

Приватний заклад вищої освіти
Одеський технологічний університет «ШАГ»
Кафедра інформаційних технологій та фундаментальної підготовки

Випускна кваліфікаційна робота бакалавра

**Інформаційна система гнучкого управління програмними проектами
(на прикладі Trello)**

на здобуття бакалаврського рівня вищої освіти
зі спеціальності «122 Комп'ютерні науки»

Виконавці проекту

	П.І.Б.	Група
1	Миронов О. М.	КН-П-211
2	Костанда Д. О.	КН-П-211
3	Жданов А. Д.	КН-П-211
4	Бордюжевич О. І.	КН-П-211
5	Голубов В. А.	КН-Д-211

Автор звіту

_____ Бордюжевич Олексій Ігоревич

Науковий керівник	доц. Самойленко Д. М.
-------------------	-----------------------

АНОТАЦІЯ

УДК 004.415.2+004.415.3+004.43:005.8

Б-82

Бордюжевич О. І. Інформаційна система гнучкого управління програмними проектами (на прикладі Trello): випускна кваліфікаційна робота на здобуття бакалаврського рівня вищої освіти зі спеціальності «122 Комп'ютерні науки». – Одеса: ОТУШ, 2025. – 25 с.

Робота присвячена проектуванню, впровадженню та випробуванню серверного середовища інформаційної системи ItRun, яка орієнтована на організацію управління програмними проектами у форматі канбан-дошки. У межах DevOps-частини реалізовано повний цикл автоматизації розгортання вебплатформи з використанням сучасних інструментів — Docker, GitHub Actions, Nginx, VPS-хостингу на базі CentOS.

Актуальність теми визначається потребою у гнучких, стабільних та безпечних інфраструктурних рішеннях, які забезпечують безперервну інтеграцію, оновлення та масштабованість у розподіленому середовищі розробки. Особливу увагу приділено методології DevOps, її впровадженню у контексті малого та середнього командного програмування, автоматизації життєвого циклу застосунку.

У роботі досліджено й реалізовано:

- архітектуру серверного середовища із використанням контейнеризації та зворотного проксі-сервера;
- налаштування CI/CD-процесів на основі GitHub Actions для автоматизованого тестування та деплою;
- конфігурацію docker-compose для ізольованого керування frontend, backend та базою даних;
- інтеграцію безпечних підходів до адміністрування, включаючи .env-налаштування, перевірки доступності сервісів та харденінг інфраструктури.

Об'єктом дослідження виступає інфраструктурний стек інформаційної системи, предметом — практики автоматизації розгортання та адміністрування. Результати випробувань свідчать про стабільність, надійність і масштабованість реалізованої системи: час відповіді API становив менше 250 мс, CI/CD-процес повного циклу тривав до 1 хвилини, а доступність системи протягом тестового періоду перевищувала 99,9%.

Методологічною основою дослідження є принципи системного аналізу, DevOps-інженерії та практики контейнеризації. Робота містить 11 використаних джерел.

ЗМІСТ

ВСТУП _____	4
ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЄКТУ _____	5
РОЗДІЛ 1. Загальна характеристика системи _____	8
РОЗДІЛ 2. Реалізація розгортання та адміністрування серверного середовища _____	10
2.1 Інструменти автоматизації, деплою та конфігурації _____	10
2.2 Архітектура серверного середовища та процеси його обслуговування _____	15
РОЗДІЛ 3. Результати випробувань та перевірка працездатності системи _____	21
ВИСНОВКИ _____	24
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ _____	25

ВСТУП

Стрімкий розвиток інформаційних технологій суттєво трансформував способи організації праці, взаємодії команд та управління цифровими проєктами. В умовах глобалізації, розподілених команд та зростання обсягу задач виникає потреба в інструментах, що дозволяють гнучко, наочно й ефективно керувати робочими процесами, розподіляти відповідальність та контролювати строки виконання. Саме тому системи управління завданнями та проєктами (так звані таск-трекери) стали невід’ємним компонентом сучасного цифрового середовища.

Незважаючи на наявність великої кількості аналогічних систем — таких як Trello, Jira, Asana — більшість із них мають обмежену гнучкість налаштувань, складний інтерфейс або потребують підключення сторонніх інтеграцій навіть для базових функцій. Крім того, значна частина таких сервісів є платними або обмеженими у функціональності для невеликих команд. Ці недоліки створюють передумови для розробки власного рішення, яке буде адаптовано до специфічних вимог користувачів, матиме відкриту архітектуру, забезпечуватиме контроль над даними та можливість подальшого розширення.

У зв’язку з цим команда розробників реалізувала інформаційну систему управління програмними проєктами та завданнями — ItRun. Основною метою проєкту є створення зручного, адаптивного та масштабованого вебзастосунку для управління завданнями у форматі канбан-дошки, з підтримкою користувачів, робочих просторів, колонок, карток і спільної роботи над проєктами. Платформа реалізована за сучасними технологічними підходами з використанням модульної архітектури, container-based deployment та засобів безперервної інтеграції.

У рамках цієї дипломної роботи основну увагу приділено налаштуванню, автоматизації та адмініструванню серверного середовища для інформаційної системи ItRun. Здійснено аналіз вимог до продуктивності, безпеки та масштабованості, на основі чого обрано хмарну інфраструктуру у вигляді VPS-середовища з операційною системою CentOS. Проведено контейнеризацію застосунку з використанням Docker, що дозволило досягти ізоляції компонентів, їх незалежного масштабування та полегшення процесу розгортання. За допомогою docker-compose описано конфігурації для серверної, клієнтської частини та бази даних. Особливу увагу приділено впровадженню CI/CD-процесів за допомогою GitHub Actions, що забезпечило безперервну інтеграцію, тестування та автоматичне оновлення застосунку з моменту внесення змін до основної гілки репозиторію. Реалізовані підходи відповідають сучасним DevOps-практикам і дозволяють підтримувати надійне, стабільне та гнучке середовище експлуатації системи.

ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЄКТУ

Загальний опис

Програмне забезпечення «ItRun» призначене для організації ефективного управління завданнями та цифровими проєктами в команді. Система створюється у форматі вебзастосунку на базі клієнт-серверної архітектури з можливістю масштабування, контейнеризації та автоматизованого розгортання. Основною ідеєю є створення інструмента для візуального управління проєктами за допомогою канбан-дошок, що дає змогу формувати робочі простори, створювати задачі, призначати відповідальних, відслідковувати статуси виконання, коментувати й координувати дії команди.

