НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Мобільний застосунок для віддаленого моніторингу та керування серверами

Виконав студент IV курсу, групи ІТ-91
(шифр групи)

Вовченко Артем Євгенійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник асистент Сопов О. О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації ст.викл. Головченко М. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ, к.т.н., доц., Шимкович В. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Вовченку Артему Євгенійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Мобільний застосунок для віддаленого моніторингу та керування серверами

керівник проєкту Сопов Олексій Олександрович, асистент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31» травня 2023 р. №2101-с

2. Термін подання студентом проєкту « 17 » червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Загальні положення: основні визначення та терміни, опис предметної області, огляд ринку програмних продуктів, аналіз вимог, постановка задачі.
2) Моделювання, конструювання програмного забезпечення, аналіз роботи з даними.
3) Аналіз якості програмного забезпечення, опис процесів тестування
4) Розгортання та підтримка програмного забезпечення

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань_____

2) Схема бази даних_____

3) Креслення вигляду екранних форм_____

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «10» березня 2023 року_____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв’язання задачі	17.03.2023	
3	Постановка та формалізація задачі	24.03.2023	
4	Розробка інформаційного забезпечення	31.03.2023	
5	Алгоритмізація задачі	07.04.2023	
6	Обґрунтування вибору використаних технічних засобів	14.04.2023	
7	Розробка програмного забезпечення	28.04.2023	
8	Налагодження програми	05.05.2023	
9	Виконання графічних документів	12.05.2023	
10	Оформлення пояснювальної записки	26.05.2023	
11	Подання ДП на попередній захист	02.06.2023	
12	Подання ДП рецензенту	12.06.2023	
13	Подання ДП на основний захист	16.06.2023	

Студент _____ Артем ВОВЧЕНКО

(підпис) (ініціали, прізвище)

Керівник _____ Олексій СОПОВ

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 41 таблицю, 41 рисунок та 12 джерел – загалом 71 сторінка.

Дипломний проєкт присвячений розробці мобільного додатку для віддаленого моніторингу та керування серверами.

Мета: спрощення задач системних адміністраторів та DevOps інженерів при взаємодії з серверами.

У розділі аналізу вимог був здійснений огляд предметної області, дослідження існуючих рішень та підходів. Були висунуті функціональні та нефункціональні вимоги та поставлена задача.

Розділ моделювання програмного забезпечення присвячений детальному опису архітектури та реалізації програмного рішення.

У розділі аналізу якості та тестування програмного забезпечення описано процес аналізу розробки на наявність помилок, відповідність стандартам та вимогам.

У розділі впровадження та супроводу описано процес встановлення програми кінцевим користувачем.

КЛЮЧОВІ СЛОВА: МОНІТОРИНГ, СЕРВЕРНЕ АДМІНІСТРУВАННЯ, МОБІЛЬНИЙ ДОДАТОК, ANDROID, SSH, УПРАВЛІННЯ ФАЙЛАМИ, ВІДДАЛЕНИЙ ДОСТУП.

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 41 tables, 41 figures and 12 sources – in total 71 pages.

The diploma project is devoted to the development of a mobile application for remote monitoring and server administration.

Purpose: simplifying the tasks of interacting with servers for system administrators and DevOps engineers.

In the requirements analysis section, a review of the subject area, a study of existing solutions and approaches was carried out. Functional and non-functional requirements were put forward and the task was set.

The software modeling section is devoted to a detailed description of the architecture and implementation of the software solution.

The software quality analysis and testing section describes the process of analyzing the work for errors, compliance with standards and requirements.

The implementation and maintenance section describes the process of installing the application by the end user.

KEYWORDS: MONITORING, SERVER ADMINISTRATION, MOBILE APP, ANDROID, SSH, FILE MANAGEMENT, REMOTE ACCESS.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-9105.045490.01.91	Технічне завдання	30	
3	А4	КПІ.ІТ-9105.045490.02.81	Пояснювальна записка	71	
4	А4	КПІ.ІТ-9105.045490.03.12	Текст програми	46	
5	А4	КПІ.ІТ-9105.045490.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-9105.045490.05.34	Керівництво користувача	17	
7	А3	КПІ.ІТ-9105.045490.06.99	Схема структурна варіантів	1	
			використань		
8	А3	КПІ.ІТ-9105.045490.07.99	Схема бази даних	1	
9	А3	КПІ.ІТ-9105.045490.08.99	Креслення вигляду екранних форм	1	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-9105.045490.00.90		
Розроб.		Вовченко А. Є.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Сопов О. О.					
Н.контр.		Головченко М.М.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-91	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДДАЛЕНОГО МОНІТОРИНГУ ТА КЕРУВАННЯ СЕРВЕРАМИ

Технічне завдання

КПІ.ІТ-9105.045490.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
Олексій СОПОВ

Нормоконтроль:
Максим ГОЛОВЧЕНКО

Виконавець:
Артем ВОВЧЕНКО

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: мобільний застосунок для віддаленого моніторингу та керування серверами.

Галузь застосування:

Наведене технічне завдання поширюється на розробку мобільного застосунку для віддаленого моніторингу та керування серверів, котрий використовується для спостереження за серверами та управління ними та призначений для використання системними адміністраторами та DevOps інженерами в сучасних хмарних середовищах.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки мобільного застосунку для віддаленого моніторингу та керування серверами є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для спостереження за станом серверів та використанням ними ресурсів, а також віддаленого адмініструваня серверів.

Метою розробки є спрощення процесу моніторингу серверів та управління ними для системних адміністраторів та DevOps інженерів.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати реалізацію функцій, що наведений у наступних підрозділах.

4.1.1 Користувацького інтерфейсу

На головному екрані (рисунок 4.1) повинні відображатися всі сервери у вигляді списку, а також кнопка для додавання нового сервера.

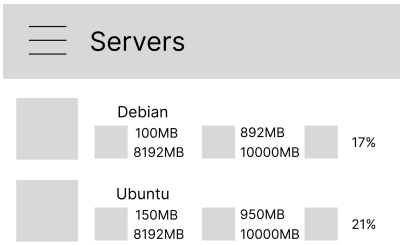


Рисунок 4.1 – Головний екран

При натисканні кнопки “+” має відкритися вікно для додавання сервера (рисунок 4.2) з полями для введення назви, IP адреси, порту, ім’я користувача, пароля та вибору приватного ключа.

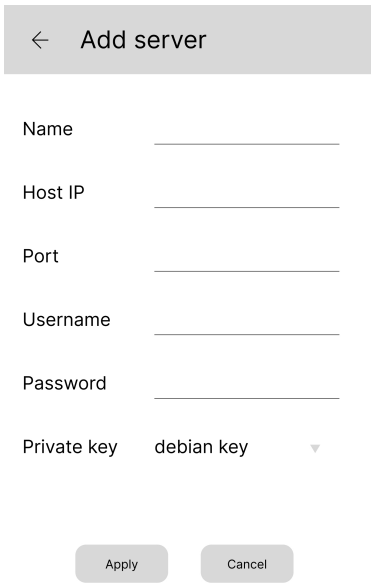


Рисунок 4.2 – Додавання сервера

При довгому натисканні на сервер повинно відкриватися контекстне меню для цього сервера (рисунок 4.3).

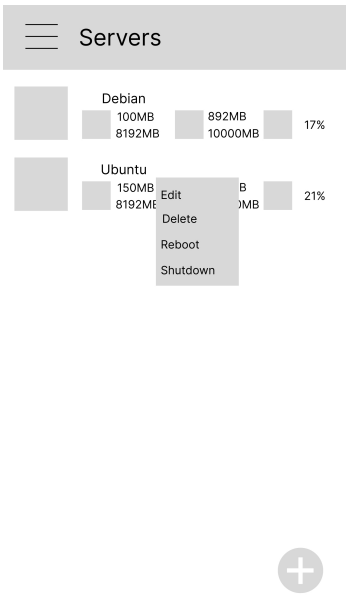


Рисунок 4.3 – Контекстне меню сервера

Екран редагування сервера (рисунок 4.4) має бути схожим на екран додавання, але поля мають бути заповненими існуючими даними при відкритті цього екрану.

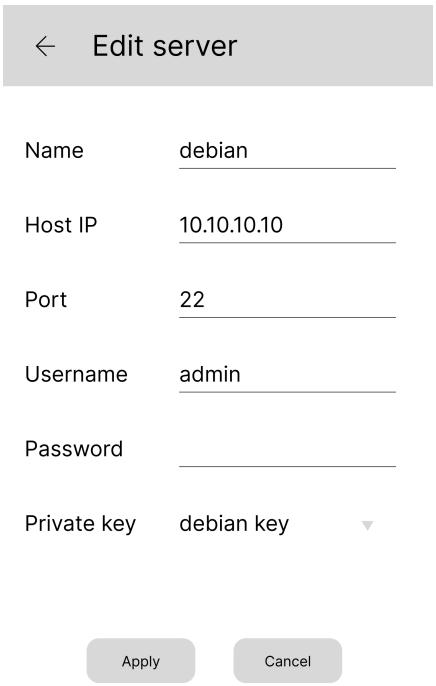


Рисунок 4.4 – Редагування сервера

Для переходу між основними розділами програми потрібно використовувати головне меню (рисунок 4.5), що знаходиться збоку.

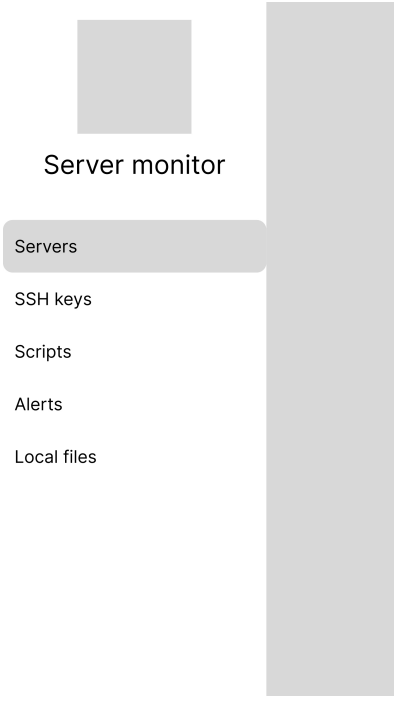


Рисунок 4.5 – Головне меню програми

На вкладці SSH keys (рисунок 4.6) повинен бути список із доданими SSH ключами та кнопкою “+” для додавання нового ключа.

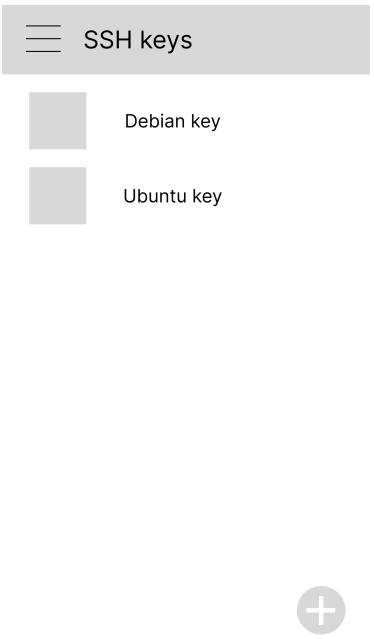


Рисунок 4.6 – Список з SSH ключами

При натисканні кнопки “+” відкривається вікно для додавання SSH ключа (рисунок 4.7).

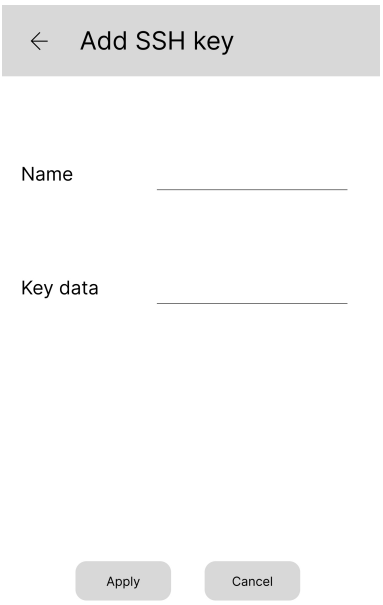


Рисунок 4.7– Додавання SSH ключа

При довгому натисканні на SSH ключ повинно відкриватися контекстне меню для цього ключа (рисунок 4.8).

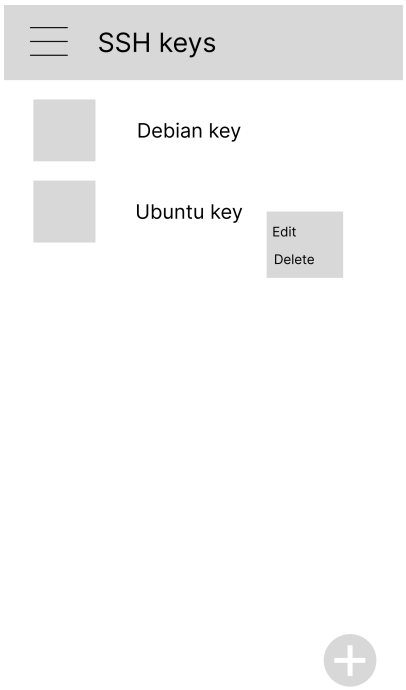


Рисунок 4.8– Контекстне меню SSH ключа

Екран редагування SSH ключа (рисунок 4.9) має бути схожим на екран додавання, але поля мають бути заповненими існуючими даними при відкритті цього екрану.

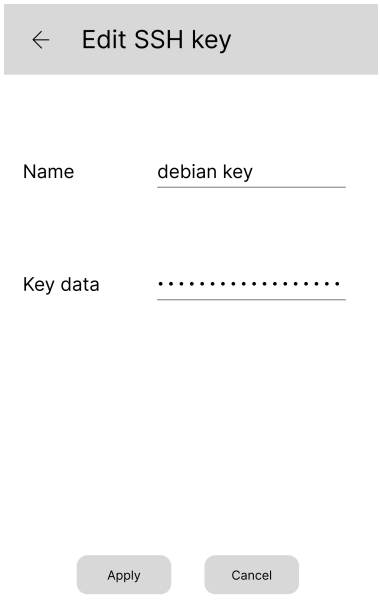


Рисунок 4.9 – Редагування SSH ключа

Сторінка сервера (рисунок 4.10) повинна відображати поточні метрики на графікаxx типу PieChart, кнопки для початку фонового моніторингу, відкриття терміналу, відкриття сторінки для перегляду файлів.

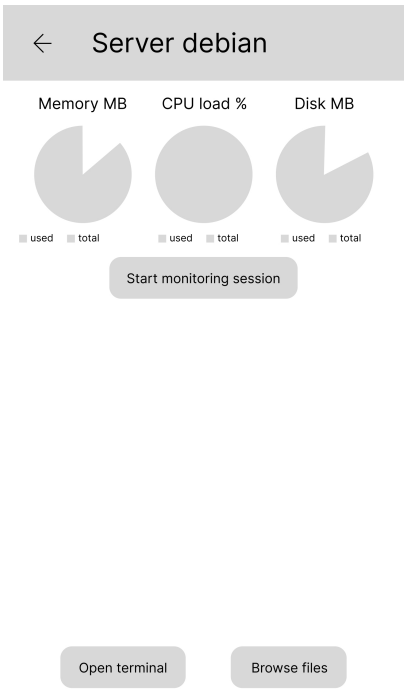


Рисунок 4.10 – Сторінка сервера

Натискання кноки Start monitoring session повинно починати моніторингову сесію та вмикати відображення трьох лінійних графік для відображення зміни метрик з часом (рисунок 4.11).

Рисунок 4.11 – Вигляд відображення моніторингової сесії

При натисканні на кнопку Open terminal повинен відкриватися екран з інтерактивним терміналом (рисунок 4.12). Він повинен містити поле для виводу результатів команд, поле для вводу команди та кнопку для запуску команди.

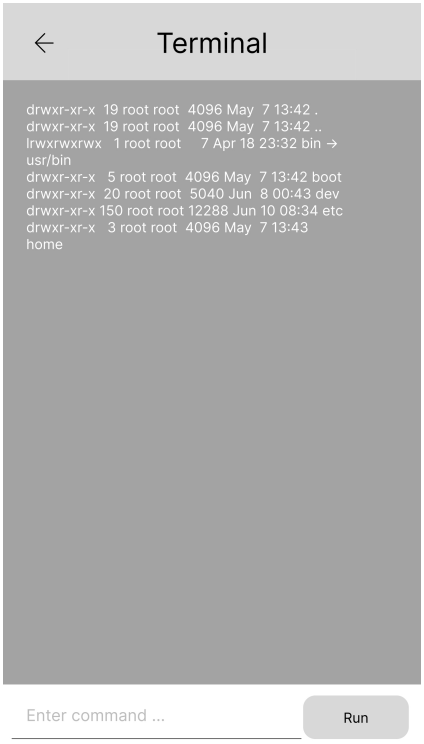


Рисунок 4.12 – Екран з інтерактивним терміналом

При натисканні на кнопку Browse files на екрані сервера повинен відкриватися екран зі списком файлів та директорій (рисунок 4.13). Він також має містити кнопки для переходу в кореневу, домашню директорії, оновлення сторінки, переходу до локальних файлів та кнопку меню. Внизу екрана повинен відображатися поточний шлях.

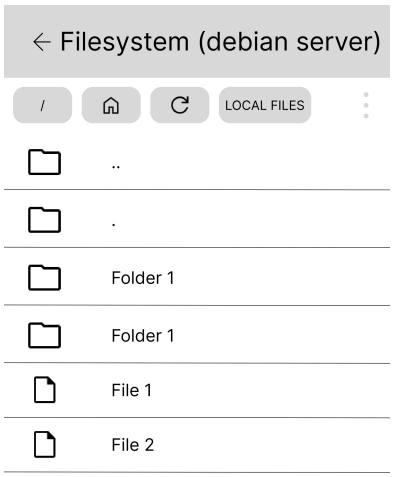


Рисунок 4.13 – Екран перегляду файлів на сервері

Контекстне меню екрану з файлами сервера (рисунок 4.14) повинно містити функції створення нового файлу, нової директорії та вставки попередньо скопійованого файлу.

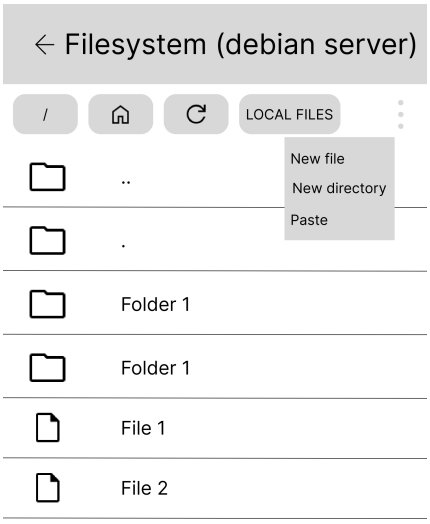


Рисунок 4.14 – Меню на екрані серверних файлів

Контекстне меню файла (рисунок 4.15) повинно містити такі пункти: Завантажити, Редагувати, Видалити, Перейменувати, Копіювати.

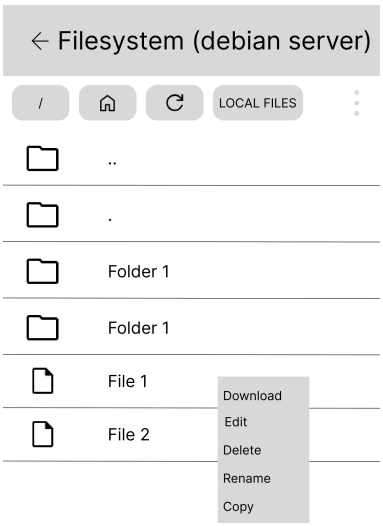


Рисунок 4.15 – Контекстне меню файла на екрані серверних файлів

При натисканні на кнопку Local file повинно відкриватися вікно з файлами, що розміщені локально на мобільному пристрої (рисунок 4.16). Воно повинно містити кнопки для повренення до початкової директорії, оновлення списку, переходу до файлів на сервері.

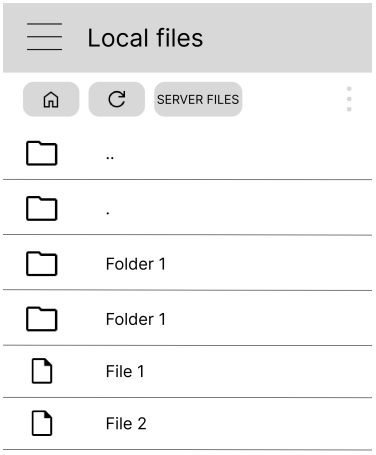


Рисунок 4.16 – Екран з файлами на локальному пристрої

Меню на цьому вікні (рисунок 4.17) повинно містити такі пункти: створення нового файлу, директорії, вставка попередньо скопійованого файлу.

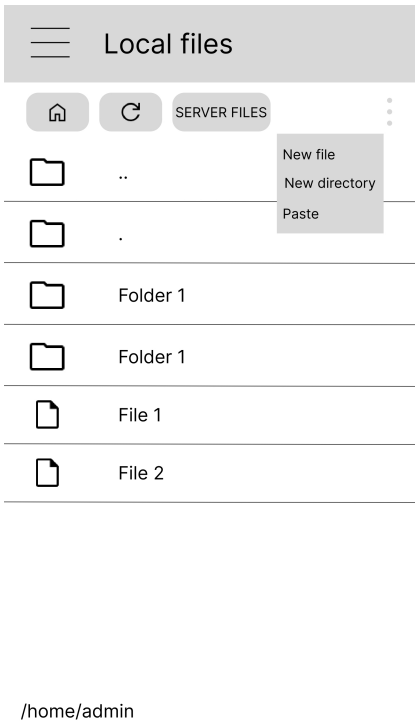


Рисунок 4.17 – Меню на екрані з файлами на локальному пристрої

Контекстне меню файла на цьому екрані (рисунок 4.18) повинно містити такі пункти: Редагувати, Видалити, Перейменувати, Копіювати.

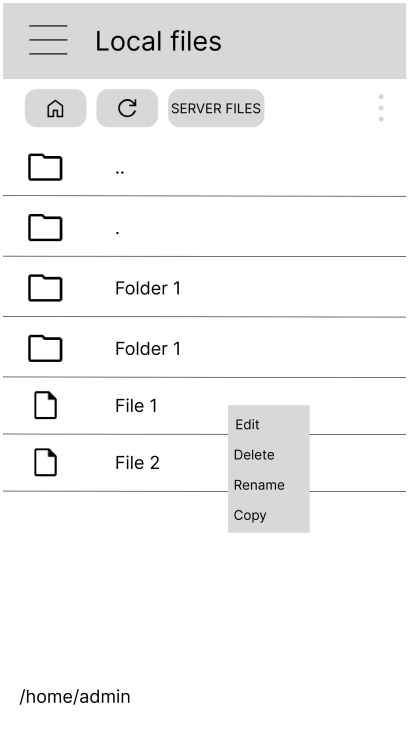


Рисунок 4.18 – Контекстне меню файлу на екрані з файлами на локальному пристрої

На екрані для shell скриптів (рисунок 4.19) повинні відображатися список скриптів та кнопка для додавання нового скрипта.

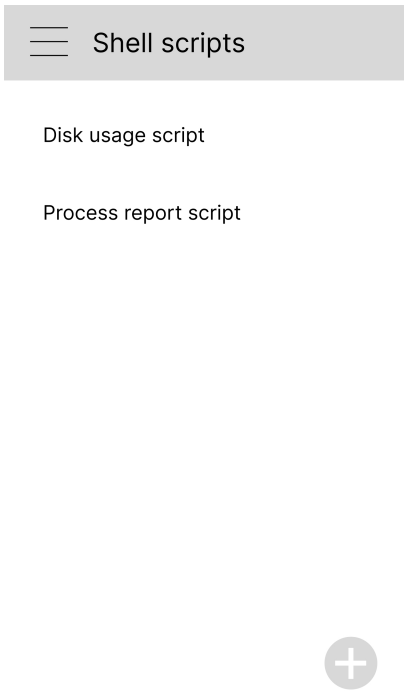


Рисунок 4.19 – Екран зі списком shell скриптів

При натисканні кнопки “+” повинен відкриватися екран для додавання нового скрипта (рисунок 4.20).

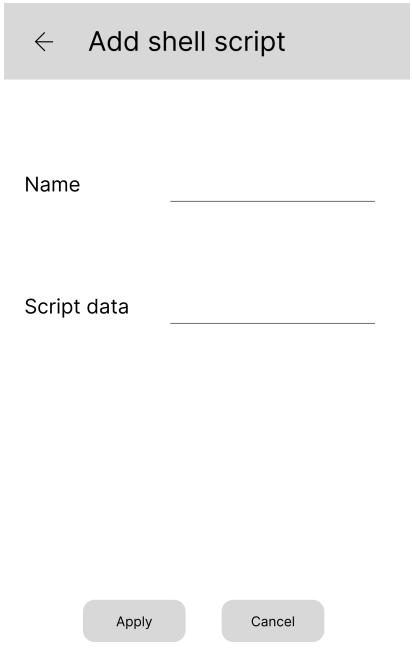


Рисунок 4.20 – Екран для додавання shell скрипта

Контекстне меню shell скрипта (рисунок 4.21) повинно містити такі пункти: Редагувати, Видалити.

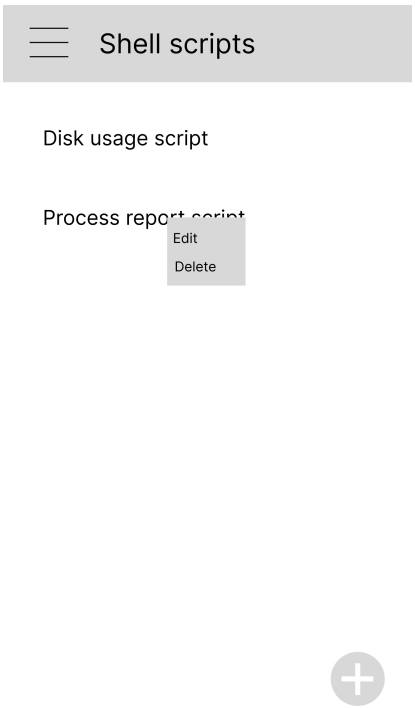


Рисунок 4.21 – Контекстне меню shell скрипта

Екран редагування shell скрипта (рисунок 4.22) має бути схожим на екран додавання, але поля мають бути заповненими існуючими даними при відкритті цього екрану.

Рисунок 4.22 – Екран для редагування shell скрипта

При натисканні на shell скрипт повинен відкриватися екран для вибору серверів (рисунок 4.23), на яких користувач бажає запустити скрипт. Користувач повинен мати можливість обрати будь-яку кількість серверів.

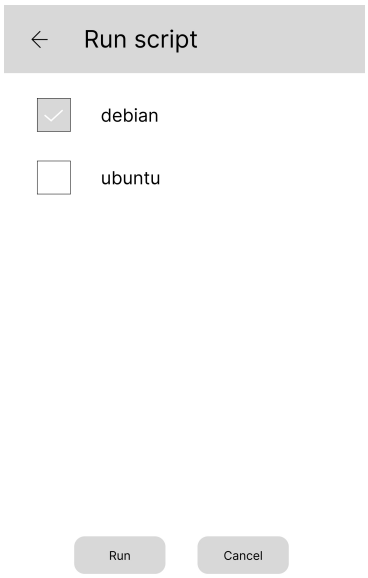


Рисунок 4.23 – Вибір серверів для запуску скрипта

При натисканні на кнопку Run, скрипт має почати виконання. Коли скрипт завершить виконання, мають відобразитися кнопки для показу результатів (рисунок 4.24).

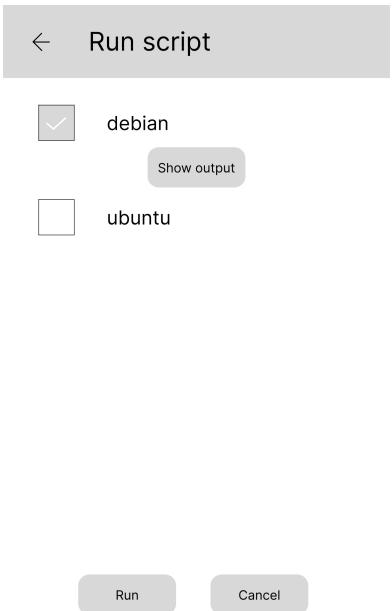


Рисунок 4.24 – Відображення кнопок для відкриття результатів виконання скрипта

При натисканні на кнопку Show output має відкриватися вікно з виводом скрипта (рисунок 4.25).

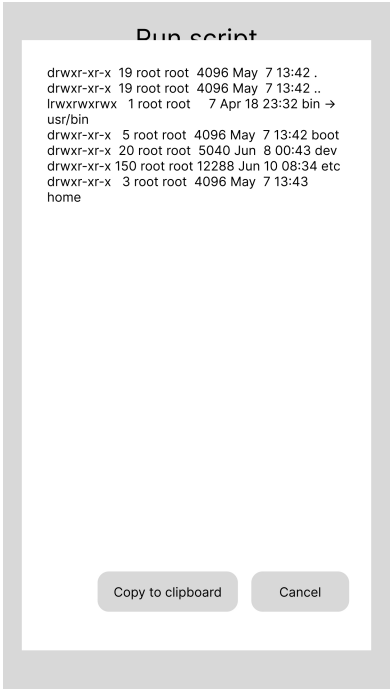


Рисунок 4.25 – Відображення результатів виконання скрипта

На екрані для сповіщень (рисунок 4.26) повинні відображатися список сповіщень та кнопка для додавання нового сповіщення.

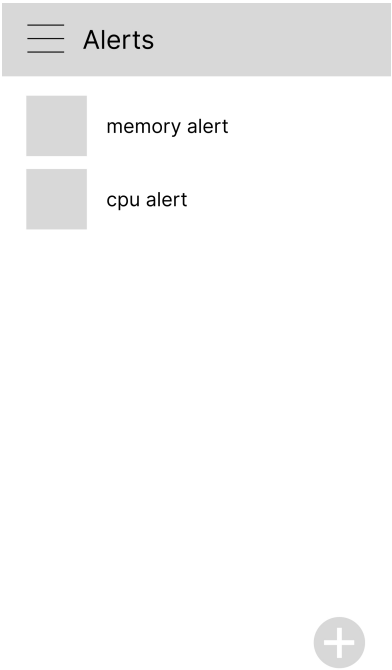


Рисунок 4.26 – Екран зі списком сповіщень

При натисканні кнопки “+” повинен відкриватися екран для додавання нового сповіщення (рисунок 4.27).

