Науковий вісник НЛТУ України
Scientific Bulletin of UNFU
https://doi.org/10.36930/40360309
Article received: 09.04.2026
Article accepted: 25.05.2026
Article published: 29.06.2026
Correspondence author: B. O. Bekas bohdan.bekas@nltu.edu.ua
ISSN 1994-7836 (print)
ISSN 2519-2477 (online)

UDC 004.[02+03+42+451+47]

Б. О. Бекас[1], А. М. Проць[1], Ю. І. Грицюк[1,2]

[1] Національний лісотехнічний університет України, м. Львів, Україна
[2] Національний університет "Львівська політехніка", м. Львів, Україна

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ІЗОЛЬОВАНОГО РОЗГОРТАННЯ ВЕБЗАСТОСУНКУ У КОНТЕЙНЕРИЗОВАНОМУ СЕРЕДОВИЩІ

Наведено результати розроблення програмного забезпечення (ПЗ) для ізольованого розгортання вебзастосунку у контейнеризованому середовищі. Встановлено, що традиційним підходам до розгортання застосунків (традиційний деплой) характерні складність налаштування середовища виконання, управління залежностями та відмінностями між локальним і серверним середовищами виконання, що призводить до збільшення тривалості розгортання застосунку та підвищення ймовірності виникнення помилок. Визначено, що використання технологій контейнеризації дає змогу реалізувати ізольоване середовище виконання застосунку та істотно підвищити його відтворюваність, портативність і надійність роботи. Розроблено вебзастосунок для управління заявками користувачів у сфері соціального захисту населення з повністю контейнеризованим його розгортанням на основі технології Docker та інструменту Docker Compose. Клієнтську частину вебзастосунку реалізовано засобами бібліотеки React та інструменту збірки Vite з використанням утиліти TailwindCSS для адаптивного інтерфейсу. Серверну частину вебзастосунку побудовано на середовищі виконання Node.js з фреймворком Express.js, ефективність роботи яких забезпечує REST API (англ. *Representational State Transfer Application Programming Interface*), JWT-автентифікацію, оброблення файлів засобами Multer та інтеграцію з AI-асистентом через хмарну інфраструктуру OpenAI API. Дані зберігаються у СУБД MySQL 8.0. Кожен компонент системи розгортається в окремому Docker-контейнері з оркестрацією через інструмент Docker Compose. Порівняльний аналіз ефективності різних підходів до розгортання вебзастосунку засвідчив, що контейнеризований підхід істотно перевершує традиційний за критеріями відтворюваності середовища виконання, ізоляції компонентів та швидкості його повторного розгортання. Тестування 44 випадків розгортання вебзастосунків підтвердило 100 % функціональність та надійність системи. Розроблене ПЗ можуть застосовувати органи місцевого самоврядування, соціальні служби та інші установи, що здійснюють приймання й оброблення заявок від населення в режимі онлайн. Упровадження контейнеризованого підходу до розгортання вебзастосунків є перспективним напрямом підвищення ефективності цифрової трансформації у сфері соціального захисту населення.

Ключові слова: контейнеризація застосунків; платформа Docker; інструмент Docker Compose; архітектурний стиль для створення вебслужб REST API; клієнт-серверна модель; ізольоване розгортання вебзастосунку; бібліотека React; середовище виконання Node.js.

Вступ / Introduction

Вебзастосунки є невід'ємною складовою частиною сучасної цифрової інфраструктури, які широко застосовують у різних галузях, зокрема – у сфері надання державних послуг, бізнесу та соціального забезпечення. Важливим етапом життєвого циклу є розгортання вебзастосунку, яке безпосередньо впливає на стабільність, надійність та ефективність функціонування програмного забезпечення (ПЗ). Розгортання (deployment) вебзастосунку передбачає перенесення готового до використання ПЗ із середовища виконання його розроблення на серверну інфраструктуру або хмарну платформу, де воно стає доступним для безпосередніх користувачів.

Традиційні підходи до розгортання вебзастосунків (традиційний деплой) передбачають розміщення коду безпосередньо на фізичних серверах або класичних віртуальних машинах (VPS/VDS) без автоматичного використання сучасних хмарних сервісів, контейнеризації (на кшталт Docker) чи Serverless-архітектури. У таких системах зазвичай використовується монолітна архітектура та стандартний триланковий шаблон: вебсервер, сервер застосунків та база даних.

Класичне розгортання (або традиційне деплоювання) застосунків часто супроводжується труднощами, пов'язаними з налаштуванням середовища виконання, управлінням залежностями та відмінностями між локальним і серверним середовищами. Розробник вебзастосунку локально його тестує на одній конфігурації операційної системи (ОС), але в разі розгортання на виробничому сервері з іншою версією ОС, інтерпретатора

Бекас Богдан Олексійович, ст. викладач, кафедра інформаційних систем і комп'ютерного моделювання.
Email: bohdan.bekas@nltu.edu.ua; https://orcid.org/0000-0002-3571-9600
Проць Андрій Миколайович, асистент, кафедра комп'ютерних наук.
Email: andrii.prots@nltu.edu.ua; https://orcid.org/0009-0005-5063-2828
Грицюк Юрій Іванович, д-р техн. наук, професор, кафедра програмного забезпечення.
Email: yurii.i.hrytsiuk@lpnu.ua; https://orcid.org/0000-0001-8183-3466

Цитування за ДСТУ: Бекас Б. О., Проць А. М., Грицюк Ю. І. Програмне забезпечення для ізольованого розгортання вебзастосунку у контейнеризованому середовищі. Науковий вісник НЛТУ України. 2026, т. 36, № 3. С. 86–97.
Citation APA: Bekas, B. O., Prots, A. M., & Hrytsiuk, Yu. I. (2026). Software for isolated deployment of a web application in a containerized environment. *Scientific Bulletin of UNFU*, 36(3), 86–97. https://doi.org/10.36930/40360309

або бібліотек можуть виникнути непередбачувані помилки. Це призводить до збільшення тривалості розгортання вебзастосунку, ускладнення процесу його супроводу та підвищення ймовірності виникнення помилок під час експлуатації системи.

Ефективним підходом до вирішення зазначених проблем є використання технологій контейнеризації СТ (англ. *Containerisation Technology*), які дають змогу реалізувати ізольоване середовище виконання ПЗ. Контейнеризація вебзастосунку передбачає його упакування разом з усіма залежностями (бібліотеки, фреймворки, системні утиліти) у вигляді окремого контейнера, що забезпечує уніфікованість середовища виконання, незалежність від конкретної інфраструктури та спрощення процесу його розгортання.

Наразі однією з найпопулярніших технологій контейнеризації вебзастосунків є Docker – це програмна платформа, яка дає змогу "упакувати" застосунок разом з усім його оточенням (бібліотеками, конфігураціями та залежностями) в один ізольований контейнер. Головна мета Docker – зробити так, щоб програма працювала однаково стабільно на комп'ютері розробника, тестовому сервері та реальному продакшені, усуваючи проблему "на моєму комп'ютері все працювало".

За даними статистики, понад 85 % підприємств, які використовують контейнеризацію застосунків, обирають саме програмну платформу Docker. Водночас, з точки зору науково-практичного дослідження, перевага контейнеризованого розгортання застосунку над традиційним підходом до його розгортання потребує додаткової верифікації через порівняльний аналіз метрик відтворюваності середовища виконання, тривалості розгортання застосунку та складності конфігурації ОС. Отримані результати в роботі [23] підтверджують цю гіпотезу, де тестування контейнеризованого розгортання вебзастосунку продемонструвало 100-відсоткову відтворюваність усіх функцій системи управління заявками користувачів під час запуску на різних операційних системах.

Актуальність теми дослідження полягає у необхідності розроблення ПЗ, яке забезпечує ізольоване розгортання вебзастосунків у контейнеризованому середовищі, що дає змогу підвищити ефективність процесу його розгортання, зменшити вплив зовнішніх факторів середовища виконання та забезпечує стабільну роботу системи.

Об'єкт дослідження – розгортання вебзастосунків у контейнеризованому середовищі.

Предмет дослідження – методи, засоби та технології забезпечення ізольованого розгортання вебзастосунків у контейнеризованому середовищі, які забезпечать відтворюваність, портативність, надійність роботи та ізоляцію середовища виконання вебзастосунків.

Мета роботи – розробити програмне забезпечення для ізольованого розгортання вебзастосунку у контейнеризованому середовищі, що дасть змогу провести порівняльний аналіз ефективності різних підходів до його розгортання та обґрунтувати переваги контейнеризації застосунків над традиційним ручним налагодженням його роботи.

Для досягнення зазначеної мети визначено такі *основні завдання дослідження*:

- проаналізувати сучасні підходи до розгортання вебзастосунків та дослідити принципи їх контейнеризації та ізоляції середовища виконання, що дасть змогу систематизувати наявні підходи до ізольованого розгортання застосунків та обґрунтувати вибір технологій їх контейнеризації;
- розробити функціональний вебзастосунок у визначеній предметній галузі (систему управління заявками користувачів у сфері соціального захисту), який забезпечить автоматизоване оброблення заявок громадян, здійснить управління їхніми статусами та уможливить взаємодію між заявниками й адміністраторами в режимі онлайн;
- реалізувати розгортання вебзастосунку традиційним монолітним методом, що дасть можливість отримати базові метрики тривалості та складності його розгортання для подальшого порівняльного аналізу;
- реалізувати ізольоване розгортання вебзастосунку із використанням технології Docker та інструменту Docker Compose, які забезпечать відтворюваність, портативність та ізоляцію всіх компонентів системи в єдиному контейнеризованому середовищі;
- провести комплексне тестування функціональності та надійності розгорнутої системи, що підтвердить відповідність розробленого ПЗ функціональним вимогам й обґрунтує надійність обраного підходу до розгортання вебзастосунку;
- здійснити детальний порівняльний аналіз ефективності обох підходів до розгортання вебзастосунку, що дасть змогу обґрунтувати рекомендації щодо їхнього застосування.

