МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп'ютерних наук та кібербезпеки

МАТЮХІН ВАДИМ АНДРІЙОВИЧ

АНАЛІЗ ТА ВИКОРИСТАННЯ API VIRTUALBOX ДЛЯ СТВОРЕННЯ GUI ДЛЯ POCKET MOODLE

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
БУЛАТЕЦЬКИЙ ВІТАЛІЙ ВІКТОРОВИЧ,
доцент кафедри комп'ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол № _____
засідання кафедри_____
від_____2025 р.
Завідувач кафедри
(_____)_____
(підпис) ПІБ

Луцьк 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

GUI — Graphical User Interface (Графічний інтерфейс користувача)

VM — Virtual Machine (Віртуальна машина)

API — Application Programming Interface (Інтерфейс програмування додатків)

VBox — VirtualBox (Програмне забезпечення для віртуалізації)

LMS — Learning Management System (Система управління навчанням)

IVMManager — Interface for Virtual Machine Manager (Інтерфейс для менеджера віртуальних машин)

DI — Dependency Injection (Ін’єкція залежностей)

ViewModel — Модель-представлення (зв’язок між даними та інтерфейсом користувача)

FluentTheme — Стиль для інтерфейсу користувача

VMManager — Менеджер віртуальних машин

ВСТУП

Актуальність теми обумовлена зростаючою потребою в інтеграції віртуалізації у сферу освіти та управління інформаційними системами. Віртуалізація дозволяє ефективно використовувати ресурси серверів і знижувати витрати на інфраструктуру, що особливо важливо для навчальних установ, де часто виникає необхідність у використанні різноманітних операційних систем для проведення лабораторних робіт, тестів та інших завдань.

У роботі [1] представлено концепцію та реалізацію портативної системи Pocket Moodle, яка базується на віртуальній машині з гостьовою ОС Ubuntu Server, розгорнутій у середовищі Oracle VirtualBox. Дана система показала високу стабільність, простоту використання та мобільність, що дозволяє педагогічним працівникам переносити її між різними мережами та пристроями без необхідності втручання в її роботу. Проте, попри технічну ефективність такого рішення, його використання може бути ускладнене відсутністю зручного графічного інтерфейсу для управління та взаємодії з Pocket Moodle, особливо для користувачів без глибоких IT-компетенцій. Отже, створення інтуїтивно зрозумілого графічного інтерфейсу, який би спрощував запуск, налаштування та взаємодію з системою, є логічним кроком у напрямі підвищення її доступності та практичного використання в освітньому середовищі. Це підкреслює актуальність дослідження та розробки зручного графічного інтерфейсу для Pocket Moodle, який розширить її потенціал і сприятиме впровадженню в закладах освіти різного рівня. Система Pocket Moodle GUI, яка поєднає можливості віртуалізації та інтеграції з платформою Moodle, дозволить створити універсальний інтерфейс для управління віртуальними машинами, що є основою для успішного навчання та наукових досліджень.

Мета роботи полягає в розробці графічного інтерфейсу користувача для системи Pocket Moodle, що інтегрує можливості VirtualBox через використання API, а також у реалізації зручного інтерфейсу для взаємодії з платформою

Moodle, забезпечуючи повний контроль за віртуальними машинами та їх ресурсами.

Завданнями роботи є:

- розробка архітектури програми для взаємодії з VirtualBox та Moodle;
- створення графічного інтерфейсу користувача, який надає зручний доступ до основних функцій управління віртуальними машинами;
- інтеграція API VirtualBox для отримання та управління метриками віртуальних машин;
- розробка тестових сценаріїв для перевірки функціональності та продуктивності програми;
- аналіз ефективності програми на основі тестів продуктивності та функціональних характеристик.

Об’єктом дослідження є програмне забезпечення, що інтегрує VirtualBox API для керування віртуальними машинами та взаємодії з навчальною платформою Moodle.

Предметом дослідження є процес розробки графічного інтерфейсу користувача та програмних компонентів, що забезпечують інтеграцію системи Pocket Moodle з VirtualBox і Moodle для покращення процесу навчання через віртуалізацію.

Публікації:

Матюхін В., Булатецький В. Аналіз способів взаємодії з VirtualBox для автоматизації інфраструктури. *Проблеми комп’ютерних наук, програмного моделювання та безпеки цифрових систем*.: матеріали II міжнар. науково-практ. конф. Луцьк, 9–11 червн. 2025 р. Луцьк, 2025. С. 90-91

РОЗДІЛ 1
ТЕХНОЛОГІЇ ВІРТУАЛІЗАЦІЇ НА ОСНОВІ ORACLE VIRTUALBOX ПРИ РЕАЛІЗАЦІЇ ПОРТАТИВНИХ LMS

1.1 Використані технології

Система управління навчанням (LMS) Moodle є одним із найпоширеніших рішень для організації дистанційного навчання, що характеризується відкритим кодом, підтримкою розширень і широким функціональним охопленням. Однак традиційна модель розгортання Moodle передбачає використання серверної інфраструктури з доступом до глобальної мережі, що створює низку обмежень у випадках, коли доступ до хостингових платформ або інтернет-з'єднання є нестабільним чи відсутнім [2].

У рамках розробки програмного забезпечення Pocket Moodle, яке базується на готовому образі віртуальної машини з встановленим Moodle-середовищем, було поставлено завдання створити умови для автономної, швидкої та повторюваної інсталяції LMS [3] у локальному середовищі користувача. Відповідно, до середовища розгортання системи висуваються наступні функціональні та технічні вимоги.

1. Автономність. Система повинна бути здатною до повноцінної роботи в умовах відсутності підключення до Інтернету, функціонуючи виключно в межах локальної або VPN-мережі. Це є критично важливим, зокрема в умовах аварійних ситуацій або обмеженого доступу до зовнішніх хостинг-сервісів.

2. Простота розгортання. Кінцевий користувач (викладач або адміністратор) не повинен володіти спеціальними знаннями в області налаштування серверів, баз даних або вебсервісів. Розгортання повинно виконуватись за допомогою імпорту попередньо налаштованого образу віртуальної машини у форматі .ova, що містить усі необхідні компоненти: серверну ОС, вебсервер Apache, СУБД MySQL, PHP, VPN-адаптер і власне LMS Moodle [4].

3. Портативність**.** Уся система має зберігатись у вигляді одного або кількох файлів, що легко копіюються на інші пристрої. Після імпорту та запуску віртуальної машини, конфігурація мережі й серверного середовища має виконуватись автоматично або вимагати мінімального втручання з боку користувача. Це дозволяє переносити платформу з одного комп'ютера на інший без необхідності її повторного налаштування.

4. Кросплатформеність. Платформа повинна функціонувати незалежно від операційної системи основного пристрою користувача (Windows, Linux, MacOS). Це означає, що середовище розгортання повинно підтримуватись універсальним віртуалізаційним програмним забезпеченням, сумісним з усіма основними ОС.

5. Мережева інтеграція. Віртуальна машина повинна отримувати IP-адресу безпосередньо з мережевого середовища користувача (через проміжний адаптер), що забезпечить можливість звернення до неї через браузер як з локальної, так і з віддаленої машини. Для роботи через Інтернет передбачається використання VPN-з'єднання з сервером навчального закладу або викладача.

6. Мінімальне споживання ресурсів. Віртуальне середовище повинно бути оптимізоване під використання на звичайних персональних комп'ютерах. У проєкті Pocket Moodle використано серверну версію Ubuntu без графічного інтерфейсу, що дозволило забезпечити стабільну роботу LMS навіть при 1 GB оперативної пам'яті, а також розширити кількість одночасних підключень без значного навантаження на систему.

Система розгортання Moodle у вигляді готової віртуальної машини дозволяє забезпечити стабільне функціонування навчального середовища як у межах локальної мережі, так і з віддаленим доступом. Вибір такої архітектури зумовлений необхідністю зменшення технічного порогу входу для користувачів, підвищення мобільності, а також забезпечення можливості автономної роботи в умовах, що змінюються. Водночас, задоволення зазначених вимог стало основою для подальшої реалізації програмної бібліотеки взаємодії з VirtualBox та розробки графічного інтерфейсу, що автоматизує процеси імпорту, запуску та керування віртуальним середовищем Moodle [5].

Для реалізації системи Pocket Moodle необхідно було обрати платформу віртуалізації, яка б відповідала вимогам до портативності, простоти використання, сумісності з різними операційними системами та підтримки мережевої взаємодії. Серед найпопулярніших засобів віртуалізації на сьогоднішній день можна виокремити такі програмні продукти, як VMware Workstation, Microsoft Hyper-V, QEMU, Parallels та Oracle VirtualBox [6]. Вибір конкретного інструменту повинен базуватись на технічних, функціональних і практичних критеріях, що визначають зручність використання та доцільність застосування у межах навчального процесу.

За результатами порівняльного аналізу, проведеного в межах дослідження, було встановлено, що Oracle VirtualBox найкраще відповідає специфіці використання системи Pocket Moodle [7]. По-перше, VirtualBox — це повністю безкоштовне та відкрите програмне забезпечення, що не потребує жодних ліцензій або підписок навіть для комерційного використання. Це особливо актуально в освітньому середовищі, де обмеженість фінансових ресурсів часто є перешкодою до впровадження платних технологій.

По-друге, Oracle VirtualBox підтримує всі основні операційні системи як на стороні хосту (Windows, Linux, MacOS), так і на стороні гостьових систем (у тому числі Ubuntu Server [8], що використовується у даному проєкті), що забезпечує високу гнучкість при розгортанні середовища Moodle на різних пристроях. Така кросплатформеність дозволяє розробленій віртуальній машині функціонувати практично на будь-якому комп’ютері користувача, не вимагаючи адаптації під конкретне ПЗ чи «залізо».

Окремої уваги заслуговує підтримка формату .ova — відкритого стандарту для імпорту/експорту повністю налаштованих віртуальних машин. Це дає змогу підготувати один типовий образ із уже встановленою ОС, серверним оточенням, Moodle та необхідним ПЗ (VPN-клієнт, SSH-сервер тощо) та швидко розгорнути його на будь-якому комп’ютері через механізм імпорту [9]. Цей підхід суттєво знижує технічний бар’єр для викладачів і студентів, адже не потребує

встановлення чи налаштування окремих компонентів вручну — усе вже включено у заздалегідь сконфігуровану збірку.

Ще однією перевагою Oracle VirtualBox є підтримка так званого «проміжного адаптера» (Bridged Adapter) — режиму мережевої взаємодії, який дозволяє віртуальній машині отримати IP-адресу безпосередньо з маршрутизатора локальної мережі. Таким чином, система Moodle, яка працює всередині ВМ, стає доступною для всіх пристроїв у тій же мережі, ніби вона розгорнута на фізичному сервері. Це є важливою перевагою для локального або віддаленого доступу до платформи, зокрема при організації навчального процесу в аудиторіях або за умов роботи через VPN [10].

Також варто відзначити, що VirtualBox має CLI-інтерфейс (VBoxManage), який, хоча і не є повноцінним API у звичному сенсі, забезпечує доступ до широкого спектру функціональності — від запуску й зупинки ВМ до збору системних метрик. Це дозволяє будувати надбудови, зокрема графічні інтерфейси або служби автоматизації, на основі викликів CLI-команд, як це реалізовано в розробленій бібліотеці VBoxAPI у даному проєкті [11].

Oracle VirtualBox у межах поставленого завдання має низку істотних переваг: відкритий код, відсутність ліцензійних обмежень, підтримка імпорту/експорту .ova, широка сумісність із ОС, гнучке налаштування мережевих параметрів, а також можливість автоматизації через CLI. Усі ці характеристики дозволили забезпечити швидке розгортання системи Pocket Moodle, її ефективну роботу у локальних і віддалених мережах та розширюваність через спеціалізоване ПЗ, що було реалізоване на мові C# у вигляді графічного клієнта для користувача.

У процесі створення системи Pocket Moodle [12], яка має забезпечити швидке, зручне та надійне розгортання платформи дистанційного навчання Moodle, було прийнято рішення використати віртуалізаційне середовище Oracle VirtualBox як основну технологічну базу. Такий вибір дозволив сформувати архітектуру, що відповідає ключовим вимогам до портативності, автономності

та простоти використання в умовах обмеженого доступу до інтернету або сторонніх серверних ресурсів.

Однією з центральних ідей реалізації є створення наперед підготовленого образу віртуальної машини у форматі .ova, який включає повністю налаштоване середовище для запуску Moodle. Цей образ містить операційну систему Ubuntu Server 20.04, попередньо встановлений та налаштований стек LAMP (Apache, MySQL, PHP), сервер SSH для віддаленого адміністрування, Webmin для графічного моніторингу, а також інсталяцію LMS Moodle [13] з відповідною конфігурацією. Образ підтримує мережеву взаємодію як через локальну мережу, так і за допомогою VPN, що дає змогу користувачам підключатись до Moodle ззовні, зберігаючи при цьому гнучкість розгортання.

Особливу увагу було приділено мережевим налаштуванням віртуальної машини. Використання режиму «Проміжний адаптер» (Bridged Adapter) у VirtualBox дозволяє кожній ВМ отримати повноцінну IP-адресу в тій же мережі, де знаходиться фізичний хост. Це означає, що користувачі можуть звертатись до LMS Moodle безпосередньо через браузер, використовуючи IP-адресу ВМ, як до реального сервера. Такий підхід значно спрощує доступ до платформи з боку студентів та викладачів, а також дозволяє ефективно організувати доступ до Moodle [14] у комп’ютерних класах, лабораторіях чи через VPN-інфраструктуру навчального закладу.

Віртуальна машина також включає автоматизоване VPN-з’єднання, яке активується під час запуску системи. При цьому Moodle стає доступною в глобальній мережі, навіть якщо її фізичне розміщення обмежено локальними умовами. Для роботи з такою системою користувачам достатньо лише імпортувати .ova-образ через графічний інтерфейс VirtualBox або через спеціалізований застосунок, що був розроблений у межах цього дослідження з використанням мови програмування C# та бібліотеки VBoxManage. Подальше керування ВМ (вмикання, вимикання, перегляд стану, моніторинг ресурсів) відбувається автоматично або через зручний графічний інтерфейс, без необхідності використання стандартного інтерфейсу VirtualBox.

Окремим етапом реалізації стала розробка проміжного програмного забезпечення, яке забезпечує взаємодію користувача з VirtualBox безпосередньо через графічний інтерфейс Pocket Moodle GUI. Це програмне забезпечення використовує CLI-інтерфейс VBoxManage [15], що дозволяє ініціювати запуск віртуальної машини, отримувати перелік доступних ВМ, переглядати метрики (використання процесора, пам’яті), а також виконувати базові дії адміністрування. Завдяки реалізації асинхронного оброблення процесів у C# та правильному парсингу результатів CLI-команд, користувач отримує інтуїтивно зрозумілий інтерфейс без необхідності працювати з консоллю або знати специфіку командної взаємодії з VirtualBox.

Використання VirtualBox у проєкті Pocket Moodle дозволило забезпечити запуск LMS Moodle на будь-якому ПК без налаштування серверів, перенесення системи у вигляді одного .ova-файлу, доступ через локальну мережу або VPN, а також централізоване та спрощене керування через спеціальний GUI.

Ця реалізація не лише забезпечила відповідність сучасним вимогам до портативних освітніх рішень, але й створила основу для подальшої автоматизації, масштабування та використання у різних навчальних сценаріях, з мінімальним технічним навантаженням на викладача чи адміністратора системи.

1.2 Дослідження роботи з бібліотекою API VirtualBox

Для створення програмного інтерфейсу до віртуалізованого середовища VirtualBox у межах проєкту Pocket Moodle було досліджено всі наявні способи програмної взаємодії з віртуальними машинами. Відповідно до офіційної документації VirtualBox та практичного тестування, можна виділити три основні API-інтерфейси, які потенційно придатні для інтеграції з користувацьким застосунком на платформі .NET.

1. COM/XPCOM API — низькорівневий інтерфейс для прямої взаємодії з ядром VirtualBox [16].

2. Web API (XML-RPC) — серверний API [17], доступний через HTTP-запити.

3. CLI (VBoxManage) — командний рядок, що дозволяє запускати всі адміністративні операції у вигляді зовнішніх процесів [18].

Кожен з інтерфейсів має свої переваги та обмеження, які потрібно враховувати під час проєктування бібліотеки взаємодії з VirtualBox. Нижче подано порівняльну таблицю 1.1 основних характеристик зазначених API.

Таблиця 1.1

Порівняння способів взаємодії з VirtualBox

Параметр	**COM/XPCOM API**	**Web API (XML-RPC)**	**CLI (VBoxManage)**
Повнота функціоналу	Повний доступ до всіх можливостей VirtualBox	Частковий доступ, частина функцій недоступна або не документована	Повний функціонал через окремі CLI-команди
Сумісність із .NET/C#	Ускладнена. Відсутні готові бібліотеки, складне підключення через IDL	Відсутні офіційні бібліотеки для C#, структура орієнтована на Python	Висока — можна запускати процеси з парсингом результатів
Документація та приклади	Обмежена. Більшість прикладів для C++/Java	Вкрай обмежена. Неофіційні джерела, застарілі URI	Повна документація, численні приклади, стабільні версії
Надійність/стабільність	Високий ризик нестабільної поведінки при генерації C#-обгорток	Нестабільно, часто повертає помилки HTTP 404 або невідповідні типи	Надійна, стабільна поведінка CLI-команд
Підтримка асинхронності	Обмежена, потребує ручної реалізації	Обмежена. HTTP-запити не завжди відповідають у визначений час	Підтримка через Process API та async/await у .NET
Складність реалізації	Висока. Необхідна генерація COM-інтерфейсів вручну, труднощі з налагодженням	Висока. Недокументовані методи, зворотне інжинірингування	Помірна. Потрібен парсинг виводу, але реалізація пряма і передбачувана
Можливість кросплатформенності	Обмежена (залежить від COM/IDL механізмів ОС)	Потенційно широка, але недопрацьована	Висока — CLI доступне на всіх ОС, де встановлено VirtualBox

Аналіз показав, що COM/XPCOM API, хоча й забезпечує найповніший доступ до внутрішнього API VirtualBox, на практиці є малопридатним для інтеграції з .NET [19]. Зокрема, при спробі згенерувати обгортки C# на основі IDL-описів спостерігалися критичні проблеми сумісності, відсутність документації, нестабільна компіляція типів і відсутність прикладів використання. Більшість офіційних матеріалів орієнтовані на C++ або Java, що суттєво ускладнює розробку.

Web API у формі XML-RPC-сервера виглядає привабливо з погляду віддаленого керування, але його використання на практиці виявилось надзвичайно обмеженим. Зокрема, сервер вимкнений за замовчуванням, вимагає ручного запуску, а документація відсутня. Частина запитів не працює, повертаючи HTTP-помилки (404, 500), що свідчить про відсутність зворотної сумісності між версіями або про експериментальний характер API. Крім того, структура відповіді не узгоджується з типізацією C# і часто орієнтована на скриптові мови типу Python.