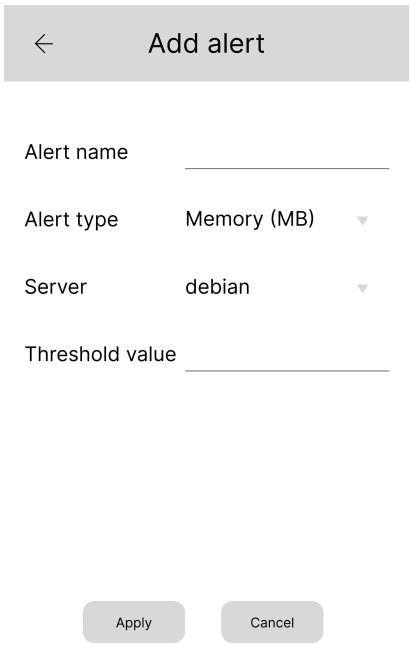


Рисунок 4.27 – Екран для додавання сповіщення

Контекстне меню сповіщення (рисунок 4.28) повинно містити такі пункти: Редагувати, Видалити.

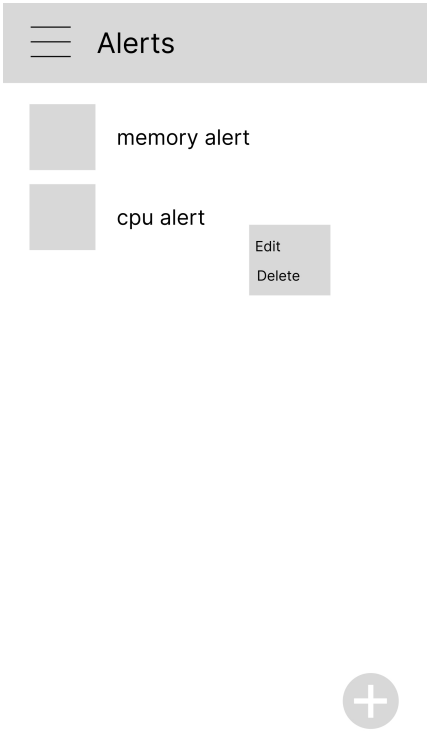


Рисунок 4.28 – Контекстне меню сповіщення

Екран редагування сповіщення (рисунок 4.29) має бути схожим на екран додавання, але поля мають бути заповненими існуючими даними при відкритті цього екрану.

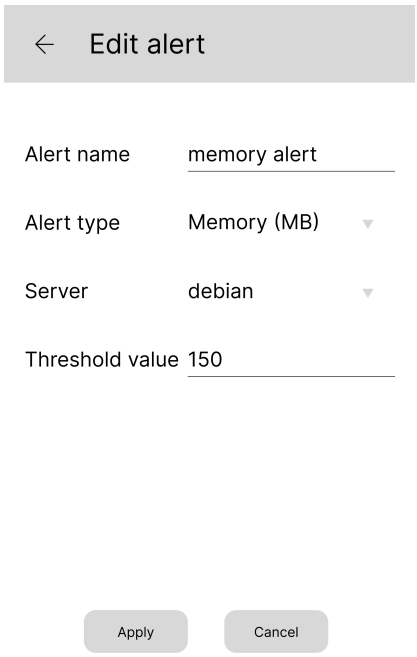


Рисунок 4.29 – Екран для редагування сповіщення

4.1.2 Функції для роботи з серверами:

4.1.2.1 Додавання сервера;

4.1.2.1.1 Введення назви сервера;

4.1.2.1.2 Введення IP адреси сервера;

4.1.2.1.3 Введення номера порта підключення;

4.1.2.1.4 Введення ім’я користувача сервера;

4.1.2.1.5 Введення пароля користувача сервера;

4.1.2.1.6 Вибір приватного ключа;

4.1.2.2 Редагування сервера;

4.1.2.2.1 Зміна назви сервера;

4.1.2.2.2 Зміна IP адреси сервера;

4.1.2.2.3 Зміна номера порта підключення;

4.1.2.2.4 Зміна імені користувача сервера;

4.1.2.2.5 Зміна пароля користувача сервера;

4.1.2.2.6 Зміна приватного ключа;

4.1.2.3 Видалення сервера;

4.1.2.4 Підключення до сервера за SSH для використання інтерактивної сесії термінала;

4.1.2.4.1 Перегляд виводу команд;

4.1.2.4.2 Введення та виконання нової команди;

4.1.2.5 Вимкнення сервера;

4.1.2.6 Перезавантаження сервера;

4.1.3 Функції для роботи з SSH ключами:

4.1.3.1 Додавання SSH ключа;

4.1.3.1.1 Введення назви приватного ключа;

4.1.3.1.2 Введення вмісту приватного ключа;

4.1.3.2 Редагування SSH ключа;

4.1.3.2.1 Зміни назви приватного ключа;

4.1.3.2.2 Зміна вмісту приватного ключа;

4.1.3.3 Видалення SSH ключа;

4.1.4 Моніторингові функції:

4.1.4.1 Перегляд останніх метрик використання ресурсів для кожного сервера;

4.1.4.2 Почати фоновий моніторинг для сервера;

4.1.4.3 Перегляд графіків використання ресурсів в реальному часі;

4.1.5 Функції для роботи з файлами:

4.1.5.1 Перегляд файлової системи сервера;

4.1.5.2 Перегляд файлової системи локального сховища;

4.1.5.3 Копіювання файла з сервера до локального сховища;

4.1.5.4 Копіювання файла з локального сховища до сервера;

4.1.5.5 Видалення файла;

4.1.6 Функції для роботи зі скриптами:

4.1.6.1 Додавання скрипта;

4.1.6.1.1 Введення назви скрипта;

4.1.6.1.2 Введення вмісту скрипта;

4.1.6.2 Редагування скрипта;

4.1.6.2.1 Зміна назви скрипта;

4.1.6.2.2 Зміна вмісту скрипта;

4.1.6.3 Видалення скрипта;

4.1.7 Функції для роботи зі сповіщеннями:

4.1.7.1 Додавання сповіщення;

4.1.7.1.1 Введення назви для сповіщення;

4.1.7.1.2 Вибір типу сповіщення (про пам’ять, диск, процесор, завантаження мережі, завантаження диску);

4.1.7.1.3 Введення порогового значення для метрики;

4.1.7.2 Редагування сповіщення;

4.1.7.2.1 Зміна назви для сповіщення;

4.1.7.2.2 Зміна типу сповіщення;

4.1.7.2.3 Зміна порогового значення для метрики;

4.1.7.3 Видалення сповіщення;

4.1.8 Додаткові вимоги:

– зручний та інтуїтивний користувацький інтерфейс;

– додаток повинен коректно відображатися на пристроях із різною роздільною здатністю.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

Не висуваються.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на мобільних пристроях з системами на чипі на базі технологій ARM.

Мінімальна конфігурація технічних засобів:

– мобільний пристрій з процесором Snapdragon 415 або аналогічним за продуктивністю процесором;

– об’єм оперативної пам’яті – 1 ГБ;

– об’єм вбудованого дискового простору – 16 ГБ ;

– зв’язок з мережею Інтернет зі швидкістю 1 Мбіт/с.

Рекомендована конфігурація технічних засобів:

– мобільний пристрій з процесором Snapdragon 695 або аналогічним за продуктивністю процесором;

– об’єм оперативної пам’яті – 4 ГБ;

– об’єм вбудованого дискового простору – 64 ГБ ;

– зв’язок з мережею Інтернет зі швидкістю 20 Мбіт/с.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційної системи Android версії 7 (версія SDK 24) та вище.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в форматі JSON (дані для імпорту).

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в вигляді файлу JSON (дані для експорту).

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Java.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі Android за допомогою інтегрованого середовища розробки Android Studio.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді текстових файлів із програмним кодом.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати інсталяційну версію програмного забезпечення у вигляді APK файла.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема бази даних;
– креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення рекомендованої літератури	10.03	
2.	Аналіз існуючих методів розв’язання задачі	17.03	
3.	Постановка та формалізація задачі	24.03	Технічне завдання
4.	Розробка інформаційного забезпечення	31.03	
5.	Алгоритмізація задачі	07.04	
6.	Обґрунтування вибору використаних технічних засобів	14.04	
7.	Розробка програмного забезпечення	28.04	Тексти програмного забезпечення
8.	Налагодження програми	05.05	Тести, результати тестування
9.	Виконання графічних документів	12.05	Графічний матеріал проекту
10	Оформлення пояснювальної записки	26.05	Пояснювальна записка

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: Мобільний застосунок для віддаленого моніторингу та керування серверами

КПІ.ІТ-9105.045490.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ	–	Програмне забезпечення
IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс
SDK	–	Software development kit
IT	–	Інформаційні технології
ER	–	Entity-Relation diagram
ОС	–	Операційна система.
БД	–	База даних.

ВСТУП

Сучасний світ усе більше покладається на цифрові рішення, що допомагають розв’язувати реальні задачі. Хоча деякі з них можна вирішити за допомогою програми, що працює офлайн та розміщена локально на комп’ютері, у більшості випадків зручніше та ефективніше використовувати онлайн-сервіси, що доступні безперебійно. Люди вже звикли до таких зручних функцій, як синхронізація даних між пристроями, колективна робота над одним документом чи проєктом, відсутність необхідності встановлювати додаткове програмне забезпечення на власний комп’ютер тощо. Поняття хмарних обчислень уже закріпилося в лексиконі програмних архітекторів. І незалежно від того, чи сервіс працює на власному обладнанні, чи у хмарному середовищі, чи на окремому сервері, чи в кластері – усе одно базовою одиницею, що дозволяє запускати безперебійно та надійно працюючі застосунки є сервер.

Постійне збільшення кількості серверів ускладнює керування ними та процес збирання інформації про їхній стан. Виникає необхідність у спрощенні та автоматизації цих процесів. Для системного інженера можливість спостерігати стан усіх серверів в одному місці та виконувати дії над ними автоматично або в декілька кліків, уникаючи необхідність ручної роботи, є важливою. Для цього існують різні рішення, наприклад, вбудовані від розробників хмарних середовищ, або стороннє програмне забезпечення. Проте, рішення в хмарних середовищах є специфічним для конкретного провайдера, в той час як адміністратор може мати сервери в різних провайдерів. Також, функціонал може бути недостатнім. Сторонні рішення, зазвичай, вимагають додаткової конфігурації та встановлення додаткових програм на сервер, що не завжди є зручним. Також, цей спосіб може бути занадто складним для користувачів, які працюють над любительськими проєктами, і не мають часу і бажання встановлювати ще щось, а хочуть одразу бачити результат.

Ще однією проблемою є те, що більшість наявних рішень не адаптовані для мобільних пристроїв. Іноді, виникає необхідність виконати якесь налаштування чи провести спостереження, в той час, як адміністратор не має доступу до комп’ютера. На допомогу приходять мобільні додатки. Сьогодні більшість людей тримають при собі смартфон чи планшет майже в будь-який час. Мобільний пристрій дає легкий доступ до майже невичерпних можливостей, що доступні на ходу.

Тому, враховуючи сучасні потреби користувачів, доцільно реалізувати програмне забезпечення, що вирішує проблеми, описані вище, на мобільній платформі, що і було обрано об’єктом дослідження даної роботи.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Предметна область віддаленого моніторингу та адміністрування є дуже обширною. Підходи до моніторингу різняться. Архітектурно, моніторинг може бути двох типів:

– push-based;

– pull-based.

Pull-based моніторинг не вимагає встановлення додаткового ПЗ (програми-агента) на сервери. Система містить центральний контролер, який самотужки збирає необхідні метрики з певною частотою. Якщо ж використовується push-based моніторинг, то на кожен сервер встановлюється додаткове ПЗ, яке надсилає дані необхідних метрик на центральний контролер. Ці два підходи порівняні на рисунку 1.1.

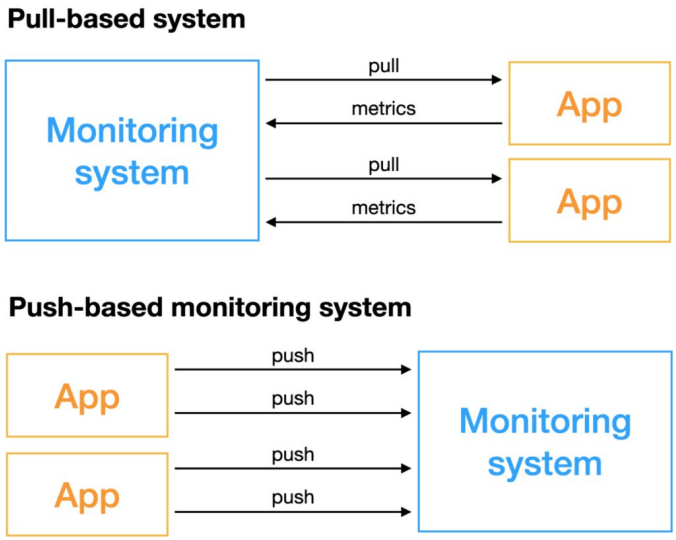


Рисунок 1.1 – Порівняльна схема роботи push-basxed та pull-based моніторингу [1]

Інша характеристика, за якою наявні рішення різняться, це тип застосунку. Більшість моніторингових програм заточені під роботу з великими та складними системами, де потрібен дуже детальний аналіз. Така складність не очікується від мобільного застосунку, тому, зазвичай, такі рішення є окремими серверними програмами. Зазвичай, вони

використовуються через веб-браузер або десктопний застосунок. Це не відповідає очікуванням користувача в тому випадку, коли йому важлива мобільність та простота. Вибір мобільних додатків з необхідним функціоналом є досить невеликим.

Популярним способом є використання нативних API хмарних платформ. Google Cloud, Azure, AWS дозволяють проводити базовий моніторинг серверів у графічній веб-консолі.

Також важливою різницею зі звичними системами моніторингу є потреба користувача зручно виконувати інтерактивні дії над серверами. Для користувача важливим є як можливість спостерігати за станом серверів, так і виконувати маніпуляції з файлами, програмами та самими серверами.

Моніторингові системи не задовольняють таку комбінацію вимог, так само і більшість наявних мобільних застосунків. Користувач подібних рішень очікує наступні можливості:

– моніторинг стану сервера – перевірка, чи сервер доступний;

– спостереження за використанням ресурсів – оперативної пам’яті, процесорного часу, дискового простору в реальному часі у вигляді графіків, які легко сприймати та читати;

– спостереження за програмами – можливість відображення встановлених на сервері програм, перевірка статусу важливих сервісів (наприклад, Docker), можливість читати логи;

– маніпулювання файлами – можливість переглядати файлову систему сервера, копіювати дані з сервера на мобільний пристрій та навпаки, а також між серверами;

– виявлення проблем та сповіщення про їх виникнення;

– можливість виконувати будь-які команди та скрипти на сервері.

1.2 Змістовний опис і аналіз предметної області

Pull-based моніторинг простіший в реалізації. Попри цю простоту, цей підхід має чимало недоліків [2]. По-перше, він вимагає, щоб на кожному

сервері був відкритий хоча б один порт, що погіршує загальну безпеку системи. По-друге, такий підхід погано масштабується. Якщо серверів стає багато, це створить велике навантаження на центральний контролер, і його потрібно буде вертикально масштабувати. Push-based рішення масштабується горизонтально (додавання великої кількості серверів не сильно шкодить продуктивності головної програми, що збирає метрики). Іншим суттєвим недоліком є статичність конфігурації. Для pull-based системи потрібно вказати конкретні IP адреси серверів. У сучасному IT середовищі це далеко не завжди є практичним, тому що, зазвичай, сервери працюють у кластерах, часто зникають і поновлюються з новими адресами. Проте, попри ці недоліки, для цієї роботи був обраний pull-based підхід, через свою простоту та швидкість налаштування.

Користувач зможе додати будь-який сервер, до якого можна підключитися по ssh, і моніторинг одразу працюватиме. Якби використовувався підхід зі встановленням агента на сервери, то для нього потрібно було б прописати IP адресу центрального контролера. У випадку з мобільним застосунком, це є проблематичним, тому що, зазвичай, мобільний пристрій не має власної IP адреси, і виходить в Інтернет з IP адресою базової станції оператора, яка є однаковою для багатьох користувачів. Користувач не зможе відкрити на своєму телефоні порт, до якого агент міг би під’єднатися. До того ж, адреса, видана провайдером, часто змінюється. Тут також варто згадати рішення mosh, яке цю проблему обходить. Воно зберігає сесію термінала, навіть якщо IP адреса користувача змінюється. Але в нашому випадку збирання даних проводиться з певним інтервалом, тому немає необхідності постійно тримати ssh сесію.

Також, функціональні вимоги є такими, що часто виникатиме необхідність виконувати певні дії над серверами в конкретний час за бажанням користувача, тому pull-based підхід більш доцільний у даному випадку.

Використання API хмарних платформ теж має свої недоліки. Якщо користувач має сервери на різних платформах, то йому потрібно використовувати інструменти кожної платформи, а не єдиний інструмент. А якщо сервер знаходиться у приватному датацентрі, чи сервером є, наприклад, комп'ютер Raspberry Pie для експериментів удома, то користувач взагалі не матиме готового API та веб-інтерфейсу.

1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації мобільного застосунку для віддаленого моніторингу та керування серверами. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

Існує багато варіантів архітектури ПЗ. Найпопулярнішими на сьогодні є монолітна, клієнт-серверна, розподілена, мікросервісна та багатошарова архітектури.

Монолітна архітектура є найдавнішою та найпростішою для розуміння. У результаті її застосування, кінцева програма являє собою один виконуваний бінарний файл, який є готовим до використання. Усі функції знаходяться всередині цього файлу. У ньому також можуть міститися необхідні для роботи бібліотеки, хоча вони також можуть бути надані операційною системою. Моноліт може взагалі не взаємодіяти з зовнішніми компонентами, або може зберігати дані в окремій базі даних чи з'єднуватися із зовнішніми сервісами, які не розробляються розробником цієї програми. Переваги такого підходу включають простоту розробки та наявність єдиної кодової бази. Також, монолітний застосунок може мати вищу продуктивність та менші накладні витрати, тому що немає поділу на компоненти та відсутня комунікація між ними.

Клієнт-серверна архітектура передбачає наявність двох різних програм – клієнта, який встановлюється на машини користувачів та сервера, який працює в датацентрі компанії, що надає послуги. Сервер є центральною програмою, яка отримує та опрацьовує запити від усіх клієнтів, надає необхідні сервіси та ресурси, зберігає дані. Клієнт та сервер взаємодіють через мережу. Цей підхід має багато переваг. По-перше, основне навантаження переноситься на сервер, який нескладно зробити потужним, в той час як клієнтська програма може бути легкою і запускатися навіть на непродуктивних пристроях. Сервер може зберігати набагато більше даних, ніж це можна зробити на клієнтському пристрої, і користувач може запитувати тільки те, що йому потрібно. Також плюсом є централізоване управління, що дозволяє легко змінювати спектр послуг, що надаються, оновлювати конфігурацію та ПЗ без необхідності будь-яких дій з боку клієнтів. Проте, є й очевидні недоліки. Програма-клієнт частково або повністю втрачає функціональність, якщо відсутнє з’єднання з мережею. Користувач втрачає контроль над конфігурацією сервісу та списком послуг, що надаються. Розробник може обмежити надання певних послуг, що може призвести до розчарування користувача та його бажання використовувати іншу програму або відмовитися від продукту, що надається як сервіс. Також можливі загрози безпеки, якщо персональні дані багатьох осіб зберігаються в одній точці. Якщо на сервері виникають проблеми, ніхто не зможе користуватися сервісом. Цей підхід не є оптимальним для даної роботи. Якщо розбити програму на клієнтський застосунок на платформі Android та бекенд, що буде працювати на сервері та надавати послуги, то продуктивність, масштабованість програми покращиться, і ще буде можливість додати зручні функції синхронізації даних. Але для цього потрібно буде тримати програму-сервер постійно запущеною, що вимагає певних фінансових витрат. Тобто, для даної роботи буде потрібно знайти модель монетизації, що не є предметом дослідження. І тоді сама клієнтська програма не буде повнофункціональною, тому що не зможе працювати без

сервера. Також, імовірно, потрібно буде додати реєстрацію для користувача та зберігати його персональні ти приватні дані. Це має ризики безпеки, а також необхідність слідувати багатьом регуляційним вимогам. Тому в нашому випадку було вирішено використовувати монолітну архітектуру.

Розподілена архітектура передбачає наявність багатьох рівноправних вузлів (комп'ютерів), які взаємодіють між собою за допомогою мережі. Робота розподілена між вузлами, що покращує продуктивність системи загалом. Немає необхідності у потужному центральному сервері. Така система є надійною – вихід з ладу одного чи декількох вузлів не призведе до припинення надання сервісу.

При використанні мікросервісної архітектури, програма розбивається на менші компоненти, кожен з яких має свій окремий цикл розробки. Вони взаємодіють між собою за допомогою API. Це дає значну гнучкість, тому що кожен компонент можна розробляти, використовуючи будь-яку мову програмування та довільний стек технологій. Головне, щоб кожен мікросервіс відкривав доступ до стандартизованого API, яке розуміють інші компоненти. Цей підхід добре підходить для систем, де є високі вимоги до масштабованості. Згідно 6 принципу 12 факторної розробки програм [3], кожен екземпляр сервісу не має стану. Тому така система масштабується горизонтально (додаванням кількості екземплярів сервісів). Найпопулярнішою технологією, яка дозволяє реалізувати цей підхід є Kubernetes.

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

Реалізувати програмний код можна навіть у найпростішому текстовому редакторі, проте він не надасть достатні функціональні можливості для зручної розробки.

Для багатьох мов програмування можна використовувати легкий текстовий редактор Vim, або Neovim [4] (форк Vim з розширеним функціоналом). Цей редактор є зручним для багатьох інженерів, які хочуть

виконувати всі дії за допомогою клавіатури. Він особливий тим, що має два основних режими – нормальний (Normal) та режим введення (Input). У режимі введення розробник пише код, а нормальний режим дозволяє дуже швидко маніпулювати текстом за допомогою різних комбінацій клавіш. Цей редактор (особливо Neovim) має величезну кількість розширень, які дозволяють зробити його функціонально подібним до більшості сучасних IDE. Існують проєкти, які додають подібну функціональність до Neovim, наприклад LunarVim [5]. Вигляд його інтерфейсу можна побачити на рисунку 1.2.

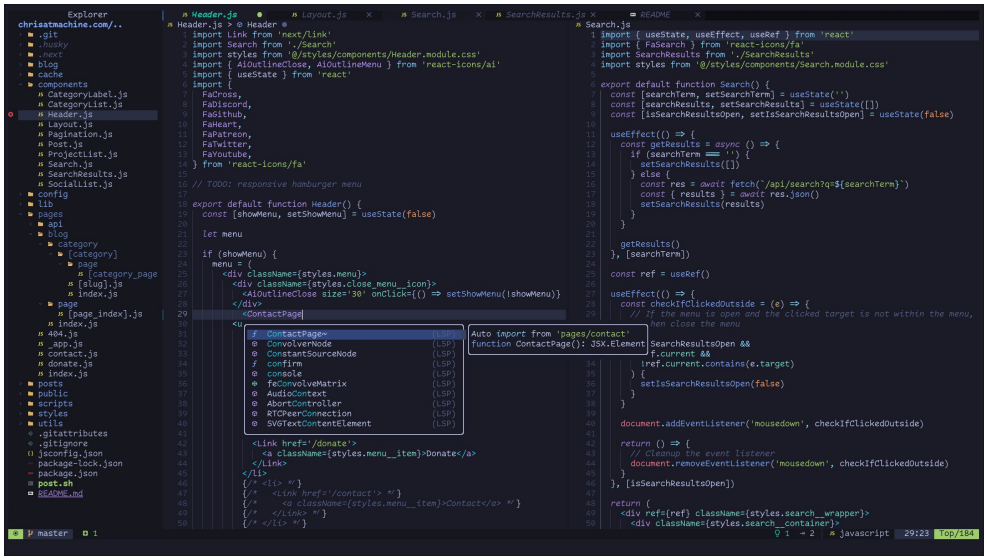


Рисунок 1.2 – Можливості текстового редактора Neovim у модифікації LunarVim [5]

Visual Studio Code – дуже популярний редактор коду. Має велику кількість розширень, є зручним та легким. Його можна використовувати для написання коду будь-якою мовою програмування. Розширення роблять його функціонал порівняним з більш важкими та потужними IDE. Вигляд його інтерфейсу зображений на рисунку 1.3.

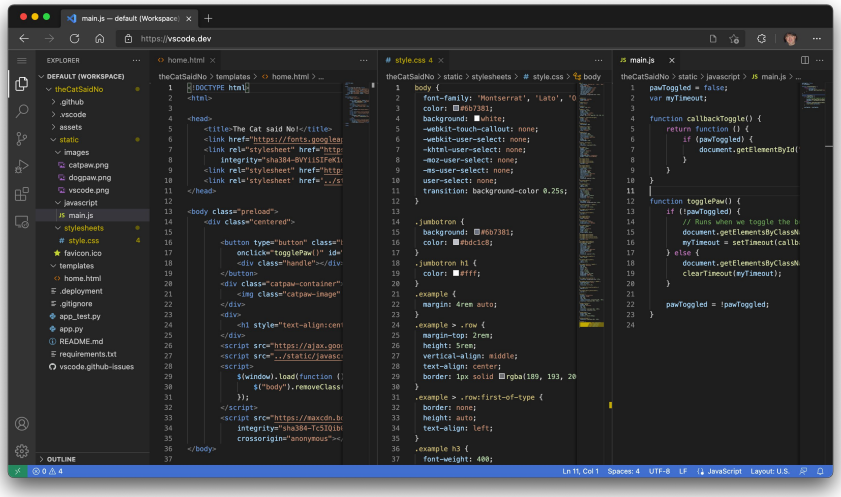


Рисунок 1.3 – Редактор коду Visual studio code (версія, що відкрита у веб-браузері)

Android Studio – це найпопулярніша IDE для розробки на платформі Android. Як базу, ця програма використовує продукт IntelliJ IDEA, що також є популярною IDE, але не спеціалізованою для Android. Порівняно з нею, Android Studio додає специфічний для Android розробки функціонал – графічний редактор макетів, профайлер для моніторингу споживаних додатком ресурсів, менеджер пристроїв, який дозволяє запустити застосунок на віртуалізованому чи реальному мобільному пристрої тощо. Оскільки ця IDE має найбільш багатий функціонал для розробки на платформі Android, а також є рекомендованим середовищем для цього, то розробка велася з її допомогою. Інтерфейс цієї IDE показано на рисунку 1.4.

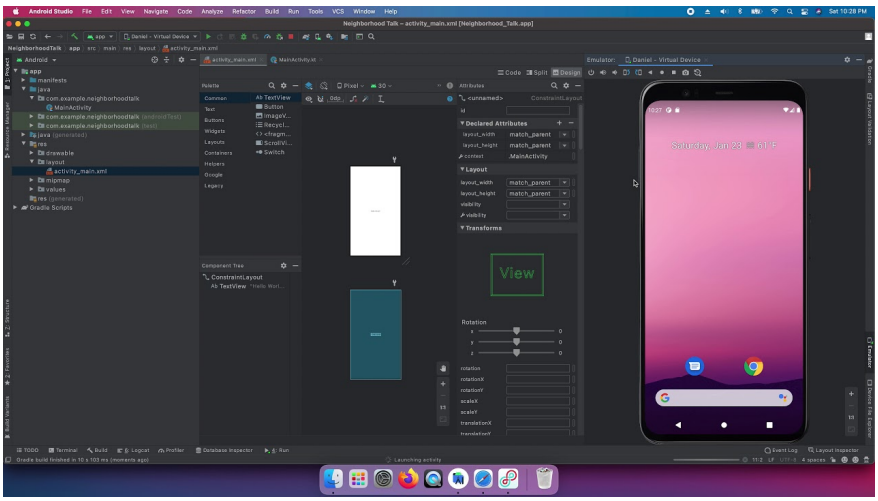


Рисунок 1.4 – Інтегроване середовище розробки Android Studio

1.3.3 Аналіз відомих програмних продуктів

Найближчими аналогами застосунку, що проєктується, є Android програми Termius та DaRemote.

Termius – це зручна програма з сучасним інтерфейсом користувача, що дозволяє підключатися до серверів через ssh та працювати з їх файловою системою. Проте, в цьому застосунку відсутні функції для зручного моніторингу.

DaRemote надає можливість моніторингу ресурсів, та ще деякі додаткові функції, як, наприклад, перегляд наявних на сервері Docker контейнерів. Проте, деякі функції доступні лише в Pro версії.

Для порівняння даної роботи з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал	Ця робота	Termius	DaRemote
Додавання сервера	Можливе	Можливе	Можливе (для необмеженої кількості підключень необхідно купити Pro версію)
Моніторинг статусу сервера та використання ресурсів	Моніторинг завантаження пам’яті, процесора, диску, мережі, I/O активності	Відсутній	Відображає завантаження пам’яті, процесора, диску, мережі, I/O активності

Продовження таблиці 1.1

Функціонал	Ця робота	Termius	DaRemote
Управління файлами сервера	Можливість переглядати файли та директорії, перейменовувати, видаляти файли, створювати нову директорію, переміщувати файли між локальним пристроєм та сервером	Можливість переглядати файли та директорії, видаляти файли, створювати нову директорію, переміщувати файли між локальним пристроєм та сервером	Можливість переглядати файли та директорії, перейменовувати, видаляти файли, створювати нову директорію, завантажувати файл з пристрою на сервер
Експорт та імпорт даних	Наявний файловий експорт та імпорт	Можливо зберігати дані в хмарі	Наявний файловий експорт та імпорт, а також збереження файлу в хмарі (Pro-версія)
Сповіщення про помилки	Сповіщення про перевищення меж використання ресурсів, а також при зупинці сервера	Відсутні	Відсутні

Продовження таблиці 1.1

Функціонал	Ця робота	Termius	DaRemote
Спрощення адміністрування за рахунок готових команд та можливості додавання власних скриптів	Можливість додавання скриптів та виконання їх на одному чи кількох серверах	Наявна функція додавання скриптів, а також доповнення команди в терміналі за рахунок історії попередніх введень	Відсутнє
Перегляд наявних на сервері застосувань та їх логів	Перегляд наявних systemd сервісів та їх логів	Відсутній	Перегляд наявних docker контейнерів, запущених процесів

1.4 Аналіз вимог до програмного забезпечення

Головною функцією програмного забезпечення є перегляд стану серверів та розміру використаних ними ресурсів. Більше варіантів використання можна побачити у графічному матеріалі, Креслення 1 – схема структурна варіантів використання.

В таблицях 1.2 - 1.17 наведені основні варіанти використання програмного забезпечення.