Аналіз останніх досліджень та публікацій. Проблема розгортання вебзастосунків та їх контейнеризації набула широкого відображення у сучасній науковій і технічній літературі. Головна мета методології полягає у пришвидшенні циклу розроблення, забезпеченні високої якості продукту проєкту та скороченні часу до виходу на ринок (Time to Market).

Набір практик та інструментів, що об'єднують розроблення ПЗ та його ІТ-експлуатацію. У роботі [14] систематизовано принципи DevOps (від англ. Development – розроблення та Operations – експлуатація) і безперервного доставлення (англ. *Continuous Delivery*) ПЗ, де його контейнеризацію розглядають як ключовий елемент автоматизації процесу розгортання. Автори наголошують, що уніфіковані контейнерні образи усувають клас проблем, пов'язаний із відмінностями середовищ виконання, що дає змогу скоротити цикл розроблення та розгортання ПЗ з місяців до днів або годин. Водночас, зв'язка Prometheus і Grafana [28] – це галузевий стандарт у сфері DevOps для збирання, збереження та візуалізації метрик системи в реальному часі. Разом вони працюють як єдине ціле: Prometheus виконує роль бази даних та збирача інформації, а Grafana забезпечує створення інтерактивних графіків і панелей керування.

У роботі [22] досліджено вплив контейнеризації на процес розгортання застосунків у DevOps. Контейнеризацію розглядають як підхід до ізольованого пакування застосунків разом із залежностями, що забезпечує їхню портативність, узгодженість і масштабованість. Проаналізовано її вплив на ключові параметри розгортання, зокрема швидкість, безпеку та керованість. На основі аналізу літературних джерел і практичних досліджень визначено основні переваги та обмеження використання контейнерів у DevOps, а також сформульовано рекомендації щодо їх впровадження. Отримані результати свідчать, що контейнеризація сприяє оптимізації процесів розгортання застосунків і підтримує прийняття обґрунтованих рішень щодо інтеграції цієї технології в життєвий цикл розроблення програмного забезпечення.

Технологія Docker та інструмент Docker Compose. Офіційна документація платформи Docker [8] описує архітектуру та принципи її роботи. Технологія Docker реалізує ізоляцію застосунків на рівні ОС через механізм просторів імен (namespaces) та контрольних груп (cgroups). На відміну від віртуальних машин, контейнери використовують спільне ядро ОС, що дає змогу зменшити споживання пам'яті та обчислювальних ресурсів, а також досягти вищої продуктивності системи. Документація інструменту Docker Compose [7] описує декларативний підхід до визначення та запуску багатоконтейнерних застосунків за допомогою YAML-конфігурації. Документація до платформи Kubernetes [16] охоплює принципи оркестрації контейнерів для великомасштабних систем, проте, на відміну від інструменту Docker Compose, призначена для промислових розподілених систем і для неї характерна вища складність налаштування.

У роботі [10] розглянуто особливості розгортання застосунків із акцентом на вплив контейнеризації, де Docker, як провідна платформа контейнеризації, забезпечує ефективне пакування, розгортання та керування застосунками, оптимізуючи використання обчислювальних ресурсів. Контейнеризацію визначають як метод ізольованого виконання застосунків, що забезпечує вищу ефективність порівняно з традиційною віртуалізацією. Використання Docker сприяє підвищенню узгодженості середовищ, продуктивності та керованості, а також частково вирішує проблеми безпеки й оркестрації кількох контейнерів. Реалізовано підхід, що передбачає пакування вебзастосунку в контейнер, його розгортання за допомогою Docker і подальше оцінювання продуктивності та зручності керування порівняно з традиційними методами.

У роботі [30] проаналізовано сучасні інструменти створення ізольованих середовищ для розроблення ПЗ. Розглянуто технології Docker і Vagrant, які реалізують різні підходи до віртуалізації: Vagrant ґрунтується на використанні віртуальних машин VM (англ. *Virtual Machines*), тоді як Docker застосовує контейнеризацію з використанням ізоляції на рівні ОС. Наголошено, що до впровадження цих інструментів існували значні труднощі перенесення, встановлення та відтворюваності ПЗ через відмінності в конфігураціях середовищ. Вказано на те, що технології віртуалізації (VM і контейнери) здатні ефективно вирішувати зазначені проблеми, однак їх практичне застосування ускладнюється відсутністю належних інструментів інтеграції у процеси розроблення. Проведено порівняльний аналіз Docker і Vagrant та визначено сфери їх доцільного застосування залежно від конкретних вимог.

У роботі [27] наведено результати контейнеризації навчального середовища для машинного навчання задля кібербезпеки. Автори вважають, що багато студентів мають труднощі з налаштуванням відповідного середовища кодування та отримання наборів даних на власних комп'ютерах, що певною мірою призводить до втрати дорогоцінного часу на вивчення основних тем машинного навчання та кібербезпеки. У роботі запропоновано підхід з контейнеризацією навчального середовища алгоритмом машинного навчання та набором даних. Це допоможе студентам отримати цінний практичний досвід роботи з контейнером Docker, а також позбутися проблем з налаштуванням середовища кодування та отримання набору даних.

Веброзроблення та технологічний стек. Документація до бібліотеки React [25] описує компонентний підхід до розроблення користувацьких інтерфейсів, що забезпечує повторне використання коду, ефективне оновлення об'єктної моделі документа DOM (англ. *Document Object Model*) та зручну організацію навігації через бібліотеку React Router. Tailwind CSS [31] є утилітарним CSS-фреймворком, що пришвидшує розроблення адаптивних інтерфейсів. Фреймворк Express.js [9] забезпечує гнучку маршрутизацію HTTP-запитів та систему middleware. Довідник MySQL 8.0 [20] охоплює можливості реляційної СУБД з підтримкою транзакцій, зовнішніх ключів та розширеного синтаксису SQL.

У роботі [29] наведено особливості застосування контейнеризації в наукових розрахунках. Автори вважають, що контейнеризація ПЗ корисна для відтворення обчислювальних експериментів та для його поширення. Однак, складно запускати розподілені завдання в контейнерах, оскільки багато з них використовують MPI для міжпроцесної комунікації. У своїй роботі вони оцінюють такі рішення, як інструмент mpiexec_docker, який дає змогу запускати контейнеризовані програми аналогічно до монолітних. Він підтримує Docker та Podman як ПЗ для контейнеризації, а також OpenMPI, MVAPICH2 та Intel MPI – як реалізації MPI. Цей інструмент може працювати зі Slurm та використовувати пристрої, популярні в HPC, такі як графічні процесори GPUs або адаптери InfiniBand.

У роботі [2] наведено архітектурну основу для проєктування IoT-застосунків – від проєктування до їх розгортання. Автори вважають, що проєктування IoT-систем вимагає управління складними архітектурними проблемами, що охоплюють логіку ПЗ, конфігурацію обладнання та просторове розгортання, а також перевірки нефункціональних властивостей, таких як споживання енергії та ефективність зв'язку. Дослідники використали модельно-орієнтований підхід MDE (англ. *Model Driven Engineering*) для розроблення CAPS-платформи, яка унікально інтегрує багатовидове архітектурне моделювання, моделювання з урахуванням енергії та трафіку за допомогою CupCarbon, а також безперешкодне генерування програмного коду Arduino з високорівневих моделей проєктування. CAPS-платформа забезпечує відстежуваний та цілісний процес розроблення застосунків від архітектурного їх проєктування до фізичного розгортання.

За результатами аналізу сучасних досліджень та публікацій виявлено прогалини, згідно з якими у наявній літературі відсутні комплексні роботи, що описують методику ізольованого розгортання повноцінного вебзастосунку з клієнт-серверною архітектурою, реляційною базою даних та інтеграцією AI-сервісів у єдиному контейнеризованому середовищі з детальним порівняльним аналізом ефективності використання та тестуванням надійності.

Матеріали та методи дослідження. Для дослідження проблеми розгортання вебзастосунків у контейнеризованому середовищі потрібно використовувати комплексну методологію, яка поєднує теоретичний аналіз їх архітектур та практичні експерименти. Для розроблення системи управління заявками користувачів у сфері соціального захисту населення застосовано комплекс сучасних методів та інструментів, що забезпечують повну відтворюваність результатів дослідження та можливість їх тиражування на інші подібні системи.

Архітектура та технологічний стек. Систему управління заявками користувачів спроєктовано за методологією клієнт-серверної архітектури з дотриманням принципу розмежування відповідальностей (англ. *Separation of Concerns*). Архітектуру системи поділяють на три логічні рівні: клієнтський інтерфейс (frontend), серверна бізнес-логіка та API (backend), а також рівень збереження даних (СУБД). Клієнтську частину вебзастосунку реалізовано на основі бібліотеки React із функціональними компонентами та хуками (useState, useEffect, useContext), маршрутизацію даних здійснено через react-router-dom з захищеними маршрутами та HTTP-запитами через axios зі перехопленням JWT (англ. *JSON Web Token*). Серверну частину вебзастосунку здійснено засобами середовища виконання Node.js з фреймворком Express.js для маршрутизації даних, відкритий стандарт JWT-автентифікації та авторизації реалізовано на рівні middleware, популярну npm-бібліотека Multer використано для оброблення завантаження файлів, а інтеграцію хмарної інфраструктури OpenAI API – для AI-асистента. Зберігання даних реалізовано у MySQL 8.0 з підтримкою транзакцій та цілісності даних.

