Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Лабораторна робота №2

з дисципліни «Технології розроблення програмного забезпечення»

Виконала:
студентка групи ІА-23
Проценко. В. І.

Перевірив:
Мягкий М. Ю

Київ 2024

Зміст

Тема ……..……..... 3
Завдання ……..……..... 3
1. Короткі теоретичні відомості ……..…….....…..……...…… 3
2. Хід роботи ……..……..…….... 5
2.1. Вибір індивідуальної теми ...…...……..... 5
2.2. Діаграма прецендентів …….....…..... 5
2.3. Діаграма класів ……...…….....…..…..... 7
2.4. Опис прецедентів ……...…….....…..…..... 8
2.5. Основні класи і структура системи баз даних …..…..... 10
2.6. Реалізація шаблону Репозиторію …….....…..…..... 12
2.6. Вихідні коди класів системи …….....…..…..... 12
3. Висновок ……...…….....……....…..... 27

Тема: Діаграма варіантів використання. Сценарії варіантів використання. Діаграми UML. Діаграми класів. Концептуальна модель системи.

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.
2. Проаналізуйте тему та намалюйте схему прецеденту, що відповідає обраній темі лабораторії.
3. Намалюйте діаграму класів для реалізованої частини системи.
4. Виберіть 3 прецеденти і напишіть на їх основі прецеденти.
5. Розробити основні класи і структуру системи баз даних.
6. Класи даних повинні реалізувати шаблон Репозиторію для взаємодії з базою даних.
7. Підготувати звіт про хід виконання лабораторних робіт. Звіт, що подається повинен містити: діаграму прецедентів, діаграму класів системи, вихідні коди класів системи, а також зображення структури бази даних.

1. Короткі теоретичні відомості

1. Діаграми варіантів використання (Use Case Diagrams)
Діаграми варіантів використання в UML показують взаємодію між користувачами (акторами) і системою через різні варіанти використання (прецеденти). Прецеденти відображають функції або дії, які система виконує на запит користувача. Актори — це зовнішні сутності (користувачі або інші системи), які взаємодіють із системою.
Основні елементи діаграми:

- Актор: зовнішній учасник, який взаємодіє з системою.
- Прецедент (Use Case): дія або функція, яку виконує система.
- Зв'язки: асоціації між акторами і прецедентами, що показують, які дії доступні конкретному актору.

2. Діаграми класів (Class Diagrams)
Діаграми класів відображають структуру системи у вигляді класів і їх взаємодій. Вони показують класи, їх атрибути, методи та зв'язки між класами (асоціації, наслідування, агрегації тощо).

Основні елементи:

- Клас: шаблон для створення об'єктів із певними атрибутами і методами.
- Атрибути: характеристики класу.
- Методи: функції, які виконуються класом.
- Зв'язки між класами: показують, як класи взаємодіють між собою.

3. Шаблон репозиторію (Repository Pattern)

Програмний шаблон проєктування, який абстрагує доступ до даних і операції з ними. Основна ідея полягає в тому, щоб створити клас або об'єкт, який відповідає за взаємодію з джерелом даних (базою даних, файловою системою, веб-сервісами тощо), забезпечуючи таким чином чітке розділення між бізнес-логікою програми та операціями з даними.

Основні переваги шаблону репозиторію:

- Інкапсуляція логіки доступу до даних. Репозиторій обмежує прямий доступ до джерела даних, що зменшує залежності від конкретної бази даних або методу зберігання даних.
- Легкість тестування. Завдяки абстрагуванню доступу до даних, репозиторії можна легко підміняти під час тестування (наприклад, використовувати підробки або "mock").
- Чітке розділення відповідальностей. Бізнес-логіка не займається питаннями збереження чи вибірки даних, а зосереджена на власній функціональності.
- Зменшення дублювання коду. У репозиторії можна централізовано управляти всіма операціями з даними, уникнувши повторень у різних частинах коду.

Структура репозиторію:

- Інтерфейс: Визначає базові методи для взаємодії з даними, такі як findById(), findAll(), save(), delete() тощо.
- Реалізація: В класах реалізації виконується конкретна взаємодія з базою даних, веб-сервісами або іншими сховищами.

2. Хід роботи

2.1. Вибір індивідуальної теми.

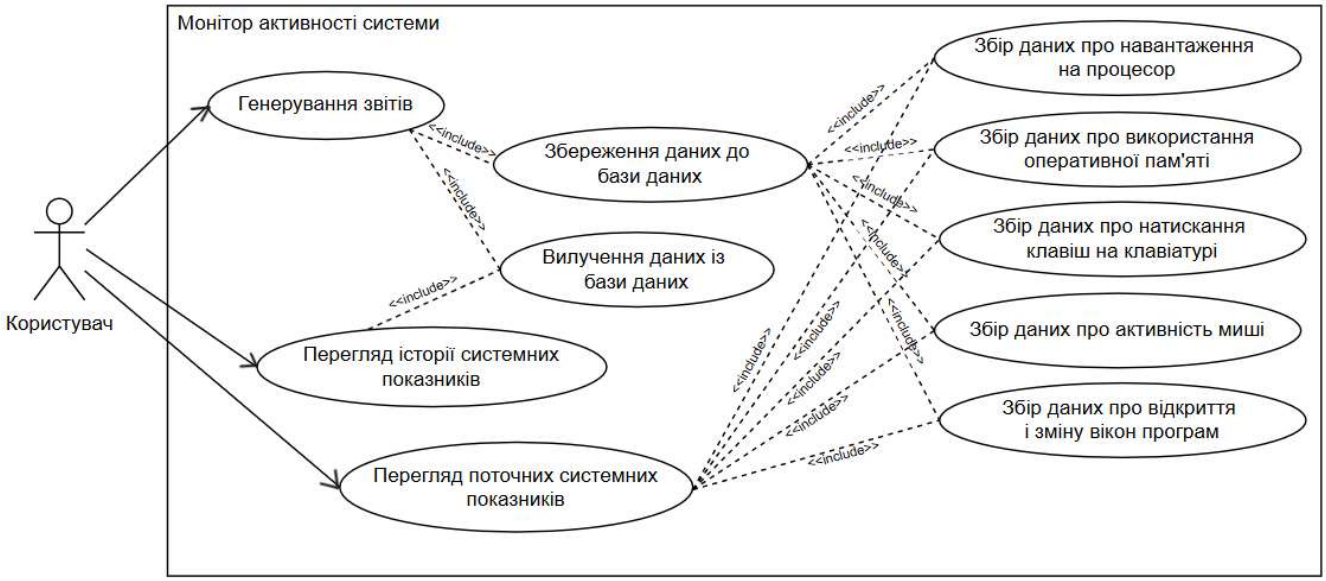
З переліку тем для виконання лабораторних робіт обрано індивідуальну тему. Проаналізовано тему на базі якої будемо у подальшому виконувати лабораторні роботи. А також було ознайомлено з короткими теоретичними відомостями.