Натомість CLI-інтерфейс (VBoxManage) виявився найбільш стабільним, зрозумілим і практичним для використання у .NET-додатках. Усі основні дії — створення, запуск, зупинка, видалення ВМ, а також отримання інформації про ресурси — реалізуються за допомогою окремих CLI-команд, які легко обробляються в C# через клас Process. Незважаючи на потребу в ручному парсингу текстового виводу, саме цей інтерфейс став основою для створеної бібліотеки VBoxAPI, що реалізує асинхронну взаємодію з VirtualBox та дозволяє інтегрувати її в користувацький інтерфейс GUI Pocket Moodle [20].

За сукупністю технічних та практичних критеріїв CLI був визнаний найбільш доцільним варіантом взаємодії з VirtualBox у межах розробки прикладного програмного забезпечення для локального розгортання та адміністрування віртуального навчального середовища.

1.3 Аналіз наявних аналогів

Для визначення унікальності та конкурентних переваг системи Pocket Moodle GUI було проведено аналіз наявних програмних засобів, які забезпечують управління віртуальними машинами (ВМ) та інтеграцію з навчальними платформами, такими як Moodle. Основними критеріями порівняння обрано функціональність управління ВМ, кросплатформність, інтеграцію з Moodle, зручність користувацького інтерфейсу (UX) та підтримку автоматизації. Розглянуто три аналоги: VirtualBox Manager (вбудований GUI VirtualBox), Vagrant і Proxmox VE. Кожен із них має свої сильні та слабкі сторони, які порівнюються з можливостями Pocket Moodle GUI.

VirtualBox Manager є стандартним графічним інтерфейсом, що постачається разом із VirtualBox. Він дозволяє створювати, налаштовувати, запускати та зупиняти віртуальні машини, а також моніторити базові метрики продуктивності, такі як використання CPU і RAM. Інтерфейс підтримує базові операції, такі як імпорт/експорт образів (.ova), управління снапшотами та налаштування мережевих параметрів.

Переваги:

- тісна інтеграція з VirtualBox, що забезпечує повний доступ до всіх функцій платформи;
- простий і зрозумілий інтерфейс для базового управління ВМ;
- кросплатформність (підтримує Windows, Linux, macOS);
- безкоштовність і відкритий вихідний код.

Недоліки:

- відсутність прямої інтеграції з навчальними платформами, такими як Moodle, що обмежує його використання в навчальних середовищах;
- обмежені можливості автоматизації та скриптування, що ускладнює масове розгортання ВМ;
- моніторинг метрик продуктивності є базовим і не підтримує потокове оновлення в реальному часі;

- інтерфейс не оптимізовано для складних сценаріїв, таких як одночасне керування кількома ВМ у навчальному контексті.

Vagrant — це інструмент для автоматизації створення та управління віртуальними машинами, який часто використовується в розробці програмного забезпечення. Він підтримує VirtualBox як один із провайдерів і дозволяє налаштовувати ВМ через декларативні конфігураційні файли (Vagrantfile). Vagrant фокусується на автоматизації розгортання середовищ для тестування та розробки.

Переваги:

- потужна автоматизація через Vagrantfile, що дозволяє швидко створювати та налаштовувати ВМ;
- підтримка кількох провайдерів віртуалізації (VirtualBox, VMware, Hyper-V), що забезпечує гнучкість;
- активна спільнота та багата документація;
- можливість інтеграції з інструментами оркестрації, такими як Ansible або Puppet.

Недоліки:

- орієнтація на командний рядок, що робить його менш зручним для користувачів без технічних навичок;
- відсутність вбудованого графічного інтерфейсу, що обмежує доступність для викладачів і студентів;
- немає прямої інтеграції з Moodle або іншими навчальними платформами;
- складність налаштування для нестандартних сценаріїв, таких як моніторинг продуктивності в реальному часі.

Proxmox Virtual Environment (Proxmox VE) — це платформа з відкритим вихідним кодом для управління віртуалізацією, яка підтримує KVM і LXC, а також має вебінтерфейс для адміністрування ВМ і контейнерів. Вона орієнтована на серверні середовища та може бути використана для розгортання навчальних платформ.

Переваги:

- Потужний вебінтерфейс із підтримкою моніторингу продуктивності в реальному часі.
- Підтримка кластеризації та високої доступності, що корисно для великих навчальних центрів.
- Можливість інтеграції з зовнішніми системами через API.
- Безкоштовна базова версія з відкритим вихідним кодом.

Недоліки:

- орієнтація на серверне розгортання, що вимагає значних ресурсів і складного налаштування;
- відсутність прямої підтримки VirtualBox, що робить його несумісним із проєктом Pocket Moodle GUI;
- складний інтерфейс для початківців, що може бути перешкодою для викладачів і студентів;
- обмежена інтеграція з Moodle без додаткових плагінів або налаштувань.

Для чіткого порівняння аналогів із Pocket Moodle GUI наведено таблицю, яка оцінює їх за ключовими критеріями.

Таблиця 1.2.

Порівняння аналогів Pocket Moodle GUI

Критерій/Аналог	**VirtualBox Manager**	**Vagrant**	**Proxmox VE**	**Pocket Moodle GUI**
Управління ВМ	Базове	Автоматизоване	Розширене	Розширене
Кросплатформ-ність	Часткова (Windows, Linux)	Присутня	Немає	Часткова (Windows, Linux)
Інтеграція з Moodle	Немає	Немає	Обмежена	Присутня
Графічний інтерфейс	Присутній	Немає	Присутній (веб)	Присутній
Моніторинг продуктивності	Базовий	Немає	Розширений	Розширений
Автоматизація	Відсутня	Присутня	Присутня	Присутня
Зручність для користувачівок	Висока	Низька	Середня	Висока
Підтримка VirtualBox CLI	Присутня	Присутня	Відсутня	Присутня

Аналіз аналогів показує, що жоден із розглянутих засобів не поєднує всіх необхідних функцій для ефективного управління віртуальними машинами в навчальному середовищі з інтеграцію з Moodle. VirtualBox Manager є зручним для базового використання, але не підтримує автоматизацію чи інтеграцію з навчальними платформами. Vagrant потужний для автоматизації, але його орієнтація на командний рядок і відсутність GUI роблять його непридатним для викладачів і студентів. Proxmox VE пропонує розширені можливості, але його складність і невідповідність VirtualBox обмежують його використання в контексті Pocket Moodle.

Pocket Moodle GUI вирізняється унікальним поєднанням кросплатформного графічного інтерфейсу, інтеграції з Moodle, розширеного моніторингу продуктивності в реальному часі та автоматизації через бібліотеку VBoxAPI. Використання CLI VirtualBox дозволяє забезпечити повний контроль над ВМ, а асинхронна модель із гнучким парсером і кешуванням забезпечує оптимальну продуктивність. Реактивний інтерфейс на основі Avalonia UI і ReactiveUI забезпечує зручність і швидкість взаємодії, що робить систему ідеальною для навчальних середовищ, де потрібна простота, стабільність і автоматизація.

Таким чином, Pocket Moodle GUI заповнює прогалину між базовими інструментами управління ВМ і спеціалізованими навчальними платформами, пропонуючи зручне, автоматизоване та інтегроване рішення, яке відповідає потребам сучасного віртуального навчання.

РОЗДІЛ 2
РОЗРОБКА GUI ДЛЯ POCKET MOODLE

2.1 Постановка задачі та призначення

У результаті аналізу існуючих технічних рішень, досвіду використання платформи VirtualBox, а також особливостей розгортання навчального середовища Moodle у форматі портативної системи Pocket Moodle було виявлено ряд актуальних проблем, пов’язаних з управлінням віртуальними машинами. Серед них — складність у використанні інструментів керування для користувачів без технічної підготовки, відсутність зручного інтерфейсу для типових дій з віртуальними машинами, а також потреба в автоматизації процесу запуску та контролю стану середовища Moodle.

Платформа Oracle VirtualBox, попри свою функціональність і відкритість, не забезпечує готового рішення для створення легкого, інтуїтивного інтерфейсу керування віртуальними машинами. Особливої складності набувають такі дії, як імпорт .ova-файлів, налаштування мережевого режиму (наприклад, "Bridged Adapter"), запуск і зупинка машин, визначення IP-адреси або перегляд продуктивності системи.

Враховуючи ці обмеження, виникла потреба у розробці спеціалізованого прикладного програмного забезпечення, яке забезпечить зручне, автоматизоване й безпечне керування віртуальним середовищем Pocket Moodle. Це програмне забезпечення має стати надбудовою над VirtualBox, використовуючи наявні інструменти керування CLI (VBoxManage), але надаючи користувачу повністю графічний інтерфейс для взаємодії з системою.

Відтак, мета кваліфікаційної роботи полягає у розробці бібліотеки для роботи з CLI VirtualBox та створенні графічного інтерфейсу для управління віртуальними машинами Pocket Moodle на основі платформи .NET та фреймворку Avalonia.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- дослідити можливості інтеграції VirtualBox у C#-середовище через CLI-інтерфейс VBoxManage та розробити обгортку у вигляді керованої бібліотеки VBoxAPI, що забезпечує асинхронну взаємодію з віртуальними машинами;
- розробити механізм виявлення, імпорту та запуску віртуальних машин із готових .ova-файлів, уникаючи необхідності ручного втручання користувача;
- реалізувати інтерфейс керування віртуальним середовищем, який надає доступ до основних операцій: запуск, зупинка, перевірка статусу машини, перегляд IP-адреси;
- інтегрувати в інтерфейс моніторинг використання системних ресурсів віртуальної машини (CPU, RAM) у реальному часі;
- забезпечити зворотний зв’язок у GUI у вигляді повідомлень про стан операцій, помилки, інструкцій для подальших дій користувача.

Робота повинна об’єднати в собі програмну реалізацію бібліотеки взаємодії з VirtualBox, створення GUI-інтерфейсу для зручного користування, а також забезпечити технічні й методичні засади використання створеного інструменту в системах типу Pocket Moodle. Це рішення повинно сприяти ширшому впровадженню LMS Moodle у навчальний процес за умов обмежених технічних ресурсів, підвищуючи доступність освітніх технологій у незалежному локальному форматі.

Розроблення окремої бібліотеки взаємодії з VirtualBox на базі CLI-інтерфейсу VBoxManage стало ключовим завданням проєкту. Система взаємодіє з VBoxManage через запуск зовнішніх процесів у фоновому режимі. Для інкапсуляції цієї взаємодії створено сервісний рівень, який включає інтерфейс IVMManager та його реалізацію VMManager. Це забезпечує гнучкий зв’язок між ViewModel і функціоналом CLI.

У процесі розробки застосунку Pocket Moodle GUI першочерговим завданням стало проєктування архітектури, яка дозволить реалізувати інтуїтивний інтерфейс для керування віртуальними машинами VirtualBox, не

вимагаючи від користувача знання CLI-команд чи роботи з конфігураціями віртуалізатора. Система мала бути максимально автономною, кросплатформенною та модульною, щоби спростити майбутнє супроводження та масштабування.

Під час проєктування інтерфейсу для керування віртуальними машинами в рамках системи Pocket Moodle було визначено потребу в реалізації окремої підсистеми, яка б дозволяла відслідковувати технічні характеристики виконання ВМ у режимі реального часу. Така функціональність має практичну цінність як для моніторингу продуктивності під час роботи Moodle-сервера, так і для своєчасного виявлення перевантаження, проблем із мережею або недоступності віртуального середовища.

2.2 Вибір моделі розробки програмного засобу

Для розроблення системи Pocket Moodle GUI було необхідно обрати модель розробки, яка б відповідала вимогам проєкту, включаючи кросплатформність, складність інтеграції з CLI-інтерфейсом VirtualBox, необхідність реактивного інтерфейсу та можливість поступового розширення функціоналу. Після аналізу різних моделей розробки — каскадної, ітеративної, спіральної та гнучких методологій (Agile) — було обрано спіральну модель, яка забезпечує баланс між систематичним плануванням, ітеративним удосконаленням і управлінням ризиками.

Спіральна модель, запропонована Баррі Боемом, базується на циклічному процесі, що включає чотири основні фази: визначення цілей, аналіз ризиків, розроблення та тестування, а також оцінка результатів із плануванням наступної ітерації. Ця модель була обрана через її здатність адаптуватися до складних проєктів із високим рівнем невизначеності, що є характерним для Pocket Moodle GUI. Основними факторами вибору стали потреба в поетапній реалізації складних компонентів, таких як бібліотека VBoxAPI для взаємодії з VirtualBox,

і необхідність ітеративного тестування для забезпечення стабільності та сумісності з різними версіями VirtualBox і операційними системами.

На першій ітерації спіральної моделі було проведено аналіз вимог і ризиків, пов’язаних із інтеграцією з CLI VirtualBox. Зокрема, основними ризиками вважалися можливі зміни у форматі виводу команд VBoxManage між версіями, нестабільність асинхронної обробки тривалих операцій, таких як моніторинг метрик, і складність створення кросплатформеного інтерфейсу. Для їхнього мінімізації було вирішено розробити гнучкий парсер для CLI-виводу, використовувати асинхронну модель із підтримкою CancellationToken і обрати фреймворк Avalonia UI для забезпечення кросплатформності. На цій фазі також створено прототип бібліотеки VBoxAPI, який включав базові операції, такі як отримання списку віртуальних машин (VBoxManage list vms) і запуск машини (startvm).

Наступні ітерації спіральної моделі дозволили поступово розширювати функціонал. Наприклад, друга ітерація зосередилася на реалізації підсистеми збору метрик продуктивності з використанням команди VBoxManage metrics collect і розробленні класу MetricParser для обробки потокового виводу. Ця ітерація включала тестування стабільності при довготривалому моніторингу та оптимізацію продуктивності через кешування часто використовуваних метрик. Третя ітерація була присвячена розробленню реактивного інтерфейсу користувача на основі Avalonia UI та ReactiveUI, що забезпечило динамічне оновлення вкладок і зручну навігацію. Кожна ітерація супроводжувалася оцінкою результатів і коригуванням плану для усунення виявлених недоліків, таких як затримки при запуску кількох віртуальних машин, які вирішувалися шляхом впровадження асинхронного виконання.

Спіральна модель виявилася оптимальною завдяки її здатності враховувати ризики на ранніх етапах і дозволявати ітеративне вдосконалення. На відміну від каскадної моделі, яка передбачає лінійний процес і погано підходить для проєктів із невизначеністю в реалізації (наприклад, залежність від CLI VirtualBox), спіральна модель дала змогу поетапно тестувати критичні

компоненти, такі як асинхронна обробка команд і парсинг даних. Порівняно з гнучкими методологіями, такими як Scrum, спіральна модель краще підходить для технічно складних проєктів, де потрібен ретельний аналіз ризиків перед кожною ітерацією, а не лише фокус на швидкій доставці функціоналу.

Додатковою перевагою спіральної моделі стало її відповідність вимогам масштабованості та розширюваності системи. Наприклад, можливість додавання нових функцій, таких як підтримка снапшотів або інтеграція з іншими платформами віртуалізації (VMware, Hyper-V), була передбачена на етапі планування завдяки модульній архітектурі бібліотеки VBoxAPI та чіткому поділу відповідальностей у патерні MVVM. Кожна ітерація дозволяла уточнювати вимоги, тестувати нові компоненти та забезпечувати сумісність із навчальним середовищем Moodle.

Таким чином, вибір спіральної моделі розробки для Pocket Moodle GUI обґрунтовано її гнучкістю, здатністю управляти ризиками та підтримкою ітеративного розвитку, що дозволило створити стабільну, масштабовану та зручну систему. Ця модель забезпечила поетапну реалізацію складних компонентів, таких як асинхронна взаємодія з VirtualBox і реактивний інтерфейс, а також дала змогу адаптуватися до змін вимог і технічних обмежень протягом розроблення.

2.3 Загальний опис проєкту

Проєкт Pocket Moodle GUI розроблено для створення зручного та ефективного інструменту управління віртуальними машинами VirtualBox у навчальному середовищі, з інтеграцією з платформою Moodle. Основною метою було забезпечити кросплатформений, стабільний і реактивний застосунок, який дозволяє користувачам керувати віртуальними машинами, моніторити їхню продуктивність і отримувати доступ до навчальних ресурсів через інтуїтивний графічний інтерфейс.

Архітектура системи побудована на багаторівневій моделі з чітким поділом відповідальностей, що забезпечує гнучкість, тестованість і легкість супроводу. Верхній рівень — графічний інтерфейс користувача, створений за допомогою фреймворку Avalonia UI, який підтримує кросплатформність на Windows і Linux. Для ізоляції логіки представлення від бізнес-логіки застосовано архітектурний шаблон MVVM (Model–View–ViewModel). ViewModel-шар виступає посередником між інтерфейсом (XAML Views) і програмною логікою, забезпечуючи реактивну взаємодію та оновлення даних у реальному часі. Нижній рівень системи включає бібліотеку VBoxAPI, яка інкапсулює взаємодію з CLI-інтерфейсом VirtualBox (VBoxManage), ізолюючи складність роботи з зовнішніми процесами від решти застосунку.

Бібліотека VBoxAPI спроєктована як універсальний низькорівневий модуль для взаємодії з VirtualBox через CLI-команди. Вона забезпечує виконання операцій, таких як отримання списку віртуальних машин, збір метрик продуктивності, керування станом машин (запуск, зупинка, перезавантаження) та імпорт образів .ova. Клас VBoxManager є центральним компонентом бібліотеки, відповідаючи за виклик CLI-команд, аналіз їхнього виводу та формування об’єктів для подальшої обробки на вищих рівнях. Його архітектура підтримує асинхронне виконання, обробку помилок і можливість розширення функціоналу, наприклад, додавання підтримки снапшотів або експорту машин, без зміни існуючої структури. Модульність VBoxAPI дозволяє ізолювати CLI-залежності, що спрощує інтеграцію нових функцій і забезпечує стабільність системи.

Підсистема збору метрик продуктивності спроєктована для безперервного моніторингу стану віртуальних машин, включаючи рівень завантаження процесора, обсяг споживаної пам’яті, швидкість операцій із диском, мережевий обмін, час безперервної роботи та кількість активних процесів. Для цього використовується команда VBoxManage metrics collect, яка забезпечує періодичний збір даних із заданим інтервалом. Дані передаються асинхронно через IAsyncEnumerable до ViewModel, що дозволяє оновлювати інтерфейс у

реальному часі без блокування основного потоку. У графічному інтерфейсі метрики відображаються у вигляді таблиці або блоку з підписаними полями, прив’язаними до ViewModel через механізм DataContext, що гарантує автоматичне оновлення без ручного опитування.

Графічний інтерфейс користувача побудовано як багатовкладкову систему з уніфікованим головним вікном, що забезпечує логічну сегментацію функцій і просту навігацію. Структура вікна включає ліву панель зі списком віртуальних машин, реалізованим через ListBox із прив’язкою до властивості VMs у ViewModel, і правий блок із вкладками (TabControl), що відповідають за керування машиною, перегляд метрик, доступ до файлової системи, Moodle і вбудованого терміналу. Кожна вкладка активується лише за наявності вибраної віртуальної машини, а її вміст оновлюється динамічно через прив’язку до ViewModel. Наприклад, вибір машини в списку змінює властивість SelectedVM, що автоматично оновлює всі пов’язані вкладки через механізм RaisePropertyChanged. Кнопки керування (запуск, зупинка, перезавантаження) прив’язані до ICommand-властивостей у VMViewModel, які викликають методи IVMManager, а для тривалих операцій передбачено індикатори активності (IsBusy).