У роботі використано JSON Web Token (JWT), тобто відкритий стандарт (RFC 7519) для безпечного передавання інформації між сторонами у вигляді об'єкта JSON. Його безпосередньо застосовано для автентифікації та авторизації користувачів у веб- і мобільних застосунках та мікросервісній архітектурі. Кросплатформне середовище виконання Node.js (JavaScript) із відкритим кодом дає змогу виконувати JS-код поза веббраузером. Воно працює на основі високопродуктивного рушія V8 від Google Chrome і його використано для розроблення серверного вебзастосунку, вебслужби REST API, інструментів командного рядка та скриптів. Водночас, архітектурний стиль для створення вебслужб REST API (англ. *Representational State Transfer Application Programming Interface*) дає змогу різним програмам обмінюватися даними через мережу Інтернет за допомогою протоколу HTTP/HTTPS. Він працює за принципом клієнт-серверної взаємодії, де клієнт (наприклад, вебзастосунок) надсилає запит, а сервер повертає структуровану відповідь, найчастіше у форматі JSON або XML.

Методи контейнеризації вебзастосунків та їх розгортання. Ізольоване розгортання вебзастосунку реалізовано за допомогою технології Docker та інструменту Docker Compose. Для кожного компонента розроблено окремий Docker-образ на основі node:20-alpine для оптимізації розміру образу. Інструмент Docker Compose визначає три сервіси (frontend, backend, db) з мережевими зв'язками, змінними середовища виконання, томами для персистентного зберігання та механізмом healthcheck для координації запуску залежних компонентів. Традиційний підхід розгортання вебзастосунку реалізовано через безпосереднє встановлення Node.js, Express, React та MySQL на хост-машину без засобів ізоляції. Обидва підходи послідовно розгорталися та тестувалися на одній і тій самій апаратній конфігурації ОС для забезпечення порівнюваності результатів.

Методи тестування та вимірювання метрик. Застосовано комплексне тестування розробленого ПЗ: функціональне (28 тестових випадків розгортання вебзастосунку за модулями системи: користувачі, заявки, документи, сповіщення, адміністрування), тестування безпеки (4 випадки: автентифікація, авторизація, захист від CSRF (англ. *Cross-Site Request Forgery*, або міжсайтова підробка запиту), оброблення конфіденційних даних), тестування GUI (англ. *Graphical User Interface*) (7 випадків розгортання: навігація, адаптивність, доступність), тестування стабільності контейнеризованого розгортання вебзастосунку (5 випадків розгортання: перезапуск контейнерів, збереження даних, відновлення після збоїв). Для вимірювання тривалості розгортання вебзастосунку використовувалися системні утиліти (time, docker stats), для вимірювання відтворюваності середовища виконання проводилися повторні розгортання на різних ОС (Windows 11, Ubuntu 20.04, macOS 12).

Результати дослідження та їх обговорення / Research results and their discussion

Архітектура та проєктування системи управління заявками користувачів. Вебзастосунок для управління заявками користувачів у сфері соціального захисту населення – це цифрова платформа, яка автоматизує оброблення звернень громадян, призначення допомоги та взаємодію між заявниками та соціальними працівниками. Впровадження такої системи мінімізує паперову рутину та значно пришвидшує надання послуг. Основні модулі системи: кабінет заявника, призначений для подання заяв, завантаження документів, відстеження статусів; кабінет соцпрацівника, призначений для перевірки документів, ведення бази клієнтів, прийняття рішень; модуль аналітики, призначений для формування звітів для керівництва, ведення статистики виплат, моніторингу демографічних даних; адмін-панель, призначена для налаштування прав доступу користувачів, безпосереднього перегляду заявок та зміни станів поданих заявок.

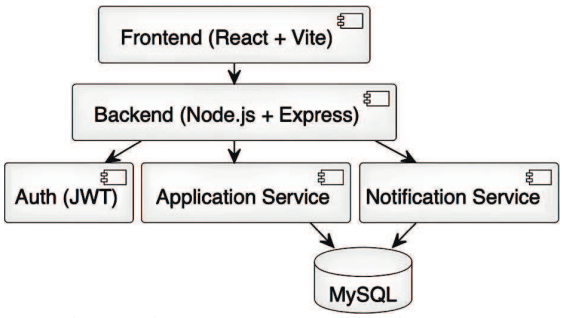

Рис. 1. Логічна архітектура системи управління заявками користувачів (компонентна діаграма) / Logical architecture of the user request management system (component diagram)

Розроблений вебзастосунок реалізовано за багаторівневою клієнт-серверною моделлю з дотриманням принципів SOLID та чистого коду. Головна мета цих принципів – зробити ПЗ гнучким, зрозумілим, простим для масштабування системи та подальшої підтримки. Логічну архітектуру системи управління заявками користувачів (компонентна діаграма) наведено на рис. 1. Кожен компонент цієї системи розгортається в окремому Docker-контейнері, що забезпечує ізоляцію середовища виконання вебзастосунку та незалежність від конфігурації цільового сервера. Взаємодія між контейнерами організовується через інструмент Docker Compose через внутрішню Docker-мережу, де сервіси спілкуються за DNS-іменами (frontend, backend, db). Тут DNS (англ. *Domain Name System*) – технологія, яка працює як "телефонна книга" мережі Інтернет, перетворюючи зрозумілі людині текстові імена сайтів (наприклад, google.com) на числові IP-адреси (наприклад, 142.250.190.46), які використовують комп'ютери для зв'язку між собою.

Об'єктна модель системи управління заявками користувачів (Service Desk, CRM або Issue Tracking системи) описує структуру даних, сутності, їхні атрибути, методи та взаємозв'язки, що потрібні для автоматизації оброблення звернень. Для опису об'єктної моделі системи управління заявками користувачів розроблено діаграму класів (рис. 2), яка відображає основні сутності предметної області, їх атрибути та зв'язки. Основними класами діаграми є User (ідентифікатор, email, пароль, роль, ПІБ, телефон), Application (назва, опис, статус, категорія, пріоритет, коментар), Notification (текст, прочитано), ApplicationStatusHistory (історія змін), ApplicationDocument (файли), AdminLog (операції адміністратора). Зв'язки 1: М між класами забезпечують повний цикл управління заявками користувачів.

Проєктування бази даних системи управління заявками користувачів. База даних системи управління заявками (англ. *Helpdesk/Service Desk*) – це структуроване сховище для реєстрації, оброблення та відстеження звернень клієнтів або співробітників. Вона об'єднує інформацію про запити, виконавців і клієнтів, забезпечуючи повний життєвий цикл кожного завдання.

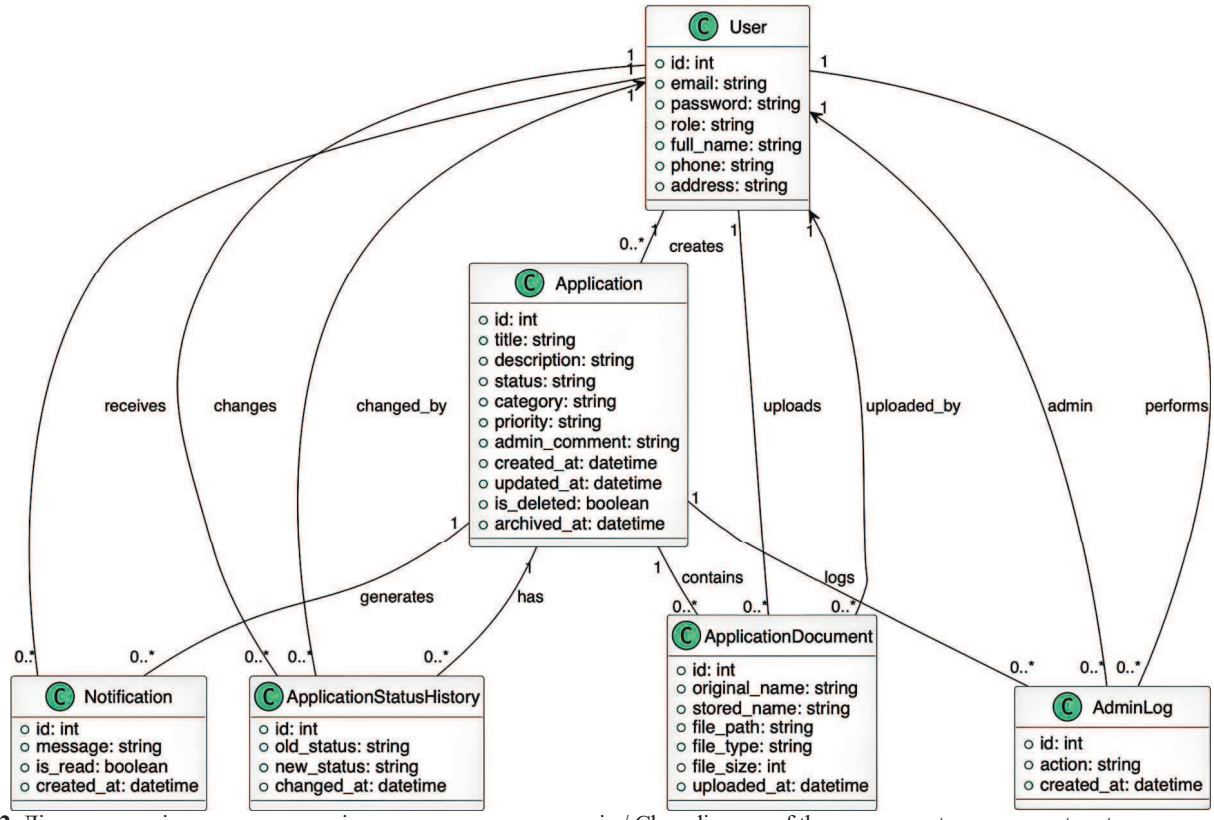

Рис. 2. Діаграма класів системи управління заявками користувачів / Class diagram of the user request management system

