелі, виводиться відповідне повідомлення користувачу і виконується перехід до кроку j).

f) Визначається потужність панелі.

g) Порівнюємо значення отриманої потужності з критичним значенням. Якщо поточне значення менше критичного рівня, виводиться відповідне повідомлення і виконується автоматичне встановлення панелі напрямком на схід, якщо панель знаходилась в іншому положенні.

h) Виконується алгоритм пошуку оптимального положення панелі, описаний у п. 1.4.2. Оновлюється статистика виробленої енергії.

i) Перевіряємо чи є необроблені панелі, якщо так, переходимо до кроку c).

j) Очікуємо завершення виконання потоків.

k) Оновлюється статистика проданої енергії.

l) Кінець. Очікуємо на команду повтору алгоритму.

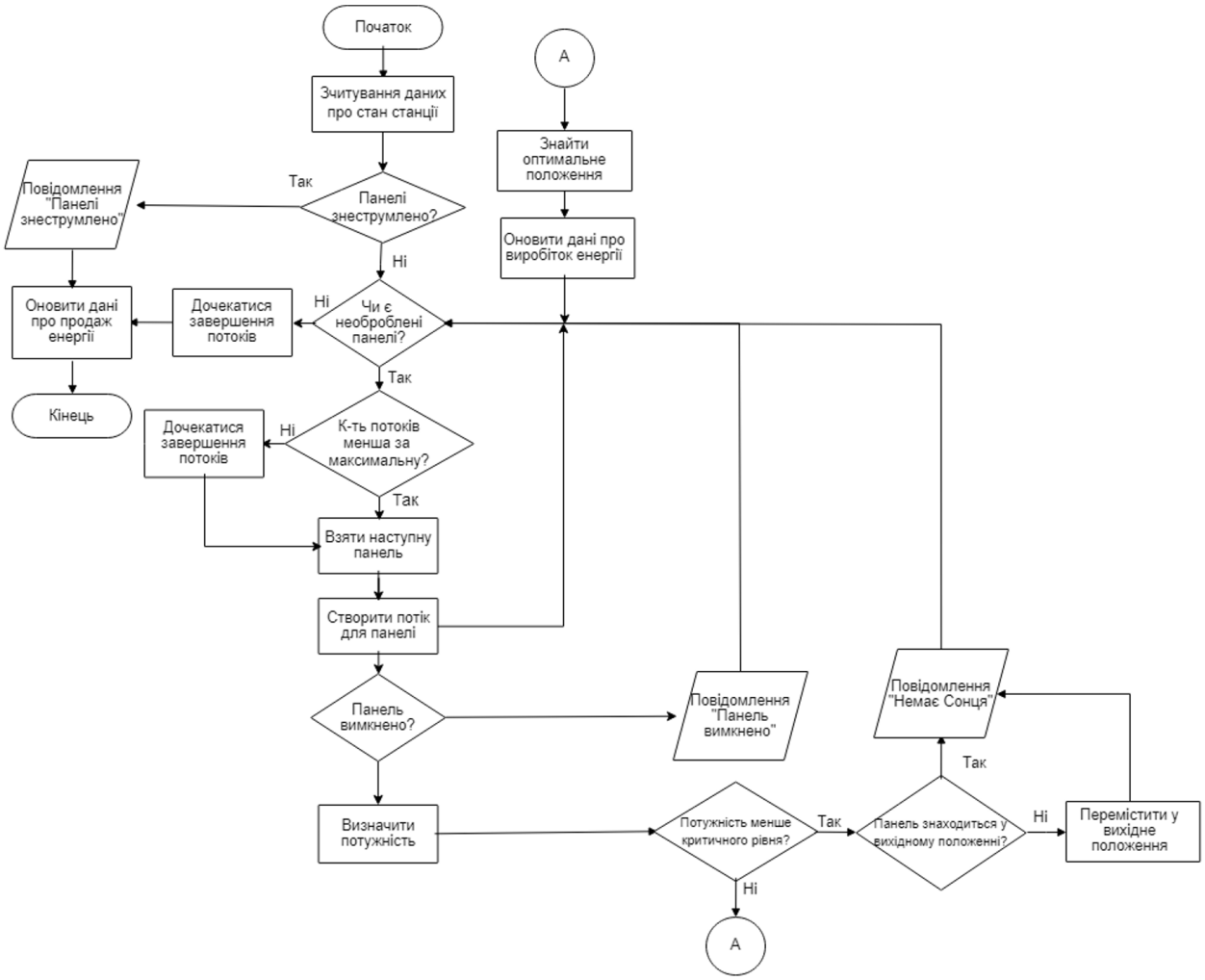


Рисунок 3.1 – Алгоритм роботи ПЗ станції

3.2 Вдосконалення архітектури системи

3.2.1 Оптимізація навантаження на серверну частину системи

Під час тестування прототипу було виявлено критичну вразливість системи. При коригуванні положення сонячних панелей виконується велика кількість запитів до бази даних, і всі вони проходять через центральний сервер. Виведення його з ладу не лише унеможливить перегляд інформації користувачем, а й заблокує процес стеження панелей за Сонцем. Таким чином, навіть виконання планових технічних робіт на сервері буде заважати користувачам отримувати максимальний прибуток від своїх електростанцій.

Для виправлення цього недоліку запропоновано реалізувати прямий доступ програмного забезпечення станцій до бази даних. Використання даного підходу дозволяє станціям працювати у випадку проблем із доступом до сервера. Також це значно знижує навантаження на серверну частину системи та прискорює обробку запитів. Схема бази даних прототипу зображена на рис. 3.2.

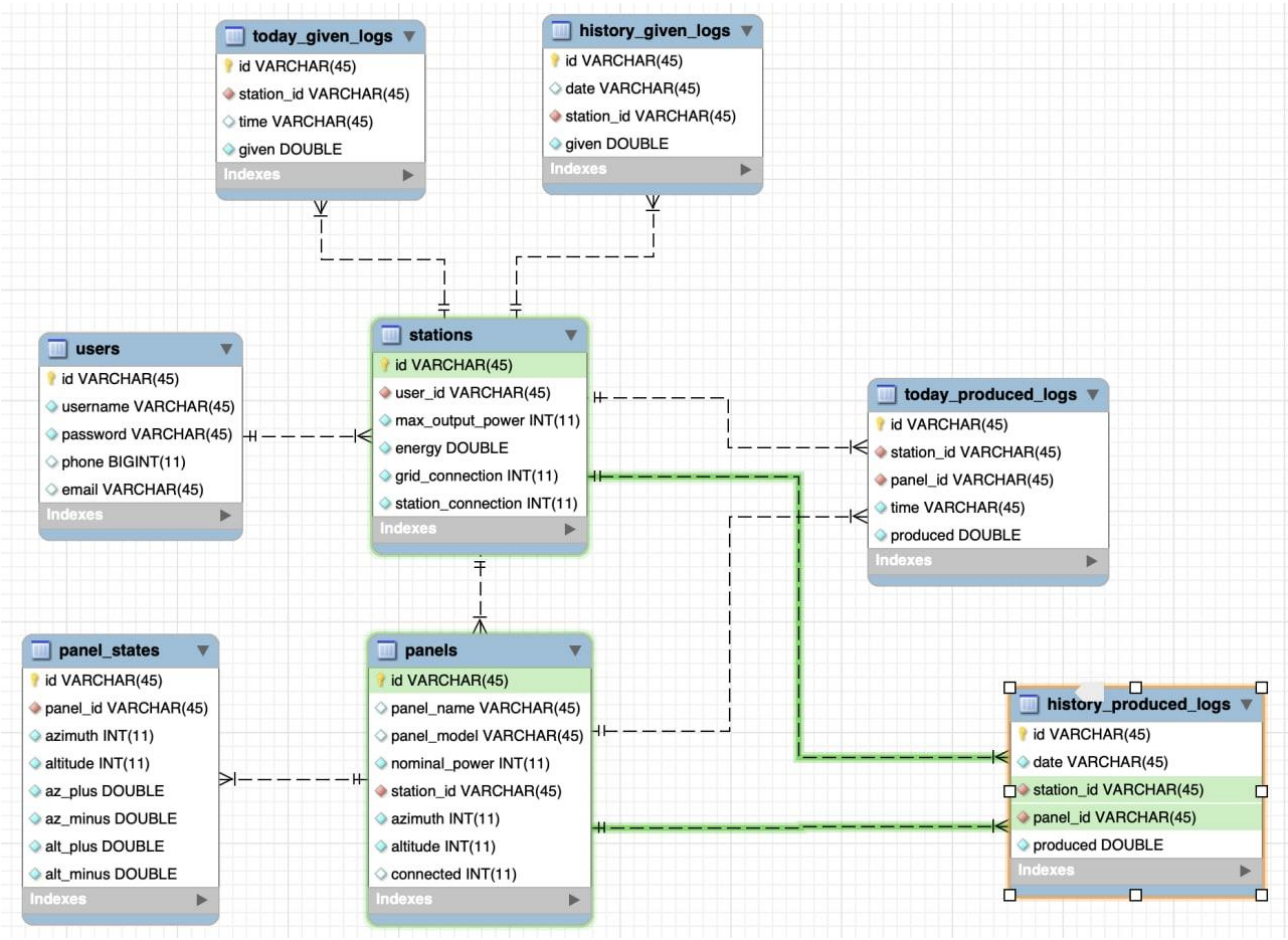


Рисунок 3.2 – Схема БД прототипу

На схемі бачимо таблицю `panel_states`, яка зберігає стани сонячних панелей зі значеннями корисності можливих дій `az_plus`, `az_minus`, `alt_plus`, `alt_minus`. Відповідно до алгоритму роботи програмного забезпечення станції, зміна інформації про стани відбувається в десятки разів частіше, ніж оновлення статистики та ще частіше, ніж отримання команд користувача. Також інформація про стани панелей ніде більше не використовується. У зв’язку з цим пропонується

винести таблицю `panel_states` в окрему бази даних, доступ до якої надати сервісам станцій. Таким чином, процес слідування сонячних панелей за Сонцем буде відбуватися незалежно від серверної частини системи. З одного боку, реалізація запропонованого підходу знизить навантаження на основну базу, а з іншого – контролери електростанцій не будуть мати доступ до зайвої інформації. Також це дасть можливість обирати оптимальну СУБД для кожної бази даних залежно від навантаження та інших факторів.

3.2.2 Масштабування серверної частини системи

Важливою характеристикою програмних систем є масштабованість – здатність системи легко адаптуватися до збільшеного навантаження [24]. Система називається масштабованою, якщо вона здатна обробляти все більший обсяг роботи пропорційно додатковим ресурсам. Також під масштабованістю розуміють можливість залучення додаткових ресурсів без структурних змін архітектури системи. Масштабованість є бажаною характеристикою системи, що створюється, оскільки дозволяє оперативно реагувати на збільшення кількості клієнтів.

Якість масштабованості оцінюється через відношення приросту продуктивності системи до збільшення кількості залучених ресурсів. Чим ближче значення до одиниці, тим краще. Якщо виникає ситуація при якої додавання ресурсів призводить лише до незначного підвищення продуктивності, а з деякого моменту додавання ресурсів не дає жодного корисного ефекту, можна говорити про погану масштабованість.

Існує два види масштабованості: вертикальна та горизонтальна. Вертикальне масштабування – збільшення потужності та продуктивності кожного компонента системи, що повинно призвести до підвищення загальної продуктивності. Можливість замінювати в існуючій обчислювальній системі компоненти на більш потужні та швидкі у міру зростання вимог та розвитку технологій і означає масштабованість. Це найпростіший спосіб масштабування, тому що не вимагає жодних змін у коді прикладних програмах, які працюють на таких системах.

Горизонтальне масштабування — розбиття системи на дрібніші структурні компоненти та рознесення їх по окремих фізичних машинах (або їх групах), та (або) збільшення кількості серверів, що паралельно виконують одну й ту саму функцію. Можливість додавати до системи нові вузли, сервери та інші пристрої для збільшення загальної продуктивності і є масштабованістю. Цей спосіб масштабування в деяких випадках може вимагати внесення змін до програмного коду, щоб програми могли повною мірою використовувати збільшену кількість ресурсів.

Для забезпечення горизонтальної масштабованості системи управління кластером сонячних електростанцій у даній роботі використовується мікросервісна архітектура, яка передбачає розбиття додатку на багато відносно простих функціональних компонентів (мікросервісів), які взаємодіють між собою. Основні переваги даного підходу наведено у пункті 2.1.3.

Після аналізу функціоналу прототипу було вирішено розділити центральний сервер на наступні мікросервіси:

- сервіс авторизації – для реєстрації та авторизації користувачів;
- сервіс статистики – для збору даних про виробіток та продаж електроенергії для кожної станції;
- сервіс управління – для передачі команд користувача потрібній станції;
- сервіс обробки запитів – точка входу системи, перенаправляє запити користувачів на потрібний сервіс.

Для реалізації було вирішено використати технологію Spring Cloud Netflix Eureka [25] мови програмування Java. Дана технологія дозволяє створювати мікросервісні додатки, в яких окремі мікросервіси можуть розміщуватися на різних робочих станціях. Кожний мікросервіс має своє ім’я і може звертатися до інших мікросервісів, але важливо щоб всі вони були зареєстровані на одному сервісі виявлення, по імені сервіса. Сервіс виявлення позначається Eureka Server. У результаті сервісу управління для отримання доступу до сервісів контролю станцій (їх багато) не потрібно знати ip-адреси всіх станцій, а треба знати їх

ідентифікатори (які є іменами відповідних сервісів) та ip-адресу сервісу виявлення. Цей механізм дає перевагу, оскільки інформація про ip-адреси електростанцій користувачів не потрібна. Схематичне зображення роботи сервісу виявлення представлене на рисунку 3.3.

