МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ІНСТРУМЕНТУ ІМПОРТУ ШАБЛОНУ ЦІЛЬОВОГО САЙТУ З ВИКОРИСТАННЯМ RUBY ON RAILS»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Д.А. Герасименко
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Горбенко В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Герасименко Дмитру Андрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка інструменту імпорту шаблону цільового сайту з використанням Ruby on Rails

керівник роботи Горбенко Віталій Іванович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Програмна реалізація інструменту для імпорту шаблону цільового сайту.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	30.01.2024	
3.	Обробка методичних та теоретичних	16.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	28.03.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2024	

Студент _____ Д.А. Герасименко
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.І. Горбенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка інструменту імпорту шаблону цільового сайту з використанням Ruby on Rails»: 48 с., 27 рис., 13 джерел.

БАЗА ДАНИХ, ГЕМ, КОНТЕНТ, ПАРСИНГ, САЙТ, САЙТМАП, СЕРВЕР, ШАБЛОН, ФРЕЙМВОРК, NOKOGIRI, RUBY ON RAILS.

Об’єкт дослідження – перенесення веб ресурсів між програмними середовищами та платформами.

Предмет дослідження – реалізація перенесення шаблону цільового сайту до іншого програмного середовища (платформи) засобами Ruby on Rails.

Мета роботи: створення інструменту, що дозволяє реалізувати необхідну функціональність для перенесення шаблону цільового сайту.

Методи дослідження – аналіз предметної області та функціональності застосунку, експерименти щодо реалізації функціональності застосунку.

У кваліфікаційній роботі розглянуто створення інструменту для перенесення вебсайтів між програмними середовищами (платформами) засобами Ruby on Rails. Проведено аналіз існуючих інструментів для переносу вебзастосунків. Виконано реалізацію зручного та ефективного механізму, що дозволяє автоматизувати процес перенесення через механізм імпорту шаблону цільового сайту. Проаналізовано практичне застосування отриманих результатів для використання розробниками вебресурсів.

SUMMARY

Bachelor's qualifying paper "Development of the Import Tool of a Target Site Template Using Ruby on Rails": 48 pages, 27 figures, 13 references.

BASE DATA, CONTENT, FRAMEWORK, GEM, NOKOGIRI, PARSING, RUBY ON RAILS, SERVER, SITE, SITEMAP, TEMPLATE.

The object of the study is the transfer of web resources between software environments and platforms. The subject of the study is the implementation of transferring the template of the target site to another software environment (platform) using Ruby on Rails.

The purpose of the work: creation of a tool that allows you to implement the necessary functionality for transferring the template of the target site.

Research methods – analysis of the subject area and functionality of the application, experiments on the implementation of the functionality of the application.

The qualification paper considered the creation of a tool for transferring websites between software environments (platforms) using Ruby on Rails. An analysis of existing web application migration tools has been carried out. A convenient and effective mechanism has been implemented that allows you to automate the transfer process through the target site template import mechanism. The practical application of the obtained results for use by developers of web resources is analyzed.

ЗМІСТ

ВСТУП

У сучасному світі веброзробка є одним з найважливіших напрямків інформаційних технологій, адже вебсайти стали невід'ємною частиною бізнесу, освіти, розваг та багатьох інших сфер діяльності. Проте, розвиток технологій і постійна еволюція вебплатформ вимагають оновлення та перенесення існуючих вебсайтів на нові, більш сучасні платформи. Це створює потребу у високоякісних та ефективних інструментах для автоматизації процесу перенесення вебсайтів.

Кваліфікаційну роботу присвячено розробці інструменту для перенесення вебсайтів на платформу Ruby on Rails. Ruby on Rails є популярною веброзробницькою платформою, яка забезпечує високу продуктивність, зручність у використанні та потужні можливості для розробки складних вебдодатків. Метою даного дослідження є створення інструменту, який автоматизує процес перенесення контенту та структури вебсайту на цю платформу, забезпечуючи збереження функціональності та адаптацію під нове середовище.

У роботі розглянуто концептуальні та логічні моделі інструменту, а також детальний опис його структури. Основна увага приділяється автоматизації таких процесів, як завантаження файлу sitemap.xml, парсинг сторінок, обробці контенту, налаштування маршрутів та підтримці мультимовності. Використання сучасних бібліотек та сервісів дозволяє значно спростити та прискорити процес перенесення, зберігаючи при цьому високу якість кінцевого продукту.

Розробка інструменту для перенесення вебсайтів на платформу Ruby on Rails включає кілька ключових етапів: аналіз вимог та потреб користувачів, проєктування структури інструменту, реалізація основних функцій, тестування та впровадження. Особлива увага приділяється забезпеченню гнучкості та налаштовуваності інструменту, що дозволяє адаптувати його до різних сценаріїв використання та потреб користувачів.

Таким чином, кваліфікаційну роботу спрямовано на вирішення актуальної проблеми автоматизації процесу перенесення вебсайтів, запропоновано рішення, яке забезпечує ефективність, надійність та простоту використання. Результати кваліфікаційної роботи можуть бути корисними для веброзробників, яким необхідно виконати перенесення вебсайтів на нові платформи, а також для компаній, що прагнуть оптимізувати свої вебресурси та зберегти конкурентоспроможність.

1 ОГЛЯД СТЕКА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ІНСТРУМЕНТУ ПЕРЕНОСУ ШАБЛОНУ ЦІЛЬОВОГО ВЕБСАЙТУ З ЗАСТОСУВАННЯМ ФРЕЙМВОРКУ RUBY ON RAILS

1.1 Загальні поняття стека технологій

Загальні поняття стека технологій визначають основні складові, які будуть використані для розробки інструменту імпорту шаблону цільового вебсайту з використанням Ruby on Rails. Вибір правильних компонентів та їх інтеграція є критично важливими для досягнення успішного результату та забезпечення ефективності розробки.

Стек технологій, також відомий як технологічний стек або стек розробки, включає набір інструментів, фреймворків, мов програмування, бібліотек, сервісів та інших компонентів, які використовуються для розробки, тестування, розгортання та підтримки програмного забезпечення. Кожен елемент стека виконує свою специфічну роль і впливає на загальну архітектуру та функціональність кінцевого продукту [3].

Платформа Ruby on Rails, або скорочено Rails, є потужним вебфреймворком, побудованим на мові програмування Ruby. Вона забезпечує високу продуктивність розробки завдяки використанню парадигм «конвенції понад конфігурацією» та «неповторюваності коду» (DRY – Don’t Repeat Yourself). Rails надає розробникам зручні інструменти для швидкого створення функціональних вебдодатків, зокрема завдяки вбудованим засобам для роботи з базами даних, маршрутизації запитів, керування сеансами користувачів та іншим важливим аспектам веброзробки [10].

До основних компонентів стека технологій для розробки інструменту імпорту шаблонів на платформі Ruby on Rails входять:

– мова програмування Ruby;

- фреймворк Ruby on Rails;
- система керування базами даних (СКБД);
- мова програмування JavaScript;
- HTML та CSS;
- системи контролю версій (VCS);
- інструменти для тестування;
- системи управління залежностями;
- системи розгортання та публікації;
- контейнеризація та оркестрація;
- інструменти моніторингу та логування [5].

Мова програмування Ruby відома своєю простотою та читабельністю, вона дозволяє швидко писати та підтримувати код. Її синтаксис наближений до природної мови, що зменшує кількість помилок і прискорює процес розробки.

Фреймворк Ruby on Rails надає структуровану архітектуру для вебдодатків, що включає вбудовані рішення для роботи з базами даних, кешування, маршрутизації, безпеки, тощо. Завдяки цьому розробники можуть зосередитися на бізнес-логіці і не вирішувати низькорівневі задачі [7].

Важливим елементом є вибір системи керування базами даних (СКБД), яку буде використано для зберігання даних вебсайту. Найчастіше використовують PostgreSQL або MySQL, які забезпечують достатню продуктивність, надійність та масштабованість [6].

Для додавання динамічності та інтерактивності до вебдодатку використовують JavaScript разом з відповідними бібліотеками та фреймворками, такими як jQuery або React.js. Вони дозволяють створювати швидкі та чутливі інтерфейси користувача. Основними технологіями для створення структури та стилів вебсторінок є HTML та CSS [3].

Використання такої системи контролю версій (VCS), як Git, є необхідним для ефективного керування базою програмного коду, відстеження

змін та спільної роботи над проєктом. GitHub або GitLab є популярними сервісами для хостингу репозиторіїв і спрощення колаборації між розробниками.

Якість програмного забезпечення перевіряється за допомогою тестування. Для проєктів на Ruby on Rails популярними є RSpec та Capybara, які надають можливості для написання різних видів тестів, від юніт-тестів, за допомогою яких можливе тестування компонентів або складових до інтеграційних та тестів сприйняття.

Для управління бібліотеками та пакетами використовують Bundler, який дозволяє легко додавати, оновлювати та керувати залежностями проєкту.

Автоматизація процесу розгортання та публікації є важливою для швидкого та безперебійного оновлення версій веб застосунку. Інструменти на кшталт Capistrano або Chef допомагають автоматизувати цей процес і забезпечити консистентність середовищ розгортання [5].

Масштабованість та висока доступність вебдодатків забезпечується завдяки використання Docker для контейнеризації додатків та Kubernetes для їх оркестрації.

Інструменти моніторингу та логування: для забезпечення стабільної роботи вебдодатку важливим є моніторинг та аналіз логів. Використання інструментів, таких як New Relic, Prometheus або ELK Stack (Elasticsearch, Logstash, Kibana), допомагає виявляти та усувати проблеми у роботі системи.

Успішність та ефективність розробки інструменту для переносу вебсайту на платформу Ruby on Rails значною мірою залежить від правильного вибору та інтеграції цих компонентів стека технологій. Кожен з них виконує свою роль у забезпеченні стабільності, продуктивності та зручності використання кінцевого продукту. Тому ретельний аналіз та обґрунтування кожного елемента є критично важливими для досягнення поставленої мети дослідження [11].