Таблиця 1.2 - Варіант використання UC-01

Use case name	Під’єднатися до сервера з ssh
Use case ID	UC-01

Продовження таблиці 1.2

Goals	Під’єднання до сервера з ssh та виконання будь-яких команд у терміналі
Actors	Користувач
Trigger	Користувач хоче під’єднатися до сервера
Extension	Якщо з’єдання не пройшло успішно, користувач отримує сповіщення про це у спливаючому вікні
Post-Condition	Створення ssh з’єднання

Таблиця 1.3 - Варіант використання UC-02

Use case name	Вставити файл
Use case ID	UC-02
Goals	Вставити файл, що було скопійовано раніше, на сервер чи локальне сховище
Actors	Користувач
Trigger	Користувач хоче вставити файл
Pre-conditions	Файл повинен бути скопійованим
Flow of Events	Користувач відкриває директорію на сервері чи локально, куди він хоче вставити об’єкт. Натискаю кнопку Меню, натискає “Paste”
Extension	Якщо при виконанні дії сервер чи локальна машина повідомляє про помилку, вона відображається користувачу
Post-Condition	Відображається вікно, яке показує прогрес операції. Його можна згорнути

Таблиця 1.4 - Варіант використання UC-03

Use case name	Копіювати файл
Use case ID	UC-03
Goals	Позначити файл для копіювання
Actors	Користувач
Trigger	Користувач хоче скопіювати файл
Pre-conditions	-
Flow of Events	Користувач переходить у директорію на сервері чи локально, звідки він хоче виконати копіювання. Відкриває контекстне меню для потрібного файлу, обирає пункт “Copy”
Extension	-

Продовження таблиці 1.4

Post-Condition	Обраний об’єкт можна побачити в меню

Таблиця 1.5 - Варіант використання UC-04

Use case name	Вимкнути сервер
Use case ID	UC-04
Goals	Вимкнути обраний сервер
Actors	Користувач
Trigger	Користувач хоче вимкнути сервер
Pre-conditions	До сервера здійснено підключення та він увімкнений
Flow of Events	Користувач обирає сервер та відкриває контекстне меню, натискає “Shutdown”
Extension	Якщо команда вимкнення не пройшла успішно, користувач отримує про це повідомлення у спливаючому вікні
Post-Condition	Інтерфейс відображає за допомогою індикації, що сервер вимкнуто

Таблиця 1.6 - Варіант використання UC-05

Use case name	Додати сервер
Use case ID	UC-05
Goals	Додати сервер та встановити з ним успішне з’єднання
Actors	Користувач
Trigger	Користувач хоче додати сервер
Pre-conditions	-
Flow of Events	Користувач натискає на кнопку “+”. Вводить необхідні дані – назва сервера, його IP адреса, порт, ім’я користувача на сервері, пароль або приватний ключ. Натискає кнопку “Apply”. Чекає, поки відобразиться індикація про успішність чи неуспішність з’єднання. Якщо з’єднання не встановилося, користувач може відредагувати дані та спробувати ще раз.
Extension	Якщо з’єднання не встановилося, відображається індикація про це, але при цьому дані, введені користувачем, зберігаються
Post-Condition	Сервер додано до списку та збережено дані про з’єднання.

Таблиця 1.7 - Варіант використання UC-06

Use case name	Запустити фоновий моніторинг
Use case ID	UC-06
Goals	Запустити фоновий моніторинг сервера зі збереженням статистики
Actors	Користувач
Trigger	Користувач хоче почати фоновий моніторинг сервера
Pre-conditions	Сервер уже додано та початкове з'єднання з ним пройшло успішно
Flow of Events	Користувач обирає сервер, який він хоче моніторити. Натискає на кнопку Меню, обирає пункт Почати фоновий моніторинг. Коли користувач бажає завершити моніторингову сесію, він натискає кнопку Зупинити моніторинг в меню. Статистичні дані моніторингу зберігаються в файл
Extension	Якщо сервер повідомив про помилку, то користувач отримує відповідне повідомлення у спливаючому вікні, моніторинг припиняється
Post-Condition	Моніторинг у фоновому режимі почато. Якщо його зупинити, статистичні дані зберігаються в файл

Таблиця 1.8 - Варіант використання UC-07

Use case name	Переглянути стан сервера та використання ним ресурсів
Use case ID	UC-07
Goals	Переглянути статус сервера (працюючий чи вимкнений), а також побачити поточну кількість використаних ресурсів
Actors	Користувач
Trigger	Користувач хоче переглянути стан сервера
Pre-conditions	Сервер уже додано, та початкове з'єднання пройшло успішно
Flow of Events	Користувач відкриває головне меню зі списком усіх серверів. У цей час система опитує кожен доданий та працюючий сервер і відображає отримані дані. Користувач може спостерігати їх у головному меню, або клікнути на конкретний сервер, який його цікавить, і побачити більш детальні дані, які стосуються саме його

Продовження таблиці 1.8

Extension	Якщо на сервері виникає помилка при опрацюванні запиту програми, то біля сервера відображається індикатор, який сповіщає про помилку
Post-Condition	Дані про сервери відображають на головному меню та на сторінці сервера та регулярно оновлюються

Таблиця 1.9 - Варіант використання UC-08

Use case name	Переглянути файлову систему
Use case ID	UC-08
Goals	Перегляд файлової система сервера або локальної машини, переміщення між директоріями
Actors	Користувач
Trigger	Користувач хоче переглянути файлову систему
Pre-conditions	-
Flow of Events	Користувач відкриває сторінку сервера, файли якого хоче переглянути, натискає кнопку Browse files або обирає Local files в меню і переглядає файли на поточному пристрої
Extension	Якщо сервер сповіщає про помилку, користувач бачить відповідне повідомлення у спливаючому вікні
Post-Condition	Користувач переглядає файли та директорії

Таблиця 1.10 - Варіант використання UC-09

Use case name	Переглянути список відкритих портів
Use case ID	UC-09
Goals	Перегляд списку відкритих портів на сервері
Actors	Користувач
Trigger	Користувач хоче переглянути список відкритих портів на сервері
Pre-conditions	Сервер уже додано та початкове підключення до нього спрацювало
Flow of Events	Користувач відкриває сторінку потрібного сервера. Натискає на кнопку View open ports
Extension	Якщо на сервері виникла помилка, користувач бачить про це сповіщення у спливаючому вікні

Продовження таблиці 1.10

Post-Condition	Користувач бачить список відкритих на сервері портів

Таблиця 1.11 - Варіант використання UC-10

Use case name	Переглянути список запущених сервісів та їхні логи
Use case ID	UC-10
Goals	Перегляд списку запущених сервісів та їхні логи
Actors	Користувач
Trigger	Користувач хоче переглянути запущені сервіси та їхні логи
Pre-conditions	Сервер уже додано, та початкове з'єднання з ним пройшло успішно
Flow of Events	Користувач відкриває сторінку з сервером, який його цікавить. Натискає кнопку View services. Бачить список systemd сервісів. Натискає кнопку View logs біля сервісу, який його цікавить
Extension	Якщо на сервері виникла помилка, користувач бачить про це сповіщення у спливаючому вікні
Post-Condition	Користувач бачить список сервісів та потрібні йому логи

Таблиця 1.12 - Варіант використання UC-11

Use case name	Виконати скрипт на одному чи декількох серверах
Use case ID	UC-11
Goals	Виконання обраного скрипта на одному чи кількох серверах
Actors	Користувач
Trigger	Користувач хоче виконати обраний скрипт
Pre-conditions	Потрібні сервери додані та до них здійснено успішне початкове з'єднання. Бажаний скрипт уже створено
Flow of Events	Користувач переходить на сторінку з власними скриптами, обирає потрібний скрипт, обирає необхідні сервери (мульти-вибір). Починається виконання скрипта. По завершенню виконання скрипта біля відповідного сервера відобразиться кнопка для відображення згенерованого виводу
Extension	Якщо на сервері виникла помилка, користувач бачить про це сповіщення у спливаючому вікні. Якщо скрипт завершився з помилкою, користувач теж буде повідомлений

Продовження таблиці 1.12

Post-Condition	Скрипт виконано та користувач повідомлений про завершення виконання

Таблиця 1.13 - Варіант використання UC-12

Use case name	Створити скрипт
Use case ID	UC-12
Goals	Створення та збереження власного скрипта, який потім можна багаторазово використовувати
Actors	Користувач
Trigger	Користувач хоче додати власний скрипт
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку з власними скриптами, натискає на кнопку “+”, вводить назву скрипта та його текст, натискає кнопку Apply
Extension	-
Post-Condition	Скрипт створено та збережено

Таблиця 1.14 - Варіант використання UC-13

Use case name	Під’єднатися до сервера з ssh
Use case ID	UC-13
Goals	Під’єднання до сервера з ssh та виконання будь-яких команд у терміналі
Actors	Користувач
Trigger	Користувач хоче під’єднатися до сервера
Pre-conditions	-
Flow of Events	Користувач натискає на сервер, до якого хоче під’єднатися. Відкривається сторінка сервера. Користувач натискає кнопку Open terminal
Extension	Якщо виникла помилка при з’єднанні, про це з’явиться спливаюче сповіщення
Post-Condition	Термінал відкритий, користувач може виконувати команди на сервері

Таблиця 1.15 - Варіант використання UC-14

Use case name	Експорт та імпорт даних
Use case ID	UC-14
Goals	Експортувати та імпортувати дані
Actors	Користувач
Trigger	Користувач хоче експортувати або імпортувати дані
Pre-conditions	-
Flow of Events	Користувач відкриває пункт меню Manage data. Натискає Export data або Import data. При експорті, обирає застосунок, якому хоче відправити файл. При імпорті, обирає файл, який хоче зчитати
Extension	Якщо виникла помилка при зчитуванні даних, буде відображено сповіщення
Post-Condition	Експорт чи імпорт пройшов успішно

1.4.1 Розроблення функціональних вимог

У таблиці 1.16 наведено загальну модель вимог, а в таблицях 1.17 – 1.23 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити у таблиці 1.24.

Таблиця 1.16 – Модель вимог у загальному вигляді

	Вимога	Пріоритет	Ризик
FR-1	Додавання сервера	Високий	Високий
FR-2	Моніторинг статусу сервера та використання ресурсів	Високий	Середній
FR-3	Управління файлами сервера	Високий	Середній
FR-4	Експорт та імпорт даних	Середній	Високий
FR-5	Сповіщення про помилки	Високий	Низький
FR-6	Спрощення адміністрування за рахунок готових команд та можливості додавання власних скриптів	Середній	Низький

Продовження таблиці 1.16

FR-7	Перегляд наявних на сервері застосувань та їх логів	Середній	Середній

Таблиця 1.17 – Функціональна вимога FR-1

Назва	Додавання сервера
Опис	Система повинна надавати можливість додавання та підключення до серверів по ssh

Таблиця 1.18 – Функціональна вимога FR-2

Назва	Моніторинг статусу сервера та використання ресурсів
Опис	Система повинна надавати можливість перегляду використаних ресурсів (пам’ять, дисковий простір, процесорний час, мережева активність) система на даний момент, а також надавати історію використання ресурсів

Таблиця 1.19 – Функціональна вимога FR-3

Назва	Управління файлами сервера
Опис	Система повинна надавати можливість переглядати список файлів на сервері, переміщатися між директоріями, видаляти, копіювати файли між сервером та локальним пристроєм

Таблиця 1.20 – Функціональна вимога FR-4

Назва	Експорт та імпорт даних
Опис	Система повинна надавати можливість експортування наявних підключень, а також зібраних даних на протязі моніторингових сесій

Таблиця 1.21 – Функціональна вимога FR-5

Назва	Сповіщення про помилки

Продовження таблиці 1.21

Опис	Система повинна мати можливість працювати в фоні, і повідомляти за допомогою push-сповіщення, якщо певний сервер зупинився, або використовує занадто багато ресурсів

Таблиця 1.22 – Функціональна вимога FR-6

Назва	Спрощення адміністрування за рахунок готових команд та можливості додавання власних скриптів
Опис	Система повинна надавати можливість додавання власних скриптів, які можна багаторазово запускати на одному чи кількох серверах одночасно

Таблиця 1.23 – Функціональна вимога FR-7

Назва	Перегляд наявних на сервері застосувань та їх логів
Опис	Система повинна надавати можливість переглядати список запущених процесів, а також сервісів. Має бути можливість перегляду логів сервісів

Таблиця 1.24 – Матриця трасування вимог

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7
UC-01						+	
UC-02			+				
UC-03			+				
UC-04							
UC-05	+						
UC-06		+			+		
UC-07		+					
UC-08			+				
UC-09							+

Продовження таблиці 1.24

UC-10							+
UC-11						+	
UC-12						+	
UC-13	+						
UC-14				+			

1.4.2 Розроблення нефункціональних вимог

Система має наступні нефункціональні вимоги: безпека, продуктивність та зручність.

Система передбачає зберігання приватних ключів та паролів для доступу до серверів. Ці дані є конфіденційними, тому, згідно вимог безпеки, мають надійно зберігатися і не передаватися третім особам.

Вимога продуктивності встановлює, що програма повинна швидко та плавно працювати, а також споживати невелику кількість оперативної пам’яті, процесорного часу ти дискового простору. Ці показники не повинні надто сильно погіршуватися при збільшенні кількості підключень.

Усі функції програми повинні бути легко доступними та інтуїтивно зрозумілими для користувача. Інтерфейс має бути зручним для використання і приємним на вигляд.

1.5 Постановка задачі

Розробка призначена для спостереження за станом серверів та використанням ними ресурсів, а також віддаленого адміністрування серверів.

Метою розробки є спрощення процесу моніторингу серверів та управління ними для системних адміністраторів та DevOps інженерів.

Функціональні задачі включають:

– забезпечення можливості моніторингу серверів;

– забезпечення можливості інтерактивно виконувати будь-які команди на серверах;

– забезпечення можливості виконання на серверах скриптів, створених користувачем;

– забезпечення можливості роботи з файлами на серверах;

– реалізація сповіщень про перевикористання ресурсів чи недоступність сервера.

Висновки до розділу

У даному розділі була розглянута предметна область, спектр існуючих на ринку рішень, були сформульовані вимоги до програмного продукту і поставлена задача.

Було розглянуто загальні засади моніторингу систем. Розглянуто push-based та pull-based моніторинг та здійснено порівняння. Було обґрунтовано, чому в даній роботі застосовується pull-based моніторинг. Описані типи застосунків для моніторингу. Наведено посилання на профільну літературу для подальшого вивчення даних питань. Здійснено опис функціональних вимог, які є стандартними для застосунків, подібних до того, що проєктується.

У цьому розділі також подано опис програмних архітектур, проведено аналіз їх переваг та недоліків і зроблено вибір на користь однієї з них. Зроблено вибір та порівняння наявних рішень. Описані інструменти, за допомогою яких була проведена розробка.

Створена діаграма способів використання, кожен із них детально описаний. Розроблені функціональні вимоги та матриця трасування.

Поставлена задача для даної роботи.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для опису бізнес процесу програмного забезпечення використовується BPMN модель (рисунок 2.1).

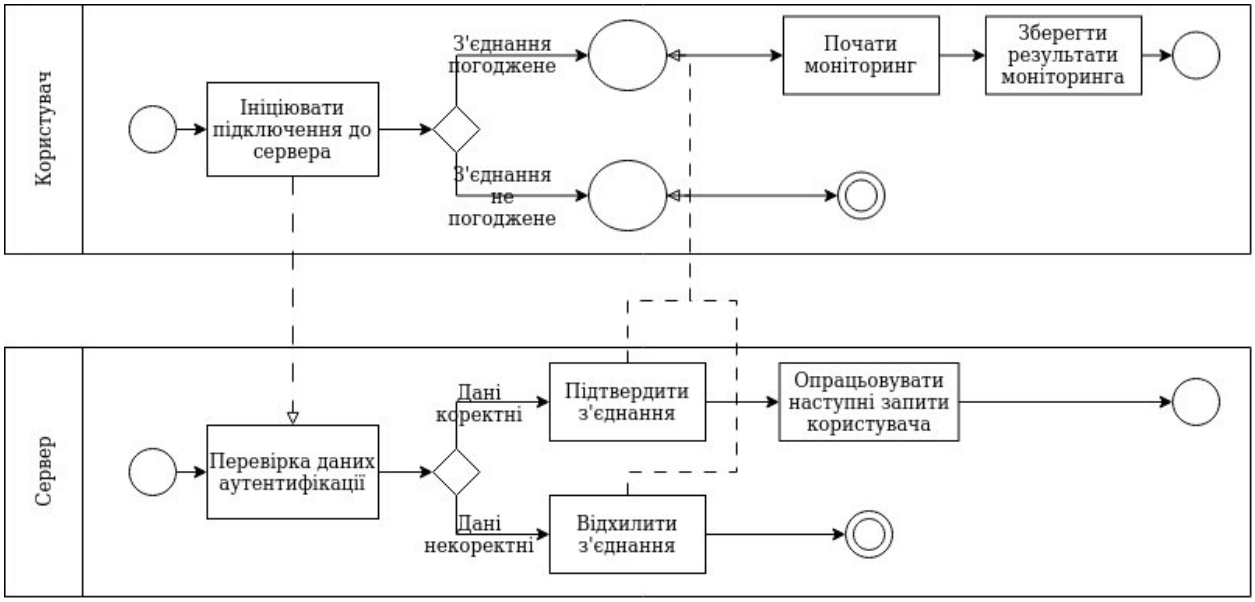


Рисунок 2.1 – Схема бізнес-процесу

Опис послідовності процесів при додаванні сервера та подальшому його моніторингу:

– користувач заповнює необхідні дані для підключення (IP адреса, порт, ім’я користувача, приватний ключ чи пароль);

– дані надходять на сервер, відбувається перевірка їх коректності;

– користувач отримує сповіщення про успішність аутентифікації у вигляді візуального індикатора;

– користувач переглядає статистику використання сервером ресурсів, починає моніторинг у фоні та виконує інші дії, які йому потрібні;

– користувач зберігає дані моніторингової сесії.

2.2 Архітектура програмного забезпечення

Як уже зазначалося, програма, що розробляється, є монолітом. При цьому, вона розділена на декілька шарів, тобто має багатошарову архітектуру. Застосунок складається з таких шарів: користувацький інтерфейс (активності та фрагменти), адаптери (для моделювання відображення елементів у списках), шар логіки (сервіси), моделі, мапери (об’єкти, що перетворюють моделі та сутності в обох напрямках), сутності, шар доступу до даних. Взаємозв’язок між даними шарами можна побачити на архітектурній діаграмі, що зображена на рисунку 2.2.

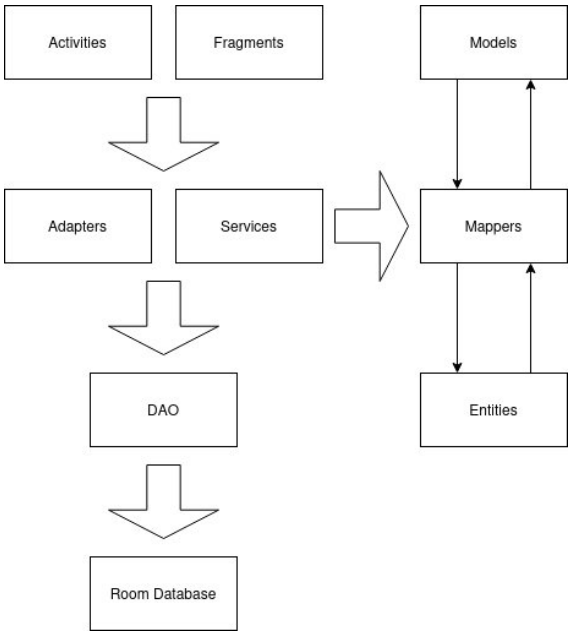


Рисунок 2.2 – Архітектура програми

Для реалізації користувацького інтерфейсу в Android використовуються активності (Activities) та фрагменти (Fragments). Кожна activity має свій layout – макет віджетів. Він допомагає в розташуванні віджетів (Views) на екрані. Фрагменти розташовуються в контейнері для фрагментів та змінюють одне одного в процесі навігації. Взаємодія між активностями та фрагментами показана на рисунку 2.3.

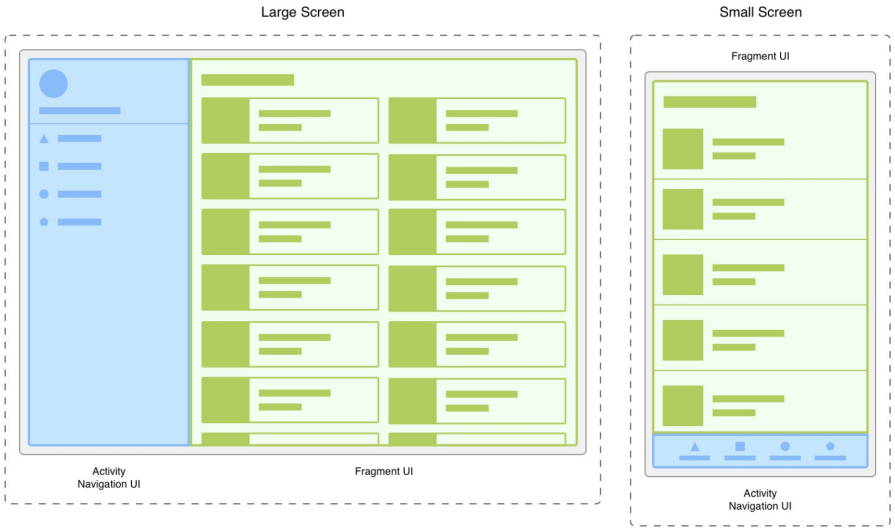


Рисунок 2.3 – Відображення відносин компонентів Activity, Fragment та Navigation в архітектурі типового Android додатка [6]

Адаптери використовуються для зв’язування джерела даних, яким є список об’єктів, з елементом графічного представлення, який відповідає за відображення цього списку об’єктів (наприклад, RecyclerView).

Шар логіки містить основний функціонал та алгоритми програми. Для збереження даних він викликає шар даних, і отримані дані віддає шару представлення.

Моделі – це шар, який містить структури даних, які використовуються в застосунку при відображенні даних. Він містить основні сутності та їхні атрибути.

Мапери можуть перетворювати модель одного типу в сутність того ж типу, і навпаки.

Сутності – це відповідники для моделей, але вони містять лише ті атрибути, які повинні зберігатися в таблицях бази даних, і не містять додаткових атрибутів, які можуть знадобитися при відображенні.

Рівень доступу до даних надає рівню логіки інструменти для збереження даних у джерелі даних. Це реалізується за допомогою DAO (Data Access Objects).

2.3 Конструювання програмного забезпечення

Програма побудована з використанням однієї головної активності (MainActivity) та фрагментів, кожен з яких відповідає за окремий екран у застосунку. Кожен фрагмент, і так само активність, поділений на два файли – код логіки та графічний макет. Програміст пише файл з кодом мовою Java, а макет – мовою розмітки XML. XML файли компілюються в Java під час процесу збирання. Фрагмент звретаються до сервісів для виконання бізнес-логіки. Сервіси отримують дані у вигляді сутностей (Entity) від шару доступу до даних, за допомогою маперів перетворюють їх на моделі, які віддають до фрагментів. Шар доступу до даних містить класи, які відповідають архітектурі бібліотеки збереження даних Room [7]. Структуру пакетів даного проєкту можна побачити на рисунку 2.4.

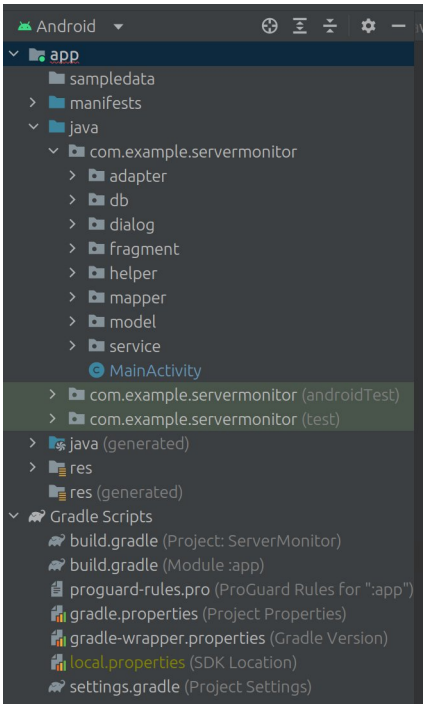


Рисунок 2.4 – Структура пакетів у даному проєкті

Головна активність зберігає в собі список із моделями серверів, який наповнюється з бази даних при першому запуску програми. Це зроблено для того, щоб моніторинг серверів працював у фоні на протязі всієї роботи програми, на якому б екрані користувач не знаходився. Для кожного наявного сервера створюється об’єкт класу SshSessionWorker, у якому

створюється окрема SSH сесія (об'єкт Session бібліотеки JSch) (рисунок 2.5) для кожного сервера. SSH сесія закріплюється за відповідним сервером і тримається під'єднаною на протязі всього часу роботи програми. Це дозволяє не перестворювати сесію при кожному запиті на моніторинг, що робить виконання команд досить швидким.

```
1 usage   Artem Vovchenko
private Boolean createSshSession() {
    String user = server.getUserName();
    String host = server.getHostIp();
    String password = server.getPassword();
    int port = server.getPort();
    try {
        if (sshKey.isPresent()) {
            String privateKey = sshKey.get().getKeyData();
            File tempFile = getTemporaryFile(privateKey);
            String privateKeyPath = tempFile.getAbsolutePath();
            jsch.addIdentity(privateKeyPath);
        }
        session = jsch.getSession(user, host, port);
        if (password != null) {
            session.setPassword(password);
        }
        session.setConfig("StrictHostKeyChecking", "no");
        session.setTimeout(10000);
        session.connect();
    } catch (JSchException e) {
        Log.d(TAG, msg: "JSch exception occurred while getting the session");
        return false;
    } catch (IOException e) {
        Log.d(TAG, msg: "IO exception occurred when working with SSH key and trying to establish ssh session");
        return false;
    }
    return true;
}
```

Рисунок 2.5 – Фрагмент коду, що створює SSH сесію за допомогою бібліотеки JSch (клас SshSessionWorker)

Кожен сервер отримує запит на моніторинг з рівним інтервалом часу (за замовчуванням, 5 секунд). Для виконання запиту SshSessionWorker створює об'єкт ChannelExec, який дозволяє виконати одну команду [8]. При цьому використовується попередньо створена SSH сесія. Команда складається з кількох внутрішніх команд, кожна з яких відповідає за збір конкретної метрики (використання пам'яті, диску тощо).

Для організації інтерактивної роботи з терміналом застосовується об'єкт ChannelShell бібліотеки JSch (створення цього об'єкту показане на рисунку 2.6). При переході на екран з терміналом створюється об'єкт класу SshShellSessionWorker, в якому здійснюється під'єднання до каналу ChannelShell, з'єднання з яким тримається під час всієї роботи з терміналом. Кожна введена команда надсилається до цього каналу, і в окремому потоці

відбувається зчитування виводу, що приходить від сервера. Оскільки канал перевикористовується, взаємодія з терміналом є досить швидкою.

Рисунок 2.6 – Фрагмент коду, що створює ChannelShell для подальшого виконання команд в інтерактивному режимі (клас SshShellSessionWorker)

У цьому застосунку часто використовуються окремі потоки для виконання обчислень (рисунок 2.7). Це дозволяє не зупиняти оновлення графічного інтерфейсу, і при цьому виконувати фонову роботу. Для фонового збору метрик з заданим часовим інтервалом використовується метод scheduleAtFixedRate класу ScheduledExecutorService, який дозволяє виконувати певний шматок коду, обгорнутий в об’єкт Runnable, постійно з певною частотою. Також багато фрагментів звертаються до бази даних. Це не варто робити на головному потоці, на якому працює графічний інтерфейс. Тому для таких потреб створюється об’єкт Thread, що являє собою новий потік. Він також приймає Runnable, який містить шматок коду, який потрібно виконати в потоці. Зазвичай, після того, як дані отримано з бази даних, потрібно оновити графічний інтерфейс. Для цього в кінці виконання потоку, який звертається до бази даних, додається виклик методу runOnUiThread класу Activity. Цей метод дозволяє повернутися на головний потік та оновити інтерфейс, відобразивши дані, які було щойно отримано. Такий підхід дозволяє розробляти динамічний інтерфейс, який може відображати користувачу повідомлення про завантаження, коли дані ще не надійшли, а потім відображати самі дані.

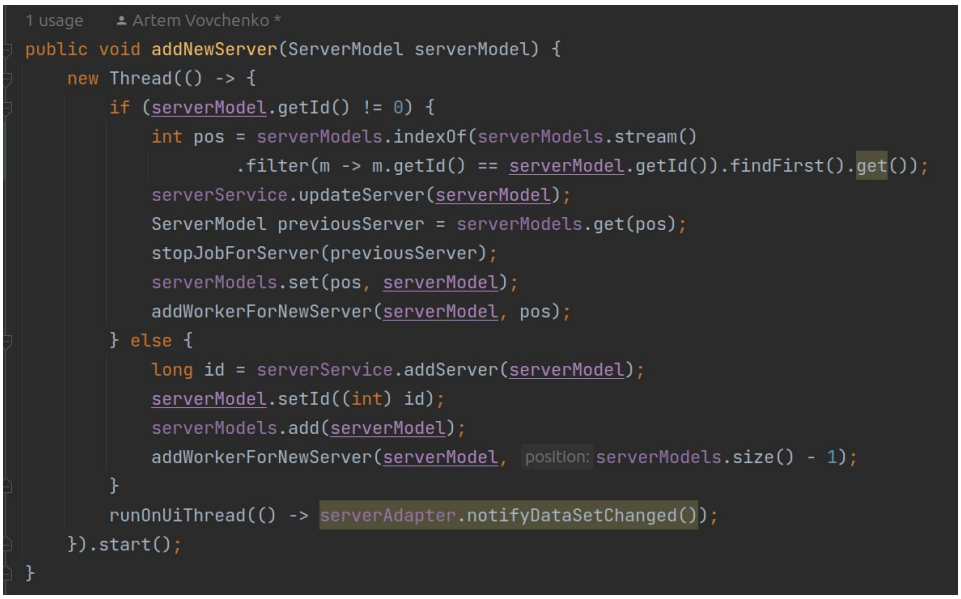


Рисунок 2.7 – Приклад коду з використанням окремого потоку та виконанням дій на графічному потоці (функція для додавання або редагування сервера, клас MainActivity). Дії, що стосуються роботи з базою даних, запущені в окремому потоці, і одразу після цих дій на графічному потоці запускається оновлення відображення списку серверів

Під час роботи з файлами користувач може копіювати файли між сервером та локальним сховищем та в іншому напрямку. Щоб перенести файл із сервера на сервер, потрібно спочатку скопіювати його в локальне сховище, а потім – на інший сервер. У якості локального сховища використовується приватна директорія застосунку (Documents), яка не є видимою для інших програм. Для файлових операцій на сервері використовується об’єкт ChannelSftp бібліотеки JSch. Логіка цих операцій прописана в класі SshShellSessionWorker. ChannelSftp містить зручні та корисні методи такі, як put для копіювання файлу на сервер, get для копіювання файлу з сервера на локальний пристрій, ls для отримання списку файлів у поточній директорії, cd для зміни поточної директорії тощо. Для копіювання файлів реалізовано відстеження прогресу завантаження (рисунок 2.8). Для реалізації цього, метод get класу ChannelSftp, крім файлових шляхів до джерела та призначення, приймає ще об’єкт, який наслідує SftpProgressMonitor. ChannelSftp передає цьому об’єкту дані про поточний

стан прогресу завантаження. Зміни в цьому об’єкті можна моніторити в окремому потоці та відповідно оновлювати графічний інтерфейс. Реалізація копіювання файлу показана на рисунку 2.9.