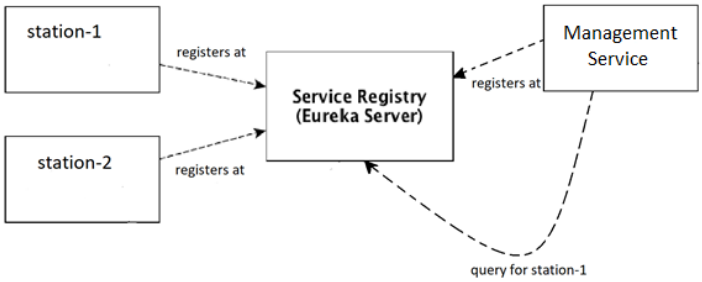


Рисунок 3.3 – Реєстрація сервісів у Spring Cloud Netflix Eureka

Таким чином, архітектура системи, що створюється, набула наступного вигляду (рис. 3.4):

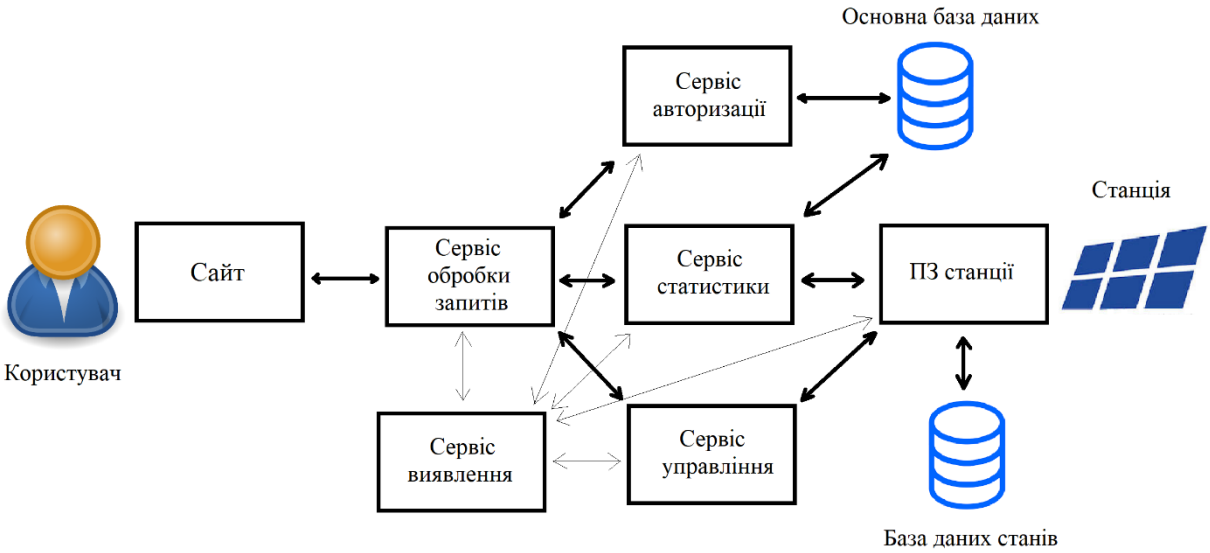


Рисунок 3.4 – Архітектура розробленої системи

Програмне забезпечення станції встановлюється на контролері електростанції та контролює її роботу. Ця програма з заданою періодичністю коригує положення сонячних панелей за допомогою бази даних станів панелей,

зчитує дані як про стан станції, так і кожної панелі, та передає отриману інформацію на сервіс статистики. Також вона виконує команди управління, отримані з сервіса управління.

Сервіс обробки запитів є точкою входу системи. Він призначений для переадресації запитів користувачів на потрібний сервіс.

Сервіс авторизації відповідає за реєстрацію та авторизацію користувачів у системі.

Сервіс статистики відповідає за дані про виробіток та продаж електроенергії для кожної станції. Сервіс авторизації та сервіс статистики мають доступ до основної бази даних.

Сервіс управління призначений для передачі команд управління користувача до відповідної станції.

Сервіс виявлення виконує реєстрацію вище наведених сервісів та сервісів електростанцій. Таким чином зареєстровані сервіси отримують можливість звертатися один до одного по імені сервісу, не використовуючи ip-адрес. У розробленій системі це дозволяє сервісу управління передавати запити користувачів потрібній станції по її ідентифікатору, не знаючи реальних ip-адрес.

На веб-сайті відображаються корисні дані для користувача а саме: актуальна інформація про роботу своєї станції на сайті компанії. Також представлена можливість керувати підключенням окремих блоків. З сайту запити користувача відправляються на сервіс обробки запитів, а потім, з нього, переадресовуються на необхідний сервіс.

3.2.3 Схема інформаційних потоків системи

У системі, що створюється, можна виділити наступні види інформаційних потоків (рис. 3.5):

a) Коригування положення панелі.

b) Авторизація користувача в системі.

c) Передача станції команд управління від користувача.

d) Запит статистики користувачем.

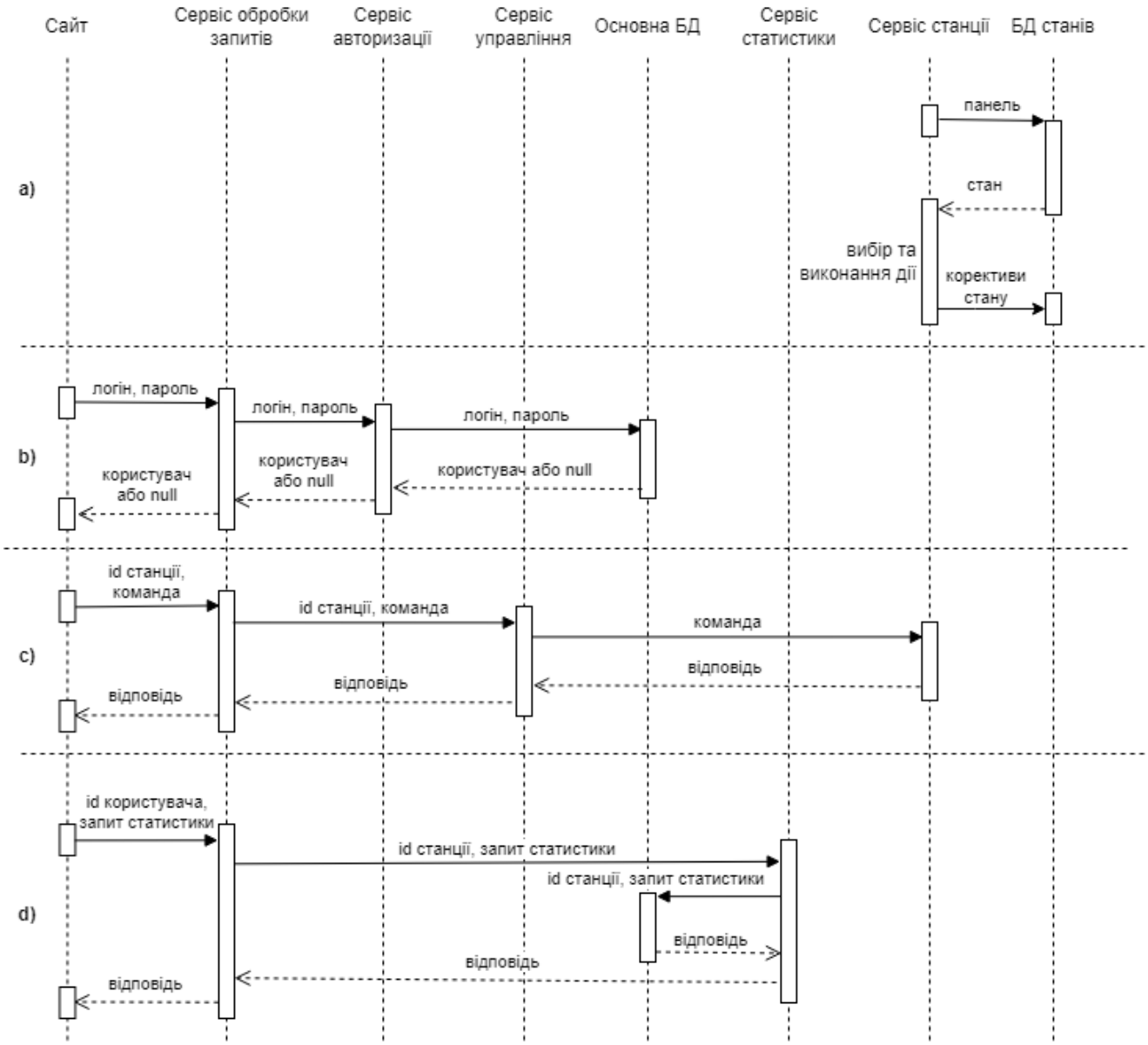


Рисунок 3.5 – Схема інформаційних потоків

Для коригування положення деякої панелі ПЗ станції робить запит до бази даних станів, у якому вказується ідентифікатор панелі та координати поточного положення. В базі даних станів відбувається пошук стану, який відповідає одержаним ідентифікатору та координатам панелі. Якщо під час пошуку нічого не знайшли то такий стан створюється. БД станів видає станції шуканий стан. Після того як стан панелі був отриманий, станція виконує рекомендовану дію, в

результаті панель переходить у новий стан. Під час роботи станція обчислює результат виконання дії, коригує корисність початкового стану та зберігає зміни в базі даних станів.

Для авторизації користувачу потрібно на сайті вести логін та пароль, після чого сайт відправляє дані сервісу обробки запитів. Сервіс обробки передає запит сервісу авторизації, який робить запит в основну базу даних для пошуку користувача за логіном та паролем. Якщо користувач ідентифікований, БД повертає його сервісу авторизації. Якщо користувача не ідентифіковано, повертається null. Сервіс авторизації передає одержаний результат сервісу обробки запитів, який передає результат сайту. Якщо отриманий результат не null, то авторизація підтверджується, інакше в доступі буде відмовлено

Для управління станцією користувач на сайті обирає можливу команду, сайт передає сервісу обробки запитів обрану команду та ідентифікатор станції. Сервіс обробки запитів передає запит сервісу управління, той шукає потрібну станцію за ідентифікатором та передає їй отриману команду. Станція виконує команду та повертає сервісу управління результат. Сервіс управління передає отриманий результат сервісу обробки запитів, той передає його сайту.

Для отримання статистики роботи станції сайт відправляє сервісу обробки запитів ідентифікатор станції та запит статистики. Сервіс обробки передає запит сервісу статистики, той передає запит в основну базу даних для пошуку статистики за ідентифікатором станції. БД повертає відповідь сервісу статистики, той передає її сервісу обробки запитів, який передає результат сайту.

3.3 Методи тестування створеної системи

Для тестування системи було створено сервіс, який зберігає дані реального положення Сонця 1-4 червня 2022 року у місті Києві. Інформацію взято з сайту Stellarium-web [26]. Створений сервіс містить масив даних у форматі «ключ-значення», де ключем є дата і час у форматі «рррр-мм-ддТгг:хх:сс», наприклад, «2022-06-01T10:30:00». Таким чином ми отримуємо координати положення Сонця

в даний час. Далі, знаючи положення Сонця та положення панелі, визначаємо коефіцієнт

$$k = \cos \varphi, \tag{3.1}$$

де φ – кут між сонячним променем та перпендикуляром до сонячної панелі (рис. 3.6), який визначається за математичними формулами. Тоді поточну потужність панелі визначаємо

$$P_i = P_{nom} * k_i, \tag{3.2}$$

де P_{nom} – номінальна потужність панелі, k_i – поточний коефіцієнт.

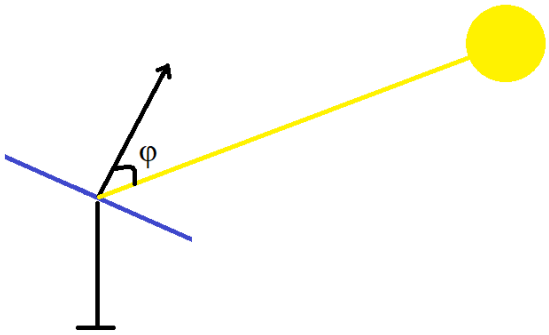


Рисунок 3.6 – Кут між сонячним променем та перпендикуляром

Для оцінки ефективності виробітку енергії ми фіксуємо потужність панелі в остаточному положенні. За час експериментів (4 тестових дні) кінцева потужність прототипу переважно встановлювалася вище за 279 ват при номінальній потужності 280 ват, тобто втрати не перевищують 0.4%. Таким чином, робимо висновок про ефективність даного методу пошуку оптимального положення.

