МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»

Направление подготовки
09.03.01 Информатика и вычислительная техника

Направленность (профиль)
«Технологии разработки программного обеспечения»

Выпускная квалификационная работа

Разработка фреймворка для генерации статических сайтов

Обучающейся 4 курса
очной формы обучения
Крючковой Анастасии Сергеевны

Руководитель выпускной квалификационной работы:
кандидат физико-математических наук,
доцент кафедры ИТиЭО
Жуков Николай Николаевич

Санкт-Петербург
2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время генерация статических сайтов становится все более востребованной в различных областях веб-разработки. Навык работы с такими инструментами является крайне важным для многих специалистов, занимающихся созданием и поддержкой веб-контента. Именно поэтому изучение и использование фреймворков для генерации статических сайтов стало неотъемлемой частью образовательного процесса в вузах и даже некоторых школах.

Однако, существующие решения часто ограничены в функциональности и гибкости. Это приводит к двум основным проблемам: сложности в адаптации под специфические требования проектов и значительным временным затратам на настройку и развертывание. Решение этих проблем обосновывает актуальность работы – разработка нового фреймворка для генерации статических сайтов, который был заказан кафедрой информационных технологий, института информационных технологий и технологического образования.

Цель дипломного проекта – разработка фреймворка для генерации статических сайтов.

Предметом исследования является процесс разработки и оптимизации фреймворка для генерации статических сайтов, включая его архитектуру, функциональные возможности и методы интеграции с современными веб-технологиями.

Для достижения поставленной цели требуется выполнить следующие задачи:

1) Анализ существующих решений: Изучить текущие фреймворки для генерации статических сайтов, их преимущества и недостатки.

2) Определение требований: Сформулировать функциональные и нефункциональные требования к новому фреймворку.

3) Проектирование архитектуры: Разработать архитектуру фреймворка, обеспечивающую гибкость и расширяемость.

4) Реализация прототипа: Создать прототип фреймворка с базовыми функциями генерации статических сайтов.

5) Интеграция с веб-технологиями: Обеспечить поддержку современных веб-технологий и инструментов.

6) Тестирование и оптимизация: Провести тестирование производительности и оптимизацию фреймворка.

7) Документация и обучение: Подготовить документацию и обучающие материалы для пользователей и разработчиков.

Данная работа состоит из введения, первой главы, включающей в себя анализ имеющихся фреймворков для генерации статических сайтов и второй главы, содержащей в себе проектирование и разработку фреймворка.

INTRODUCTION

Nowadays, static site generation is becoming increasingly in demand in various areas of web development. The ability to work with such tools is extremely important for many professionals involved in the creation and maintenance of web content. Therefore, the study and use of frameworks for generating static sites have become an integral part of the educational process in universities and even some schools.

However, existing solutions are often limited in functionality and flexibility. This leads to two main problems: difficulties in adapting to specific project requirements and significant time costs for setup and deployment. Solving these problems justifies the relevance of the project – the development of a new framework for generating static sites, commissioned by the Department of Information Technology, Institute of Information Technology and Technological Education.

The aim of the diploma project is to develop a framework for generating static sites.

The subject of the research is the process of developing and optimizing the framework, including its architecture, functional capabilities, and integration methods with modern web technologies.

To achieve the set goal, the following tasks need to be completed:

1) Analyze existing solutions: Study current frameworks for generating static sites, their advantages, and disadvantages.

2) Define requirements: Formulate functional and non-functional requirements for the new framework.

3) Design architecture: Develop a framework architecture that ensures flexibility and scalability.

4) Implement a prototype: Create a prototype of the framework with basic static site generation functions.

5) Integrate with web technologies: Ensure support for modern web technologies and tools.

6) Test and optimize: Conduct performance testing and optimization of the framework.

7) Document and train: Prepare documentation and training materials for users and developers.

This work consists of an introduction, the first chapter, which includes an analysis of existing frameworks for generating static sites and the definition of functional and non-functional requirements, the second chapter, which contains framework design and development.

ГЛАВА I. ИССЛЕДОВАНИЕ СУЩЕСТВУЮЩИХ ФРЕЙМВОРКОВ ДЛЯ ГЕНЕРАЦИИ СТАТИЧЕСКИХ САЙТОВ

1.1 Назначение и преимущество фреймворков для генерации статических сайтов

Фреймворки для генерации статических сайтов (Static Site Generators, SSG) представляют собой специализированные программные решения, предназначенные для преобразования исходных файлов в форматах Markdown, reStructuredText, JSON и других в статические веб-страницы. Данная технология получила широкое распространение благодаря совокупности технических и экономических преимуществ, которые она предоставляет разработчикам и владельцам веб-ресурсов.