1.2 Фреймворк Ruby on Rails

Фреймворк Ruby on Rails, часто згадуваний просто як Rails, є одним з найпопулярніших і найбільш вживаних вебфреймворків у світі. Створений Девідом Хайнемейером Ханссоном у 2004 році, Rails базується на мові програмування Ruby і значно спрощує та прискорює процес розробки вебдодатків завдяки своїй орієнтації на продуктивність, конвенції та не повторюваність коду [10].

Основні принципи та концепції Ruby on Rails визначають його архітектуру та підхід до розробки. Одним з ключових принципів є «конвенція понад конфігурацією» (Convention over Configuration, CoC), що передбачає використання набору стандартів і шаблонів, які мінімізують необхідність ручного налаштування та конфігурування. Це дозволяє розробникам зосередитися на унікальних аспектах своєї програми, а не на налаштуванні конфігураційних файлів. Інший важливий принцип – «не повторюй себе» (Don’t Repeat Yourself, DRY), який спрямований на зменшення дублювання коду. Цей принцип досягається завдяки модульності, реюзабельності компонентів і використанню абстракцій, що дозволяє писати код, який можна легко повторно використовувати в різних частинах додатку [1].

Rails використовує патерн Active Record для роботи з базами даних. Це дозволяє розробникам маніпулювати даними, використовуючи об’єктно-орієнтований підхід, що спрощує написання SQL-запитів та управління базою даних. Rails також включає набір компонентів, таких як Action Pack, який відповідає за обробку HTTP-запитів, маршрутизацію та рендеринг відповідей. До складу Action Pack входять Action Controller, який керує запитами, взаємодіє з моделями і обирає відповідний вигляд для рендерингу, та Action View, що відповідає за рендеринг HTML-шаблонів і забезпечує відображення даних, отриманих з моделей.

Фреймворк також містить компонент Action Mailer, який дозволяє легко відправляти електронні листи з додатка. Він підтримує генерацію листів на

основі шаблонів, надсилання їх через SMTP або інші протоколи та забезпечує інтеграцію з іншими частинами фреймворку. Active Support є набором утиліт та розширень для мови Ruby, які полегшують розробку, включаючи додаткові методи для роботи з рядками, масивами, датами та іншими типами даних. Action Cable є компонентом для додавання функціональності вебсокетів до Rails-додатків, що дозволяє створювати реальні часом додатки з інтерактивними можливостями, такими як чати або сповіщення [2].

Ruby on Rails має ряд переваг, які роблять його привабливим вибором для розробників вебдодатків. Перш за все, Rails значно прискорює процес створення вебдодатків завдяки вбудованим засобам автоматизації, генераторам коду та зручним інструментам для тестування. Завдяки принципам CoC та DRY, код, написаний на Rails, зазвичай є чистим та легко читається, що спрощує його підтримку та розширення. Rails також має активну спільноту розробників, яка постійно вдосконалює фреймворк, додає нові можливості та забезпечує підтримку через форуми, блоги, конференції та інші ресурси.

Іншою перевагою є велика кількість готових до використання гемів та плагінів, які розширюють функціональність Rails-додатків і дозволяють швидко додавати нові можливості без необхідності писати багато коду з нуля. Це включає в себе все, від інтеграції з популярними API до додавання складних функцій, таких як аутентифікація користувачів або обробка платежів. Завдяки цим та іншим характеристикам, Ruby on Rails є потужним інструментом для швидкої, ефективної та зручної розробки вебдодатків, що робить його одним з найкращих виборів для сучасних веброзробників [3].

1.3 Висновки до розділу 1

У першому розділі було розглянуто основні поняття стека технологій та особливості фреймворку Ruby on Rails, що є ключовими для розробки

інструменту переносу вебсайту на цю платформу. Аналіз технологічного стека показав, що правильний вибір і інтеграція його компонентів є критично важливими для успішного та ефективного розвитку програмного забезпечення. Важливими елементами стека є мова програмування Ruby, системи керування базами даних, HTML, CSS, JavaScript, а також інструменти для тестування, контролю версій, розгортання та публікації, контейнеризації та моніторингу. Кожен з цих компонентів відіграє свою роль у забезпеченні стабільності, продуктивності та зручності використання кінцевого продукту.

Фреймворк Ruby on Rails, завдяки своїм принципам «конвенції понад конфігурацією» та «не повторюй себе», значно спрощує та прискорює процес розробки вебдодатків. Rails надає структуровану архітектуру, зручні інструменти та велику кількість готових рішень для різноманітних задач, що робить його привабливим вибором для розробників. Переваги Rails включають швидку розробку, легкість підтримки та розширення, активну спільноту, а також велику кількість гімів та плагінів, які розширюють функціональність додатків.

Таким чином, вибір Ruby on Rails як основного фреймворку для переносу вебсайту є добре обґрунтованим рішенням. Він забезпечує ефективний процес розробки, дозволяє швидко адаптуватися до змін та вимог проєкту, а також сприяє створенню якісного, масштабованого та легко підтримуваного програмного забезпечення. Розуміння загальних понять стека технологій та глибокий аналіз можливостей Ruby on Rails є фундаментом для подальшого успішного впровадження інструменту для переносу вебсайту на цю платформу.

2 ПРОЄКТУВАННЯ ТА МОДЕЛЮВАННЯ ІНСТРУМЕНТУ ДЛЯ ІМПОРТУ ШАБЛОНУ ЦІЛЬОВОГО САЙТУ

Перенесення вебсайту з однієї платформи на іншу є складним процесом, що вимагає ретельного планування та використання сучасних технологій. Інструмент, призначений для автоматизації цього процесу, повинен забезпечувати високу точність та ефективність перенесення, зберігаючи при цьому всю функціональність та структуру оригінального сайту.

Мета інструменту полягає в автоматизації перенесення контенту та структури вебсайту на платформу Ruby on Rails. Такий підхід дозволяє зменшити час і зусилля, необхідні для цього завдання, забезпечуючи при цьому високу якість кінцевого продукту.

У процесі розробки інструменту важливо враховувати різноманітні потреби користувачів, а також специфіку різних типів контенту, що дозволяє забезпечити універсальність та гнучкість рішення.

2.1 Концептуальна та логічна модель інструменту

Розробка інструменту для переносу вебсайту на платформу Ruby on Rails є складним і багатоступінчастим процесом, що потребує глибокого аналізу і детального планування. Основною метою було автоматизувати процес перенесення контенту та структури сайту з одночасним збереженням функціональності і адаптацією під нове середовище.

Концептуальна модель інструменту включає кілька ключових компонентів та етапів, кожен з яких виконує певну функцію. Перший компонент – це завантаження файлу sitemap.xml з цільового вебсайту. Sitemap.xml є важливим елементом, оскільки він містить повний перелік URL-адрес, що використовуються на сайті, та їх ієрархію. Це дозволяє отримати

детальне уявлення про структуру сайту, що є критично важливою вимогою для створення відповідних маршрутів у Ruby on Rails [4].

На наступному етапі було виконано парсинг файлу sitemap.xml з використанням бібліотеки Nokogiri, яка забезпечує потужні інструменти для обробки XML та HTML документів. Цей процес дозволяє ідентифікувати всі сторінки, які потрібно перенести, та визначити їхні взаємозв'язки. На основі отриманої інформації було створено маршрути у Rails, що відповідають структурі оригінального сайту. Цей крок включає очищення імен доменів для уникнення конфліктів і забезпечення логічної організації контролерів та дій.
Наступним важливим етапом є завантаження і обробка кожної сторінки сайту. Використовуючи Nokogiri, завантажується HTML вміст кожної сторінки і виконується його парсинг. Це дозволяє виділити основні елементи, такі як заголовки, параграфи, зображення, таблиці і т.д. Після цього ці елементи адаптуються для зберігання у вигляді файлів представлення (views) у Rails. Один з критичних аспектів цього етапу – вилучення мета-тегів, які не є необхідними, і збереження тих, що критично важливі для SEO [8].

Далі інструмент перевіряє всі посилання на зовнішні CSS та JavaScript файли. Якщо такі файли виявлено, вони завантажуються і зберігаються у відповідних директоріях нового Rails проєкту. Всі внутрішні посилання в цих файлах коригуються, щоб забезпечити їхню правильну роботу в новому середовищі. Цей процес включає автоматичне редагування шляхів до ресурсів, таких як зображення та шрифти, щоб вони правильно відображалися на новому сайті [10].

Особливу увагу приділено підтримці мультимовності. Використовуючи сервіс Amazon Translate, автоматизовано процес перекладу текстових елементів на додаткові мови, задані користувачем. Це включає переклад тексту всередині HTML тегів, атрибутів (наприклад, alt для зображень) та мета-тегів. Кожна мовна версія сторінок зберігається окремо, що полегшує управління різними мовними версіями сайту. Цей підхід дозволяє зберегти повну функціональність сайту при його використанні різними мовними

групами користувачів.

Для зручності підтримки і уникнення дублювання коду, інструмент створює часткові представлення (partials) для загальних елементів сторінок, таких як шапка та підвал сторінки. Це дозволяє зменшити кількість коду і зробити його більш організованим. Інструмент автоматично ідентифікує ці елементи на основі заданих класів і зберігає їх у вигляді partials, що значно спрощує їх подальше редагування і оновлення [13].

Після завершення парсингу і адаптації всіх сторінок сайту, інструмент редагує основний шаблон layout у Rails. Це додає включення всіх необхідних стилів та скриптів, забезпечуючи коректне відображення всіх сторінок сайту у новому середовищі. Цей процес включає також налаштування загального вигляду і стилю сайту, зберігаючи його оригінальний дизайн [8].