Для оцінки ефективності навчання системи фіксується кількість дій, яка знадобилася панелі для встановлення в оптимальне положення (рис. 3.7). В середньому за перший день роботи прототипу (база зі значеннями корисності була порожня) панель виконувала 33.11 дій. За другий день (враховуючи досвід попереднього дня) панель виконувала 23.33 дій. Для третього дня цей показник становив 22.48 дій, для четвертого – 20.18 дій. Ці дані дозволяють зробити висновок про корисність навчання для розробленої системи.

```
Console   Actuator
Panel Altek 3: final azimuth: 55; altitude: 5; index: 62; power: 379.36623565476
Iterations: 25
Task executed on Sun Feb 20 23:28:01 EET 2022
Panel Risen 3: final azimuth: 59; altitude: 5; index: 64; power: 279.8061019316789
Iterations: 3
Panel Ja Solar 2: final azimuth: 56; altitude: 5; index: 64; power: 539.3922244819956
Iterations: 5
Panel Altek 2: final azimuth: 56; altitude: 6; index: 64; power: 379.2981828060729
Iterations: 7
Panel Ja Solar 4: final azimuth: 57; altitude: 5; index: 64; power: 539.6337709006343
Panel Altek 3: final azimuth: 58; altitude: 5; index: 64; power: 379.79713763096805
Iterations: 9
Iterations: 10
Panel Ja Solar 3: final azimuth: 57; altitude: 5; index: 64; power: 539.6337709006343
Iterations: 14
Panel Altek 1: final azimuth: 56; altitude: 5; index: 64; power: 379.5723061169599
Iterations: 17
Panel Risen 2: final azimuth: 56; altitude: 5; index: 64; power: 279.68485713881256
Iterations: 18
Panel Ja Solar 1: final azimuth: 59; altitude: 5; index: 64; power: 539.6260537253808
Iterations: 32
Panel Risen 1: final azimuth: 57; altitude: 5; index: 64; power: 279.81010342995853
Iterations: 44
Task executed on Sun Feb 20 23:28:11 EET 2022
```

Рисунок 3.7 – Дослідження кількості ітерацій

Для оцінки швидкодії системи ми фіксуємо час, необхідний для встановлення в оптимальне положення десяти панелей (рис. 3.8). В середньому за перший день роботи прототипу даний час становив 8.04 секунд, за другий день – 6.63 секунд. Для третього дня цей показник становив 6.37 секунд, для четвертого – 5.85 секунд.

```
Console   Actuator
Time to correct all panels: 514
Task executed on Sun Feb 20 23:21:17 EET 2022
Time to correct all panels: 625
Task executed on Sun Feb 20 23:21:27 EET 2022
Time to correct all panels: 1183
Task executed on Sun Feb 20 23:21:37 EET 2022
Time to correct all panels: 2343
Task executed on Sun Feb 20 23:21:48 EET 2022
Time to correct all panels: 2086
Task executed on Sun Feb 20 23:21:58 EET 2022
Time to correct all panels: 1785
Task executed on Sun Feb 20 23:22:08 EET 2022
Time to correct all panels: 1900
Task executed on Sun Feb 20 23:22:18 EET 2022
Time to correct all panels: 1864
Task executed on Sun Feb 20 23:22:28 EET 2022
Time to correct all panels: 1587
Task executed on Sun Feb 20 23:22:38 EET 2022
Time to correct all panels: 2247
Task executed on Sun Feb 20 23:22:48 EET 2022
Time to correct all panels: 1003
Task executed on Sun Feb 20 23:22:57 EET 2022
Time to correct all panels: 867
Task executed on Sun Feb 20 23:23:07 EET 2022
```

Рисунок 3.8 – Дослідження витраченого часу

3.4 Оцінка ефективності запропонованого рішення

Для оцінки ефективності запропонованих рішень використовуємо симулятор Сонця, детально описаний у попередньому пункті. Досліджується середній час за добу, необхідний для встановлення 10 сонячних панелей в оптимальне положення. У ході роботи було проведено дослідження для наступних ситуацій (рис. 3.9):

a) Прототип – до початку роботи над удосконаленням системи.

b) Прототип після реалізації прямого доступу станцій до БД (проміжний результат).

c) Прототип після реалізації прямого доступу станцій до БД та алгоритму паралельного обчислення оптимального положення (кінцевий результат).

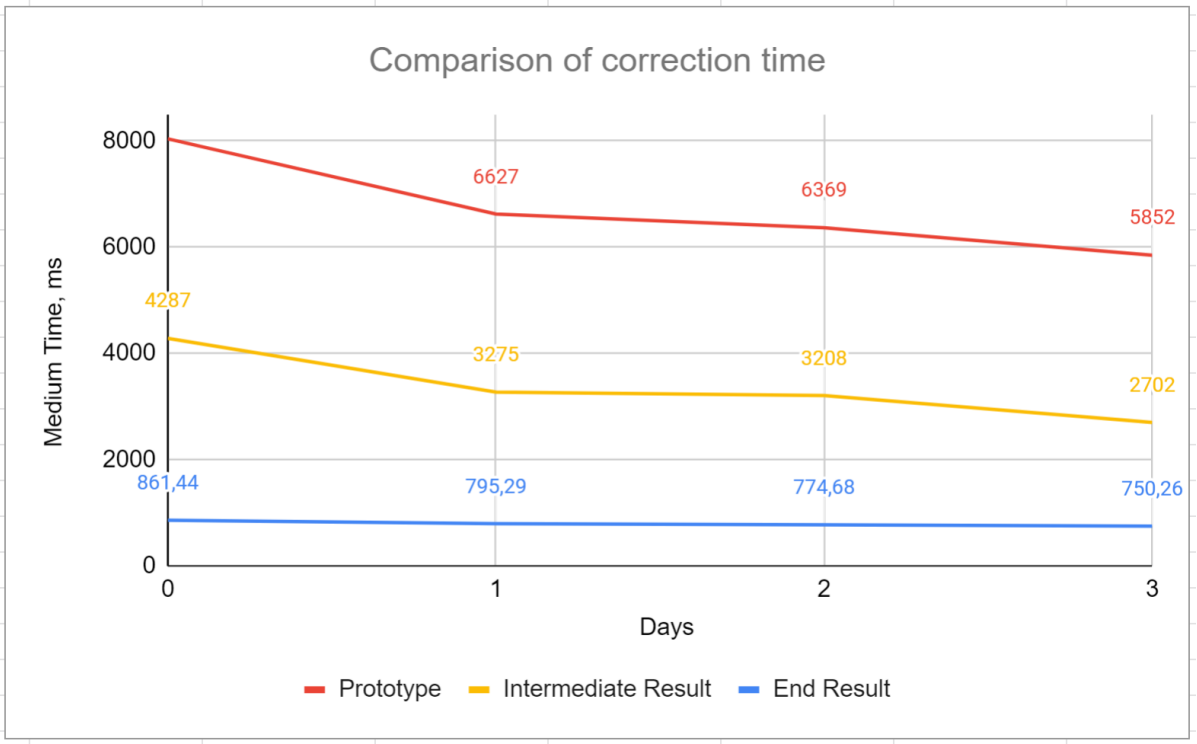


Рисунок 3.9 – Порівняння часу корекції 10 панелей, мс

З рисунку видно, що реалізація прямого доступу станцій до бази даних зменшує необхідний час в середньому на 50%. Це пояснюється тим, що при корекції положення сонячної панелі запит проходить удвічі менший шлях між вузлами за рахунок оптимізації архітектури системи. Використання паралельних обчислень зменшує даний час у 4.24 рази. Даний результат відповідає очікуванням, оскільки комп’ютер, на якому проводилися експерименти, має 4 ядра.

3.5 Опис інтерфейсу та інструкція користувача

Для роботи з розробленою системою необхідно скористатися будь-яким браузером та перейти на сайт продукту. Після цього користувачу відкриється сторінка авторизації. Поля логіну та паролю обов’язкові для заповнення (рис. 3.10).

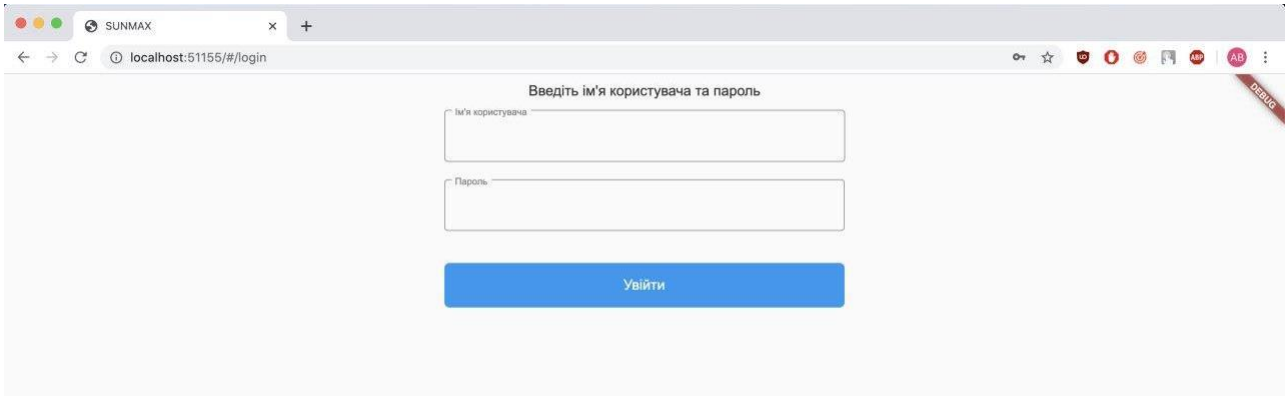


Рисунок 3.10 – Сторінка входу на сайт

Якщо користувача за введеними даними не знайдено, то з'являється повідомлення про помилку (рис. 3.11).

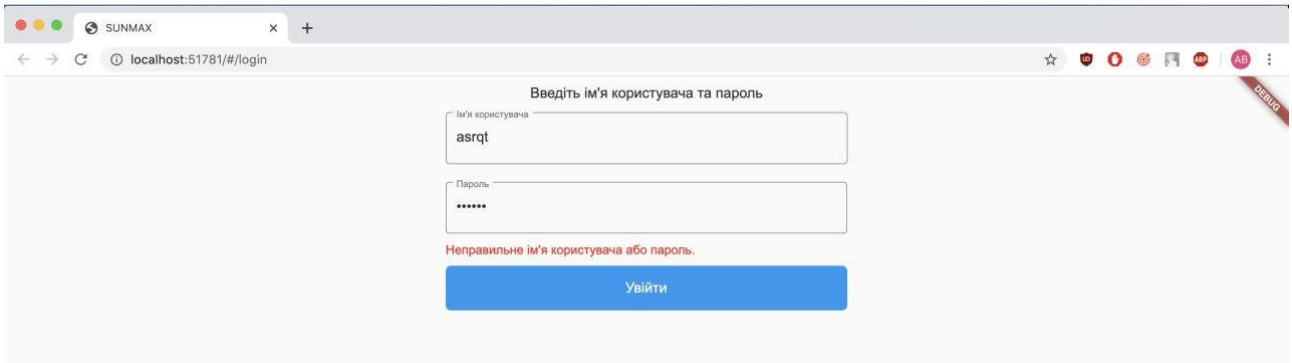


Рисунок 3.11 – Некоректно введені дані

Якщо введені дані коректні, то відкривається головна сторінка (рис. 3.12). На ній ми бачимо електростанції, які належать даному користувачеві. Елементи списку мають вигляд прямокутних віджетів, на кожному з яких показується схематичне зображення відповідної станції, стан перемикачів, назва та поточна потужність сонячних панелей.

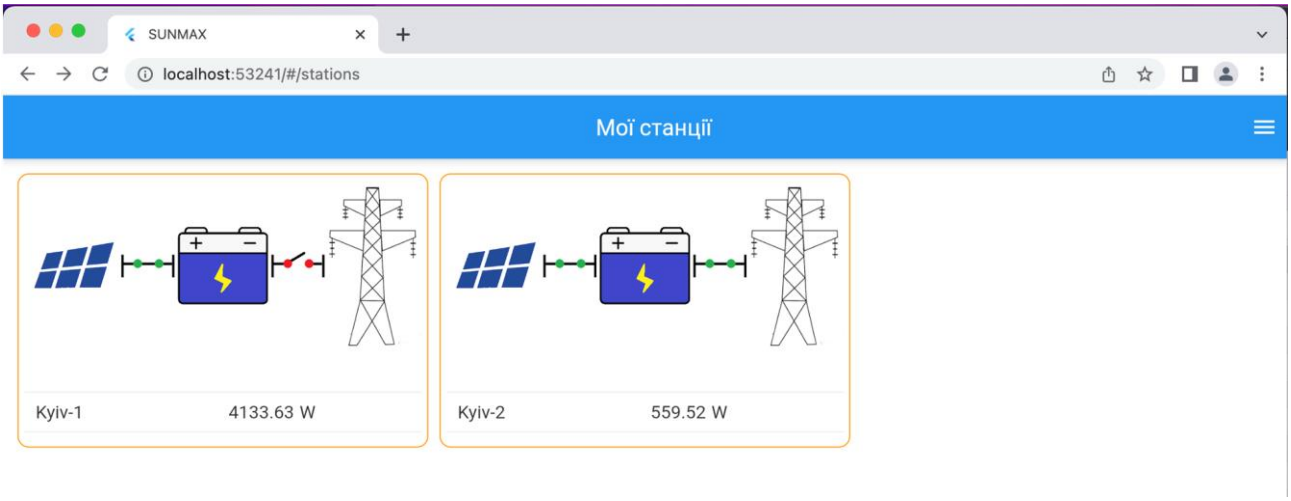


Рисунок 3.12 – Список електростанцій користувача

При натисканні на будь-який елемент відкривається сторінка відповідної станції (рис. 3.13). На ній ми бачимо збільшену схему з інтерактивними елементами, а нижче – поточні дані стану станції (поточна потужність у ватах, кількість виробленої, накопиченої та реалізованої електроенергії у кВт * год). Користувач може натиснути на зображення сонячних панелей (про це мова буде йти далі) та перемикачів. Призначення лівого перемикача – увімкнути або знеструмити всі панелі, призначення правого перемикача – увімкнути або вимкнути постачання енергії в зовнішню мережу. Стан перемикачів на екрані відповідає поточному стану відповідних фізичних перемикачів на станції.