В контексте оптимизации производительности веб-приложений, SSG демонстрируют значительные преимущества. Предварительный рендеринг HTML-страниц существенно сокращает время загрузки контента, а отсутствие необходимости в серверной обработке запросов минимизирует нагрузку на инфраструктуру. Это приводит к значительному улучшению пользовательского опыта за счет сокращения времени отклика системы.

С точки зрения информационной безопасности, статические сайты обладают существенными преимуществами по сравнению с динамическими решениями. Отсутствие серверной части и базы данных исключает целый класс уязвимостей, включая SQL-инъекции и многие типы XSS-атак. Статическая природа контента значительно усложняет возможность его несанкционированной модификации.

Архитектура SSG обеспечивает эффективное разделение контента и представления, что существенно упрощает процесс разработки и поддержки веб-ресурсов. Использование шаблонов гарантирует консистентность дизайна, а поддержка различных форматов ввода позволяет разработчикам выбирать наиболее подходящие инструменты для создания контента.

Масштабируемость статических сайтов достигается за счет отсутствия зависимостей от серверных компонентов, что обеспечивает высокую отказоустойчивость системы. При увеличении нагрузки масштабирование

осуществляется путем простого добавления серверных мощностей без необходимости модификации архитектуры.

Экономическая эффективность SSG проявляется в снижении затрат на хостинг благодаря минимальным требованиям к серверной инфраструктуре. Это позволяет использовать недорогие или даже бесплатные хостинг-решения, что особенно актуально для небольших проектов и стартапов.

В контексте поисковой оптимизации (SEO) статические сайты демонстрируют превосходные результаты благодаря быстрой загрузке страниц и чистой структуре HTML-кода. Возможность детальной настройки метаданных для каждой страницы позволяет оптимизировать контент под требования поисковых систем.

Современные SSG обеспечивают широкие возможности интеграции с актуальными инструментами разработки и системами непрерывной интеграции и доставки (CI/CD), что делает их привлекательным решением для профессиональной разработки. Данные преимущества обуславливают широкое применение SSG в разработке блогов, новостных порталов, корпоративных сайтов, а также персональных веб-ресурсов и портфолио.

1.2 Самые популярные фреймворки для генерации статических сайтов на языке программирования Python и их сравнительная характеристика

По данным источника https://dev.to/stokry/5-best-static-site-generators-in-python-56a8, 5 наиболее популярных фреймворка для генерации статических сайтов это: Pelican, MkDocs, Nikola, Sphinx, Lektor. Рассмотрим документацию к каждому фреймворку и составим их сравнительную таблицу. [1][2][3][4][5].

Таблица 1 демонстрирует сравнение перечисленных выше табличных процессоров.

Таблица 1. – Сравнение фреймворков

Характеристика	Pelican	MkDocs	Nikola	Sphinx	Lektor
Основное назначение	Блоги и новостные сайты	Документация	Универсальный	Техническая документация	Универсальный
Формат контента	Markdown, reStructuredText, AsciiDoc	Markdown	Markdown, reStructuredText, HTML, Jupyter Notebooks	reStructuredText, Markdown	Markdown, HTML
Шаблонизатор	Jinja2	Jinja2	Mako, Jinja2	Jinja2	Jinja2
Поддержка тем	Да	Да	Да	Да	Да
Встроенные темы	3+	2+	10+	1+	5+
Поддержка плагинов	Да	Да	Да	Да	Да
Поиск	Да	Да	Да	Да	Да
Многоязычность	Да	Да	Да	Да	Да
Поддержка тегов	Да	Да	Да	Да	Да
Поддержка категорий	Да	Нет	Да	Нет	Да
API документации	Нет	Да	Нет	Да	Нет
Поддержка комментариев	Через плагины	Нет	Через плагины	Нет	Через плагины
Поддержка RSS	Да	Нет	Да	Нет	Да
Поддержка Sitemap	Да	Да	Да	Да	Да

Характеристика	Pelican	MkDocs	Nikola	Sphinx	Lektor
Поддержка MathJax	Через плагины	Да	Да	Да	Через плагины
Поддержка Code Highlighting	Да	Да	Да	Да	Да
Поддержка изображений	Да	Да	Да	Да	Да
Поддержка видео	Да	Да	Да	Да	Да
Поддержка PDF	Через плагины	Нет	Да	Да	Нет
Поддержка LaTeX	Через плагины	Да	Да	Да	Через плагины
Поддержка Webpack	Нет	Да	Нет	Нет	Да
Поддержка Docker	Да	Да	Да	Да	Да
Поддержка CI/CD	Да	Да	Да	Да	Да
Активность разработки	Средняя	Высокая	Средняя	Высокая	Низкая
Сложность настройки	Средняя	Низкая	Средняя	Средняя	Низкая
Кривая обучения	Средняя	Низкая	Средняя	Высокая	Низкая

Вывод по главе 1

В данной главе было рассмотрено назначение и преимущество использования фреймворков для генерации статических сайтов, а также самые популярные из них на языке программирования Python. Впоследствии была составлена их сравнительная таблица, на основе которой можно составить функциональные и нефункциональные требования к собственному фреймворку.

