Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра інженерії програмного забезпечення
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Методи автоматизованого розгортання та тестування програмного-забезпечення з використанням Hugo фреймворку

Виконав(ла): студент(ка) 4 курсу, групи СПс-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

		Романський В. В.
	(підпис)	(прізвище та ініціали)
Керівник		Бревус В. М.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Стоянов Ю. М.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М. Р.
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

Тернопіль 2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *програмної інженерії*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *121 «Інженерія програмного забезпечення»*
(шифр і назва спеціальності)

студенту Романському Владиславу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи *«Методи автоматизованого розгортання та тестування програмного забезпечення з використанням Hugo фреймворку»*

Керівник роботи Бревус Віталій Миколайович PhD
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *6* » *квітня* 2026 року № *4/9-172*

2. Термін подання студентом завершеної роботи *22.06.2026*

3. Вихідні дані до роботи *Технічне завдання*

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
1. Аналіз вимог до програмної системи
2. Проєктування та тестування програмного забезпечення
3. Реалізація вебсайту та автоматизованого розгортання
4. Безпека життєдіяльності, основи охорони праці
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Слайди презентації

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Мариненко С. Ю. к.т.н., доц. каф. МТ		
Нормоконтроль	Стоянов Ю. М. к.т.н. доц. каф. ПІ		

7. Дата видачі завдання *06.04.2026р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
	Розробка технічного завдання	*06.04 – 12.04*	
	Робота над першим розділом «Аналіз вимог до програмної системи»	*13.04 – 24.04*	
	Робота над другим розділом «Проєктування архітектури системи автоматизованого розгортання та тестування програмного забезпечення»	*25.04 – 04.05*	
	Робота над третім розділом «Реалізація вебсайту та автоматизованого розгортання»	*05.05 – 18.05*	
	Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»	*19.05 – 24.05*	
	Оформлення пояснювальної записки та графічного матеріалу	*25.05 – 07.06*	
	Перевірка на академічний плагіат, перевірка керівником та консультантами	*08.06 – 14.06*	
	Попередній захист кваліфікаційної роботи бакалавра	*15.06 – 21.06*	
	Захист кваліфікаційної роботи бакалавра		

Студент _____ Романський В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Бревус В.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2026. Сторінок 66, таблиць 2, рисунків 22, джерел 29, 1 лістинг, додатків 4, презентація.

Тема: Методи автоматизованого розгортання та тестування програмного забезпечення з використанням HUGO фреймворк.

Мета дипломної роботи полягає у дослідженні та практичній реалізації автоматизованого процесу розгортання і тестування програмного забезпечення з використанням генератора статичних вебсайтів Hugo та засобів безперервної інтеграції і доставки (CI/CD). У сучасних умовах швидкого розвитку вебтехнологій автоматизація процесів розробки, тестування та розгортання програмного забезпечення є важливим чинником підвищення якості програмних продуктів, зменшення кількості помилок та скорочення часу їх впровадження.

У процесі виконання роботи проведено аналіз сучасних підходів до автоматизованого тестування програмного забезпечення, концепцій безперервної інтеграції та безперервного розгортання, а також особливостей використання генератора статичних вебсайтів Hugo. Розроблено архітектуру програмного рішення, побудовано UML-моделі системи та реалізовано демонстраційний вебзастосунок університету на базі Hugo. Для автоматизації процесів збірки, тестування та публікації вебзастосунку використано платформу GitHub Actions і сервіс GitHub Pages.

У результаті виконання роботи створено систему автоматизованого розгортання вебзастосунку, яка забезпечує оновлення контенту, автоматичну перевірку коректності змін та спрощує супровід програмного забезпечення.

Ключові слова: Hugo, CI/CD, GitHub Actions, Github Pages, Автоматизоване Розгортання, тестування програмного забезпечення, вебзастосунок.

ABSTRACT

Bachelor’s thesis. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, specialty 121 “Software Engineering”. TNTU, 2026. The thesis comprises 66 pages, 2 tables, 22 figures, 29 references, 1 code listing, 4 appendices, and a presentation.

Topic: Methods of automated deployment and testing of software using the Hugo framework.

The aim of this thesis is to study and implement an automated process of software deployment and testing using the Hugo static site generator and continuous integration and continuous delivery (CI/CD) tools. In the context of rapidly evolving web technologies, automation of development, testing, and deployment processes is a key factor in improving software quality, reducing the number of errors, and shortening release time.

During the work, modern approaches to automated software testing, concepts of continuous integration and continuous deployment, as well as features of using the Hugo static site generator were analyzed. A system architecture was designed, UML models of the system were developed, and a demonstration university website based on Hugo was implemented. The GitHub Actions platform and GitHub Pages service were used to automate the processes of building, testing, and publishing the web resource.

As a result, an automated deployment system for a web resource was developed, which enables content updates, automatic validation of changes, and simplifies software maintenance.

Keywords: Hugo, CI/CD, GitHub Actions, GitHub Pages, automated deployment, software testing, web resource.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CI – Continuous Integration

CD – Continuous Delivery

CI/CD – Continuous Integration and Continuous Delivery

ПЗ – Програмне забезпечення

ТЗ – Технічне завдання

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

JS – JavaScript

JSON – JavaScript Object Notation

YAML – YAML Ain't Markup Language

URL – Uniform Resource Locator

SEO – Search Engine Optimization

UI – User Interface

UX – User Experience

IDE – Integrated Development Environment

Git – система контролю версій

GitHub – платформа для зберігання та супроводу програмного коду

GitHub Actions – сервіс автоматизації збірки, тестування та розгортання програмного забезпечення

Hugo – генератор статичних вебсайтів

VS Code – Visual Studio Code

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій програмне забезпечення стало невід'ємною складовою діяльності практично всіх сфер суспільства. Вебзастосунки використовуються для забезпечення доступу до інформації, надання електронних послуг, організації навчального процесу, підтримки комунікації між користувачами та автоматизації бізнес-процесів. Зі збільшенням складності програмних систем зростають вимоги до швидкості їх розробки, надійності функціонування та оперативності впровадження змін.

Традиційні методи розгортання програмного забезпечення передбачають виконання значної кількості ручних операцій. Адміністратор або розробник повинен самостійно виконувати збірку проєкту, копіювання файлів на сервер, перевірку працездатності системи та контроль коректності внесених змін. Такий підхід є трудомістким, потребує значних часових витрат та збільшує ризик виникнення помилок через людський фактор. Навіть незначна помилка під час розгортання може призвести до порушення роботи вебзастосунку, втрати даних або тимчасової недоступності сервісу для користувачів.

Одним із найефективніших способів вирішення зазначених проблем є впровадження автоматизації процесів розробки, тестування та розгортання програмного забезпечення. Використання концепцій Continuous Integration та Continuous Deployment дозволяє створити безперервний цикл доставки програмного продукту, у межах якого всі етапи перевірки та публікації змін виконуються автоматично. Застосування CI/CD-конвеєрів забезпечує підвищення якості програмного забезпечення, скорочення часу впровадження нових функцій та зменшення кількості помилок, що виникають у процесі супроводу системи [2,3].

Особливо актуальним питання автоматизації є для вебзастосунків освітніх закладів. Офіційні сайти університетів містять значний обсяг інформації про освітні програми, структуру закладу, наукову діяльність, вступну кампанію та інші аспекти функціонування установи. Оновлення такого контенту повинно

відбуватися швидко та безперервно, що вимагає використання сучасних підходів до адміністрування та супроводу вебзастосунків.

Одним із перспективних інструментів для створення сучасних вебсайтів є генератор статичних сайтів Hugo. Даний програмний продукт забезпечує швидке формування готових вебсторінок на основі текстових файлів та шаблонів оформлення. На відміну від традиційних систем керування контентом, Hugo не потребує використання баз даних та серверних інтерпретаторів під час роботи сайту, що позитивно впливає на швидкодію, безпеку та надійність вебзастосунку. Завдяки високій продуктивності та простоті інтеграції з інструментами автоматизації Hugo широко використовується для створення документації, корпоративних сайтів, блогів та інформаційних порталів.

У межах даної роботи досліджуються методи автоматизованого розгортання та тестування програмного забезпечення з використанням генератора статичних сайтів Hugo. Практична частина роботи передбачає створення прототипу вебзастосунку університету, реалізацію процесу автоматичного розгортання за допомогою GitHub Actions та впровадження автоматизованих механізмів перевірки працездатності системи.

Об'єктом дослідження є процеси розробки, тестування та розгортання веборієнтованого програмного забезпечення.

Предметом дослідження є методи автоматизації процесів розгортання та тестування програмного забезпечення з використанням генератора статичних сайтів Hugo та технологій CI/CD.

Метою роботи є дослідження та практична реалізація автоматизованої системи розгортання і тестування вебзастосунку на базі Hugo з використанням сучасних інструментів безперервної інтеграції та доставки.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз сучасних підходів до автоматизованого розгортання програмного забезпечення;
- дослідити методи тестування веборієнтованих програмних систем;
- виконати аналіз генератора статичних сайтів Hugo;

- спроєктувати структуру вебзастосунку університету;
- реалізувати автоматизований процес розгортання за допомогою GitHub Actions;
- виконати тестування розробленого рішення та оцінити його ефективність.

Практична цінність роботи полягає у створенні прототипу системи автоматизованого розгортання вебзастосунку, який може бути використаний для спрощення процесу супроводу інформаційних ресурсів освітніх закладів та інших організацій.

РОЗДІЛ 1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

У цьому розділі здійснюється комплексне дослідження предметної області, пов’язаної з процесами вибору, оцінювання та використання програмного забезпечення в умовах сучасних підприємств та організацій. Розглянуто існуючі підходи до автоматизації процесів розгортання, тестування та оцінки програмних рішень, зокрема у контексті веброзробки із застосуванням статичних сайт-генераторів, таких як Hugo [2,4].

На основі аналізу наявних ринкових та технологічних аналогів визначено ключові проблеми та технологічні обмеження існуючих рішень, серед яких: недостатній рівень автоматизації процесів тестування, складність інтеграції різних інструментів у єдиний CI/CD-процес, а також відсутність уніфікованого підходу до оцінювання якості згенерованого програмного продукту.

З метою формалізації вимог до програмного продукту здійснюється ідентифікація потенційних користувачів системи, а також аналіз їхніх потреб у контексті автоматизованого розгортання та тестування вебзастосунків. Визначаються основні ролі користувачів (розробник, DevOps-інженер, адміністратор системи), а також моделюється їхня взаємодія з системою шляхом визначення та детального опису ключових варіантів використання (use cases), що охоплюють процеси збірки, тестування та розгортання проєкту на базі Hugo.

1.1 Поняття та роль автоматизованого розгортання програмного забезпечення

У процесі створення програмного забезпечення одним із найважливіших етапів є його розгортання. Під розгортанням програмного забезпечення розуміють сукупність дій, спрямованих на перенесення готового програмного продукту із середовища розробки до середовища експлуатації, де ним можуть користуватися кінцеві споживачі. Традиційно даний процес передбачав виконання великої кількості ручних операцій, зокрема копіювання файлів на

сервер, налаштування конфігураційних параметрів, запуск необхідних служб та перевірку працездатності системи після внесення змін.

З розвитком інформаційних технологій та збільшенням складності програмних систем ручне розгортання почало створювати значні труднощі для розробників та адміністраторів. Кількість помилок, пов'язаних із людським фактором, суттєво зростала, а час, необхідний для впровадження нових версій програмного забезпечення, ставав дедалі більшим. Особливо гостро дана проблема проявляється під час супроводу вебзастосунків, які потребують частого оновлення контенту та регулярного випуску нових функціональних можливостей.

Для вирішення зазначених проблем було запропоновано концепцію автоматизованого розгортання програмного забезпечення. Основна ідея даного підходу полягає у максимальному усуненні ручних операцій та передачі більшості процедур спеціалізованим програмним засобам. У результаті процес розгортання стає стандартизованим, повторюваним та незалежним від людського фактору.

Автоматизоване розгортання передбачає створення певного сценарію або конвеєра, який виконує необхідні дії у визначеній послідовності. Після внесення змін до вихідного коду система автоматично здійснює перевірку проєкту, виконує його збірку, запускає тестування та публікує нову версію програмного забезпечення на сервері. Завдяки цьому суттєво скорочується час між завершенням розробки та появою оновлень у робочому середовища [2].

Важливою перевагою автоматизації є підвищення надійності програмних систем. Якщо процес розгортання виконується вручну, існує ризик пропустити окремі етапи або допустити помилку під час налаштування середовища. У випадку використання автоматизованих засобів усі операції виконуються за попередньо визначеним алгоритмом, що забезпечує однаковий результат незалежно від кількості запусків. Таким чином досягається стабільність роботи програмного забезпечення та спрощується процес його супроводу [2,3].

Не менш важливим фактором є швидкість впровадження змін. У сучасних умовах розробники прагнуть максимально скоротити час між створенням нового

функціоналу та його доступністю для користувачів. Автоматизоване розгортання дозволяє виконувати оновлення системи практично одразу після завершення

розробки та проходження тестування. Це особливо актуально для вебзастосунків, інформаційних порталів та корпоративних сайтів, де оперативність оновлення інформації має важливе значення.

Автоматизація процесів розгортання тісно пов'язана з концепцією DevOps, яка передбачає інтеграцію процесів розробки та адміністрування програмних систем. Основною метою DevOps є забезпечення безперервної взаємодії між командами розробників та фахівцями з експлуатації програмного забезпечення. У межах даного підходу автоматизоване розгортання виступає одним із ключових інструментів, що дозволяє забезпечити швидке та стабільне впровадження нових версій програмного продукту.

У сучасній практиці автоматизоване розгортання реалізується за допомогою різноманітних програмних платформ. Найбільшого поширення набули GitHub Actions, GitLab CI/CD, Jenkins, CircleCI та Azure DevOps. Дані інструменти дозволяють створювати спеціальні сценарії, які описують послідовність виконання всіх етапів розгортання програмного забезпечення. При цьому кожен етап може містити перевірку вихідного коду, запуск тестів, генерацію програмних файлів, формування звітів та автоматичне розміщення готового продукту на сервері.

Особливого значення автоматизоване розгортання набуває під час роботи зі статичними вебзастосуноками. На відміну від традиційних систем керування контентом, статичні сайти не потребують складного серверного середовища та можуть бути опубліковані практично на будь-якому вебсервері. Це створює сприятливі умови для впровадження повністю автоматизованого циклу розробки та розгортання.

Одним із найбільш популярних інструментів для створення статичних вебзастосунків є Hugo. Даний генератор сайтів забезпечує надзвичайно високу швидкість збірки проєктів та підтримує інтеграцію із сучасними платформами автоматизації. Використання Hugo дозволяє формувати готовий набір HTML-сторінок, які можуть бути автоматично розгорнуті після кожного оновлення

вихідного коду. У результаті забезпечується швидке оновлення вебзастосунку без необхідності виконання додаткових ручних операцій.

Таким чином автоматизоване розгортання програмного забезпечення є важливим напрямком розвитку сучасних інформаційних технологій. Його використання дозволяє скоротити час впровадження змін, підвищити якість програмного забезпечення, зменшити кількість помилок та забезпечити стабільність функціонування програмних систем. Саме тому дослідження методів автоматизації розгортання та їх практична реалізація є актуальним завданням у сфері програмної інженерії.

1.2 Основні підходи до тестування програмного забезпечення

Тестування програмного забезпечення є одним із найважливіших етапів життєвого циклу розробки програмних систем. Основною метою тестування є виявлення помилок, перевірка відповідності програмного продукту встановленим вимогам та забезпечення його стабільної роботи в реальних умовах експлуатації. Незважаючи на постійне вдосконалення технологій розробки, створення програмного забезпечення без помилок залишається практично неможливим завданням. Саме тому тестування виступає невід'ємною складовою процесу забезпечення якості програмних продуктів.

У сучасній програмній інженерії тестування розглядається не як окремий етап після завершення розробки, а як безперервний процес, який супроводжує створення програмного забезпечення від початкових етапів проєктування до його впровадження та подальшої експлуатації. Такий підхід дозволяє своєчасно виявляти помилки та зменшувати витрати на їх усунення.