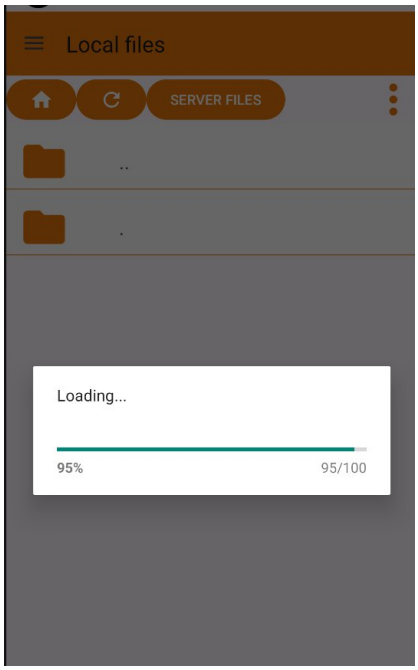


Рисунок 2.8 – Вигляд вікна, що відображає прогрес завантаження файлу

```
2 usages   Artem Vovchenko
public List<Object> copyFromServer(String remotePath, String remoteName, String localDirectory) {
    ArrayList<Object> results = new ArrayList<>();
    if (channelSftp == null) {
        try {
        channelSftp = (ChannelSftp) session.openChannel( type: "sftp");
        channelSftp.connect();
        } catch (JSchException e) {
            results.add(false);
            return results;
        }
    }
    FileLoadingProgressMonitor monitor = new FileLoadingProgressMonitor();
    new Thread(() -> {
        try {
            channelSftp.get(remotePath,  dst: localDirectory + "/" + remoteName, monitor);
        } catch (SftpException e) {}
    }).start();
    results.add(true);
    results.add(monitor);
    return results;
}
```

Рисунок 2.9 – Функція для копіювання файлу з сервера на локальний пристрій. FileLoadingProgressMonitor – це реалізація SftpProgressMonitor

Для відображення списків використовується RecyclerView. Цей клас схожий на ListView, але має кращу продуктивність, тому що повторно використовує об’єкти View, які перестають бути видимими на екрані під час

прокручування екрана користувачем [9]. Щоб знати, де брати дані та як саме відображати кожен елемент, RecyclerView потребує об’єкт клсу RecyclerView.Adapter. У собі адаптер містить масив об’єктів, а також логіку по відображенню даних, збережених в об’єктах, у відповідних View на екрані. Також в адаптері вказаний графічний макет. Він дозволяє специфічно відображати об’єкт. Наприклад, на екрані зі списком серверів відображені останні метрики, а також графічні малюнки для кращого сприйняття інформації. Клас-адаптер містить внутрішній клас, який наслідується від RecyclerView.ViewHolder. Цей клас, по суті, моделює один елемент зі списку, і містить View, які потрібні для встановлення даних одного об’єкта зі списку та відображення його на екрані. Метод адаптера onBindViewHolder отримує номер елемента в списку та саму модель, і заповнює View даними.

Графічні макети містять контейнери для об’єктів View та самі View. Контейнер (ConstraintLayout, RelativeLayout, FrameLayout та ін.) зберігає в собі View та визначає спосіб їх розміщення. Для того, щоб View можна було маніпулювати в коді, йому призначається ідентифікатор. Щоб його знайти в коді, потрібно передати ідентифікатор в метод findViewById. Щоб не викликати цей метод для кожного View, у цій роботі використано ViewBinding. Ця функція була ввімкнена конфігураційному файлі build.gradle для основного модуля програми. При цьому для кожного фрагмента генерується клас, в якому можна знайти кожен View відповідного макета. Приклад декларативного коду графічного макету наведено на рисунку 2.10.

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".fragment.ServersFragment">
    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_height="match_parent"
        android:layout_width="match_parent">
        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/rvServers"
            android:layout_width="0dp"
            android:layout_height="0dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />
        <com.google.android.material.floatingactionbutton.FloatingActionButton
            android:id="@+id/fabAddServer"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom|end"
            android:layout_margin="16dp"
            android:src="@drawable/baseline_add_24"
            app:backgroundTint="@color/main_color"
            app:elevation="6dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:circularflow_radiusInDP="1000dp"
            android:background="@drawable/fab_circle" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</FrameLayout>
```

Рисунок 2.10 – Приклад графічного макету (fragment_server.xml)

Навігація налаштована з використанням компонента навігації. Даний застосунок розроблений так, що макет головної активності містить контейнер для фрагментів (FragmentContainerView). Перший фрагмент, який завантажується при відкритті програми – ServersFragment. Надалі при навігації відбувається зміна фрагмента в контейнері. Для реалізації потрібно створити макет навігації (рисунок 2.11). Він містить опис існуючих фрагментів та можливих переходів між ними.

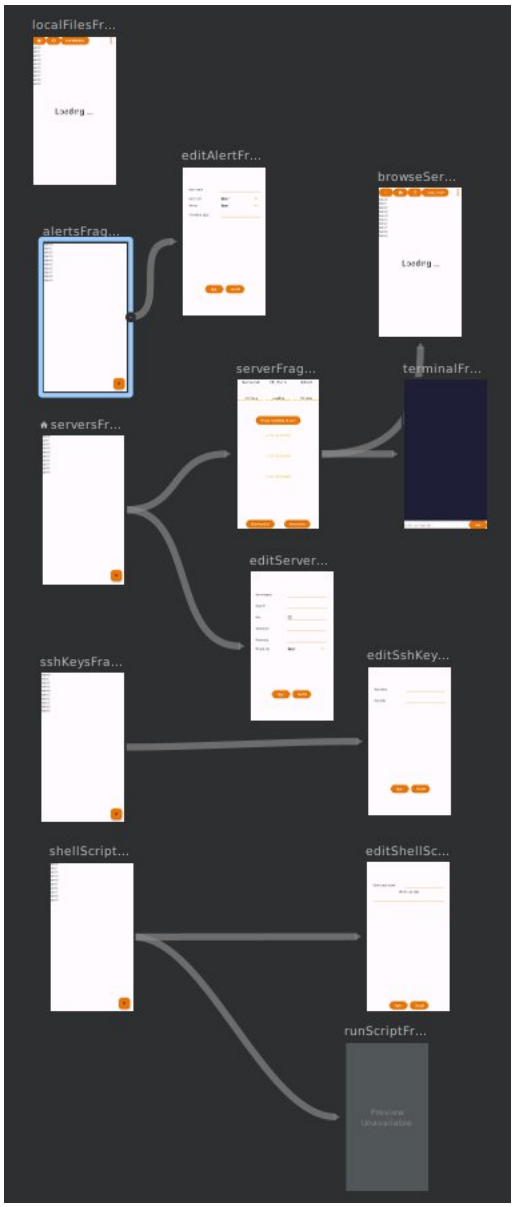


Рисунок 2.11 – Макет навігації (nav_graph.xml)

Перехід між фрагментами можна налаштувати, наприклад, при натисканні на кнопку (рисунок 2.12). Для виконання переходу потрібно знайти об’єкт NavController та викликати метод navigate [10]. При цьому можна передати об’єкт Bundle, в який можна помістити дані, які треба передати на інший фрагмент.

```
holder.itemView.setOnClickListener((v) -> {
    NavController controller = Navigation.findNavController(v);
    Bundle args = new Bundle();
    args.putParcelable("shellScriptModel", shellScripts.get(position));
    controller.navigate(R.id.action_shellScriptsFragment_to_runScriptFragment, args);
});
```

Рисунок 2.12 – Приклад коду навігації

Для відображення графіків застосовано бібліотеку MPAndroidChart [11]. Вона містить View, які відображають графіки (наприклад, лінійний або PieChart). Механізм підключення бібліотеки показано на рисунку 2.13.

```
implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'
```

Рисунок 2.13 – Підключення бібліотеки MPAndroidChart у файлі build.gradle головного модуля програми

Більшість об’єктів у програмі відгукуються на довге натискання, і при цьому відкривається контекстне меню. Для його реалізації у класі фрагмента перевизначаються методи onCreateContextMenu та onContextItemSelected. Для визначення елементів, які входять до меню, описується макет меню.

В якості системи управління базами даних використовується SQLite. База даних призначена для зберігання даних про сервери, моніторингові сесії та їх записи, скрипти тощо. Для роботи з базою даних застосовано фреймворк Room, який спрощує роботу з сутностями, значно зменшує необхідність написання власних SQL запитів. Для кожної сутності (Entity) в базі даних створюється окрема таблиця з відповідними атрибутами. Для роботи з об’єктами сутності створюється клас доступу до даних (data access object – DAO), який містить методи для додавання, видалення сутності, отримання її за ідентифікатором тощо. Також в класі Dao можна прописати власні методи, які будуть використовувати окремо складені SQL запити. Наприклад, це потрібно для отримання саме тих записів, які пов’язані з певним іншим записом в іншій таблиці бази даних. Щоб уникнути використання об’єктів Dao в коді класів фрагментів та полегшити отримання даних, були створені сервіси для кожної сутності (ServerService, SshKeyService, MonitoringRecordService тощо). Сервіси взаємодіють з об’єктами Dao, отримують від них сутності (Entity) та перетворюють їх на моделі, які краще підходять для відображення даних на екрані. Для кожної Entity є відповідна їй Model. Це зроблено тому, що іноді бажано зберігати в

сутності додаткові дані, які корисні для відображення користувачу, але які не потрібно зберігати до бази даних. Також, наприклад, SQLite не має типу даних для дати, тому дата зберігається в ній як число мілісекунд, і для зручної роботи з датою це число потрібно конвертувати в об'єкт дати. Тому іноді Entity та Model відрізняються, і для перетворення об'єкта з одного типу на інший створені відповідні перетворювачі (Mappers). Також моделі в цій програмі реалізують інтерфейс Parcelable. Це потрібно для того, щоб моделі можна було передавати між фрагментами, коли користувач переміщується від одного екрана до іншого. Приклад реалізації одного з класів DAO наведений на рисунку 2.14.

```
9 usages   1 implementation   Artem Vovchenko
@Dao
public interface ServerDao {
    1 usage   1 implementation   Artem Vovchenko
    @Insert
    long addServer(ServerEntity serverEntity);

    1 usage   1 implementation   Artem Vovchenko
    @Update
    void updateServer(ServerEntity serverEntity);

    1 usage   1 implementation   Artem Vovchenko
    @Delete
    void deleteServer(ServerEntity serverEntity);
    1 usage   1 implementation   Artem Vovchenko
    @Query("delete from servers where id == :id")
    void deleteServerById(int id);

    1 usage   1 implementation   Artem Vovchenko
    @Query("select * from servers")
    List<ServerEntity> getAllServers();