Завершальний етап включає тестування створеного проєкту для виявлення можливих помилок і недоліків. Це важливий крок, який дозволяє упевнитися в тому, що всі сторінки сайту правильно відображаються і функціонують у новому середовищі. Інструмент надає звіти про результати тестування і дозволяє швидко виправляти виявлені проблеми.

В цілому, концептуальна і логічна модель інструменту забезпечує автоматизований, надійний і ефективний процес перенесення вебсайтів на платформу Ruby on Rails. Використання сучасних бібліотек і сервісів дозволяє зберегти функціональність сайту, забезпечити підтримку мультимовності і спростити подальше управління і підтримку створеного проєкту. Цей підхід значно зменшує час і зусилля, необхідні для перенесення сайту, і забезпечує високу якість кінцевого результату.

2.2 Проєктування структури інструменту

Проєктування структури інструменту включає в себе ретельне визначення функціональних можливостей та компонентів, необхідних для

успішного імпорту шаблону цільового сайту з використанням Ruby on Rails. Перед проєктуванням необхідно чітко визначити вимоги до інструменту та з'ясувати потреби користувачів.

Центральним елементом структури інструменту є клас TestGenerator, який включає в себе основну логіку для клонування сайту та імпорту шаблону. Цей клас реалізує важливі функції, такі як завантаження сторінок, переклад контенту, створення файлів зображень та стилів, а також генерація маршрутів та контролерів для створення нових сторінок у Rails додатку [9].

Для забезпечення гнучкості та можливості налаштування інструменту використовуються опції, які дозволяють користувачеві вказувати різноманітні параметри, такі як URL цільового сайту, мова сайту, додаткові мови для перекладу, API ключі для перекладу та інші. Це робить інструмент універсальним та придатним для різноманітних сценаріїв використання.

Також важливою частиною структури є методи для завантаження та обробки різних видів ресурсів, таких як CSS файли, JavaScript файли та медіа контент. Ці методи забезпечують коректний імпорт та інтеграцію різноманітних елементів шаблону цільового сайту [12].

Урахування потреб користувачів та забезпечення простоти використання інструменту є ключовими пріоритетами при проєктуванні структури. Детальний аналіз вимог та участь користувачів у процесі розробки дозволяють створити інструмент, який задовольнить потреби широкого кола користувачів та забезпечить зручний та ефективний процес імпорту шаблону цільового сайту на платформу Ruby on Rails.

Додатковою складовою структури інструменту є логіка для обробки мовних версій сайту. Це включає в себе автоматичний переклад контенту на різні мови за допомогою зовнішніх сервісів, таких як Amazon Translate. Забезпечення можливості робити сайт доступним для різноманітних аудиторій є важливим аспектом розробки інструменту.

Окрему увагу слід звернути на можливості налаштування інструменту для роботи з різними типами контенту, такими як зображення, відео, PDF

файли тощо. Додаткові опції дозволяють користувачам вказувати, які типи контенту ігнорувати або як обробляти їх в залежності від мовної версії сайту.

Структура інструменту також передбачає можливість робити різні операції зі стилізацією та скриптами. Це включає в себе можливість вбудовувати стилі та скрипти безпосередньо в HTML сторінки або зберігати їх у окремі файли для подальшого використання. Такий підхід дозволяє забезпечити чистоту коду та покращити підтримку інструменту [7].

Загальна структура інструменту побудована з урахуванням принципів модульності та розширюваності, що дозволяє легко додавати нові функції та можливості в майбутньому. Це робить інструмент гнучким та придатним для використання в різних проєктах та сценаріях розробки вебдодатків.

Крім того, важливою складовою структури інструменту є можливість робити додаткові операції з метаданими сторінок. Це може включати автоматичне створення тегів Open Graph для соціальних мереж, що покращує представлення сайту при його поширенні в соціальних мережах.

Проєктування інструменту також передбачає можливість робити інтеграцію з іншими сервісами та API для отримання додаткової функціональності. Наприклад, інтеграція з сервісами аналітики або моніторингу дозволяє користувачам отримувати додаткову інформацію про використання та продуктивність їх вебсайту.

Загалом, структура інструменту повинна бути добре організованою та легко розширюватися. Це дозволить забезпечити ефективну розробку, підтримку та розвиток інструменту в майбутньому. Чітко визначені модулі та компоненти дозволяють розробникам працювати над окремими частинами інструменту незалежно, що підвищує швидкість розробки та знижує ризик виникнення помилок [11].

Проєктування структури інструменту вимагає ретельного аналізу вимог та уважного планування. Тільки таким чином можна створити ефективний та потужний інструмент для імпорту шаблону цільового сайту з використанням Ruby on Rails.

Ще однією ключовою складовою структури інструменту є можливість налаштовувати його параметри з метою забезпечення гнучкості та адаптованості до різних потреб користувачів. Це може включати параметри, які дозволяють вказати URL-адресу цільового сайту, мову сайту, API-ключі для сервісів перекладу чи інтеграції з Amazon Web Services (AWS) [6].

Також важливим аспектом є можливість налаштовувати структуру файлів та каталогів, щоб забезпечити зручність управління та організацію проєкту. Це дозволяє користувачам легко знаходити необхідні файли та ресурси та швидко робити зміни в їх структурі.

Крім того, структура інструменту повинна бути добре документованою, щоб забезпечити зрозумілість та легкість використання для розробників. Чітка документація допомагає знизити час на ознайомлення з інструментом та підвищує його придатність для використання в командному середовищі [9].

Усі ці аспекти структури інструменту допомагають забезпечити його ефективність, надійність та зручність використання. Планування та проєктування структури інструменту є важливою частиною процесу розробки, яка визначає успішність та придатність інструменту для вирішення конкретних завдань користувачів.

2.3 Висновки до розділу 2

У розділі 2 було детально розглянуто концептуальну та логічну модель інструменту для перенесення вебсайтів на платформу Ruby on Rails. Було визначено основні компоненти та етапи процесу перенесення, включаючи завантаження файлу sitemap.xml, парсинг сторінок, обробку контенту, налаштування маршрутів та підтримку мультимовності. Кожен етап детально описано з акцентом на використанні сучасних бібліотек та сервісів, що забезпечує ефективний і автоматизований процес перенесення.

Проєктування структури інструменту включало визначення ключових

класів, таких як TestGenerator, а також методів для обробки різноманітних ресурсів і контенту. Особлива увага приділялася забезпеченню гнучкості та налаштовуваності інструменту для задоволення потреб різних користувачів. Було підкреслено важливість належної документації, модульності та можливості розширення інструменту для забезпечення його тривалої підтримки та розвитку.

Загалом, розділ 2 висвітлив всі необхідні аспекти проєктування та моделювання інструменту, що створює надійну основу для його подальшої розробки та використання. Такий підхід забезпечує високу якість кінцевого продукту та значно спрощує процес перенесення вебсайтів, зберігаючи їх функціональність і адаптуючи до нового середовища.

У ході проєктування інструменту було враховано різноманітні потреби користувачів та специфіку різних типів контенту, що дозволяє забезпечити універсальність та гнучкість рішення. Створення інструменту з використанням принципів модульності та розширюваності сприяє легкому додаванню нових функцій і адаптації до нових вимог, що виникають у процесі розробки та експлуатації вебдодатків.

Також особливу увагу було приділено забезпеченню підтримки мультимовності, що є важливим аспектом для сучасних вебсайтів. Використання сервісів автоматичного перекладу, таких як Amazon Translate, дозволяє значно спростити процес локалізації контенту та зробити сайт доступним для користувачів з різних країн.

Висновок, що можна зробити з розглянутого матеріалу, полягає в тому, що ретельне проєктування та моделювання інструменту для імпорту є критично важливими етапами, що визначають успіх усього проєкту. Завдяки структурованому підходу та використанню сучасних технологій, вдалося створити інструмент, що відповідає найвищим стандартам якості та функціональності, забезпечуючи простоту використання та ефективність процесу перенесення вебсайтів на платформу Ruby on Rails.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНСТРУМЕНТУ ІМПОРТУ ШАБЛОНУ ЦІЛЬОВОГО САЙТУ

3.1 Впровадження інструменту, огляд середовища розробки

Перед початком розробки інструменту для імпорту шаблону цільового сайту необхідно визначити та налаштувати середовище розробки. Для розробки було обрано платформу Ruby on Rails завдяки її потужним можливостям для розробки вебдодатків, модульній структурі та багатому набору інструментів для роботи з вебконтентом. Підготовка середовища розробки включала вибір операційної системи, системи управління версіями коду, текстових редакторів або інтегрованих середовищ розробки (IDE), а також додаткових бібліотек та фреймворків.

Для даного проєкту було обрано наступне середовище розробки. Операційною системою стала Ubuntu 20.04 LTS завдяки її стабільності та поширеності серед розробників. Для управління версіями коду використовується Git, який забезпечує надійність та зручність в роботі з кодом. Основним текстовим редактором став Visual Studio Code через його гнучкість та підтримку великої кількості розширень для Ruby on Rails. Платформою для розробки обрано фреймворк Ruby on Rails 7, який надає багатий набір інструментів для створення вебдодатків. Системою управління базами даних стала PostgreSQL, відома своєю надійністю та високою продуктивністю.

Для налаштування середовища розробки були виконані кілька кроків. Спершу було встановлено Ruby за допомогою менеджера пакетів RVM (Ruby Version Manager). Це дозволило легко керувати версіями Ruby та встановлювати необхідні залежності. Встановлення Ruby включало виконання команд для завантаження та налаштування RVM, після чого була встановлена конкретна версія Ruby 2.7.2.

Наступним кроком було встановлення Rails. Це було зроблено за допомогою команди gem install rails, що дозволило завантажити та встановити останню стабільну версію Rails. Для роботи з базою даних була встановлена PostgreSQL, що забезпечує високу продуктивність та масштабованість. Встановлення PostgreSQL включало оновлення пакетів системи та встановлення необхідних компонентів.