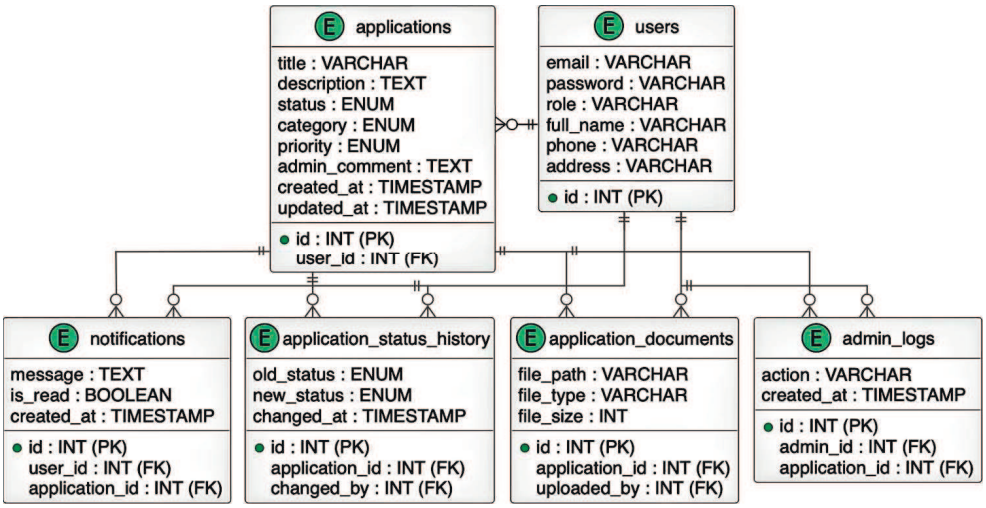

Рис. 3. ER-діаграма бази даних системи управління заявками користувачів / ER diagram of the user request management system database

Базу даних системи управління заявками користувачів реалізовано в СУБД MySQL 8.0 відповідно до бізнес-середовища виконання типу OLTP (англ. *On-line Transaction Processing*) з інтенсивними операціями CRUD – акронім, що означає чотири базові функції для роботи з даними: Create (створення), Read (читання), Update (оновлення) та Delete (видалення). Операції CRUD є фундаментом для проєктування більшості вебзастосунків, інтерфейсів та систем керування базами даних. Інформаційну модель бази даних подано ER-діаграмою (рис. 3) з шістьма таблицями та зовнішніми ключами. База даних системи управління заявками побудована навколо головної сутності – Заявка (Ticket). Вона об'єднує користувачів, категорії, статуси та історію обговорення попередніх заявок. Таблиця users зберігає 9 атрибутів користувача, applications – 10 атрибу-

тів заявки, notifications – 4 атрибути сповіщення та ін. Встановлено каскадне оновлення та видалення для забезпечення цілісності даних.

Поведінкові моделі системи управління заявками користувачів. Поведінкові моделі системи управління заявками (Help Desk, Service Desk, CRM) описують динаміку взаємодії потенційних користувачів, операторів та самої системи у процесі оброблення запитів. На відміну від структурних моделей, які показують будову системи, поведінкові моделі фокусуються на зміні станів, на потоках даних, на алгоритмах автоматизації та сценаріях реагування на події. Ці моделі будуються за допомогою стандартних нотацій (UML, BPMN) і охоплюють декілька ключових аспектів. Для подання динамічних особливостей моделі системи управління заявками користувачів розроблено діаграму послідовності (рис. 4), що показує взаємодію під час створення заявки: заповнення форми → POST /api/applications → перевірка JWT → INSERT applications → INSERT notifications → 201 Created.

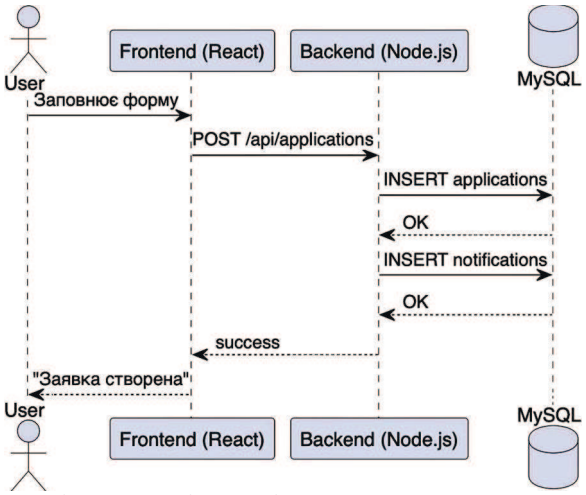

Рис. 4. Діаграма послідовності процесу створення заявки користувачем / Sequence diagram of the user request creation process

Діаграма станів (англ. *State Machine Diagram*) для сутності "Заявка користувача" описує повний життєвий цикл заявки від моменту її створення до закриття або скасування. Діаграма станів заявки користувача (рис. 5) показує цикл життя заявки зі зміною статусів: new → in_progress → approved/rejected. Кожна зміна фіксується у application_status_history та генерує сповіщення.

Рис. 5. Діаграма станів заявки користувача у системі управління заявками / State diagram of a user request in a request management system

Діаграма діяльності (англ. *Activity Diagram*) відображає послідовність дій та потік керування у системі управління заявками від початкового стану до кінцевого. Діаграма діяльності (рис. 6) описує основні бізнес-процеси: автентифікація, подання заявки, адміністрування.

Рис. 6. Діаграма діяльності основних процесів у системі управління заявками користувачів / Activity diagram of the main processes in a user request management system

Проєктування інтерфейсу користувача. Графічний інтерфейс користувача (GUI) системи управління заявками (Helpdesk, Service Desk або тикет-системи) – це візуальне середовище, яке забезпечує взаємодію між заявниками (клієнтами або працівниками) та операторами технічної підтримки. Головна мета такого інтерфейсу – зробити процес подання, оброблення та відстеження звернень користувачів максимально швидким, інтуїтивно зрозумілим та структурованим. Оскільки система управління має різні типи користувачів, її графічний інтерфейс зазвичай розділений на дві основні ролі (панелі):

- інтерфейс клієнта (портал самообслуговування) розроблено для звичайних користувачів, яким потрібно надіслати запит або знайти відповідь на питання самостійно;
- панель адміністратора та аналітика призначена для керівників підтримки прийняття рішень та системних адміністраторів для контролю та налаштування виробничих процесів.

Графічний інтерфейс системи управління заявками користувачів спроєктовано для двох ролей: звичайного користувача та адміністратора. На рис. 7 подано макет (wireframe) адміністративної панелі з табличним переглядом заявок, їх фільтрацією за статусом, пошуком та можливістю зміни статусу.

Реалізація контейнеризованого розгортання вебзастосунку. Контейнеризоване розгортання вебзастосунку – це сучасний стандарт деплою, за якого код програми, її оточення, системні бібліотеки та конфігурації упаковуються в один ізольований контейнер. Це гарантує, що застосунок працюватиме абсолютно однаково на комп'ютері розробника, тестовому сервері та у продакшені.

Контейнеризоване розгортання вебзастосунку для управління заявками користувачів реалізовано трьома Docker-сервісами: frontend (React/Vite:5173), backend (Node.js /Express:5001), db (MySQL 8.0:3307). Сервіс mySQL використовує healthcheck, що гарантує запуск backend тільки після реальної готовності СУБД. На рис. 8 показано результат успішного запуску: усі три контейнери запущені, мережа створена, томи під'єднані.

