НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Кафедра інформаційних систем та технологій

Тема_____System activity monitor_____

Курсова робота
З дисципліни «Технології розроблення програмного забезпечення»

Керівник
доц. Амонс О.А.
«Допущений до захисту»
_____
(Особистий підпис керівника)
« » _____2025р.
Захищений з оцінкою
_____
(оцінка)

Виконавець
ст. Сльота М. Б.
залікова книжка № ___ – _____
гр. ІА-34
_____
(особистий підпис виконавця)
« »_____2025р.

Члени комісії:

_____ _____
(особистий підпис) (розшифровка підпису)

_____ _____
(особистий підпис) (розшифровка підпису)

Київ – 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

(назва навчального закладу)

Кафедра ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Дисципліна «Технології розроблення програмного забезпечення»

Курс_3_ Група___ІА-34____Семестр____5____

ЗАВДАННЯ

на курсову роботу студента

Сльота Максим Богданович

(прізвище, ім’я, по батькові)

1.Тема роботи__System_activity_monitor

2. Строк здачі студентом закінченої роботи__25.11.2025

3. Вихідні дані до роботи:

Монітор активності системи повинен зберігати і запам'ятовувати статистику використовуваних компонентів системи, включаючи навантаження на процесор, обсяг займаної оперативної пам'яті, натискання клавіш на клавіатурі, дії миші (переміщення, натискання), відкриття вікон і зміна вікон; будувати звіти про використання комп'ютера за різними критеріями (% часу перебування у веб-браузері, середнє навантаження на процесор по годинах, середній час роботи комп'ютера по днях і т.д.); правильно поводитися з «простоюванням» системи – відсутністю користувача.

4. Зміст розрахунково – пояснювальної записки (перелік питань, що підлягають розробці)

Аналіз задачі моніторингу активності користувача, розробка архітектури та бази даних, реалізація збору системної статистики, формування звітів і створення графічного інтерфейсу.

Додатки:

5.Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

6. Дата видачі завдання_16.09.2025

КАЛЕНДАРНИЙ ПЛАН

№, п/п	Назва етапів виконання курсової роботи	Строк виконання етапів роботи	Підписи або примітки
1.	Підбір та вивчення літератури	30.09.2025	
2.	Проектування та написання розділу 1	31.10.2025	
3.	Розробка та написання розділу 2	20.11.2025	
4.	Подання курсової роботи на перевірку	25.11.2025	
5.	Захист курсової роботи	08.12.2025	
6.			
7.			
8.			
9.			
10.			
11.			
12.			
13.			
14.			
15.			
16.			
17.			
18.			

Студент _____ _____Максим СЛЬОТА__
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник_____ __Олександр АМОНС__
(підпис) (Ім’я ПРІЗВИЩЕ)

«____»_____ 20___ р.

ЗМІСТ

ВСТУП

На сьогоднішній день комп’ютери використовуються скрізь – у навчанні, роботі та для виконання щоденних завдань. Зі зростанням кількості програм і процесів, які одночасно працюють у системі, збільшується навантаження на апаратні ресурси. Користувач часто навіть не помічає, як окремі програми споживають надмірну кількість ресурсів. Усі ці фактори впливають на стабільність і продуктивність системи. Саме тому виникає потреба у створенні програмного забезпечення, яке дозволить відстежувати активність системи в реальному часі, аналізувати навантаження та ефективність її використання.

Суть розробки системи моніторингу активності полягає в тому, що вона дає змогу користувачеві контролювати стан комп’ютера, своєчасно виявляти проблеми та запобігати нераціональному використанню ресурсів. Така система може бути корисною як звичайним користувачам, так і системним адміністраторам.

Метою даної курсової роботи є розробка системи, яка забезпечує збір та збереження даних про роботу компонентів комп’ютера. Для досягнення цієї мети система повинна мати певний набір властивостей і виконувати такі завдання:

- система повинна відстежувати навантаження на процесор, використання оперативної пам’яті, обсяг сховища та активність користувача;
- система повинна формувати звіти за різними критеріями, зокрема за середнім навантаженням, часом роботи системи та відсотком простою;
- забезпечувати коректну роботу у разі переходу системи в режим простою.

Розроблення такої системи дозволить підвищити ефективність і стабільність роботи комп’ютера, а також зробить його використання більш контрольованим і зручним для користувача.

1 ПРОЄКТУВАННЯ СИСТЕМИ

1.1. Огляд існуючих рішень

На даний час існує велика кількість програмного забезпечення, яке дозволяє здійснювати моніторинг системних ресурсів комп’ютера. Більшість із них надає користувачеві можливість переглядати показники завантаження процесора, використання оперативної пам’яті та активність дискової підсистеми. Проте кожне рішення має свої особливості, переваги й недоліки.

Найвідомішим інструментом є «Диспетчер завдань», який вбудований в операційну систему Windows. Він надає базову інформацію про активні процеси, рівень використання процесора, оперативної пам’яті, мережі та сховища. Однак «Диспетчер завдань» не дозволяє зберігати історію навантаження або будувати детальні звіти за певний період.

Ще одним прикладом є HWMonitor – програмне забезпечення, яке спеціалізується переважно на контролі апаратної частини комп’ютера. Воно дозволяє відстежувати температуру компонентів, напругу та швидкість обертання вентиляторів, проте не призначене для аналізу активності користувача чи побудови статистики щодо роботи програм.

Також варто згадати Process Explorer – це більш розширена версія «Диспетчера завдань», яка дозволяє переглядати детальні характеристики процесів, їхні залежності та поточне використання ресурсів. Проте ця програма орієнтована переважно на технічних спеціалістів і не має можливості формувати аналітичні звіти або відстежувати активність користувача.

Інші програмні продукти, зокрема AIDA64, також не повністю відповідають вимогам комплексного моніторингу. Хоча вони надають детальну інформацію про стан системи, такі програми не фіксують поведінку користувача (натискання клавіш, рухи миші, відкриття вікон) і не враховують час простою.

Отже, більшість існуючих рішень не забезпечують повного аналізу взаємодії користувача із системою. Вони не дозволяють об’єднати дані про апаратні ресурси та поведінкову активність у єдину базу для подальшої побудови звітів.

Розроблювана в межах цієї курсової роботи система моніторингу активності покликана усунути ці недоліки. Вона має об’єднати відстеження навантаження на процесор, використання оперативної пам’яті, активність миші й клавіатури, а також час роботи та простою комп’ютера. Це дозволить створити більш повну картину взаємодії користувача із системою та забезпечити можливість побудови статистичних звітів за обраний період.

1.2. Загальний опис проєкту

Розроблювана система моніторингу активності комп’ютера призначена для збору, обробки та збереження інформації про стан системних ресурсів і дії користувача. Основна ідея полягає в тому, щоб створити програму, яка дозволить у реальному часі відстежувати навантаження на процесор, використання оперативної пам’яті, обсяг сховища, активність користувача та загальний час роботи комп’ютера. На основі отриманих даних система зможе формувати звіти й надавати користувачеві зібрану інформацію у зручному вигляді.

Система міститиме кілька основних компонентів: модуль збору даних, базу даних для збереження статистики, сервіси для обробки та аналізу інформації, а також інтерфейс користувача для перегляду результатів. Модуль збору даних працюватиме у фоновому режимі, регулярно отримуючи показники навантаження на процесор і пам’ять, визначаючи активне вікно та фіксуючи активність користувача. Усі дані передаватимуться до бази даних, де зберігатимуться з можливістю подальшого формування звітів.

Проєкт розробляється з використанням мови програмування Java, середовища JavaFX для створення графічного інтерфейсу та системи керування базами даних MySQL. Використання такого набору технологій дозволить створити ефективний інструмент для контролю стану комп’ютера, аналізу ефективності його використання та підвищення продуктивності роботи користувача.

1.3. Вимоги до застосунків системи

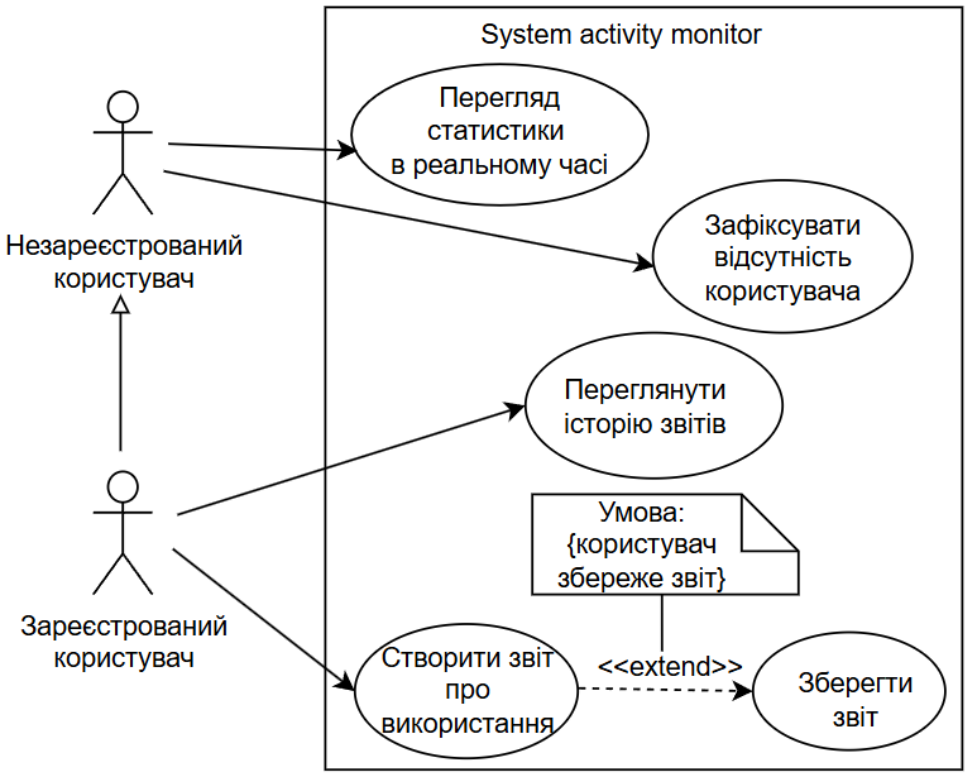


Рис 1. Діаграма варіантів використання

1.3.1. Функціональні вимоги до системи

Функціональні вимоги, які повинна забезпечувати система моніторингу активності відповідно до діаграми варіантів використання:

- незареєстрований користувач повинен мати можливість у реальному часі відстежувати навантаження на центральний процесор, визначати обсяг використаної оперативної пам’яті, показувати загальний обсяг пам’яті на диску, кількість вільного місця;
- незареєстрований користувач повинен мати можливість переглядати яке саме вікно є активним у поточний момент;
- незареєстрований користувач повинен мати можливість переглядати кількість натиснутих клавіш, кліки миші та переміщення курсора;
- незареєстрований користувач повинен мати можливість переглядати загальний час роботи операційної системи від моменту запуску та зафіксувати періоди відсутності користувача;

– зареєстрований користувач повинен мати усі можливості, що є у незареєстрованого користувача, а також він повинен мати можливість створювати звіти про використання, переглянути свою історію звітів і зберігати звіти.

1.3.2. Нефункціональні вимоги до системи

Нефункціональні вимоги, які повинна забезпечувати система моніторингу активності:

– система повинна працювати у фоновому режимі;
– у разі виникнення збоїв або помилок система має коректно відновлювати роботу без втрати основних даних;
– система повинна мати веб-інтерфейс;
– інтерфейс користувача має бути інтуїтивно-зрозумілим і зручним;
– система не повинна передавати зібрані дані стороннім сервісам;

1.4. Сценарії використання системи

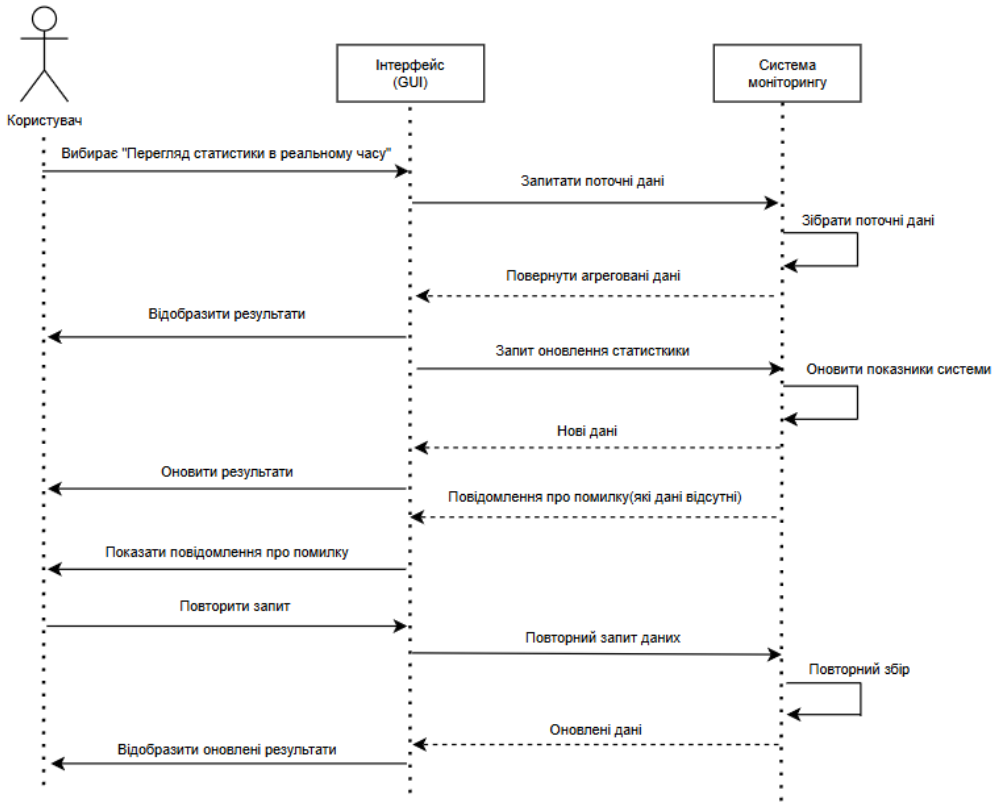


Рис 2. Діаграма послідовностей

Варіант використання: Перегляд статистики в реальному часі

Передумови: Користувач має доступ до функції моніторингу.