Таким чином, налаштування середовища розробки забезпечило необхідні умови для ефективної роботи над інструментом імпорту шаблону цільового сайту, створивши базу для подальшої розробки, тестування та впровадження проєкту.

3.2 Розробка основних компонентів

Розробка основних компонентів інструменту для імпорту шаблону цільового сайту на платформу Ruby on Rails включає в себе кілька ключових етапів. Кожен з цих етапів спрямований на реалізацію окремих функцій, які забезпечують коректний та ефективний процес перенесення контенту та структури сайту. Основні компоненти включають завантаження та парсинг файлу sitemap.xml, обробку HTML контенту, генерацію маршрутів та контролерів, а також інтеграцію з зовнішніми сервісами.

3.2.1 Завантаження та парсинг файлу sitemap.xml

Перший етап розробки включав створення модуля для завантаження файлу sitemap.xml з цільового вебсайту. Для цього було використано бібліотеку OpenURI, яка дозволяє завантажувати файли з інтернету. OpenURI забезпечує простий і зручний спосіб завантаження даних через HTTP, що робить її ідеальним вибором для нашого завдання. Код, що реалізує цей процес, показано на рисунку 3.1.

```
require 'open-uri'
sitemap_url = 'http://example.com/sitemap.xml'
sitemap_content = URI.open(sitemap_url).read
```

Рисунок 3.1 – Код для завантаження файлу sitemap.xml

Після завантаження файлу sitemap.xml було розроблено функціонал для його парсингу за допомогою бібліотеки Nokogiri. Nokogiri є потужною бібліотекою для роботи з XML та HTML документами в Ruby. Вона дозволяє легко обробляти та витягати інформацію з XML-файлів. Для парсингу файлу sitemap.xml було створено документ Nokogiri з отриманого контенту, як показано на рисунку 3.2.

```
require 'nokogiri'
sitemap_doc = Nokogiri::XML(sitemap_content)
```

Рисунок 3.2 – Підключення бібліотеки Nokogiri

Цей процес включав виділення всіх URL-адрес, їх ієрархії та метаданих, що зберігаються у файлі sitemap.xml. Для цього ми використовували XPath-запити, які дозволяють ефективно знаходити необхідні елементи в документі. Наприклад, щоб витягти всі URL-адреси з sitemap.xml, було використано код, що наведено на рисунку 3.3.

```
urls = sitemap_doc.xpath('//url/loc').map(&:text)
```

Рисунок 3.3 – Парсинг sitemap

Цей код знаходить всі елементи <loc> всередині елементів <url> і витягує текстовий вміст цих елементів, який є URL-адресами сторінок сайту.

Також було важливо отримати метадані для кожної URL-адреси, такі як частота оновлення (change frequency) та пріоритет (priority), які можуть бути корисними для подальшого аналізу та обробки. Це можна зробити

аналогічним чином, використовуючи XPath-запити, як показано на рисунку 3.4.

```
sitemap_data = sitemap_doc.xpath('//url').map do |url|
  {
    loc: url.xpath('loc').text,
    lastmod: url.xpath('lastmod').text,
    changefreq: url.xpath('changefreq').text,
    priority: url.xpath('priority').text
  }
end
```

Рисунок 3.4 – Збереження метаданих сторінок

Цей код дозволяє створити масив хешів, де кожен хеш представляє собою одну сторінку сайту з усіма необхідними метаданими.

Парсинг дозволив отримати детальну інформацію про структуру сайту, необхідну для подальшої обробки. Ця інформація є основою для створення маршрутів, контролерів та файлів представлення у Rails. В результаті ми отримали чітке уявлення про всі сторінки, які потрібно перенести, їхню ієрархію та взаємозв’язки, що забезпечило коректність і повноту процесу перенесення.

3.2.2 Обробка HTML контенту

Наступним етапом стала розробка модуля для обробки HTML контенту кожної сторінки сайту. Цей модуль відігравав ключову роль у забезпеченні точного переносу вмісту та структури сторінок на платформу Ruby on Rails. Використовуючи Nokogiri, було реалізовано функції для завантаження HTML вмісту сторінок та їх парсингу. Nokogiri надає ефективні інструменти для

аналізу та виділення елементів з HTML документів, що дозволяє легко витягувати необхідні дані.

Для початку, кожна сторінка сайту завантажувалась з використанням OpenURI, аналогічно до завантаження файлу sitemap.xml. Код для завантаження HTML вмісту виглядав як показано на рисунку 3.5.

```
require 'open-uri'
require 'nokogiri'
page_url = 'http://example.com/page.html'
html_content = URI.open(page_url).read
html_doc = Nokogiri::HTML(html_content)
```

Рисунок 3.5 – Завантаження HTML коду сторінок

Після завантаження HTML вмісту, було реалізовано функції для його парсингу. Цей процес включав виділення основних елементів сторінок, таких як заголовки, параграфи, зображення, таблиці та інші. Для кожного типу елементів використовувалися відповідні XPath або CSS селектори. Наприклад, для виділення заголовків і параграфів використовувався код, наведений на рисунку 3.6.

```
headings = html_doc.css('h1, h2, h3, h4, h5, h6').map(&:text)
paragraphs = html_doc.css('p').map(&:text)
```

Рисунок 3.6 – Збереження заголовків та параграфів

Аналогічно, для виділення зображень та таблиць використовувався код, який показано на рисунку 3.7.

```
images = html_doc.css('img').map { |img| img['src'] }
tables = html_doc.css('table')
```

Рисунок 3.7 – Збереження зображень та таблиць

Отримані дані використовувалися для створення файлів представлення (views) у Rails. Ці файли відображають структуру та вміст оригінальних сторінок, забезпечуючи збереження їх вигляду та функціональності. Наприклад, для кожної сторінки створювався відповідний файл представлення, як показано на рисунку 3.8.

```
<%# app/views/pages/show.html.erb %>
<% headings.each do |heading| %>
  <h1><%= heading %></h1>
<% end %>
<% paragraphs.each do |paragraph| %>
  <p><%= paragraph %></p>
<% end %>
<% images.each do |image| %>
  <img src="<%= image %>" alt="Image">
<% end %>
<% tables.each do |table| %>
  <table><%= table.inner_html %></table>
<% end %>
```

Рисунок 3.8 – Приклад готової сторінки в Rails додатку

Особлива увага приділялася обробці мета-тегів, які мають важливе значення для SEO. Мета-теги, такі як meta description і meta keywords, були виділені та збережені для подальшого використання. Непотрібні мета-теги видалялися, а критично важливі зберігалися. Для цього використовувалися наступні методи, показані на рисунку 3.9.

Цей підхід дозволив забезпечити збереження важливих SEO-метаданих, що сприяє кращому ранжуванню сайту в пошукових системах. Таким чином, модуль для обробки HTML контенту забезпечив точний і ефективний перенос вмісту сторінок, зберігаючи їх оригінальний вигляд і функціональність у новому середовищі Ruby on Rails.

```
meta_tags = html_doc.css('meta')
important_meta_tags = meta_tags.select do |meta|
  %w[description keywords].include?(meta['name'])
end
# Видалення непотрібних мета-тегів
html_doc.css('meta').each do |meta|
  meta.remove unless important_meta_tags.include?(meta)
end
```

Рисунок 3.9 – Видалення зайвих елементів коду

3.2.3 Генерація маршрутів та контролерів

Для забезпечення відповідності структурі оригінального сайту, було розроблено функціонал для генерації маршрутів та контролерів у Rails. Цей процес був критично важливим для забезпечення правильної обробки запитів до нових сторінок сайту, що переносилися з оригінального середовища.

Перший крок полягав у аналізі даних, отриманих з файлу sitemap.xml. Ці дані містили інформацію про всі URL-адреси та їх ієрархічну структуру, що дозволяло чітко визначити, які сторінки повинні бути перенесені та як вони взаємопов’язані. На основі цієї інформації створювалися відповідні маршрути у Rails. Для цього використовувалася стандартна конфігурація маршрутизації Rails, яка задається у файлі config/routes.rb.

Основний код для генерації маршрутів виглядав наступним чином (рис. 3.10).

Наступним кроком було очищення імен доменів та шляхів для уникнення конфліктів та забезпечення логічної організації контролерів та дій. Це включало заміну спеціальних символів, таких як косі риски, тире та крапки, на підкреслення, що дозволяло уникнути проблем з іменуванням методів у Rails. Очищення імен було виконано за допомогою методу gsub, як показано у рисунку 3.10.

```
require 'nokogiri'
require 'open-uri'
# Завантаження та парсинг файлу sitemap.xml
sitemap_url = 'http://example.com/sitemap.xml'
sitemap_xml = Nokogiri::XML(URI.open(sitemap_url))
# Виділення URL-адрес з файлу sitemap.xml
urls = sitemap_xml.xpath('//url/loc').map(&:text)
# Генерація маршрутів у файлі config/routes.rb
Rails.application.routes.draw do
  urls.each do |url|
    path = URI.parse(url).path
    clean_path = path.gsub('/', '_').gsub('-', '_').gsub('.', '_')
    get path, to: "pages##{clean_path}"
  end
end
```

Рисунок 3.10 – Генерація маршрутів

Кожен згенерований маршрут був зв’язаний з відповідним контролером та дією, що обробляла запит. Для цього створювалися контролери, які містили методи для обробки запитів до кожної сторінки. Наприклад, якщо URL-адреса сторінки була /about-us, маршрут генерувався як get '/about-us', to: 'pages#about_us', а відповідний метод about_us додавався до контролера PagesController, як це показано на рисунку 3.11.

```
class PagesController < ApplicationController
  def about_us
    # Логіка для обробки запиту до сторінки "About Us"
    render template: 'pages/about_us'
  end
  # Інші методи для інших сторінок...
end
```

Рисунок 3.11 – Контролер сторінок

Для забезпечення масштабованості та підтримки в майбутньому, також було реалізовано механізми автоматичного оновлення маршрутів та контролерів при змінах у файлі sitemap.xml. Це дозволяло швидко адаптуватися до будь-яких змін у структурі оригінального сайту без необхідності ручного втручання.

