МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

М.М. Букасов, Д.О. Галушко, П.Ю. Катін, Я.В. Хіцко

ІНФРАСТРУКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ WEB-ЗАСТОСУВАНЬ

Лабораторний практикум

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як навчальний посібник для здобувачів ступеня бакалавра студентів Інституту післядипломної освіти Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського». Може бути використаний для студентів денної та заочної форми навчання за освітніми програмами «Інтегровані інформаційні системи», «Інформаційні управляючі системи та технології», «Інформаційне забезпечення робототехнічних систем», «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»
спеціальностей
121 Інженерія програмного забезпечення, 126 Інформаційні системи та технології

Київ
КПІ ім. Ігоря Сікорського
2023

Рецензенти — Волокита А.М., канд. техн. наук, доц. кафедри обчислювальної техніки ФІОТ, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

Боярінова Ю.Є., канд. техн. наук, доц. кафедри спеціалізованих комп'ютерних систем і системного програмування ФПМ, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

Відповідальний редактор — Катін П.Ю., канд. техн. наук, доцент

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 5 від 23.02.2023 р.) за поданням Вченої ради факультету інформатики та обчислювальної техніки (протокол № 8 від 30.01.2023 р.)

Електронне мережне навчальне видання

Автори:

Букасов Максим Михайлович, канд. техн. наук,
Галушко Дмитро Олександрович, канд. тех. наук,
Катін Павло Юрійович, канд. техн. наук, доцент.
Хіцко Яна Володимирівна, канд. техн. наук.

ІНФРАСТРУКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ WEB-ЗАСТОСУВАНЬ

Інфраструктура програмного забезпечення WEB-застосувань. Лабораторний практикум [Електронний ресурс]: навч. посіб. для студ. спеціальностей 121 «Інженерія програмного забезпечення», 126 «Інформаційні системи та технології» / КПІ ім. Ігоря Сікорського; автор.: М.М. Букасов, Д.О. Галушко, П.Ю. Катін, Я.В. Хіцко. – Електронні текстові дані (1 файл: 1,1 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2023. – 53 с.

В навчальному посібнику наведені завдання на дослідження та визначений порядок документування результатів. Основне призначення навчального посібника є самостійна робота здобувачів вищої освіти та дистанційного навчання. Матеріали навчального посібника можуть бути використані для проведення лабораторних робіт студентів денної і заочної форми навчання. Також посібник містить додаткову інформацію для виконання досліджень.

Для студентів, які навчаються за спеціальністю 121 «Інженерія програмного забезпечення» факультету прикладної математики, 126 «Інформаційні системи та технології» факультету інформатики та обчислювальної техніки та Інституту післядипломної освіти Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

ЗМІСТ

ПЕРЕДМОВА

Основна задача даного посібника полягає у наданні завдань та методичних рекомендацій для самостійної роботи здобувачів вищої освіти та дистанційного навчання. Незамінна складова матеріалу отримана з досвіду практичної роботи Галушко Д.О., Букасова М.М. і Хіцко Я.В.. У навчальний посібник також включена інформація, що отримана на досвіді роботи Катіна П.Ю. під час викладання і організації курсу «Інфраструктура програмного забезпечення WEB-застосувань» у Інституті післядипломної освіти Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

Практика роботи показала, що даний курс зацікавив студентів бакалаврату старшого курсу денної і заочної форми навчання факультету інформатики та обчислюваної техніки і факультету прикладної математики.

Всі завдання і рекомендації розроблені з урахуванням практичних рішень і спрямовані для вдосконалення знань і навичок досвідчених студентів і початківців у цій галузі.

Завдання, що вирішуються у ЛР сприяють розумінню предмета, дають можливість подальших досліджень для підготовки до магістерського рівня і можуть бути корисними для роботи у аспірантурі.

Для роботи з курсом «Інфраструктура програмного забезпечення WEB-застосувань» студент має успішно опанувати курси: основ програмування, вищої математики, англійської мови.

ВСТУП

Особливістю даного навчального посібника є те, що він може використовуватися для фінального етапу підготовки бакалаврів по трьом спеціальностям, а саме: 121 – Інженерія програмного забезпечення, 126 – Інформаційні системи та технології. Це обумовлено тим, що технологія, яка вивчається, може бути використана для вирішення різних завдань у контексті побудови розподіленої програмної системи. Наприклад, сучасна телекомунікація і радіотехніка містить програмно-апаратні комплекси, що забезпечуються складним розподіленим програмним забезпеченням і передбачають певну інфраструктуру.

Потрібно відмітити, що матеріал навчального посібника розгалужується з напрямком створення ІТ-інфраструктури підприємства і аналогічних складних глобальних рішень. У курсі не розглядаються питання апаратних елементів складних розподілених програмних систем. Дослідження у рамках ЛР даного посібника провадяться виключно у межах відносно простих розподілених програмних систем, що побудовані на різних фреймворках і з різною архітектурою. Посібник містить перелік основних джерел [1], та перелік додаткових джерел [1-28д.], що можуть бути корисними у процесі досліджень.

Питанням створення і дослідженням інфраструктури для розподілених програмних систем завжди приділялося багато уваги. Прикладом таких досліджень є джерело [1д.], що передбачає процес системного аналізу і дизайну під час розробки складних програмних систем. У [1д.], на первинному етапі системного аналізу, разом з бізнес аналізом, визначенням технічних умов, програмною архітектурою, велика увага приділялася оцінюванню і розробці інфраструктури для розподілених програмних систем (РПС).

Звичайно, що у [1д.] інфраструктура РПС розглядалася виключно як апаратне рішення, і це – застарілий підхід. На теперішній час програмна

інфраструктура РПС може бути побудована з використанням різних програмних технологій, що мають три базових етапу розвитку.

Першим етапом розвитку таких технологій є віртуальні машини, до яких можна віднести: VMware Workstation (Microsoft Windows, Linux), VMware Fusion (macOS); Microsoft Hyper-V; Oracle VM VirtualBox; QEMU (Quick Emulator).

Наступним етапом розвитку технологій, є системи контейнеризації (СК). Вони доволі часто побудовані на основі програмного забезпечення (ПЗ) з відкритим вихідним кодом. Системи контейнеризації мають структуру певних контейнерів, що є меншим обсягом за віртуальні машини, дуже добре підлягають процесу автоматизації управління, розповсюдження і створення. Ці контейнери можуть працювати у хмарних системах або на локальному ПК. Прикладом таких систем є Docker та інші подібні системи, що активно розвиваються на теперішній час.

Найбільш сучасним етапом розвитку технологій створення інфраструктури РПС на базі СК є системи оркестрації контейнерів. Звичайно ці системи являють собою набір API для автоматизації і управління РПС, що побудована на базі контейнерів. Найбільш часто вони реалізують масштабування або управління декількома контейнерами у РПС для вирішення практичних завдань з метою управління навантаженням, забезпечення надійності РПС та її енергозбереження. Прикладом таких технологій є Kubernetes.

У навчальному посібнику уміщено завдання для лабораторних робіт (ЛР), рекомендації щодо їх виконання та порядку оформлення і аналізу отриманих результатів. Рівень дослідження в посібнику орієнтований на виконання студентами старших курсів бакалаврату.

Лабораторні роботи № 1–3 є базовими для обох спеціальностей (121, 126) і можуть виконуватися без змін. Цей блок лабораторних робіт є обов'язковим

для отримання позитивної оцінки і дає можливість набрати 60 балів. Лабораторні роботи 4, 5 мають два варіанта виконання.

Лабораторні роботи № 4.2–5.2 відокремлюються відповідно напрямку дослідження спеціальності 126. Для спеціальності 121 у лабораторних роботах № 4.1–5.1 досліджується процес створення і тестування розподіленого web-застосунку. Цей web-застосунок може бути основою РПС для управління радіотехнічними системами або використаний для вирішення задач прикладної математики.

Підготовлені студенти можуть, по узгодженню з викладачем, поєднувати лабораторні роботи або відокремлювати їх частини, дотримуючись загального завдання. У разі успішного захисту дається можливість отримати відповідну кількість балів без необхідності захищати лабораторні роботи відповідно їх розподілу у навчальному посібнику.

Навчальний посібник може бути корисним також для магістрантів і аспірантів, що працюють у галузі створення РПС. Більша частина теоретичного матеріалу, програмного коду, рисунків, що наведені у посібнику, є авторськими.

Виконання лабораторних робіт у даному курсі реалізується на основі СК Docker. Дозволяється, по узгодженню з викладачем, після успішного виконання ЛР №1-3, використовувати інші СК для використання курсу лабораторних робіт. Це передбачено для підтримки дослідження студентів, що працюють професійно на інших СК, мають практичний досвід або провадять самостійні дослідження.

Вибір системи контейнеризації Docker для даного посібника обумовлений її умовною безкоштовністю і широким розповсюдженням у професійній розробці. Опанування СК та вироблення навичок роботи з нею, є основою для роботи над бакалаврським проектом.

Основою матеріалу для навчального посібника є результати практичної і професійної діяльності і досліджень авторів-викладачів. У навчальний посібник

додані результати багаторічної професійної діяльності з створення РПС, інженерами з розробки програмного забезпечення, викладачами і науковцями к.т.н. Галушко Д.О. і Букасова М.М.

Основою посібника є курс дисципліни «Інфраструктура програмного забезпечення», що читалася на ФПМ студентам бакалаврату під керівництвом к.т.н. Катіна П.Ю.

Також незамінною складовою посібника є професійний досвід створення складних програмних систем к.т.н. Хіцко Я.В. Автором лабораторних робіт № 1 є Галушко Д.О. Завдання і хід дослідження ЛР № 3 розроблені Катіним П.Ю., теоретичні відомості укладені Хіцко Я.В. Автором лабораторних робіт ЛР № 4.1, 5.1, вступу, передмови, додатку 1, порядку оформлення робіт є Катін П.Ю. Лабораторні роботи 2, 4.2, 5.2 розроблені Букасовим М.М.

Загальна редакція Катіна П.Ю. Редакція ЛР 1, 2, 4.2, 5.2 Галушко Д.О.

ПРОГРАМНІ ІНСТРУМЕНТИ ДЛЯ ЛАБОРАТОРНИХ РОБІТ

Курс лабораторних робіт бажано реалізувати на операційних системах родини Microsoft або на POSIX сумісних (сертифікованих) операційних системах. Дозволяється проводити дослідження на всіх операційних системах, що підтримують технології СК Docker та технології мікросервісної архітектури і web-розробки.

Лабораторні роботи можна реалізовувати з текстовим або графічним інтерфейсом користувача. Дозволяється використовують інтегровані середовища розробки (Integrated Development Environment (IDE)) та інші допоміжні програмні засоби.

Автори, зважаючи на те, що навчальний посібник призначений для старшого курсу бакалаврату, жодним чином не обмежують студентів у виборі технології реалізації досліджень.

ЛАБОРАТОРНА РОБОТА № 1. Дослідження системи контейнерів Docker

Мета роботи: полягає у дослідженні специфіки запуску Docker контейнерів, ознайомленні з репозиторієм Docker Hub та, за потреби, Docker Desktop. Навчитися прокидати порти з гостьової на хостову машини, що дасть змогу працювати з власним веб-сервером nginx.