Постумови: Користувач отримує оновлені дані про стан системи (використання процесора, оперативної пам’яті, кількість натискань клавіш і кліків миші, активність вікон, час простою тощо).

Взаємодіючі сторони: Користувач, Інтерфейс (GUI), Система моніторингу.

Короткий опис: Користувач запускає режим перегляду статистики в реальному часі, після чого інтерфейс надсилає запит до системи моніторингу для збору поточних даних. Система повертає зібрану інформацію, яка відображається у вікні програми. У разі виникнення помилки користувач отримує повідомлення та може повторити запит.

Основний потік подій:

1. Користувач у графічному інтерфейсі обирає функцію «Перегляд статистики в реальному часі».
2. Інтерфейс надсилає запит через кнопку «Старт» до системи моніторингу на отримання поточних даних.
3. Система моніторингу збирає актуальні показники (CPU, RAM, активні вікна, дії користувача) і повертає агреговані результати інтерфейсу.
4. Інтерфейс відображає отриману статистику у зручному вигляді (текстові значення, графіки або діаграми).
5. Через певний інтервал часу інтерфейс надсилає запит на оновлення статистики, після чого система моніторингу виконує оновлення показників системи.
6. Нові дані повертаються до інтерфейсу, який оновлює результати на екрані користувача.

Виняткові ситуації:

Виняток №1: Якщо система моніторингу не може зчитати деякі дані, вона надсилає повідомлення про помилку до інтерфейсу.

Виняток №2: Інтерфейс показує повідомлення про помилку, інформуючи користувача про відсутність або недоступність даних.

Виняток №3: Користувач може натиснути «Повторити запит», після чого система виконує повторний збір даних і оновлює відображені результати.

Примітка: Оновлення статистики відбувається автоматично через встановлений інтервал часу, забезпечуючи безперервний моніторинг стану системи.

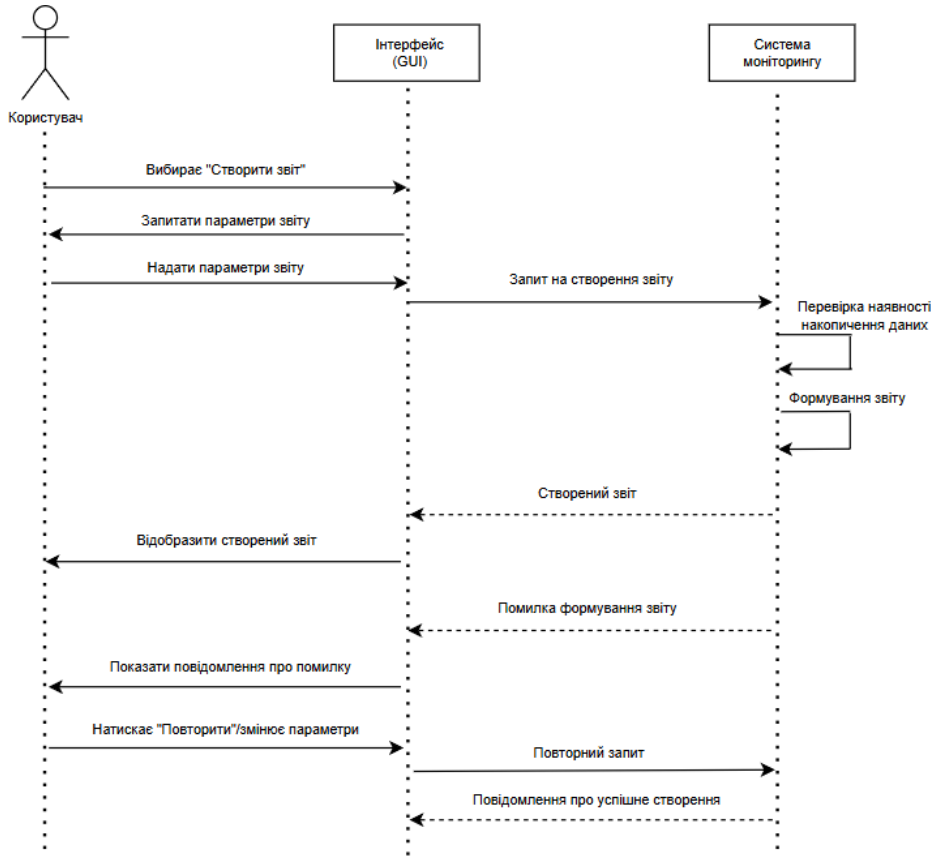


Рис 3. Діаграма послідовностей

Варіант використання: Створення звіту про використання системи

Передумови: Користувач повинен увійти у свій власний аккаунт у системі моніторингу. У базі даних вже накопичена статистика за певний період (CPU, RAM, активність користувача, простої, тощо).

Постумови: Після успішного виконання процесу система формує звіт за заданими параметрами, який відображається користувачу у графічному інтерфейсі та може бути збережений для подальшого перегляду.

Взаємодіючі сторони: Користувач, Інтерфейс (GUI), Система моніторингу.

Короткий опис: Користувач ініціює створення звіту, надає параметри для фільтрації (період, тип даних тощо), після чого система моніторингу перевіряє

наявність необхідних даних і формує звіт. У разі помилки система повідомляє користувача, який може змінити параметри або повторити спробу.

Основний потік подій:

1. Користувач у графічному інтерфейсі обирає функцію «Створити звіт».
2. Інтерфейс надсилає запит до системи моніторингу, щоб отримати необхідні параметри для створення звіту.
3. Користувач надає параметри звіту (наприклад, період часу).
4. Інтерфейс передає ці параметри до системи моніторингу у вигляді запиту на створення звіту.
5. Система моніторингу перевіряє наявність накопичених даних у базі.
6. Якщо дані доступні, система формує звіт і передає його до інтерфейсу.
7. Інтерфейс відображає створений звіт користувачу.

Виняткові ситуації:

Виняток №1: Якщо система виявляє відсутність необхідних даних, вона надсилає повідомлення про помилку до інтерфейсу. Інтерфейс відображає повідомлення про помилку користувачу з пропозицією змінити параметри звіту.

Виняток №2: Якщо виникає помилка під час формування звіту, система надсилає повідомлення про помилку, після чого користувач може натиснути «Повторити» або змінити параметри запиту.

Варіант використання: Зафіксувати відсутність користувача

Примітки: Після успішного формування звіту система надсилає повідомлення про успішне створення, а користувач може переглянути або зберегти звіт локально через інтерфейс.

Передумови: Користувач перебуває в активному режимі (онлайн), система моніторингу активна.

Постумови: Система фіксує час початку простою, а після повернення користувача - час завершення. Дані зберігаються в базу даних для подальшого аналізу.

Взаємодіючі сторони: Користувач, Інтерфейс (GUI), Система моніторингу.

Короткий опис: Користувач вручну керує своїм статусом активності через інтерфейс. Після натискання кнопки “Офлайн” система починає відлік часу простою. Коли користувач натискає “Онлайн”, система завершує фіксацію простою і записує дані до бази.

Основний потік подій:

1. Користувач натискає кнопку “Офлайн” у GUI.
2. Система фіксує час початку простою.
3. Користувач повертається і натискає “Онлайн”.
4. Система фіксує час завершення простою.
5. Дані про тривалість простою зберігаються у базу даних.

Виняткові ситуації:

Виняток №1: Якщо під час простою система не може записати дані у базу, користувачу відображається повідомлення про помилку.

Варіант використання: Переглянути історію звітів

Передумови: Користувач авторизований у системі.

Постумови: Користувач отримує список створених раніше звітів із можливістю перегляду деталей кожного.

Взаємодіючі сторони: Користувач, Інтерфейс (GUI), Система моніторингу.

Короткий опис: Користувач може відкрити розділ історії звітів і переглянути збережені файли або інформацію про попередні звіти.

Основний потік подій:

1. Користувач відкриває розділ “Історія звітів” у графічному інтерфейсі.
2. GUI надсилає запит до системи моніторингу.
3. Система зчитує дані про збережені звіти з бази даних.
4. Система повертає список звітів до інтерфейсу.
5. Користувач переглядає список і може відкрити будь-який звіт для детального перегляду.

Винятки:

Виняток №1: Якщо звіти відсутні, система виводить повідомлення “Історія порожня”.

Варіант використання: Зберегти звіт

Передумови: Користувач створив звіт про використання системи.

Постумови: Звіт успішно збережено у файловій системі або базі даних, і доступний для перегляду в історії.

Взаємодіючі сторони:Користувач, Інтерфейс (GUI), Система моніторингу.

Короткий опис: Після формування звіту користувач має можливість зберегти його для подальшого перегляду або експорту.

Основний потік подій:

1. Користувач натискає кнопку “Зберегти звіт”.
2. Інтерфейс надсилає запит на збереження до системи моніторингу.
3. Система перевіряє коректність сформованого звіту.
4. Користувачу відображається повідомлення про успішне збереження.

Винятки:

Виняток №1: Якщо виникла помилка при записі файлу, система пропонує повторити спробу або обрати інший каталог.

Примітки: Збережений звіт з’являється в “Історії звітів” і може бути відкритий повторно.

1.5. Концептуальна модель системи

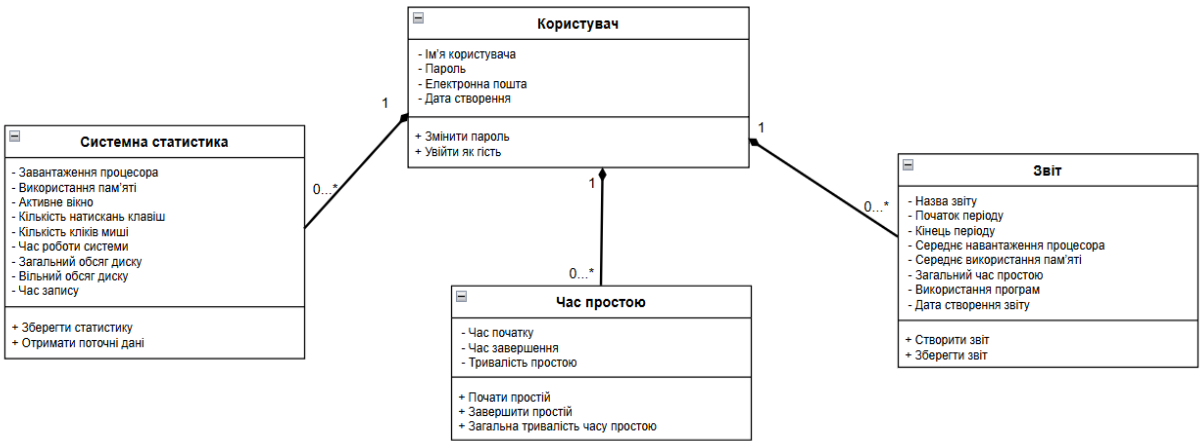


Рис 4. Діаграма класів

У центрі діаграми розташований клас «Користувач», який є головною сутністю системи. Він містить інформацію про ім’я користувача, пароль, електронну пошту та дату створення облікового запису. Користувач має можливість змінити пароль, а також увійти до системи як гість. Саме користувач ініціює процеси моніторингу системної активності, фіксації простою та створення звітів.

Клас «Системна статистика» відповідає за збирання даних про поточний стан комп’ютера. У ньому зберігається інформація про завантаження процесора, використання оперативної пам’яті, активне вікно, кількість натискань клавіш і кліків миші, час роботи системи, обсяг вільного і зайнятого дискового простору, а також час запису даних. Цей клас пов’язаний із користувачем відношенням «один до багатьох», оскільки один користувач може мати безліч записів статистики. Методи класу дозволяють зберігати нові показники системи та отримувати поточні дані в реальному часі. Таким чином, саме через цей клас здійснюється постійний моніторинг стану системи.

Клас «Час простою» описує періоди неактивності користувача, коли він не взаємодіє з комп’ютером. У ньому зберігаються дані про час початку та завершення простою, а також про його тривалість у секундах. Користувач може розпочати або завершити період простою вручну. Клас має асоціацію «один до багатьох» із користувачем, що означає, що для одного користувача може бути збережено багато інтервалів неактивності.

Клас «Звіт» призначений для збереження зведеної інформації про роботу користувача за вибраний період. Він містить атрибути: назва звіту, початок і кінець періоду, середнє навантаження процесора, середнє використання пам’яті, загальний час простою, використання програм та дату створення звіту. Методи класу дозволяють створювати нові звіти і зберігати, які формуються на основі даних класів «Системна статистика» та «Час простою». Зв’язок між класами «Користувач» і «Звіт» має тип «один до багатьох», оскільки кожен користувач може створювати кілька звітів за різні часові періоди.

1.6. Вибір бази даних

Для розробки системи моніторингу активності комп’ютера було обрано систему керування базами даних MySQL, оскільки це одна з найпоширеніших реляційних СКБД, зручна у використанні з мовою програмування Java. MySQL добре підходить для проєктів різної складності, забезпечуючи високу швидкість роботи навіть за значного обсягу інформації.

MySQL характеризується високою продуктивністю обробки запитів, що особливо важливо для систем, які постійно записують оновлені дані. Крім того, вона дозволяє легко масштабувати базу без змін у логіці застосунку, що є суттєвою перевагою у разі розширення функціональності системи.