Інтерфейс спроектовано з урахуванням принципів UX-дизайну, орієнтованих на зручність і швидкість. Текст неактивних вкладок відображається сірим кольором, активна вкладка виділяється синьою лінією, а всі функції доступні в одному вікні, що зменшує когнітивне навантаження. Запуск віртуальної машини здійснюється одним кліком, а редагування конфігурації (кількість ядер, пам’ять, мережеві інтерфейси) у вкладці Config передається через MachineConfigViewModel до API VirtualBox. Вкладки Filesystem і Moodle надають доступ до внутрішніх ресурсів віртуального середовища через спільні папки або вбудовані сервіси, а вкладка Terminal дозволяє вводити команди для гостьової чи хостової системи з відображенням результатів у реальному часі.

Типові сценарії взаємодії користувача включають вибір віртуальної машини зі списку, що активує вкладки та оновлює їхній вміст, керування

машиною (запуск, зупинка, перезавантаження), перегляд метрик у реальному часі, редагування конфігурації, доступ до файлової системи, Moodle або терміналу. Усі дії виконуються реактивно завдяки використанню бібліотеки ReactiveUI та патерну MVVM, що забезпечує швидку реакцію інтерфейсу та чіткий зворотний зв’язок. Система підтримує асинхронне виконання команд, стабільність при довготривалій роботі та мінімальну залежність між шарами, що робить її універсальною для використання в навчальних середовищах.

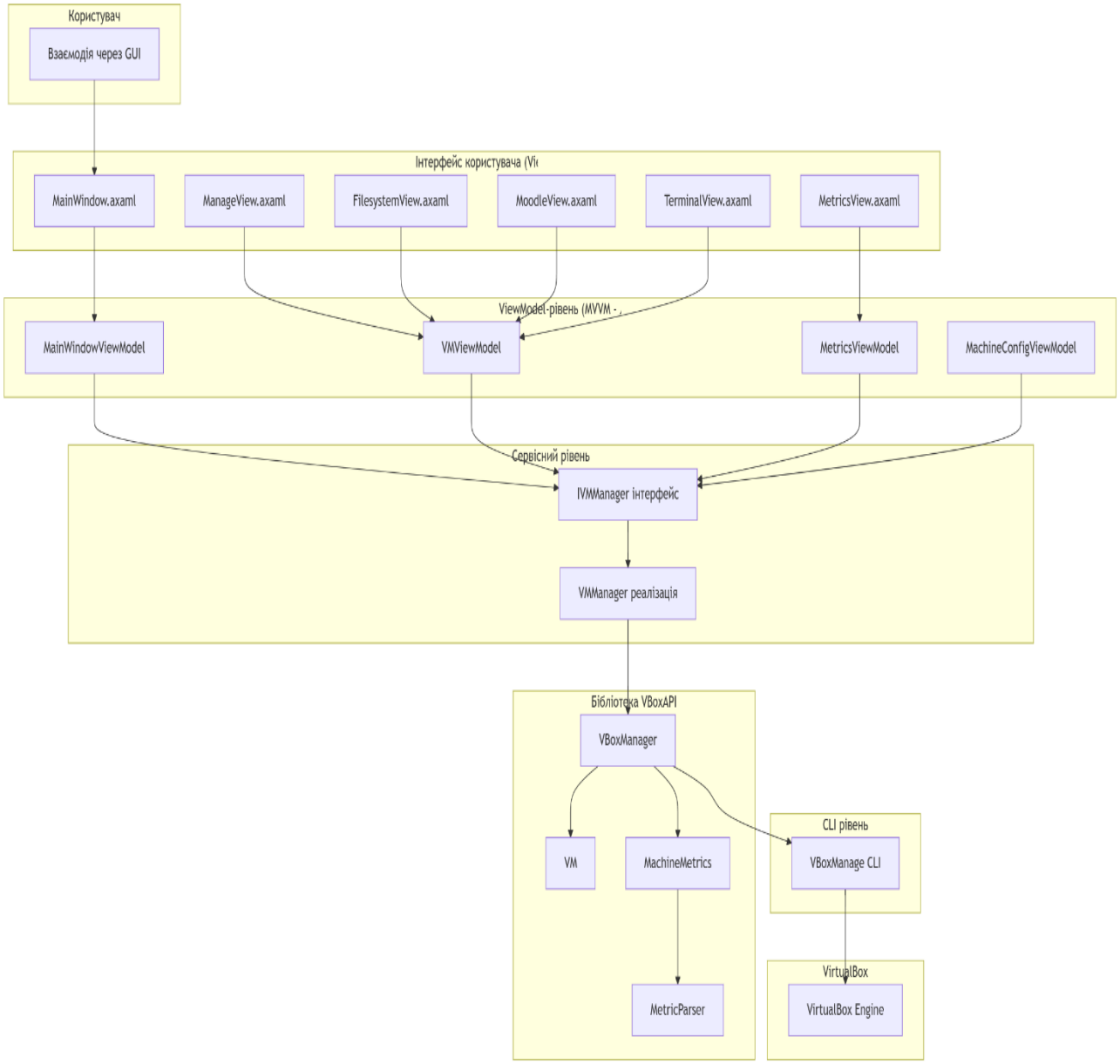

Рисунок 2.1 – Основні компоненти застосунку

2.4 Обґрунтування вибору інструментів засобів розробки

Розроблення системи Pocket Moodle GUI вимагало вибору інструментів, які б забезпечували ефективну взаємодію з віртуальним середовищем VirtualBox, стабільність роботи застосунку, кросплатформність і зручний користувацький інтерфейс. Основними компонентами технологічного стеку стали платформа C# .NET, фреймворк Avalonia UI для створення графічного інтерфейсу та CLI-інтерфейс VirtualBox (VBoxManage) для взаємодії з віртуальними машинами. Вибір цих інструментів ґрунтується на їхній надійності, гнучкості та відповідності вимогам проєкту.

Платформа C# .NET була обрана як основа для розроблення Pocket Moodle GUI завдяки її потужним можливостям для створення структурованих, масштабованих і кросплатформених застосунків. .NET забезпечує розвинену екосистему бібліотек і інструментів, що спрощують роботу з асинхронними операціями, управлінням процесами та обробкою даних. Зокрема, простір імен System.Diagnostics надає зручні засоби для запуску та керування зовнішніми процесами, що є критично важливим для взаємодії з CLI VirtualBox. Крім того, C# підтримує сучасні парадигми програмування, такі як асинхронне виконання через Task API (async/await), що дозволяє уникнути блокування основного потоку застосунку під час виконання тривалих операцій, таких як запуск віртуальних машин чи збирання метрик. Кросплатформна підтримка .NET, особливо в останніх версіях, забезпечує можливість розгортання застосунку на Windows і Linux, що відповідає вимогам навчального середовища Pocket Moodle.

Ще однією перевагою C# .NET є розвинена підтримка об’єктно-орієнтованого програмування та патернів, таких як MVVM, що використовується в проєкті для ізоляції логіки від інтерфейсу. Наявність бібліотек, таких як ReactiveUI, дозволяє реалізувати реактивну модель даних, що забезпечує миттєве оновлення інтерфейсу при зміні стану. Крім того, .NET має потужні інструменти для тестування (наприклад, xUnit) і налагодження, що полегшує розроблення та супровід системи. Вибір C# .NET також обґрунтовано

широкою спільнотою розробників і багатою документацією, що знижує ризики технічних проблем і спрощує розширення функціоналу в майбутньому.

Для створення графічного інтерфейсу обрано фреймворк Avalonia UI, який забезпечує кросплатформну підтримку та сучасний підхід до розроблення інтерфейсів на основі XAML. Avalonia UI дозволяє створювати однаковий вигляд і поведінку застосунку на Windows, Linux і macOS, що є важливим для розгортання Pocket Moodle GUI в різних навчальних середовищах. На відміну від інших фреймворків, таких як WPF, який прив’язаний до Windows, або UWP з обмеженою функціональністю, Avalonia UI пропонує гнучкість і незалежність від платформи, що відповідає вимогам кросплатформності проєкту.

Avalonia UI підтримує декларативну розмітку через XAML, що спрощує створення складних інтерфейсів, таких як багатовкладкова структура Pocket Moodle GUI з динамічним оновленням даних. Інтеграція з ReactiveUI та патерном MVVM дозволяє реалізувати реактивний інтерфейс, де зміни в ViewModel автоматично відображаються в інтерфейсі через механізм Binding і INotifyPropertyChanged. Наприклад, вибір віртуальної машини чи оновлення метрик миттєво оновлює вкладки без затримок. Крім того, Avalonia UI підтримує інструменти дизайну, такі як <Design.DataContext>, що полегшує розроблення інтерфейсу у Visual Studio, дозволяючи відображати демо-дані під час проєктування. Ці можливості, разом із легкістю масштабування та підтримки сучасних принципів UX-дизайну, зробили Avalonia UI оптимальним вибором для створення інтуїтивного та продуктивного інтерфейсу.

Для взаємодії з віртуальним середовищем VirtualBox було проаналізовано три доступні API: COM/XPCOM API, Web API (XML-RPC) і CLI-інтерфейс VBoxManage. Аналіз показав, що CLI-інтерфейс є найбільш стабільним і практичним для інтеграції з .NET. COM/XPCOM API, хоча й забезпечує повний доступ до функціоналу VirtualBox, виявився складним у реалізації через відсутність готових бібліотек для C#, проблеми з генерацією обгорток і обмежену документацію, орієнтовану на C++ і Java. Web API (XML-RPC) має обмежений функціонал, нестабільну поведінку (наприклад, HTTP-помилки 404

або 500), відсутність офіційної документації та орієнтацію на скриптові мови, що ускладнює його використання в C#.

Натомість CLI-інтерфейс VBoxManage забезпечує повний доступ до всіх адміністративних операцій VirtualBox, таких як створення, запуск, зупинка, моніторинг і видалення віртуальних машин, через прості та добре документовані команди. У .NET ці команди легко виконуються через клас System.Diagnostics.Process із підтримкою асинхронного зчитування виводу (StandardOutput, StandardError) і обробки помилок. Для спрощення взаємодії створено бібліотеку VBoxAPI, яка інкапсулює логіку виконання команд, парсинг текстового виводу та формування об’єктів для інтерфейсу. Гнучкий парсер у складі VBoxAPI дозволяє адаптуватися до змін у форматі виводу CLI між версіями VirtualBox, а асинхронна модель із використанням Task API і CancellationToken забезпечує неблокуюче виконання та можливість скасування операцій. Висока кросплатформність CLI, доступного на всіх ОС, де встановлено VirtualBox, робить його ідеальним для проєкту Pocket Moodle GUI, який потребує локального розгортання та адміністрування віртуального навчального середовища.

Для інтеграції CLI VirtualBox у середовищі .NET використано простір імен System.Diagnostics, зокрема клас Process, який дозволяє запускати VBoxManage як зовнішній процес, налаштовувати його через ProcessStartInfo та асинхронно зчитувати вивід. Це забезпечує надійний і контрольований механізм виконання команд, таких як list vms, startvm чи metrics collect. Щоб уникнути блокування основного потоку застосунку, усі операції реалізовано асинхронно з використанням Task API (async/await), а для довготривалих процесів, таких як моніторинг метрик, застосовано IAsyncEnumerable для потокової передачі даних. Механізм CancellationToken дозволяє примусово завершувати процеси, наприклад, при закритті застосунку, що підвищує стабільність.

Для ізоляції складності роботи з CLI створено бібліотеку VBoxAPI з центральним класом VBoxManager, який реалізує методи ExecuteCommandAsync, StartExecutingCommandAsync і StartMonitoringMetrics.

Ці методи забезпечують виконання одноразових і потокових команд, обробку помилок і передачу даних у ViewModel. Метод FindVBoxManagePath автоматично визначає шлях до VBoxManage залежно від ОС, забезпечуючи кросплатформність без ручного налаштування. Гнучкий парсер у складі бібліотеки адаптується до змін у форматі виводу CLI, а механізм кешування часто використовуваних метрик зменшує кількість запитів до VirtualBox, оптимізуючи продуктивність.

Інтерфейс користувача, побудований на Avalonia UI, інтегрується з бібліотекою VBoxAPI через ViewModel-шар, який використовує інтерфейс IVMManager для гнучкого зв'язку. Реактивна модель, реалізована через ReactiveUI, забезпечує миттєве оновлення інтерфейсу при зміні стану, наприклад, при виборі віртуальної машини чи оновленні метрик. Зміни конфігурації застосовуються без перезавантаження, а UX-елементи, такі як сірий текст неактивних вкладок, синя лінія для активної вкладки та навігація в одному вікні, підвищують зручність. Ця інтеграція дозволяє створити стабільну, продуктивну та зручну систему, яка відповідає вимогам кросплатформності, масштабованості та зручності для користувачів у навчальному середовищі.

2.5 Особливості програмної реалізації

Програмна реалізація системи Pocket Moodle GUI зосереджена на створенні стабільної, асинхронної та масштабованої підсистеми для взаємодії з VirtualBox через CLI-інтерфейс VBoxManage, а також на забезпеченні зручного та реактивного графічного інтерфейсу. Основна мета полягала в ізоляції складної логіки роботи з CLI, ефективному зборі метрик продуктивності та інтеграції з користувацьким інтерфейсом, побудованим на основі фреймворку Avalonia UI.

Центральним елементом системи є бібліотека VBoxAPI, яка інкапсулює логіку виконання CLI-команд VirtualBox, обробки їхнього виводу та формування об'єктної моделі для взаємодії з вищими рівнями застосунку. Основним класом бібліотеки є VBoxManager (Додаток Д), який реалізує інтерфейс IVMManager і

забезпечує асинхронне виконання команд, обробку помилок і підтримку кросплатформності. Клас спроєктовано з урахуванням повної асинхронності операцій, можливості моніторингу ресурсів у реальному часі та універсальності викликів для використання різними компонентами системи.

VBoxManager підтримує кілька ключових методів. Метод ExecuteCommandAsync виконує одноразові CLI-команди, такі як VBoxManage list vms, зчитуючи стандартний вивід (stdout) і помилки (stderr) через об’єкт ProcessStartInfo з увімкненою редирекцією потоків. Метод StartExecutingCommandAsync призначений для довготривалих процесів, таких як моніторинг, забезпечуючи асинхронне зчитування потокового виводу в реальному часі. Для моніторингу метрик реалізовано метод StartMonitoringMetrics, який викликає команду VBoxManage metrics collect --period 1 і повертає асинхронний потік об’єктів MachineMetrics через IAsyncEnumerable. Цей метод підтримує скасування операцій через CancellationToken, що дозволяє примусово завершувати процес за вимогою користувача або при закритті застосунку.

Для запуску CLI-команд використовується клас System.Diagnostics.Process із конфігурацією через ProcessStartInfo, де задаються шлях до VBoxManage (визначається автоматично методом FindVBoxManagePath), аргументи команди, редирекція stdout і stderr, а також параметри для приховування консольного вікна (CreateNoWindow = true, UseShellExecute = false). Після виконання команди перевіряється код завершення (ExitCode), а в разі помилки зчитується stderr і генерується виняток InvalidOperationException із деталями. Для тривалих процесів, таких як моніторинг, реалізовано асинхронне зчитування потоків через StandardOutput.ReadLineAsync, що унеможливлює блокування основного потоку.

Підсистема збору метрик (Додаток Г) продуктивності забезпечує безперервний моніторинг стану віртуальних машин, включаючи використання процесора (CpuUsage), пам’яті (MemoryUsage), швидкість операцій із диском (DiskRead, DiskWrite), мережевий обмін, час безперервної роботи (Uptime) і

кількість активних процесів (ActiveProcesses). Основою є команда VBoxManage metrics collect, яка повертає текстовий вивід у табличному форматі з періодичністю, заданою параметром --period.

Для обробки цього виводу створено клас MetricParser, який пострічно аналізує дані, виділяє релевантні значення та формує об’єкти типу MachineMetrics. Парсер працює інкрементально: кожен рядок доповнює поточний об’єкт MachineMetrics, доки не будуть зібрані всі необхідні поля, після чого об’єкт передається через yield return у ViewModel. Структура MachineMetrics використовує nullable-властивості, що забезпечує гнучкість при обробці неповних даних. Цей підхід дозволяє адаптуватися до змін у форматі виводу VBoxManage, мінімізуючи залежність від конкретної версії CLI.

Передача метрик до інтерфейсу користувача здійснюється через MetricsViewModel, який виступає посередником між даними MachineMetrics і графічним інтерфейсом. Властивості ViewModel, такі як CpuUsage, MemoryUsage і Uptime, реалізовано з підтримкою INotifyPropertyChanged, що забезпечує автоматичне оновлення XAML-компонентів у MetricsView.axaml при зміні даних. Завдяки використанню IAsyncEnumerable і CancellationToken підсистема метрик працює асинхронно, не блокує основний потік і підтримує стабільність при довготривалому моніторингу без необхідності перезапуску.

Інтерфейс користувача побудовано на основі фреймворку Avalonia UI з використанням XAML для декларативної розмітки. Головне вікно (MainWindow) організовано через контейнер DockPanel, де ліва панель містить список віртуальних машин (ListBox із прив’язкою до властивості VMs у MainWindowViewModel), а правий блок — вкладки (TabControl) для керування машиною, перегляду метрик, доступу до файлової системи, Moodle (Додаток В) і вбудованого терміналу. Список машин має фіксовану мінімальну ширину (200 пікселів) і притиснутий до лівого краю (Dock="Left"), а вкладки динамічно заповнюють решту простору. Кожна вкладка активується лише за наявності вибраної віртуальної машини (IsVMSelected), а її вміст формується через

прив’язку до властивостей ViewModel, таких як ManageViewContent чи MetricsViewContent.

Реактивність інтерфейсу забезпечується бібліотекою ReactiveUI та патерном MVVM. Усі ViewModel реалізують INotifyPropertyChanged, що дозволяє Binding в Avalonia автоматично відстежувати зміни стану. Наприклад, вибір віртуальної машини змінює властивість SelectedVM, що оновлює всі вкладки через RaisePropertyChanged. Кнопки керування (запуск, зупинка, перезавантаження) прив’язані до ICommand-властивостей у VMViewModel, які викликають методи IVMManager. Для тривалих операцій передбачено індикатори активності (IsBusy). Текст неактивних вкладок відображається сірим кольором, а активна вкладка виділяється синьою лінією, що полегшує навігацію. Усі функції доступні в одному вікні, а запуск машини виконується одним кліком, що відповідає принципам UX-дизайну.

Для спрощення розробки в XAML використано блок <Design.DataContext>, який дозволяє відображати демо-дані у Visual Studio, навіть якщо реальний DataContext призначається в App.axaml.cs. Це забезпечує зручність дизайну інтерфейсу та відповідність MVVM.