Таким чином, розроблений функціонал для генерації маршрутів та контролерів у Rails забезпечив відповідність структурі оригінального сайту, дозволивши коректно обробляти запити до нових сторінок та забезпечити їх безперебійну роботу у новому середовищі Ruby on Rails.

3.2.4 Інтеграція з зовнішніми сервісами

Особливу увагу приділялося інтеграції з зовнішніми сервісами, такими як Amazon Translate, для автоматичного перекладу контенту. Це забезпечувало мультимовність сайту та його доступність для користувачів з різних країн, що є важливим аспектом сучасних вебдодатків.

Першим кроком у цьому процесі було створення відповідних класів та методів, які забезпечували взаємодію з API сервісу Amazon Translate. Це включало налаштування з’єднання з сервісом, відправлення тексту для перекладу та отримання перекладеного тексту. Основним інструментом для роботи з API був гем aws-sdk-translate, який надає зручний інтерфейс для взаємодії з сервісом.

Приклад класу для роботи з Amazon Translate наведено на рисунку 3.12.

Цей клас містив метод translate_text, який приймав текст для перекладу, код вихідної мови та код цільової мови. Використовуючи клієнт AWS SDK, цей метод відправляв запит до сервісу Amazon Translate та отримував перекладений текст.

Для перекладу тексту всередині HTML тегів та атрибутів, таких як атрибут alt для зображень, був створений окремий модуль, що відповідав за

парсинг HTML контенту та відправлення тексту для перекладу. За допомогою бібліотеки Nokogiri HTML контент розбивався на окремі елементи, які потрібно було перекласти.

Приклад коду для перекладу контенту наведено на рисунку 3.13.

```
require 'aws-sdk-translate'
class AmazonTranslateService
  def initialize
    @translate_client = Aws::Translate::Client.new(region: 'us-east-1')
  end
  def translate_text(text, source_lang, target_lang)
    response = @translate_client.translate_text({
      text: text,
      source_language_code: source_lang,
      target_language_code: target_lang
    })
    response.translated_text
  end
end
```

Рисунок 3.12 – Підключення AWS API

```
require 'nokogiri'
class HtmlTranslator
  def initialize(translate_service)
    @translate_service = translate_service
  end
  def translate_html(html_content, source_lang, target_lang)
    doc = Nokogiri::HTML(html_content)
    # Переклад тексту всередині тегів
    doc.traverse do |node|
      if node.text?
        node.content = @translate_service.translate_text(node.text, source_lang,
target_lang)
      elsif node.element?
        node.attributes.each do |attr_name, attr_value|
          if attr_name == 'alt'
            node[attr_name] = @translate_service.translate_text(attr_value.value,
source_lang, target_lang)
          end
        end
      end
    end
    doc.to_html
  end
end
```

Рисунок 3.13 – Переклад сторінок під час парсингу

Цей клас HtmlTranslator приймав на вхід об’єкт сервісу перекладу та HTML контент, який потрібно було перекласти. Використовуючи метод traverse бібліотеки Nokogiri, він проходив через всі елементи HTML документа, перекладаючи текст всередині тегів та атрибутів, таких як alt для зображень. Перекладений контент повертався у вигляді HTML коду.

Отримані переклади зберігалися у окремих мовних версіях сторінок, що полегшувало управління різними мовними версіями сайту. Для цього використовувалася структура директорій, де кожна мова мала свою окрему директорію з файлами представлення. Приклад наведено на рисунку 3.14.

```
app/views/pages/en/about_us.html.erb
app/views/pages/es/about_us.html.erb
app/views/pages/fr/about_us.html.erb
```

Рисунок 3.14 – Мовні версії сторінки

Це дозволяло легко керувати різними мовними версіями сторінок та забезпечувати їх актуальність. Для кожної мови було створено окремий файл представлення, що містив перекладений контент. Такий підхід спрощував локалізацію сайту та забезпечував його доступність для широкої аудиторії користувачів.

Таким чином, інтеграція з Amazon Translate дозволила автоматизувати процес перекладу контенту, забезпечивши підтримку мультимовності на сайті. Це було досягнуто завдяки використанню спеціалізованих класів та методів для взаємодії з API сервісу, парсингу HTML контенту та збереження перекладених даних у окремих мовних версіях сторінок.

3.2.5 Обробка ресурсів

Ще одним важливим компонентом розробки інструменту імпорту шаблону цільового сайту була обробка зовнішніх ресурсів, таких як CSS та

JavaScript файли. Цей процес забезпечував правильне відображення стилів та поведінки на новому сайті, що є критично важливим для збереження користувацького досвіду.

Першим етапом було завантаження всіх необхідних зовнішніх ресурсів з оригінального сайту. Для цього використовувалися HTTP-запити, що дозволяло автоматично отримати всі CSS та JavaScript файли. Вони завантажувалися та зберігалися у відповідних директоріях Rails проєкту, таких як app/assets/stylesheets для CSS файлів та app/assets/javascripts для JavaScript файлів.

Приклад коду для завантаження та збереження файлів виглядав наступним чином, як показано на рисунку 3.15.

```
require 'open-uri'
class AssetDownloader
 def initialize(base_url, asset_urls)
  @base_url = base_url
  @asset_urls = asset_urls
 end
 def download_assets
  @asset_urls.each do |asset_url|
   full_url = "#{@base_url}/#{asset_url}"
   file_path = asset_url_to_path(asset_url)
   open(full_url) do |file|
    File.open(file_path, "wb") do |out|
     out.write(file.read)
    end
   end
  end
 end
 private
 def asset_url_to_path(asset_url)
  if asset_url.end_with?(".css")
   "app/assets/stylesheets/#{File.basename(asset_url)}"
  elsif asset_url.end_with?(".js")
   "app/assets/javascripts/#{File.basename(asset_url)}"
  else
   "app/assets/images/#{File.basename(asset_url)}"
  end
 end
end
```

Рисунок 3.15 – Завантаження та збереження файлів

Після завантаження та збереження зовнішніх ресурсів, необхідно було коригувати внутрішні посилання в цих файлах. Це включало автоматичне редагування шляхів до ресурсів, таких як зображення та шрифти, щоб вони правильно відображалися на новому сайті. Для цього використовувалася бібліотека Nokogiri, яка дозволяла парсити CSS та JavaScript файли та змінювати шляхи до внутрішніх ресурсів.

Приклад коду для коригування шляхів виглядав наступним чином, як показано на рисунку 3.16.

```
require 'nokogiri'
class AssetLinkUpdater
  def initialize(asset_files)
    @asset_files = asset_files
  end
  def update_links
    @asset_files.each do |file_path|
      content = File.read(file_path)
      updated_content = update_asset_links(content)
      File.write(file_path, updated_content)
    end
  end
  private
  def update_asset_links(content)
    doc = Nokogiri::HTML(content)
    doc.css('link[rel="stylesheet"]').each do |link|
      link['href'] = adjust_path(link['href'])
    end
    doc.css('script[src]').each do |script|
      script['src'] = adjust_path(script['src'])
    end
    doc.to_html
  end
  def adjust_path(path)
    if path.start_with?('/')
      "/assets#{path}"
    else
      path
    end
  end
end
```

Рисунок 3.16 – Коригування шляхів

Важливою частиною коригування внутрішніх посилань було автоматичне редагування шляхів до ресурсів. Це включало зміну абсолютних шляхів на відносні, або додавання префіксу /assets до шляхів, щоб вони відповідали новій структурі проєкту на Rails. Це дозволяло зберегти правильне відображення зображень, шрифтів та інших ресурсів на новому сайті.

Після коригування шляхів та збереження оновлених файлів, важливою частиною процесу було тестування та валідація правильності відображення зовнішніх ресурсів. Це включало перевірку коректності завантаження стилів та скриптів, а також їх впливу на вигляд та поведінку сторінок. Для цього використовувалися як ручні методи тестування, так і автоматичні інструменти для перевірки валідності HTML та CSS коду.

Таким чином, обробка зовнішніх ресурсів, таких як CSS та JavaScript файли, включала декілька важливих етапів: завантаження файлів, їх збереження у відповідних директоріях проєкту, коригування внутрішніх посилань та тестування правильності відображення. Це забезпечувало збереження зовнішнього вигляду та функціональності сайту після його перенесення на нову платформу Rails.

3.2.6 Створення часткових представлень (partials)

Для зручності підтримки та уникнення дублювання коду, було вирішено реалізувати функціонал для автоматичного створення часткових представлень (partials) для загальних елементів сторінок, таких як хедер та футер. Цей підхід дозволяв значно зменшити кількість повторюваного коду та зробити структуру проєкту більш організованою. Завдяки використанню partials, редагування та оновлення загальних елементів сторінок стало більш простим і ефективним.

Першим кроком було ідентифікування загальних елементів на сторінках цільового сайту. Для цього використовувалася бібліотека Nokogiri, яка

дозволяла парсити HTML-код сторінок та знаходити елементи з заданими класами. Зазвичай, загальні елементи, такі як хедер та футер, мають певні класи або ідентифікатори, які використовуються на всіх сторінках сайту.

Приклад коду для ідентифікації загальних елементів наведено на рисунку 3.17.

```
require 'nokogiri'

class CommonElementsExtractor
  def initialize(html_content)
    @doc = Nokogiri::HTML(html_content)
  end

  def extract_common_elements
    header = @doc.at_css('.header') || @doc.at_css('#header')
    footer = @doc.at_css('.footer') || @doc.at_css('#footer')
    { header: header.to_html, footer: footer.to_html }
  end
end
```

Рисунок 3.17 – Ідентифікація загальних елементів

Після ідентифікації загальних елементів, наступним кроком було створення partials для цих елементів. Часткові представлення (partials) зберігалися у відповідних директоріях app/views/layouts у проєкті Rails. Це дозволяло легко включати їх до інших представлень за допомогою методу render.