ГЛАВА II. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ФРЕЙМВОРКА ДЛЯ ГЕНЕРАЦИИ СТАТИЧЕСКИХ САЙТОВ

2.1 Анализ требований к фреймворку

На основе анализа самых популярных фреймворков для генерации статических сайтов на Python можно сформулировать функциональные и нефункциональные требования к разработке. Данные требования основаны на исследовании существующих решений и отражают современные тенденции в области веб-разработки.

2.1.1 Функциональные требования

Функциональные требования к системе можно разделить на несколько ключевых компонентов. Ядро системы должно обеспечивать базовую функциональность посредством реализации конфигурации в формате TOML, что обеспечивает превосходную читаемость и структурированность по сравнению с традиционными форматами YAML и JSON. Система кэширования призвана оптимизировать процесс сборки, существенно сокращая время генерации при повторных запусках. Критически важным является корректная реализация маршрутизации для генерации URL-адресов и функционирование сервера разработки с поддержкой hot-reload. Система должна обеспечивать гибкое управление метаданными страниц, поддерживать многоязычность и включать механизмы организации контента через систему тегов и категорий.

В области обработки контента фреймворк должен обеспечивать поддержку основных форматов представления данных, включая Markdown и HTML. Особое внимание уделяется обработке медиа-контента, включая оптимизацию и преобразование различных форматов изображений. Расширенные возможности должны включать поддержку математических формул и диаграмм, что критически важно для создания технической документации и образовательных ресурсов.

Система шаблонизации должна предоставлять мощный механизм создания страниц посредством реализации наследования шаблонов, что позволяет создавать базовые шаблоны и расширять их для конкретных страниц. Фильтры и теги должны обеспечивать богатые возможности для манипуляции данными в шаблонах. Управление темами и поддержка адаптивного дизайна являются необходимыми условиями для создания современных, отзывчивых веб-ресурсов.

Административный интерфейс должен предоставлять комплексный набор инструментов для управления сайтом, включая предварительный просмотр изменений, управление контентом и темами, мониторинг процесса сборки и управление плагинами. Это обеспечивает полный контроль над сайтом без необходимости прямого редактирования файлов.

Система развертывания должна поддерживать инкрементальную сборку, оптимизируя процесс публикации изменений. Требуется реализация оптимизации ресурсов, включая минификацию CSS и JavaScript, генерацию sitemap.xml для SEO, поддержку CDN и интеграцию с системами непрерывной интеграции и доставки.

Фреймворк должен предоставлять расширенный API для создания плагинов, включая управление зависимостями, версионирование, подробную документацию и механизмы тестирования.

2.1.1 Нефункциональные требования

Производительность системы определяется временными и ресурсными характеристиками. Время сборки сайта не должно превышать 30 секунд для 1000 страниц, время отклика сервера разработки ограничено 100 мс, время загрузки страницы в браузере - 2 секунды, а время обработки одного файла контента - 50 мс. Ресурсные требования ограничивают использование оперативной памяти 500 МБ при сборке и CPU 80%, с обязательной оптимизацией размера сгенерированных файлов.

Надежность системы обеспечивается через корректную обработку ошибок, информативные сообщения об ошибках, механизмы восстановления после сбоев и

атомарность критических операций. Требуется строгая валидация всех входных данных и подробное логирование операций.

Безопасность системы включает защиту данных, контроль доступа и безопасность файлов. Реализуется через санитизацию пользовательских данных, шифрование конфиденциальной информации, защиту от XSS и CSRF атак, систему аутентификации и авторизации, а также защиту файловой системы.

Масштабируемость обеспечивается поддержкой горизонтального и вертикального масштабирования, включая распределенную сборку, работу в кластере, балансировку нагрузки, эффективное использование ресурсов, поддержку многопроцессорной обработки и оптимизацию работы с большими файлами.

Удобство использования определяется простотой установки и настройки (не более 5 минут), интуитивно понятной конфигурацией, подробной документацией и удобным интерфейсом. Система должна поддерживать многоязычность интерфейса и адаптивный дизайн.

Совместимость обеспечивается поддержкой Python 3.8 и выше, различных операционных систем, кроссплатформенностью и интеграцией с Docker, CI/CD системами и Git.

Поддерживаемость системы достигается через модульную архитектуру, чистый и документированный код, следование стандартам PEP 8, использование типизации, высокое покрытие кода тестами, семантическое версионирование и поддержку обратной совместимости.