У межах даної курсової роботи база даних MySQL використовується для збереження інформації про користувачів, статистику використання процесора, оперативної пам’яті, часу простою, активності клавіатури та миші, а також для формування звітів на основі зібраних даних.

1.7. Вибір мови програмування та середовища розробки

Для розробки системи моніторингу активності комп’ютера було обрано мову програмування Java та інтегроване середовище розробки IntelliJ IDEA. Мова Java забезпечує незалежність від операційної системи та дозволяє запускати програму на будь-якому комп’ютері без зміни коду завдяки використанню Java Virtual Machine.

Також Java підтримує велику кількість бібліотек і фреймворків, що значно спрощує процес розробки. Крім того, вона надає можливість легкої інтеграції з базами даних через технологію JDBC.

Як середовище розробки було обрано IntelliJ IDEA, оскільки воно має зручний інтерфейс, підтримує Maven і Git, а також містить вбудовані інструменти для роботи з базами даних. Завдяки цьому процес розробки системи стає швидшим, структурованішим і зручнішим.

1.8. Проєктування розгортання системи

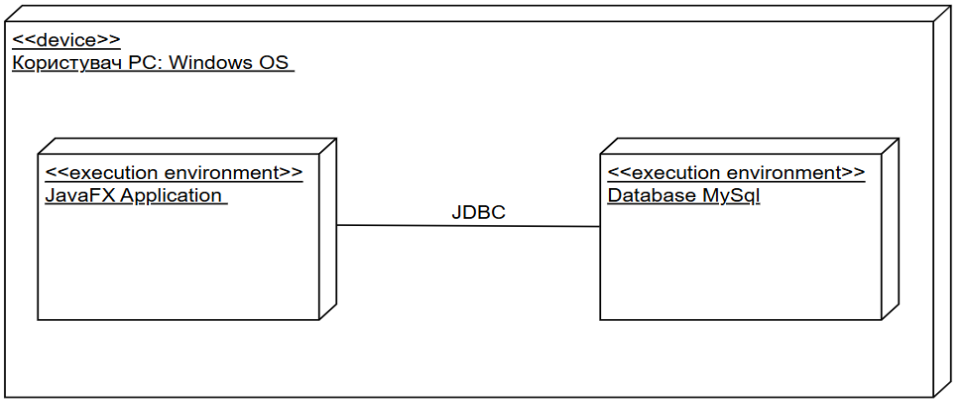


Рис 5. Діаграма розгортання

Діаграма розгортання містить один фізичний пристрій – комп'ютер користувача, позначений як <<device>> Користувач PC: Windows OS. На ньому розгортається та виконується вся система. Операційна система Windows є платформою, на якій запускаються обидва програмні середовища.

Клієнтський застосунок JavaFX Application, позначений як <<execution environment>> JavaFX Application, має графічний інтерфейс, модулі збору системної інформації та сервіси, що відповідають за обробку й відображення даних.

Другий елемент – <<execution environment>> Database MySQL – представляє середовище виконання бази даних MySQL, яка також розміщується на комп'ютері користувача. У ній зберігаються всі зібрані дані: інформація про користувачів, статистика використання процесора, пам'яті, активності користувача та звіти.

Між двома середовищами виконання встановлено зв'язок JDBC, який забезпечує обмін даними між Java-додатком і базою даних MySQL.

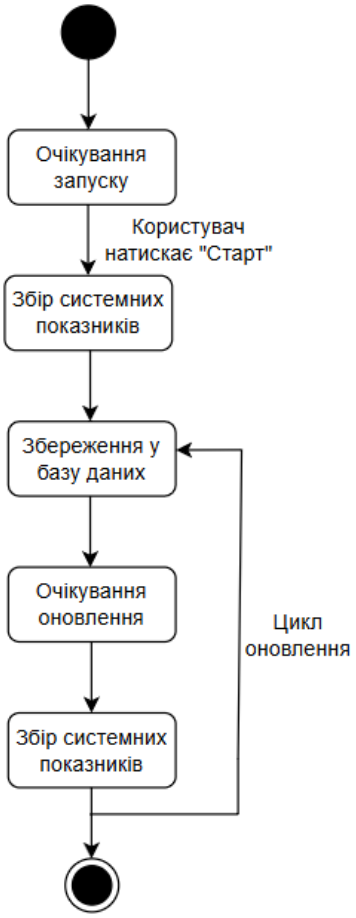


Рис 6. Діаграма станів

Процес починається зі стану «Очікування запуску», на цьому етапі збір статистичних даних не виконується, а система очікує дії користувача. Коли користувач натискає кнопку «Старт», програма переходить до стану «Збір системних показників».

У стані збору даних система автоматично отримує поточні показники роботи комп’ютера, зокрема завантаження процесора, використання оперативної пам’яті, активне вікно, кількість натискань клавіш і кліків миші. Зібрана інформація передається далі у стан «Збереження у базу даних», де всі дані записуються у відповідні таблиці MySQL для подальшого аналізу та формування звітів.

Після успішного збереження система переходить у стан «Очікування оновлення», у якому відбувається коротка пауза перед повторним збором нових даних. Потім процес циклічно повертається до стану «Збір системних показників», утворюючи безперервний цикл оновлення та запису метрик системи в базу даних.

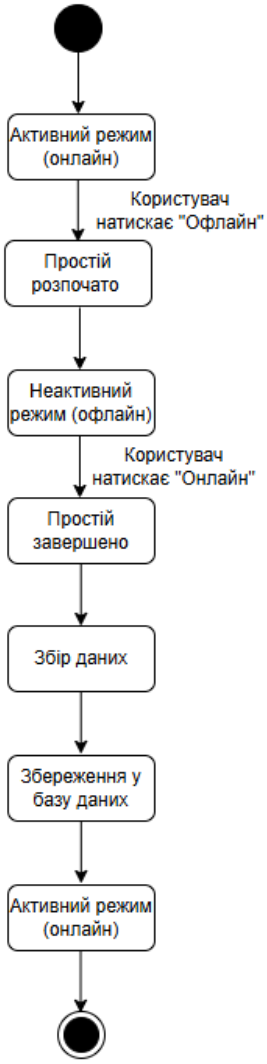


Рис 7. Діаграма станів

Процес починається зі стану «Активний режим (онлайн)», у якому користувач активно взаємодіє із системою — натискає клавіші, рухає мишу або змінює вікна програм. Система постійно відстежує ці дії через фонові процеси моніторингу.

Коли користувач припиняє будь-яку активність і натискає кнопку «Офлайн», настає стан «Простій розпочато». У цьому режимі програма фіксує час початку простою та переходить у стан «Неактивний режим (офлайн)», де користувач вважається відсутнім, а система перебуває у пасивному спостереженні.

Після повернення користувача, коли він натискає «Онлайн» відбувається перехід до стану «Простій завершено». Програма фіксує час завершення простою та обчислює загальну тривалість неактивності користувача.

Далі система переходить у стан «Збір даних», де формує коротке зведення про період простою. Ця інформація зберігається у базі даних у стані «Збереження у базу даних», після чого програма знову повертається до активного режиму (онлайн) і продовжує моніторинг у звичайному режимі.

2 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

2.1. Структура бази даних

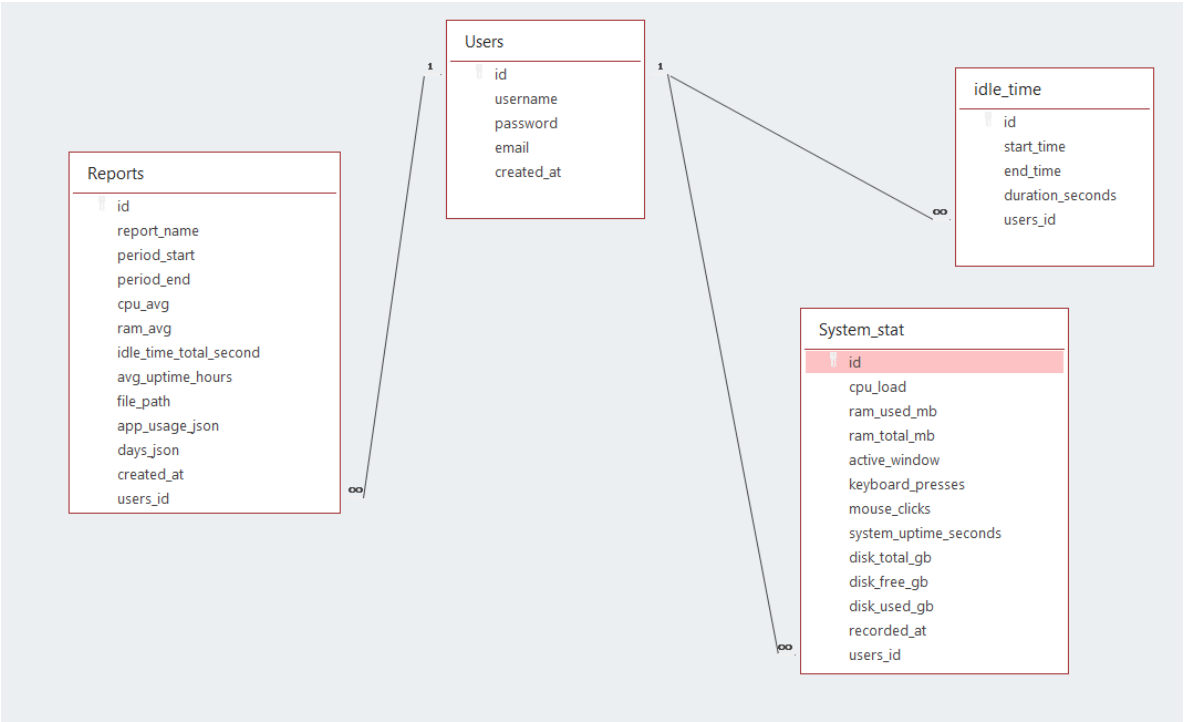


Рис 8. Структура бази даних

База даних була розроблена з метою забезпечення зберігання, аналізу та формування звітів щодо активності користувачів та стану їхніх систем у межах програмного комплексу моніторингу. Модель побудована на принципах реляційної структури з чітким розділенням сутностей, що підвищує масштабованість, швидкість доступу та узгодженість даних.

База даних містить чотири основні таблиці: users, idle_time, system_stats, reports. Вони пов’язані між собою відношенням «один-до-багатьох» через зовнішні ключі, де ключова сутність – це таблиця користувачів.

Таблиця users зберігає реєстраційні та облікові дані користувачів. Саме вона є центральною точкою зв’язку для всіх інших таблиць. Створення окремої таблиці користувачів дозволяє централізовано керувати доступом та зв’язувати з ними інші сутності системи.

Таблиця idle_time фіксує періоди, протягом яких користувач не взаємодіє з комп’ютером. Зберігання інтервалів бездіяльності необхідне для аналізу

продуктивності користувачів, обрахунку загального часу простою та подальшого включення цих даних у звіти.

Таблиця system_stats містить дані, зібрані у режимі реального часу з комп'ютера користувача. Це важлива частина системи, оскільки забезпечує можливість аналізу навантаження та поведінки системи. Збір таких показників дозволяє формувати повноцінну статистику використання системи, визначати рівень продуктивності, навантаження та взаємодію користувача з комп'ютером. Це також дає змогу генерувати детальні звіти за будь-який період.

Таблиця reports використовується для зберігання сформованих звітів, що містять агреговану інформацію за обраний період. Використання окремої таблиці для звітів дозволяє зберігати історію всіх сформованих документів, забезпечує швидкий доступ до результатів аналізу та уможливлює експорт у файли. Формат JSON дає змогу зберігати складні структуровані дані без створення додаткових таблиць.

2.2. Архітектура системи

2.2.1. Специфікація системи

Моя система має декілька рівнів, які взаємодіють між собою через інтерфейси. Діаграма архітектури взаємодії зображена на рисунку 9.

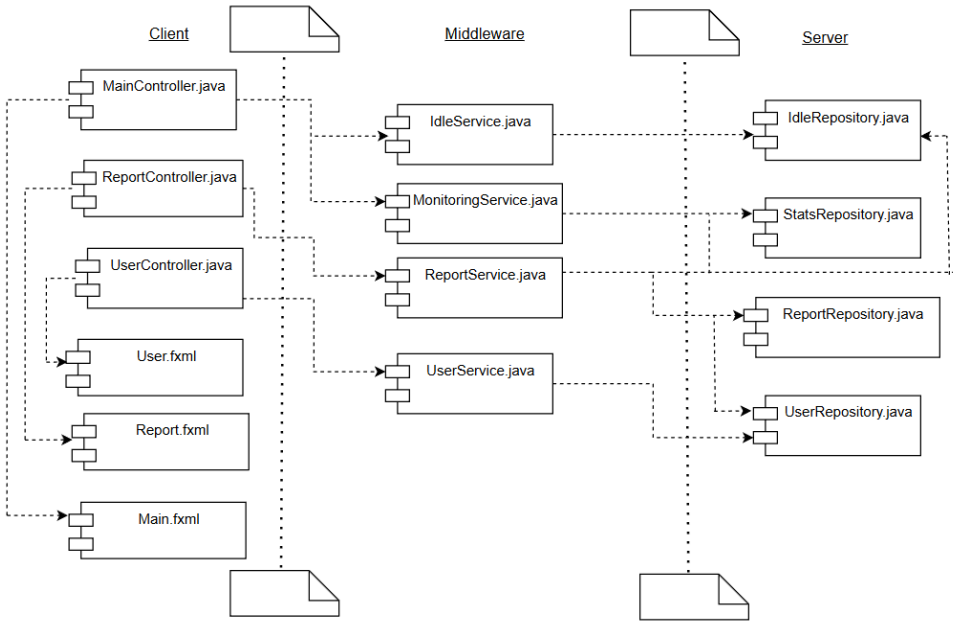


Рис 9. Діаграма архітектури взаємодії

На діаграмі зображена архітектура взаємодії системи, яка поділена на три основні рівні: Client, Middleware та Server. Такий поділ дозволяє зрозуміти, як різні частини програми взаємодіють між собою.