Після створення partials, необхідно було інтегрувати їх у відповідні представлення сайту. Це включало заміну безпосередніх вставок HTML-коду хедера та футера на виклики partials за допомогою методу render в представленнях Rails. Приклад коду для інтеграції partials наведено на рисунку 3.18.

```
class PartialsCreator
  def initialize(common_elements)
    @common_elements = common_elements
  end
  def create_partials
    create_partial('header', @common_elements[:header])
    create_partial('footer', @common_elements[:footer])
  end
  private
  def create_partial(name, content)
    File.open("app/views/layouts/_#{name}.html.erb", 'w') do |file|
      file.write(content)
    end
  end
end
```

Рисунок 3.18 – Створення partials

Використання partials дозволяло значно зменшити кількість коду в представленнях, оскільки загальні елементи вставлялися тільки один раз і могли використовуватися на всіх сторінках сайту. Це спрощувало процес редагування та оновлення цих елементів, оскільки зміни в partials автоматично відображалися на всіх сторінках, де вони використовуються. Такий підхід робив код проєкту більш організованим та легшим для підтримки.

Для редагування загальних елементів достатньо було змінити відповідний partial. Наприклад, щоб змінити вміст хедера, потрібно було відредагувати файл app/views/layouts/_header.html.erb. Це значно спрощувало підтримку проєкту, оскільки всі зміни зосереджувалися в одному місці, замість пошуку та редагування однакових елементів на кожній сторінці окремо.

Таким чином, реалізація функціоналу для автоматичного створення часткових представлень (partials) дозволила зробити код проєкту більш організованим, зменшити кількість дублювання коду та спростити процес редагування та оновлення загальних елементів сторінок, як це показано на рисунку 3.19.

```
<!-- app/views/layouts/application.html.erb -->
<!DOCTYPE html>
<html>
 <head>
  <title>MyWebsite</title>
  <%= csrf_meta_tags %>
  <%= csp_meta_tag %>
  <%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track': 'reload'
%>
  <%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
 </head>
 <body>
  <%= render 'layouts/header' %>
  <%= yield %>
  <%= render 'layouts/footer' %>
 </body>
</html>
```

Рисунок 3.19 – Приклад застосування partials

3.3 Тестування інструменту для імпорту шаблону цільового сайту на платформу Ruby on Rails

Першим кроком у процесі тестування стало створення та налаштування нового проєкту на базі Rails. На наведеному скріншоті показано процес ініціалізації нового проєкту з назвою “example_for_diploma”. Команда rails new example_for_diploma створює структуру директорій та файлів, необхідних для Rails додатку.

Під час ініціалізації проєкту також автоматично створюється порожній Git-репозиторій. Це дозволяє відслідковувати зміни в коді та підтримувати контроль версій, що є важливим аспектом при розробці та тестуванні. Як показано на рисунку 3.20.

На рисунку 3.21 представлено файл Gemfile, який містить список бібліотек (гемів), необхідних для роботи Rails-проєкту. Один з ключових компонентів у проєкті – бібліотека parsergem, яка є основним об’єктом дипломного проєкту.

```
master@master-System-Product-Name:~$ rails new example_for_diploma
      create
      create  README.md
      create  Rakefile
      create  .ruby-version
      create  config.ru
      create  .gitignore
      create  .gitattributes
      create  Gemfile
         run  git init from "."
подсказка: Using 'master' as the name for the initial branch. This default branch name
подсказка: is subject to change. To configure the initial branch name to use in all
подсказка: of your new repositories, which will suppress this warning, call:
подсказка:
подсказка:      git config --global init.defaultBranch <name>
подсказка:
подсказка: Names commonly chosen instead of 'master' are 'main', 'trunk' and
подсказка: 'development'. The just-created branch can be renamed via this command:
подсказка:
подсказка:      git branch -m <name>
Инициализирован пустой репозиторий Git в /home/master/example_for_diploma/.git/
      create  app
      create  app/assets/config/manifest.js
      create  app/assets/stylesheets/application.css
      create  app/channels/application_cable/channel.rb
      create  app/channels/application_cable/connection.rb
      create  app/controllers/application_controller.rb
      create  app/helpers/application_helper.rb
      create  app/jobs/application_job.rb
      create  app/mailers/application_mailer.rb
      create  app/models/application_record.rb
      create  app/views/layouts/application.html.erb
      create  app/views/layouts/mailer.html.erb
      create  app/views/layouts/mailer.text.erb
      create  app/assets/images
      create  app/assets/images/.keep
```

Рисунок 3.20 – Ініціалізація нового Rails проєкту

Gemfile U

Gemfile

```
source "https://rubygems.org"
git_source(:github) { |repo| "https://github.com/#{repo}.git" }

ruby "3.0.0"

# Bundle edge Rails instead: gem "rails", github: "rails/rails", branch: "main"
gem "rails", "~> 7.0.8", ">= 7.0.8.1"

# The original asset pipeline for Rails [https://github.com/rails/sprockets-rails]
gem "sprockets-rails"

# Use sqlite3 as the database for Active Record
gem "sqlite3", "~> 1.4"

# Use the Puma web server [https://github.com/puma/puma]
gem "puma", "~> 5.0"

# Use JavaScript with ESM import maps [https://github.com/rails/importmap-rails]
gem "importmap-rails"

# Hotwire's SPA-like page accelerator [https://turbo.hotwired.dev]
gem "turbo-rails"

# Hotwire's modest JavaScript framework [https://stimulus.hotwired.dev]
gem "stimulus-rails"

# Build JSON APIs with ease [https://github.com/rails/jbuilder]
gem "jbuilder"

# Use Redis adapter to run Action Cable in production
gem "redis", "~> 4.0"

gem 'parsergem' # Підключення розробленої бібліотеки до rails проєкту

# Use Kredis to get higher-level data types in Redis [https://github.com/rails/kredis]
# gem "kredis"

# Use Active Model has_secure_password [https://guides.rubyonrails.org/active_model_basics.html#securepassword]
# gem "bcrypt", "~> 3.1.7"

# Windows does not include zoneinfo files, so bundle the tzinfo-data gem
gem "tzinfo-data", platforms: %i[ mingw mswin x64_mingw jruby ]

# Reduces boot times through caching; required in config/boot.rb
gem "bootsnap", require: false

# Use Sass to process CSS
# gem "sassc-rails"
```

Рисунок 3.21 – Список залежностей проєкту

Бібліотека parsergem була додана в список бібліотек для забезпечення її використання в рамках проєкту. Parsergem є спеціалізованою бібліотекою, розробленою для парсингу та імпорту шаблонів цільового сайту. Вона забезпечує функціонал для аналізу структури вебсторінок, вилучення контенту та підготовки даних для подальшої обробки в Rails-додатку.

Додавання бібліотеки в Gemfile дозволяє автоматично завантажувати та встановлювати її разом з іншими залежностями проєкту, використовуючи команду bundle install. Це спрощує процес управління залежностями та забезпечує, що всі необхідні компоненти завжди будуть доступні під час розробки та тестування.

На рисунку 3.22 зображено виконання команди bundle install, яка використовується для встановлення всіх залежностей, зазначених у файлі Gemfile. Команда bundle install завантажує та встановлює необхідні бібліотеки (геми) для проєкту, забезпечуючи таким чином їх доступність для розробки та виконання додатка.

На екрані видно, як команда спочатку отримує метадані з репозиторію rubygems.org, потім завантажує та встановлює різні геми, включаючи aws-partitions, aws-sdk-core, та aws-sdk-translate. Після завершення процесу встановлення відображається повідомлення, що 16 залежностей з Gemfile були успішно встановлені, а також загальна кількість встановлених гемів становить 85.

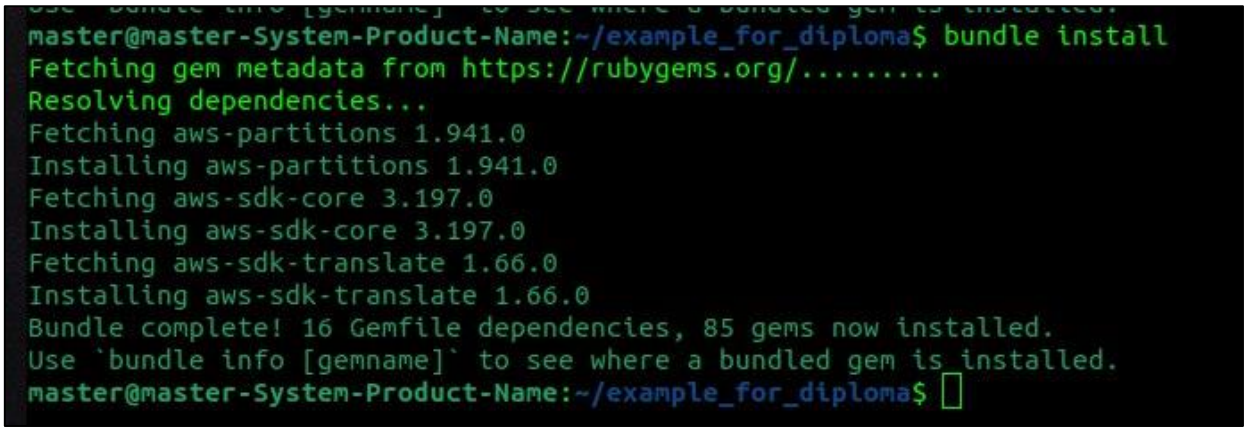

Рисунок 3.22 – Процес встановлення залежностей проєкту

Для імпорту цільового сайту був обраний сайт profistylegroup.com. Цей вебресурс був вибраний завдяки його чіткій структурі та наявності sitemap, що спрощує процес імпорту та аналізу вмісту. Сайтмап (карта сайту) profistylegroup.com складається з п’яти сторінок, що охоплюють ключові розділи вебресурсу. На рисунку 3.23 представлено скріншот головної сторінки сайту, який дає загальне уявлення про його дизайн та основний контент. Це дозволяє продемонструвати загальний вигляд та навігацію сайту перед початком процесу імпорту.

