Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка автоматизованої системи для вивчення відкритих операційних систем на основі хмарних технологій інструментами GNU/LINUX»

Виконав: студент 4 курсу
групи 1ПІ-18б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Ісаков А.В.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Войтко В.В.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. КН Арсенюк І.Р.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____
«____» _____ 2022 р.

ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії
Кафедра програмного забезпечення
Ступінь вищої освіти – бакалавр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
“_25_” __березня__2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Ісакову Андрієві Васильовичу

1. Тема роботи – розробка автоматизованої системи для вивчення відкритих операційних систем на основі хмарних технологій інструментами GNU/LINUX.
Керівник роботи: Войтко Вікторія Володимирівна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від “_24_” _березня_2022 року №_66_
2. Строк подання студентом роботи _____13 червня 2022 року_____
3. Вихідні дані до роботи: середовище розробки GNU Emacs, мова розробки Python, BASH та SQL, операційна система – GNU/Linux, система управління базою даних – PostgreSQL, базові методи розгортання навчального середовища та алгоритми перевірки виконання завдань студентами.
4. Зміст розрахунково-пояснювальної записки: вступ; обґрунтування вибору методу розробки та постановка задач дослідження; розробка моделей системи вивчення відкритих ОС та методу перевірки виконання завдань; програмна розробка системи вивчення відкритих ОС; тестування системи вивчення відкритих ОС; висновки; список використаних джерел; додатки.
5. Перелік графічного матеріалу: графічний інтерфейс рейтингової таблиці; метод перевірки виконання завдань; моделі роботи системи “зсередини” та з позиції

студента; блок-схеми алгоритму перевірки виконання завдань; ER-модель та ER-діаграма бази даних; розробка системи; графічний інтерфейс веб-сайту VNTU Linuxoids; тестування системи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Войтко В. В., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання 25 березня 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз систем для вивчення відкритих операційних систем та вибір постановка чітких задач	26.03.2022 – 10.04.2022	Вик.
2	Розробка моделі та методу системи	11.04.2022 – 20.04.2022	Вик.
3	Розробка системи вивчення відкритих операційних систем	21.04.2022 – 01.05.2022	Вик.
4	Тестування роботи системи	02.05.2022 – 20.05.2022	Вик.
5	Оформлення матеріалів до захисту БДР	21.05.2022 – 10.06.2022	Вик.

Студент _____ Ісаков А.В.
(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ Войтко В.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 95 сторінок формату А4, на яких є 33 рисунки, 3 таблиці, список використаних джерел містить 26 найменувань.

У бакалаврській дипломній роботі проаналізовано наявні системи для вивчення відкритих ОС. Було запропоновано власну розробку, яка задовольняє потреби якомога більшої кількості студентів чи всіх тих, хто вивчає відкриті ОС.

Розроблено модель системи вивчення відкритих операційних систем, що дозволяє ефективніше вивчати операційні системи майбутнім IT-спеціалістам.

Розроблено метод перевірки виконання завдань для системи вивчення відкритих операційних систем, що дозволяє швидко перевіряти завдання у великої кількості студентів.

Розроблено скрипт, з допомогою якого здійснюється керування системою, зокрема, автоматизоване розгортання навчального середовища та перевірка завдань. Скрипт було створено з допомогою мови програмування Python. Для автоматизованого розгортання було використано Ansible, для контейнеризації – LXC, для збереження виконаних завдань PostgreSQL, для перетворення образів віртуальних машин на контейнери – BASH, для веб-інтерфейсу рейтингової таблиці – Python Flask, CSS/HTML, JavaScript та NGINX, для інструкції користувача – Markdown. Розробка велась в GNU Emacs та VIM.

Ключові слова: відкриті операційні системи (ОС), LXC, автоматизоване розгортання, Python, скрипт.

ABSTRACT

The bachelor's thesis consists of 95 A4 pages, which have 33 figures, 3 tables, the list of sources used contains 26 items.

In this bachelor's thesis existing systems for studying open operating systems have been analyzed. It was proposed to develop a new system, which would meet the needs of as many students as possible or any one who studies open source OSs.

A model of a system for studying open OSs has been developed, which allows future IT specialists to study operating systems more efficiently.

A method for validating the tasks for the system of studying open operating systems has been developed. This method allows one to quickly check the tasks of a large number of students.

A script has been developed to control the system, including automated learning environment deployment and task validation. The script was created using the Python programming language. Ansible was used for automated deployment, LXC for containerization, PostgreSQL to store completed tasks, BASH to convert virtual machine images into containers, Python Flask, CSS / HTML, JavaScript and NGINX for the rating table's web interface, Markdown for user README. Development was carried out in GNU Emacs and VIM.

Keywords: open operating systems (OS), LXC, automated provisioning, Python, script.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Нині професії в IT-сфері є одними з найпопулярніших [1] у світі.

Нетривіальною задачею кожного IT-спеціаліста є організація його робочих процесів. Одним із найважливіших виборів, який суттєво впливатиме на продуктивність фахівця, є вибір операційної системи.

Операційна система (ОС) – це базовий комплекс програм, що виконує керування апаратною складовою комп'ютера або віртуальної машини; забезпечує керування обчислювальним процесом і організовує взаємодію з користувачем [2].

ОС поділяють на відкриті та власницькі (proprietary). Відкриті ОС мають відкритий сирцевий код і кожен охочий може прочитати його, провести аудит, внести власні зміни і т.п. Прикладом відкритих ОС є Linux, BSD, Plan9, OpenSolaris, FreeDOS. Код власницьких ОС – закритий, найчастіше такі ОС виробляє окрема корпорація чи організація, наприклад, Microsoft (Windows) чи Apple (macOS).

Серед ОС для IT-спеціалістів популярним вибором є саме відкриті ОС [3].

Відкриті ОС є особливо привабливими як для бізнесу, так і для навчальних установ, які готують майбутніх фахівців, через наступні переваги:

- зазвичай [4], за ОС та програмне забезпечення (ПЗ) до неї не потрібно платити, оскільки вони ліцензуються відкритими ліцензіями;
- різноманіття ОС дозволяє вибрати саме таку ОС (чи її дистрибутив), що найкраще задовольнить конкретні потреби користувача, наприклад, існує дистрибутив Linux – DebianEdu для використання в навчальних установах, чи ScientificLinux – для науковців;
- вільне і відкрите ПЗ забезпечує кращу безпеку – будь-хто може провести аудит і впевнитись у відсутності слабких місць, а також безпека такого ПЗ буде вищою, бо залучене широке коло експертів (розробники з усього світу), ніж безпека власницького ПЗ (вузьке коло експертів – розробники конкретної компанії);

- можна легко отримати допомогу завдяки величезній спільноті розробників та користувачів – якщо є якась проблема, то її вже скоріш за все вирішили і вона описана на форумах чи в технічній документації, якщо ні, то її швидко вирішать.

Таким чином, і бізнесу, і навчальним установам, і окремим індивідам, зацікавленим розвинутися в IT, важливо знаходити можливості для вивчення вільних операційних систем.

На разі існує чимало можливостей, які загалом можна поділити на:

- MOOC (масові онлайн-курси) – вони в основному складаються з теорії, тестів для самоперевірки та іноді надають можливість поекспериментувати в тимчасово створеному середовищі (sandbox).
- Спеціалізовані курси – курси від розробників комерційних відкритих ОС чи розробників хмарних та мережевих сервісів, наприклад, RedHat, Amazon AWS, Google Cloud, Microsoft Azure, Cisco – націлені на здобуття сертифікацій, що їх надає відповідна компанія.
- Спеціалізовані системи для вивчення кібербезпеки та пентестингу (pentest – penetration testing – тестування проникнення). Це, зазвичай, віртуальні машини, до яких треба отримати адміністративний доступ скориставшись слабкими місцями системи безпеки. Подібні системи в основному сфокусовані на практичній складовій навчання, а теорію користувач мусить знаходити та вивчати самотужки.

Основним недоліком цих ad hoc систем вивчення ОС є вузька предметна галузь, яку вони дозволяють вивчати, – це або загальні теоретичні навички (MOOC), або підготовка до сертифікації (спеціалізовані курси), або вивчення кібербезпеки. Систему для вивчення одного неможливо використати для вивчення іншого. Це позбавляє викладачів гнучкості в організації навчального процесу, а також обтяжує як викладачів, так і студентів – для кожного нового курсу їм необхідно щоразу вчитися користуватися відповідною системою.

Для розв'язання цієї проблеми доцільною є розробка універсальної розширюваної системи, яку викладачі могли б гнучко конфігурувати та

адаптовувати для власних курсів і від якої студенти отримували б користь у вигляді персоналізованих курсів зі збалансованою практикою та теорією. Тому актуальною є розробка автоматизованої системи для вивчення відкритих операційних систем на основі хмарних технологій інструментами GNU/LINUX.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою бакалаврської дипломної роботи є підвищення ефективності вивчення відкритих ОС шляхом розробки системи, яка дозволяє автоматизувати розгортання навчального середовища та здійснювати автоматичну перевірку виконання завдань відповідно до потреб викладача.

Основними задачами роботи є:

- розробити моделі системи вивчення відкритих ОС;
- розробити метод перевірки виконання завдань;
- розробити базу даних для збереження виконаних завдань;
- розробити рейтингову таблицю;
- розробити формат навчального середовища (для його реплікації);
- розробити шар сумісності (перетворення форматів) між віртуальними машинами та контейнерами;
- розробити інструкцію користувача;
- провести тестування розробленої системи.

Об’єкт дослідження – процес розробки систем для вивчення відкритих ОС.

Предмет дослідження – система для вивчення відкритих ОС.

Методи дослідження. Для реалізації поставлених задач були використані такі методи дослідження:

- методи розгортання з допомогою парадигми “Інфраструктура як код” для реалізації автоматизованого конфігурування;
- методи автоматизації оцінки виконання завдань для швидкої та ефективної перевірки завдань;

- методи зберігання (архівування) інформації для перетворення формату образів віртуальних машин на формат контейнерів;
- методи організації та нормалізації баз даних для оптимізації бази даних розроблюваної системи.

Наукова новизна отриманих результатів.

- Подальшого розвитку отримав метод автоматизації перевірки виконання завдань студентів, який, на відміну від існуючих, використовує протокол безпечного віддаленого з’єднання SSH, щоб отримувати доступ до контейнерів студентів, що дозволяє швидко та ефективно перевіряти завдання студентів, а також одразу відображати їхні успіхи у рейтинговій таблиці.
- Подальшого розвитку отримала модель системи вивчення відкритих ОС, яка, на відміну від існуючих, має більший функціонал та є гнучкою у конфігуруванні – її можна розгортати як на одному сервері, так і на багатьох, оскільки вона має клієнт-серверну архітектуру, тому може бути пристосованою до будь-якого типу курсу, від базового до складного, що дозволяє розширити можливості застосування системи в навчальному процесі.

Практична цінність отриманих результатів. Практична цінність полягає у можливості використання розробленої системи вивчення відкритих ОС навчальними установами, групами ентузіастів чи розробниками комерційних курсів.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У науковій роботі, опублікованій у співавторстві [5], автору належать аналіз різновидів NoSQL баз даних та аналіз сфер застосування NoSQL баз даних в PayPal, eBay, NASA. У другій науковій роботі, опублікованій у співавторстві [6], автору належать обґрунтування головних причин для використання мови Markdown, опис особливостей та приклади синтаксису Markdown .

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на XLIX Науково-технічній конференції Інституту соціально-гуманітарних наук (2020), XLIX Науково-технічній конференції факультету інформаційних технологій та комп’ютерної інженерії (2020) та на LI Науково-технічній конференції факультету інформаційних технологій та комп’ютерної інженерії (2022).

Публікації. Основні результати дослідження опубліковані в наукових роботах – тезах доповідей на конференціях: на [5] XLIX Науково-технічній конференції факультету інформаційних технологій та комп’ютерної інженерії (2020), на [6] XLIX Науково-технічній конференції Інституту соціально-гуманітарних наук (2020) та на [7] LI Науково-технічній конференції факультету інформаційних технологій та комп’ютерної інженерії (2022).

Аналіз. У пояснювальній записці до бакалаврської дипломної роботи було розглянуто 4 розділи та було використано 26 літературних джерел.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану систем для вивчення відкритих операційних систем

На разі існує багато різноманітних систем для вивчення ОС, які створені, щоб задовільнити різні потреби – чи вивчення теорії, чи підготовка до сертифікації (навчання роботі з ПЗ компанії, яка надає сертифікацію, наприклад, AWS), чи практика визначення кіберзагроз.

Явною перевагою таких систем є те, що вони доповнюють традиційні способи навчання (книжки та технічну документацію) забезпечуючи ту чи іншу міру практичного навчання. Ці системи подають уроки таким чином, аби теоретичні та практичні навички легше засвоювалися. Також вони здатні автоматизовано чи напів-автоматизовано (вчитель+система) оцінювати студента (користувача), що більш об’єктивно ніж на класичних курсах без використання таких систем.

Отже, основними перевагами таких систем є те, що студенту **не** потрібно:

- шукати технічну документацію;
- вирішувати в якій послідовності вивчати матеріал;
- створювати середовище для виконання практичних робіт;
- оцінювати себе.

Попри це, є й недоліки. Не всі з цих систем дозволяють вивчати саме відкриті ОС чи вчитися у такий спосіб, як це було б зручно студентові. Наприклад, вони не дозволяють самому студентові розгортати віртуальну машину (VM) за необхідності прямісінько під час проходження курсу чи зберігати свій поступ у VM протягом всього курсу та після його завершення. Велика частина подібних систем побудована за парадигмою I/P/SaaS (Infrastructure / Platform / Software as a Service – Інфраструктура / Платформа / Програма як сервіс), тобто щоб ними користуватися необхідне постійне якісне інтернет-з’єднання. Також їх не можна розгорнути на власній інфраструктурі та пристосовувати до власних потреб.

Тож, проаналізувавши стан наявних систем, закладено загальну ідею – розроблювана система має стати каркасом, який з допомогою невеликої кількості конфігурацій можна буде адаптувати для конкретного застосування – чи для курсу в університеті, чи для груп ентузіастів (для, наприклад, VNTU Linuxoids [8]), чи для комерційних курсів. При цьому вона повинна вдосконалити наявні недоліки – її можна буде встановити на власне обладнання та використовувати без інтернету.

1.2 Порівняльний аналіз аналогів

В попередньому розділі були висвітлені можливі ресурси для вивчення відкритих ОС. Для порівняльного аналізу було обрано ті ресурси, які є максимально наближеними за своєю сутністю до розроблюваної системи:

- HackTheBox.
- Vulnhub.
- AWS Academy.
- Coursera.

HackTheBox (далі HTB) (рисунок 1.1) – це SaaS (Software as a Service), доступ до якого здійснюється з допомогою веб-сайту hackthebox.org. На цій платформі можна отримати доступ до набору вразливих додатків та віртуальних машин (VM). Кожна VM є унікальною та містить набір вразливостей, які хакер повинен використати аби отримати необхідні привілеї. Доступ до VM здійснюється через вікно веб-браузера (рисунок 1.2) та потребує підключення до мережі інтернет.

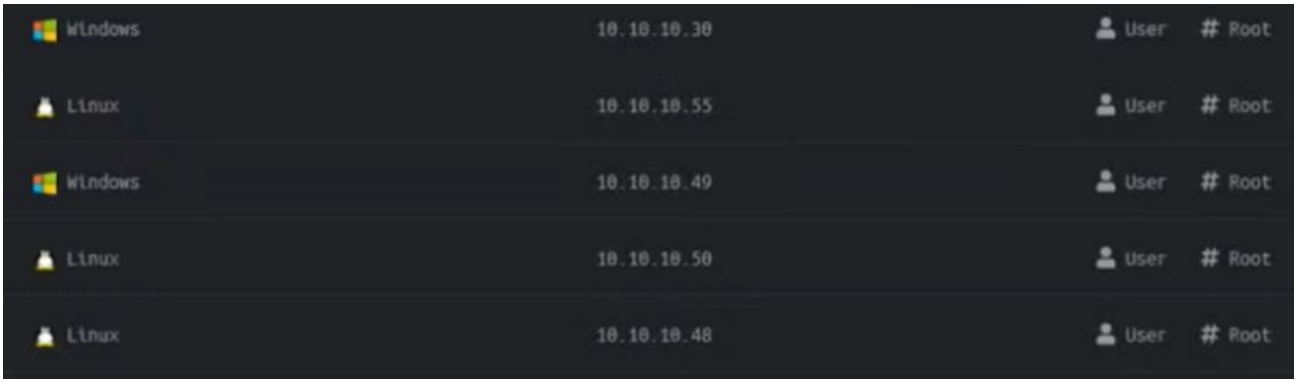

Рисунок 1.1 – Фрагмент сторінки HTB з вибору VM

Перевагами HTB є різноманіття VM [9] та покрокові посібники на випадок, якщо користувач “застряг”. Також розробники платформи регулярно додають нові VM із “найсвіжішими” вразливостями. Значним недоліком є те, що безкоштовна версія HTB надає доступ лише до обмеженої кількості VM, а необмежена кількість VM та покрокові посібники доступні за платною підпискою.

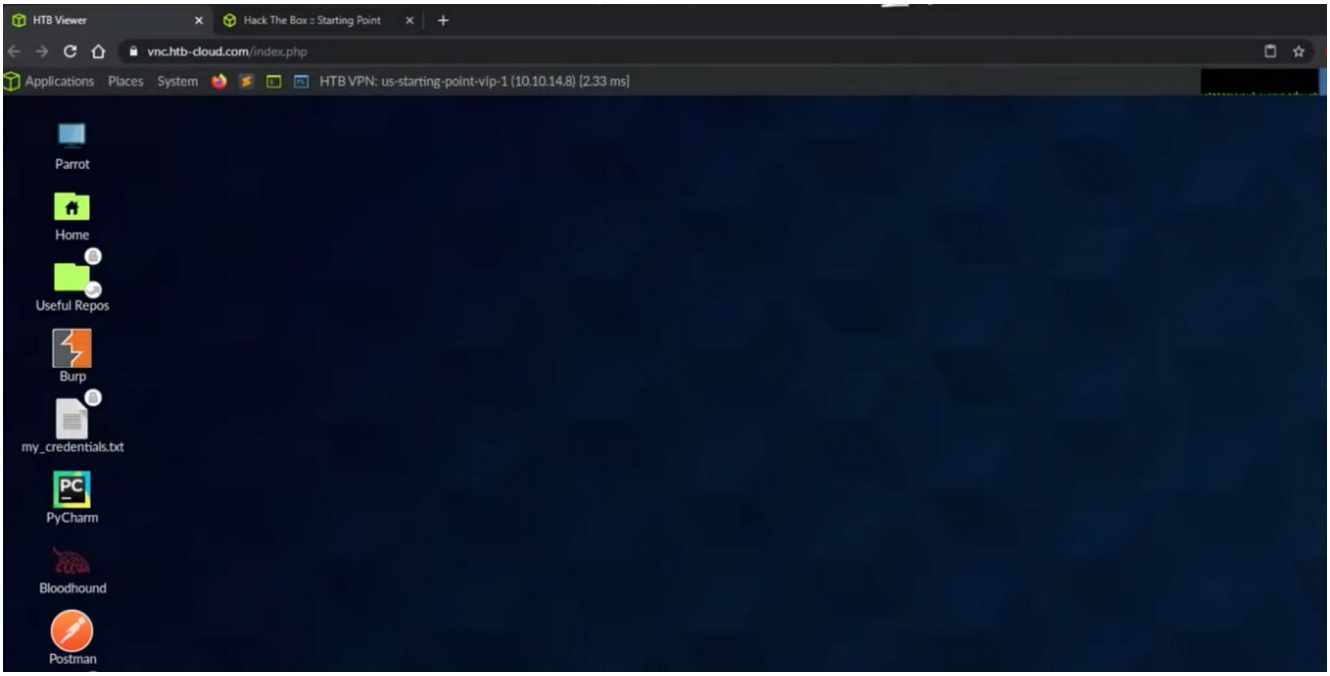

Рисунок 1.2 – Віртуальний робочий стіл HTB у вікні веб-браузера

Vulnhub (рисунок 1.3) – це платформа, що фокусується на кібербезпеці.

Рисунок 1.3 – Частина сторінки Vulnhub з переліком VM

Vulnhub індексує VM зі спеціально закладеними вразливостями [10] з різних джерел та дозволяє завантажити VM для навчання кібербезпеці. Vulnhub користується популярністю завдяки якісним та різноманітним VM на основі реальних додатків, від банківських веб-програм до CTF (Capture The Flag) базового рівня. Одним з його недоліків є те, що користувач мусить самотужки встановлювати скачану VM на власну інфраструктуру або хостинг.