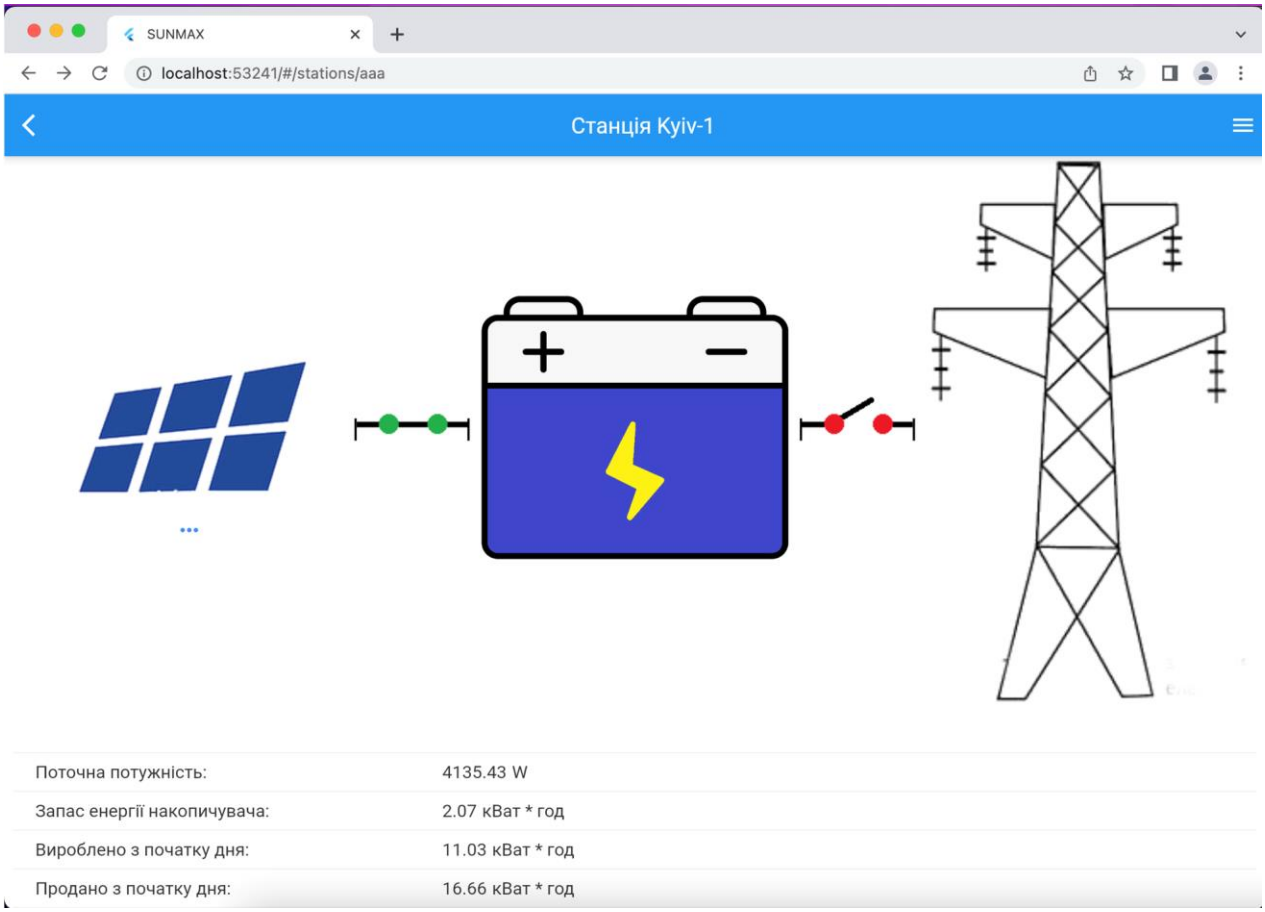


Рисунок 3.13 – Сторінка електростанції

При прокручуванні вниз користувач бачить графік сумарної генерації та реалізації електроенергії за сьогоднішній день (рис. 3.14). Графік зеленого кольору показує кількість енергії, виробленої за годину; графік помаранчевого кольору показує кількість енергії, проданої в мережу.

Рисунок 3.14 – Графік виробітку та продажу енергії протягом дня

Під графіком розташована кнопка “Історія”. При натисканні на неї буде показана статистика денного виробітку та продажу електричної енергії (рис. 3.15).

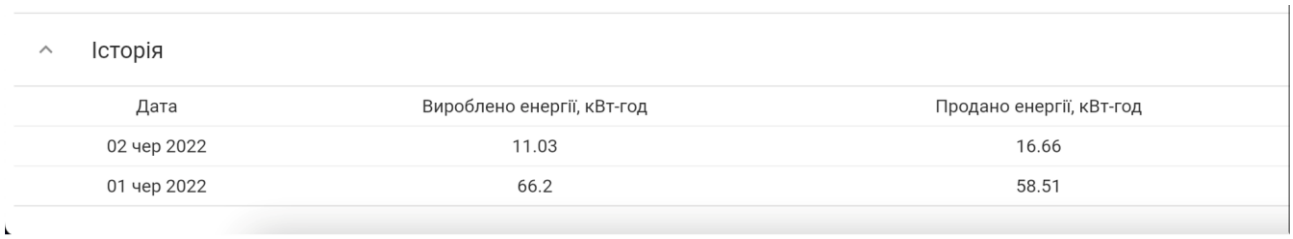


Рисунок 3.15 – Історія денної генерації та продажу енергії

При натисканні на зображення сонячних панелей користувач переходить на сторінку списку панелей (рис. 3.16). Сторінка відображає в табличному вигляді ім’я, модель, положення та потужність кожної панелі. Праворуч від зображених панелей є перемикач для вмикання / вимикання відповідної панелі.

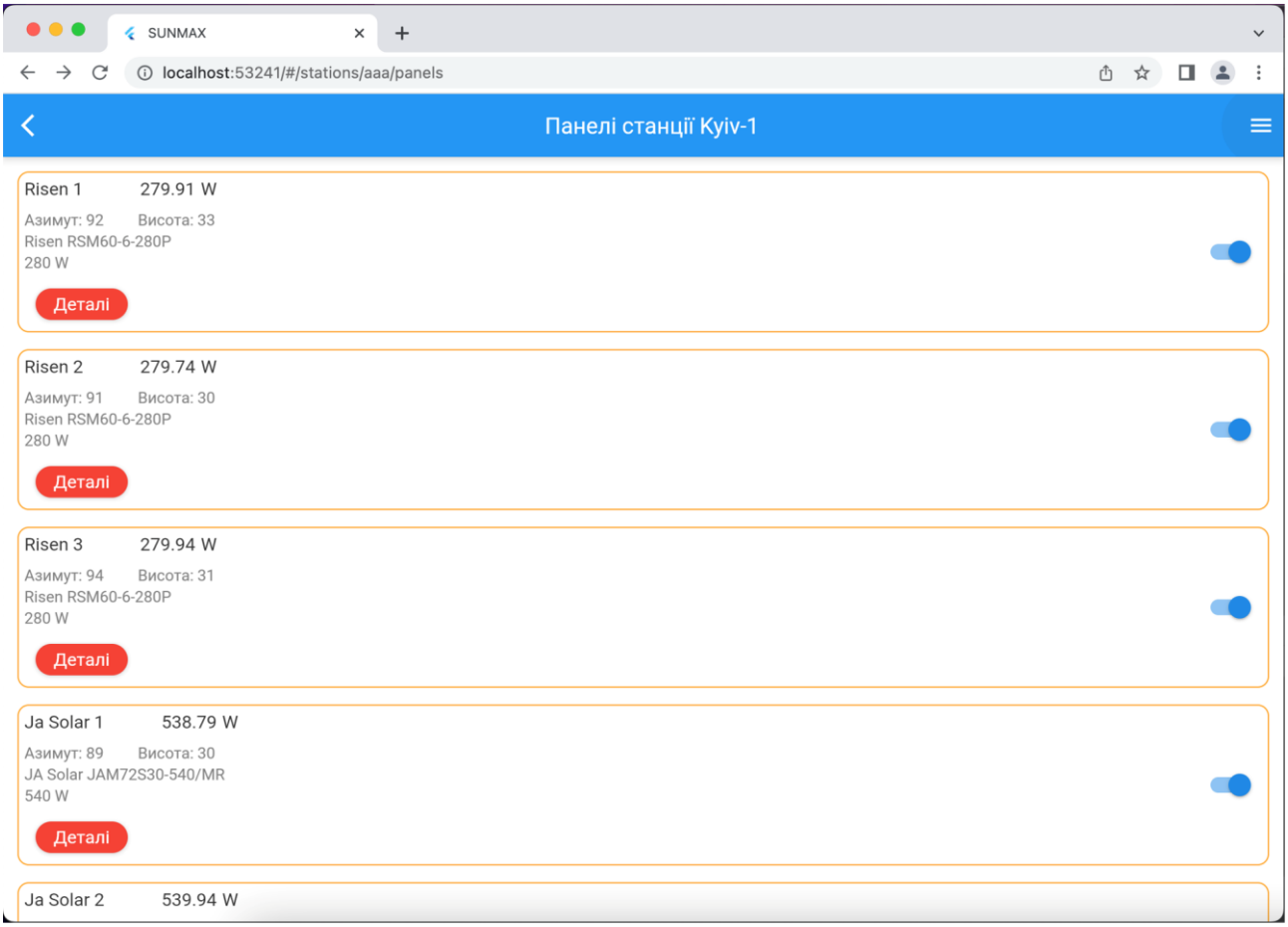


Рисунок 3.16 – Сторінка списку панелей

Натискання кнопки “Деталі” відкриває сторінку з детальною інформацію по обраній панелі (рис. 3.17).

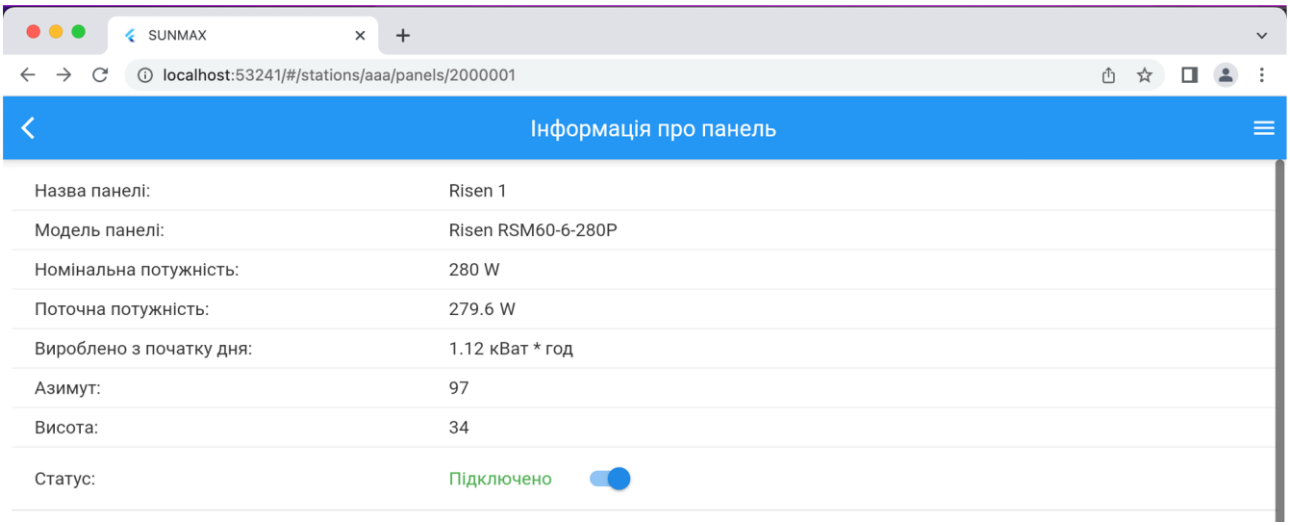


Рисунок 3.17 – Детальна інформація про панель

Після загальної інформації зображено графік генерації енергії даною панеллю за сьогоднішній день (рис. 3.18), а нижче – історія денного виробітку для даної панелі (рис. 3.19). На сторінці панелі не відображається кількість реалізованої електроенергії, оскільки постачання енергії в зовнішню мережу відбувається загалом з акумулятора, а не напряму з кожної панелі. За допомогою графіка можна аналізувати зміну генерації енергії для даної панелі.

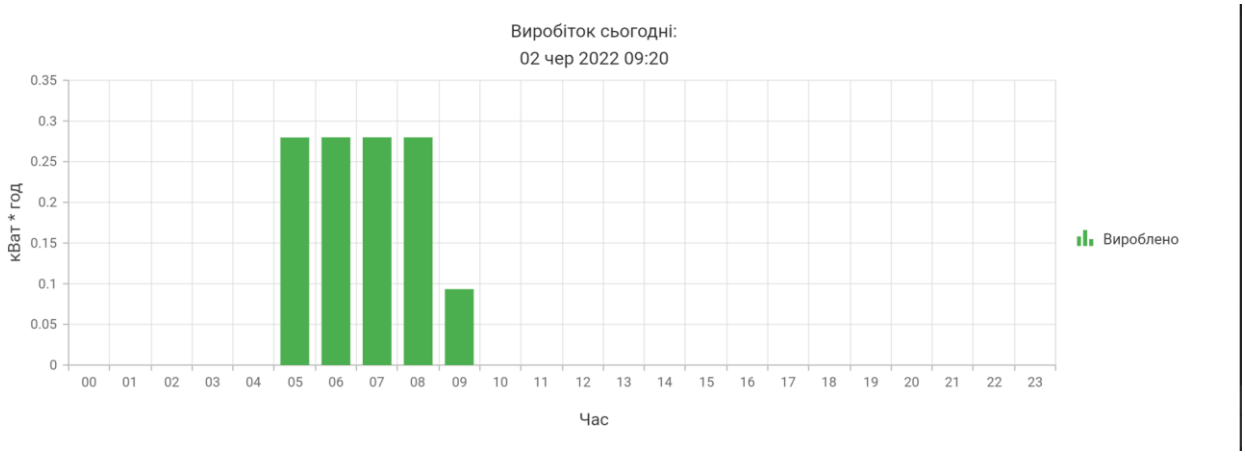


Рисунок 3.18 – Графік виробітку енергії панеллю протягом дня

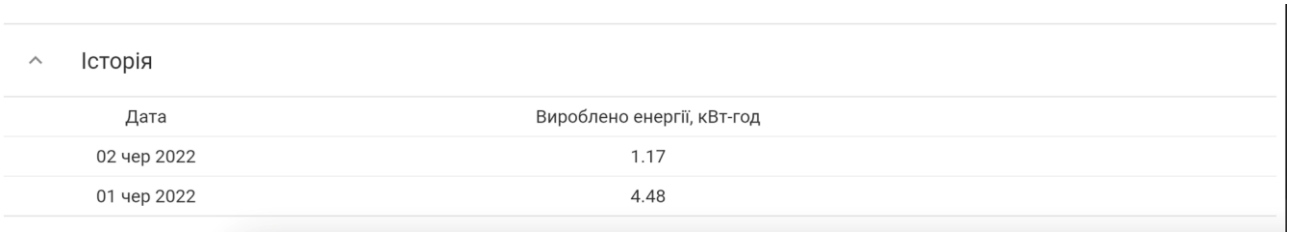


Рисунок 3.19 – Історія денної генерації енергії панеллю

Справа від заголовка сторінки знаходиться кнопка меню. При натисканні на неї з'являється бокове меню, яке має наступні пункти (рис. 3.20):

- «Головна» – переводить користувача на сторінку списку станцій.
- «Вихід» – здійснює вихід користувача з системи та переводить його на сторінку входу.