..17 System activity monitor (iterator, command, abstract factory, bridge, visitor, SOA)

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.

2.2. Діаграма прецендентів.

Створено діаграму прецедентів для монітору активності системи.



На діаграмі прецедентів показана система "Монітор активності системи", яка збирає та обробляє дані про використання системних ресурсів і надає їх користувачу.

Опис компонентів діаграми:

Актор "Користувач": Це зовнішній учасник, який взаємодіє з системою. Він може переглядати поточні та історичні системні показники, а також генерувати звіти.

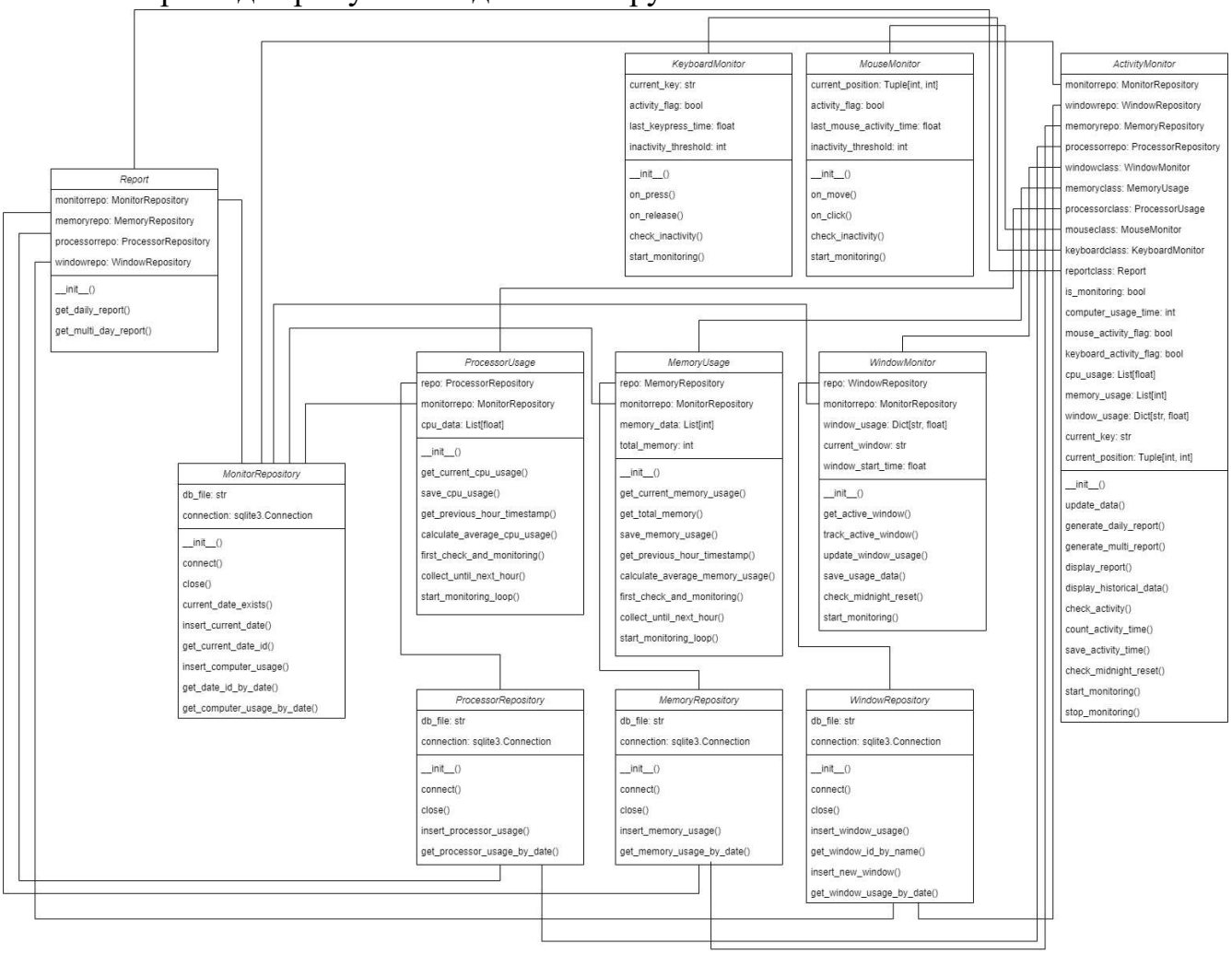
Прецеденти (Use Cases):

- Генерування звітів: Користувач може запитувати звіти, які включають системні дані.
- Збереження даних до бази даних: для того щоб звіти базувалися на актуальних даних, система зберігає інформацію в базі даних.
- Збір даних про навантаження на процесор: система збирає дані про поточний стан використання CPU.
- Збір даних про використання оперативної пам'яті: система відстежує використання оперативної пам’яті.
- Збір даних про натискання клавіш на клавіатурі: фіксуються дії користувача на клавіатурі.
- Збір даних про активність миші: фіксується переміщення та натискання миші.
- Збір даних про відкриття і зміну вікон програм: відстежується відкриття та зміна активних вікон.
- Збереження даних до бази даних: система зберігає зібрані дані для подальшого аналізу або створення звітів.
- Вилучення даних із бази даних: при запиті звітів чи перегляді історичних даних система отримує необхідну інформацію з бази.
- Перегляд історії системних показників: користувач може переглядати збережені в базі дані про минулу активність системи.
- Перегляд поточних системних показників: користувач може переглядати поточний стан системи, включаючи використання ресурсів, активність пристроїв тощо.