Типові сценарії взаємодії користувача включають вибір віртуальної машини зі списку, керування її станом, перегляд метрик, редагування конфігурації, доступ до файлової системи, Moodle або терміналу. Наприклад, сценарій запуску машини виглядає так: користувач вибирає машину в MainWindow, подія передається до MainWindowViewModel, де оновлюється SelectedVM, після чого викликається IVMManager.StartVM. VMManager делегує запит до VBoxManager, який виконує команду startvm --type headless через Process.Start. Вивід зчитується асинхронно, перевіряється код завершення, а результат оновлює статус у GUI.

Інші сценарії, такі як отримання списку машин (VBoxManage list vms), збір метрик (metrics collect), вимкнення машини (controlvm acpipowerbutton) або імпорт .ova-файлу (import), реалізовано за аналогічною асинхронною схемою.

Це забезпечує швидку реакцію інтерфейсу, стабільність при довготривалій роботі та мінімальне навантаження на ViewModel-рівень.

Система спроєктована за принципом багаторівневої архітектури з чітким поділом відповідальностей. Рівень представлення (Avalonia UI) забезпечує кросплатформений інтерфейс, ViewModel-шар координує дії користувача та стан застосунку, а сервісний рівень (VBoxAPI, IVMManager, VMManager) відповідає за взаємодію з CLI VirtualBox, парсинг даних і оновлення об’єктів. Моделі VM, MachineMetrics і MachineConfig інкапсулюють дані про машини, їхній стан і конфігурацію, підтримуючи відстеження змін через ConfigChangeTracker.

Така архітектура забезпечує ізоляцію шарів, полегшує тестування логіки без запуску інтерфейсу та дозволяє масштабувати систему, додаючи нові функції, наприклад, підтримку снапшотів чи інтеграцію з іншими платформами, без зміни базової структури. Гнучкий парсер і асинхронна модель із підтримкою CancellationToken гарантують адаптивність до змін у версіях VBoxManage та стабільність при тривалій роботі.