На рівні Client знаходяться всі елементи, з якими взаємодіє користувач. Це контролери: MainController, ReportsController та UserController, а також відповідні їм FXML-файли інтерфейсу: Main.fxml, Report.fxml та User.fxml. Контролери отримують події від графічного інтерфейсу, обробляють дії користувача і передають їх далі в бізнес-логіку. FXML-файли з'єднані з контролерами, бо саме через них описується вигляд вікон і елементів інтерфейсу.

Наступний рівень – Middleware. Тут розташована вся бізнес-логіка системи: IdleService, MonitoringService, ReportService та UserService. Ці сервіси виконують основну роботу: збір системних метрик, формування звітів, облік часу простою комп'ютера, роботу з користувачами. Контролери звертаються саме до цих сервісів. Кожен сервіс приховує внутрішню логіку і не дозволяє клієнтській частині напряму працювати з даними.

Останній рівень – Server. Він відповідає за взаємодію з постійним сховищем даних. Тут знаходяться інтерфейси репозиторіїв і їхні реалізації. Саме через ці репозиторії сервіси отримують або зберігають інформацію в базу даних. Такий поділ дозволяє відокремити логіку роботи програми від деталей зберігання даних.

Стрілки на діаграмі показують напрямки взаємодії між компонентами. Клієнтські контролери звертаються до сервісів, а сервіси – до репозиторіїв. Клієнт не має прямого доступу до бази даних, і не взаємодіє з серверним шаром.

Загальна діаграма класів системи зображена в додатку А. Діаграма класів показує загальну архітектуру програмної системи, яка складається з кількох взаємодіючих рівнів: контролерів інтерфейсу, сервісної логіки, репозиторіїв даних. Контролери працюють лише з сервісним рівнем, сервіси звертаються до репозиторіїв, а репозиторії взаємодіють із базою даних. Логіка збору статистики, обробки даних, генерації звітів та їх експорту ізольована в окремі модулі.

Сутності та інтерфейси контролерів наведені на рисунку 9.1. На рисунку 9.2 представлені сутності та інтерфейси рівня доступу до даних.

Рис. 9.1 Сутності та інтерфейси контролерів.

У проєкті контролери відповідають за взаємодію між користувачем та внутрішньою логікою системи. Кожен контролер отримує події інтерфейсу (натискання кнопок, перемикання вікон, зміни стану), викликає необхідні сервісні методи та оновлює графічне представлення вікон JavaFX. Контролери не містять логіки обробки даних – вони є лише посередниками між UI та сервісними компонентами.

Уся прикладна логіка зосереджена в сервісах, а контролери лише керують сценами, відображенням даних, запуском генерації звітів, авторизацією та перемиканням вікон.

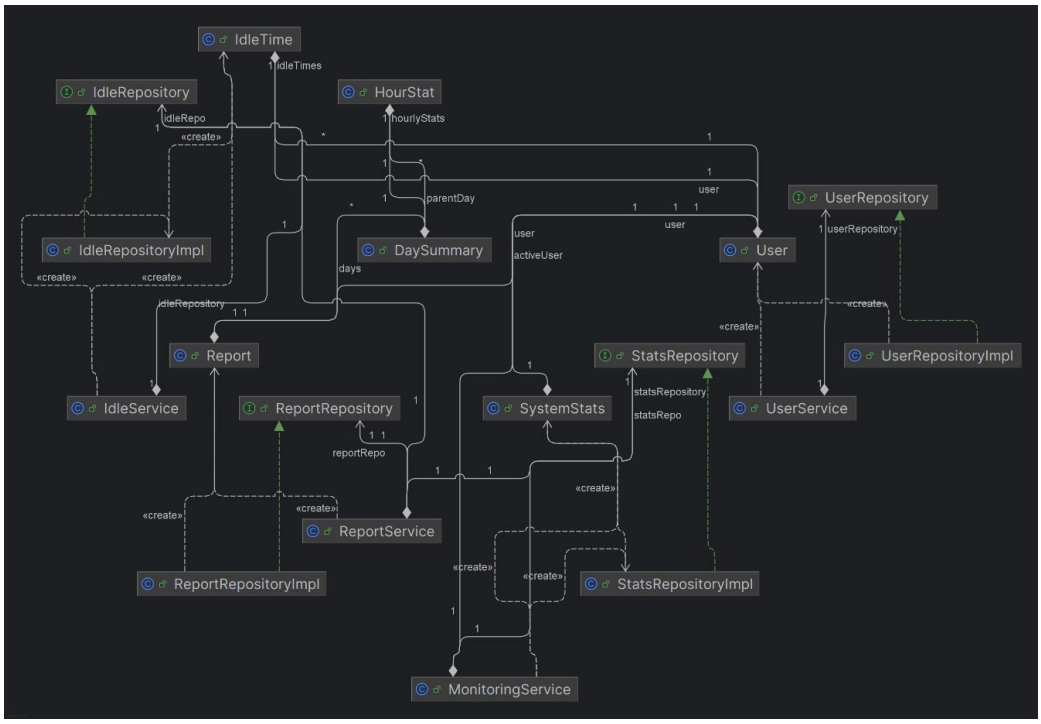


Рис. 9.2 Сутності та інтерфейси рівня доступу до даних

На наведеній діаграмі зображено основні сутності, репозиторії та сервіси, що утворюють логічний рівень обробки даних у системі. Вона відображає, як дані переміщуються між моделями, сервісами та репозиторіями, і показує загальну архітектуру доменної логіки.

У центрі діаграми знаходиться сутність User, яка представляє зареєстрованого користувача системи. Користувач пов’язаний з декількома іншими структурами: він має свої статистичні дані роботи за допомогою сутності SystemStats, список записів простою IdleTime, а також список сформованих звітів Report. Реальні операції взаємодії із користувачами виконуються через сервіс UserService, який, у свою чергу, працює з інтерфейсом UserRepository та його конкретною реалізацією UserRepositoryImpl. Таким чином забезпечується розділення відповідальностей між прикладною логікою та доступом до бази даних.

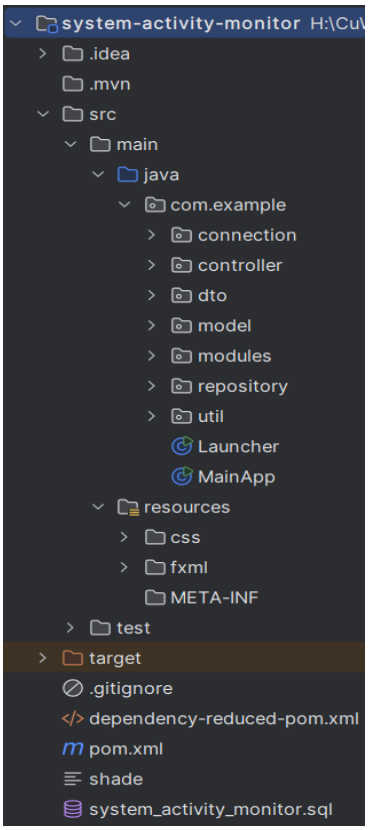


Рис 9.3 Структура проєкту

При розробці системи я звертався до шаблонів проєктування, які допомогли вирішити складні задачі і дотримуватись принципів об’єктно-орієнтованого програмування.

2.2.2. Вибір та обґрунтування патернів реалізації

У процесі реалізації програмної системи було застосовано декілька класичних патернів проєктування Iterator, Command, Strategy, Builder. Кожен із них вирішує конкретну проблему в архітектурі, полегшує підтримку проєкту та покращує масштабованість.

2.2.3 Обхід вкладених структур звіту

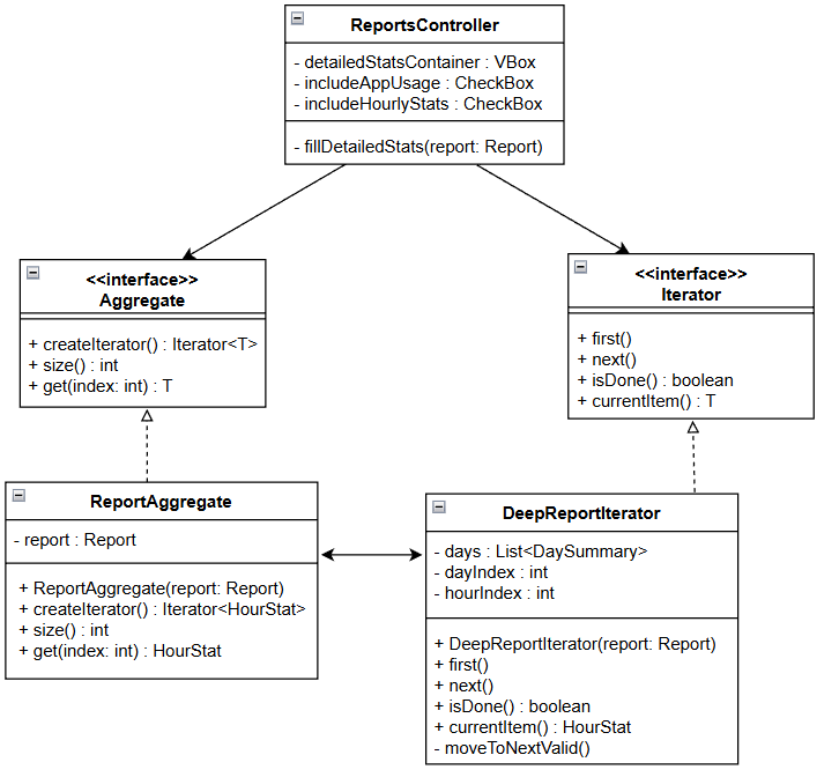


Рис 10. Діаграма класів патерна проєктування Iterator

У моєму проєкті використано патерн Iterator [1], оскільки структура звіту, з якою працює система, є вкладеною і потенційно складною. Один звіт містить список днів, а кожен день містить список годинних показників. Такі дані неможливо зручно опрацьовувати простими циклами без того, щоб не змішувати логіку обходу і логіку відображення в одному місці. Як результат, контролер стає перевантаженим вкладеними циклами, умовами, перевірками на пусті колекції та переходами між рівнями структури. Саме для вирішення цієї проблеми застосований патерн Iterator.

Iterator дозволив заховати всю складність всередині окремого класу DeepReportIterator. У цьому класі зосереджена логіка переходів між днями, пропуску пустих списків, руху між годинам та визначення моменту, коли дані закінчились. Таким чином, контролер не знає, як саме зберігаються дані всередині Report, йому не потрібно турбуватися, чи є в дні години, чи закінчився список, чи потрібно переходити на наступний день. Контролер просто отримує ітератор і працює з ним послідовно, як з рівною плоскою колекцією годинних статистичних записів.

Також Iterator робить систему гнучкішою. Якщо у майбутньому структура звіту зміниться, контролеру не доведеться змінювати свій код. Достатньо буде оновити лише логіку ітератора. Це підвищує інкапсуляцію та підтримуваність проєкту.

2.2.4 Виконання різних дій зі звітами

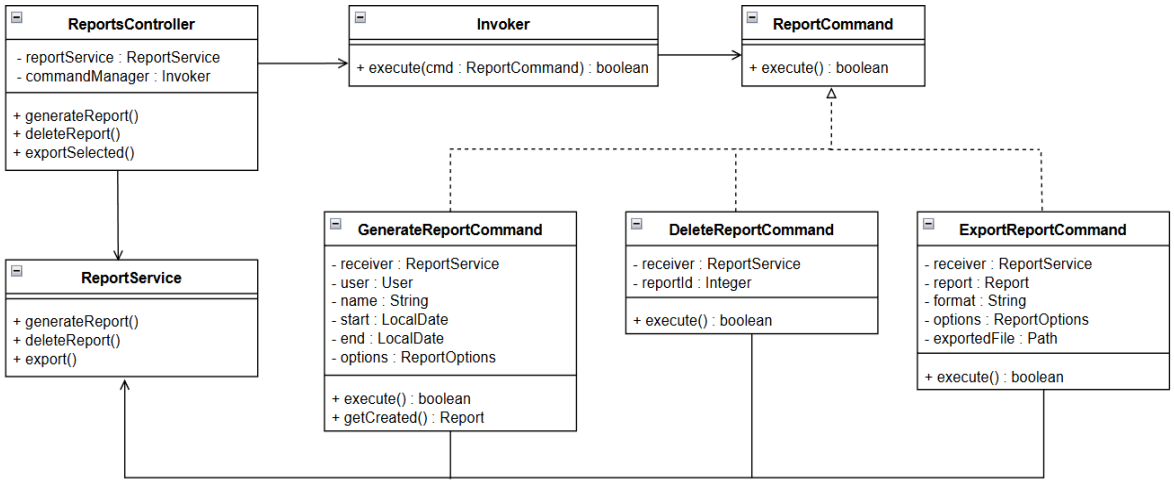


Рис 11. Діаграма класів патерна проєктування Command

У моєму проєкті використано патерн Command [1] для того, щоб відокремити логіку виконання операцій над звітами від коду користувацького інтерфейсу. У контролері є кілька різних дій: створення звіту, видалення звіту, експорт у різні формати. Усі ці дії мають свою бізнес-логіку, власні параметри та взаємодіють із ReportService. Якщо виконувати ці операції прямо в контролері, код став би важким, громіздким і жорстко пов’язаним з сервісним шаром. Патерн Command дозволив винести кожну дію в окремий об’єкт-команду, який описує усе, що потрібно для її виконання. Це зробило код чистішим, більш структурованим та масштабованим.