AWS Academy (рисунок 1.4) – це навчальна платформа від Amazon яка зосереджена на вивченні хмарних технологій AWS. Курси на цій платформі слугують до підготовки до сертифікацій AWS [11], наприклад – AWS Certified Cloud Practitioner. Основним недоліком курсів AWS Academy є обмеженість її можливостей: створені VM можуть працювати до 4 годин (є можливість продовження цього терміну знову на 4 години), після завершення цього часу VM знищується – немає можливості постійно вдосконалюватися на попередньо виконаних змінах чи налаштуваннях у VM; також для роботи з AWS Academy потрібне якісне інтернет з’єднання, з VM неможливо працювати оффлайн.

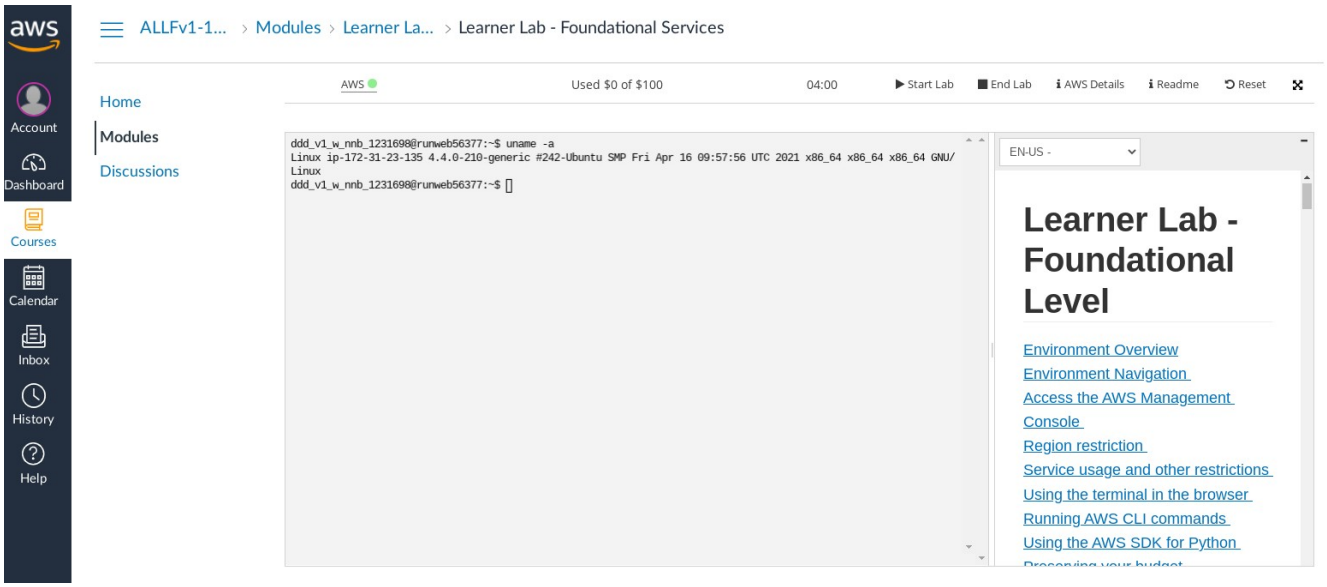

Рисунок 1.4 – Термінальний інтерфейс курсу AWS Academy Learner Lab

Coursera (рисунок 1.5) – це платформа онлайн-курсів, на якій є і курси з вивчення як конкретних відкритих технологій, так і вільних ОС. Платформа надає тільки класичні курси які практично повністю націлені на теорію (лекції та

розділи з теоретичними відомостями (Reading)), а єдине що можна назвати практикою – це тести для самоперевірки (Quiz). Тож “справжню” практику користувач повинен забезпечити собі сам, повторюючи, наприклад, команди надані в теоретичних відомостях курсу (рисунок 1.5) у власному терміналі. Доступу до VM чи VM, як таких, на Coursera не існує. Після успішного завершення курсу користувач може отримати безкоштовний чи придбати платний сертифікат (залежить від курсу) [12].

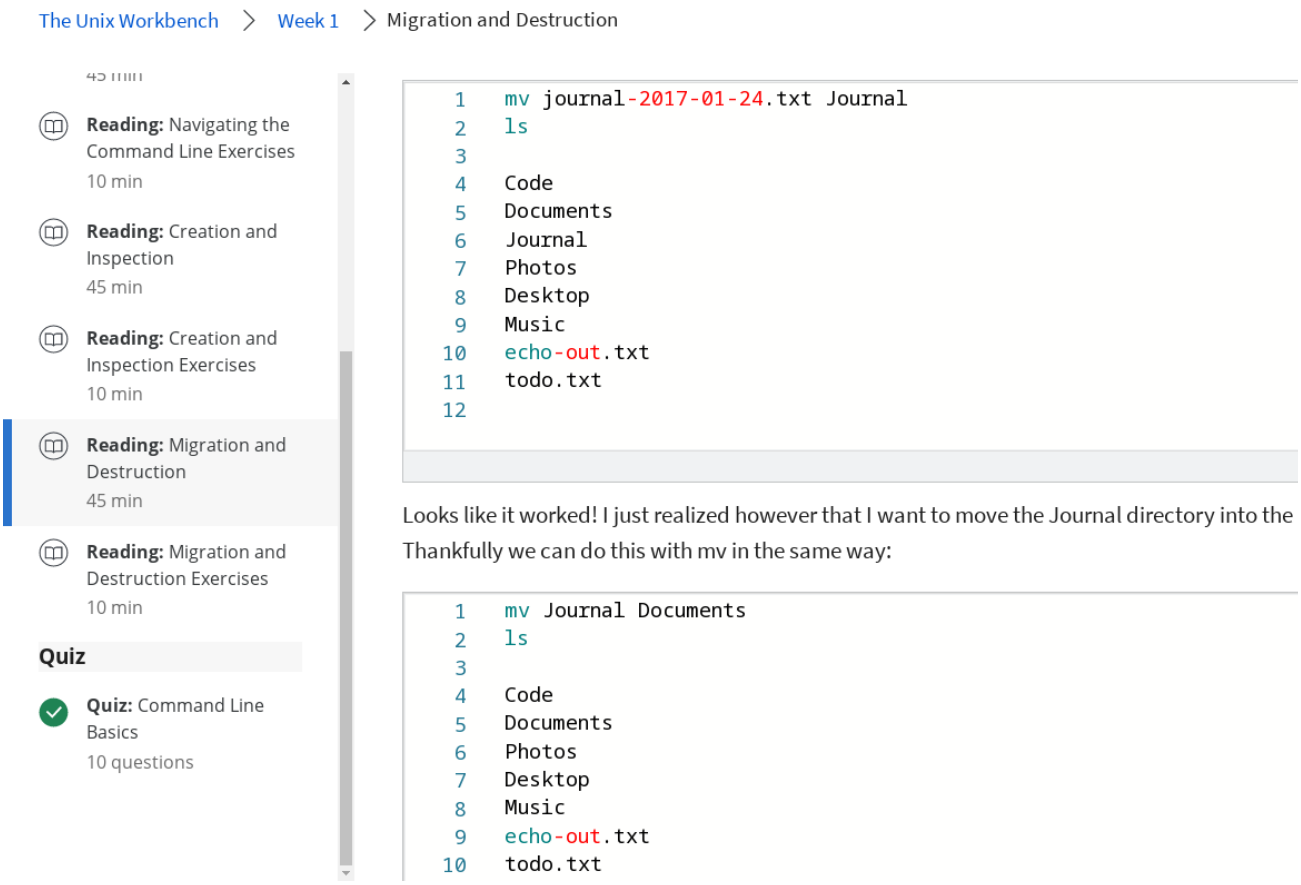

Рисунок 1.5 – Інтерфейс Coursera курсу “Unix Workbench”

Порівняння аналогів за обраними критеріями наведено в таблиці 1.1.

Власну систему вирішено назвати “Click” (“Клац”). В цю назву вкладено легкість з якою вона конфігурується.

Відповідно до загальної оцінки у таблиці порівняльних характеристик розробка власної автоматизованої системи для вивчення відкритих операційних систем є доцільною.

Таблиця 1.1 – Порівняльна характеристика ресурсів вивчення відкритих ОС

Критерії	**HTB**	**Vulnhub VM**	**AWS Academy**	**Coursera**	**Click**
Працює оффлайн	-	+	-	-	+
Можна встановити на власну інфраструктуру	-	+	-	-	+
Адаптується до різних предметних областей	-	-	-	+	+
Можливість SSH доступу	+	-	+	-	+
Дозволяє створювати власні VM	-	-	-	-	+
Загальна оцінка	**20%**	**40%**	**20%**	**20%**	**100%**

Така система візьме до уваги переваги наявних аналогів, а також додасть власні – для більш гнучкого керування навчальним процесом.

1.3 Аналіз методів розв’язання задачі

Розв’язати задачу розробки автоматизованої системи для вивчення відкритих операційних систем на основі хмарних технологій інструментами GNU/LINUX можна декількома способами. Можна написати систему цілком “з нуля”, тобто власноруч реалізувати кожен її модуль – модуль віртуалізації, розгортання, логування і т.д. У цього способу є явні недоліки. Перш за все, він вимагатиме багато часу як для планування і розробки, так і для налагодження та тестування. По-друге, система розроблена таким чином ризикує бути надто тісно зв’язаною [13], через що вона чи її модулі будуть непридатними, без внесення суттєвих змін, для застосування у трохи іншій предметній області.

Найшвидшим і найоптимальнішим є спосіб, в якому будуть поєднуватися вже наявні окремі компоненти (системи віртуалізації та контейнеризації, системи автоматизованого розгортання, системи логування) з допомогою скриптової мови програмування, наприклад, Python. Таким чином, скрипти створені задля поєднання цих систем (підсистем) будуть саме тим місцем з якого вестиметься контроль над станом розробленої системи вивчення ОС, подібно тому, як єдина панель керує різними системами літака (шасі, двигун, склоочисники).

Вибір саме мови програмування Python матиме кілька позитивних наслідків.

Оскільки ця мова є високорівневою та динамічно типізованою, то не потрібно приділяти багато часу розподілу пам’яті та задавати типи змінних; оскільки вона інтерпретована – не потрібно щоразу під час розробки (як з компільованими мовами) чекати на завершення компіляції коду та тільки тоді запускати та перевіряти його роботу. Всі ці речі в рази пришвидшують розробку та налагодження.

Другою вагомою перевагою Python є його популярність – оскільки багато людей ним користується [14], більша кількість людей зможе прочитати код розробленої системи та, можливо, використати його для власних потреб.

1.4 Постановка задач для системи вивчення відкритих ОС

Проаналізувавши наявні переваги та недоліки систем для вивчення відкритих ОС, було сформульовано такі задачі бакалаврської дипломної роботи:

- розробити моделі системи вивчення відкритих ОС;
- розробити метод для перевірки виконання завдань;
- розробити формат для розгортання навчального середовища;
- розробити базу даних для збереження завдань;
- розробити рейтингову таблицю;
- розробити формат навчального середовища (для його реплікації);
- розробити шар сумісності (перетворення форматів) між віртуальними машинами та контейнерами;
- розробити інструкцію користувача;
- провести тестування системи вивчення відкритих ОС.

1.5 Висновки

У першому розділі було розглянуто стан систем вивчення ОС. Було проведено порівняння відомих систем вивчення відкритих ОС, таких як: HackTheBox, Vulnhub, AWS Academy та Coursera.

У результаті порівняння виявлено, що досліджувані системи є

вузькоспеціалізованими SaaS рішеннями, які користуються популярністю у вузьких колах розробників. Провівши аналіз переваг та недоліків наявних систем для вивчення відкритих ОС, було виявлено, що вони не дозволяють користувачеві створювати власні VM чи редагувати наявні курси розгортаючи власні інфраструктурні одиниці (бази даних, системи розподілу нагрузки, модулі машинного навчання і т.п.). Таким чином було доведено доцільність розробки власної системи для вивчення відкритих ОС. Також, на основі проведеного аналізу, сформовано перелік задач, які необхідно виконати для розробки власної системи для вивчення відкритих ОС.

2 РОЗРОБКА МОДЕЛЕЙ СИСТЕМИ ВИВЧЕННЯ ВІДКРИТИХ ОС ТА МЕТОДУ ПЕРЕВІРКИ ВИКОНАННЯ ЗАВДАНЬ

2.1 Розробка моделей системи вивчення відкритих ОС

Система – це комплекс компонентів. Ці компоненти повинні якимось чином взаємодіяти аби система існувала як така.

Як вже було зазначено в попередньому розділі, в ролі системи виступатиме скрипт – “клей”, що зв’яже до купи спільним інтерфейсом різні автономні програми (композиція) і таким чином дозволить досягти нової поведінки, яку ці програми поодинці не здатні проявляти. Це явище називають емерджентністю [15] (emergence, дослівно “поява”). Основною ідеєю під час створення архітектури була ідея композиції та слабкої зв’язності. Композиція вже висвітлена вище, а от слабка зв’язність (loose coupling) означає незалежність [16] компонентів між собою – зміни в одному компоненті (підсистемі) не вплинуть на інші, ба більше, якщо позбутися будь-якого неключового компоненту системи, наприклад, рейтингової таблиці, система й далі працюватиме й матиме сенс, дарма-що з дещо обмеженим функціоналом; якщо ж позбутися ключового компоненту, наприклад, системи розгортання чи системи перевірки завдань – система втратить сенс, бо не зможе виконувати задачі, що роблять її “системою”, втратить власні якості.

Отже, розглянемо яким чином працює система вивчення відкритих ОС “зсередини” та з позицій взаємодії двох типів користувачів з цією системою – студента (будь-кого, хто використовує систему для навчання) та викладача (адміністратора).

На рисунку 2.1 зображена модель системи “зсередини”, вона складається з таких підсистем:

- Скрипт (точка управління системою).
- Ansible – підсистема, з допомогою якої здійснюється конфігурування віртуальних машин чи LXC контейнерів (далі просто LXC).
- PostgreSQL – база даних для збереження завдань та відповідей.
- NGINX – веб-сервер, що видає сторінку із рейтинговою таблицею.

Саму систему можна розгортати як в хмарному середовищі (на рисунку зображено, для прикладу, логотип Amazon Web Services (AWS)), так і на окремому фізичному комп’ютері або декількох комп’ютерах – її підсистеми можуть знаходитись на окремих комп’ютерах, наприклад, Ansible на одному, віртуальні машини на другому, база даних на третьому, веб-сервер на четвертому, скрипт на п’ятому. Для простоти розробки та управління системою, в цій роботі розглядається варіант де вся система разом із підсистемами знаходиться на одному комп’ютері чи хмарному середовищі. З опису та рисунку очевидно, що вибраною була клієнт-серверна архітектура системи, де клієнтами виступає як скрипт по відношенню до своїх підсистем, так і користувачі (див. “система з позиції студента”), що по SSH з’єднуються із власними LXC чи по HTTP з’єднуються з NGINX для перегляду рейтингової таблиці.

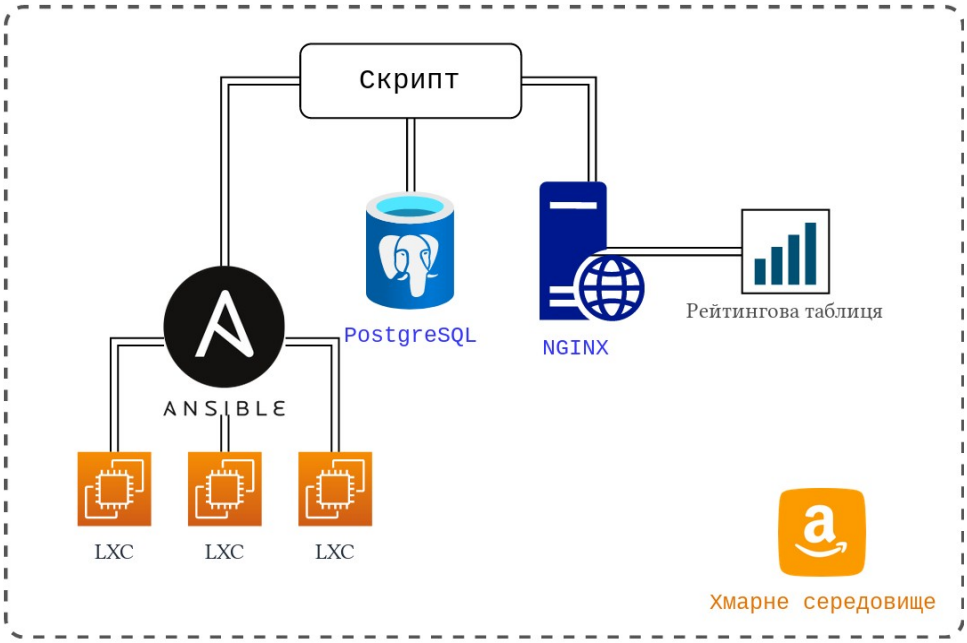

Рисунок 2.1 – Модель системи “зсередини”

З позиції студента модель системи зображено на рисунку 2.2.

Оскільки система для вивчення відкритих ОС є гнучкою, то є можливість перевіряти знання як окремого студента, так і команди, влаштовуючи, наприклад, змагання. Від цього, проте, сутність роботи системи не змінюється.

Студенти (учасники) отримують ключ який дозволяє їм здійснювати

віддалений доступ з допомогою (SSH) до LXC. Потім студенти поодинці або разом виконують певні попередньо розроблені завдання. Система перевірки завдань є частиною скрипту – це підпрограма яка час від часу перевіряє стан LXC студента та реагує на зміни внесені студентом. Якщо завдання виконано успішно, це відобразиться на табло, яке розміщується на веб-сайті який NGINX видає (рисунок 2.3).

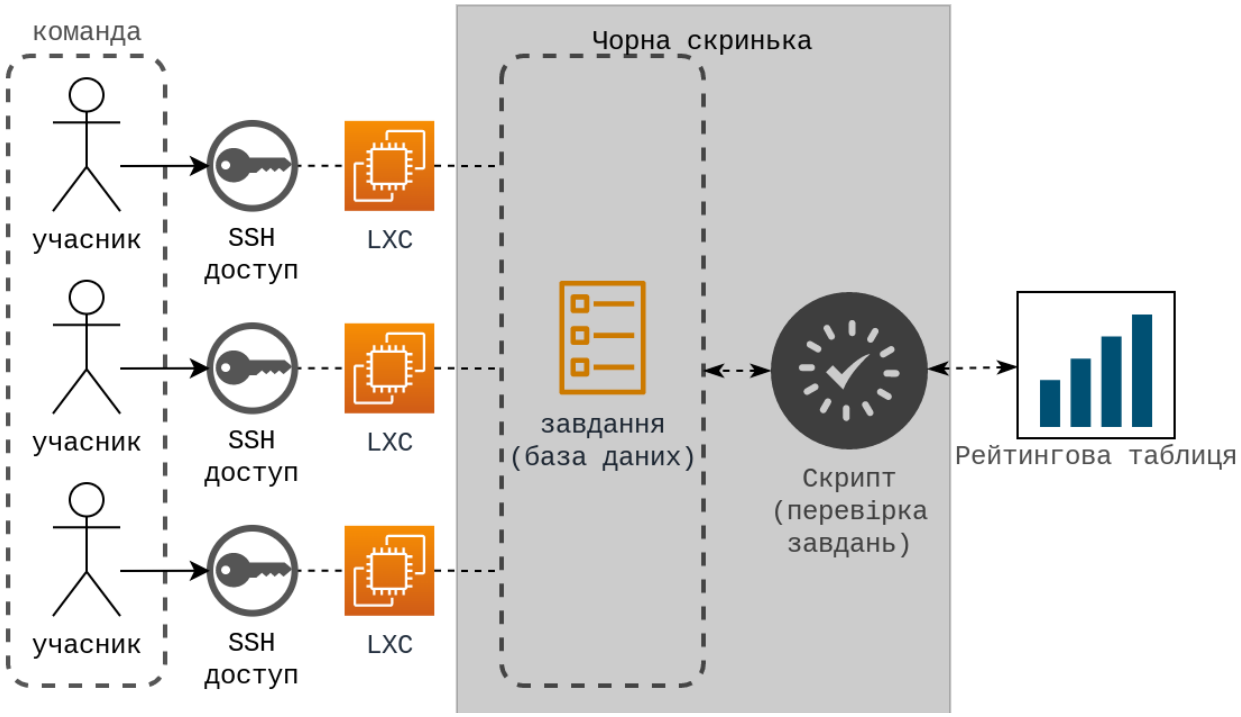

Рисунок 2.2 – Модель системи з позиції студента

П.І.Б.	Країна	Назва НЗ	Уч.Ф.Linuxation	Onl_Linuxation	Бали Linuxation-Фінал	Місце Linuxation-Фінал
Ісаков Андрій Васильович	Україна	Вінницький національний технічний університет	1	70	45	1
Лавров Вадим Валерійович	Україна	Вінницький національний технічний університет	1	70	40	2
Буняк Віталій Михайлович	Україна	Вінницький національний технічний університет	1	70	30	3

Рисунок 2.3 – Приклад табло з результатами змагань

З боку викладача існує набір текстових файлів конфігурації, змінюючи які він може досягнути того чи іншого стану системи. Завдання для системи також спершу повинні бути у вигляді текстового файлу, який потім скрипт передасть в базу даних, для швидкого доступу до них.