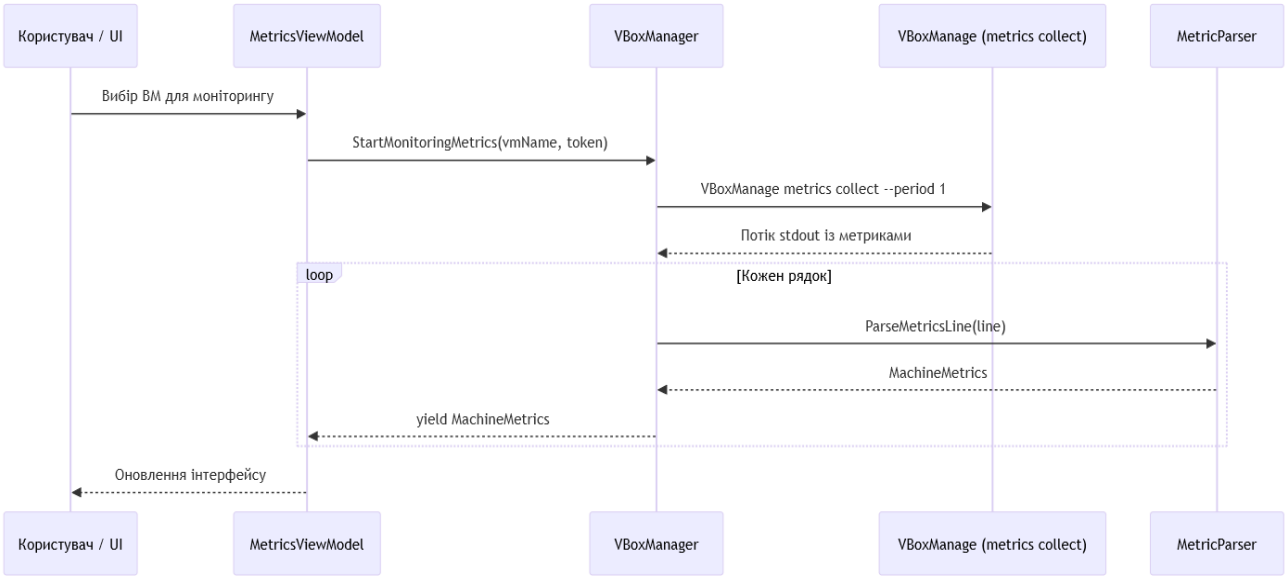

Рисунок 2.7 – Схема обміну даними в підсистемі збору метрик

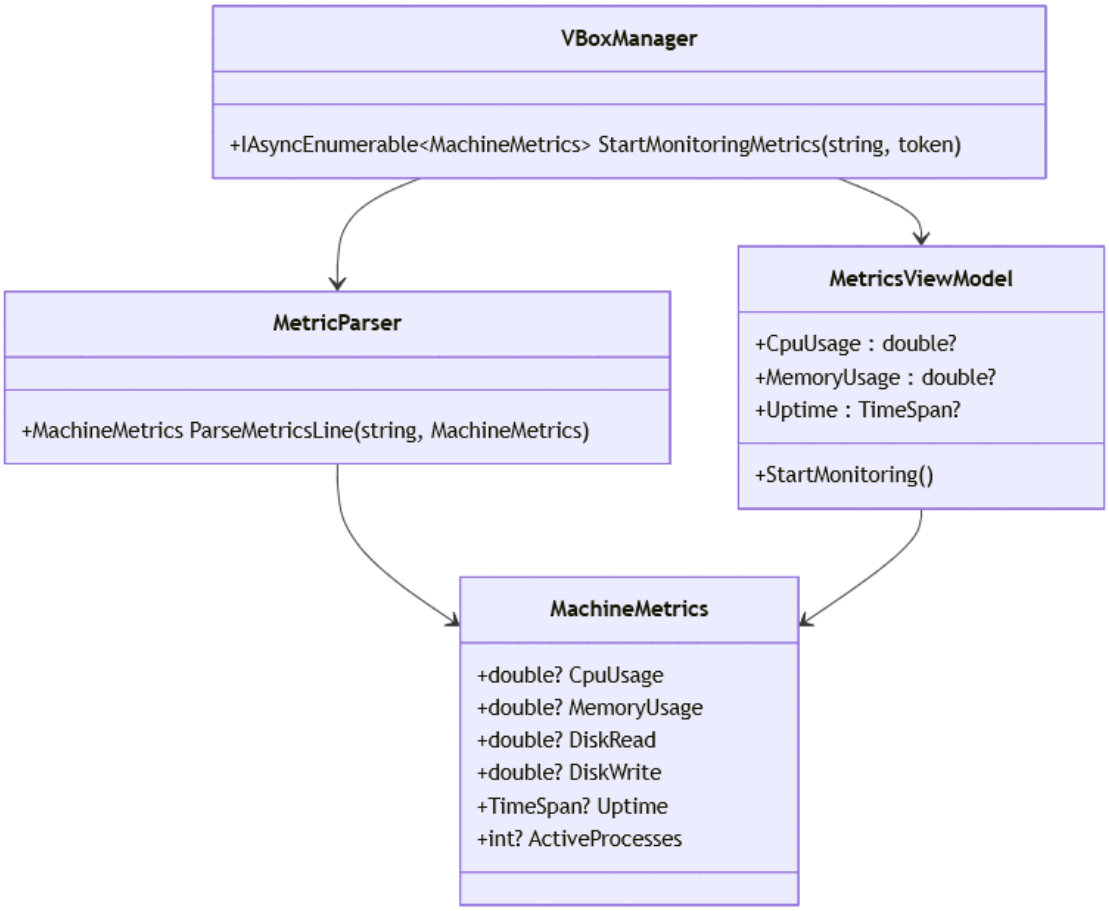

Рисунок 2.8 – UML-структура об’єктів метрик

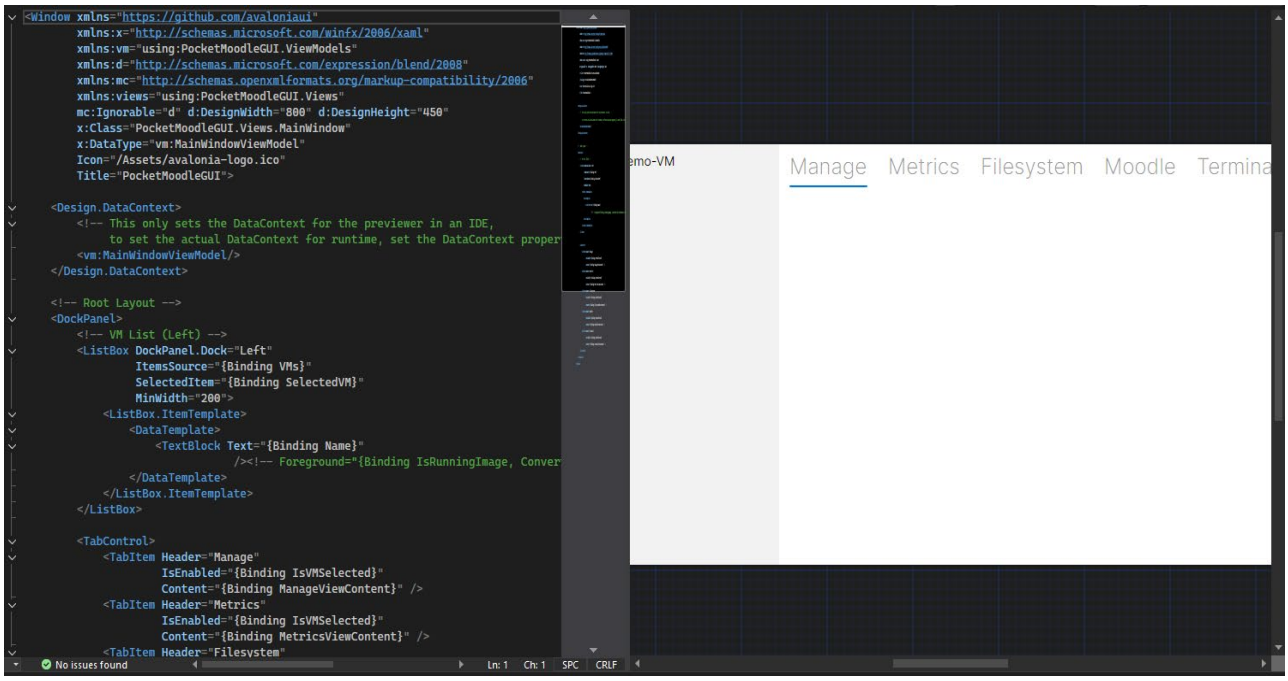

Рисунок 2.9 – Загальний вигляд інтерфейсу Pocket Moodle GUI

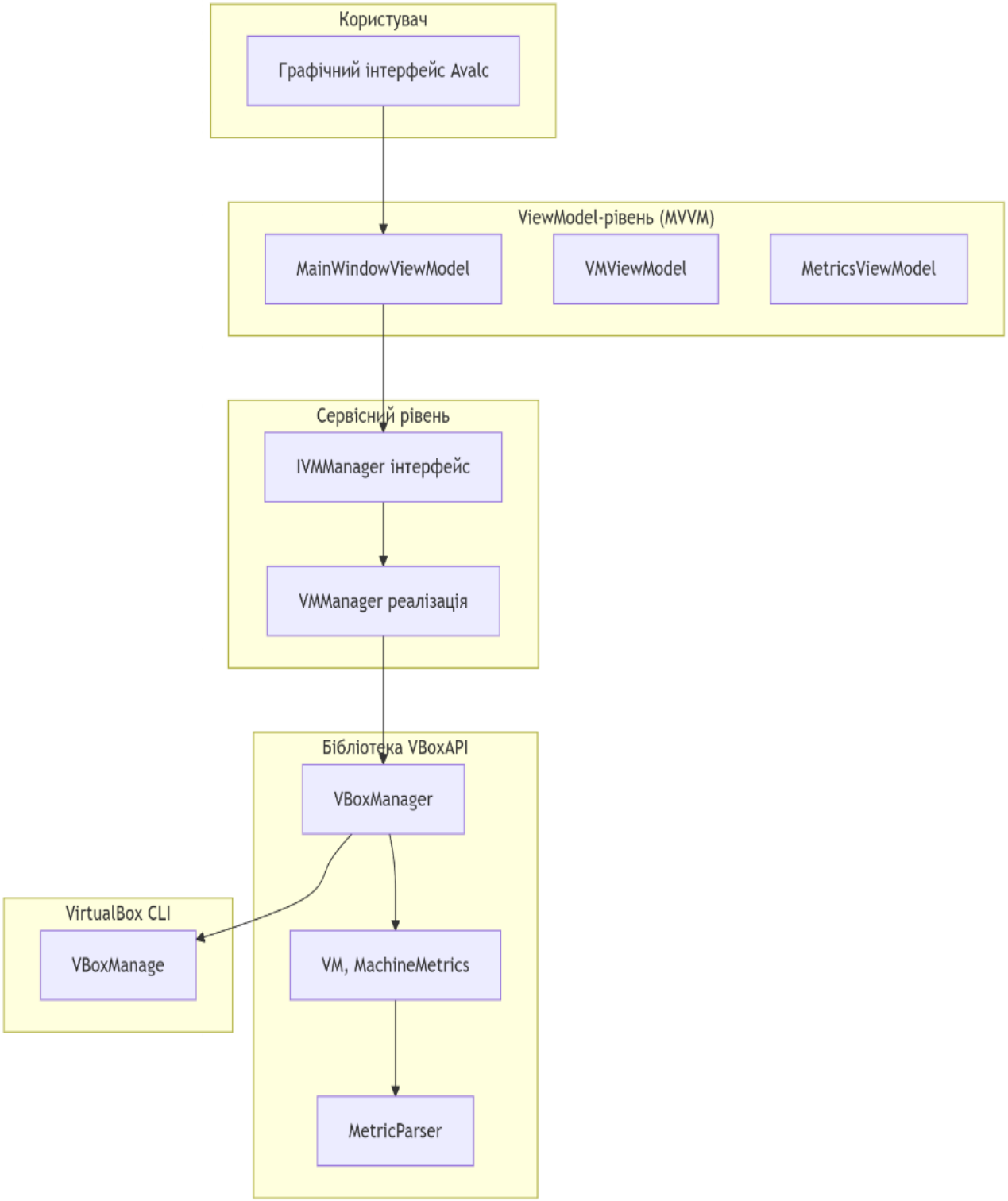

Рисунок 2.2 – Шарова архітектура Pocket Moodle GUI

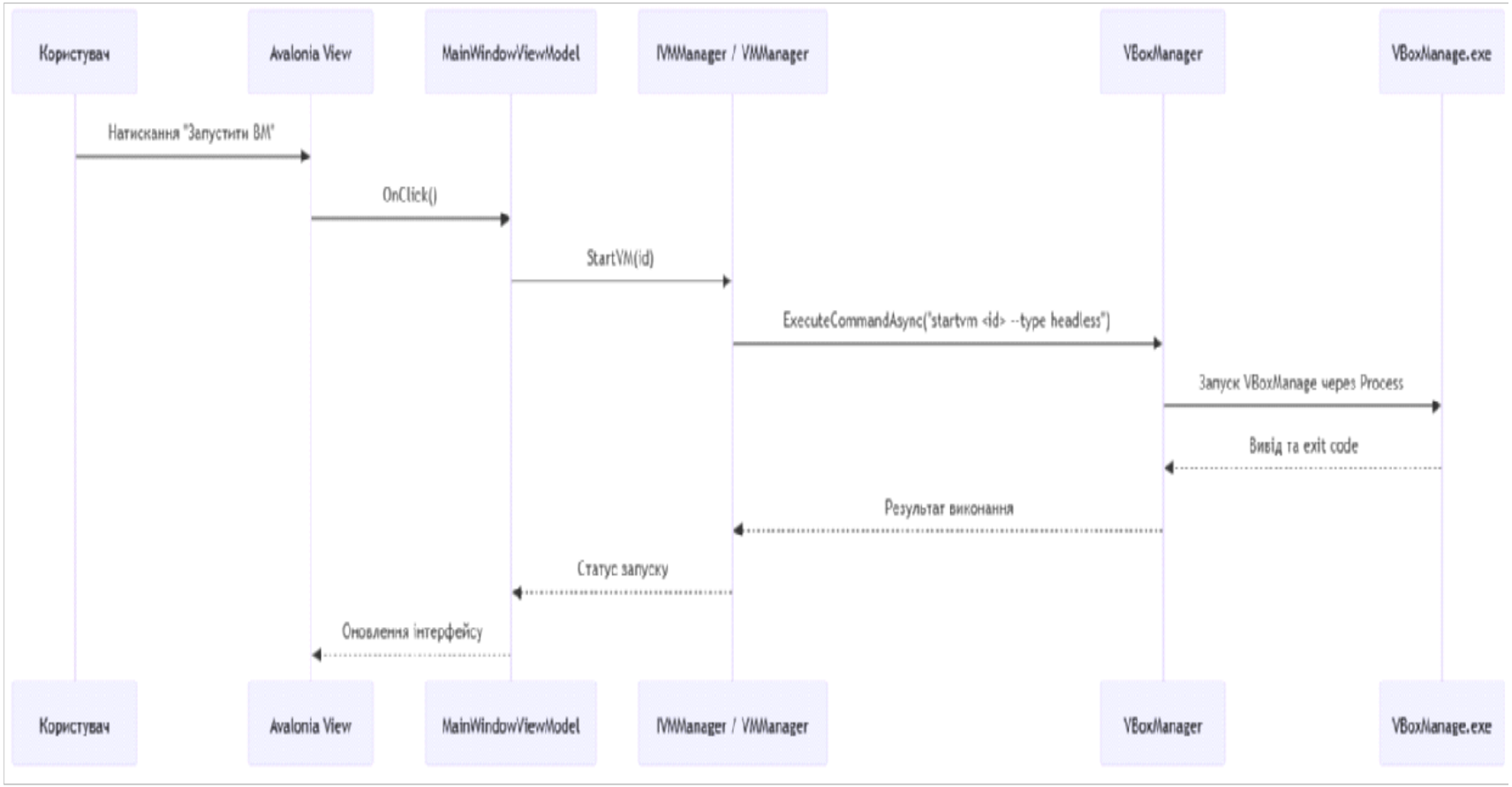

Рисунок 2.3 – Діаграма послідовності

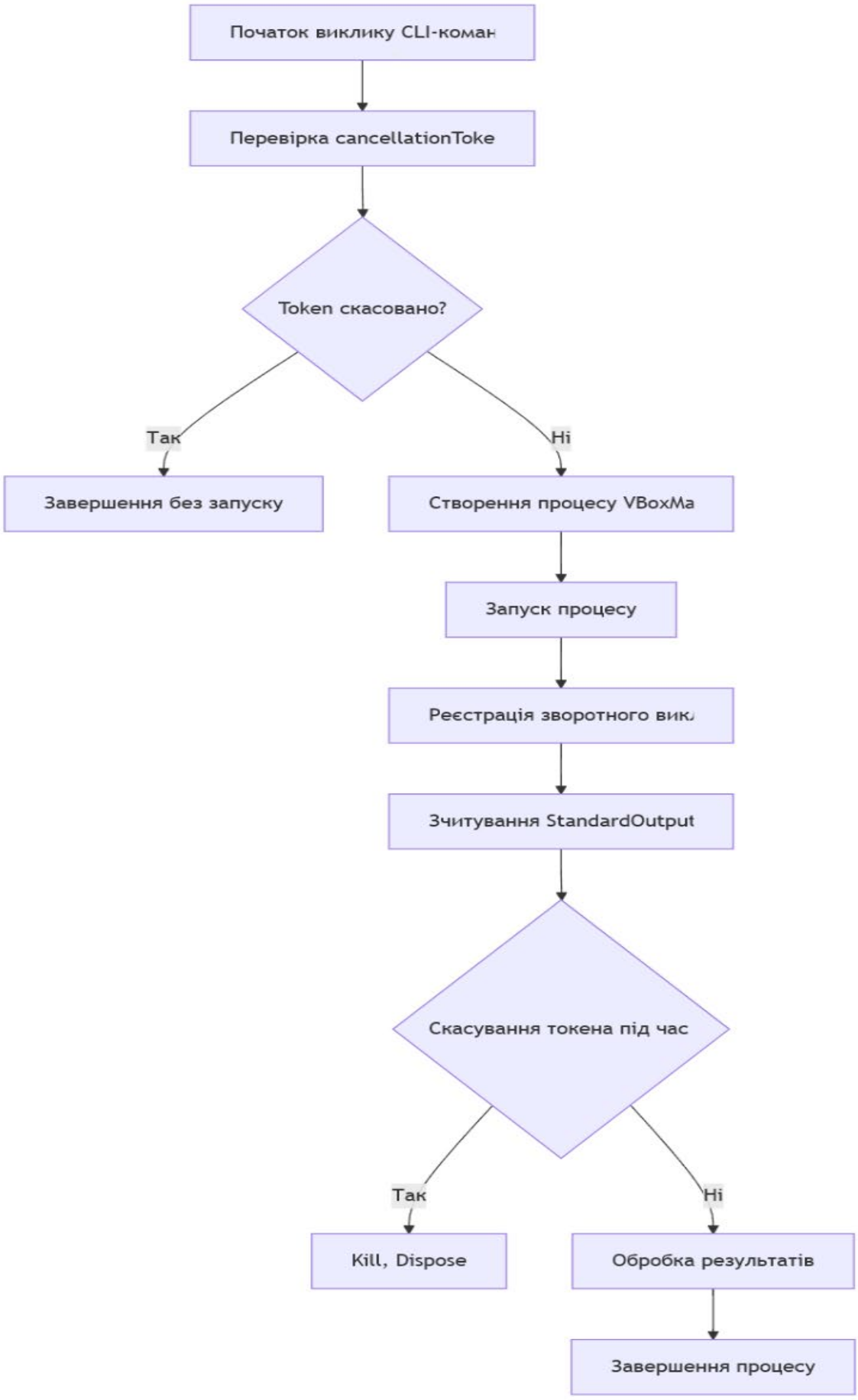

Рисунок 2.6 – Блок-схема життєвого циклу CLI-процесу у VBoxManager

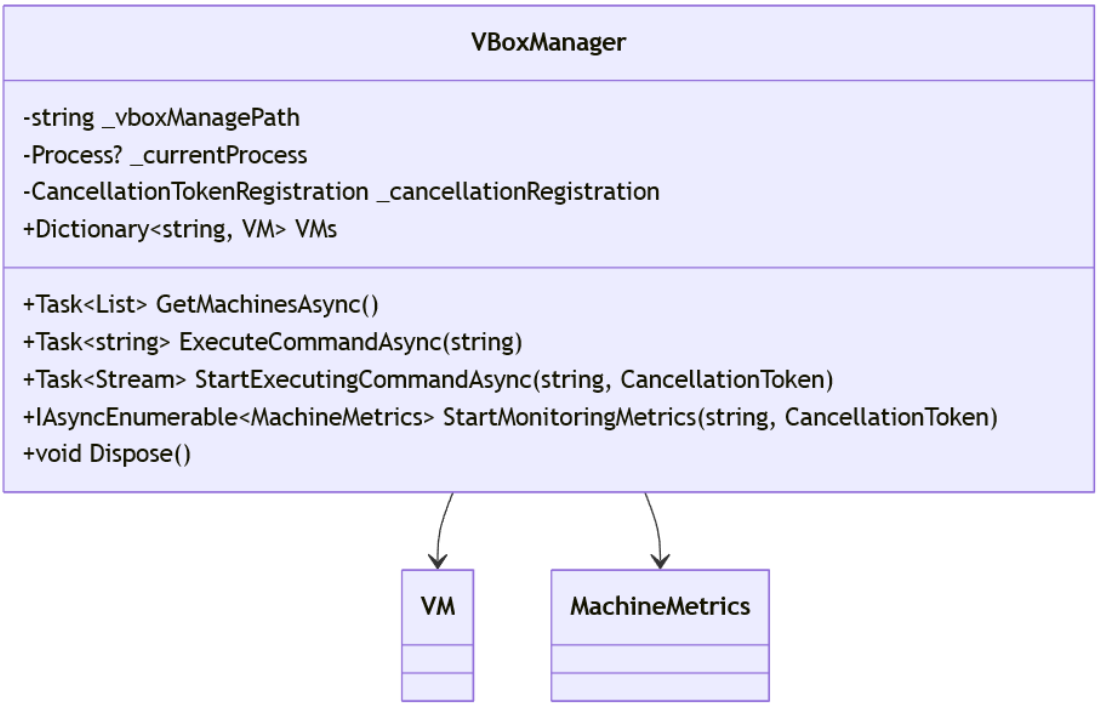

Рисунок 2.5 – Загальна структура класу VBoxManage

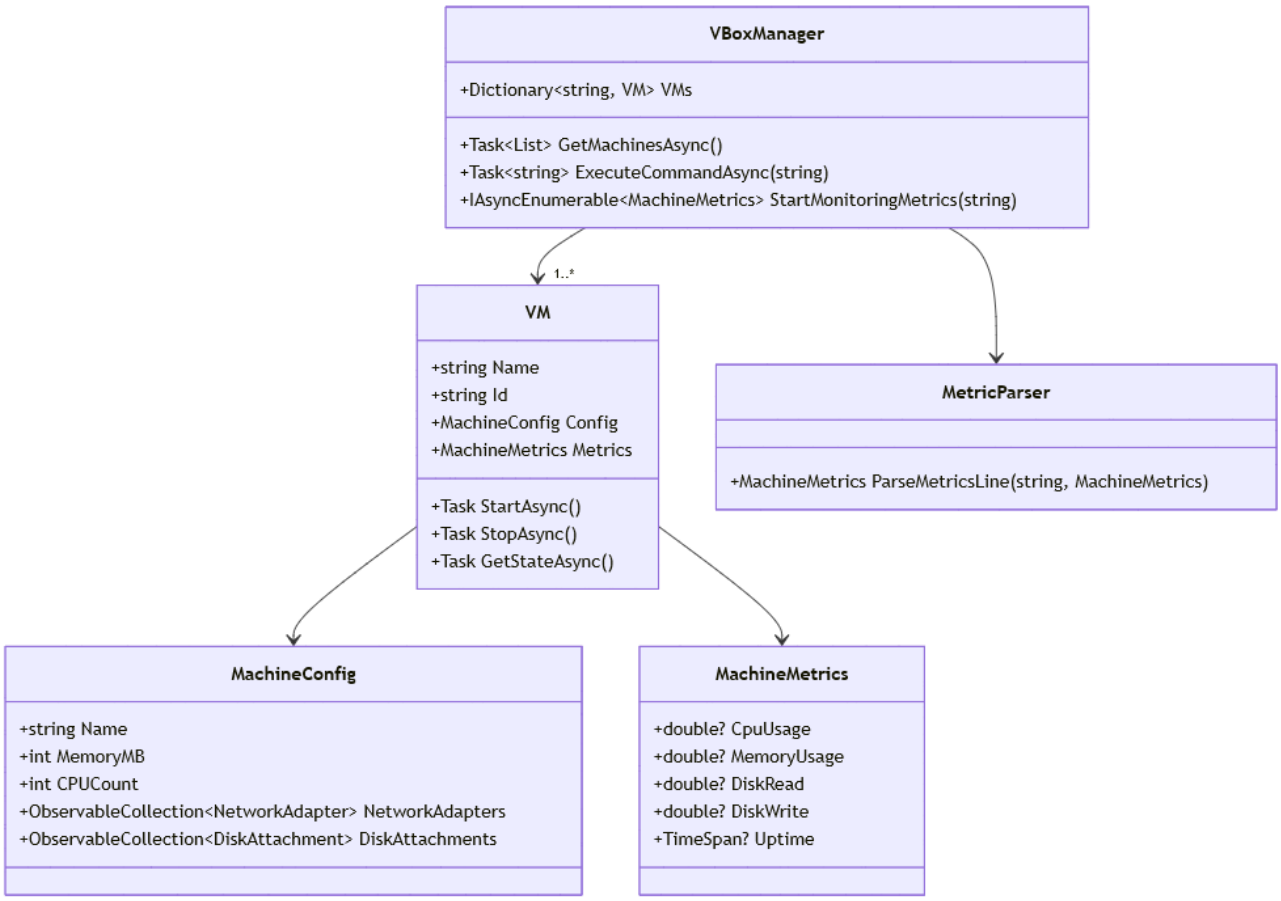

Рисунок 2.4 – Архітектура бібліотеки VBoxAPI

2.6 Організація тестування та налагодження програмного засобу

Функціональне тестування спрямоване на перевірку відповідності розробленого програмного забезпечення вимогам користувача і функціональним характеристикам. Для Pocket Moodle GUI було визначено кілька ключових сценаріїв, що охоплюють основні функції роботи з віртуальними машинами, налаштуваннями, моніторингом та інтеграцією з Moodle. Кожен з цих сценаріїв був протестований як автоматично, так і вручну для забезпечення максимальної точності та надійності результатів. Сценарії тестування:

1. Завантаження списку віртуальних машин.

   Кроки тестування:

- запуск програми;
- перевірка наявності в списку доступних віртуальних машин (ВМ);
- вибір кількох машин для перевірки коректності відображення даних.

Очікуваний результат полягає в тому, що всі доступні віртуальні машини мають бути завантажені та відображені в інтерфейсі користувача, причому кожна з них повинна містити коректні дані, зокрема ім’я та статус.

Таблиця 2.1

Тестовий сценарій 1

Функція	**Кроки для тестування**	**Очікуваний результат**
Завантаження ВМ	1. Запустити програму. 2. Перевірити список ВМ у інтерфейсі.	Усі доступні ВМ мають відображатися з коректними даними.

2. Запуск віртуальної машини.

   Кроки тестування:

- вибір ВМ зі списку доступних;
- натискання кнопки "Запустити";
- перевірка зміни статусу машини на "Running".

Очікуваний результат полягає в тому, що віртуальна машина повинна успішно запуститися, її статус має змінитися на "Running", а інтерфейс — оновити відповідне відображення.

Таблиця 2.2

Тестовий сценарій 2

Функція	**Кроки для тестування**	**Очікуваний результат**
Конфігурація ВМ	1. Вибір ВМ. 2. Зміна налаштувань (CPU, пам’ять). 3. Застосування налаштувань.	Зміни зберігаються і оновлюються в інтерфейсі.

3. Конфігурація параметрів ВМ

Кроки тестування:

- вибір ВМ зі списку;
- зміна конфігураційних параметрів (CPU, пам’ять, мережа);
- перевірка, чи зберігаються зміни після натискання кнопки "Застосувати".

Очікуваний результат полягає в тому, що всі зміни повинні коректно зберігатися і відображатися в інтерфейсі, і статус машини має бути оновлений відповідно до нових налаштувань.

Таблиця 2.3

Тестовий сценарій 3

Функція	**Кроки для тестування**	**Очікуваний результат**
Конфігурація ВМ	1. Вибір ВМ. 2. Зміна налаштувань (CPU, пам’ять). 3. Застосування налаштувань.	Зміни зберігаються і оновлюються в інтерфейсі.

4. Моніторинг метрик ВМ.

Кроки тестування:

- вибір ВМ для моніторингу;
- перехід до вкладки "Метрики";

- перевірка відображення показників CPU, пам’яті, аптайму та інших параметрів.

Очікуваний результат полягає в тому, що показники повинні оновлюватися в реальному часі, а інтерфейс має відображати актуальну інформацію без затримок.

Таблиця 2.4

Тестовий сценарій 4

Функція	**Кроки для тестування**	**Очікуваний результат**
Моніторинг метрик	1. Вибір ВМ. 2. Перехід на вкладку "Метрики". 3. Перевірка оновлення метрик.	Показники метрик оновлюються в реальному часі.

5. Підключення до Moodle

Кроки тестування:

- вибір ВМ, в якій встановлено Moodle;
- перехід до вкладки Moodle;
- перевірка доступності інтерфейсу Moodle для входу в систему.

Очікуваний результат полягає в тому, що повинен відкриватися інтерфейс Moodle, де користувач може увійти в систему за допомогою своїх облікових даних.

Таблиця 2.5

Тестовий сценарій 5

Функція	**Кроки для тестування**	**Очікуваний результат**
Підключення до Moodle	1. Вибір ВМ, що містить Moodle. 2. Перехід на вкладку Moodle.	Moodle має відкритися без помилок і бути доступним для взаємодії.

Тестування функціональності завершено успішно. Усі сценарії виконано без помилок, запуск і налаштування віртуальних машин, моніторинг метрик та

підключення до Moodle працюють стабільно відповідно до вимог, а продуктивність системи під час тестів залишалася стабільною без значних затримок чи збоїв.

Тестування на надійність та продуктивність є важливими етапами перевірки працездатності програми при високих навантаженнях та протягом тривалого часу. Воно дозволяє впевнитись, що система працюватиме стабільно та без збоїв навіть у складних умовах, таких як одночасний запуск кількох віртуальних машин або тривале використання ресурсоємних функцій. Для Pocket Moodle GUI були визначені кілька тестів для перевірки продуктивності, стабільності роботи програми та її здатності витримувати високі навантаження.

Наведемо сценарії тестування на продуктивність та надійність.

1. Навантаження на процесор (CPU).

Кроки тестування:

- запуск програми та одночасне завантаження кількох віртуальних машин із високим навантаженням на процесор;
- перевірка метрик, пов’язаних із використанням процесора через вкладку моніторингу;
- вимірювання часового інтервалу між оновленнями та їх точність при високому навантаженні.

Очікуваний результат полягає в тому, що програма має коректно працювати, не перевищуючи 80% навантаження на процесор навіть при великій кількості одночасно працюючих віртуальних машин. Всі показники повинні відображатися без затримок.

Таблиця 3.6

Тестовий сценарій 6

Функція	**Кроки для тестування**	**Очікуваний результат**
Вимірювання навантаження CPU	1. Завантажити кілька ВМ з високим навантаженням. 2. Перевірити відображення метрик CPU.	Використання процесора не повинно перевищувати 80%, оновлення відображаються в реальному часі.

2. Навантаження на пам’ять (RAM).

Кроки тестування:

- запуск кількох віртуальних машин, кожна з яких використовує велику кількість оперативної пам’яті;
- перевірка наявності проблем із відображенням пам’яті та оновленням даних на вкладці метрик;
- стеження за системними ресурсами через операційну систему, щоб визначити, чи є перевищення в пам’яті.

Очікуваний результат полягає в тому, що всі параметри пам’яті повинні відображатися правильно без значних відставань у графіках. Система не повинна перевищувати обсяг пам’яті, доступний на машині.

Таблиця 2.7

Тестовий сценарій 7

Функція	**Кроки для тестування**	**Очікуваний результат**
Вимірювання навантаження пам’яті	1. Запустити кілька ВМ з великим обсягом пам’яті. 2. Перевірити відображення пам’яті.	Всі параметри пам’яті повинні оновлюватися без помилок, і система не повинна перевищувати доступну пам’ять.

Продуктивність інтерфейсу користувача.

Кроки тестування:

- запуск кількох віртуальних машин одночасно;
- перехід між вкладками (наприклад, між вкладками "Метрики", "Файлова система", "Moodle");
- перевірка відгуку інтерфейсу під час швидких змін.

Очікуваний результат полягає в тому, що інтерфейс не повинен зависати або ставати некоректним, навіть при швидкому перемиканні між вкладками, а програма повинна залишатися відгукливою, навіть при високому навантаженні.

Таблиця 2.8

Тестовий сценарій 8

Функція	**Кроки для тестування**	**Очікуваний результат**
Продуктивність інтерфейсу	1. Запустити кілька ВМ з високим навантаженням. 2. Швидко перемикатись між вкладками.	Інтерфейс повинен залишатись відгукливим, навіть при швидкому перемиканні між вкладками.

Надійність при тривалому використанні.

Кроки тестування:

- програма працює протягом кількох годин без перерви.
- вивіряється стан ресурсів та стабільність роботи інтерфейсу під час довготривалого використання.
- перевірка на наявність витоків пам’яті або уповільнення в роботі програми.

Очікуваний результат полягає в тому, що система повинна працювати стабільно без витоків пам’яті чи інших проблем, таких як значне сповільнення роботи програми.

Таблиця 2.9

Тестовий сценарій 9

Функція	**Кроки для тестування**	**Очікуваний результат**
Перевірка на надійність	1. Програма працює більше 4 годин. 2. Перевірити використання ресурсів.	Система повинна працювати стабільно без витоків пам’яті або сповільнень.

Проведене тестування на надійність і продуктивність системи показало стабільну роботу в різних умовах. Під час тестування на навантаження CPU система успішно справлялася з навантаженням до 80% при одночасному запуску до п’яти віртуальних машин, забезпечуючи оновлення метрик у реальному часі без затримок. Тестування на навантаження пам’яті не виявило значних проблем,

а програма залишалася стабільною навіть при високому використанні пам’яті віртуальними машинами. Інтерфейс користувача демонстрував високу продуктивність, дозволяючи швидко перемикатися між вкладками без затримок чи зависань навіть за умов значного навантаження. При довготривалому використанні програма працювала стабільно протягом понад восьми годин без уповільнень чи витоків пам’яті. Усі тести завершено успішно, система показала високу продуктивність і надійність, не мала значних збоїв чи помилок, а метрики оновлювалися в реальному часі без затримок чи неточностей у відображенні.

2.7 Рекомендації по використанню та впровадженню програмного засобу

Реалізація сценарію старту, налаштування та моніторингу віртуальної машини у системі Pocket Moodle є одним з центральних етапів функціонування програмного комплексу. Цей сценарій охоплює повний цикл дій, які користувач виконує для підготовки середовища, запуску віртуальної машини, її налаштування та подальшого спостереження за ключовими параметрами продуктивності. Архітектура реалізації побудована за принципом поетапної взаємодії View → ViewModel → CLI-інтерфейс, що забезпечує стабільність, реактивність та гнучкість користувацького досвіду.

Першим етапом взаємодії є ініціалізація графічного інтерфейсу та завантаження переліку доступних віртуальних машин. Цей список формується на основі запиту до VBoxAPI через інтерфейс IVMManager, реалізований у вигляді VBoxManager. Результати виконання команди VBoxManage автоматично передаються у ViewModel, де обгортаються у структури VMViewModel. Ці структури містять усю необхідну інформацію для відображення у візуальному списку — назву машини, її унікальний ідентифікатор, а також поточний статус.