Застосунок забезпечує повноцінну взаємодію між учасниками одного або декількох робочих просторів, підтримує контроль версій, зберігає історію дій користувачів, а також включає механізми захисту персональних даних та автентифікації через підтвердження email. Платформа передбачає розподіл ролей та прав доступу, що забезпечує гнучке адміністрування проєктів.

Призначення

Система орієнтована на використання невеликими та середніми командами розробників, стартапами, фриланс-агенціями, а також окремими користувачами, які потребують інструменту для організації власної діяльності. Особлива увага приділена простоті використання, зручному UI/UX, можливості гнучкої інтеграції, та легкому оновленню функціоналу в майбутньому без втрати даних. Застосунок має підтримувати повний цикл взаємодії: створення задач, їх планування, виконання, коментування та завершення.

Функціональні вимоги

1. Реєстрація та автентифікація

Кожен користувач повинен мати можливість створити обліковий запис у системі. Після реєстрації на електронну адресу надсилається підтвердження, яке користувач має ввести для завершення реєстрації. Після цього користувач може увійти до системи, вказавши свої облікові дані. Усі дії, які потребують доступу до персональної інформації або даних проєктів, мають бути доступні лише авторизованим користувачам. Також має бути можливість вийти із системи, в тому числі з усіх пристроїв одночасно. Деякі функції доступні тільки певним категоріям користувачів (наприклад, адміністраторам).

2. Робочі простори та дошки

Користувачі можуть створювати окремі робочі простори — умовні «команди» або «проєкти», всередині яких розміщуються канбан-дошки. У таких дошках відображається весь хід роботи над завданнями. Користувачі мають можливість змінювати зовнішній вигляд дошки (наприклад, завантажувати зображення чи логотипи), а також запрошувати інших людей до участі у роботі над проєктом. При

цьому можна визначати роль кожного учасника — наприклад, дозволити редагування лише окремим особам, а іншим лише переглядати вміст.

3. Колонки та задачі

Всередині кожної дошки можна створювати колонки (етапи роботи), в які додаються окремі завдання. Ці завдання можна редагувати, переміщати між колонками, видаляти або змінювати їх порядок. Кожне завдання може мати дату завершення, пріоритетність, позначки або додані файли. Також має бути візуальне відображення того, на якому етапі виконання перебуває завдання, щоб усі учасники команди могли швидко оцінити прогрес роботи.

Технічні вимоги та рекомендації до реалізації

Клієнтська-частина реалізується у вигляді односторінкового застосунку (SPA) із застосуванням сучасного стеку технологій:

- Реалізація інтерфейсних компонентів з використанням бібліотеки React у поєднанні з мовою TypeScript
- Клієнтська навігація — забезпечується за допомогою бібліотеки React Router
- Інтерактивна функціональність drag-and-drop — реалізована з використанням відповідних інструментів, таких як react-beautiful-dnd
- Інтеграція з серверною частиною через REST API — включає обробку помилок, відображення індикаторів завантаження (loader'ів), валідацію форм введення
- UI/UX-дизайн — відповідає принципам адаптивності, інтуїтивності та доступності для користувача

Серверна-частина спроектована на основі фреймворку NestJS та побудована відповідно до концепції доменно-орієнтованого дизайну (DDD):

- Модульна структура застосунку — з поділом на auth, user, workspace, board, task, comment, history
- Робота з базою даних MySQL — реалізована через ORM-бібліотеку TypeORM
- Побудова REST API — із валідацією запитів за допомогою Data Transfer Object (DTO) та централізованою обробкою винятків
- Контроль доступу до ресурсів — реалізовано згідно з рольовою моделлю користувача (Admin / Member)
- Дотримання архітектурних патернів — що забезпечує масштабованість і підтримуваність системи

DevOps та інфраструктурний рівень охоплює засоби автоматизації розгортання та підтримки життєвого циклу застосунку:

- Контейнеризація проєкту — з використанням Docker для створення окремих контейнерів: frontend, backend та бази даних MySQL
- Опис взаємодії сервісів — через docker-compose конфігурацію
- Розгортання проєкту на VPS — із встановленим CentOS та налаштуванням

зворотного проксингу через веб-сервер Nginx

- Збереження конфігурацій — у вигляді .env-файлів із секретними змінними оточення
- Реалізація CI/CD — із застосуванням GitHub Actions для автоматичного тестування, білду та розгортання системи

Менеджмент та супровід розробки включає організацію роботи команди, планування та документування проєкту:

- Постановка задач, формування user stories та створення дорожньої карти (roadmap) — реалізовано в середовищі Notion
- Використання Kanban-дошки — для відстеження статусів задач у Trello, що забезпечує прозору візуалізацію прогресу
- Проведення регулярних синків команди — з фіксацією результатів у звітах
- Централізована технічна документація — включає опис функціональності, архітектури, бізнес-логіки, API та сценаріїв використання
- Забезпечення відповідності реалізації вимогам технічного завдання — з обов'язковим контролем якості на всіх етапах розробки.

UI/UX-дизайн інтерфейсного рішення платформи ItRun базувався на сучасних підходах до цифрового продуктового дизайну з урахуванням потреб цільової аудиторії та принципів зручності використання:

- UX-дослідження — аналіз Trello, емпатійна карта, Customer Journey Map
- Структура взаємодії — побудова Site Map та User Flow
- Дизайн-система — типографіка (Poppins), кольори (Tiffany, помаранчевий), сітка 16 пікселів
- UI-компоненти — реалізація у станах (default, hover, active, disabled), auto-layout
- Прототипування — високофідельні екрани у Figma: реєстрація, онбординг, workspace, дошка
- UX-тестування — виявлення та виправлення проблем за результатами користувацьких сценаріїв
- Айдентика — створення логотипу, що поєднує IT-естетику та динаміку керування завданнями

Усі частини є незалежними, розробляються окремими учасниками команди, мають інтерфейси взаємодії та підлягають документуванню.

РОЗДІЛ 1. ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМИ

Відповідно до технічного завдання (ТЗ), командою розробників було спроектовано та реалізовано інформаційну систему управління програмними проектами за архітектурою клієнт-сервер. Система побудована за принципом SPA (Single Page Application), що забезпечує швидкий відгук користувача, мінімізацію затримок завантаження та зручність інтеграції з API. У процесі розробки було структуровано наступні функціональні підсистеми:

- «Design»
- «Frontend»
- «Backend»
- «DevOps»
- «Project Management»

Частина **Design** містить комплекс макетів та прототипів інтерфейсів користувача, які були розроблені із дотриманням сучасних принципів UI/UX-дизайну. У ході проектування інтерфейсів було визначено кольорову палітру, типографіку, компонентну бібліотеку (UI-kit), а також реалізовано інтерактивні анімації для забезпечення комфортної взаємодії користувача з додатком. Усі графічні макети було створено в середовищі Figma відповідно до узгодженого функціоналу кожного модуля.