Command дав змогу зробити контролер простим і універсальним. Контролер тепер не знає, як саме виконується створення звіту або його експорт. Він просто створює відповідну команду і передає її Invoker-у, який викликає метод execute. Таким чином, контролер перестає бути місцем, де змішується бізнес-логіка, робота з сервісами, UI-обробка та формування результатів. Усі дії зводяться до єдиного механізму виконання команд.

Команда, у свою чергу, інкапсулює всю необхідну інформацію: об’єкти, параметри, формат експорту, діапазони дат, користувача. Це дозволяє легко

додавати нові операції, не змінюючи код контролера. Патерн Command робить архітектуру розширюваною, оскільки нові варіанти поведінки оформлюються в окремі об’єкти і не впливають на вже існуючу логіку.

У моєму проєкті Command також підвищує тестованість. Кожну команду можна перевіряти окремо, без UI, без контролера, просто створивши об’єкт і виконавши його. Це суттєво спрощує розробку. Крім того, Command чітко визначає функцію Receiver, тобто клас ReportService. Саме він виконує реальну бізнес-логіку, а команди лише передають йому контроль у потрібний момент. Така взаємодія відповідає класичному патерну і дозволяє легко управляти операціями.

2.2.5 Експорт звітів у різні формати.

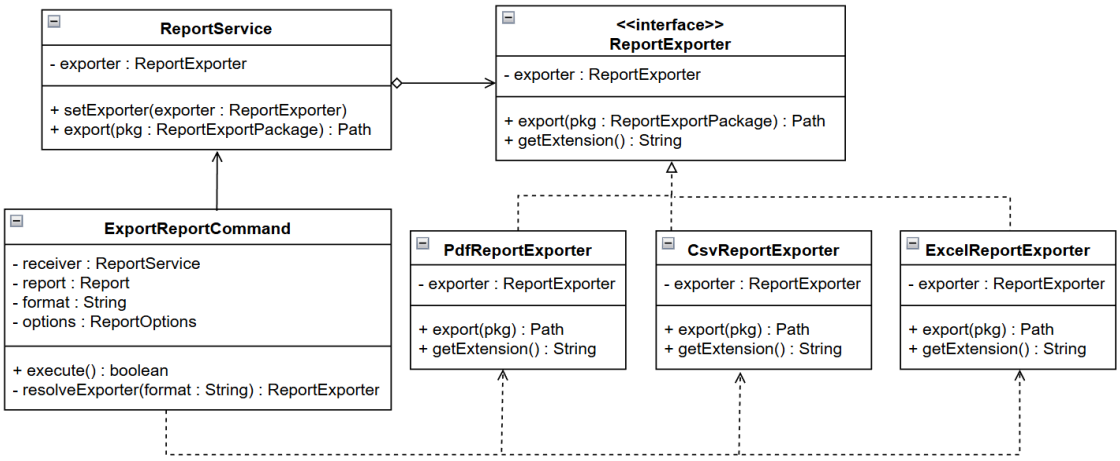


Рис 11. Діаграма класів патерна проєктування Strategy

У моєму проєкті патерн Strategy [3] використовується для організації системи експорту звітів у різні формати. Я обрав саме цей патерн, тому що він дозволяє розділити логіку створення звіту та логіку його збереження у певному форматі, не змішуючи все в одному класі і не створюючи громіздкі конструкції з умовами. У моїй системі користувач може експортувати звіт у PDF, Excel або CSV, але сам ReportService не повинен знати, яким саме способом це робиться, оскільки його завдання полягає тільки у підготовці даних. Формат експорту залежить від вибору користувача, тому відповідну реалізацію потрібно підставляти динамічно.

Strategy вирішує саме цю задачу. ReportService виступає контекстом і містить посилання на інтерфейс ReportExporter, який визначає єдиний контракт експорту.

Реальні алгоритми знаходяться у трьох окремих класах: PdfReportExporter, CsvReportExporter і ExcelReportExporter. Вони повністю ізольовані один від одного, не дублюють логіку і не перевантажують сервіс.

ExportReportCommand визначає, який формат обрав користувач, і встановлює відповідну стратегію в ReportService. Завдяки цьому сам сервіс ніколи не визначає, що саме він експортує, і не містить логіки вибору. Він лише виконує метод export у будь-якої стратегії, яку йому передали.

2.2.5 Побудова звітів за різними критеріями

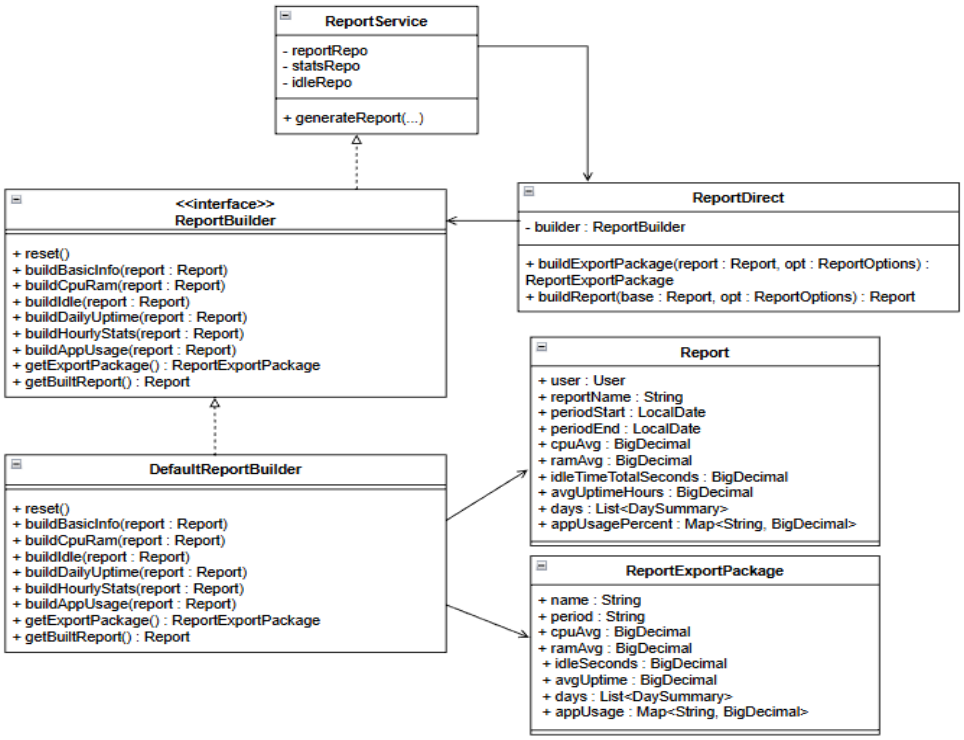


Рис 11. Діаграма класів патерна проєктування Builder

Патерн Builder [6] у моєму проєкті був обраний тому, що процес побудови звіту є складним, багатокроковим і залежить від великої кількості умов. Звіт може містити базову інформацію статистику CPU і RAM, загальний час простою, середній час роботи комп’ютера, погодинну статистику і відсоток використання програм. Кожен із цих блоків може бути присутнім або відсутнім у кінцевому звіті залежно від налаштувань, які вибирає користувач у графічному інтерфейсі. Через те, що структура звіту постійно змінюється, а порядок побудови його частин завжди має

бути відрегульований і керований, використання простих конструкторів або фабрик створює надлишкову складність та робить код непідтримуваним.

Builder дозволяє розділити процес побудови складного об’єкта на окремі етапи, кожен з яких відповідає за формування конкретної частини звіту. Конкретний Builder реалізує всю логіку наповнення структури звіту, а Director керує тим, які етапи мають бути виконані і в якій послідовності. Логіка побудови звіту стає чистою, гнучкою і легко розширюваною. Якщо в майбутньому виникне потреба додати новий блок статистики, достатньо модифікувати або створити новий Builder без змін у сервісах або контролерах.

2.3. Інструкція користувача

Для того щоб користуватись системою потрібно мати встановлену СКБД MySQL і середовище виконання Java від 21 версії. Завантажте проєкт з репозиторію і створіть базу даних (код до бази даних є в проєкті під назвою system_activity_monitor.sql), тепер можна запустити файл exe. Після запуску системи буде показана головна сторінка.

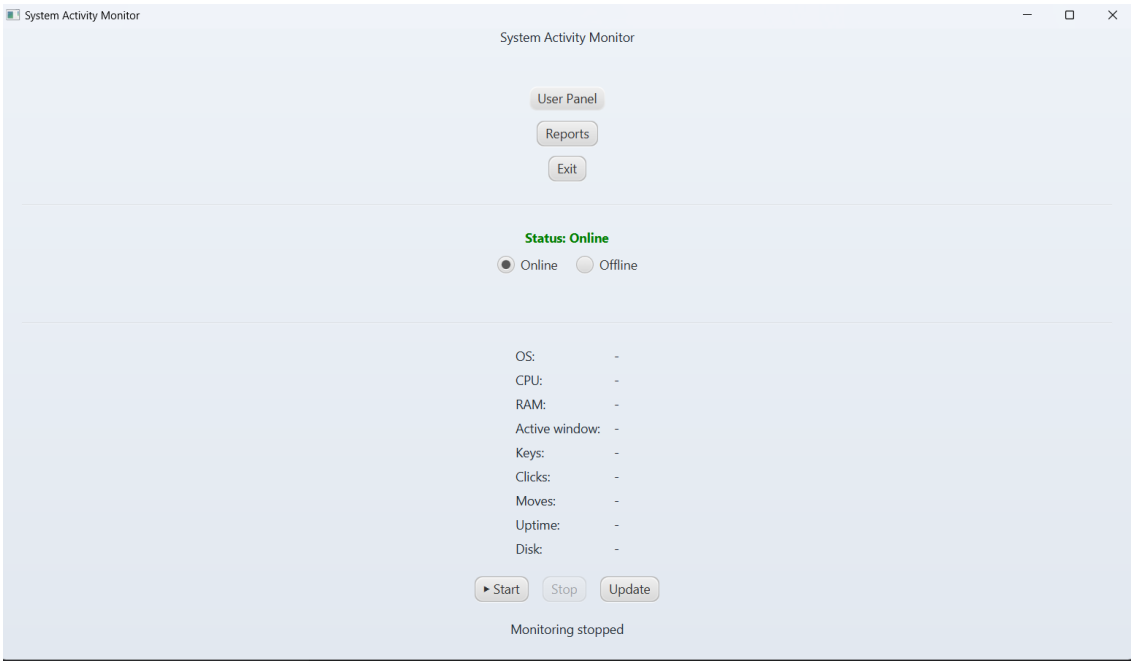


Рис 12. Головна сторінка

Для того щоб користуватись системою потрібно перейти в “User Panel” і зареєструвати аккаунт або працювати в режимі “гість” (в режимі “Гість” дані не

зберігаються у базу даних і неможливо працювати з звітами). На головній сторінці можна почати або зупинити моніторинг системних ресурсів системи, або керувати простоєм (режим Online/Offline). Коли активний простій (режим Offline) блокується весь функціонал кнопок програми окрім кнопки exit.

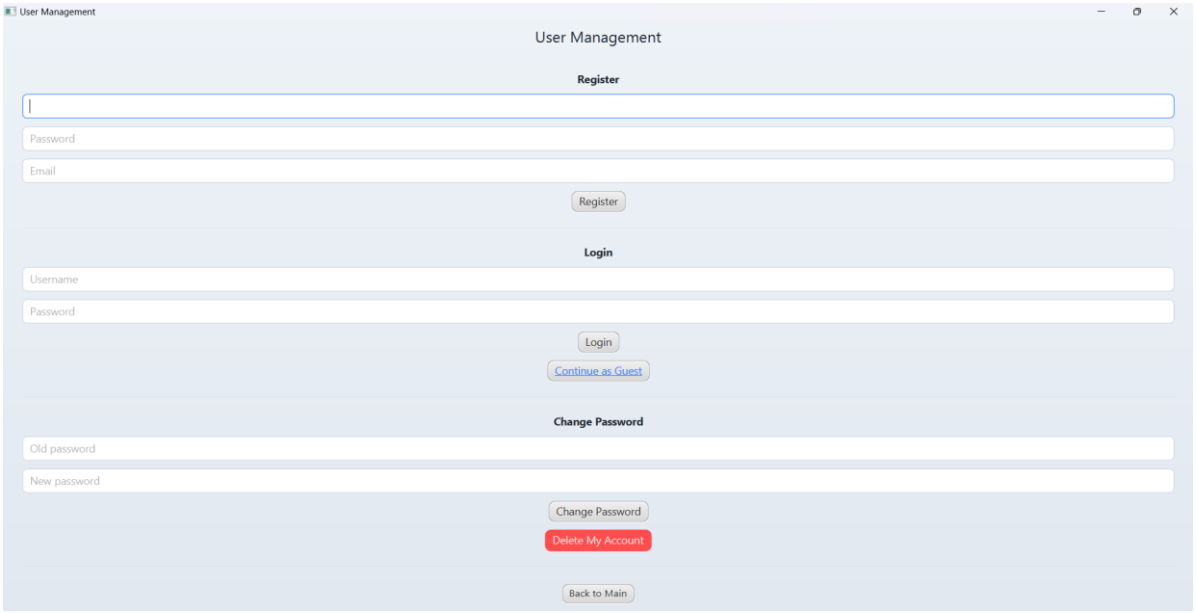


Рис 13. User panel сторінка

На User panel сторінці можна зареєструвати, увійти, видалити або змінити пароль аккаунту. Можна не реєструвати аккаунт і працювати в режимі “Гість”.