Важливість тестування обумовлена тим, що навіть незначна помилка в програмному коді може призвести до серйозних наслідків. У випадку вебзастосунків помилки можуть спричинити недоступність окремих сторінок, втрату інформації, порушення роботи сервісів або погіршення користувацького досвіду. Особливо актуальним це питання є для інформаційних ресурсів освітніх

закладів, які повинні забезпечувати безперервний доступ до актуальної інформації для студентів, викладачів та абітурієнтів.

Одним із найпоширеніших підходів до тестування є модульне тестування. Його сутність полягає у перевірці окремих компонентів програмної системи незалежно від інших частин проєкту. У межах модульного тестування кожен функціональний елемент перевіряється окремо, що дозволяє швидко локалізувати помилки та спростити процес їх усунення. Модульне тестування зазвичай виконується розробниками на ранніх етапах створення програмного забезпечення та є основою для побудови більш складних механізмів контролю якості.

Наступним видом тестування є інтеграційне тестування. Його основним завданням є перевірка взаємодії між окремими компонентами програмної системи. Навіть якщо кожен модуль працює коректно окремо, під час їх об'єднання можуть виникати помилки обміну даними або несумісності між компонентами. Інтеграційне тестування дозволяє виявляти такі проблеми та забезпечувати коректну взаємодію всіх складових програмного продукту.

Важливу роль відіграє системне тестування. Даний підхід передбачає перевірку всієї програмної системи як єдиного цілого. У процесі системного тестування аналізується відповідність реалізованого програмного продукту початковим вимогам та очікуванням користувачів. Особлива увага приділяється функціональним можливостям системи, стабільності її роботи та здатності виконувати поставлені завдання в різних умовах експлуатації.

Завершальним етапом перевірки програмного забезпечення часто виступає приймальне тестування. Воно проводиться безпосередньо замовником або кінцевими користувачами з метою підтвердження готовності програмного продукту до використання. У межах даного тестування перевіряється відповідність системи бізнес-вимогам та сценаріям реального використання.

Окрім класифікації за рівнями тестування, існує поділ на функціональне та нефункціональне тестування. Функціональне тестування орієнтоване на перевірку правильності виконання функцій програмного забезпечення. Воно дозволяє встановити, чи відповідають результати роботи системи поставленим вимогам.

Нефункціональне тестування спрямоване на оцінювання таких характеристик, як продуктивність, надійність, безпека, масштабованість та зручність використання.

Для вебзастосунків важливого значення набуває тестування продуктивності. Даний вид тестування дозволяє оцінити швидкість роботи вебсайту, час завантаження сторінок та поведінку системи під час високого навантаження. Результати таких перевірок допомагають визначити потенційні вузькі місця в архітектурі програмного забезпечення та оптимізувати його роботу.

Ще одним важливим напрямком є тестування безпеки. Сучасні вебзастосунки постійно піддаються ризику несанкціонованого доступу, атак на серверну інфраструктуру та спроб викрадення даних. Тестування безпеки дозволяє виявляти потенційні вразливості та мінімізувати ризики, пов'язані з експлуатацією програмного забезпечення.

У процесі розробки вебзастосунків особливе значення мають автоматизовані методи тестування. Автоматизація дозволяє виконувати перевірки без участі людини та інтегрувати їх у процес безперервної інтеграції та доставки програмного забезпечення. У такому випадку кожна зміна програмного коду автоматично перевіряється перед розгортанням у робочому середовищі.

Для статичних сайтів, створених за допомогою генератора Hugo, автоматизоване тестування має певні особливості. Оскільки такі вебзастосунки не містять складної серверної логіки та баз даних, основна увага приділяється перевірці коректності процесу збірки, валідності сформованих HTML-документів та працездатності навігаційних елементів.

Однією з найбільш поширених перевірок є тестування процесу збірки проєкту. У випадку Hugo система повинна успішно сформувати набір готових вебсторінок без виникнення помилок. Якщо під час генерації сайту виявляються проблеми у структурі контенту або конфігураційних файлах, процес розгортання автоматично зупиняється до моменту їх усунення.

Наступним важливим етапом є перевірка гіперпосилань. З часом окремі сторінки можуть змінювати свої адреси або бути видаленими, що призводить до появи так званих «битих» посилань. Автоматизовані інструменти дозволяють

перевіряти всі внутрішні та зовнішні посилання вебзастосунку та повідомляти про виявлені проблеми.

Крім того, часто використовується перевірка валідності HTML-коду. Коректна структура вебсторінок позитивно впливає на їх відображення в браузерах, покращує доступність для користувачів та сприяє ефективнішій індексації пошуковими системами. Автоматичні валідатори дозволяють своєчасно виявляти помилки у сформованих документах та забезпечувати відповідність сучасним вебстандартам.

Особливу роль у сучасній розробці відіграє концепція безперервного тестування. Її сутність полягає у виконанні перевірок після кожної зміни програмного коду. У результаті розробники отримують оперативну інформацію про можливі помилки та можуть швидко їх усунути. Такий підхід значно підвищує якість програмного забезпечення та скорочує час, необхідний для випуску нових версій продукту.

Таким чином тестування програмного забезпечення є комплексним процесом, спрямованим на забезпечення якості, надійності та безпеки програмних систем. Використання сучасних автоматизованих методів тестування дозволяє значно підвищити ефективність процесу розробки, скоротити кількість помилок та забезпечити стабільну роботу програмного забезпечення. Для вебзастосуноків, створених на базі Hugo, автоматизоване тестування виступає важливою складовою CI/CD-конвеєра та забезпечує надійність процесу автоматичного розгортання.

1.3 Концепція безперервної інтеграції та безперервного розгортання (CI/CD)

У сучасних умовах розвитку програмної інженерії швидкість випуску нових версій програмного забезпечення стала одним із ключових факторів конкурентоспроможності інформаційних систем. Користувачі очікують оперативного впровадження нових функцій, швидкого виправлення помилок та стабільної роботи програмних продуктів. Для забезпечення таких вимог

традиційні підходи до розробки та розгортання програмного забезпечення поступово поступаються місцем сучасним методам автоматизації, серед яких особливе місце займає концепція безперервної інтеграції та безперервного розгортання, відома під скороченням CI/CD.

Термін Continuous Integration (CI) перекладається як безперервна інтеграція. Даний підхід передбачає регулярне об'єднання змін, внесених розробниками, до спільного репозиторію програмного проєкту. Кожна зміна автоматично перевіряється за допомогою спеціалізованих інструментів, що дозволяє своєчасно виявляти помилки та конфлікти між окремими компонентами системи.

До появи концепції безперервної інтеграції розробники часто працювали над окремими частинами проєкту протягом тривалого часу без синхронізації змін із центральним репозиторієм. У результаті під час об'єднання великої кількості модифікацій виникали численні конфлікти, усунення яких потребувало значних часових витрат. Безперервна інтеграція дозволила вирішити дану проблему шляхом регулярного внесення змін до спільного середовища та автоматичної перевірки їхньої коректності.

Основна ідея CI полягає у тому, що після кожного оновлення вихідного коду запускається автоматизований процес перевірки. Він може включати компіляцію програмного продукту, запуск модульних тестів, аналіз якості коду, перевірку конфігураційних файлів та інші процедури контролю. Якщо на будь-якому етапі виявляються помилки, процес інтеграції зупиняється, а розробник отримує повідомлення про необхідність виправлення виявлених недоліків.

Наступним етапом розвитку автоматизації стала концепція Continuous Delivery (безперервна доставка). Її основною метою є забезпечення готовності програмного продукту до розгортання у будь-який момент часу. Після успішного проходження всіх перевірок система автоматично формує готову до використання версію програмного забезпечення та розміщує її у спеціальному середовищі. Остаточне рішення щодо публікації оновлення приймається адміністратором або відповідальною особою.

Подальшим розвитком безперервної доставки стала концепція Continuous Deployment (безперервне розгортання). У цьому випадку після завершення всіх етапів перевірки програмний продукт автоматично публікується у робочому середовищі без додаткового втручання людини. Таким чином користувачі отримують доступ до нової версії системи одразу після успішного проходження всіх тестів.

Концепція CI/CD поєднує безперервну інтеграцію, безперервну доставку та безперервне розгортання в єдиний автоматизований процес. Завдяки цьому забезпечується повний цикл роботи з програмним забезпеченням — від внесення змін до вихідного коду до їх появи у продуктивному середовищі.

Типовий CI/CD-конвеєр складається з кількох основних етапів. Першим етапом є отримання вихідного коду з репозиторію. Для цього використовуються системи контролю версій, серед яких найбільш поширеною є Git. Після отримання актуальної версії проєкту запускається процес збірки програмного забезпечення. Залежно від типу проєкту цей етап може включати компіляцію вихідного коду, формування пакетів розгортання або генерацію статичних файлів.

Наступним етапом є тестування. На даній стадії виконуються різноманітні перевірки, спрямовані на виявлення помилок та підтвердження працездатності системи. До таких перевірок можуть належати модульне тестування, інтеграційне тестування, функціональне тестування, перевірка безпеки та аналіз якості коду.

Після успішного завершення тестування виконується розгортання програмного забезпечення. У межах автоматизованого конвеєра дана процедура відбувається без участі користувача. Система самостійно копіює необхідні файли на сервер, оновлює конфігурацію та забезпечує доступність нової версії програмного продукту.

Використання CI/CD надає низку важливих переваг. Насамперед суттєво скорочується час між створенням нової функції та її появою у робочому середовищі. Розробники можуть оперативно впроваджувати зміни та швидко реагувати на потреби користувачів. Крім того, автоматичне тестування дозволяє виявляти помилки на ранніх етапах розробки, що зменшує витрати на їх усунення.

Ще однією важливою перевагою є підвищення стабільності програмного забезпечення. Оскільки кожна зміна проходить серію автоматизованих перевірок, ризик потрапляння помилок у продуктивне середовище значно знижується. У результаті підвищується надійність системи та покращується якість кінцевого продукту.

Важливу роль у реалізації CI/CD відіграють сучасні інструменти автоматизації. До найбільш поширених платформ належать Jenkins, GitLab CI/CD, TeamCity, CircleCI, Travis CI та GitHub Actions. Дані системи дозволяють створювати сценарії автоматичного виконання необхідних операцій та забезпечують інтеграцію з різноманітними середовищами розробки.

Особливої популярності останніми роками набув сервіс GitHub Actions. Його перевагою є тісна інтеграція із платформою GitHub, що дозволяє створювати автоматизовані робочі процеси без використання додаткових серверів. Усі етапи CI/CD описуються у вигляді спеціальних конфігураційних файлів формату YAML, які зберігаються безпосередньо у репозиторії проєкту.

Для вебзастосуноків, створених за допомогою генератора статичних сайтів Hugo, використання CI/CD є особливо доцільним. Після внесення змін до контенту або структури сайту система автоматично запускає процес генерації вебсторінок, перевіряє коректність їх формування та публікує оновлену версію ресурсу. Завдяки цьому підтримка сайту значно спрощується, а ризик виникнення помилок під час оновлення суттєво зменшується.

У межах даної роботи концепція CI/CD використовується як основа для побудови автоматизованої системи розгортання вебзастосунку університету. Передбачається використання GitHub Actions для автоматичного запуску збірки Hugo-проєкту, виконання необхідних перевірок та публікації готової версії сайту після кожного оновлення вихідного коду.

Таким чином концепція безперервної інтеграції та безперервного розгортання є одним із найважливіших напрямків розвитку сучасної програмної інженерії. Використання CI/CD дозволяє автоматизувати значну частину процесів розробки та супроводу програмного забезпечення, підвищити його якість, скоротити час

випуску нових версій та забезпечити стабільність функціонування інформаційних систем. Саме тому впровадження даних підходів є актуальним завданням для сучасних вебпроєктів, зокрема для інформаційних ресурсів освітніх установ.

1.4 Аналіз фреймворку Hugo та його переваг

Hugo є сучасним генератором статичних вебсайтів з відкритим вихідним кодом, який використовується для швидкого створення інформаційних ресурсів різного призначення. Проєкт був створений у 2013 році Бйорном Еріком Педерсеном та реалізований мовою програмування Go. Основною особливістю Hugo є надзвичайно висока швидкість генерації вебсторінок, що дозволяє формувати великі вебзастосунки за лічені секунди.

На відміну від традиційних систем керування контентом, таких як WordPress, Joomla або Drupal, Hugo не використовує бази даних та серверні інтерпретатори під час роботи сайту. Весь контент зберігається у вигляді текстових файлів формату Markdown, а під час збірки система автоматично генерує готові HTML-сторінки, CSS-файли та інші необхідні ресурси. Після завершення генерації сайт може бути розміщений на будь-якому вебсервері або сервісі статичного хостингу.

Архітектура Hugo базується на розділенні контенту, шаблонів оформлення та конфігураційних параметрів. Контент представлений у вигляді окремих текстових файлів, шаблони визначають зовнішній вигляд сторінок, а конфігураційний файл містить параметри функціонування сайту. Такий підхід забезпечує зручність супроводу та дозволяє розробникам швидко вносити зміни до структури вебзастосунку.

Важливою перевагою Hugo є підтримка великої кількості готових тем оформлення. Це дозволяє створювати сучасні вебсайти без необхідності розробки дизайну з нуля. Крім того, система підтримує багатомовність, автоматичне формування карт сайту, генерацію RSS-стрічок та інтеграцію із зовнішніми сервісами.

Особливого значення Hugo набуває у контексті автоматизації процесів розробки та розгортання програмного забезпечення. Завдяки відсутності залежності від серверної логіки сайт може бути повністю сформований під час виконання CI/CD-конвеєра. Після завершення збірки достатньо скопіювати сформовані файли на сервер або до хмарного сховища, що значно спрощує процедуру розгортання.

Порівняно з іншими генераторами статичних сайтів Hugo має низку переваг. Наприклад, Jekyll працює на базі Ruby та потребує встановлення додаткових залежностей, що ускладнює процес налаштування середовища. Gatsby використовує React і характеризується більш складною архітектурою. Hugo є самодостатнім програмним продуктом, який не потребує додаткових компонентів для роботи та демонструє значно вищу швидкість генерації сторінок.

Ще однією перевагою Hugo є високий рівень безпеки. Оскільки статичні сайти не виконують серверний код та не використовують бази даних, значно зменшується кількість потенційних вразливостей. Відсутність серверної логіки практично виключає можливість SQL-ін'єкцій, атак типу Remote Code Execution та інших поширених загроз.

РОЗДІЛ 2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ АВТОМАТИЗОВАНОГО РОЗГОРТАННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Проєктування архітектури є одним із найважливіших етапів розробки будь-якої інформаційної системи, оскільки саме на цьому етапі визначається структура програмного продукту, взаємодія між його компонентами та принципи функціонування окремих модулів. Грамотно спроєктована архітектура забезпечує високу продуктивність системи, простоту її супроводу, можливість масштабування та інтеграції з іншими сервісами.

У рамках дипломної роботи було розроблено архітектуру системи автоматизованого розгортання та тестування вебпроєкту, побудованого за допомогою генератора статичних сайтів Hugo. Основною метою розробки є забезпечення безперервної інтеграції та безперервного розгортання програмного забезпечення без участі адміністратора після внесення змін до репозиторію.

В основі запропонованої системи лежить концепція CI/CD (Continuous Integration / Continuous Delivery), яка передбачає автоматичне виконання всіх необхідних етапів після оновлення вихідного коду. Такий підхід дозволяє мінімізувати кількість людських помилок, скоротити час публікації нових версій програмного продукту та забезпечити стабільність роботи інформаційної системи.

Під час проєктування було визначено основні компоненти системи та взаємозв'язки між ними. До складу системи входять локальне середовище розробника, система контролю версій GitHub, сервіс автоматизації GitHub Actions, генератор статичних сайтів Hugo, модуль автоматизованого тестування та сервер публікації.

Розробник створює або редагує контент сайту, після чого виконує фіксацію змін у локальному репозиторії та передає їх до віддаленого сховища GitHub. Саме ця дія є тригером запуску автоматизованого конвеєра CI/CD.

Після отримання нової версії коду GitHub Actions автоматично запускає сценарій збірки. На цьому етапі виконується встановлення необхідного програмного забезпечення, перевірка конфігураційних файлів, завантаження залежностей та запуск генератора Hugo. У результаті формується готовий статичний вебсайт, який надалі проходить комплекс автоматичних перевірок.