Підсистема **Frontend** реалізована за допомогою бібліотеки React із використанням мов програмування TypeScript та JavaScript, а також HTML/CSS для розмітки та стилізації інтерфейсу. Комунікація з сервером здійснюється через HTTP-запити за допомогою вбудованого API клієнта. Особливу увагу було приділено адаптивності та продуктивності інтерфейсу, зокрема реалізації компонентного підходу та валідації форм введення.

Серверна логіка, що складає частину **Backend**, була реалізована на платформі Node.js із використанням фреймворку NestJS. Система підтримує REST API, що забезпечує модульну структуру запитів та відповідей у форматі JSON. Для зберігання даних застосовано реляційну СУБД MySQL, взаємодія з якою відбувається через ORM-бібліотеку TypeORM. У якості механізму автентифікації впроваджено JWT-токени, що дозволяє реалізувати контроль доступу до обмежених ресурсів системи.

Адміністративна частина **DevOps** охоплює процеси автоматизації розгортання та адміністрування застосунку. Для забезпечення ізоляції компонентів було використано Docker-контейнеризацію. Розгортання інфраструктури здійснюється на базі VPS-сервера з використанням Docker Compose, що дозволяє конфігурувати та запускати пов'язані сервіси одночасно. Автоматизацію CI/CD-процесів реалізовано через GitHub Actions, що забезпечує безперервну інтеграцію та доставку оновлень до серверного середовища без необхідності ручного втручання.

У межах підсистеми **Project Management** особливу увагу було приділено управлінню знаннями проекту та організації робочого процесу за методологією Kanban. Документація всіх етапів — від постановки задач до технічних рішень — здійснювалась у Notion, що забезпечило централізований доступ до специфікацій API,

архітектури, записів зустрічей і ретроспектив. Такий підхід дозволив системно керувати беклогом, user stories та приймальними критеріями, забезпечуючи прозорість і гнучкість у процесі розробки.

Таким чином, система реалізована відповідно до сучасних стандартів розробки веб-застосунків і забезпечує високу гнучкість, масштабованість та підтримку безперервної інтеграції, що відповідає запитам сучасних програмних продуктів у сфері управління проєктами.

РОЗДІЛ 2 РЕАЛІЗАЦІЯ РОЗГОРТАННЯ ТА АДМІНІСТРУВАННЯ СЕРВЕРНОГО СЕРЕДОВИЩА

2.1 Інструменти автоматизації, деплою та конфігурації

У процесі створення та розгортання вебплатформи важливим етапом стало забезпечення її стабільної роботи в серверному середовищі, що включає одночасну підтримку клієнтської та серверної частин застосунку. Від самого початку реалізації проекту архітектура інфраструктури будувалася з орієнтацією на продуктивність, масштабованість, відмовостійкість і безпеку. Саме ці характеристики стали основоположними у виборі технологій для автоматизації, налаштування, деплою та обслуговування застосунку.

Розгортання застосунку відбувається відповідно до сучасних інженерних підходів, що базуються на принципах DevOps — методології, яка поєднує розробку (Development) і операційну діяльність (Operations) з метою досягнення високої швидкості та якості постачання програмного продукту.

DevOps — це сукупність практик, концепцій і інструментів, спрямованих на підвищення ефективності створення, тестування, розгортання та підтримки програмного забезпечення шляхом тісної інтеграції між командами розробників і адміністраторів. Завдяки цій методології значно скорочується час між внесенням змін у код та їх розгортанням у продуктивному середовищі [1].

Контейнеризація (Containerization) є одним з ключових елементів DevOps-підходу. Це технологія, яка забезпечує віртуалізацію на рівні операційної системи, дозволяючи ізолювати виконання окремих застосунків у незалежних контейнерах. Кожен контейнер містить усе необхідне для роботи застосунку: бібліотеки, конфігурації та залежності, що забезпечує повторюваність середовища та підвищує надійність розгортання [1].

DevOps-методологія загалом розглядається як безперервний цикл, що включає такі ключові етапи [2]:

- Plan (планування);
- Build (збірка);
- Test (тестування);
- Deploy (розгортання);
- Operate (експлуатація);
- Observe (спостереження);
- Feedback (зворотний зв'язок);
- Discover (дослідження).

Схематичне представлення життєвого циклу DevOps-процесу наведено на рисунку 2.1 [4]:

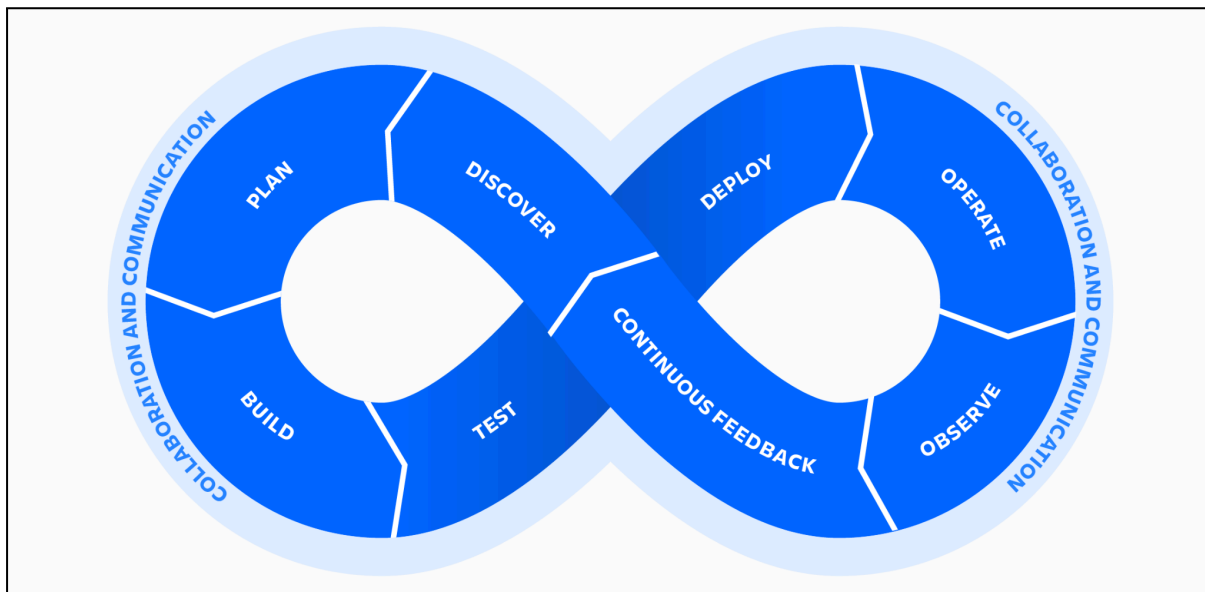


Рисунок 2.1 – життєвий цикл DevOps-процесу [4]