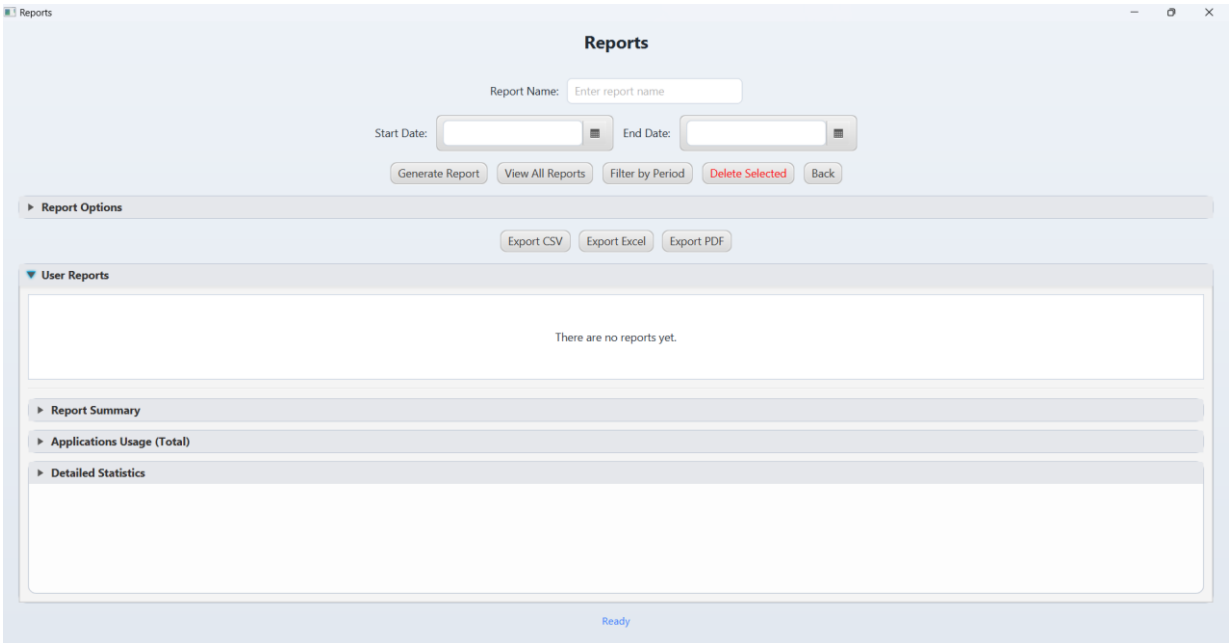


Рис 14. Reports сторінка

На Reports сторінці можна створити звіт за певний період часу, також вивести всі звіти, відфільтрувати звіти за періодом і видалити звіти. Можна формувати звіти за різними критеріями, тобто включати в звіт ту інформацію яка потрібна користувачу, і також експортувати звіт в різних форматах на локальний комп’ютер.

ВИСНОВКИ

У ході виконання даної курсової роботи було проведено повний цикл аналізу, проєктування та реалізації програмного забезпечення для системи моніторингу активності комп’ютера. На першому етапі були визначені вимоги до функціоналу, який мав забезпечувати збір системних показників, облік простою, формування структурованих звітів. На основі цих вимог було сформовано архітектуру застосунку, що включає клієнтську частину, внутрішній сервісний шар і модуль роботи з базою даних.

Під час створення системи було досліджено сучасні підходи до побудови інформаційних систем і реалізовано низку архітектурних та алгоритмічних рішень. Значну увагу приділено використанню шаблонів проєктування, які дозволили упорядкувати бізнес-логіку, ізолювати складні обчислення, уникнути дублювання коду й підвищити гнучкість системи. Застосування патернів Command, Builder, Strategy та Iterator дало змогу розділити відповідальності між компонентами та створити застосунок, у якому кожна частина виконує свою чітко визначену роль.

У процесі реалізації були розроблені модулі збору системної статистики, механізми розрахунку показників, сервіс формування звітів із можливістю експорту у різні формати та зручний графічний інтерфейс. Використання бази даних забезпечило надійне зберігання історії активності, а багатошаровий підхід у внутрішній архітектурі зробив застосунок розширюваним та керованим.

Під час розробки стало очевидно, що сучасні технології моніторингу та аналітики дозволяють створювати ефективні інструменти для контролю комп’ютерної активності, оптимізації робочого часу та аналізу продуктивності. Програмна реалізація довела, що поєднання коректного архітектурного підходу, алгоритмів обчислення статистики та застосування патернів дає змогу створити надійну і масштабовану систему, здатну адаптуватися до потреб користувача.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Рефакторинг.Гуру. [Електронний ресурс]. Доступ: https://refactoring.guru/uk

[2] JavaFx Documentation. [Електронний ресурс]. Доступ: https://openjfx.io/javadoc/25/.

[3] “Занурення в патерни проектування”. Автор: О. Швець.

[4] Java Docs. [Електронний ресурс]. Доступ: https://docs.oracle.com/en/java/.

[5] IntelliJ IDEA. [Електронний ресурс]. Доступ: https://www.jetbrains.com/idea/.

[6] "Java Design Patterns: A tour of 23 gang of four design patterns in Java" by Vaskaran Sarcar

[7] "Head First Design Patterns" by Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra

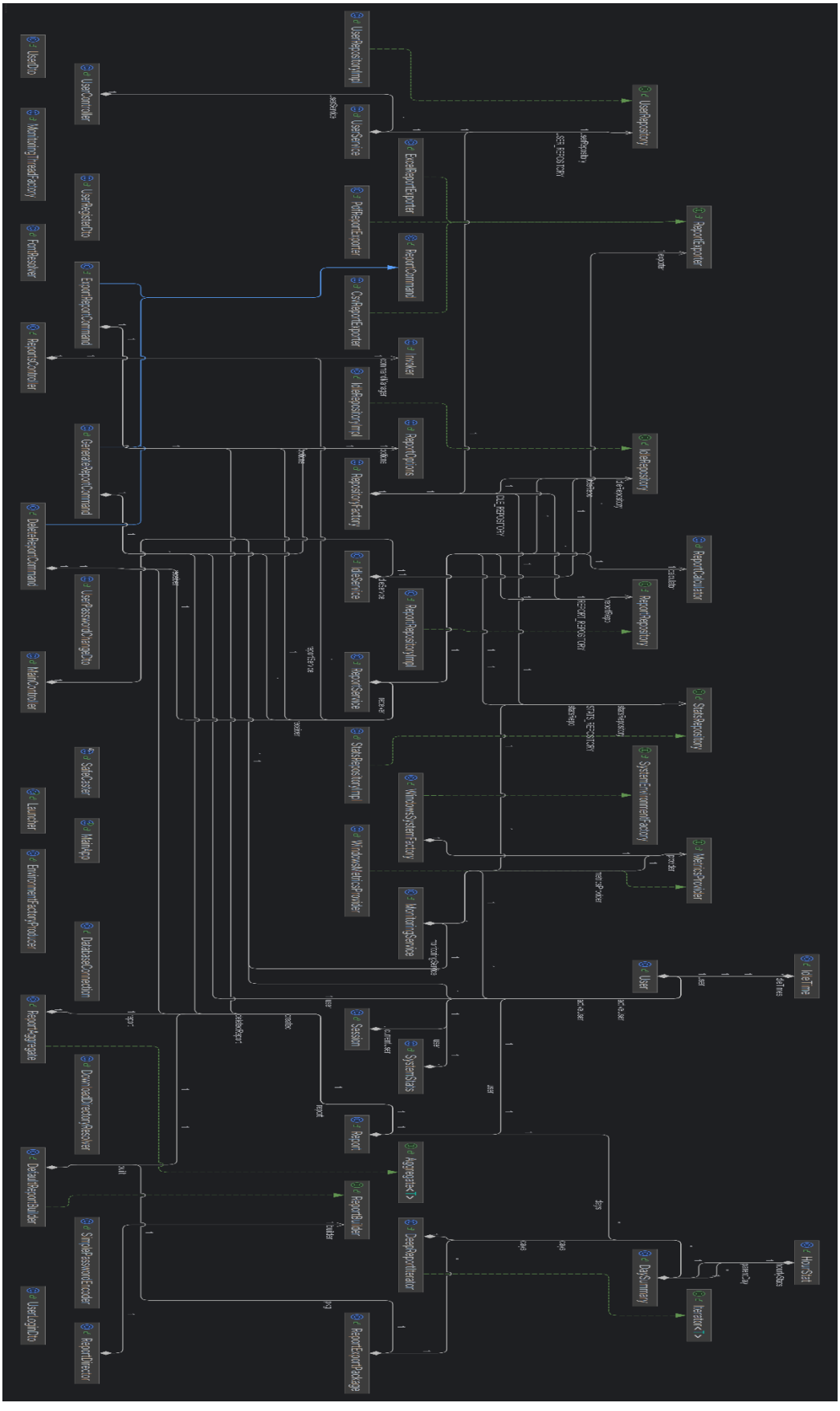
[8] "Effective Java" by Joshua Bloch

[9] “Занурення в рефакторинг”. Автор: О. Швець.

[10] “Грокаємо алгоритми. Ілюстрований посібник для програмістів і допитливих”. Автор: А. Бхаргава