Вхідні дані ЛР1

У якості вхідних даних для ЛР1 є:

– за використання Windows: встановлений Docker Desktop або віртуальну машину Linux зі встановленим Docker;

– за використання Linux: встановлений Docker.

Додатково слід пам’ятати що в цій лабораторній роботі:

* – без символів “<>”;

** – прізвища AAA=1+1+1=3=8003, а ZYXZ=26+25+24+26=101=8101.

Вихідні дані ЛР1

У якості вихідних даних для ЛР3 є: робочий контейнер з веб-сервером nginx, а контент його сторінок можна переглянути з хостової машини.

Завдання

1. Навчитися піднімати контейнери Docker.
2. Навчитися прокидати порти з гостьової на хостову машини.
3. Навчитися працювати з вебсервером nginx.

Програма проведення експерименту ЛР 1

1. На базі alpine/nginx створити власний image NAME=LAB01_1BR<перші літери прізвищ кожного з членів бригади>*. Команди і результат роботи системи показаний на рис.1.

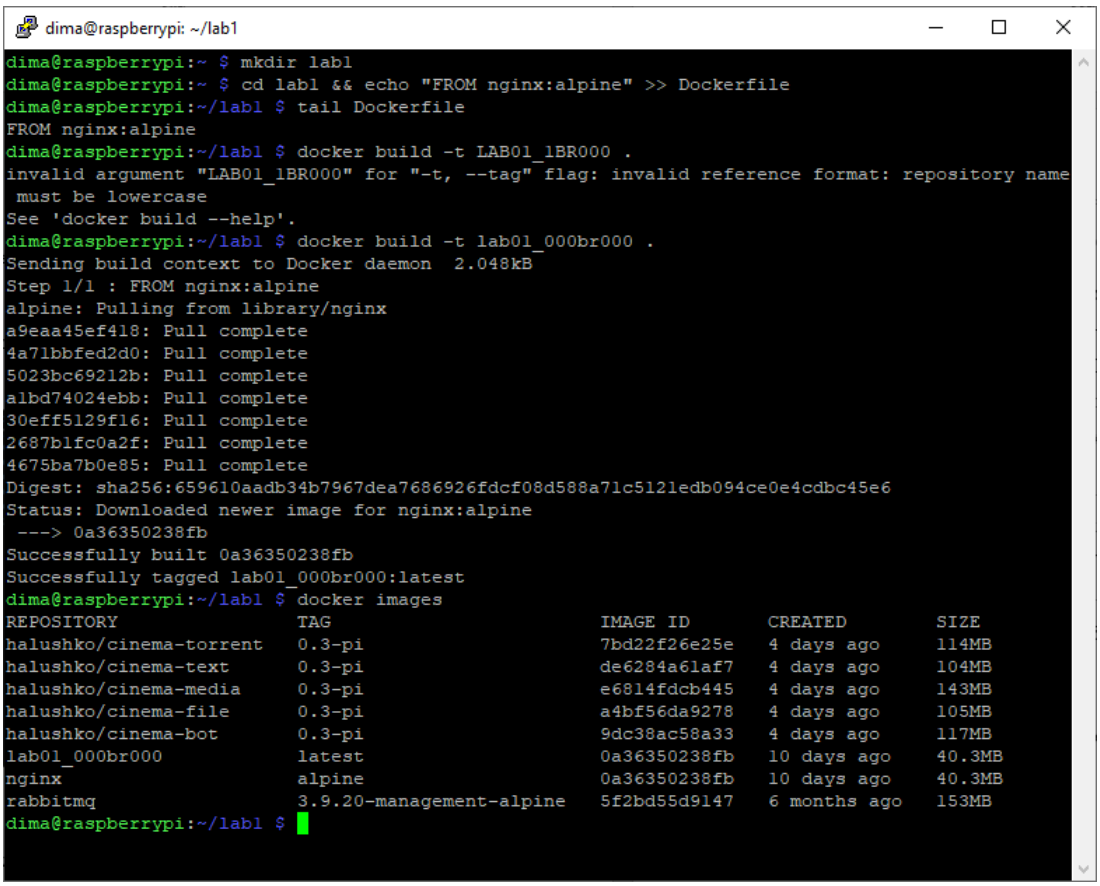

Рисунок 1 – Створення власного образу

2. Користуючись командою docker images, вивести перелік образів (images). Результат зафіксувати у вигляді скриншоту.

3. Запустити контейнер, вивести інформацію по активним контейнерам. Приклад команд і результати роботи показані на рис.2. На цьому рисунку показані приклад запуску контейнеру з лабораторної роботи і результат її виконання.

Після отримання результату, потрібно зафіксувати його у вигляді скриншоту. Результат додати до звіту.

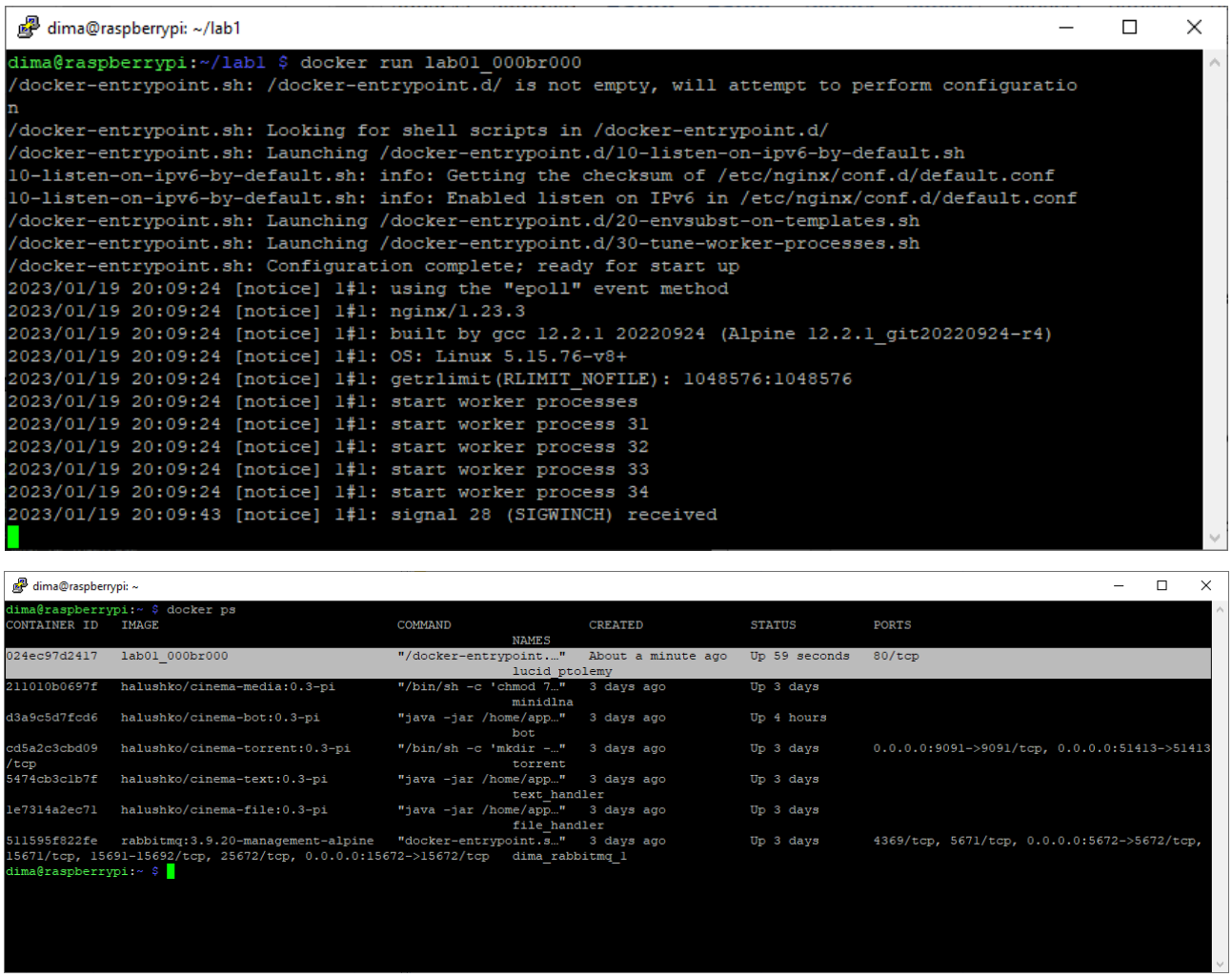

Рисунок 2 – Приклад запуску контейнерів

4. Зупинити контейнер, користуючись командою, що показана на рис.3. Вивести інформацію по активним контейнерам. Після отримання результату, потрібно зафіксувати його у вигляді скриншоту. Результат додати до звіту.

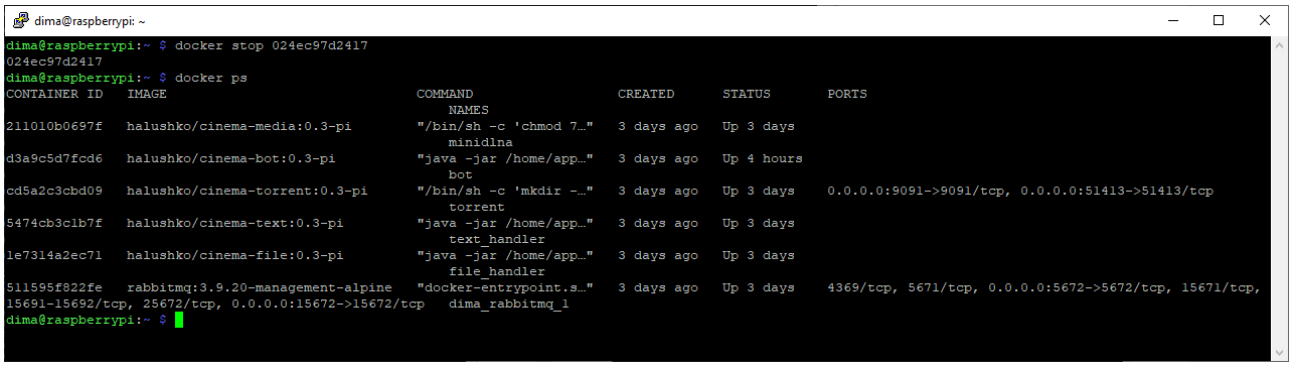

Рисунок 3 – Приклад зупинки контейнеру

5. Вивести інформацію по усім контейнерам (ключ -a). Після отримання результату, потрібно зафіксувати його у вигляді скриншоту. Результат додати до звіту. Сформувати звіт.

Контрольні запитання до лабораторної роботи № 1

1. Поясніть у чому відмінність між Docker IMAGE та Docker CONTAINER?

2. Що означає тег ALPINE в докер образах?

3. Яким чином можна переглянути контейнери, що були зупинені?

5. Яка роль Dockerfile при створенні образів?