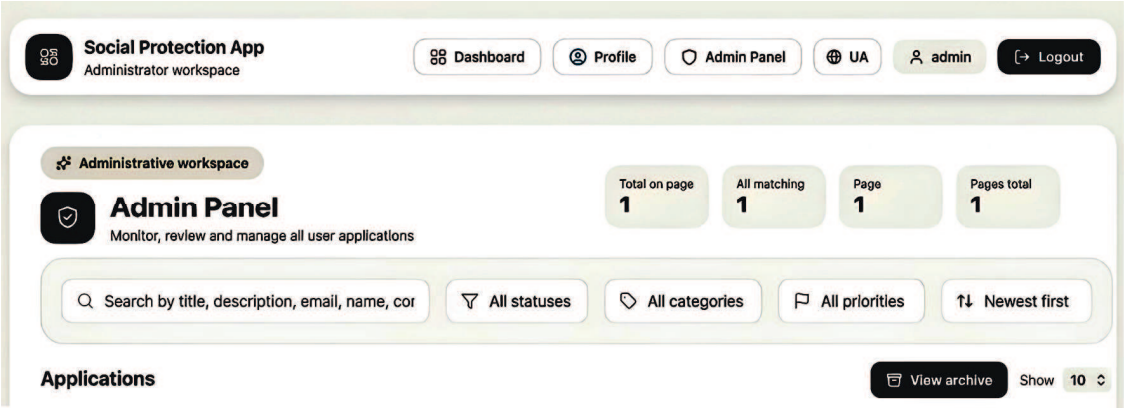

Рис. 7. Макет панелі адміністратора / Wireframe of the administrator panel

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

nastya@Nastyas-MacBook-Pro social-protection-app % docker compose up --build
[+] up 5/5
 ✓ Image social-protection-app-backend  Built
 ✓ Image social-protection-app-frontend Built
 ✓ Container social_app_mysql           Running
 ✓ Container social_app_backend         Recreated
 ✓ Container social_app_frontend        Recreated
Attaching to social_app_backend, social_app_frontend, social_app_mysql
Container social_app_mysql Waiting
Container social_app_mysql Healthy
social_app_backend  | ◇ injected env (0) from .env // tip: ⌘ enable debugging { debug: true }
social_app_backend  | ◇ injected env (0) from .env // tip: ⌘ auth for agents [www.vestauth.com]
social_app_backend  | Server running on port 5001
social_app_backend  | MySQL connected
social_app_frontend |
social_app_frontend  | > frontend@0.0.0 dev
social_app_frontend  | > vite --host
social_app_frontend |
social_app_frontend |
social_app_frontend  |   VITE v8.0.8  ready in 121 ms
social_app_frontend |
social_app_frontend  |   ➜  Local:   http://localhost:5173/
social_app_frontend  |   ➜  Network: http://172.18.0.3:5173/
```

Рис. 8. Запуск системи управління заявками користувачів за допомогою команди docker compose up / Launching the user request management system with the docker compose up command

Інтерфейс системи управління заявками користувачів та результати його тестування. Інтерфейс системи управління заявками (англ. *Helpdesk/Service Desk*) – це візуальний та інтерактивний простір, який забезпечує взаємодію між клієнтами (що створюють запити) та операторами або технічною підтримкою (що їх обробляють). Результати тестування інтерфейсу (UI/UX) системи управління заявками демонструють загальну готовність системи до експлуатації, визначають критичні помилки відображення та вказують на зони для покращення взаємодії з користувачем.

Реалізований інтерфейс системи управління заявками користувачів відображено на рис. 9-12. Головна сторінка користувача (Dashboard) містить узагальнену статистику власних заявок, прямий доступ до форми створення нової заявки та перелік поданих звернень із статусами у вигляді кольорових міток (new=синій, in_progress=жовтий, approved=зелений, rejected=червоний). Головна сторінка особистого кабінету користувача – це його власний захищений простір на сайті або в додатку, де зібрана вся найважливіша інформація про його акаунт, послуги та замовлення. Вона дає змогу швидко перевірити наявний баланс, змінити налаштування системи або зв'язатися з адміністратором для надання підтримки без пошуку по всьому сайту.

Адміністративна панель (рис. 10) надає адміністратору табличний перегляд усіх заявок системи управління заявками з можливістю їх фільтрації за статусом/категорією, пошуку за назвою, сортування за датою подання та пріоритетом виконання, інтерактивної зміни статусу замовлення та додавання коментарів без перезавантаження сторінки.

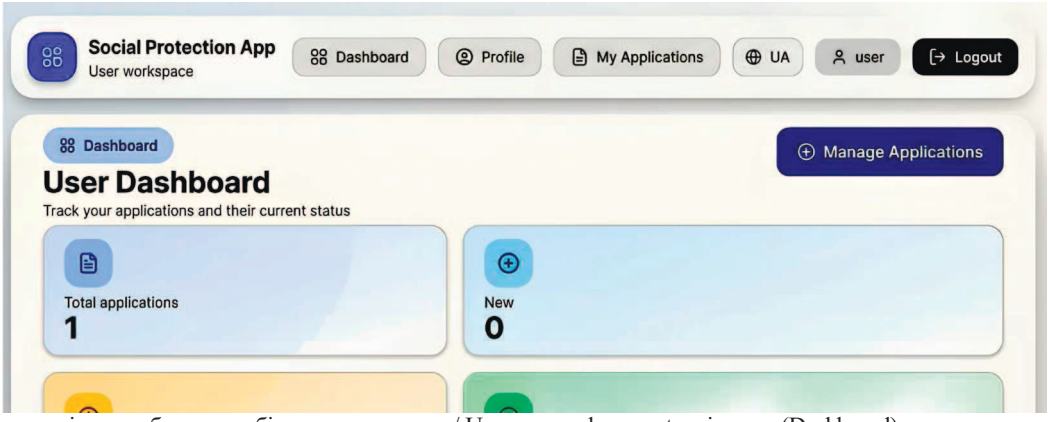

Рис. 9. Головна сторінка особистого кабінету користувача / User personal account main page (Dashboard)

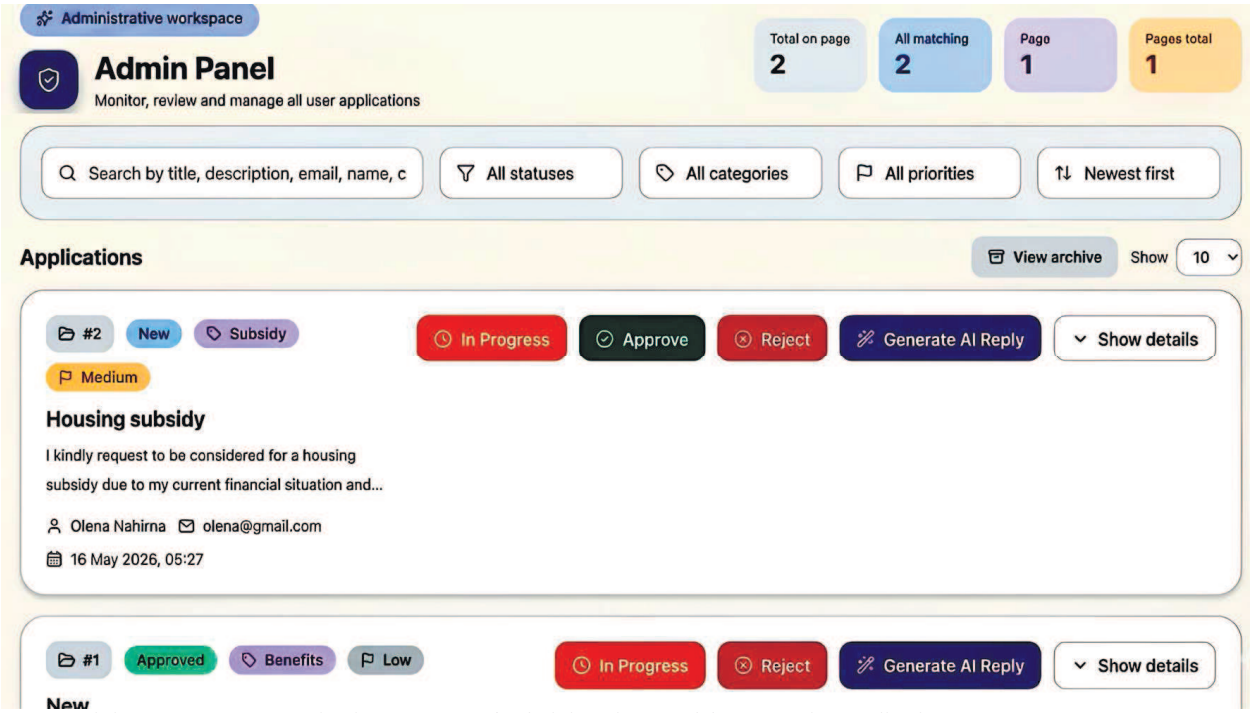

Рис. 10. Адміністративна панель оброблення заявок / Administrative panel for processing applications

Сторінка деталей заявки (рис. 11) відображає повну інформацію про заявку, враховуючи прикріплені документи, коментар адміністратора та повну хронологію змін статусу.

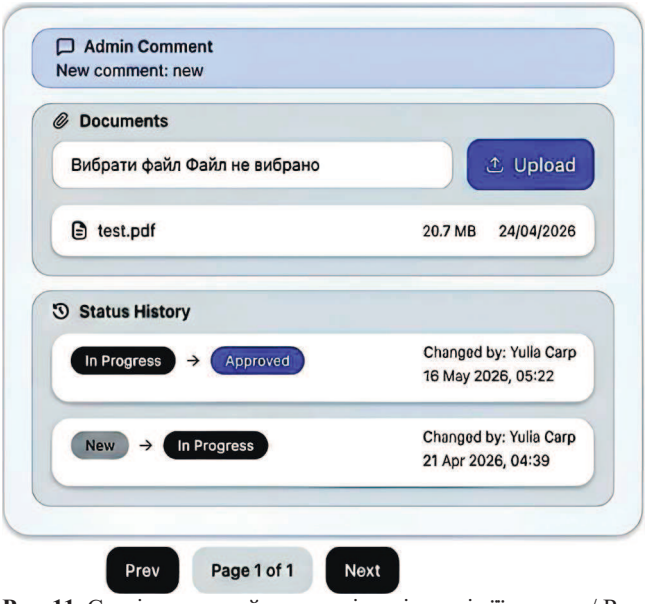

Рис. 11. Сторінка деталей заявки з історією змін її статусу / Request details page with the history of status changes

Сторінка сповіщень (рис. 12) відображає повний перелік повідомлень для користувача про зміни статусів заявок та дії адміністратора.

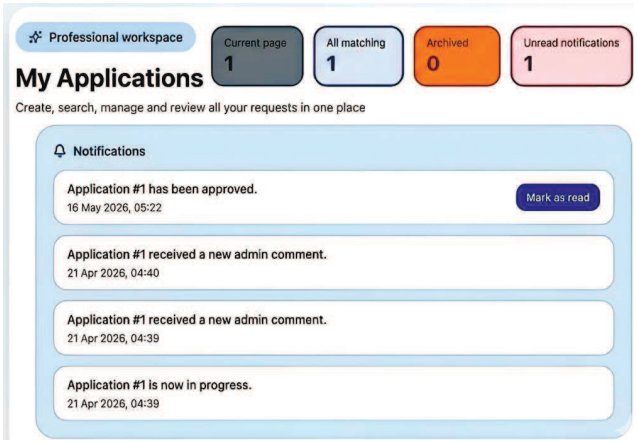

Рис. 12. Модуль сповіщень користувача / User notification module

Порівняльний аналіз підходів до розгортання вебзастосунку. Проведено порівняльний аналіз підходів до розгортання вебзастосунку за сімома критеріями (табл. 1). Оскільки єдиного ідеального способу для цього не існує, вибір залежить від розміру проєкту, бюджету та навичок розробника. Тривалість розгортання вебзастосунку вимірювалась від моменту видачі команди до повної готовності системи управління заявками користувачів. Традиційний підхід до розгортання вебзастосунку вимагав ручного встановлення Node.js, npm install, MySQL, налаштування.env файлів, запуску обох сервісів – усього 20-40 хв ручного налаштування. Інструмент Docker Compose розгортає систему за 5-10 хв за одну команду.

Табл. 1. Порівняльний аналіз підходів до розгортання вебзастосунку / Comparative analysis of approaches to web application deployment

Критерій розгортання вебзастосунку	Традиційний підхід до розгортання	Контейнеризація вебзастосунку (Docker)
Тривалість першого розгортання вебзастосунку	20-40 хв (ручне налаштування)	5-10 хв (docker compose up)
Відтворюваність середовища виконання	Низька (залежить від ОС, версій)	Висока (однакові образи на всіх платформах)
Складність налаштування	Висока (ручні дії для кожного компонента)	Низька (один конфіг-файл YAML)
Ізоляція компонентів	Відсутня (змагання за ресурси)	Повна (процеси ізольовані в контейнерах)
Масштабованість	Складна (копіювання вручну)	Проста (docker compose scale)
Управління залежностями	Ручне, конфлікти версій можливі	Автоматичне всередині образу
Перенесення на новий сервер	Складне (повторна конфігурація системи)	Просте (образи портативні, одна команда)

Результати (табл. 2) показують, що контейнеризований підхід до розгортання вебзастосунку істотно перевершує традиційний підхід за переважною більшістю критеріїв. Найбільш значущими перевагами є відтворюваність середовища виконання (критична для DevOps-практики), швидкість розгортання вебзастосунку (скорочення у 4-8 разів) та простота масштабування системи.

Табл. 2. Результати комплексного тестування технологій розгортання вебзастосунку / Results of comprehensive testing of web application deployment technologies

Категорія тестування	Кількість тестів	Пройдено *
Функціональне	28	28
Безпека	4	4
Графічний інтерфейс	7	7
Контейнеризація & надійність	5	5
Разом	44	44

Примітка: *невдачі – 0 %, покриття – 100 %.

Отримані результати тестування технологій розгортання вебзастосунку засвідчують ефективність запропонованого підходу до контейнеризованого розгортання: 100-відсоткове проходження 44 тестових випадків підтверджує функціональну повноту та надійність системи, скорочення тривалості розгортання у 4-8 разів порівняно з традиційним підходом демонструє його практичну перевагу, а стабільна відтворюваність на різних операційних системах (Windows, Linux, macOS) доводить переваги контейнеризованого підходу над традиційним монолітним.

Отже, контейнеризація застосунків є однією з ключових парадигм DevOps-практики та сучасного веброзроблення. Наше дослідження порівняло два підходи до розгортання вебзастосунку на конкретній системі управління заявками користувачів та отримало результати, що добре узгоджуються з теоретичними прогнозами та опублікованими дослідженнями.

Скорочення тривалості розгортання вебзастосунку від 20-40 до 5-10 хв підтверджує заяви багатьох практиків про пришвидшення циклу його розроблення. Досягнута висока відтворюваність середовища виконання (однакова поведінка на Windows, Linux, macOS без будь-яких змін) вирішує давню проблему "works on my machine" (WoMM), коли програма працює у розробника, але є проблемною на сервері. Ізоляція компонентів у Docker дає змогу розробляти, тестувати та розгортати frontend, backend та БД незалежно. Це особливо важливо для великих проєктів з розподіленими командами.

Інтеграція AI-асистента (OpenAI API) продемонструвала, як контейнеризовані системи управління заявками користувачів легко розширюються новими функціональностями: достатньо додати http-клієнта та змінити конфігурацію середовища виконання без зміни структури контейнерів. Обмеження дослідження: розглянуто невеликий застосунок (3 контейнери); для великих мікросервісних систем (50+ контейнерів) потрібно розглядати платформу Kubernetes замість інструменту Docker Compose.