Взаємозв'язки (Include): Прецеденти Генерування звітів, Перегляд історії системних показників та Перегляд поточних системних показників залежать від дій зі збору та збереження даних. Вони включають (include) прецеденти, що стосуються збору даних про системні ресурси (процесор, пам'ять, клавіатура, миша, вікна), щоб забезпечити актуальні або завантажити історичні дані.

2.3. Діаграма класів.

Створено діаграму класів для монітору активності системи.



Дана діаграма класів відображає структуру та взаємозв’язки між класами в системі моніторингу активності. Основні компоненти діаграми:

- 1. ActivityMonitor
  Основний клас, що відповідає за моніторинг системної активності. Він управляє початком і завершенням моніторингу, ініціацією збереження і оновлення всіх даних про активність.
- 2. Report
  Клас для формування звітів про використання системи.
- 3. MemoryUsage
  Відповідає за моніторинг використання оперативної пам'яті.
- 4. ProcessorUsage
  Відповідає за моніторинг навантаження на процесор.

- 5. WindowMonitor
  Відповідає за моніторинг активності вікон програм.
- 6. KeyboardMonitor
  Відповідає за моніторинг натискання клавіш.
- 7. MouseMonitor
  Відповідає за моніторинг дій миші.
- 8. Класи Repository Patterns
  Кожен компонент що має зв’язок з датабазою та потребує збереження даних має відповідний клас репозиторію (наприклад, MemoryRepository, ProcessorRepository, WindowRepository, MonitorRepository), що реалізує зв’язок із датабазою.

2.4. Опис прецедентів.

Прецедент 1: Перегляд поточних системних показників

- Опис: Користувач запитує перегляд поточних системних показників, таких як навантаження на процесор, використання оперативної пам'яті, активність миші, клавіатури та вікон програм.
- Актор: Користувач
  Передумови: Система моніторингу активності запущена та активно збирає дані.
- Основний сценарій:
  1. Користувач відкриває інтерфейс системи моніторингу активності.
  2. Користувач вибирає опцію "Перегляд поточних системних показників".
  3. Система викликає прецеденти збору даних: Збір даних про навантаження на процесор. Збір даних про використання оперативної пам'яті. Збір даних про активність миші. Збір даних про натискання клавіш. Збір даних про відкриття і зміну вікон програм.
  4. Система обробляє зібрані дані та відображає поточні системні показники користувачу.
  5. Користувач переглядає відображені показники. Користувач закриває інтерфейс перегляду поточних показників.
- Постумови: Користувач успішно переглянув поточні системні показники.

Прецедент 2: Генерування звітів

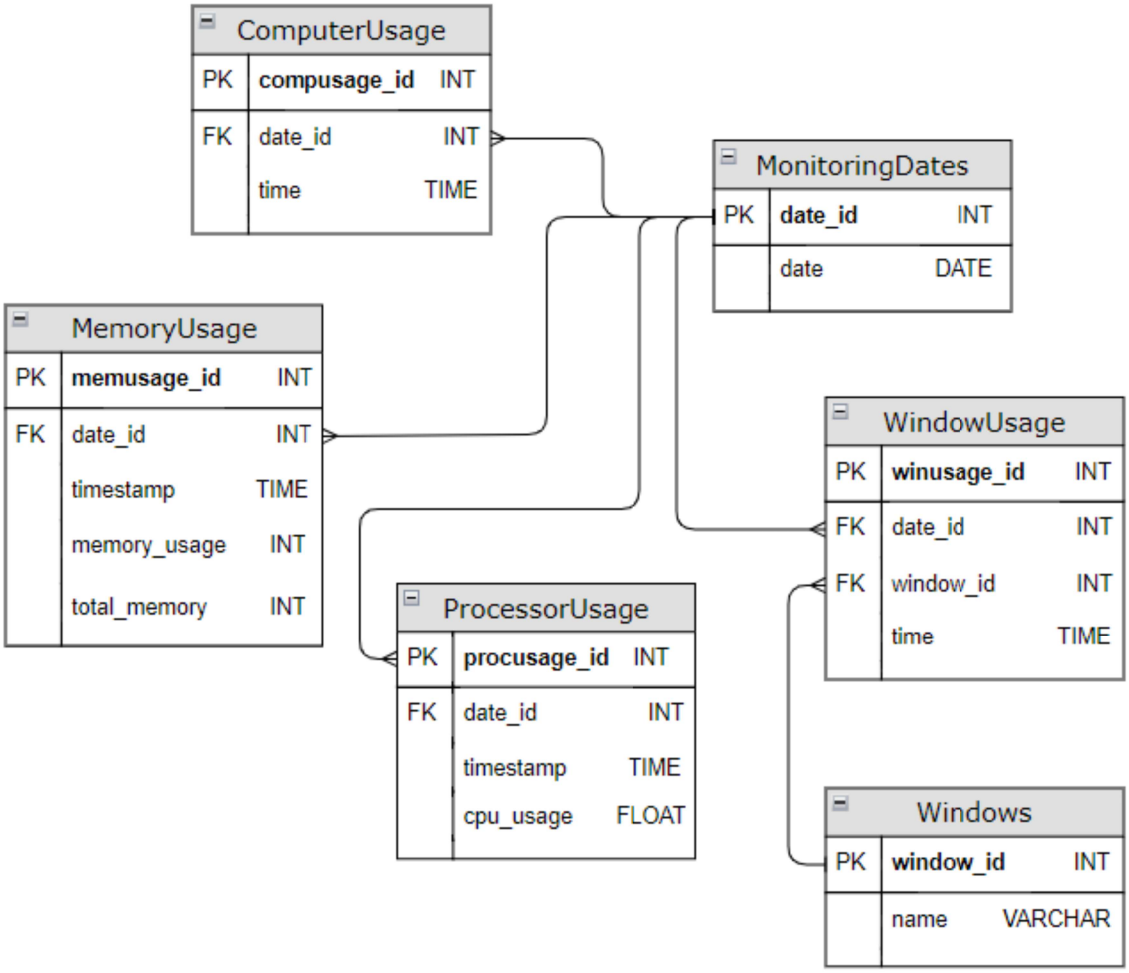
- Опис: Користувач ініціює процес генерації звіту для аналізу використання системних ресурсів за певний період.
- Актор: Користувач

