МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра комп'ютерних наук та прикладної математики

«До захисту допущений»

Завідувач кафедри

_____

___ _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

Безперервна інтеграція та доставка (CI/CD) для серверних застосунків на Python із використанням Jenkins

Виконав: Пахальчук Богдан Ігорович _____
(підпис)

група ІПЗ-41інт

Керівник: к. т. н. доц. Ярощак Сергій Вікторович _____
(науковий ступінь, вчене звання, посада, прізвище, ініціали) (підпис)

Рівне - 2025

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 54 с., 67 рисунків, 10 джерел.

Мета роботи: Дослідження особливостей використання Jenkins при реалізації процесу CI/CD для серверних застосунків на Python

Об'єкт дослідження: Використання CI/CD при розгортанні серверних застосунків на Python

Предмет дослідження: Процес проєктування та впровадження CI/CD-інфраструктури для серверного застосунку на Python з використанням Jenkins

Методи вивчення є моделювання та тестування

Побудовано CI/CD-архітектуру для автоматизованої доставки та обслуговування серверного застосунку. Спроєктовано й реалізовано REST-веб-службу управління готелем, що охоплює адміністраторів, гостей, номери й бронювання. Використовуючи Python 3.10 / FastAPI, систему контролю версій GitHub та Jenkins, здійснено повний цикл безперервної інтеграції, тестування і розгортання. Проведено серію експериментів із API-тестами, виконано їх детальний аналіз і підтверджено стабільність та масштабованість розробленої системи.

Ключові слова: Безперервна інтеграція та доставка (CI/CD), Jenkins, Pipeline, Rest API, MySQL

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – Операційна система

ВМ – Віртуальна машина

CI/CD - Continuous Integration and Continuous Delivery.

VCS - Version Control System

Worker – сервер, на якому виконуються задачі, надані Jenkins

ВСТУП

Під час створення великої програми потрібно зробити багато дрібних, але важливих кроків: зібрати весь код докупи, перевірити, що він працює, а тоді встановити нову версію на сервер. Якщо це робити вручну, легко щось переплутати — натиснути «не ту» кнопку, пропустити тест, скопіювати файли не в ту папку. Через такі дрібні помилки програма може зламатися або затриматися з виходом.

Щоб цього уникнути, розробники дедалі частіше покладають ці рутинні дії на сам комп’ютер. Спеціальні інструменти стежать за кожною зміною у коді: щойно хтось із команди додає новий файл або виправляє помилку, система одразу сама — без участі людини — перевіряє програму, запускає всі тести й, якщо все гаразд, встановлює оновлену версію на тестовому чи робочому сервері.

Завдяки такій автоматизації:

- помилок, спричинених «людським фактором», стає набагато менше;
- нові можливості й виправлення потрапляють до користувачів швидше;
- команда замість монотонної ручної роботи зосереджується на створенні нового функціоналу.

Саме опис і впровадження такої автоматизованої схеми для серверних програм на Python буде детально розглянуто та описано в цій роботі.

РОЗДІЛ 1. БЕЗПЕРЕРВНА ІНТЕГРАЦІЯ ТА БЕЗПЕРЕРВНА ДОСТАВКА. ПРИНЦИП РОБОТИ CI/CD

1.1. Поняття та сутність безперервної інтеграції

Безперервна інтеграція – це одна з DevOps практик, яка використовується для оптимізації процесу доставки коду, цілями якої є мінімізувати помилки, пришвидшити збирання коду та підвищити якість продукту. Безперервна інтеграція дозволяє розробникам регулярно об'єднувати зміни коду в репозиторії, в якому відбувається розробка, після чого автоматично виконується збирання, тестування та запуск [1-3].

За останні роки безперервна інтеграція (CI) стала однією з провідних практик у розробці програмного забезпечення, тож більшість команд уже працюють із нею або поступово впроваджують цей підхід.

Передумови успішного впровадження CI:

- повністю автоматизоване збирання коду;
- автоматичний запуск тестів;
- миттєвий доступ до актуальної версії коду для кожного члена команди;
- спільний репозиторій вихідних файлів;
- прозорість та відстежуваність усіх етапів процесу.

1.2. Принципи та етапи роботи безперервної інтеграції

Весь вихідний код зберігається у спільному репозиторії, куди розробники постійно завантажують (комітять) свої зміни за допомогою системи контролю версій, наприклад Git. Із цього репозиторію автоматизований CI-сервер ініціює збірку та паралельно запускає юніт- і інтеграційні тести, щоби впевнитися, що нові правки не порушили роботу інших компонентів. Якщо збірка завершується невдачею, сервер точно визначає етап, на якому сталася помилка, даючи команді змогу оперативно її усунути. Такий CI-процес виконується багато разів на день: після кожного коміту система одразу тестує й збирає код. Після успішної збірки реліз можна зробити вручну, але, як правило, команда DevOps додатково автоматизує проєкт, налаштовуючи безперервну доставку та розгортання коду.

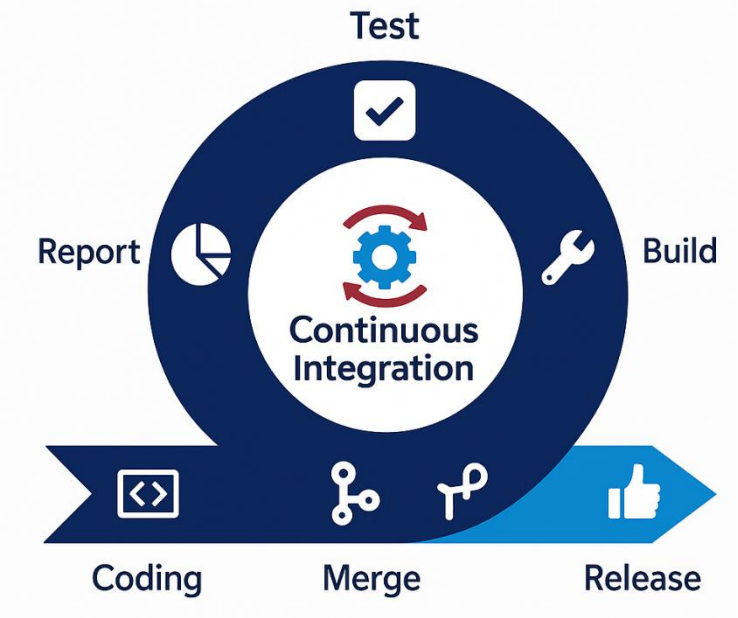

Рис. 1.1. Ілюстрація циклу безперервної інтеграції

1.3. Поняття безперервної доставки й безперервного розгортання

Безперервна доставка – це підхід в розробці програмного забезпечення, при якому програмний продукт після виконання безперервної інтеграції автоматично готується і ведеться реліз в виробництво [2].

За такого підходу розробку виконують короткими циклами, що підтримує стабільність і дає змогу виводити програму в експлуатацію будь-коли.

Безперервне розгортання – це процес, який націлений на розгортання нової версії застосунку в виробниче середовище, вносячи зміни, які є видимими для користувачів програмного забезпечення. Зазвичай цей етап не виділяють як окремий, включаючи розгортання в процес доставки [2].

Процеси неперервної інтеграції, доставки й розгортання об’єднуються в єдину методологію — CI/CD (Continuous Integration / Continuous Delivery), завдяки якій розробники створюють програмний продукт зручніше, швидше та надійніше.

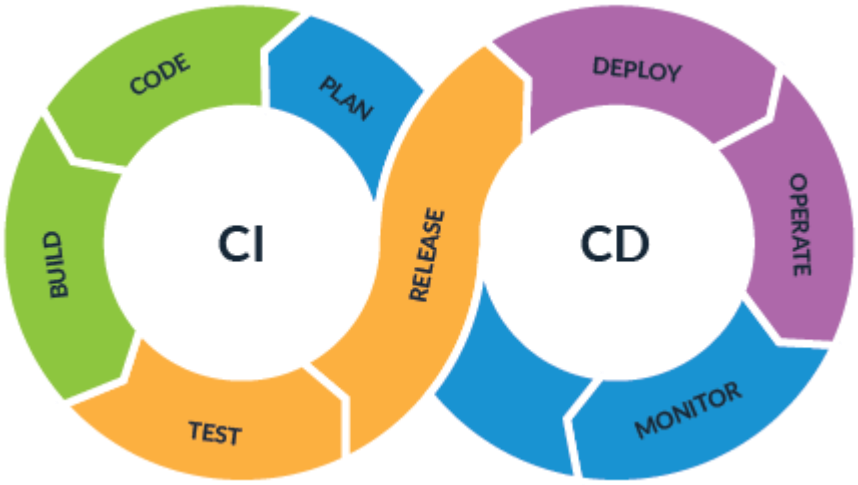

Рис. 1.2. Ілюстрацію циклу CI/CD

1.4. Основні принципи CI/CD

Для CI/CD існують чотири керуючих принципи:

- **Спільна відповідальність**

  Усі учасники проєкту несуть відповідальність за кожну фазу життєвого циклу продукту. Бізнес-логістика проектується командою, DevOps-інженери керують «логістикою» коду, виконуються приймальні тести, а користувачі надають зворотний зв'язок щодо роботи продукту.

- **Мінімізація ризиків**

  Кожна команда прагне знижувати ризики: перевіряє правильність бізнес-процесів, відстежує досвід користувачів, забезпечує надійне зберігання й обробку даних тощо.

- **Короткий цикл фідбеку**

  Щоб швидко додавати нові функції й оперативно узгоджувати правки, необхідно автоматизувати збірку та тестування. Там, де без участі людини не обійтися, слід зменшувати кількість інформаційних посередників.

- **Єдине робоче середовище**

  Розробники працюють у спільному середовищі з основною та допоміжними гілками, що дозволяє контролювати версії, якість, прийнятність і відмовостійкість коду. На фінальному етапі тестування програму також оцінюють із погляду безпеки.

1.5. Етапи життєвого циклу CI/CD

До основних етапів циклу CI/CD належать:

- **Написання коду**

  Розробники створюють або змінюють функціонал у локальних гілках та регулярно пушать коміти до спільного репозиторію Git. Це забезпечує актуальний та спільно підтримуваний код-бейс.

- **Збирання**

  Щоразу після коміту CI-сервер автоматично запускає збірку: встановлює залежності, компілює або пакує проєкт і формує артефакт (наприклад, Docker-образ чи wheel-пакет). У результаті маємо відтворюваний пакет, готовий до тестування.

- **Ручне тестування**

  Після автоматичних перевірок QA-фахівці або розробники вручну проганяють критичні сценарії, UI-кейси та нестандартні ситуації, які складно охопити автотестами. Це підтверджує, що зміни не порушують ключову бізнес-логіку.

- **Реліз**

  Коли всі тести пройдено, обрану версію коду позначають тегом як стабільну, формують реліз-ноти та погоджують випуск. Готовий реліз-пакет можна зафіксувати в білд-сховищі та підготувати до виробництва.

- **Розгортання**

  CD-конвеєр автоматично (або «по кнопці») доставляє білд у цільове середовище — staging чи production, виконує міграції баз даних, і, за потреби, відкату версії. Користувачі отримують оновлений сервіс із мінімальними простоями.

1.6. Основні засоби реалізації CI/CD

Існує безліч інструментів для безперебійної інтеграція та доставки коду програм кінцевим користувачам, однак в цьому розділі буду описані найпопулярніші:

- **Spacelift**

Spacelift — це гнучка платформа CI/CD, яка підтримує робочі процеси Terraform, OpenTofu, Terragrunt, Pulumi, CloudFormation, Ansible та Kubernetes. Вона спеціалізується на інфраструктурі як коді та забезпечує оптимізований підхід до CI/CD, спрощуючи визначення конвеєрів та інтеграцію з іншими інструментами.

Основні особливості:

- Гнучкість: використовуйте власні образи, контролюйте, що відбувається до і після кожної фази виконання, та інтегруйте будь-які сторонні інструменти.
- Інтеграція VCS: інтегрується з популярними постачальниками VCS, такими як GitHub, GitLab, BitBucket та Azure DevOps.
- Політики на декількох рівнях прийняття рішень: ви можете контролювати, скільки схвалень потрібно для запуску, які ресурси можна створити, які параметри можуть мати ці ресурси, що відбувається, коли відкрито запит на витяг, і куди надсилати дані сповіщень.
- Інтеграція з хмарою: динамічні тимчасові облікові дані для AWS, Azure та GCP.
- Виявлення та усунення відхилень: забезпечує надійність вашої інфраструктури шляхом виявлення та усунення відхилень.
- Контексти: багаторазові змінні середовища та підключені файли.
- Інфраструктура самообслуговування з Blueprints: ви можете визначити шаблони інфраструктури, які можна легко розгорнути. Ці шаблони можуть містити вбудовані політики/інтеграції/контексти/виявлення відхилень для надійного розгортання.
- Залежності стеків: створюйте залежності між стеками та передавайте результати з одного до іншого. Це може допомогти легко побудувати конвеєр просування середовища.
- Видимість: легко переглядайте всі розгорнуті ресурси та детальну інформацію про них.

- Можливість самостійного розміщення: Spacelift можна розмістити самостійно в AWS та AWS Gov Cloud.

Ліцензія/ціна: комерційна ліцензія з пробним періодом до двох місяців

❖ **Jenkins**

Jenkins — це високорозширюваний сервер автоматизації CI/CD на базі Java. Він є відкритим і самохостним і дозволяє автоматизувати, створювати та розгортати програмне забезпечення. Цей інструмент безперешкодно інтегрується з різними системами контролю версій, хмарними провайдерами та сторонніми додатками, що робить його універсальним вибором для сучасних середовищ розробки.

Ключові особливості Jenkins:

- Багатий набір плагінів, що інтегруються з усіма інструментами розробки, тестування та розгортання в галузі
- Простий інтерфейс
- Вбудовані вузли для розподіленого створення на декількох машинах
- Надійний підхід «пайплайн як код» з використанням Jenkinsfile (пайплайни на базі Groovy)
- Планування випуску білдів
- Просте налаштування середовища

Будучи одним з найстаріших інструментів CI/CD на ринку, Jenkins все ще має багато прихильників, але його популярність повільно знижується.

Ліцензія/ціна: відкритий код (ліцензія MIT)

❖ **Buddy**

Buddy CI/CD — це платформа автоматизації, призначена для оптимізації та прискорення процесів розробки та розгортання програмного забезпечення. З акцентом на простоті та ефективності, Buddy пропонує інтуїтивно зрозумілий інтерфейс, що дозволяє командам налаштовувати, контролювати та виконувати конвеєри з мінімальними зусиллями.

Основні особливості Buddy:

- Візуальний конструктор конвеєрів: інтуїтивно зрозумілий інтерфейс з функцією перетягування для створення робочих процесів CI/CD
- Кешування шарів Docker: швидше створення завдяки повторному використанню шарів кешування Docker.
- Великі можливості інтеграцій: можливість інтеграції з популярними службами VCS, хмарними службами та службами сповіщень.
- Паралельність: одночасне виконання завдань або їх постановка в чергу для оптимального використання ресурсів

Ліцензія/ціна: комерційна з безкоштовним тарифом

❖ **GitLab CI/CD**

GitLab CI/CD — це інтегрована функція платформи GitLab VCS, яка автоматизує робочий процес CI/CD. Завдяки уніфікованому інтерфейсу, що охоплює весь цикл розробки програмного забезпечення (SDLC), GitLab CI/CD забезпечує швидкі ітерації, надійне тестування та безпечне розгортання — і все це в середовищі GitLab. Вбудовані інструменти CI/CD дають командам можливість швидше та впевненіше впроваджувати зміни в код.

Основні функції GitLab CI/CD:

- Детальний огляд етапів конвеєра, завдань та статусів для оптимізованого моніторингу.
- Функція Auto DevOps — автоматична конфігурація CI/CD на основі найкращих практик, що скорочує процес ручного налаштування.
- Пряме підключення до Kubernetes для ефективного розгортання та масштабування додатків.
- Вбудований реєстр контейнерів — зберігання та управління образами Docker.
- Автоматичне сканування безпеки на наявність вразливостей та перевірка відповідності вимогам.

Ліцензія/ціна: відкрита та комерційна версії.

❖ **CircleCI**

CircleCI — це провідна хмарна платформа CI/CD, яка дає розробникам можливість швидко створювати, тестувати та розгортати свої додатки в будь-якому масштабі. Вона має широкі можливості налаштування, великий набір інструментів інтеграції та оптимізації продуктивності. Ці особливості зробили її улюбленим інструментом сучасних команд розробників, які прагнуть гнучкості та швидкості.

Основні особливості CircleCI:

- Створення складних конвеєрів CI/CD з паралельним, послідовним і ручним виконанням завдань
- Матричні збірки: одночасне виконання тестів у декількох версіях і середовищах
- Пакети конфігурацій, які можна ділитися та використовувати повторно, для спрощення створення та інтеграції конвеєрів
- Можливість налаштовувати ресурси процесора та оперативної пам'яті відповідно до конкретних вимог завдань.

Висновки до розділу 1

У цьому розділі було розглянуто ключові поняття, пов’язані з безперервною інтеграцією, безперервною доставкою та безперервним розгортанням, а також проаналізовано особливості їх функціонування. Розкрито основні етапи та принципи, що лежать в основі циклу CI/CD.

Встановлено, що впровадження безперервної інтеграції потребує дотримання низки умов, зокрема: автоматизованого збирання програмного коду, проведення тестування без участі людини, забезпечення постійного доступу до актуального стану репозиторію, а також прозорості всіх процесів розробки.

Окрім того, було проаналізовано інструменти, які використовуються для автоматизації тестування та доставки програмного забезпечення. Серед них виокремлено найбільш популярні CI/CD-системи, коротко охарактеризовано їхні основні можливості та функціонал.

РОЗДІЛ 2. ВСТАНОВЛЕННЯ ТА НАЛАШТУВАННЯ CI/CD СИСТЕМИ JENKINS

2.1. Проєктування CI/CD системи

Для створення CI/CD системи необхідно мати 3 середовища або віртуальних машини – одна для Jenkins, друга для автоматично розгортання тестового середовища, на якому будуть виконуватися тестування та третя для виробничого серверу.

Для віртуальних машин було обрано операційну систему Ubuntu Server 22.04, яка є найпоширеніша системою для серверних застосунків.

В CI/CD застосунку буде створено 3 завдання:

1. Автоматичне розгортання серверного застосунку на тестовому сервері, завантаження тестових даних в базу даних
2. В разі успішного розгортання серверного застосунку, виконання тестових API запитів з серверу Jenkins до тестового серверу для перевірки працездатності Rest API.
3. В разі успішного проходження всіх тестових API запитів, розгортання серверного застосунку на виробничому сервері.

Виконання завдання повинно виконуватися в такому порядку:

1. Першим завдання повинно виконуватися розгортання та компіляція серверного застосунку на тестовому сервері. Запускається воно вручну адміністратором Jenkins.
2. Друге завдання – виконання тестів – запускається автоматично після першого в разі успішного розгортання серверного застосунку на тестовому сервері
3. Третє завдання запускається відразу після успішного виконання другого для розгортання серверного застосунку на виробничому сервері

2.2. Встановлення та налаштування Jenkins

Програмне забезпечення Jenkins розроблене на Java, тому для його роботи необхідне встановлення Java на операційну систему:

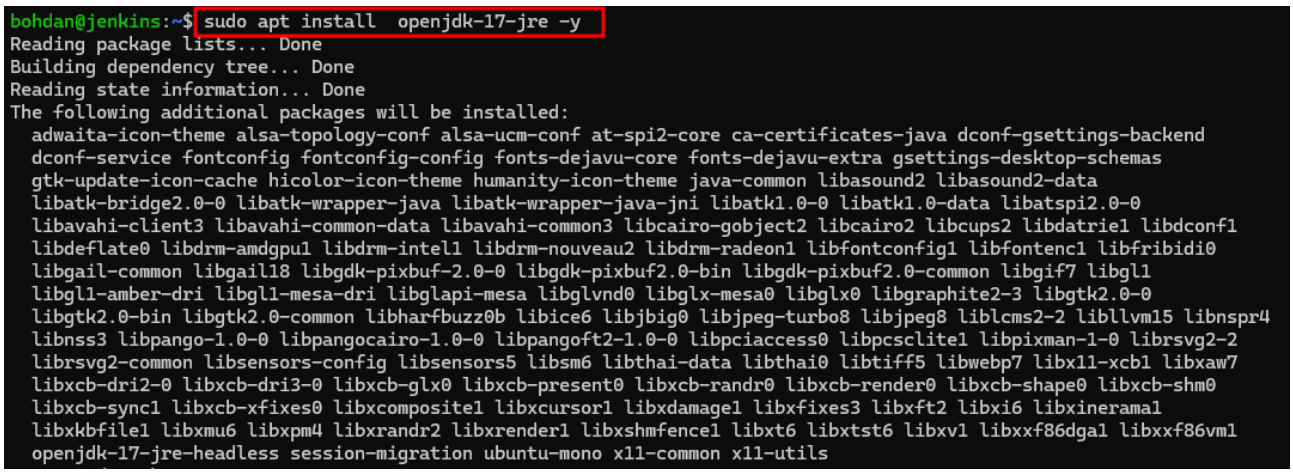

Рис. 2.1. Встановлення Java на ВМ

Перевіряємо версію Java, щоб переконатися що вона встановлена успішно:

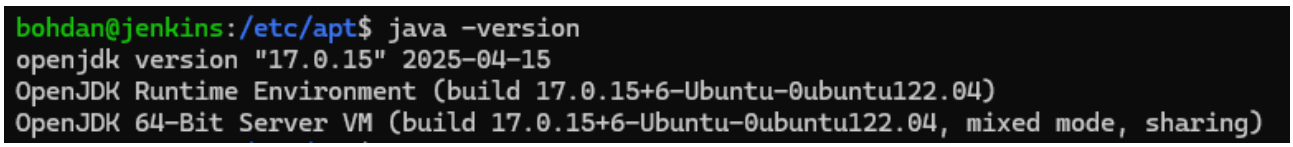

Рис. 2.2. Перевірка версії Java

Після встановлення Java можемо розпочати встановлення Jenkins. Добавляємо репозиторій Jenkins в список репозиторіїв та добавляємо згенерований ключ для доступу до репозиторію в пакетному менеджері apt.