ЛАБОРАТОРНА РОБОТА № 2. Дослідження спільних ресурсів хостової та гостьової систем в Docker

Мета роботи: полягає дослідженні специфіки запуску Docker контейнерів, ознайомленні з репозиторієм Docker Hub та, за потреби, Docker Desktop. Навчитися прокидати порти з гостьової на хостову машини, що дасть змогу працювати з власним веб-сервером nginx.

Вхідні дані ЛР2

У якості вхідних даних для ЛР2 є:

- виконана ЛР1 та її Docker-образи.

Вихідні дані ЛР2

У якості вихідних даних для ЛР2 є: робочий контейнер з веб-сервером nginx.

Завдання

1. Навчитися використовувати спільні ресурси хостової та гостьової систем на прикладі Docker nginx.

2. Переглянути контет сторінок Docker nginx з хостової машини.

Програма проведення експерименту ЛР2

1. Прокинемо порти. В команду docker run додати такі ключі, щоб на сайт контейнера можна було зайти через 127.0.0.1:<80 + двозначна сума номерів перших літер ваших прізвищ>**. Вивести інформацію по активним контейнерам. Приклад позитивного виконання показаний на рис.4. Після отримання результату, потрібно зафіксувати його у вигляді скриншоту. Результат додати до звіту.

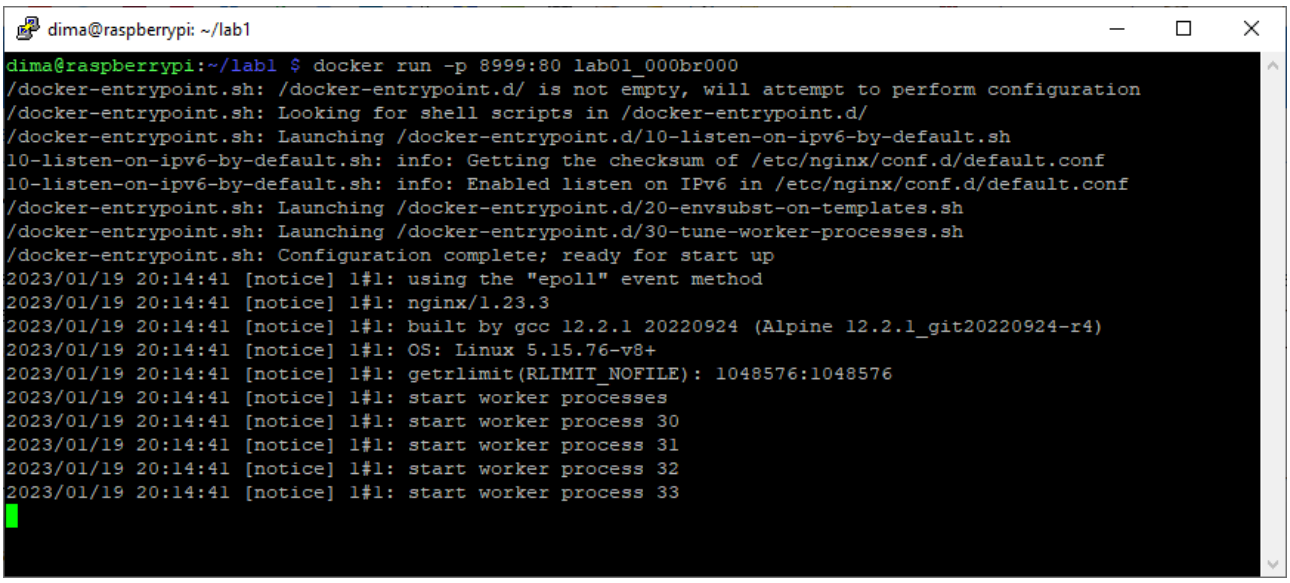

Рисунок 4 – Запуск контейнера з прокиданням портів.

Для перевірки результату через браузер гостьової машини потрібно звернутися до відповідної адреси і порту, а саме перейти на 127.0.0.1:<порт> у браузері, як це показано на рис. 5. Після отримання результату, потрібно зафіксувати його у вигляді скриншоту. Результат додати до звіту.

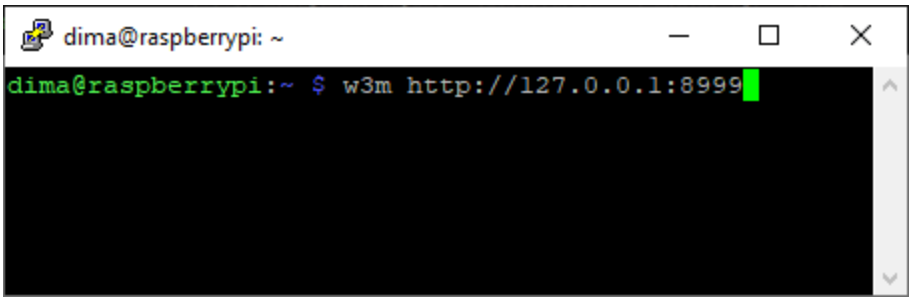

Рисунок 5 – Приклад запиту

2. Замінити контент дефолтної сторінки nginx на власний – перелік ПІБ всіх членів бригади + поточна дата створення image. Замінити через ADD/COPY та, за потреби RUN, у Dockerfile. Створити НОВИЙ image NAME=LAB01_2BR<перші літери прізвищ кожного з членів бригади> на базі імейджу LAB01_1BRxxx. Зайти на сторінку через браузер. Приклад показаний на рис.6.

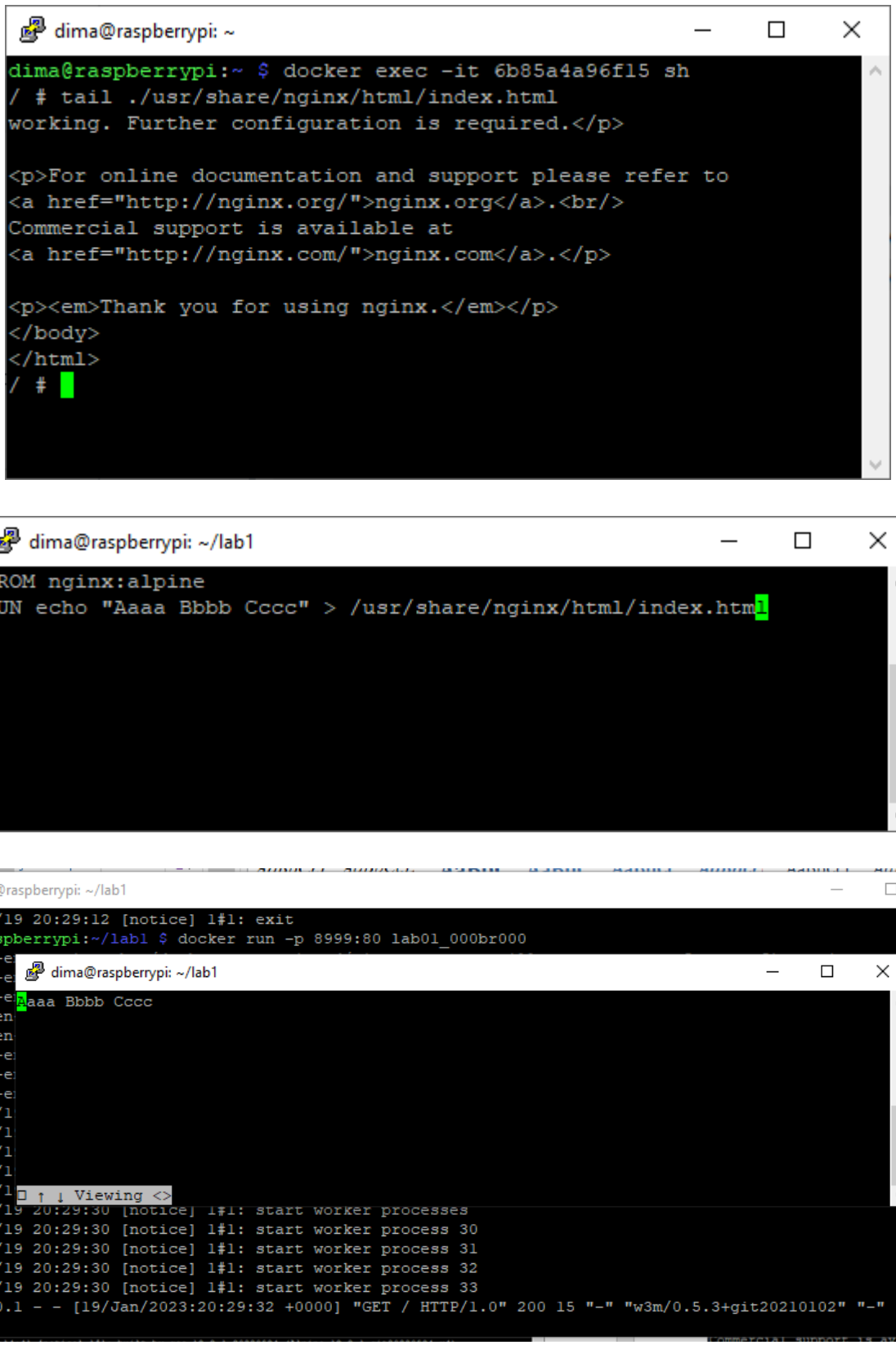

Рисунок 6 – Заміна контенту дефолтної сторінки

3. Замінити контент дефолтної сторінки nginx на власний, але розшарений з каталогу ./lab01 – так само перелік ПІБ всіх членів бригади, але створіть файл у папці ./lab01 і зробити так, щоб через VOLUME цей файл ставав сторінкою. Створити НОВИЙ image NAME=LAB01_3BRxxx на базі імейджу LAB01_1BRxxx. Зайти на сторінку через браузер. Після отримання результату, потрібно зафіксувати його у вигляді скриншоту. Результат додати до звіту.

4. Сформувати звіт.

Контрольні запитання до лабораторної роботи № 2

1. Що таке прокидання портів та навіщо його робити при запуску докер контейнерів?

2. Що таке VOLUME у докері?

3. Що виконує інструкція RUN у Dockerfile?

5. Яка команда дозволяє зайти в командний рядок контейнера Docker?

ЛАБОРАТОРНА РОБОТА № 3. Створення і розгортання програмної інфраструктури на основі docker-compose

Мета роботи: полягає у дослідженні процесу автоматичного розгортання відносно складної програмної інфраструктури розподіленого веб-застосунку за обраним напрямом технології. Зважаючи на те, що сучасні РПС являють собою систему програмних модулів, що взаємодіють між собою і реалізовані на різних технологіях, їх автоматичне розгортання потребує додаткових програмних механізмів, що спрощують процес розгортання і розробки.

Одним з таких механізмів є docker-compose. У цій ЛР відбувається розгортання РПС з різних модулів відповідно до наданого завдання, вивчення синтаксисту файлів docker-compose і тестування отриманих результатів.

Лабораторну роботу можна умовно розподілити на три частини:

- вивчення і тестування складових частин відносно складної РПС, дозволяється по узгодженню з викладачем використовувати власні напрацювання і досвід роботи за фахом студентів, що навчаються;

- підготовка первинного файлу docker-compose, вивчення елементів синтаксису і формування стилю цього файлу;

- розгортання РПС з використанням отриманого docker-compose, тестування роботи складної програмної системи, виправлення помилок, що були виявлені.