Все інше конфігурування викладач здійснює змінюючи файли конфігурації відповідних підсистем (рисунок 2.4).

```
---
- hosts: localhost
  connection: local
  become: true
  vars:
  - interface: lxcbr0

  tasks:
  - name: apt lxc packages are installed on host
    apt:
      name:
      - tmux
      - mdp
      - lxc
      - lxc-dev
      - python3-pip
      state: present

  - copy:
      dest: /etc/default/lxc-net
      content: |
        USE_LXC_BRIDGE="true"

  - copy:
      dest: /etc/lxc/default.conf
      content: |
        lxc.net.0.type = veth
        lxc.net.0.link = {{ interface }}
        lxc.net.0.flags = up
        lxc.net.0.hwaddr = 00:16:3e:xx:xx:xx

  - service:
      name: lxc-net
      state: started

  - name: pip lxc packages are installed on host
    pip:
      name:
      - ansible-lint
      - yamllint
    run_once: true

- hosts: all
```

Рисунок 2.4 – Приклад файлу конфігурації системи розгортання

Викладачеві доведеться навчитись користуватись системою та писати файли конфігурацій, що складніше ніж просто використовувати GUI (графічний інтерфейс користувача). Проте після цього він зможе пристосувати її до найвитонченіших потреб, оскільки файли конфігурацій дають змогу контролювати кожну деталь цієї системи.

2.2 Розробка формату зберігання завдань

Джерелом натхнення для створення формату завдань стала змагальна платформа зі спортивного програмування Codeforces. Кожне завдання має вхідні та вихідні дані в певному попередньо зазначеному вигляді, обмеження по часу виконання програми та рівень складності. Той студент, чия програма виконалась швидше посідає вище місце в рейтинговій таблиці. Ці основні ідеї були перейняті для розробки системи вивчення відкритих ОС з єдиною відмінністю – оскільки система вивчає саме відкриті ОС, а не спортивне програмування, час виконання програми було замінено часом виконання завдання студентом.

Для створення формату завдань необхідно відповісти на питання – “В якому форматі повинні бути завдання, текстовому (plain text) чи двійковому (binary)?”. Вибір було зроблено на користь текстового формату, оскільки він найкраще підходить для редагування людиною (вчителем), його легко читати та налагоджувати. Існують такі популярні текстові формати зберігання даних – JSON, YAML, TOML, XML. З них було обрано саме JSON, тому що по-перше, більшість баз даних (як SQL, так і NoSQL) підтримують JSON, що дозволяє вибирати між широким спектром баз даних [5]; по-друге, тому що JSON не настільки багатослівний (verbose) як XML, а в якості роздільних знаків використовуються дужки, коми, інші писані символи, що важливо для простоти читання та усунення помилок, на відміну від пробілів та символів табуляції (tab), неписаних символів, у форматах YAML та TOML. Описана вище специфікація формату є рекомендацією та не є примусовою – завдяки формату JSON викладач може додавати та перевіряти ті атрибути які він вважає за потрібне. Приклад завдання, відформатованого з допомогою JSON, зображено на рисунку 2.5.

```
{
  "solution": "22/tcp open ssh\n80/tcp open http\n139/tcp open netbios-ssn\n445/tcp open microsoft-ds",
  "check_command": "nmap localhost | awk '/open/{print $1 \" \" $2 \" \" $3}'"
}
```


Рисунок 2.5 – Приклад завдання для системи вивчення відкритих ОС в форматі JSON

Варто зауважити, що символ лапок у JSON має особливе значення і тому перед всіма лапками всередині інших лапок необхідно ставити символ зворотної скісної риски (“\”).

2.3 Розробка методу перевірки виконання завдань

Вивчення ОС зазвичай полягає в тому, щоб виконати якусь послідовність команд або ж змінити стан LXC якимось чином. Для того, щоб це перевірити, існують такі способи:

- Перевірка вмісту файлів в які студент мав записати правильні відповіді.
- Перевірка стану LXC студента з допомогою команд, наприклад, перевірка відкритих портів з допомогою nmap – отримавши цю інформацію можна перевірити чи піднятий, наприклад, веб-сервер на LXC студента та чи приймає він підключення).
- Перевірка історії термінальних команд, що їх виконував студент (файлу .bash_history) або логів системи (dmesg, journalctl).

Метод перевірки завдань реалізується послідовністю дій (рисунок 2.6):

1. Скрипт віддає задачу підсистемі конфігурування перевірити стан конкретного LXC.
2. Підсистема конфігурування з допомогою SSH доступу входить до LXC.
3. Читає файл або виконує команду.
4. Перевіряє чи отримані дані збігаються з правильною відповіддю (при цьому беручи до уваги, що на LXC можуть бути інше ім’я користувача, шлях до його домашньої директорії і т.п.).
5. Записує відповідь студента, навіть якщо вона є неправильною до бази даних в таблицю з відповідями студентів, що необхідно, наприклад, для

випадку, коли студент оскаржує рішення системи і вимагає “ручної” перевірки.

6. Записує бали у випадку правильного виконання завдання.
7. Створює файл логування (log file) виконання перевірки.

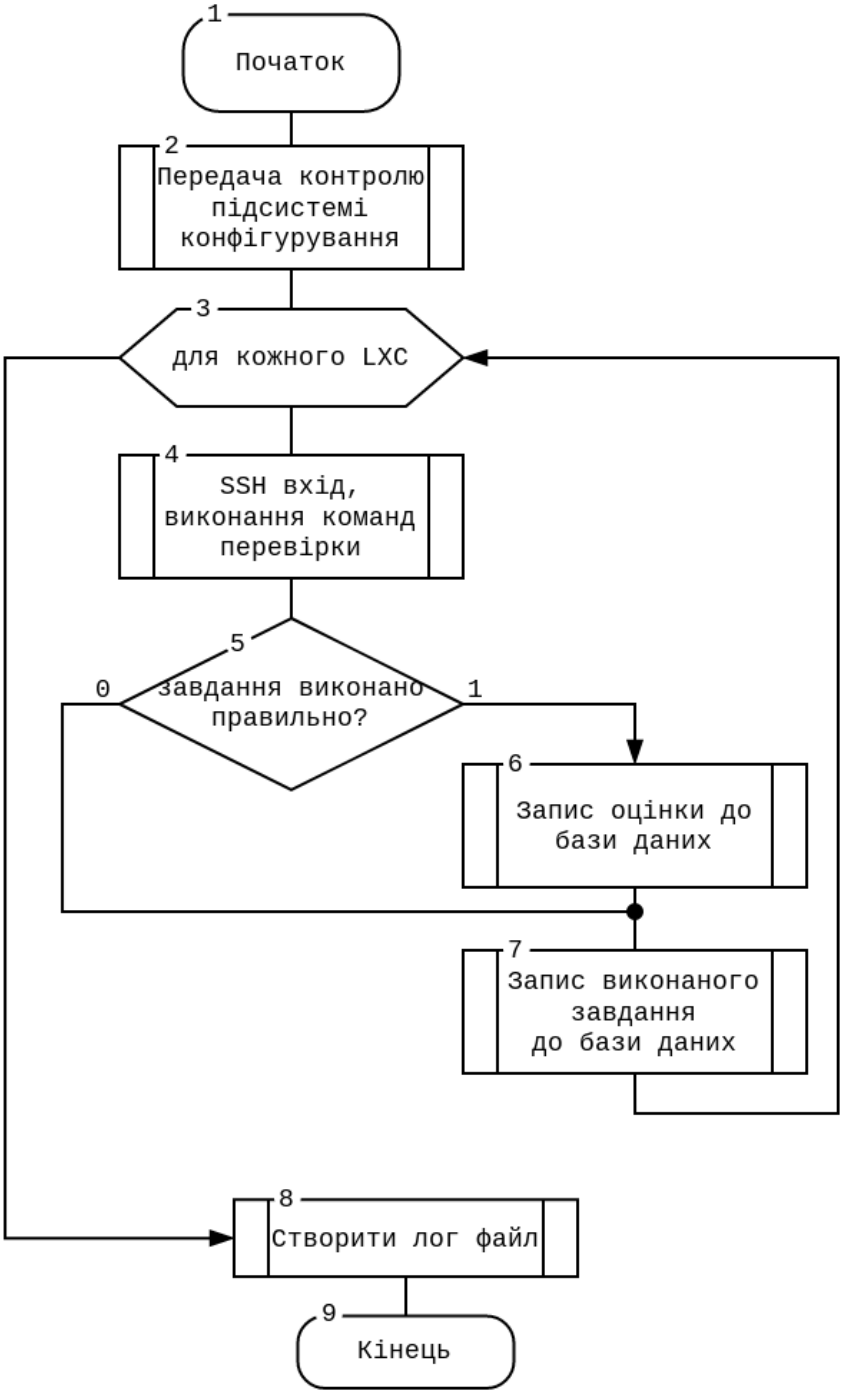

Рисунок 2.6 – Алгоритм перевірки виконання завдань

Таким чином, на блок-схемі було візуалізовано алгоритм перевірки завдань.

2.4 Обґрунтування вибору хмарного середовища для розгортання системи вивчення відкритих ОС

Основною перевагою використання хмарних сервісів з точки зору користувача є модель оплати ресурсів в міру їх використання, завдяки чому капітальні витрати перетворюються на поточні витрати [17]. Користувачеві (адміністратору) не потрібно наперед купувати апаратне забезпечення, опікуватись його обслуговуванням, купувати нове устаткування (якщо не вистачає потужності наявного) і т.д. Всі ці класичні проблеми утримання власної інфраструктури вирішується з допомогою моделі надання обчислень як послуги. Постачальник таких послуг опікується апаратурою, а користувач абстрагується з допомогою цієї моделі та користується програмним забезпеченням.

Система для вивчення відкритих ОС може працювати як на власному сервері, так і на хмарній платформі. Якщо, наприклад, навчальний заклад має власну приватну хмару, то зручно буде розгорнути систему саме там; або ж можна використовувати систему у якомусь одному комп’ютерному класі локально та без необхідності доступу до Інтернет.

Публічне хмарне середовище допоможе тим в кого не надто потужний комп’ютер почати користуватися системою, тому нижче обрано найоптимальніше хмарне середовище для розгортання системи.

Нині найпопулярнішими хмарними сервісами [18] є AWS, Microsoft Azure та Google Cloud. Всі вони пропонують дуже різні тарифні плани, які, наприклад, залежать від того є користувач студентом чи звичайним користувачем (різні типи підписок).

Google Cloud та Microsoft Azure схожі тим, що надають безкоштовні віртуальні гроші ($200) для використання на власних хмарних платформах. Після закінчення терміну дії цих безкоштовних грошей (на обох платформах – один місяць), користувача переводять на платну підписку. Кількість сервісів, які можна використовувати користуючись віртуальними грошима є обмеженою.

Для навчальних установ, особливо корисною є платформа на базі хмарного середовища AWS – AWS Academy, оскільки вона надає $100 [19] для кожного

студента у кожному курсі створеному на платформі; один студент може мати, наприклад, $80 на курсі по машинному навчанню на AWS Academy та $100 на курсі по мережевих технологіях) і ці кошти автоматично відновлюються кожні 12 місяців. Це означає, що послуги AWS безкоштовні, якщо не перевищувати ліміту.

AWS Academy надає середовище Learner Lab (рисунок 2.7) в якому можна розгорнути систему вивчення відкритих ОС. В цьому середовищі можна користуватися всіма основними сервісами AWS за винятком IAM (політика доступу та дозволи), які необхідні коли AWS використовують для комерційних проектів чи організацій.

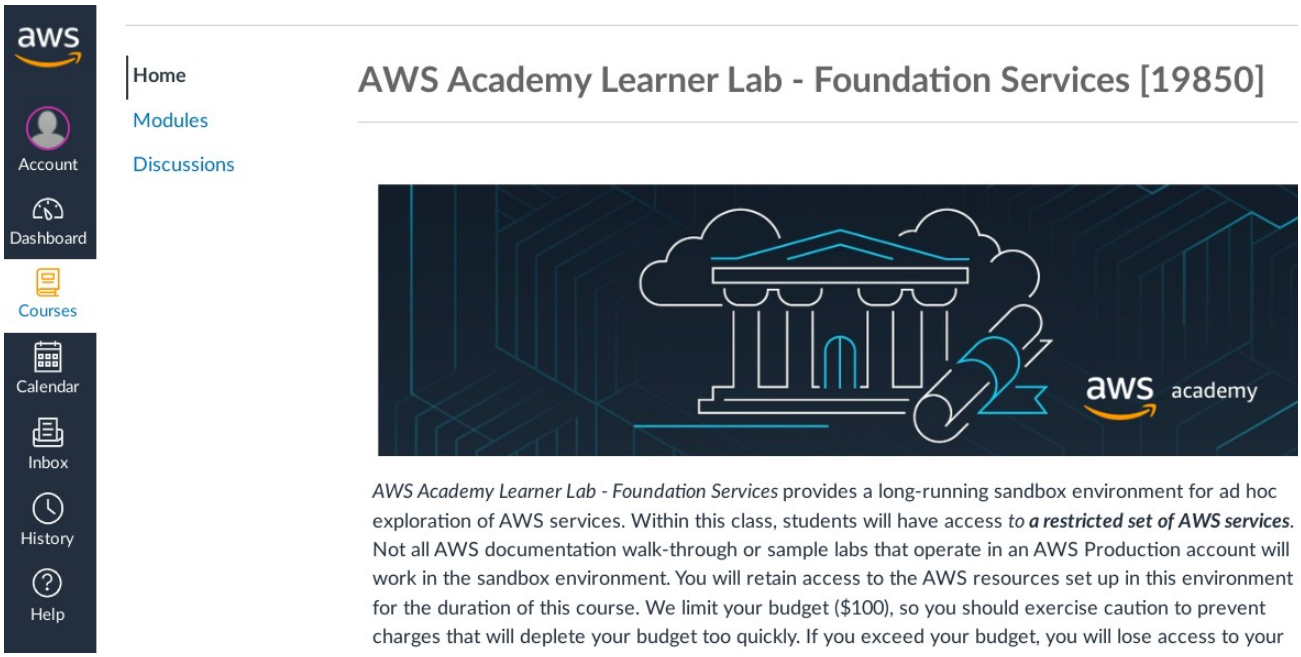

Рисунок 2.7 – AWS Academy Learner Lab

З фінансової точки зору за одні змагання (інтенсивне використання хмарних ресурсів) тривалістю близько двох годин, може бути використано в середньому $2, тобто за рік можна провести близько 50 змагань, майже одне змагання щотижня. Враховуючи, що AWS Academy дозволяє створювати декілька середовищ Learner Lab, одне середовище можна використовувати для змагань, а інше для звичайних занять, таким чином отримуючи користь без необхідності платити гроші.

Отже, найоптимальнішим хмарним середовищем для навчальних установ, якщо в них немає власної приватної хмари, є середовище AWS Academy.

2.5 Висновки

У другому розділі було розроблено та розглянуто моделі системи вивчення відкритих ОС з боку самої системи, а також з боку студента та викладача. Розроблено формат, в якому зберігатимуться завдання, та розроблено метод перевірки їх виконання. Обґрунтовано вибір хмарного середовища AWS Academy для розгортання системи. Також було використано численні рисунки для полегшення розуміння описового матеріалу.

3 ПРОГРАМНА РОЗРОБКА СИСТЕМИ ВИВЧЕННЯ ВІДКРИТИХ ОС

3.1 Варіантний аналіз і обґрунтування вибору інструментів розробки та методів їх зв’язування між собою

Щоб побудувати систему, яка б існувала та розвивалась тривалий час, важливо вибрати компоненти (підсистеми), які протягом цього часу будуть самі вдосконалюватися та не застаріють. Тривалість існування, а також використання, розвитку та підтримки системи визначатиметься мінімальною тривалістю існування її компонентів, у разі якщо їх неможливо замінити без внесення суттєвих змін до системи. Також ці компоненти повинні бути достатньо стабільними, щоб у разі виходу для них головних оновлень систему не потрібно було переписувати заново. Важливим є і вибір зручних інструментів розробки системи. Оскільки майбутнє за вільним програмним забезпеченням, всі компоненти системи, сама система й інструменти використані для її розробки є вільним програмним забезпеченням.

В основі системи скрипт на платформонезалежній скриптовій мові програмування Python. Python підтримується всіма популярними відкритими та власницькими операційними системами [20], що дозволяє виконувати скрипт і розгортати систему на них (за умови доступності інших компонентів). Для Python розроблена велика кількість API які дозволяють пов’язувати різні системи та бібліотеки між собою, ця особливість є ключовою перевагою Python, щоб використовувати його для цієї предметної області.

Ansible – це програмне забезпечення для автоматизованого розгортання програм, керування конфігураціями та оркестровки, виконання рутинних адміністративних завдань на групі систем. Ansible написана мовою Python і розповсюджується під ліцензією GPLv3. Основними перевагами Ansible над іншими подібними програмами (Puppet, Chef, Salt Stack) є те, що їх не треба встановлювати на кожний сервер [21] , конфігурування якого здійснюється – вона використовує звичайне SSH з’єднання для комунікації та виконує ті команди на кінцевому сервері, що очікує його ОС. Також, оскільки Ansible написана на Python

набагато легше розробляти спеціальні модулі для неї, та не потрібно встановлювати інтерпретатори чи компілятори інших мов – Chef та Puppet написані на Ruby. Ansible використовує декларативну мову конфігурацій, тобто замість того, щоб описувати як досягти певної конфігурації (імперативний підхід), мова конфігурацій Ansible задає стан, якого треба досягти, і вже сама Ansible вирішує яким чином це зробити.

Для створення віртуальних навчальних середовищ для студентів є три шляхи – використання фізичних комп’ютерів (bare metal), використання віртуалізації та використання контейнеризації.

Використання фізичних комп’ютерів може зумовлювати значні витрати на інфраструктуру. Використання віртуалізації суттєво зменшує можливості масштабування системи, оскільки кожна віртуальна машина віртуалізує апаратні компоненти, для чого витрачаються додаткові ресурси. Тому було обрано контейнеризацію для використання в системі вивчення відкритих ОС. Існують дві популярні технології контейнеризації – LXC (Linux Containers) та Docker. Відмінності між ними зображені на рисунку 3.1.

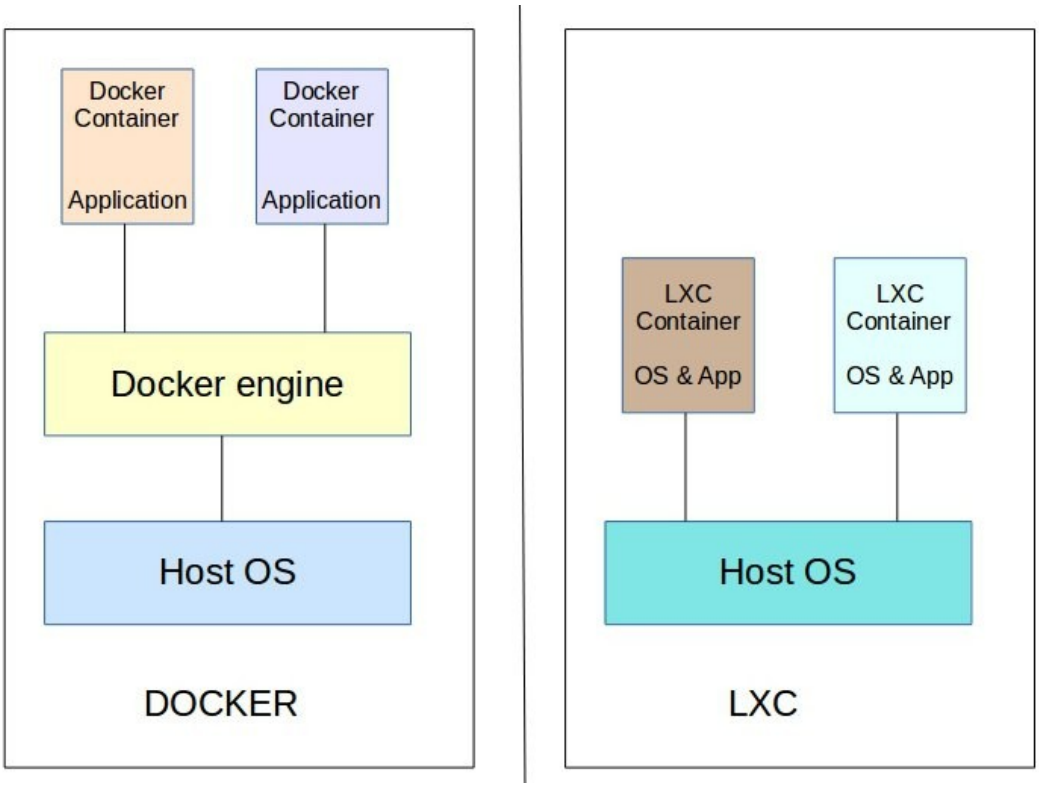

Рисунок 3.1 – Відмінності в роботі LXC та Docker

Docker в основному використовують для контейнеризації окремих програм (через Docker Engine), в той час як LXC контейнеризує цілий дистрибутив Linux (все програмне забезпечення) окрім ядра, яке є спільним (через ізоляцію процесів ядра – namespaces [22]). Тому обрано саме LXC для створення віртуальних навчальних середовищ.