- Передумови: Дані про активність системи зберігаються в базі даних.
- Основний сценарій:
  1. Користувач заходить у систему і вибирає опцію "Генерація звітів".
  2. Система пропонує користувачу вибрати період часу, за який потрібно згенерувати звіт.
  3. Користувач вказує період часу.
  4. Система отримує дані з бази даних, викликаючи прецедент "Вилучення даних із бази даних".
  5. Система генерує звіт, включаючи такі показники: Середнє навантаження на процесор. Використання оперативної пам'яті. Час використання програм.
  6. Згенерований звіт відображається користувачу або зберігається в системі для подальшого перегляду.
- Постумови: Звіт успішно згенеровано, користувач може його переглянути або зберегти.

<u>Прецедент 3</u>: Перегляд системних показників за певну дату

- Опис: Користувач ініціює запит на перегляд системних показників за конкретну дату для аналізу роботи системи в минулому.
- Актор: Користувач
- Передумови: Дані про активність системи за обрану дату зберігаються в базі даних.
- Основний сценарій:
  1. Користувач входить у систему моніторингу активності.
  2. Користувач обирає опцію "Перегляд історії системних показників".
  3. Система запитує вказати конкретну дату для перегляду.
  4. Користувач обирає потрібну дату.
  5. Система викликає прецедент "Вилучення даних із бази даних", отримуючи інформацію про системні показники за обрану дату.
  6. Система обробляє отримані дані: Середнє навантаження на процесор за день. Використання оперативної пам'яті протягом дня. Процент використання вікон програм.
  7. Користувач переглядає відображені системні показники за обрану дату.
- Альтернативні сценарії: Якщо за обрану дату відсутні дані, система повідомляє користувача про відсутність записів і пропонує обрати іншу дату.
- Постумови: Користувач успішно переглянув системні показники за обрану дату.

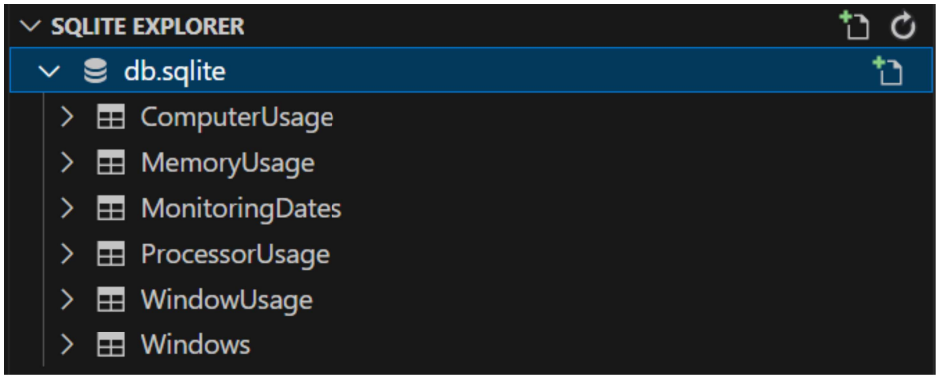
2.5. Основні класи і структура системи баз даних.
Структура бази даних наведена на схемі.



Створення бази даних за наведеною схемою.

Query1.sql

```sql
CREATE TABLE MonitoringDates (
    date_id INT PRIMARY KEY,
    date DATE
);

CREATE TABLE ComputerUsage (
    compusage_id INT PRIMARY KEY,
    date_id INT NOT NULL,
    time TIME NOT NULL,
    FOREIGN KEY (date_id) REFERENCES MonitoringDates(date_id)
);

CREATE TABLE MemoryUsage (
    memusage_id INT PRIMARY KEY,
    date_id INT NOT NULL,
    timestamp TIME NOT NULL,
    memory_usage INT NOT NULL,
    total_memory INT NOT NULL,
    FOREIGN KEY (date_id) REFERENCES MonitoringDates(date_id)
);

CREATE TABLE ProcessorUsage (
    procusage_id INT PRIMARY KEY,
    date_id INT NOT NULL,
    timestamp TIME NOT NULL,
    cpu_usage FLOAT NOT NULL,
    FOREIGN KEY (date_id) REFERENCES MonitoringDates(date_id)
);

CREATE TABLE Windows (
    window_id INT PRIMARY KEY,
    name VARCHAR(255) NOT NULL
);

CREATE TABLE WindowUsage (
    winusage_id INT PRIMARY KEY,
    date_id INT NOT NULL,
    window_id INT NOT NULL,
    time TIME NOT NULL,
    FOREIGN KEY (date_id) REFERENCES MonitoringDates(date_id),
    FOREIGN KEY (window_id) REFERENCES Windows(window_id)
);
```



2.6. Реалізація шаблону Репозиторію.

Реалізація шаблону досягається використанням класів MemoryRepository, ProcessorRepository, WindowRepository, MonitorRepository що реалізують зв'язок із базою даних та відповідають за обмін та збереження даних.

2.7. Вихідні коди класів системи.