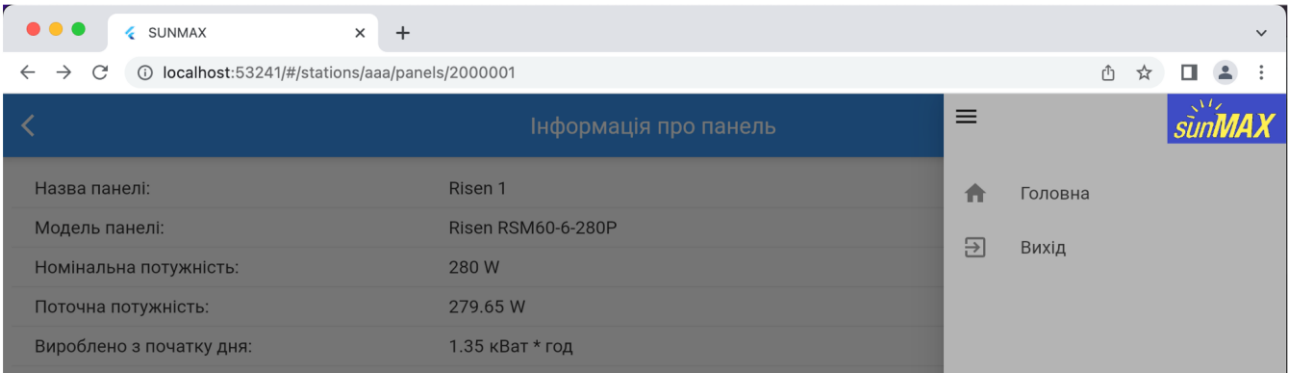


Рисунок 3.20 – Бокове меню

Створений додаток є крос-платформним. Крім веб-версії, описаної вище, підтримуються версії для iOS (рис. 3.21) та Android (рис. 3.22). Додаток має адаптивний дизайн, тобто вміст однаково добре відображається на пристроях з різним розміром екрану. Функціонал однаковий для всіх версій.

Рисунок 3.21 – Інтерфейс користувача на iPhone

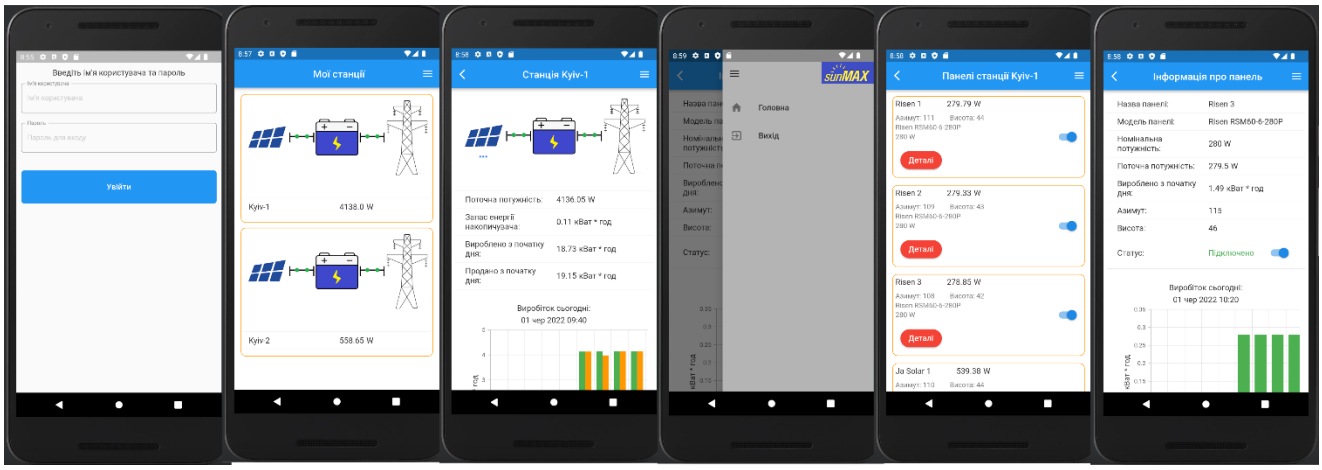


Рисунок 3.22 – Інтерфейс користувача на Android

Висновки до розділу

У даному розділі описано шляхи підвищення продуктивності системи управління сонячними електростанціями. Вдосконалено архітектуру системи для оптимізації навантаження на компоненти. Реалізовано алгоритм паралельного обчислення оптимального положення сонячних панелей з використанням машинного навчання.

Проведено аналіз ефективності запропонованих рішень порівняно з прототипом. Перевірено відповідність показників роботи вимогам, визначеним у розділі 1. Досліджено підвищення швидкості роботи системи в умовах, наближених до реальних. Результати експериментів показали, що вдосконалення

архітектури підвищує швидкодію системи мінімум у 2 рази. Використання паралельних обчислень підвищує швидкодію ще у 4.24 рази. Отримані результати відповідають очікуванням.

Розглянуто інтерфейс користувача на стороні клієнта, описано функціонал створеного додатку.

ВИСНОВКИ

У ході виконання магістерської дисертації було розглянуто питання, пов’язані з підвищення швидкості роботи, якості та надійності систем управління кластером сонячних електростанцій, що використовують машинне навчання.

На основі даних, отриманих у процесі аналізу, сформульовано задачі вдосконалення архітектури для підвищення стійкості системи, а також модифікації алгоритму пошуку оптимального положення сонячних панелей за допомогою методів паралельного програмування.

Для розробки серверної частини системи було використано технологію Spring Cloud мови програмування Java, що дозволяє створювати легко масштабовані мікросервісні додатки. Для зберігання даних було використано СУБД MySQL, яка є простою у використанні та надійною. Для розробки клієнтського додатку було використано фреймворк Flutter, який забезпечує зручний інтерфейс користувача. Все перераховане є широко вживаними та безкоштовними інструментам.

Розроблено програмне забезпечення, яке поєднує в собі функції контролю стану станції, віддаленого управління сонячною електростанцією та онлайн-моніторингу. Результати експериментів показали підвищення швидкодії та надійності розробленої системи порівняно з прототипом. Переваги розробленого рішення:

a) Забезпечено автономність програмного забезпечення електростанцій при коригуванні положення сонячних панелей. Пошук оптимального положення панелі відбувається незалежно від серверної частини, що підвищує стійкість системи. Також це зменшує час пошуку в 2 рази.

b) Реалізовано паралельний пошук оптимального положення за рахунок використання багатопотоковості. В результаті пошук відбувається одночасно для N панелей, де N – кількість ядер процесора контролера

станції. Відповідно, загальний процес коригування всіх панелей станції відбувається в N разів швидше.

c) За рахунок використання мікросервісної архітектури розроблену систему можна горизонтально масштабувати. Для кожного сервісу можна створити потрібну кількість екземплярів.

Результати роботи над магістерською дисертацією опубліковані в науковій статті:

Мокрий А.В. Методи та програмні засоби для управління кластером сонячних електростанцій / А.В. Мокрий, І.В. Баклан // Науковий журнал «Адаптивні системи автоматичного управління» (АСАУ-2022) – м. Київ: НТУУ «КПІ ім. Ігоря Сікорського», 2022 р.

Наукова новизна запропонованого рішення полягає в тому, що вперше створено бібліотеку для паралельного обчислення оптимального положення сонячних панелей з використанням машинного навчання. Також набуло подальшого розвитку використання мікросервісної архітектури для вирішення прикладних задач.

Практичне значення систем віддаленого управління важко переоцінити в умовах бойових дій. Їх використання дозволяє продовжити керувати об’єктами, які розташовані у важкодоступних районах. Розробка даної системи відбувалася в мирний час, але концепцію паралельного обчислення оптимального положення сонячних панелей можна використати для реалізації одночасного наведення артилерійських батарей або засобів протиповітряної оборони по координатах.

ПЕРЕЛІК ПОСИЛАНЬ

1) На газорозподільних станціях встановлять сонячні панелі. URL: https://ua-energy.org/uk/posts/na-hazorozpodilnykh-stantsiiakh-vstanovliat-soniachni-paneli (дата звернення: 20.03.2022).

2) Як “зелена” енергія допомагає виживати під час військових дій. URL: https://ua-energy.org/uk/posts/yak-zelena-enerhiia-dopomahaie-vyzhyvaty-pid-chas-viiskovykh-dii (дата звернення: 05.05.2022).

3) Сонячна енергетика в Україні. URL: https://avenston.com/articles/solar/ (дата звернення: 20.01.2022).

4) Демонстрація системи моніторингу "Solar.web". URL: https://www.solarweb.com/PvSystems/Widgets (дата звернення: 22.05.2022).

5) Демонстрація системи моніторингу "SolarEdge Monitoring System". URL: https://monitoring.solaredge.com/solaredge-web/p/site/315737/#/dashboard (дата звернення: 22.05.2022).

6) *Richard S. Sutton and Andrew G. Barto*. Reinforcement Learning: An Introduction. [Електронний ресурс]. — 2005. — Режим доступу до ресурсу: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf (дата звернення: 17.09.2021).

7) 4 типи архітектури програмного забезпечення. URL: https://nuancesprog.ru/p/12019/ (дата звернення: 22.11.2021).

8) Концепції Сервіс-Орієнтованої Архітектури. URL: https://learn.ztu.edu.ua/mod/book/view.php?id=278&chapterid=72 (дата звернення: 23.11.2021).

9) What Is Microservices – Introduction To Microservice Architecture. URL: https://www.edureka.co/blog/what-is-microservices/ (дата звернення: 24.11.2021).

10) SOA vs. Microservices: What’s the Difference? URL: https://learn.ztu.edu.ua/mod/book/view.php?id=278&chapterid=72 (дата звернення: 24.11.2021).

11) Шаховська Н. Б. Модель великих даних – сутність-характеристика / Н. Б. Шаховська, Ю. Я. Болюбаш // Вісник Національного університету "Львівська політехніка". Серія: Інформаційні системи та мережі: збірник наукових праць. – 2015. – № 814. – С. 186–196

12) Основні поняття реляційних БД: нормалізація, зв’язок та ключі. URL: https://bondarenko.dn.ua/osnovni-ponyattya-relyatsijnih-bd-normalizatsiya-zv-yazok-ta-klyuchi/ (дата звернення: 15.12.2021).

13) SQLite Reference Doc. URL: https://www.sqlite.org/about.html (дата звернення: 16.12.2021).

14) PostgresQL Reference Doc. URL: https://www.postgresql.org/about/ (дата звернення: 17.12.2021).

15) MySQL Reference Doc. URL: https://www.mysql.com/ (дата звернення: 18.12.2021).

16) What is document-oriented database? URL: https://aws.amazon.com/nosql/document/ (дата звернення: 10.01.2022).

17) CouchDB Technical documentation. URL: http://docs.couchdb.org/en/1.6.1/intro/why.html (дата звернення: 11.01.2022).

18) MongoDB Technical documentation. URL: https://docs.mongodb.com/manual/ (дата звернення: 12.01.2022).

19) NoSQL Database Couchbase. URL: http://www.couchbase.com/nosql-resources/what-is-no-sql (дата звернення: 13.01.2022).

20) Процеси і потоки. URL: https://tehnar.net.ua/protsesi-i-potoki/ (дата звернення: 23.01.2022).

21) What is Flutter: main features and 6 advantages for mobile development. URL: https://logap.com.br/en/blog/what-is-flutter/ (дата звернення: 23.04.2022).

22) Progit. URL: https://github.com/progit/progit2-uk (дата звернення: 30.04.2022).

23) What is Git. URL: https://www.atlassian.com/git/tutorials/what-is-git (дата звернення: 30.04.2022).

24) Масштабованість як вимога до програмного забезпечення. URL: https://uk.itpedia.nl/2021/07/20/schaalbaarheid-als-software-requirement-betekenis-en-definitie/ (дата звернення: 23.11.2021).

25) Spring Cloud Netflix Reference Doc. URL: https://cloud.spring.io/spring-cloud-netflix/2.2.x/reference/html/ (дата звернення: 25.04.2022).

26) Карта зоряного неба. URL: https://stellarium-web.org/ (дата звернення: 15.05.2022).