    1 implementation   Artem Vovchenko
    @Query("select * from servers where id == :id")
    ServerEntity getServer(int id);
}
```

Рисунок 2.14 – Один із класів DAO (ServerDao)

База даних містить такі сутності: Alert, MonitoringRecord, MonitoringSession, Server, ShellScript, SshKey.

Alert – сутність, яка позначає сповіщення. Містить тип сповіщення (про використання пам'яті чи процесора чи диску або швидкість зчитування чи запису даних на диску або швидкість завантаження чи вивантаження даних

мережі) та порогове значення (число), при якому сповіщення повинно активуватися.

MonitoringRecord відображає зібрані метрики в конкретний момент часу з конкретного сервера. Містить позначку часу та інформації про використання пам’яті та повний об’єм пам’яті, використання процесора, використання диску та його повний об’єм, а також дані про завантаження вводу-виводу мережі та диску. Відноситься до певної моніторингової сесії, тому містить її ідентифікатор.

MonitoringSession – відображає моніторингову сесію. Містить момент початку та завершення моніторингу.

Server – позначає сервер. Містить назву, IP адресу, порт, ім’я користувача, пароль, ID приватного ключа (якщо використовується).

ShellScript – позначає створений користувачем скрипт. Містить його назву та текстовий вміст.

SshKey – позначає приватний SSH ключ. Містить його назву та його дані (вміст).

Опис таблиць бази даних для відповідних сутностей наведений у таблицях 2.1 - 2.6.

Таблиця 2.1 – опис таблиці Alert

Назва колонки	Тип даних	Опис
id	integer	Унікальний ідентифікатор сповіщення
tame	text	Назва сповіщення
alertType	integer	Номер типу сповіщення (кожне числове значення типу співставляється з відповідним типом у коді програми)
thresholdValue	integer	Порогове значення метрики, при переході якого сповіщення спрацьовує
serverId	integer	Ідентифікатор сервера, для моніторингу якого створюється дане сповіщення

Таблиця 2.2 – опис таблиці MonitoringRecord

Назва колонки	Тип даних	Опис
id	integer	Унікальний ідентифікатор моніторингового запису
monitoringSessionId	integer	Ідентифікатор моніторингової сесії, якій належить даний запис
timeRecorded	integer	Час збирання інформації (позначка часу, яка в програмі конвертується в дату та час)
memoryUsedMb	integer	Дані про використання пам’яті (в мегабайтах)
memoryTotalMb	integer	Дані про повний обсяг пам’яті (в мегабайтах)
cpuUsagePercent	integer	Дані про використання процесорного ресурсу (у відсотках)
diskUsedMb	integer	Дані про використання дискового простору (в мегабайтах)
diskTotalMb	integer	Дані про повний обсяг дискового простору (в мегабайтах)

Таблиця 2.3 – опис таблиці MonitoringSession

Назва колонки	Тип даних	Опис
id	integer	Унікальний ідентифікатор моніторингової сесії
name	text	Назва моніторингової сесії
dateStarted	integer	Час початку запису моніторингової сесії (позначка часу)
dateEnded	integer	Час завершення запису моніторингової сесії (позначка часу)
serverId	integer	Ідентифікатор сервера, моніторинг якого проводився

Таблиця 2.4 – опис таблиці Server

Назва колонки	Тип даних	Опис
id	integer	Унікальний ідентифікатор сервера
name	text	Назва сервера
hostIp	text	IP адреса сервера
port	integer	Номер порту, до якого виконується SSH підключення (зазвичай, 22)
username	text	Ім’я користувача, з яким буде здійснене підключення до сервера
password	text	Пароль користувача (якщо потрібно)
privateKeyId	integer	Ідентифікатор приватного SSH ключа

Таблиця 2.5 – опис таблиці ShellScript

Назва колонки	Тип даних	Опис
id	integer	Унікальний ідентифікатор скрипта
name	text	Назва скрипта
scriptData	text	Текстовий вміст скрипта

Таблиця 2.6 – опис таблиці SshKey

Назва колонки	Тип даних	Опис
id	integer	Унікальний ідентифікатор приватного SSH ключа
name	text	Назва приватного SSH ключа
keyData	text	Текстовий вміст приватного SSH ключа

На рисунку 2.15 наведено ER діаграму. Вона показує основні сутності в базі даних та зв’язки між ними.

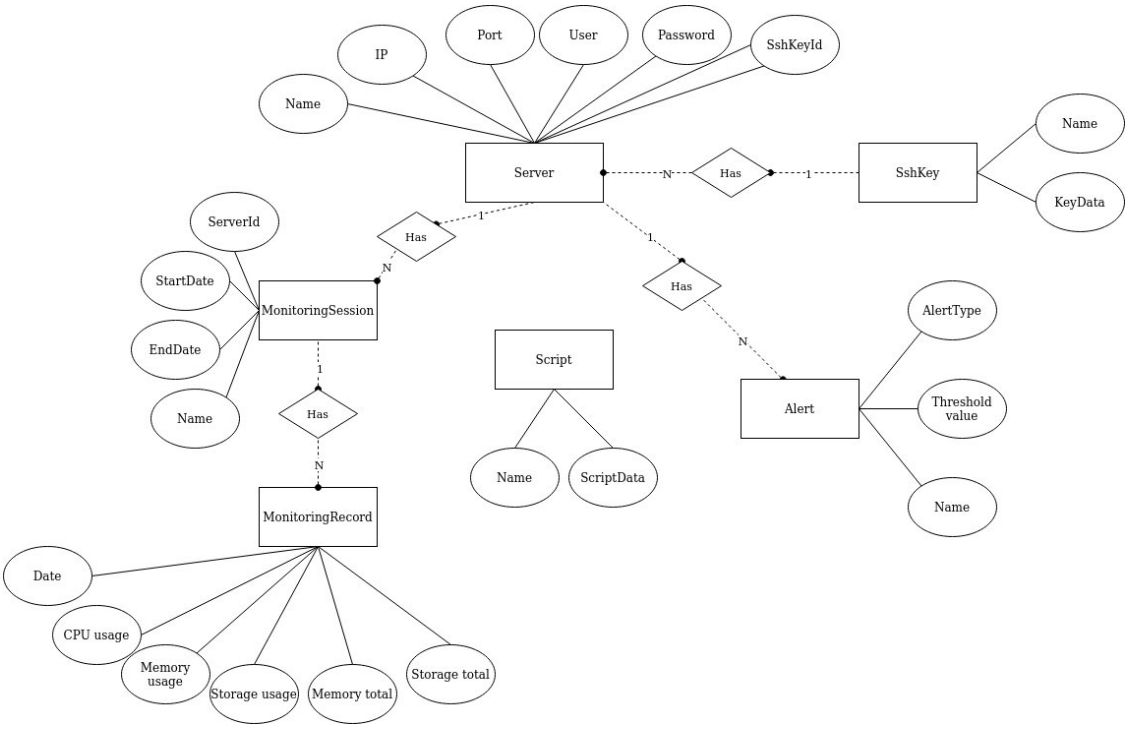


Рисунок 2.15 – ER діаграма сутностей

2.4 Аналіз безпеки даних

Безпека даних є важливою складовою розробки будь-якого застосунку, а особливого того, який зберігає приватні дані користувачів.

Поточний застосунок зберігає такі приватні дані: приватні ssh-ключі та паролі від серверів. Також, уся комунікація між застосунком та серверами повинна буде захищена. Для захисту комунікації використовується протокол ssh (Secure Shell). Це є стандарт індустрії для комунікації з віддаленими системами.

Паролі та приватні ключі зберігаються в базі даних у приватному сховищі, яке видно лише цьому застосунку.

Висновки до розділу

У цьому розділі були продемонстровані аспекти розробки програми. Показана модель бізнес-процесів. Описана архітектура програмного

забезпечення, зображена діаграма відношень сутностей та проаналізовані вимоги щодо безпеки застосунку.

Архітектура програми складається з декількох шарів (шари представлення, логіки, доступу до даних тощо). Кожен шар містить набір класів. У цьому розділі описана відповідальність основних класів та показані фрагменти коду, які ілюструють важливі технічні концепції. Дані зберігаються в таблицях бази даних. У цьому розділі показано, як програма працює з даними. Описані сутності бази даних та їхні атрибути.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Інструменти для тестування програмного коду поділяються на два типи: статичні та динамічні.

Статичні аналізатори не вимагають запуск програми для своєї роботи. Вони лише перевіряють текст коду на наявність вразливостей, потенційних помилок та використання рекомендованих практик розробки програмного забезпечення. Вони можуть дати корисні рекомендації для покращення якості коду, його продуктивності, усунення помилок, використання рекомендованих практик, які стосуються безпеки.

Інструменти динамічного аналізу перевіряють роботу коду в той час, коли він виконується і відстежує потрібні характеристики. Динамічний аналіз дозволяє виявити проблеми, які не можна знайти шляхом статичного аналізу. Його можна використовувати для перевірки коректності роботи функцій та компонентів з різними вхідними даними, споживання пам’яті в реальному часі, наявності проблем при багатопоточному чи конкурентному доступі до ресурсів, ступеня покриття тестами тощо.

Для статичного аналізу Android програм можна використовувати різні інструменти, наприклад, PMD, SonarQube, Android Lint.

PMD – аналізатор з відкритим вихідним кодом, який може працювати з різними мовами програмування та може відстежувати проблеми зі стилем коду, дублікацією частин коду, порушенням рекомендованих практик. Він дозволяє створення власних правил для аналізу. Його можна інтегрувати з популярними build системами.

SonarQube – популярна платформа з відкритим вихідним кодом для аналізу коду. Надає докладні доповіді з потрібними метриками та дозволяє відстежувати їхню зміну в часі.

Android Lint – статичний аналізатор коду, вбудований в Android SDK. Може генерувати HTML чи XML доповіді з метриками коректності, продуктивності, безпечності, доступності. Цей інструмент можна запускати з командного рядку або інтегрувати його з build системою.

Для даного проєкту був обраний інструмент SonarQube (https://sonarcloud.io). Він надає зручну інтеграцію з GitHub. Після надання доступу до свого GitHub профілю, інструмент дозволяє зручно проаналізувати код в репозиторії та відобразити результати в графічному веб-інтерфейсі. Результати сканування коду даного проєкту показано на рисунку 3.1. Можна побачити, що загалом код проходить перевірку, проте має 1 потенційну проблему з безпекою та 9 потенційних помилок в коді.

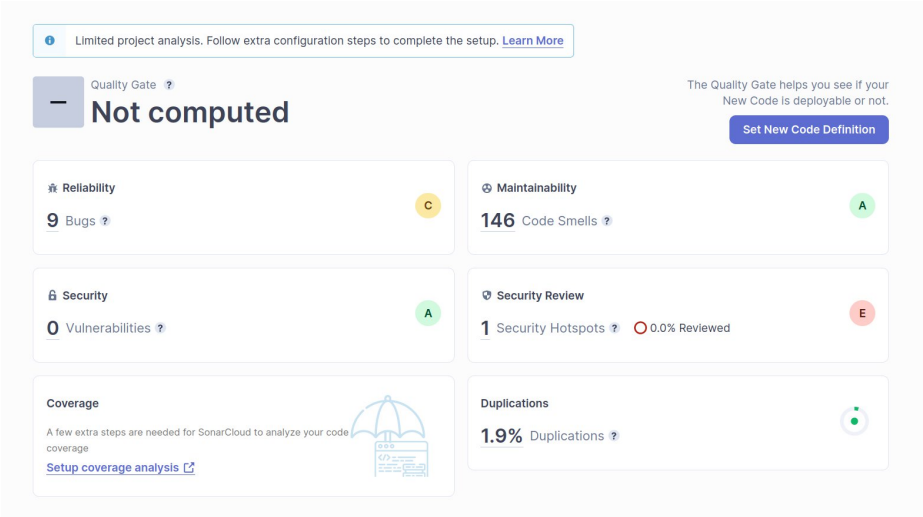


Рисунок 3.1 – Результат аналізу коду даної програми інструментом SonarQube

Нефункціональні вимоги, які були поставлені раніше, досягнуто. Застосунок продуктивно працює, відповідає базовим критеріям безпеки. Програмою зручно користуватися, інтерфейс є інтуїтивно зрозумілим.

3.2 Опис процесів тестування

Тестування може проводитися на різних рівнях. В залежності від цих рівнів, різні види тестів часто зображають на піраміді тестування (рисунок 3.2).

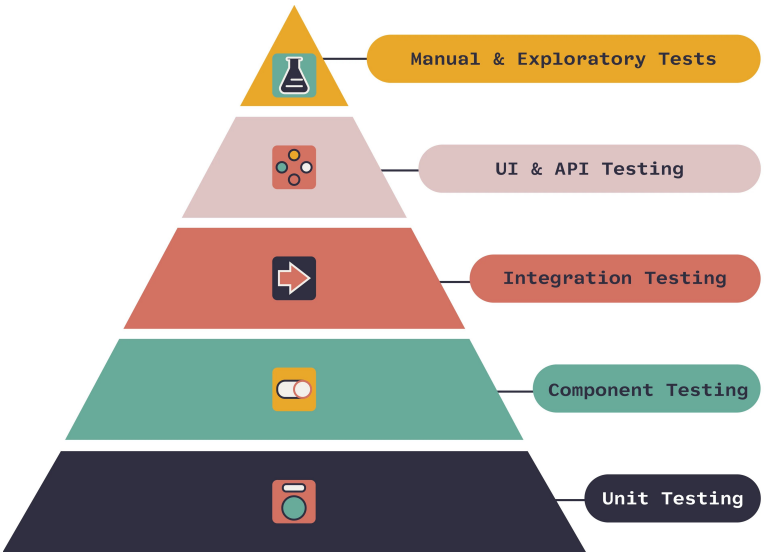


Рисунок 3.2 – Піраміда тестування [12]

Unit тести перевіряють коректність роботи окремих частин коду (методів, функцій, класів). Інтеграційні тести перевіряють взаємодію декількох компонентів складної системи. Зазвичай, компоненти взаємодіють через API інтерфейс. End to end тестування перевіряє коректність роботи системи вцілому з симуляцією введення реальних даних від користувача.

У даній роботі було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 3.1 – 3.30.

Таблиця 3.1 – Тест 1.1

Тест	Додавання сервера
Модуль	Робота з серверами
Номер тесту	1.1
Початковий стан системи	Користувач знаходиться на головному екрані
Вхідні дані	Ім’я, IP-адреса, порт сервера, ім’я користувача, пароль (за необхідності), приватний ключ (за необхідності)
Опис проведення тесту	Користувач натискає на кнопку “+”, вводить відповідні дані у потрібні поля. Натискає кнопку “Apply”.

Продовження таблиці 3.1

Очікуваний результат	Сервер успішно додається, користувач перенаправляється на головний екран та при успішному з’єднанні бачить дані моніторингу.
Фактичний результат	Додавання сервера проводиться успішно, користувач перенаправляється на екран із серверами та бачить сервер, що додався. При успішному з’єднанні починають відображатися дані моніторингу.

Таблиця 3.2 – Тест 1.2

Тест	Додавання SSH ключа
Модуль	Робота з SSH ключами
Номер тесту	1.2
Початковий стан системи	Користувач знаходиться на екрані з SSH ключами
Вхідні дані	Назва ключа та сам SSH ключ у вигляді тексту
Опис проведення тесту	На головному екрані користувач натискає на кнопку Меню (три риски в лівому верхньому кутку) та обирає вкладку SSH keys. Користувач натискає на кнопку “+”, вводить відповідні дані у потрібні поля. Натискає кнопку “Apply”.
Очікуваний результат	Додавання SSH ключа проходить успішно. Користувач повертається на екран з ключами і бачить новий ключ
Фактичний результат	Додавання SSH ключа проходить успішно. Користувач повертається на екран з ключами і бачить новий ключ

Таблиця 3.3 – Тест 1.3

Тест	Перегляд статусу сервера та моніторингових метрик по кожному серверу

Продовження таблиці 3.3

Модуль	Робота з серверами
Номер тесту	1.3
Початковий стан системи	Користувач знаходиться на екрані з серверами (головний екран)
Вхідні дані	-
Опис проведення тесту	Користувач відкриває головний екран, на якому відображений список усіх доданих серверів, і спостерігає за індикатором та метриками
Очікуваний результат	При успішному підключенні до серверів, індикатор світиться зеленим кольором та метрики оновлюються кожні декілька секунд.
Фактичний результат	При успішному з’єднання з серверами, біля кожного сервера індикатор світиться зеленим кольором. Біля кожного сервера зображений список основних метрик, і вони оновлюються з однаковим інтервалом у декілька секунд відповідно до реальних показників кожного з серверів.

Таблиця 3.4 – Тест 1.4

Тест	Тривалий моніторинг сервера
Модуль	Робота з серверами
Номер тесту	1.4
Початковий стан системи	Користувач знаходиться на екрані певного сервера
Вхідні дані	-

Продовження таблиці 3.4

Опис проведення тесту	На головному екрані користувач натискає на один із серверів, метрики якого він бажає моніторити. На екрані, що відобразився, натискає кнопку Start monitoring session. По завершенню моніторингу натискає на кнопку Stop monitoring session.
Очікуваний результат	Моніторингові дані відображаються на графіках в реальному часі.
Фактичний результат	Починається моніторинг сервера. Кожні декілька секунд з’являються нові значення метрик. Вони відображаються на лінійних графіках на екрані з даним сервером. У процесі дані зберігаються до бази даних. При натисканні кнопки Stop monitoring session, моніторинг зупиняється. Коли моніторинг працює, користувач може вийти на головний екран або вийти з застосунку, не згортаючи його, на головний екран.

Таблиця 3.5 – Тест 1.5

Тест	Підключення до сервера по SSH для роботи з інтерактивним терміналом
Модуль	Робота з серверами
Номер тесту	1.5
Початковий стан системи	Користувач знаходиться на екрані певного сервера
Вхідні дані	-

Продовження таблиці 3.5

Опис проведення тесту	На головному екрані користувач натискає на один із серверів, до якого він хоче підключитися. На екрані, що відобразився, натискає кнопку Open terminal. На екрані, що з'явився, вводить бажані команди в рядок вводу. Натискає кнопку Run для виконання кожної з команд.
Очікуваний результат	Користувач може виконувати команди одна за одною (забезпечується інтерактивна сесія терміналу). Вивід відображається у відповідному полі.
Фактичний результат	При натисканні на кнопку Run, команда виконується на сервері, її результат з'являється у вікні для текстового виводу. Поле для вводу стає порожнім. Користувач може ввести наступну команду. Текст у полі для виводу прокручується до останнього рядка виводу, якщо у полі текст не вміщається у видимій площині.

Таблиця 3.6 – Тест 1.6

Тест	Редагування даних сервера
Модуль	Робота з серверами
Номер тесту	1.6
Початковий стан системи	Користувач знаходиться на головному екрані
Вхідні дані	Ім'я, IP-адреса, порт сервера, ім'я користувача, пароль (за необхідності), приватний ключ (за необхідності)
Опис проведення тесту	На головному екрані користувач натискає довгим тапом на потрібний сервер, з'являється контекстне меню. Користувач натискає кнопку Edit. Користувач змінює значення в потрібних полях і натискає кнопку Apply.

Продовження таблиці 3.6

Очікуваний результат	Редагування сервера проходить успішно
Фактичний результат	При переході на екран редагування сервера, усі попередні значення полів автоматично підставляються. Користувач змінює лише ті поля, які вважає за потрібне. Після натискання кнопки Apply користувач повертається на головний екран і одразу бачить сервер з оновленими даними. Підключення здійснюється повторно, що дозволяє виправити проблему підключення, якщо користувач вказав невірний пароль чи SSH ключ.

Таблиця 3.7 – Тест 1.7

Тест	Створення скрипта
Модуль	Робота зі скриптами
Номер тесту	1.7
Початковий стан системи	Користувач знаходиться на екрані зі скриптами
Вхідні дані	Назва скрипта, текст скрипта
Опис проведення тесту	На головному екрані користувач натискає на кнопку Меню та обирає вкладку Scripts. Користувач натискає на кнопку “+”, вводить відповідні дані у потрібні поля. Натискає кнопку “Apply”
Очікуваний результат	Додавання скрипта проходить успішно
Фактичний результат	Користувач повертається на екран зі скриптами і бачить доданий скрипт у списку зі скриптами.

Таблиця 3.8 – Тест 1.8

Тест	Виконання скрипта на одному чи декількох серверах
Модуль	Робота зі скриптами
Номер тесту	1.8
Початковий стан системи	Користувач знаходиться на екрані зі скриптами
Вхідні дані	-
Опис проведення тесту	На головному екрані користувач натискає на кнопку Меню та обирає вкладку Scripts. Користувач натискає на скрипт, який хоче виконати. На екрані, що відкрився, обирає сервери, на яких він хоче виконати скрипт (натискає чекбокси біля потрібних серверів). Натискає кнопку Run. Чекає, поки напис “Running ...” не зміниться на “View output” біля обраних серверів. Натискає на кнопку “View output” біля бажаного сервера і переглядає результати виконання скрипта.
Очікуваний результат	Скрипт виконується на всіх обраних серверах. Можливо побачити результат виконання для кожного сервера.
Фактичний результат	Після виконання скрипта результати можна переглянути, натиснувши кнопку “View output”.

Таблиця 3.9 – Тест 1.9

Тест	Виконання скрипта на одному чи декількох серверах
Модуль	Робота зі скриптами
Номер тесту	1.9
Початковий стан системи	Користувач знаходиться на екрані зі скриптами
Вхідні дані	-

Продовження таблиці 3.9

Опис проведення тесту	На головному екрані користувач натискає на кнопку Меню та обирає вкладку Scripts. Користувач натискає на скрипт, який хоче виконати. На екрані, що відкрився, обирає сервери, на яких він хоче виконати скрипт (натискає чекбокси біля потрібних серверів). Натискає кнопку Run. Чекає, поки напис “Running ...” не зміниться на “View output” біля обраних серверів. Натискає на кнопку “View output” біля бажаного сервера і переглядає результати виконання скрипта.
Очікуваний результат	Скрипт виконується на всіх обраних серверах. Можливо побачити результат виконання для кожного сервера.
Фактичний результат	Після виконання скрипта результати можна переглянути, натиснувши кнопку “View output”.

Таблиця 3.10 – Тест 1.10

Тест	Додавання сповіщення та перевірка його роботи
Модуль	Робота зі сповіщеннями
Номер тесту	1.10
Початковий стан системи	Користувач знаходиться на екрані зі сповіщеннями
Вхідні дані	Назва сповіщення, його тип, порогове значення метрики, сервер, до якого застосовується сповіщення
Опис проведення тесту	На головному екрані користувач натискає на кнопку Меню та обирає вкладку Alerts. Користувач натискає на кнопку “+”, вказує назву для сповіщенням, його тип, порогове значення метрики та сервер, що відстежується. Натискає Apply. Якщо значення метрики стало більше за порогове, користувач отримує push-сповіщення про це.

Продовження таблиці 3.10

Очікуваний результат	Сповіщення працює при перевищенні метрикою порогового значення.
Фактичний результат	Сповіщення працює при перевищенні метрикою порогового значення.

Таблиця 3.11 – Тест 1.11

Тест	Копіювання файлу з сервера на локальний пристрій
Модуль	Робота з сервером
Номер тесту	1.11
Початковий стан системи	Користувач знаходиться на сторінці сервера
Вхідні дані	Шлях до файлу, який потрібно скопіювати
Опис проведення тесту	Користувач переходить на сторінку певного сервера. Натиска кнопку “Browse files”. Обирає потрібний файл. Відкриває контекстне меню файла, натискає “Copy to local”. Виходить на головний екран. У головному меню відкриває вкладку “Local storage”. Перевіряє наявність скопійованого файлу.
Очікуваний результат	Файл успішно скопійовано.
Фактичний результат	Файл успішно скопійовано.

3.3 Опис контрольного прикладу

На початку роботи з програмою користувач опиняється на головному екрані (рисунок 3.3).

Рисунок 3.3 – Головний екран

Для додавання сервера, користувач натискає “+” та заповнює потрібні дані на екрані для додавання сервера (рисунок 3.4).

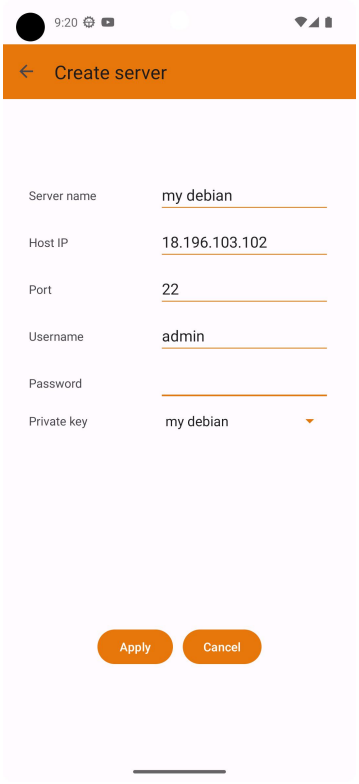


Рисунок 3.4 – Екран для додавання сервера

Для перегляду більш детальної інформації користувач обирає сервер та переходить на його сторінку (рисунок 3.5).

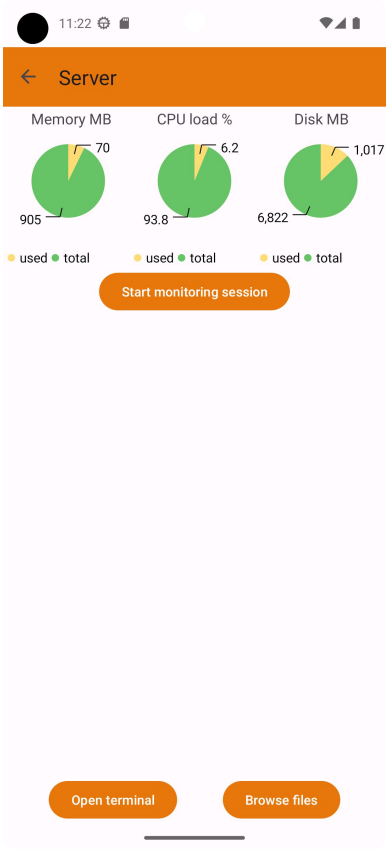


Рисунок 3.5 – Екран сервера з інформацією про нього

Щоб почати постійний моніторинг, користувач натискає “Start monitoring session”. З’являються графіки, що відображають зміну метрик з часом (рисунок 3.6).

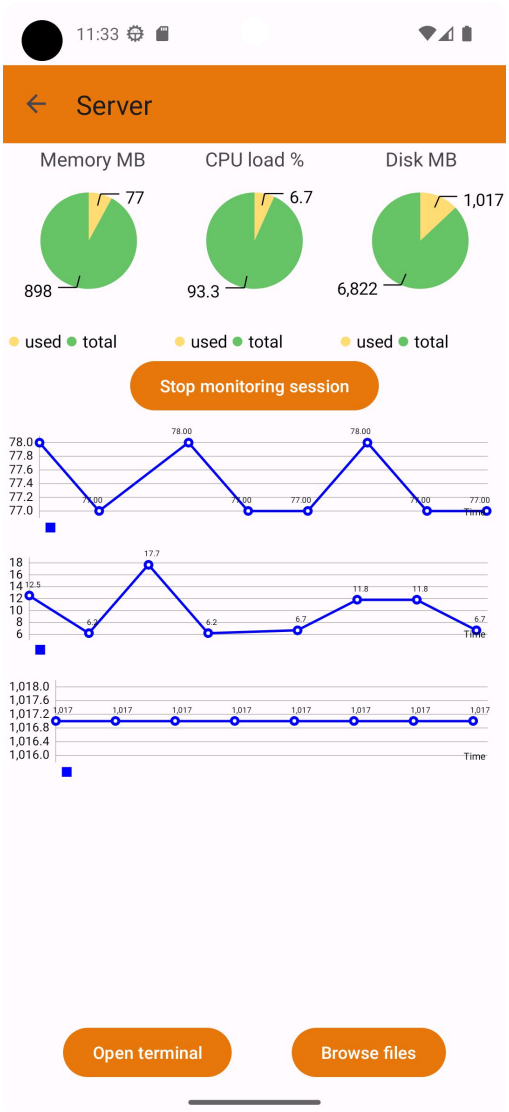


Рисунок 3.6 – Відображення моніторингу з графіками

Щоб зупинити моніторинг, потрібно натиснути Stop monitoring session.

Щоб розпочати інтерактивну SSH сесію, користувач натискає “Open terminal” і переходить на екран з терміналом (рисунок 3.7). Далі він вводить команди в рядок для вводу та натискає кнопку “Run”.

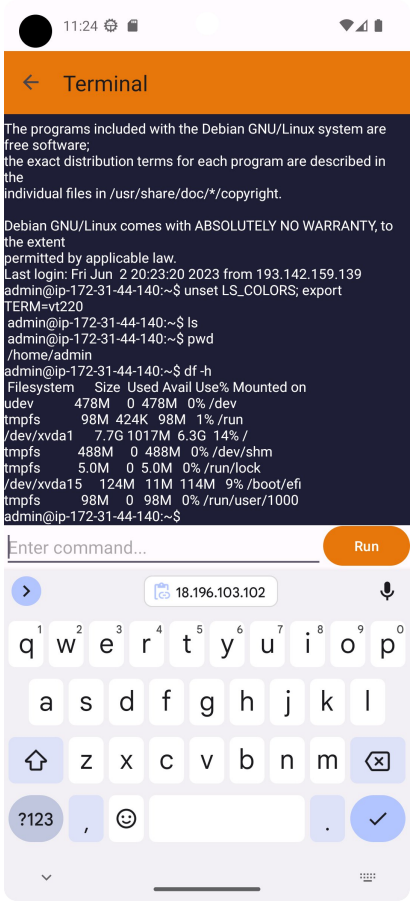


Рисунок 3.7 – Робота з терміналом

Для того щоб повернутися до сторінки сервера, користувач натискає кнопку Назад на панелі дій.

Щоб перейти до перегляду файлів, він натискає “Browse files”. На цьому екрані (рисунок 3.8) користувач може переміщатися між директоріями, натискаючи на рядок “..” (на директорію вище) або на рядок з назвою директорії. Також він може скористатися кнопкою “Root” для переміщення в корінь файлової системи та “Home” для переміщення в домашню директорію.

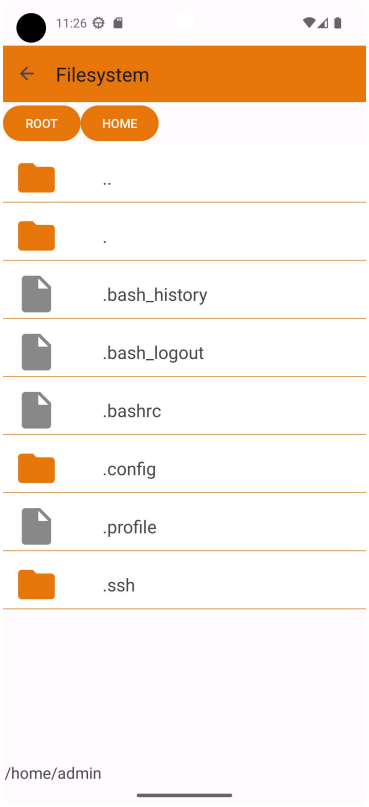


Рисунок 3.8 – Перегляд файлової системи сервера

Для створення сповіщення потрібно обрати вкладку Alerts у головному меню (рисунок 3.9).

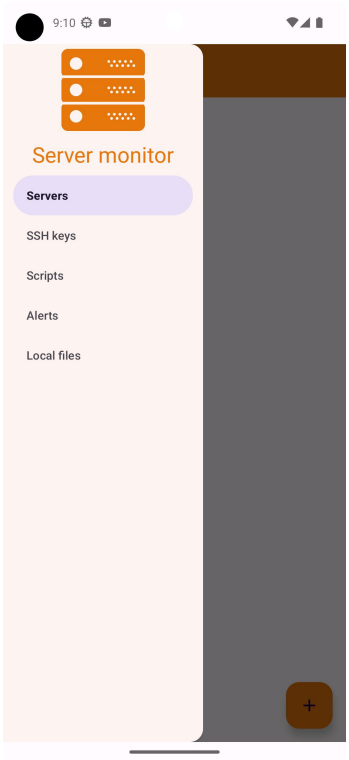


Рисунок 3.9 – Бокове меню навігації

Користувач вводить необхідні дані на екрані для створення сповіщення (рисунок 3.10).

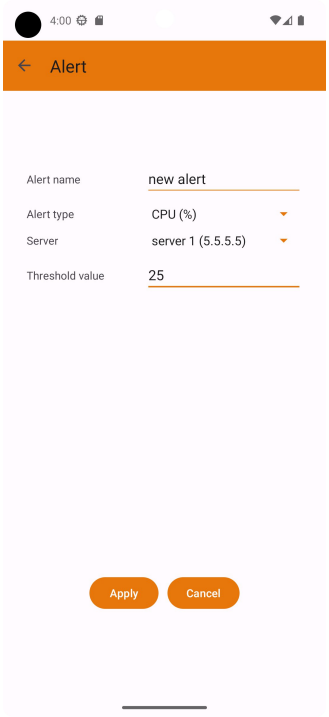


Рисунок 3.10 – Створення сповіщення

Для виконання дій з файлами на сервері користувач використовує контекстне меню.

Файли, що збережені в самому застосунку, можна переглянути на вкладці “Local files” у головному меню.

Висновки до розділу

У цьому розділі був проведений аналіз існуючих засобів для тестуванн ПЗ на платформі Android. Було обрано інструмент для статичного аналізу коду та зібрані метрики кодової бази даного проєкту за його допомогою. Було здійснене мануальне тестування застосунку згідно основних сценаріїв використання. Продемонстровано контрольний приклад використання програми з різними сценаріями.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для розгортання програм на платформі Android є два основних способи: розміщення програми в спеціалізованому магазині програм для Android або поширення програми у вигляді APK файлу, який користувач має встановити вручну.

Найпопулярнішим магазином програм для Android є Google Play. Є також альтернативи – F-Droid, Samsung GalaxyStore, APKMirror, Amazon Appstore, Huawei AppGallery та інші.

Для даного проєкту було обрано поширювати встановлюваний APK файл в репозиторії проєкту на GitHub в розділі для релізів. Ось кроки, необхідні для встановлення програми:

– перейти на сторінку https://github.com/artem-vovchenko01/android-server-monitoring;

– клацнути на посилання Releases;

– завантажити APK файл для останнього релізу.

Для створення APK файлу використовувалися вбудовані інструменти в Android Studio. Для створення файлу потрібно:

– в меню натиснути Build, Generate signed bundle or APK (рисунок 4.1);

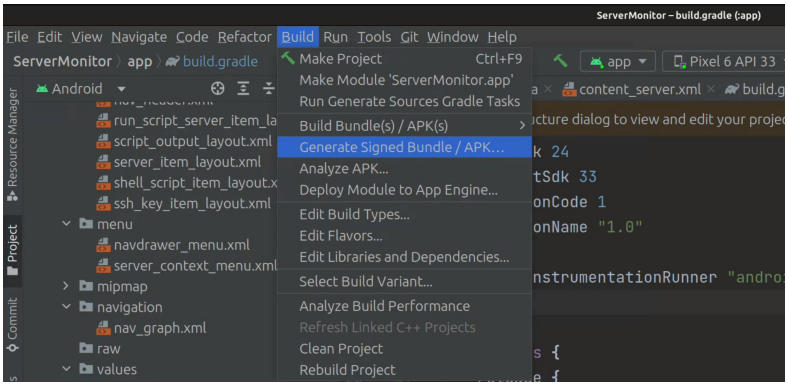


Рисунок 4.1 – Пункт меню для генерації APK файлу

– обрати APK (рисунок 4.2);

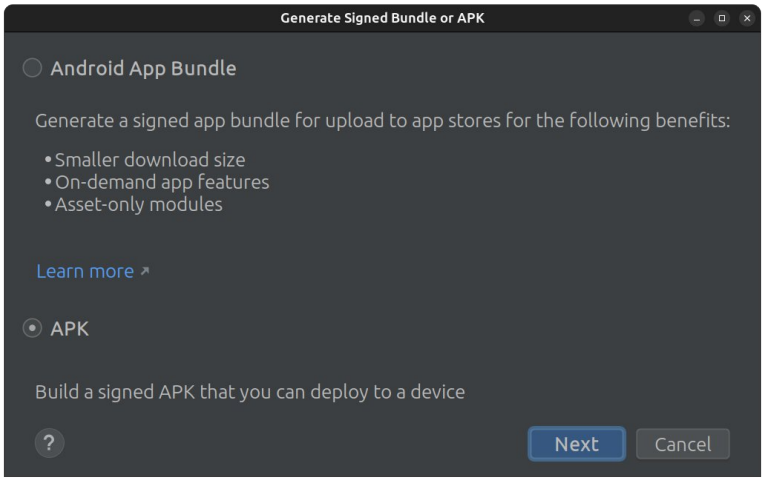


Рисунок 4.2 – Вибір формату готового артефакту

– залишити збережені дані про ключ для підписання APK або створити новий ключ (рисунок 4.3);

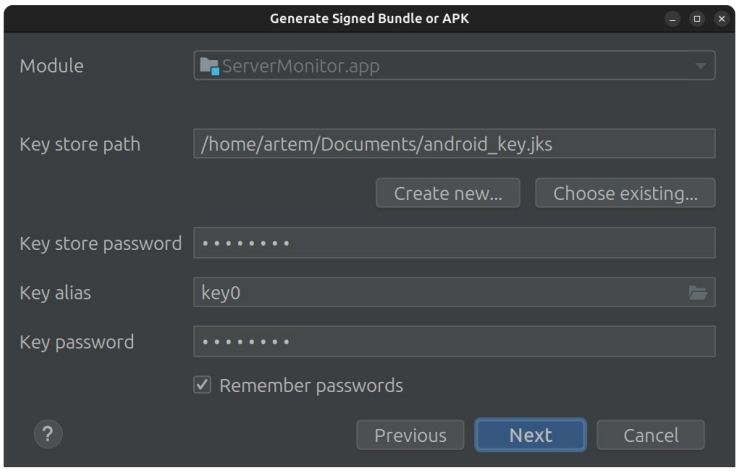


Рисунок 4.3 – Налаштування ключа

– обрати версію Release (рисунок 4.4);

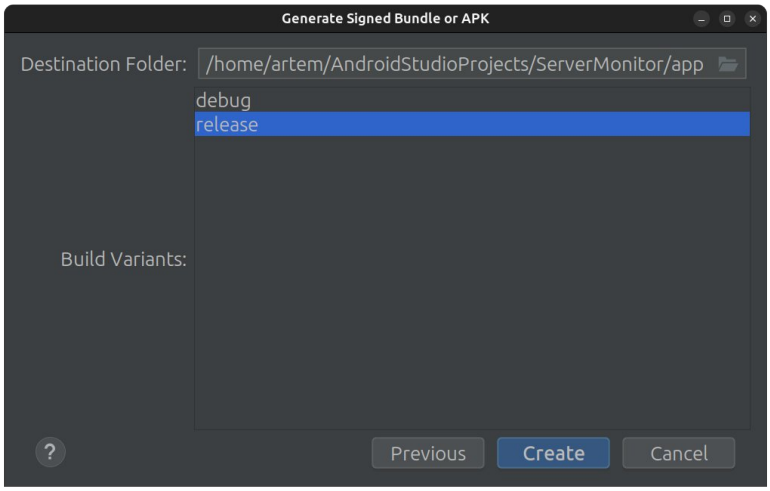


Рисунок 4.4 – Вибір варіанту збирання

– у спливаючому вікні, що сигналізує про успішне завершення операції, натиснути Locate (рисунок 4.5);

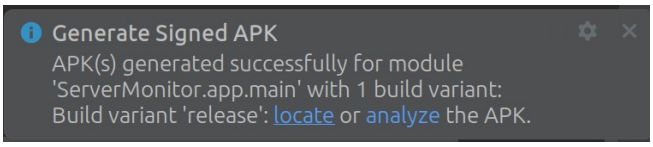


Рисунок 4.5 – Сповіщення про завершення генерування APK файлу

– у вікні файлового менеджера буде відображений створений APK файл (рисунок 4.6). Його, за потреби, можна перейменувати.

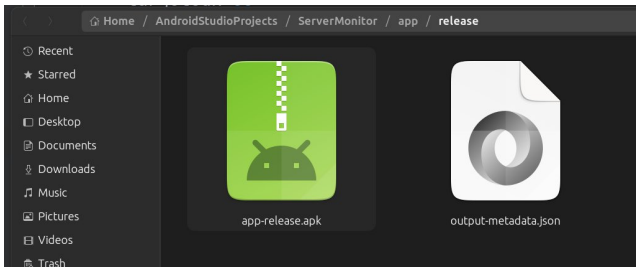


Рисунок 4.6 – Готовий APK файл у файловому менеджері

4.2 Підтримка програмного забезпечення

Користувачі можуть самостійно встановити нову версію застосунку. Завантажити нову версію можна на сторінці репозиторію на платформі GitHub у секції Releases. Процедури встановлення програми першого разу та наступного її оновлення є аналогічно схожими. Для їх реалізації потрібно виконати такі кроки:

– перейти на сторінку репозиторія за адресою https://github.com/artem-vovchenko01/android-server-monitoring та відкрити секцію Releases (рисунок 4.7);

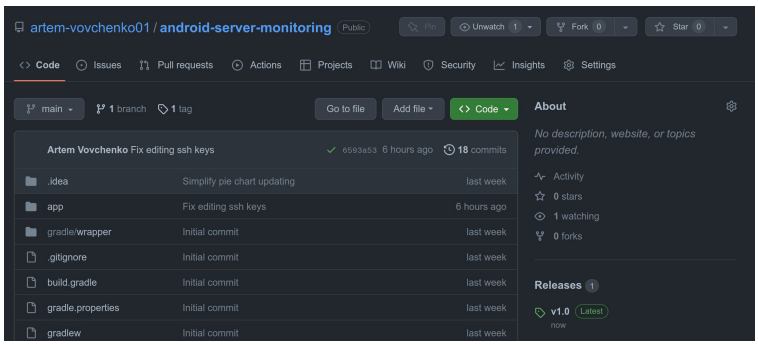


Рисунок 4.7 – Сторінка репозиторію даного проєкту на GitHub. З правого боку внизу знаходиться секція Releases

– завантажити APK файл останньої версії (рисунок 4.8);

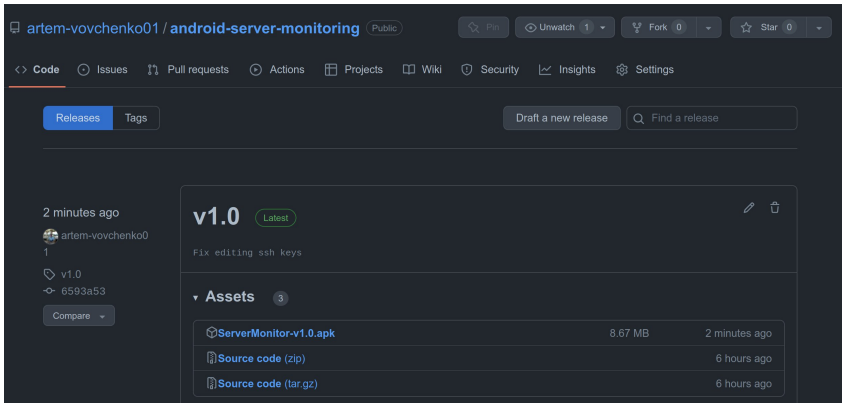


Рисунок 4.8 – Останній реліз програми та варіанти завантаження

– розмістити цей файл на мобільному пристрої та знайти його в будь-якому файловому менеджері;

– якщо це ще не виконано, надати необхідні дозволи (рисунок 4.9) для встановлення з нового джерела (в даному випадку – файлового менеджера) при виникненні відповідного діалогу (рисунок 4.10);

Рисунок 4.9 – Екран з налаштуванням встановлення програм з невідомих джерел

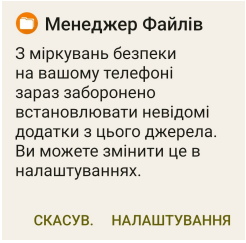


Рисунок 4.10 – Сповіщення про обмеження встановлення APK файлів зі сторонніх джерел

– встановити програму (рисунок 4.11) або оновити, якщо вона вже була встановлена (рисунок 4.12);

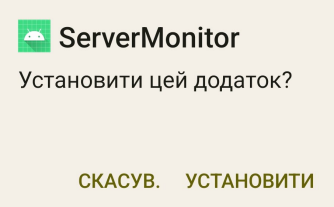


Рисунок 4.11 – Сповіщення про встановлення додатка

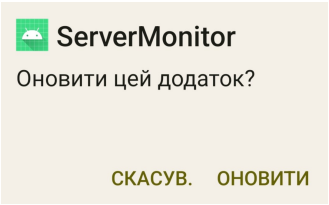


Рисунок 4.12 – Сповіщення про оновлення додатка

Висновки до розділу

У цьому розділі зроблено огляд способів розгортання програм на платформі Android. Продемонстровано процес створення готового для розгортання артефакту в інтегрованому середовищі розробки Android Studio. Показано, як кінцевий користувач може встановити програму то оновлювати її.

ВИСНОВКИ

У цій роботі було запропоноване рішення, яке спрощує системним адміністраторам певний список завдань, які може бути не так зручно виконувати іншими інструментами. На сьогоднішній день керування віддаленими системами та їхній моніторинг є дуже актуальними. Додаток, який було розроблено, добре підходить для виконання задач у цій сфері. Його можна надалі покращити для реалізації ще більшої кількості функціоналу, потрібного користувачам. Додаток можна опублікувати в магазинах для Android програм, наприклад, Google Play, для охоплення більшої аудиторії.

У результаті роботи над даним проєктом було створено Android застосунок, який дозволяє:

– віддалено моніторити Linux сервери;

– запускати постійний моніторинг та відстежувати результати на графіках;

– використовувати інтерактивну SSH сесію для виконання будь-яких команд на сервері;

– запускати скрипти одночасно на кількох серверах тощо.

У якості середовища розробки обрано Android Studio. Для постійного зберігання даних обрано базу даних SQLite. Бібліотека JSch дозволила створювати SSH сесії на серверах та виконувати на них команди. Після реалізації застосунок була протестовано на реальному пристрої.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Pull or Push: How to Select Monitoring Systems? URL: https://www.alibabacloud.com/blog/pull-or-push-how-to-select-monitoring-systems_599007 (дата звернення 04.06.2023)

2) James Turnbull. The Art of Monitoring. James Turnbull, 2016. сторінки. 23-25

3) Методологія застосунку дванадцяти факторів. URL: https://12factor.net (дата звернення 04.06.2023)

4) Текстовий редактор Neovim. URL: https://neovim.io (дата звернення 04.06.2023)

5) Текстовий редактор Lunarvim. URL: https://www.lunarvim.org/ (дата звернення 04.06.2023)

6) Fragments and the Navigation Component Codelab. URL: https://developer.android.com/codelabs/basic-android-kotlin-training-fragments-navigation-component (дата звернення 04.06.2023)

7) Save data in a local database using Room. URL: https://developer.android.com/training/data-storage/room (дата звернення 04.06.2023)

8) Java JSch Example to run Shell Commands on SSH Unix Server. URL: https://www.digitalocean.com/community/tutorials/jsch-example-java-ssh-unix-server (дата звернення 04.06.2023)

9) Google I/O 2010 - The world of ListView. URL: https://youtu.be/wDBM6wVEO70 (дата звернення 04.06.2023)

10) Get started with the Navigation component. URL: https://developer.android.com/guide/navigation/get-started (дата звернення 04.06.2023)

11) Android бібліотека MPAndroidChart. URL: https://github.com/PhilJay/MPAndroidChart (дата звернення 04.06.2023)

12) Your QA tester's hierarchy of needs: what is the agile testing pyramid? URL: https://www.onpathtesting.com/blog/qa-testers-what-is-the-agile-testing-pyramid (дата звернення 04.06.2023)

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015473707

Дата перевірки:
07.06.2023 07:23:52 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
07.06.2023 07:24:58 EEST

ID користувача:
76913

Назва документа: **ІТ-91_Вовченко_ПЗ**

Кількість сторінок: **72** Кількість слів: **9667** Кількість символів: **74774** Розмір файлу: **8.92 MB** ID файлу: **1015131910**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.25%
Схожість

Найбільша схожість: **4.03%** з джерелом з Бібліотеки (ID файлу: **1015067687**)

2.78% Джерела з Інтернету 167 ..... Сторінка 74

8.25% Джерела з Бібліотеки 265 ..... Сторінка 77

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 72

Підозріле форматування 11 сторінок

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДДАЛЕНОГО МОНІТОРИНГУ ТА КЕРУВАННЯ СЕРВЕРАМИ

Текст програми

КПІ.ІТ-9105.045490.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
Олексій СОПОВ

Нормоконтроль:
Максим ГОЛОВЧЕНКО

Виконавець:
Артем ВОВЧЕНКО

Файл MainActivity.java

```
package com.example.servermonitor;

import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;
import androidx.navigation.NavController;
import androidx.navigation.fragment.NavHostFragment;
import androidx.navigation.ui.AppBarConfiguration;
import androidx.navigation.ui.NavigationUI;
import androidx.room.Room;

import android.os.Bundle;

import com.example.servermonitor.adapter.ServerAdapter;
import com.example.servermonitor.databinding.ActivityMainBinding;
import com.example.servermonitor.db.Converters;
import com.example.servermonitor.db.ServerDatabase;
import com.example.servermonitor.db.entity.MonitoringRecordEntity;
import com.example.servermonitor.model.ServerModel;
import com.example.servermonitor.model.SshKeyModel;
import com.example.servermonitor.service.ServerService;
import com.example.servermonitor.service.SshKeyService;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.HashMap;
import java.util.Optional;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;

public class MainActivity extends AppCompatActivity {
    private static final int MONITORING_INTERVAL = 3;
    public static ServerDatabase database;
    public ServerService serverService;
    public ServerAdapter serverAdapter;
    public ArrayList<ServerModel> serverModels;
    public HashMap<ServerModel, ScheduledFuture> scheduledJobs;
    public HashMap<ServerModel, ExecutorService> executors;
    public HashMap<ServerModel, SshSessionWorker> serverSessions;
    private ActivityMainBinding binding;
    private SshKeyService sshKeyService;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
        setupUiComponents();
        setupOnClickListeners();
        database = Room.databaseBuilder(
                getApplicationContext(),
                ServerDatabase.class,
                "ServerDB")
                .fallbackToDestructiveMigration()
                .build();
```