Вхідні дані ЛР3

У якості вхідних даних для ЛР3 є:

- типова база даних, що містить відповідний стандартний контейнер, який може бути активований з docker-compose;

- два або три контейнера з базовим веб-застосунком, що побудовані на основі типового веб-фреймворку і взаємодіють з базою даних.

Вихідні дані ЛР3

Система каталогів з файлом docker-compose.yml.

Завдання

1. У якості індивідуального завдання, на першому етапі вивчити переваги і недоліки бази даних, що надана у переліку варіантів, за необхідністю побудувати файл docker-compose для бази даних і протестувати роботу цього файлу на реальної системі, користуючися доступними засобами тестування БД.

Вивчити веб-застосунок, що побудований на основі фреймворку, наданого відповідно до варіанту. При необхідності провести його дослідження з docker-compose, аналогічно БД.

2. Підготувати файл docker-compose, що дозволяє побудувати узагальнену систему з БД і двома-трьома контейнерами з веб-застосунку, що взаємодіють відповідно до функціоналу з БД.

3. Провести розгортання РПС з використанням отриманого docker-compose. Після розгортання провести тестування роботи складної програмної системи, що запущена з використанням результатів п.2. У випадку наявності, провести виправлення помилок. При неможливості поєднати БД і фреймворк веб-застосунку, повідомити викладача, обґрунтовано обрати іншу БД або фреймворк.

4. Отриманні результати, вихідний програмний код, каталоги і файли розмістити на https://github.com/. Дозволяється розміщувати або у відкритому або у приватному варіанті. Надати викладачеві права доступу для первинної перевірки.

5. За потребою створити звіт. Дозволяється надавати викладачеві результати відповідно до звіту без захисту ЛР. За потребою викладач надає запитання для захисту лабораторної роботи.

Варіанти завдання

Далі надані варіанти завдань, що обираються студентами відповідно до списку у групі. По узгодженням з викладачем можна обрати іншу базу даних або фреймворк та інший рівень дослідження.

Варіант 1.
База даних postgres:14.1-alpine,
фреймворк Django (два контейнера).

Варіант 2.
База даних mysql,
фреймворк Django (один контейнер).

Варіант 3.
База даних postgres:14.1-alpine,
фреймворк Django (один контейнер).

Варіант 4.
База даних mysql,
фреймворк Django (два контейнера).

Програма проведення експерименту ЛР3

Програма проведення експерименту буде описана у контексті використання postgres:14.1-alpine та двох проектів з використанням Django. Звичайно при використанні інших рішень програма може відрізнятися.

1. Перший етап.

Перед початком досліджень перевіряємо працездатність Docker для обраного типу ОС. Перевіряємо працездатність Docker compose для обраного типу операційної системи. Для деяких ОС засіб Docker compose постачається за замовчуванням разом з Docker, для деяких ОС потрібно його додатково інсталювати. При необхідності провести додаткову інсталяцію з використанням стандартних засобів, для обраної операційної системи.

Створюємо робочий каталог проекту. У робочий каталог проекту розміщуємо файл docker-compose.yml.

2. Другий етап.

Користуючись мануалом [4 д.], створюємо два мінімальних проекти Django у робочому каталозі проекту (РКП) разом з веб-застосунком на базі типового мануала. Достатньо виконати простий проект на рівні двох частин мануалу [4 д.] «Writing your first Django app, part 1», «Writing your first Django app, part 2».

Результатом мають бути два внутрішніх каталоги у РКП з проектами Django. Провести тестування двох проектів на працездатність. Як це зробити показано у мануалі [4 д.] частина 1.

Користуючись мануалом [4 д.] переналаштувати отримані контейнери на базу даних Postgres.

3. Третій етап.

Для кожного проекту у РКП додаємо Dockerfile з відповідними налаштуваннями, з урахуванням БД Postgres. У результаті роботи система каталогу кожного проекту Django отримає вигляді, що показаний на рис.7.

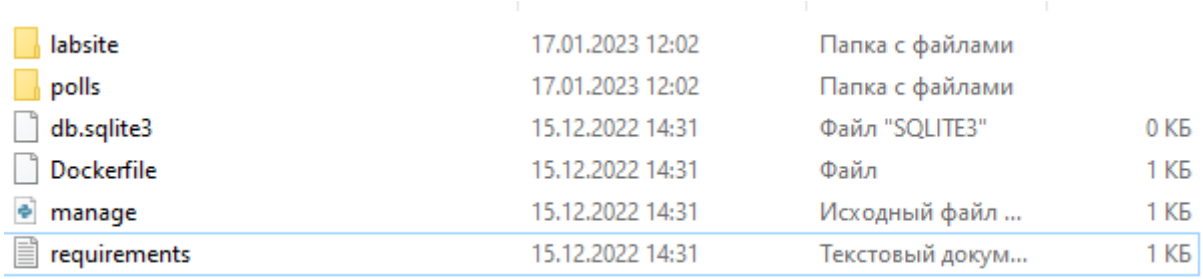

Рисунок 7 – Зміст каталогу одного з проектів Django.

Як можна побачити з рис.7, до стандартного проекту додається Dockerfile і текстовий файл requirements.txt з базовими налаштуваннями проекту. Файл з базою даних db.sqlite3 можна видалити. Виконуємо всі потрібні дії для підготовки системи контейнерів.

4. Четвертий етап.

Відповідно до функціоналу вносимо зміни до файлу docker-compose.yml. Передбачено, що будуть працювати три контейнера. Два контейнера з проектом

Django і контейнер з базою даних Postgres. Після внесення змін до файлу docker-compose.yml перевіряємо результат docker-compose build. При успішної побудові доступним є тест двох проектів Django.

У випадку помилок, внести відповідні зміни до файлів docker-compose.yml і провести додаткове тестування.

5. П'ятий етап.

Далі налаштовуємо Postgres. Для цього вносимо відповідні зміни до файлу docker-compose.yml. Оновлюємо налаштування у двох проектах Django налаштовуємо додаткові елементи. Після внесених змін проводимо тестування результатів. Після отримання результату, потрібно зафіксувати його у вигляді скриншоту. Результат додати до звіту.

Теоретичні відомості ЛР3

На теперішній час активно розвивається технологія Docker і споріднена з нею технологія Docker compose. Використання технології Docker compose доцільно у виробничому та перед-виробничому (staging) процесі, у ході тестування і підтримки, процесах безперервної інтеграції і створення мікросервісних архітектурних рішень.

Для роботи зі складними РПС, що побудовані на основі СК Docker використовується система Docker Compose. За допомогою цього інструменту відбувається автоматизація процесу розгортання сервісів та заданої конфігурації. Зручність цієї системи полягає у тому, що всі елементи можуть бути створені та запущені однією командою з терміналу (консолі).

Доступні команди управління життєвим циклом РПС різного рівня складності, а саме: запуск, зупинка та відновлення сервісів, додавання нових сервісів, визначення їх технічного стану, документування рішень, тощо.

Система Docker Compose має наступні базові кроки виконання:

- визначення загальної архітектури РПС та базових елементів;

- документування результатів і створення відповідного файлу у якому записаний перелік всіх елементів автоматизації;

- безпосередня робота і запуск файлу автоматизації Docker compose;

- контроль і моніторинг результатів під час роботи РПС.

Приклад простого файлу показаний на рис.8. Він містить простий приклад роботи з базою даних. У контейнері відбувається налаштування бази даних, що приведені далі. У такому вигляді система docker-compose не дає можливості переконатися у її перевагою перед технологією Dockerfile. Проте цей приклад, наведений з коментарями, надає можливість чітко визначити основу і принципи побудови docker-compose.

```
version: '3.8'
services: #Визначаємо потрібні сервіси для роботи РПС
  db: #Сервіс для роботи з базою даних
    image: postgres:14.1-alpine
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    volumes:
      - db:/var/lib/postgresql/data # Надає шлях до файлу,
                                    # що призначений для автоматизації команд
  web:
    build:
      context: labsite
    depends_on:
      - db
    ports:
      - '8000:8000'
volumes:
  db:
    driver: local
```

Рисунок 8 – Приклад простої роботи з базою даних.

Більш цікавий приклад наведений далі на рис.9. На даному прикладі наведено формування відносно складної РПС, що містить базу даних і підключений до неї фреймворк Django. Фреймворк містить типове початкове рішення з веб-застосунком polls і helloworld. Крім того у РПС включена база даних postgres.

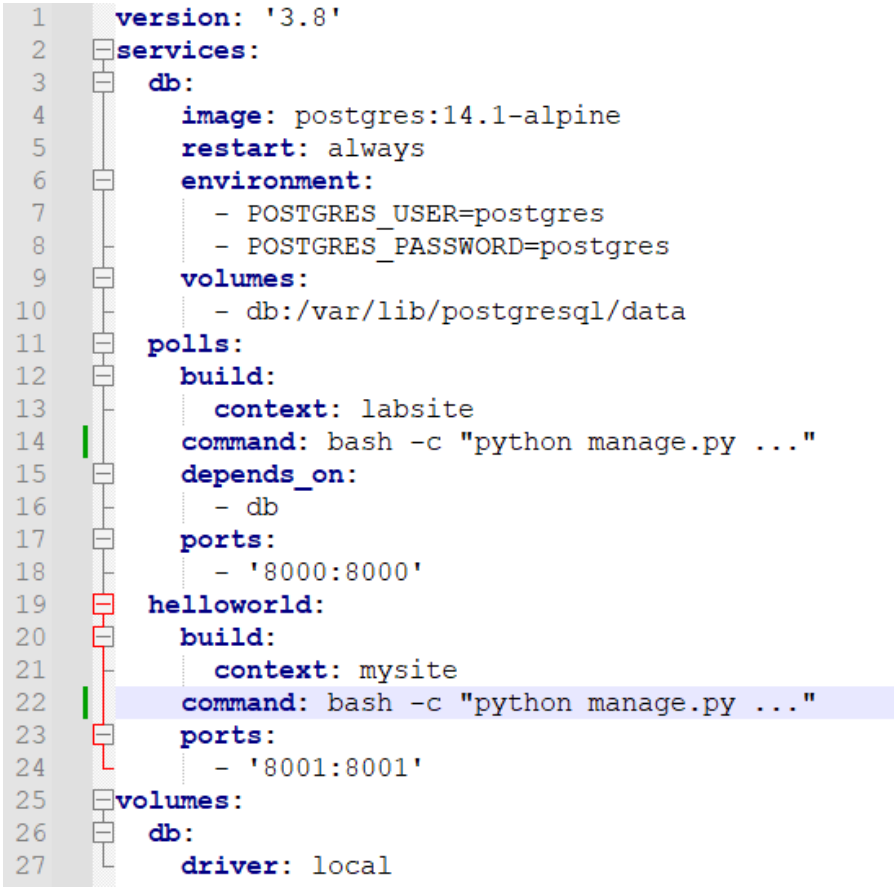

Рисунок 9 – Складна розподілена програмна система

Зважаючи на рішення, що приведені на рис.8 і рис.9 можна визначитися з типовими вихідними даними для реалізації лабораторної роботи.