До складу перевірок входить аналіз правильності побудови HTML-документів, перевірка структури каталогів, контроль відсутності пошкоджених внутрішніх посилань, аналіз конфігураційних параметрів та контроль успішності виконання процесу збірки. Якщо на будь-якому етапі виникає помилка, процес автоматично припиняється, а розробник отримує відповідне повідомлення.

У випадку успішного проходження всіх перевірок запускається процедура автоматичного розгортання. Готовий вебсайт копіюється на сервер або до сервісу GitHub Pages, після чого нова версія стає доступною користувачам.

Запропонована архітектура має модульну структуру, що значно спрощує її модернізацію. За необхідності до неї можуть бути додані нові модулі автоматичного тестування, аналізу безпеки, перевірки продуктивності або інтеграції з зовнішніми сервісами моніторингу.

Однією з головних переваг використання Hugo є швидкість генерації статичних сторінок. На відміну від динамічних CMS, статичний сайт не потребує виконання серверних скриптів під час обробки запиту користувача, що позитивно впливає на продуктивність, безпеку та швидкість завантаження сторінок.

Використання GitHub Actions забезпечує централізоване зберігання сценаріїв автоматизації разом із вихідним кодом проєкту. Це дозволяє відтворити процес розгортання на будь-якому комп'ютері без необхідності додаткового налаштування середовища.

Запропонована архітектура також відповідає сучасним принципам DevOps, відповідно до яких процеси розробки, тестування та розгортання повинні бути максимально автоматизованими та інтегрованими між собою. Це сприяє підвищенню якості програмного забезпечення, скороченню часу випуску нових

версій та спрощенню командної роботи. На рисунку 2.1 зображено UML-діаграму вавріантів використання системи автоматизованого розгортання.

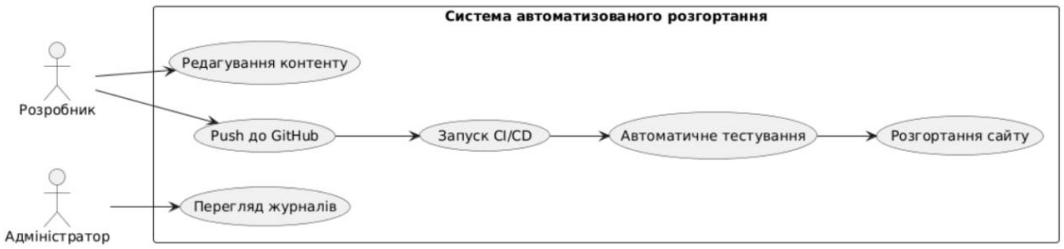


Рисунок 2.1 – UML-діаграма варіантів використання системи автоматизованого розгортання

Підсумовуючи, можна зробити висновок, що розроблена архітектура забезпечує повністю автоматизований цикл створення, перевірки та розгортання програмного забезпечення. Її використання дозволяє підвищити якість кінцевого продукту, скоротити час оновлення вебдодатку та мінімізувати ризики, пов'язані з ручним виконанням операцій.

2.1 UML-моделювання системи автоматизованого розгортання та тестування програмного забезпечення

Під час проєктування сучасних програмних систем важливим етапом є створення формалізованої моделі, яка дозволяє описати структуру системи, взаємозв'язки між окремими компонентами та алгоритми виконання основних процесів. Одним із найбільш поширених засобів моделювання програмного забезпечення є уніфікована мова моделювання UML (Unified Modeling Language), яка широко використовується під час розробки інформаційних систем різного рівня складності.

Застосування UML дає можливість ще на етапі проєктування визначити функціональні можливості системи, встановити взаємозв'язки між користувачами та програмними компонентами, а також виявити можливі недоліки архітектури до початку практичної реалізації програмного продукту. Використання графічних

моделей значно спрощує розуміння логіки роботи системи та покращує взаємодію між розробниками.

У рамках даної дипломної роботи UML-моделювання застосовується для опису системи автоматизованого розгортання та тестування вебпроєкту, створеного за допомогою генератора статичних сайтів Hugo. Побудовані діаграми відображають функціональні можливості системи, алгоритм її роботи та взаємодію між окремими компонентами автоматизованого конвеєра CI/CD [5,7].

Для опису функціонування системи було використано три основні типи UML-діаграм: діаграму варіантів використання (Use Case Diagram), діаграму діяльності (Activity Diagram) та діаграму послідовності (Sequence Diagram). Сукупність цих моделей дозволяє комплексно описати процес автоматизованого розгортання програмного забезпечення та забезпечує повне уявлення про логіку роботи інформаційної системи.

2.1.1 Діаграма варіантів використання

Першим етапом UML-моделювання є побудова діаграми варіантів використання. Її основною метою є визначення функціональних можливостей програмної системи та встановлення взаємодії між користувачами і програмними компонентами.

У розробленій системі основними акторами є розробник програмного забезпечення та адміністратор системи. Розробник здійснює створення та редагування контенту, внесення змін до репозиторію GitHub, контроль результатів автоматичного тестування та аналіз журналів виконання процесу збірки. Адміністратор відповідає за налаштування інфраструктури, підтримку середовища розгортання та контроль працездатності системи.

Після внесення змін до вихідного коду або контенту вебсайту розробник виконує операцію push до віддаленого репозиторію. Це автоматично запускає процес безперервної інтеграції, у рамках якого система виконує збірку проєкту, перевірку його коректності та автоматичне розгортання нової версії.

Діаграма варіантів використання демонструє, що більшість операцій виконується автоматично, без необхідності ручного втручання адміністратора. Це дозволяє скоротити час доставки нових версій програмного забезпечення та мінімізувати кількість помилок, пов'язаних із людським фактором.На рисунку 2.2 зображено UML-діаграму діяльності процесу автоматизованого розгортання.

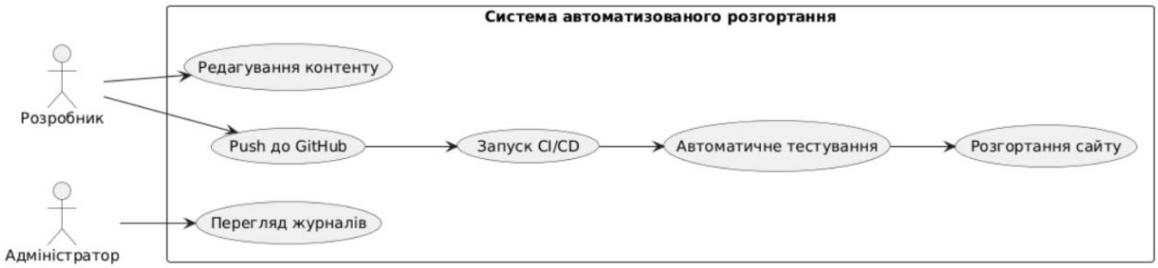


Рисунок 2.2 – UML-діаграма діяльності процесу автоматизованого розгортання

Аналіз наведеної діаграми дозволяє зробити висновок, що функціональні можливості системи повністю охоплюють усі етапи створення, тестування та розгортання програмного забезпечення. При цьому користувач взаємодіє лише з мінімальною кількістю функцій, а більшість процесів автоматизується засобами CI/CD.

2.1.2 Діаграма діяльності

Для опису алгоритму роботи системи була побудована UML-діаграма діяльності. Вона дозволяє представити послідовність виконання операцій від моменту внесення змін до програмного коду до завершення автоматичного розгортання вебдодатку.

Процес починається зі створення або редагування контенту. Після завершення роботи розробник виконує commit та push до репозиторію GitHub. Ця дія активує сценарій GitHub Actions, який автоматично запускає конвеєр безперервної інтеграції.

На наступному етапі здійснюється встановлення середовища виконання та запуск генератора Hugo. У процесі генерації формується статична версія вебсайту, яка проходить комплекс перевірок.

Особливістю алгоритму є наявність умовного переходу після завершення тестування. Якщо перевірки завершилися успішно, система автоматично переходить до етапу розгортання. Якщо ж під час тестування виявлено помилки, процес завершується формуванням повідомлення для розробника без публікації нової версії сайту.

Такий механізм забезпечує високий рівень надійності програмного забезпечення та запобігає потраплянню некоректних змін у продуктивне середовище. На рисунку 2.3 зображено UML-діаграму діяльності процесу автоматизованого розгортання та тестування програмного забезпечення

Рисунок 2.3 – UML-діаграма діяльності процесу автоматизованого розгортання та тестування програмного забезпечення

Використання діаграми діяльності дозволяє наочно представити логіку виконання автоматизованого процесу та спрощує аналіз можливих точок виникнення помилок. Крім того, дана модель є основою для подальшого вдосконалення CI/CD-конвеєра.

2.1.3 Діаграма послідовності

Для моделювання взаємодії між окремими компонентами системи використовується UML-діаграма послідовності. Вона демонструє порядок обміну повідомленнями між програмними модулями та дозволяє простежити логіку роботи системи в часі.

Основними учасниками процесу є розробник, GitHub, GitHub Actions, генератор Hugo та сервер публікації. Після виконання push GitHub формує подію, яка активує GitHub Actions. Далі запускається процес складання проєкту, створення статичних сторінок та автоматичного тестування.

Після успішного завершення всіх перевірок GitHub Actions передає сформований вебзастосунок до сервера публікації, де відбувається оновлення поточної версії сайту. Якщо на будь-якому етапі виникає помилка, процес автоматично припиняється, а користувач отримує повідомлення про необхідність усунення виявлених недоліків.

Завдяки використанню діаграми послідовності можна детально простежити взаємодію між окремими компонентами системи та оцінити ефективність реалізованого алгоритму автоматизованого розгортання. На рисунку 2.4 зображено UML-діаграму послідовності взаємодії компонентів системи автоматизованого розгортання та тестування програмного забезпечення

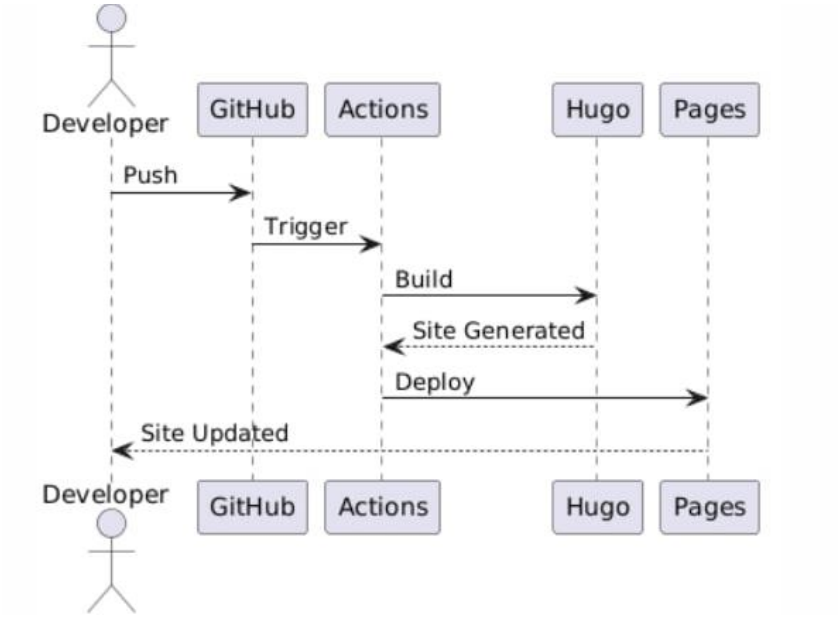


Рисунок 2.4 – UML-діаграма послідовності взаємодії компонентів системи автоматизованого розгортання та тестування програмного забезпечення

Отже, використання UML-моделювання дозволило сформувати повну модель розроблюваної системи автоматизованого розгортання та тестування програмного забезпечення. Побудовані діаграми відображають функціональні можливості системи, алгоритм її роботи та взаємодію між програмними компонентами. Це значно спрощує процес розробки, супроводу та подальшого вдосконалення програмного продукту, а також забезпечує чітке документування його архітектури відповідно до сучасних вимог програмної інженерії [4,5].

2.2 Реалізація автоматизованого розгортання програмного забезпечення з використанням HUGO

Автоматизоване розгортання програмного забезпечення є невід'ємною складовою сучасного процесу розробки, оскільки дозволяє значно скоротити час доставки нових версій програмного продукту, мінімізувати вплив людського фактора та забезпечити стабільність процесу публікації. Використання автоматизованих засобів розгортання особливо актуальне для вебпроєктів, які регулярно оновлюються та потребують швидкого внесення змін.

У межах даної роботи реалізовано систему автоматизованого розгортання статичного вебсайту, побудованого за допомогою генератора Hugo. Основою реалізації виступає концепція безперервної інтеграції та безперервного розгортання (CI/CD), яка забезпечує автоматичне виконання всіх етапів після внесення змін до репозиторію [11,12].

Процес автоматизованого розгортання починається з локального середовища розробника. Після внесення змін до контенту або шаблонів сайту виконується створення нового коміту та передача його до віддаленого репозиторію GitHub. Виконання операції push автоматично створює подію, яка активує сценарій GitHub Actions.

GitHub Actions є сервісом автоматизації, що дозволяє виконувати сценарії безпосередньо у хмарному середовищі. Конфігурація процесу описується у YAML-файлі та зберігається разом із вихідним кодом проєкту. Це забезпечує повторюваність процесу розгортання незалежно від робочого середовища.

Після запуску конвеєра виконується клонування репозиторію та підготовка середовища виконання. Система автоматично встановлює необхідну версію Hugo та додаткові залежності, після чого запускається процес генерації статичного вебсайту.

На наступному етапі сформований сайт проходить автоматичну перевірку. Контролюється успішність збірки, правильність структури каталогів, наявність усіх необхідних ресурсів, коректність створення HTML-документів та відсутність критичних помилок під час компіляції.

Після успішного завершення перевірок запускається процедура автоматичного розгортання. Сформовані файли копіюються до каталогу публікації GitHub Pages або на віддалений вебсервер. У результаті користувачі отримують доступ до нової версії сайту без виконання будь-яких ручних операцій.

Використання автоматизованого розгортання значно скорочує час публікації нових матеріалів. Якщо традиційний процес передбачає ручне копіювання файлів, перевірку структури каталогів та налаштування сервера, то запропонована система виконує всі ці операції автоматично протягом кількох хвилин.

Важливою перевагою є можливість автоматичного повернення до попередньої стабільної версії у випадку виникнення критичних помилок. Зберігання історії змін у Git забезпечує контроль версій та спрощує процес супроводу програмного забезпечення.

Ще однією перевагою є централізоване зберігання конфігурації CI/CD. Опис усіх етапів розгортання знаходиться у YAML-файлі, що дозволяє легко модифікувати процес, додавати нові перевірки або інтегрувати додаткові інструменти тестування.

Практична реалізація автоматизованого розгортання підтвердила ефективність використання Hugo у поєднанні з GitHub Actions. Генерація статичного сайту займає мінімальний час, а автоматизація всіх операцій значно знижує ризик появи помилок під час оновлення вебзастосунку.

Таким чином, реалізована система автоматизованого розгортання відповідає сучасним принципам DevOps, забезпечує швидку доставку програмного продукту, підвищує його надійність та створює основу для подальшої інтеграції автоматизованого тестування, моніторингу та контролю якості.

2.3 Реалізація автоматизованого тестування програмного забезпечення

Автоматизоване тестування є невід'ємною складовою сучасного процесу розробки програмного забезпечення та одним із ключових елементів методології DevOps. Його використання дозволяє своєчасно виявляти помилки, контролювати якість програмного продукту та забезпечувати стабільність роботи інформаційної системи після внесення змін до вихідного коду. Інтеграція автоматизованого тестування до процесу безперервної інтеграції та безперервного розгортання (CI/CD) дозволяє виконувати перевірку програмного забезпечення без участі користувача, що значно скорочує час розробки та мінімізує ризик виникнення критичних помилок.

У розробленій системі автоматизоване тестування реалізоване за допомогою сервісу GitHub Actions, який автоматично запускає необхідні перевірки після

кожного внесення змін до репозиторію. Такий підхід дозволяє виконувати тестування ще до моменту публікації нової версії вебзастосунку, забезпечуючи контроль якості на всіх етапах життєвого циклу програмного забезпечення.

Після виконання операції Commit та Push до віддаленого репозиторію GitHub активується сценарій автоматизованого тестування. На першому етапі здійснюється отримання актуальної версії проєкту та підготовка середовища виконання. Далі встановлюється необхідна версія генератора статичних сайтів Hugo та перевіряється коректність конфігураційних параметрів проєкту.