Після вибору користувачем однієї з машин активуються функціональні вкладки інтерфейсу, зокрема вкладка конфігурації. У цій частині системи реалізовано можливість перегляду та редагування основних параметрів машини:

обсягу оперативної пам’яті, кількості процесорних ядер, типу мережевого адаптера та обсягу виділеного сховища. Усі ці значення представлені у вигляді інтерактивних полів або повзунків, зміна яких негайно фіксується у ViewModel.

За потреби користувач може застосувати нові конфігураційні параметри перед запуском машини, що забезпечує контроль над ресурсами ще до моменту активації віртуального середовища.

Наступним кроком є безпосередній запуск віртуальної машини. Цей процес ініціюється користувачем натисканням відповідної кнопки у вкладці керування. Після цього ViewModel [37] формує команду запуску, яка передається в CLI-обгортку VBoxManager, де формально генерується команда запуску через VBoxManage. Завдяки асинхронному виконанню ця операція не блокує інтерфейс і дозволяє користувачу продовжувати взаємодію з системою. Після успішного запуску статус машини оновлюється у ViewModel, що автоматично відображається у вікні інтерфейсу.

Особливе місце у цьому сценарії посідає механізм моніторингу. Для його реалізації було впроваджено функціонал підключення до потоку системних метрик віртуальної машини, які надає VBoxManage через спеціальну команду. Результати її виконання зчитуються з потоку у режимі реального часу та парсяться у ViewModel, яка містить властивості для відображення навантаження на процесор, використання оперативної пам’яті, тривалості безперервної роботи (аптайму) та інших характеристик. Ці властивості інтегруються з інтерфейсом через механізм двостороннього зв’язування, що дозволяє візуально відстежувати зміну параметрів у режимі live-спостереження.

Реалізація цього сценарію об’єднує одразу кілька ключових підсистем: від взаємодії з CLI VirtualBox через програмну обгортку до реактивного інтерфейсу з високим рівнем зворотного зв’язку. Усе це в комплексі забезпечує стабільну роботу застосунку, дозволяючи користувачеві зручно керувати віртуальними машинами в рамках одного цілісного GUI середовища. Сценарій старту, налаштування та моніторингу реалізовано з урахуванням як технічної ефективності, так і загальної зручності використання, що відповідає сучасним

вимогам до програмного забезпечення у сфері віртуалізації та локальних освітніх платформ [38].

Однією з ключових функцій, реалізованих у межах Pocket Moodle GUI, є забезпечення зручної взаємодії з локальним екземпляром системи управління навчанням Moodle, що розгортається всередині віртуального середовища. Основна ідея полягала в тому, щоб надати користувачу простий та інтуїтивний інтерфейс для доступу до Moodle без потреби ручного пошуку IP-адреси, конфігурації портів чи окремих клієнтів. Для цього в інтерфейсі Pocket Moodle було реалізовано окрему функціональну вкладку, яка відповідає за з'єднання з середовищем Moodle та візуалізацію його інтерфейсу у вікні програми.

Вкладка Moodle активується після запуску віртуальної машини, яка містить попередньо встановлене середовище Moodle. У ViewModel ця вкладка пов'язана з об'єктом типу VMViewModel, що містить інформацію про параметри мережі та дозволяє побудувати локальне посилання для доступу до веб-інтерфейсу Moodle. Таким чином, незалежно від внутрішніх налаштувань VirtualBox, користувач завжди має прямий доступ до LMS через вікно Pocket Moodle GUI.

Для візуалізації інтерфейсу Moodle застосовується компонент вбудованого перегляду вебсторінок, що забезпечує рендеринг HTML-контенту без виходу з програми. Це дозволяє відкривати Moodle у вкладці на кшталт стандартного браузера, підтримуючи функції аутентифікації, навігації між курсами, перегляду контенту тощо. У випадку, якщо віртуальна машина ще не повністю завантажена, вкладка відображає повідомлення про очікування запуску сервісів або пропонує користувачеві оновити вміст.

Окрім інтеграції з веб-інтерфейсом, реалізовано також взаємодію з файловою системою віртуального середовища. Відповідна вкладка дозволяє отримати доступ до спільних папок, налаштованих у конфігурації VirtualBox, де користувач може розміщувати SCORM-пакети, матеріали курсів, резервні копії або експортовані дані. Це стало можливим завдяки підтримці механізму guest

additions та автоматичного монтування спільних директорій, налаштованих через VBoxManage [37].

Ще одним елементом інтерактивної взаємодії стала вкладка з вбудованим терміналом. Вона дає змогу адміністратору отримати доступ до командного рядка всередині віртуальної машини без потреби зовнішніх SSH-клієнтів. Такий підхід значно спрощує технічну підтримку Moodle: можна швидко виконати оновлення системи, перезапустити сервіси, перевірити статус бази даних або журнал помилок.

Усі ці елементи в сукупності формують завершений сценарій роботи користувача з освітнім середовищем на базі Moodle. Починаючи з запуску віртуальної машини, користувач отримує повноцінний доступ до LMS-системи, її адміністрування та вмісту курсів — без залишення застосунку, без налаштування мережевих тунелів і без потреби у технічній експертизі в питаннях віртуалізації.

Підсистема взаємодії з Moodle була реалізована як одна з центральних функціональних частин Pocket Moodle GUI. Її інтеграція дозволила досягти не лише технічної завершеності рішення, а й зробити продукт доступним для користувачів, які не мають глибоких знань у сфері адміністрування серверів чи систем віртуалізації. Це цілком відповідає вимогам до сучасного освітнього програмного забезпечення, орієнтованого на доступність, автономність та простоту розгортання.

Аналіз продуктивності є критично важливим етапом у розробці програмного забезпечення, оскільки він дозволяє виявити потенційні вузькі місця та оптимізувати систему для кращої роботи при високих навантаженнях. У цьому розділі здійснюється аналіз продуктивності системи Pocket Moodle GUI на основі показників використання пам’яті, процесора та часу відгуку інтерфейсу користувача.

У ході тестування були зафіксовані основні показники продуктивності програми: використання процесора (CPU) та пам’яті (RAM). Використання пам’яті програмою становить 78 МБ, що є нормальним значенням для

застосунку, який працює з віртуальними машинами та має візуальний інтерфейс. Важливо було перевірити, чи спостерігаються значні коливання у використанні пам’яті або її постійне збільшення, що могло б указувати на витоки пам’яті. Аналіз показав, що програма стабільно використовує 78 МБ пам’яті протягом тестування без значних змін, що свідчить про відсутність великих витоків пам’яті чи небажаного зростання її використання. Це підтверджує високу ефективність програми. Щодо використання процесора, результати демонструють практично нульове навантаження навіть під час виконання таких операцій, як запуск віртуальних машин або моніторинг метрик, що є позитивним показником мінімального споживання ресурсів.

Перевірка продуктивності інтерфейсу користувача включала швидкість оновлення метрик, зокрема CPU, пам’яті та інших показників в реальному часі. Програма продовжує оновлювати ці дані без затримок або значних лагів, навіть при швидкому перемиканні між вкладками та при великому навантаженні на ресурси.

Інтерфейс працює стабільно, з невеликими перервами при переході між вкладками (як видно на скріншоті). Користувач може перемикатися між вкладками, такими як "Метрики", "Файлова система", "Moodle", без будь-яких затримок, що свідчить про гарну продуктивність графічної частини програми.

Аналіз продуктивності системи Pocket Moodle GUI показує наступні результати (табл. 2.10): програма продемонструвала високу продуктивність і стабільність навіть при роботі з великим навантаженням. Ресурси використовуються економно, і система працює без витоків пам’яті та неоптимізованих процесів. Тестування інтерфейсу показало, що він не страждає від значних затримок або проблем з відгуком.

Ці результати свідчать про те, що система Pocket Moodle GUI готова до реального використання при високих навантаженнях, і не потребує додаткової оптимізації на поточному етапі.

Таблиця 2.10

Аналіз продуктивності системи

Показник	**Результат**	**Інтерпретація**
Використання процесора (CPU)	0% протягом тесту, без значних змін	Програма використовує мінімальні ресурси процесора, що є позитивним результатом.
Використання пам’яті (RAM)	78 MB	Система стабільно використовує пам’ять, без витоків або значних коливань.
Швидкість оновлення метрик	Оновлення в реальному часі без затримок	Всі метрики оновлюються коректно, навіть при високому навантаженні.
Інтерфейс користувача	Плавний, відгукливий інтерфейс при перемиканні вкладок	Програма не демонструє проблем з відгуком або відставаннями в інтерфейсі.

ВИСНОВКИ

Розробка графічного інтерфейсу користувача (GUI) для системи Pocket Moodle GUI була успішно завершена. Розроблений інтерфейс дозволяє користувачам зручно керувати віртуальними машинами через API VirtualBox та взаємодіяти з платформою Moodle, що підтверджує ефективність розробленого програмного рішення.

Інтеграція VirtualBox API з платформою Moodle та створення єдиного інтерфейсу для управління віртуальними машинами є важливим досягненням. Віртуальні машини можуть бути зручно запущені, зупинені, налаштовані та моніторингуються без необхідності використання додаткових інструментів або зовнішніх клієнтів.

Тестування функціональності та продуктивності показало високу ефективність системи, зокрема з точки зору стабільності роботи, швидкості відгуку інтерфейсу та використання системних ресурсів. Програма стабільно працює при високих навантаженнях і не демонструє значних витоків пам’яті або збоїв у роботі.

Негативні результати та проблеми: незважаючи на загальну стабільність системи, були виявлені деякі дрібні затримки при великому навантаженні, зокрема при одночасному запуску кількох віртуальних машин з великими вимогами до пам’яті. Також були зафіксовані незначні проблеми з інтеграцією нових оновлень VirtualBox у систему.

Для підвищення продуктивності системи пропонується оптимізувати обробку запитів до API VirtualBox і впровадити механізм кешування для часто використовуваних метрик. Щоб усунути можливі затримки при запуску кількох віртуальних машин, доцільно реалізувати асинхронний запуск і моніторинг, що зменшить навантаження на користувацький інтерфейс. Також рекомендується регулярно оновлювати систему для забезпечення сумісності з новими версіями VirtualBox і покращення взаємодії з Moodle.

У майбутніх дослідженнях варто зосередитися на інтеграції системи з іншими платформами управління віртуальними машинами, такими як VMware

або Hyper-V, що розширить функціональність Pocket Moodle GUI. Крім того, дослідження можливостей автоматизації навчальних процесів через інтеграцію з іншими системами навчання та ресурсами сприятиме створенню більш повноцінного середовища для студентів і викладачів.

Розроблене програмне забезпечення демонструє високу ефективність у виконанні основних функцій і може бути впроваджене в освітні установи для полегшення адміністрування та організації навчального процесу. У майбутньому важливо продовжувати вдосконалення системи, враховуючи нові вимоги та технологічні оновлення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Булатецький В. В., Булатецька Л. В., Павленко Ю.С. Портативна LMS Moodle з віддаленим доступом. *Прикладні проблеми комп'ютерних наук, безпеки та математики.* 2023. № 2. С. 70–78. URL: https://apcssm.vnu.edu.ua/index.php/Journalone/article/view/19
2. Colvin H. VirtualBox: An Ultimate Guide Book on Virtualization with VirtualBox. North Charleston, SC : CreateSpace Independent Publishing Platform, 2015. 70 p.
3. Reddy S., Roth R. Moodle 3.x Developer's Guide. Birmingham : Packt Publishing, 2017. 440 p.
4. Smith Nash S. Moodle Course Design Best Practices. Design and develop outstanding Moodle learning experiences. 2nd ed. Birmingham : Packt Publishing, 2018. 248 p.
5. Dougiamas M., Taylor P. Moodle: E-Learning Course Development. Birmingham : Packt Publishing, 2005. 304 p.
6. Oleksandr D. A., Lytvyn O. L., Shyshkina M. P., Ponomarchuk P. P., Kukharenko K. K. Digital Virtualization Technologies in Distance Learning. International Journal of Advanced Trends in Computer Science and Engineering. 2020. Vol. 9 (2). P. 1808–1813. DOI: 10.30534/ijatcse/2020/138922020.
7. Davis B., Wilson C. Impact of virtual labs on student learning outcomes in STEM education. Innovative Development in Educational Activities. 2024. Vol. 3 (9). P. 45–58.
8. Bond M., Bedenlier S., Marín V. I., Händel M. Quality Standards in Online Education: The ISO/IEC 40180 Framework. International Review of Research in Open and Distributed Learning. 2021. Vol. 22 (1). P. 134–155.
9. Al-Said K. M. Students' Perceptions of Moodle: A Case Study. International Journal of Emerging Technologies in Learning (iJET). 2015. Vol. 10 (4). P. 25–30. DOI: 10.3991/ijet.v10i4.4605.
10. Gkountouma A., Palaigeorgiou G., Tsihouridis C. Designing Educational Software with a Focus on User Experience: A Case Study. Education and

Information Technologies. 2021. Vol. 26. P. 345–367. DOI: 10.1007/s10639-020-10291-3.

11. Citrix. Open Virtualization Format (OVF and OVA). XenServer Product Documentation. URL: https://docs.xenserver.com/en-us/xencenter/current-release/vms-exportimport-ovf.html (дата звернення: 15.05.2025).
12. Avalonia UI Team. Avalonia UI: Cross-Platform UI Frameworks for .NET Developers. Avalonia UI. URL: https://avaloniaui.net/ (дата звернення: 15.05.2025).
13. Moodle HQ. API Guides - Moodle Developer Resources. Moodle Developer Resources. URL: https://moodledev.io/docs/5.0/apis (дата звернення: 15.05.2025).
14. Eastman D. Avalonia: An Open Source Option for Cross-Platform UI Work. The New Stack. Опубліковано: 21.01.2025. URL: https://thenewstack.io/avalonia-an-open-source-option-for-cross-platform-ui-work/ (дата звернення: 15.05.2025).
15. Moodle. Developer documentation. MoodleDocs. URL: https://docs.moodle.org/en/Developer_documentation (дата звернення: 15.05.2025).
16. Microsoft. .NET documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/ (дата звернення: 15.05.2025).
17. Oracle. Oracle® VM VirtualBox® User Manual for Release 7.0. Oracle VirtualBox Documentation. URL: https://docs.oracle.com/en/virtualization/virtualbox/7.0/user/ (дата звернення: 15.05.2025).
18. W3C Web Accessibility Initiative (WAI). Web Content Accessibility Guidelines (WCAG) Overview. W3C Web Accessibility Initiative. URL: https://www.w3.org/WAI/standards-guidelines/wcag/ (дата звернення: 15.05.2025).

19. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley Professional, 1994. 395 p. ISBN: 9780201633610.
20. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. London : Prentice Hall, 2017. 432 p. ISBN: 9780134494166.
21. Price M. J. C# 10 and .NET 6 – Modern Cross-Platform Development: Build apps, websites, and services with ASP.NET Core 6, Blazor, and EF Core 6 using Visual Studio 2022 and Visual Studio Code. 6th ed. Birmingham : Packt Publishing, 2021. 826 p. ISBN: 9781801077361.
22. Troelsen A., Japikse P. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming. 11th ed. New York : Apress, 2022. 1705 p. ISBN: 9781484278680.
23. Nielsen J. Usability Engineering. Boston : Morgan Kaufmann, 1993. 384 p. ISBN: 9780125184069.
24. Norman D. The Design of Everyday Things: Revised and Expanded Edition. New York : Basic Books, 2013. 368 p. ISBN: 9780465050659.
25. Skeet J. C# in Depth. 4th ed. New York : Manning Publications, 2019. 528 p. ISBN: 9781617294532.
26. Rosen P., Shneiderman B. Representing and Managing Virtual Machines: A User Interface Design Study. Journal of Systems and Software. 2006. Vol. 79 (8). P. 1137–1149.
27. Abualese H., Mdanat M., Saleh M. Towards Evaluating User Experience of Cross-Platform Mobile UI Toolkits. International Journal of Advanced Computer Science and Applications (IJACSA). 2019. Vol. 10 (5). P. 557–564. DOI: 10.14569/IJACSA.2019.0100569.
28. Ferreira D., Silva W., Vale T., Fernandes A. M. A Systematic Literature Review on Software Metrics and Human-Computer Interaction. Journal of Systems and Software. 2019. Vol. 157. Article 110399. DOI: 10.1016/j.jss.2019.110399.
29. Scott S. D., Myers B. A. The Most Important Design Guideline? IEEE Software. 2004. Vol. 21 (3). P. 96. DOI: 10.1109/MS.2004.1293082.

30. Meyer B. The Many Faces of Virtualization. Communications of the ACM. 2009. Vol. 52 (3). P. 27–29. DOI: 10.1145/1467247.1467260.
31. Oracle Corporation. Oracle® VM VirtualBox® User Manual for Release 7.0. Oracle VM VirtualBox. URL: https://www.virtualbox.org/manual/UserManual.html (дата звернення: 15.05.2025).
32. Oracle Corporation. Chapter 11. Oracle VM VirtualBox Programming Interfaces. Oracle VM VirtualBox User Manual. URL: https://www.virtualbox.org/manual/ch11.html (дата звернення: 15.05.2025).
33. Avalonia UI Team. Welcome to Avalonia Docs. Avalonia UI Documentation. URL: https://docs.avaloniaui.net/docs/welcome (дата звернення: 15.05.2025).
34. Microsoft. C# Guide. .NET Documentation. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 15.05.2025).
35. Microsoft. Asynchronous programming with async and await (C#). .NET Documentation. URL: https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/task-asynchronous-programming-model (дата звернення: 15.05.2025).
36. Microsoft. Process Class (System.Diagnostics). .NET API Browser. URL: https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.process (дата звернення: 15.05.2025).
37. National Institute of Standards and Technology. Security and Privacy Controls for Information Systems and Organizations. NIST Special Publication 800-53 Revision 5. U.S. Department of Commerce, 2020. URL: https://csrc.nist.gov/pubs/sp/800/53/r5/upd1/final (дата звернення: 15.05.2025).
38. Moodle HQ. External Services API. Moodle Developer Resources. URL: https://moodledev.io/docs/general/external (дата звернення: 15.05.2025).

ДОДАТКИ

ДОДАТОК А

Технічне завдання на розробку програмного продукту Pocket Moodle GUI

Загальні положення

Назва: Pocket Moodle GUI

Призначення: Кросплатформений програмний засіб із графічним інтерфейсом для управління віртуальними машинами (ВМ) VirtualBox і інтеграції з навчальною платформою Moodle.

Мета: Створити зручний графічний інтерфейс для автоматизації управління ВМ, моніторингу їхньої продуктивності та доступу до Moodle у навчальних середовищах.