ДОДАТКИ

ДОДАТОК А

Лістинг програми

```
package ParallelSolarPanelsPackage;

import ParallelSolarPanelsPackage.Model.*;
import ParallelSolarPanelsPackage.Utils.StateUtils;
import org.springframework.http.HttpMethod;
import org.springframework.web.client.RestTemplate;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Random;

public class WorkProcess {
    private static WorkProcess workProcess;

    private WorkProcess(){
        panels = new ArrayList<PanelDTO>();
        station = null;
        index = 25;
    }

    public static WorkProcess getInstance(){
        if(workProcess == null){
            workProcess = new WorkProcess();
        }
        return workProcess;
    }

    public final String managementUrl = "http://management-
service/";
    public final String statisticsUrl = "http://statistics-
service/";
    public final String sunUrl = "http://sun-service/sun/power-
coef/";
    public final String dateTimeUrl = "http://sun-
service/sun/datetime/";

    public List<PanelDTO> panels;
    private RestTemplate restTemplate;
    private StateUtils stateUtils;
```

```
        public StationDTO station;

        private int index;

        private double mediumIterations = 0;
        private double mediumTime = 0;

        public void execute(){
            Date startTime = new Date();
            if(!(station == null)) {
                if(station.getStationConnection() == 1) {
                    List<Thread> threads = new ArrayList<>();
                    for (PanelDTO panel : panels) {
                        Thread thread = new Thread(() ->
doTaskForPanel(panel));
                        thread.start();
                        threads.add(thread);
                        if (threads.size() >= panels.size()) {
                            waitForThreads(threads);
                        }
                    }
                } else {
                    System.out.println("Station is disconnected");
                }
                updateGivenLogs();
                index += 1;

                Date endTime = new Date();
                processTime(endTime.getTime() - startTime.getTime());

                System.out.println("Task executed on " + new Date());
            }
        }

        private void processTime(long time) {
            System.out.println("Time to correct all panels: " +
time);

            int _index = index % 144;
            if (_index == 123) {
                int _day = index / 144;
                System.out.println("-------Day " + _day + ":  medium
time: " + (mediumTime / 84.0) + "------------");

                mediumTime = 0;
            }
```

```
            if(_index >= 36 && _index < 120) {
                mediumTime += time;
            }
        }

        public void doTaskForPanel(PanelDTO panel){
            if (panel.getConnected() == 1) {
                double power = getPanelPower(panel);
                if (power >= 10) {
                    preparePanel(panel);
                    findSun(panel);
                    updateProducedLogs(panel);
                } else {
                    turnPanelEast(panel);
                    System.out.println("No sun found");
                }
            } else {
                System.out.println("Panel " + panel.getName() + " is
disconnected");
            }
        }

        private void waitForThreads(List<Thread> threads) {
            for (Thread thread : threads) {
                try {
                    thread.join();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            threads.clear();
        }

        private void turnPanelEast(PanelDTO panel) {
            if(panel.getAzimuth() != 90 && panel.getAltitude() !=
10){
                panel.setAzimuth(90);
                panel.setAltitude(10);
                reduceForPanel(panel.getId());
                updatePanel(panel);
            }
        }

        public void findSun(PanelDTO panel) {
            double azPlus = 0;
```

```
            double azMinus = 0;
            double altPlus = 0;
            double altMinus = 0;

            int iterator = 0;
            while (azPlus >= 0 || azMinus >= 0 || altPlus >= 0 ||
altMinus >= 0) {
                double prevPower = getPanelPower(panel);
                Random random = new Random();

                StateDTO prevState = getState(panel);
                azPlus = prevState.getAzPlus();
                azMinus = prevState.getAzMinus();
                altPlus = prevState.getAltPlus();
                altMinus = prevState.getAltMinus();

                int code = 0;

                int randomInt = random.nextInt(20);
                if (randomInt == 5) {
                    code = random.nextInt(4);
                } else {
                    if (azPlus >= azMinus && azPlus >= altPlus &&
azPlus >= altMinus) {
                        code = 2;
                    }
                    if (azMinus >= azPlus && azMinus >= altPlus &&
azMinus >= altMinus) {
                        code = 3;
                    }
                    if (altPlus >= azMinus && altPlus >= azPlus &&
altPlus >= altMinus) {
                        code = 0;
                    }
                    if (altMinus >= azMinus && altMinus >= azPlus &&
altMinus >= altPlus) {
                        code = 1;
                    }
                }

                PreviousDTO previousDTO = new PreviousDTO();

                switch (code) {
                    case 0:
                        if (panel.getAltitude() < 90) {
                            panel.setAltitude(panel.getAltitude() +
```

```
1);
                                break;
                            } else {
                                previousDTO.setAltPlus(-1);
                                code++;
                            }

                        case 1:
                            if (panel.getAltitude() > 5) {
                                panel.setAltitude(panel.getAltitude() -
1);

                                break;
                            } else {
                                previousDTO.setAltMinus(-1);
                                code++;
                            }

                        case 2:
                            if (panel.getAzimuth() < 359) {
                                panel.setAzimuth(panel.getAzimuth() + 1);
                            } else {
                                panel.setAzimuth(0);
                            }
                            break;
                        case 3:
                            if (panel.getAzimuth() > 0) {
                                panel.setAzimuth(panel.getAzimuth() - 1);
                            } else {
                                panel.setAzimuth(359);
                            }
                            break;
                    }

                    StateDTO newState = getState(panel);
                    double newPower = getPanelPower(panel);

                    double diff = newPower - prevPower;

                    PreviousDTO currentDTO = new PreviousDTO();
                    previousDTO.setId(prevState.getId());
                    currentDTO.setId(newState.getId());

                    double k = 10;
                    switch (code) {
                        case 0:
                            previousDTO.setAltPlus(diff);
```

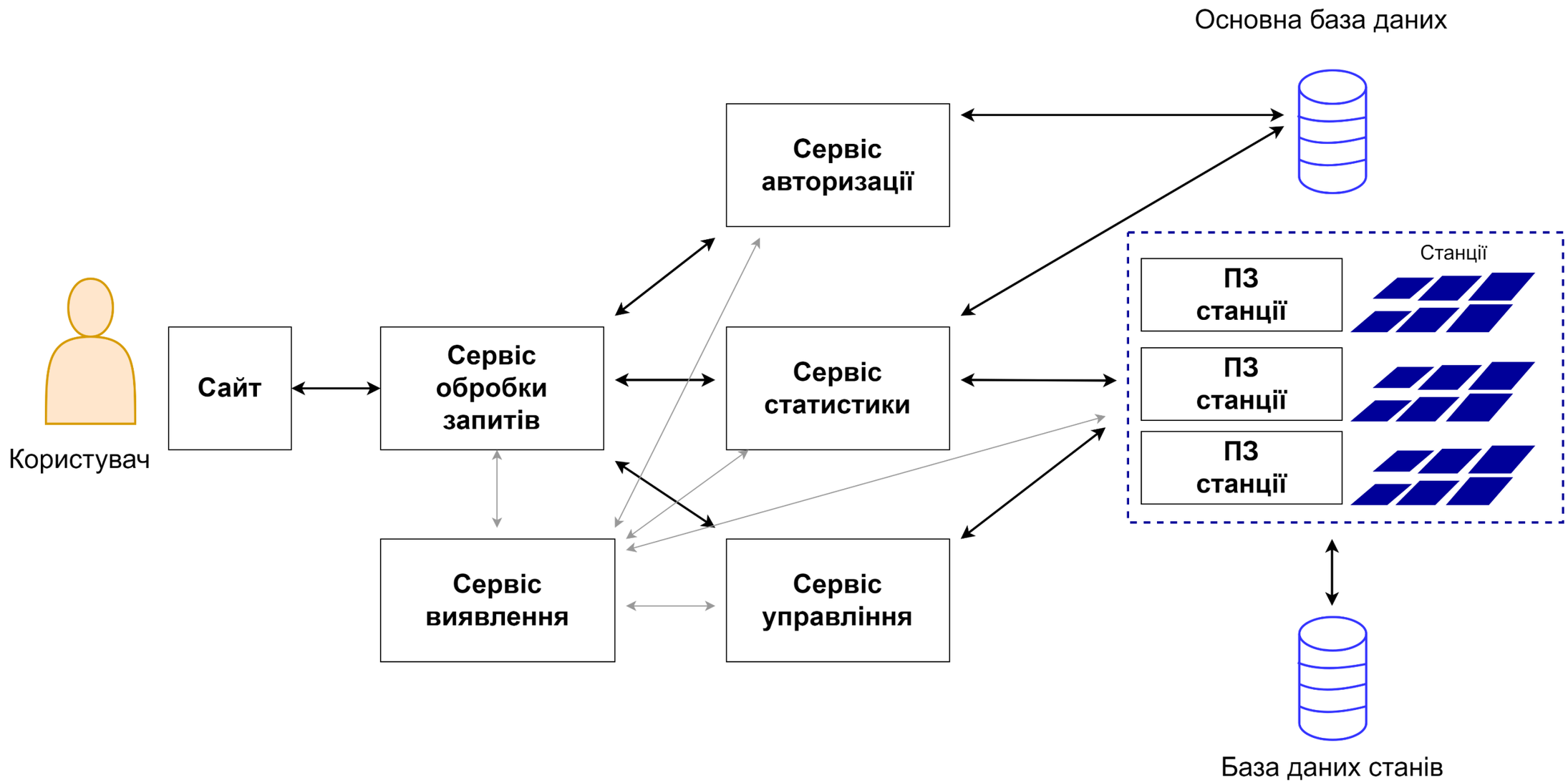
```
                    currentDTO.setAltPlus(diff / k);

                    previousDTO.setAltMinus(-diff / k);
                    currentDTO.setAltMinus(-diff / k / k);
                    break;
                case 1:
                    previousDTO.setAltMinus(diff);
                    currentDTO.setAltMinus(diff / k);

                    previousDTO.setAltPlus(-diff / k);
                    currentDTO.setAltPlus(-diff / k / k);
                    break;
                case 2:
                    previousDTO.setAzPlus(diff);
                    currentDTO.setAzPlus(diff / k);

                    previousDTO.setAzMinus(-diff / k);
                    currentDTO.setAzMinus(-diff / k / k);
                    break;
                case 3:
                    previousDTO.setAzMinus(diff);
                    currentDTO.setAzMinus(diff / k);

                    previousDTO.setAzPlus(-diff / k);
                    currentDTO.setAzPlus(-diff / k / k);
                    break;
            }

            sendUpdate(previousDTO);
            sendUpdate(currentDTO);

            iterator++;
        }

        updatePanel(panel);
        printPanelInfo(panel, iterator);
    }

    public double getTotalPower(){
        if(station.getStationConnection() == 0) {
            return 0;
        }

        double power = 0;
        for (PanelDTO panel : panels){
            power += getPanelPower(panel);
```

```
            }
            return power;
        }

        public double getPanelPower(PanelDTO panel){
            if (panel.getConnected() == 0 ||
station.getStationConnection() == 0){
                return 0;
            }
            double coef = 0;
            try {
                coef = restTemplate.postForObject(sunUrl + index, new
CoordinatesDTO(panel.getAzimuth(), 0,0,panel.getAltitude(),0,0),
Double.class);
            }
            catch (Exception e){
                coef = restTemplate.postForObject(sunUrl + "0", new
CoordinatesDTO(panel.getAzimuth(), 0,0,panel.getAltitude(),0,0),
Double.class);
                index = 0;
            }
            return coef > 0 ? coef * panel.getNominalPower() : 0;
        }

        public StateDTO getState(PanelDTO panel){
            StateDTO stateDTOSent = new StateDTO();
            stateDTOSent.setPanelId(panel.getId());
            stateDTOSent.setAzimuth(panel.getAzimuth());
            stateDTOSent.setAltitude(panel.getAltitude());

            return (stateUtils.fetchState(stateDTOSent)).toDTO();
        }

        public void sendUpdate(PreviousDTO previousDTO){
            stateUtils.updatePrevState(previousDTO);
        }

        public void updatePanel(PanelDTO panelDTO){
            try {
                Void response =
restTemplate.postForObject(managementUrl + "panels/" +
panelDTO.getId(), panelDTO, void.class);
            } catch (Exception e) {
                System.out.println("--------WARNING: Could not update
panel information----------");
            }
```

```
        }

        public void preparePanel(PanelDTO panelDTO){
            stateUtils.preparePanel(panelDTO);
        }

        public void reduceForPanel(String panelId){
            stateUtils.reduceByPanelId(panelId);
        }

        public void updateProducedLogs(PanelDTO panelDTO){
            LogDTO logDTO = new LogDTO();
            logDTO.setStationId(station.getId());
            logDTO.setPanelId(panelDTO.getId());
            String dateTime = restTemplate.exchange(dateTimeUrl +
index, HttpMethod.GET, null, String.class).getBody();
            logDTO.setDateTime(dateTime);
            logDTO.setProduced(getPanelPower(panelDTO) * 60 * 10);

            station.setEnergy(station.getEnergy() +
logDTO.getProduced());
            updateStation(station);
            updateLog(logDTO);
        }

        public void updateGivenLogs(){
            LogDTO logDTO = new LogDTO();
            logDTO.setStationId(station.getId());
            String dateTime = restTemplate.exchange(dateTimeUrl +
index, HttpMethod.GET, null, String.class).getBody();
            logDTO.setDateTime(dateTime);

            if (station.getGridConnection() == 1){
                double maxEnergyGiven = station.getMaxOutputPower() *
60 * 10;
                double additionalEnergy =
Math.min(station.getEnergy(), maxEnergyGiven);
                logDTO.setGiven(additionalEnergy);
                station.setEnergy(station.getEnergy() -
additionalEnergy);
            }
            else {
                logDTO.setGiven(0);
            }

            updateStation(station);
```