Наступним кроком є автоматична збірка вебзастосунку. Під час виконання цієї операції Hugo генерує статичні HTML-сторінки, використовуючи шаблони оформлення, контент та конфігураційні файли. У разі виникнення синтаксичних помилок або відсутності необхідних ресурсів процес збірки автоматично припиняється, а система формує повідомлення про помилку.

Одним із важливих етапів автоматизованого тестування є перевірка правильності структури сформованого вебсайту. Аналізується наявність необхідних HTML-файлів, таблиць стилів CSS, JavaScript-ресурсів, зображень та інших компонентів, що використовуються під час роботи вебзастосунку. Така перевірка дозволяє запобігти появі сторінок із відсутніми елементами або некоректним відображенням інформації.

Додатково система виконує перевірку конфігураційних файлів Hugo. Контролюється правильність параметрів генерації сайту, налаштування тем оформлення, структури меню, мовних конфігурацій та шляхів до статичних ресурсів. Виявлення помилок на цьому етапі дозволяє уникнути некоректної роботи вебсайту після його розгортання.

Особлива увага приділяється перевірці внутрішніх посилань. Автоматичний аналіз дозволяє виявити неіснуючі сторінки, помилкові URL-адреси та порушення навігації між розділами вебзастосунку. Це позитивно впливає на якість користувацького інтерфейсу та покращує індексацію сайту пошуковими системами.

Під час тестування також здійснюється перевірка сценаріїв GitHub Actions, описаних мовою YAML. Аналіз правильності синтаксису та структури файлів

забезпечує стабільне виконання конвеєра безперервної інтеграції та виключає можливість помилок під час автоматичного запуску процесу розгортання.

У разі успішного завершення всіх перевірок система переходить до етапу автоматизованого розгортання. Якщо хоча б один із тестів завершується з помилкою, процес публікації припиняється, а розробник отримує повідомлення про виявлені недоліки через журнал виконання GitHub Actions. Такий механізм дозволяє запобігти публікації нестабільних або некоректних версій програмного забезпечення.

Автоматизоване тестування забезпечує не лише контроль якості програмного продукту, а й підвищує ефективність командної розробки. Кожна зміна незалежно перевіряється перед інтеграцією до основної гілки репозиторію, що значно знижує ймовірність появи конфліктів між окремими компонентами системи та спрощує супровід проєкту.

Практичне використання автоматизованого тестування у поєднанні з генератором статичних сайтів Hugo підтвердило ефективність запропонованого підходу. Автоматичне виконання перевірок дозволило скоротити час контролю нових версій вебзастосунку, підвищити стабільність його роботи та мінімізувати кількість помилок, пов'язаних із людським фактором.

Таким чином, реалізована система автоматизованого тестування забезпечує високий рівень контролю якості програмного забезпечення, інтегрується із процесом безперервної інтеграції та автоматизованого розгортання, а також відповідає сучасним принципам DevOps і дозволяє підвищити ефективність розробки та супроводу вебпроєктів. На рисунку 2.5 зображено Алгоритм автоматизованого тестування програмного забезпечення.

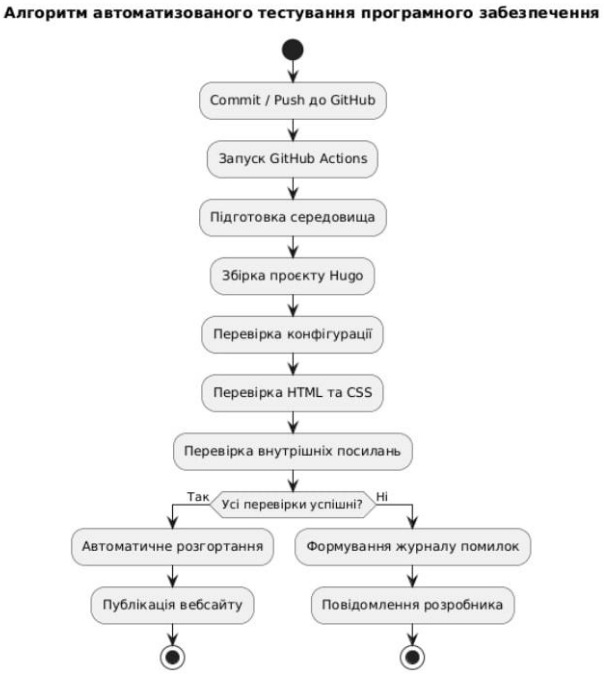


Рисунок 2.5 – Алгоритм автоматизованого тестування програмного забезпечення

Після наведеної схеми можна зробити висновок, що процес автоматизованого тестування є невід'ємною частиною CI/CD-конвеєра та забезпечує перевірку програмного забезпечення перед його публікацією. Використання автоматичних перевірок дозволяє своєчасно виявляти помилки, підвищувати якість вебзастосунку та гарантувати стабільність його функціонування.

2.4 Оцінка ефективності використання автоматизованого розгортання та тестування програмного забезпечення

Сучасні підходи до розробки програмного забезпечення орієнтовані на максимальну автоматизацію процесів створення, тестування та розгортання програмних продуктів. Використання технологій безперервної інтеграції та безперервного розгортання (CI/CD) дозволяє значно підвищити якість програмного забезпечення, скоротити час випуску нових версій та мінімізувати кількість помилок, пов'язаних із людським фактором.

У межах даної роботи реалізовано систему автоматизованого розгортання вебпроєкту, створеного за допомогою генератора статичних сайтів Hugo. Для

автоматизації процесів складання, тестування та публікації використано платформу GitHub Actions, а кінцеве розміщення вебзастосунку здійснюється на сервісі GitHub Pages. Така архітектура дозволяє забезпечити повністю автоматизований цикл доставки програмного продукту без необхідності ручного виконання проміжних операцій.

Однією з основних переваг запропонованого рішення є значне скорочення часу розгортання нової версії програмного забезпечення. У традиційному підході розробнику необхідно вручну виконувати збірку проєкту, перевірку його працездатності та копіювання файлів на сервер. При використанні автоматизованого конвеєра всі зазначені операції виконуються автоматично після внесення змін до репозиторію, що дозволяє скоротити час оновлення вебзастосунку до кількох хвилин.

Не менш важливою перевагою є підвищення якості програмного забезпечення. Інтеграція автоматизованого тестування до процесу CI/CD забезпечує перевірку кожної нової версії проєкту перед її публікацією. У разі виявлення помилок процес розгортання автоматично припиняється, що запобігає потраплянню нестабільної версії до продуктивного середовища.

Використання генератора статичних сайтів Hugo позитивно впливає на швидкодію вебзастосунку. На відміну від динамічних систем керування контентом, Hugo генерує готові HTML-сторінки ще на етапі складання проєкту. Завдяки цьому під час відкриття сайту не виконується обробка серверних сценаріїв або запитів до бази даних, що забезпечує швидке завантаження сторінок та зменшує навантаження на серверну інфраструктуру.

Автоматизоване розгортання також сприяє підвищенню надійності процесу оновлення вебзастосунку. Усі дії виконуються відповідно до визначеного сценарію GitHub Actions, що виключає можливість пропуску окремих етапів або помилок, пов'язаних із ручним копіюванням файлів. Крім того, історія виконання всіх процесів зберігається у журналі GitHub Actions, що значно спрощує аналіз та усунення можливих несправностей.

Таблиця 2.1 – Порівняння ручного та автоматизованого розгортання

Критерії	**Ручне розгортання**	**Автоматизоване розгортання**
Час оновлення	Від 3 до 10 хвилин	1-3 хвилини
Ймовірність помилки	Висока	Мінімально
Автоматичне тестування	Відсутнє	Виконується автоматично
Контроль версій	Частково	ПовністюРеалізований
Масштабованість	Обмежена	Висока
Повторюваність процесу	Не гарантується	Гарантується
Журнал виконання	Відсутній або вводиться вручну	Автоматично формується

З наведеного порівняння видно, що автоматизоване розгортання має суттєві переваги перед традиційним підходом [13]. Використання GitHub Actions та Hugo дозволяє забезпечити повторюваність усіх операцій, скоротити час доставки нових версій програмного забезпечення та підвищити загальний рівень надійності інформаційної системи [15].

Економічний ефект від впровадження автоматизації проявляється у скороченні трудових витрат на виконання рутинних операцій, зменшенні кількості помилок та прискоренні процесу випуску нових версій програмного забезпечення. Для невеликих вебпроєктів це дозволяє відмовитися від окремого серверного програмного забезпечення та використовувати безкоштовні сервіси GitHub, що додатково знижує витрати на підтримку інформаційної системи.

Отримані результати підтверджують доцільність використання генератора статичних сайтів Hugo у поєднанні із сервісами GitHub Actions та GitHub Pages для автоматизації процесів тестування та розгортання програмного забезпечення. Запропонований підхід забезпечує високу швидкодію, надійність та простоту супроводу вебпроєкту, а також відповідає сучасним принципам DevOps та безперервної інтеграції.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ВЕБСАЙТУ ТА АВТОМАТИЗОВАНОГО РОЗГОРТАННЯ

Після проведення аналізу предметної області було виконано проєктування структури демонстраційного вебсайту, який реалізовано з використанням генератора статичних сайтів Hugo. Основною метою створення вебзастосунку є демонстрація можливостей автоматизованого розгортання та оновлення статичних вебпроєктів із застосуванням сучасних технологій безперервної інтеграції та доставки (CI/CD).

При розробленні структури особлива увага приділялася простоті навігації, швидкості завантаження сторінок та зручності користування. Було обрано класичну схему побудови вебзастосунку, що складається із шапки сайту, навігаційного меню, основної області відображення контенту та нижнього колонтитулу.

Структура сайту побудована за модульним принципом, що дозволяє незалежно редагувати окремі сторінки без внесення змін [11]. Такий підхід значно спрощує подальший супровід і розвиток вебпроєкту.

До складу вебзастосунку входять такі основні сторінки:

- ·головна сторінка;
- ·сторінка «Про університет»;
- ·сторінка «Наші сервіси»;
- ·сторінка «Новини»;
- ·сторінка «Контакти».

Для переходу між розділами використовується горизонтальне меню навігації, яке розташоване у верхній частині сайту. Завдяки цьому користувач може швидко перейти до необхідної інформації незалежно від поточної сторінки.

Окрему увагу приділено візуальному оформленню вебзастосунку. Для стилізації використано корпоративні кольори університету, що забезпечує впізнаваність ресурсу та покращує його зовнішній вигляд.

На рисунку 3.1 представлено головну сторінку реалізованого вебсайту.

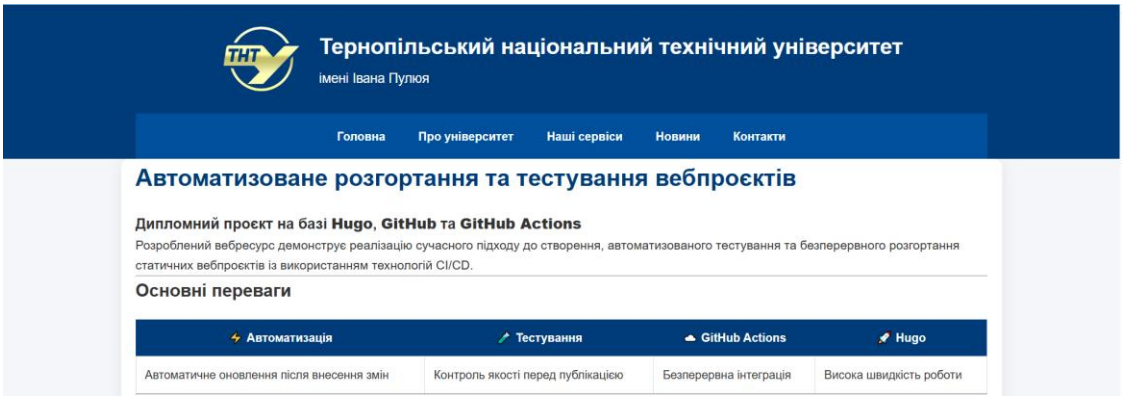


Рисунок 3.1 – Головна сторінка вебсайту

3.1 Реалізація головної сторінки

Головна сторінка виконує функцію інформаційного центру вебзастосунку та містить основні відомості про дипломний проєкт. Саме вона є першою сторінкою, яку бачить користувач після відкриття сайту.

У верхній частині сторінки розміщено логотип Тернопільського національного технічного університету імені Івана Пулюя та навігаційне меню, яке забезпечує доступ до інших інформаційних розділів.

Нижче розташовано інформаційний блок із чотирма картками, що демонструють ключові переваги використання автоматизованого розгортання вебпроєктів. Кожна картка містить іконку, заголовок та короткий опис функціональних можливостей системи.

До основних переваг належать:

- швидке оновлення вебзастосунку після внесення змін;
- автоматизована перевірка працездатності програмного забезпечення;
- використання технології безперервної інтеграції та доставки;
- підвищення надійності та стабільності роботи вебсайту.

Основний інформаційний блок сторінки містить тему дипломної роботи, короткий опис реалізованої системи та інформацію про використані технології.

Для побудови інтерфейсу застосовано HTML5 та CSS3, що дозволило створити сучасний дизайн із використанням адаптивної верстки [17]. Завдяки цьому сторінка коректно відображається на різних пристроях та екранах різної роздільної здатності [18].

Додатковою перевагою є мінімальний час завантаження, оскільки Hugo генерує статичні HTML-файли без необхідності виконання серверних сценаріїв. На рисунку 3.2 представлено інтерфейс головної сторінки вебсайту.

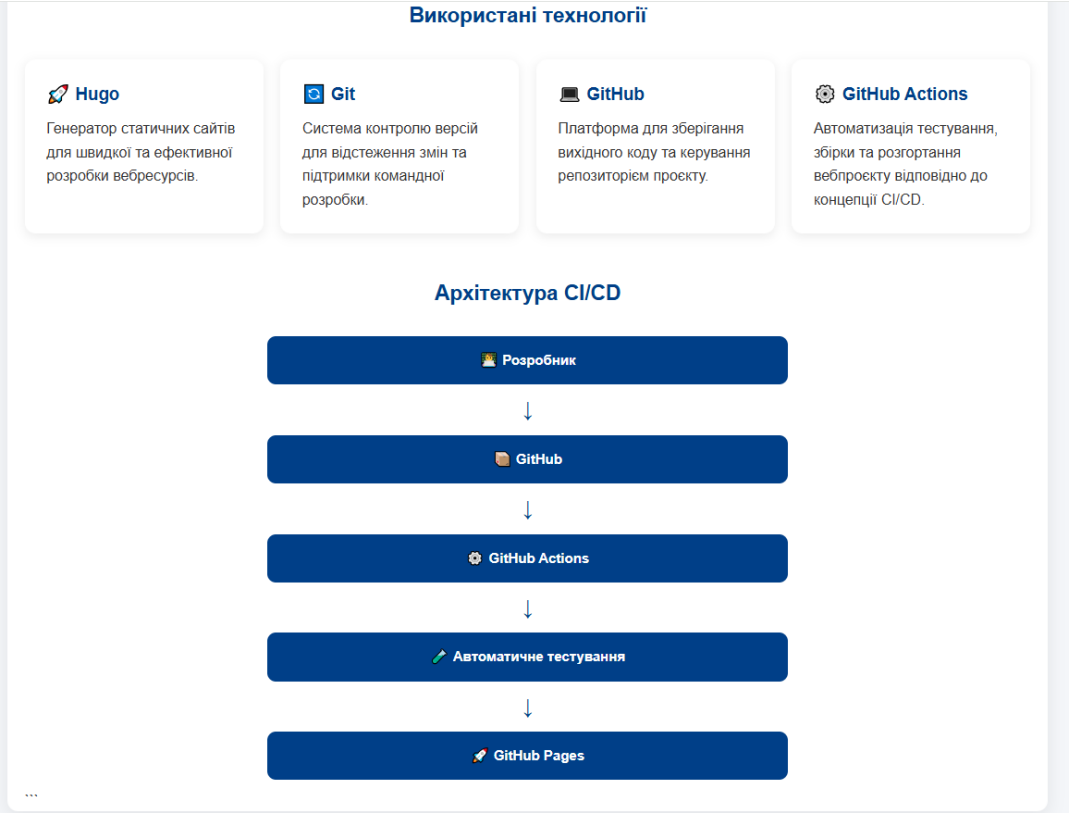


Рисунок 3.2 - Інтерфейс головної сторінки вебсайту

3.2 Архітектура вебзастосунку

Під час розроблення вебзастосунку використано модульний принцип організації проєкту, який передбачає розділення інформаційного наповнення, шаблонів оформлення та стилів між окремими компонентами системи.