Вимоги до функціональних характеристик

Програмний продукт має забезпечувати управління віртуальними машинами через команди VirtualBox (отримання списку ВМ, запуск, зупинка, перезавантаження, імпорт .ova-файлів, редагування конфігурації, наприклад, CPU, RAM чи мережевих налаштувань). Для моніторингу продуктивності система повинна в реальному часі збирати метрики (використання CPU, RAM, швидкість операцій із диском, мережевий обмін, аптайм) за допомогою команди VBoxManage metrics collect і відображати їх у таблиці чи блоці з оновленням кожну секунду. Інтеграція з Moodle передбачає доступ до навчальних ресурсів через спільні папки або сервіси у ВМ із виділеною вкладкою в інтерфейсі.

Графічний інтерфейс, побудований на Avalonia UI, має містити багатовкладкову структуру (керування, метрики, конфігурація, файли, Moodle, термінал) із лівою панеллю для списку ВМ і реактивним оновленням вкладок через ReactiveUI. Дизайн інтерфейсу включає сірий текст для неактивних вкладок, синю лінію для активної та індикатори активності. Вкладка терміналу дозволяє вводити команди в гостьовій або хостовій системі з відображенням результатів у реальному часі. Автоматизація забезпечується асинхронним

виконанням команд із підтримкою CancellationToken і кешуванням метрик для оптимізації продуктивності.

Вимоги до надійності

Система має гарантувати безперебійну роботу під час 24-годинного моніторингу, обробляти помилки CLI з логуванням і винятками, а також відновлюватися після збоїв протягом 5 хвилин.

Умови експлуатації

Продукт розроблено для роботи на Windows 10/11 і Ubuntu 20.04 або новіших версіях із мінімальними вимогами до обладнання: 4-ядерний процесор, 8 ГБ RAM, 20 ГБ SSD. Система сумісна з VirtualBox 6.1+ і Moodle 4.0+.

Вимоги до технічних засобів

Розробка ведеться на C# (.NET 8.0 або новіша) із використанням фреймворку Avalonia UI (XAML, ReactiveUI). Для взаємодії з VirtualBox застосовується CLI (VBoxManage) через бібліотеку VBoxAPI, а також бібліотеки System.Diagnostics.Process і IAsyncEnumerable для асинхронної обробки.

Вимоги до сумісності

Система інтегрується з Moodle через спільні папки або API, автоматично визначаючи шлях до VBoxManage для забезпечення кросплатформності.

Вимоги до документації

Для розробників передбачено опис архітектури (UML), документацію VBoxAPI та інструкцію з розгортання. Користувачам надається посібник із керування ВМ, доступу до Moodle і встановлення системи.

Техніко-економічні показники

Продукт підвищує ефективність навчання завдяки автоматизації управління ВМ і знижує витрати на адміністрування через інтеграцію з Moodle та зручний інтерфейс.

Етапи розробки

Розробка включає аналіз вимог і ризиків (0.5 місяця), створення прототипу з базовою бібліотекою VBoxAPI та GUI (1 місяць), реалізацію підсистеми метрик

(1 місяць), завершення інтерфейсу з інтеграцією Moodle та оптимізацією UX (1 місяць), а також тестування й впровадження (0.5 місяця).

Порядок контролю і приймання

Контроль передбачає тестування функціоналу (запуск ВМ, моніторинг, інтеграція), перевірку продуктивності (затримка оновлення метрик до 1 секунди, стабільність протягом 24 годин) і затвердження замовником після демонстрації та відгуків тестової групи.

ДОДАТОК Б

Інструкції користувача

1. Загальні відомості

Позначення програми: Графічний інтерфейс до платформи Pocket Moodle

Найменування програми: Pocket Moodle GUI

2. Функціональне призначення

Pocket Moodle GUI призначена для викладачів, студентів і адміністраторів навчальних закладів, які використовують віртуалізацію для організації навчального процесу. Програма забезпечує зручне управління віртуальними машинами (ВМ) на базі VirtualBox через графічний інтерфейс і інтеграцію з навчальною платформою Moodle. Вона дозволяє розв’язувати такі задачі:

- управління ВМ;
- моніторинг продуктивності;
- інтеграція з Moodle;
- встановлення Moodle.

3. Умови застосування програми

Системні вимоги.

Комп’ютер: персональний комп’ютер із 4-ядерним процесором, 8 ГБ оперативної пам’яті, 20 ГБ вільного місця на SSD.

Операційна система: Windows 10/11 або Ubuntu 20.04 (чи новіші).

Програмне забезпечення:

- VirtualBox 6.1 або новіша (з доступним VBoxManage);
- стабільне підключення до Інтернету для віддаленого завантаження до Moodle;
- .NET 9 Runtime.

4. Повідомлення оператору

Під час роботи програми можуть виникати наступні повідомлення.

"Помилка завантаження ВМ": VirtualBox не виявлено або ВМ не зареєстровані. Перевірте встановлення VirtualBox і виконайте команду VBoxManage list vms у терміналі.

"Недоступна вкладка Moodle": Відсутнє підключення до Moodle або спільні папки не налаштовані. Перевірте налаштування ВМ і мережу.

"Метрики не оновлюються": ВМ не активна або проблема з CLI. Перезапустіть ВМ або програму.

"Помилка виконання команди": Некоректна команда в терміналі. Перевірте синтаксис і права доступу. Для вирішення зверніться до лог-файлів у папці Logs або до адміністратора.

5. Опис роботи програми

Доступний функціонал.

Управління ВМ: Програма дозволяє переглядати список ВМ, запускати, зупиняти, перезавантажувати їх, імпортувати нові машини Pocket Moodle.

Моніторинг продуктивності: відображає метрики ВМ (CPU, RAM, диск, мережа, аптайм) у реальному часі на вкладці «Метрики» з оновленням кожну секунду.

Інтеграція з Moodle: надає доступ до налаштувань ВМ з Moodle через вкладку «Moodle».

Встановлення Moodle: кнопка «Install Moodle» на вкладці «Керування» автоматизує встановлення Moodle у вибраній ВМ.

Інструкції до виконання операцій:

1. Запуск програми та перегляд ВМ. Запустіть Pocket Moodle GUI через. На лівій панелі відобразиться список ВМ, зареєстрованих у VirtualBox. Якщо список порожній, перевірте VirtualBox за допомогою команди VBoxManage list vms.
2. Запуск ВМ. Виберіть ВМ на лівій панелі, перейдіть на вкладку «Керування» і натисніть «Запустити». Статус ВМ зміниться на «Running»,

а інтерфейс оновиться. Для зупинки або перезавантаження використовуйте відповідні кнопки.

3. Конфігурація ВМ. Виберіть ВМ, перейдіть на вкладку «Конфігурація», змініть параметри (наприклад, кількість ядер CPU або обсяг RAM) і натисніть «Застосувати». Зміни збережуться, а статус ВМ оновиться в інтерфейсі.
4. Моніторинг метрик. Виберіть активну ВМ, перейдіть на вкладку «Метрики». Перевірте показники (CPU, RAM, диск, мережа, аптайм), які оновлюються в реальному часі.
5. Підключення до Moodle. Виберіть ВМ із встановленим Moodle, перейдіть на вкладку «Moodle» та перейдіть за згенерованим посиланням.
6. Встановлення Moodle. На вкладці «Керування» виберіть ВМ і натисніть «Install Moodle». Програма автоматично налаштує Moodle у ВМ. Після завершення перейдіть на вкладку «Moodle» для доступу.

ДОДАТОК В

Лістинг коду «PMVM.cs»

```
public class PMVM : VM
{
    protected override SshCredentials? DefaultCredentials =>
new(){
        User = "sysadmin",
        Password = "1Qaaaaaa$"
    };
    internal PMVM(VBoxManager manager, string name, string
id) : base(manager, name, id)
    {
        _enableVPN = GetVPN();
    }

    private bool _enableVPN;
    public bool EnableVPN{
        get
        {
            return _enableVPN;
        }
        set
        {
            if(_enableVPN != value)
                SetVPN(value).Wait();
            _enableVPN = value;
        }
    }

    private async Task SetVPN(bool enabled){
        await SSHExecAsync($"sudo {(enabled ? "pon" :
"poff")} scs");
    }
    private bool GetVPN(){
        try{
            GetIP("ppp0");
```

```
        }
        catch(ArgumentException e) {return false;}
        return true;
    }
    public string MoodleURL{
        get
        {
            string ip = GetIP(EnableVPN ? "ppp0" : "enp0s3");
            return $"http://{ip}/moodle";
        }
    }

    private string GetIP(string dev_name)
    {
        var command = $"ifconfig {dev_name} | awk '/inet /
{{print $2}}'";
        var ip = SSHExec(command);
        if(!ip.Any())
            throw new ArgumentException("Invalid device
name");
        return ip.Trim();
    }

    public static bool IsMoodleMachine(VM vm)
    {
        try{
            using (var disk = vm.GetDisk(0)){
                var volumeManager = new VolumeManager(disk);
                var volumes =
volumeManager.GetLogicalVolumes();
                if (volumes.Length == 0)
                {
                    //Debug.WriteLine($"No logical volumes
found in disk.");
                    return false;
                }
```

```
                var volume = volumes[2];
                using (var fsStream = volume.Open()){
                    try{
                        using (var fs = new
ExtFileSystem(fsStream, null))
                        {
                            string moodlePath =
"\\var\\www\\moodledata";
                            bool exists =
fs.DirectoryExists(moodlePath);
                            //Debug.WriteLine($"Checked
{moodlePath}: Exists={exists}");
                            return exists;
                        }
                    }
                    catch (Exception e){
                        //Debug.WriteLine($"Invalid or
unsupported filesystem in disk. {e.Message}.");
                        return false;
                    }
                }
            }
            catch(Exception e){
                Debug.WriteLine("Exception while checking if
machine is PMVM: " + e.Message);
                return false;
            }
        }

        public override string GetMachineHealth()
        {
            string baseHealth = base.GetMachineHealth();
            if (baseHealth != "Healthy")
            {
                return baseHealth;
            }
```

```
        try
        {
            // Moodle-specific check: Apache status
            string apacheCommand = "systemctl is-active
apache2";
            string apacheStatus = SSHExec(apacheCommand);
            bool isApacheRunning = apacheStatus.Trim() ==
"active";

            return isApacheRunning ? "Healthy" : "Warning:
Apache not running";
        }
        catch (Exception ex)
           {
         Debug.WriteLine($"Error in Moodle-specific health
check: {ex.Message}");
            return "Unknown";
        }
    }
}
```