Базовий функціонал Docker compose передбачає можливість збереження даних томів під час створення контейнерів. Тобто будуть збережені всі томи, що використовуються створеними сервісами і у такий спосіб відбувається захист від втрат. Є можливість створення копії томів зі старого контейнера в новий створений контейнер.

Наступною важливою перевагою Docker compose є те, що він кешує конфігурацію яка є базою для створення контейнера. Ця система дозволяє будувати кілька ізольованих середовищ на одному хості.

При цьому використовується назва проекту, яка дозволяє реалізувати ізоляцію певних контейнерів відповідно до завдання і архітектури. Можна використовувати технологію ізоляції у наступних контекстах, а саме:

- унеможливити перешкоджанню різних проектів, що можуть використовувати однакові служби на загальному хості або хості розробника;

- створювати кілька незалежних копій одного середовища виконання, наприклад, потрібно запустити стабільну копію для кожної гілки проекту та зробити аналогічну для налагодження і вдосконалення;

- зробити неможливим процес, коли збірки заважатимуть роботі одна одної на сервері, для цього встановлюється унікальний номер збірки.

Система Файл Docker compose надає для розробників, тестувальників і системних інженерів наступні можливості: надає багатий функціонал і можливості для розробників, а саме: налаштовувати всі залежності служб РПС (кеші, черги, бази даних, API веб-служб тощо) за допомогою однієї команди (docker compose up). У цьому варіанті можна створити та запустити один або кілька контейнерів для кожної залежності.

Для тестувальників РПС забезпечується зручний спосіб створення та видалення ізольованих середовищ автоматизованого тестування потрібного набору тестів за допомогою відповідних команд.

Хід і алгоритм встановлення системи Docker compose залежить від операційної системи. Для операційної системи родини Windows Docker Compose може бути встановлений за допомогою програмного застосунку Docker Desktop.

Для операційних систем родини Linux передбачено встановлення відповідно до технологій. Наприклад інсталяція Compose plugin для Ubuntu передбачає виконання кроків, що надані далі. На першому етапі треба отримати нову стабільну версію Docker Compose, що можна зробити шляхом завантаження з офіційного репозиторію Github – https://github.com/docker/compose. Далі потрібно зробити інструмент доступним глобально, шляхом виконання команди [5, 7-9]:

$ sudo curl -L

На наступному етапі потрібно надати відповідні дозволи для виконання команди у середовище операційної системи. Це відбувається шляхом зміни відповідних прав доступу, а саме:

$ sudo chmod +x /usr/local/bin/docker-compose

Для перевірки працездатності системи виконується наступна команда:

$ docker-compose --version.

Відображення поточної версії програми свідчить про її правильну інсталяцію і готовність до роботи, інші повідомлення свідчать про потребу повторного встановлення або налаштування.

Опишемо приклад створення і простого прикладу файлу docker-compose.yml, що взятий з офіційного джерела [5, 7-9]. У цьому файлі відбувається автоматизація створення контейнера з веб-сервером, що побудований на базі образу Nginx із Docker Hub, загальнодоступного реєстру Docker. У якості прикладу цей веб-серевер буде хостити один статичний файл.

На першому етапі потрібно створити новий робочий каталог, у обраному місці операційної системи, перейти до нього. У цьому каталозі потрібно створити каталог app і розмістити у ньому статичний файл index.html. Все це робиться шляхом виконання наступної послідовності команд:

$ mkdir ~/compose-demo

$ cd ~/compose-demo

$ mkdir app

$ nano app/index.html

Далі у файлі index.html реалізувати контент, що буде відображатися. На цьому завершується підготовчий етап створення демо-інфраструктури. На наступному етапі створюється безпосередньо елемент системи Docker Compose, а саме файл docker-compose.yml. Як варіант можна використати наступну команду:

$ nano docker-compose.yml

Після відкриття текстового редактора до файлу потрібно додати контент, що приведений далі і взятий з офіційного джерела [5, 7-9].

docker-compose.yml

```
version: '3.7'
services:
web:
image: nginx:alpine
ports:
- "8000:80"
volumes:
- ./app:/usr/share/nginx/html
```

Розглянемо складові елементи файлу docker-compose.yml. Він включає версії конфігурації (version).

Далі йде блок сервісів (services), якій містить відповідні налаштування, що формують складові елементи контейнерів, або контейнеру, який представлений у даному прикладі. Фактично це є єдиний контейнер з web-сервером на основі образу nginx:alpine.

Крім того передбачено у блоці services внутрішній блок для переспрямування портів (директива ports). Усі запити на порт 8000 хост-комп’ютера (системи, з якої запускається Docker Compose) будуть переспрямовані на веб-контейнер на порту 80 Nginx.

Директива volumes створить спільний том між головною машиною та контейнером. Це надасть контейнеру спільний доступ до каталогу програми. Зміст тому буде розміщено за адресою /usr/share/nginx/html усередині контейнера.

Звичайно, що складні РПС містять набагато більше блоків у файлах docker-compose.yml. Приклад можна побачити на рис.8, 9.

Після створення файлу docker-compose.yml його можна запустити у середовще Docker Compose. Наступна команда активує записи у docker-compose.yml та запустить контейнерне середовище у фоновому режимі:

$ docker-compose up -d

При необхідності система Docker Compose шукає образи або на локальній системі або його з Docker Hub.

Для тестування РПС, що активована після роботи docker-compose.yml, потрібно запустити команду “docker-compose ps”. Ця команда виводить всю інформацію відповідно РПС, що активована.

Для доступу до демонстраційного додатку, що створений у вищеописаному прикладі потрібно набрати у браузері localhost:8000. Відповіддю на це буде відображення сторінки, що записана у файлі index.html.

Контрольні запитання до лабораторної роботи № 3

1. Поясніть у чому відмінність технології Docker і Docker compose?

2. Які етапи створення складної розподіленої програмної системи з використанням технології Docker compose?

3. Розкрийте основу структури файлу docker-compose.yml.

5. Який зв'язок файлу docker-compose.yml і dockerfile?

6. Розкрийте і опішить структуру складного розподіленого проекту, що побудований за технологією Docker compose?

7. Які команди подаються для розгортання складного розподіленого проекту з використанням Docker compose?

8. Наведіть порядок створення складного розподіленого проекту з використанням Docker compose?

ЛАБОРАТОРНА РОБОТА № 4.1 (Напрям 121) Створення веб-застосунку на основі типової об'єктно-реляційної проекції з використанням контейнеризації

Мета роботи: полягає у дослідженні процесу створення, тестування веб-застосунку на основі типового фреймворку у системі контейнеризації з використанням Dockerfile. Робота виконується на основі фреймворку Django на основі бази даних за замовчуванням і відповідно заданому варіанту БД.

Вхідні дані ЛР4.1

Вхідними даними для реалізації лабораторної роботи є:

- типовий фреймворк Django на рівні складності мануалу [4 д.] «Writing your first Django app, part 1-4», з типовою базою даних за замовчуванням, що реалізується на основі типової об'єктно-реляційної проекції;

- завдання для розробки бази даних, що надано у додатку 1, у якості вихідних даних для ЛР можуть бути використані таблиці, що використовувалися у якості курсу лабораторних робіт попередніх предметів.

Вихідні дані ЛР4.1

У якості вихідних даних є: робочий контейнер з проектом Django.

Завдання

1. Перевірити працездатність системи Docker. У випадку несправності у роботі відновити працездатність або перевстановити систему.

2. Створити каталог для робочого проекту. Створити проект Django, що містить налаштування БД за замовчуванням.

Користуючись завданням і описом бази даних у додатку 1, або розробленими проектами БД створити на основі типової об'єктно-реляційної проекції базу даних у рамках проекту відповідно до завдання і обраного варіанту. Користуючись [4 д.], протестувати отримане рішення.

Довести складність проекту до рівня «Writing your first Django app, part 5» [4 д.]. Провести тестування проекту.

3. Внести контейнер для отриманого рішення з Dockerfle. Протестувати отримане рішення. Результати розмістити на GitHub.

Варіанти завдання

Варіанти завдань надані у додатку 1, що обираються студентами відповідно до списку у групі. По узгодженням з викладачем можна обрати іншу базу даних.

Програма проведення експерименту ЛР 4.1.

1. Перший етап.

Створюємо проект Django у робочому каталозі проекту на основі системи контейнеризації.

2. Другий етап.

Користуючись мануалом [4 д.], допрацьовуємо проект Django у робочому каталозі проекту на базі типового мануала до рівня «Writing your first Django app, part 5» з урахуванням бази даних. Проводимо тестування на працездатність.

3. Третій етап.

Після отримання результатів, потрібно зафіксувати їх у вигляді скриншотів. Результат додати до звіту. Оформлюємо звіт, робимо висновки.

Контрольні запитання до лабораторної роботи № 4.1

1. Поясніть у чому особливість роботи з БД у Django ?

2. Які етапи створення проекту на Django?

3. Розкрийте основу структури Django.

4. У чому переваги і недоліки роботи з об'єктно-реляційної проекції баз даних?

ЛАБОРАТОРНА РОБОТА № 4.2 (Напрям 126) Створення проекту на базі Spring Framework з використанням контейнеризації

Мета: навчитися створювати консольні застосунки та RESTful вебсервіси за допомогою Spring Framework та Spring Boot.

Вхідні дані ЛР4.2

Вхідними даними для реалізації лабораторної роботи є:

- контейнер з JDK.

Вихідні дані ЛР4.2

Працездатний контейнер "Hello World" з JDK з дефолтним ендпойнтом.

Завдання

1. Створити контейнер "Hello World" з JDK.
2. Створити власні REST-full API за допомогою фреймворку Spring.

Програма проведення експерименту ЛР4.2

1. Встановіть одну з двох IDE:

- Spring Tool Suit (Free, Open Source, https://spring.io/tools);
- IntelliJ IDEA Ultimate Edition (як студенти КПІ ви можете отримати безкоштовну ліцензію, https://www.jetbrains.com/student/).

2. Створіть консольний застосунок HelloWorld.

2.1. За допомогою Spring Starter створіть проект з наступними параметрами, що показані на рис. 10.

Версії Java або інших елементів конфігурації можуть відрізнятися від тих, що вказані на рисунках. При першому використанні процес завантаження залежностей у локальний репозиторій може зайняти деякий час в залежності від швидкості Вашого підключення до мережі.