ДОДАТКИ

ДОДАТОК А



ДОДАТОК Б

```
package com.example.modules.reports.iterator;

public interface Aggregate<T> {
    Iterator<T> createIterator();
    int size();
    T get(int index);
}
```

```
package com.example.modules.reports.iterator;

import com.example.model.DaySummary;
import com.example.model.HourStat;
import com.example.model.Report;

import java.util.List;

public class DeepReportIterator implements Iterator<HourStat> {

    private final List<DaySummary> days;

    private int dayIndex = 0;
    private int hourIndex = 0;

    public DeepReportIterator(Report report) {
        this.days = report != null ? report.getDays() : List.of();
    }

    @Override
    public void first() {
        dayIndex = 0;
        hourIndex = 0;
```

```
    moveToNextValid();
}

@Override
public void next() {
    if (isDone()) return;

    hourIndex++;
    moveToNextValid();
}

@Override
public boolean isDone() {
    return dayIndex >= days.size();
}

@Override
public HourStat currentItem() {
    if (isDone()) return null;

    DaySummary day = days.get(dayIndex);
    List<HourStat> hours = day.getHourlyStats();

    if (hours == null || hourIndex >= hours.size()) return null;

    return hours.get(hourIndex);
}

private void moveToNextValid() {
    while (dayIndex < days.size()) {
        DaySummary day = days.get(dayIndex);
```

```
            List<HourStat> hours = (day != null) ? day.getHourlyStats() : null;

            if (hours == null || hours.isEmpty()) {
                dayIndex++;
                hourIndex = 0;
                continue;
            }

            if (hourIndex >= hours.size()) {
                dayIndex++;
                hourIndex = 0;
                continue;
            }

            return;
        }
    }
}
```

```
package com.example.modules.reports.iterator;

public interface Iterator<T> {
    void first();
    void next();
    boolean isDone();
    T currentItem();
}
```

```
package com.example.modules.reports.iterator;

import com.example.model.HourStat;
import com.example.model.Report;
```

```
public class ReportAggregate implements Aggregate<HourStat> {

    private final Report report;

    public ReportAggregate(Report report) {
        this.report = report;
    }

    @Override
    public Iterator<HourStat> createIterator() {
        return new DeepReportIterator(report);
    }

    @Override
    public int size() {
        Iterator<HourStat> it = createIterator();
        int count = 0;

        for (it.first(); !it.isDone(); it.next()) {
            if (it.currentItem() != null) count++;
        }
        return count;
    }

    @Override
    public HourStat get(int index) {
        if (index < 0) return null;

        Iterator<HourStat> it = createIterator();
        int i = 0;
```

```
        for (it.first(); !it.isDone(); it.next()) {
            HourStat stat = it.currentItem();
            if (stat == null) continue;

            if (i == index) return stat;
            i++;
        }
        return null;
    }
}
```

```
package com.example.modules.reports.export;

import com.example.modules.reports.builder.ReportExportPackage;

import java.nio.file.Path;

public interface ReportExporter {
    Path export(ReportExportPackage pkg) throws Exception;
    String getExtension();
}
```

```
package com.example.modules.reports.export;

import com.example.modules.reports.builder.ReportExportPackage;
import com.example.model.DaySummary;
import com.example.model.HourStat;
import com.example.util.FontResolver;
import com.itextpdf.text.*;
import com.itextpdf.text.pdf.*;
```

```
import java.io.FileOutputStream;
import java.nio.file.*;

public class PdfReportExporter implements ReportExporter {

    private static final Path EXPORT_DIR =
DownloadDirectoryResolver.getDownloadDirectory();

    @Override
    public Path export(ReportExportPackage pkg) throws Exception {

        Files.createDirectories(EXPORT_DIR);

        Path path = EXPORT_DIR.resolve(pkg.name + ".pdf");

        Document document = new Document();
        PdfWriter.getInstance(document, new FileOutputStream(path.toFile()));
        document.open();

        BaseFont bf = FontResolver.resolveCyrillicFont();
        Font font = new Font(bf, 12);
        Font bold = new Font(bf, 16, Font.BOLD);

        document.add(new Paragraph("Звіт: " + pkg.name, bold));
        document.add(new Paragraph("Період: " + pkg.period, font));

        if (pkg.cpuAvg != null)
            document.add(new Paragraph("CPU Avg: " + pkg.cpuAvg + "%", font));

        if (pkg.ramAvg != null)
            document.add(new Paragraph("RAM Avg: " + pkg.ramAvg + " MB", font));
```

```
if (pkg.idleSeconds != null)
    document.add(new Paragraph("Idle Time: " + pkg.idleSeconds + " сек", font));

if (pkg.avgUptime != null)
    document.add(new Paragraph("Avg Uptime: " + pkg.avgUptime + " год/день", font));

document.add(new Paragraph("\n"));

// DAILY CPU/RAM
document.add(new Paragraph("Середній CPU/RAM по днях:", bold));
for (DaySummary d : pkg.days) {
    document.add(new Paragraph(
            String.format("%s — CPU: %.2f%%, RAM: %.2f MB",
                    d.getDate(),
                    d.getCpuAvg(),
                    d.getRamAvg()
            ),
            font
    ));
}
document.add(new Paragraph("\n"));

// DAILY APP USAGE
document.add(new Paragraph("Використання програм по днях:", bold));
PdfPTable appDailyTable = new PdfPTable(3);
appDailyTable.addCell("Дата");
appDailyTable.addCell("Програма");
appDailyTable.addCell("Відсоток");

for (DaySummary d : pkg.days) {
```

```
    if (d.getAppUsagePercentByDay() != null) {
        for (var entry : d.getAppUsagePercentByDay().entrySet()) {
            appDailyTable.addCell(d.getDate().toString());
            appDailyTable.addCell(entry.getKey());
            appDailyTable.addCell(entry.getValue().toString());
        }
    }
}
document.add(appDailyTable);
document.add(new Paragraph("\n"));

// DAILY UPTIME
document.add(new Paragraph("Аптайм по днях:", bold));
for (DaySummary d : pkg.days) {
    document.add(new Paragraph(
            d.getDate() + " — " + d.getUptimeHours() + " год",
            font
    ));
}
document.add(new Paragraph("\n"));

// HOURLY STATS
boolean hasHourly = pkg.days.stream().anyMatch(d -> d.getHourlyStats() != null);
if (hasHourly) {

    document.add(new Paragraph("Погодинна статистика:", bold));

    PdfPTable table = new PdfPTable(4);
    table.addCell("Дата");
    table.addCell("Година");
    table.addCell("CPU (%)");
```

```
    table.addCell("RAM (MB)");

    for (DaySummary d : pkg.days) {
        if (d.getHourlyStats() == null) continue;

        for (HourStat h : d.getHourlyStats()) {
            table.addCell(d.getDate().toString());
            table.addCell(String.format("%02d:00", h.getHour()));
            table.addCell(h.getAvgCpu().toString());
            table.addCell(h.getAvgRam().toString());
        }
    }

    document.add(table);
    document.add(new Paragraph("\n"));
}

// TOTAL APP USAGE
if (pkg.appUsage != null && !pkg.appUsage.isEmpty()) {
    document.add(new Paragraph("Використання програм (Загалом):", bold));

    PdfPTable appTable = new PdfPTable(2);
    appTable.addCell("Програма");
    appTable.addCell("Відсоток");

    pkg.appUsage.forEach((app, value) -> {
        appTable.addCell(app);
        appTable.addCell(value + "%");
    });

    document.add(appTable);
```

```
        }

        document.close();
        return path;
    }

    @Override
    public String getExtension() {
        return "pdf";
    }
}
```

```
package com.example.modules.reports.export;

import com.example.modules.reports.builder.ReportExportPackage;
import com.example.model.DaySummary;
import com.example.model.HourStat;
import org.apache.poi.ss.usermodel.*;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.FileOutputStream;
import java.nio.file.*;

public class ExcelReportExporter implements ReportExporter {

    private static final Path EXPORT_DIR =
DownloadDirectoryResolver.getDownloadDirectory();

    @Override
    public Path export(ReportExportPackage pkg) throws Exception {
        Files.createDirectories(EXPORT_DIR);
```

```
        Path path = EXPORT_DIR.resolve(pkg.name + ".xlsx");

        try (Workbook workbook = new XSSFWorkbook()) {

            Sheet sheet = workbook.createSheet("Report");
            int row = 0;

            sheet.createRow(row++).createCell(0).setCellValue("Report Name: " + pkg.name);
            sheet.createRow(row++).createCell(0).setCellValue("Period: " + pkg.period);

            if (pkg.cpuAvg != null)
                sheet.createRow(row++).createCell(0).setCellValue("CPU Avg: " + pkg.cpuAvg);

            if (pkg.ramAvg != null)
                sheet.createRow(row++).createCell(0).setCellValue("RAM Avg: " + pkg.ramAvg);

            if (pkg.idleSeconds != null)
                sheet.createRow(row++).createCell(0).setCellValue("Idle: " + pkg.idleSeconds);

            if (pkg.avgUptime != null)
                sheet.createRow(row++).createCell(0).setCellValue("Avg Uptime: " +
pkg.avgUptime);

            row += 2;

            // DAILY CPU/RAM
            if (pkg.days != null && !pkg.days.isEmpty()) {
                Row title = sheet.createRow(row++);
                title.createCell(0).setCellValue("Date");
                title.createCell(1).setCellValue("Daily CPU Avg (%)");
                title.createCell(2).setCellValue("Daily RAM Avg (MB)");
```

```
    for (DaySummary d : pkg.days) {
        Row r = sheet.createRow(row++);
        r.createCell(0).setCellValue(d.getDate().toString());
        r.createCell(1).setCellValue(d.getCpuAvg().doubleValue());
        r.createCell(2).setCellValue(d.getRamAvg().doubleValue());
    }

    row += 2;
}

// DAILY APP USAGE
if (pkg.days != null && !pkg.days.isEmpty()) {
    Row title = sheet.createRow(row++);
    title.createCell(0).setCellValue("Date");
    title.createCell(1).setCellValue("Application");
    title.createCell(2).setCellValue("Usage (%)");

    for (DaySummary d : pkg.days) {
        if (d.getAppUsagePercentByDay() != null) {
            for (var entry : d.getAppUsagePercentByDay().entrySet()) {
                Row r = sheet.createRow(row++);
                r.createCell(0).setCellValue(d.getDate().toString());
                r.createCell(1).setCellValue(entry.getKey());
                r.createCell(2).setCellValue(entry.getValue().doubleValue());
            }
        }
    }

    row += 2;
}
```

```
// DAILY UPTIME
if (pkg.days != null && !pkg.days.isEmpty()) {
    Row title = sheet.createRow(row++);
    title.createCell(0).setCellValue("Date");
    title.createCell(1).setCellValue("Uptime (h)");

    for (DaySummary d : pkg.days) {
        Row r = sheet.createRow(row++);
        r.createCell(0).setCellValue(d.getDate().toString());
        r.createCell(1).setCellValue(d.getUptimeHours().doubleValue());
    }

    row += 2;
}

// HOURLY STATS
boolean hasHourly = pkg.days.stream().anyMatch(d -> d.getHourlyStats() != null);
if (hasHourly) {

    Row h = sheet.createRow(row++);
    h.createCell(0).setCellValue("Date");
    h.createCell(1).setCellValue("Hour");
    h.createCell(2).setCellValue("CPU (%)");
    h.createCell(3).setCellValue("RAM (MB)");

    for (DaySummary d : pkg.days) {
        if (d.getHourlyStats() == null) continue;

        for (HourStat hs : d.getHourlyStats()) {
            Row r = sheet.createRow(row++);
```

```
                r.createCell(0).setCellValue(d.getDate().toString());
                r.createCell(1).setCellValue(hs.getHour());
                r.createCell(2).setCellValue(hs.getAvgCpu().doubleValue());
                r.createCell(3).setCellValue(hs.getAvgRam().doubleValue());
            }
        }

        row += 2;
    }

    // TOTAL APP USAGE
    if (pkg.appUsage != null && !pkg.appUsage.isEmpty()) {
        Row t = sheet.createRow(row++);
        t.createCell(0).setCellValue("Application Usage (Total)");

        for (var entry : pkg.appUsage.entrySet()) {
            Row r = sheet.createRow(row++);
            r.createCell(0).setCellValue(entry.getKey());
            r.createCell(1).setCellValue(entry.getValue().doubleValue());
        }
    }

    try (FileOutputStream out = new FileOutputStream(path.toFile())) {
        workbook.write(out);
    }
  }

  return path;
}

@Override
```

```
    public String getExtension() {
        return "xlsx";
    }
}
```

```
package com.example.modules.reports.export;

import com.example.modules.reports.builder.ReportExportPackage;
import com.example.model.DaySummary;
import com.example.model.HourStat;

import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.*;

public class CsvReportExporter implements ReportExporter {

    private static final Path EXPORT_DIR =
DownloadDirectoryResolver.getDownloadDirectory();

    @Override
    public Path export(ReportExportPackage pkg) throws IOException {

        Files.createDirectories(EXPORT_DIR);

        Path path = EXPORT_DIR.resolve(pkg.name + ".csv");

        try (FileWriter writer = new FileWriter(path.toFile())) {

            writer.write("Report," + pkg.name + "\n");
            writer.write("Period," + pkg.period + "\n");
```

```
if (pkg.cpuAvg != null)
    writer.write("CPU Avg," + pkg.cpuAvg + "\n");

if (pkg.ramAvg != null)
    writer.write("RAM Avg," + pkg.ramAvg + "\n");

if (pkg.idleSeconds != null)
    writer.write("Idle Time," + pkg.idleSeconds + "\n");

if (pkg.avgUptime != null)
    writer.write("Avg Uptime," + pkg.avgUptime + "\n");

writer.write("\n");

if (pkg.days != null && !pkg.days.isEmpty()) {
    writer.write("Date,Daily CPU Avg,Daily RAM Avg\n");
    for (DaySummary d : pkg.days) {
        writer.write(String.format(
                "%s,%.2f,%.2f\n",
                d.getDate(),
                d.getCpuAvg() != null ? d.getCpuAvg() : 0,
                d.getRamAvg() != null ? d.getRamAvg() : 0
        ));
    }
    writer.write("\n");
}

// DAILY APP USAGE
if (pkg.days != null && !pkg.days.isEmpty()) {
    writer.write("Date,Application,Usage (%)\n");
    for (DaySummary d : pkg.days) {
```

```
        if (d.getAppUsagePercentByDay() != null) {
            for (var entry : d.getAppUsagePercentByDay().entrySet()) {
                writer.write(String.format(
                        "%s,%s,%.2f\n",
                        d.getDate(),
                        entry.getKey(),
                        entry.getValue()
                ));
            }
        }
    }
    writer.write("\n");
}

// DAILY UPTIME
if (pkg.days != null && !pkg.days.isEmpty()) {
    writer.write("Date,Uptime Hours\n");
    for (DaySummary d : pkg.days) {
        writer.write(d.getDate() + "," + d.getUptimeHours() + "\n");
    }
    writer.write("\n");
}

// HOURLY
if (pkg.days != null && !pkg.days.isEmpty()) {
    writer.write("Date,Hour,CPU,RAM\n");
    for (DaySummary d : pkg.days) {
        if (d.getHourlyStats() == null) continue;
        for (HourStat h : d.getHourlyStats()) {
            writer.write(String.format(
                    "%s,%02d:00,%.2f,%.2f\n",
```

```
                        d.getDate(),
                        h.getHour(),
                        h.getAvgCpu(),
                        h.getAvgRam()
                    ));
                }
            }
            writer.write("\n");
        }

        // TOTAL APP USAGE
        if (pkg.appUsage != null && !pkg.appUsage.isEmpty()) {
            writer.write("Total Application Usage\n");
            writer.write("Application,Usage (%)\n");
            pkg.appUsage.forEach((k, v) -> {
                try {
                    writer.write(k + "," + v + "\n");
                } catch (IOException ignored) {}
            });
        }
    }

    return path;
  }

  @Override
  public String getExtension() {
    return "csv";
  }
}
```

```