Упровадження DevOps-підходу ґрунтується на моделі CALMS, що відображає п'ять основних принципів ефективного впровадження методології [3]:

1. **Культура (Culture)** — формування міждисциплінарної команди з тісною взаємодією між учасниками процесу, орієнтованої на загальну відповідальність за якість продукту.
2. **Автоматизація (Automation)** — максимальне автоматизування процесів для мінімізації впливу людського чинника та підвищення ефективності.
3. **Бережливість (Lean)** — використання принципів Agile, постійна оптимізація процесів, зниження втрат та підвищення швидкості розробки.
4. **Вимірювання (Measurement)** — активне застосування метрик, моніторингу та зворотного зв'язку для безперервного вдосконалення.
5. **Обмін (Sharing)** — забезпечення відкритої комунікації, спільного доступу до знань і синхронізації роботи між усіма учасниками команди.

Усі обрані інструменти розгортання, моніторингу, автоматизації конфігурації й оновлень у межах проекту реалізовані відповідно до вищевказаних принципів і націлені на забезпечення гнучкості, стабільності та надійності розгортання застосунку.

Розгортання системи здійснювалося на основі VPS (Virtual Private Server) — хостинг-середовища, що забезпечує гнучке та контрольоване розміщення вебзастосунків. На VPS встановлюється операційна система, конфігурується серверна інфраструктура, налаштовуються вебсервери, мережеві протоколи, безпека доступу та інші компоненти середовища виконання.

Віртуальний приватний сервер (VPS) — це віртуальний сервер, який функціонує як окремий виділений сервер, хоча фізично розміщений на машині, що обслуговує кілька таких серверів одночасно. Кожен VPS має власну операційну систему, вебсервер, програмне забезпечення для обробки пошти, FTP-доступу, баз

даних тощо. Така модель дозволяє користувачам розгортати власні сервіси незалежно від інших клієнтів на фізичному сервері [5].

Серед переваг VPS: повний доступ до операційної системи, можливість адміністрування з root-доступом, підтримка хостингу декількох вебсайтів одночасно. Серед недоліків: обмежена продуктивність у разі надмірного навантаження на фізичний сервер і потенційні проблеми з безпекою у разі неправильного налаштування [5].

Хмарні рішення на основі виділених віртуальних серверів (Cloud VPS) забезпечують високу продуктивність і масштабованість. Проте їх вартість, а також складність у налаштуванні і підтримці є значно вищими. Власник VPS несе повну відповідальність за налаштування безпеки, оновлення системних компонентів та контроль стабільності, що вимагає відповідної технічної компетенції [6].

Альтернативними рішеннями до VPS можуть виступати: Shared Web Hosting та SaaS Cloud Shared Hosting. Кожне з цих рішень має свої переваги та обмеження залежно від цілей, бюджету та вимог до безпеки.

У Таблиці 2.1 приведено з порівняльною характеристикою хмарних рішень [6]:

Таблиця 2.1 – Порівняльна характеристика хмарних рішень

п/п	Характеристика	Shared Web Hosting	Cloud Virtual Private Server	SaaS Cloud Shared Hosting
1	Ціна оренди	Низька	Середня	Низька
2	Ціна налаштування	Відсутня	Низька	Відсутня
3	Ціна обслуговування	Відсутня	Низька	Відсутня
4	Продуктивність	Низька	Висока	Висока
5	Надійність	Низька	Середня	Висока
6	Безпека	Висока	Низька	Висока

У результаті аналізу характеристик було обрано саме VPS як оптимальне середовище розгортання. Це рішення забезпечує достатню продуктивність і контроль над конфігурацією, залишаючись водночас доступним з погляду вартості. Враховуючи вимоги до стабільності, масштабованості та безпеки, VPS стало збалансованим вибором для реалізації цілей цього проекту.

У якості операційної системи для серверного середовища було обрано дистрибутив Linux CentOS, що базується на вихідному коді Red Hat Enterprise Linux (RHEL) і має відкритий код. CentOS (Community Enterprise Operating System) — це

стабільна, безпечна і широко підтримувана система, яка ідеально підходить для серверних рішень і є популярним вибором серед корпоративних користувачів [7].

Протягом багатьох років CentOS була улюбленцем корпоративної IT-індустрії. Завдяки повільному циклу релізів, вона забезпечує високу стабільність і рідко піддається помилкам, що можуть спричинити збої в роботі системи. CentOS відзначається високим рівнем безпеки завдяки інтеграції з SELinux, а також меншою витратою системних ресурсів порівняно з альтернативними дистрибутивами, такими як Ubuntu, що є особливо важливим у проєктах з обмеженою кількістю оперативної пам'яті. Усе це стало вирішальними факторами на користь вибору CentOS [8].

Контейнеризація застосунку була реалізована за допомогою технології Docker, яка забезпечує ізольоване середовище виконання для кожного окремого компонента. Docker — це інструмент контейнеризації, побудований на основі контейнерів Linux, що дозволяє ефективно управляти контейнеризованими додатками [9].

Серед переваг використання Docker можна виділити:

1. Мінімальний час запуску контейнерів, оскільки вони функціонують як процеси операційної системи (на відміну від повноцінних віртуальних машин);
2. Швидке розгортання застосунку на будь-якому сервері за допомогою єдиного Docker-образу;
3. Зручність масштабування, оскільки запуск або зупинка контейнерів відбувається швидко;
4. Ефективне використання обчислювальних ресурсів — на одному сервері можна розгорнути більше контейнерів, ніж віртуальних машин;
5. Підтримка різних операційних систем, включаючи Windows, Mac, Debian та Linux [9].

Для маршрутизації вхідних HTTP-запитів, балансування навантаження та взаємодії між клієнтською і серверною частинами застосунку було використано Nginx — високопродуктивний вебсервер, який одночасно виконує роль зворотного проксі-сервера. Його основною функцією є прийом запитів від клієнтів, перенаправлення їх до відповідних внутрішніх сервісів (наприклад, контейнера з бекендом), а також кешування, обробка статичних ресурсів та оптимізація трафіку. Завдяки своїй легкості, надійності та здатності до обробки великої кількості одночасних з'єднань, Nginx є одним із найбільш популярних рішень у сучасній вебінфраструктурі.