Розв’язки та завдання для змагань та навчання, а також дані студентів вирішено зберігати в реляційній базі даних PostgreSQL. Як і інші бази даних, PostgreSQL дозволяє зберігати JSON (у PostgreSQL завдяки двом типам – JSON та JSONB), але, на відміну від MariaDB та MySQL, PostgreSQL підтримує більшу кількість [23] SQL операцій (lateral joins, grouping sets, WITH RECURSIVE clause, FILTER clause, TABLESAMPLE clause та інші). Також PostgreSQL надає інтерфейс (API) для Python – psycopg2, який можна використовувати напряму, або через ORM SQLAlchemy.

Для розробки веб-інтерфейсу із рейтинговою таблицею використано невід’ємні для будь-якого веб-сайту HTML/CSS та JavaScript, а також мікровебфреймворк Python Flask. Для запуску веб-сайту використано веб-сервер NGINX, таблицю розроблено з допомогою фреймворка Tabulator.

Під час розробки були використані виключно вільні та відкриті програми, ОС GNU/Linux, сама розробка велась в GNU Emacs та термінальному текстовому редакторі VIM. Всі програми використані для розробки системи активно підтримуються спільнотою розробників та є достатньо популярними й перевіреними часом, щоб стверджувати, що за найближчі п’ять років вони не застаріють та не перестануть існувати, а отже і система існуватиме.

3.2 Розробка бази даних

База даних є одним із невід’ємних компонентів системи вивчення відкритих ОС. В ній зберігаються завдання, відомості про студентів та журнал розв’язків наданих студентами.

Спершу, перед тим як писати SQL запити та створювати таблиці, варто візуалізувати та описати ідею (підхід зверху-вниз, top-down approach), щоб

зрозуміти як будуть пов’язані сутності між собою.

Для системи вивчення відкритих ОС було виділено три сутності та їх атрибути:

1. Task – Завдання:
    - id_task – унікальний ідентифікатор;
    - name – назва завдання;
    - description – словесний опис завдання;
    - rating – рівень складності завдання;
    - creation_date – дата створення завдання;
    - solution – рішення завдання в JSON форматі.
2. Student – Студент:
    - id_student – унікальний ідентифікатор;
    - name – ім’я студента або нікнейм;
    - email – електронна пошта студента;
    - ssh_key – SSH ключ студента для доступу до контейнера.
3. Journal – Журнал:
    - id_journal – унікальний ідентифікатор;
    - id_task – зовнішній ключ, що вказує на сутність із таблиці Завдання;
    - id_student – зовнішній ключ, що вказує на сутність із таблиці Студент;
    - student_solution – рішення завдання в JSON форматі як було виконано студентом;
    - date_submitted – дата виконання завдання;
    - grade – оцінка за завдання.

Зв’язок між сутностями зображено на ER-моделі на рисунку 3.2.

ER-модель (з англ. “Entity-relationship model” – модель сутність-зв’язок) – це семантична модель даних, яка призначена для спрощення процесу проектування бази даних [24]. Зв’язок між сутністю Студент та Журнал – один-до-багатьох (1:n), оскільки один студент може виконувати одне завдання декілька разів чи виконувати різні завдання; Журнал містить записи кожної спроби студента, вдалої чи невдалої. Зв’язок між сутністю Завдання та Журнал – один-до-багатьох (1:n),

оскільки одне завдання можуть виконати декілька студентів.

Таким чином сутність Журнал робить декомпозицію зв’язку багато-до-багатьох між сутностями Студент та Завдання (один студент робить багато завдань та одне завдання може бути виконане багатьма студентами).

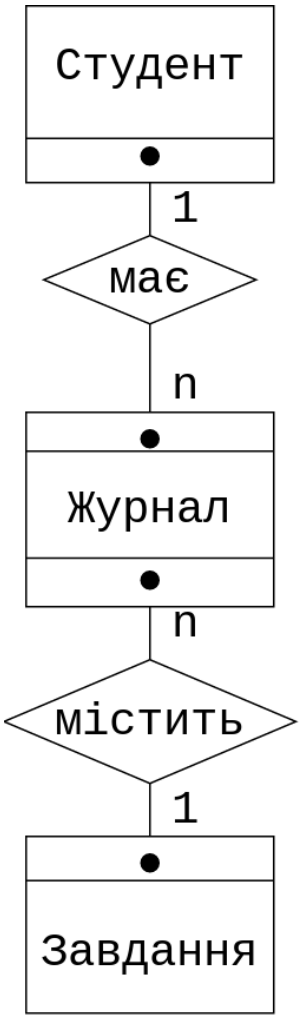

Рисунок 3.2 – ER-модель для розробки бази даних для системи вивчення відкритих ОС

Варто зазначити, що в PostgreSQL, починаючи з версії v9.2, було додано підтримку зберігання JSON формату, що до цього зазвичай використовувався NoSQL (нереляційними) базами даних. PostgreSQL, проте, залишається SQL базою даних, бо єдиний спосіб отримувати доступ до елементів PostgreSQL – це мова SQL. Базу даних розроблено використовуючи термінальний інтерфейс PostgreSQL “psql” та веб-інтерфейс “pgAdmin4”.

На рисунку 3.3 продемонстровано створення бази даних “clickDB”. Оскільки PostgreSQL – це система керування базами даних (СКБД), вона може керувати декількома окремими базами даних із власними таблицями, повністю ізольованими одна від одної.

```
CREATE DATABASE "clickDB"
    WITH
    OWNER = admin
    ENCODING = 'UTF8'
    LC_COLLATE = 'uk_UA.UTF-8'
    LC_CTYPE = 'uk_UA.UTF-8'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1;

    DROP TABLE IF EXISTS student CASCADE;
DROP TABLE IF EXISTS task CASCADE;
DROP TABLE IF EXISTS journal;
```

Рисунок 3.3 – SQL-код для створення бази даних “clickDB”

На рисунку 3.4 надано код створення таблиць task, student та journal.

```
CREATE TABLE task (
    id_task         int PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    name            varchar(80),
    description     varchar(800),
    rating          int,
    creation_date   date,
    solution        jsonb
);

CREATE TABLE student (
    id_student      int PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    name            varchar(80),
    email           varchar(100),
    ssh_key         varchar(1000)
);

CREATE TABLE journal (
    id_journal          int PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    id_task             int references task(id_task),
    id_student          int references student(id_student),
    student_solution    jsonb NOT NULL,
    date_submitted      date,
    grade               int
);
```

Рисунок 3.4 – SQL-код для створення таблиць task, student та journal для “clickDB”

На рисунку 3.5 зображено чотири SQL-запити, які демонструють правильність запису до створених таблиць.

```
INSERT INTO task (name, description, rating, creation_date, solution)
    VALUES ('tricky nmap',
'Try to set up http and smb servers. Output nmap output to prove which ports are open.
Output format: <port>/<connection_type> open <service_type>',
2,
'2022-05-15',
'{"nmap":"22/tcp open ssh\n80/tcp open http\n139/tcp open netbios-ssn\n445/tcp open microsoft-ds"}'
);

INSERT INTO student (name, email, ssh_key)
    VALUES ('Andrew', 'andrew@example.com',
'ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCgcPOpb1zRgzGN7NJ0NryWWztkrE954/HSpQnkHoWyMmcnJ3KvDHAb14KWVhd
);

INSERT INTO student (name, email, ssh_key)
    VALUES ('Max', 'max@example.com',
'ssh-rsa Meg93G/CmcnJryWWztkrE954HSpQnkHoWyM1yc2EAAAADAQABAAABgQCgcPOpb1zRgzGN7NJ0NIzTdNaP9HFzxxolPZ
);

INSERT INTO journal (id_task, id_student, student_solution, date_submitted, grade)
    VALUES (1, 1,
'{"nmap":"22/tcp open ssh\n80/tcp open http\n139/tcp open netbios-ssn\n445/tcp open microsoft-ds"}',
'2022-05-16',
10
);
```

Рисунок 3.5 – SQL-код для створення записів в таблицях task, student та journal

Щоб продемонструвати правильність отримання даних зі створених таблиць, зображено запит на рисунку 3.6, що виконує операцію лівого зовнішнього з’єднання (left outer join) – видає рядки з Журнал об’єднані з тільки тими рядками Студент та Завдання, на які посилаються рядки в Журнал з допомогою зовнішнього ключа (рисунок 3.7).

```
SELECT * FROM journal, student, task
WHERE journal.id_student = student.id_student AND
journal.id_task = task.id_task
ORDER BY id_journal ASC
```

Рисунок 3.6 – SQL-код запиту лівого зовнішнього з’єднання з таблиці journal

rnal r	id_task integer	id_student integer	student_solution jsonb	date_submitted date	grade integer	id_student integer	name character varying (80)	email character varyin
1	1	1	{"nmap": "22/tcp …	2022-05-16	10	1	Andrew	andrew@examp

Рисунок 3.7 – Результат виконання запиту з таблиці journal

У результаті створення таблиць, з допомогою інтерфейсу pgAdmin4 було візуалізовано схему даних (рисунок 3.8).

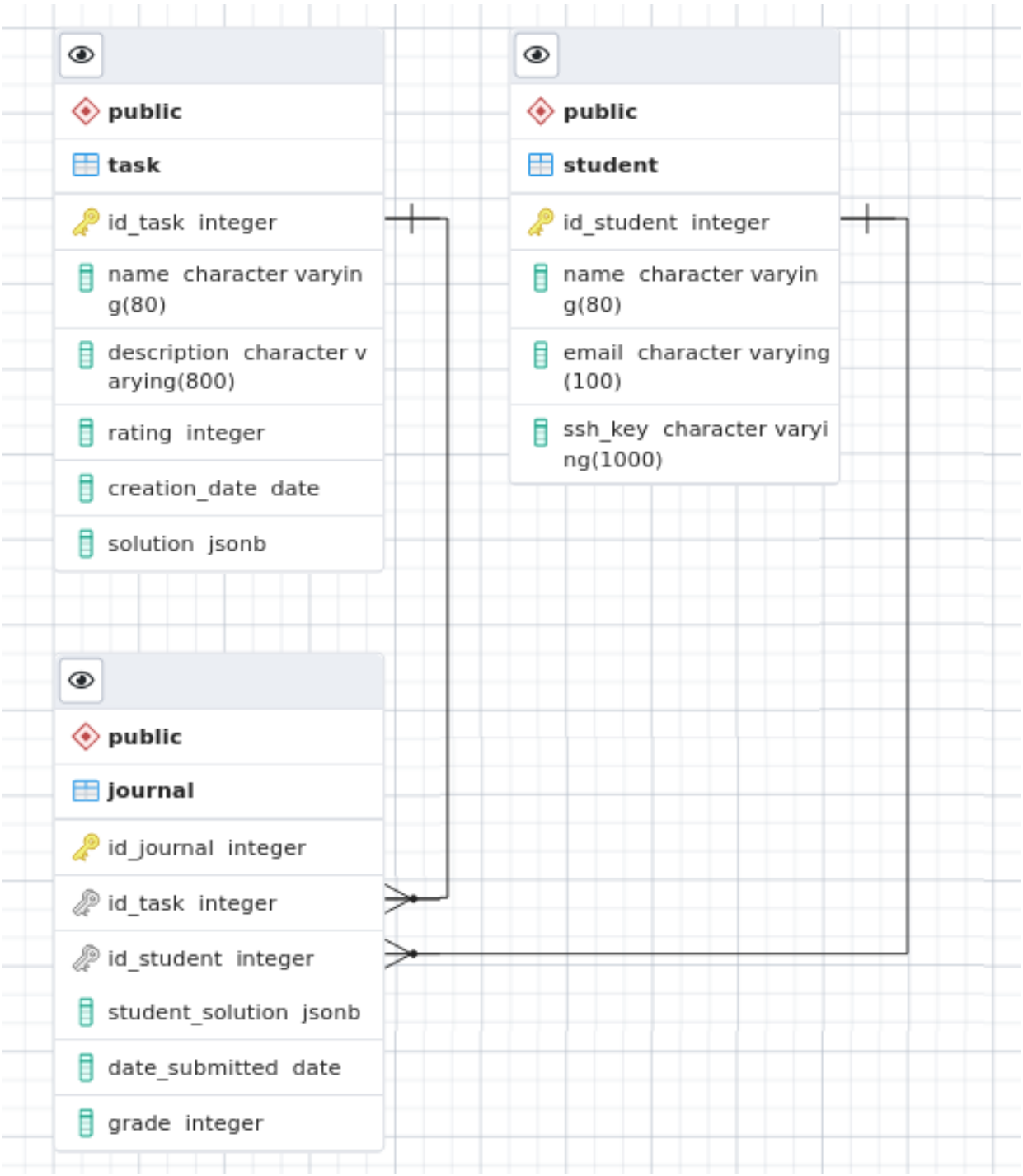

Рисунок 3.8 – Схема даних системи вивчення відкритих ОС

На схемі даних відображені атрибути сутностей, позначені головні та зовнішні ключі (жовтим та сірими ключиками відповідно), а також зв’язок між сутностями (один-до-багатьох).

3.3 Розробка рейтингової таблиці

Рейтингова таблиця відображає успіхи студентів (рисунок 3.9). До неї включено імена студентів та кількість балів яку ці студенти заробили виконуючи завдання. Відповідно до зароблених балів, якщо, наприклад, систему використовували для проведення змагань, можна створити стовпчик з місцем, яке посів студент.

💪 Рейтингова таблиця 👌

Студент	Всього набрано балів
Ада Гупало	51
Веніямин Масоха	48
Дем'ян Стельмах	47
пан Григорій Бабій	44
Федір Шутько	30
Роксолана Гайденко	29
Давид Дробаха	22
пані Мар'яна Даниленко	22

Рисунок 3.9 – Табло з результатами змагань

Для створення веб-сайту цілком використано веб-фреймворк Flask. На рисунку 3.10 наведено Python код серверної частини (backend), який отримує дані для таблиці з PostgreSQL.

```
107 @app.route("/")
108 @app.route("/home")
109 @app.route("/index")
110 def home():
111 #    q = db.session.query(Student.name, Journal.id_journal).all()
112     plain_sql=text("""
113         select student.name, sum(journal.grade) as "sum"
114         from student join journal
115         on student.id_student = journal.id_student
116         group by student.name order by student.name;
117     """)
118     with db.engine.connect() as connection:
119         query = connection.execute(plain_sql)
120         data=[{'name':i[0], 'grade':i[1]} for i in query]
121         return render_template('index.html', data=data)
```

Рисунок 3.10 – Фрагмент коду, що заповнює таблицю виконавши SQL запит

Для реалізації рейтингової таблиці використано JavaScript фреймворк для створення інтерактивних веб-таблиць Tabulator [25]. На рисунку 3.11 наведений код клієнтської частини (frontend).

```
<!DOCTYPE html>
<html lang="en">

  <head>
    <title>Рейтингова таблиця</title>
    <meta charset="utf-8"/>
    <link rel="shortcut icon" href="/favicon.ico" type="image/x-icon" />
    <!-- <link rel='stylesheet' type='text/css' href='/style.css'> -->
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="keywords" content="">
    <link href="../static/css/tabulator.css" rel="stylesheet">
  </head>
  <body>
      <header><h1>🏆 Рейтингова таблиця 🏆</h1></header>
      <div id="example-table"></div>
      <style>
        h1 {
            text-align: center ;
        }
      </style>
      <!-- This thing here is needed to sort column etries with the dates -->
      <script type="text/javascript" src="../static/js/luxon.js"></script>
      <script type="text/javascript" src="../static/js/tabulator.js"></script>
      <script type="text/javascript">
        //sample data
        var tabledata = {{ data | tojson }};

        var table = new Tabulator("#example-table", {
            //height:200, // set height of table to enable virtual DOM
            // better to set hight depending on device's screensize?
            // or not set it and let the user scroll...
            data:tabledata, //load initial data into table
            layout:"fitColumns", //fit columns to width of table (optional)
            columns:[ //Define Table Columns
                {title:"Студент", field:"name", sorter:"string"},//, width:150},
                {title:"Всього набрано балів", field:"grade", sorter:"integer",
                 hozAlign:"left"},
            ],
        });
      </script>
  </body>
</html>
```

Рисунок 3.11 – Фрагмент коду клієнтської частини

Таким чином було розроблено інформаційний інтерфейс, по якому студенти можуть зорієнтуватись як вони справляються із завданнями. Також таблиця дозволяє сортувати рядки за іменами студентів чи їх балами за зростанням та спаданням.

3.4 Розробка скрипта сумісності для перетворення образів віртуальних машин

Для розширення можливостей системи вивчення відкритих ОС, вирішено зробити окремий скрипт сумісності здатний конвертувати образи завантажені з Vulnhub (формат OVA, Open Virtualization Appliance) у формат з яким працює LXC (rootfs). OVA це фактично gzip архів з файлами формату віртуального диску VMDK та OVF (Open Virtualization Format) – XML файлу з деталями апаратної конфігурації віртуальної машини (в ньому прописані віртуальні мережеві адаптери, кількість зарезервованої оперативної пам’яті, SATA контролери і т.д.). Оскільки готового рішення, яке б перетворювало OVA в rootfs, не існує, процес перетворення матиме два кроки – перетворення з OVA в raw.img (формат KVM), і потім з raw.img в rootfs. На рисунку 3.12 подано код скрипта на BASH, що конвертує з OVA в rootfs. Скрипт поділено на дві частини – перетворення OVA на IMG та IMG на rootfs.

```
1   #!/bin/bash
  1 # ---------part 1---------
  2 # -------OVA > IMG--------
  3 # takes .ova file as input
  4 # listing contents of the ova
  5 tar tvf "$1" # add checking if it is truly .ova or let tar fail and complain
  6 # extracting vmdk from ova
  7 tar xvf "$1" --wildcards *vmdk
  8 # convert vmdk into img (KVM)
  9 # put new names into array for later reference
 10 # typedef -a images
 11 # for i in vmdk; do
 12 # images+="${i/vmdk/img}"
 13 imgname="${i/vmdk/img}"
 14 qemu-img -p convert -f vmdk -O raw "$i" "$imgname"
 15 # done
 16 # ---------part 2---------
 17 # -------IMG > ROOTFS-----
 18 # dont foget to rename imap
 19 # create lxc image with the name of .ova file
 20 lxc-create --name "${1/.ova/}"
 21 # get the start block of an img
 22 start_block=$(sudo fdisk -l "$imgname" | grep ^`realpath "$imgname" |  awk -F" "  '{ print $3 }')
 23 # get the sector size of an img
 24 sector_size=$(sudo fdisk -l "$imgname" | grep ^Sector |  awk -F" "  '{ print $4 }')
 25 # mount image; may be buggy, do check
 26 mount -o loop,offset=$(($start_block*$sector_size)) "$imgname" /mnt-source
 27 # sync image filesystem contents to rootfs
 28 rsync -av /mnt-source/* /var/lib/lxc/"${1/.ova/}"/rootfs
```

Рисунок 3.12 – Фрагмент коду скрипта конвертування OVA в rootfs

Наступні кроки виконання скрипта є такими:

1. До rootfs додаються файли пристроїв, таких як /dev/null, /dev/zero, /dev/tty0 і т.д.
2. Створюється LXC конфігурація для rootfs.
3. Виконується запуск створеного контейнера.

Весь інший сирцевий код для налаштування отриманої rootfs та запуску контейнера подано в додатку В.