Экономичность системы проявляется в эффективном использовании ресурсов, минимальных требованиях к системе, оптимизации использования CPU и памяти, минимизации внешних зависимостей и оптимизации сетевого взаимодействия.

Представленный набор требований формирует комплексную основу для разработки современного, надежного и эффективного фреймворка для генерации статических сайтов, способного удовлетворить потребности как небольших проектов, так и крупных корпоративных решений.

2.2 Архитектурное проектирование

При проектировании будем придерживаться модульного подхода, где каждый компонент системы имеет четко определенную ответственность и интерфейсы взаимодействия, так как это обеспечивает масштабируемость и гибкость, надежность и отказоустойчивость, возможность замены компонентов системы, а также расширяемость через плагины.

2.2.1 Проектирование движка

Архитектура движка статического генератора сайтов базируется на модульной структуре, включающей следующие ключевые компоненты: Engine (ядро системы), Config (модуль конфигурации), Site (управление структурой сайта), Page (обработка страниц), Router (маршрутизация) и TemplateEngine (система шаблонизации). Взаимодействие между компонентами реализуется посредством четко определенных интерфейсов и протоколов обмена данными.

Процесс инициализации системы начинается с инициализации Engine, который последовательно загружает конфигурацию проекта через модуль Config. На основе полученных настроек создается экземпляр Site, который определяет структуру директорий проекта. Модуль Site выполняет сканирование исходных файлов и создает соответствующие объекты Page для каждого обнаруженного файла контента. Router генерирует корректные пути для каждой страницы на основе предоставленных метаданных, а TemplateEngine осуществляет финальный рендеринг страниц с применением соответствующих шаблонов. Engine координирует весь процесс сборки, управляя вызовом обработчиков плагинов в определенной последовательности.

Для обеспечения расширяемости системы разрабатывается модульная система плагинов с четко определенным интерфейсом взаимодействия. Система плагинов реализует механизм хуков, которые интегрируются в различные этапы процесса построения сайта, что позволяет расширять функциональность без необходимости модификации ядра системы.

Процесс генерации статического сайта реализуется в виде последовательности взаимосвязанных этапов:

1) Загрузка и валидация конфигурации проекта.
2) Инициализация и настройка плагинов.
3) Выполнение предварительных обработчиков (pre_build).
4) Сканирование и индексация исходных файлов контента.
5) Парсинг содержимого и извлечение метаданных.
6) Последовательная обработка контента через цепочку плагинов.
7) Генерация выходных путей для страниц.
8) Рендеринг HTML-страниц с применением шаблонов.
9) Копирование и оптимизация статических файлов.
10) Выполнение финальных обработчиков (post_build).

Для оптимизации производительности системы разрабатывается многоуровневая система кэширования, которая минимизирует повторную обработку неизмененных файлов. Система кэширования реализуется на следующих уровнях:

- Кэширование результатов парсинга Markdown и reStructuredText документов;
- Кэширование преобразованного HTML-контента;
- Кэширование результатов рендеринга шаблонов;
- Реализация инкрементальной сборки с отслеживанием изменений в исходных файлах.

2.2.2 Проектирование парсеров

Система парсеров представляет собой критически важный компонент архитектуры StaticFlow, обеспечивающий преобразование исходного контента в HTML-представление. Эффективность данной системы напрямую влияет на

производительность и надежность всего фреймворка, особенно в контексте интеграции с блочным редактором.

Архитектура системы парсеров базируется на абстрактном классе ContentParser, который определяет унифицированный интерфейс для всех реализаций парсеров. Данный класс инкапсулирует следующие ключевые компоненты:

1) Базовый функционал обработки контента реализуется через три основных метода:

- parse() - осуществляет первичное преобразование контента,
- parse_with_metadata() - обеспечивает обработку контента с учетом метаданных,
- validate() - выполняет валидацию входных данных.

2) Система конфигурации предоставляет гибкие механизмы настройки:

- параметры обработки блоков контента,
- настройки форматирования,
- опции рендеринга.

Для оптимизации производительности разрабатывается многоуровневая система кэширования, которая включает:

- кэширование результатов парсинга,
- механизм инвалидации кэша при модификации контента,
- оптимизацию повторной обработки неизмененных блоков.

Критически важным аспектом является обеспечение двусторонней синхронизации с блочным редактором, которая реализуется через:

- преобразование структуры блоков редактора в HTML-представление,
- обратное преобразование HTML в структуру блоков,
- сохранение форматирования и стилевых атрибутов.

Система валидации обеспечивает целостность данных посредством:

- проверки корректности структуры блоков,
- валидации атрибутов элементов,
- контроля правильности вложенности элементов.