```
        serverService = new ServerService(database);
        sshKeyService = new SshKeyService(database);
        serverSessions = new HashMap<>();
        scheduledJobs = new HashMap<>();
        executors = new HashMap<>();
        new Thread(() -> {
            serverModels = serverService.getAllServers();
            addPreviousServers(serverModels);
        }).start();
    }
    public void setupUiComponents() {
        NavHostFragment navHostFragment =
                (NavHostFragment)
getSupportFragmentManager().findFragmentById(R.id.navHostFragment);

        NavController navController = navHostFragment.getNavController();
        AppBarConfiguration appBarConfiguration =
                new AppBarConfiguration.Builder(R.id.serversFragment,
R.id.sshKeysFragment,
R.id.shellScriptsFragment).setDrawerLayout(binding.drawerLayout).build();
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        NavigationUI.setupWithNavController(
                toolbar, navController, appBarConfiguration);
        NavigationUI.setupWithNavController(binding.navView, navController);
    }
    public void setupOnClickListeners() {
    }

    public void addPreviousServers(ArrayList<ServerModel> serverModels) {
        for (int i = 0; i < serverModels.size(); i++) {
            ServerModel model = serverModels.get(i);
            model.setServerStatusImg(R.drawable.redcircle);
            model.setConnected(false);
            addWorkerForNewServer(model, i);
        }
    }

    public void addNewServer(ServerModel serverModel) {
        new Thread(() -> {
            if (serverModel.getId() != 0) {
                int pos = serverModels.indexOf(serverModels.stream().filter(m
-> m.getId() == serverModel.getId()).findFirst().get());
                serverService.updateServer(serverModel);
                ServerModel previousServer = serverModels.get(pos);
                stopJobForServer(previousServer);
                serverModels.set(pos, serverModel);
                addWorkerForNewServer(serverModel, pos);
            } else {
                long id = serverService.addServer(serverModel);
                serverModel.setId((int) id);
                serverModels.add(serverModel);
                addWorkerForNewServer(serverModel, serverModels.size() - 1);
            }
            runOnUiThread(() -> serverAdapter.notifyDataSetChanged());
        }).start();
    }
    public void stopJobForServer(ServerModel serverModel) {
        ScheduledFuture<?> future = scheduledJobs.get(serverModel);
        ExecutorService executor = executors.get(serverModel);
        future.cancel(true);
        executor.shutdown();
        scheduledJobs.remove(serverModel);
```

```
        executors.remove(serverModel);
    }


    private void addWorkerForNewServer(ServerModel serverModel, int position)
{
        ScheduledExecutorService executor =
Executors.newScheduledThreadPool(1);
        ScheduledFuture<?> future = executor.scheduleAtFixedRate(() -> {
            Optional<SshKeyModel> sshKeyModel =
Optional.of(sshKeyService.getSshKeyById(serverModel.getPrivateKeyId()));
            SshSessionWorker worker =
serverSessions.getOrDefault(serverModel, null);
            if (worker == null) {
                try {
                    worker = new SshSessionWorker(getApplicationContext(),
serverModel, sshKeyModel);
                    serverSessions.put(serverModel, worker);
                } catch (Exception e) {
                    e.printStackTrace();
                    serverModel.setConnected(false);
                    serverModel.setServerStatusImg(R.drawable.redcircle);
                    runOnUiThread(() -> {
                        if (serverAdapter != null)
                            serverAdapter.notifyItemChanged(position);
                    });
                    return;
                }
            }
            MonitoringRecordEntity monitoringRecordEntity =
worker.getMonitoringStats();
            updateServerModel(position, serverModel, monitoringRecordEntity);
            runOnUiThread(() -> {
                if (serverAdapter != null)
                    serverAdapter.notifyItemChanged(position);
            });
        }, 0, MONITORING_INTERVAL, TimeUnit.SECONDS);
        scheduledJobs.put(serverModel, future);
        executors.put(serverModel, executor);
    }
    private void updateServerModel(int position, ServerModel serverModel,
MonitoringRecordEntity monitoringRecord) {
        if (monitoringRecord == null) {
            serverModel.setConnected(false);
            serverModel.setServerStatusImg(R.drawable.redcircle);
            return;
        }
        serverModel.setMemoryUsedMb(monitoringRecord.memoryUsedMb);
        serverModel.setMemoryTotalMb(monitoringRecord.memoryTotalMb);
        serverModel.setDiskUsedMb(monitoringRecord.diskUsedMb);
        serverModel.setDiskTotalMb(monitoringRecord.diskTotalMb);
        serverModel.setCpuUsagePercent(monitoringRecord.cpuUsagePercent);
        serverModel.setConnected(true);
        serverModel.setServerStatusImg(R.drawable.greencircle);
        if (serverModel.getMonitoringSessionId() != -1) {
            monitoringRecord.monitoringSessionId =
serverModel.getMonitoringSessionId();
            monitoringRecord.timeRecorded =
Converters.dateToTimestamp(Calendar.getInstance().getTime());

database.getMonitoringRecordDao().addMonitoringRecord(monitoringRecord);
        }
    }
```

```
    @Override
    protected void onDestroy() {
        super.onDestroy();
        serverModels = null;
    }

}
```

Файл SshSessionWorker.java

```
package com.example.servermonitor;


import android.content.Context;
import android.util.Log;


import com.example.servermonitor.db.entity.MonitoringRecordEntity;
import com.example.servermonitor.model.ServerModel;
import com.example.servermonitor.model.SshKeyModel;
import com.jcraft.jsch.Channel;
import com.jcraft.jsch.ChannelExec;
import com.jcraft.jsch.ChannelSftp;
import com.jcraft.jsch.JSch;
import com.jcraft.jsch.JSchException;
import com.jcraft.jsch.Session;
import com.jcraft.jsch.SftpException;


import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Optional;


public class SshSessionWorker implements AutoCloseable {
    private static final String TAG = "sshSessionWorker";
    private Session session;
    private Context context;
    private Optional<SshKeyModel> sshKey;
    private ServerModel server;
    private JSch jsch;
    private ArrayList<File> tempFiles;
```

```
    private static String MEMORY_USED_MB_COMMAND = "free -m | grep Mem | awk
'{print $3}';";
    private static String MEMORY_TOTAL_MB_COMMAND = "free -m | grep Mem | awk
'{print $2}';";
    private static String DISK_USED_MB_COMMAND = "df -mP / | tail -n -1 | awk
'{print $3}';";
    private static String DISK_TOTAL_MB_COMMAND = "df -mP / | tail -n -1 |
awk '{print $2}';";
    private static String CPU_USAGE_COMMAND = "top -bn 1 | grep '%Cpu' | awk
'{print $2 + $4}';";
    public SshSessionWorker(Context context, ServerModel server,
Optional<SshKeyModel> sshKey) throws Exception {
        this.context = context;
        this.server = server;
        this.sshKey = sshKey;
        this.jsch = new JSch();
        this.tempFiles = new ArrayList<>();
        if (!createSshSession()) {
            throw new Exception("Couldn't establish session with a server");
        }
    }


    private Boolean createSshSession() {
        String user = server.getUserName();
        String host = server.getHostIp();
        String password = server.getPassword();
        int port = server.getPort();
        try {
            if (sshKey.isPresent()) {
                String privateKey = sshKey.get().getKeyData();
                File tempFile = getTemporaryFile(privateKey);
                String privateKeyPath = tempFile.getAbsolutePath();
                jsch.addIdentity(privateKeyPath);
            }
            session = jsch.getSession(user, host, port);
            if (password != null) {
                session.setPassword(password);
            }
            session.setConfig("StrictHostKeyChecking", "no");
            session.setTimeout(10000);
            session.connect();
        } catch (JSchException e) {
```

```
            e.printStackTrace();
            Log.d(TAG, "JSch exception occurred while getting the session");
            return false;
        } catch (IOException e) {
            e.printStackTrace();
            Log.d(TAG, "IO exception occurred when working with SSH key and
trying to establish ssh session");
            return false;
        }
        return true;
    }


    public MonitoringRecordEntity getMonitoringStats() {
        String output = "";
        String commandList =
                MEMORY_USED_MB_COMMAND +
                MEMORY_TOTAL_MB_COMMAND +
                DISK_USED_MB_COMMAND +
                DISK_TOTAL_MB_COMMAND +
                CPU_USAGE_COMMAND;
        output = executeSingleCommand(commandList);
        return parseMonitoringOutput(output);
    }
    private MonitoringRecordEntity parseMonitoringOutput(String output) {
        MonitoringRecordEntity monitoringRecordEntity = new
MonitoringRecordEntity();
        try {
           String[] split = output.split("\n");
           monitoringRecordEntity.id = 0;
           monitoringRecordEntity.monitoringSessionId = -1;
           monitoringRecordEntity.memoryUsedMb = Integer.parseInt(split[0]);
           monitoringRecordEntity.memoryTotalMb = Integer.parseInt(split[1]);
           monitoringRecordEntity.diskUsedMb = Integer.parseInt(split[2]);
           monitoringRecordEntity.diskTotalMb = Integer.parseInt(split[3]);
           monitoringRecordEntity.cpuUsagePercent =
Double.parseDouble(split[4]);
       } catch (Exception e) {
            Log.d(TAG, "Exception occurred while filling monitoring data from
output: " + e.getMessage());
            Log.d(TAG, Log.getStackTraceString(e));
           return null;
       }
```

```
        return monitoringRecordEntity;
    }
    public String executeShellScript(String scriptContents) {
        String homeDirPath = executeSingleCommand("pwd").replace("\n", "");
        String serverFileName = homeDirPath + "/scriptToExecute.sh";
        putFileFromText(scriptContents, serverFileName);
        executeSingleCommand("chmod +x " + serverFileName);
        String output = executeSingleCommand(serverFileName);
        executeSingleCommand("rm " + serverFileName);
        return output;
    }
    public Boolean putFileFromText(String text, String serverPath) {
        File file = null;
        try {
            file = getTemporaryFile(text);
        } catch (IOException e) {
            e.printStackTrace();
            Log.d(TAG, "Exception occurred when trying to place text in a
temporary file");
            return false;
        }
        return putFile(file, serverPath);
    }
    public Boolean putFile(File file, String serverPath) {
        ChannelSftp channelSftp = null;
        try {
            channelSftp = (ChannelSftp) session.openChannel("sftp");
            channelSftp.connect();
        } catch (JSchException e) {
            e.printStackTrace();
            Log.d(TAG, "JSch exception during SFTP connection occurred.");
            return false;
        }
        try {
            channelSftp.put(file.getAbsolutePath(), serverPath);
        } catch (SftpException e) {
            e.printStackTrace();
            Log.d(TAG, "sftp exception occurred when trying to put file to
the server");
            return false;
        }
        channelSftp.disconnect();
        return true;
```

```
    }

    public String executeSingleCommand(String command) {
        String result = "";
        try {
            ChannelExec channelExec = (ChannelExec)
session.openChannel("exec");
            channelExec.setCommand(command);
            channelExec.connect();
            result = readChannelOutput(channelExec);
            channelExec.disconnect();
        } catch (JSchException e) {
            throw new RuntimeException(e);
        }
        return result;
    }

    private String readChannelOutput(Channel channel){
        StringBuilder result = new StringBuilder();
        byte[] buffer = new byte[1024];
        try{
            InputStream in = channel.getInputStream();
            String line = "";
            while (true){
                while (in.available() > 0) {
                    int i = in.read(buffer, 0, 1024);
                    if (i < 0) {
                        break;
                    }
                    line = new String(buffer, 0, i);
                    result.append(line);
                }

                if (channel.isClosed()){
                    break;
                }
                try {
                    Thread.sleep(1000);
                } catch (Exception ee){}
            }
        }catch(Exception e){
```

```
        }
        return result.toString();
    }

    private File getTemporaryFile(String content) throws IOException {
        File tempFile = File.createTempFile("file_" + content.hashCode(),
".txt", context.getCacheDir());
        writeStringToFile(tempFile, content);
        tempFiles.add(tempFile);
        return tempFile;
    }

    private void writeStringToFile(File file, String content) {
        try {
            FileWriter writer = new FileWriter(file);
            writer.write(content);
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void close() throws Exception {
        session.disconnect();
        for (File file : tempFiles) {
            file.delete();
        }
    }
}
```

Файл SshShellSessionWorker.java

```
package com.example.servermonitor.service;

import android.content.Context;
import android.util.Log;

import com.example.servermonitor.fragment.BrowseServerFilesFragment;
import com.example.servermonitor.model.ServerModel;
import com.example.servermonitor.model.SshKeyModel;
```

```
import com.jcraft.jsch.ChannelSftp;
import com.jcraft.jsch.ChannelShell;
import com.jcraft.jsch.JSch;
import com.jcraft.jsch.JSchException;
import com.jcraft.jsch.Session;
import com.jcraft.jsch.SftpException;


import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.Optional;
import java.util.Vector;


public class SshShellSessionWorker implements AutoCloseable {
    private static final String TAG = "sshSessionWorker";
    private OutputStream outputStream;
    private InputStream inputStream;
    private ChannelShell channel;
    private ChannelSftp channelSftp;
    private Optional<SshKeyModel> sshKey;
    private ServerModel server;
    private Session session;
    private JSch jsch;
    private ArrayList<File> tempFiles;
    private Context context;


    public SshShellSessionWorker(Context context, ServerModel server,
Optional<SshKeyModel> sshKey) throws Exception {
        this.context = context;
        this.server = server;
        this.sshKey = sshKey;
        this.jsch = new JSch();
        this.tempFiles = new ArrayList<>();
        if (!createSshSession()) {
            throw new Exception("Couldn't establish session with a server");
        }
```

```
        establishShell();
    }

    public String tryFetchNewOutput() {
        StringBuilder result = new StringBuilder();
        int SIZE = 2048;
        byte[] tmp = new byte[SIZE];
        while (true) {
            int i = 0;
            try {
                if (inputStream.available() == 0) break;
                i = inputStream.read(tmp, 0, SIZE);
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
            if (i < 0)
                break;
            result.append(new String(tmp, 0, i));
        }
        return result.toString()
            .replace("\u001B[?2004l","")
            .replace("\u001B[?2004h","");
    }

    private Boolean createSshSession() {
        String user = server.getUserName();
        String host = server.getHostIp();
        String password = server.getPassword();
        int port = server.getPort();
        try {
            if (sshKey.isPresent()) {
                String privateKey = sshKey.get().getKeyData();
                File tempFile = getTemporaryFile(privateKey);
                String privateKeyPath = tempFile.getAbsolutePath();
                jsch.addIdentity(privateKeyPath);
            }
            session = jsch.getSession(user, host, port);
            if (password != null) {
                session.setPassword(password);
            }
            session.setConfig("StrictHostKeyChecking", "no");
            session.setTimeout(10000);
```

```
            session.connect();
        } catch (JSchException e) {
            e.printStackTrace();
            Log.d(TAG, "JSch exception occurred while getting the session");
            return false;
        } catch (IOException e) {
            e.printStackTrace();
            Log.d(TAG, "IO exception occurred when working with SSH key and
trying to establish ssh session");
            return false;
        }
        return true;
    }
    public Vector<ChannelSftp.LsEntry> listDir(String dir,
BrowseServerFilesFragment fragment) {
        Vector<ChannelSftp.LsEntry> lsEntries = null;
        try {
            if (channelSftp == null) {
                channelSftp = (ChannelSftp) session.openChannel("sftp");
                channelSftp.connect();
            }
            channelSftp.cd(dir);
            fragment.currentPath = channelSftp.pwd();
            lsEntries = channelSftp.ls(".");
            lsEntries.sort(lsEntryComparator);
        } catch(JSchException | SftpException e){
            throw new RuntimeException(e);
        }
        return lsEntries;
    }


    Comparator<ChannelSftp.LsEntry> lsEntryComparator = (entry1, entry2) -> {
        String name1 = entry1.getFilename();
        String name2 = entry2.getFilename();
        if (name1.equals(".") && ! name2.equals("..")) {
            return -1;
        } else if (name1.equals("..") && !name2.equals(".")) {
            return -1;
        }  else if (name1.equals("..") && name2.equals(".")) {
            return -1;
        } else {
            return name1.compareToIgnoreCase(name2);
```

```
        }
    };
    private void establishShell() {
        try {
            channel = (ChannelShell) session.openChannel("shell");
            outputStream = channel.getOutputStream();
            channel.connect();
            inputStream = channel.getInputStream();
            outputStream = channel.getOutputStream();
            executeCommand("unset LS_COLORS; export TERM=vt220");
        } catch (JSchException | IOException e) {
            e.printStackTrace();
            Log.d(TAG, "Exception occurred when establishing SSH shell");
        }
    }

    public void executeCommand(String command) {
        new Thread(() -> {
            try {
                PrintStream printStream = new PrintStream(outputStream,
true);
                printStream.print(command + "\n");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }).start();
    }

    private File getTemporaryFile(String content) throws IOException {
        File tempFile = File.createTempFile("file_" + content.hashCode(),
".txt", context.getCacheDir());
        writeStringToFile(tempFile, content);
        tempFiles.add(tempFile);
        return tempFile;
    }

    private void writeStringToFile(File file, String content) {
        try {
            FileWriter writer = new FileWriter(file);
            writer.write(content);
            writer.close();
```

```
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        @Override
        public void close() throws Exception {
            inputStream.close();
            outputStream.close();
            channel.disconnect();
            session.disconnect();
            if (channelSftp != null)
                channelSftp.disconnect();
            for (File file : tempFiles) {
                file.delete();
            }
        }
    }
```

Файл ServerService.java

```
package com.example.servermonitor.service;

import com.example.servermonitor.db.ServerDatabase;
import com.example.servermonitor.db.dao.ServerDao;
import com.example.servermonitor.db.entity.ServerEntity;
import com.example.servermonitor.mapper.ServerMapper;
import com.example.servermonitor.model.ServerModel;

import java.util.ArrayList;
import java.util.List;

public class ServerService {
    private ServerDatabase database;
    private ServerDao serverDao;
    public ServerService(ServerDatabase database) {
        this.database = database;
        this.serverDao = database.getServerDao();
    }
    public ArrayList<ServerModel> getAllServers() {
        ArrayList<ServerModel> models = new ArrayList<>();
        List<ServerEntity> entities = serverDao.getAllServers();
        for (ServerEntity serverEntity : entities) {
```

```
            models.add(ServerMapper.serverEntityToModel(serverEntity));
        }
        return models;
    }
    public long addServer(ServerModel serverModel) {
        return
serverDao.addServer(ServerMapper.serverModelToEntity(serverModel));
    }
    public void updateServer(ServerModel serverModel) {

serverDao.updateServer(ServerMapper.serverModelToEntity(serverModel));
    }
    public void deleteServerById(int id) {
        serverDao.deleteServerById(id);
    }
    public void deleteServer(ServerModel model) {
        serverDao.deleteServer(ServerMapper.serverModelToEntity(model));
    }
}
```

Файл ServerModel.java

```
package com.example.servermonitor.model;


import android.os.Parcel;
import android.os.Parcelable;


import androidx.annotation.NonNull;


public class ServerModel implements Parcelable {
    private int id;
    private String name;
    private String hostIp;
    private int port;
    private String userName;
    private String password;
    private boolean connected;
    private int memoryUsedMb;
    private int memoryTotalMb;
    private double cpuUsagePercent;
    private double diskUsedMb;
```

```
    private double diskTotalMb;
    private int serverStatusImg;
    private int monitoringSessionId = -1;
    private int privateKeyId;

    public int getPrivateKeyId() {
        return privateKeyId;
    }

    public void setPrivateKeyId(int privateKeyId) {
        this.privateKeyId = privateKeyId;
    }

    public ServerModel() {}
    public ServerModel(int id, String name, String hostIp, int port, String
userName, String password, int privateKeyId, boolean connected, int
memoryUsedMb, int memoryTotalMb, double cpuUsagePercent, double diskUsedMb,
double diskTotalMb, int serverStatusImg) {
        this.id = id;
        this.name = name;
        this.hostIp = hostIp;
        this.port = port;
        this.userName = userName;
        this.password = password;
        this.privateKeyId = privateKeyId;
        this.connected = connected;
        this.memoryUsedMb = memoryUsedMb;
        this.memoryTotalMb = memoryTotalMb;
        this.cpuUsagePercent = cpuUsagePercent;
        this.diskUsedMb = diskUsedMb;
        this.diskTotalMb = diskTotalMb;
        this.serverStatusImg = serverStatusImg;
    }

    public int getMonitoringSessionId() {
        return monitoringSessionId;
    }

    public void setMonitoringSessionId(int monitoringSessionId) {
        this.monitoringSessionId = monitoringSessionId;
    }
```

```
public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getHostIp() {
    return hostIp;
}

public void setHostIp(String hostIp) {
    this.hostIp = hostIp;
}

public int getPort() {
    return port;
}

public void setPort(int port) {
    this.port = port;
}

public String getUserName() {
    return userName;
}

public void setUserName(String userName) {
    this.userName = userName;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
```

```
        this.password = password;
    }

    public int getServerStatusImg() {
        return serverStatusImg;
    }

    public void setServerStatusImg(int serverStatusImg) {
        this.serverStatusImg = serverStatusImg;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public boolean isConnected() {
        return connected;
    }

    public void setConnected(boolean connected) {
        this.connected = connected;
    }

    public int getMemoryUsedMb() {
        return memoryUsedMb;
    }

    public void setMemoryUsedMb(int memoryUsedMb) {
        this.memoryUsedMb = memoryUsedMb;
    }

    public int getMemoryTotalMb() {
        return memoryTotalMb;
    }
```

```
public void setMemoryTotalMb(int memoryTotalMb) {
    this.memoryTotalMb = memoryTotalMb;
}

public double getCpuUsagePercent() {
    return cpuUsagePercent;
}

public void setCpuUsagePercent(double cpuUsagePercent) {
    this.cpuUsagePercent = cpuUsagePercent;
}

public double getDiskUsedMb() {
    return diskUsedMb;
}

public void setDiskUsedMb(double diskUsedMb) {
    this.diskUsedMb = diskUsedMb;
}

public double getDiskTotalMb() {
    return diskTotalMb;
}

public void setDiskTotalMb(double diskTotalMb) {
    this.diskTotalMb = diskTotalMb;
}

@Override
public int describeContents() {
    return 0;
}

@Override
public void writeToParcel(@NonNull Parcel dest, int flags) {
    dest.writeString(name);
    dest.writeString(hostIp);
    dest.writeInt(port);
    dest.writeString(userName);
```

```
        dest.writeString(password);
        dest.writeInt(privateKeyId);
    }

    public static final Parcelable.Creator<ServerModel> CREATOR = new
Parcelable.Creator<ServerModel>() {
        @Override
        public ServerModel createFromParcel(Parcel source) {
            return new ServerModel(source);
        }

        @Override
        public ServerModel[] newArray(int size) {
            return new ServerModel[size];
        }
    };
    private ServerModel(Parcel in) {
        this.name = in.readString();
        this.hostIp = in.readString();
        this.port = in.readInt();
        this.userName = in.readString();
        this.password = in.readString();
        this.privateKeyId = in.readInt();
    }
}
```

Файл ServerMapper.java

```
package com.example.servermonitor.mapper;

import com.example.servermonitor.model.ServerModel;
import com.example.servermonitor.db.entity.ServerEntity;

public class ServerMapper {
    public static ServerEntity serverModelToEntity(ServerModel serverModel) {
        return new ServerEntity(
                serverModel.getId(),
                serverModel.getName(),
                serverModel.getHostIp(),
                serverModel.getPort(),
```

```
                serverModel.getUserName(),
                serverModel.getPassword(),
                serverModel.getPrivateKeyId()
        );
    }

    public static ServerModel serverEntityToModel(ServerEntity serverEntity)
{
        return new ServerModel(
               serverEntity.id,
               serverEntity.name,
               serverEntity.hostIp,
               serverEntity.port,
               serverEntity.userName,
               serverEntity.password,
               serverEntity.privateKeyId,
              false,
                0,
                0,
                0,
                0,
                0,
                0
        );
    }
}
```

Файл ServersFragment.java

```
package com.example.servermonitor.fragment;


import android.content.Context;
import android.os.Bundle;


import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import androidx.navigation.NavController;
import androidx.navigation.Navigation;
import androidx.recyclerview.widget.DefaultItemAnimator;
import androidx.recyclerview.widget.LinearLayoutManager;
```

```
import androidx.recyclerview.widget.RecyclerView;


import android.view.ContextMenu;
import android.view.LayoutInflater;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;


import com.example.servermonitor.MainActivity;
import com.example.servermonitor.R;
import com.example.servermonitor.SshSessionWorker;
import com.example.servermonitor.adapter.ServerAdapter;
import com.example.servermonitor.databinding.FragmentServersBinding;
import com.example.servermonitor.model.ServerModel;
import com.example.servermonitor.service.ServerService;


public class ServersFragment extends Fragment {
    private FragmentServersBinding binding;
    private MainActivity activity;
    private Context context;
    private ServerService serverService;
    private ServerAdapter serverAdapter;


    public ServersFragment() {
        // Required empty public constructor
    }


    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        binding = FragmentServersBinding.inflate(inflater, container, false);
        activity = (MainActivity) getActivity();
        activity.getSupportActionBar().setTitle("Servers");
```

```
        context = activity.getApplicationContext();
        serverService = new ServerService(MainActivity.database);
        Bundle args = getArguments();
        if (args != null) {
            if (args.getInt("success") == 1) {
                ServerModel newServerModel =
args.getParcelable("serverModel");
                activity.addNewServer(newServerModel);
            }
            getArguments().clear();
        }
        return binding.getRoot();
    }


    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle
savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        setupUiComponents();
        setupOnClickListeners();
        registerForContextMenu(binding.rvServers);
        new Thread(() -> {
            while (activity.serverModels == null) {
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }
            }
            serverAdapter = new ServerAdapter(context, activity.serverModels,
activity);
            activity.serverAdapter = serverAdapter;
            activity.runOnUiThread(() -> {
                binding.rvServers.setAdapter(serverAdapter);
            });
        }).start();
    }


    public void setupUiComponents() {
        RecyclerView.LayoutManager layoutManager = new
LinearLayoutManager(context);
        binding.rvServers.setLayoutManager(layoutManager);
```

```
        binding.rvServers.setItemAnimator(new DefaultItemAnimator());
    }

    public void setupOnClickListeners() {
        binding.fabAddServer.setOnClickListener(v -> {
            NavController controller =
Navigation.findNavController(binding.getRoot());

controller.navigate(R.id.action_serversFragment_to_editServerFragment);
        });
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        activity.serverAdapter = null;
    }

    @Override
    public boolean onContextItemSelected(MenuItem item) {
        int pposition = serverAdapter.selectedItemPosition;
        ServerModel server = activity.serverModels.get(pposition);

        switch (item.getTitle().toString()) {
            case "Edit":
                NavController controller =
Navigation.findNavController(binding.getRoot());
                Bundle bundle = new Bundle();
                bundle.putInt("edit", 1);
                bundle.putParcelable("serverModel", server);

controller.navigate(R.id.action_serversFragment_to_editServerFragment,
bundle);
                break;
            case "Delete":
                new Thread(() -> {
                    serverService.deleteServer(server);
                    activity.serverModels.remove(pposition);
                    activity.stopJobForServer(server);
                    activity.runOnUiThread(() ->
serverAdapter.notifyItemRemoved(pposition));
```

```
                }).start();
                break;
            case "Reboot":
                new Thread(() -> {
                    SshSessionWorker worker =
activity.serverSessions.get(server);
                    worker.executeSingleCommand("reboot");
                }).start();
                break;
            case "Shutdown":
                new Thread(() -> {
                    SshSessionWorker worker =
activity.serverSessions.get(server);
                    worker.executeSingleCommand("shutdown now");
                }).start();
                break;
        }
        return super.onContextItemSelected(item);
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
                                    ContextMenu.ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);
        MenuInflater inflater = activity.getMenuInflater();
        inflater.inflate(R.menu.server_context_menu, menu);
    }
}
```

Файл ShowScriptOutputDialog.java

```
package com.example.servermonitor.dialog;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;

import androidx.annotation.NonNull;
```

```
import androidx.annotation.Nullable;
import androidx.fragment.app.DialogFragment;

import com.example.servermonitor.R;

public class ShowScriptOutputDialog extends DialogFragment {
    private TextView tvScriptOutput;
    private Button btnClose;
    private Button btnCopy;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable
ViewGroup container, @Nullable Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.script_output_layout,
container, false);
        tvScriptOutput = rootView.findViewById(R.id.tvOutput);
        btnClose = rootView.findViewById(R.id.btnClose);
        btnCopy = rootView.findViewById(R.id.btnCopy);
        setupListeners();
        return rootView;
    }
    public void setOutputText(String text) {
        new Thread(() -> {
            while (tvScriptOutput == null) {
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }
            }
            getActivity().runOnUiThread(() -> {
                tvScriptOutput.setText(text);
            });
        }).start();
    }

    private void setupListeners() {
        btnClose.setOnClickListener((v) -> {
           dismiss();
        });
```

```
        btnCopy.setOnClickListener((v) -> {


        });
    }
}
```

Файл ServerDatabase.java

```
package com.example.servermonitor.db;


import androidx.room.Database;
import androidx.room.RoomDatabase;
import androidx.room.TypeConverters;


import com.example.servermonitor.db.dao.MonitoringRecordDao;
import com.example.servermonitor.db.dao.MonitoringSessionDao;
import com.example.servermonitor.db.dao.ServerDao;
import com.example.servermonitor.db.dao.ShellScriptDao;
import com.example.servermonitor.db.dao.SshKeyDao;
import com.example.servermonitor.db.entity.MonitoringRecordEntity;
import com.example.servermonitor.db.entity.MonitoringSessionEntity;
import com.example.servermonitor.db.entity.ServerEntity;
import com.example.servermonitor.db.entity.ShellScriptEntity;
import com.example.servermonitor.db.entity.SshKeyEntity;


@Database(entities = {ServerEntity.class,
        MonitoringRecordEntity.class,
        MonitoringSessionEntity.class,
        SshKeyEntity.class,
        ShellScriptEntity.class},
        version = 5
)
@TypeConverters({Converters.class})
public abstract class ServerDatabase extends RoomDatabase {
    public abstract ShellScriptDao getShellScriptDao();
    public abstract SshKeyDao getSshKeyDao();
    public abstract ServerDao getServerDao();
    public abstract MonitoringRecordDao getMonitoringRecordDao();
    public abstract MonitoringSessionDao getMonitoringSessionDao();
}
```

Файл ServerDao.java

```
package com.example.servermonitor.db.dao;


import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Update;


import com.example.servermonitor.db.entity.ServerEntity;


import java.util.List;


@Dao
public interface ServerDao {
    @Insert
    long addServer(ServerEntity serverEntity);