3.5 Розробка розгортання системи з допомогою GNU make

Аби систему було легше встановити на власний комп’ютер, прийнято рішення автоматизувати цей процес (рисунок 3.13). Популярним вибором є make (в цій роботі використано GNU make – версія make для систем GNU/Linux) – програма яка дозволяє описувати послідовність дій для компіляції, побудови, встановлення, розгортання програм чи програмних систем. Головною перевагою make є те, що вона виконує вищезазначені дії лише над файлами, які змінилися з часу попереднього виклику make. Тому час на повторне розгортання програмної системи, наприклад, у разі виходу оновлень чи під час розробки, значно зменшується. Таким чином, завдяки можливостям GNU make процес розгортання системи вивчення відкритих ОС займатиме лічені хвилини.

```
# set ID from /etc/os-release
$(eval $(shell grep ^ID /etc/os-release)) # e.g. ID=debian
# set VERSION_ID from /etc/os-release
$(eval $(shell grep ^VERSION_ID /etc/os-release)) # e.g. VERSION_ID=11

setup-env:
# If there is no indentation, Make will treat it as a directive for itself; otherwise, it's regarded as a shel
cript.
ifeq ($(ID),debian)
ifeq ($(VERSION_ID),11)
    sudo apt-get install ansible lxc lxc-dev python2
# install python2-lxc. Is required by ansible to run
# https://docs.ansible.com/ansible/latest/collections/community/general/lxc_container_module.html
    git clone https://github.com/lxc/python2-lxc
    cd python2-lxc
    python setup.py build
    python setup.py install
else
    echo -e "Change the Makefile to suit your needs to use the different version of Debian than "$(VERSION_ID)
    ".\nBeware of difficulties which might arise."
endif
endif
```

Рисунок 3.13 – Фрагмент коду GNU make для розгортання системи

Двома першими директивами в Makefile отримано ім’я та версію дистрибутиву Linux на якому буде розгорнуто систему. Тоді виконується блок коду “setup-env:”. Відбувається перевірка імені та версії дистрибутиву і встановлюються необхідні бібліотеки та програми – компоненти системи. Це здійснюється з допомогою пакетного менеджера “apt-get”, а програма якої нема в репозиторії “apt-get” (python2-lxc) встановлюється вручну – спершу отримано її сирцевий код з GitHub репозиторію та виконано скрипт налаштування (setup.py).

3.6 Висновки

Отже, у третьому розділі було проведено варіантний аналіз та обґрунтовано вибір інструментів для розробки системи, а саме: Ansible та LXC, PostgreSQL, Python, Flask, JavaScript Tabulator, Emacs, VIM. Розроблено базу даних на рівні концепції (ER-модель) та реалізації (ER-діаграма та SQL запити). Розроблено рейтингову таблицю, в якій відображаються успіхи студентів (користувачів системи). Створено систему сумісності для можливості перетворення вже наявних віртуальних машин на контейнери. Також було реалізовано процес розгортання системи вивчення відкритих ОС з допомогою GNU make.

4 ТЕСТУВАННЯ СИСТЕМИ ВИВЧЕННЯ ВІДКРИТИХ ОС

4.1 Тестування елементів системи вивчення відкритих ОС

Щоб перевірити складну систему на наявність помилок, не достатньо перевірити її складові. Це є наслідком емерджентності. Для тестування подібної складної системи потрібно розроблювати ще одну систему яка б цим займалась, проте і тоді не було б абсолютних гарантій щодо правильності роботи першої, оскільки немає гарантій щодо правильності роботи другої. Тим не менш, це не означає, що не потрібно тестувати окремі компоненти системи. Великою перевагою системи є те, що використані підсистеми розробляють та тестують окремі спільноти розробників (через що ці підсистеми доволі стабільні). Розробники, проте, не можуть попередити помилок в конфігуруванні власних систем, тому для тестування системи відкритих ОС варто зосередитись на тестуванні того її елементу, який є найкритичнішим – на базі даних. Якщо неправильно проходить запис чи зчитування даних, викладач може отримувати фальшиву картину успіхів своїх студентів, а студенти не зорієнтуються в власному прогресі. На рисунках 4.1, 4.2 та 4.3 наведені Python функції які здійснюють запис тестових даних до бази даних.

```
73 def add_student():
74     student = Student(name = fake.name(),
75                       email = fake.email(),
76                       ssh_key = uuid.uuid4().hex
77                       )
78     db.session.add(student)
79     db.session.commit()
```

Рисунок 4.1 – Функція створення запису в таблиці Student (Студент)

```
81 def add_task():
82     task = Task(name = fake.last_name(),
83                 description = "describing stuff",
84                 rating = random.randint(1,10),
85                 creation_date = fake.date(),
86                 solution = {'solved' : 'solved'}
87                 )
88     db.session.add(task)
89     db.session.commit()
```

Рисунок 4.2 – Функція створення запису в таблиці Task (Завдання)

```
 91 def add_journal():
 92     # a list of existing student and task ids
 93     stud_ids = [id[0] for id in Student.query.with_entities(Student.id_student).all()]
 94     task_ids = [id[0] for id in Task.query.with_entities(Task.id_task).all()]
 95     # choose random id out of stud/task ids
 96     journal = Journal(id_task = random.choice(task_ids),
 97                       id_student = random.choice(stud_ids),
 98                       student_solution = {'solved' : 'solved'},
 99                       date_submitted = fake.date(),
100                       grade = random.randint(0,10)
101                       )
102     db.session.add(journal)
103     db.session.commit()
```

Рисунок 4.3 – Функція створення запису в таблиці Journal (Журнал)

В основі цих функцій використана бібліотека Faker (дослівно “дурисвіт”, “той, що дурить”) – бібліотека, що генерує тестові (фейкові) дані [26] для тестування бази даних. Після створення функцій, їх виконано (рисунок 4.4)

```
Python 3.9.12 (main, Mar 24 2022, 14:31:07)
[GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from clickrating import add_task, add_student, add_journal
>>> for i in range(10):
...     add_student()
...
>>> for i in range(30):
...     add_task()
...
>>> for i in range(60):
...     add_journal()
...
>>>
```

Рисунок 4.4 – Виконання тестових функцій

Після виконання запису до бази даних, зроблено декілька SQL запитів, щоб перевірити його правильність.

На рисунках 4.5, 4.6 та 4.7 зображено термінальний інтерфейс PostgreSQL “psql”. З допомогою звичайних запитів SELECT із кожної таблиці розробленої для системи вивчення відкритих операційних систем було отримано всі її рядки. Тестування показало, що запис та читання відбуваються коректно.

```
clickDB=> select * from  journal;
 id_journal | id_task | id_student |   student_solution   | date_submitted | grade
------------+---------+------------+----------------------+----------------+-------
          1 |      18 |          9 | {"solved": "solved"} | 1994-12-28     |     6
          2 |      10 |          2 | {"solved": "solved"} | 2006-09-20     |     1
          3 |       9 |          4 | {"solved": "solved"} | 1971-02-06     |     0
          4 |      30 |          2 | {"solved": "solved"} | 1982-01-22     |     3
          5 |      29 |          7 | {"solved": "solved"} | 2006-04-10     |     2
          6 |      27 |          2 | {"solved": "solved"} | 1975-05-16     |     3
          7 |      26 |          9 | {"solved": "solved"} | 2011-08-20     |     0
          8 |      29 |          1 | {"solved": "solved"} | 2018-02-05     |     8
          9 |      19 |          5 | {"solved": "solved"} | 1975-01-16     |     9
         10 |      26 |          5 | {"solved": "solved"} | 1973-04-06     |     4
         11 |       3 |          9 | {"solved": "solved"} | 2008-04-15     |     1
         12 |      15 |         10 | {"solved": "solved"} | 1985-10-25     |     8
         13 |      25 |          7 | {"solved": "solved"} | 1973-02-15     |     7
         14 |      18 |         10 | {"solved": "solved"} | 1989-05-18     |     5
         15 |       4 |          2 | {"solved": "solved"} | 1994-11-09     |     7
         16 |      12 |          8 | {"solved": "solved"} | 1992-03-10     |     2
         17 |       4 |         10 | {"solved": "solved"} | 2011-07-31     |     2
         18 |      25 |         10 | {"solved": "solved"} | 2016-08-29     |     1
         19 |      14 |          7 | {"solved": "solved"} | 1979-10-19     |     0
         20 |      13 |          8 | {"solved": "solved"} | 1997-04-04     |     0
         21 |      13 |          1 | {"solved": "solved"} | 2003-05-19     |     1
         22 |      28 |          4 | {"solved": "solved"} | 1998-08-04     |     5
         23 |       7 |          4 | {"solved": "solved"} | 2012-01-25     |     7
         24 |       3 |          1 | {"solved": "solved"} | 1973-11-27     |     7
         25 |      17 |          6 | {"solved": "solved"} | 2010-06-09     |     8
         26 |       8 |          4 | {"solved": "solved"} | 2014-03-18     |     8
         27 |      13 |          9 | {"solved": "solved"} | 2015-08-27     |     7
         28 |      16 |          7 | {"solved": "solved"} | 2014-11-18     |     2
```

Рисунок 4.5 – Тестування таблиці Journal (Журнал)

```
clickDB=> select * from  student;
 id_student |           name           |               email                |             ssh_key
------------+--------------------------+------------------------------------+----------------------
          1 | панı Мар’яна Даниленко   | bvashchenko-zakharchenko@example.net | 0ecafc4a461548fb94d7
          2 | пан Григорій Бабій       | dannaskyba@example.org             | 84ea7d6fc974452eaecf
          3 | Леон Товстолıс           | ivelychko@example.org              | 63331b2c18514d4da9dd
          4 | Федıр Шутько             | zlatoslava46@example.com           | eff981b773f74c66836e
          5 | Роксолана Гайденко       | matiashvadym@example.net           | c5f437b6bb264df49c2f
          6 | Венıямин Масоха          | veniiamyn38@example.org            | 406a47594c964c04b42f
          7 | Давид Дробаха            | arsen52@example.org                | 982e6093aeb6491fabd5
          8 | панı Василина Нестеренко | volodymyrtereshchenko@example.com  | c0afb77f9983414daca5
          9 | Дем’ян Стельмах          | pavlychenkosnizhana@example.com    | 118354d18d6942198898
         10 | Ада Гупало               | wvasylechko@example.com            | f94070ae79e543dda3f6
(10 rows)
```

Рисунок 4.6 – Тестування таблиці Student (Студент)

Варто зазначити, що Faker підтримує різні локалізації, зокрема й українську, тобто він вміє створювати українські імена, поштові адреси, інші назви.

Задати об’єкту Faker локалізацію можна одним з двох способів, які є еквівалентним – Faker(['uk-UA', 'uk_UA']). Навіть попри те, що тут об’єкту Faker передано локалізацію двічі, він все одно використовуватиме єдину локалізацію – 'uk-UA'.

```
clickDB=> select id_task, description, rating, creation_date, solution from  task;
 id_task |   description    | rating | creation_date |       solution
---------+------------------+--------+---------------+----------------------
       1 | describing stuff |      8 | 2017-06-18    | {"solved": "solved"}
       2 | describing stuff |      7 | 1978-08-04    | {"solved": "solved"}
       3 | describing stuff |      2 | 2011-07-18    | {"solved": "solved"}
       4 | describing stuff |      7 | 1982-04-06    | {"solved": "solved"}
       5 | describing stuff |      6 | 1998-07-12    | {"solved": "solved"}
       6 | describing stuff |     10 | 2009-01-30    | {"solved": "solved"}
       7 | describing stuff |      5 | 1971-05-11    | {"solved": "solved"}
       8 | describing stuff |      3 | 2017-07-13    | {"solved": "solved"}
       9 | describing stuff |      5 | 1983-11-11    | {"solved": "solved"}
      10 | describing stuff |      2 | 1998-08-15    | {"solved": "solved"}
      11 | describing stuff |      8 | 2006-03-19    | {"solved": "solved"}
      12 | describing stuff |     10 | 1983-01-22    | {"solved": "solved"}
      13 | describing stuff |      6 | 1991-11-30    | {"solved": "solved"}
      14 | describing stuff |      3 | 2019-05-01    | {"solved": "solved"}
      15 | describing stuff |      2 | 1995-10-01    | {"solved": "solved"}
      16 | describing stuff |      6 | 1999-12-01    | {"solved": "solved"}
      17 | describing stuff |      5 | 2016-10-06    | {"solved": "solved"}
      18 | describing stuff |      8 | 1991-12-16    | {"solved": "solved"}
      19 | describing stuff |      8 | 2014-04-22    | {"solved": "solved"}
      20 | describing stuff |      4 | 1973-10-16    | {"solved": "solved"}
      21 | describing stuff |      5 | 2000-01-20    | {"solved": "solved"}
      22 | describing stuff |      4 | 1997-09-18    | {"solved": "solved"}
      23 | describing stuff |      7 | 1993-01-07    | {"solved": "solved"}
      24 | describing stuff |      4 | 1977-08-21    | {"solved": "solved"}
      25 | describing stuff |     10 | 1980-02-18    | {"solved": "solved"}
      26 | describing stuff |      2 | 1981-04-04    | {"solved": "solved"}
      27 | describing stuff |      3 | 2011-01-12    | {"solved": "solved"}
      28 | describing stuff |     10 | 2014-08-10    | {"solved": "solved"}
      29 | describing stuff |      4 | 1973-06-14    | {"solved": "solved"}
      30 | describing stuff |      2 | 2018-01-24    | {"solved": "solved"}
(30 rows)
```

Рисунок 4.7 – Тестування таблиці Task (Завдання)

Створює користувача, завдання та відповіді до них. Таким чином перевіряється правильність роботи бази даних.

Отже, тестування бази даних успішно завершене. Після нього проведено

тестування розгортання контейнерів Debian Linux на LXC. На рисунку 4.8 наведено процес розгортання контейнера “test5”.

```
---> lxc-create --name test5 --template=download -- --dist=debian -
Using image from local cache
Unpacking the rootfs

---
You just created a Debian stretch amd64 (20220603_05:25) container.

To enable SSH, run: apt install openssh-server
No default root or user password are set by LXC.
---> lxc-ls --fancy
NAME      STATE   AUTOSTART GROUPS IPV4       IPV6 UNPRIVILEGED
deb1      RUNNING 0         -      10.0.3.26  -    false
deb2      RUNNING 0         -      10.0.3.80  -    false
deb3      RUNNING 0         -      10.0.3.201 -    false
t1        RUNNING 0         -      10.0.3.54  -    false
t1-clone  STOPPED 0         -      -          -    false
t1-clone2 RUNNING 0         -      10.0.3.244 -    false
t2        RUNNING 0         -      10.0.3.127 -    false
test1     RUNNING 0         -      10.0.3.220 -    false
test2     RUNNING 0         -      10.0.3.120 -    false
test3     RUNNING 0         -      10.0.3.21  -    false
test4     RUNNING 0         -      10.0.3.2   -    false
test5     STOPPED 0         -      -          -    false
---> lxc-start test5
---> lxc-ls --fancy
NAME      STATE   AUTOSTART GROUPS IPV4       IPV6 UNPRIVILEGED
deb1      RUNNING 0         -      10.0.3.26  -    false
deb2      RUNNING 0         -      10.0.3.80  -    false
deb3      RUNNING 0         -      10.0.3.201 -    false
t1        RUNNING 0         -      10.0.3.54  -    false
t1-clone  STOPPED 0         -      -          -    false
t1-clone2 RUNNING 0         -      10.0.3.244 -    false
t2        RUNNING 0         -      10.0.3.127 -    false
test1     RUNNING 0         -      10.0.3.220 -    false
test2     RUNNING 0         -      10.0.3.120 -    false
test3     RUNNING 0         -      10.0.3.21  -    false
test4     RUNNING 0         -      10.0.3.2   -    false
test5     RUNNING 0         -      10.0.3.184 -    false
```

Рисунок 4.8 – Тестування одного з контейнерів LXC

Після виклику команди “lxc-create”, створений контейнер знаходиться у зупиненому стані, тому його потрібно запустити з допомогою “lxc-start”. “lxc-ls” здійснює виведення інформації щодо стану контейнерів у вікно терміналу.

4.2 Розробка інструкції користувача

Завдяки правильно написаній інструкції шанси ширшого прийняття системи з боку викладачів чи ентузіастів зростають, оскільки інструкція економить їхній час – їм не потрібно читати сирцевий код системи, щоб зрозуміти як вона працює та як її налаштовувати. Відомості про мінімальні необхідні та рекомендовані параметри комп’ютера наведено в таблиці 4.1 та 4.2.

Таблиця 4.1 – Мінімальна конфігурація

Процесор	64-розрядний (x64) процесор з тактовою частотою від 2 ГГц
Оперативна пам’ять	4 ГБ
Місце на жорсткому диску	30 ГБ
Операційна система	Debian 11

Таблиця 4.2 – Рекомендована конфігурація

Процесор	64-розрядний (x64) процесор з тактовою частотою від 2.5 ГГц
Оперативна пам’ять	8 ГБ
Місце на жорсткому диску	50 ГБ
Операційна система	Debian 11

Завантаживши файли із сирцевим кодом системи вивчення відкритих ОС, викладач (користувач), уважно прочитавши README та LICENSE файли, що розповсюджуються разом із сирцевим кодом, повинен буде виконати файл розгортання (Makefile). Окрім декількох конфігурацій, як от задання розміру навчальної групи (кількості LXC) чи налаштовування адміністративних паролів, на етапі розгортання втручання викладача є мінімальним.

Для запуску системи і, відповідно, її підсистем, викладач має мати права адміністратора на комп'ютері на якому він її запускає, а також адміністративний

обліковий запис для бази даних. Налаштування останнього передбачено в Makefile. В результаті він отримує змогу комунікувати з базою даних (записувати нові завдання, додавати студентів, переглядати розв’язки задач), створювати нові LXC, конвертувати образи віртуальних машин і т.д.

Викладач може керувати відображенням рейтингової таблиці, запускаючи чи вимикаючи відповідну підсистему (веб-сервер), а також обнуляти рейтинг.

4.3 Висновки

У четвертому розділі проведено ручне тестування системи для вивчення відкритих ОС. Перевірено найкритичнішу підсистему – базу даних з допомогою функцій, що створюють тестові дані та SQL запити. Разом із тим перевірено правильність розгортання контейнерів LXC. Розроблено інструкцію користувача, яка допоможе зробити перші кроки у розгортанні та застосуванні системи, а також суттєво зекономить час користувача.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено систему для вивчення відкритих операційних систем на основі хмарних технологій інструментами GNU/LINUX. Для розробки було використано виключно вільні та відкриті програми ОС GNU/Linux, розробка здійснювалась в середовищі GNU Emacs та термінальному текстовому редакторі VIM.

Під час виконання бакалаврської дипломної роботи було проаналізовано нинішній стан проблеми. Було розглянуто основні аналоги системи. Було визначено їхні недоліки та переваги й порівняно їх з власною розробкою. На основі проведеного аналізу було сформовано перелік задач, які повинні бути виконані у бакалаврській дипломній роботі.

У варіантному аналізі технологій розробки було обґрунтовано вибір мови програмування Python та підсистем включених у систему: Ansible – для автоматизованого розгортання, LXC – для контейнеризації, PostgreSQL – для збереження виконаних завдань, BASH – для перетворення образів віртуальних машин на контейнери, Python Flask, CSS/HTML, JavaScript та NGINX – для веб-інтерфейсу рейтингової таблиці, Markdown для інструкції користувача (README).

У бакалаврській дипломній роботі було виконано такі задачі:

- розроблено моделі системи вивчення відкритих ОС;
- розроблено метод перевірки виконання завдань;
- розроблено формат для розгортання навчального середовища;
- розроблено базу даних для збереження завдань;
- розроблено рейтингову таблицю;
- розроблено формат навчального середовища (для його реплікації);
- розроблено шар сумісності (перетворення форматів) між віртуальними машинами та контейнерами;
- розроблено інструкцію користувача;
- проведено тестування системи вивчення відкритих ОС.

Було розроблено моделі системи вивчення відкритих ОС з боку самої системи, студента та викладача. Розроблено базу даних системи на рівні концепції у вигляді ER-моделі та на рівні реалізації – з допомогою SQL запитів, що відображено в ER-діаграмі. Також було розроблено метод перевірки виконання завдань та модель роботи системи вивчення відкритих ОС.

Тестування системи довело її працездатність та відповідність поставленому технічному завданню. Також було розроблено інструкцію користувача.

Наступними векторами розвитку системи для вивчення відкритих ОС є її оптимізація для роботи з іншими популярними відкритими ОС, окрім GNU/Linux, наприклад, ОС сімейства BSD. Важливим покращенням буде створення скрипту конфігурування форвардингу X Window System через SSH протокол, що уможливить використання графічних програм GNU/Linux з контейнеру LXC всього-на-всього підключившись до нього по SSH. Також однією з провідних задач буде розробка низки скриптів для тестування, налагодження та резервного копіювання системи та розробка більш глибокої документації з типовими “рецептами” конфігурування системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Top 100 careers [Електронний ресурс] – Режим доступу до ресурсу: https://www.careerprofiles.info/top-100-careers.html.
2. Операційна система [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Операційна_система.
3. PC operating system distribution for software development worldwide in 2018 to 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/869211/worldwide-software-development-operating-system.
4. Comparison of Linux distributions [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Comparison_of_Linux_distributions.
5. Ісаков А.В. Сфери застосування нереляційних баз даних / А.В. Ісаков, В.Ю. Ліміна, О.В. Романюк // Матеріали XLIX Науково-технічної конференції факультету інформаційних технологій та комп’ютерної інженерії (2020), Секція програмного забезпечення. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2020/paper/download/9416/7928.
6. Ісаков А.В. Markdown language as an essential tool for digital writing / А.В. Ісаков, В.Г. Дерун // Матеріали XLIX Науково-технічної конференції Інституту соціально-гуманітарних наук (2020), Секція англійської мови. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-hum/all-hum-2020/paper/download/8806/7444.
7. Ісаков А.В. Розробка інфраструктури для вивчення відкритих операційних систем на основі хмарних технологій інструментами GNU/Linux / А.В. Ісаков // Матеріали LI Науково-технічної конференції факультету інформаційних технологій та комп’ютерної інженерії (2022), Секція програмного забезпечення. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/

download/16149/13556.