Безопасность обработки контента обеспечивается через реализацию следующих механизмов:

- очистку потенциально опасного HTML-кода,
- валидацию URL и ссылок,
- защиту от XSS-атак.

2.2.3 Проектирование CLI

Интерфейс командной строки (CLI) является основным способом взаимодействия пользователя с системой StaticFlow для создания и запуска серввера разработки проекта. В рамках проектирования CLI предлагается реализовать следующие основные команды:

- create - создание нового проекта,
- serve - запуск локального сервера разработки.

Для улучшения пользовательского опыта команда “create” должна предлагать интерактивный режим с пошаговым созданием нового проекта с выбором опций, а команда “serve” должна предоставлять систему логирования.

2.2.4 Проектирование панели администратора

Разработка административной панели представляет собой критически важный компонент системы управления статическим сайтом, обеспечивающий эффективное создание, редактирование и публикацию контента. Архитектура панели администратора должна обеспечивать оптимальный баланс между простотой использования и функциональной полнотой, позволяя управлять всеми аспектами веб-ресурса.

В основу проектирования административной панели заложены четыре ключевых архитектурных принципа. Первый принцип - легковесность, подразумевает интеграцию в статический генератор без существенного усложнения системной архитектуры. Второй принцип - модульность, обеспечивает логическое разделение функциональности на независимые компоненты. Третий принцип - расширяемость, позволяет простое добавление новых функциональных возможностей. Четвертый принцип - отзывчивость, гарантирует адаптивный интерфейс с минимальным временем отклика.

Технологический стек административной панели формируется из трех основных компонентов. Серверная часть реализована на Python с использованием асинхронного веб-фреймворка aiohttp, что обеспечивает высокую производительность при обработке запросов. Для генерации HTML-страниц используется система шаблонизации Jinja2, предоставляющая мощные возможности для создания динамического контента. Клиентская часть построена на JavaScript с минимальным набором зависимостей, что обеспечивает быструю загрузку и отзывчивость интерфейса.

Архитектура административной панели включает четыре ключевых модуля: модуль управления контентом, модуль системных настроек, модуль развертывания

и систему предварительного просмотра. Каждый из этих модулей выполняет специфические функции и взаимодействует с другими компонентами системы.

Модуль системных настроек обеспечивает комплексное управление конфигурацией сайта. Он включает управление базовыми настройками, такими как название сайта, URL и описание, конфигурацию многоязычности, параметры маршрутизации и выбор темы с шаблонами. Все эти параметры могут быть изменены через удобный пользовательский интерфейс.

Модуль развертывания реализует полный цикл публикации сайта на GitHub Pages. Он обеспечивает конфигурацию репозитория и ветки, настройку пользовательского домена, безопасное хранение учетных данных и ведение истории публикаций. Особое внимание уделено безопасности хранения конфиденциальной информации.

Система предварительного просмотра предоставляет возможность мгновенного просмотра изменений. Она обеспечивает рендеринг страниц с применением актуальных шаблонов, отображение в отдельном окне браузера и актуализацию метаданных. Это позволяет разработчикам и контент-менеджерам оперативно оценивать результаты своих изменений.

Взаимодействие административной панели с основным движком StaticFlow осуществляется через четыре основных интерфейса. Интерфейс доступа к конфигурации обеспечивает чтение и модификацию настроек сайта. Интерфейс управления контентом предоставляет функциональность создания и редактирования страниц. Интерфейс генерации сайта управляет процессом сборки, а интерфейс предпросмотра обеспечивает рендеринг страниц без сохранения на диск.

2.2.5 Проектирование блочного редактора в панеле администратора

Блочный редактор контента представляет собой критически важный компонент административной панели, обеспечивающий создание структурированного контента через интуитивно понятный интерфейс.

Архитектура редактора ориентирована на визуальное представление публикации, что позволяет пользователям эффективно работать с контентом.

Функциональные возможности редактора включают комплексный набор операций с контентными блоками. Пользователи могут добавлять, удалять, перемещать и редактировать различные типы блоков, включая параграфы, заголовки, списки, цитаты и медиа-элементы. Интерфейс редактора обеспечивает работу в визуальном режиме, максимально приближенном к конечному виду страницы. Особое внимание уделено современному UX/UI, включающему поддержку drag & drop, быстрый доступ к типам блоков и адаптивность под различные устройства.

Архитектура блочного редактора предусматривает реализацию трех основных категорий контентных блоков. Первая категория - текстовые блоки, включает параграфы, заголовки различных уровней (H1, H2, H3), маркированные и нумерованные списки, цитаты и блоки кода. Вторая категория - медиа-блоки, обеспечивает работу с изображениями, аудио и видео контентом. Третья категория - специальные блоки, включает математические формулы на основе KaTeX, диаграммы на основе Mermaid и информационные блоки различных типов (info, warning, danger).