Основою реалізації став генератор статичних сайтів Hugo, який автоматично формує HTML-сторінки на основі шаблонів та текстових файлів у форматі Markdown. Такий підхід дозволяє значно спростити процес адміністрування сайту та забезпечує швидке оновлення інформації [17].

Структура проєкту містить декілька основних каталогів:

- content/ – інформаційне наповнення сторінок;
- layouts/ – HTML-шаблони відображення;
- static/ – статичні файли (зображення, логотипи);
- themes/ – тема оформлення;
- config.toml – файл конфігурації проєкту.

Розділення компонентів дозволяє незалежно змінювати дизайн, текстове наповнення та службові параметри без внесення змін до інших частин проєкту.

Для генерації сторінок використовується шаблонізація Hugo, що забезпечує повторне використання однакових елементів інтерфейсу, таких як шапка сайту, меню навігації та нижній колонтитул. На рисунку 3.3 представлено структуру директорії проєкту Hugo.

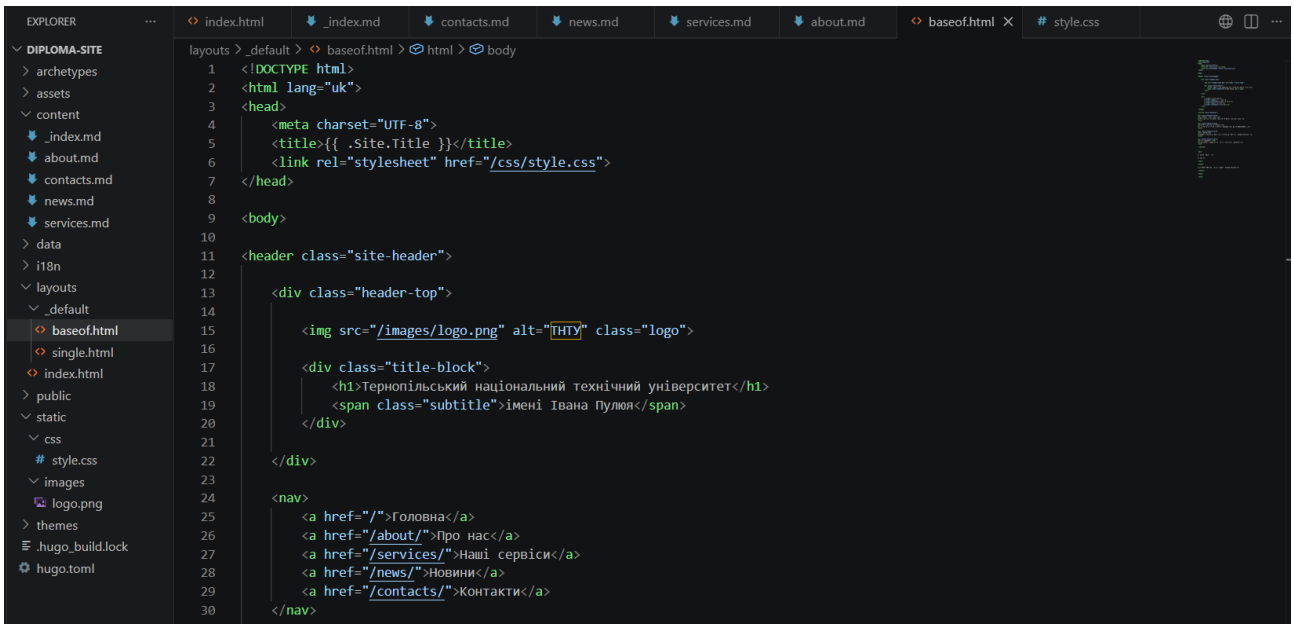


Рисунок 3.3 – Структура директорій проєкту Hugo

3.3 Реалізація сторінки «Про університет»

Для надання інформації про навчальний заклад створено окрему сторінку «Про університет». Вона містить короткий опис діяльності університету, основні напрями роботи та місію закладу вищої освіти.

Інформація структурована за тематичними блоками, що покращує її сприйняття користувачем. Заголовки виділено більшим шрифтом та корпоративним кольором, а текст подано у логічній послідовності [18].

На сторінці наведено перелік основних напрямів діяльності університету, серед яких:

- освітня діяльність;
- наукові дослідження;
- міжнародне співробітництво;
- розвиток інноваційних технологій;
- підтримка студентського самоврядування.

Окремим блоком представлено місію університету, яка полягає у підготовці висококваліфікованих фахівців шляхом поєднання сучасної освіти, науки та практичної діяльності.

При реалізації сторінки використовувалися семантичні HTML-елементи, що позитивно впливає на доступність та індексацію вебзастосунку пошуковими системами. На рисунку 3.2 представлено сторінку про університет

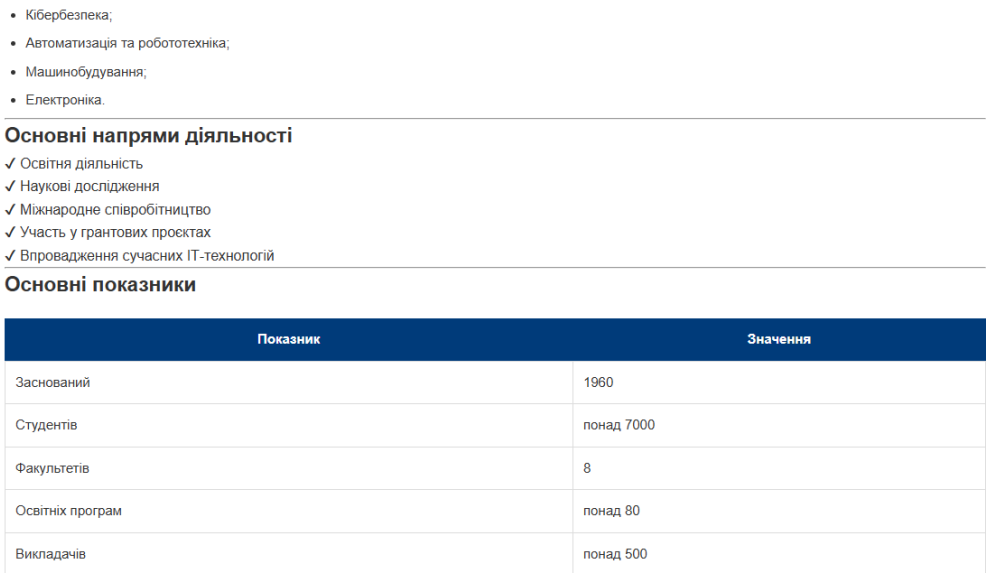


Рисунок 3.4 – Сторінка «Про університет»

На рисунку 3.3 наведено реалізовану сторінку «Про університет», яка містить структуровану інформацію про діяльність закладу освіти та демонструє використання єдиного стилю оформлення вебзастосунку.

3.4 Реалізація сторінки «Наші сервіси»

Для демонстрації цифрової інфраструктури університету реалізовано окрему сторінку «Наші сервіси». Вона містить інформацію про основні електронні ресурси, що використовуються студентами, викладачами та співробітниками.

Інформація представлена у вигляді таблиці, яка складається з двох стовпців: назви сервісу та його призначення.

До переліку включено такі сервіси:

- Atutor;
- електронна бібліотека;
- корпоративна пошта;
- розклад занять;
- репозиторій наукових праць.

Використання табличного представлення дозволяє швидко знайти потрібну інформацію та забезпечує її зручне сприйняття.

Застосування CSS-стилізації дозволило оформити таблицю відповідно до загального дизайну вебзастосунку, а також забезпечити однаковий стиль усіх сторінок сайту. На рисунку 3.4 представлено сервіси вебсайту.

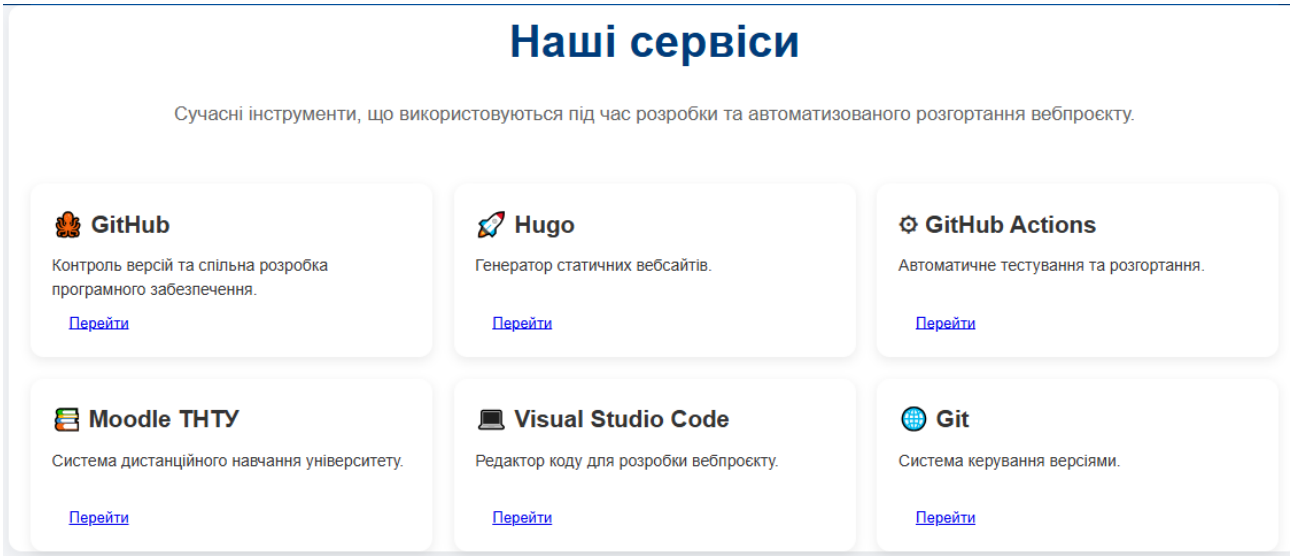


Рисунок 3.4 - Сторінка «Наші сервіси»

На рисунку 3.4 представлено сторінку із переліком електронних сервісів університету, яка демонструє використання структурованого подання інформації та єдиного стилю оформлення.

3.5 Реалізація сторінки новин

Однією з функціональних можливостей вебзастосунку є публікація актуальної інформації у вигляді новин.

Для цього створено окремий розділ, у якому кожна новина представлена окремим інформаційним блоком із заголовком та коротким описом.

Такий підхід дозволяє легко оновлювати інформацію шляхом додавання нових Markdown-файлів без необхідності редагування HTML-коду [16].

Після внесення змін Hugo автоматично генерує нову версію сторінки, а GitHub Actions виконує її публікацію на сервері [11,15].

Таким чином процес оновлення новин повністю автоматизований і не потребує ручного втручання адміністратора сайту.

Подібний механізм особливо ефективний для інформаційних вебзастосуноків, де регулярно публікуються нові матеріали. На рисунку 3.5 представлено сторінку новини.

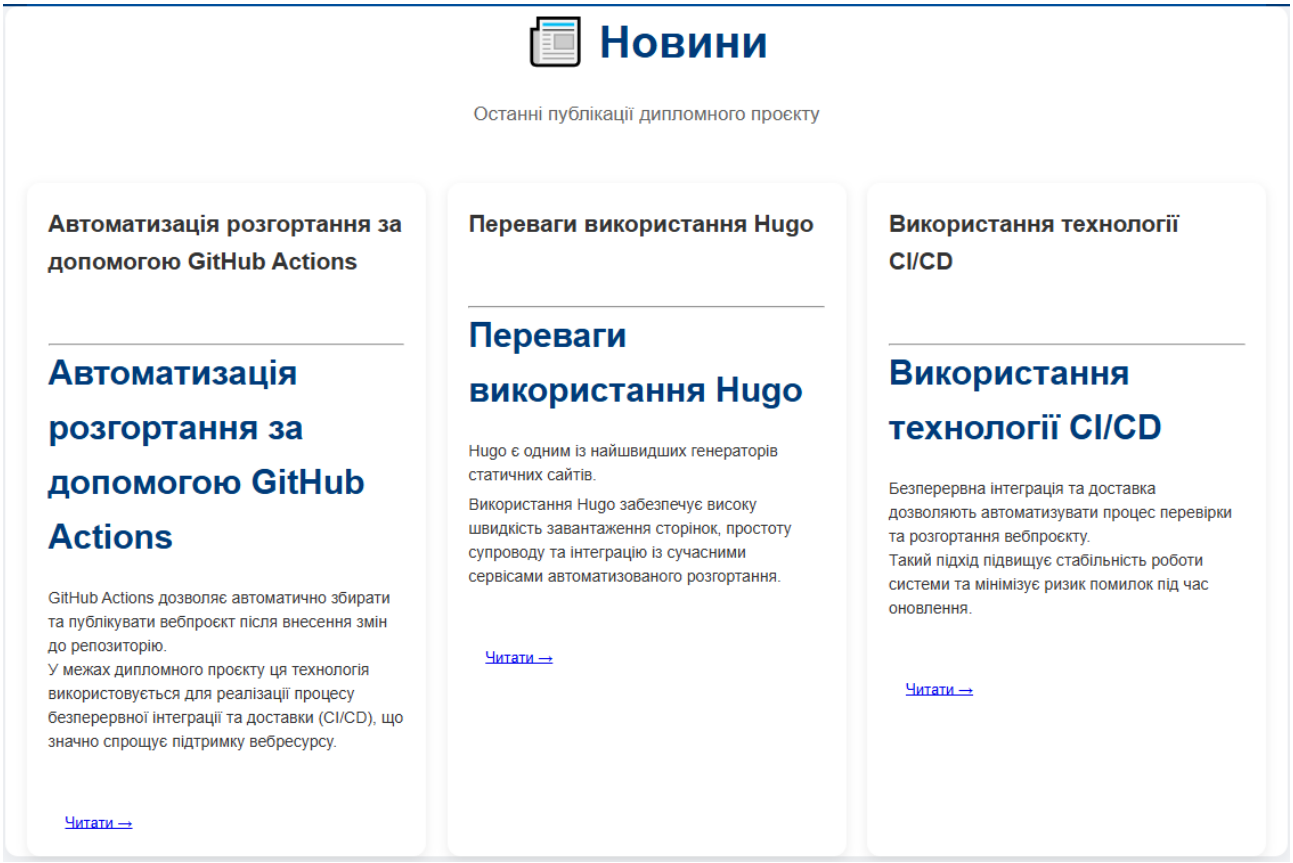


Рисунок 3.5 - Сторінка «Новини»

На рисунку 3.5 показано реалізацію сторінки новин, яка забезпечує відображення актуальної інформації та демонструє можливість швидкого оновлення контенту за допомогою генератора Hugo.

3.6 Реалізація сторінки контактів

Для забезпечення можливості комунікації між користувачами та адміністрацією університету реалізовано окрему сторінку контактної інформації.

На сторінці розміщено основні контактні дані:

- адресу;
- номер телефону;
- електронну пошту;
- офіційний вебсайт;
- графік роботи.

Інформація подана у простій та зрозумілій формі, що дозволяє швидко знайти необхідні контактні відомості.

Використання окремої сторінки для контактної інформації відповідає сучасним принципам побудови вебзастосуноків та покращує зручність користування сайтом.

Єдине стилістичне оформлення забезпечує гармонійний зовнішній вигляд сторінки та підтримує загальну концепцію дизайну всього вебзастосунку. На рисунку 3.6 представлено інтерфейс контактної сторінки.