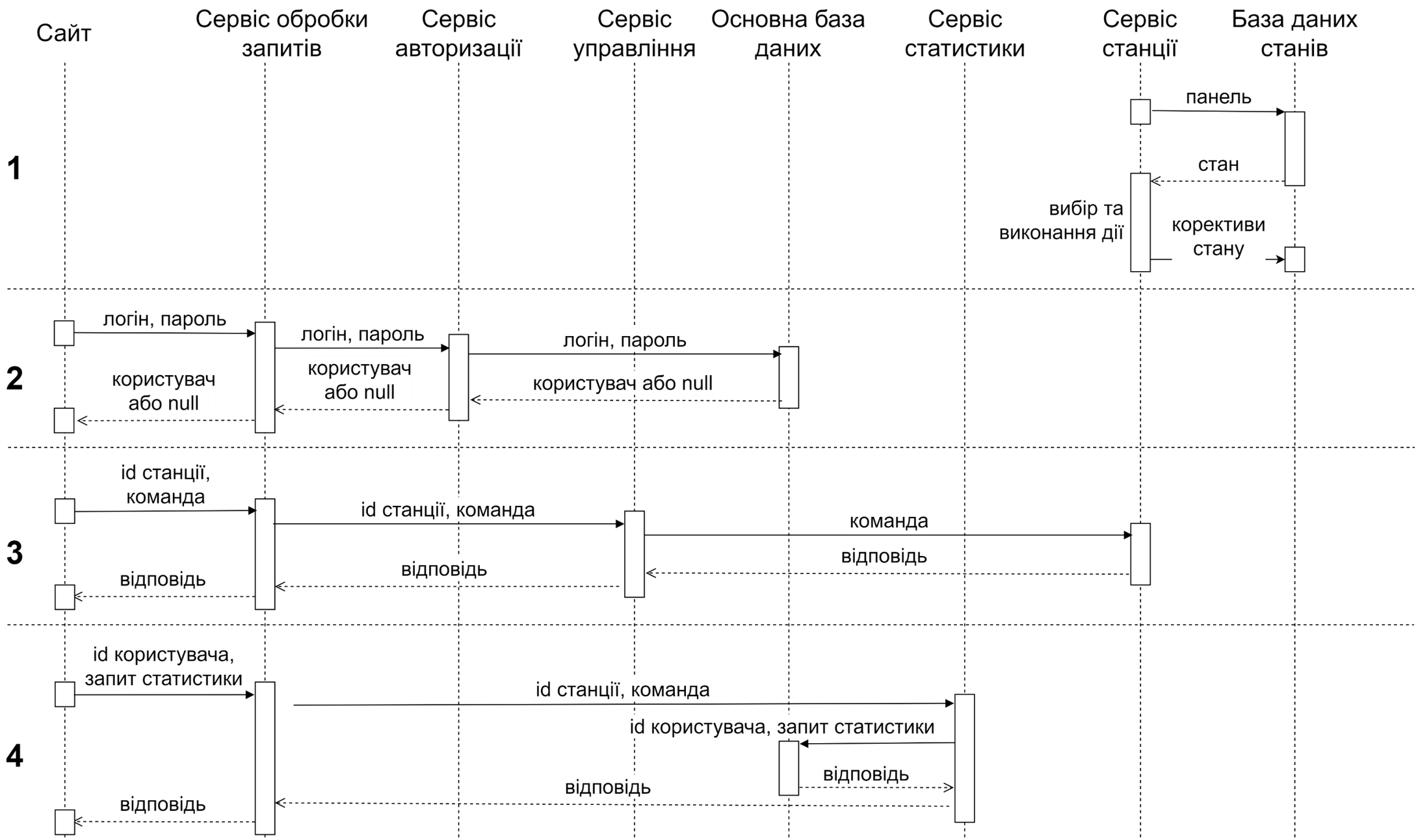
```
            updateLog(logDTO);
        }

        private void updateStation(StationDTO stationDTO) {
            try {
                Void response =
restTemplate.postForObject(managementUrl + "stations/" +
station.getId(), stationDTO, void.class);
            } catch (Exception e) {
                System.out.println("--------WARNING: Could not update
station information----------");
            }
        }

        private void updateLog(LogDTO logDTO){
            try {
                restTemplate.postForObject(statisticsUrl +
"logs/update/", logDTO, void.class);
            } catch (Exception e) {
                System.out.println("--------WARNING: Could not update
statistics----------");
            }
        }

        private void printPanelInfo(PanelDTO panel, int iterator){
            System.out.println("Panel " + panel.getName() + ": final
azimuth: " + panel.getAzimuth() + "; altitude: " +
panel.getAltitude() + "; index: " + index + "; power: " +
getPanelPower(panel));
            System.out.println("Iterations: " + iterator);

            int _index = index % 144;
            if (_index == 120) {
                int _day = index / 144;
                System.out.println("-------Day " + _day + ":  medium
iterations: " + (mediumIterations / panels.size() / 84.0) + "--------
----");

                mediumIterations = 0;
            }

            if(_index >= 36 && _index < 120) {
                mediumIterations += iterator;
            }
        }
```

```
        public String getDateTime() {
            String dateTime = restTemplate.exchange(dateTimeUrl +
index, HttpMethod.GET, null, String.class).getBody();
            return dateTime;
        }

        public RestTemplate getRestTemplate(){
            return restTemplate;
        }

        public void setRestTemplate(RestTemplate restTemplate) {
            this.restTemplate = restTemplate;
        }

        public StateUtils getStateUtils() {
            return stateUtils;
        }

        public void setStateUtils(StateUtils stateUtils) {
            this.stateUtils = stateUtils;
        }
    }
```



					IT.01.07.001 ВЗ		
Зм.	Лист	№ докум	Підпис	Дата	Архітектура розробленої системи	Літ.	Маса	Мірило
Розроб.		Мокрий А.В.				Лист	Листів
Перев.		Баклан І.В.					
Н.контр.		Баклан І.В.			Кафедра Інформатики та програмної інженерії	Група ІТ-01мн	
Затв.		Жаріков Е.В.					

Підпис і дата
Інв. № дубл.
Взам. інв. №
Підпис і дата
Інв. № ориг.

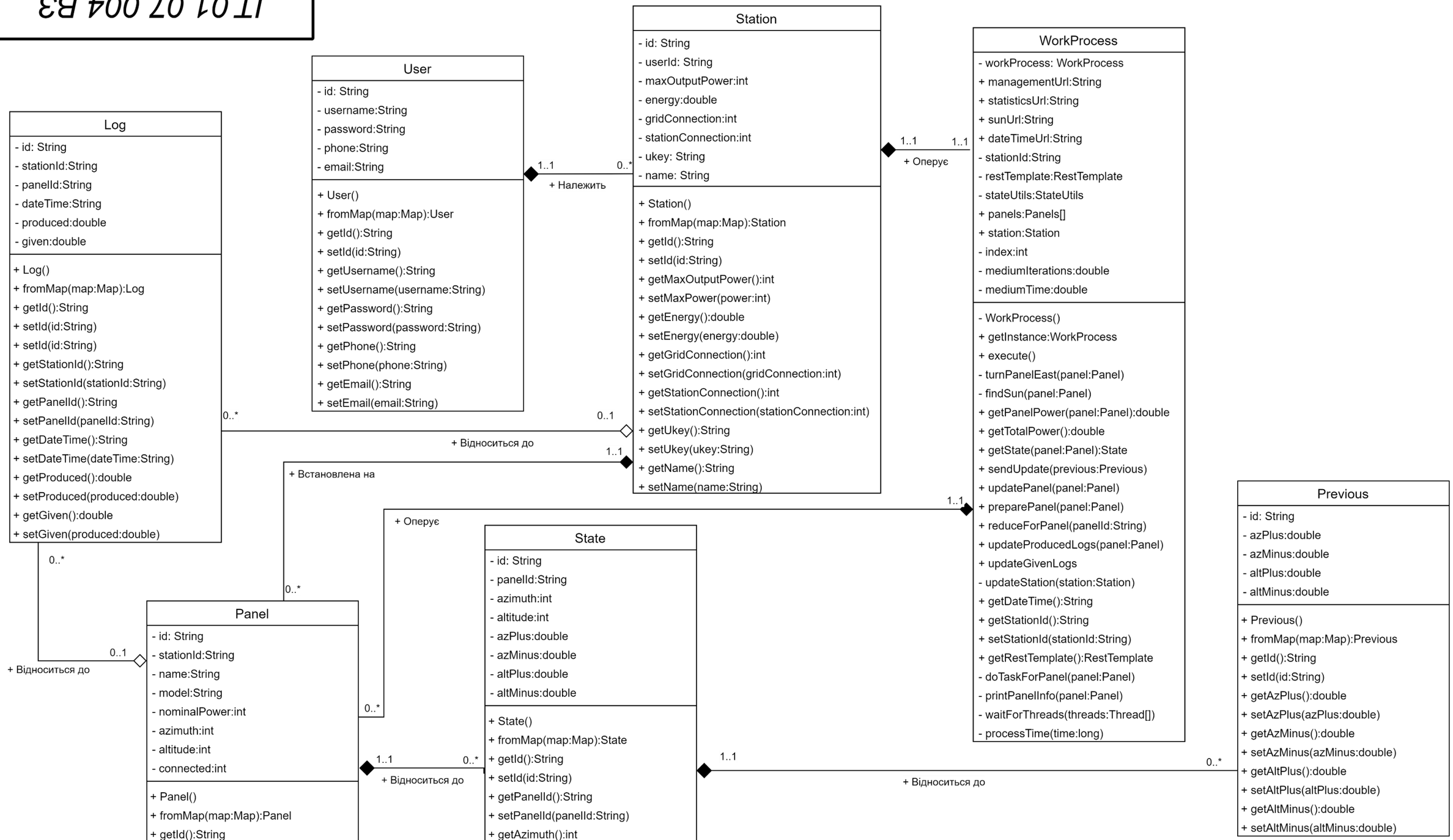


					IT.01.07.002 ВЗ		
Зм.	Лист	№ докум	Підпис	Дата	Схема інформаційних потоків системи	Літ.	Маса	Мірило
Розроб.		Мокрий А.В.						
Перев.		Баклан І.В.				Лист	Листів	
Н.контр.		Баклан І.В.			Кафедра Інформатики та програмної інженерії	Група ІТ-01мн		
Затв.		Жаріков Е.В.						

Підпис і дата
Інв. № дубл.
Взам. інв. №
Підпис і дата
Інв. № ориг.

					ІТ.01.07.003 ВЗ		
Зм.	Лист	№ докум	Підпис	Дата	Схема алгоритму q-навчання	Літ.	Маса	Мірило
Розроб.		Мокрий А.В.						
Перев.		Баклан І.В.				Лист	Листів	
Н.контр.		Баклан І.В.			Кафедра Інформатики та програмної інженерії	Група ІТ-01мн		
Затв.		Жаріков Е.В.						

Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.



Зм.	Лист	№ докум	Підпис	Дата
Розроб.		Мокрий А.В.		
Перев.		Баклан І.В.		
Н.контр.		Баклан І.В.		
Затв.		Жаріков Е.В.		

ІТ.01.07.004 ВЗ

Діаграма класів сервісу станції

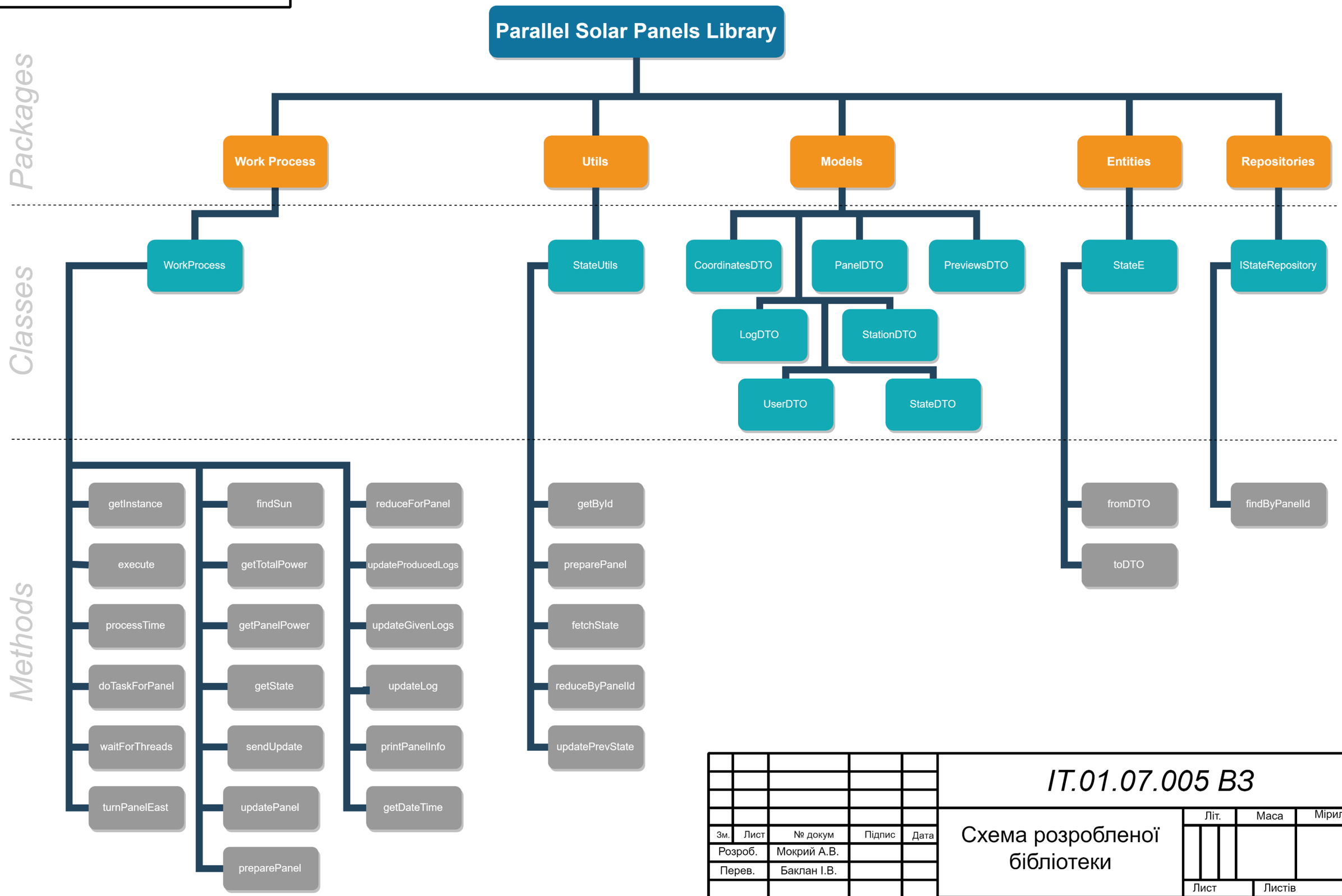
Літ.	Маса	Мірило

Лист | Листів

Кафедра Інформатики та програмної інженерії

Група ІТ-01мн

Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.