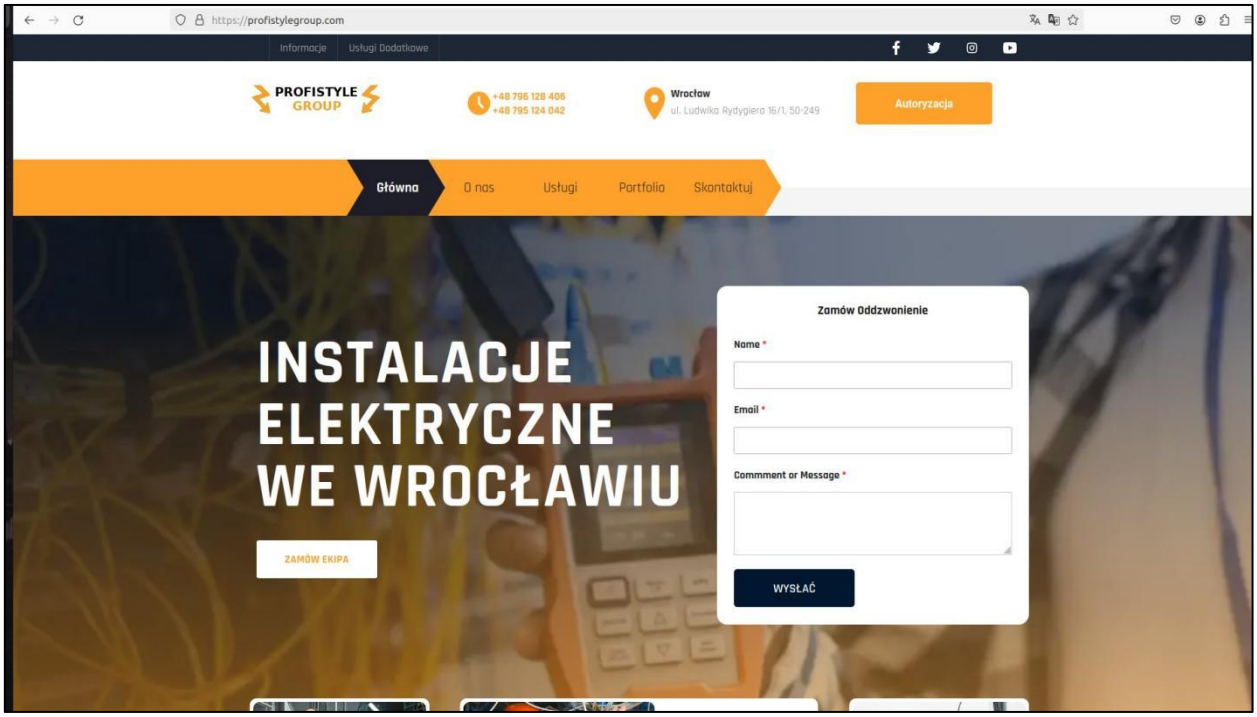

Рисунок 3.23 – Головна сторінка цільового сайту

Далі за допомогою команди запускаємо генератор інструменту імпорту шаблону цільового сайту, задаємо параметр «**--target-url**» в якому вказуємо доменне ім’я цільового сайту.

Після цього генератор зчитавши sitemap цільового сайту, створив структуру сторінок у rails застосунку, а також налаштував маршрутизацію сторінок. Потім за допомогою бібліотеки Nokogiri генератор почав копіювати контент сторінок у Rails застосунок, як це показано на рисунку 3.24.

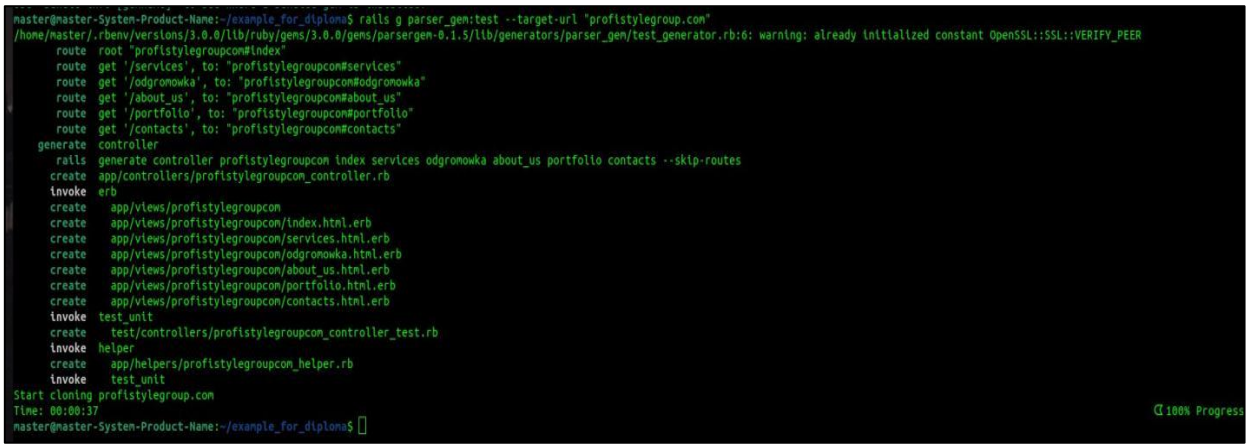

Рисунок 3.24 – Робота інструмента для імпорту шаблону цільового сайту

На рисунку 3.25 показано результат імпорту сторінки index.html, було імпортовано мета-теги, html контент сторінки, а також збережено та підключено css-таблиці стилів.

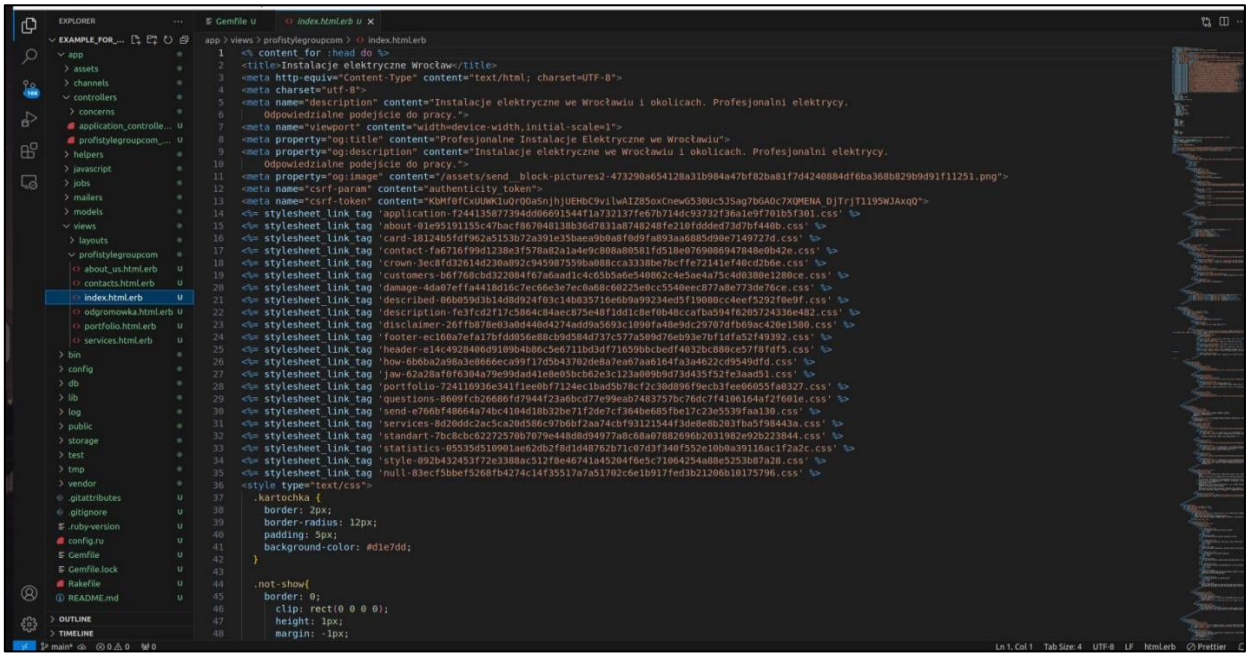

Рисунок 3.25 – Імпортований код сторінки index.html

Для перевірки результатів імпорту був запущений локальний сервер за допомогою вбудованої в Rails бібліотеки Puma, за адресою localhost:3000 став доступний rails застосунок в який було імпортовано шаблон цільового сайту, як це показано на рисунку 3.26.

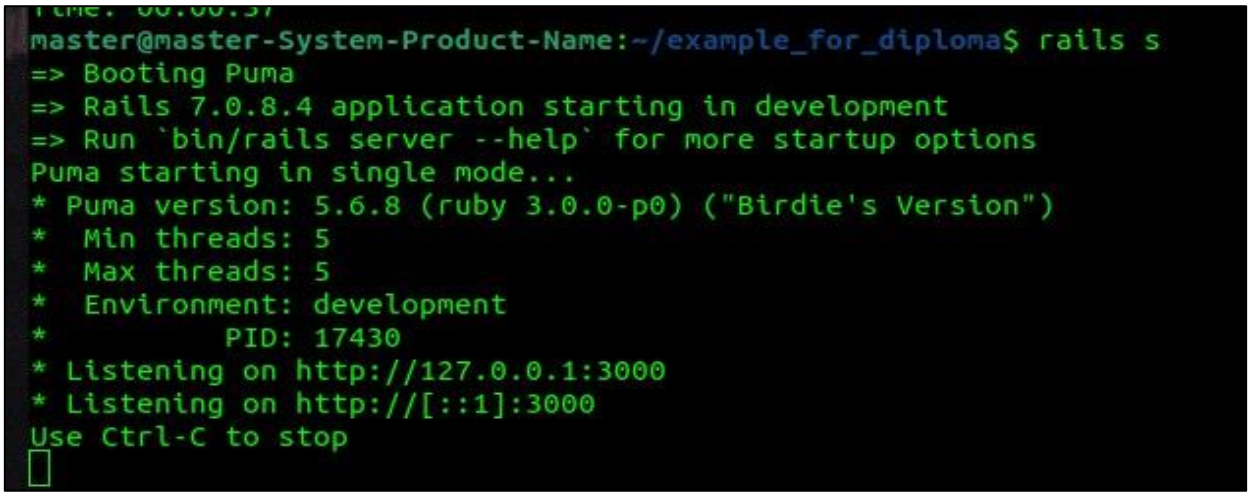

Рисунок 3.26 – Запуск локального сервера для тестування

Результат імпорту шаблону цільового сайту можна побачити на рисунку 3.27. В результаті було успішно імпортовано контент цільового сайту.