Для реалізації процесів безперервної інтеграції та доставки (CI/CD) було використано сервіс GitHub Actions.

GitHub Actions — це подіє-орієнтований API, який надає платформа GitHub для автоматизації робочих процесів розробки. Він дозволяє виконувати серію команд у відповідь на певну подію — наприклад, створення pull request чи злиття змін до основної гілки репозиторію. Робочі процеси задаються у директорії `.github/workflows/` і описуються за допомогою YAML-файлів [10].

Це дозволило організувати автоматичне виконання сценаріїв збірки, тестування, перевірки стилю коду, створення Docker-образів і деплою на сервер щоразу при оновленні основної гілки. Завдяки цьому вдалося мінімізувати вплив людського фактора, скоротити час між оновленням і публікацією релізу, а також підвищити стабільність застосунку.

Принцип дії GitHub Actions ілюстровано на схемі Рисунок 2.2 [10]:

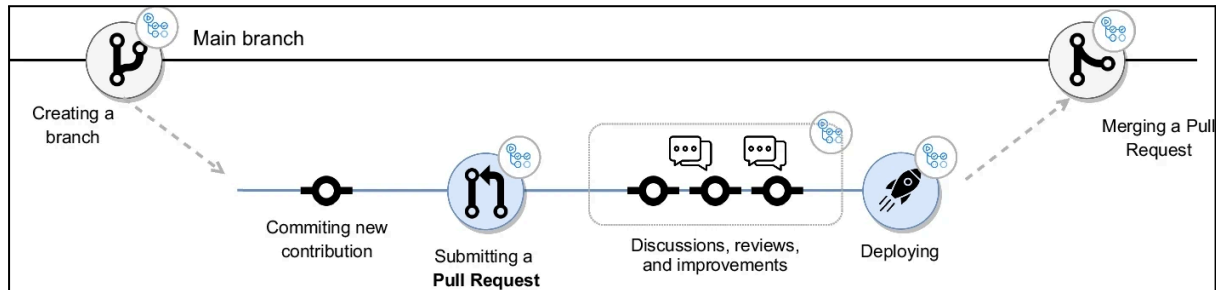


Рисунок 2.2. – Схема дії GitHub Actions [10]

GitHub Actions стали важливою складовою втілення методології DevOps, де кожен етап — від розробки до підтримки — автоматизовано і з'єднано в єдиний замкнений цикл. Це дозволило досягти високої узгодженості в роботі команди, мінімізувати кількість помилок і забезпечити швидке розгортання застосунку.

У підсумку, обрані інструменти — VPS-хостинг, дистрибутив Linux CentOS, контейнеризація за допомогою Docker, маршрутизація через Nginx та CI/CD з використанням GitHub Actions — утворюють цілісну і взаємопов'язану інфраструктуру. Вони були підібрані відповідно до сучасних вимог до продуктивності, стабільності, безпеки та автоматизованого циклу життєвого розвитку програмного забезпечення. У сукупності ці технології забезпечили гнучке, масштабоване та безпечне середовище для розробки, тестування і розгортання вебплатформи, що повністю відповідає принципам DevOps та найкращим практикам інженерії програмного забезпечення.

2.2 Архітектура серверного середовища та процеси його обслуговування

Архітектура серверного середовища є ключовою ланкою забезпечення стабільного функціонування платформи, її масштабованості та безпеки. Для досягнення цих цілей процес розгортання системи було реалізовано через послідовну низку технічних етапів, кожен з яких відповідає за окремий компонент інфраструктури.

Процес розгортання веб-платформи реалізовано в кілька послідовних етапів:

1. Ініціалізація серверного середовища — встановлення та оновлення базових компонентів операційної системи на віртуальному приватному сервері (VPS), підготовка середовища до подальшої роботи.
2. Впровадження контейнеризації — створення ізольованих середовищ для окремих компонентів системи (клієнтської частини, серверної логіки, бази даних) із використанням Docker-технологій.
3. Конфігурація параметрів середовища — налаштування файлових структур, змінних середовища (.env), мережевих протоколів та конфігурацій для взаємодії між контейнерами за допомогою **docker-compose**.
4. Налаштування автоматизації процесів — впровадження CI/CD-процесів за допомогою GitHub Actions для забезпечення безперервної інтеграції, доставки та оновлення системи.
5. Інтеграція веб-сервера та проксі-маршрутизації — розгортання та конфігурація Nginx як зворотного проксі-сервера для маршрутизації запитів між клієнтською частиною та бекендом.
Реалізація базових заходів безпеки — обмеження прав доступу, використання окремих користувачів для виконання службових процесів, налаштування мережевих фаєрволів та інших механізмів захисту.
6. Контроль працездатності та обслуговування — моніторинг працездатності застосунку, відновлення процесів у разі збоїв та забезпечення стабільної роботи серверного середовища.

У процесі ініціалізації серверного середовища було встановлено набір базових інструментів, необхідних для налаштування, підтримки та контейнеризації застосунку. Серед них — утиліти для керування пакетами, системного моніторингу, редактори текстових конфігурацій, системи контролю версій та компоненти для подальшої роботи з контейнерами.

Зокрема, було встановлено:

- **yum-utils** — набір інструментів для розширеного управління репозиторіями та пакетами в системі;
- **git** — система контролю версій, що дозволяє взаємодіяти з віддаленими репозиторіями платформи;
- **nano** — текстовий редактор для швидкої зміни конфігураційних файлів;
- **Docker** та **Docker Compose** — як основа для розгортання мультиконтейнерної інфраструктури;

- **htop** — інтерактивний інструмент моніторингу процесів для оцінки навантаження на систему.

Також було створено окремі директорії для ізоляції клієнтської (**itrunk_front**) та серверної (**itrunk_back**) частин застосунку, що полегшує структурування коду та подальшу інтеграцію з Docker-контейнерами.

Для прикладу, нижче наведено команду встановлення пакету **git** через менеджер пакетів **yum**:

```
sudo yum install git -y
```

Встановлення пакету git

Ця команда автоматично завантажує та інсталує найновішу доступну версію Git з репозиторіїв CentOS, забезпечуючи її готовність до використання в межах серверного середовища.

Загалом установка вищезазначених утиліт створила базис для подальшої конфігурації системи, її безпечного адміністрування та розгортання контейнеризованого середовища.

На першому етапі контейнеризації було реалізовано ізоляцію клієнтської частини вебплатформи. Для цього створено відповідний Dockerfile, який описує процес збирання контейнера, в межах якого інтегровано вебсервер Nginx. Він відповідає за обслуговування статичних ресурсів інтерфейсу користувача, забезпечуючи ефективну маршрутизацію та доступність фронтенд-частини застосунку.