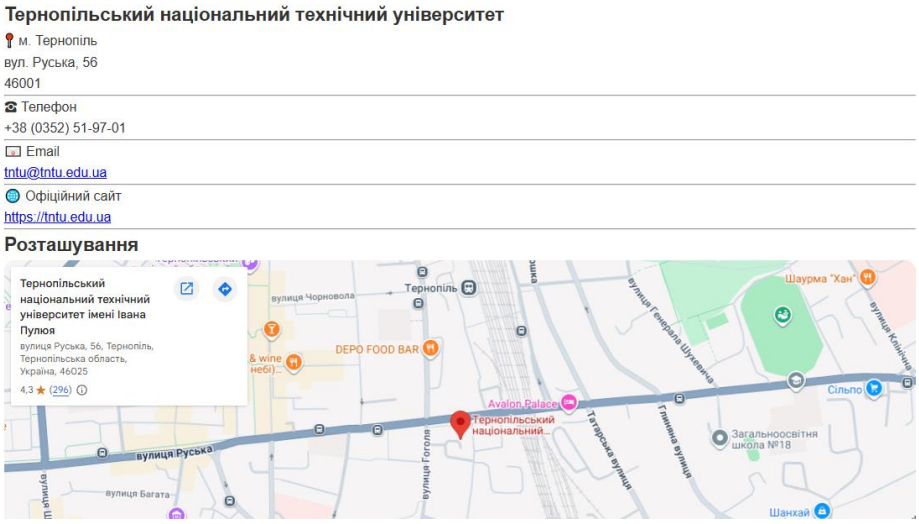


Рисунок 3.6 – Сторінка «Контакти»

На рисунку 3.6 представлено сторінку контактної інформації, яка містить основні засоби зв'язку з університетом та забезпечує користувачам швидкий доступ до необхідних контактних даних.

3.7 Автоматизація розгортання вебзастосунку

Після завершення розроблення вебзастосунку було проведено тестування його функціональних можливостей з метою перевірки коректності роботи всіх реалізованих компонентів та підтвердження відповідності вимогам [13].

У процесі тестування перевірялася працездатність навігаційного меню, правильність відображення інформації на сторінках, функціонування інформаційних блоків, таблиць та гіперпосилань. Також виконано перевірку коректності роботи вебзастосунку після внесення змін до вихідного коду та його автоматичної збірки за допомогою генератора Hugo.

Особливу увагу приділено перевірці механізму автоматизованого оновлення вебсайту.

Для перевірки працездатності вебзастосунку було проведено функціональне тестування основних розділів сайту. Результати тестування наведено у таблиці 3.1.

Таблиця 3.1 – Результати функціонального тестування вебзастосунку

Тестова операція	**Очікуваний результат**	**Результат**
Відкриття головної сторінки	Сторінка завантажується без помилок	Успішно
Перехід до сторінки «Про університет»	Коректне відображення інформації	Успішно
Перехід до сторінки «Наші сервіси»	Відображення сервісів	Успішно
Перегляд сторінки контактів	Відображення контактної інформації	
Перегляд сторінки новин	Коректне відображення новин	Успішно

Після внесення змін до Markdown-файлів виконувалася повторна генерація статичних сторінок, що дозволило переконатися у правильності роботи шаблонів та відображення оновленої інформації

3.8 Тестування вебзастосунку

Однією з основних переваг реалізованого вебзастосунку є використання засобів автоматизації процесів тестування та розгортання програмного забезпечення. Для реалізації цих процесів було використано систему контролю версій Git, платформу GitHub та сервіс GitHub Actions, які забезпечують централізоване зберігання вихідного коду, контроль змін і автоматичне виконання необхідних операцій після оновлення проєкту.

Автоматизація процесу тестування дозволяє перевіряти коректність роботи вебсайту після внесення будь-яких змін до вихідного коду або інформаційного наповнення. Після завантаження змін до репозиторію автоматично запускається робочий процес GitHub Actions. На початковому етапі здійснюється отримання актуальної версії проєкту та встановлення необхідного програмного забезпечення для роботи генератора статичних сайтів Hugo.

Наступним етапом є автоматична збірка вебзастосунку. Під час виконання збірки система генерує HTML-сторінки, таблиці стилів, скрипти та інші файли, необхідні для функціонування вебсайту. У разі виникнення помилок процес автоматично припиняється, що дозволяє запобігти розгортанню некоректної версії програмного продукту.

З метою реалізації автоматизованого тестування у робочому процесі було додано набір перевірок результатів збірки. Після генерації вебсайту виконується перевірка наявності головної сторінки index.html, а також контроль успішного створення HTML-файлів у каталозі результатів збірки. Такі перевірки дозволяють автоматично виявляти помилки, пов’язані з некоректною генерацією сторінок, пошкодженням структури проєкту або помилками конфігурації.

Використання автоматизованих перевірок значно підвищує надійність процесу розробки, оскільки дозволяє виявляти потенційні проблеми ще до етапу публікації вебзастосунку. Якщо хоча б одна перевірка завершується невдало, процес розгортання автоматично скасовується, а розробник отримує повідомлення про помилку. Таким чином забезпечується додатковий контроль якості програмного забезпечення.

Після успішного завершення всіх етапів тестування автоматично запускається процес розгортання вебсайту. Для цього використовується сервіс GitHub Pages, який забезпечує публікацію згенерованої версії вебзастосунку без необхідності ручного копіювання файлів на сервер. Завдяки цьому оновлення сайту відбувається швидко та без додаткового втручання користувача. На рисунку 3.8 зображено структуру репозиторію вебсайту на платформі GitHub, що містить основні каталоги та файли проєкту Hugo, зокрема шаблони сторінок,статичні ресурси, контент сайту та конфігураційний файл hugo.toml. На рисунку 3.8 представлено опублікований сайт на GitHub Pages.

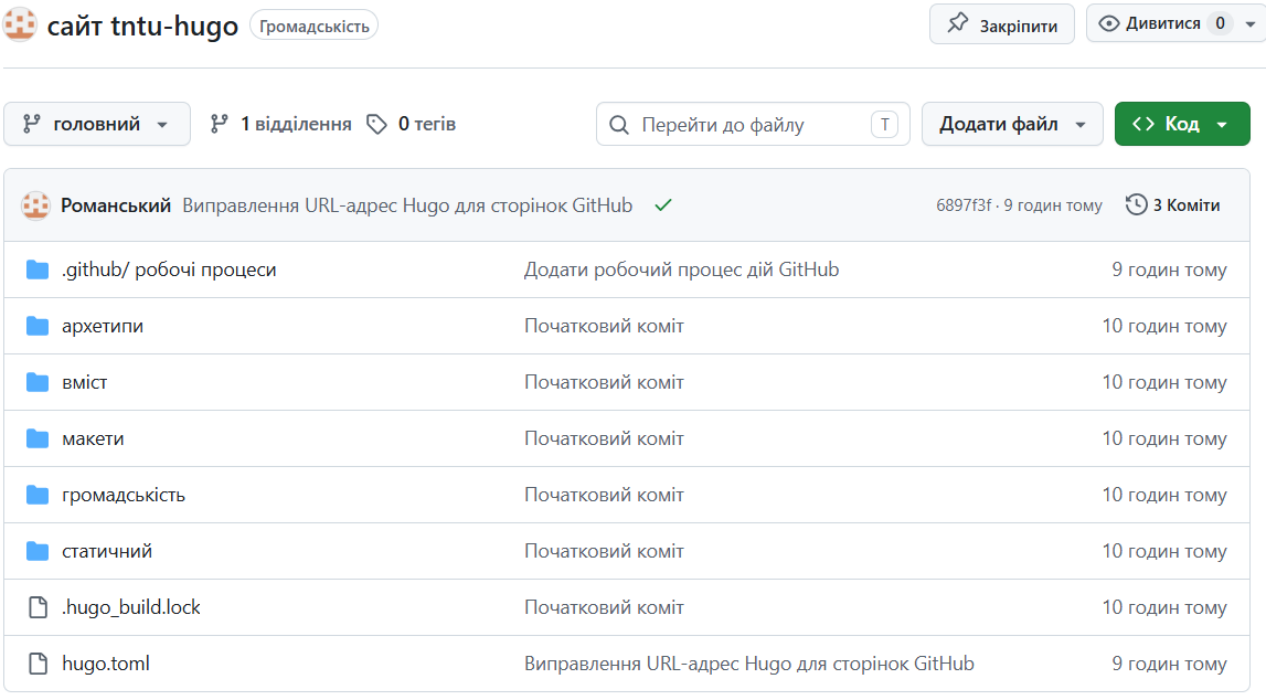


Рисунок 3.8 – Опублікований сайт на GitHub Pages.

На рисунку 3.8 зображено структуру репозиторію вебсайту на платформі GitHub, що містить основні каталоги та файли проєкту Hugo, зокрема шаблони сторінок,статичні ресурси, контент сайту та конфігураційний файл hugo.toml.

Під час практичної реалізації було проведено тестування роботи автоматизованого конвеєра шляхом внесення змін до інформаційного наповнення вебсайту та подальшого завантаження оновлених файлів до репозиторію. Після цього автоматично виконувалися збірка проєкту, перевірка результатів генерації сторінок і розгортання нової версії вебзастосунку. Усі перевірки були успішно пройдені, що підтвердило правильність роботи реалізованого механізму автоматизованого тестування та розгортання. На рисунку 3.9 представлено результат виконання автоматизованого тестування та розгортання вебсайту засобами GitHub Actions.

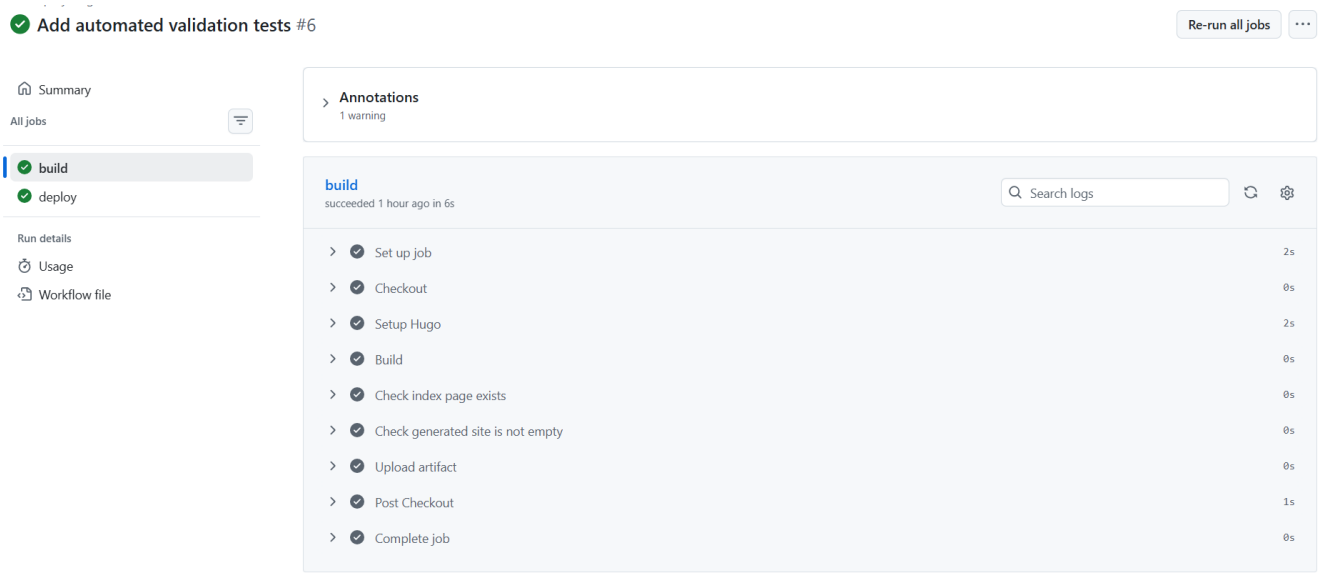


Рисунок 3.9 – Результат виконання автоматизованого тестування та розгортання вебсайту засобами GitHub Actions

Отримані результати підтверджують ефективність використання GitHub Actions для реалізації процесів безперервної інтеграції та доставки (CI/CD). Використання автоматизованого тестування дозволило підвищити надійність

вебзастосунку, скоротити час оновлення сайту та мінімізувати ризик виникнення помилок, пов’язаних із людським фактором. Реалізований підхід забезпечує стабільну роботу програмного продукту та відповідає сучасним практикам розробки вебзастосунків.

3.9 Перевірка адаптивності та працездатності вебзастосунку

Для забезпечення комфортної роботи користувачів вебзастосунок було перевірено на різних роздільних здатностях екрана та в сучасних браузерах.

Під час тестування встановлено, що сторінки коректно відображаються у браузерах Google Chrome, Microsoft Edge та Mozilla Firefox без втрати функціональності.

Завдяки використанню CSS та адаптивної верстки всі інформаційні блоки автоматично перебудовуються відповідно до ширини екрана. Навігаційне меню, картки переваг та основний контент зберігають правильне форматування як на широкоформатних моніторах, так і на ноутбуках.

Також проведено перевірку швидкості завантаження сторінок. Оскільки вебсайт побудовано як статичний ресурс за допомогою Hugo, його сторінки завантажуються практично миттєво без необхідності виконання серверних запитів до баз даних.

Використання статичної архітектури позитивно впливає на продуктивність, безпеку та стійкість до великих навантажень. На рисунку 3.9 відображається перевірка коректного відображення вебзастосунку у браузері та інших пристроях.

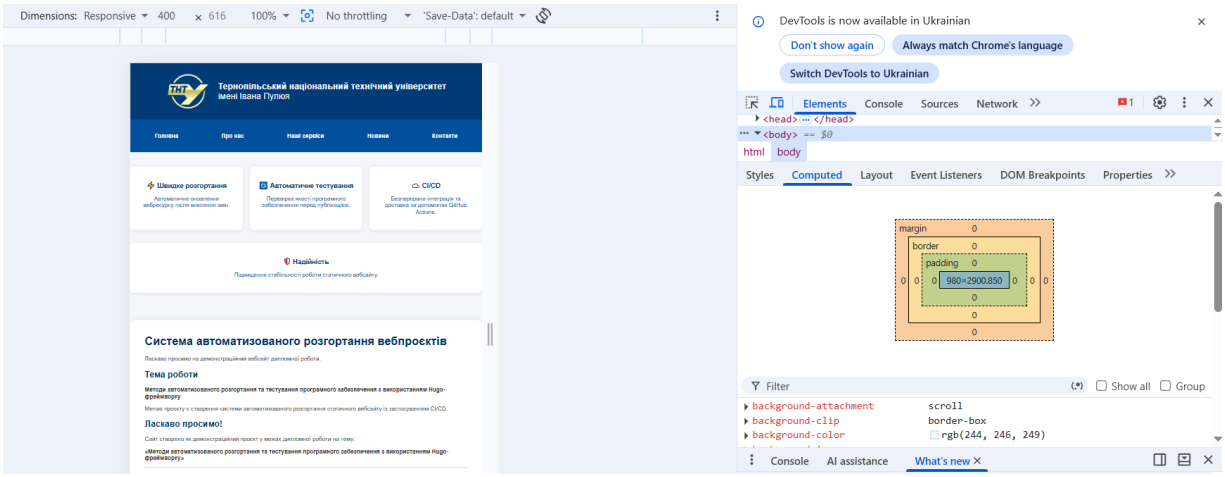


Рисунок 3.9 – Перевірка коректного відображення вебзастосунку у браузері

У результаті виконання третього розділу було реалізовано структуру демонстраційного вебсайту з використанням генератора статичних сайтів Hugo. Розроблено головну сторінку та окремі інформаційні розділи, що містять відомості про університет, електронні сервіси, новини та контактну інформацію.

Особливістю реалізованого рішення є інтеграція механізму автоматичного розгортання на основі GitHub Actions, що забезпечує автоматичну збірку та публікацію вебзастосунку після внесення змін до репозиторію.

Проведене тестування підтвердило коректність роботи всіх функціональних компонентів системи, правильність навігації та високу швидкість завантаження сторінок. Отриманий вебзастосунок відповідає поставленим вимогам та демонструє ефективність використання сучасних технологій автоматизації розгортання статичних вебпроєктів [17,19].

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Під час розроблення, тестування та автоматизованого розгортання програмного забезпечення важливим аспектом є забезпечення безпечних умов праці та дотримання вимог безпеки життєдіяльності [21]. Діяльність інженера-програміста пов'язана з тривалою роботою за персональним комп'ютером, використанням сучасних інформаційних технологій та програмних засобів, що може супроводжуватися впливом низки небезпечних і шкідливих виробничих факторів [22].