8. Клуб VNTU Linuxoids [Електронний ресурс] – Режим доступу до ресурсу: https://vntu-linuxclub.github.io.
9. HackTheBox [Електронний ресурс] – Режим доступу до ресурсу: https://help.hackthebox.com/en/collections/2919917-main-platform.
10. Vulnhub [Електронний ресурс] – Режим доступу до ресурсу: https://www.vulnhub.com/about.
11. AWS Academy [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/training/awsacademy.
12. Coursera [Електронний ресурс] – Режим доступу до ресурсу: https://about.coursera.org.
13. Coupling [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Coupling_(computer_programming)#Disadvantages_of_tight_coupling.
14. Top programming languages 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://spectrum.ieee.org/top-programming-languages-2021.
15. W. Ross Ashby, An Introduction to Cybernetics, Chapman & Hall, London, 1956 [Електронний ресурс] – Режим доступу до ресурсу: http://pespmc1.vub.ac.be/ASHBBOOK.html.
16. Loose coupling [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Loose_coupling#In_programming.
17. Зінченко О.В. Хмарні технології / О.В. Зінченко, С.М. Іщеряков, С.В. Прокопов, С.О. Сєрих, В.В. Василенко // Навчальний посібник. – Київ: ФОП Гуляєва В.М., 2020. – 74 с.
18. Top cloud providers: AWS, Microsoft Azure, and Google Cloud, hybrid, SaaS players [Електронний ресурс] – Режим доступу до ресурсу: https://www.zdnet.com/article/the-top-cloud-providers-of-2021-aws-microsoft-azure-google-cloud-hybrid-saas.
19. AWS Educate [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/blogs/aws/aws-educate-credits-training-content-and-

collaboration-for-students-educators.

20. Python: supported platforms and architectures [Електронний ресурс] – Режим доступу до ресурсу: https://pythondev.readthedocs.io/platforms.html.
21. Russ McKendrick Learn Ansible, Automate cloud, security, and network infrastructure using Ansible 2.x : навч. посіб. / Russ McKendrick, Packt Publishing; 2018. – 558 с.
22. Konstantin Ivanov Containerization with LXC : навч. посіб. / Konstantin Ivanov, Packt Publishing; 2017. – 341 с.
23. Hans-Jürgen Schönig Mastering PostgreSQL 13 : навч. посіб. / Hans-Jürgen Schönig, Packt Publishing; 4 edition, 2020. – 363 с.
24. Поняття ER-моделі [Електронний ресурс] – Режим доступу до ресурсу: https://www.bestprog.net/uk/2019/01/24/the-concept-of-er-model-the-concept-of-essence-and-communication-attributes-attribute-types-ua.
25. JavaScript Tabulator [Електронний ресурс] – Режим доступу до ресурсу: http://tabulator.info.
26. Python Faker [Електронний ресурс] – Режим доступу до ресурсу: https://faker.readthedocs.io/en/master.

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"_25_" _березня_2022 р.

Технічне завдання

на бакалаврську дипломну роботу «Розробка автоматизованої системи для вивчення відкритих операційних систем на основі хмарних технологій інструментами GNU/LINUX»

за спеціальністю

121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:

_____ к.т.н., доц. В.В. Войтко

"____"_____2022 р.

Виконав:

_____студент гр. 1ПІ-18б А.В. Ісаков

"____"_____2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка автоматизованої системи для вивчення відкритих операційних систем на основі хмарних технологій інструментами GNU/LINUX».

Галузь застосування – операційні системи.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № _12_ від «_07_» _лютого_ 2022 р.

3. Мета та призначення розробки.

Метою бакалаврської дипломної роботи є підвищення ефективності вивчення відкритих ОС шляхом розробки системи, яка дозволяє автоматизувати розгортання навчального середовища та здійснювати автоматичну перевірку виконання завдань відповідно до потреб викладача.

Призначення роботи – для автоматизації процесу розгортання навчального середовища та перевірки завдань студентів.

4. Вихідні дані для проведення НДР

1. W. Ross Ashby, An Introduction to Cybernetics, Chapman & Hall, London, 1956 [Електронний ресурс] – Режим доступу до ресурсу: http://pespmc1.vub.ac.be/ASHBBOOK.html.
2. Зінченко О.В. Хмарні технології / О.В. Зінченко, С.М. Іщеряков, С.В. Прокопов, С.О. Сєрих, В.В. Василенко // Навчальний посібник. – Київ: ФОП Гуляєва В.М., 2020. – 74 с.
3. Russ McKendrick Learn Ansible, Automate cloud, security, and network infrastructure using Ansible 2.x : навч. посіб. / Russ McKendrick, Packt Publishing; 2018. – 558 с.
4. Konstantin Ivanov Containerization with LXC : навч. посіб. / Konstantin

Ivanov, Packt Publishing; 2017. – 341 с.

5. Hans-Jürgen Schönig Mastering PostgreSQL 13 : навч. посіб. / Hans-Jürgen Schönig, Packt Publishing; 4 edition, 2020. – 363 с.

5. Технічні вимоги

Вхідні дані – інформація користувача, кількість вакансій; вихідні дані – графічний інтерфейс зі списком доступних вакансій.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до бакалаврської дипломної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка автоматизованої системи для вивчення відкритих операційних систем на основі хмарних технологій інструментами GNU/LINUX

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: доц. каф. ПЗ Войтко В.В.

Оригінальність	98,8%
Схожість	1,2%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
🐦 Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
🐦 Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Ісаков Андрій Васильович

Керівник роботи _____ Войтко Вікторія Володимирівна

Додаток В – Лістинг програми

SQL-запити для створення таблиць та запису даних до бази даних PostgreSQL

```
DROP TABLE IF EXISTS journal;
DROP TABLE IF EXISTS student CASCADE;
DROP TABLE IF EXISTS task CASCADE;

CREATE TABLE task (
    id_task         int PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    name            varchar(80),
    description     varchar(800),
    rating          int,
    creation_date   date,
    solution        jsonb
);

CREATE TABLE student (
    id_student      int PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    name            varchar(80),
    email           varchar(100),
    ssh_key         varchar(1000)
);

CREATE TABLE journal (
    id_journal          int PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    id_task             int references task(id_task),
    id_student          int references student(id_student),
    student_solution    jsonb NOT NULL,
    date_submitted      date,
    grade               int
```

```
);
INSERT INTO task (name, description, rating, creation_date, solution)
    VALUES ('tricky nmap',
'Try to set up http and smb servers. Output nmap output to prove which ports are open.
Output format: <port>/<connection_type> open <service_type>',
2,
'2022-05-15',
'{"nmap":"22/tcp open ssh\n80/tcp open http\n139/tcp open netbios-ssn\n445/tcp open microsoft-ds","check_command":"nmap localhost"}'
);

INSERT INTO student (name, email, ssh_key)
    VALUES ('Andrew', 'andrew@example.com',
'ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQCgcPOpb1zRgzGN7NJ0NryWWztkrE
954/HSpQnkHoWyMmcnJ3KvDHAb14KWVhdbHxOTdhU+pxRIzTdNaP9HFzxxolP
Z6RhhXW1UMeg93G/CirUpKUaY06Ma7XHjdo/
d1CMG0eemXVDZXeb3Zrw8DERHijVSY4e3ypntsLDA8aUDuh0SfeAUdIBAG5JVR
PBJbFRYCXE2z5U5TgilSD45aKWeZk4nEGwaqCxb/
cqgTAE0SiXaWSMN9hH2hmq0E5D/M/BFX2EIQF/8vi47+Qar/
lAk6IEb3z7AY9Di9toroSMsapW2XQ+DyS+kAilwUI7vLQwYTahqObO2VETA1TpD
FLn5bylqJmo1uclXb0smkuX0rXeXlku6L/7EaTOMxNsUx7jkybqJr+/
jEarsMfJ40q5zjYZX35G2v/3rsv0cvrg6Rsn+RlvamYqP3MfoGOqh/GNl/
It7YSHVkidCPCrFU8iEALf6nU/bTPvJp/Wt1ovQTq5VyuV5Q/1XvhW8UlYhxJc=
andrew@pc'
);

INSERT INTO journal (id_task, id_student, student_solution, date_submitted, grade)
    VALUES (1, 1,
'{"nmap":"22/tcp open ssh\n80/tcp open http\n139/tcp open netbios-ssn\n445/tcp open microsoft-ds","check_command":"nmap localhost"}',
'2022-05-16',
10);
```

```
SELECT * FROM journal, student, task
WHERE journal.id_student = student.id_student AND
journal.id_task = task.id_task
ORDER BY id_journal ASC
```

Makefile (файл розгортання для GNU make)

```
# makefile for deploying Click - a system for deploying an infrastructure for learning open operating systems.
# On Debian-based Linux distributions.
# Slackware support is experimental.
# See LICENSE file for copyright and license details.
# MVP (minimum viable product) version

# set ID from /etc/os-release
$(eval $(shell grep ^ID /etc/os-release)) # e.g. ID=debian
# set VERSION_ID from /etc/os-release
$(eval $(shell grep ^VERSION_ID /etc/os-release)) # e.g. VERSION_ID=11

setup-env:
# If there is no indentation, Make will treat it as a directive for itself; otherwise, it's regarded as a shell script.
ifeq ($(ID),debian)
ifeq ($(VERSION_ID),11)
	sudo apt-get install -y ansible lxc lxc-dev python2 postgresql-13 postgresql-doc-13 nginx
# install python2-lxc. Is required by ansible to run
# https://docs.ansible.com/ansible/latest/collections/community/general/lxc_container_module.html
	git clone https://github.com/lxc/python2-lxc
```

```
        cd python2-lxc
        python setup.py build
        python setup.py install
    pip install -r requirements.txt


### database setup To Be Done (TBD) ###
# https://stackoverflow.com/questions/39850860/disable-wrapping-in-psql-output
sudo echo "export PAGER=less -S" >> /etc/profile
su postgres -c "psql -c \"CREATE USER admin WITH CREATEDB CREATEROLE LOGIN PASSWORD 'admin123'\""
# su postgres -c "psql -c \"DROP USER admin\"" # to drop=delete=remove user
su postgres -c "createdb clickDB -O admin" # need to give password for "postgres" system user # or not depending on /etc/sudoers configuration
#su postgres -c "psql -d clickDB -h localhost -U admin"
#Пароль:
#Пароль користувача admin:

su postgres
echo "localhost:5432:clickDB:admin:admin123" >> ~/.pgpass
chmod 600 ~/.pgpass
exit
# USING COPY TO or FROM:
#ERROR: ПОМИЛКА:  потрібно бути суперкористувачем або членом ролі pg_write_server_files, щоб виконати COPY з записом у файл
#HINT:  Будь-хто може використати COPY to stdout або from stdin, а також команду psql \copy.
#GRANT pg_read_server_files,pg_write_server_files TO admin; # to be able to use COPY write to the filesystem
su postgres -c "psql -c \"GRANT pg_read_server_files,pg_write_server_files TO admin\""
```

```
# pgAdmin 4
# email: admin@admin.lan
# passw: admin12

# sudo apt-get install postgresql-doc-13 # to read docs offline
# look at it at /usr/share/doc/postgresql-doc-13/html/index.html

else
	echo -e "Change the Makefile to suit your needs to use the different version of Debian than "$(VERSION_ID) \
	".\nBeware of difficulties which might arise."
endif
endif

# slackware 15.0 setup
ifeq ($(ID),slackware)
ifeq ($(VERSION_ID),15.0)
#sudo sboui-backend install ansible
# need a better SBO cli tool for using in scripts, or just
# installpkg packagename.tgz
# but need to supply prebuilt packagename.tgz
else
	echo -e "Change the Makefile to suit your needs to use the different version of Slackware than "$(VERSION_ID)".\nBeware of difficulties which might arise."
endif
endif
.PHONY: run clean setup-env
```

Рейтингова таблиця

clickrating.py

```
from datetime import datetime
from flask import Flask, render_template, url_for, flash, redirect
from flask_sqlalchemy import SQLAlchemy
#from sqlalchemy import create_engine
#from forms import RegistrationForm, LoginForm
import random
import urllib.parse
from sqlalchemy.dialects import postgresql
from sqlalchemy.sql import func
# for random data generation
from faker import Faker
import uuid
from sqlalchemy import text


app = Flask(__name__)
fake = Faker(['uk_UA'])
# As the URL is like any other URL, special characters such as those that may be used in the password need to be URL encoded to be parsed correctly.
# https://docs.sqlalchemy.org/en/14/core/engines.html#postgresql
password=urllib.parse.quote_plus("admin123")
# for the simple pass it's fine, but
# "...an example of a URL that includes the password "kx%jj5/g", where the percent sign and slash characters are represented as %25 and %2F, respectively"
db_string = "postgresql://admin:"+password+"@localhost:5432/clickDB"


app.config['SECRET_KEY'] = 'e5c7e94b260341f0a003d14189928ca0'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

```
app.config['SQLALCHEMY_DATABASE_URI'] = db_string
db = SQLAlchemy(app)


class Task(db.Model):
    id_task = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(80), nullable=False)
    description = db.Column(db.String(800), nullable=False)
    rating = db.Column(db.Integer, nullable=False)
    creation_date = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    solution = db.Column(postgresql.JSONB(astext_type=db.Text()))

    journals = db.relationship('Journal', backref='task', lazy=True)
    def __repr__(self):
        return f"Task('{self.name}', '{self.description}', '{self.rating}', '{self.solution}')"


class Student(db.Model):
    id_student = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(80), nullable=False)
    email = db.Column(db.String(100), nullable=False)
    ssh_key = db.Column(db.String(1000), nullable=False)

    # get a Student object which is referenced by the Journal entry
    # eg -> J = Journal (student1_id, task_id, student_solution, etc)
    # J.student -> gives student object Student(id, name, email, ssh_key)
    # ! this is not a column in PostgreSQL, but a feature of SQLAlchemy
    journals = db.relationship('Journal', backref='student', lazy=True)
    def __repr__(self):
```

```
        return f"Student('{self.name}', '{self.email}')"


class Journal(db.Model):
    id_journal = db.Column(db.Integer, primary_key=True)
    id_task = db.Column(db.Integer, db.ForeignKey('task.id_task'), nullable=False)
    id_student = db.Column(db.Integer, db.ForeignKey('student.id_student'),
nullable=False)
    student_solution = db.Column(postgresql.JSONB(astext_type=db.Text()))
    date_submitted = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    grade = db.Column(db.Integer, nullable=False, default=0)


    def __repr__(self):
        return f"Journal('{self.id_journal}', '{self.id_task}', '{self.id_student}',
'{self.date_submitted}')"


    # # table 1
    # select id_student, grade
    # from journal
    # where grade > 0


def add_student():
    student = Student(name = fake.name(),
                      email = fake.email(),
                      ssh_key = uuid.uuid4().hex
                      )
    db.session.add(student)
    db.session.commit()


def add_task():
```

```
    task = Task(name = fake.last_name(),
                description = "describing stuff",
                rating = random.randint(1,10),
                creation_date = fake.date(),
                solution = {'solved' : 'solved'}
                )
    db.session.add(task)
    db.session.commit()


def add_journal():
    # a list of existing student and task ids
    stud_ids = [id[0] for id in Student.query.with_entities(Student.id_student).all()]
    task_ids = [id[0] for id in Task.query.with_entities(Task.id_task).all()]
    # choose random id out of stud/task ids
    journal = Journal(id_task = random.choice(task_ids),
                      id_student = random.choice(stud_ids),
                      student_solution = {'solved' : 'solved'},
                      date_submitted = fake.date(),
                      grade = random.randint(0,10)
                      )
    db.session.add(journal)
    db.session.commit()



# https://stackoverflow.com/questions/67166839/how-to-show-a-table-with-tabulator-using-flask-and-a-json-variable
@app.route("/")
@app.route("/home")
@app.route("/index")
```

```
def home():
#    q = db.session.query(Student.name, Journal.id_journal).all()
    plain_sql=text("""
        select student.name, sum(journal.grade) as "sum"
        from student join journal
        on student.id_student = journal.id_student
        group by student.name order by student.name;
    """)
    with db.engine.connect() as connection:
        query = connection.execute(plain_sql)
        data=[{'name':i[0], 'grade':i[1]} for i in query]
        return render_template('index.html', data=data)
    # query = db.session.query(Student.name, func.sum(Journal.grade).label("sum of grades")).filter(Journal.grade > 0).filter(Journal.id_student==Student.id_student).group_by(Student.name,Journal.grade).all()
    # print(query)
    # data=[{'name':i[0], 'grade':i[1]} for i in query]


    # for i in tup:
    #     d.append({'name':i[0], 'grade':i[1]})


    # >>> print(db.session.query(func.sum(Journal.grade).label("sum of grades"), Student.name).filter(Journal.grade > 0).group_by(Student.name))
# SELECT sum(journal.grade) AS "sum of grades", student.name AS student_name
# FROM journal, student
# WHERE journal.grade > %(grade_1)s GROUP BY student.name
# >>> db.session.query(Student.name, func.sum(Journal.grade).label("sum of grades")).filter(Journal.grade > 0).group_by(Student.name).all()
```

```
    #
db.session.query(Order).filter(Order.id==Order_line.order_id).filter(Order_line.status_i
d!=1).group_by(Order.id).all()
    # db.session.query(Journal, Student).filter(Journal.grade > 0
    #                                 ).filter(Journal.id_student == Student.id_student
    #                                 ).group_by(Student.id_student)



    #   db.session.select(Journal, Student).filter(Journal.grade > 0
    #                                     ).filter(Journal.id_student == Student.id_student
    #                                         ).group_by(Student.id_student)

    # p
    # .all()
    # Journal.query.

@app.route("/about")
def about():
    return render_template('about.html', title='About')
if __name__ == '__main__':
    app.run(debug=True)
```

index.html — Сторінка з таблицею

```
<!DOCTYPE html>
<html lang="en">

 <head>
  <title>Рейтингова таблиця</title>
  <meta charset="utf-8"/>
  <link rel="shortcut icon" href="/favicon.ico" type="image/x-icon" />
  <!-- <link rel='stylesheet' type='text/css' href='/style.css'> -->
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="keywords" content="">
  <link href="../static/css/tabulator.css" rel="stylesheet">
 </head>
 <body>
    <header><h1>💪 Рейтингова таблиця 👌</h1></header>
    <div id="example-table"></div>
    <style>
     h1 {
         text-align: center ;
     }
    </style>
    <!-- This thing here is needed to sort column etries with the dates -->
    <script type="text/javascript" src="../static/js/luxon.js"></script>
    <script type="text/javascript" src="../static/js/tabulator.js"></script>
    <script type="text/javascript">
   //sample data
   var tabledata = {{ data | tojson }};

   var table = new Tabulator("#example-table", {
            //height:200, // set height of table to enable virtual DOM
```

```
        // better to set hight depending on device's screensize?
        // or not set it and let the user scroll...
            data:tabledata, //load initial data into table
            layout:"fitColumns", //fit columns to width of table (optional)
        columns:[ //Define Table Columns
                    {title:"Студент", field:"name", sorter:"string"},//, width:150},
                    {title:"Всього набрано балів", field:"grade", sorter:"integer",
            hozAlign:"left"},
            ],
    });
    </script>
  </body>
</html>
```

setup.yaml — Конфігурація Ansible

```
---
- hosts: localhost
  connection: local
  become: true
  vars:
  - interface: lxcbr0

  tasks:
  - name: apt lxc packages are installed on host
    apt:
      name:
      - tmux
      - lxc
      - lxc-dev
      - python3-pip
      state: present

  - copy:
      dest: /etc/default/lxc-net
      content: |
        USE_LXC_BRIDGE="true"

  - copy:
      dest: /etc/lxc/default.conf
      content: |
        lxc.net.0.type = veth
        lxc.net.0.link = {{ interface }}
        lxc.net.0.flags = up
        lxc.net.0.hwaddr = 00:16:3e:xx:xx:xx
```

```
  - service:
      name: lxc-net
      state: started

  - name: pip lxc packages are installed on host
    pip:
      name:
      - ansible-lint
      - yamllint
    run_once: true