ДОДАТОК Г

Лістинг коду «VM.cs»

```
public class VM
{
    //Override DefaultCredentials to provide them for your
type of VM
    protected virtual SshCredentials? DefaultCredentials
{get;} = null;
    private SshClient _client;
    private readonly TimeSpan _connectTimeout =
TimeSpan.FromSeconds(5);
    private readonly TimeSpan _commandTimeout =
TimeSpan.FromSeconds(10);
    protected readonly VBoxManager _manager;
    private AbstractConfigStorage? _configStorage;
    public string Name { get; }
    public string Id { get; }
    private CancellationTokenSource _cts;
    private Task _monitoringTask;
    private MachineMetrics _metrics = new();
    public MachineMetrics? Metrics {
        get{
            return _metrics is not null && _cts != null &&
!_cts.IsCancellationRequested ? _metrics : null;
        }
    }

    public MachineState GetState()
    {
        var getState = GetStateAsync();
        getState.Wait();
        return getState.Result;
    }

    private MachineConfig _machineConfig;
```

```
    public MachineConfig Config {
        get
        {
            if(_machineConfig is null){
                GatherConfiguration().Wait();
                _configTracker = new(_machineConfig);
            }
            return _machineConfig;
        }
        private set => _machineConfig = value;
    }
    private ConfigChangeTracker _configTracker;

    public ConfigChangeTracker ConfigTracker {get =>
_configTracker;}

    internal VM(VBoxManager manager, string name, string id,
AbstractConfigStorage? configStorage = null)
    {
        _manager = manager;
        Name = name;
        Id = id;
        _configStorage = configStorage;
    }
    public async Task SSHConnect(string username, string
password, CancellationToken ct = default)
    {
        var MAC = Regex.Replace(
            Config.NetworkAdapters.First(a=>a.Type ==
NetworkAdapterType.Bridged).MAC,
            @"(.{2})(.{2})(.{2})(.{2})(.{2})(.{2})",
            "$1:$2:$3:$4:$5:$6")
        .ToLower();
        string host =
PocketMoodleGUI.Tools.Killmepls.FindIpByMac(MAC);
        int port = 22;
```

```
        // Create and configure the client
        _client = new SshClient(host, port, username,
password)
        {
            // This Timeout applies to the TCP+SSH handshake
            ConnectionInfo =
            {
                Timeout = TimeSpan.FromSeconds(5)
            }
        };

        try
        {
            _client.Connect();    // throws
SshOperationTimeoutException on timeout
        }
        catch (SshOperationTimeoutException tex)
        {
            throw new TimeoutException($"SSH connect timed
out after 5s", tex);
        }
        catch (SshAuthenticationException aex)
        {
            throw new InvalidOperationException("SSH
authentication failed", aex);
        }
        catch (SshConnectionException cex)
        {
            throw new InvalidOperationException("SSH
connection failure", cex);
        }

        if (!_client.IsConnected)
            throw new InvalidOperationException("Unknown
error: client not connected after Connect().");
    }
    public async Task SSHConnect(){
```

```
        if(_configStorage is not null)
            try{
                SshCredentials? credentials = await
_configStorage.GetCredentialsAsync(Id);
                if(credentials is null) throw new
InvalidOperationException("No stored credentials found");
                await SSHConnect(credentials?.User,
credentials?.Password);
                return;
            }
        catch(InvalidOperationException){}
        if(DefaultCredentials != null){
            try{
                await SSHConnect(DefaultCredentials?.User,
DefaultCredentials?.Password);
                return;
            }
            catch(InvalidOperationException){}
        }
        throw new SshAuthenticationException("No stored or
default credentials found");
    }
    public async Task<string> SSHExecAsync(string command,
CancellationToken cancellationToken = default)
    {
        // 1) Ensure VM is running (your check)
        if (await GetStateAsync() != MachineState.Running)
            throw new InvalidOperationException("The virtual
machine is not running.");

        // 2) Connect with timeout
        if (_client is null || !_client.IsConnected)
        {
            await SSHConnect();
        }
        // 3) Execute command with timeout
```

```
        string res = "";

        Thread sshThread = new(()=>{
            var cmd = _client.CreateCommand(command);
            //cmd.CommandTimeout = _commandTimeout;
            var execTask = cmd.Execute();
            res = cmd.Result;
        });
        sshThread.Start();
        sshThread.Join();
        return res;
    }
    public string SSHExec (string command)
    {
        var task = SSHExecAsync(command);
        task.Wait();
        return task.Result;
    }
    public async Task<MachineState> GetStateAsync()
    {
        var output = await
_manager.ExecuteCommandAsync($"showvminfo {Id} --
machinereadable");
        using (var reader = new StringReader(output))
        {
            string line;
            while ((line = reader.ReadLine()) != null)
//TODO: Rewrite with string.Find
            {
                if (line.StartsWith("VMState="))
                {
                    var state = line.Split('=')[1].Trim('"');
                    return ParseMachineState(state);
                }
            }
        }
        return MachineState.Unknown;
```

```
    }

    public async Task StartAsync(bool headless = true) =>
        await _manager.ExecuteCommandAsync($"startvm {Id} --
type {(headless ? "headless" : "gui")}", true);

    public async Task StopAsync() =>
        await _manager.ExecuteCommandAsync($"controlvm {Id}
poweroff", true);

    public async Task PauseAsync() =>
        await _manager.ExecuteCommandAsync($"controlvm {Id}
pause", true);

    public async Task ResumeAsync() =>
        await _manager.ExecuteCommandAsync($"controlvm {Id}
resume", true);

    public async Task TakeSnapshotAsync(string snapshotName,
string description = "") =>
        await _manager.ExecuteCommandAsync($"snapshot {Id}
take \"{snapshotName}\" --description \"{description}\"");

    private MachineState ParseMachineState(string state)
    {
        switch (state.ToLower())
        {
            case "running": return MachineState.Running;
            case "poweredoff": return
MachineState.PoweredOff;
            case "saved": return MachineState.Saved;
            case "paused": return MachineState.Paused;
            case "aborted": return MachineState.Aborted;
            default: return MachineState.Unknown;
        }
    }
```

```
    public async Task StartMonitoringMetrics()
    {
        if (_cts != null && !_cts.IsCancellationRequested)
        {
            // Already monitoring; do nothing
            return;
        }

        _cts?.Dispose(); // Clean up any previous CTS
        _cts = new CancellationTokenSource();

        _monitoringTask = Task.Run(async () =>
        {
            try
            {
                await foreach (var metrics in
_manager.StartMonitoringMetrics(Name, _cts.Token))
                {
                    UpdateMetrics(metrics); // Update metrics
as they arrive
                }
            }
            catch (OperationCanceledException)
            {
                // Expected when canceled; reset metrics
                _metrics = new MachineMetrics();
            }
            catch (Exception ex)
            {
                Debug.WriteLine($"Monitoring error for
{Name}: {ex.Message}");
            }
        }, _cts.Token);
        var waitMetricsTask = Task.Run(() => {
            while (_metrics is null) Thread.Sleep(100);
        });
        var timeoutTask =
```

```
Task.Delay(TimeSpan.FromSeconds(10));
        Task.WaitAny(waitMetricsTask, timeoutTask);
        if(!waitMetricsTask.IsCompleted)
            throw new TimeoutException("Machine metrics was
not updated in 10 seconds.");

    }

    private void UpdateMetrics(MachineMetrics newMetrics)
    {
        _metrics.Timestamp = newMetrics.Timestamp;
        _metrics.CpuUsage = newMetrics.CpuUsage;
        _metrics.MemoryUsage = newMetrics.MemoryUsage;
        _metrics.DiskRead = newMetrics.DiskRead;
        _metrics.DiskWrite = newMetrics.DiskWrite;
        _metrics.NetworkRx = newMetrics.NetworkRx;
        _metrics.NetworkTx = newMetrics.NetworkTx;
        _metrics.ProcessCount = newMetrics.ProcessCount;
        _metrics.Uptime = newMetrics.Uptime;
    }

    public void StopMonitoringMetrics()
    {
        if (_cts != null && !_cts.IsCancellationRequested)
        {
            _cts.Cancel();
            _monitoringTask?.Wait(); // Optional: wait for
task to complete
            _cts.Dispose();
            _cts = null;
            _metrics = new MachineMetrics(); // Reset metrics
when stopped
        }
    }
    private void ParseMetricLine(string line, MachineMetrics
metrics)
    {
```

```
        var parts = line.Split(new[] { ':' }, 2);
        if (parts.Length != 2) return;

        var key = parts[0].Trim();
        var value = parts[1].Trim();

        try
        {
            switch (key)
            {
                case "Guest/CPU/Load/User":
                    metrics.CpuUsage =
(int)(double.Parse(value) * 100);
                    break;
                case "Guest/RAM/Usage/Total":
                    metrics.MemoryUsage = long.Parse(value) /
(1024 * 1024);
                    break;
                case "Guest/Storage/Read/Bytes":
                    metrics.DiskRead = long.Parse(value);
                    break;
                case "Guest/Storage/Write/Bytes":
                    metrics.DiskWrite = long.Parse(value);
                    break;
                case "Guest/Net/Rate/Rx":
                    metrics.NetworkRx = long.Parse(value);
                    break;
                case "Guest/Net/Rate/Tx":
                    metrics.NetworkTx = long.Parse(value);
                    break;
                case "Guest/System/Process/Count":
                    metrics.ProcessCount = int.Parse(value);
                    break;
                case "Guest/System/Uptime":
                    metrics.Uptime =
TimeSpan.FromMilliseconds(long.Parse(value));
                    break;
```

```
            }
        }
        catch (FormatException) { /* Ignore parse errors */ }
    }

    public virtual string GetMachineHealth()
    {
        return GetState() == MachineState.Running ? "Healthy"
: "Stoped";
    }
    private async Task<MachineConfig> GatherConfiguration()
    {
        var output = await
_manager.ExecuteCommandAsync($"showvminfo \"{Name}\" --
machinereadable");
        var parser = new Tools.VMConfigParser(output);
        var name = Name;
        var osType = parser.GetOSType();
        var memoryMB = parser.GetMemoryMB();
        var cpuCount = parser.GetCPUCount();
        var vramMB = parser.GetVRAMMB();
        var enableAudio = parser.IsAudioEnabled();
        var enableNestedPaging =
parser.IsNestedPagingEnabled();
        ObservableCollection<NetworkAdapter> networkAdapters
= new (parser.ParseNetworkAdapters());
        ObservableCollection<StorageController>
storageControllers = new(parser.ParseStorageControllers());
        ObservableCollection<DiskAttachment> diskAttachments
= new(parser.ParseDiskAttachments());

        var enableEFI = parser.IsEFIEnabled();
        var enable3DAcceleration =
parser.Is3DAccelerationEnabled();
        var conf =  new MachineConfig
        {
            //Linking gathered config to fields in
```

```
MachineConfig class
        };
        Config = conf;
        return conf;
    }

    public async Task CommitConfig()
    {
        if (_configTracker==null &&
!_configTracker.ChangedProperties.Any())
        {
            return;
        }

        MachineState currentState = GetState();

        if(currentState == MachineState.Running){
            await StopAsync();
        }

        var commands = new List<string>();
        var changedProps = _configTracker.ChangedProperties;
// For efficient lookup

        // Basic Identification
        if
(changedProps.Contains(nameof(MachineConfig.Name)))
            commands.Add($"modifyvm \"{Id}\" --name
\"{Config.Name}\"");
        if
(changedProps.Contains(nameof(MachineConfig.Description)))
            commands.Add($"modifyvm \"{Id}\" --description
\"{Config.Description}\"");
        if
(changedProps.Contains(nameof(MachineConfig.OSType)))
            commands.Add($"modifyvm \"{Id}\" --ostype
\"{Config.OSType}\"");
```

```
        // Hardware Resources
        if
(changedProps.Contains(nameof(MachineConfig.MemoryMB)))
            commands.Add($"modifyvm \"{Id}\" --memory
{Config.MemoryMB}");
        if
(changedProps.Contains(nameof(MachineConfig.CPUCount)))
            commands.Add($"modifyvm \"{Id}\" --cpus
{Config.CPUCount}");
        if
(changedProps.Contains(nameof(MachineConfig.EnablePAE)))
            commands.Add($"modifyvm \"{Id}\" --pae
{(Config.EnablePAE ? "on" : "off")}");
        if
(changedProps.Contains(nameof(MachineConfig.EnableNestedPagin
g)))
            commands.Add($"modifyvm \"{Id}\" --nestedpaging
{(Config.EnableNestedPaging ? "on" : "off")}");
        if
(changedProps.Contains(nameof(MachineConfig.VRAMMB)))
            commands.Add($"modifyvm \"{Id}\" --vram
{Config.VRAMMB}");
        if
(changedProps.Contains(nameof(MachineConfig.Enable3DAccelerat
ion)))
            commands.Add($"modifyvm \"{Id}\" --accelerate3d
{(Config.Enable3DAcceleration ? "on" : "off")}");
        if
(changedProps.Contains(nameof(MachineConfig.EnableAudio)))
            commands.Add($"modifyvm \"{Id}\" --audio
{(Config.EnableAudio ? "enabled" : "none")}");

        // Storage Controllers
        if
(changedProps.Contains(nameof(MachineConfig.StorageController
s)))
```

```
        {
            // For simplicity, recreate all controllers if
the collection changed
            foreach (var controller in
Config.StorageControllers)
            {
                commands.Add($"storagectl \"{Id}\" --name
\"{controller.Name}\" --remove");
                commands.Add($"storagectl \"{Id}\" --name
\"{controller.Name}\" --add
{controller.Type.ToString().ToLower()} --controller
{controller.Type} --portcount {controller.PortCount}");
            }
        }

        // Disk Attachments
        if
(changedProps.Contains(nameof(MachineConfig.DiskAttachments))
)
        {
            // Recreate all attachments if the collection
changed
            foreach (var attachment in
Config.DiskAttachments)
            {
                commands.Add($"storageattach \"{Id}\" --
storagectl \"{attachment.ControllerName}\" --port
{attachment.Port} --device {attachment.Device} --type hdd --
medium \"{attachment.Path}\"");
            }
        }

        // Network Adapters
        if
(changedProps.Contains(nameof(MachineConfig.NetworkAdapters))
)
        {
```

```
            for (int i = 0; i < Config.NetworkAdapters.Count;
i++)
            {
                var adapter = Config.NetworkAdapters[i];
                commands.Add($"modifyvm \"{Id}\" --nic{i + 1}
{adapter.Type.ToString().ToLower()}");
            }
        }

        // Boot Configuration
        if
(changedProps.Contains(nameof(MachineConfig.BootOrder)))
        {
            commands.Add($"modifyvm \"{Id}\" --boot1
{Config.BootOrder.First.ToString().ToLower()}");
            commands.Add($"modifyvm \"{Id}\" --boot2
{Config.BootOrder.Second.ToString().ToLower()}");
            commands.Add($"modifyvm \"{Id}\" --boot3
{Config.BootOrder.Third.ToString().ToLower()}");
            commands.Add($"modifyvm \"{Id}\" --boot4
{Config.BootOrder.Fourth.ToString().ToLower()}");
        }
        if
(changedProps.Contains(nameof(MachineConfig.EnableEFI)))
            commands.Add($"modifyvm \"{Id}\" --firmware
{(Config.EnableEFI ? "efi" : "bios")}");

        // Advanced Features
        if
(changedProps.Contains(nameof(MachineConfig.EnableClipboard))
)
            commands.Add($"modifyvm \"{Id}\" --clipboard
{(Config.EnableClipboard ? "bidirectional" : "disabled")}");
        if
(changedProps.Contains(nameof(MachineConfig.EnableDragAndDrop
)))
            commands.Add($"modifyvm \"{Id}\" --draganddrop
```

```
{(Config.EnableDragAndDrop ? "bidirectional" :
"disabled")}");
        if
(changedProps.Contains(nameof(MachineConfig.EnableRemoteDispl
ay)) ||
changedProps.Contains(nameof(MachineConfig.RemoteDisplayPort)
))
        {
            if (Config.EnableRemoteDisplay)
                commands.Add($"modifyvm \"{Id}\" --vrde on --
vrdeport {Config.RemoteDisplayPort}");
            else
                commands.Add($"modifyvm \"{Id}\" --vrde
off");
        }
        if
(changedProps.Contains(nameof(MachineConfig.EnableUSBControll
er)))
            commands.Add($"modifyvm \"{Id}\" --usb
{(Config.EnableUSBController ? "on" : "off")}");
        if
(changedProps.Contains(nameof(MachineConfig.EnableUSBOHCI)))
            commands.Add($"modifyvm \"{Id}\" --usbohci
{(Config.EnableUSBOHCI ? "on" : "off")}");
        if
(changedProps.Contains(nameof(MachineConfig.EnableUSBXHCI)))
            commands.Add($"modifyvm \"{Id}\" --usbxhci
{(Config.EnableUSBXHCI ? "on" : "off")}");
        // Display Configuration
        if
(changedProps.Contains(nameof(MachineConfig.MonitorCount)))
            commands.Add($"modifyvm \"{Id}\" --monitorcount
{Config.MonitorCount}");
        if
(changedProps.Contains(nameof(MachineConfig.Enable2DVideoAcce
leration)))
            commands.Add($"modifyvm \"{Id}\" --
```

```
accelerate2dvideo {(Config.Enable2DVideoAcceleration ? "on" :
"off")}");
        // Serial Ports
        if
(changedProps.Contains(nameof(MachineConfig.SerialPorts)))
        {
            for (int i = 0; i < Config.SerialPorts.Count;
i++)
            {
                var port = Config.SerialPorts[i];
                commands.Add($"modifyvm \"{Id}\" --uart{i +
1} {port.Mode} \"{port.Path}\"");
            }
        }
        // Shared Folders
        if
(changedProps.Contains(nameof(MachineConfig.SharedFolders)))
        {
            // Remove all existing shared folders and re-add
the current ones
            commands.Add($"sharedfolder remove \"{Id}\" --
all");
            foreach (var folder in Config.SharedFolders)
            {
                commands.Add($"sharedfolder add \"{Id}\" --
name \"{folder.Name}\" --hostpath \"{folder.HostPath}\" --
automount {folder.AutoMount}");
            }
        }
        //TODO: Add machine restart request

        // Execute commands
        foreach (var command in commands)
        {
            await _manager.ExecuteCommandAsync(command);
        }
```

```
        if(currentState != MachineState.PoweredOff){ //TODO:
Should change to corresponding state, not just turn on in any
case
            await StartAsync();
        }
        // Reset the tracker after successful commit
         _configTracker.Reset();
         _manager.StartUpdatingMachines();
    }

    public Disk GetDisk(int diskIndex){ //FIXME: Disk type
can be not VMDK
        string diskPath =
Config.DiskAttachments[diskIndex].Path;
        var diskStream = File.OpenRead(diskPath);
        var disk = new Disk(diskStream, Ownership.None );
        return disk;
    }

    public void Dispose()
    {
        _client?.Disconnect();
    }
}
```

ДОДАТОК Д

Лістинг коду «VBoxManager.cs»

```
public class VBoxManager: IDisposable, IVMManager
{
    private Dictionary<string, VM>? _vms;
    private List<(Func<VM, bool> Predicate, Func<VM, VM> Converter)>
_vmTypesRegistry = new();
    private Process? _currentContinuousProcess; //TODO: Remake with
list or similar
    private Task? _updatingTask;
    public bool IsUpdating {get; private set;}
    private CancellationTokenRegistration _cancellationRegistration;
    public delegate void MachinesChangedHandler();
    public event MachinesChangedHandler OnMachinesChanged =
new(()=>{});
    public Dictionary<string, VM> VMs {
        get {
                if(IsUpdating)
                    _updatingTask.Wait();
                if(_vms == null){
                    UpdateMachines();
                }
                if(_vms != null)
                    return _vms;
                throw new Exception("Failed to load VMs"); //TODO:Find
better exception base
            }
    }
    public void RegisterVMSubtype(Func<VM, bool> predicate, Func<VM,
VM> convertor){
        _vmTypesRegistry.Add((predicate, convertor));
    }
    public void UpdateMachines()
    {
        if(!IsUpdating)
            _updatingTask = UpdateMachinesAsync();
        _updatingTask?.Wait();
    }
    public void StartUpdatingMachines(){
        _updatingTask = UpdateMachinesAsync();
    }
```

```
private async Task UpdateMachinesAsync()
{
    if(IsUpdating)
        return;
    IsUpdating = true;
    var vms = await GetMachinesAsync();
    _vms = vms.ToDictionary(vm => vm.Id);
    IsUpdating = false;
    _updatingTask = null;
    OnMachinesChanged?.Invoke();
}

private readonly string _vboxManagePath;
public VBoxManager()
{
    // Find VBoxManage in standard locations
    _vboxManagePath = FindVBoxManagePath();
}

public async Task<List<VM>> GetMachinesAsync()
{
    var output = await ExecuteCommandAsync("list vms");

    var machines = output
    .Split('\n', StringSplitOptions.RemoveEmptyEntries)
    .Select(
        mi => new VM(
            this,
            mi.GetBetween("\"").First(),
            mi.GetBetween("{", "}").First()
        )
    )
    .Select(vm => {
        foreach (var (predicate, convertor) in _vmTypesRegistry)
        {
            if (predicate(vm))
                return convertor(vm);
        }
        return vm;
    })
    .ToList();
    return machines;
}
```

```
    private string FindVBoxManagePath()
    {
        // Check common installation paths
        var paths = new[]
        {
            "/usr/bin/VBoxManage",
            "/usr/local/bin/VBoxManage",
            "C:\\Program Files\\Oracle\\VirtualBox\\VBoxManage.exe"
        };

        foreach (var path in paths)
        {
            if (File.Exists(path))
                return path;
        }

        throw new FileNotFoundException("VBoxManage not found in
standard locations");
    }

    internal async Task<string> ExecuteCommandAsync(string arguments,
bool isDirtyCommand = false)
    {
        var process = new Process
        {
            StartInfo = new ProcessStartInfo
            {
                FileName = _vboxManagePath,
                Arguments = arguments,
                RedirectStandardOutput = true,
                RedirectStandardError = true,
                UseShellExecute = false,
                CreateNoWindow = true
            }
        };

        process.Start();
        Task.Delay(TimeSpan.FromSeconds(1)).Wait();//TODO: Remove
later. This code temporaarly avoids deadlock somhere under here.
        Task<string> outputTask =
process.StandardOutput.ReadToEndAsync();
        Task<string> errorTask =
```

```
process.StandardError.ReadToEndAsync();
        string output = await outputTask;
        string error = await errorTask;
        await process.WaitForExitAsync();

        if (process.ExitCode != 0)
            throw new InvalidOperationException($"VBoxManage failed:
{error}");
        if(isDirtyCommand)
            StartUpdatingMachines();
        return output;
    }

    async Task<Stream> StartExecutingCommandAsync(string arguments,
CancellationToken cancellationToken = default)
    {
        cancellationToken.ThrowIfCancellationRequested();

        _currentContinuousProcess = new Process
        {
            StartInfo = new ProcessStartInfo
            {
                FileName = _vboxManagePath,
                Arguments = arguments,
                RedirectStandardOutput = true,
                RedirectStandardError = true,
                UseShellExecute = false,
                CreateNoWindow = true
            }
        };

        _currentContinuousProcess.Start();
        Task.Delay(TimeSpan.FromSeconds(1)).Wait();//TODO: Remove
later. This code temporaarly avoids deadlock somhere under here.

        // Register cancellation callback to kill the process even
after method exit
        _cancellationRegistration = cancellationToken.Register(() =>
        {
            if (_currentContinuousProcess != null &&
!_currentContinuousProcess.HasExited)
            {
                _currentContinuousProcess.Kill();
```

```
                _currentContinuousProcess.Dispose();
                _currentContinuousProcess = null;
            }
        });

        // Check cancellation immediately after starting
        if (cancellationToken.IsCancellationRequested)
        {
            _currentContinuousProcess.Kill();
            _currentContinuousProcess.Dispose();
            _currentContinuousProcess = null;
            _cancellationRegistration.Dispose(); // Clean up
registration
            cancellationToken.ThrowIfCancellationRequested();
        }

        await Task.Yield(); // Allow process to start
        return _currentContinuousProcess.StandardOutput.BaseStream;
    }

    internal async IAsyncEnumerable<MachineMetrics>
StartMonitoringMetrics(string vmName,
[EnumeratorCancellation]CancellationToken cancellationToken = default)
    {
        string arguments = $"metrics collect --period 1 {vmName}";
        using (var stream = await StartExecutingCommandAsync(arguments,
cancellationToken))
        using (var reader = new StreamReader(stream))
        {
            var parser = new MetricParser();
            var metrics = new MachineMetrics();

            while (!reader.EndOfStream &&
!cancellationToken.IsCancellationRequested)
            {
                string line = await
reader.ReadLineAsync(cancellationToken);
                if (string.IsNullOrEmpty(line)) continue;

                metrics = parser.ParseMetricsLine(line, metrics);

                // Yield when a complete set of metrics is collected
                if (metrics.CpuUsage.HasValue &&
```

```
metrics.MemoryUsage.HasValue)
                {
                    yield return metrics;
                    metrics = new MachineMetrics(); // Reset for next
batch
                }
            }

            // If canceled, ensure the process is terminated
            if (cancellationToken.IsCancellationRequested &&
_currentContinuousProcess != null &&
!_currentContinuousProcess.HasExited)
            {
                _currentContinuousProcess.Kill();
            }
        }
    }

    public void Dispose()
    {
        if (_currentContinuousProcess != null &&
!_currentContinuousProcess.HasExited)
        {
            _currentContinuousProcess.Kill(); // Terminate the process
if still running
            _currentContinuousProcess.Dispose();
            _currentContinuousProcess = null;
        }
    }
}
```

АНОТАЦІЯ

Матюхін В.А. Аналіз та використання API Virtual Box для створення GUI для Pocket Moodle.

Рукопис

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 122 Комп'ютерні науки. Волинський національний університет імені Лесі Українки, Луцьк, 2025 р.

У кваліфікаційній роботі представлено розробку системи Pocket Moodle GUI — кросплатформенного програмного засобу з графічним інтерфейсом для управління віртуальними машинами (ВМ) на базі VirtualBox та інтеграції з навчальною платформою Moodle. Актуальність дослідження зумовлена зростаючою потребою в інтеграції віртуалізації в освітні процеси, що дозволяє оптимізувати ресурси, забезпечувати гнучкість у використанні різних операційних систем і знижувати витрати на інфраструктуру. Метою роботи є створення зручного та ефективного інтерфейсу для керування ВМ, моніторингу їхньої продуктивності та взаємодії з Moodle у навчальному середовищі.

Основні завдання включали розробку архітектури системи на основі багаторівневої моделі з використанням патерну MVVM, створення бібліотеки VBoxAPI для асинхронної взаємодії з CLI VirtualBox, реалізацію графічного інтерфейсу на базі фреймворку Avalonia UI, інтеграцію з Moodle та аналіз ефективності системи через тестові сценарії. Для взаємодії з VirtualBox обрано CLI-інтерфейс (VBoxManage) через його стабільність, повноту функціоналу та кросплатформність, що підтверджено порівняльним аналізом із COM/XPCOM і Web API. Підсистема збору метрик продуктивності (CPU, RAM, диск, мережа, аптайм) реалізована з використанням асинхронного парсингу виводу команди VBoxManage metrics collect, забезпечуючи оновлення даних у реальному часі.

Порівняльний аналіз аналогів показав, що Pocket Moodle GUI вирізняється унікальним поєднанням реактивного GUI, інтеграції з Moodle, розширеного моніторингу та автоматизації. Результати тестування підтвердили стабільність,

зручність і високу продуктивність системи, що робить її ефективним інструментом для віртуалізованих навчальних середовищ.

Ключові слова: віртуалізація, VirtualBox, Moodle, графічний інтерфейс, CLI, кросплатформність, MVVM, Avalonia UI, асинхронність, моніторинг продуктивності.