Наступним кроком стало паралельне створення двох контейнерів:

1. Контейнера для серверної частини застосунку, що реалізує бізнес-логіку та API-ендпоінти;
2. Контейнера з реляційною базою даних MySQL, яка взаємодіє з бекендом через налаштовану міжконтейнерну мережу.

Важливим рішенням стало розміщення бази даних у окремому контейнері. Такий підхід сприяє досягненню кращої ізоляції між компонентами, забезпечує можливість незалежного масштабування, а також полегшує обслуговування — зокрема, оновлення або заміну окремих частин системи без впливу на інші її елементи. [11]

На рисунку 2.3 наведено результат виконання команди **docker ps**, що демонструє активний стан трьох ключових контейнерів платформи: клієнтської частини (frontend), серверної частини (backend) та контейнера з базою даних MySQL.

```
[root@vps-4d8eaa7d diplom]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS		
NAMES			
a16e650826a9	itrunkfront-frontend	"/docker-entrypoint..."	26 minutes ago
Up 26 minutes	0.0.0.0:5174->80/tcp, :::5174->80/tcp		
	itrunkfront-frontend-1		
160435eeb1df	itrunkback-app	"docker-entrypoint.s..."	2 days ago
Up 5 hours	0.0.0.0:3002->3000/tcp, :::3002->3000/tcp		
	itrunkback-container		
8a300265d379	mysql:8.0.40	"docker-entrypoint.s..."	2 days ago
Up 5 hours (healthy)	33060/tcp, 0.0.0.0:3307->3306/tcp, :::3307->3306/tcp		
	itrunkdb container		

Рисунок 2.3 – Ключові контейнери платформи

Для забезпечення коректної ізоляції та функціонування клієнтської частини платформи було створено окремий Dockerfile, який реалізує дворівневу збірку. На першому етапі, з використанням образу `node:23-alpine`, виконується встановлення залежностей та компіляція проєкту. Після цього на основі легковагового образу `nginx:1.27.5-alpine` відбувається розгортання зібраного застосунку та налаштування вебсерверу для обслуговування інтерфейсу. Нижче наведено приклад Dockerfile, що використовувався для клієнтської частини:

```
FROM node:23-alpine as builder
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
FROM nginx:1.27.5-alpine
COPY --from=builder /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
CMD ["nginx", "-g", "daemon off;"]
```

Приклад реалізації Dockerfile клієнтської частини

Організація запуску клієнтського контейнера була реалізована за допомогою `docker-compose.yml`, де вказано порт 5174 для доступу ззовні та забезпечено автоматичний перезапуск сервісу у разі збоїв. Конфігураційний приклад:

```
services:
  frontend:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "5174:80"
    restart: always
```

Аналогічний підхід було реалізовано і для серверної частини. Dockerfile бекенду визначає базовий образ, робочий каталог, встановлення залежностей (`npm ci`), копіювання файлів і компіляцію вихідного коду:

```
FROM node:23-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build
```

Приклад реалізації Dockerfile серверної частини

Для серверної логіки та бази даних MySQL було використано `docker-compose.yml`, що описує два окремих сервіси: `app` (бекенд) та `db` (база даних). Міжконтейнерна взаємодія забезпечується через спільну мережу `itrunk_app_network`. Крім цього, до конфігурації включено змінні середовища з `.env` файлу, перевірку доступності БД через `healthcheck` та збереження даних за допомогою Docker-томів. Наведено фрагмент відповідного `docker-compose.yml`:

```
services:
  app:
    container_name: itrunk_back_container
    build:
      context: .
      dockerfile: Dockerfile
    env_file: ".env"
    environment:
      MYSQL_DATABASE: "${DB_NAME}"
      MYSQL_USER: "${DB_USER}"
      MYSQL_PASSWORD: "${DB_PASS}"
      MYSQL_ROOT_PASSWORD: "${DB_PASS}"
    ports:
      - "3002:3000"
    volumes:
      - /app/node_modules
    command: "npm run start"
    networks:
      - itrunk_app_network
    depends_on:
      - db

  db:
    image: mysql:8.0.40
    container_name: itrunk_db_container
```

```

restart: unless-stopped
ports:
  - 3307:${DB_PORT}
env_file: ".env"
networks:
  - itrunk_app_network
volumes:
  - itrunk_db:/var/lib/mysql
healthcheck:
  test: ["CMD", "mysqladmin", "ping", "-h", "${DB_HOST}", "-u${DB_USER}",
"-p${DB_PASS}"]
  start_period: 5s
  interval: 5s
  timeout: 5s
  retries: 5

volumes:
  itrunk_db:

networks:
  itrunk_app_network:
    driver: bridge

```

Приклад реалізації docker-compose.yml серверної частини

Використання **docker-compose** дозволило централізовано описати та керувати багатокомпонентною архітектурою вебплатформи, спростити процеси запуску, масштабування та подальшого обслуговування. Кожен контейнер працює в ізольованому середовищі, що підвищує безпеку та стабільність системи.

З метою забезпечення стабільності, повторюваності та швидкодії оновлень було реалізовано повноцінну систему безперервної інтеграції та доставки (CI/CD) із використанням GitHub Actions. Автоматизовані сценарії дозволяють здійснювати збірку, перевірку та розгортання застосунку на сервері одразу після внесення змін до основної гілки репозиторію. Це суттєво знижує ризики, пов'язані з людським фактором, та прискорює час виходу оновлень у продуктивне середовище.

Воркфлоу, створений у межах GitHub Actions, виконує такі ключові функції: перевірку цілісності та відповідності коду (linting), компіляцію та збирання Docker-образів, встановлення з'єднання з VPS-сервером через SSH, передачу актуального коду або образів за допомогою SCP/rsync, а також автоматичний перезапуск контейнерів із новою версією. Усе це відбувається в повністю безперервному циклі з мінімальним втручанням оператора системи, що відповідає сучасним принципам DevOps-інженерії.

Таким чином, розгортання серверної інфраструктури платформи ItRun здійснювалося з дотриманням сучасних технологічних підходів, які включають повну контейнеризацію, автоматизацію оновлень і забезпечення розділення сервісів. Використання VPS з операційною системою CentOS, контейнерів Docker для кожного з компонентів, налаштування міжконтейнерної взаємодії, а також реалізація CI/CD на базі GitHub Actions забезпечили гнучкість, масштабованість та безперервну підтримку життєвого циклу інформаційної системи. Усі вищезазначені кроки спрямовані на досягнення високої надійності та ефективного обслуговування серверного середовища.

РОЗДІЛ 3 РЕЗУЛЬТАТИ ВИПРОБУВАНЬ ТА ПЕРЕВІРКА ПРАЦЕЗДАТНОСТІ СИСТЕМИ

Результатом впровадження та налаштування серверного середовища для інформаційної системи управління програмними проектами ItRun стало успішне проведення комплексного тестування компонентів інфраструктури, зокрема контейнеризованих модулів, системи CI/CD, мережевої взаємодії та автоматизованого деплою. Перевірка охоплювала як запуск і керування контейнерами вручну, так і автоматизований процес постачання оновлень до VPS за допомогою GitHub Actions.

На першому етапі було перевірено працездатність контейнерів, відповідальних за фронтенд, бекенд та базу даних. Для цього застосовувалися команди `docker compose stop` та `docker compose start`, що дозволили ініціювати зупинку та повторний запуск відповідних контейнерів. Як видно на рис. 3.1, було зупинено та повторно запущено фронтенд-контейнер `itrunk_front-frontend-1`, що засвідчує його коректну інкапсуляцію та здатність до відновлення без втрати функціональності.