1. ActivityMonitor

activity_monitor.py > ...

```python
import time
import datetime
import threading
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk
from report import Report
from keyboard_monitor import KeyboardMonitor
from mouse_monitor import MouseMonitor
from memory_usage import MemoryUsage
from processor_usage import ProcessorUsage
from window_monitor import WindowMonitor
from repositories.memory_repository import MemoryRepository
from repositories.processor_repository import ProcessorRepository
from repositories.window_repository import WindowRepository
from repositories.monitor_repository import MonitorRepository


class ActivityMonitor:
    def __init__(self, db_file):
        self.monitorrepo = MonitorRepository(db_file)
        self.windowrepo = WindowRepository(db_file)
        self.memoryrepo = MemoryRepository(db_file)
        self.processorrepo = ProcessorRepository(db_file)
        self.windowclass = WindowMonitor(db_file)
        self.memoryclass = MemoryUsage(db_file)
        self.processorclass = ProcessorUsage(db_file)
        self.mouseclass = MouseMonitor(db_file)
        self.keyboardclass = KeyboardMonitor(db_file)
```

```python
        self.reportclass = Report(db_file)
        self.is_monitoring = False
        self.computer_usage_time = 0
        self.mouse_activity_flag = True
        self.keyboard_activity_flag = True
        self.cpu_usage = []
        self.memory_usage = []
        self.window_usage = {}
        self.current_key = ""
        self.current_position = (0, 0)

    def update_data(self, cpu_data, memory_data, window_usage, current_key, current_position):
        # Implement

    def generate_daily_report(self, date):
        # Implement

    def generate_multi_report(self, date_from, date_to):
        # Implement

    def display_report(self):
        # Implement

    def display_report(self, report_data):
        # Implement

    def display_historical_data(self, historical_data):
        # Implement

    def check_activity(self, historical_data):
        # Implement

    def count_activity_time(self):
        # Implement

    def save_activity_time(self):
        # Implement

    def check_midnight_reset(self):
        # Implement

    def start_monitoring(self):
        # Implement

    def stop_monitoring(self):
        # Implement
```

2. Report

report.py > ...

```python
import datetime
import json
from repositories.monitor_repository import MonitorRepository
from repositories.memory_repository import MemoryRepository
from repositories.processor_repository import ProcessorRepository
from repositories.window_repository import WindowRepository

class Report:
    def __init__(self, db_file):
        self.monitorrepo = MonitorRepository(db_file)
        self.memoryrepo = MemoryRepository(db_file)
        self.processorrepo = ProcessorRepository(db_file)
        self.windowrepo = WindowRepository(db_file)

    def get_daily_report(self, date):
        date_id = self.monitorrepo.get_current_date_id()
        if date_id is None:
            print("Date not found.")
            return

        total_usage_time = self.monitorrepo.get_computer_usage_by_date(date_id)
        window_usage = self.windowrepo.get_window_usage_by_date(date_id)

        all_memory = self.memoryrepo.get_memory_usage_by_date(date_id)
        if not all_memory:
            print("No memory usage data found.")
            memory_data_list = []
            average_memory_usage = 0
        else:
            memory_data_list = []
            for record in all_memory:
                timestamp, memory_usage, total_memory = record
                memory_data_list.append({
                    "timestamp": timestamp,
                    "memory_usage": memory_usage,
                    "total_memory": total_memory
                })
        memory_data_json = json.dumps(memory_data_list, indent=4)
        total_memory_usage = 0
        memory_record_count = 0
        for memory_data in memory_data_list:
            total_memory_usage += memory_data['memory_usage']
            memory_record_count += 1
        if memory_record_count == 0:
            print("No records found to calculate the average memory.")
            average_memory_usage = 0
        else:
            average_memory_usage = total_memory_usage / memory_record_count
```

```python

        all_cpu = self.processorrepo.get_processor_usage_by_date(date_id)
        if not all_cpu:
            print("No cpu usage data found.")
            cpu_data_list = []
            average_cpu_usage = 0
        else:
            cpu_data_list = []
            for record in all_cpu:
                timestamp, cpu_usage, = record
                cpu_data_list.append({
                    "timestamp": timestamp,
                    "cpu_usage": cpu_usage,
                })
        cpu_data_json = json.dumps(cpu_data_list, indent=4)
        total_cpu_usage = 0
        cpu_record_count = 0
        for cpu_data in cpu_data_list:
            total_cpu_usage += cpu_data['cpu_usage']
            cpu_record_count += 1
        if cpu_record_count == 0:
            print("No records found to calculate the average memory.")
            average_cpu_usage = 0
        else:
            average_cpu_usage = total_cpu_usage / cpu_record_count

        report = {
            "date": date,
            "total_usage_time": sum(total_usage_time),
            "window_usage": window_usage,
            "memory_json": memory_data_json,
            "average_memory": average_memory_usage,
            "cpu_json": cpu_data_json,
            "average_cpu": average_cpu_usage
        }
        return report

    def get_multi_day_report(self, date_from, date_to):
        date_from = datetime.datetime.strptime(date_from, "%Y-%m-%d")
        date_to = datetime.datetime.strptime(date_to, "%Y-%m-%d")
        current_date = date_from

        daily_reports = []
        total_usage_time = 0
        total_days = 0

        while current_date <= date_to:
            date_str = current_date.strftime("%Y-%m-%d")
            daily_report = self.get_daily_report(date_str)
```

```python

            if daily_report:
                daily_reports.append(daily_report)
                total_usage_time += daily_report["total_usage_time"]
                total_days += 1

            current_date += datetime.timedelta(days=1)

        average_usage_time = total_usage_time / total_days if total_days > 0 else 0

        average_memory = self.memoryrepo.get_average_memory_over_period(date_from, date_to)
        # Implement this
        average_cpu = self.processorrepo.get_average_cpu_over_period(date_from, date_to)
        # Implement this

        summary_report = {
            "daily_reports": daily_reports,
            "average_usage_time": average_usage_time,
            "average_memory": average_memory,
            "average_cpu": average_cpu
        }
        return summary_report
```

3. MemoryUsage

memory_usage.py > ...

```python
import psutil
import time
import datetime
from repositories.memory_repository import MemoryRepository
from repositories.monitor_repository import MonitorRepository


class MemoryUsage:
    def __init__(self, db_file):
        self.repo = MemoryRepository(db_file)
        self.monitorrepo = MonitorRepository(db_file)
        self.memory_data = []
        self.total_memory = self.get_total_memory()

    def get_current_memory_usage(self):
        memory_info = psutil.virtual_memory()
        return int(memory_info.used / (1024 ** 2))  # Return in MB

    def get_total_memory(self):
        memory_info = psutil.virtual_memory()
        return int(memory_info.total / (1024 ** 2))  # Return in MB

    def save_memory_usage(self, average_used_memory):
        current_date = datetime.datetime.now().strftime("%Y-%m-%d")
```

```python

        if not self.monitorrepo.current_date_exists(current_date):
            self.monitorrepo.insert_current_date()

        date_id = self.monitorrepo.get_current_date_id()
        timestamp = self.get_previous_hour_timestamp()

        self.repo.insert_memory_usage(date_id, timestamp, average_used_memory, self.total_memory)

    def get_previous_hour_timestamp(self):
        previous_hour = datetime.datetime.now() - datetime.timedelta(hours=1)
        return previous_hour.strftime("%H:%M:%S")

    def calculate_average_memory_usage(self):
        if not self.memory_data:
            return 0  # Avoid division by zero
        return sum(self.memory_data) / len(self.memory_data)

    def first_check_and_monitoring(self):
        current_time = datetime.datetime.now()
        if current_time.minute == 0 and current_time.second == 0:
            print("Start of an hour detected. Beginning regular monitoring.")
            self.start_monitoring_loop()
        else:
            next_hour = (current_time + datetime.timedelta(hours=1)).replace(minute=0, second=0, microsecond=0)
            time_to_next_hour = (next_hour - current_time).total_seconds()
            print(f"Waiting for the next full hour in {time_to_next_hour // 60} minutes.")
            self.collect_until_next_hour(time_to_next_hour)

    def collect_until_next_hour(self, time_to_next_hour):
        start_time = time.time()
        while time.time() - start_time < time_to_next_hour:
            used_memory = self.get_current_memory_usage()
            print(used_memory)
            self.memory_data.append(used_memory)
            time.sleep(1)

        average_used = self.calculate_average_memory_usage()
        self.save_memory_usage(average_used)
        self.memory_data = []
        self.start_monitoring_loop()

    def start_monitoring_loop(self):
        while True:
            start_time = time.time()
            for _ in range(3600):
                used_memory = self.get_current_memory_usage()
                print(used_memory)
                self.memory_data.append(used_memory)
                time.sleep(1)

            average_used = self.calculate_average_memory_usage()
            self.save_memory_usage(average_used)
            self.memory_data = []
```

4. ProcessorUsage

processor_usage.py > ...

```python
import psutil
import time
import datetime
from repositories.processor_repository import ProcessorRepository
from repositories.monitor_repository import MonitorRepository


class ProcessorUsage:
    def __init__(self, db_file):
        self.repo = ProcessorRepository(db_file)
        self.monitorrepo = MonitorRepository(db_file)
        self.cpu_data = []

    def get_current_cpu_usage(self):
        return psutil.cpu_percent(interval=1)

    def save_cpu_usage(self, average_cpu_usage):
        current_date = datetime.datetime.now().strftime("%Y-%m-%d")
        if not self.monitorrepo.current_date_exists(current_date):
            self.monitorrepo.insert_current_date()

        timestamp = self.get_previous_hour_timestamp()

        self.repo.insert_cpu_usage(timestamp, average_cpu_usage)

    def get_previous_hour_timestamp(self):
        previous_hour = datetime.datetime.now() - datetime.timedelta(hours=1)
        return previous_hour.strftime("%H:%M:%S")

    def calculate_average_cpu_usage(self):
        if not self.cpu_data:
            return 0  # Avoid division by zero
        return sum(self.cpu_data) / len(self.cpu_data)

    def first_check_and_monitoring(self):
        current_time = datetime.datetime.now()
        if current_time.minute == 0 and current_time.second == 0:
            print("Start of an hour detected. Beginning regular monitoring.")
            self.start_monitoring_loop()
        else:
            next_hour = (current_time + datetime.timedelta(hours=1)).replace(minute=0, second=0, microsecond=0)
            time_to_next_hour = (next_hour - current_time).total_seconds()
            print(f"Waiting for the next full hour in {time_to_next_hour // 60} minutes.")
            self.collect_until_next_hour(time_to_next_hour)

    def collect_until_next_hour(self, time_to_next_hour):
        start_time = time.time()
        while time.time() - start_time < time_to_next_hour:
            used_cpu = self.get_current_cpu_usage()
            print(used_cpu)
            self.cpu_data.append(used_cpu)
            time.sleep(1)
```

```python

        average_used = self.calculate_average_cpu_usage()
        self.save_cpu_usage(average_used)
        self.cpu_data = []
        self.start_monitoring_loop()

    def start_monitoring_loop(self):
        while True:
            start_time = time.time()
            for _ in range(3600):
                used_cpu = self.get_current_cpu_usage()
                print(used_cpu)
                self.cpu_data.append(used_cpu)
                time.sleep(1)

            average_used = self.calculate_average_cpu_usage()
            self.save_cpu_usage(average_used)
            self.cpu_data = []
```

5. WindowMonitor

window_monitor.py > ...

```python
import time
import datetime
import pygetwindow as gw  # on Windows
from repositories.window_repository import WindowRepository
from repositories.monitor_repository import MonitorRepository

class WindowMonitor:
    def __init__(self, db_file):
        self.repo = WindowRepository(db_file)
        self.monitorrepo = MonitorRepository(db_file)
        self.current_window = None
        self.window_start_time = None
        self.window_usage = {}

    def get_active_window(self):
        active_window = gw.getActiveWindow()
        if active_window:
            return active_window.title
        return None

    def track_active_window(self):
        active_window = self.get_active_window()

        if active_window != self.current_window:
            if self.current_window is not None:
                elapsed_time = time.time() - self.window_start_time
                elapsed_time_minutes = round(elapsed_time / 60, 2)
                self.update_window_usage(self.current_window, elapsed_time_minutes)
            self.current_window = active_window
            self.window_start_time = time.time()

        time.sleep(1)
```

```python

    def update_window_usage(self, window_name, elapsed_time_minutes):
        if window_name not in self.window_usage:
            self.window_usage[window_name] = elapsed_time_minutes
        else:
            self.window_usage[window_name] += elapsed_time_minutes

    def save_usage_data(self):
        current_date = datetime.datetime.now().strftime("%Y-%m-%d")
        if not self.monitorrepo.current_date_exists(current_date):
            self.monitorrepo.insert_current_date()
        for window_name, usage_time in self.window_usage.items():
            self.repo.insert_window_usage(window_name, usage_time)

    def check_midnight_reset(self):
        current_time = datetime.datetime.now().time()
        if current_time.hour == 0 and current_time.minute == 0:
            self.save_usage_data()
            self.window_usage = {}

    def start_monitoring(self):
        try:
            while True:
                self.track_active_window()
                self.check_midnight_reset()
        except KeyboardInterrupt:
            print("Window monitoring stopped.")
```

6. KeyboardMonitor

keyboard_monitor.py > ...

```python
import time
from pynput import keyboard
import threading

class KeyboardMonitor:
    def __init__(self):
        self.current_key = None
        self.activity_flag = True
        self.last_keypress_time = time.time()
        self.inactivity_threshold = 10 * 60

    def on_press(self, key):
        try:
            self.current_key = key.char
            print(f"Key pressed: {self.current_key}")
        except AttributeError:
            self.current_key = str(key)
            print(f"Special key pressed: {self.current_key}")
```

```python

        self.last_keypress_time = time.time()
        self.activity_flag = True

    def on_release(self, key):
        self.current_key = None
        print("None")

    def check_inactivity(self):
        while True:
            current_time = time.time()
            if current_time - self.last_keypress_time > self.inactivity_threshold:
                if self.activity_flag:
                    print("No activity for 10 minutes.")
                self.activity_flag = False
            time.sleep(1)

    def start_monitoring(self):
        inactivity_thread = threading.Thread(target=self.check_inactivity)
        inactivity_thread.daemon = True
        inactivity_thread.start()

        with keyboard.Listener(on_press=self.on_press, on_release=self.on_release) as listener:
            listener.join()
```

7. MouseMonitor

mouse_monitor.py > ...

```python
import time
from pynput import mouse
import threading

class MouseMonitor:
    def __init__(self):
        self.current_position = (0, 0)
        self.activity_flag = True
        self.last_mouse_activity_time = time.time()
        self.inactivity_threshold = 10

    def on_move(self, x, y):
        self.current_position = (x, y)
        print(f"Mouse moved to: {self.current_position}")
        self.last_mouse_activity_time = time.time()
        self.activity_flag = True

    def on_click(self, x, y, button, pressed):
        action = 'pressed' if pressed else 'released'
        print(f"Mouse {action} at ({x}, {y}) with {button}.")
        self.last_mouse_activity_time = time.time()
        self.activity_flag = True
```

```python

    def check_inactivity(self):
        while True:
            current_time = time.time()
            if current_time - self.last_mouse_activity_time > self.inactivity_threshold:
                if self.activity_flag:
                    print("No mouse activity for 10 minutes.")
                self.activity_flag = False
            time.sleep(1)

    def start_monitoring(self):
        inactivity_thread = threading.Thread(target=self.check_inactivity)
        inactivity_thread.daemon = True
        inactivity_thread.start()

        with mouse.Listener(on_move=self.on_move, on_click=self.on_click) as listener:
            listener.join()
```

8. MonitorRepository

repositories > monitor_repository.py > ...

```python
import sqlite3
import datetime

class MonitorRepository:
    def __init__(self, db_file):
        self.db_file = db_file
        self.connect()

    def connect(self):
        try:
            self.connection = sqlite3.connect(self.db_file)
            print("Connected to database successfully.")
        except sqlite3.Error as e:
            print(f"Error connecting to database: {e}")

    def close(self):
        if self.connection:
            self.connection.close()
            print("Database connection closed.")

    def current_date_exists(self, current_date):
        query = "SELECT 1 FROM MonitoringDates WHERE date = ?"
        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (current_date,))
            return cursor.fetchone() is not None
        except sqlite3.Error as e:
            print(f"Error checking for current date: {e}")
            return False
```

```python

    def insert_current_date(self):
        current_date = datetime.datetime.now().strftime("%Y-%m-%d")

        if not self.current_date_exists(current_date):
            query = """
            INSERT INTO MonitoringDates (date_id, date)
            VALUES ((SELECT IFNULL(MAX(date_id), 0) + 1 FROM MonitoringDates), ?)
            """
            try:
                cursor = self.connection.cursor()
                cursor.execute(query, (current_date,))
                self.connection.commit()
                print(f"Current date {current_date} inserted successfully.")
            except sqlite3.Error as e:
                print(f"Error inserting current date: {e}")
        else:
            print(f"Date {current_date} already exists in the database.")

    def get_current_date_id(self):
        current_date = datetime.datetime.now().strftime("%Y-%m-%d")
        query = "SELECT date_id FROM MonitoringDates WHERE date = ?"
        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (current_date))
            result = cursor.fetchone()
            return result[0] if result else None
        except sqlite3.Error as e:
            print(f"Error fetching current date_id: {e}")
            return None

    def insert_computer_usage(self, time):
        query = """
        INSERT INTO ComputerUsage (compusage_id, date_id, time)
        VALUES ((SELECT IFNULL(MAX(compusage_id), 0) + 1 FROM ComputerUsage), (SELECT date_id FROM M
        """
        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (time))
            self.connection.commit()
            print(f"Computer usage data saved: {time} minutes for today.")
        except sqlite3.Error as e:
            print(f"Error saving computer usage data: {e}")

    def get_date_id_by_date(self, date):
        query = "SELECT date_id FROM MonitoringDates WHERE date = ?"
        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (date))
            result = cursor.fetchone()
            return result[0] if result else None
        except sqlite3.Error as e:
            print(f"Error fetching current date_id: {e}")
            return None
```

```python

    def get_computer_usage_by_date(self, date_id):
        query = "SELECT time FROM ComputerUsage WHERE date_id = ?"
        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (date_id,))
            results = cursor.fetchall()
            return [row[0] for row in results]
        except sqlite3.Error as e:
            print(f"Error fetching computer usage by date: {e}")
            return None
```

9. MemoryRepository

repositories > memory_repository.py > ...

```python
import sqlite3

class MemoryRepository:
    def __init__(self, db_file):
        self.db_file = db_file
        self.connect()

    def connect(self):
        try:
            self.connection = sqlite3.connect(self.db_file)
            print("Connected to database successfully.")
        except sqlite3.Error as e:
            print(f"Error connecting to database: {e}")

    def close(self):
        try:
            self.connection.close()
        except sqlite3.Error as e:
            print(f"Error closing the database: {e}")

    def insert_memory_usage(self, timestamp, memory_usage, total_memory):
        query = """
        INSERT INTO MemoryUsage (memusage_id, date_id, timestamp, memory_usage, total_memory)
        VALUES ((SELECT IFNULL(MAX(memusage_id), 0) + 1 FROM MemoryUsage), (SELECT date_id FROM MonitoringDates
        """
        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (date_id, timestamp, memory_usage, total_memory))
            self.connection.commit()
            print(f"Memory usage data saved: {memory_usage} MB at {timestamp}")
        except sqlite3.Error as e:
            print(f"Error saving memory usage data: {e}")

    def get_memory_usage_by_date(self, date_id):
        query = """
        SELECT timestamp, memory_usage, total_memory
        FROM MemoryUsage
        WHERE date_id = ?
        """
        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (date_id,))
            return cursor.fetchall()
        except sqlite3.Error as e:
            print(f"Error retrieving memory usage data: {e}")
            return None
```

10. ProcessorRepository

repositories > processor_repository.py > ...

```python
import sqlite3

class ProcessorRepository:
    def __init__(self, db_file):
        self.db_file = db_file
        self.connect()

    def connect(self):
        try:
            self.connection = sqlite3.connect(self.db_file)
            print("Connected to database successfully.")
        except sqlite3.Error as e:
            print(f"Error connecting to database: {e}")

    def close(self):
        try:
            self.connection.close()
        except sqlite3.Error as e:
            print(f"Error closing the database: {e}")

    def insert_processor_usage(self, timestamp, cpu_usage):
        query = """
        INSERT INTO ProcessorUsage (procusage_id, date_id, timestamp, cpu_usage)
        VALUES ((SELECT IFNULL(MAX(procusage_id), 0) + 1 FROM ProcessorUsage), (SELECT date_id FROM MonitoringDat
        """
        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (timestamp, cpu_usage))
            self.connection.commit()
            print(f"CPU usage for {timestamp} inserted successfully.")
        except sqlite3.Error as e:
            print(f"Error inserting CPU usage: {e}")

    def get_processor_usage_by_date(self, date_id):
        query = """
        SELECT timestamp, cpu_usage
        FROM ProcessorUsage
        WHERE date_id = ?
        """
        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (date_id,))
            return cursor.fetchall()
        except sqlite3.Error as e:
            print(f"Error retrieving memory usage data: {e}")
            return None
```

11. WindowRepository

repositories > window_repository.py > ...

```python
import sqlite3

class WindowRepository:
    def __init__(self, db_file):
        self.db_file = db_file
        self.connect()

    def connect(self):
        try:
            self.connection = sqlite3.connect(self.db_file)
            print("Connected to database successfully.")
        except sqlite3.Error as e:
            print(f"Error connecting to database: {e}")

    def close(self):
        try:
            self.connection.close()
        except sqlite3.Error as e:
            print(f"Error closing the database: {e}")

    def insert_window_usage(self, window_name, usage_time):
        window_id = self.get_window_id_by_name(window_name)
        query = """
        INSERT INTO WindowUsage (winusage_id, date_id, window_id, time)
        VALUES ((SELECT IFNULL(MAX(winusage_id), 0) + 1 FROM WindowUsage), (SELECT date_id FROM
        """
        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (window_id, usage_time))
            self.connection.commit()
            print(f"Window usage for {window_name} saved: {usage_time} seconds.")
        except sqlite3.Error as e:
            print(f"Error saving window usage: {e}")

    def get_window_id_by_name(self, window_name):
        query = "SELECT window_id FROM Windows WHERE name = ?"
        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (window_name,))
            result = cursor.fetchone()
            if result:
                return result[0]
            else:
                return self.insert_new_window(window_name)
        except sqlite3.Error as e:
            print(f"Error getting window ID for {window_name}: {e}")
            return None
```

```python

    def insert_new_window(self, window_name):
        query = "INSERT INTO Windows (name) VALUES (?)"
        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (window_name,))
            self.connection.commit()
            new_window_id = cursor.lastrowid
            print(f"New window {window_name} inserted with ID: {new_window_id}")
            return new_window_id
        except sqlite3.Error as e:
            print(f"Error inserting new window {window_name}: {e}")
            return None

    def get_window_usage_by_date(self, date):
        query = """
        SELECT window_name, usage_time FROM WindowUsage
        JOIN MonitoringDates ON WindowUsage.date_id = MonitoringDates.date_id
        WHERE MonitoringDates.date = ?
        """
        try:
            cursor = self.connection.cursor()
            cursor.execute(query, (date,))
            return cursor.fetchall()
        except sqlite3.Error as e:
            print(f"Error fetching window usage for date {date}: {e}")
            return []
```

3. Висновок

У ході даної лабораторної роботи було проаналізовано тему проєкту та надані теоретичні відомості створено діаграму прецедентів, діаграму класів системи, попередні вихідні коди класів системи, а також зображення структури бази даних згідно вимог до обраної системи.