package com.example.modules.reports.command;

public abstract class ReportCommand {
    public abstract boolean execute();
}
```

```
package com.example.modules.reports.command;


public class Invoker {
    public boolean execute(ReportCommand command) {
        return command.execute();
    }
}
```

```
package com.example.modules.reports.command;

import com.example.modules.reports.builder.ReportOptions;
import com.example.model.Report;
import com.example.model.User;
import com.example.modules.reports.service.ReportService;

import java.time.LocalDate;

public class GenerateReportCommand extends ReportCommand {

    private final ReportService receiver;
    private final User user;
    private final String name;
    private final LocalDate start;
    private final LocalDate end;
    private final ReportOptions options;
```

```
    private Report created;

    public GenerateReportCommand(
            ReportService receiver,
            User user,
            String name,
            LocalDate start,
            LocalDate end,
            ReportOptions options
    ) {
        this.receiver = receiver;
        this.user = user;
        this.name = name;
        this.start = start;
        this.end = end;
        this.options = options;
    }

    @Override
    public boolean execute() {
        created = receiver.generateReport(user, name, start, end, options);
        return created != null && created.getId() != null;
    }
    public Report getCreated() {
        return created;
    }
}
```

```
package com.example.modules.reports.command;

import com.example.modules.reports.builder.*;
```

```
import com.example.model.Report;
import com.example.modules.reports.service.ReportService;
import com.example.modules.reports.export.*;

import java.nio.file.Path;

public class ExportReportCommand extends ReportCommand {

    private final ReportService receiver;
    private final Report report;
    private final String format;
    private final ReportOptions options;

    private Path exportedFile;

    public ExportReportCommand(
            ReportService receiver,
            Report report,
            String format,
            ReportOptions options
    ) {
        this.receiver = receiver;
        this.report = report;
        this.format = format;
        this.options = options;
    }

    @Override
    public boolean execute() {
        try {
            ReportBuilder builder = new DefaultReportBuilder();
```

```
            ReportDirector director = new ReportDirector(builder);

            ReportExportPackage pkg = director.buildExportPackage(report, options);

            receiver.setExporter(resolveExporter(format));

            exportedFile = receiver.export(pkg);

            return exportedFile != null;

        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    private ReportExporter resolveExporter(String format) {
        return switch (format.toLowerCase()) {
            case "pdf" -> new PdfReportExporter();
            case "csv" -> new CsvReportExporter();
            case "excel", "xlsx" -> new ExcelReportExporter();
            default -> throw new IllegalArgumentException("Unknown export format: " + format);
        };
    }
}
```

```
package com.example.modules.reports.command;

import com.example.model.Report;
import com.example.modules.reports.service.ReportService;

public class DeleteReportCommand extends ReportCommand {
```

```
    private final ReportService receiver;
    private final Integer reportId;
    private Report deletedReport;

    public DeleteReportCommand(ReportService receiver, Integer reportId) {
        this.receiver = receiver;
        this.reportId = reportId;
    }

    @Override
    public boolean execute() {
        deletedReport = receiver.findById(reportId);
        if (deletedReport == null) return false;

        receiver.deleteReport(reportId);
        return true;
    }
}
```

```
package com.example.modules.reports.builder;

import com.example.model.Report;

public interface ReportBuilder {

    void reset();

    void buildBasicInfo(Report report);
    void buildCpuRam(Report report);
    void buildIdle(Report report);
    void buildDailyUptime(Report report);
```

```
    void buildHourlyStats(Report report);
    void buildAppUsage(Report report);

    ReportExportPackage getExportPackage();
    Report getBuiltReport();
}
```

```
package com.example.modules.reports.builder;

public class ReportOptions {
    public boolean includeCpuRam = true;
    public boolean includeDailyUptime = true;
    public boolean includeHourlyStats = true;
    public boolean includeAppUsage = true;
    public boolean includeIdle = true;
}
```

```
package com.example.modules.reports.builder;

import com.example.model.Report;

import java.util.Collections;

public class DefaultReportBuilder implements ReportBuilder {

    private ReportExportPackage pkg;
    private Report built;

    @Override
    public void reset() {
        pkg = new ReportExportPackage();
        built = new Report();
```

```
    pkg.days = Collections.emptyList();
}

@Override
public void buildBasicInfo(Report report) {
    pkg.name = report.getReportName();
    pkg.period = report.getPeriodStart() + " - " + report.getPeriodEnd();

    built.setUser(report.getUser());
    built.setReportName(report.getReportName());
    built.setPeriodStart(report.getPeriodStart());
    built.setPeriodEnd(report.getPeriodEnd());
}

@Override
public void buildCpuRam(Report report) {
    pkg.cpuAvg = report.getCpuAvg();
    pkg.ramAvg = report.getRamAvg();

    built.setCpuAvg(report.getCpuAvg());
    built.setRamAvg(report.getRamAvg());
}

@Override
public void buildIdle(Report report) {
    pkg.idleSeconds = report.getIdleTimeTotalSeconds();
    built.setIdleTimeTotalSeconds(report.getIdleTimeTotalSeconds());
}

@Override
```

```
    public void buildDailyUptime(Report report) {
        pkg.avgUptime = report.getAvgUptimeHours();
        built.setAvgUptimeHours(report.getAvgUptimeHours());
    }

    @Override
    public void buildHourlyStats(Report report) {
        pkg.days = report.getDays() != null ? report.getDays() : Collections.emptyList();
        built.setDays(pkg.days);
    }

    @Override
    public void buildAppUsage(Report report) {
        pkg.appUsage = report.getAppUsagePercent();
        built.setAppUsagePercent(report.getAppUsagePercent());
    }

    @Override
    public ReportExportPackage getExportPackage() {
        return pkg;
    }

    @Override
    public Report getBuiltReport() {
        return built;
    }
}
```

```
package com.example.modules.reports.builder;

import com.example.model.Report;
```

```
public class ReportDirector {

    private final ReportBuilder builder;

    public ReportDirector(ReportBuilder builder) {
        this.builder = builder;
    }

    public ReportExportPackage buildExportPackage(Report report, ReportOptions opt) {

        builder.reset();

        builder.buildBasicInfo(report);

        if (opt.includeCpuRam) builder.buildCpuRam(report);
        if (opt.includeIdle) builder.buildIdle(report);
        if (opt.includeDailyUptime) builder.buildDailyUptime(report);
        if (opt.includeHourlyStats) builder.buildHourlyStats(report);
        if (opt.includeAppUsage) builder.buildAppUsage(report);

        return builder.getExportPackage();
    }

    public Report buildReport(Report base, ReportOptions opt) {

        builder.reset();

        builder.buildBasicInfo(base);

        if (opt.includeCpuRam) builder.buildCpuRam(base);
        if (opt.includeIdle) builder.buildIdle(base);
```

```
        if (opt.includeDailyUptime) builder.buildDailyUptime(base);
        if (opt.includeHourlyStats) builder.buildHourlyStats(base);
        if (opt.includeAppUsage) builder.buildAppUsage(base);

        return builder.getBuiltReport();
    }
}
```

```
package com.example.modules.monitoring.metrics.impl;

import com.example.modules.monitoring.metrics.MetricsProvider;
import com.sun.jna.Native;
import com.sun.jna.platform.win32.*;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.AtomicLong;

public class WindowsMetricsProvider implements MetricsProvider {

    private long lastIdle = 0, lastKernel = 0, lastUser = 0;
    private volatile BigDecimal cpuLoad = BigDecimal.ZERO;

    private final ScheduledExecutorService cpuScheduler =
            Executors.newSingleThreadScheduledExecutor();

    public WindowsMetricsProvider() {
        cpuScheduler.scheduleAtFixedRate(this::updateCpu, 0, 1, TimeUnit.SECONDS);
```

```
}

private long filetime(WinBase.FILETIME ft) {
   return (((long) ft.dwHighDateTime) << 32) | (ft.dwLowDateTime & 0xffffffffL);
}

private void updateCpu() {
   Kernel32 k = Kernel32.INSTANCE;

   WinBase.FILETIME idle = new WinBase.FILETIME();
   WinBase.FILETIME kernel = new WinBase.FILETIME();
   WinBase.FILETIME user = new WinBase.FILETIME();

   if (!k.GetSystemTimes(idle, kernel, user)) return;

   long idleNow = filetime(idle);
   long kernNow = filetime(kernel);
   long userNow = filetime(user);

   if (lastIdle == 0) {
      lastIdle = idleNow;
      lastKernel = kernNow;
      lastUser = userNow;
      return;
   }

   long idleDiff = idleNow - lastIdle;
   long kernDiff = kernNow - lastKernel;
   long userDiff = userNow - lastUser;

   lastIdle = idleNow;
```

```
        lastKernel = kernNow;
        lastUser = userNow;

        long total = kernDiff + userDiff;
        if (total <= 0) return;

        double usage = (double) (total - idleDiff) / total * 100.0;
        usage = Math.max(0, Math.min(100, usage));

        cpuLoad = BigDecimal.valueOf(usage).setScale(2, RoundingMode.HALF_UP);
    }

    @Override
    public BigDecimal getCpuLoad() {
        return cpuLoad;
    }

    private BigDecimal ramTotal = BigDecimal.ZERO;

    @Override
    public BigDecimal getRamUsed() {
        Kernel32 k = Kernel32.INSTANCE;

        WinBase.MEMORYSTATUSEX mem = new WinBase.MEMORYSTATUSEX();
        if (!k.GlobalMemoryStatusEx(mem)) return BigDecimal.ZERO;

        long total = mem.ullTotalPhys.longValue();
        long free = mem.ullAvailPhys.longValue();

        ramTotal = BigDecimal.valueOf(total / 1024.0 / 1024.0).setScale(2,
RoundingMode.HALF_UP);
```

```
    double usedMb = (total - free) / 1024.0 / 1024.0;
    return BigDecimal.valueOf(usedMb).setScale(2, RoundingMode.HALF_UP);
}

@Override
public BigDecimal getRamTotal() {
    return ramTotal;
}

private BigDecimal diskTotal = BigDecimal.ZERO;
private BigDecimal diskFree = BigDecimal.ZERO;
private String diskDetails = "Unknown";

@Override
public void updateDiskStats() {
    Kernel32 k = Kernel32.INSTANCE;

    double tot = 0, free = 0;
    StringBuilder sb = new StringBuilder();

    for (char d = 'C'; d <= 'Z'; d++) {
        String root = d + ":\\";

        if (k.GetDriveType(root) != WinBase.DRIVE_FIXED) continue;

        WinNT.LARGE_INTEGER freeAvail = new WinNT.LARGE_INTEGER();
        WinNT.LARGE_INTEGER totalBytes = new WinNT.LARGE_INTEGER();
        WinNT.LARGE_INTEGER freeBytes = new WinNT.LARGE_INTEGER();

        if (k.GetDiskFreeSpaceEx(root, freeAvail, totalBytes, freeBytes)) {
```

```
            double tGb = totalBytes.getValue() / 1e9;
            double fGb = freeBytes.getValue() / 1e9;

            tot += tGb;
            free += fGb;

            sb.append(String.format("%s: %.2f / %.2f GB | ", d, (tGb - fGb), tGb));
        }
    }

    diskTotal = BigDecimal.valueOf(tot).setScale(2, RoundingMode.HALF_UP);
    diskFree = BigDecimal.valueOf(free).setScale(2, RoundingMode.HALF_UP);
    diskDetails = sb.isEmpty() ? "Unknown" : sb.toString();
}

@Override public BigDecimal getDiskTotal() { return diskTotal; }
@Override public BigDecimal getDiskFree()  { return diskFree;  }
@Override public BigDecimal getDiskUsed()  { return diskTotal.subtract(diskFree); }

@Override
public String getUptime() {
    long sec = Kernel32.INSTANCE.GetTickCount64() / 1000;

    long d = sec / 86400;
    long h = (sec % 86400) / 3600;
    long m = (sec % 3600) / 60;

    return d + " d " + h + " h " + m + " m";
}
```

```
public long getSystemUptimeSecondsRaw() {
   return Kernel32.INSTANCE.GetTickCount64() / 1000;
}

private final AtomicInteger keyPressCount = new AtomicInteger();
private final AtomicInteger mouseClickCount = new AtomicInteger();
private final AtomicLong mouseMoveCount = new AtomicLong();
private long lastActivity = System.currentTimeMillis();

private ScheduledExecutorService inputScheduler;
private boolean inputMonitoringActive = false;
private int lastX = -1, lastY = -1;

@Override
public void startInputMonitoring() {
   if (inputMonitoringActive) return;

   inputMonitoringActive = true;

   inputScheduler = Executors.newSingleThreadScheduledExecutor();
   inputScheduler.scheduleAtFixedRate(this::checkInput, 0, 80, TimeUnit.MILLISECONDS);
}

private void checkInput() {
   User32 u = User32.INSTANCE;

   for (int i = 0x08; i <= 0xFE; i++) {
      if ((u.GetAsyncKeyState(i) & 0x0001) != 0) {
         keyPressCount.incrementAndGet();
         lastActivity = System.currentTimeMillis();
      }
```

```
    }

    // мишка
    if ((u.GetAsyncKeyState(0x01) & 1) != 0) mouseClickCount.incrementAndGet();
    if ((u.GetAsyncKeyState(0x02) & 1) != 0) mouseClickCount.incrementAndGet();

    WinDef.POINT p = new WinDef.POINT();
    u.GetCursorPos(p);

    if (lastX != -1 && lastY != -1) {
        if (Math.abs(p.x - lastX) > 3 || Math.abs(p.y - lastY) > 3) {
            mouseMoveCount.incrementAndGet();
            lastActivity = System.currentTimeMillis();
        }
    }

    lastX = p.x;
    lastY = p.y;
}

@Override
public void stopInputMonitoring() {
    inputMonitoringActive = false;
    if (inputScheduler != null) inputScheduler.shutdownNow();
}

@Override
public Map<String, Long> getInputStats() {
    if (!inputMonitoringActive) return Map.of(
            "keys", 0L,
            "clicks", 0L,
```

```
            "moves", 0L
    );

    Map<String, Long> map = new HashMap<>();
    map.put("keys", (long) keyPressCount.get());
    map.put("clicks", (long) mouseClickCount.get());
    map.put("moves", mouseMoveCount.get());
    return map;
}

@Override
public String getActiveWindowTitle() {
    try {
        char[] buffer = new char[512];

        WinDef.HWND hwnd = User32.INSTANCE.GetForegroundWindow();
        if (hwnd == null) return "Unknown";

        User32.INSTANCE.GetWindowText(hwnd, buffer, 512);
        return Native.toString(buffer).trim();
    } catch (Exception e) {
        return "Unknown";
    }
}

@Override
public Map<String, Object> collectAllMetrics() {
    updateDiskStats();

    Map<String, Object> map = new HashMap<>();
    map.put("cpuLoad", getCpuLoad());
```

```
        map.put("ramUsed", getRamUsed());
        map.put("ramTotal", getRamTotal());

        map.put("diskTotal", getDiskTotal());
        map.put("diskFree", getDiskFree());
        map.put("diskUsed", getDiskUsed());
        map.put("diskDetails", diskDetails);

        map.put("uptime", getUptime());
        map.put("uptimeSeconds", getSystemUptimeSecondsRaw());

        map.put("activeWindow", getActiveWindowTitle());
        map.put("osName", "Windows");

        if (inputMonitoringActive) map.putAll(getInputStats());

        return map;
    }
}
```

```
package com.example.modules.monitoring.metrics;

import java.math.BigDecimal;
import java.util.Map;

public interface MetricsProvider {

    BigDecimal getCpuLoad();
    BigDecimal getRamUsed();
    BigDecimal getRamTotal();
    void updateDiskStats();
    BigDecimal getDiskTotal();
```

```
    BigDecimal getDiskFree();
    BigDecimal getDiskUsed();
    String getActiveWindowTitle();
    String getUptime();
    void startInputMonitoring();
    void stopInputMonitoring();
    Map<String, Long> getInputStats();
    Map<String, Object> collectAllMetrics();
}
```