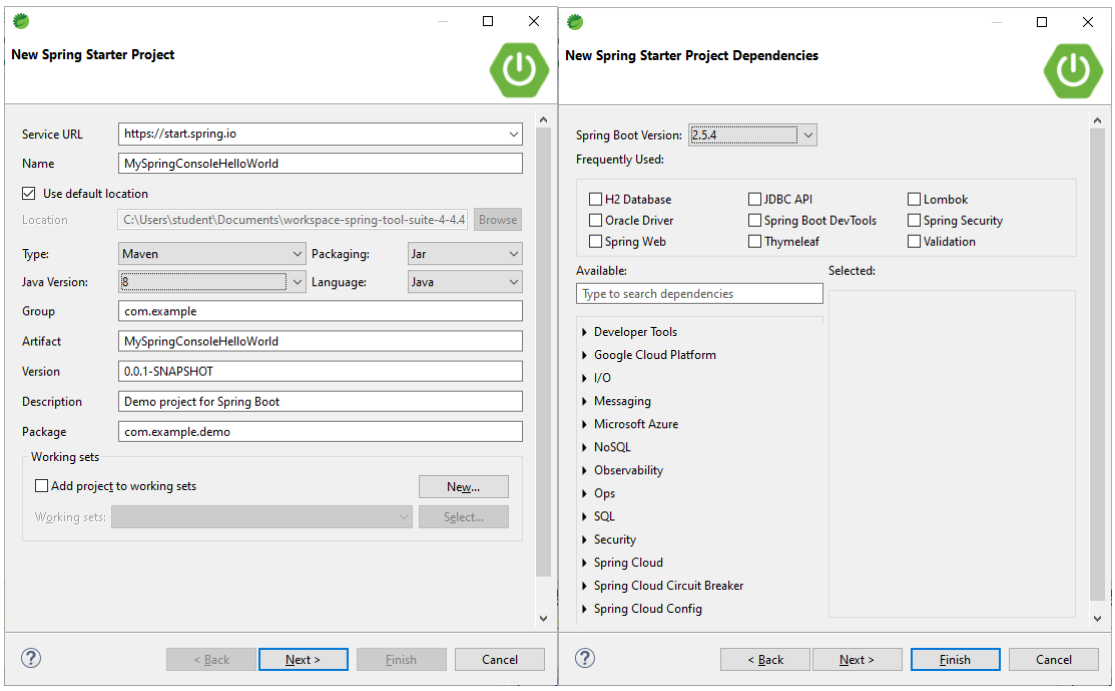

Рисунок 10 – Параметри фреймворку

Після створення проекту його структура має вигляд, що показана на рис.11.

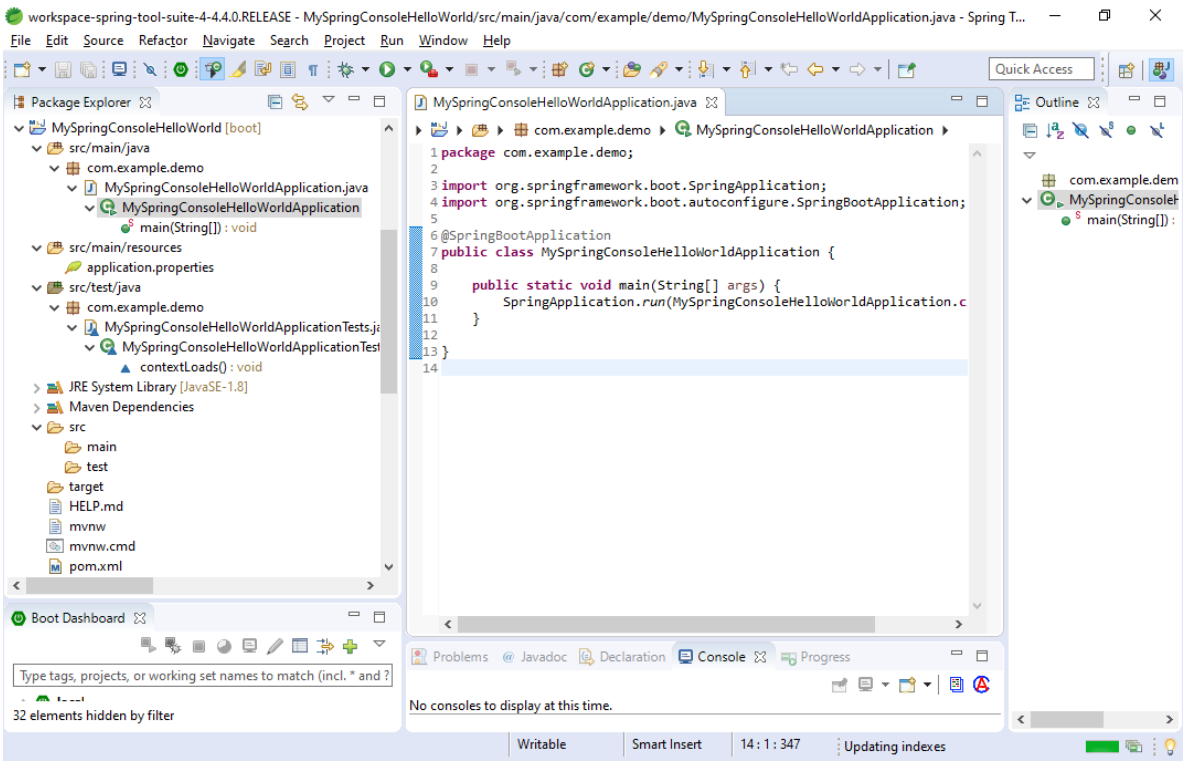

Рисунок 11 – Структура проекту

2.2. Додайте до декларації класу інтерфейс CommandLineRunner та відповідний метод run(String... agrs), як це показано на рис.12.

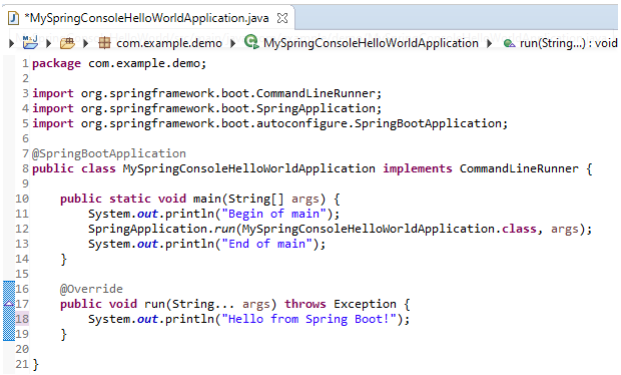

Рисунок 12 – Додавання інтерфейсу до декларації класу

2.3. Запустіть проект. Зверніть увагу на порядок виводу рядочків у консолі.

3. Створіть і запустіть RESTful вебсервіс HelloWorld.

3.1. Оберіть в меню File -> New -> Spring Starter Project. Задайте параметри, як це показано на рис.13.

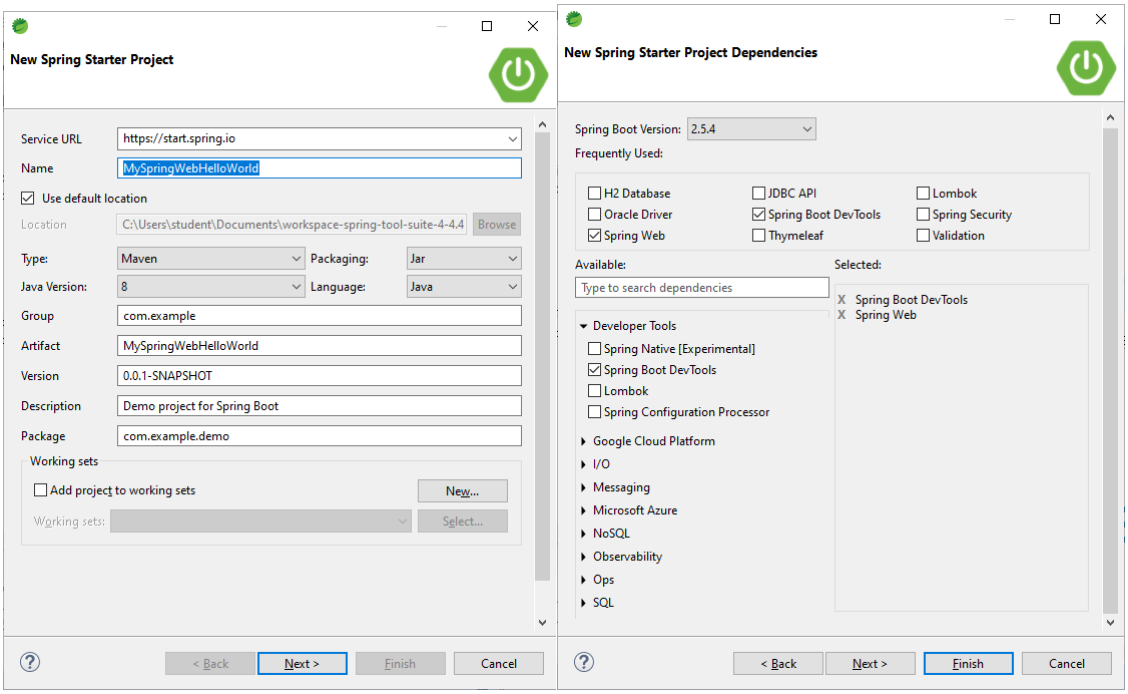

Рисунок 13 – Параметри проекту

3.2. Додайте контролер, як це показано на рис.14.

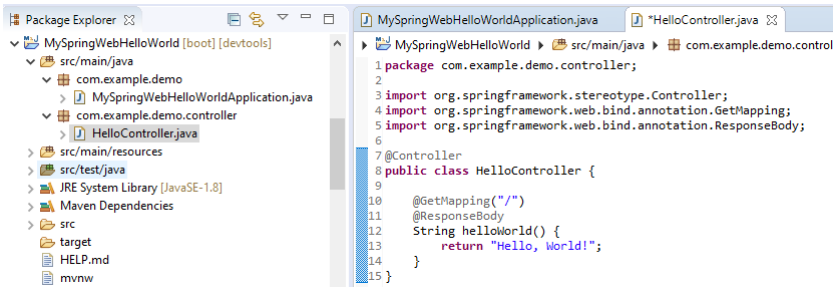

Рисунок 14 – Контролер застосунку

3.3. Запустіть проект, відкрийте браузер та перейдіть за посиланням http://localhost:8080/, має бути відображений результат, що показаний на рис.15.

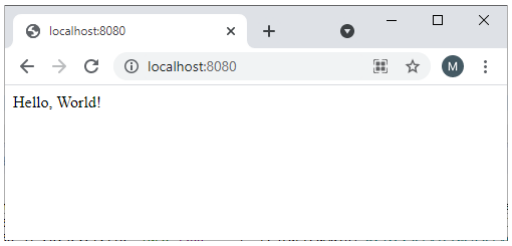

Рисунок 15 – Результат роботи програми з контейнера

3.4. Замість анотації @Controller перед класом та @ReponseBody перед методом спробуйте використати лише анотацію @RestController перед класом.

3.5. Створіть в контролері ще декілька методів, які будуть повертати масив рядочків, колекцію рядочків, та об'єкт, в якому є декілька полів, гетери та сетери. За допомогою анотації @GetMapping задайте для них різні шляхи. Перевірте, що вказані дані правильно відображаються в браузері в форматі JSON.

Контрольні питання

1. В чому полягає різниця між Spring Framework та Spring Boot?
2. Для чого в структурі проекту потрібен файл pom.xml?

3. Для чого потрібна анотація @SpringBootApplication? Що буде, якщо її прибрати?

4. Для чого потрібен інтерфейс CommandLineRunner?

5. Чим інтерфейс CommandLineRunner відрізняється від ApplicationRunner?

6. Що таке сервлет?