Рис. 2.3. Добавлення репозиторію та ключа для встановлення Jenkins

Оновляємо список пакетів та встановлюємо Jenkins:

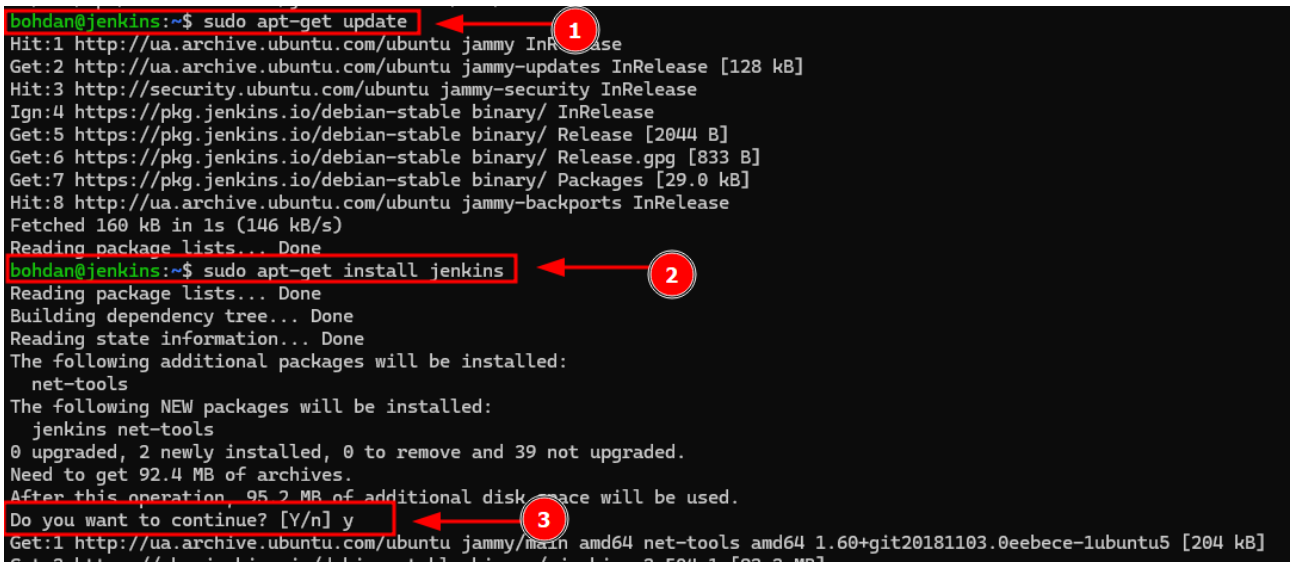

Рис. 2.4. Оновлення списку пакетів та встановлення Jenkins

Після успішного встановлення запускаємо службу та перевіряємо її статус:

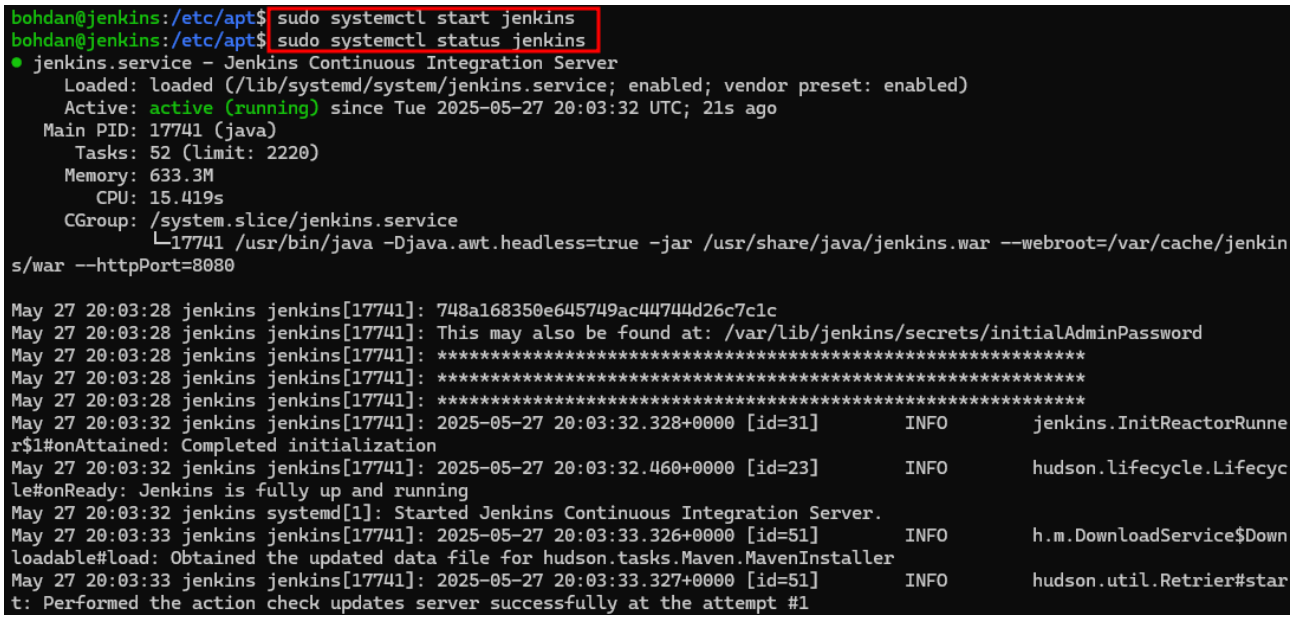

Рис. 2.5. Запуск та перевірка статусу служби Jenkins

Веб інтерфейс Jenkins за замовчуванням працює по порту 8080, і оскільки було налаштовано firewall на сервері, необхідно добавити в ufw правило на дозвіл підключення по порту 8080:

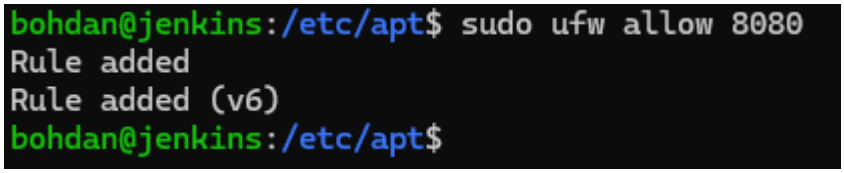

Рис. 2.6. Добавлення в ufw правило на дозвіл підключення по порту 8080

Спробуємо перейти в браузера на наступним посиланням: *http://<ip_address_server>:8080/*

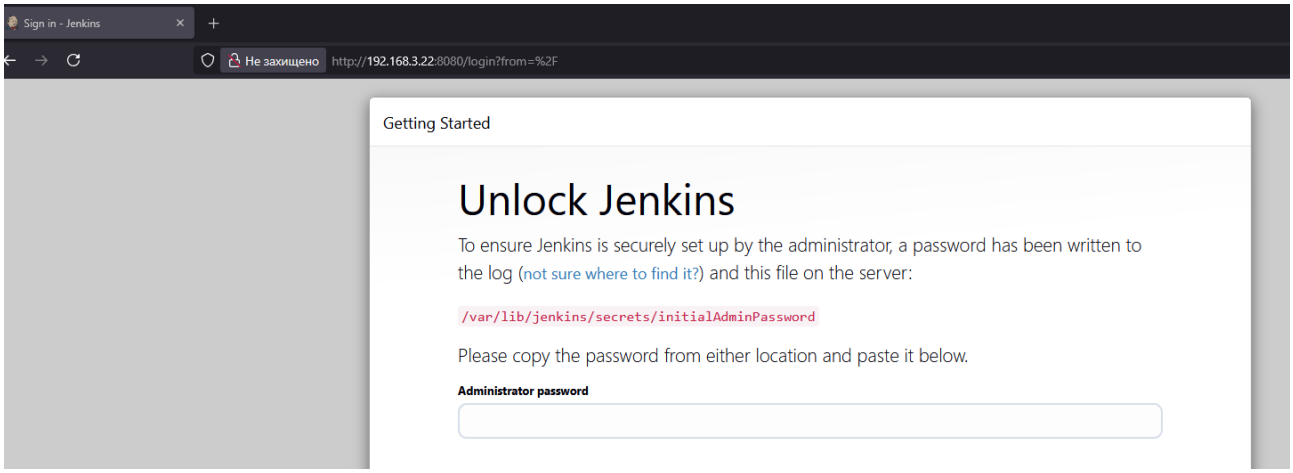

Рис. 2.7. Успішне завантаження веб інтерфейсу Jenkins

Для початкового налаштування Jenkins необхідно ввести пароль, який знаходить в вказаному файлі на сервері. Це зроблено для того, що якщо Jenkins розташовується на сервері з публічної IP адресою, то щоб ніхто посторонній не зміг провести початкове налаштування Jenkins.

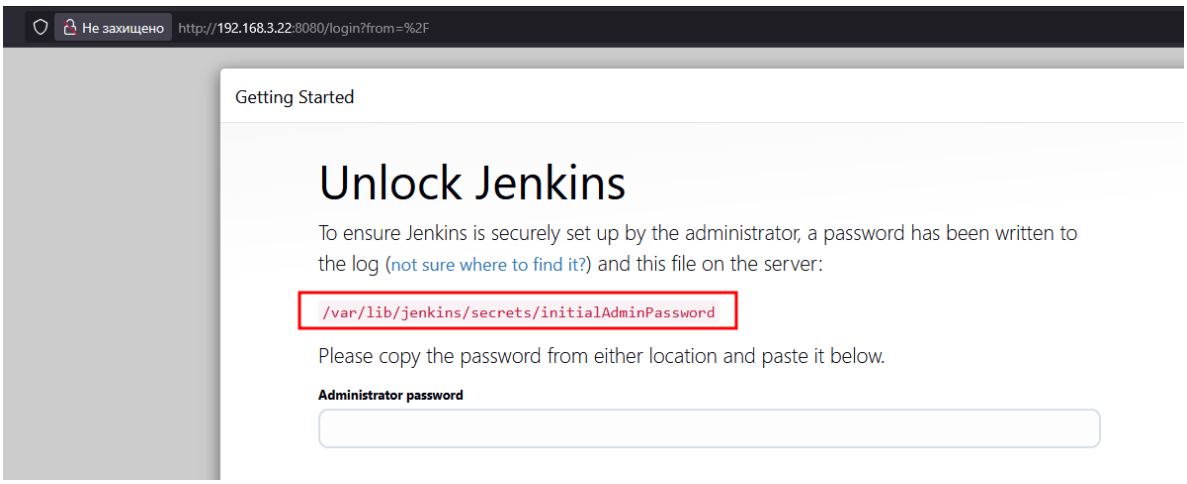

Рис. 2.8. Початкова сторінка Jenkins

Переходимо до вказаної директорії на сервері, копіюємо та вставляємо пароль на початку сторінку налаштування Jenkins:

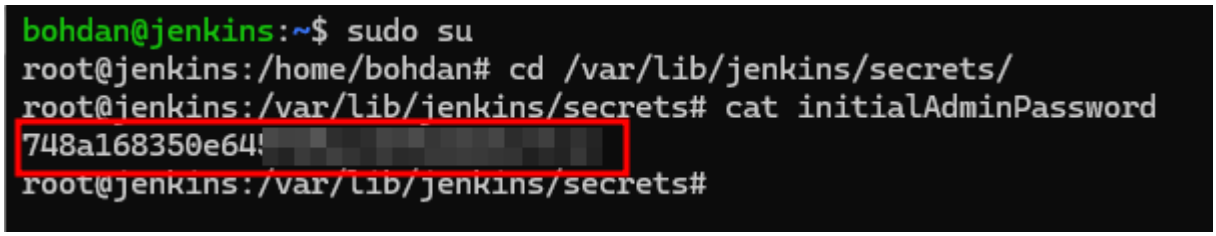

Рис. 2.9. Файл з початковим паролем на сервері Jenkins

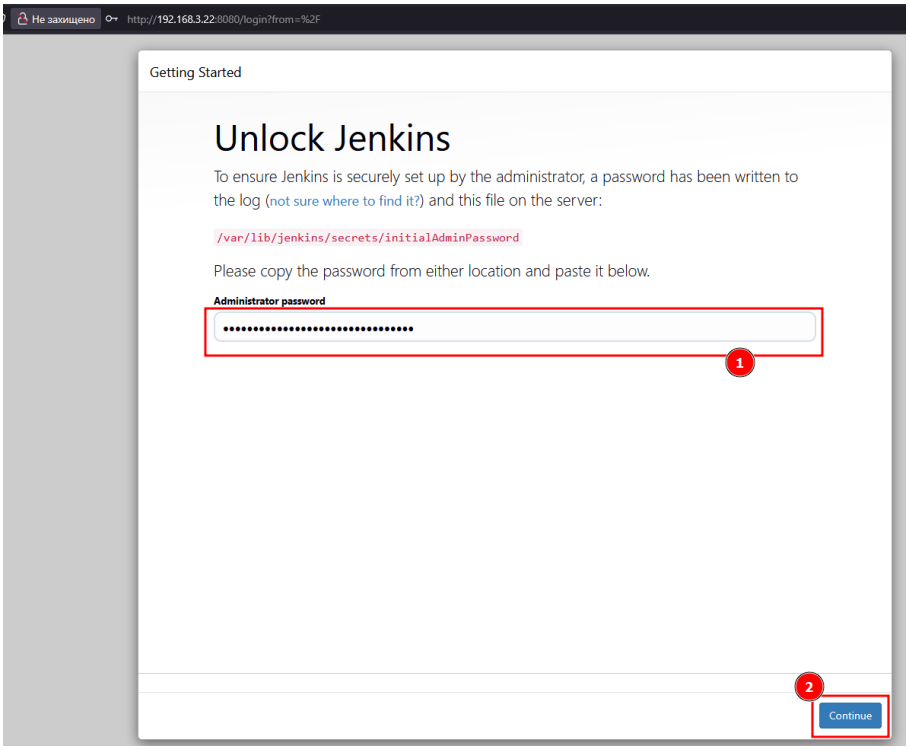

Рис. 2.10. Задання паролю адміністратора в веб інтерфейсі для продовження початкового налаштування Jenkins

Продовжуємо початкове налаштування Jenkins з встановлення бажаних плагінів:

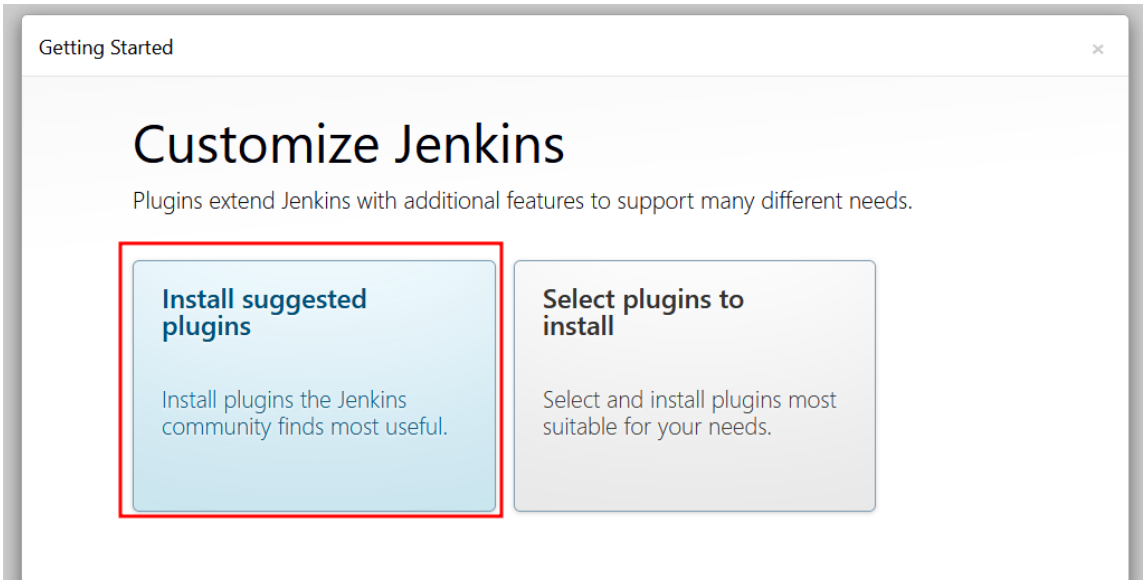

Рис. 2.11. Встановлення бажаних плагінів під час початкового налаштування Jenkins

Очікуємо завантаження та встановлення плагінів та продовжуємо налаштування Jenkins:

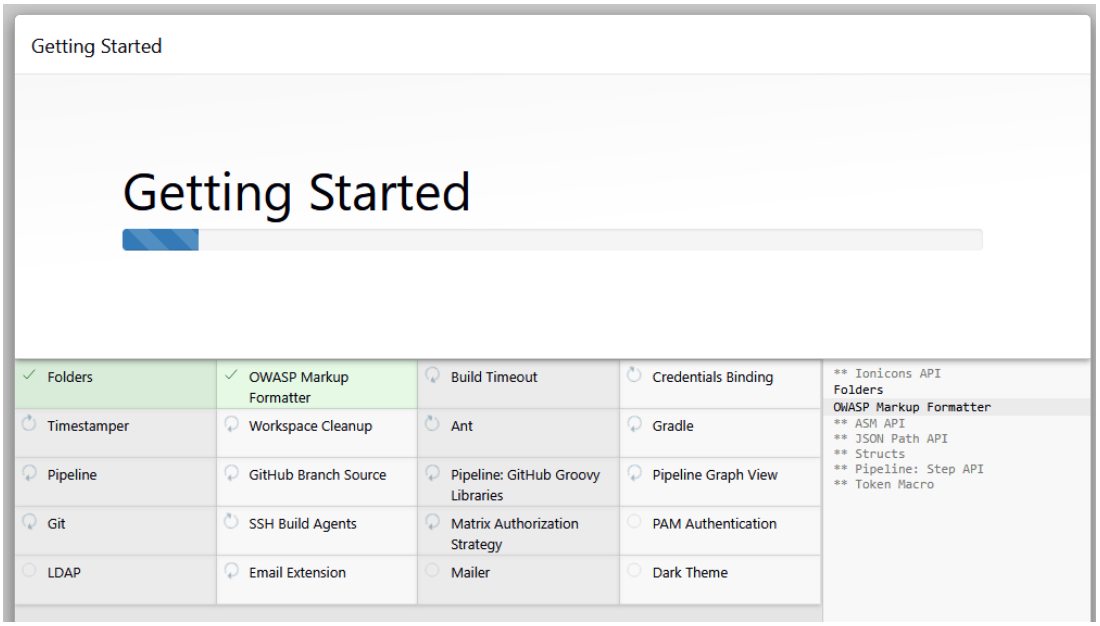

Рис. 2.12. Завантаження та встановлення плагінів Jenkins

На наступному етапі налаштування необхідно задати облікові дані користувача Jenkins:

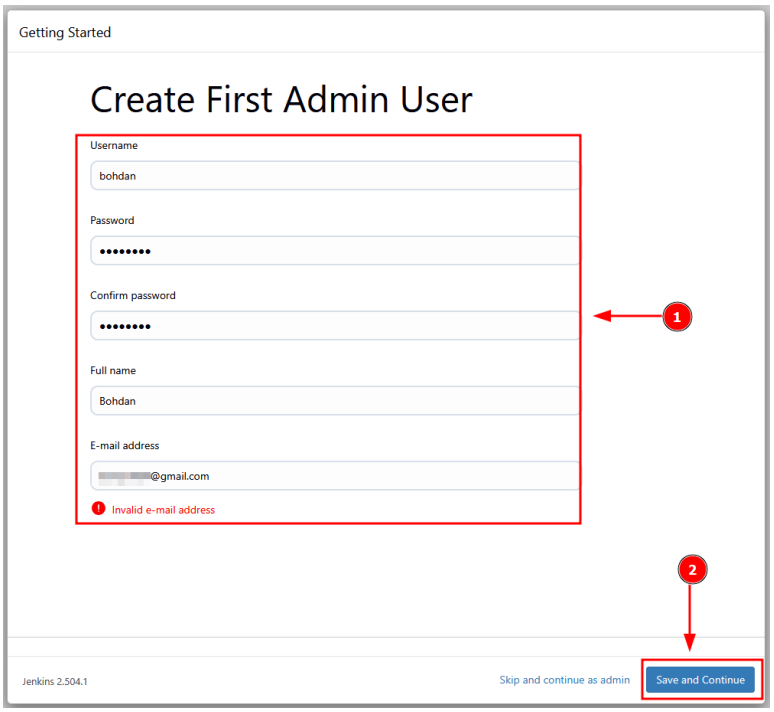

Рис. 2.13. Задання облікових даних користувача Jenkins

Підтверджуємо URL, по якому буде працювати Jenkins та завершуємо налаштування Jenkins:

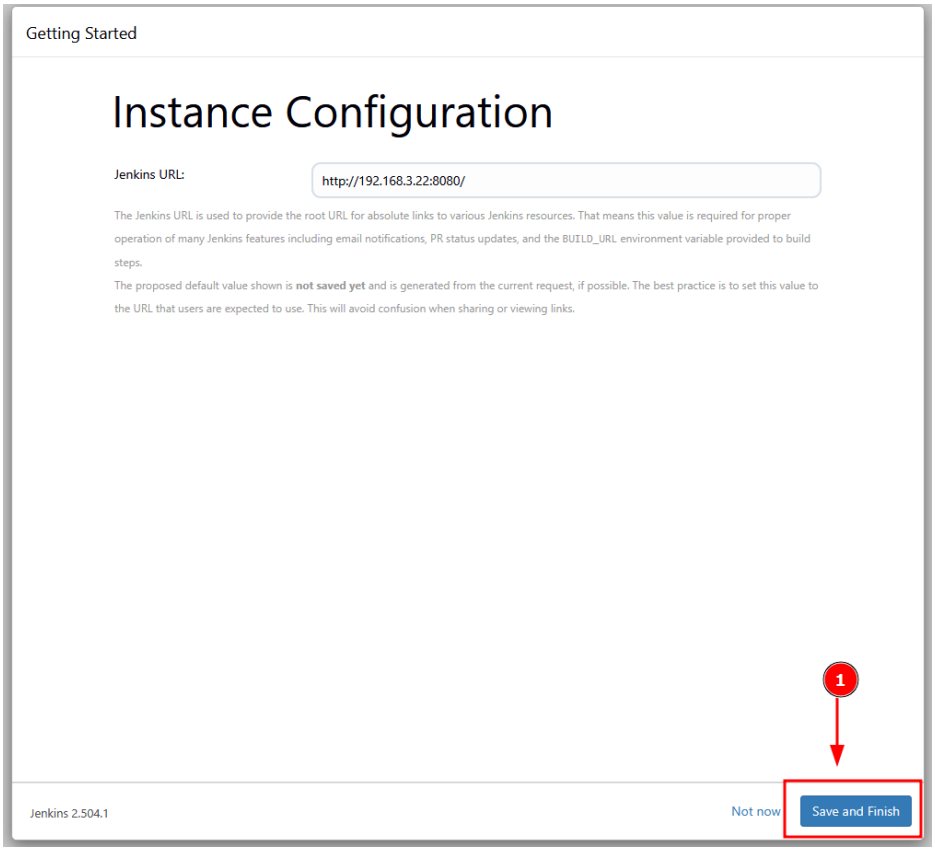

Рис. 2.14. Підтвердження URL та завершення початкового налаштування Jenkins

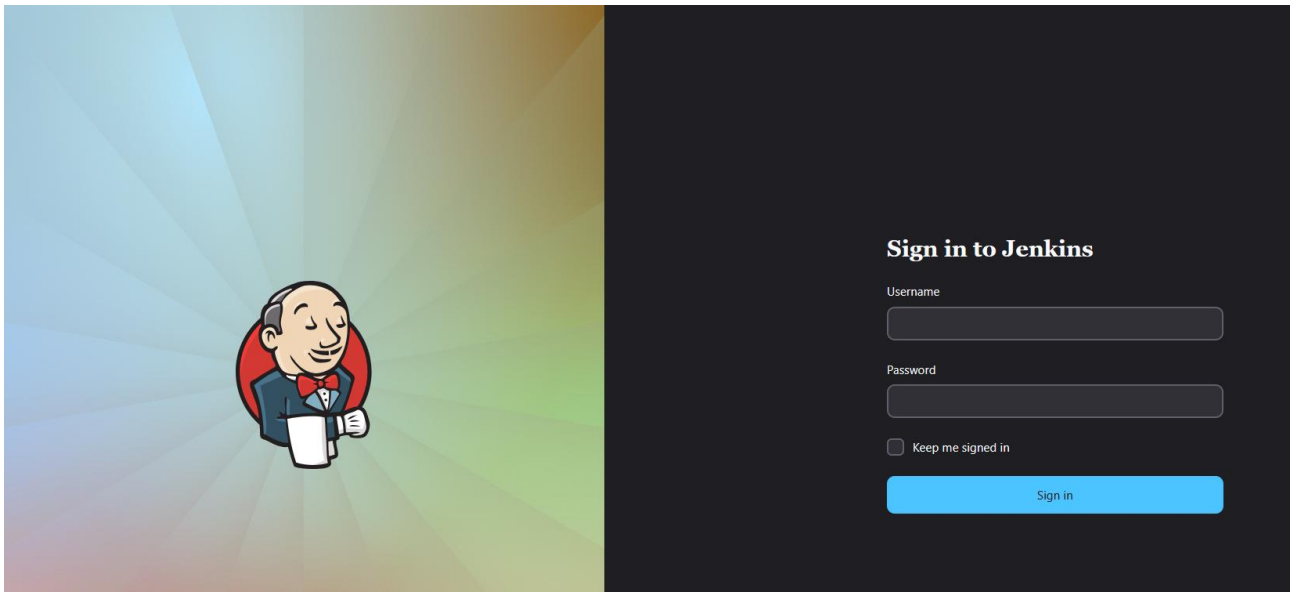

Рис. 2.15. Сторінка авторизації Jenkins

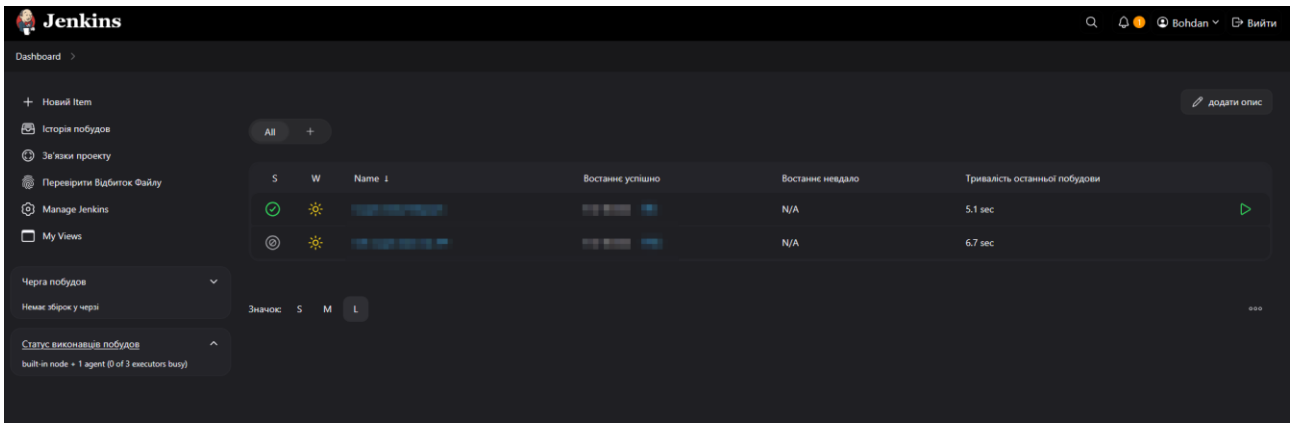

Рис. 2.16. Відображення дашборду Jenkins

2.3. Налаштування віртуальної машини для Jenkins Worker

Jenkins буде підключатися до тестового та виробничого через SSH, тому необхідно створити SSH ключ на сервері Jenkins:

```
bohdan@jenkins:~$ ssh-keygen -t ed25519 -C "jenkins [redacted]" -f ~/.ssh/jenkins_key
Generating public/private ed25519 key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/bohdan/.ssh/jenkins_key
Your public key has been saved in /home/bohdan/.ssh/jenkins_key.pub
The key fingerprint is:
SHA256:T+HfytWTYMpHVrdEZ+TCYt9MI8SR/j5uQq/uh2YW+ZY jenkins.[redacted]
The key's randomart image is:
+--[ED25519 256]--+
|            .ooo+|
|            .+.o.|
|          . +.o++|
|         . o ++=+|
|        S o  =+.o|
|         o..==.o.|
|          .oo.Ooo|
|           ..O E.|
|           O+B..|
+----[SHA256]-----+
```

Рис. 2.17. Створення SSH ключа

Приватну частину цього ключа необхідно завантажити в облікові дані (Credentials) Jenkins:

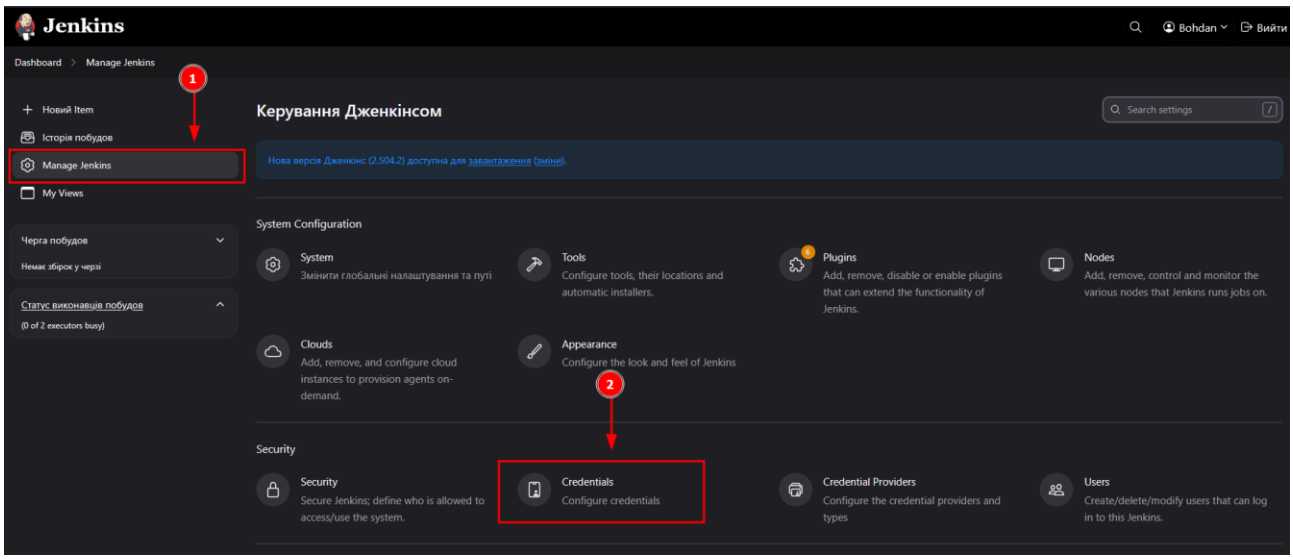

Рис. 2.18. Налаштування облікових даних (Credentials) в веб інтерфейсі Jenkins

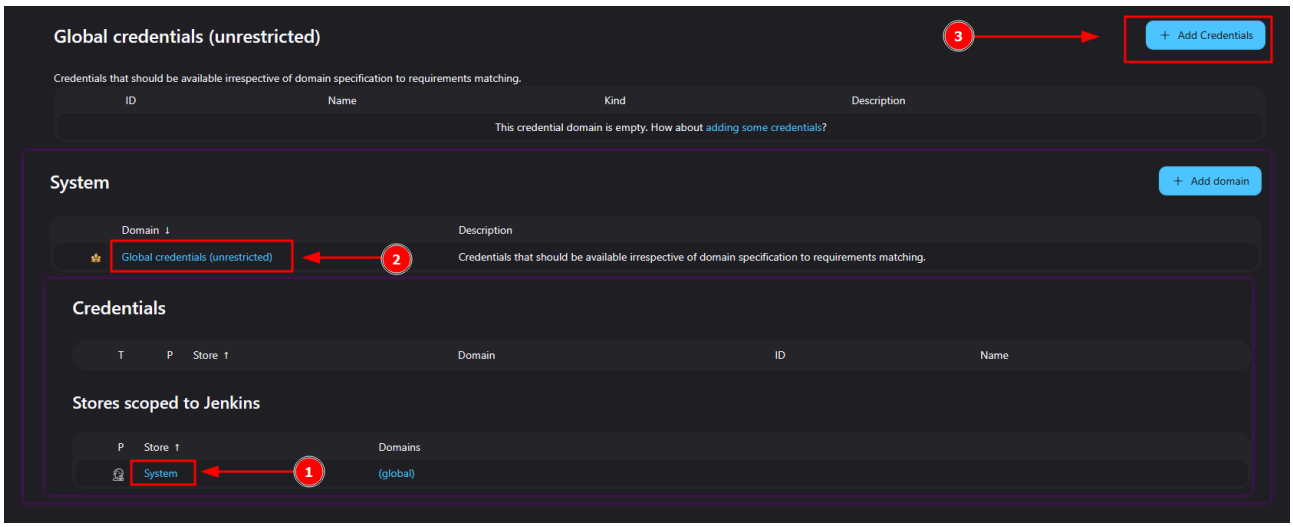

Рис. 2.19. Створення нових облікового запису в Jenkins

Рис. 2.20. Завантаження приватної частини ключ до облікового запису та його створення

Після чого необхідно публічну частину SSH ключу завантажити на Worker Jenkins сервери.

Після успішно налаштування підключення по SSH між серверами, необхідно добавити сервери worker як агентів в веб-інтерфейсі Jenkins:

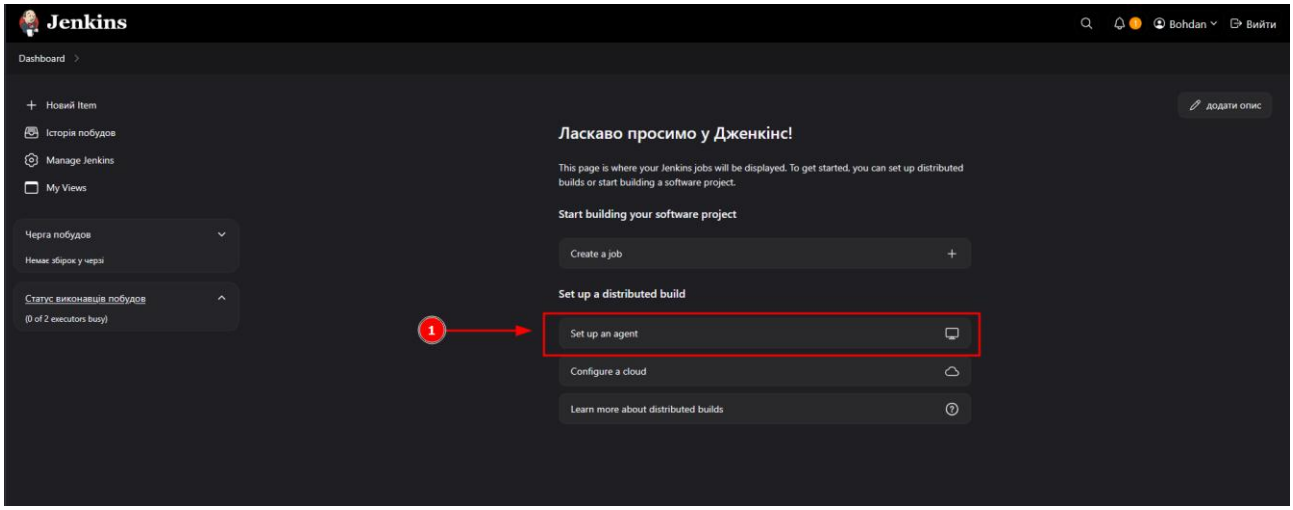

Рис. 2.21. Добавлення агента в Jenkins

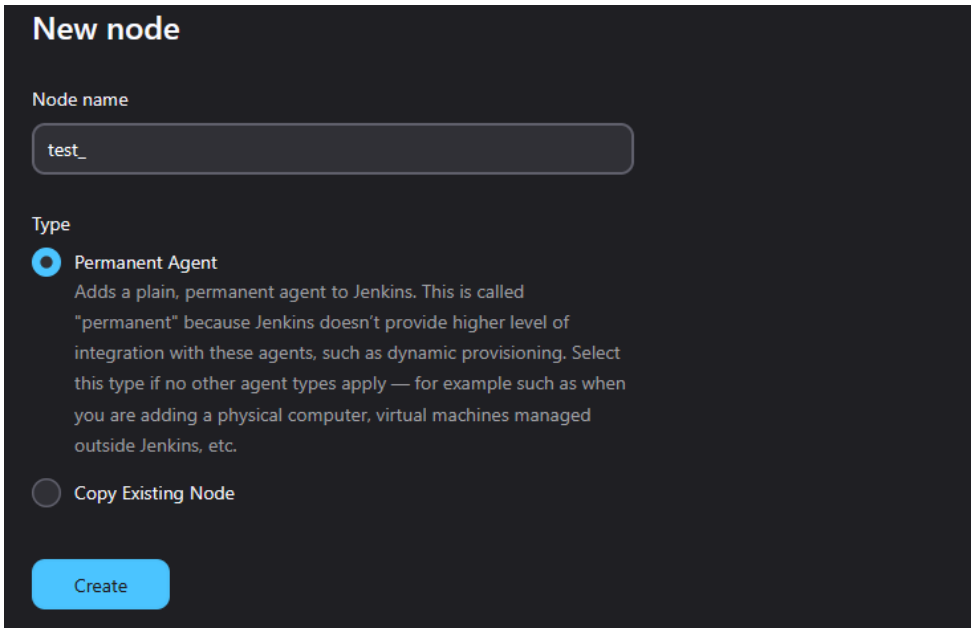

Рис. 2.22. Задання назви агента та вибір відповідного типу в Jenkins

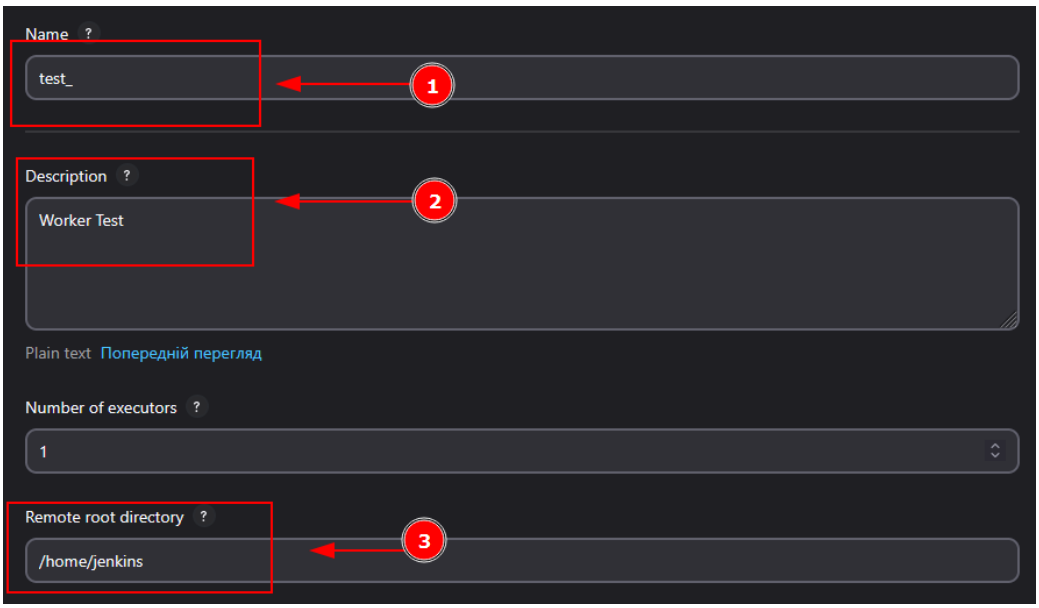

Рис. 2.23. Задання відповідних параметрів під час створення агенту

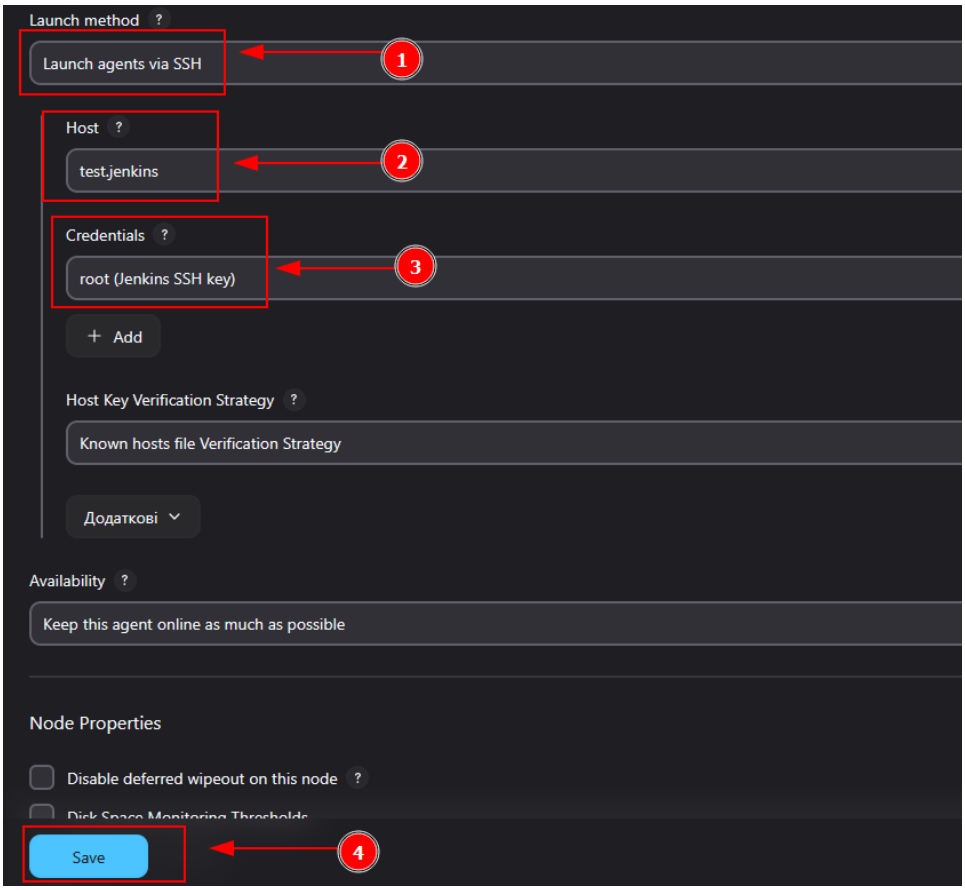

Рис. 2.24. Задання відповідних параметрів під час створення агенту

Після успішно добавлення агенту (worker) можна переглянути його логування і переконатися що він успішно підключений:

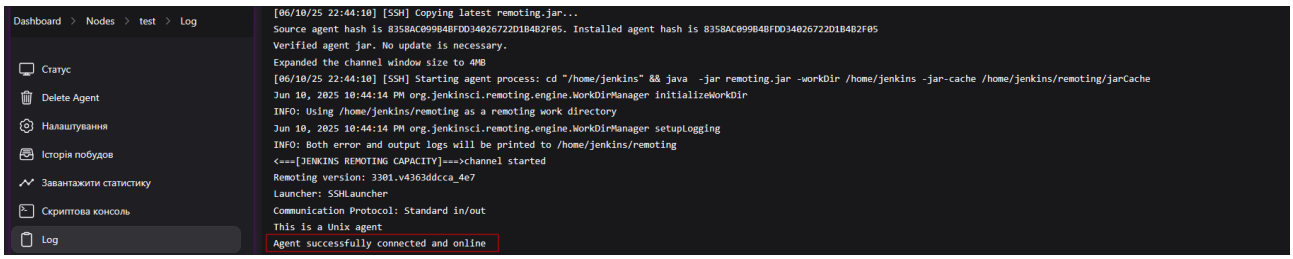

Рис. 2.25. Відображення файлу логування worker в Jenkins

Також переглянути стан підключення worker можна в списку nodes:

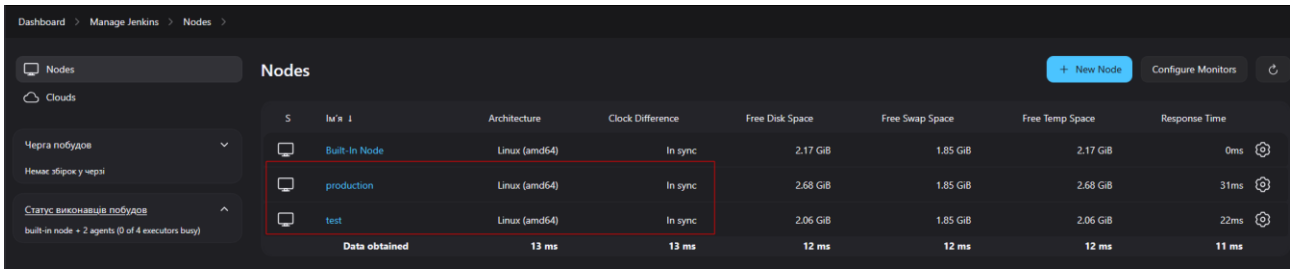

Рис. 2.26. Перегляд список nodes (workers) в Jenkins

2.3. Налаштування інтеграції Jenkins з GitHub

Для початку необхідно переконатися що необхідні плагіни для роботи GitHub встановлені на Jenkins. Для цього переходимо в налаштування плагінів та перевіряє наявність вказаних плагінів:

- Git plugin
- GitHub plugin
- GitHub Integration Plugin
- GitHub Branch Source Plugin

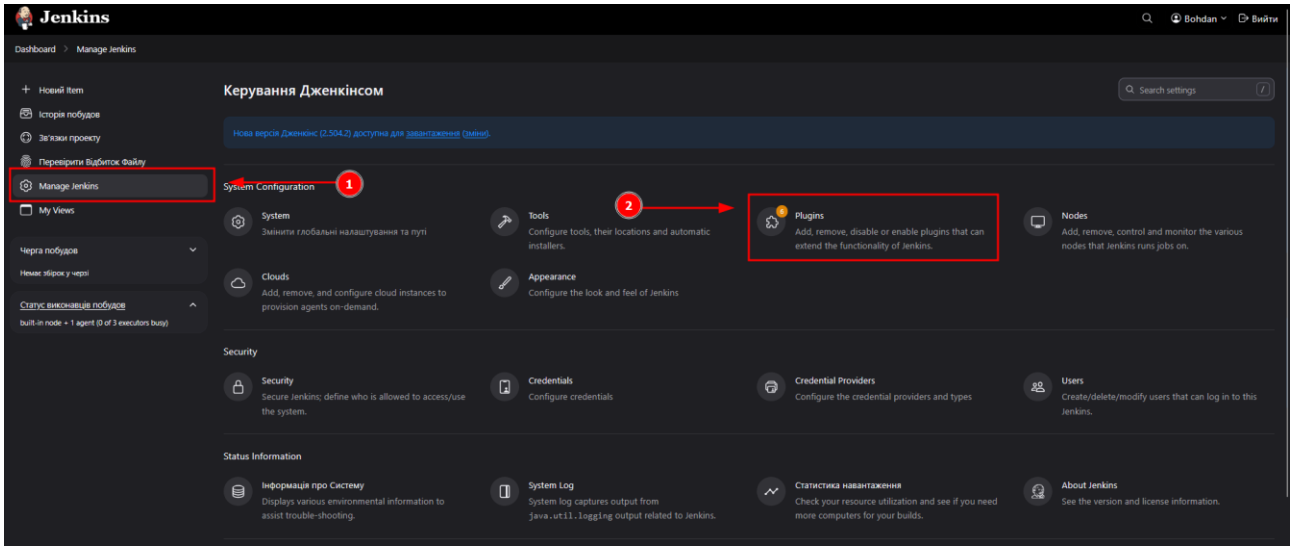

Рис. 2.27. Перехід в модуль плагінів в Jenkins

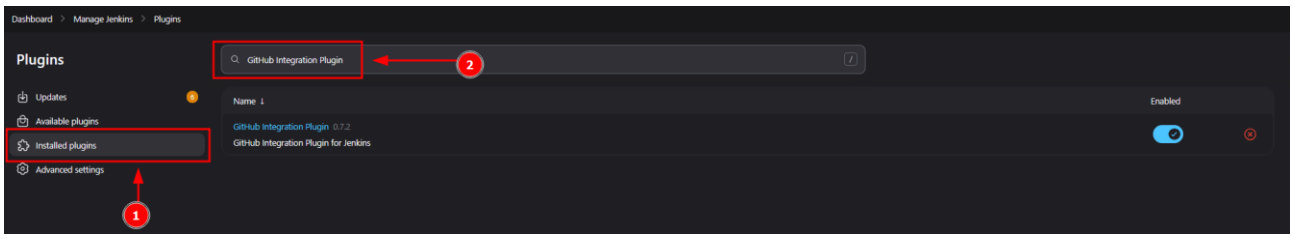

Рис. 2.28. Перевірка встановлення необхідні плагінів для інтеграції з GitHub

За необхідністю встановлюємо необхідні плагіни.

Оскільки до GitHub також буде підключатися по ssh, то для цього підключення необхідно створити новий ключ. Створюємо новий SSH ключ під назвою github та завантажуємо публічну частину в облікових запис GitHub:

Рис. 2.29. Створені новий SSH ключ для підключення до GitHub

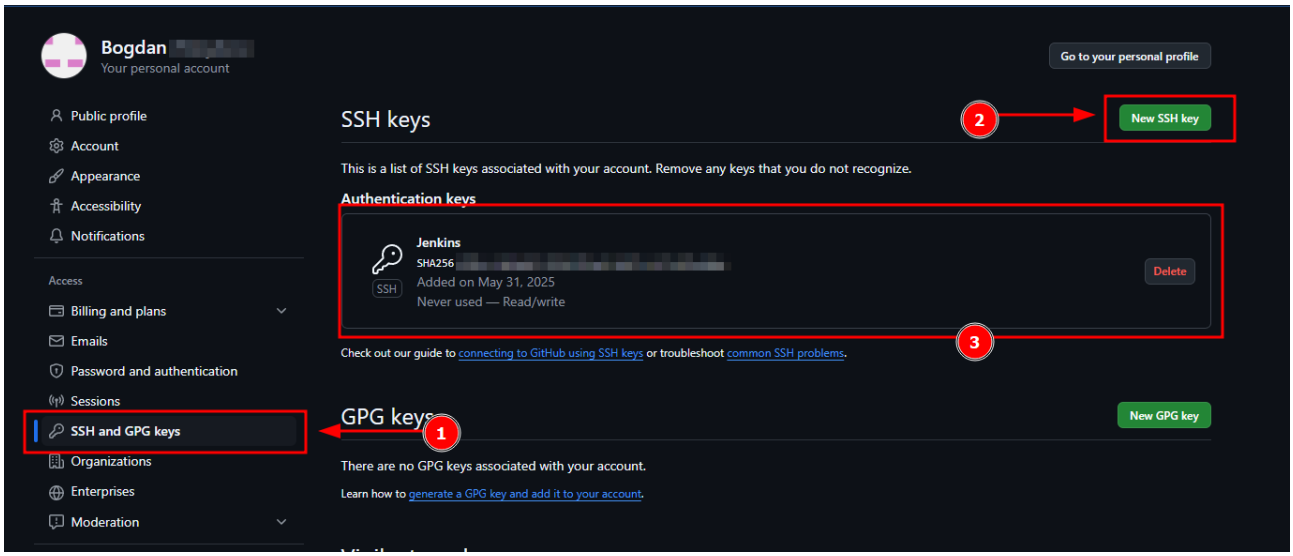

Рис. 2.30. Задання публічної частини ключа в обліковому записі GitHub

Приватну частину цього ключа необхідно задати в налаштування облікових даних в Jenkins:

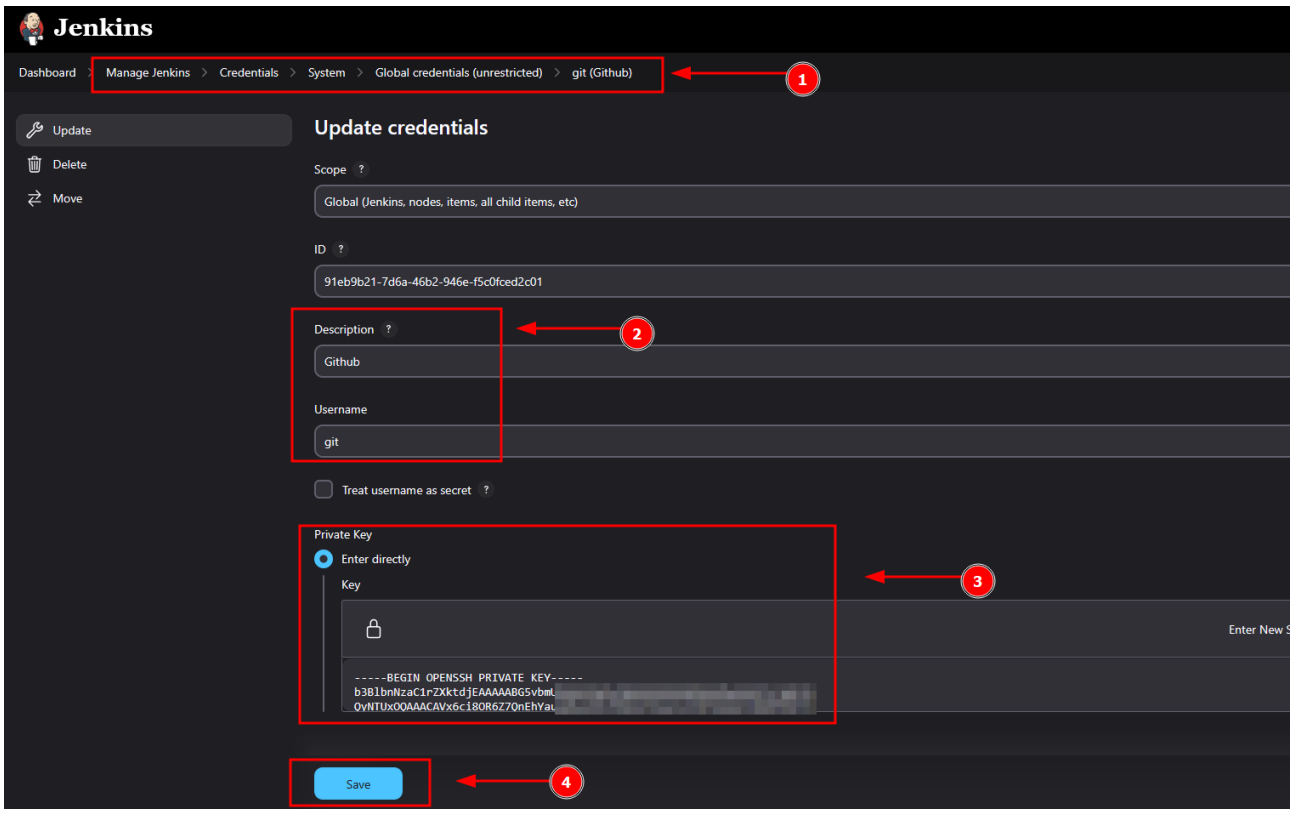

Рис. 2.31. Задання приватної частини ключа в налаштування облікових даних (credentials) в веб-інтерфейсі Jenkins

Висновки до розділу 3

В даному розділі було детально описано етапи встановлення та налаштування CI/CD системи Jenkins на операційній системі Ubuntu Server 22.04, підключення агентів до неї, підключення систему контролю версій GitHub для можливості автоматично завантаження коду останньої версії серверного застосунку.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РОЗРОБКА СЕРВЕРНОГО ЗАСТОСУНКУ

3.1. Вимоги серверного застосунку

У межах цієї роботи буде створено серверний застосунок — REST API, що взаємодіятиме з базою даних готелю для отримання, додавання, оновлення та видалення інформації. На тому ж веб-сервері, використовуючи Python, бібліотеку **Jinja2** та сервер **Uvicorn**, буде реалізовано сайт-інтерфейс до цього API.

База даних складатиметься з окремих таблиць, у яких зберігатимуться:

- відомості про гостей;
- дані адміністраторів готелю;
- перелік готельних номерів;
- інформація про бронювання кімнат;
- список адміністраторів API.

Також, UI інтерфейс повинен мати функцію реєстрації та авторизації для користувачів та адміністраторів, дії яких повинні виконуватись через API запити з локальною авторизацією (для запобігання можливого взлому).

3.2. База даних

В базі даних MySQL таблиці виглядають наступним чином:

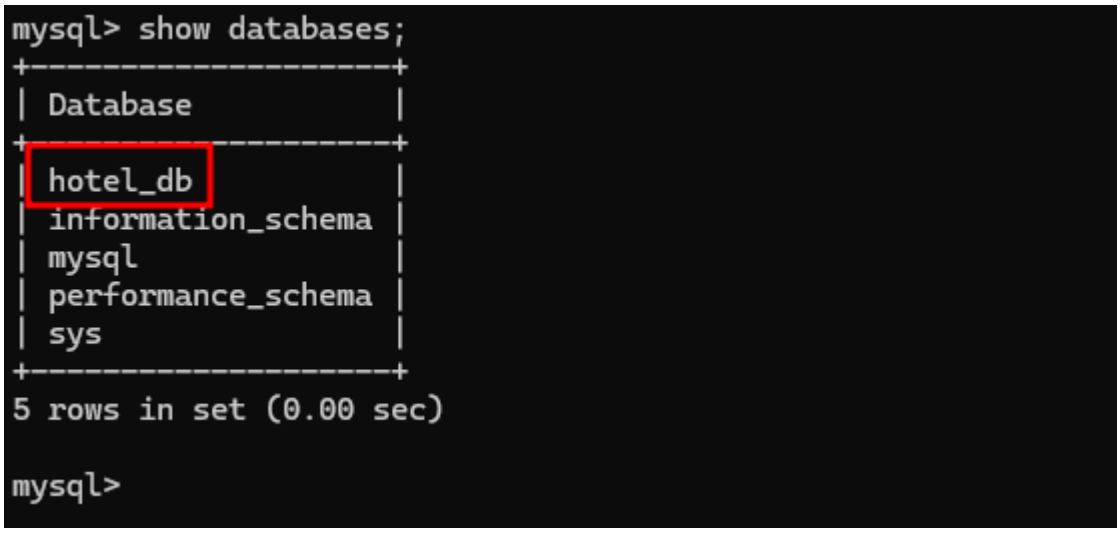

Рис. 3.1. Відображення бази даних hotel_db в MySQL

```
mysql> DESCRIBE administrators_hotel;
+---------------+-----------------------------------+------+-----+-------------------+-------------------+
| Field         | Type                              | Null | Key | Default           | Extra             |
+---------------+-----------------------------------+------+-----+-------------------+-------------------+
| admin_id      | int unsigned                      | NO   | PRI | NULL              | auto_increment    |
| username      | varchar(50)                       | NO   | UNI | NULL              |                   |
| password_hash | varchar(255)                      | NO   |     | NULL              |                   |
| first_name    | varchar(100)                      | NO   |     | NULL              |                   |
| last_name     | varchar(100)                      | NO   |     | NULL              |                   |
| email         | varchar(255)                      | NO   | UNI | NULL              |                   |
| phone         | varchar(30)                       | YES  |     | NULL              |                   |
| role          | enum('front_desk','manager','admin') | NO   |     | NULL              |                   |
| status        | enum('active','inactive')         | NO   |     | NULL              |                   |
| created_at    | timestamp                         | NO   |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+---------------+-----------------------------------+------+-----+-------------------+-------------------+
10 rows in set (0.01 sec)
```

Рис. 3.2. Структура таблиці administrators_hotel в MySQL

```
mysql> DESCRIBE admins_api;
+------------+-----------------------------------+------+-----+---------+----------------+
| Field      | Type                              | Null | Key | Default | Extra          |
+------------+-----------------------------------+------+-----+---------+----------------+
| id         | int unsigned                      | NO   | PRI | NULL    | auto_increment |
| login      | varchar(30)                       | NO   | UNI | NULL    |                |
| password   | varchar(60)                       | NO   |     | NULL    |                |
| privileges | enum('admin','user','reviewer')   | NO   |     | user    |                |
+------------+-----------------------------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Рис. 3.3. Структура таблиці admins_api в MySQL

```
mysql> DESCRIBE guests;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| guest_id      | int unsigned | NO   | PRI | NULL    | auto_increment |
| first_name    | varchar(100) | NO   |     | NULL    |                |
| last_name     | varchar(100) | NO   |     | NULL    |                |
| email         | varchar(255) | NO   | UNI | NULL    |                |
| phone         | varchar(30)  | YES  |     | NULL    |                |
| password_hash | varchar(255) | NO   |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

Рис. 3.4. Структура таблиці guests в MySQL

```
mysql> DESCRIBE rooms;
+-----------------+-----------------------------------------+------+-----+---------+----------------+
| Field           | Type                                    | Null | Key | Default | Extra          |
+-----------------+-----------------------------------------+------+-----+---------+----------------+
| room_id         | int unsigned                            | NO   | PRI | NULL    | auto_increment |
| room_number     | varchar(10)                             | NO   | UNI | NULL    |                |
| room_type       | enum('single','double','suite','family') | NO   |     | NULL    |                |
| capacity        | tinyint unsigned                        | NO   |     | NULL    |                |
| price_per_night | decimal(10,2)                           | NO   |     | NULL    |                |
| status          | enum('available','out_of_service')      | NO   |     | NULL    |                |
+-----------------+-----------------------------------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

Рис. 3.5. Структура таблиці rooms в MySQL