    @Update
    void updateServer(ServerEntity serverEntity);


    @Delete
    void deleteServer(ServerEntity serverEntity);
    @Query("delete from servers where id == :id")
    void deleteServerById(int id);


    @Query("select * from servers")
    List<ServerEntity> getAllServers();


    @Query("select * from servers where id == :id")
    ServerEntity getServer(int id);
}
```

Файл ServerFilesAdapter.java

```
package com.example.servermonitor.adapter;


import android.content.Context;
import android.view.ContextMenu;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;


import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;


import com.example.servermonitor.MainActivity;
import com.example.servermonitor.R;
import com.example.servermonitor.fragment.BrowseServerFilesFragment;
import com.example.servermonitor.model.SshKeyModel;
import com.jcraft.jsch.ChannelSftp;
import com.jcraft.jsch.SftpATTRS;


import java.util.ArrayList;
import java.util.Vector;


public class ServerFilesAdapter extends
RecyclerView.Adapter<ServerFilesAdapter.ServerFilesViewHolder> {
    public Vector<ChannelSftp.LsEntry> lsEntries;
    private Context context;
    private MainActivity activity;
    public int selectedItemPosition;
    private BrowseServerFilesFragment fragment;


    public ServerFilesAdapter(Context context, Vector<ChannelSftp.LsEntry>
lsEntries, MainActivity activity, BrowseServerFilesFragment fragment) {
        this.context = context;
        this.activity = activity;
        this.lsEntries = lsEntries;
        this.fragment = fragment;
    }
```

```
    @NonNull
    @Override
    public ServerFilesViewHolder onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());
        View itemView = inflater.inflate(R.layout.file_item_layout, parent,
false);
        return new ServerFilesViewHolder(itemView);
    }


    @Override
    public void onBindViewHolder(@NonNull ServerFilesViewHolder holder, int
position) {
        ChannelSftp.LsEntry entry = lsEntries.get(position);
        holder.tvFileName.setText(entry.getFilename());
        holder.itemView.setOnLongClickListener((v) -> {
            selectedItemPosition = position;
            return false;
        });
        holder.itemView.setOnClickListener(v -> {
            if (entry.getAttrs().isDir())
                fragment.goToPath(entry.getFilename());
        });
        if (entry.getAttrs().isDir()) {
           holder.fileIcon.setImageResource(R.drawable.baseline_folder_24);
        } else {

holder.fileIcon.setImageResource(R.drawable.baseline_insert_drive_file_24);
        }
        activity.registerForContextMenu(holder.itemView);
    }


    @Override
    public int getItemCount() {
        return lsEntries.size();
    }


    public static class ServerFilesViewHolder extends RecyclerView.ViewHolder
implements View.OnCreateContextMenuListener {
        public TextView tvFileName;
```

```
        public ImageView fileIcon;
        public ServerFilesViewHolder(@NonNull View itemView) {
            super(itemView);
            tvFileName = itemView.findViewById(R.id.tvFileName);
            fileIcon = itemView.findViewById(R.id.fileIcon);
        }

        @Override
        public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo) {
            menu.add("Rename");
            menu.add("Delete");
        }
    }
}
```

Файл ServerAdapter.java

```
package com.example.servermonitor.adapter;

import android.content.Context;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.navigation.NavController;
import androidx.navigation.Navigation;
import androidx.recyclerview.widget.RecyclerView;

import com.example.servermonitor.MainActivity;
import com.example.servermonitor.R;
import com.example.servermonitor.fragment.ServerFragment;
import com.example.servermonitor.model.ServerModel;
```

```
import java.util.ArrayList;


public class ServerAdapter extends
RecyclerView.Adapter<ServerAdapter.ServerViewHolder>  {
    private ArrayList<ServerModel> servers;
    private Context context;
    private MainActivity mainActivity;
    public int selectedItemPosition;


    public ServerAdapter(Context context, ArrayList<ServerModel> servers,
MainActivity mainActivity) {
        this.context = context;
        this.mainActivity = mainActivity;
        this.servers = servers;
    }


    @NonNull
    @Override
    public ServerViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());
        View itemView = inflater.inflate(R.layout.server_item_layout, parent,
false);
        return new ServerViewHolder(itemView);
    }


    @Override
    public void onBindViewHolder(@NonNull ServerViewHolder holder, int
position) {
        ServerModel serverModel = servers.get(position);
        holder.tvServerName.setText(serverModel.getName());

holder.tvMemoryUsed.setText(String.valueOf(serverModel.getMemoryUsedMb()) +
"MB");

holder.tvMemoryTotal.setText(String.valueOf(serverModel.getMemoryTotalMb()) +
"MB");
        holder.tvDiskUsed.setText((int)serverModel.getDiskUsedMb() + "MB");
        holder.tvDiskTotal.setText((int)serverModel.getDiskTotalMb() + "MB");

holder.imvServerStatus.setImageResource(serverModel.getServerStatusImg());
```

```
        holder.tvCpuUsage.setText(serverModel.getCpuUsagePercent() + "%");
        holder.itemView.setOnClickListener(v -> {
            NavController navController =
Navigation.findNavController(mainActivity, R.id.navHostFragment);
            Bundle fragmentData = new Bundle();
            fragmentData.putParcelable("serverModel", servers.get(position));

navController.navigate(R.id.action_serversFragment_to_serverFragment,
fragmentData);
        });
        holder.itemView.setOnLongClickListener(v -> {
            selectedItemPosition = position;
            return false;
        });
        mainActivity.registerForContextMenu(holder.itemView);
    }


    @Override
    public int getItemCount() {
        return servers.size();
    }


    public static class ServerViewHolder extends RecyclerView.ViewHolder {
        public Button btnEdit;
        public Button btnDelete;
        public TextView tvServerName;
        public TextView tvMemoryUsed;
        public TextView tvMemoryTotal;
        public TextView tvDiskUsed;
        public TextView tvDiskTotal;
        public TextView tvCpuUsage;
        public ImageView imvServerStatus;
        public ServerViewHolder(@NonNull View itemView) {
            super(itemView);
            tvServerName = itemView.findViewById(R.id.tvServerName);
            tvMemoryUsed = itemView.findViewById(R.id.tvMemoryUsed);
            tvMemoryTotal = itemView.findViewById(R.id.tvMemoryTotal);
            tvDiskUsed = itemView.findViewById(R.id.tvDiskUsed);
            tvDiskTotal = itemView.findViewById(R.id.tvDiskTotal);
            tvCpuUsage = itemView.findViewById(R.id.tvCpuUsage);
            imvServerStatus = itemView.findViewById(R.id.imvServerStatus);
            btnEdit = itemView.findViewById(R.id.btnEdit);
```

```
            btnDelete = itemView.findViewById(R.id.btnDelete);
        }
    }
}
```

Файл TerminalFragment.java

```
package com.example.servermonitor.fragment;

import android.content.Context;
import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.example.servermonitor.MainActivity;
import com.example.servermonitor.R;
import com.example.servermonitor.SshSessionWorker;
import com.example.servermonitor.databinding.FragmentTerminalBinding;
import com.example.servermonitor.mapper.ServerMapper;
import com.example.servermonitor.model.ServerModel;
import com.example.servermonitor.model.SshKeyModel;
import com.example.servermonitor.service.SshKeyService;
import com.example.servermonitor.service.SshShellSessionWorker;
import com.jcraft.jsch.ChannelShell;
import com.jcraft.jsch.JSch;
import com.jcraft.jsch.JSchException;
import com.jcraft.jsch.Session;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintStream;
import java.util.Optional;

public class TerminalFragment extends Fragment {
```

```
    private static final int OUTPUT_SIZE_LIMIT = 100000;
    private int currentOutputSize = 0;
    private FragmentTerminalBinding binding;
    private SshKeyService sshKeyService;
    private MainActivity activity;
    public static ServerModel serverModel;
    private SshShellSessionWorker shellSessionWorker;
    private Context context;


    public TerminalFragment() {
        // Required empty public constructor
    }


    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        binding = FragmentTerminalBinding.inflate(inflater, container,
false);
        activity = (MainActivity) getActivity();
        activity.getSupportActionBar().setTitle("Terminal");
        context = activity.getApplicationContext();
        sshKeyService = new SshKeyService(MainActivity.database);
        binding.userInput.setActivated(true);
        runTerminalSession();
        setupListeners();
        return binding.getRoot();
    }
    private void setupListeners() {
        binding.btnRunCommand.setOnClickListener((v) -> {
            executeCommand(binding.userInput.getText().toString());
            binding.userInput.setText("");
        });
    }
    private void runTerminalSession() {
        new Thread(() -> {
```

```
            Optional<SshKeyModel> sshKeyModel =
Optional.of(sshKeyService.getSshKeyById(serverModel.getPrivateKeyId()));
            try {
                shellSessionWorker = new SshShellSessionWorker(context,
serverModel, sshKeyModel);
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
            fetchOutput();
        }).start();
    }
    private void executeCommand(String command) {
        shellSessionWorker.executeCommand(command);
    }

    private void fetchOutput() throws RuntimeException {
        Thread thread = new Thread(() -> {
            while (true) {
                if (shellSessionWorker == null) break;
                String result = shellSessionWorker.tryFetchNewOutput();
                if (result != "")
                    activity.runOnUiThread(() ->
updateTerminalWithNewText(result));
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }
            }
        });
        try {
            thread.start();
            thread.join();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

    public void updateTerminalWithNewText(String newPart) {
        binding.terminalOutput.append(newPart);
        currentOutputSize += newPart.length();
        if (currentOutputSize > OUTPUT_SIZE_LIMIT) {
```

```
            String text = binding.terminalOutput.getText().toString();
            String newText = text.substring(text.length() -
(int)(OUTPUT_SIZE_LIMIT * 0.8));
            binding.terminalOutput.setText(newText);
            currentOutputSize = newText.length();
        }
        binding.terminalScrollView.fullScroll(View.FOCUS_DOWN);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        shellSessionWorker = null;
    }
}
```

Файл activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout
    android:id="@+id/drawerLayout"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    >
    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    <androidx.appcompat.widget.Toolbar
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        android:layout_width="match_parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:id="@+id/toolbar" />
```

```
    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/navHostFragment"
        android:name="androidx.navigation.fragment.NavHostFragment"
        app:layout_constraintTop_toBottomOf="@id/toolbar"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:defaultNavHost="true"
        app:navGraph="@navigation/nav_graph" />
    </androidx.constraintlayout.widget.ConstraintLayout>

<com.google.android.material.navigation.NavigationView
    android:id="@+id/navView"
    android:layout_width="240dp"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    app:menu="@menu/navdrawer_menu"
    app:headerLayout="@layout/nav_header"
    />

</androidx.drawerlayout.widget.DrawerLayout>
```

Файл fragment_server.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".fragment.ServerFragment">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:id="@+id/tvLabelMemory"
            android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_marginStart="30dp"
        android:layout_marginTop="2dp"
        android:text="Memory MB"
        android:textSize="16sp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/tvLabelCpu"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="2dp"
        android:text="CPU load %"
        android:textSize="16sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/tvLabelDisk"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="2dp"
        android:layout_marginEnd="40dp"
        android:text="Disk MB"
        android:textSize="16sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/tvLabelMemoryNoData"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="80dp"
        android:text="No data"
        android:textSize="16sp"
        app:layout_constraintEnd_toEndOf="@id/tvLabelMemory"
        app:layout_constraintStart_toStartOf="@id/tvLabelMemory"
        app:layout_constraintTop_toTopOf="parent" />
```

```
<TextView
    android:id="@+id/tvLabelCpuNoData"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="80dp"
    android:text="No data"
    android:textSize="16sp"
    app:layout_constraintEnd_toEndOf="@id/tvLabelCpu"
    app:layout_constraintStart_toStartOf="@id/tvLabelCpu"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/tvLabelDiskNoData"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="80dp"
    android:text="No data"
    android:textSize="16sp"
    app:layout_constraintEnd_toEndOf="@id/tvLabelDisk"
    app:layout_constraintStart_toStartOf="@id/tvLabelDisk"
    app:layout_constraintTop_toTopOf="parent" />

<com.github.mikephil.charting.charts.PieChart
    android:id="@+id/pcMemory"
    android:layout_width="140dp"
    android:layout_height="140dp"
    android:layout_marginTop="10dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/tvLabelMemory" />

<com.github.mikephil.charting.charts.PieChart
    android:id="@+id/pcCpu"
    android:layout_width="140dp"
    android:layout_height="140dp"
    android:layout_marginTop="10dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/tvLabelCpu" />

<com.github.mikephil.charting.charts.PieChart
    android:id="@+id/pcDisk"
```

```
            android:layout_width="140dp"
            android:layout_height="140dp"
            android:layout_marginTop="10dp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toBottomOf="@id/tvLabelDisk" />

        <Button
            android:id="@+id/btnChangeMonitoringState"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="175dp"
            android:layout_marginBottom="10dp"
            android:text="Record monitoring session"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <ScrollView
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_marginTop="2dp"
            android:layout_marginBottom="2dp"
            app:layout_constraintBottom_toTopOf="@id/btnOpenTerminal"

app:layout_constraintTop_toBottomOf="@id/btnChangeMonitoringState">

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical">

                <com.github.mikephil.charting.charts.LineChart
                    android:id="@+id/lcMemory"
                    android:layout_width="match_parent"
                    android:layout_height="100dp" />

                <com.github.mikephil.charting.charts.LineChart
                    android:id="@+id/lcCpu"
                    android:layout_width="match_parent"
                    android:layout_height="100dp" />
```

```
                <com.github.mikephil.charting.charts.LineChart
                    android:id="@+id/lcStorage"
                    android:layout_width="match_parent"
                    android:layout_height="100dp" />

            </LinearLayout>
        </ScrollView>

        <Button
            android:id="@+id/btnOpenTerminal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginBottom="5dp"
            android:text="Open terminal"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toStartOf="@id/btnBrowseFiles"
            app:layout_constraintBottom_toBottomOf="parent"
            />

        <Button
            android:id="@+id/btnBrowseFiles"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginBottom="5dp"
            android:text="Browse files"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toEndOf="@id/btnOpenTerminal"
            />

    </androidx.constraintlayout.widget.ConstraintLayout>

</FrameLayout>
```

Файл drawer_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
```

```
        <group android:checkableBehavior="single">
            <item
                android:id="@+id/serversFragment"
                android:title="Servers" />
            <item
                android:id="@+id/sshKeysFragment"
                android:title="SSH keys" />
            <item
                android:id="@+id/shellScriptsFragment"
                android:title="Scripts" />
            <item
                android:id="@+id/alertsFragment"
                android:title="Alerts" />
        </group>
</menu>
```