- hosts: all
  connection: local
  become: true
  vars:
  - interface: lxcbr0
  tasks:
  - name: trying to clone one container t1
    delegate_to: localhost
    lxc_container:
      name: t1
      container_log: true
      template: debian
      state: started
      template_options: --release stretch
      container_config:
        - "lxc.net.0.type = veth"
        - "lxc.net.0.flags = up"
        - "lxc.net.0.link = {{ interface }}"
```

```
      - "lxc.net.0.ipv4.address = {{ ansible_host }}/24"
      - "lxc.net.0.ipv4.gateway = auto"
    container_command: |
      if [ ! -d ~/.ssh ]; then
        mkdir ~/.ssh
        echo "hello!" | tee -a ~/some_file
      fi
    clone_name: t1-clone2
#     clone_snapshot: true
  - name: trying to clone one container t1
    delegate_to: localhost
    lxc_container:
      name: t1-clone2
      state: started
# doing cool stuff - execute commands on container with shellscript
- hosts: all
  connection: local
  become: true
  vars:
  - interface: lxcbr0
  tasks:
  - name: Load in local SSH key path
    set_fact:
      my_ssh_key: "{{ lookup('env','HOME') }}/.ssh/id_rsa.pub"

  - name: set ssh key for the containers - it will be cool!!!!
    shell: |
      lxc-attach -n {{inventory_hostname}} --clear-env -e -- bash -c " \
                                              if [ ! -d ~/.ssh ]; then \
                                              mkdir ~/.ssh ; \
```

```
                                                    echo 'hello!' | tee -a ~/some_file ; \
                                                    echo \"{{ lookup('file', my_ssh_key) }}\" |
tee -a ~/.ssh/authorized_keys ; \
                                                    sed -i 's/dhcp/manual/' /etc/network/interfaces
&& systemctl restart network ; \
                                                    fi \
                                                    "
      # vm_name may not work
      # may want try to change to inventory_hostname or ansible_hostname
      delegate_to: localhost

    - hosts: all
      connection: local
      become: true
      vars:
      - interface: lxcbr0
      tasks:
      - name: Load in local SSH key path
        set_fact:
          my_ssh_key: "{{ lookup('env','HOME') }}/.ssh/id_rsa.pub"

      - name: interface device exists
        command: ip addr show {{ interface }}
        changed_when: false
        run_once: true

      - name: Local user has an SSH key
        command: stat {{ my_ssh_key }}
        changed_when: false
        run_once: true
```

```
  - name: containers exist and have local SSH key
    delegate_to: localhost
    lxc_container:
      name: "{{ inventory_hostname }}"
      container_log: true
      template: debian
      state: started
      template_options: --release stretch
      container_config:
        - "lxc.net.0.type = veth"
        - "lxc.net.0.flags = up"
        - "lxc.net.0.link = {{ interface }}"
        - "lxc.net.0.ipv4.address = {{ ansible_host }}/24"
        - "lxc.net.0.ipv4.gateway = auto"
      container_command: |
        if [ ! -d ~/.ssh ]; then
          mkdir ~/.ssh
          echo "{{ lookup('file', my_ssh_key) }}" | tee -a ~/.ssh/authorized_keys
          sed -i 's/dhcp/manual/' /etc/network/interfaces && systemctl restart network
        fi


- hosts: all
  connection: local
  become: false
  serial: 1
  tasks:
  - wait_for: host={{ ansible_host }} port=22

  - name: container key is up-to-date locally
```

```
        shell: ssh-keygen -R {{ ansible_host }}; ssh-keygen -R
{{ inventory_hostname }}; (ssh-keyscan {{ ansible_host }} {{ inventory_hostname }}
>> ~/.ssh/known_hosts)

    - hosts: all
      gather_facts: no
      vars:
      - ansible_user: root
      tasks:
      - name: install python on target machines
        raw: which python3 || (apt-get -y update && apt-get install -y python3)

    # Retrieval of local and remote file content
    # blog: https://fabianlee.org/2021/05/25/ansible-creating-a-variable-from-a-
remote-or-local-file-content/
    ---

    - hosts: all
      gather_facts: yes

      vars:
        local_path: local-README.md
        remote_path: /tmp/remote-README.md

      tasks:

        - name: get content of local file
          set_fact:
            readme_contents: "{{ lookup('file',playbook_dir + '/' + local_path) }}"
```

```
    - debug:
        msg: "content of local file {{local_path}}: {{readme_contents}}"


    - name: create emulated 'remote' file
      delegate_to: localhost
      copy:
        dest: "{{remote_path}}"
        mode: '0666'
        content: |
          {"json":"is", "beautiful":42, "obfuscated":"code}

    - name: get content of remote file
      slurp:
        src: "{{remote_path}}"
      register: remote_content_encoded
    - name: decode remote content
      set_fact:
        remote_content: "{{remote_content_encoded.content | b64decode}}"

    - debug:
        msg: "content of remote file {{remote_path}}: {{remote_content}}"
  # useful, but not here https://stackoverflow.com/questions/43096404/ansible-use-of-diff-command-using-ansible/43096880#43096880
  # https://stackoverflow.com/questions/26638180/write-variable-to-a-file-in-ansible
  # now write down the content of remote into the
    # destination path is relative to the folder where setup.yml is located
    - copy: content="{{ remote_path }}" dest=task1.json
```

requirements.txt (необхідні Python бібліотеки для роботи системи)

(імпорт з допомогою “”)

Faker==13.13.0

Flask==2.1.2

Flask-SQLAlchemy==2.5.1

greenlet==1.1.2

importlib-metadata==4.11.4

itsdangerous==2.1.2

Jinja2==3.1.2

MarkupSafe==2.1.1

numpy==1.22.4

psycopg2==2.9.3

python-dateutil==2.8.2

six==1.16.0

SQLAlchemy==1.4.37

SQLAlchemy-serializer==1.4.1

Werkzeug==2.1.2

zipp==3.8.0

ova2rootfs.sh (перетворення OVA на rootfs)

```
#!/bin/bash
# ---------part 1---------
# -------OVA > IMG--------
# takes .ova file as input
tar tvf "$1"  # listing contents of the ova
# extracting vmdk from ova
tar xvf "$1" --wildcards *vmdk
# convert vmdk into img (KVM)
# put new names into array for later reference
# typedef -a images
# for i in vmdk; do
# images+="${i/vmdk/img}"
imgname="${i/vmdk/img}"
qemu-img -p convert -f vmdk -O raw "$i" "$imgname"
# done
# ---------part 2---------
# -------IMG > ROOTFS-----
# create lxc image with the name of .ova file
lxc-create --name "${1/.ova/}"
# get the start block of an img
start_block=$(sudo fdisk -l "$imgname" | grep ^`realpath "$imgname"` |  awk -F"
"  '{ print $3 }')
# get the sector size of an img
sector_size=$(sudo fdisk -l "$imgname" | grep ^Sector |  awk -F" "  '{ print $4 }')
# mount image; may be buggy, do check
mount -o loop,offset=$(($start_block*$sector_size)) "$imgname" /mnt-source
# sync image filesystem contents to rootfs
rsync -av /mnt-source/* /var/lib/lxc/"${1/.ova/}"/rootfs
```

```
# create device nodes
ROOT=/var/lib/lxc/"${1/.ova/}"/rootfs
DEV=${ROOT}/dev
mv "${DEV}" "${DEV}".old
mkdir -p "${DEV}"
mknod -m 666 "${DEV}"/null c 1 3
mknod -m 666 "${DEV}"/zero c 1 5
mknod -m 666 "${DEV}"/random c 1 8
mknod -m 666 "${DEV}"/urandom c 1 9
mkdir -m 755 "${DEV}"/pts
mkdir -m 1777 "${DEV}"/shm
mknod -m 666 "${DEV}"/tty c 5 0
mknod -m 600 "${DEV}"/console c 5 1
mknod -m 666 "${DEV}"/tty0 c 4 0
mknod -m 666 "${DEV}"/tty1 c 4 1
mknod -m 666 "${DEV}"/tty2 c 4 2
mknod -m 666 "${DEV}"/tty3 c 4 3
mknod -m 666 "${DEV}"/tty4 c 4 4
mknod -m 666 "${DEV}"/tty5 c 4 5
mknod -m 666 "${DEV}"/tty6 c 4 6
mknod -m 666 "${DEV}"/full c 1 7
mknod -m 600 "${DEV}"/initctl p
mknod -m 666 "${DEV}"/ptmx c 5 2
# create LXC config
# Template used to create this container: (null)
# Parameters passed to the template:
cat > /var/lib/lxc/"${1/.ova/}"/config <<EOF
lxc.mount.entry = proc proc proc nodev,noexec,nosuid 0 0
lxc.mount.entry = sysfs sys sysfs defaults  0 0
```

```
lxc.tty = 2
lxc.pts = 1024
lxc.cgroup.devices.deny = a
lxc.cgroup.devices.allow = c 1:3 rwm
lxc.cgroup.devices.allow = c 1:5 rwm
lxc.cgroup.devices.allow = c 5:1 rwm
lxc.cgroup.devices.allow = c 5:0 rwm
lxc.cgroup.devices.allow = c 4:0 rwm
lxc.cgroup.devices.allow = c 4:1 rwm
lxc.cgroup.devices.allow = c 1:9 rwm
lxc.cgroup.devices.allow = c 1:8 rwm
lxc.cgroup.devices.allow = c 136:* rwm
lxc.cgroup.devices.allow = c 5:2 rwm
lxc.cgroup.devices.allow = c 254:0 rm
lxc.utsname = ${1/.ova/}
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = virbr0
lxc.network.hwaddr = 00:16:3e:f1:35:ab
lxc.network.ipv4.gateway = 192.168.122.1
lxc.network.ipv4 = 192.168.122.15
lxc.cap.drop = sys_module
lxc.cap.drop = mac_admin
lxc.cap.drop = mac_override
lxc.cap.drop = sys_time
lxc.rootfs = $ROOT
EOF
# don't forget to provide an unique MAC address (hwaddr) to each container.
# start the container
lxc-start --name ${1/.ova/} -d
```

Додаток Г.

(обов’язковий)

ГРАФІЧНА ЧАСТИНА

РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ ДЛЯ ВИВЧЕННЯ ВІДКРИТИХ ОПЕРАЦІЙНИХ СИСТЕМ НА ОСНОВІ ХМАРНИХ ТЕХНОЛОГІЙ ІНСТРУМЕНТАМИ GNU/LINUX

(Назва бакалаврської кваліфікаційної роботи)

Додаток Г – Ілюстративна частина

Рисунок Г.1 – Назва роботи

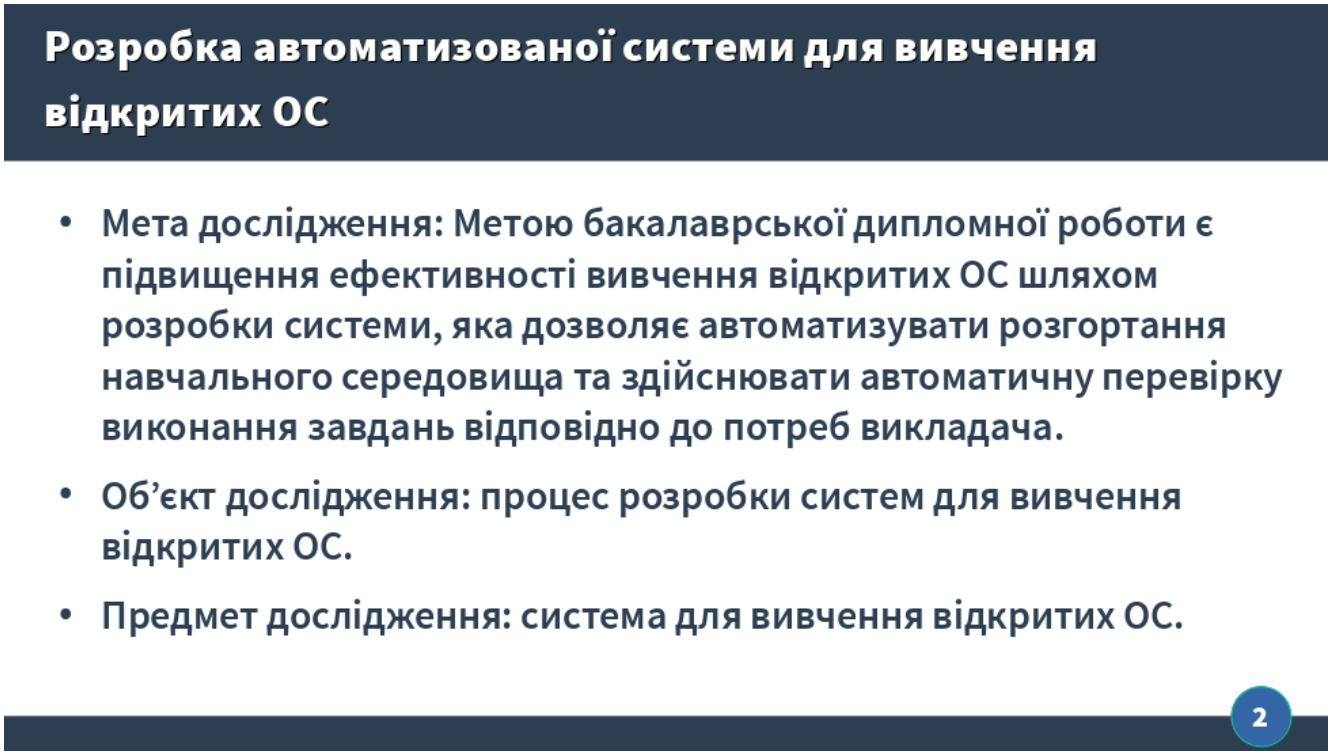

Рисунок Г.2 – Мета, об’єкт і предмет дослідження

Розробка автоматизованої системи для вивчення відкритих ОС

Наукова новизна:

- Подальшого розвитку отримав метод автоматизації перевірки виконання завдань студентів, який, на відміну від існуючих, використовує протокол безпечного віддаленого з'єднання SSH, щоб отримувати доступ до контейнерів студентів, що дозволяє швидко та ефективно перевіряти завдання студентів, а також одразу відображати їхні успіхи у рейтинговій таблиці.
- Подальшого розвитку отримала модель системи вивчення відкритих ОС, яка, на відміну від існуючих, має більший функціонал та є гнучкою у конфігуруванні – її можна розгортати як на одному сервері, так і на багатьох, оскільки вона має клієнт-серверну архітектуру, тому може бути пристосованою до будь-якого типу курсу, від базового до складного, що дозволяє розширити можливості застосування системи в навчальному процесі.

Практична цінність:

- практична цінність полягає у можливості використання розробленої системи вивчення відкритих ОС навчальними установами, групами ентузіастів чи розробниками комерційних курсів.

Рисунок Г.3 – Наукова новизна та практична цінність

Задачі бакалаврської дипломної роботи

- Розробити моделі системи вивчення відкритих ОС
- Розробити метод перевірки виконання завдань
- Розробити базу даних для збереження виконаних завдань
- Розробити рейтингову таблицю
- Розробити формат навчального середовища (для його реплікації)
- Розробити шар сумісності (перетворення форматів) між віртуальними машинами та контейнерами
- Розробити інструкцію користувача
- Провести тестування розробленої системи

Рисунок Г.4 – Структура бакалаврської дипломної роботи

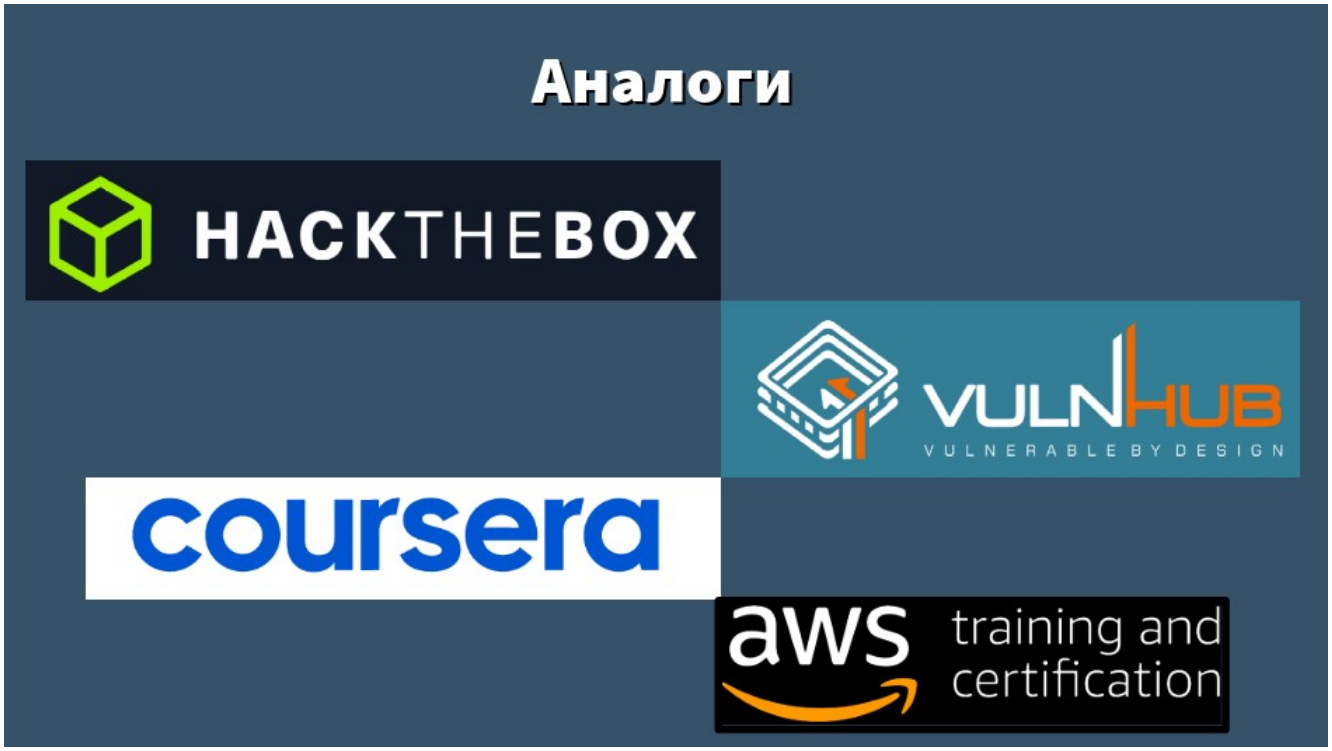

Рисунок Г.5 – Аналоги

Порівняння аналогів

Критерії	HTB	Vulnhub VM	AWS Academy	Coursera	Click
Працює оффлайн	-	+	-	-	+
Можна встановити на власну інфраструктуру	-	+	-	-	+
Адаптується до різних предметних областей	-	-	-	+	+
Можливість SSH доступу	+	-	+	-	+
Дозволяє створювати власні VM	-	-	-	-	+
Загальна оцінка	**20%**	**40%**	**20%**	**20%**	**100%**

Рисунок Г.6 – Порівняння аналогів

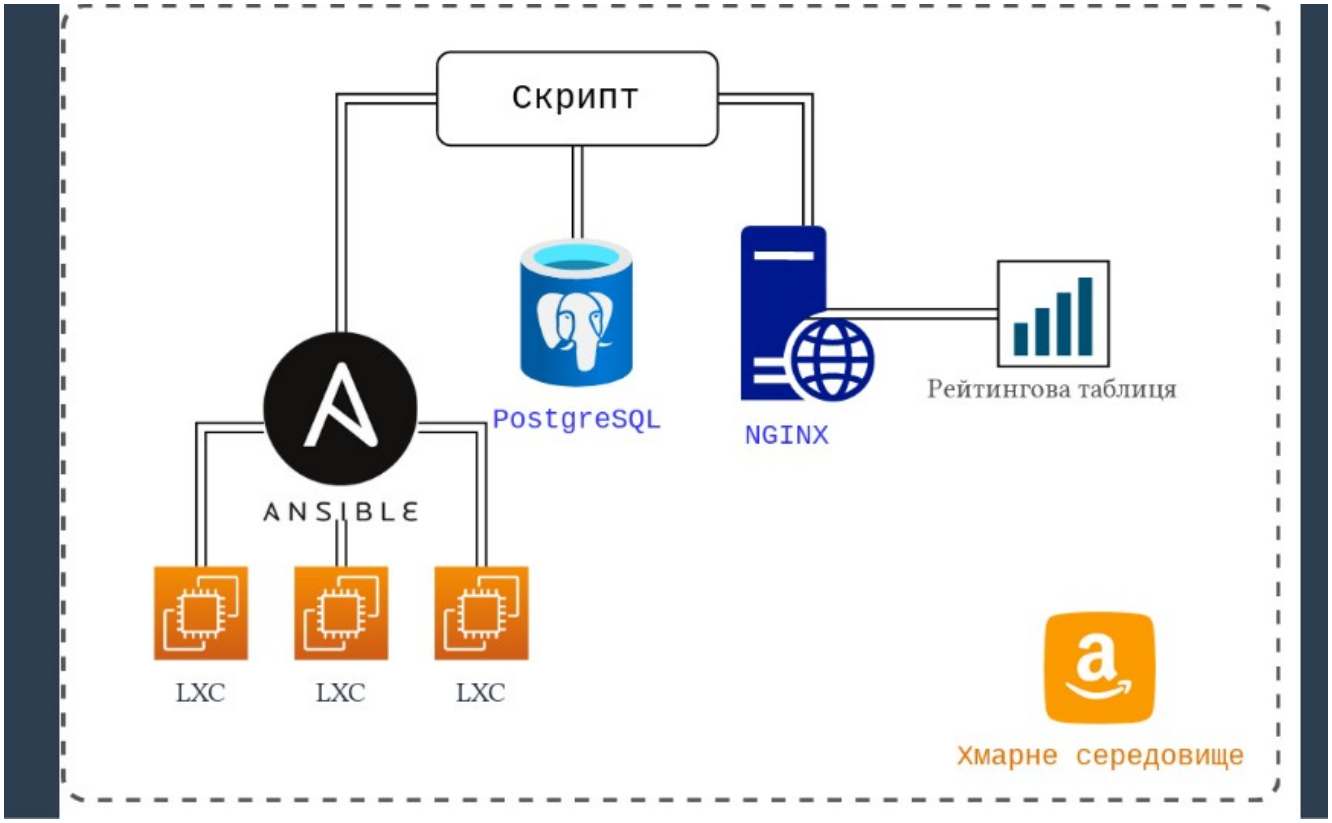

Рисунок Г.7 – Модель системи “зсередини”

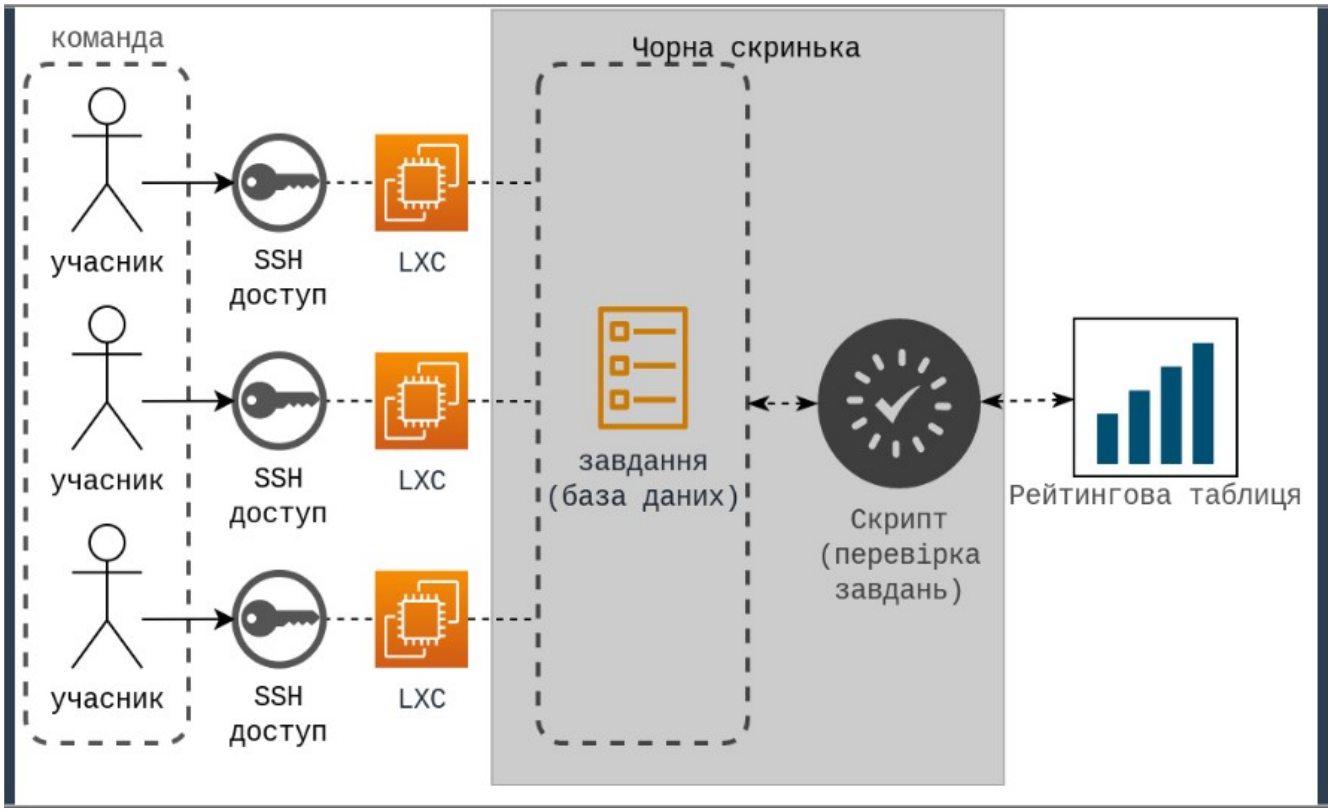

Рисунок Г.8 – Модель системи з позиції студента