7. Для чого потрібен шаблон проектування MVC.

8. В чому полягають переваги використання шаблону проектування Front Controller?

9. Поясніть особливості використання шаблону проектування Front Controller при реалізації веб-застосунків та RESTful веб-сервісів.

10. В чому полягають відмінності методів GET та POST протоколу HTTP?

ЛАБОРАТОРНА РОБОТА № 5.1 (Напрям 121) Створення і розгортання програмної інфраструктури на основі бази даних і фреймворку Django

Мета роботи: полягає у дослідженні процесу автоматичного розгортання відносно складної програмної інфраструктури. Вона містить окрему базу даних і програмний застосунок на основі Django.

Вхідні дані ЛР5.1

У якості вхідних даних для ЛР5.1 є:

- типова база даних, таблиці якої будуть побудовані на основі варіантів додатку 1, може бути використані результати ЛР 4.1.

Вихідні дані ЛР5.1

Система каталогів з веб-застосунком на основі фреймворку Django і файл docker-compose. Система має передаватися через GitHab і розгортатися на будь-якому хості, де встановлений Docker і docker-compose.

Завдання

1. Підготувати файл docker-compose, що дозволяє побудувати два контейнера. Перший контейнер з веб-застосунком, другий контейнер з базою даних.

2. Внести зміни до веб-застосунку, що дозволяють перебудувати базу даних за замовчуванням фреймворку Django на базу даних, яка задана у ЛР3 у варіантах.

3. Внести зміни до веб-застосунку, що реалізують функціонал відображення вибірок і таблиць бази даних.

4. Провести розгортання РПС з використанням отриманого docker-compose. Після розгортання провести тестування роботи складної програмної

системи, що містить веб-застосунок і окрему базу даних у різних контейнерах. У випадку наявності, провести виправлення помилок.

5. Отриманні результати, вихідний програмний код, каталоги і файли розмістити на https://github.com/. Дозволяється розміщувати або у відкритому або у приватному варіанті. Надати викладачеві права доступу для первинної перевірки.

6. За потребою створити звіт.

Варіанти завдання

Варіанти завдань, що обираються студентами відповідно до списку у групі для виконання лабораторної роботи надані у ЛР3.

Варіанти завдання для створення БД надані у додатку 1. По узгодженням з викладачем можна обрати іншу базу даних або фреймворк та інший рівень дослідження.

Програма проведення експерименту ЛР 5.1

1. Перший етап.

Перед початком досліджень перевіряємо працездатність Docker і Docker compose для обраного типу операційної системи. При необхідності провести додаткову інсталяцію з використанням стандартних засобів.

Створюємо робочий каталог проекту. У робочий каталог проекту розміщуємо файл docker-compose.yml.

2. Другий етап.

Користуючись мануалом [4 д.] і результатами ЛР4.1, створюємо проект Django у робочому каталозі проекту, що дає можливість відобразити зміст таблиць БД і відповідних вибірок додатку 1.

Результатом мають бути внутрішній каталог у РКП з проектом Django. Провести тестування проекту на працездатність. Як це зробити показано у мануалі [4 д.] частина 1.

Користуючись мануалом [4 д.] переналаштувати веб-застосунок на базу даних відповідно варіанту, що вказаний у ЛР3.

3. Третій етап.

Для веб-застосуноку у РКП додаємо Dockerfile з відповідними налаштуваннями, з урахуванням БД. До стандартного проекту додається Dockerfile і текстовий файл requirements.txt з базовими налаштуваннями проекту.

4. Четвертий етап.

Відповідно до функціоналу вносимо зміни до файлу docker-compose.yml. Передбачено, що будуть працювати два контейнера. Контейнер з проектом Django і контейнер з базою даних. Після внесення змін до файлу docker-compose.yml перевіряємо результат docker-compose build. При успішної побудові доступним є тест проекту Django.

У випадку помилок, внести відповідні зміни до файлів docker-compose.yml і провести додаткове тестування.

5. П'ятий етап.

Після отримання результату, потрібно зафіксувати його у вигляді скриншотів і розмістити на https://github.com/.

ЛАБОРАТОРНА РОБОТА № 5.2 (Напрям 126) Створення RESTful вебсервісів за допомогою Spring Framework

Мета роботи: навчитися створювати RESTful вебсервіси за допомогою Spring Framework та Spring Boot.

Вхідні дані ЛР 5.2

контейнер з JDK. При цьому зверніть увагу:

- версія JDK – не нижче ніж JDK8;
- не переплутайте JDK (Java Development Kit) з JRE (Java Runtime Environment).

Вихідні дані ЛР 5.2

Контейнер JDK з реалізованими не дефолтними ендпойнтами.

Завдання

- створити власні REST-full API за допомогою фреймворку Spring;
- створити сервіси на базі Docker та фреймворку Spring.

Програма проведення експерименту ЛР5.2

1. Створіть обліковий запис на ресурсах «https://swagger.io/» та «https://www.postman.com/» (реєстрація безкоштовна). Встановіть на свій комп'ютер Postman Desktop Agent (встановлюється безкоштовно).

2. Для заданої викладачем предметної області сплануйте схему представлення даних у вигляді RESTful ресурсів та реалізуйте рівень представлення. Проект має функціонувати як RESTful вебсервіс. При цьому:

2.1 має бути присутнім що найменше один ресурс, для якого реалізовані усі чотири CRUD-операції (create, read, update, delete);

2.2 щонайменше для одного ресурсу мають бути реалізовані функції фільтрації та пагінації;

2.3 усі операції мають повертати відповідний код стану HTTP в залежності від успішності чи неуспішності операції;

2.4 в процесі виконання потрібно показати вміння користуватися наступними анотаціями та класами: @RestController, @RequestMapping, @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @RequestBody, @PathVariable, @RequestParam, @Valid, ResponseEntity.

2.5 функції, пов’язані з безпекою, в цій роботі можуть бути не реалізовані;

2.6 функції, пов'язані зі збереженням даних (рівень репозиторію), можуть бути реалізовані у вигляді заглушки, яка зберігає дані у пам'яті.

3. До розроблених RESTful вебсервісів створіть документацію у форматі OpenAPI. Для кожної операції в документації має бути присутніми:

- короткий опис операції;

- повний опис операції;

- опис параметрів та/або моделей даних, які використовуються в цій операції;

- перелік кодів стану HTTP, які можуть надходити в результаті спроби виконання операції.

Рекомендується використати підхід, при якому документація генерується автоматично за допомогою включення в проект бібліотеки Swagger (OpenApi), і додавання в контролер таких анотацій як @Tag, @Operation.

4. За допомогою Postman провести тестування розробленого API на відповідність опису у документації.

Короткі теоретичні відомості

Як відомо у протоколі HTTP є багато методів (GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, PATH), рис.16, але для реалізації RESTful API як правило використовують лише GET, POST, PUT, DELETE і іноді PATCH. Для реалізації операцій CRUD (Create, Read, Update, Delete) ці методи можуть використовуватися наступним чином.

	Назва	GET	POST	PUT	PATCH	DELETE
Методи HTTP	Безпечність (read only)	+	-	-	-	-
	Ідемпотентність	+	-	+	-	+
Операції CRUD	**Read**	+	-	-	-	-
	Create	-	+ ID генерується сервером	+ ID генерується клієнтом	-	-
	Update	-	-	+ Передається новий стан ресурсу (replace)	+ Передаються інструкції щодо зміни ресурсу (modify)	-
	Delete	-	-	-	-	+

Рисунок 16 – Методи протоколу HTTP

Метод GET.

Цей метод слід використовувати лише для операції читання. Він має бути реалізований розробником сервісу таким чином, щоб в жодному разі виклик цього методу не призводив до побічних ефектів у вигляді створення, зміни чи видалення даних, які перманентно зберігаються на сервері (за виключенням створення записів в лог-файлах, зміни лічильника відвідувань тощо). Для операцій, які передбачають створення, зміну та видалення даних або мають відповідні побічні ефекти, слід використовувати інші методи. Також слід зазначити, що даний метод є ідемпотентним, тобто його повторний виклик не повинен приводити до принципово іншого результату.

Відповіді на запити такого типу можуть кешуватися з метою більш ефективного використання пропускної здатності каналів зв’язку, зменшення затримки при отриманні відповіді та зменшення навантаження на сервер. Чи може відповідь кешуватися чи ні, якщо так то ким саме та як довго, вказується у заголовку відповіді в полях Expires, Cache-Control, ETag.

Приклади використання методу GET:

GET /users – отримати усіх користувачів;

GET /users/1234 – отримати користувача з ID=1234;

GET /users?name=Petro – отримати усіх користувачів з іменем Petro (фільтрація);

GET /users?page=1&size=10 – отримати усіх користувачів зі сторінки 1, якщо виводити по 10 користувачів на сторінку.

Метод POST.

Цей метод слід використовувати лише для створення нових ресурсів на сервері. При цьому унікальний ідентифікатор має генеруватися саме сервером. Він має надсилатися клієнту разом з відповіддю у заголовку в полі Location.

Метод POST не є ідемпотентним тому що якщо з клієнта надіслати кілька однакових POST-запитів один за одним з одними й тими самими даними, то це призведе до створення кількох нових ресурсів на сервері з однаковим змістом але різними ідентифікаторами. Саме тому дуже важливо при реалізації клієнтського UI захистити користувача від помилкової повторної відправки POST-запитів на сервер.

Приклад використання методу POST: створити нового користувача с заданим ім’ям та адресою електронної пошти.

Запит:

POST /users

Тіло запиту:

```
{
    "name": "Petro",
    "email": "petro@example.com"
}
```

Відповідь:

Status code: 201 Created

Header: Location: http://localhost:8080/users/1234

Метод PUT.

Цей метод може використовуватися або для створення нового ресурсу у випадку, якщо унікальний ідентифікатор для нього генерується на клієнті, або для зміни стану існуючого ресурсу. В обох випадках унікальний ідентифікатор ресурсу має бути частиною URL, а повне представлення ресурсу (стан об’єкта) має передаватися в тілі запиту. Як правило для представлення ресурсу використовують формат JSON, але можуть використовуватись і інші формати такі як XML, YAML та інші, якщо вони підтримуються і клієнтом і сервером.

Метод PUT є ідемпотентним тому що разом з запитом передаються як ідентифікатор ресурсу так і його повне представлення (усі поля об’єкту). Тому незалежно від того, чи був ресурс присутній на сервері чи ні, якщо на сервер надіслати декілька однакових PUT-запитів, то перший з них створить новий чи змінить існуючий ресурс, а другий, третій та наступні запити просто перезапишуть його стан ще раз фактично без внесення в нього жодних змін.

Приклад використання методу PUT: створити нового користувача (чи змінити існуючого, якщо такий є) с заданим ідентифікатором, ім’ям та адресою електронної пошти.

Запит:

PUT /users/1234

Тіло запиту:

```
{
    "name": "Petro",
    "email": "petro@example.com"
}
```

Метод PATCH.