Обговорення результатів дослідження. Ізольоване розгортання вебзастосунків у контейнерах – це стандарт сучасного розроблення ПЗ, який відокремлює код і його залежності від фізичного сервера. Головними проблемами розгортання вебзастосунків у контейнерах є вразливості безпеки спільного ядра, складність керування станом (Stateful даних), надлишкове споживання ресурсів через великі образи та високий поріг входження в інструменти оркестрації. Ізольоване розгортання вебзастосунку у хмарі – це архітектурний підхід, який захищає програму від зовнішніх загроз та інших користувачів хмари через створення виділеного закритого середовища. Ізольоване розгортання (Air-gapped deployment) передбачає запуск застосунку у хмарі без доступу до мережі Інтернет чи публічних мереж. Основна проблема такого підходу – критична складність оновлення залежностей та доставлення образів до закритого середовища, що вимагає ручного перенесення всіх даних через захищені шлюзи.

У роботі [13] охарактеризовано адаптивну контейнеризацію для мікросервісів у розподілених хмарних системах. Автори вважають, що традиційна монолітна локальна модель розгортання застосунків швидко замінюється парадигмою хмарних мікросервісів, що частково зумовлено зростанням кількості постачальників хмарної інфраструктури, що забезпечують безперешкодний доступ до різноманітного обчислювального обладнання, а також необхідністю в застосунках обслуговувати постійно зростаючу аудиторію, що вимагає масштабованості. Хоча віртуалізація на основі контейнерів була кращим методом розгортання мікросервісів, споживачі хмарних послуг досі не мали значних можливостей для оптимізації витрат і ресурсів. Задля цього в роботі наведено структуру розподілу ресурсів для контейнерного розгортання мікросервісів, яку називають "Адаптивна контейнеризація для мікросервісів у розподілених хмарних системах" (англ. *Adaptive Containerization for Microservices in Distributed Cloud Systems*), що допомагає знизити експлуатаційні витрати, забезпечуючи при цьому мінімальний гарантований рівень обслуговування.

У роботі [28] наведено результати керування мультихмарними розгортаннями на платформі Kubernetes за допомогою Istio, Prometheus та Grafana. Автори стверджують, що наразі хмарно-орієнтований підхід до розгортання застосунків змінює спосіб їх створення та розгортання. Мультихмарність означає використання багатьох хмарних постачальників і послуг в єдиній хмарній мережевій інфраструктурі. Мультихмарні середовища виконання застосунків використовують всі переваги від різних постачальників, такі як розподіл обчислювальних ресурсів, мінімальна тривалість простою та висока доступність даних.

У роботі [24] наведено результати контейнеризації програм та особливості її реалізації у хмарі PaaS. Автор вважає, що контейнеризацію програм широко обговорюють як легке рішення для віртуалізації застосунків. Окрім переваг над традиційними віртуальними машинами у хмарі, контейнери особливо актуальні для хмар типу "платформа як послуга" (PaaS) для керування та оркестрації застосунків за допомогою контейнерів як механізму їх пакування. У своїй роботі дослідник розглядає вимоги, що виникають через необхідність сприяння роботі застосунків засобами розподілених мультихмарних платформ.

У роботі [1] наведено результати оптимізованого розгортання та моніторингу хмарних застосунків на AWS (англ. *Amazon Web Services*) за допомогою платформи Kubernetes та Prometheus та Grafana. Автори вважають, що керування журналами та моніторинг ресурсів для кожного розгортання застосунків може призвести до ще більших витрат. Щоб вирішити ці проблеми,

вони пропонують впровадити стратегію автоматизації для розгортання застосунків у хмарі. Завдяки автоматизації процесу розгортання застосунків можна впровадити централізовану систему ведення журналів для збирання та керування логами з різних розгортань в єдиному середовищі. Це забезпечує централізований аналіз усіх журналів і цим самим спрощує виявлення та усунення несправностей.

У роботі [3] наведено особливості вивчення технології контейнеризації, такі як Docker та Kubernetes, та їхню роль у сучасних хмарних розгортаннях. Автори вважають, що поява хмарних обчислень змінила ландшафт розгортання та управління застосунками, для якого характерні безпрецедентна гнучкість, масштабованість та спритність. У цьому контексті технології контейнеризації, прикладом яких є Docker, та платформи оркестрації контейнерів, втіленими Kubernetes, стали ключовими факторами сучасного хмарного розгортання застосунків. Ці технології обіцяють ефективне пакування застосунків, безперешкодну масштабованість та спрощені практики DevOps.

У роботі [17] охарактеризовано поняття монолітності до контейнеризації на основі хмарних обчислень з використанням служби еластичних контейнерів для філогенетичного аналізу. Автори вважають, що значне зростання в галузі біотехнологій та нові винаходи поставили багато викликів перед технологічною галуззю, яка відтворює модель попиту та пропозиції. Побудова такої монолітної моделі має когнітивну складність кодової бази, а її масштабованість є основним недоліком, який можна вирішити за допомогою технології контейнеризації застосунків. У цьому підході розподілені застосунки можуть бути розгорнуті та виконані шляхом спільного використання ресурсів віртуальної або фізичної машини з іншими контейнеризованими застосунками. Автори пропонують модель контейнеризації застосунку на основі хмарних обчислень для філогенетичного аналізу насамперед у публічній хмарі AWS (англ. *Amazon Web Services*) за допомогою служби еластичних контейнерів.

У роботі [11] охарактеризовано портативний фреймворк із узагальненими функціями середовища виконання для виконання графів завдань та одночасного розгортання кількох програм на гетерогенних системах. Автори вважають, що гетерогенні обчислювальні платформи широко використовують для задоволення різноманітних вимог до продуктивності сучасних програм, але вони створюють ключову проблему балансування програмованості з портативністю продуктивності. У роботі наведено уніфіковану та портативну платформу, яка вирішує цей компроміс шляхом інтеграції системи виконання CEDR з моделлю програмування графів завдань Taskflow. Запропонована система спрощує розроблення програм, забезпечуючи ефективне виконання на різних графічних процесорах та FPGA. Вони демонструють її портативність на широкому діапазоні гетерогенних систем SoC та HPC. Платформа підтримує одночасне виконання кількох програм через централізований рівень координації, долаючи обмеження ізольованих контекстів виконання та досягаючи швидкості до 1,23.× пришвидшення на дуже гетерогенних платформах.

У роботі [35] проаналізовано адаптивний багатоцільовий ройовий інтелект для розгортання контейнерних мікросервісів. Автори зазначають, що архітектура мікросервісів на основі контейнерів є важливою для сучасних застосунків. У роботі запропоновано формалізовану модель розгортання контейнерних мікросервісів із систематичним аналізом взаємозалежностей у ланцюгах функцій сервісів SFC (англ. *Service Function Chains*). Для досягнення цієї мети автори розробили новий алгоритм оптимізації ройового інтелекту – багатоцільову оптимізацію рою Sand Cat із гібридними стратегіями (MSCSO-HS), призначений для оптимізації розгортання мікросервісів. Запропонований алгоритм ефективно мінімізує витрати на зв'язок між мікросервісами та підвищує щільність агрегації контейнерів, що сприяє підвищенню надійності роботи застосунків і максимальному використанню ресурсів.