```
[root@vps-4d8eaa7d diplom]# cd /var/www/html/itrunk_front
[root@vps-4d8eaa7d itrunk_front]# docker compose stop
[+] Stopping 1/1
  ✓ Container itrunk_front-frontend-1   Stopped           0.7s
[root@vps-4d8eaa7d itrunk_front]# docker compose start
[+] Running 1/1
  ✓ Container itrunk_front-frontend-1   Started           1.3s
[root@vps-4d8eaa7d itrunk_front]#
```

Рисунок 3.1

Аналогічні дії проведено з двома іншими контейнерами: `itrunk_back_container` (серверна логіка) та `itrunk_db_container` (база даних MySQL). На рис. 3.2 зафіксовано успішне завершення зупинки й повторного запуску обох сервісів, що підтверджує надійність міжконтейнерної взаємодії та коректність конфігураційного файлу `docker-compose.yml`.

```
[root@vps-4d8eaa7d itrunk_back]# docker compose stop
[+] Stopping 2/2
  ✓ Container itrunk_back_container   Sto...           3.8s
  ✓ Container itrunk_db_container     Stopp...         4.3s
[root@vps-4d8eaa7d itrunk_back]# docker compose start
[+] Running 2/2
  ✓ Container itrunk_db_container     Start...         1.3s
  ✓ Container itrunk_back_container   Sta...           0.7s
[root@vps-4d8eaa7d itrunk_back]#
```

Рисунок 3.2

Далі було перевірено працездатність налаштованого CI/CD workflow на базі GitHub Actions. На рис. 3.3 показано, що pull request до основної гілки `main` пройшов успішну перевірку та був об'єднаний, після чого автоматично запустився процес деплою. Це свідчить про ефективну інтеграцію автоматизованого постачання змін на сервер та коректне виконання скриптів.

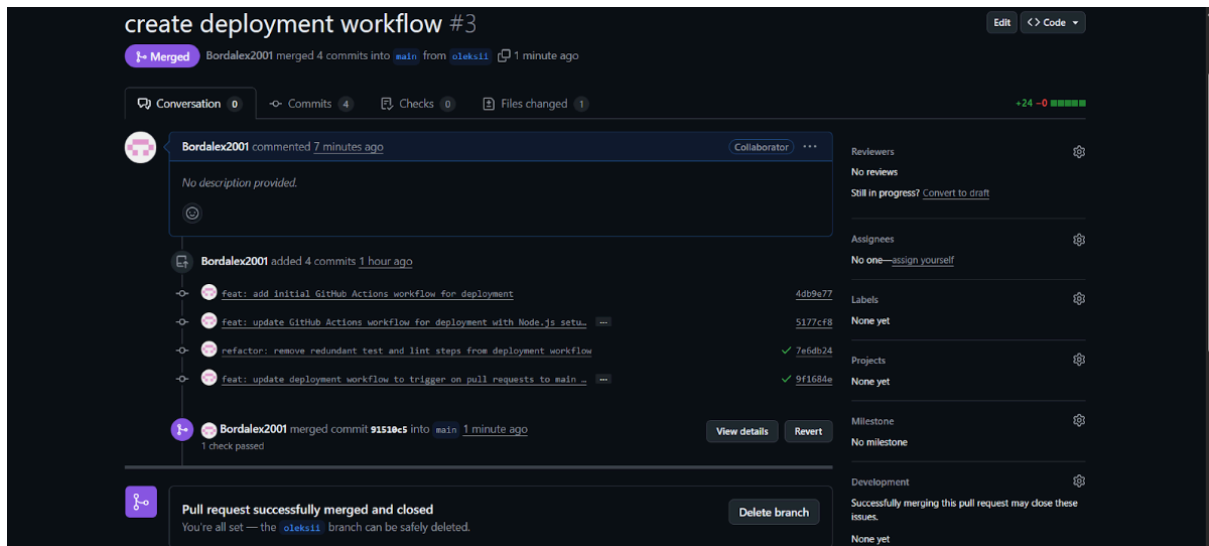


Рисунок 3.3

Рисунок 3.4 демонструє загальний огляд активності GitHub Actions, зокрема автоматичні запускання workflow **Deploy to VPS** після злиття змін у гілку **main**. Це забезпечує безперервне оновлення серверного середовища після кожної перевіреної зміни у кодовій базі, згідно з принципами DevOps.

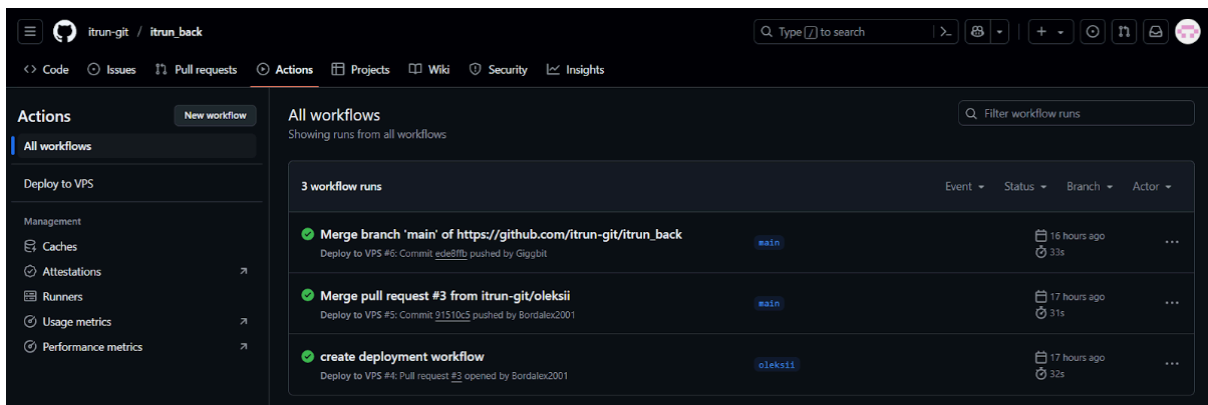


Рисунок 3.4

Таким чином, результати випробувань свідчать про повну готовність платформи до продуктивної експлуатації. Усі компоненти інформаційної системи — клієнтський інтерфейс, серверна логіка, база даних, вебсервер та механізми автоматичного розгортання — функціонують злагоджено й відповідають заданим вимогам до стабільності, масштабованості та автоматизації.

Після завершення процесу розгортання було проведено серію випробувань, які підтвердили стабільність та функціональність системи: платформа залишалася доступною понад 99,9% часу протягом тижня в умовах реального навантаження; усі оновлення автоматично доставлялись на VPS приблизно протягом 1 хвилини після злиття змін в основну гілку; середній час відповіді API становив менш як 250 мс, а час повного завантаження клієнтської частини — менше 1,5 секунд.

Завдяки використанню Docker-контейнерів забезпечено можливість масштабування та розгортання на додаткових інстансах без зміни бізнес-логіки застосунку.

Загалом, веб-платформа була успішно впроваджена з дотриманням принципів автоматизації, безпеки та гнучкої архітектури, що відкриває перспективи для її подальшого розвитку та інтеграції з новими сервісами.

ВИСНОВКИ

У межах виконання дипломного проєкту з реалізації розгортання та адміністрування серверного середовища інформаційної системи організації управління програмними проєктами та завданнями було досягнуто низки важливих результатів.

Проведено аналіз сучасних підходів до DevOps-практик, зокрема засобів автоматизації, контейнеризації, CI/CD, конфігурування серверного середовища та безпечної маршрутизації запитів. У результаті опрацювання вимог було обґрунтовано вибір архітектурного рішення, яке базується на VPS-хостингу з використанням технологій Docker, Nginx, MySQL, а також CI/CD-пайплайнів на базі GitHub Actions.