Цей метод може використовуватися для зміни існуючого ресурсу на сервері. При цьому на відміну від методу PUT він не є ідемпотентним, бо в запиті замість повного представлення стану ресурсу передається інструкція про те, як треба змінювати об’єкт. Розглянемо приклад, коли для бухгалтерської системи треба передбачити операцію підвищення зарплатні співробітника. Нехай в системі зареєстрований такий співробітник з ID=1234:

Запит:

```
GET /users/1234
```

Результат:

```
{
    "name": "Petrenko",
    "salary": 10000
}
```

Якщо цьому співробітнику потрібно підвищити зарплатню на 5000, то така операція за допомогою запитів PUT або PATCH може виглядати наступним чином. Очевидно, що повторне виконання того самого PUT-запиту не буде призводити до іншого результату, бо ті самі дані будуть просто перезаписуватись без видимих наслідків, в той час як повторне виконання PATCH-запиту буде збільшувати зарплатню співробітнику на 5000 кожен раз. Саме тому метод PATCH не є ідемпотентним.

Запит:

PUT /users/1234

Тіло запиту:

```
{
    "name": "Petrenko",
    "salary": 15000
}
```

Запит:

PATCH /users/1234

Тіло запиту:

```
{
    "operation": "increase salary",
    "amount": 5000
}
```

Слід зазначити, що в такий спосіб метод PATCH використовують не дуже часто у порівнянні з іншими методами HTTP, бо для його реалізації потрібно передбачити як набір команд для зміни стану ресурсу, так і механізм їх парсингу.

Також слід зазначити, що іноді метод PATCH використовують для часткової зміни ресурсу, коли замість його повного представлення в запиті передають JSON-об'єкт, який містить лише ті поля, які треба змінити, а відсутні – залишити у старому стані.

Метод DELETE.

Цей метод використовується для видалення ресурсу. URL повинен містити унікальний ідентифікатор ресурсу, який видаляється. Цей метод є ідемпотентним тому що при повторному виконанні того самого запиту результат не зміниться: після виконання запиту заданий ресурс буде видалений. При цьому може бути таке, що при першій спробі видалити ресурс у відповіді буде статус код 200 Ok, а при другій спробі видалити той самий ресурс – 404 Not Found через те, що ресурс був видалений ще в перший раз. Але відмінність у статус кодах, які надсилаються клієнту, чи різні повідомлення, які клієнт побачить на екрані, не є ознакою того, що метод не є ідемпотентним.

Приклад використання методу DELETE: видалити користувача по ID:

DELETE /users/1234

Контрольні питання

1. Поясніть різницю між вебзастосунками та RESTful вебсервісами.
2. Які технології реалізації концепції сервісно-орієнтованої архітектури ви знаєте? Чим RESTful вебсервіси відрізняються від інших підходів?
3. Поясніть особливості використання шаблону проектування Front Controller при реалізації вебзастосунків та RESTful вебсервісів.
4. Для реалізації якої чи яких операцій CRUD (create, read, update, delete) можуть використовуватися такі методи протоколу HTTP як GET, POST, PUT, PATCH, DELETE?
5. Що означають поняття «безпечний» та «ідемпотентний» метод HTTP? Чи будь який безпечний метод є ідемпотентним? Чи будь який ідемпотентний метод є безпечним?
6. В чому полягає різниця між анотаціями @Controller та @RestController?
7. В чому полягає різниця між анотаціями @GetMapping та @RequestMapping?
8. В яких випадках можуть знадобитися анотації @RequestParam, @PathVariable, @RequestBody?
9. Для чого потрібен клас ResponseEntity?
10. Як за допомогою анотації @Valid здійснювати валідацію даних, які надходять від клієнта?

ОФОРМЛЕННЯ ЗВІТУ ТА ПОРЯДОК ЙОГО ПОДАННЯ

Звіт надається у вигляді складової частини проекту на GitHab. Затримка подання роздрукованого звіту не повинна перевищувати дві лабораторні роботи. У випадку більшої затримки можуть нараховуватися штрафні бали.

У режимі роботи особливого стану виконувати роботу відповідно до поточних керівних документів Університету. Необхідною умовою зарахування ЛР:

– у вихідному коді, або проекті GitHub коментарі з прізвищем, номером групи, номером лабораторної роботи, не підписані роботи не розглядаються;

– наявність файлів, що реалізують всі вимоги завдання до лабораторних роботи, забезпечують працездатність, не надають повідомлення про синтаксичні помилки на етапі виконання, побудови контейнерів або програмної системи;

– здатність студента пояснити створену програмну систему і зміст відповідних файлів для побудови РПС або контейреру;

– здатність студента виконувати відповідні команди для тестування отриманих результатів;

– здатність студента надати незначні зміни до керівничих файлів і налагодити систему після цього.

У тому випадку, коли робота захищається без захисту, до звіту додаються фотографії екранів (скриншоти). Зміст проекту дозволено використовувати під час захисту роботи.

У звіті має бути висновок, що містить інформацію про помилки, що виникли під час розробки і налагодження програми, відповідність результатів довідникової системи. Титульний аркуш, пропонується далі як варіант.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
НАЗВА ФАКУЛЬТЕТУ
НАЗВА КАФЕДРИ

ЛАБОРАТОРНА РОБОТА № 1

З ДИСЦИПЛІНИ «Інфраструктура програмного забезпечення»

Виконав:

студент групи _____

_____

«__»_____

Перевірив:

_____

«__»_____

Київ – 202_

СПИСОК ЛІТЕРАТУРИ

Базова література:

1. Проектування інформаційних систем: Загальні питання теорії проектування ІС (конспект лекцій) [Електронний ресурс]: навч. посіб. для студ. спеціальності 122 «Комп'ютерні науки» / КПІ ім. Ігоря Сікорського; уклад.: О. С. Коваленко, Л. М. Добровська. – Електронні текстові дані (1 файл: 2,02 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 192с. https://ela.kpi.ua/bitstream/123456789/33651/1/PIS_KL.pdf

Додаткова література:

1. ALAN DENNIS, BARBARA HALEY WIXOM, ROBERTA M. ROTH. System Analisis amd desigm.Fifth Edition. John Wiley & Sons, Inc.2012. 563 c.

2. Django документація [Електронний ресурс] –https://docs.djangoproject.com/en/3.2/.

3. Adam Freeman. Essential Docker for ASP.NET Core MVC.ISBN-13 (pbk): 978-1-4842-2777-0 ISBN-13 (electronic): 978-1-4842-2778-7.2017p.

4. https://docs.djangoproject.com/en/4.1/intro/tutorial01/

5. https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-20-04

6. Rebeka Mukherjee. Python Scripting for System Administration.Department of Computer Science and Engineering Netaji Subhash Engineering College, Kolkata.

7. https://docs.docker.com/samples/django/

8. https://docs.docker.com/compose/reference/

9. Adam Freeman. Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages.2020.

10. Docker документація [Електронний ресурс] – https://docs.docker.com/get-started/.

11. Matthes E. Python Crash Course (2nd Edition) : A Hands-On, Project-Based Introduction to Programming / Eric Matthes. – San Francisco, United States: No Starch. Press, US, 9. – 544 c. – (2nd Edition).

12. Thomas D. The Pragmatic Programmer : your journey to mastery, 20th

Anniversary Edition / D. Thomas, A. Hunt. – Boston, United States: Pearson Education. (US), 2020. – 352 c.

13. Learn Python the Hard Way : A Very Simple Introduction to the Terrifyingly

Beautiful World of Computers and Code – New Jersey, United States: Pearson Education. (US), 2013. – 320 c.

14. Learning React : Modern Patterns for Developing React Apps – Sebastopol, United States: O'Reilly Media, Inc, USA, 2020. – 300 c.

15. Docker : Complete Guide To Docker For Beginners And Intermediates, 2020. –

140 c.

16. Docker: Up & Running : Shipping Reliable Containers in Production –

Sebastopol, United States: O'Reilly Media, Inc, USA, 2018. – 347 c.

17. Docker homepage - http://www.docker.com/

18. Docker Hub - https://hub.docker.com

19. Docker blog - http://blog.docker.com/

20. Docker documentation - http://docs.docker.com/

21. Docker Getting Started Guide - http://www.docker.com/gettingstarted/

22. Docker code on GitHub - https://github.com/docker/docker

23. Docker mailing list - https://groups.google.com/forum/#!forum/docker#user

24. Docker on IRC: irc.freenode.net and channels #docker and #docker#dev

25. Docker on Twitter - http://twitter.com/docker

26. Get Docker help on Stack Overflow - http://stackoverflow.com/

search?q=docker

27. Valeria Cardellini. Matteo Nardelli. Container-based virtualization: Docker. Università degli Studi di Roma “Tor Vergata” Dipartimento di Ingegneria Civile e Ingegneria Informatica Corso di Sistemi Distribuiti e Cloud Computing A.A. 2017/18.

28. Adam Freeman. Pro Angular 6 .ISBN-13 (pbk): 978-1-4842-3648-2 ISBN-13 (electronic): 978-1-4842-3649-9/2018 p.

Додаток 1

У перелік варіантів таблиць і фільтрів можуть бути внесені зміни відповідно до результату робіт студентів. Перелік надається як варіант рівня складності і тематичної спрямованості.

Тематика лабораторних робіт

Варіант бази даних №1. Таблиці і запити агентства з реклами.

Таблиці: Клієнти (Код, ПІБ, Вік, Пол). Співробітники (Код, ПІБ, Вік, Пол). Співробітники і посади (Код, посади, Оклад, Обов'язки, Вимоги).

Відділ кадрів (Зв'язує Таблиці "Співробітники" і "Співробітники і посади" по полю "Код посади").

Фільтр: Фільтр для відображення співробітників окремих посад (На основі запиту "Відділ кадрів").

Варіант бази даних №2. Фірма з продажу персональних комп'ютерів.

Таблиці: Клієнти (Код, ПІБ, Вік, Пол, тип, вартість). Продавці (Код, ПІБ, Вік, Пол). Продавці і посади (Код, посади, Оклад, Обов'язки, Вимоги). Товар (Тип персональних комп'ютерів, вартість).

Замовлення (Зв'язує Таблиці "Клієнти" і "Товар" у контексті замовлення).

Фільтр: Фільтр для відображення клієнт, замовлення.

Варіант бази даних №3. Продаж автомобілів

Таблиці: Клієнти (Код, ПІБ, Вік, Пол, тип автомобілю, вартість). Продавці (Код, ПІБ, Вік, Пол). Продавці і посади (Код, посади, Оклад, Обов'язки, Вимоги). Автомобілі (Код, тип автомобілю, вартість, пробіг, технічний стан).

Замовлення (Зв'язує Таблиці "Клієнти" і "Автомобілі" у контексті замовлення).

Фільтр: Фільтр для відображення клієнт, замовлення.

Варіант бази даних №4. База даних кінотеатру

Таблиці: Театрали (Код, ПІБ, Вік, Пол, замовлення). Продавці (Код, ПІБ, Вік, Пол). Продавці і посади (Код, посади, Оклад, Обов'язки, Вимоги). Білет (Код, дата, номер місця, час, назва вистави).

Замовлення (Зв'язує Таблиці "Театрали" і "Білет" у контексті замовлення).

Фільтр: Фільтр для відображення замовлень по певним датам.