Редактор реализуется как самостоятельный JavaScript-компонент, инициализируемый внутри контейнера административной панели. Архитектура компонента включает несколько ключевых модулей:

BlockEditor представляет собой основной класс, ответственный за управление состоянием редактора, списком блоков и их рендерингом. BlockActions обеспечивает набор функций для манипуляции блоками, включая добавление, удаление, перемещение и обновление содержимого. BlockTypeMenu реализует всплывающее меню для выбора типа нового блока, а BlockFactory отвечает за создание DOM-элементов различных типов блоков. Отдельные модули блоков-контейнеров обеспечивают специфическую функциональность для каждого типа блока.

Управление состоянием редактора осуществляется через экземпляр класса BlockEditor, который хранит массив блоков, информацию о выделенном блоке и режимы предпросмотра. Все изменения состояния немедленно отражаются в DOM через функцию render(), что обеспечивает мгновенное обновление интерфейса при любых модификациях контента.

2.2.6 Проектирование системы шаблонов

Проектирование системы шаблонов для фреймворка StaticFlow представляет собой комплексную задачу, требующую реализации модульной архитектуры на основе шаблонизатора Jinja2. Ключевым компонентом системы является класс TemplateEngine, ответственный за инициализацию окружения Jinja2 и управление жизненным циклом шаблонов.

Архитектура системы шаблонов базируется на механизме наследования, реализованном через систему блоков Jinja2. Базовый шаблон base.html определяет фундаментальную структуру страницы, предоставляя дочерним шаблонам возможность расширения через переопределение ключевых блоков. К таким блокам относятся заголовок страницы, дополнительные элементы в head и основное содержимое, что обеспечивает гибкость при создании различных типов страниц.

Важным аспектом архитектуры является поддержка макросов и включений, которые обеспечивают создание переиспользуемых компонентов шаблонов. Система предоставляет расширенный доступ к контексту шаблонов, включая глобальные переменные, параметры страницы и пользовательские данные. Это позволяет создавать динамические и адаптивные шаблоны, сохраняя при этом чистоту кода и возможность повторного использования компонентов.

Безопасность системы обеспечивается через комплексный набор механизмов. Реализовано автоматическое экранирование HTML и XML, что предотвращает XSS-атаки. Система включает валидацию входных данных и безопасную обработку пользовательского контента. Дополнительно

предоставляется возможность расширения функциональности через пользовательские фильтры и регистрацию глобальных переменных.

Интеграция системы шаблонов с основными компонентами фреймворка обеспечивает обработку метаданных страниц, управление статическими файлами и поддержку многоязычности. Шаблоны поддерживают работу с языковыми метатегами, переключателем языков, каноническими URL и альтернативными языковыми версиями, что критически важно для создания многоязычных веб-ресурсов.

Производительность системы оптимизирована через реализацию многоуровневого кэширования шаблонов, эффективную загрузку файлов и оптимизацию процесса рендеринга. Система предоставляет удобный доступ к статическим файлам через глобальную переменную static_url и автоматическую обработку путей, что упрощает работу с ресурсами и обеспечивает корректное формирование URL.

2.2.7 Проектирование системы кэширования

Система кэширования должна обеспечивать оптимизацию производительности фреймворка. Основой системы должен стать механизм кэширования на основе файловой системы, который позволяет хранить результаты обработки контента и шаблонов. Система должна поддерживать различные уровни кэширования, включая кэширование страниц, шаблонов и результатов работы плагинов.

Важным аспектом является реализация механизма инвалидации кэша, который позволяет обновлять кэш при изменении исходных данных. Система должна обеспечивать атомарность операций с кэшем и защиту от race conditions. Для оптимизации производительности необходимо реализовать асинхронное обновление кэша и механизм предварительной загрузки часто используемых данных.

2.2.8 Проектирование системы маршутизации

Система маршрутизации должна обеспечивать гибкое управление URL-адресами сайта. Основой системы должен стать механизм маршрутизации на основе правил, который позволяет определять соответствие между URL-адресами и страницами сайта. Система должна поддерживать различные типы маршрутов, включая статические, динамические и регулярные выражения.

Важным аспектом является реализация механизма генерации URL-адресов, который позволяет создавать корректные ссылки на основе параметров маршрута. Система должна обеспечивать поддержку многоязычности URL-адресов и обработку специальных случаев, таких как редиректы и обработка ошибок. Для обеспечения производительности необходимо реализовать кэширование правил маршрутизации и оптимизацию поиска маршрутов.