У роботі [6] проаналізовано підхід до управління IoT-застосунками для універсального контейнерного шлюзу з використанням вебсервісів. Автори вважають, що різноманітність комунікаційних стандартів та протоколів, що використовуються для обміну даними між IoT-пристроями, збільшила гетерогенність системи. Ця складність вимагає використання шлюзів та проміжного ПЗ для забезпечення сумісності між різними пристроями та системами. Хоча ці підходи покращили оброблення гетерогенності, проблеми, пов'язані з динамічним управлінням, масштабованістю та оркестрацією застосунків, залишаються. Вони пропонують підхід до управління вебсервісами для динамічного, контейнерного шлюзу IoT на периферії, який підтримує багатопротокольні та багатозастосункові середовища з використанням недорогого апаратного забезпечення та програмних рішень з відкритим кодом. Запропонована архітектура шлюзу використовує контейнери для ізоляції окремих астосунків і функцій, забезпечуючи безпеку та захист виконання. Вона використовує Message Queue Telemetry Transport (MQTT) для міжкомпонентного зв'язку та використовує JavaScript Object Notation (JSON) як модель застосунку.

Обговорення результатів дослідження підтвердило доцільність проведення саме цієї роботи. Аналіз наявної літератури засвідчив, що попередні дослідження розглядали або окремі аспекти контейнеризації (теоретичні засади технології Docker, оркестрацію великомасштабних систем Kubernetes), або розгортання без детального кількісного порівняльного аналізу. Жодна з проаналізованих робіт не охоплювала в єдиному комплексі: повну контейнеризацію системи клієнт-серверної архітектури з реляційною базою даних, інтеграцію AI-сервісів та систематичне тестування 44 сценаріїв з верифікацією на трьох операційних системах – що й визначає наукову новизну та практичну цінність цього дослідження.

Отже, внаслідок виконаної роботи можна сформулювати такі наукову новизну та практичну значущість результатів дослідження.

Наукова новизна отриманих результатів дослідження – отримала подальший розвиток методика ізольованого розгортання вебзастосунків у контейнеризованому середовищі, яка, на відміну від наявних підходів, поєднує повну контейнеризацію всіх компонентів клієнт-серверної системи управління заявками користувачів з автоматизованою ініціалізацією бази даних, механізмом healthcheck для синхронізації сервісів та інтеграцією AI-сервісів для інтелектуального аналізу заявок, забезпечуючи цим самим відтворюване ізольоване розгортання вебзастосунку в єдиній Docker Compose-конфігурації з гарантією однаковості на будь-якій платформі.

Практична значущість результатів дослідження – розроблений функціональний вебзастосунок є готовим рішенням для його використання органами місцевого самоврядування, соціальними службами та установами, що здійснюють оброблення заявок від населення, який легко розгорнути однією командою docker compose up на будь-якому сервері з Docker Engine (Windows, Linux, macOS) без додаткових ручних налаштувань, що скорочує тривалість впровадження застосунку з місяців на дні.

Висновки / Conclusions

Розроблено ПЗ для ізольованого розгортання вебзастосунку у контейнеризованому середовищі, яке дало змогу провести порівняльний аналіз ефективності різних підходів до його розгортання та обґрунтувати переваги контейнеризації застосунків над традиційним ручним налагодженням його роботи. За результатами проведеного дослідження можна зробити такі основні висновки.

1. Аналіз підходів до розгортання вебзастосунку показав, що для традиційного підходу до розгортання вебзастосунків (традиційний деплой) характерні складність (ручне налаштування), низька відтворюваність середовища виконання (залежність від конфігурації серверу) та повільність реалізації (20-40 хв). Контейнеризація вебзастосунку забезпечує стандартизоване ізольоване середовище виконання, незалежне від платформи, що робить такий підхід оптимальним для систем, які потрібно розгортати на декількох серверах й розвивати різними командами.
2. Розроблено функціональний вебзастосунок для управління заявками користувачів у сфері соціального захисту населення з клієнт-серверною архітектурою (React + Node.js + MySQL), що реалізує: реєстрацію та автентифікацію користувачів за відкритим стандартом JWT, повний CRUD-цикл заявок, систему сповіщень, журналювання дій, завантаження документів та AI-асистента для класифікації звернень.
3. Реалізовано контейнеризоване розгортання системи управління заявками користувачів у вигляді трьох Docker-контейнерів з Docker Compose-оркестрацією. Ключові особливості: healthcheck MySQL гарантує коректний порядок запуску, іменовані томи забезпечують персистентність даних між перезапусками, env-файли безпечно передають конфіденційні параметри, внутрішня Docker-мережа забезпечує DNS-маршрутизацію між сервісами.
4. Порівняльний аналіз ефективності різних підходів до розгортання вебзастосунку показав переваги контейнеризованого підходу: тривалість розгортання вебзастосунку скорочується у 4-8 разів, відтворюваність середовища виконання підвищується від низької до максимальної, ізоляція компонентів усуває конфлікти залежностей, масштабування системи спрощується з ручного на декларативне. Комплексне тестування системи (44 випадки) підтвердило її функціональну коректність та надійність роботи.
5. Запропоновано напрями подальших етапів дослідження: міграція на платформу Kubernetes для горизонтального масштабування системи; імплементація CI/CD-конвеєра з автоматичною збіркою та розгортанням образів; додавання галузевого стандарту Prometheus+Grafana для моніторингу контейнерів; розширення функціоналу системи управління заявками користувачів за рахунок відеоверифікації та е-підпису документів.

Запропоноване рішення є практичним підтвердженням ефективності контейнеризованого підходу до розгортання вебзастосунків та демонструє доцільність використання Docker-технологій в органах місцевого самоврядування, соціальних службах та інших установах, що потребують надійних, масштабованих і простих в адмініструванні інформаційних систем.

References

1. Abirami, T., Mapari, S., Jayadharshini, P., Krishnasamy, L., & Vigneshwaran, R. R. (2023). Streamlined Deployment and Monitoring of Cloud-Native Applications on AWS with Kubernetes Prometheus Grafana. 2023 *International Conference on Advances in Computation, Communication and Information Technology* (ICAICCIT), Faridabad, India, pp. 1149–1155. https://doi.org/10.1109/ICAICCIT60255.2023.10465818
2. Abughazala, M., Sharaf, M., Abusair, M., & Muccini, H. (2026, April). An architecture framework for architecting IoT applications: From design to deployment. *Journal of Systems and Software*, vol. 234, article ID 112728. https://doi.org/10.1016/j.jss.2025.112728
3. Agrawal, S., & Singh, D. (2024). Study Containerization Technologies like Docker and Kubernetes and their Role in Modern Cloud Deployments. 2024 IEEE 9th *International Conference for Convergence in Technology* (I2CT), Pune, India, pp. 1–5. https://doi.org/10.1109/I2CT61223.2024.10543986
4. Bucko, A., Vishi, K., Krasniqi, B., & Rexha, B. (2023). Enhancing JWT Authentication and Authorization in Web Applications Based on User Behavior History. *Computers*, *12*(4), article number 78. https://doi.org/10.3390/computers12040078
5. Dalimunthe, S., Putra, E. H., & Ridha, M. A. F. (2023). Restful API Security Using JSON Web Token (JWT) With HMAC-Sha512 Algorithm in Session Management. *IT Journal Research and Development*, *8*(1), 81–94. https://doi.org/10.25299/itjrd.2023.12029
6. Dauda, A., Flauzac, O., & Nolot, F. (2025). IoT Applications Management Approach for Universal Container-Based Gateway Using Webservices. 2025 12th *International Conference on Wireless Networks and Mobile Communications* (WINCOM), Riyadh, Saudi Arabia, pp. 1–6. https://doi.org/10.1109/WINCOM65874.2025.11313420
7. Docker Compose Overview: official documentation Docker. URL: https://docs.docker.com/compose/
8. Docker Overview: official documentation Docker. URL: https://docs.docker.com/get-started/overview/
9. Express.js Documentation: official documentation Express. URL: https://expressjs.com/
10. Galhotra, P., Singh, K. A., Hasan, Z., & Safa, M. (2025). Application Deployment: The Game-Changing Impact of Containerization. 2025 *Fourth International Conference on Power, Control and Computing Technologies* (ICPC2T), Raipur, India, pp. 1–7. https://doi.org/10.1109/ICPC2T63847.2025.10958660