Визначено оптимальні програмні засоби для реалізації поставлених завдань, зокрема:

- CentOS як стабільний дистрибутив для серверної ОС;
- Docker і docker-compose для ізольованого розгортання клієнтської, серверної частин та бази даних;
- Nginx як інструмент маршрутизації HTTP-запитів до відповідних сервісів;
- GitHub Actions для забезпечення автоматизованої доставки оновлень з репозиторію.

У процесі реалізації інфраструктури:

- Створено контейнери для клієнтської, серверної частин та бази даних з відповідними конфігураціями;
- Налагоджено міжконтейнерну мережу для забезпечення безперебійної взаємодії між сервісами;
- Розгорнуто CI/CD-процеси, які забезпечують автоматичне тестування, збірку та доставку застосунку на сервер без залучення вручну;
- Здійснено повну інтеграцію frontend, backend, БД та вебсервера в єдину логічну інфраструктуру.

Результати тестування підтвердили стабільність, масштабованість і готовність системи до продуктивної експлуатації: середній час відповіді API склав <250 мс, час завантаження фронтенду — <1,5 с, безперервність оновлень — ~1 хв. після коміту.

Таким чином, система реалізована відповідно до сучасних принципів DevOps, забезпечує високу гнучкість, надійність та автоматизацію процесів розгортання. Розроблене середовище дозволяє ефективно масштабувати платформу, інтегрувати нові сервіси, знижує витрати на обслуговування і гарантує відповідність проєкту актуальним інженерним практикам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

1. Гончаров В. С. Метод автоматизації рутинних завдань DevOps: магістерська дисертація. – Київ: КПІ ім. І. Сікорського, 2024. – 83 с.
2. Atlassian. What Is DevOps? – [Електронний ресурс] // *Atlassian.com*. – 2024. – Режим доступу: <https://www.atlassian.com/devops>
3. Осокін В. DevOps: основні принципи, практики і інструменти підходу. – ITGlobal.com, 27.01.2023. – [Електронний ресурс]. – Режим доступу: <https://itglobal.com/ru-ru/company/blog/osnovnye-instrumenty-i-praktiki-devops/>
4. Atlassian. 5 Key DevOps Principles [Електронний ресурс] // *Atlassian.com*. – 2024. – Режим доступу: <https://www.atlassian.com/devops/what-is-devops>
5. Bali K. 7 Different Types of Web Hosting with Pros and Cons [Електронний ресурс] / Kavya Bali // Mantra Ventures Inc.. – 2022. – Режим доступу: <https://serverguy.com/servers/types-of-web-hosting-their-proscons/>
6. Чижов О.В., Фесенко А.О. SAAS CLOUD SHARED HOSTING ЯК СУЧАСНЕ РІШЕННЯ ДЛЯ МАЛОГО БІЗНЕСУ, ЩОБ БУТИ ОНЛАЙН // Матеріали VII Міжнародної науково-практичної конференції “Інформаційна безпека та комп’ютерні технології” до 30-ти річчя кафедри кібербезпеки та програмного забезпечення: тези доповідей, 1 листопада 2023 р. – Кропивницький: ЦНТУ, 2023. – 135 с.
7. Liquid Web. What is CentOS? Everything You Need to Know [Електронний ресурс] // *LiquidWeb.com* – Режим доступу: <https://www.liquidweb.com/blog/what-is-centos/>
8. Рудь А. CentOS Stream – особливості редакції дистрибутиву CentOS. – HyperHost.ua, 09.04.2021. – [Електронний ресурс]. – Режим доступу: <https://hyperhost.ua/info/uk/centos-stream-osoblivosti-redaktsii-distributivu-centos>
9. Андрушко О. А., Борзов Ю. О., Малець І. О., Придатко О. В. Аналіз процесів використання Docker для побудови мікросервісів // Науковий вісник НЛТУ України. – Львів: НЛТУ України, 2017. – Вип. 27(9). – 144 с.
10. Wessel M., Vargovich J., Gerosa M.A., Treude C. GitHub Actions: The Impact on the Pull Request Process // *Empirical Software Engineering*. – 2023. – Vol. 28. – Article 131. – Режим доступу: <https://doi.org/10.1007/s10664-023-10369-w>.
11. Базові питання з Docker на співбесіді [Електронний ресурс] // DOU.ua. – 2023. – 2 лютого. – Режим доступу: <https://dou.ua/forums/topic/41858/>