Підпис і дата	Інв. № дубл.	Взам. інв. №	Підпис і дата	Інв. № ориг.

Зм.	Лист	№ докум	Підпис	Дата
Розроб.		Мокрий А.В.		
Перев.		Баклан І.В.		
Н.контр.		Баклан І.В.		
Затв.		Жаріков Е.В.		

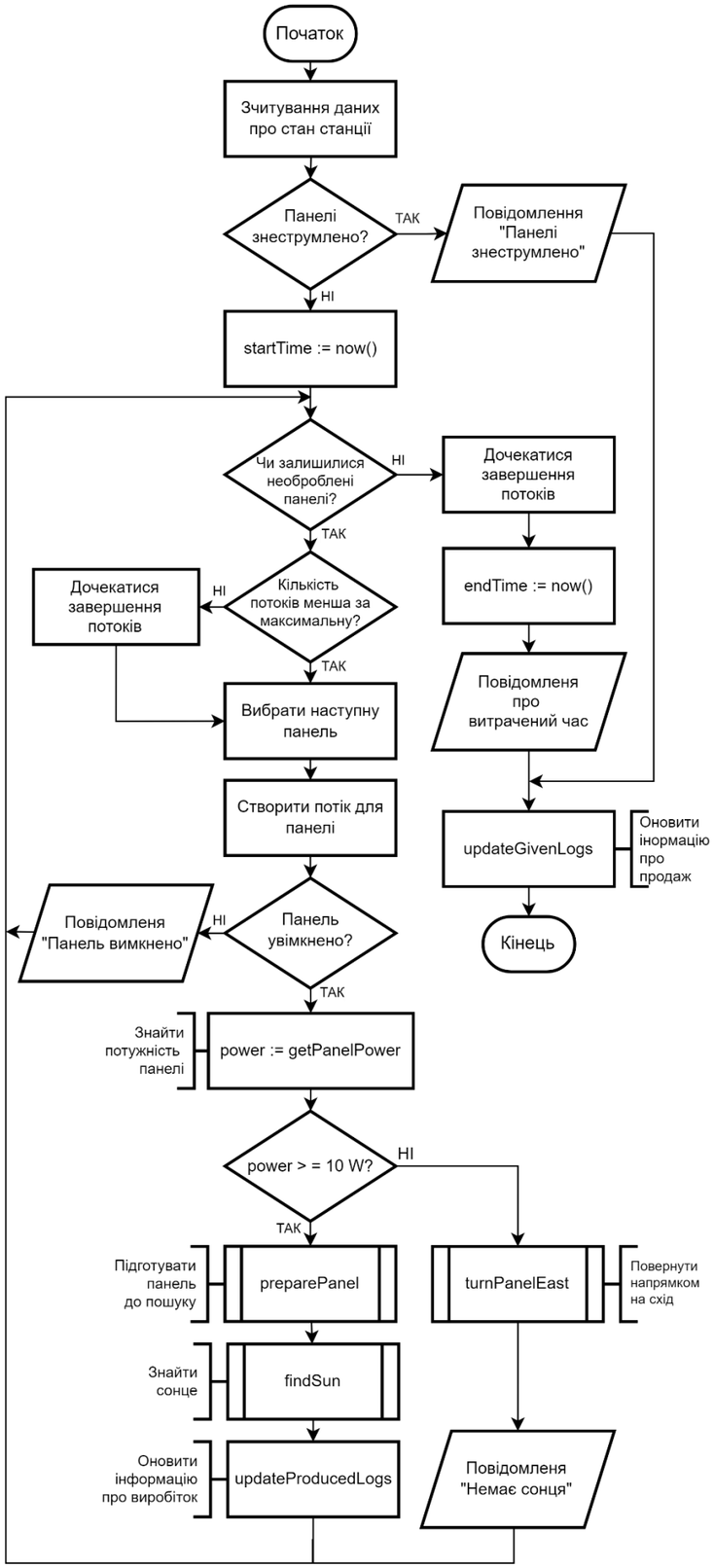
IT.01.07.005 B3

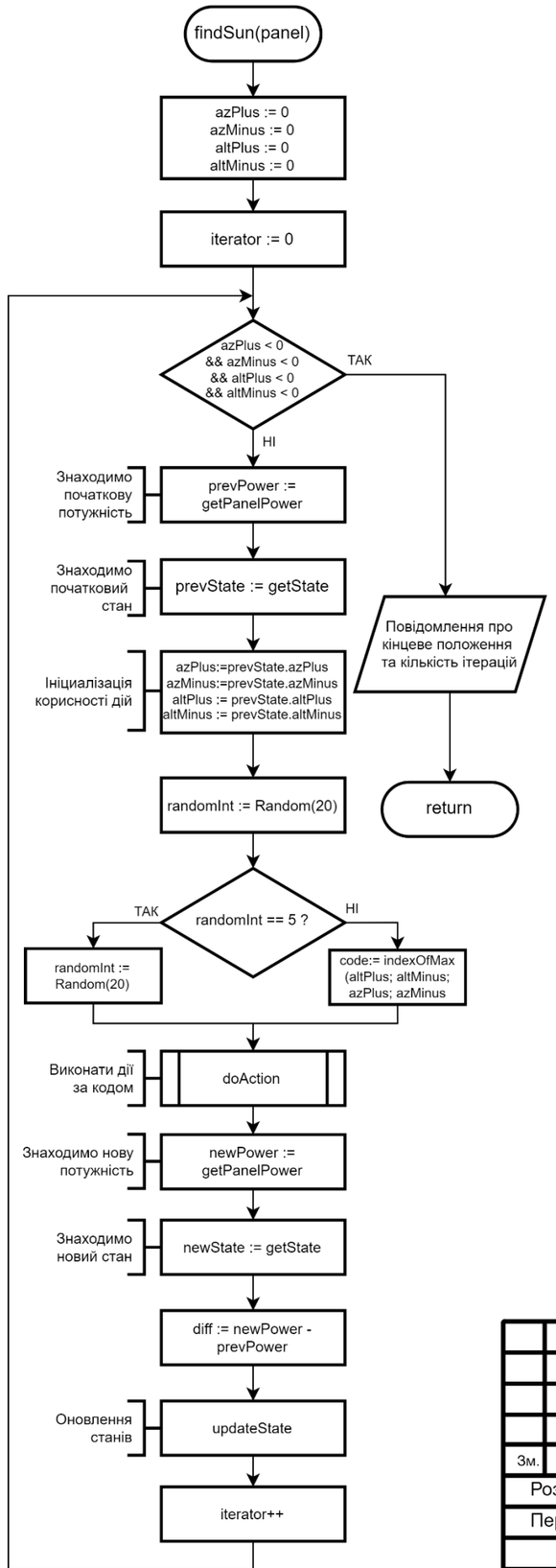
Схема розробленої бібліотеки

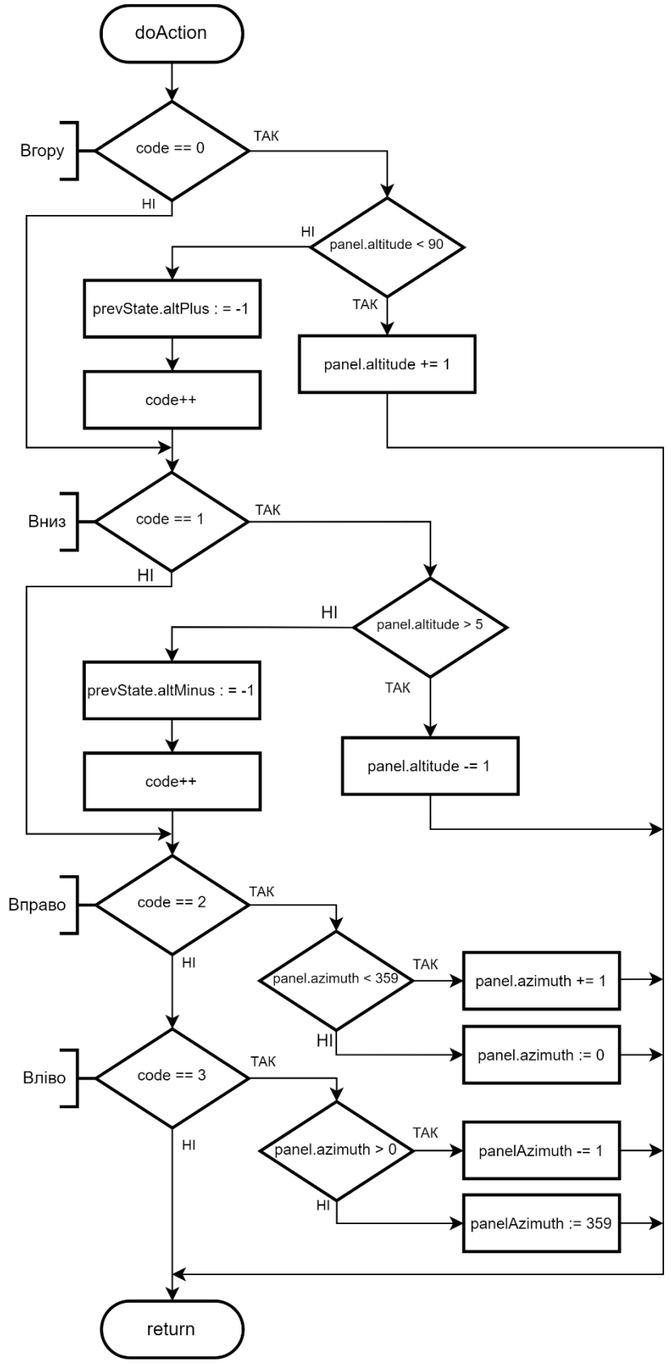
Літ.	Маса	Мірило
Лист	Листів	

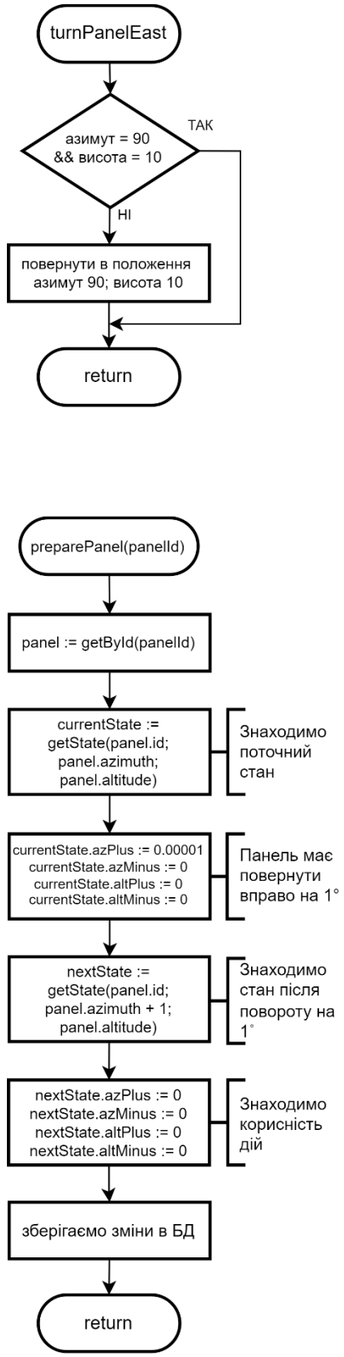
Кафедра Інформатики та програмної інженерії

Група ІТ-01мн









Зм.	Лист	№ докум	Підпис	Дата
Розроб.		Мокрий А.В.		
Перев.		Баклан І.В.		
Н.контр.		Баклан І.В.		
Затв.		Жаріков Е.В.		

IT.01.07.006 ВЗ

Алгоритм роботи сервісу станції

Літ.	Маса	Мірило
Лист	Листів	

Кафедра Інформатики та програмної інженерії

Група ІТ-01мн

Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

ДОДАТОК 3

Результат перевірки роботи на співпадіння

UNICHECK

Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1011328808

Дата проверки:
25.05.2022 07:33:30 EEST

Тип проверки:
Doc vs Internet + Library

Дата отчета:
25.05.2022 07:38:06 EEST

ID пользователя:
76913

Название файла: **IT-01мн_Мокрий_ПЗ**

Количество страниц: 59 **Количество слов:** 12166 **Количество символов:** 93444 **Размер файла:** 135.46 KB **ID файла:** 1011214956

3.96% Совпадения

Наибольшее совпадение: 1.73% с **источником из Библиотеки** (ID **файла:** 1003805001)

1.32% Источники из Интернета | 28 ..... Страница 61

3.58% Источники из Библиотеки | 150 ..... Страница 61

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников