МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА СИСТЕМИ ВІДДАЛЕНОГО МОНІТОРИНГУ ВИКОРИСТАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗА ДОПОМОГОЮ NODE.JS ТА POSTGRESQL»**

Виконав: студент 2 курсу, групи 6.1213-пі-с3
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія (зі скороченим терміном навчання)
(назва освітньої програми)
В.С. Воробець
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Лимаренко Ю.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри електроніки, інформаційних систем та програмного забезпечення ІННІ ЗНУ, доцент, к.т.н. Заяц В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія (зі скороченим терміном навчання)

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Воробцю Віталію Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка системи віддаленого моніторингу використання програмного забезпечення за допомогою Node.js та PostgreSQL

керівник роботи Лимаренко Юлія Олексіївна, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області.
2. Розробка проєкту застосунку.
3. Програмна реалізація застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.01.2025	
2.	Збір вихідних даних.	03.02.2025	
3.	Обробка методичних та теоретичних	14.03.2025	
	джерел.		
4.	Розробка першого та другого розділу.	28.04.2025	
5.	Розробка третього розділу.	19.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	20.06.2025	

Студент _____ В.С. Воробець
(підпис) (ініціали та прізвище)

Керівник роботи _____ Ю.О. Лимаренко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка системи віддаленого моніторингу використання програмного забезпечення за допомогою Node.js та PostgreSQL»: 48 с., 22 рис., 1 табл., 22 джерела.

БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ЗНІМОК, МОНІТОРИНГ, ОПЕРАЦІЙНА СИСТЕМА, ПРОЦЕС.

Об’єкт дослідження – система для відстеження часу роботи застосунків у користувачів.

Мета роботи: розробка системи для відстеження часу роботи застосунків.

Метод дослідження – аналіз та збір вимог до програмного забезпечення, підходи до моделювання, проєктування, розробки й тестування програмного застосунку.

У кваліфікаційній роботі викладено теоретичні відомості про способи отримання інформації про процеси та приклади сучасних застосунків відстеження часу. На основі даних теоретичних відомостей розроблено систему, яка відстежує роботу активні/сфокусовані вікна, скриншоти та відправляє їх до серверу. За допомогою Node.JS та Svelte був розроблений вебзастосунок для більш зручного використання системи.

SUMMARY

Bachelor’s qualifying paper “Development of a Remote Monitoring System for Software Usage Using Node.js and PostgreSQL”: 48 pages, 22 figures, 1 table, 22 references.

APPLICATION SNAPSHOT, DATABASE, MONITORING, OPERATING SYSTEM, PROCESS, WEB APPLICATION.

The object of research is the system for tracking application uptime.

The aim of the study is to develop the application for tracking application uptime.

The methods of research are analysis and collecting of software requirements, methods of modelling, design, development, and testing of the application.

The qualification work contains theoretical information about the methods of obtaining information about processes and examples of time-tracking applications. Based on this theoretical knowledge, a system has been developed that tracks the active/focused processes, takes screenshots, and sends them to a server. A web application has been developed using Node.js and Svelte for more convenient use of the system.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Аналіз предметної області 9
1.1 Поняття моніторингу та його види 9
1.2 Отримання необхідних даних з системи 10
1.3 Огляд альтернативних застосунків 10
1.3.1 Steam 11
1.3.2 ActivityWatch 11
1.3.3 Kickidler 11
1.4 Засоби для створення системи 12
1.4.1 Огляд фреймворку Qt 12
1.4.2 Огляд СУБД 13
1.4.3 Огляд платформи Node.js 14
1.4.4 Огляд фреймворку Express.js 14
1.4.5 Огляд фреймворку Svelte 15
2 Розробка проєкту застосунку 16
2.1 Технічне завдання 16
2.1.1 Найменування і область застосування 16
2.1.2 Призначення розробки 16
2.1.3 Технічні вимоги до програмного продукту 17
2.2 Етапи створення застосунку 18
2.3 Діаграма прецедентів 20
2.4 ER-діаграма бази даних 21
2.5 Діаграма класів 22
2.6 Діаграма послідовності 24
2.7 Проєктування інтерфейсу 24

3 Програмна реалізація застосунку 28
3.1 Реалізація клієнтського застосунку 28
3.1.1 Розробка модулю процесу 29
3.1.2 Розробка модулю знімку 30
3.1.3 Розробка тимчасової бази даних 31
3.1.4 Розробка системи моніторингу 32
3.1.5 Розробка графічного інтерфейсу 32
3.2 Реалізація вебсерверу 34
3.2.1 Розробка модулю бази даних 35
3.2.2 Збереження зображень на сервері 36
3.2.3 Розробка запитів 36
3.3 Реалізація вебсайту 38
3.3.1 Сторінки входу та реєстрації 40
3.3.2 Сторінка користувача 40
3.4 Інтеграція клієнтського додатку з вебсервером 41
3.5 Тестування системи 43
Висновки 45
Перелік посилань 46

ВСТУП

В наш час люди використовують багато різноманітних програм для роботи, навчання, спілкування, тощо, проте деякі з них часто відволікають увагу, особливо коли необхідна концентрація для важливих задач.

Мета дослідження – розробка програмного застосунку для моніторингу виконання програм та екрану користувача. Для досягнення мети було сформульовано наступні завдання:

– проаналізувати існуючі застосунки для моніторингу активності програм та екрану користувача;
– описати інструменти, за допомогою яких такий застосунок може бути реалізований;
– зробити програмну реалізацію застосунку.

Об’єкт дослідження – основні методи та засоби моніторингу активності програм.

Предмет дослідження – Windows API, Qt, Node.js та PostgreSQL.

Структурно робота складається з трьох розділів.

Перший розділ містить основні відомості про процес та спосіб отримання часу роботи за допомогою API операційних систем, також будуть розглянуті альтернативи.

У другому розділі описуються технології, які можуть бути використані для реалізації застосунку, діаграму варіантів використання, ER-діаграму, діаграму класів, послідовності та макет вебзастосунку.

Третій розділ містить особливості реалізації кожної частини застосунку – стеження за процесами, вебсерверу з базою даних та вебзастосунку. Виконаний опис застосунку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття моніторингу та його види

Моніторинг – це система постійного спостереження, аналізу та оцінювання будь-якого процесу для прийняття певних рішень [1]. Моніторинг широко застосовується у багатьох галузях людської діяльності, таких як екологія, охорона здоров’я, освіта, промисловість, фінанси, безпека тощо. У кожній із цих сфер моніторинг допомагає вчасно виявити проблеми, оптимізувати процеси й підвищити ефективність управління.

У цьому проєкті розглядатиметься системний моніторинг – це вид моніторингу, який відслідковує роботу різних частих операційної системи, таких як: продуктивність, мережа, програм, тощо [2]. Системний моніторинг дозволяє виявляти аномалії, падіння продуктивності, несанкціоновану активність або просто аналізувати поведінку користувача для подальшого аналізу.

Основний акцент проєкту буде зроблено на моніторингу прикладного програмного забезпечення, тобто процесів, які відповідають за роботу програм користувача, а також моніторингу екрана – періодичному знаттю скриншотів з метою фіксації активності. Таке поєднання дозволяє отримати повнішу картину використання комп’ютера: які програми були активні, коли вони працювали, та що саме відображалось на екрані.

Програма – це виконуваний файл, який містить певний набір інструкцій, написаних для виконання конкретного завдання чи операції на вашому комп’ютері [3].

Процес – це сутність, яка виконує мету програми. Кілька процесів можуть бути пов’язані з однією програмою [3].

Скриншот (англ. screenshot) – це знімок екрану; картинка, яка точно відображає те, що показує монітор в конкретний момент часу [4].

1.2 Отримання необхідних даних з системи

Для забезпечення повноцінного моніторингу прикладного програмного забезпечення система має отримувати актуальні та достовірні дані про всі активні процеси, які виконуються на комп’ютері користувача. У більшості сучасних операційних систем кожен процес, що створюється та виконується, має блок керування процесом (англ. Process Control Block, PCB), в якому зберігається вся технічна інформація, необхідній операційній системі, проте для проєкту необхідна лише його частина [5]:

- ідентифікатор процесу (англ. Process Identifier, PID) – унікальний номер процесу, який дозволяє операційній системі розрізняти різні процеси;
- час створення процесу – значення, яке вказує, коли процес був створений;
- шлях до виконуваного файлу – повна адреса, звідки було запущено програму.

Для отримання цієї інформації можна скористатися засобами операційної системи, наприклад, інтерфейсом програмування застосунків (англ. Application Programming Interface, API). Для Windows існує WinAPI (Windows API), для UNIX-подібних систем – POSIX (Portable Operating System Interface for UNIX).

1.3 Огляд альтернативних застосунків

Перед початком розробки системи слід провести аналіз існуючих популярних систем моніторингу. Це дозволяє краще зрозуміти їх основне призначення, доступний функціонал, особливості інтерфейсу, а також оцінити їхні переваги та недоліки. Такий аналіз допомагає виявити сильні сторони конкурентів, які можна врахувати при розробці, а також уникнути типових помилок або обмежень. Одним із головних недоліків більшості подібних

програм є складність у використанні, що проявляється у високому порозі входження для нових користувачів, заплутаному інтерфейсі або надмірній кількості технічних налаштувань, що ускладнюють їх впровадження у повсякденну роботу.

1.3.1 Steam

Steam – це онлайн платформа для цифрового поширення відеоігор [6]. Містить вбудований функцію відстеження часу в іграх, які є у магазині.

Переваги: відображення історії активності.

Недоліки:

– обмежений застосунками в магазині;

– відсутня детальна інформація.

1.3.2 ActivityWatch

ActivityWatch – це open source застосунок, який відслідковує, як користувач витрачає час [7].

Переваги:

– відкритий вихідний код.

– кросплатформеність;

Недоліки: складний інтерфейс.

1.3.3 Kickidler

Kickidler – це набір застосунків для моніторингу роботу співпрацівників [8].

Переваги:

– широкий набір функцій;

– безпека даних.

Недоліки: ціна.

Аналіз популярних систем моніторингу показує, що хоча кожна з них має свої переваги, жодна повністю не задовольняє потреби в гнучкому, простому та розширюваному рішенні для збору й аналізу даних активності користувача, тому було вирішено створити власний застосунок.

1.4 Засоби для створення системи

Розробка системи моніторингу вимагає використання великого набору інструментів, зокрема:

– бібліотеки для доступу до системної інформації;

– бази даних для збереження інформації;

– інструменти для передачі та візуалізації даних з серверу.

Існують безліч бібліотек для отримання процесів, зокрема Boost.Process, проте у цьому проєкті розглядатиметься написання власної реалізації. Для реалізації отримання скриншотів використовуватиметься вбудований модуль в Qt.

1.4.1 Огляд фреймворку Qt

Фреймворк (англ. Framework) – це зазвичай велика сукупність класів, які можуть мати загальне або спеціалізоване призначення залежно від сфери використання [9, с. 178]. Основна мета фреймворку – спростити процес проєктування та розробки програмного забезпечення шляхом надання готової архітектурної структури. Часто фреймворки застосовуються під час створення графічних інтерфейсів або складних багатокомпонентних застосунків.

Qt – це кросплатформений фреймворк та набір інструментів для створення графічного інтерфейсу користувача (англ. Graphical User Interface, GUI), який широко застосовується в розробці програмного забезпечення [9, с. 179].

Основні переваги Qt:

– кростплатформеність;
– висока продуктивність завдяки використанню C++;
– велика кількість вбудованих модулів;
– зручна модель подій (сигнали й слоти);
– активна спільнота та хороша документація.

1.4.2 Огляд СУБД

Система управління базами даних (СУБД) – це програмне забезпечення, призначене для створення та маніпуляцією базами даних [10]. СУБД надає зручний та ефективний інтерфейс для роботи з великими обсягами інформації, забезпечує захист та цілісність даних.

Під час розробки системи буде використано дві різні СУБД: SQLite та PostgreSQL.

SQLite – це СУБД без окремої серверної частини, яка записує безпосередньо до звичайних дискових файлів [11]. Вона ідеально підходить для використання на стороні клієнта завдяки своїй легкості, мінімальним вимогам до ресурсів та простоті використання. Використовуватиметься у клієнтському застосунку для тестування зберігання зібраних даних.

PostgreSQL – це СУБД з відкритим вихідним кодом, який використовує розширену версію мови SQL з більш безпечним та масштабованим функціоналом [12]. Вона була обрана як основна серверна СУБД для зберігання зібраних клієнтами даних у сервері.

1.4.3 Огляд платформи Node.js

Node.js – це середовище виконання JavaScript на стороні сервера [13]. Завдяки асинхронній обробці подій і неблокуючому введенню/виведенню, Node.js є ефективним рішенням для створення високопродуктивних мережевих застосунків. Node.js особливо популярний у сфері веброзробки, де він використовується для створення API, реального часу сервісів, систем мікросервісів, і навіть повноцінних бекенд-систем.

Основні переваги Node.js:

– один і той самий JavaScript на фронтенді та бекенді, що довзволяє використовувати спільний стек технології, наприклад, для Express.js та Svelte;
– швидкий запуск і легка масштабованість;
– велика екосистема пакетів;
– активна спільнота.

Недоліки:

– однопоточність;
– залежність від якості сторонніх пакетів.

1.4.4 Огляд фреймворку Express.js

Вебсервер – це програмне забезпечення, яке обробляє вхідні запити від клієнтів через мережу за протоколом HTTP(S) і відповідає на них, зазвичай передаючи вебсторінки, дані або інші ресурси [14].

У цьому проєкті вебсервер виконує роль посередником між клієнтським застосунком і базою даних PostgreSQL. Він отримує інформацію про активність застосунків, скриншоти або інші дані, що надходять від системи моніторингу, опрацьовує її та зберігає в базі.

Для передачі даних на сервер з базою даних буде створено вебсервер з використанням Express.js – це мінімалістичний та гнучкий фреймворк для

вебзастосунків, побудованих на Node.js [15]. Express.js дозволяє швидко будувати RESTful API – це архітектурний стиль, який широко використовується для організації взаємодії між клієнтами і сервером через HTTP-запити [16].

Основні переваги Express.js:

– простота інтеграції з Node.js та базами даних;
– можливість масштабування та додавання нових маршрутів без зміни основної архітектури;
– активна спільнота.

1.4.5 Огляд фреймворку Svelte

Svelte – це фреймворк для створення вебсайтів. На відміну від React та їм подібним, які використовують віртуальний DOM для відстеження змін стану, Svelte компілює свій код у JavaScript код, який безпосередньо оновлює DOM [17]. Завдяки легкій інтеграції з HTTP API вебсерверу (за допомогою fetch), Svelte забезпечує зручний механізм обміну інформацією з бекендом.

Однією з ключових особливостей Svelte є простота синтаксису – компоненти записуються у вигляді HTML-подібної структури з вбудованими стилями та скриптами. Завдяки цьому структура програми є прозорою та легкою для розуміння. Замість складних інструментів керування станом і рендерингом, Svelte використовує реактивні змінні, які автоматично оновлюють інтерфейс при зміні значень.

Основні переваги Svelte:

– мала вага кінцевого бандлу;
– простий синтаксис;
– зручна інтеграція з бекендом;
– гарна та ізольована інтеграція з CSS.

2 РОЗРОБКА ПРОЄКТУ ЗАСТОСУНКУ

2.1 Технічне завдання

Перед початком створення системи моніторингу активності користувачів необхідно сформувати технічне завдання, вимоги та етапи його розробки.

2.1.1 Найменування і область застосування

Програмний продукт, що розробляється, отримує найменування: «Система моніторингу активності користувачів».

Система призначена для використання в навчальних закладах, офісах, ІТ-компаніях, а також в інших середовищах, де необхідно контролювати продуктивність та активність користувачів за комп’ютером. Основною метою є збирання даних про те, які програми були використані, у який час, а також фіксація активності на екрані за допомогою знімків (скриншотів).

2.1.2 Призначення розробки

Даний проєкт призначений для вирішення наступних завдань:

– автоматичне відстеження активності користувача за комп’ютером (виявлення активного вікна, запущеного процесу, сфокусованої програми тощо);
– періодичне створення знімків екрана для візуального підтвердження дій користувача;
– передача даних на віддалений вебсервер за допомогою HTTP-запитів для централізованого зберігання;

- побудова зручного вебінтерфейсу для перегляду, аналізу та фільтрації інформації;
- забезпечення мінімального навантаження на ресурси комп’ютера під час фонової роботи застосунку;
- кросплатформенна сумісність для клієнтського застосунку (підтримка Windows, Linux);
- підтримка багатокористувацької архітектури з можливістю авторизації та ролей (адміністратор, звичайний користувач).

2.1.3 Технічні вимоги до програмного продукту

На початку роботи над системою були сформульовані наступні технічні вимоги:

- реалізувати збір даних про активні вікна та запущені процеси на комп’ютері користувача;
- реалізувати автоматичне знімання екрана з періодичністю;
- забезпечити збереження зібраних даних у локальну базу даних SQLite з можливістю додавання, оновлення та видалення записів;
- реалізувати клієнтську програму з використанням Qt, яка працює у фоновому режимі та має мінімалістичний інтерфейс налаштувань;
- реалізувати вебсервер на основі Express.js для прийому даних з клієнтів і запису їх у базу PostgreSQL;
- реалізувати вебінтерфейс з авторизацією, відображенням зібраної статистики, фільтрацією та переглядом скриншотів;
- реалізувати механізм синхронізації з вебсервером шляхом надсилання HTTP-запитів через API;
- забезпечити кросплатформенність клієнтського застосунку для роботи в операційних системах Windows та Linux;
- реалізувати зручну систему конфігурації для налаштування інтервалів моніторингу, шляхів збереження тощо;

– реалізувати механізм обробки помилок (наприклад, недоступність інтернету або сервера) з подальшими спробами повторної відправки.

2.2 Етапи створення застосунку

Процес створення програмного застосунку було вирішено розділити на декілька етапів:

– визначення цілей створення проєкту, складання технічного завдання та вимог до системи;
– вибір технологій для реалізації;
– проєктування бази даних;
– створення макетів сторінок вебзастосунку;
– реалізація системи;
– тестування усіх компонентів системи.

Визначення цілей та технічного завдання є ключовим етапом у процесі створення програмного забезпечення. У технічному завданні описується функціонал системи, вимоги до його розробки, архітектуру тощо. Технічне завдання допомагає розробникам чітко зрозуміти очікування замовника і розробити рішення, яке повністю відповідає поставленим цілям.

Наступним етапом є вибір технологій для реалізації програмного продукту. Необхідно визначитися з фреймворками, СУБД, мовами програмування, бібліотеками та іншими технологіями, які будуть використані під час розробки. Вибір технологій має суттєвий вплив на якість і ефективність кінцевого продукту. При прийнятті рішення важливо враховувати функціональні потреби проєкту, сумісність із цільовими платформами, можливість масштабування, продуктивність, а також вартість.

Проєктування бази даних дозволяє визначити, яку саме інформацію буде зберігати система, як вона буде структурована та яким чином взаємодіятиме між собою. Це сприяє раціональному та ефективному зберіганню даних.

Проєктування також є ключовим для забезпечення масштабованості: правильна структура даних дозволяє уникнути труднощів із доступом або збереженням інформації в умовах її зростання.

Створення макетів є важливою частиною для розробки вебзастосунку. Макети дають змогу сформувати загальне уявлення про структуру інтерфейсу, розташування основних елементів та передбачувану функціональність системи ще до початку її реалізації. Прототипування дозволяє заздалегідь оцінити зручність і логіку взаємодії користувача з додатком, виявити потенційні труднощі у використанні та покращити UX-дизайн.

Далі йде безпосередньо етап розробки. Розробка поділена на три основних компонента: клієнтський додаток, вебсервер та вебсайт. Клієнтський додаток працює на стороні користувача і відповідає за збір інформації про використання застосунків та активні вікна. Ці дані надсилаються на сервер через HTTP API. Вебсервер приймає вхідні запити від клієнтського додатка, обробляє отримані дані та зберігає їх у базі даних. Крім того, сервер надає API-інтерфейс для клієнтів, які хочуть отримати збережену інформацію. Третій компонент, вебсайт, слугує користувацьким інтерфейсом для перегляду результатів моніторингу. Він дозволяє відображати зібрані дані у зручному вигляді. Усі три компоненти функціонують як частини єдиної системи, що забезпечує збір, збереження та візуалізацію інформації.

У процесі розробки, а також після завершення реалізації системи, важливо провести повноцінне тестування для перевірки її коректної роботи. Тестування дозволяє виявити помилки, технічні несправності, проблеми з продуктивністю та сумісністю. Воно охоплює як функціональні перевірки, наприклад, чи відповідає система заданим вимогам, так і нефункціональні аспекти, зокрема зручність використання, надійність та безпеку. Завдяки тестуванню можна своєчасно усунути недоліки та підвищити стабільність роботи.

2.3 Діаграма прецедентів

В уніфікованій мові моделювання (англ. Unified Modeling Language, UML) діаграма прецедентів відображає взаємодію між акторами та системою [18]. Вона слугує для визначення функціональності, яка необхідна конкретним акторам. Опис акторів та відповідних прецедентів представлено в таблиці 2.1, а графічне представлення діаграми прецедентів наведено на рисунку 2.1.



Рисунок 2.1 – Діаграма прецедентів системи

Таблиця 2.1 – Актори та їхні прецеденти

Актор	Прецедент
Користувач	– Авторизуватись у системі; – подивитися свою статистику активності застосунків; – фільтрувати за вказаним періодом.

Продовження таблиці 2.1

Актор	Прецедент
Адміністратор	– Авторизуватись у системі; – переглядати користувачів та їхні активності; – фільтрувати за вказаним періодом; – ігнорувати певний файл/папку; – реєструвати нових користувачів.

2.4 ER-діаграма бази даних

Модель «сутність-зв’язок» або ER-діаграма – це різновид блок-схеми, яка показує взаємозв’язки між різними сутностями, такими як люди, об’єкти тощо [19]. Її часто застосовують для проєктування реляційних баз даних, як у нашому випадку.

База даних розроблюваної системи представлена на рисунку 2.2.

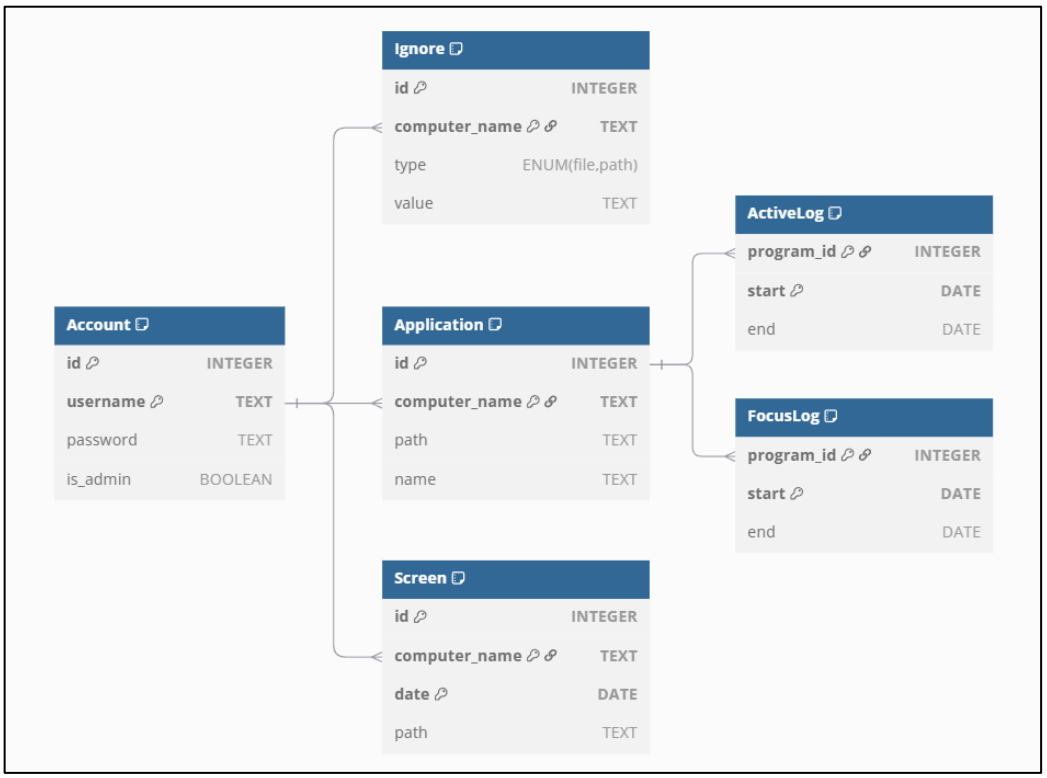


Рисунок 2.2 – ER-діаграма

База даних системи складатиметься з шістьох таблиць:

– Account – таблиця зі всіма акаунтами користувачів системи;
– Ignores – таблиця з файлами та папками, які ігноруються при спробі їх внесення до бази даних;
– Application – таблиця зі всіма застосунками, які були зафіксовані у користувачів;
– Screen – таблиця зі всіма скриншота користувачів;
– ActiveLog та FocusLog – таблиці зі записами активності кожного застосунку (активного та сфокусованого відповідно).

2.5 Діаграма класів

Діаграма класів є різновидом структурних діаграм, яка демонструє структуру системи [20]. Дуже популярна серед інженерів-програмістів для документації архітектури програмного забезпечення. Буде наведена діаграма для клієнтського додатку.

Діаграма складається з трьох ключових класів (див. рис. 2.3):

– database – база даних зі всіма записами про час роботи застосунків;
– monitoring – клас для автоматичного оновлення інформації о застосунках та відправки їх до серверу;
– window – основне вікно, яке відповідає за управління базою даних та моніторингом.

Класи record, window та database реалізують концепцію «Модель-Вигляд-Контролер» (англ. Movel-View-Controller, MVC) – це патерн, який підкреслює поділ між даними, інтерфейсом та контролером [21]. Зазвичай використовується для реалізації користувацьких інтерфейсів.

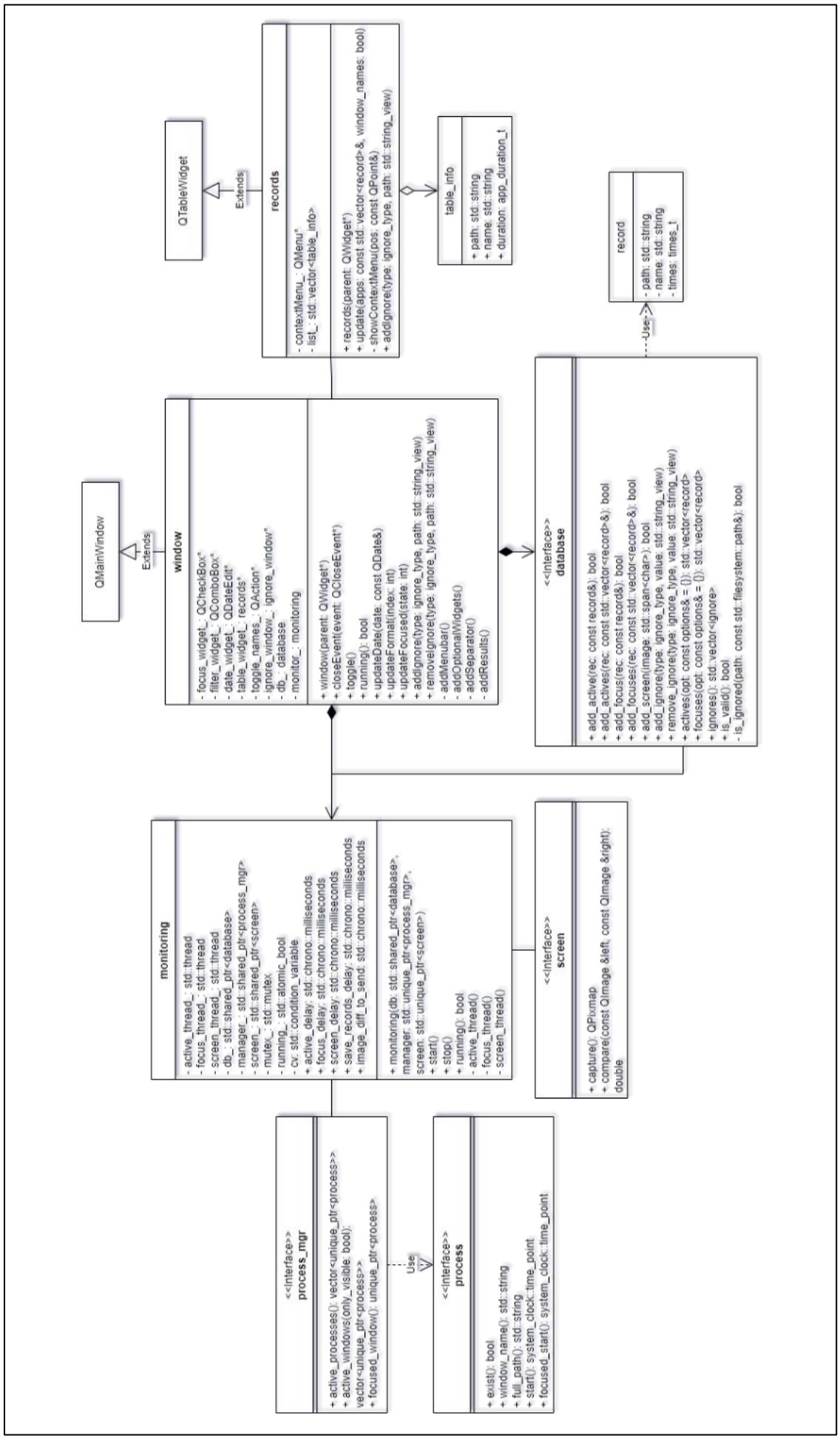


Рисунок 2.3 – Діаграма класів клієнтського додатку

2.6 Діаграма послідовності

Діаграма послідовності ілюструє, як об’єкти взаємодіють у певній послідовності [22]. Однією з найважливіших функцій у системі є передача даних до серверу (див. рис. 2.4). У послідовності беруть участь моніторинг, менеджер процесів та база даних.



Рисунок 2.4 – Діаграма послідовності передачі даних до бази даних

2.7 Проєктування інтерфейсу

На основі технічного завдання був спроєктований інтерфейс системи для вебзастосунок.

На рисунку 2.5 наведений макет головної сторінки. У верхній частині сторінки розташовано заголовок сайту з кнопками, які забезпечують швидкий перехід на головну сторінку та вихід із облікового запису. Нижче розміщено список усіх комп’ютерів, доступних користувачу, а також поле для пошуку конкретного пристрою.

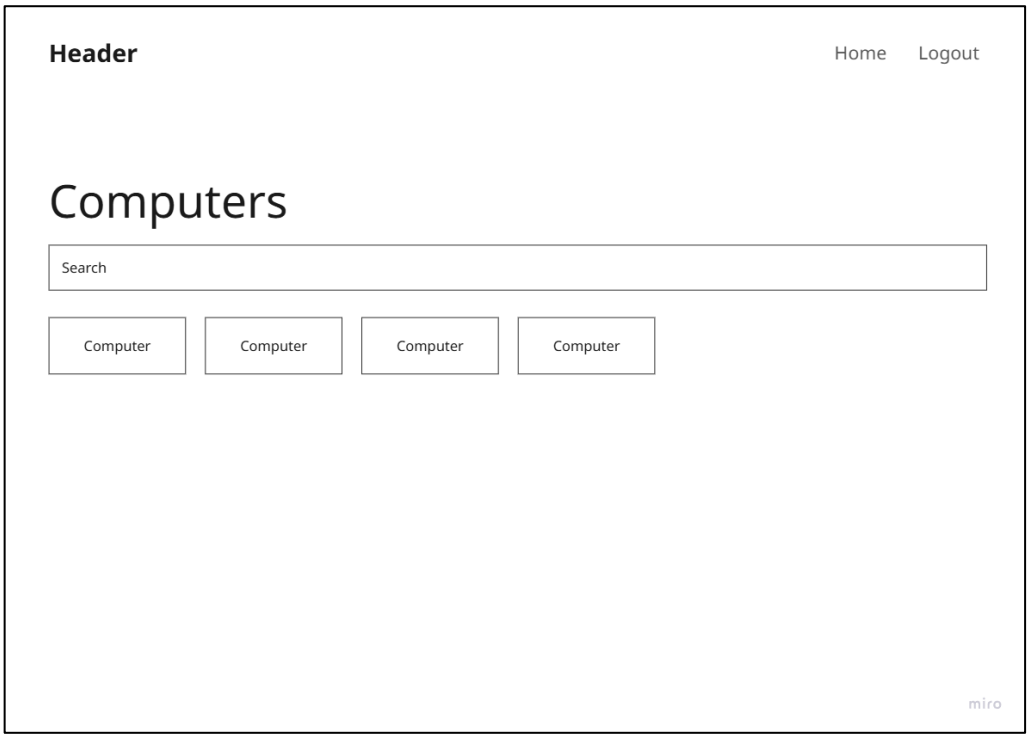


Рисунок 2.5 – Макет головної сторінки

Для авторизації та реєстрації була зроблена проста форма з вводом ім’я та паролю (див. рис. 2.6).

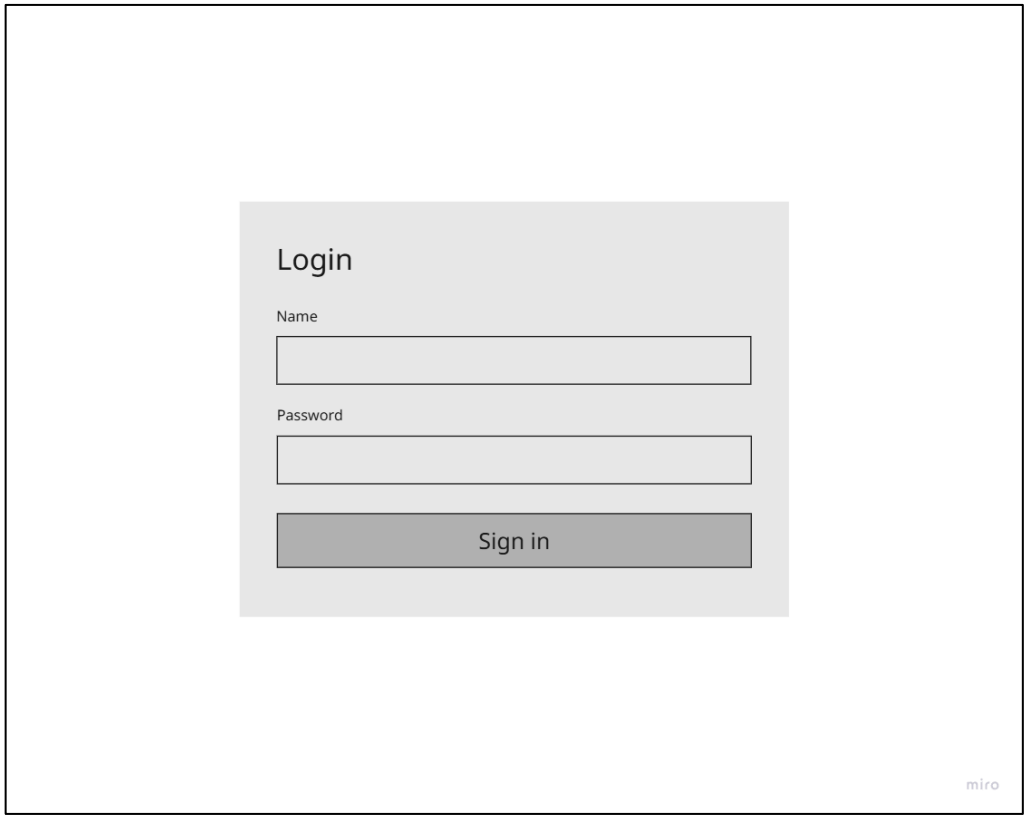


Рисунок 2.6 – Макет авторизації

Далі йде сторінка з детальною інформацією про активність користувача (див. рис. 2.7). Вона призначена для аналізу поведінки користувача за допомогою зібраних даних.

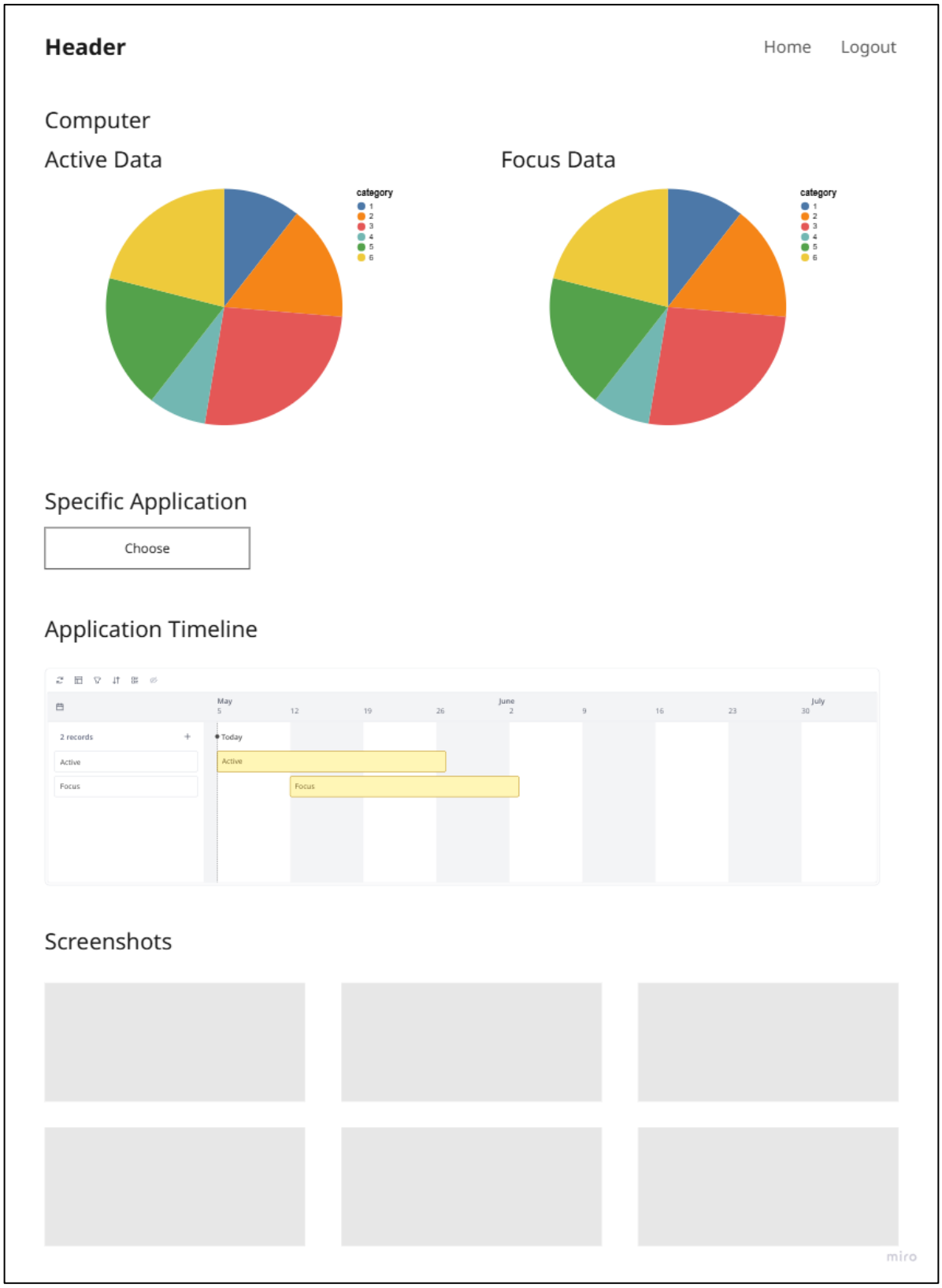


Рисунок 2.7 – Макет інформації користувача

Основні елементи сторінки:

– діаграма активності процесів – відображає відсотковий розподіл часу, коли процес був робочим під час роботи застосунку в обраному діапазоні дат;
– діаграма фокусування вікон – показує, які вікна були у фокусі та як довго, тобто були на передньому плані;
– таймлайн застосунку – графічне представлення періодів активності та фокусу обраного застосунку;
– скриншоти – відображаються зображення у хронологічному порядку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

На етапі проєктування програмного застосунку було вирішено розділити розробку на декілька етапів:

– створення застосунку на стороні користувача для отримання даних про роботу застосунків;
– створення бази даних та вебсерверу для відправки даних;
– створення вебсайту для перегляду результатів моніторингу користувачів;
– поєднання клієнтського застосунку з вебсервером через HTTP API;
– тестування усіх компонентів системи.

Під час розробки буде використовуватися система контролю версій Git.

3.1 Реалізація клієнтського застосунку

Клієнтський застосунок є основною моніторингової системи. Його основне завдання – збір, обробка та передача інформації про використання комп’ютера користувачем.

Розробка застосунку можна поділити на п’ять задач:

– реалізація модулю для роботи з процесом, що дозволяє отримувати назву вікна, ім’я процесу, повний шлях до виконуваного файлу, а також перевіряти наявність процесу у системі;
– реалізація модулю для отримання знімку екрану, який здійснює регулярне збереження скриншотів екрана, визначає їхню подібність та передає їх для подальшого аналізу;
– розробка системи моніторингу, що отримує активні й сфокусовані вікна, фіксує час фокусування, збирає дані кожні кілька секунд і готує їх для збереження або передачі;

– створення тимчасової бази даних на SQLite, яка використовується в цілях тестування;
– створення графічного інтерфейсу, що дозволяє перевірити правильність роботи моніторингу локально без потреби підключення до серверу.

3.1.1 Розробка модулю процесу

Клас процесу є обгорткою над функціями WinAPI та POSIX, щоб було зручніше використовувати для графічного інтерфейсу та запитів для відправки даних. Для застосунку необхідні методи:

– існування процесу для уникнення запису некоректних значень у базу даних – це можливо виконати, використовуючи функцію OpenProcess з правами доступу PROCESS_QUERY_INFORMATION на WinAPI або за допомогою функції kill з сигналом 0 для перевірки валідності процесу на POSIX;
– отримання ім’я вікна, повного шляху до виконуваного файлу та час початку роботи застосунку для подальшого внесення їх у базу даних – це можливо отримати, використовуючи функції SendMessageTimeout з параметром WM_GETTEXT, GetModuleFileNameEx та GetProcessTimes відповідно у Windows або за допомогою функцій readlink та openproc на UNIX-подібних системах.

Також потрібно реалізувати функції для отримання усіх активних процесів, вікон для внесення їх у базу даних. Активні процеси можливо отримати через WinAPI за допомогою функцій CreateToolhelp32Snapshot, яка робить знімок усіх процесів, та Process32First/Process32Next, які зчитують інформацію зі знімку або через POSIX за допомогою бібліотеки procps, яка зчитує необхідну інформацію з папки /proc/. У Windows активні вікна можливо

отримати, використовуючи функцію EnumWindows, яка перелічує усі вікна у задану функцію.

Слід зауважити, що WinAPI не надає готової функції для визначення часу початку фокусування вікна, тому це потрібно реалізовувати самостійно. Ідея полягає в тому, що якщо під час наступного виклику виявлено зміну активного вікна, час виклику функції буде записаний як початок фокусування вікна. На рисунку 3.1 наведено код для отримання сфокусованого вікна.

```
process_system_mgr::process_type process_system_mgr::focused_window() {
    HWND hwnd = GetForegroundWindow();
    if (!hwnd) {
        return std::make_unique<process_system>(-1);
    }
    static process_system last_focused{-1};
    const int pid = get_pid(hwnd);
    if (pid != last_focused.process_id_) {
        last_focused = process_system{pid};
        last_focused.focused_start_ = std::chrono::system_clock::now();
    }
    return std::make_unique<process_system>(last_focused);
}
```

Рисунок 3.1 – Код отримання сфокусованого вікна

3.1.2 Розробка модулю знімку

Для реалізації отримання скриншотів було використано фреймворк Qt, а саме функція QGuiApplication::screens, яка отримує усі зображення екрана користувача, та клас QScreen, який надає інструменти для роботи з ними.

Для зменшення обсягу збережених даних і підвищення ефективності аналізу, було реалізовано додаткову функцію порівняння скриншотів. Кожен новий скриншот порівнюється з попереднім, і на основі аналізу відмінностей

між пікселями обчислюється відсоток зміни зображення. Це дозволяє визначити, чи відбулися суттєві зміни на екрані, наприклад, переключення між вікнами, активність у програмі, або ж тривала бездіяльність. Тому була реалізована функція для зрівняння кожного пікселю зображення (див. рис. 3.2).

```
// calculate the average difference in color values
const double pixel_diff_red   = compare_color(pixel1.red(), pixel2.red());
const double pixel_diff_green = compare_color(pixel1.green(), pixel2.green());
const double pixel_diff_blue  = compare_color(pixel1.blue(), pixel2.blue());
value += (pixel_diff_red + pixel_diff_green + pixel_diff_blue) / color_number;
```

Рисунок 3.2 – Код порівняння двох пікселів

3.1.3 Розробка тимчасової бази даних

Перед розробкою вебсерверу для відправки даних було вирішено створити тимчасову базу даних на локальному пристрої для проєктування та тестування інтерфейсу. Було вирішено використати базу даних SQLite.

Для збереження скриншотів було створено відокремлену папку screens, у якій зберігаються всі зображення екрана, зроблені користувачем під час роботи застосунку. Імена файлів генеруються автоматично на основі дати й часу знімка, що забезпечує унікальність кожного зображення та полегшує подальший пошук і обробку.

Модуль database був спроєктований таким чином, щоб його легко було замінити на макет (mock) для автоматизованих тестів або на HTTP API-обгортку для відправлення даних на сервер. Такий підхід реалізує принцип інверсії залежностей, що дозволяє відокремити логіку збору даних від способу їх зберігання чи передавання. Це забезпечує гнучкість і розширюваність системи.

3.1.4 Розробка системи моніторингу

Цей клас об’єднує реалізації модулів процесу, скриншотів та бази даних в єдиний керуючий компонент, який відповідає за координацію збору даних у режимі реального часу. Під час ініціалізації створюються три окремі потоки, що працюють паралельно для забезпечення незалежного та ефективного збору інформації.

Потоки моніторингу активних та сфокусованих вікон, які відповідають за виявлення, який із запущених процесів наразі активний або має фокус користувача. При цьому для кожного обробленого вікна визначається найстарший процес, щоб уникнути дублювання інформації. Зібрані дані (назва вікна, шлях до виконуваного файлу, час запуску, час фокусування) фіксуються у базі даних.

Потік скриншотів, який через заданий інтервал часу виконує збереження знімка активного екрана. Кожен новий знімок порівнюється з попереднім для визначення ступеня схожості (у відсотках). Це дозволяє оцінити, чи дійсно користувач взаємодіяв із вікном, чи залишив його без змін. Результати (зображення, час, різниця) також зберігаються у базі даних.

Для фіксування активних, сфокусованих процесів та скриншотів можна визначити частоту оновлення.

3.1.5 Розробка графічного інтерфейсу

У головному вікні графічного інтерфейсу відображається таблиця з записами часу роботи застосунків (див. рис. 3.3). Кожен рядок містить інформацію про назву застосунку та тривалість активності або фокусування.

Інтерфейс також передбачає налаштування фільтрації даних, таких як:

– вибір періоду часу для перегляду записів;
– перемикач для відображення лише сфокусованих вікон.

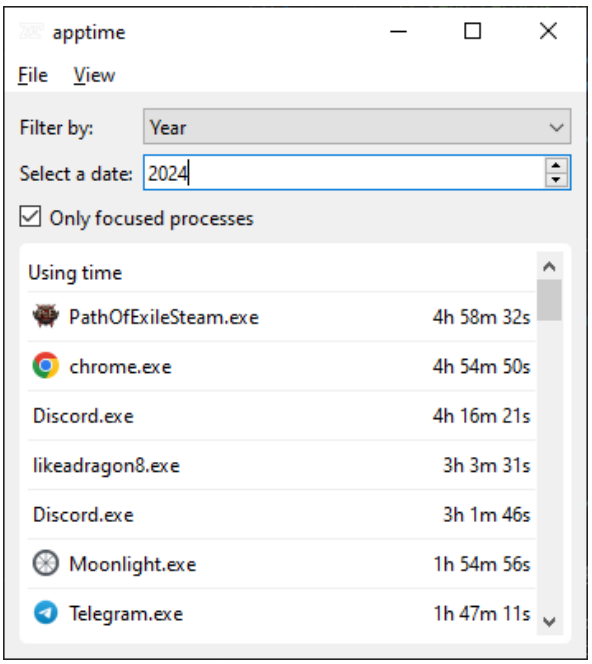


Рисунок 3.3 – Головне вікно застосунку

При виклику контекстного меню на записі користувач має доступ до додаткових функцій (див. рис. 3.4):

– копіювання повного шляху до виконуваного файлу;
– копіювання назви застосунку;
– додавання застосунку до списку ігнорування, після чого нові записи цього процесу не зберігатимуться до бази даних.

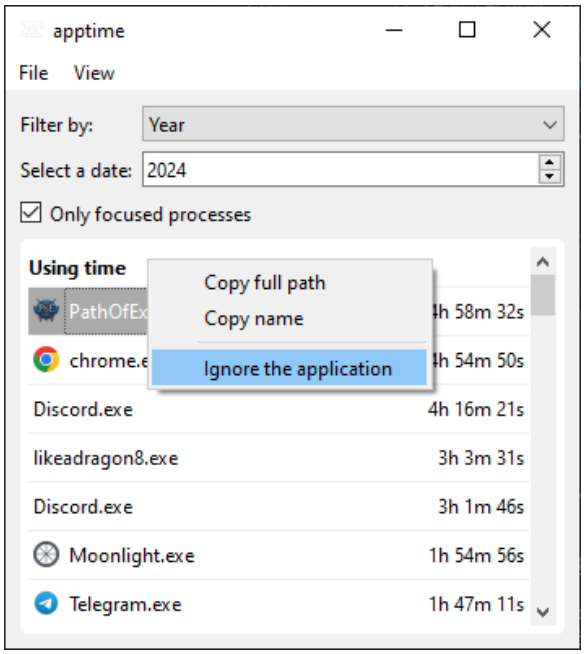


Рисунок 3.4 – Вікно застосунку з викликом контекстного меню

Також необхідно реалізувати налаштування ігнорованих файлів та папок, які не повинні відображатися в основній таблиці й не записуватимуться до бази даних надалі. Це дозволяє виключити системні процеси або особисті застосунки, що не потребують моніторингу. Користувач має можливість (див. рис. 3.5):

– додати файл або папку до списку;
– видалити елемент зі списку;
– переглядати список усіх ігнорованих файлів та папок.

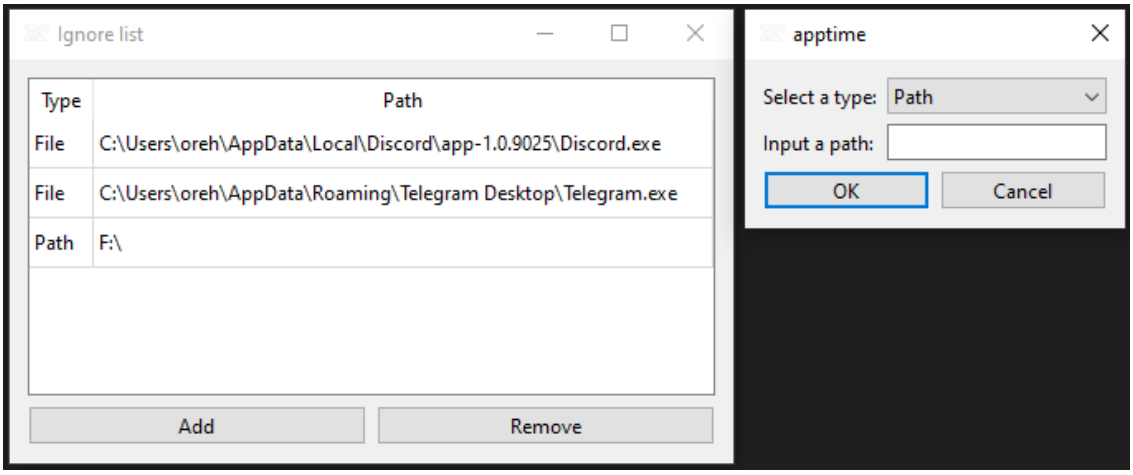


Рисунок 3.5 – Ігнорування файлів та папок

3.2 Реалізація вебсерверу

Вебсервер виконує роль проміжної ланки між клієнтським застосунком, який надсилає дані моніторингу, та вебзастосунком, який відображає цю інформацію користувачу. Основним ціллю сервера є обробка запитів, збереження отриманих даних у базу даних, а також надання інтерфейсів (API) для зчитування у зручному вигляді.

Для безпечного та гнучкого зберігання конфігураційних параметрів, наприклад, для даних для підключення до бази даних (назва БД, логін, пароль, хост, порт), використовується файл .env, який зчитується за допомогою бібліотеки dotenv. Це дозволяє легко змінювати параметри середовища без

необхідності редагування коду. Файл .env матиме такі дані, як вказано на рисунку 3.6.

```
DB_HOST=localhost
DB_PORT=5432
DB_NAME=apptime
DB_USER=postgres
DB_PASSWORD=mypassword
TOKEN_SECRET=mytoken
SCREEN_PATH=path_to_screens/
```

Рисунок 3.6 – Параметри для вебсерверу

3.2.1 Розробка модулю бази даних

База даних буде створена автоматично за допомогою бібліотеки Sequelize для зручної взаємодії з реляційною базою даних, зокрема з PostgreSQL. Моделі сутностей у базі визначаються у вигляді класів на мові програмування TypeScript, що дозволяє строго типізувати всі поля таблиць, встановлювати зв’язки між сутностями (One-to-Many, Many-to-Many), а також описувати додаткову логіку в межах класу. На рисунку 3.7 зображено код однієї з таблиць бази даних.

```
@Table({ tableName: 'active_logs', timestamps: false })
export class ActiveLog extends Model {
  @PrimaryKey
  @Column(DataType.INTEGER)
  @ForeignKey(() => Application)
  declare program_id: number;

  @PrimaryKey
  @Column(DataType.DATE)
```

```
    declare start: Date;

    @Column(DataType.DATE)
    declare end: Date;

    @BelongsTo(() => Application)
    declare application: Application;
}
```

Рисунок 3.7 – Визначення таблиці бази даних через Sequelize

3.2.2 Збереження зображень на сервері

Необхідно додати обробку скриншотів користувачів для збереження їх на серверу для подальшого використання у вебзастосунку.

Після отримання зображення, сервер автоматично виконує його масштабування до розміру: ширина не більше 800 пікселів та висота – 400 пікселів. Після масштабування зображення конвертується у формат JPEG, що є оптимальним за співвідношенням якості до розміру файлу. Оброблене зображення зберігається у відповідній директорії на сервері, а шлях до нього, разом з ідентифікатором користувача та міткою часу, фіксується в базі даних.

3.2.3 Розробка запитів

Щоб любий користувач не мав доступу до усіх даних, було додано перевірку на наявність JWT-токену при кожному запиті. Якщо користувач не має токен або він неправильний, сервер поверне помилку 403 (Forbidden), якщо термін токену минув, користувач отримає 401 (Unauthorized). Це можливо зробити за допомогою додаткового обробнику у Express.js перед виконанням основної функції, як зроблено на рисунку 3.8.

```
function authenticateToken(req: AuthenticatedRequest, res: Response, next: () => void) {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];
    if (!token) {
        return res.sendStatus(401);
    }
    jwt.verify(token, process.env.TOKEN_SECRET as string, (err: any, user: any) => {
        if (err instanceof jwt.TokenExpiredError) {
            return res.sendStatus(401);
        }
        if (err) {
            return res.sendStatus(403);
        }
        req.user = user;
        next();
    });
}
```

Рисунок 3.8 – Обробка JWT-токену перед обробкою запиту

JWT-токен містить основну інформацію про користувача, зокрема його ім’я та ознаку адміністративних прав. На основі цих даних сервер визначає рівень доступу:

– авторизований користувач: має доступ к перегляду лише своїх даних, може відправляти дані;
– адміністратор: має доступ к перегляду усіх даних, не може відправляти дані, може реєструвати користувачів для системи.

Для усіх користувачів буде доступний запит /login для авторизації у систему.

Розроблені наступні запити:

– POST/GET /active – відправка та отримання даних активних процесів;
– POST/GET /focus – відправка та отримання даних сфокусованих процесів;

– POST/GET /screen – відправка та отримання даних скриншотів;
– POST/DELETE /ignore – додавання та видалення процесів, які необхідно ігнорувати під час моніторингу;
– GET /computers – отримання усіх доступних користувачеві комп’ютери (для авторизованого користувача повертатиме лише його ім’я);
– POST /login – авторизація у систему;
– POST /register – реєстрація у систему (може лише адміністратор системи);
– GET /me – отримання даних про себе.

Для запитів на отримання даних додані параметри day, month, year, period, dayp, monthp, yearp, які забезпечують більш гнучке налаштування періоду часу під час аналізу даних. Існує 4 види періоду:

– якщо вказано параметр period, дані вибираються у проміжку з day/month/year по dayp/monthp/yearp включно;
– якщо задано лише year, буде обрано усі записи за відповідний рік;
– якщо задано month і year, повертаються дані за конкретний місяць обраного року;
– якщо вказано всі три параметри – day, month, year, то отримуються записи лише за конкретну дату.

3.3 Реалізація вебсайту

Для забезпечення зручної взаємодії користувачів зі системою був розроблений вебсайт, який дозволяє переглядати зібрану інформацію про активність застосунків.

Вебсайт реалізовано з використанням фреймворку Svelte, що дозволяє створювати користувацькі інтерфейси з високою швидкістю завантаження та

плавною взаємодією. Усі дані на сторінках завантажуються динамічно, без повного перезавантаження сторінки, за допомогою запитів до API вебсерверу.

Якщо користувач не авторизований у системі, він автоматично перенаправляється на сторінку входу (див. рис. 3.9).

```
function redirectResponse(redirect: boolean = true): Response {
    if (redirect) {
        goto("/login");
        return new Response(null, { status: 401 });
    }
    return new Response(null, { status: 403 });
}
export async function fetchWithToken(input: RequestInfo | URL, init?: RequestInit, loginRedirect: boolean = true): Promise<Response> {
    if (!localStorage.getItem('token')) {
        return redirectResponse(loginRedirect);
    }
    // ...
}
```

Рисунок 3.9 – Перенаправлення зі сторінки

Після успішної авторизації система перевіряє, чи має користувач доступ к перегляду декількох акаунтів. Якщо доступний лише один, користувач одразу перенаправляється на сторінку перегляду даних цього комп’ютера (див. рис. 3.10).

```
const response = await fetchWithToken(new URL("computers", API_URL));
computers = await response.json();
if (computers.length === 1) {
    handleClick(computers[0]);
}
```

Рисунок 3.10 – Автоматичне перенаправлення на сторінку перегляду

3.3.1 Сторінки входу та реєстрації

Під час введення даних здійснюється перевірка на заповнення обов’язкових полів на клієнтській стороні. Після натискання кнопки відправки форми виконується запит до сервера. У разі успішної авторизації користувач отримує JWT-токен і перенаправляється на головну сторінку з результатами моніторингу.

Якщо під час відправки форми виникає помилка, наприклад: неправильний пароль чи неіснуючий користувач – форма автоматично відображає повідомлення про помилку без перезавантаження сторінки (див. рис. 3.11).

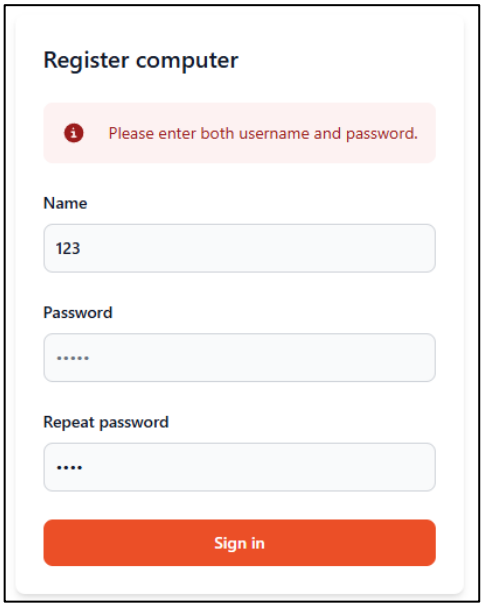


Рисунок 3.11 – Форма з помилкою

3.3.2 Сторінка користувача

На сторінці користувача реалізований функціонал для перегляду зібраної інформації користувача. Інтерфейс дозволяє обрати певний відрізок часу, після чого всі дані оновлюються відповідно до вказаного періоду (див. рис. 3.12).

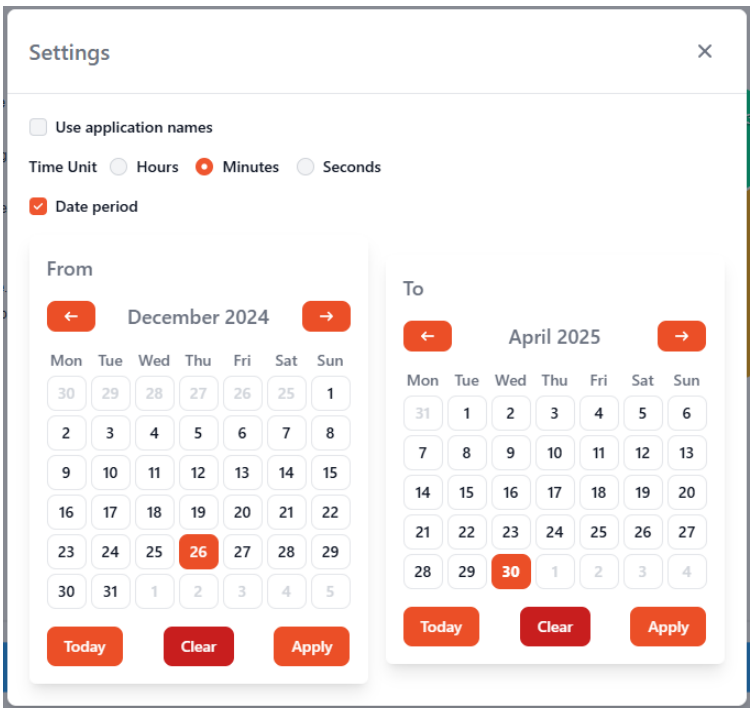


Рисунок 3.12 – Налаштування відображення даних

3.4 Інтеграція клієнтського додатку з вебсервером

Після реалізації окремих частин системи, таких як клієнтського застосунку для збору даних та вебсерверу для їх зберігання й обробки, було здійснено інтеграцію цих компонентів у єдину систему.

У клієнтському додатку створюємо клас бази даних, який є обгорткою над HTTP API запитами вебсерверу. Він замінює використання локальної бази даних і дозволяє прозоро надсилати запити до віддаленого серверу. Для реалізації запитів було обрано бібліотеку cpr, яка надає зручний інтерфейс для роботи з HTTP-протоколом.

Передача структурованих даних у форматі JSON здійснюється за допомогою бібліотеки nlohmann_json. Оскільки структура застосунку містить складні типи даних, було вирішено розширити функціональність стандартного механізму серіалізації цієї бібліотеки. Для цього було реалізовано додатковий функціонал, який дозволяє зручно перетворювати внутрішні структури застосунку у JSON-формат і навпаки (див. рис. 3.13).

```
template <>
struct adl_serializer<apptime::record> {
    static void to_json(json &json_obj, const apptime::record &rec) {
        std::vector<json_time_t> times;
        for (const auto &[start, end]: rec.times) {
            times.emplace_back(start, end);
        }
        json_obj = json{{"path", rec.path}, {"name", rec.name}, {"times", times}};
    }
    static void from_json(const json &json_obj, apptime::record &rec) {
        json_obj.at("path").get_to(rec.path);
        json_obj.at("name").get_to(rec.name);
        std::vector<json_time_t> times;
        json_obj.at("times").get_to(times);
        for (const auto &time: times) { rec.times.emplace_back(time.start, time.end); }
    }
};
```

Рисунок 3.13 – Серіалізація структури застосунку

Крім того, для налаштування параметрів взаємодії із сервером, таких як ім’я комп’ютера, адреса вебсерверу та пароль користувача, було створено окреме графічне вікно налаштувань (див. рис. 3.14).

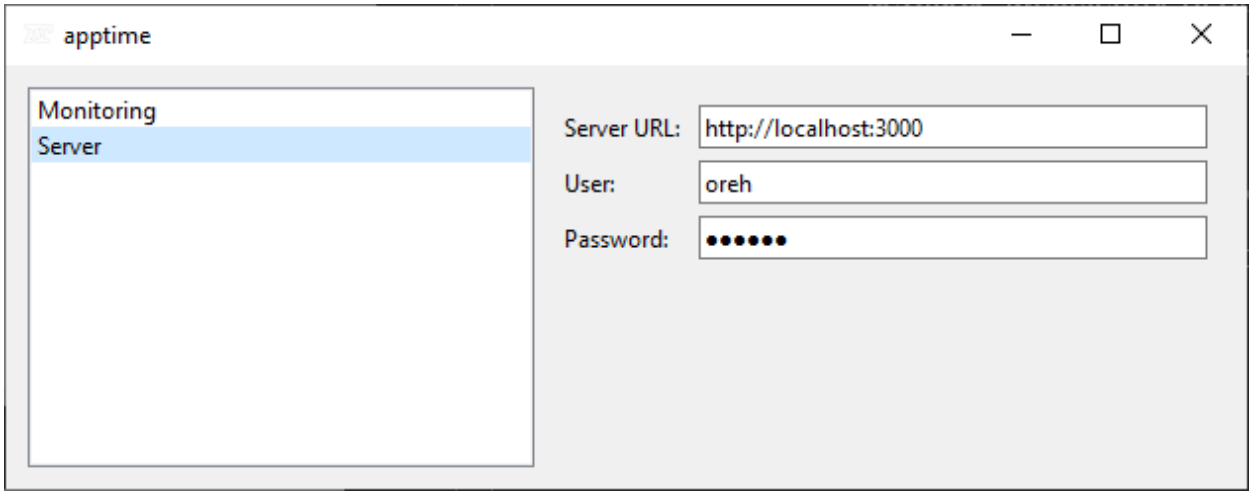


Рисунок 3.14 – Налаштування вебсерверу у застосунку

3.5 Тестування системи

Під час розробки систем проводилися тестування для перевірки коректності роботи всіх її частин. Тестування охоплювало як окремі модулі, так і інтегровану систему в цілому.

Для клієнтського застосунку було зроблено три автоматизовані тести.

Перший тест стосується модуля скриншотів, де перевіряється точність обчислення відсоткової схожості між двома зображеннями, що дозволяє виявляти зміни в активності користувача. Тест враховує випадки однакових знімків та знімків з незначними змінами.

Другий тест охоплює роботу з базою даних SQLite. Здійснюється перевірка коректності створення, збереження та вибірки записів. Тест покриває типові сценарії використання тимчасової бази, зокрема обробку повторних вставок, видалення та оновлення записів.

Останній тест зосереджений на модулі моніторингу. Перевіряється правильність роботи основного функціоналу з використанням макетованих об’єктів бази даних та менеджера процесів. Тест фіксує, чи правильно система обробляє дані про активні та сфокусовані вікна, а також коректне формування записів для подальшої відправки на сервер.

Тестування вебсерверу проводилось за допомогою інструменту Postman, який дозволяє зручно створювати та надсилати HTTP-запити до API. За його допомогою перевірялась правильність обробки запитів усіх реалізованих шляхів, таких як авторизація, реєстрація, надсилання даних, отримання статистики тощо.

Особлива увага приділялась відповідям сервера на некоректні запити, відсутність або недійсність JWT-токенів, а також правильному формуванню помилок 401 та 403. Було протестовано сценарії як для звичайного користувача, так і для адміністратора.

На рисунку 3.15 зображено тестування одного з запитів – авторизації.

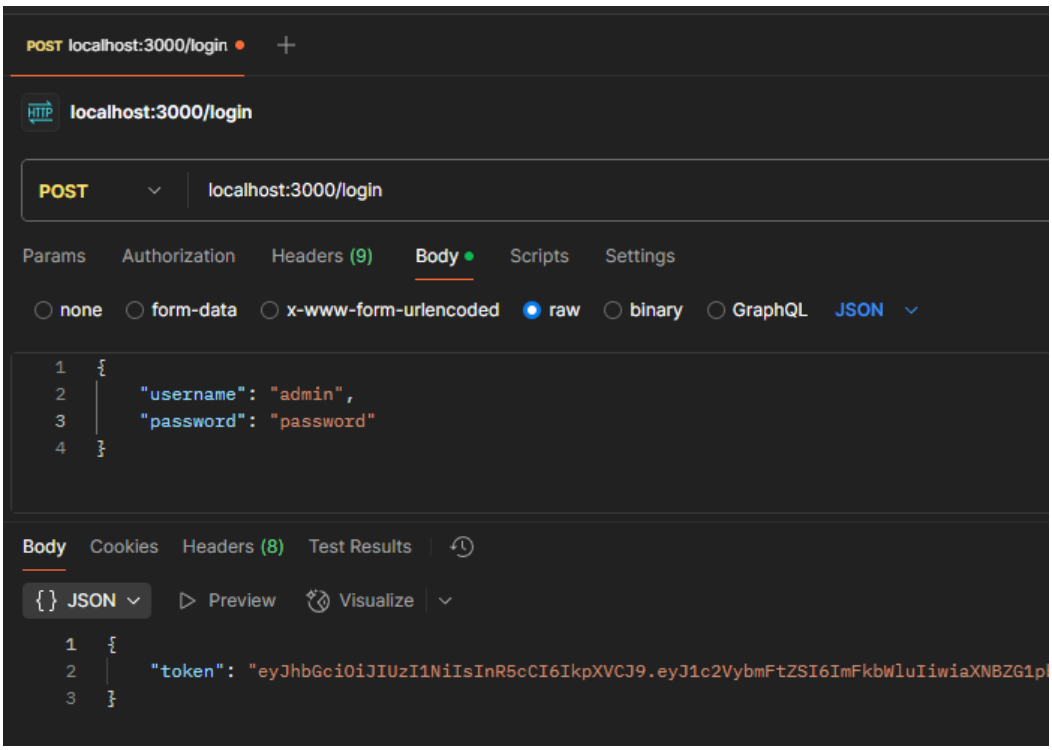


Рисунок 3.15 – Тестування авторизації в Postman

Наприкінці було проведено ручне тестування вебзастосунку, тобто його перевіряли без спеціальних автоматизованих інструментів. Тестування виконувалося вручну, імітуючи дії звичайного користувача. Це дозволяє перевірити, як система реагує на введення даних, зміну сторінок, взаємодію з елементами інтерфейсу, обробку запитів тощо. Зокрема, здійснюється перевірка коректності авторизації та виходу з акаунту, відображення списку комп’ютерів користувача, навігації між сторінками та правильності виведення даних на графіках. Особливу увагу приділяють стабільності роботи інтерфейсу в різних браузерах і при різних розширеннях екрана, а також перевірці відсутності критичних помилок при взаємодії з сервером.

ВИСНОВКИ

У кваліфікаційній роботі було досліджено процес розробки системи для відстеження часу роботи програм. Розглянуто основні підходи та інструменти, необхідні для створення подібного програмного забезпечення. У рамках проєктування були створені відповідні діаграми, а також реалізовано функціональність для моніторингу активності застосунків. Також було створено вебсервер для відправки даних на віддалений сервер та вебсайт для їхнього перегляду.

У ході роботи були виконані такі завдання:

– проаналізувати інструменти для розробки програмного забезпечення;
– спроєктувати структуру системи;
– реалізувати клієнтський додаток;
– розробити вебсервер та базу даних;
– створити вебсайт для перегляду статистики користувача;
– провести тестування системи.

Розроблений застосунок відповідає сформованому завданню та технічним вимогам.

ПЕРЕЛІК ПОСИЛАНЬ

1. Моніторинг. *Словник.ua*. URL: https://slovnyk.ua/index.php?swrd=моніторинг (дата звернення: 09.03.2025).

2. Wiesen G. What is a System Monitor? *EasyTechJunkie*. URL: https://www.easytechjunkie.com/what-is-a-system-monitor.htm (дата звернення: 09.03.2025).

3. Беннетт Л. Різниця між процесом і програмою (процес проти програми). *Guru99*. URL: https://www.guru99.com/uk/program-vs-process-difference.html (дата звернення: 09.03.2025).

4. Що таке скріншот (screenshot). *Hostpro*. URL: https://billing.hostpro.ua/index.php?rp=/knowledgebase/57/SHCHo-take-skrnshot-screenshot.html (дата звернення: 09.03.2025).

5. Process control block in Operating Systems. *PrepBytes Blog*. URL: https://www.prepbytes.com/blog/operating-system/process-control-block-in-operating-systems (дата звернення: 07.05.2025).

6. Steam, The Ultimate Online Game Platform. *Steam*. URL: https://store.steampowered.com/about/ (дата звернення: 10.03.2025).

7. ActivityWatch. *ActivityWatch – Open-source time tracker*. URL: https://activitywatch.net/ (дата звернення: 10.03.2025).

8. Employee Monitoring Software | Kickidler. *Kickidler*. URL: https://www.kickidler.com/ (дата звернення: 10.03.2025).

9. Ezurt A., Ezurt P. An introduction to design patterns in C++ with Qt 4. Upper Saddle River, NJ : Prentice Hall, 2007. 624 p.

10. Що таке система управління базами даних та як вона застосовується на практиці? *Наш Репортер*. URL: https://nashreporter.com/dovidnyk/shho-take-systema-upravlinnya-bazamy-danyh-ta-yak-vona-zastosovuyetsya-na-praktyczi-170773 (дата звернення: 10.03.2025).

11. About SQLite. *SQLite Home Page*. URL: https://www.sqlite.org/about.html (дата звернення: 04.05.2025).

12. PostgreSQL: About. *PostgreSQL: The world's most advanced open source database*. URL: https://www.postgresql.org/about/ (дата звернення: 10.03.2025).

13. Node.js – About Node.js®. *Node.js – Run JavaScript Everywhere*. URL: https://nodejs.org/en/about (дата звернення: 04.05.2025).

14. What is a web server? – Learn web development | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Howto/Web_mechanics/What_is_a_web_server (дата звернення: 04.05.2025).

15. Express – це фреймворк для веб-застосунків, побудованих на Node.js. *Express – Node.js web application framework*. URL: https://expressjs.com/uk/ (дата звернення: 14.03.2025).

16. What are REST APIs? HTTP API vs. REST API. *Educative*. URL: https://www.educative.io/blog/what-are-rest-apis (дата звернення: 04.05.2025).

17. Harris R. Svelte 3: Rethinking reactivity. *Svelte – Web development for the rest of us*. URL: https://svelte.dev/blog/svelte-3-rethinking-reactivity (дата звернення: 14.03.2025).

18. UML Use Case Diagram Tutorial. *Lucidchart*. URL: https://www.lucidchart.com/pages/uml-use-case-diagram (дата звернення: 26.04.2025).

19. What is an Entity Relationship Diagram (ERD)? *Lucidchart*. URL: https://www.lucidchart.com/pages/er-diagrams (дата звернення: 26.04.2025).

20. UML Class Diagram Tutorial. *Lucidchart*. URL: https://www.lucidchart.com/pages/uml-class-diagram (дата звернення: 26.04.2025).

21. MVC – MDN Web Docs Glossary: Definitions of Web-related terms | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC (дата звернення: 26.04.2025).

22. BasuMallick C. What Is a Sequence Diagram? Meaning, Elements, Examples, and Importance. *Spiceworks*. URL:

https://www.spiceworks.com/tech/devops/articles/sequence-diagram/ (дата звернення: 26.04.2025).