11. Gener, S., Hassan, S., Umut Suluhan, H., Chang, L., Chakrabarti, C., Huang, T.-W., Ogras, U., & Akoglu, A. (2026, March). A portable framework with generalized runtime features for task graph execution and concurrent multi-application deployment on heterogeneous systems. *Future Generation Computer Systems*, vol. 176, article ID 108184. https://doi.org/10.1016/j.future.2025.108184
12. Grytsiuk, P. Y., Ivanyshyn, A. V., & Hrytsiuk, Y. I. (2023). Quality assurance of software products in accordance with IEEE 730-2014 standard within the project implementation lifecycle. *Scientific Bulletin of UNFU,* 33(2), 101–117. https://doi.org/10.36930/40330214
13. Keni, N. D., & Kak, A. (2020). Adaptive Containerization for Microservices in Distributed Cloud Systems. 2020 IEEE 17th *Annual Consumer Communications & Networking Conference* (CCNC), Las Vegas, NV, USA, pp. 1–6. https://doi.org/10.1109/CCNC46108.2020.9045634
14. Kim, G., Humble, J., Debois, P., & Willis, J. (2021). The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. 2nd ed. Portland: IT Revolution Press, 528 p. URL: https://www.yakaboo.ua/ua/the-devops-handbook-how-to-create-world-class-agility-reliability-security-in-technology-organizations.html?srsltid=AfmBOop9IswLaFGHFT3uCkAL6xeNU5wHEJ6OOojq

15. Korbylo, R. Y., & Hrytsiuk, Y. I. (2025). System for assessing the psychological state of military personnel using natural language processing methods. *Scientific Bulletin of UNFU,* 35(6), 137–153. https://doi.org/10.36930/40350617
16. Kubernetes Documentation: official documentation Kubernetes. URL: https://kubernetes.io/docs/home/
17. Kumudavalli, M. V., & Venkatesh, G. (2021). Cloud Computing based Monolithic to Containerization using Elastic Container Service for Phylogenetic Analysis. 2021 *Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks* (ICICV), Tirunelveli, India, pp. 1540–1543. https://doi.org/10.1109/ICICV50876.2021.9388395
18. Mabotha, E., Mabunda, N. E., & Ali, A. (2025). A Dockerized Approach to Dynamic Endpoint Management for RESTful Application Programming Interfaces in Internet of Things Ecosystems. *Sensors*, *25*(10), article ID 2993. https://doi.org/10.3390/s25102993
19. Matkivskyi, R. M., & Hrytsiuk, Y. I. (2025). Book recommendation system using artificial intelligence methods and tools. *Scientific Bulletin of UNFU*, 35(6), 84–95. https://doi.org/10.36930/40350610
20. MySQL 8.0 Reference Manual: official documentation MySQL. URL: https://dev.mysql.com/doc/refman/8.0/en/
21. Nair, A. M., Ck, S., S, S., Kk, V., & Joy, J. (2024). Dockerized Application with Web Interface. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, *10*(2), 412–419. https://doi.org/10.32628/CSEIT243646
22. Narasimhulu, M., Mounika, D. V., Varshini, P., Amarendra, K., & Rao, T. R. K. (2023). Investigating the Impact of Containerization on the Deployment Process in DevOps. 2023 2nd *International Conference on Edge Computing and Applications* (ICECAA), Namakkal, India, pp. 679–685. https://doi.org/10.1109/ICECAA58104.2023.10212240
23. Oleksyuk, V. V. (2019). Fundamentals of Cloud Technologies: A Teaching Guide. Kyiv: UMO Publishing House, 120 p. URL: https://umo.edu.ua/images/content/depozitar/posibnyky/navchalyni/7_Олексюк__Основи.pdf
24. Pahl, C. (2015, May-June). Containerization and the PaaS Cloud. In IEEE *Cloud Computing*, vol. 2, no. 3, pp. 24–31. https://doi.org/10.1109/MCC.2015.51
25. React Documentation: official documentation React. URL: https://react.dev/
26. Rubel, Y., & Hrytsiuk, Y. (2026). Development of a hybrid artillery fire control system based on neural networks and uncertainty quantification methods. *Technology Audit and Production Reserves*, 2(2(88), 98–105. https://doi.org/10.15587/2706-5448.2026.357488
27. Shahriar, H., Qian, K., & Zhang, H. (2020). Learning Environment Containerization of Machine Leaning for Cybersecurity. 2020 IEEE 44th *Annual Computers, Software, and Applications Conference* (COMPSAC), Madrid, Spain, pp. 1131–1132. https://doi.org/10.1109/COMPSAC48688.2020.0-105
28. Sharma, V. (2022). Managing Multi-Cloud Deployments on Kubernetes with Istio, Prometheus and Grafana. 2022 8th *International Conference on Advanced Computing and Communication Systems* (ICACCS), Coimbatore, India, pp. 525–529. https://doi.org/10.1109/ICACCS54159.2022.9785124
29. Sheka, A., Bersenev, A., & Samun, V. (2019). Containerization in Scientific Calculations. 2019 *International Multi-Conference on Engineering, Computer and Information Sciences* (SIBIRCON), pp. 0793–0798. https://doi.org/10.1109/SIBIRCON48586.2019.8958324
30. Šimec, A., Držanić, B., & Lozić, D. (2018). Isolated Environment Tools for Software Development. 2018 *International Conference on Applied Mathematics & Computer Science* (ICAMCS), Paris, France, 48–52. https://doi.org/10.1109/ICAMCS46079.2018.00016
31. Tailwind CSS Documentation: official documentation Tailwind CSS. URL: https://tailwindcss.com/docs
32. Torskyi, O. I., & Hrytsiuk, Y. I. (2025). Application of machine learning to enhance the efficiency of automated software testing. *Scientific Bulletin of UNFU*, 35(4), 142–149. https://doi.org/10.36930/40350416
33. Vangavolu, S. V. (2022). Implementing Microservices Architecture with Node.js and Express in MEAN Applications. *International Journal of Advanced Research in Engineering and Technology, 13*(8), 56–65. https://doi.org/10.34218/IJARET_13_08_007
34. Veseliak, V. V., & Hrytsiuk, Y. I. (2024). Machine learning methods in epidemiological research. *Scientific Bulletin of UNFU*, 34(4), 59–67. https://doi.org/10.36930/40340408
35. Zhu, J., Bai, W., Zhang, H., Lin, W., Zhou, T., & Li, K. (2026, January). Adaptive multi-objective swarm intelligence for containerized microservice deployment. *Future Generation Computer Systems*, vol. 174, article ID 108012. https://doi.org/10.1016/j.future.2025.108012

B. O. Bekas[1], A. M. Prots[1], Yu. I. Hrytsiuk[1,2]

[1] Ukrainian National Forestry University, Lviv, Ukraine
[2] Lviv Polytechnic National University, Lviv, Ukraine

SOFTWARE FOR ISOLATED DEPLOYMENT OF A WEB APPLICATION IN A CONTAINERIZED ENVIRONMENT

Presents the results of the development of software for the isolated deployment of a web application within a containerized environment. It has been established that traditional approaches to application deployment are characterized by the complexity of configuring the execution environment, dependency management, and discrepancies between local and server-side environments, which lead to increased deployment time and a higher probability of errors. It has been determined that the use of containerization technologies enables the implementation of an isolated execution environment and significantly improves the reproducibility, portability, and reliability of applications. A web application for managing user requests in the field of social protection has been developed, featuring a fully containerized deployment based on Docker technology and the Docker Compose tool. The client-side of the application is implemented using the React library and the Vite build tool, with the Tailwind CSS framework employed to ensure a responsive user interface. The server-side is built on the Node.js runtime environment using the Express.js framework, whose efficiency is supported by the implementation of a REST API, JWT-based authentication, file processing via Multer, and integration with an AI assistant through the OpenAI cloud API infrastructure. Data are stored in a MySQL 8.0 database management system. Each system component is deployed in a separate Docker container, with orchestration performed using Docker Compose. A comparative analysis of different web application deployment approaches has demonstrated that the containerized approach significantly outperforms the traditional one in terms of environment reproducibility, component isolation, and redeployment speed. Testing across 44 deployment scenarios confirmed 100 % functionality and system reliability.

Keywords: application containerization; Docker; Docker Compose; REST API; client-server model; isolated deployment; React; Node.js.