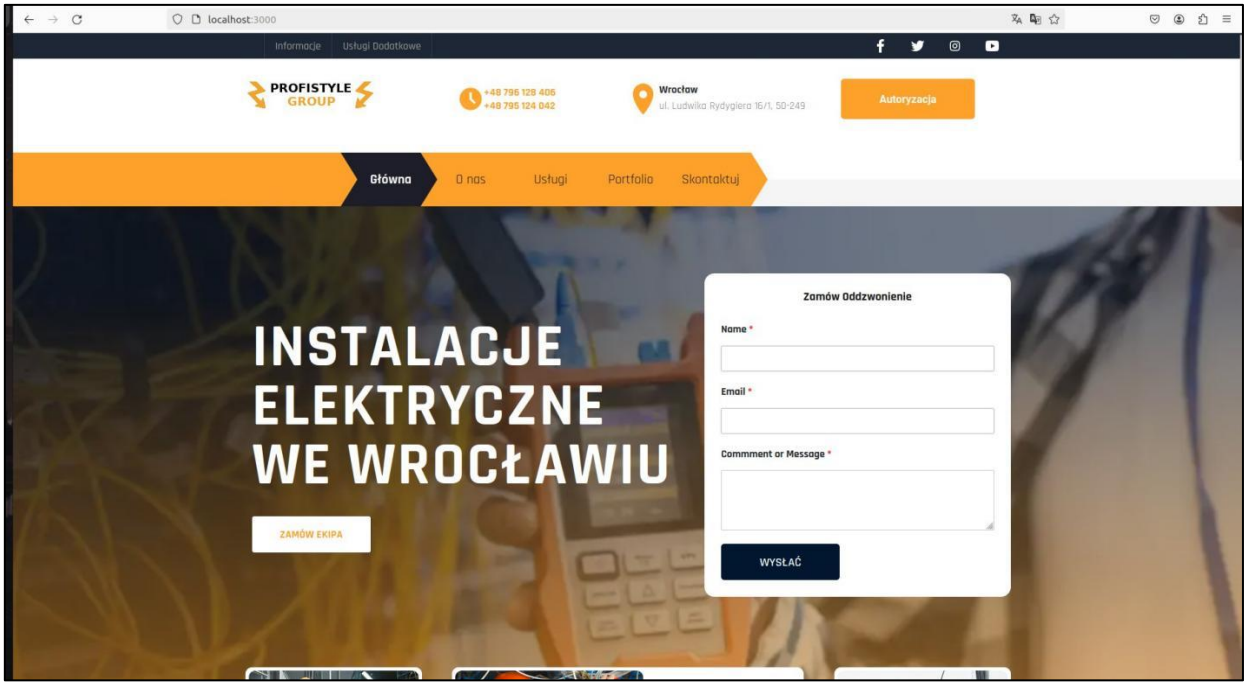

Рисунок 3.27 – Імпортована головна сторінка

3.4 Висновки до розділу 3

Реалізація інструменту імпорту шаблону цільового сайту була завершена успішно. Розроблений інструмент показав свою ефективність у

забезпеченні автоматичного перенесення структурної та контентної частин вебсайтів у нове середовище на платформі Rails. В результаті було досягнуто значного зменшення часу та зусиль, необхідних для перенесення вебсайтів, що раніше вимагало б значної ручної роботи.

Інструмент забезпечив виконання основних завдань, включаючи завантаження та парсинг файлу sitemap.xml, обробку HTML контенту сторінок, генерацію маршрутів та контролерів, інтеграцію з зовнішніми сервісами для автоматичного перекладу контенту, а також обробку зовнішніх ресурсів. Кожен з цих компонентів працював у відповідності до вимог, забезпечуючи повноту та точність імпорту вебсайту.

Використання бібліотек OpenURI та Nokogiri дозволило ефективно реалізувати завантаження та парсинг файлів і HTML контенту. Створення маршрутів та контролерів на основі даних з файлу sitemap.xml забезпечило логічну організацію нових сторінок сайту. Інтеграція з Amazon Translate продемонструвала успішну роботу з API для автоматичного перекладу, що значно полегшило управління багатомовними версіями сайту.

Однією з основних переваг розробленого інструменту є його гнучкість та масштабованість. Завдяки використанню часткових представлень (partials) вдалося значно зменшити дублювання коду та спростити процес редагування загальних елементів сторінок. Крім того, автоматизація процесу перенесення контенту та структурних елементів дозволила знизити ризик помилок, які могли б виникнути під час ручного перенесення.

Незважаючи на успіх реалізації, інструмент має певні обмеження. Наприклад, у деяких випадках можуть виникати труднощі з обробкою специфічних або дуже складних структур сторінок, які можуть вимагати додаткового налаштування. Крім того, для коректної роботи з API зовнішніх сервісів необхідно враховувати можливі обмеження та політики використання цих сервісів.

Проведені тести показали високу точність та надійність роботи інструменту. Він успішно справлявся з імпортом шаблонів різних сайтів,

зберігаючи їх структуру та контент у новому середовищі. Автоматичне створення маршрутів та контролерів, а також інтеграція з зовнішніми сервісами працювали відповідно до очікувань, забезпечуючи необхідну функціональність.

Реалізація інструменту імпорту шаблону цільового сайту продемонструвала значні переваги в автоматизації процесу перенесення вебсайтів на платформу Rails. Висока ефективність та зручність використання цього інструменту можуть суттєво спростити роботу розробників та зменшити час, необхідний для перенесення сайтів. Враховуючи результати тестування та отримані переваги, можна зробити висновок, що розроблений інструмент є корисним та ефективним рішенням для задач імпорту вебсайтів.

ВИСНОВКИ

Дипломна робота була спрямована на розробку інструменту для автоматичного перенесення структури та контенту вебсайтів на платформу Ruby on Rails. В ході роботи були вирішені різноманітні завдання, від початкового аналізу структури та контенту оригінального сайту до реалізації функціоналу інструменту та його тестування.

Під час проєктування була розроблена структура інструменту та визначені його основні функціональні вимоги. Було приділено увагу важливим компонентам, таким як завантаження та парсинг контенту, генерація маршрутів та контролерів, а також інтеграція з зовнішніми сервісами.

Під час реалізації було створено ряд модулів та функціоналу для роботи зі структурою та контентом вебсайтів. Ці модулі були ретельно протестовані для забезпечення їх коректної роботи та відповідності вимогам.

Результатом роботи став реалізований інструмент, який забезпечує автоматичний перенесення структури та контенту вебсайтів на платформу Rails. Інструмент проявив свою ефективність та надійність під час тестування, що свідчить про його потенційну корисність для розробників.

У майбутньому інструмент може бути розширений для підтримки додаткових функцій та інтеграції з іншими сервісами. Також можливе покращення швидкості та продуктивності за допомогою оптимізації алгоритмів та удосконалення процесів роботи з великими обсягами даних.

Розроблений інструмент може значно полегшити роботу розробників та зменшити час, необхідний для перенесення вебсайтів на нову платформу. Це може мати позитивний вплив на продуктивність та ефективність розробничих процесів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Colyer T., Harvie A. Using JRuby: Bringing Ruby to Java. Pragmatic Bookshelf, 2011. 450 p.
2. Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley Professional Computing Series) / R. Helm et al. Addison-Wesley Professional, 1995. 395 p.
3. Ford M., Sharkey P. Design Patterns in Ruby. Addison-Wesley Professional, 2009. 384 p.
4. Freeman E., Robson E. Head First Design Patterns. O'Reilly Media, 2004. 694 p.
5. Hart K. Agile Web Development with Rails 6. Pragmatic Bookshelf, 2019. 520 p.
6. Hartl M. Ruby on Rails Tutorial: Learn Web Development with Rails (4th Edition) (Addison-Wesley Professional Ruby Series). Addison-Wesley Professional, 2016. 816 p.
7. Hutton D. M. Clean Code: A Handbook of Agile Software Craftsmanship20092Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice-Hall, 2008. Kybernetes. 2009. Vol. 38, no. 6. P. 1035. URL: https://doi.org/10.1108/03684920910973252 (дата звернення: 15.04.2024)
8. Overview. Nokogiri. URL: https://nokogiri.org/index.html (дата звернення: 25.04.2024).
9. Pollice G., Berk S., Cangiano J. Software Development with Ruby. Addison-Wesley Professional, 2009. 480 p.
10. Офіційна документація Ruby on Rails. URL: https://rubyonrails.org/ (дата звернення: 28.03.2024).
11. Tate B., Hibbs C., Armenatzoglou B. Ruby on Rails: Up and Running, 2nd Edition. O'Reilly Media, 2016. 234 p.
12. Thomas D., Hunt A. Programming Ruby 1.9 & 2.0: The Pragmatic Programmers'

Guide. Pragmatic Bookshelf, 2013. 888 p.

13. Wold P., Lehne O. A., Reenskaug T. Working With Objects: The Ooram Software Engineering Method. Manning Pubns Co, 1995. 366 p.