```
mysql> DESCRIBE reservations;
+----------------+-----------------------------------------------------+------+-----+-------------------+-------------------+
| Field          | Type                                                | Null | Key | Default           | Extra             |
+----------------+-----------------------------------------------------+------+-----+-------------------+-------------------+
| reservation_id | int unsigned                                        | NO   | PRI | NULL              | auto_increment    |
| guest_id       | int unsigned                                        | NO   | MUL | NULL              |                   |
| room_id        | int unsigned                                        | NO   | MUL | NULL              |                   |
| created_by     | varchar(10)                                         | YES  |     | NULL              |                   |
| check_in       | date                                                | NO   |     | NULL              |                   |
| check_out      | date                                                | NO   |     | NULL              |                   |
| status         | enum('confirmed','cancelled','completed','waiting') | NO   |     | waiting           |                   |
| total_amount   | decimal(10,2)                                       | NO   |     | NULL              |                   |
| created_at     | timestamp                                           | NO   |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+----------------+-----------------------------------------------------+------+-----+-------------------+-------------------+
9 rows in set (0.00 sec)
```

Рис. 3.6. Структура таблиці reservations в MySQL

3.3. Сервер Rest API

Сервер Rest API на Python був розроблений з використанням пакетів Uvicorn та FastAPI.

Всього наявно 5 кінцевих точок API:

- /api/admins_hotel – для отримання, створення та зміну інформації про адміністраторів готелю
- /api/admins_api – для отримання, створення та зміну інформації про адміністраторів API
- /api/guests - для отримання, створення та зміну інформації про гостей
- /api/rooms - для отримання, створення та зміну інформації про кімнати
- /api/bookings - для отримання, створення та зміну інформації про бронювання

При виконані API запитів необхідно використовувати авторизацію. В даному серверному застосунку API використовується базова (Basic) авторизація, яка передбачає кодування логіну (API key) та пароль (Secret) в форматі base64.

Наприклад, логін – root, пароль – toor. Логін та пароль записуються через двокрапку (root:toor) та кодуються в base64 - cm9vdDp0b29y. Облікові дані адміністраторів API зберігаються в базі даних в таблиці admins_api.

Якщо скористатися програмою для захвату і аналізу мережевого трафіку, можна побачити в заголовку (Header) HTTP протоколу параметр авторизації:

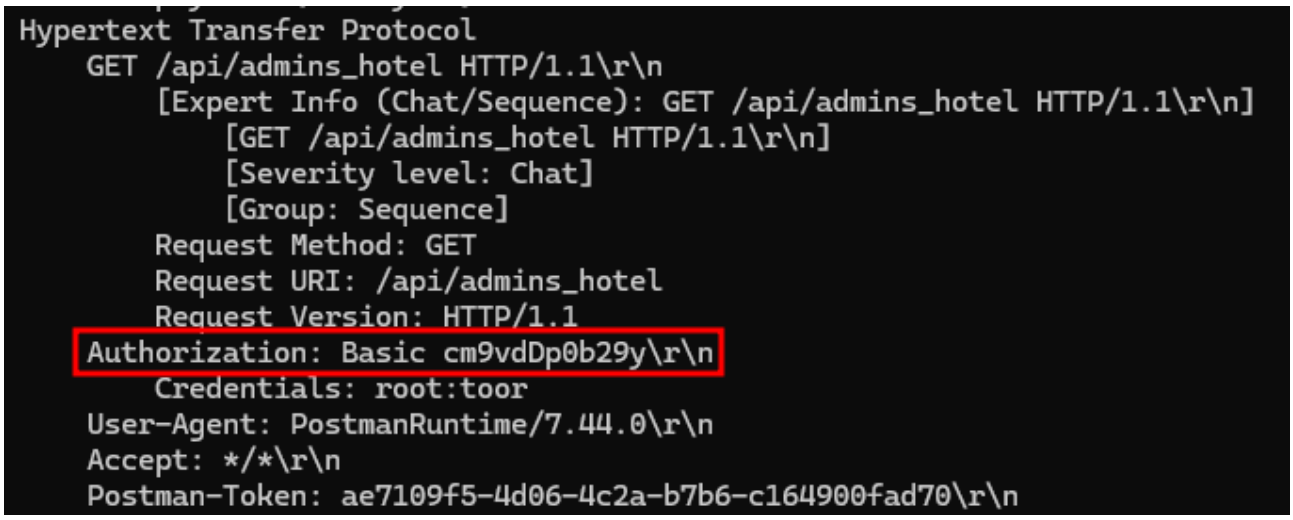

Рис. 3.7. Відображення параметру авторизації в HTTP протоколу при здійснені API запиту

При виконання API запитів (так само, як і HTTP(S)) є 4 основних методи:

- GET – Отримання інформації, даних з серверу
- POST – Надсилання інформації до серверу (при використання API це буде створення нових об’єктів)
- PUT – Оновлення вже існуючої інформації
- DELETE – Видалення інформації

Формати даних при використанні API можуть бути типу JSON, XML та YAML. JSON формат є найпопулярнішим, тому я його використовував в моєму серверному застосунку.

JSON формат передбачає об’єкти, в якому міститься пари ключ-значення, записані через двокрапку:

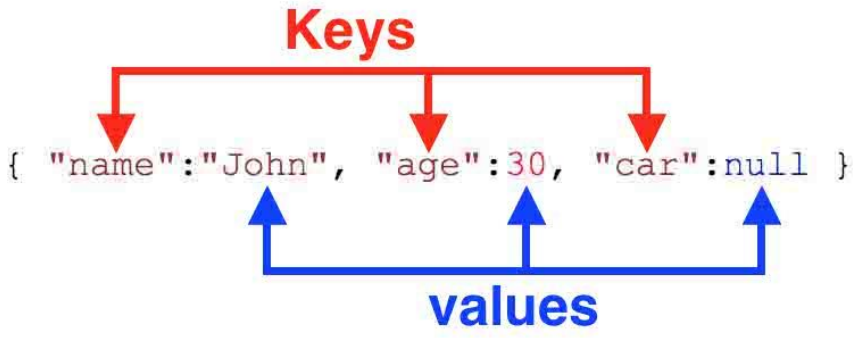

Рис. 3.8. Структура JSON формату

API запити до серверу будуть виконуватися за допомогою програми Postman – однієї з найкращої програми для роботи з API.

Нижче будуть наведені приклади використання API:

Отримання всіх адміністраторів готелю (рис. 2.!. , де 1 – позначено метод HTTP запиту, 2 – Повний URL запиту, разом з кінцевою точкою API)

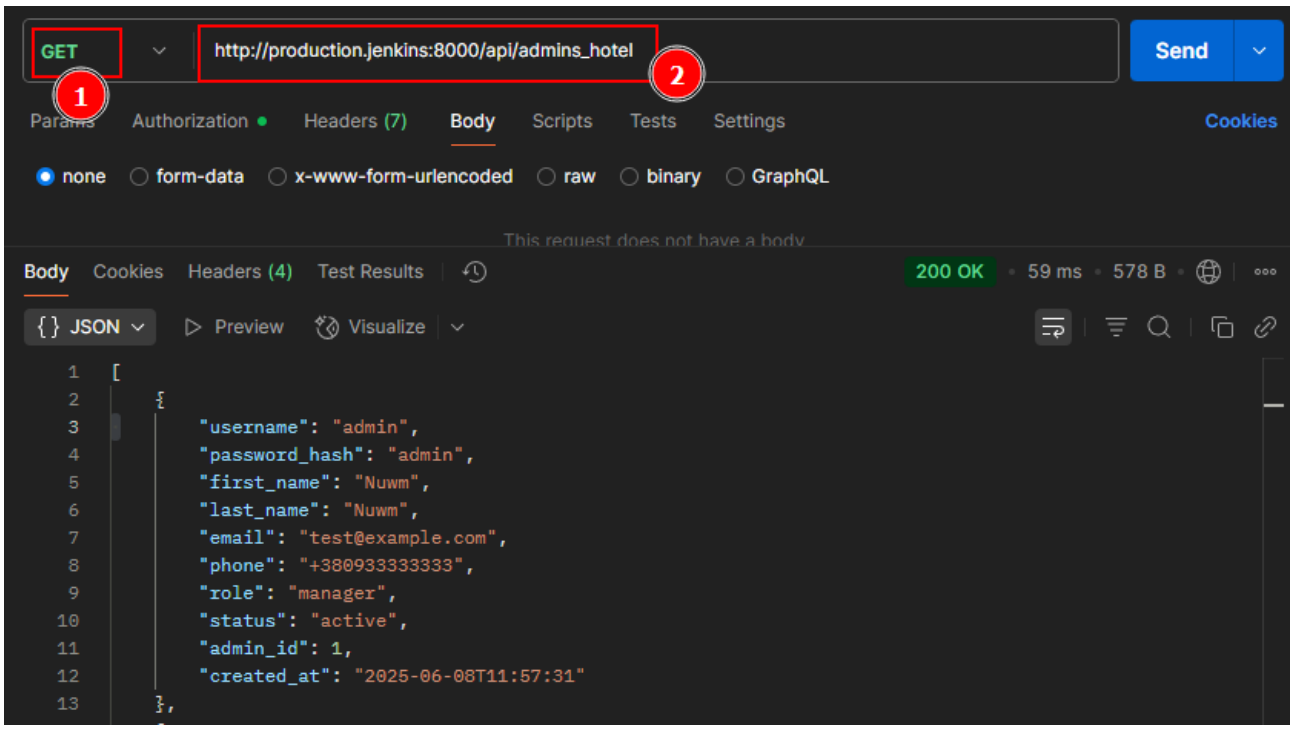

Рис. 3.9. Приклад API запиту, для отримання інформації про всіх адміністраторів готелю

Нижче наведений уривок коду з серверу Rest API, який відповідає за отримання адміністраторів готелю:

```
@api_router.get(
    "/admins_hotel",
    response_model=List[schemas.AdminHotelReadSchema],
    dependencies=[Depends(admin_only)])
def list_administrators(
    admin_id:      Optional[int]  = Query(None, description="= exact match по admin_id"),
    username:      Optional[str]  = Query(None, description="= exact match по username"),
```

first_name: Optional[str] = Query(None, description="= exact match по first_name"),

last_name: Optional[str] = Query(None, description="= exact match по last_name"),

email: Optional[str] = Query(None, description="= exact match по email"),

phone: Optional[str] = Query(None, description="= exact match по phone"),

role: Optional[str] = Query(None, description="= exact match по role"),

status: Optional[str] = Query(None, description="= exact match по status"),

admin_id_in: Optional[List[int]] = Query(None),

username_in: Optional[List[str]] = Query(None),

first_name_in: Optional[List[str]] = Query(None),

last_name_in: Optional[List[str]] = Query(None)

Для створення нових записів (гостей, адміністраторів, кімнат, тощо) необхідно використовувати метод POST з вказанням в тілі запиту всі параметрів елементу.

На рис. 3.9. відображено приклад API запиту для створення кімнати, де під номер 1 – типу методу, 2 – вкладка body, де потрібно добавити дані до запиту, 3 – JSON формат даних, 4 – вміст самих даних до запиту, 5 – відповідь сервера

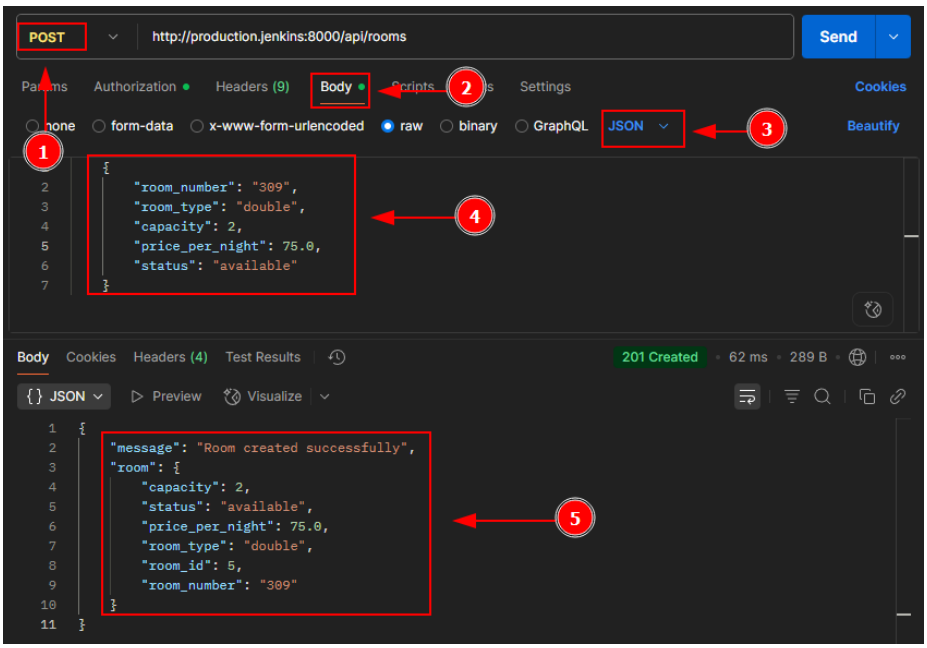

Рис. 3.10. Приклад API запиту, для створення нової кімнати

Уривок з коду серверу Rest API, який відповідає за створення кімнати:

```
api_router.post(
    "/rooms",
    status_code=201,
    dependencies=[Depends(auth.admin_required)],
)
def create_room(
    payload: schemas.RoomCreateSchema,
    db: Session = Depends(auth.get_db),
):
    room = models.Room(**payload.dict())
    try:
        created_room = crud.create(room, db)
        return { "message": "Room created successfully",
            "room": created_room }
    except IntegrityError as e:
        db.rollback()
        duplicate_msg = e.orig.args[1]
        if "Duplicate entry" in duplicate_msg:
            dup_value = duplicate_msg.split("'")[1] if "'" in duplicate_msg else
payload.room_number
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail=f"Room with number '{dup_value}' already exists"
            )
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=f"Cannot create room: {duplicate_msg}"
        )
    except Exception as e:
```