```yaml
---
- hosts: localhost
  connection: local
  become: true
  vars:
  - interface: lxcbr0

  tasks:
  - name: apt lxc packages are installed on host
    apt:
      name:
      - tmux
      - mdp
      - lxc
      - lxc-dev
      - python3-pip
      state: present

  - copy:
      dest: /etc/default/lxc-net
      content: |
        USE_LXC_BRIDGE="true"

  - copy:
      dest: /etc/lxc/default.conf
      content: |
        lxc.net.0.type = veth
        lxc.net.0.link = {{ interface }}
        lxc.net.0.flags = up
        lxc.net.0.hwaddr = 00:16:3e:xx:xx:xx

  - service:
      name: lxc-net
      state: started

  - name: pip lxc packages are installed on host
    pip:
      name:
      - ansible-lint
      - yamllint
    run_once: true

- hosts: all
```

Файли конфігурації!

Рисунок Г.9 – Модель системи з позиції викладача

Рисунок Г.10 – Метод перевірки завдань

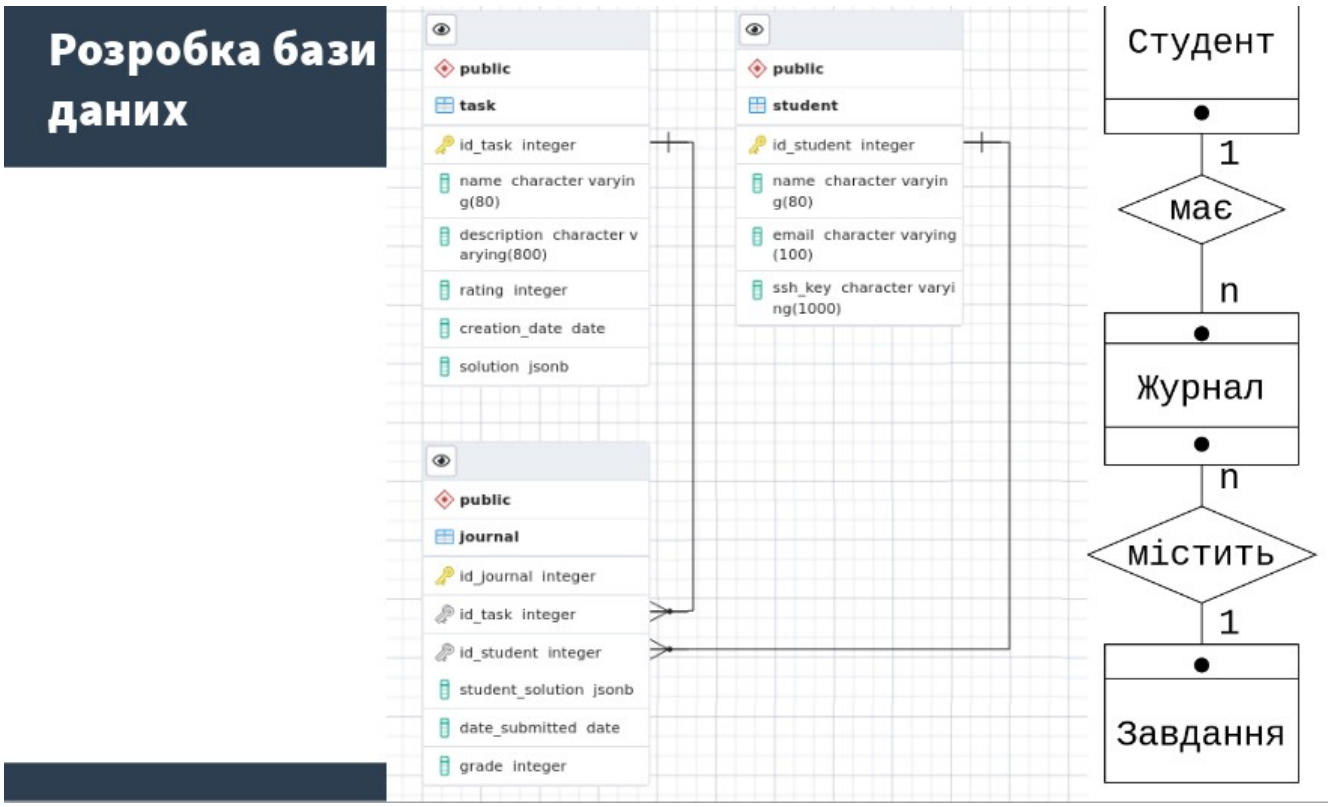

Рисунок Г.11 – Розробка бази даних

Студент	Всього набрано балів
Ада Гупало	51
Веніямин Масоха	48
Дем'ян Стельмах	47
пан Григорій Бабій	44
Федір Шутько	30
Роксолана Гайденко	29
Давид Дробаха	22
пані Мар'яна Даниленко	22
Леон Товстоліс	17
пані Василина Нестеренко	12

Рисунок Г.12 – Розробка рейтингової таблиці

Тестування бази даних з допомогою Python

```python
def add_student():
    student = Student(name = fake.name(),
                      email = fake.email(),
                      ssh_key = uuid.uuid4().hex
                      )
    db.session.add(student)
    db.session.commit()

def add_task():
    task = Task(name = fake.last_name(),
                description = "describing stuff",
                rating = random.randint(1,10),
                creation_date = fake.date(),
                solution = {'solved' : 'solved'}
                )
    db.session.add(task)
    db.session.commit()

def add_journal():
    # a list of existing student and task ids
    stud_ids = [id[0] for id in Student.query.with_entities(Student.id_student).all()]
    task_ids = [id[0] for id in Task.query.with_entities(Task.id_task).all()]
    # choose random id out of stud/task ids
    journal = Journal(id_task = random.choice(task_ids),
                      id_student = random.choice(stud_ids),
                      student_solution = {'solved' : 'solved'},
                      date_submitted = fake.date(),
                      grade = random.randint(0,10)
                      )
    db.session.add(journal)
    db.session.commit()
```

Рисунок Г.13 – Тестування бази даних з допомогою Python

```
clickDB=> select * from  student;
 id_student |           name           |                email                 |             ssh_key
------------+--------------------------+--------------------------------------+----------------------------------
          1 | пані Мар'яна Даниленко   | bvashchenko-zakharchenko@example.net | 0ecafc4a461548fb94d7dfe2ff9632bf
          2 | пан Григорій Бабій       | dannaskyba@example.org               | 84ea7d6fc974452eaecf63cf3f40906d
          3 | Леон Товстоліс           | ivelychko@example.org                | 63331b2c18514d4da9ddc0494ac481e4
          4 | Федір Шутько             | zlatoslava46@example.com             | eff981b773f74c66836e202008de5fb5
          5 | Роксолана Гайденко       | matiashvadym@example.net             | c5f437b6bb264df49c2f48e46d091b9d
          6 | Венямин Масоха           | veniiamyn38@example.org              | 406a47594c964c04b42f20be5f41f03d
          7 | Давид Дробаха            | arsen52@example.org                  | 982e6093aeb6491fabd56c0433951a55
          8 | пані Василина Нестеренко | volodymyrtereshchenko@example.com    | c0afb77f9983414daca5016c1ea4c00e
          9 | Дем'ян Стельмах          | pavlychenkosnizhana@example.com      | 118354d18d694219889867e873782531
         10 | Ада Гупало               | wvasylechko@example.com              | f94070ae79e543dda3f64c811514a97c
(10 rows)
```

Рисунок Г.14 – Результат тестування таблиці Студент

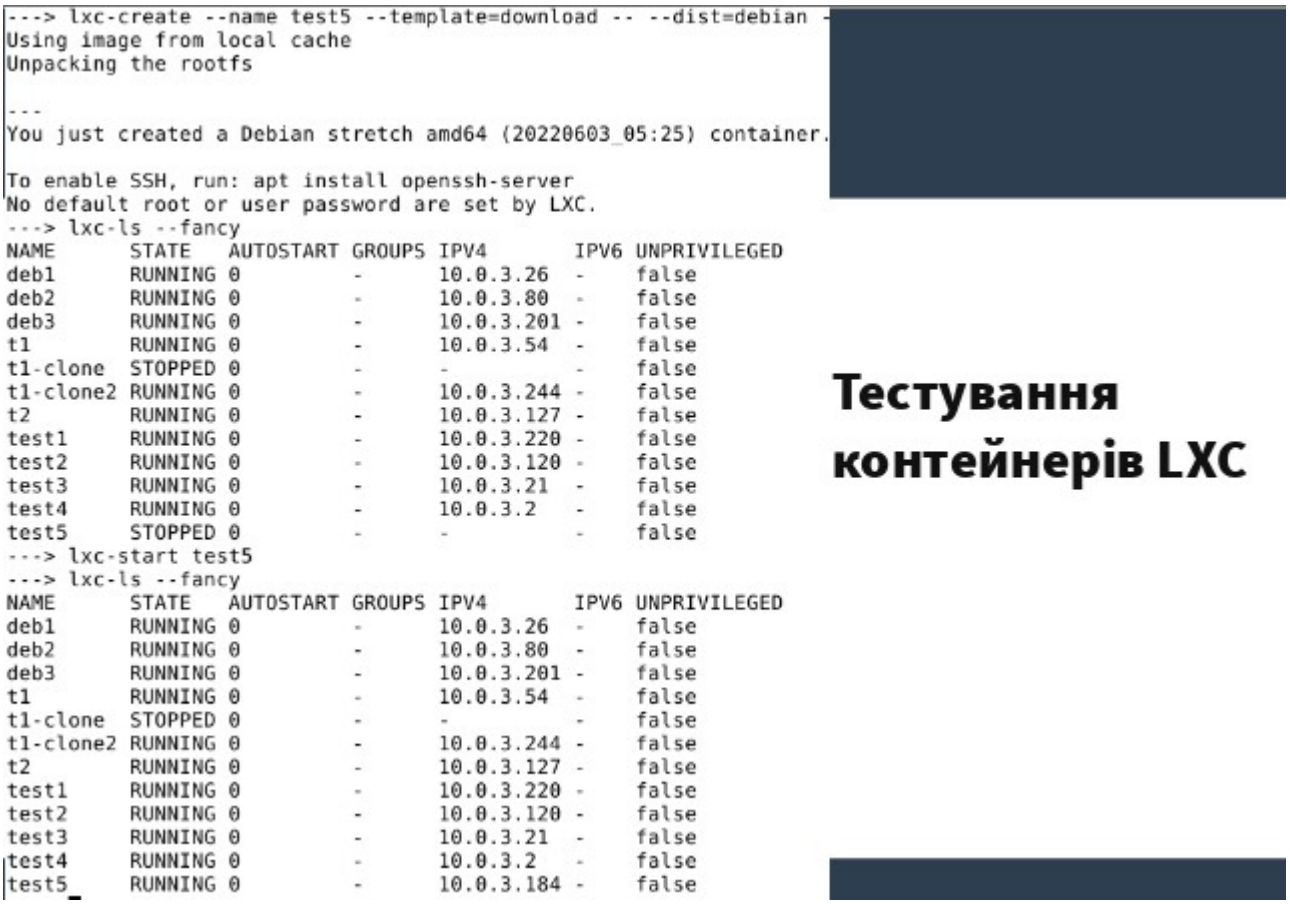

Рисунок Г.15 – Тестування контейнерів LXC

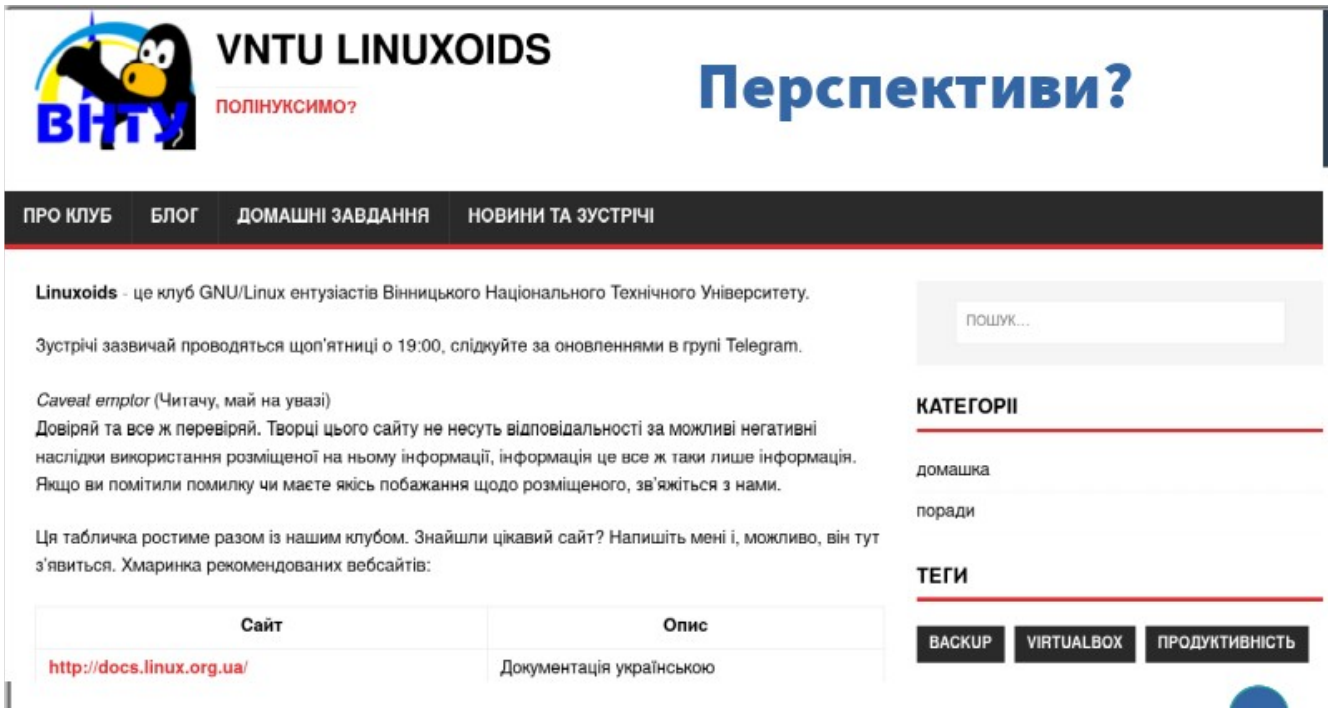

Рисунок Г.16 – Перспективи розвитку системи

Висновки

У бакалаврській дипломній роботі було виконано такі задачі:

- розроблено моделі системи вивчення відкритих ОС
- розроблено метод перевірки виконання завдань
- розроблено формат для розгортання навчального середовища
- розроблено базу даних для збереження завдань
- розроблено рейтингову таблицю
- розроблено формат навчального середовища (для його реплікації)
- розроблено шар сумісності (перетворення форматів) між віртуальними машинами та контейнерами
- розроблено інструкцію користувача
- проведено тестування системи вивчення відкритих ОС

Рисунок Г.17 – Висновки

Апробація та публікації

- Ісаков А.В. Сфери застосування нереляційних баз даних / А.В. Ісаков, В.Ю. Ліміна, О.В. Романюк // Матеріали XLIX Науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії (2020), Секція програмного забезпечення. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2020/paper/download/9416/7928.
- Ісаков А.В. Markdown language as an essential tool for digital writing / А.В. Ісаков, В.Г. Дерун // Матеріали XLIX Науково-технічної конференції Інституту соціально-гуманітарних наук (2020), Секція англійської мови. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-hum/all-hum-2020/paper/download/8806/7444.
- Ісаков А.В. Розробка інфраструктури для вивчення відкритих операційних систем на основі хмарних технологій інструментами GNU/Linux / А.В. Ісаков // Матеріали LI Науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії (2022), Секція програмного забезпечення. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/download/16149/13556.

Рисунок Г.18 – Апробації та публікації