Файл nav_graph.xml

```
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/nav_graph"
    app:startDestination="@id/serversFragment">


    <fragment
        android:id="@+id/serversFragment"
        android:name="com.example.servermonitor.fragment.ServersFragment"
        android:label="fragment_servers"
        tools:layout="@layout/fragment_servers" >
        <action
            android:id="@+id/action_serversFragment_to_serverFragment"
            app:destination="@id/serverFragment" />
        <action
            android:id="@+id/action_serversFragment_to_editServerFragment"
            app:destination="@id/editServerFragment" />
    </fragment>
    <fragment
        android:id="@+id/sshKeysFragment"
        android:name="com.example.servermonitor.fragment.SshKeysFragment"
        android:label="fragment_ssh_keys"
        tools:layout="@layout/fragment_ssh_keys" >
```

```
        <action
            android:id="@+id/action_sshKeysFragment_to_editSshKeyFragment"
            app:destination="@id/editSshKeyFragment" />
    </fragment>
    <fragment
        android:id="@+id/serverFragment"
        android:name="com.example.servermonitor.fragment.ServerFragment"
        android:label="fragment_server"
        tools:layout="@layout/fragment_server" >
        <action
            android:id="@+id/action_serverFragment_to_terminalFragment"
            app:destination="@id/terminalFragment" />
        <action

android:id="@+id/action_serverFragment_to_browseServerFilesFragment"
            app:destination="@id/browseServerFilesFragment" />
    </fragment>
    <fragment
        android:id="@+id/terminalFragment"
        android:name="com.example.servermonitor.fragment.TerminalFragment"
        android:label="fragment_terminal"
        tools:layout="@layout/fragment_terminal" />
    <fragment
        android:id="@+id/editServerFragment"
        android:name="com.example.servermonitor.fragment.EditServerFragment"
        android:label="fragment_edit_server"
        tools:layout="@layout/fragment_edit_server" />
    <fragment
        android:id="@+id/editSshKeyFragment"
        android:name="com.example.servermonitor.fragment.EditSshKeyFragment"
        android:label="fragment_edit_ssh_key"
        tools:layout="@layout/fragment_edit_ssh_key" />
    <fragment
        android:id="@+id/shellScriptsFragment"

android:name="com.example.servermonitor.fragment.ShellScriptsFragment"
        android:label="fragment_shell_scripts"
        tools:layout="@layout/fragment_shell_scripts" >
        <action

android:id="@+id/action_shellScriptsFragment_to_editShellScriptFragment"
            app:destination="@id/editShellScriptFragment" />
        <action
```

```
android:id="@+id/action_shellScriptsFragment_to_runScriptFragment"
            app:destination="@id/runScriptFragment" />
    </fragment>
    <fragment
        android:id="@+id/editShellScriptFragment"

android:name="com.example.servermonitor.fragment.EditShellScriptFragment"
        android:label="fragment_edit_shell_script"
        tools:layout="@layout/fragment_edit_shell_script" />
    <fragment
        android:id="@+id/runScriptFragment"
        android:name="com.example.servermonitor.fragment.RunScriptFragment"
        android:label="RunScriptFragment" />
    <fragment
        android:id="@+id/browseServerFilesFragment"

android:name="com.example.servermonitor.fragment.BrowseServerFilesFragment"
        android:label="fragment_browse_server_files"
        tools:layout="@layout/fragment_browse_server_files" />
    <fragment
        android:id="@+id/alertsFragment"
        android:name="com.example.servermonitor.fragment.AlertsFragment"
        android:label="fragment_alerts"
        tools:layout="@layout/fragment_alerts" >
        <action
            android:id="@+id/action_alertsFragment_to_editAlertFragment"
            app:destination="@id/editAlertFragment" />
    </fragment>
    <fragment
        android:id="@+id/editAlertFragment"
        android:name="com.example.servermonitor.fragment.EditAlertFragment"
        android:label="fragment_edit_alert"
        tools:layout="@layout/fragment_edit_alert" />
    <fragment
        android:id="@+id/localFilesFragment"
        android:name="com.example.servermonitor.fragment.LocalFilesFragment"
        android:label="fragment_local_files"
        tools:layout="@layout/fragment_local_files" />
</navigation>
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДДАЛЕНОГО МОНІТОРИНГУ ТА КЕРУВАННЯ СЕРВЕРАМИ

Програма та методика тестування

КПІ.ІТ-9105.045490.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
Олексій СОПОВ

Нормоконтроль:
Максим ГОЛОВЧЕНКО

Виконавець:
Артем ВОВЧЕНКО

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є застосунок для віддаленого моніторингу та керування серверами. Додаток написаний мовою Java для платформи Android.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка коректності збереження даних у відповідності з діями користувача;

– перевірка сумісності додатку з різними версіями Android;

– перевірка коректності відображення додатку на мобільних пристроях з різною роздільною здатністю екрану;

– знаходження проблем, помилок і недоліків з метою їх усунення;

– перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– статичне тестування – перевіряється код програми на предмет дотримання рекомендованих практик написання коду, дублювання коду, наявності потенційних помилок;

– функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;

– системне тестування – перевіряється усе програмне забезпечення в цілому;

– мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

– тестування «білої скриньки» – тестування, що фокусується на дослідженні внутрішньої структури програмного забезпечення для того, щоб переконатися, що внутрішня логіка працює коректно та програма відповідає заявленим вимогам.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування проводиться мануально з використанням наскрізного тестування (End-to-end, E2E) з метою виявлення помилок та недоліків у функціональній частині програмного забезпечення, а також у частині, що стосується інтуїтивності та зручності використання.

Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

– статичне тестування для виявлення очевидних помилок та порушень рекомендованих практик програмування;

– тестування на мобільних пристроях з різною роздільною здатністю екрану;

– тестування на мобільних пристроях з різною версією операційної системи;

– тестування на виведення повідомлень про помилку, коли це необхідно;

– тестування зміни орієнтації екрану;

– тестування поведінки програми у випадку відсутності з’єднання до мережі;

– тестування інтерфейсу користувача;

– тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДДАЛЕНОГО МОНІТОРИНГУ ТА КЕРУВАННЯ СЕРВЕРАМИ

Керівництво користувача

КПІ.ІТ-9105.045490.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
Олексій СОПОВ

Нормоконтроль:
Максим ГОЛОВЧЕНКО

Виконавець:
Артем ВОВЧЕНКО

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Мобільний застосунок для віддаленого моніторингу та керування серверами – це програма, яка спрощує процеси адміністрування та огляду інфраструктури для системних адміністраторів та DevOps інженерів. Програма надає можливість додавати сервери і відслідковувати в реальному часі використання ними системних ресурсів – процесору, пам’яті, диску. Можливості застосунку також включають перегляд файлів на сервері, копіювання файлів на пристрій та з пристрою на сервер, створення сповіщень про перевищення використання ресурсів, запуск фонового моніторингу зі збереженням даних та використання інтерактивної SSH сесії.

Програмне забезпечення доступне для встановлення як APK файл, який можна завантажити на сторінці проєкту на платформі GitHub.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність мобільного пристрою з операційною системою на базі Android з версією 7.0 або вище;

– наявність доступу до Інтернету;

– для встановлення додатку на мобільному пристрої повинно бути не менше 100 МБ вільної пам’яті.

2.2 Завантаження застосунку

На даний момент застосунок можна встановити власноруч, використовуючи відповідний APK файл. Для цього потрібно перейти на сторінку проєкту на платформі GitHub (https://github.com/artem-vovchenko01/android-server-monitoring), перейти до секції з релізами та завантажити APK файл останнього релізу на мобільний пристрій. Далі потрібно встановити цей файл, натиснувши на нього у файловому менеджері та виконавши процедуру встановлення. При цьому, можливо, доведеться ввімкнути налаштування щодо встановлення програм зі сторонніх джерел.

2.3 Перевірка коректної роботи

По завершенню встановлення додатка на робочому столі мобільного пристрою повинна відобразитись піктограма даного застосунку. При натисканні на цю піктограму повинен відкритися головний екран застосунку.

3 ВИКОНАННЯ ПРОГРАМИ

При першому запуску програми на головному екрані (рисунок 3.1) буде відображений порожній список із серверами та кнопка “+” для додавання підключення до сервера.

Рисунок 3.1 – Головна сторінка додатку

Перш ніж додати сервер, користувачу потрібно додати SSH ключ, за допомогою якого він планує підключатися. Для цього потрібно відкрити бокове меню (рисунок 3.2) і обрати вкладку SSH keys. Далі натиснути на аналогічну кнопку “+” для додавання ключа. Вигляд вікна для додавання ключа зображено на рисунку 3.3. Список ключів після створення нового ключа показаний на рисунку 3.4.

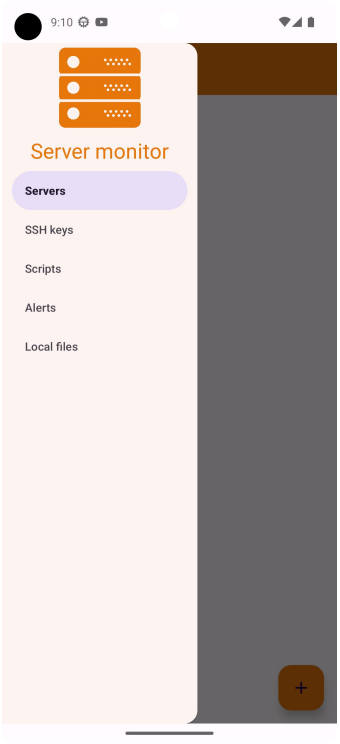


Рисунок 3.2 – Головне меню програми

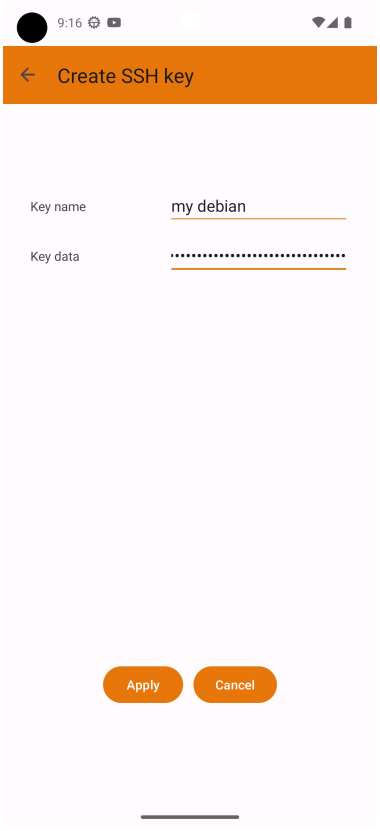


Рисунок 3.3 – Додавання SSH ключа

Рисунок 3.4 – Відображення створеного SSH ключа у списку

Далі користувач має повернутися до екрану з серверами за допомогою головного меню і натиснути на кнопку “+” для додавання нового сервера. Екран для створення нового сервера показаний на рисунку 3.5.

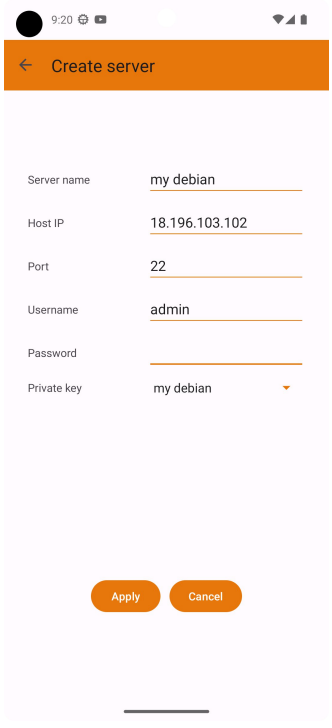


Рисунок 3.5 – Створення сервера

Після створення сервер з’являється у списку (рисунок 3.6). Якщо підключення пройшло успішно, індикатор світиться зеленим кольором та відображаються останні зібрані метрики.

Рисунок 3.6 – Відображення створеного сервера у списку

Щоб перейти на сторінку роботи з сервером (рисунок 3.7), потрібно клацути на потрібний сервер у списку.

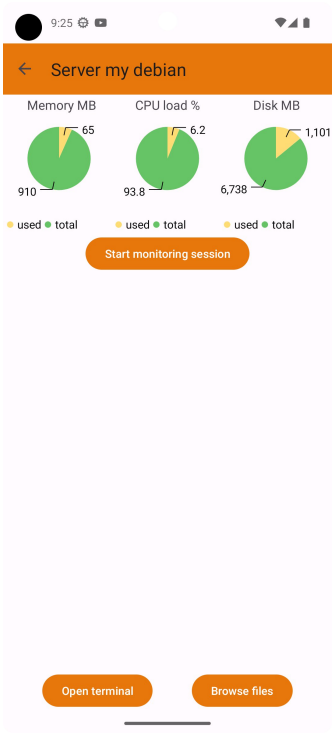


Рисунок 3.7 – Сторінка роботи з сервером

Щоб підключитися до інтерактивного терміналу (рисунок 3.8) для даного сервера, потрібно натиснути кнопку Open terminal.

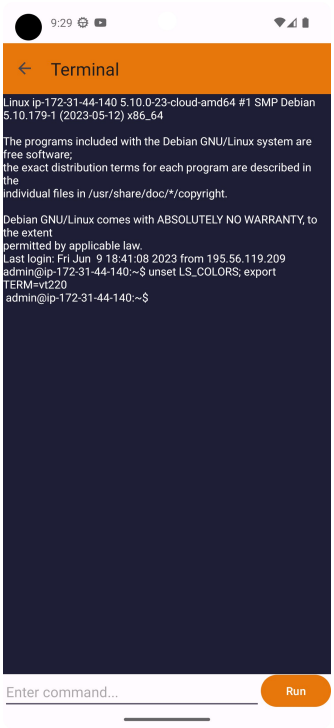


Рисунок 3.8 – Інтерактивна сесія термінала для сервера

Для виконання команди, її текст потрібно ввести в поле внизу екрана і натиснути Run (рисунок 3.9).

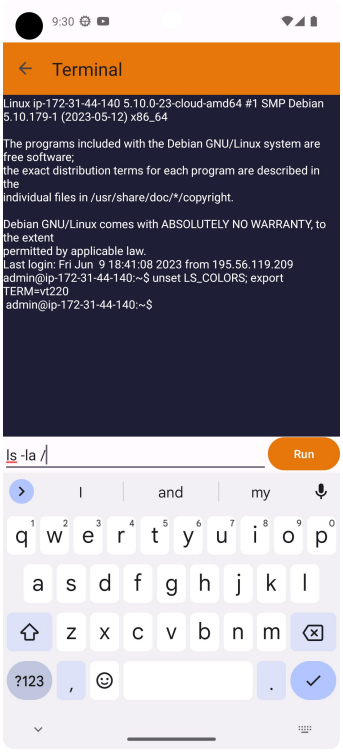


Рисунок 3.9 – Процес введення команди

Приклад результату виконання команди зображений на рисунку 3.10.

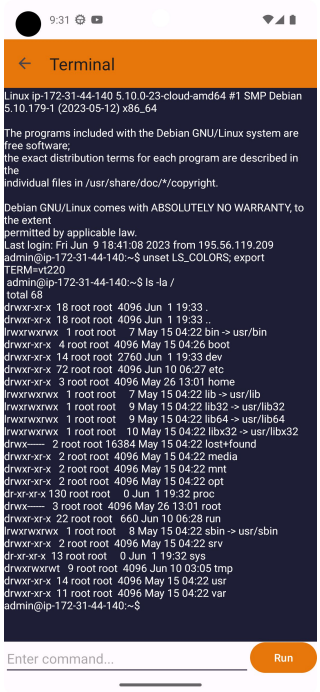


Рисунок 3.10 – Відображення результату виконання команди

Щоб повернутися до екрану для роботи з сервером, потрібно натиснути кнопку Назад.

Для початку фонового моніторингу (рисунок 3.11) потрібно натиснути кнопку Start monitoring session.

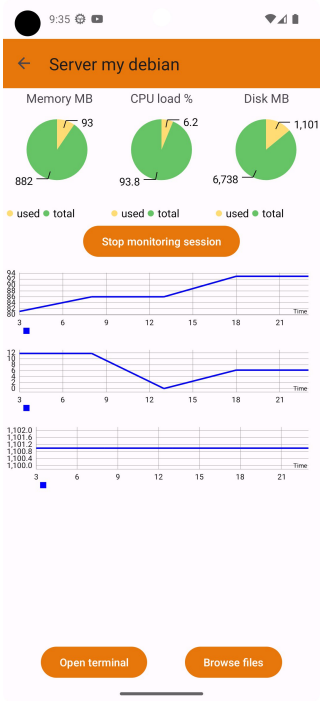


Рисунок 3.11 – Процедура фонового моніторингу

Для збереження резлуьтатів потірбно натиснути Stop monitoring session.

Щоб переглянути список файлів (рисунок 3.12), наявних на серверів, потрібно натиснути Browse files.

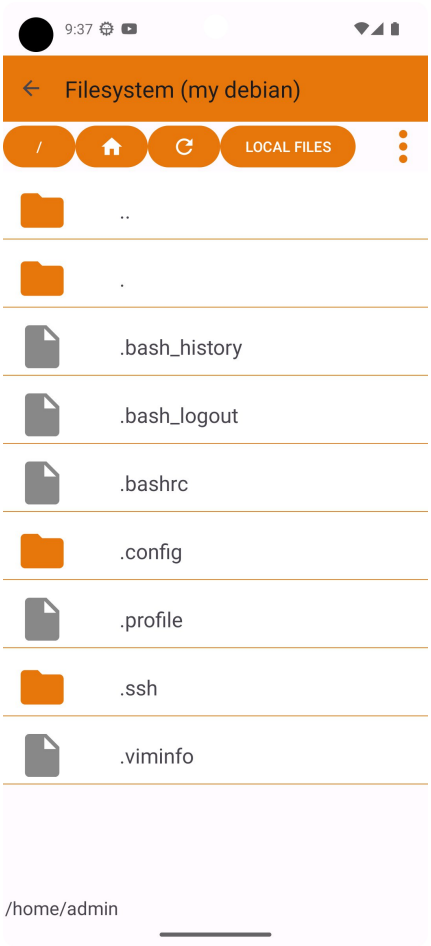


Рисунок 3.12 – Список файлів у домашній директорії

Щоб перейти на директорію вище, треба натиснути файл з назвою “..”. Щоб перейти до кореневої директорії, треба натиснути кнопку з символом “/”. Щоб повернутися до домашньої директорії, натисніть кнопку з символом домівки. Щоб оновити список файлів, натисніть кнопку з символом оновлення.

Щоб створити нову директорію, натисніть кнопку меню (має вигляд трьох вертикальних крапок) і оберіть пункт New directory (рисунок 3.13). Відобразиться діалогове вікно для введення назви директорії (рисунок 3.14). Створена директорія відобразиться у списку (рисунок 3.15).

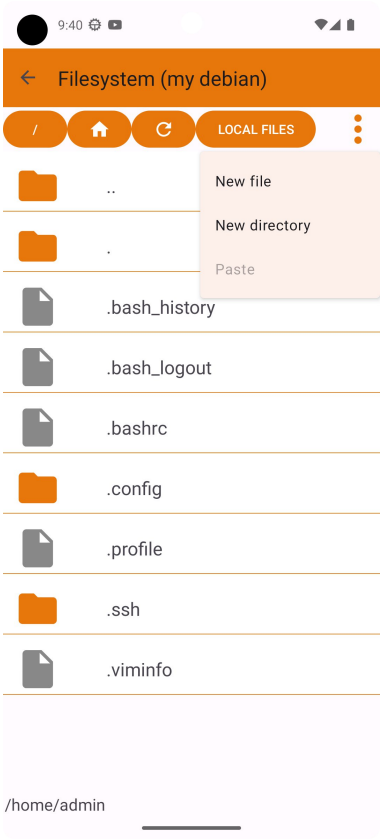


Рисунок 3.13 – Вигляд меню

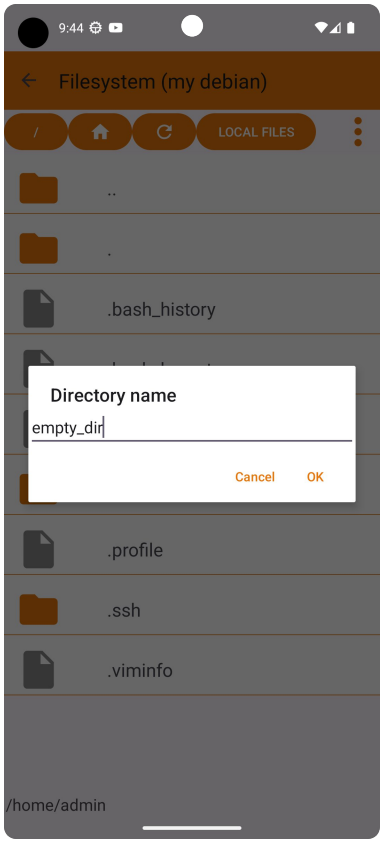


Рисунок 3.14 – Введення назви для нової директорії

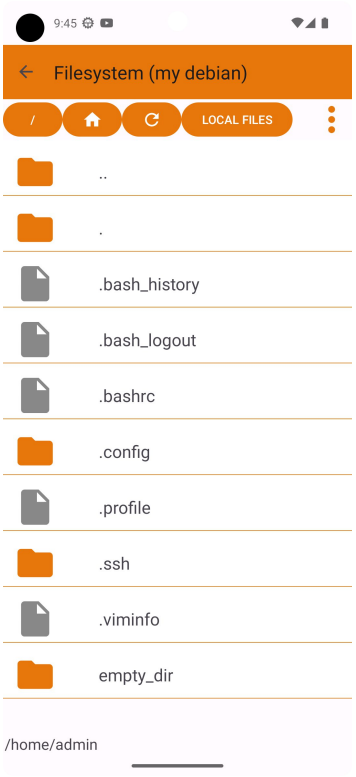


Рисунок 3.15 – Відображення створеної директорії

Щоб копіювати файл до локального сховища, потрібно натиснути довгим тапом на потрібний файл і обрати пункт Copy (рисунок 3.16).

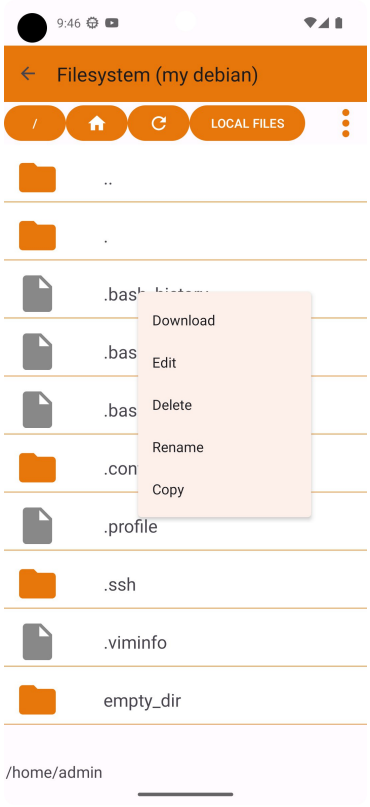


Рисунок 3.16 – Меню роботи з файлом

Далі потрібно перейти до локального сховища, натиснувши Local files, відкрити меню і обрати Paste (рисунок 3.17). Вставлений файл відобразиться у списку ((рисунок 3.18).

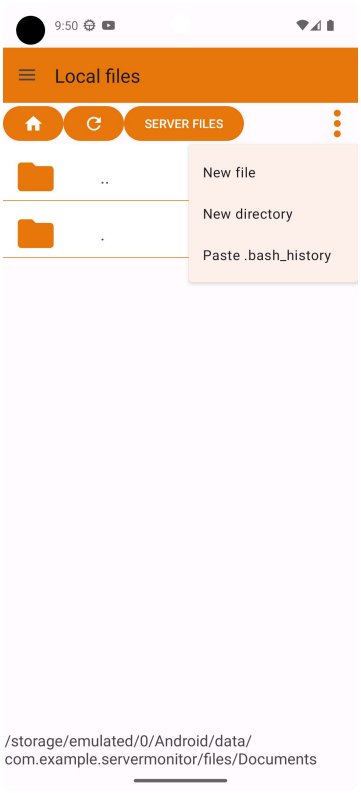


Рисунок 3.17 – Меню роботи з файлом

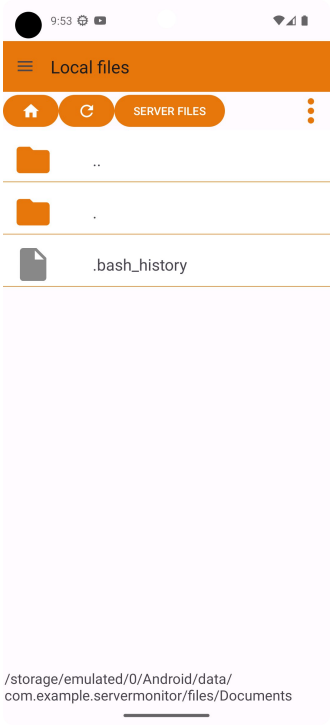


Рисунок 3.18 – Відображення вставленого файлу

Для роботи зі скриптами треба відкрити головне меню та обрати Scripts. Натисніть “+”, щоб створити новий скрипт. Відкриється вікно для введення даних для скрипта ((рисунок 3.19). Після підтвердження введення, новий скрипт з’явиться у списку (рисунок 3.20).

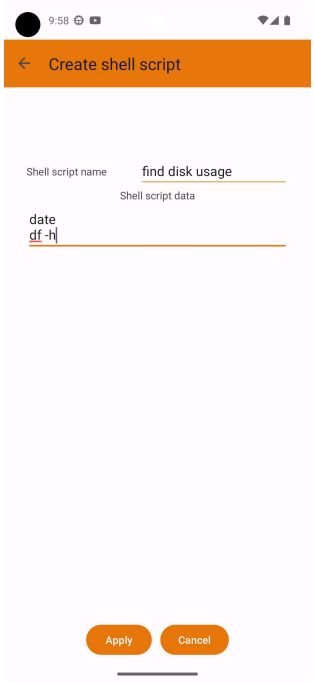


Рисунок 3.19 – Створення shell скрипта

Рисунок 3.20 – Відображення shell скрипта у списку

Щоб виконати скрипт, потрібно на нього клацнути. Відкриється вікно з вибором серверів (рисунок 3.21), на яких скрипт потрібно виконати.

Потрібно обрати необхідні сервери за допомогою натискання на відповідні чекбокси.

Рисунок 3.21 – Процес вибору серверів для запуску скрипта

Щоб запустити скрипт на обраних серверах, натисніть Run. Скрипт виконається, і біля відповідних серверів з’являться кнопки Show output (рисунок 3.22).

Рисунок 3.22 – Вигляд списку серверів після виконання скрипта

Натисніть Show output біля потрібного сервера для перегляду результату виконання скрипта (рисунок 3.23).

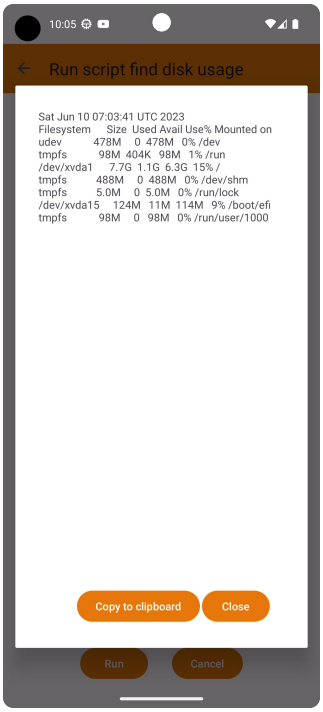


Рисунок 3.23 – Перегляд результатів виконання скрипта

Для створення сповіщення, відкрийте головне меню та натисніть Alerts. Натисніть “+” для створення нового сповіщення. Після заповнення даних у вікні, що з’явилось (рисунок 3.24), та підтвердження, скрипт додасться.

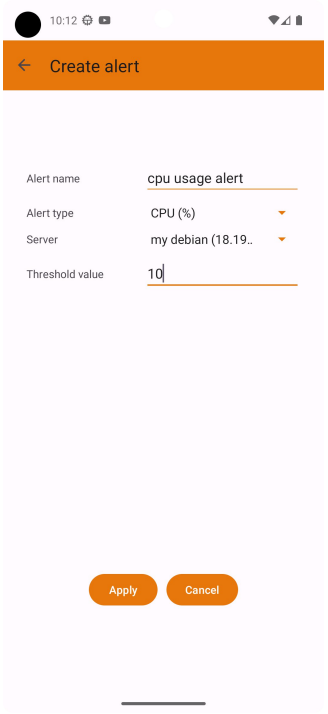


Рисунок 3.24 – Створення сповіщення

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДДАЛЕНОГО МОНІТОРИНГУ ТА КЕРУВАННЯ СЕРВЕРАМИ

Схема структурна варіантів використання

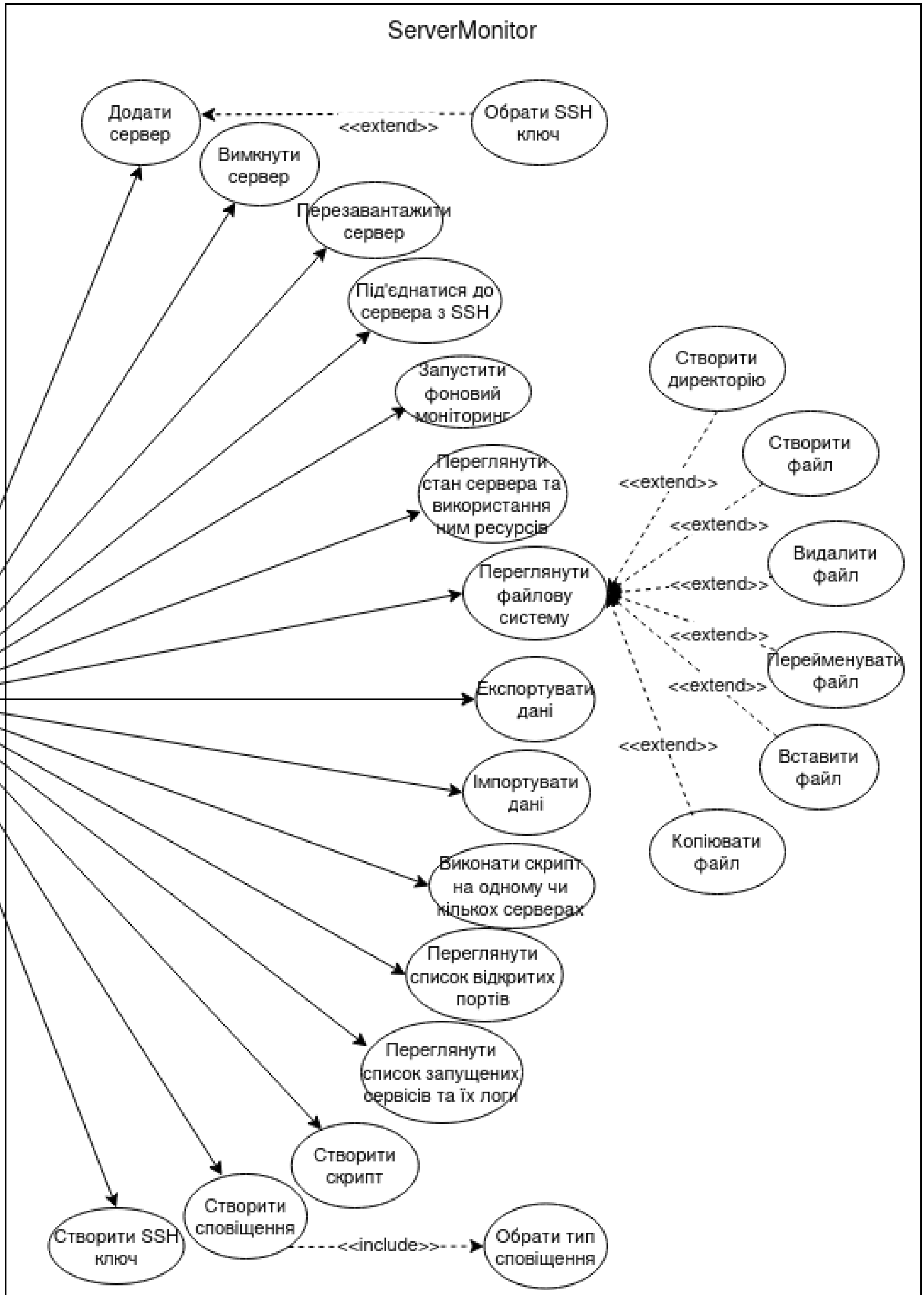
КПІ.ІТ-9105.045490.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
Олексій СОПОВ

Нормоконтроль:
Максим ГОЛОВЧЕНКО

Виконавець:
Артем ВОВЧЕНКО

Київ – 2023



					КПІ.ІТ-9105.045490.06.99.ССВ	
					Схема структурна варіантів використань	Лит. / Арк. / Аркушів: 1
Зм.	Арк.	№ докум.	Підп.	Дата		Аркуш / Аркушів
Розроб.		Вовченко А.Є.			Мобільний застосунок для віддаленого моніторингу та керування серверами	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-91
Перев.		Сопов О.О.				
Т. Кон.						
Н. Кон.		Головченко М.М.				
Затв.		Жаріков Е.В.				

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДДАЛЕНОГО МОНІТОРИНГУ ТА КЕРУВАННЯ СЕРВЕРАМИ

Схема бази даних

КПІ.ІТ-9105.045490.07.99

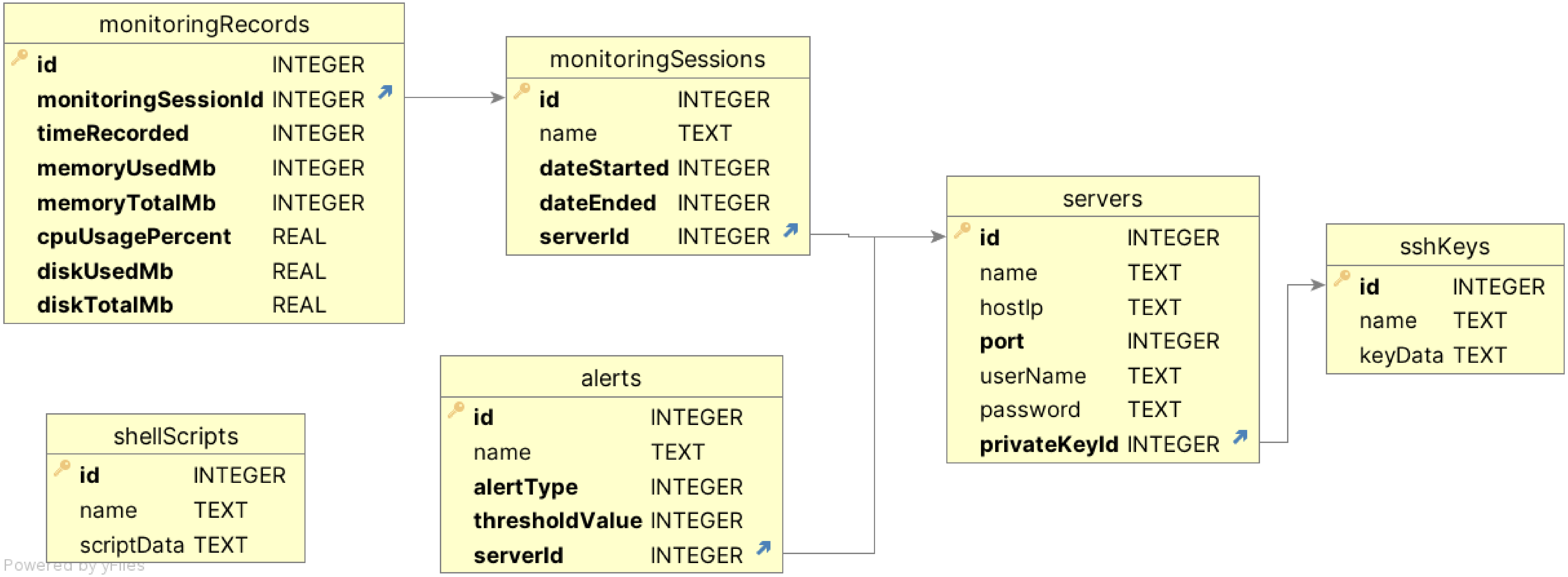
“ПОГОДЖЕНО”
Керівник проєкту:
Олексій СОПОВ

Нормоконтроль:
Максим ГОЛОВЧЕНКО

Виконавець:
Артем ВОВЧЕНКО

Київ – 2023

КПІ.ІТ-9105.045490.07.99.СБД



Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІТ-9105.045490.07.99.СБД		
Розробив		Вовченко А.Є.			Схема бази даних	Літера	Маса / Масштаб: 1
Перевірив		Сопов О.О.				Аркуш	Аркушів
Т. кон.					Мобільний застосунок для віддаленого моніторингу та керування серверами	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-91	
Н. кон.		Головченко М.М.					
Затвердив		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДДАЛЕНОГО МОНІТОРИНГУ ТА КЕРУВАННЯ СЕРВЕРАМИ

Креслення вигляду екранних форм

КПІ.ІТ-9105.045490.08.99

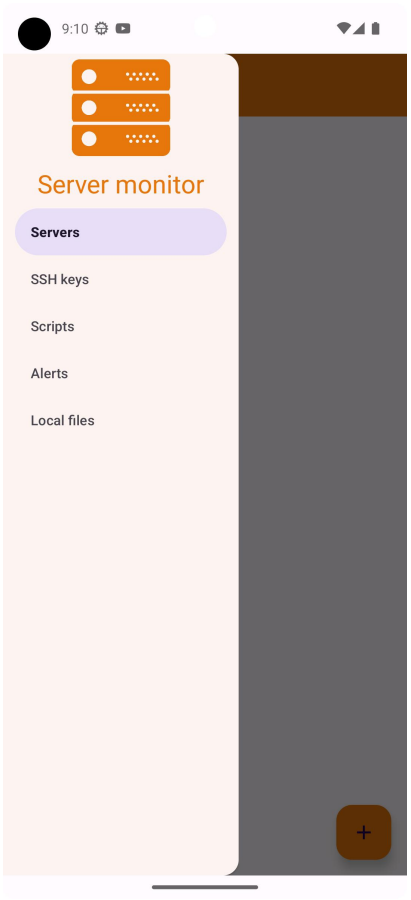
“ПОГОДЖЕНО”
Керівник проєкту:
Олексій СОПОВ

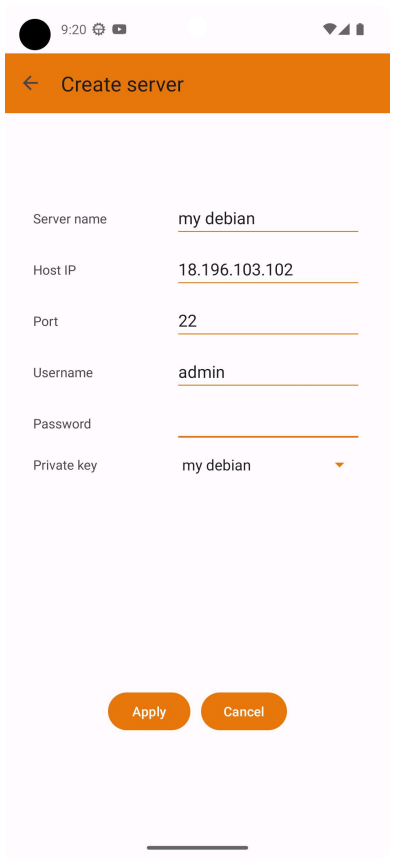
Нормоконтроль:
Максим ГОЛОВЧЕНКО

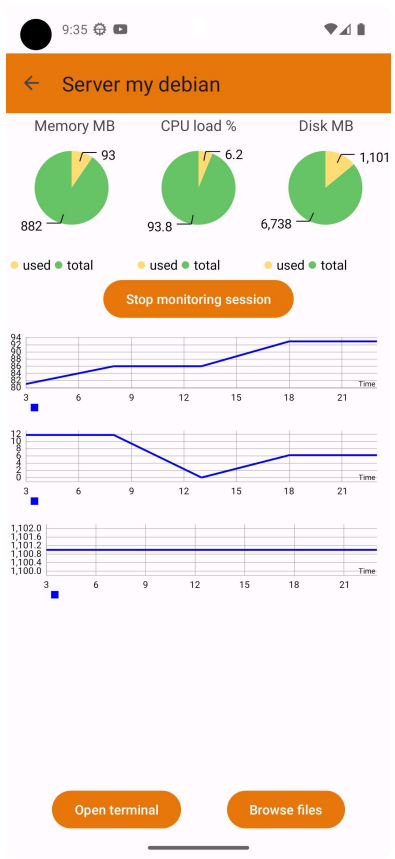
Виконавець:
Артем ВОВЧЕНКО

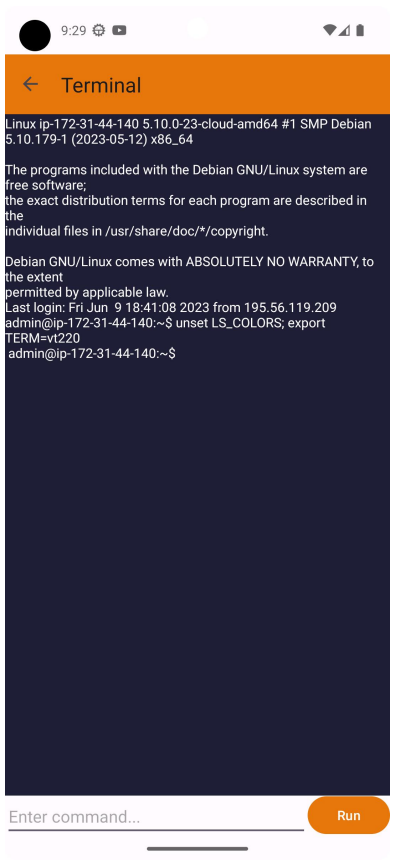
Київ – 2023

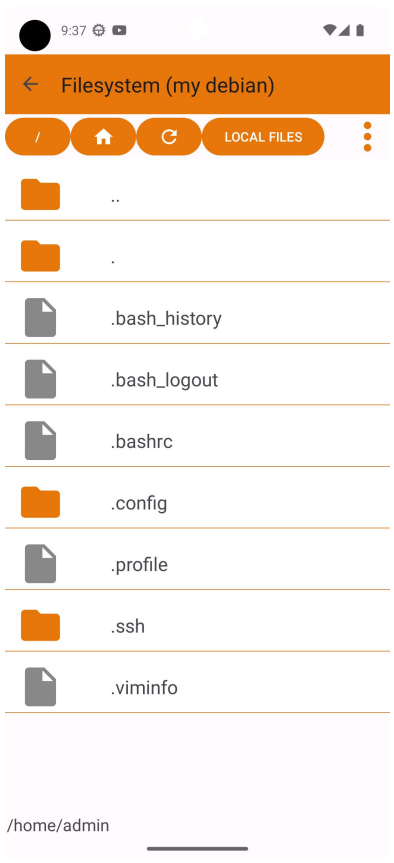
КПІ.ІТ-9105.045490.08.99.КЕ

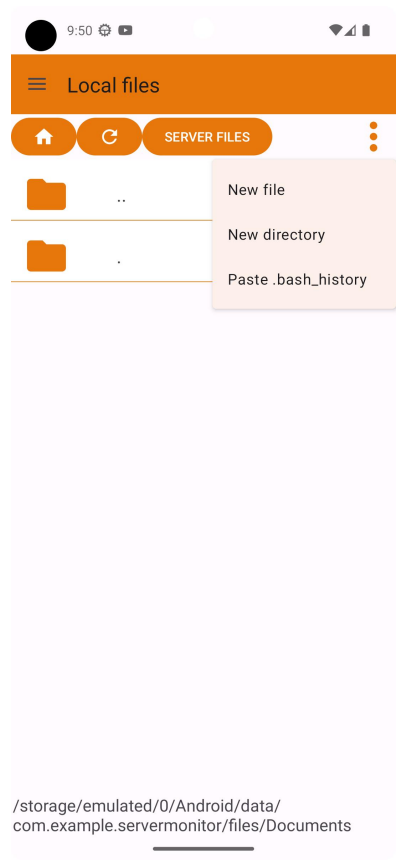


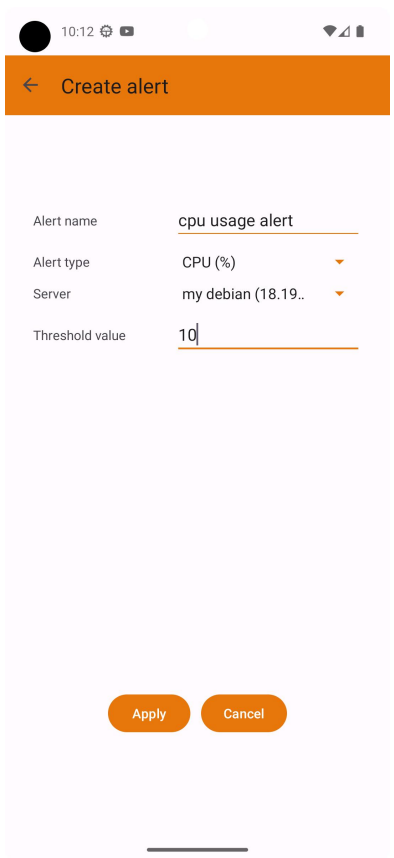


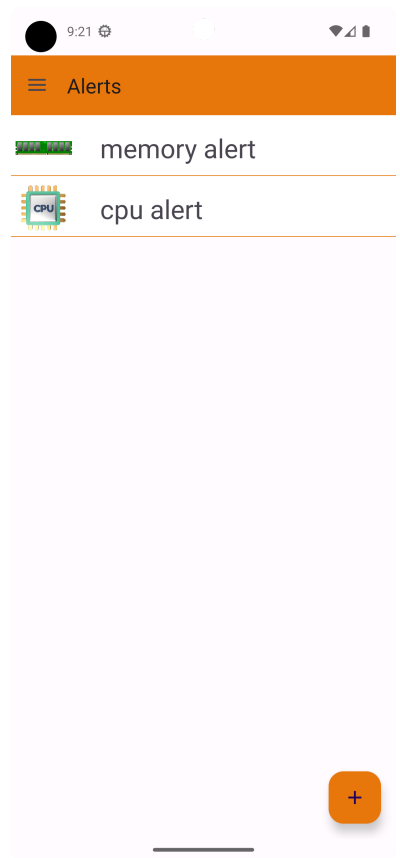












					КПІ.ІТ-9105.045490.08.99.КЕ
Зм.	Арк.	№ докум.	Підп.	Дата	Креслення вигляду екранних форм
Розроб.		Вовченко А.Є.			Лит. / Арк. / Аркушів: 1
Перев.		Сопов О.О.			Аркуш / Аркушів
Т. Кон.					Мобільний застосунок для віддаленого моніторингу та керування серверами
Н. Кон.		Головченко М.М.			КПІ ім.Ігоря Сікорського, Кафедра ІПІ, гр. ІТ-91
Затв.		Жаріков Е.В.			