2.2.9 Проектирование модуля развертывания

Процесс развертывания должен состоять из следующих этапов:

1) Подготовительный этап

Процесс развертывания начинается с подготовительного этапа, который обеспечивает корректность всех необходимых условий для успешного деплоя. На этом этапе происходит тщательная валидация конфигурации развертывания, включающая проверку всех обязательных параметров: URL репозитория GitHub, имя пользователя, email для Git-коммитов и токен доступа. Особое внимание уделяется формату URL репозитория, который должен соответствовать либо HTTPS, либо SSH формату. Если в конфигурации указан CNAME для пользовательского домена, производится его валидация.

Следующим шагом является проверка наличия и доступности директории с собранным сайтом. Система проверяет существование директории public (или указанной в конфигурации), валидирует права доступа и убеждается в наличии необходимых файлов. Для обеспечения изоляции процесса развертывания создается временная рабочая директория с автоматической очисткой после завершения операций. Завершает подготовительный этап настройка

Git-окружения, включающая конфигурацию идентичности пользователя и подготовку учетных данных для аутентификации.

2) Этап инициализации репозитория

На этапе инициализации репозитория происходит создание временного Git-репозитория в подготовленной директории. Система настраивает базовые параметры Git и подготавливает окружение для работы с удаленным репозиторием. Особое внимание уделяется настройке удаленного репозитория: добавляется origin с учетом токена доступа, настраиваются параметры аутентификации и проверяется доступность удаленного репозитория.

Важной частью этого этапа является конфигурация Git-идентичности, где устанавливаются имя пользователя и email для коммитов, а также настраивается GPG-подпись, если она используется. Завершает этап проверка существования целевой ветки (gh-pages или указанной в конфигурации), что определяет дальнейшую стратегию развертывания.

3) Этап подготовки контента

Этап подготовки контента начинается с тщательной очистки рабочей директории, где удаляются все файлы, кроме Git-конфигурации. Это обеспечивает чистое состояние для размещения нового контента. Далее происходит рекурсивное копирование всех файлов из директории собранного сайта с сохранением структуры директорий и проверкой целостности скопированных файлов.

Если в конфигурации указан CNAME для пользовательского домена, система создает соответствующий файл в корневой директории. Завершает этап подготовка к коммиту, включающая проверку статуса Git-репозитория, анализ изменений и формирование сообщения коммита.

4) Этап публикации

Этап публикации является ключевым в процессе развертывания. Начинается он с добавления всех новых файлов в Git-индекс, после чего система проверяет статус добавления и валидирует изменения. Особое внимание уделяется проверке наличия реальных изменений, чтобы избежать создания пустых коммитов.

Создание коммита происходит с тщательно сформированным сообщением, включающим информацию о времени развертывания и внесенных изменениях. После успешного создания коммита система отправляет изменения в удаленный репозиторий, обрабатывая возможные конфликты и валидируя результаты push-операции.

5) Завершающий этап

Завершающий этап обеспечивает корректное завершение процесса развертывания. Система обновляет историю развертываний, добавляя новую запись с временной меткой и статусом операции. Это позволяет отслеживать все попытки развертывания и их результаты.

Особое внимание уделяется очистке временных файлов: удаляется временная директория, очищается Git-кэш и проверяется завершение всех операций очистки. Завершает процесс подробное логирование результатов развертывания, включающее запись всех успешных операций, ошибок (если они возникли) и формирование итогового отчета.

Каждый этап пайплайна включает в себя комплексную обработку ошибок и подробное логирование всех операций. Это обеспечивает надежность процесса развертывания и предоставляет необходимую информацию для отладки в случае возникновения проблем. Система спроектирована таким образом, чтобы обеспечить атомарность операций и возможность отката изменений при возникновении ошибок на любом этапе процесса.

2.2.10 Проектирование системы плагинов

При проектировании системы плагинов для фреймворка StaticFlow необходимо реализовать модульную архитектуру, которая позволит расширять базовую функциональность фреймворка без изменения его ядра. Система плагинов должна обеспечивать гибкость, безопасность и производительность при обработке контента.

В составе фреймворка необходимо предусмотреть набор встроенных плагинов, которые будут предоставлять базовую функциональность:

- SyntaxHighlightPlugin - для подсветки синтаксиса кода в контенте;
- MathPlugin - для поддержки математических формул;
- MermaidPlugin - для создания диаграмм;
- NotionBlocksPlugin - для поддержки блоков Notion;
- MediaPlugin - для обработки медиафайлов;
- CDNPlugin - для интеграции с CDN;
- SEOPlugin - для оптимизации сайта;
- SitemapPlugin - для генерации sitemap.xml;
- RSSPlugin - для создания RSS-ленты;
- MinifierPlugin - для минификации ресурсов.