До основних факторів, які можуть негативно впливати на стан здоров'я працівника, належать статичне навантаження, тривале перебування у сидячому положенні, значне навантаження на органи зору, психоемоційне перенапруження, а також небезпека ураження електричним струмом під час експлуатації комп'ютерної техніки. Саме тому організація безпечного робочого середовища є одним із важливих завдань під час виконання професійної діяльності у сфері інформаційних технологій.

У цьому розділі розглянуто питання моделювання та прогнозування небезпечних ситуацій, а також основні вимоги охорони праці для користувачів персональних комп'ютерів, що безпосередньо стосуються виконання дипломної роботи на тему автоматизованого розгортання та тестування програмного забезпечення з використанням Hugo-фреймворку [25].

4.1 Моделювання та прогнозування небезпечних ситуацій

Одним із важливих напрямів забезпечення безпеки життєдіяльності є моделювання та прогнозування небезпечних ситуацій. Використання сучасних методів аналізу ризиків дозволяє своєчасно виявляти потенційні загрози, оцінювати можливі наслідки їх виникнення та розробляти комплекс профілактичних заходів, спрямованих на запобігання аваріям і нещасним випадкам [21].

Моделювання небезпечних ситуацій являє собою процес створення логічної або математичної моделі розвитку потенційно небезпечної події. За допомогою такого підходу можна визначити фактори, які впливають на виникнення ризиків, оцінити їх взаємозв'язок та спрогнозувати можливі наслідки для працівників і функціонування інформаційної системи [25].

У сфері інформаційних технологій потенційні небезпечні ситуації можуть бути пов'язані як із технічними несправностями обладнання, так і з програмними помилками або діями користувача. До найбільш поширених ризиків належать відмова серверного обладнання, пошкодження даних, помилки конфігурації програмного забезпечення, збій процесу автоматизованого розгортання, порушення цілісності програмного коду та несанкціонований доступ до інформаційних ресурсів.

Під час виконання дипломної роботи використовується система автоматизованого розгортання вебзастосунку на базі Hugo та GitHub Actions. Такий підхід дозволяє значно зменшити вплив людського фактора та мінімізувати кількість помилок, які можуть виникати при ручному оновленні вебсайту. Кожна зміна програмного коду проходить автоматичну перевірку, після чого виконується генерація статичного вебзастосунку та його публікація. У випадку виявлення помилки процес розгортання автоматично припиняється, що унеможливлює публікацію некоректної версії програмного продукту.

Важливим елементом прогнозування небезпечних ситуацій є постійний моніторинг стану системи. Аналіз журналів виконання автоматизованих процесів дозволяє своєчасно виявити відхилення від нормального режиму роботи, локалізувати проблему та оперативно вжити заходів щодо її усунення. Застосування автоматичного тестування програмного забезпечення перед розгортанням також сприяє підвищенню надійності інформаційної системи та зменшує ризик виникнення програмних помилок.

Особливе значення має оцінювання ризиків, пов'язаних із кібербезпекою. Несанкціонований доступ до репозиторію, компрометація облікових записів або модифікація програмного коду можуть призвести до порушення працездатності

вебзастосунку. Для запобігання таким ситуаціям використовуються механізми автентифікації, контроль доступу та резервне копіювання інформації.

Таким чином, застосування методів моделювання та прогнозування небезпечних ситуацій дозволяє підвищити рівень надійності програмного забезпечення, забезпечити стабільність роботи автоматизованої системи розгортання та своєчасно реагувати на можливі загрози. Використання сучасних засобів автоматизації значно зменшує ймовірність виникнення аварійних ситуацій та сприяє безпечній експлуатації програмного продукту.

4.2 Загальні вимоги безпеки з охорони праці для користувачів персональних комп'ютерів

Професійна діяльність розробників програмного забезпечення нерозривно пов'язана з використанням персональних комп'ютерів та інших електронних пристроїв. Тривала робота за комп'ютером супроводжується впливом факторів, які можуть негативно позначатися на здоров'ї працівника, тому організація безпечного робочого місця є одним із основних напрямів охорони праці.

Робоче місце користувача персонального комп'ютера повинно відповідати вимогам ергономіки та забезпечувати комфортні умови виконання професійних обов'язків[23]. Конструкція робочого столу і крісла повинна забезпечувати правильне положення тіла працівника, зменшувати статичне навантаження на хребет і сприяти підтриманню природної пози під час роботи [26].

Монітор рекомендується розташовувати безпосередньо перед користувачем на відстані приблизно 50–70 сантиметрів від очей. Верхня межа екрана повинна знаходитися на рівні очей або трохи нижче, що дозволяє знизити навантаження на шийний відділ хребта та органи зору. Клавіатура і маніпулятор «миша» повинні розміщуватися таким чином, щоб забезпечувати природне положення кистей рук та мінімізувати ризик розвитку професійних захворювань опорно-рухового апарату.

Особливу увагу необхідно приділяти параметрам виробничого середовища. Робоче приміщення повинно мати достатній рівень природного або штучного

освітлення, ефективну систему вентиляції та оптимальні показники температури і вологості повітря. Недостатнє освітлення або несприятливий мікроклімат можуть призводити до швидкої втоми, погіршення концентрації уваги та зниження продуктивності праці [24].

Під час роботи за персональним комп'ютером рекомендується дотримуватися раціонального режиму праці та відпочинку. Через певні проміжки часу необхідно робити короткі перерви, виконувати вправи для очей та змінювати положення тіла. Такі заходи сприяють профілактиці перевтоми, покращують кровообіг та зменшують ризик виникнення професійних захворювань.

Одним із важливих аспектів охорони праці є електробезпека. Уся комп'ютерна техніка повинна експлуатуватися лише за умови її справності та відповідності технічним вимогам. Забороняється використовувати пошкоджені кабелі живлення, несправні подовжувачі або виконувати ремонт обладнання без відповідної підготовки. У разі виявлення несправностей працівник повинен негайно припинити експлуатацію обладнання та повідомити відповідальну особу [29].

Не менш важливими є вимоги пожежної безпеки. Робоче місце повинно утримуватися у належному стані, без накопичення легкозаймистих матеріалів та захаращення проходів. Електрообладнання після завершення роботи необхідно вимикати, а працівники повинні знати порядок дій у разі виникнення пожежі та місце розташування первинних засобів пожежогасіння.

Під час виконання дипломної роботи всі етапи розроблення вебзастосунку, написання програмного коду, тестування та автоматизованого розгортання здійснювалися з дотриманням вимог охорони праці. Робоче місце було організовано відповідно до ергономічних вимог, забезпечено належне освітлення та вентиляцію, а також дотримано раціонального режиму праці й відпочинку. Використання справного обладнання та сучасних засобів автоматизації дозволило створити безпечні умови праці та підвищити ефективність виконання поставлених завдань.

ВИСНОВКИ

У дипломній роботі досліджено методи автоматизованого розгортання та тестування програмного забезпечення з використанням генератора статичних вебсайтів Hugo. Актуальність обраної теми обумовлена широким впровадженням сучасних технологій безперервної інтеграції та доставки програмного забезпечення, які дозволяють автоматизувати процес розроблення, тестування та оновлення вебзастосуноків, підвищуючи їх надійність і ефективність.

У процесі виконання роботи було проаналізовано сучасні підходи до автоматизованого розгортання програмного забезпечення, досліджено принципи функціонування CI/CD, особливості використання генератора статичних сайтів Hugo та його переваги порівняно з традиційними системами керування контентом. Проведений аналіз показав, що застосування статичних генераторів сайтів дозволяє суттєво підвищити швидкодію вебзастосуноків, покращити рівень їхньої безпеки та значно спростити процес адміністрування.

У практичній частині дипломної роботи було реалізовано демонстраційний вебсайт, структура якого відтворює основні інформаційні сторінки офіційного вебзастосунку Тернопільського національного технічного університету імені Івана Пулюя. Для створення вебзастосунку використано можливості Hugo, HTML5, CSS3 та шаблонізатора Go Templates, що дозволило сформувати сучасний інтерфейс із єдиним стилем оформлення та адаптивною структурою.

Особливу увагу приділено автоматизації процесу розгортання вебпроєкту. Використання GitHub Actions забезпечило автоматичне виконання збірки та публікації сайту після внесення змін до репозиторію, що дозволило мінімізувати вплив людського фактора, скоротити час оновлення вебзастосунку та підвищити стабільність його функціонування. Також було розглянуто особливості автоматизованого тестування, яке дає можливість своєчасно виявляти програмні помилки та забезпечувати високу якість кінцевого програмного продукту.

У четвертому розділі досліджено питання охорони праці та безпеки життєдіяльності під час роботи інженера-програміста. Розглянуто методи

моделювання небезпечних ситуацій, а також основні вимоги щодо організації безпечного робочого місця користувача персонального комп'ютера. Виконаний аналіз підтверджує важливість дотримання вимог охорони праці для забезпечення безпечних умов професійної діяльності та збереження здоров'я працівників.

Отримані результати свідчать про ефективність використання сучасних засобів автоматизації процесів розроблення, тестування та розгортання вебзастосуноків. Поставлену мету дипломної роботи досягнуто, а визначені завдання виконано у повному обсязі. Розроблений вебсайт та реалізований механізм автоматизованого розгортання можуть бути використані як приклад впровадження сучасних DevOps-підходів у процес створення та супроводу статичних вебпроєктів, а також у навчальному процесі під час підготовки фахівців у галузі інженерії програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів спеціальності 121 – Інженерія програмного забезпечення / укладачі: Михалик Д.М., Цуприк Г.Б., Бревус В.М. – Тернопіль : ТНТУ імені Івана Пулюя, 2024. – 45 с.

2. Sommerville I. Software Engineering [Електронний ресурс]. – Режим доступу до ресурсу: https://www.pearson.com/en-us/subject-catalog/p/software-engineering/P200000003462;

3. Jacobson I., Booch G., Rumbaugh J. The Unified Software Development Process. Boston : Addison-Wesley, 1999. – 463 p.

4. Петрик М.Р., Мудрик І.Я. Архітектура програмного забезпечення (на базі використання CASE-засобів IBM Rational Software Architect): навчальний посібник. Тернопіль : ТНТУ імені Івана Пулюя, 2017. – 100 с.

5. OMG Unified Modeling Language (UML) Specification [Електронний ресурс]. – Режим доступу до ресурсу: https://www.omg.org/spec/UML/2.5.1/;

6. Діаграма класів UML [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Діаграма_класів;

7. Діаграма послідовності. URL: http://flash.retejo.info/cxefpagxo/uml/ diagrama-poslidovnosti

8. Патерни проєктування: каталог патернів [Електронний ресурс]. – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns;

9. Патерн «Спостерігач» [Електронний ресурс]. – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/observer;

10. Патерн «Одинак» [Електронний ресурс]. – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/singleton;

11. Hugo Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://gohugo.io/documentation/;

12. Hugo Quick Start Guide [Електронний ресурс]. – Режим доступу до ресурсу: https://gohugo.io/getting-started/quick-start/;

13. Git Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://git-scm.com/doc;

14. GitHub Docs [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.github.com/;

15. GitHub Actions Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.github.com/actions;

16. Markdown Guide [Електронний ресурс]. – Режим доступу до ресурсу: https://www.markdownguide.org/;

17. Bootstrap: Powerful frontend toolkit [Електронний ресурс]. – Режим доступу до ресурсу: https://getbootstrap.com/docs/5.3/getting-started/introduction/;

18. Sass Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://sass-lang.com/documentation/;

19. База знань QALight. Функціональне тестування [Електронний ресурс]. – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/funktsionalne-testuvannya/;

20. Selenium Browser Automation [Електронний ресурс]. – Режим доступу до ресурсу: https://www.selenium.dev/;

21. Желібо Є.П., Зацарний В.В. Безпека життєдіяльності : підручник. Київ : Каравела, 2023. – 344 с.

22. Атаманчук П.С. Безпека життєдіяльності : навчальний посібник. Київ : Центр учбової літератури, 2020. – 276 с.

23. Жидецький В.Ц. Охорона праці користувачів комп'ютерів : підручник. Львів : Афіша, 2020. – 176 с.

24. Закон України «Про охорону праці» [Електронний ресурс]. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12;

25. Вимоги до планування та обладнання робочих місць. URL: https://pidru4niki.com/2015082666293/menedzhment/vimogi_planuvannya_obladnannya_robochih_mists

26. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями».

27. ДСТУ ISO 9241-5:2004 Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 5. Вимоги до компонування робочого місця та робочої пози.

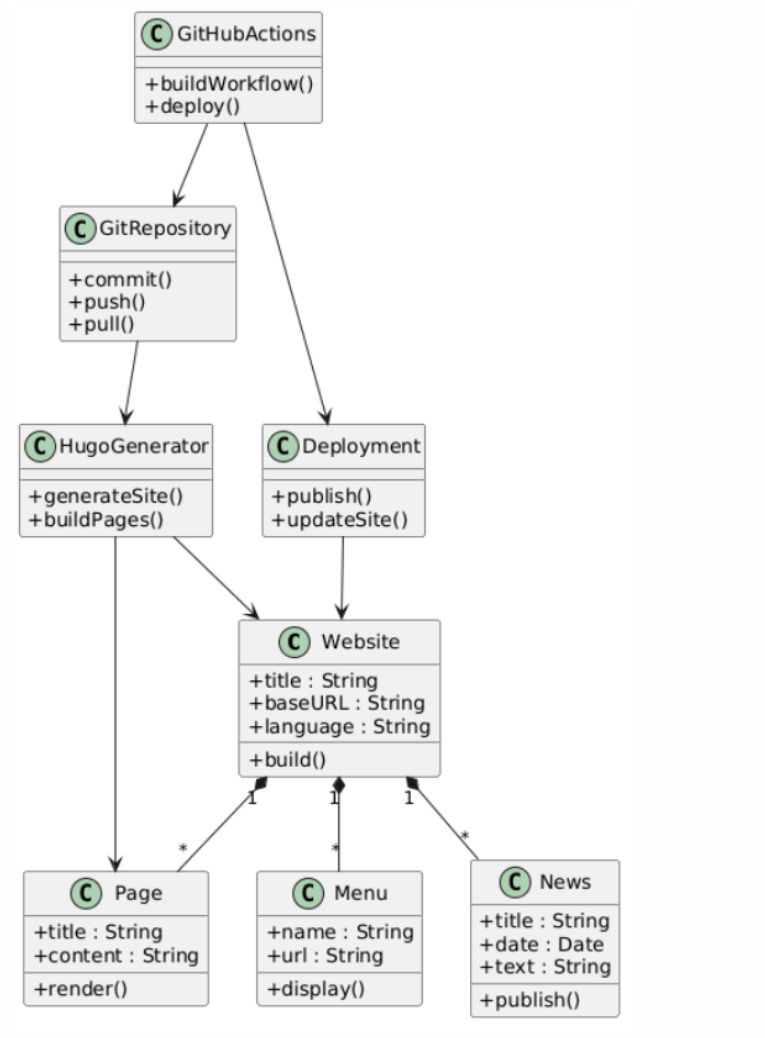
28. Андрейчук Н.І., Кіт Ю.В., Шибанов С.В., Шерстньова О.В. Охорона праці : навчальний посібник. Львів : Видавництво Львівської політехніки, 2021. – 276 с.

29. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) [Електронний ресурс]. – Режим доступу до ресурсу: https://www.computer.org/education/bodies-of-knowledge/software-engineering;

ДОДАТКИ

ДОДАТОК А

Діаграма класів програмного забезпечення



ДОДАТОК Б

Дизайн вебзастосунку

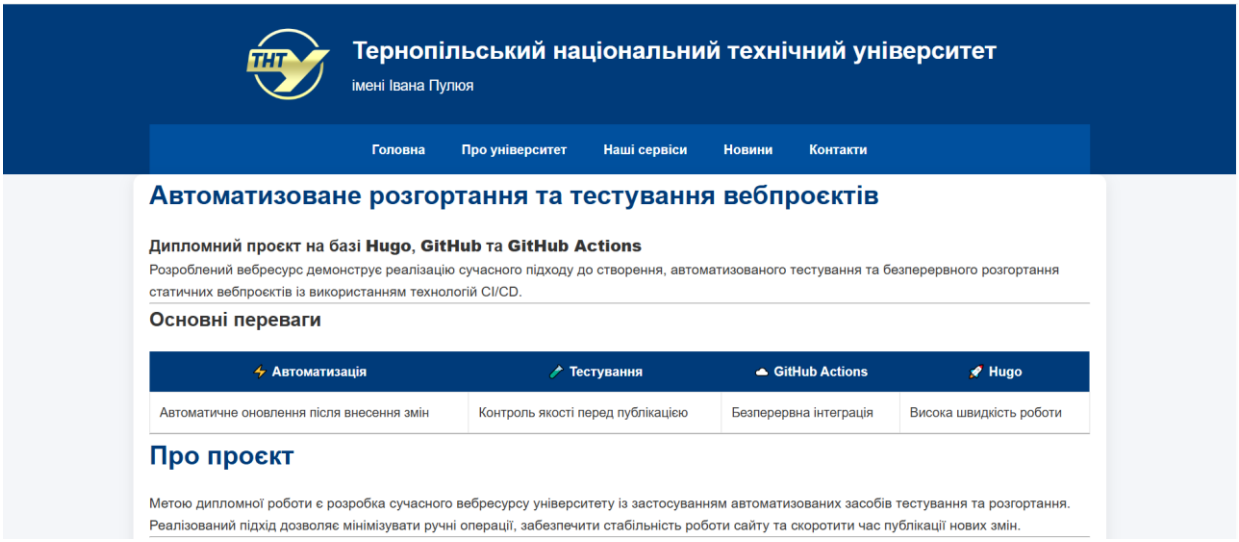


Рисунок Б.1 – Головна сторінка вебзастосунку на базі Hugo

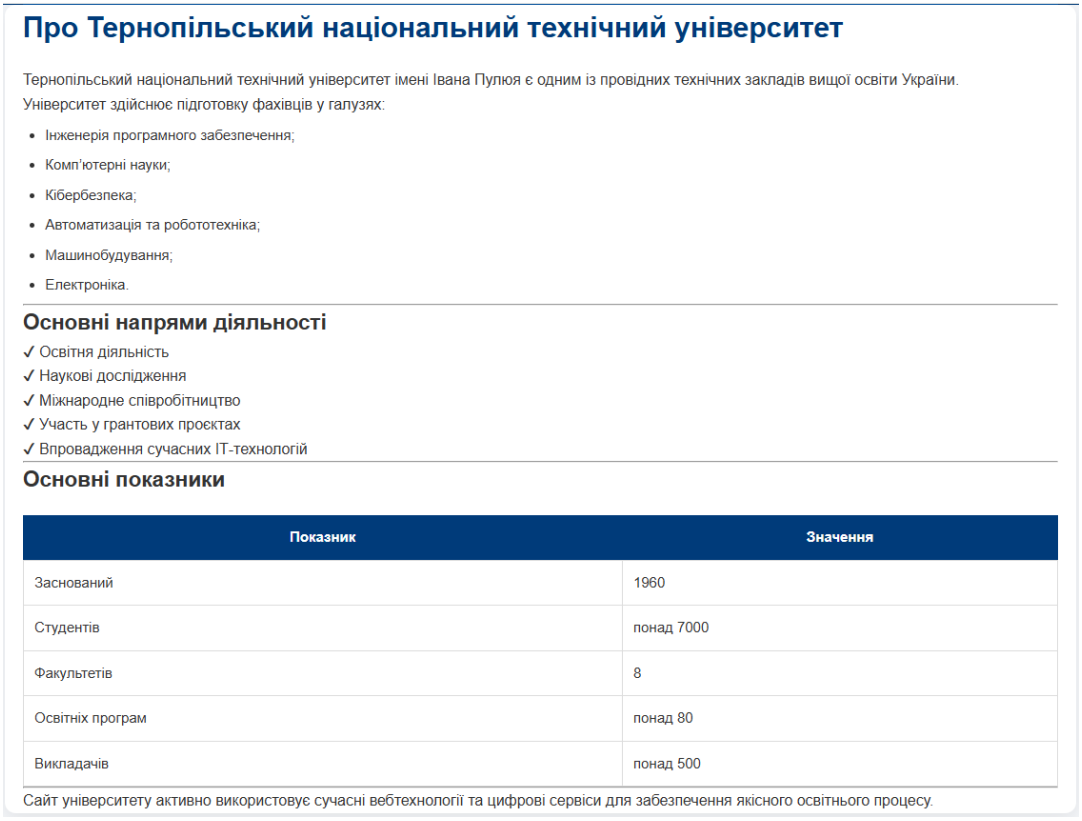


Рисунок Б.2 - Сторінка «Про університет»

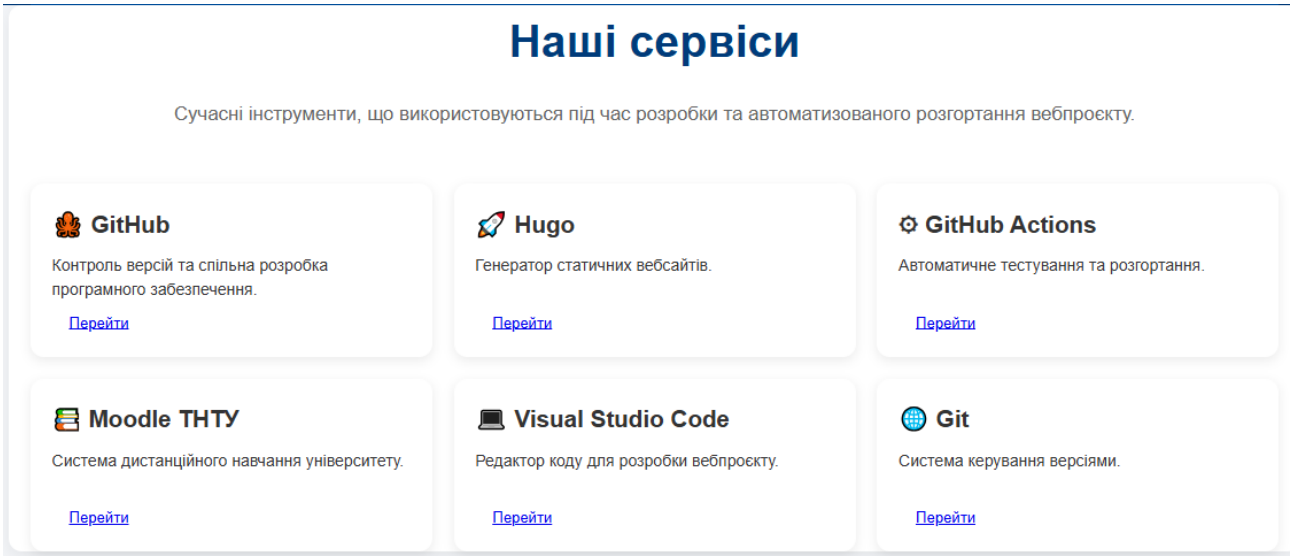


Рисунок Б.3 - Сторінка «Наші сервіси»

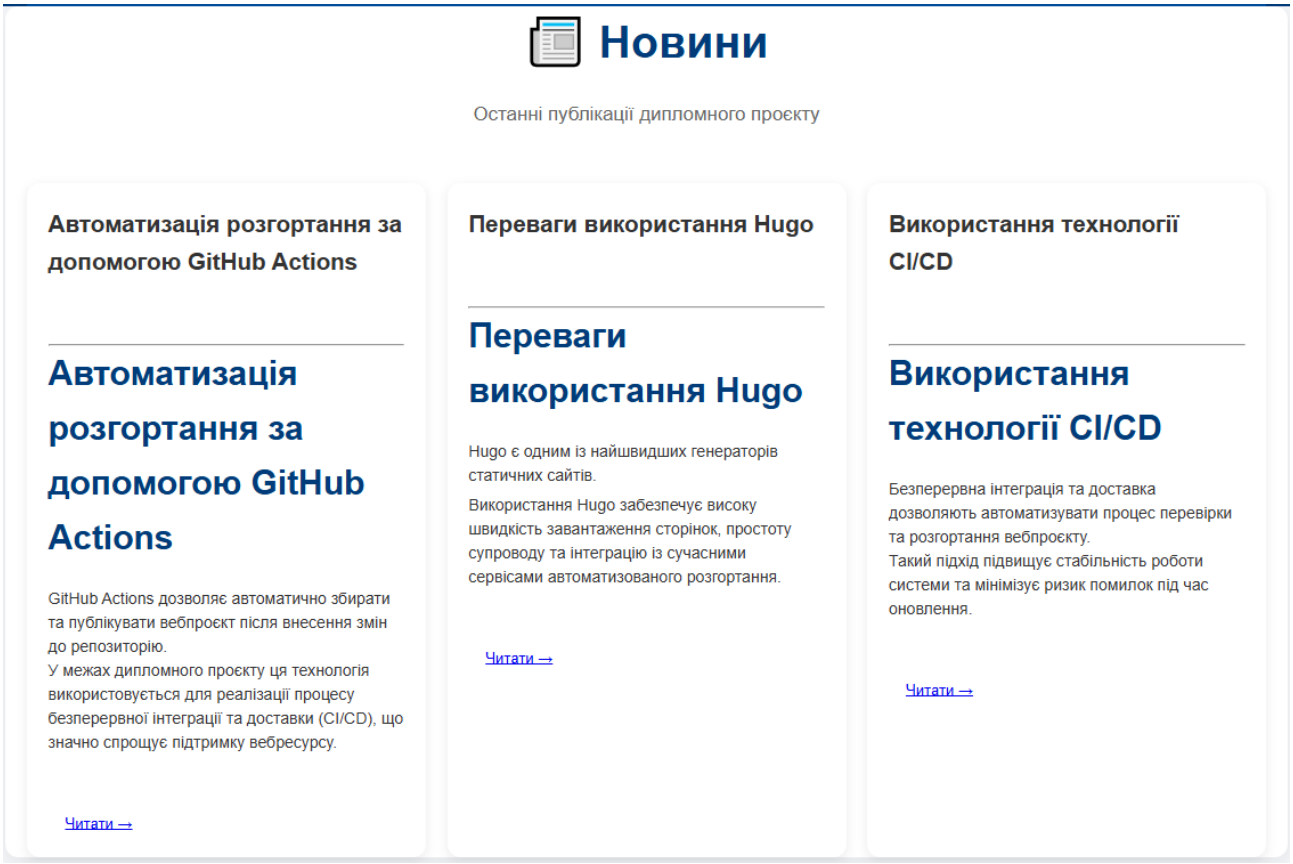


Рисунок Б.4 - Сторінка «Новини»

Тернопільський національний технічний університет

м. Тернопіль
вул. Руська, 56
46001

Телефон
+38 (0352) 51-97-01

Email
tntu@tntu.edu.ua

Офіційний сайт
https://tntu.edu.ua

Розташування

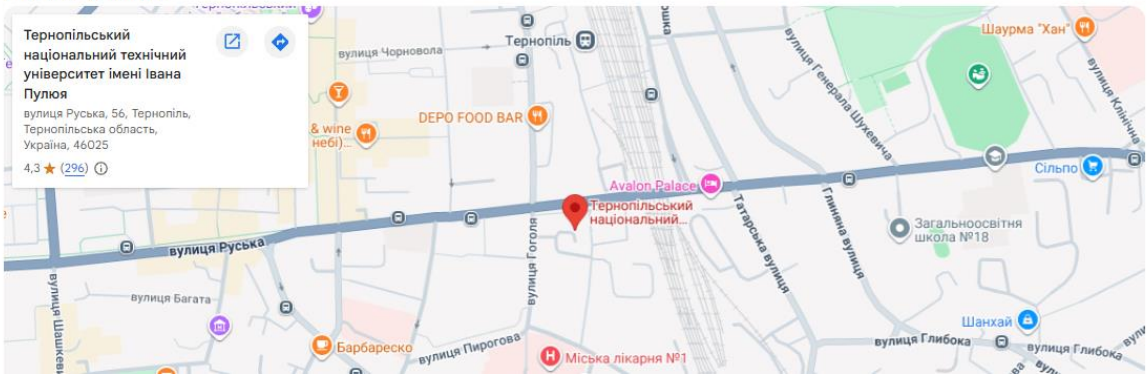


Рисунок Б.5 - Сторінка «Контакти»

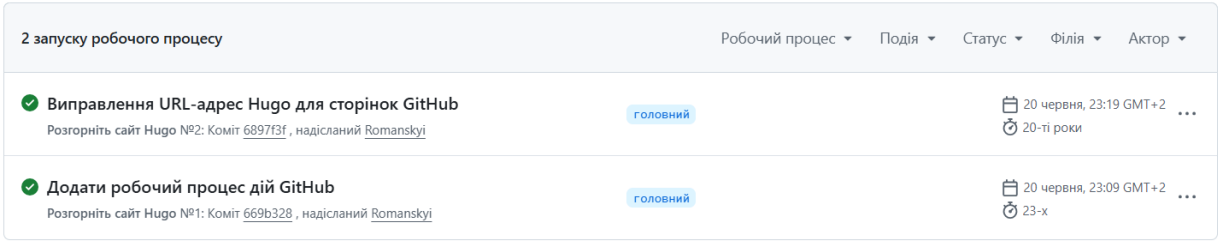


Рисунок Б.6 - Результат виконання сценарію GitHub Actions

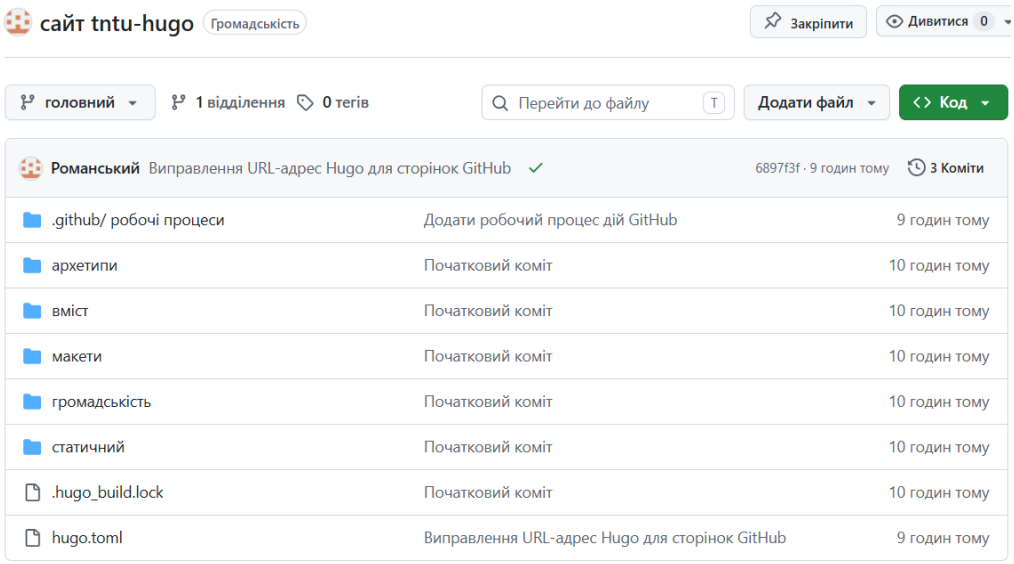


Рисунок Б.7 - Опублікований вебсайт на платформі GitHub Pages

ДОДАТОК В

Лістинг коду

Лістинг коду файлу deploy.yml для автоматизованого розгортання вебзастосунку

```
name: Deploy Hugo Site
on:
  push:
    branches:
      - main
permissions:
  contents: read
  pages: write
  id-token: write
concurrency:
  group: pages
  cancel-in-progress: true
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4
      - name: Setup Hugo
        uses: peaceiris/actions-hugo@v3
        with:
          hugo-version: latest

      - name: Build
        run: hugo
```

```
    # Автоматизовані перевірки
    - name: Check index page exists
      run: test -f public/index.html

    - name: Check generated site is not empty
      run: test "$(find public -name '*.html' | wc -l)" -gt 5

    - name: Upload artifact
      uses: actions/upload-pages-artifact@v3
      with:
        path: ./public

deploy:
  needs: build

  runs-on: ubuntu-latest

  environment:
    name: github-pages
    url: ${{ steps.deployment.outputs.page_url }}

  permissions:
    pages: write
    id-token: write
  steps:
    - name: Deploy
      id: deployment
      uses: actions/deploy-pages@v4
```

ДОДАТОК Д

Посилання на репозиторій проєкту

https://romanskyi.github.io/2025-2026-KRB-SPs-41-Vladyslav-ROMANSKYI/