```
        db.rollback()
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=f"Cannot create room: {str(e)}"
        )
```

Для оновлення даних вже в існуючих запитах необхідно використовувати метод PUT та в кінці URL запиту вказувати унікальний ідентифікатор елементу.

Наприклад, оновимо для кімнати з ID 5 наступні значення полів (на рис. 3.10. під номером 4):

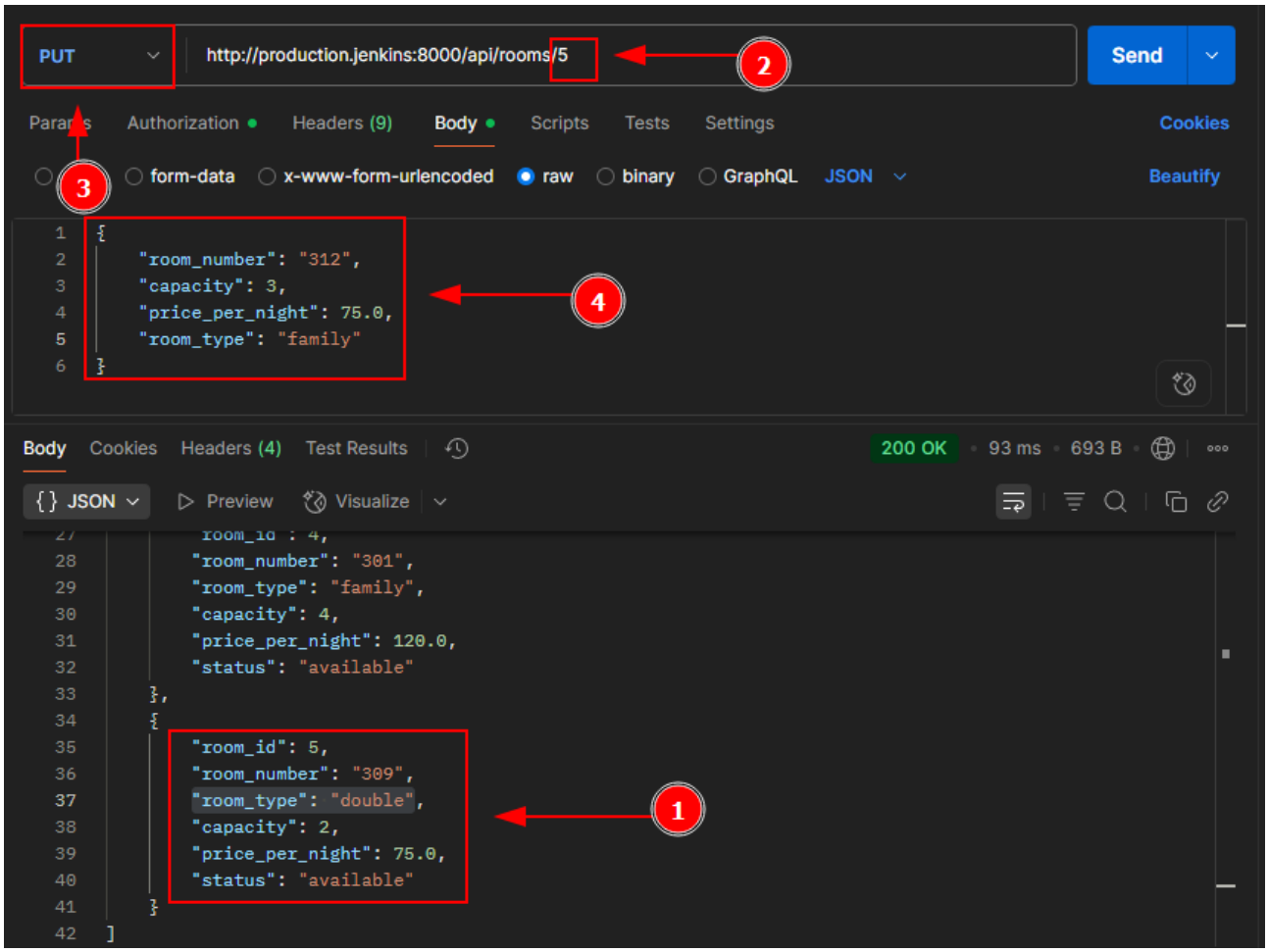

Рис. 3.11. Приклад API запиту оновлення кімнати

Для видалення запитів необхідно використовувати метод DELETE та в кінці API запиту також вказувати унікальний ідентифікатор запису, який необхідно видалити:

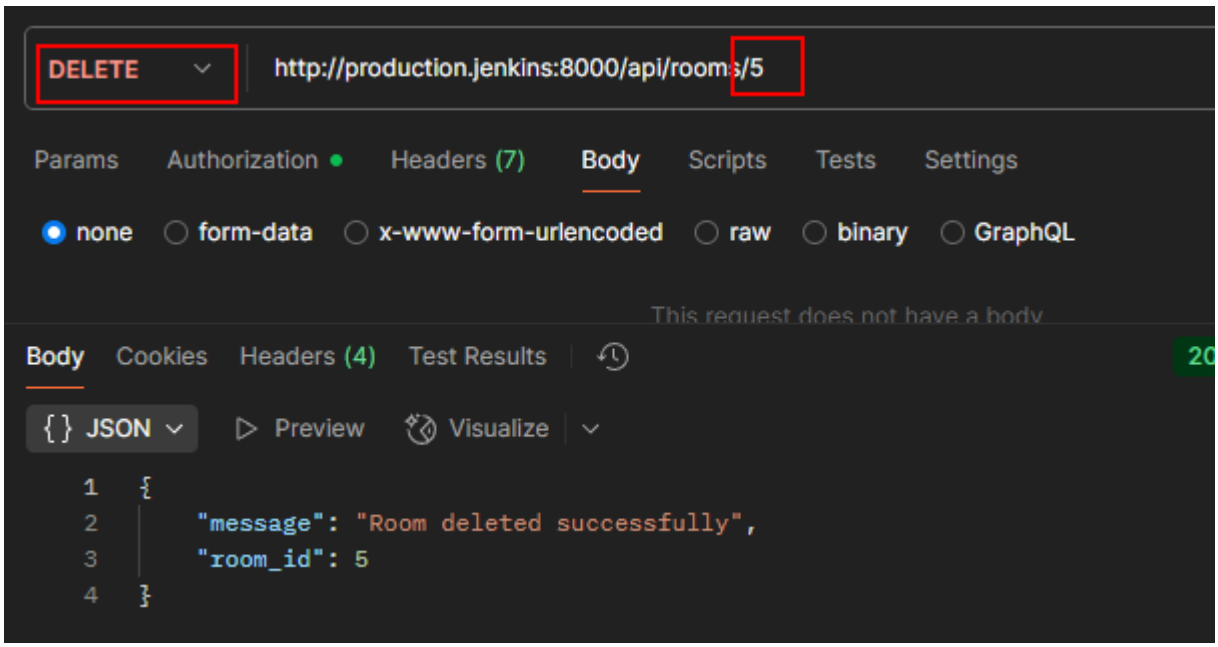

Рис. 3.12. Приклад API запиту видалення кімнати з ID 5

3.4. Веб-інтерфейс

Для прикладу використання API було розроблено веб-інтерфейс, через який можуть реєструватися та авторизуватися гості, бронювати свої кімнати, а також можуть авторизуватися адміністратори та переглядати всі бронювання з можливість їх видалення.

Головна сторінка веб сайт виглядає наступним чином:

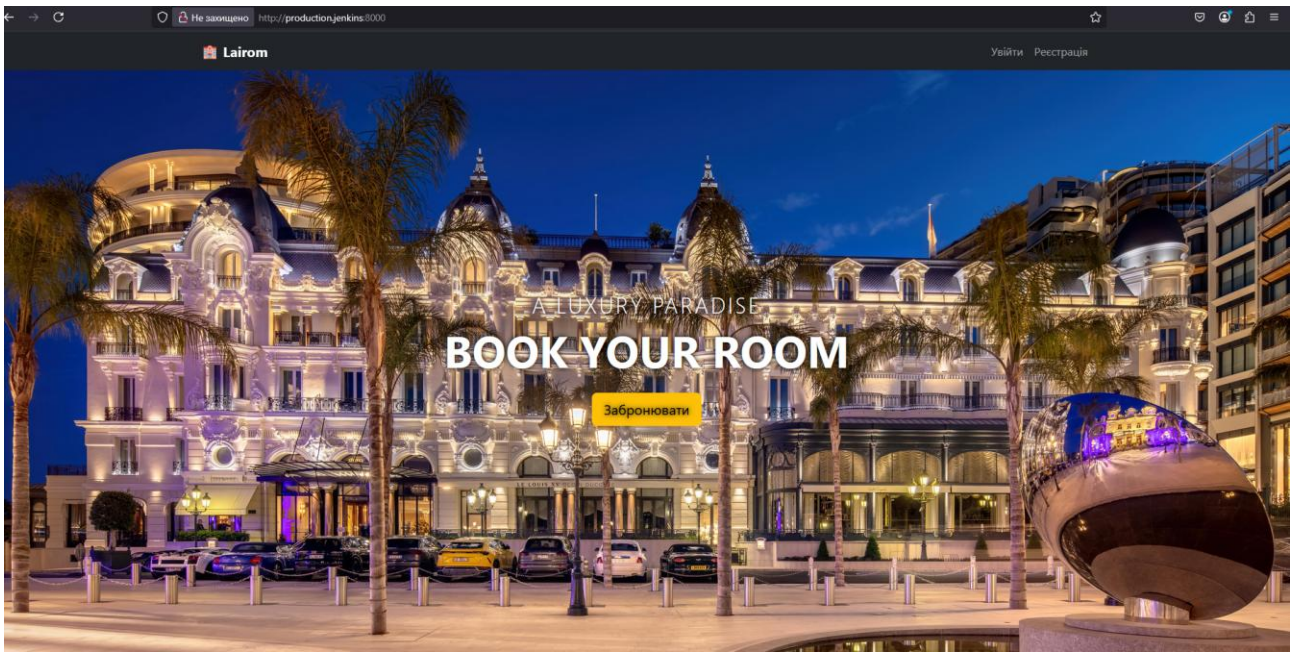

Рис. 3.13. Головна сторінка веб-сайту

Справа зверху наявні кнопки для переходу сторінку реєстрації та авторизації гостей:

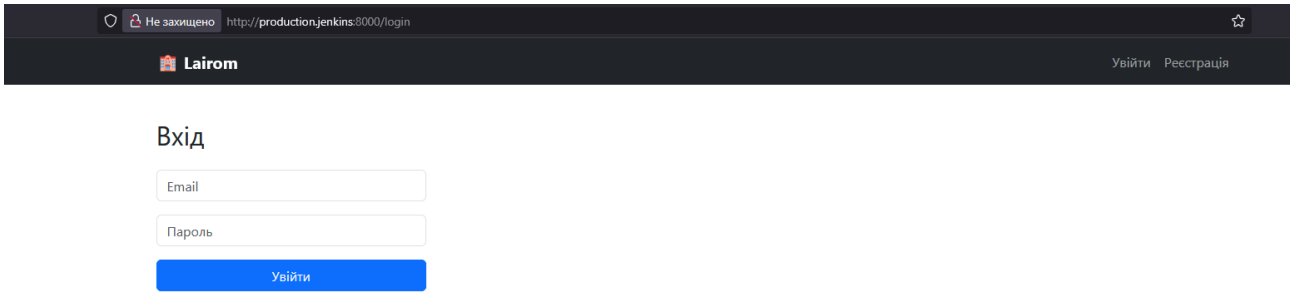

Рис. 3.14. Сторінка реєстрації гостей

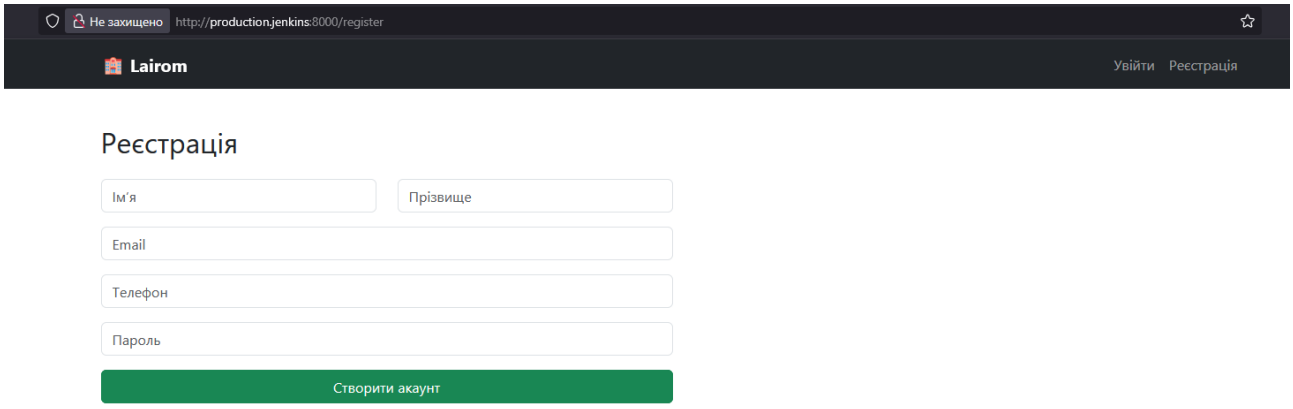

Рис. 3.15. Сторінка реєстрації гостей

Сторінка авторизації адміністраторів знаходиться за шляхом /admin:

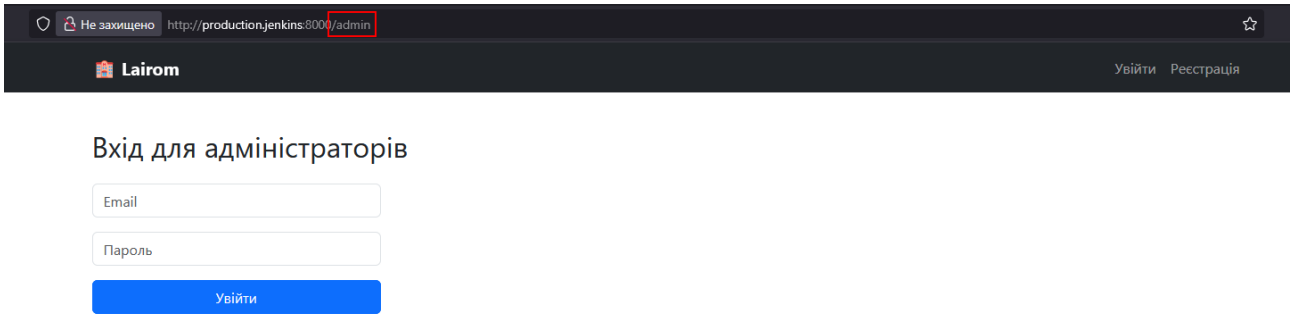

Рис. 3.16. Сторінка авторизації адміністраторів

Адміністратори та гості авторизуються по електронній пошті та паролю.

Після успішної авторизації гостя на головній сторінці з'являються функції перегляду номерів та поточні бронювання авторизованого гостя:

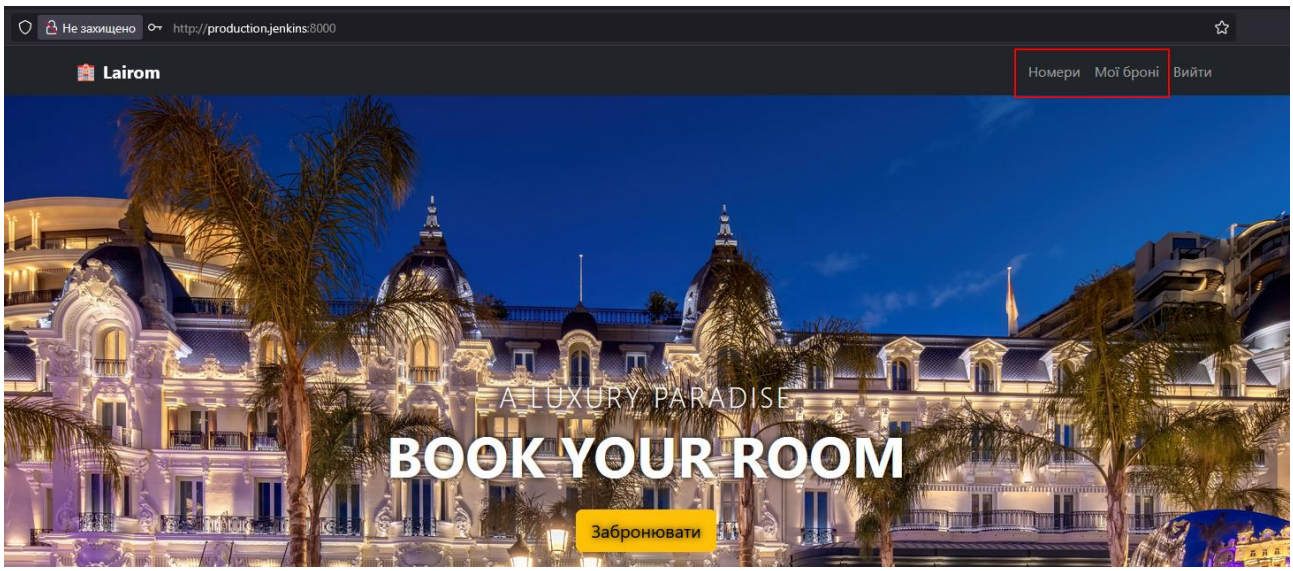

Рис. 3.17. Головна сторінка веб-сайту після авторизації гостя

На сторінка «Номери» можна побачити список всіх доступних бронювань та забронювати доступні номери:

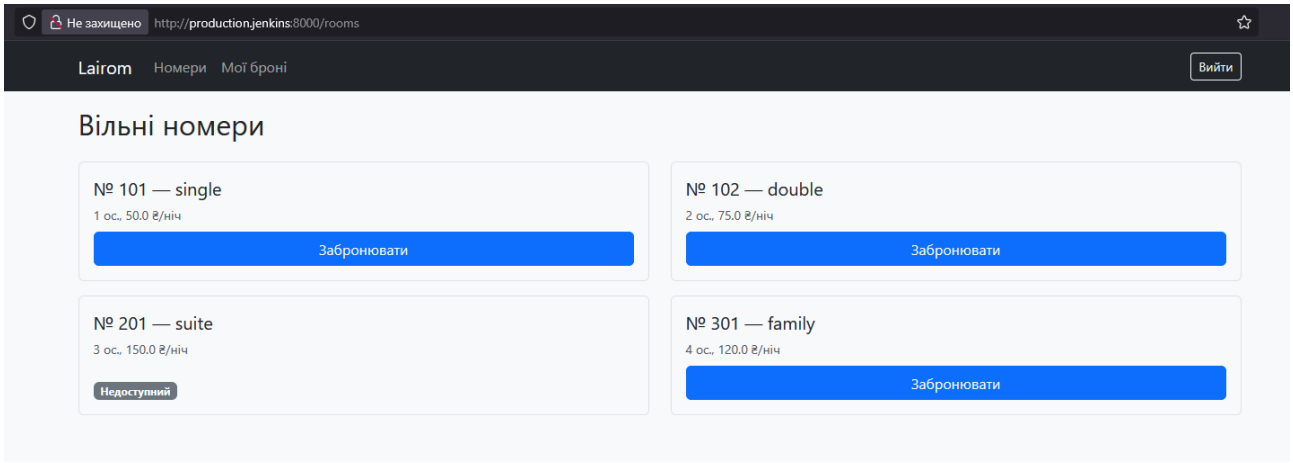

Рис. 3.18. Сторінка номери при успішному авторизації гостя

На сторінці «Мої броні» можна переглянути всі активі бронювання авторизованого гостя:

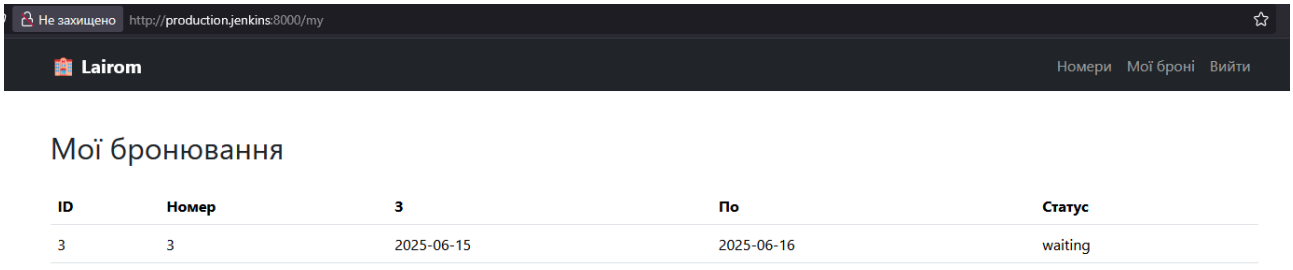

Рис. 3.19. Сторінка Мої броні при успішному авторизації гостя

Якщо авторизуватися в веб-застосунку через адміністратора (по шляху /admin) то головна сторінка буде мати наступний вигляд:

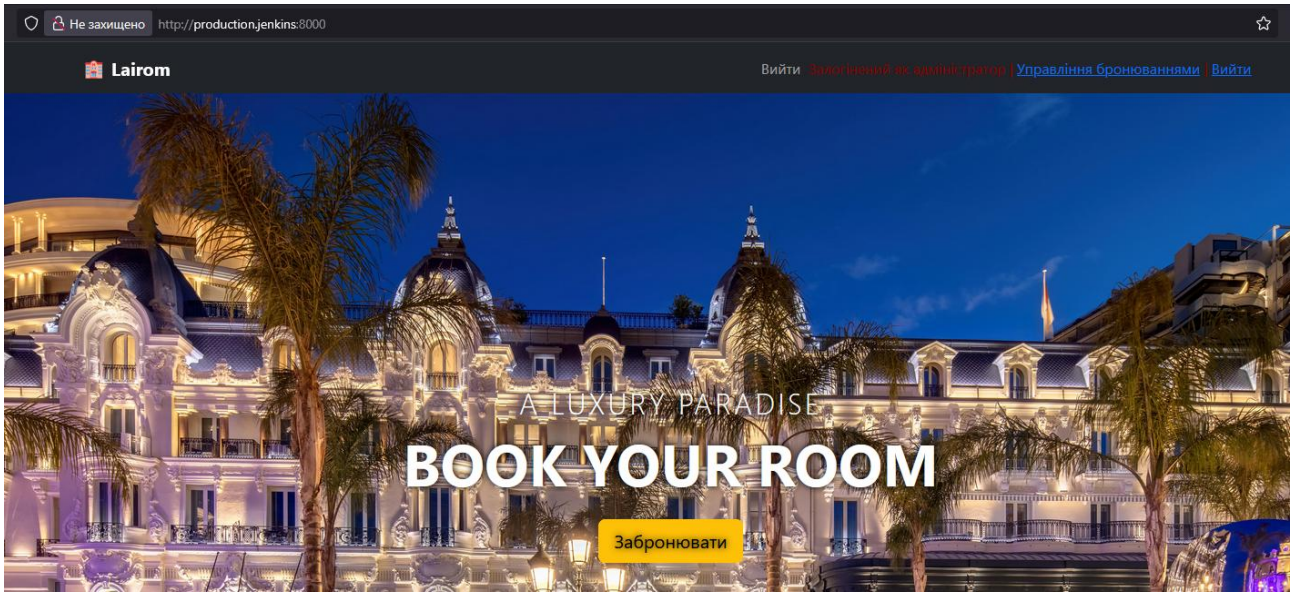

Рис. 3.20. Головна сторінка при успішній авторизації адміністратора готелю

Якщо перейти на вкладку «Управління бронюванням» то можна переглянути всі бронювання готелю, а також видалити вже існуючі бронювання:

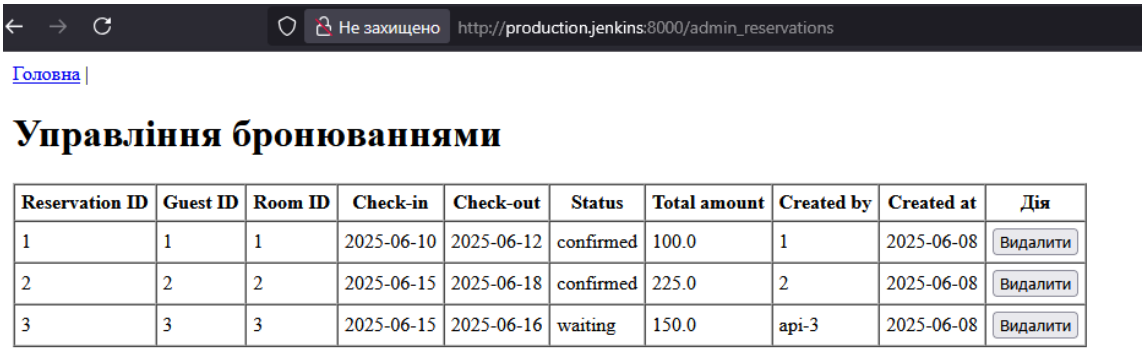

Рис. 3.21. Сторінка управління бронюванням при успішному авторизації адміністратора готелю

3.5. Розгортання та тестування серверного застосунку за допомогою Jenkins

Створюємо завдання, яке буде завантажувати з GitHub серверний застосунок на Python та запускати його тестовому сервері:

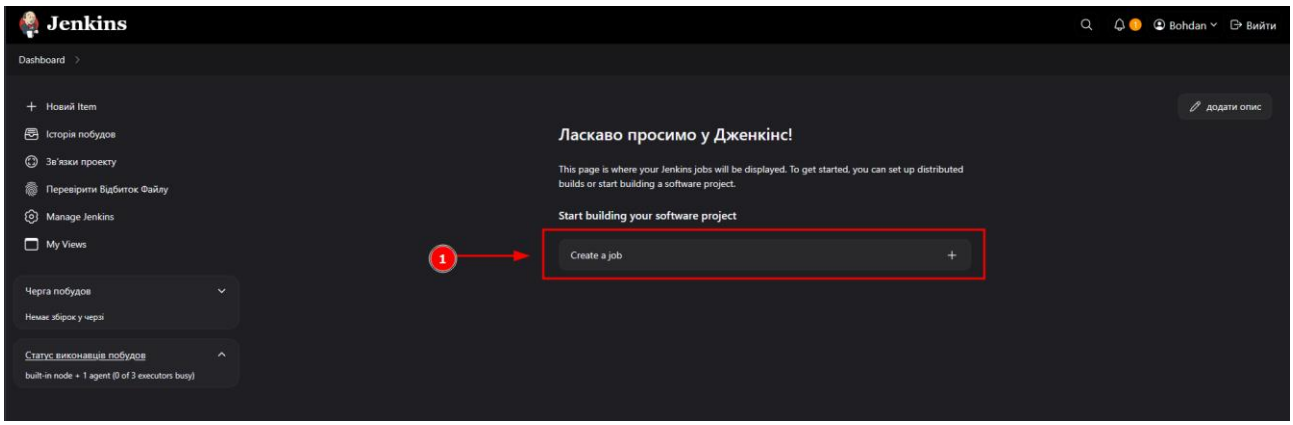

Рис. 3.22 Створення нового завдання

Задаємо йому ім’я та вибираємо тип pipeline:

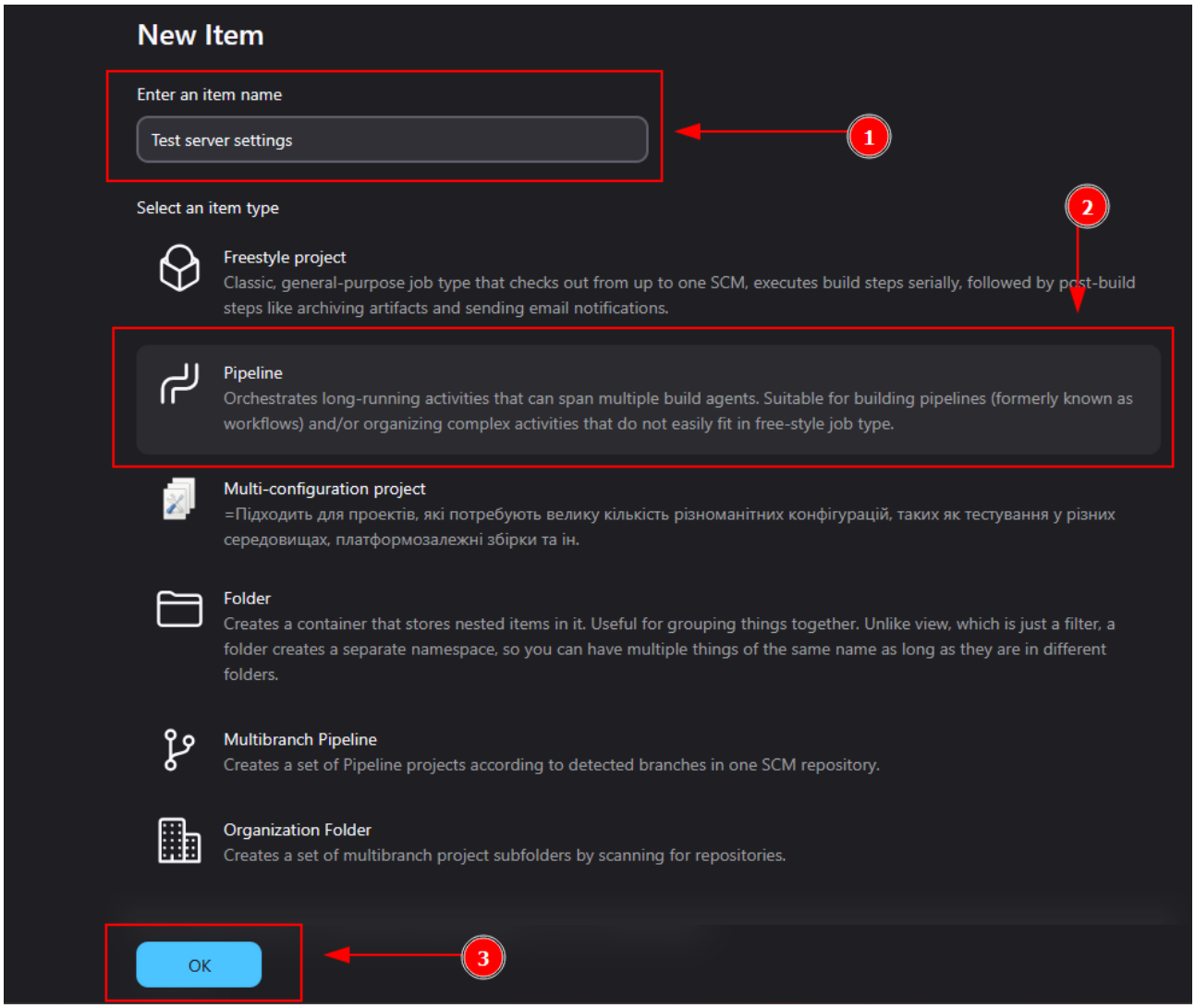

Рис. 3.23. Задання ім’я та вибір відповідного типу під час створення pipeline

В налаштуваннях самого завдання задаємо код pipeline:

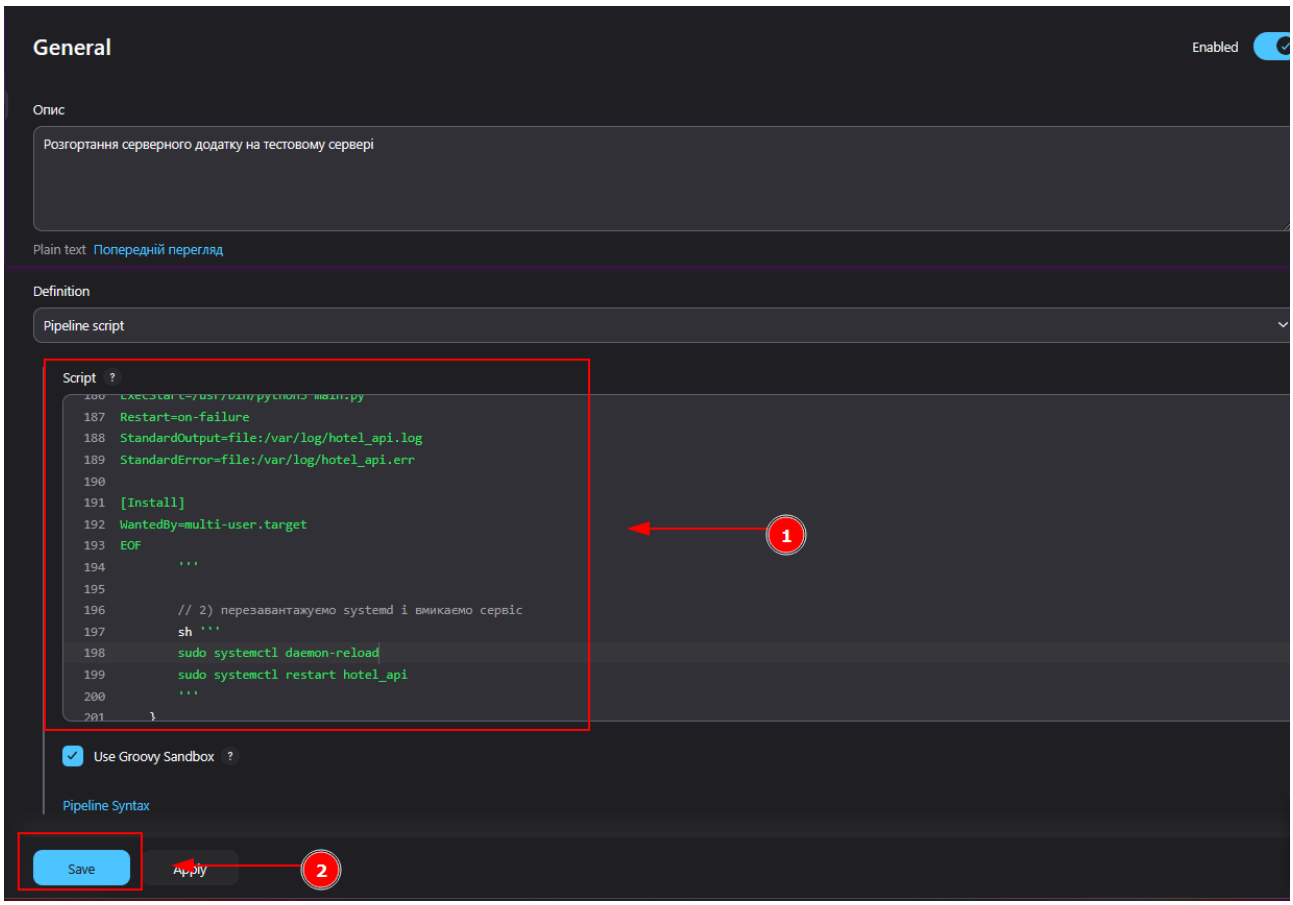

Рис. 3.24. Задання pipeline при створені завдання

Pipeline складається з певних основних частин:

- Оголошення самого pipeline
- Задання агенту (worker) на якому буде виконуватися вказаний pipeline, або можна вказати що він буде виконуватися на будь-якому агенті (any)
- Задання змінних, які можна використовувати в pipeline
- Опис всіх етапі виконання завдання, які будуть виконуватися по черзі

Код Pipeline для автоматичного розгортання серверного застосунку на тестовому сервері виглядає наступним чином:

```
pipeline {
    agent { label 'test' } // Назва агента в Jenkins

// Оголошення змінних
    environment {
        REQUIRED_PY = '3.10.12'          // очікувана точна версія
        GIT_REPO    = 'git@github.com:Maybes/server-api.git'
        GIT_CREDENTIALS = '91eb9b21-7d6a-46b2-946e-f5c0fced2c01'
        DB_HOST          = 'localhost'
        DB_PORT          = '3306'
        DB_NAME          = 'hotel_db'
        DB_USER          = 'root'
        DB_PASS          = 'root'
        Job_name    = "${env.JOB_NAME}"
        MYSQL_USER       = 'hotel'
        MYSQL_PASS       = 'hotel'
    }

// Початок етапі виконання
    stages {
    // Перший етап – зупинка служби hotel_api, якщо активна
stage('Stop hotel_api if active') {
        steps {
        sh '''
        # Перевіряємо статус служби
        if systemctl is-active --quiet hotel_api; then
            echo "hotel_api is active — stopping it..."
            sudo systemctl stop hotel_api
        else
            echo "hotel_api is not active — skipping stop."
        fi
        '''
        }
    }

// 2 етап – перевірка наявності python версії 3.10.12. Встановлення цієї версії, якщо вона відсутня.
   stage('Ensure Python 3.10.12') {
             steps {
                     def pyExists = (sh(
                         script: "command -v python3 >/dev/null 2>&1",
                         returnStatus: true
                     ) == 0)


            if (!pyExists) {
                         echo 'Python3 не знайдено, встановлюємо Python 3.10.12…'
                         sh '''
                           set -e
                           sudo apt-get update
                           sudo apt-get install -y software-properties-common
                           sudo add-apt-repository -y ppa:deadsnakes/ppa
                           sudo apt-get update
                           sudo apt-get install -y python3.10 python3.10-venv python3.10-dev
                           sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.10 1
                           sudo update-alternatives --set      python3 /usr/bin/python3.10
                         '''
                     }
                     def currentVer = sh(
                         script: "python3 --version | awk '{print \$2}'",
                         returnStdout: true
                     ).trim()

                     if (currentVer != env.REQUIRED_PY) {
                         echo «Поточна версія Python: ${currentVer}. Оновлюємо до ${env.REQUIRED_PY}…»
```

```
                        sh '''
                          set -e
                          sudo apt-get update
                          sudo apt-get install -y software-properties-common
                          sudo add-apt-repository -y ppa:deadsnakes/ppa
                          sudo apt-get update
                          sudo apt-get install -y python3.10 python3.10-venv python3.10-dev
                          sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.10 1
                          sudo update-alternatives --set      python3 /usr/bin/python3.10
                        '''
                        currentVer = sh(
                            script: "python3 --version | awk '{print \$2}'",
                            returnStdout: true
                        ).trim()
                    }

                    echo "Використовується Python ${currentVer}"
                }
            }
        }

// 3-ій етап – завантаження коду серверного застосунку з GitHub
stage('Checkout Source') {
            steps {

                git(
                    url:          env.GIT_REPO,
                    credentialsId: env.GIT_CREDENTIALS,
                    branch:       'main'
                    )
            }
        }
// 4-ий етап – перевірка встановлення всіх залежностей. Якщо щось відсутнє – встановлюється всі залежності з файлу requirements.txt
        stage('Install Deps') {
            steps {
                sh '''
                  set -e
                  echo "Перевіряємо залежності (глобальний pip)…"

                  TMP_CUR=/tmp/current_sys.txt
                  TMP_WANT=/tmp/wanted_sys.txt

                  # список уже встановлених пакетів
                  python3 -m pip freeze --all 2>/dev/null | sort > "$TMP_CUR" || true
                  # бажаний список
                  sort requirements.txt > "$TMP_WANT"

                  if diff -q "$TMP_CUR" "$TMP_WANT" >/dev/null; then
                      echo "Залежності збігаються — встановлення не потрібне."
                  else
                      echo "Встановлюємо/оновлюємо пакети з requirements.txt"
                      sudo python3 -m pip install --upgrade pip
                      sudo python3 -m pip install --no-cache-dir -r requirements.txt
                  fi
                '''
            }
        }
    // 5-етап. Перевірка встановлення MySQL сервер. Якщо відсутній – встановлюємо
        stage('Ensure MySQL Server') {
            steps {
                sh '''
                    set -e
                    echo "Перевіряємо, чи встановлено mysql-server…"
                    if ! dpkg -l | grep -q '^ii  mysql-server '; then
                        echo "🛠  mysql-server не знайдено — встановлюємо"
```

```
                            sudo DEBIAN_FRONTEND=noninteractive apt-get update
                            sudo DEBIAN_FRONTEND=noninteractive apt-get install -y mysql-server
                            sudo systemctl enable mysql
                            sudo systemctl start mysql
                        else
                            echo "mysql-server уже встановлений"
                        fi

                        # Відображення версії MySQL
                 mysql --version || true
             '''
             }
         }
    // 6-ий етап – Створення бази даних в MySQL та завантаження в неї тестових даних
             stage('Load hotel_test.sql') {
             steps {
             script {
                 sh '''
                   set -e
                   FILE="hotel_test.sql"

                   if [ ! -f "$FILE" ]; then
                       echo " $FILE не знайдено у $WORKSPACE – пропускаємо імпорт."
                       exit 0
                   fi

                      mysql --host=${DB_HOST} --port=${DB_PORT} \
                           --user=${DB_USER} --password=${DB_PASS} \
                           --execute="CREATE DATABASE IF NOT EXISTS ${DB_NAME} CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;"

                   echo "Імпортуємо $FILE у базу ${DB_NAME}"

                   mysql -u "${DB_USER}" -p"${DB_PASS}" "${DB_NAME}" < "$FILE"

                   echo "Дамп успішно завантажено."
                 '''
             }
         }

       }
    // 7-ий етап – створення сервері під серверний застосунок
    stage("Create service") {
          steps {


      script {
             sh '''
             cat <<'EOF' | sudo tee /etc/systemd/system/hotel_api.service > /dev/null
    [Unit]
    Description=Hotel REST API
    After=network.target

    [Service]
    User=root
    WorkingDirectory=/home/jenkins/workspace/Test_server_settings
    ExecStart=/usr/bin/python3 main.py
    Restart=on-failure
    StandardOutput=file:/var/log/hotel_api.log
    StandardError=file:/var/log/hotel_api.err

    [Install]
    WantedBy=multi-user.target
    EOF
             '''

             // 2) перезавантажуємо службу і вмикаємо сервіс
             sh '''
```

```
            sudo systemctl daemon-reload
            sudo systemctl restart hotel_api
            '''
        }
        }

    }
    // 8-ий етап – перевірка активності сервісу hotel_api
     stage("Check service") {
        steps {
            script {
            sh '''
            # Перевіряємо, чи служба активна
            if systemctl is-active --quiet hotel_api; then
                echo "hotel_api is running"
            else
                echo "ERROR: hotel_api service is NOT running!" >&2
                exit 1
            fi
            '''
            }
       }
     }
    }
        post {
            success { echo '✅ Pipeline успішно виконався.' }
            failure { echo '❌ Pipeline завершився з помилкою.' }
        }
    }
```

Аналогічним чином були розроблена два наступних Pipeline (Test_API та Production_settings) для тестування та розгортання серверного застосунку на виробничому сервері:

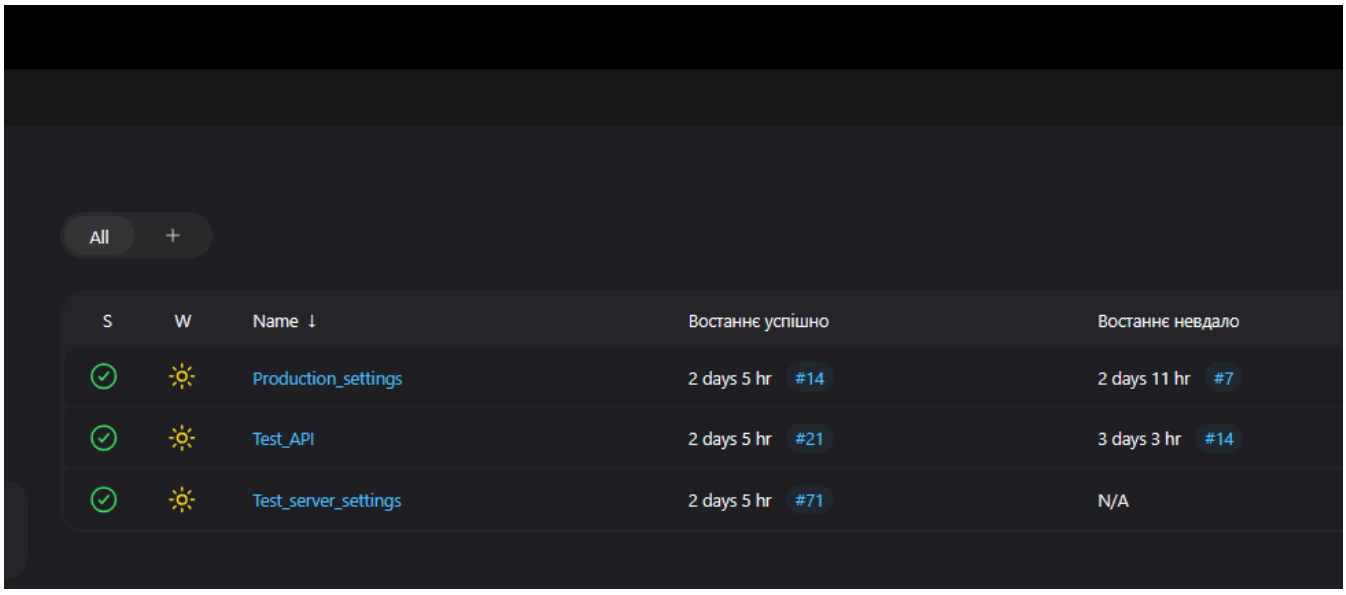

Рис. 3.22. Відображення всіх завдань в веб-інтерфейсі Jenkins

Запускаємо завдання:

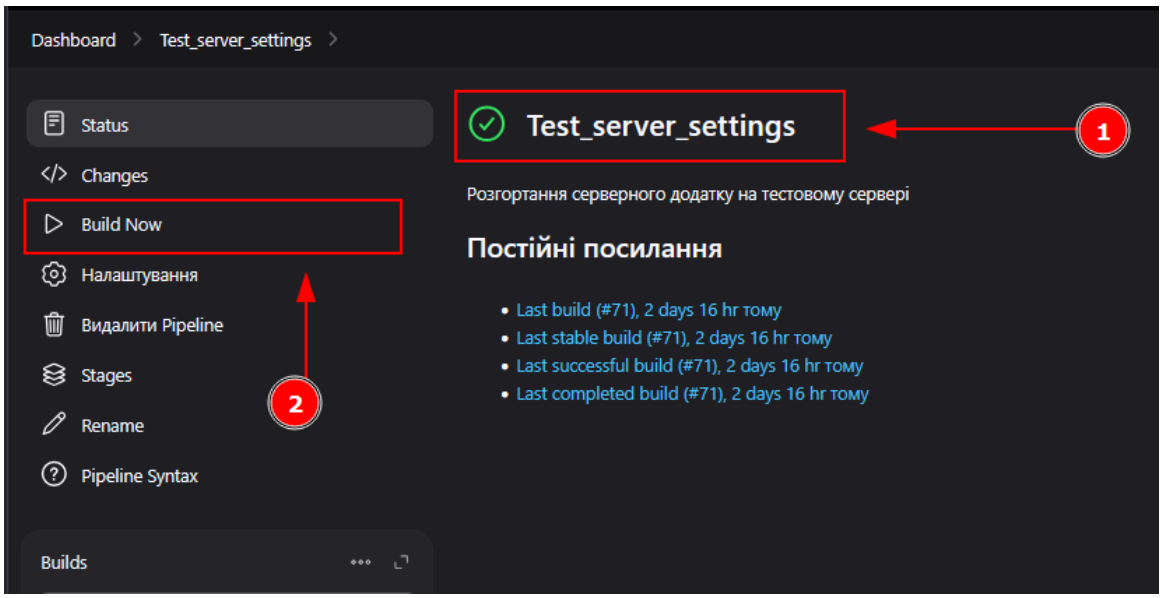

Рис. 3.23. Запуск завдання для розгортання серверного застосунку на тестовому сервері

Нижче ми можемо побачити історію запущений білдів цього завдання:

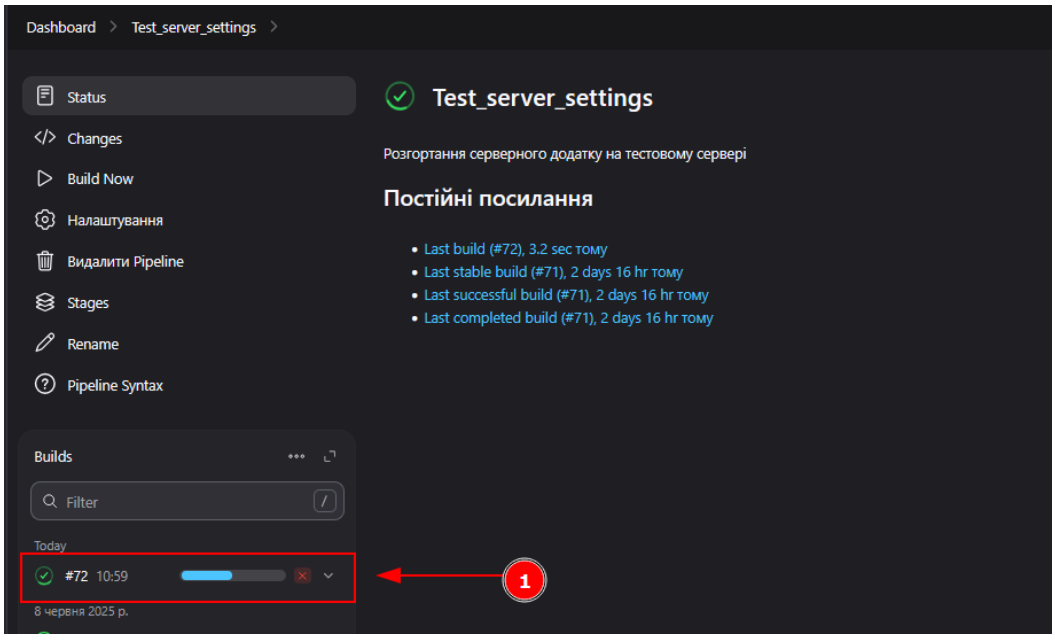

Рис. 3.24. Відображення процесу запущеного завдання в веб інтерфейсі Jenkins

Якщо перейти в самий білд та у вивід консолі, можна побачити що завдання було успішно виконано, і серверний застосунок був розгорнутий на тестовому сервері:

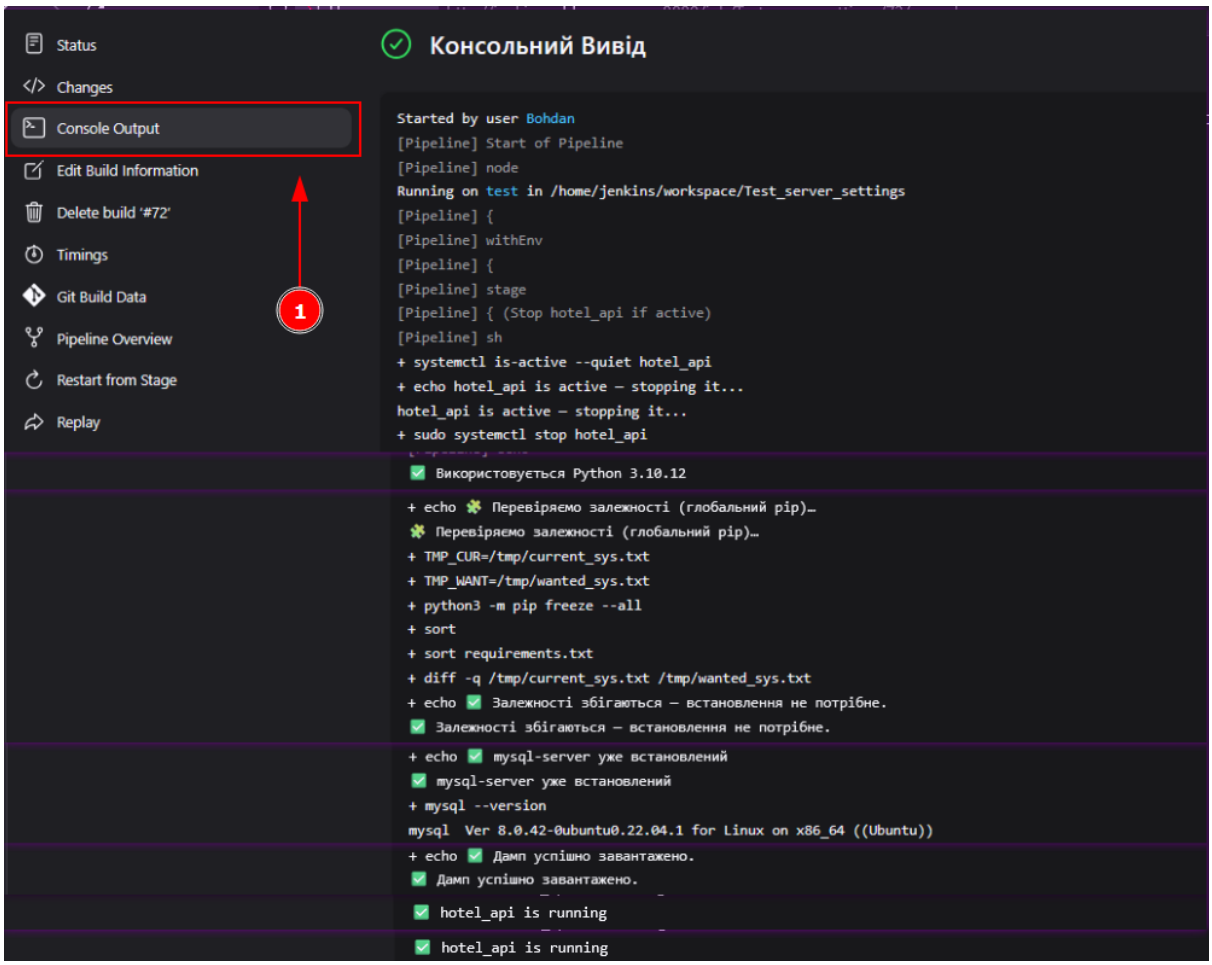

Рис. 3.25. Вивід консолі першого завдання

Оскільки перше завдання було виконано успішно, після нього автоматично запустилось друге завдання, в якому тести по Rest API були пройдені успішно:

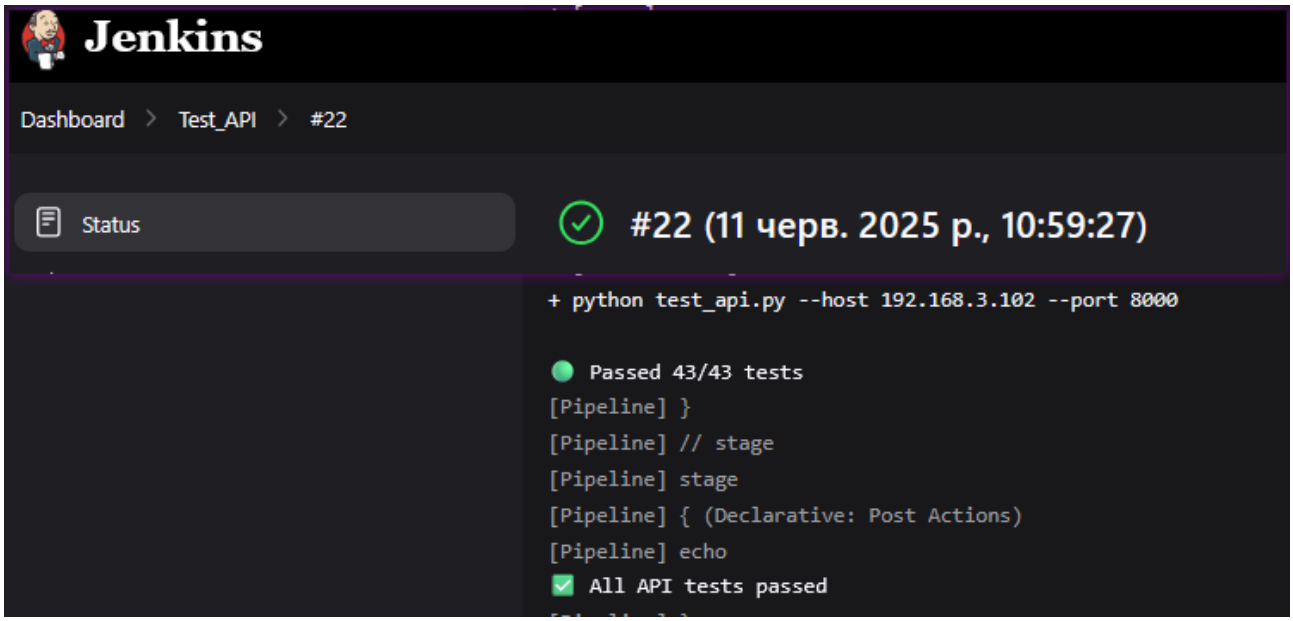

Рис. 3.26. Успішне проходження завдання з тестами для API серверу

Після чого був автоматично запущене завдання про розгортанню серверного застосунку на виробничому серверу, який також виконався успішно:

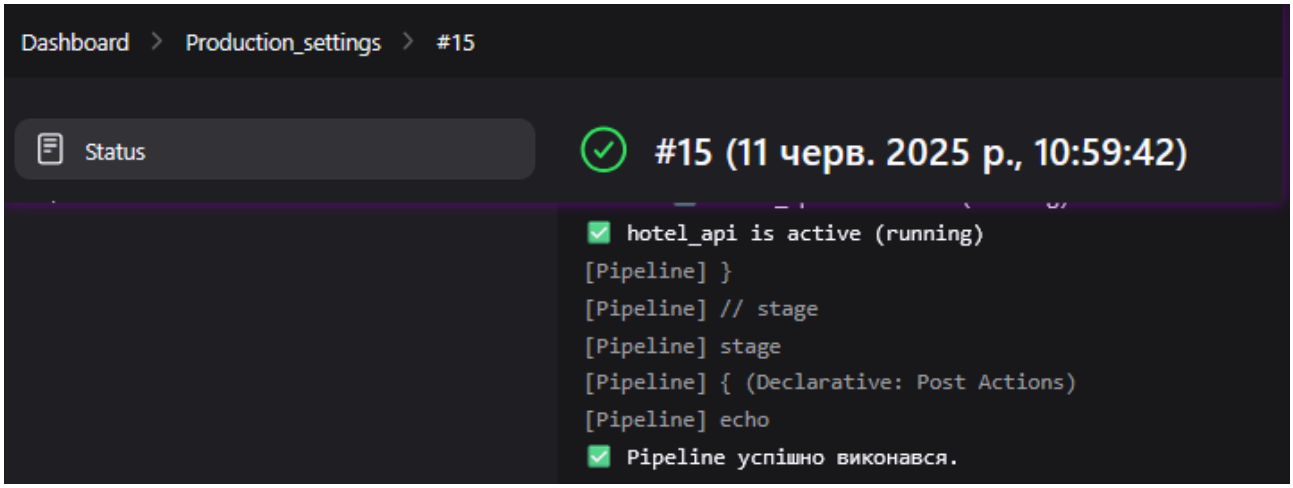

Рис. 3.27. Успішне завершення завдання з розгортання серверного застосунку на виробничому сервері

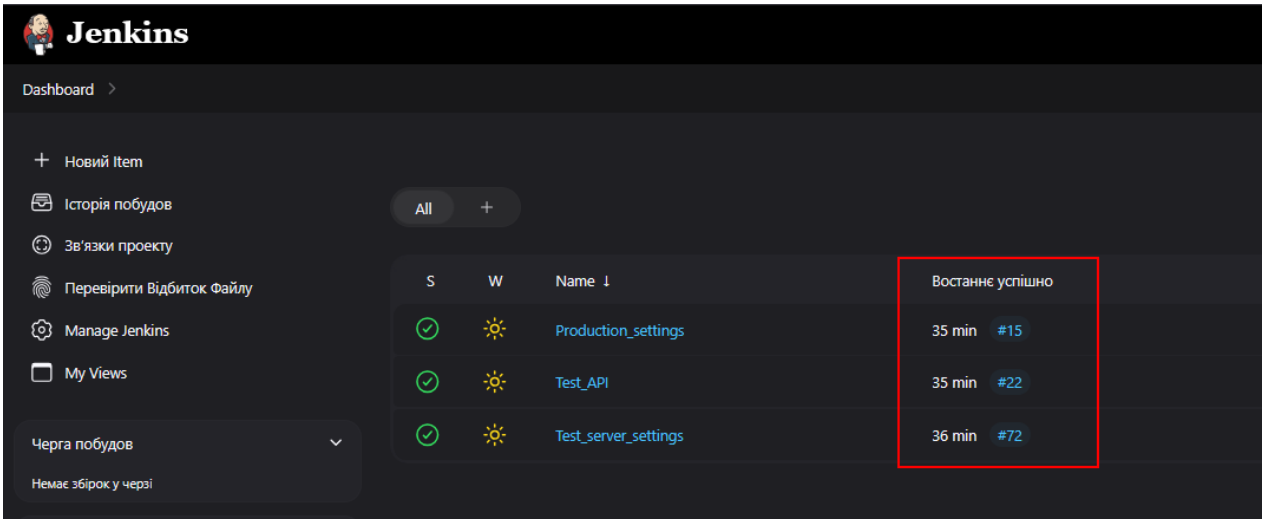

Рис. 3.28. Відображення всіх успішно завершених завдань в Jenkins

Наразі, щоб змоделювати помилку виконання API тестів, я добавлю помилку в код серверного застосунку, та зміню картинку на головній сторінці веб інтерфейсу.

Після запуску CI/CD процесу можна побачити, що розгортання серверного застосунку на тестовому сервері було успішним, однак завдання на тестування провалилось, через що серверний застосунок не був змінений на виробничому сервері:

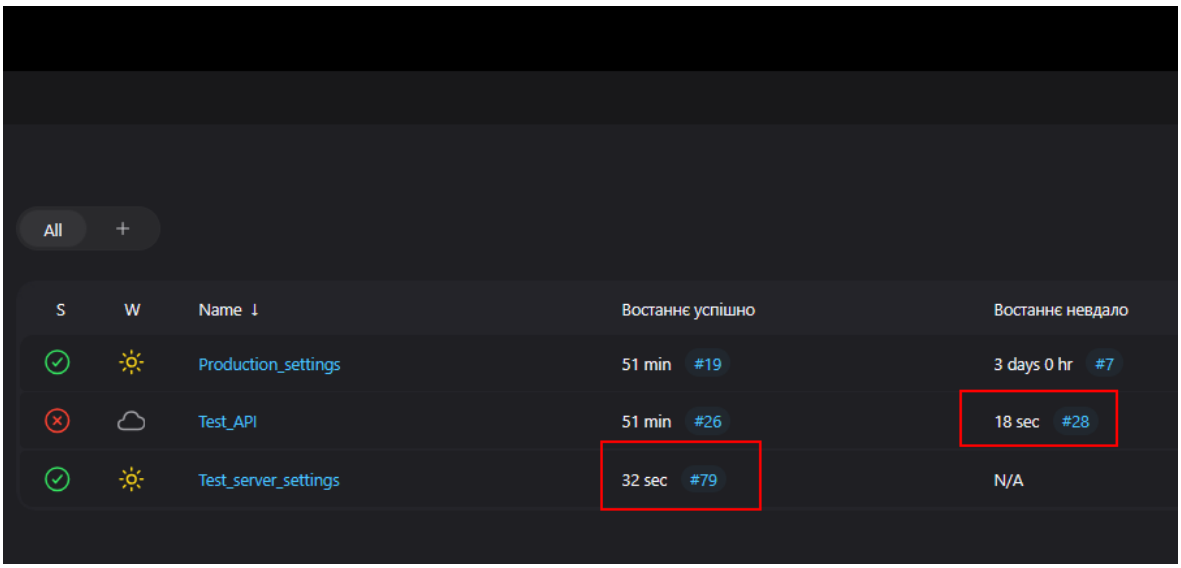

Рис. 3.29. Відображення успішних та неуспішних завершених завдань в Jenkins

Якщо переглянути логування останнього білду, то можна побачити що тест на отримання кімнат провалився:

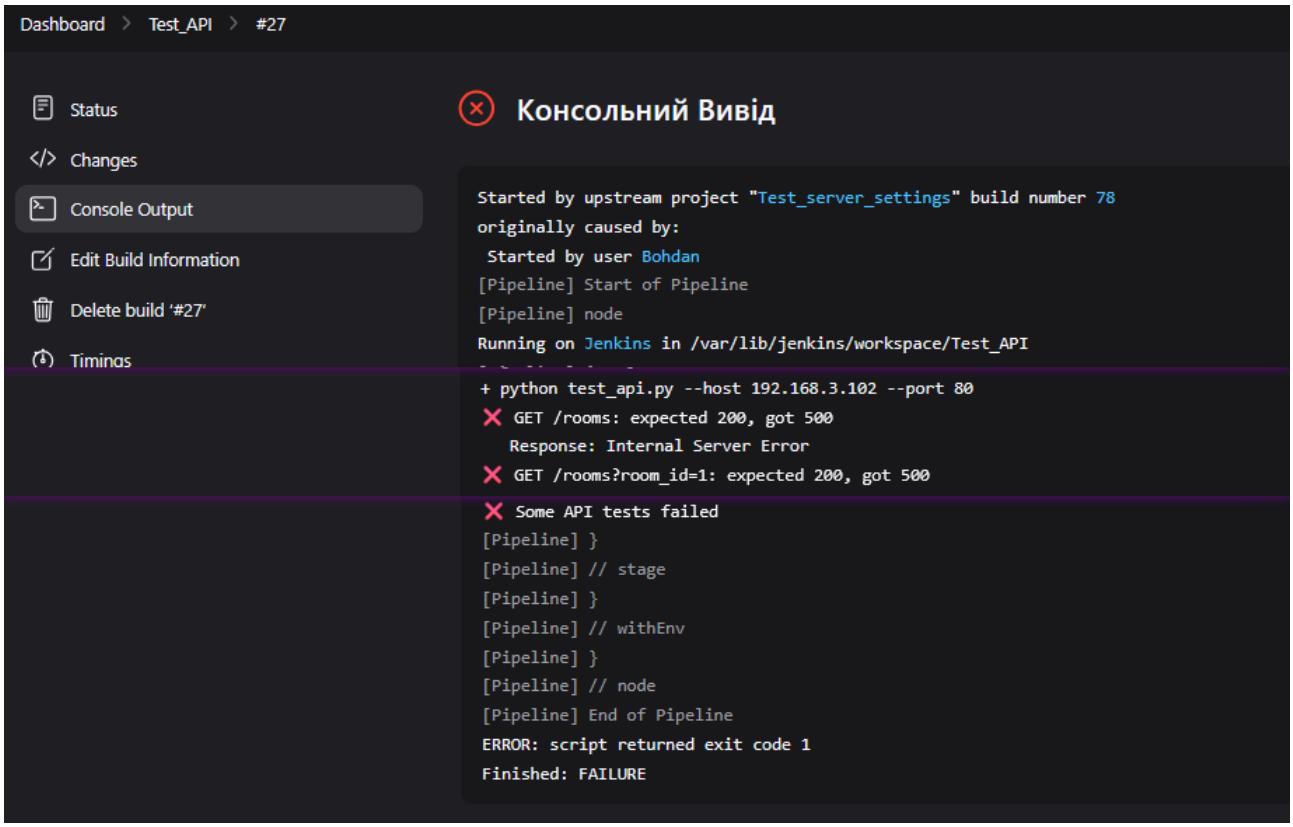

Рис. 3.30. Логування останнього завдання на тестування

Веб інтерфейс виробничого серверу працює без проблем, оскільки помилковий код серверного застосунку не був змінений на ньому через неуспішне проходження тестів, веб інтерфейс тестового серверу не працює:

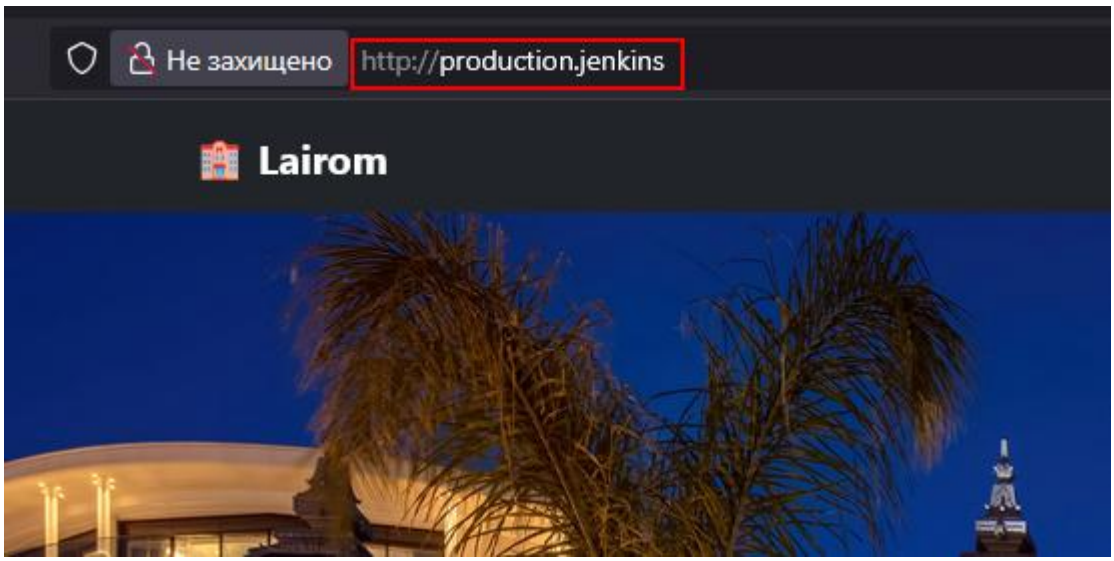

Рис. 3.31. Веб інтерфейс виробничого сервер після неуспішого проходження тестів

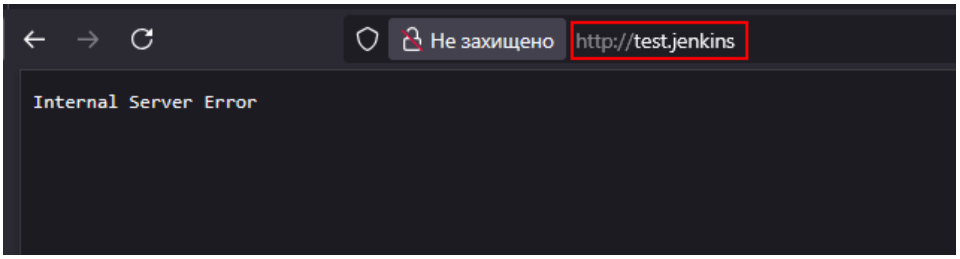

Рис. 3.32. Веб інтерфейс тестового серверу після неуспішного проходження тестів

На цьому етапі я виправлю помилку в коді, та заново запущу CI/CD процес в Jenkins. Після виправлення помилки в коді, всі завдання були виконані успішно, і веб інтерфейс на тестовому сервері виглядає наступним чином:

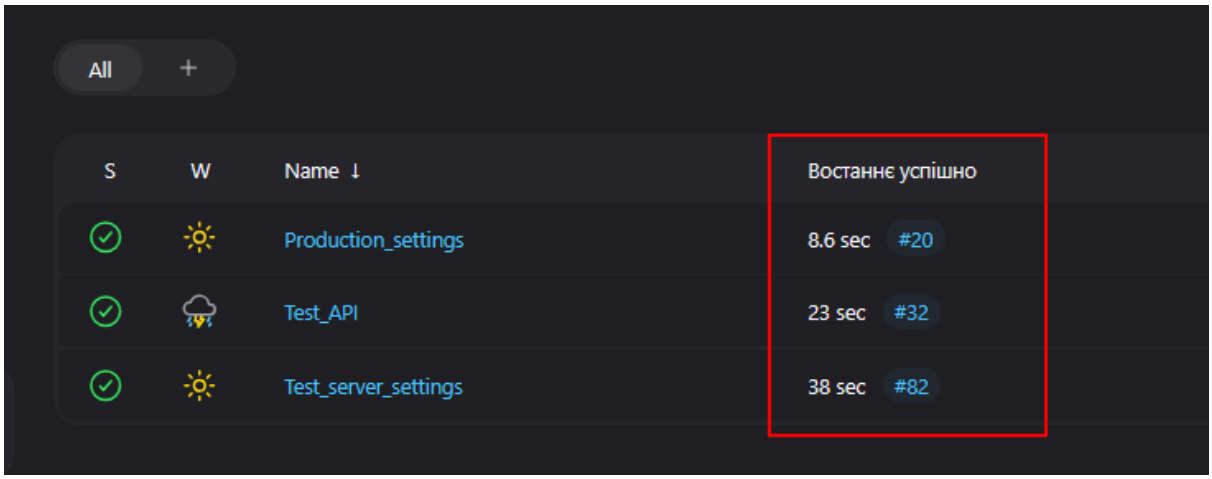

Рис. 3.33. Успішне виконання всіх завдань після виправлення помилки в коді

Рис. 3.34. Веб інтерфейс тестового серверу після виправлення помилки в коді серверного застосунку

Висновки до розділу 3

У цьому розділі було розроблено серверний застосунок який складається з 3-ьох компонентів: бази даних MySQL, серверу Rest API а також веб-сайт, який взаємодіє з базою даних готелю через API.

Було описано структури всіх таблиць в базі даних, наведені приклади API запитів для отримання, додавання, зміни та видалення даних в базі даних з використанням формату JSON, а також описано основні сторінки веб-сайту.

Також в цьому розділі було описано основні елементи pipeline коду та повний цикл використання CI/CD в Jenkins у тестовому та виробничих серверах. Було змодельовано помилку у роботі серверного застосунку API щоб забезпечити недоторканість виробничого серверу у випадку наявності будь-яких проблем у коді.

ВИСНОВКИ

1. **Проаналізовано теоретичні засади CI/CD.** Розкрито поняття безперервної інтеграції, доставки та розгортання, визначено їхні переваги й ключові принципи (спільна відповідальність, мінімізація ризиків, короткий цикл зворотного зв'язку, єдине робоче середовище). Огляд сучасних інструментів засвідчив доцільність вибору Jenkins як гнучкого й розширюваного рішення для автоматизації DevOps-процесів.
2. **Налаштовано середовище Jenkins і агенти-worker.** Встановлено Jenkins на Ubuntu Server 22.04, згенеровано SSH-ключі, під'єднано два worker-сервери (test і production) і інтегровано GitHub як систему контролю версій.
3. **Розроблено серверний застосунок**
    - Спроєктовано реляційну базу даних MySQL із п'ятьма основними таблицями (guests, administrators_hotel, admins_api, rooms, reservations).
    - Створено REST-API на FastAPI + Uvicorn, який підтримує розширений пошук та систему авторизації (Basic Auth).
    - Розроблено демонстраційний веб-інтерфейс на Jinja2 / Bootstrap, що взаємодіє з API та реалізує сценарії для гостей і адміністраторів.
4. **Спроєктовано та реалізовано три Jenkins-pipeline:**
    - Test_server_settings: автоматично розгортає застосунок на тестовому сервері, ініціалізує базу даних, запускає серверний застосунок як службу.
    - Test_API: запускає понад 40 REST-тестів Post/GET/PUT/DELETE, формує зведення успішно/помилка, зупиняючи конвеєр у разі помилки.
    - Production_settings: за умови успішних тестів виконує оновлення коду на production-сервері без переривання сервісу.

5. **Забезпечено автоматичне керування залежностями та середовищем.** Сценарії перевіряють наявність Python 3.10 , MySQL-server, ініціюють базу даних і виконують імпорт тестових даних.
6. **Проведено експериментальне тестування.** CI/CD-ланцюжок продемонстрував коректне виявлення помилки в API-коді: тестовий деплой відбувся, тести «посипалися», production-середовище залишилося недоторканим. Після виправлення коду всі етапи пройшли успішно, і оновлення безпроблемно дійшло до користувачів.

Основні результати та їхня практична цінність:

- Скорочення часу релізу — оновлення переходить із репозиторію до production-середовища в один клік без ручних дій.
- Зниження ризику людських помилок — автоматичні тести та перевірки середовища гарантують цілісність коду і даних.
- Масштабованість — додавання нових середовищ або розширення тест-набору потребує мінімальних змін у Jenkinsfile.

Отже, поставлена мета — створити працездатну CI/CD-інфраструктуру для Python-застосунку з використанням Jenkins досягнута. Запропонований підхід підвищує якість, безпечність та швидкість доставки нового функціоналу, що робить його доцільним для впровадження у реальних комерційних проєктах.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Методологія CI/CD: автоматизація, тестування і швидкі релізи. IT Education Center Blog. URL: https://itedu.center/ua/blog/review/what-is-ci-cd/?srsltid=AfmBOorumBi_nJQ0oTUXohaLkereVzvjboixIRPIl829Jzu3Py1W3xIO (дата звернення: 10.03.2025)
2. Що таке CI/CD, як він працює та коли знадобиться на проєкті. *NIX*. URL: https://www.nixsolutions.com/ua/blog/for-developer/shho-take-ci-cd-yak-vin-praczyuye-ta-koly-znadobyt/ (дата звернення: 10.03.2025)
3. 20 best CI/CD tools in 2022: continuous integration & continuous delivery - fulcrum. *Fulcrum*. URL: https://fulcrum.rocks/blog/best-ci-cd-tools/ (дата звернення: 17.03.2025)
4. FastAPI. *FastAPI*. URL: https://fastapi.tiangolo.com/ (дата звернення: 15.04.2025)
5. Installing jenkins. *Installing Jenkins*. URL: https://www.jenkins.io/doc/book/installing/ (дата звернення: 20.04.2025)
6. Jinja – jinja documentation (3.1.x). *Jinja – Jinja Documentation (3.1.x)*. URL: https://jinja.palletsprojects.com/en/stable/ (дата звернення: 13.06.2025)
7. Pipeline. *Pipeline*. URL: https://www.jenkins.io/doc/book/pipeline/ (дата звернення: 20.04.2025)
8. Using a jenkinsfile. *Using a Jenkinsfile*. URL: https://www.jenkins.io/doc/book/pipeline/jenkinsfile/ (дата звернення: 20.04.2025)
9. Uvicorn. *Uvicorn*. URL: https://www.uvicorn.org/ (дата звернення: 20.04.2025)
10. Maybes/Server-FastAPI. GitHub. URL: https://github.com/Maybes/Server-FastAPI (дата звернення: 12.06.2025).