Система плагинов должна обеспечивать расширяемость фреймворка через четкий API для создания новых плагинов. Этот API должен предоставлять разработчикам все необходимые инструменты и интерфейсы для реализации собственных плагинов, включая методы для обработки контента, управления жизненным циклом плагина и взаимодействия с ядром фреймворка.

Важным аспектом является возможность добавления пользовательских плагинов, что позволяет разработчикам расширять функциональность фреймворка в соответствии с конкретными требованиями проекта. Для этого система должна поддерживать динамическую загрузку плагинов и их регистрацию в фреймворке.

2.2.11 Проектирование системы CDN

Интеграция Content Delivery Network (CDN) представляет собой критически важный компонент современной веб-инфраструктуры, обеспечивающий эффективную доставку контента конечным пользователям. В контексте фреймворка StaticFlow реализация CDN является необходимым условием для обеспечения высокой производительности и масштабируемости системы.

Архитектура CDN в StaticFlow реализована в виде модульной системы, конфигурируемой через файл config.toml. Основу системы составляет трехуровневая архитектура, включающая CDNPlugin, CDNProvider и конкретные реализации провайдеров, такие как CloudflareCDN. Такая структура обеспечивает гибкость при выборе CDN-провайдера и возможность расширения функциональности.

CDNPlugin представляет собой центральный компонент системы, ответственный за управление взаимодействием с CDN-провайдером. Абстрактный класс CDNProvider определяет унифицированный интерфейс для работы с различными CDN-сервисами, что обеспечивает единообразие взаимодействия независимо от выбранного провайдера. Конкретные реализации, такие как CloudflareCDN, предоставляют специфичную для каждого сервиса функциональность.

Процесс работы CDN в StaticFlow включает несколько ключевых этапов. На этапе инициализации происходит создание экземпляра CDNPlugin, загрузка конфигурационных параметров и инициализация выбранного CDN-провайдера. Система обеспечивает обработку файлов из директорий static/ и media/, генерирует уникальные ключи для каждого файла и осуществляет загрузку на CDN с сохранением исходной структуры директорий.

Важным аспектом системы является обработка URL-адресов. Реализована автоматическая замена локальных URL на CDN URL, что обеспечивает прозрачность для разработчиков и конечных пользователей. Система поддерживает различные типы медиафайлов и обеспечивает обработку

адаптивных изображений, что критически важно для современных веб-приложений.

Интеграция CDN с MediaPlugin обеспечивает комплексную оптимизацию контента. Для изображений реализовано автоматическое изменение размеров и конвертация в формат WebP, что значительно улучшает производительность загрузки. Система обработки видео включает генерацию превью, оптимизацию формата и поддержку различных кодеков. Для аудио-контента обеспечена оптимизация формата и эффективное сжатие без потери качества.

2.2.12 Проектирование системы локализации

Система локализации должна обеспечивать поддержку многоязычности сайта. Основой системы должен стать механизм перевода на основе файлов локализации, который позволяет хранить переводы для различных языков. Система должна поддерживать различные форматы файлов локализации, включая JSON, YAML и PO-файлы.

Важным аспектом является реализация механизма определения языка, который позволяет автоматически выбирать язык на основе предпочтений пользователя или параметров URL-адреса. Система должна обеспечивать поддержку плюральных форм и форматирования дат, чисел и валют. Для обеспечения производительности необходимо реализовать кэширование переводов и оптимизацию загрузки файлов локализации.

Система должна поддерживать динамическое переключение языка и сохранение выбранного языка в настройках пользователя. Необходимо обеспечить возможность добавления новых языков без изменения кода фреймворка и поддержку RTL-языков. Для обеспечения качества переводов система должна поддерживать валидацию файлов локализации и проверку полноты переводов.

2.3 Программная реализация

В качестве инструментов для реализации программного продукта используются редактор кода Visual Studio Code и инструмент для управления

зависимостями Poetry, язык программирования Python и такие сторонние библиотеки как:

- pyyaml и toml, предоставляющие набор функций для работы с конфигурационными файлами;
- jinja2, управляющий системой шаблонов;
- markdown, предлагающий функции для обработки Markdown;
- python-frontmatter, предоставляющий возможности для работы с frontmatter в Markdown файлах;
- aiohttp, предоставляющий набор функций для административной панели и сервера разработки;
- pillow, использующийся для обработки изображений;
- pygments для подсветки синтаксиса;
- csscomprsessor и jsmin для минификации CSS и JavaScript;
- watchdog, предоставляющий функционал для отслеживания изменений файлов в реальном времени;
- click, обеспечивающий создание интерфейса командной строки;
- rich, предоставляющий набор инструментов для красивого форматированного вывода в консоль;
- aiofiles, обеспечивающий асинхронную работу с файловой системой;

- python-slugify, предоставляющий функции для создания URL-friendly строк из обычного текста;
- aiohttp-jinja2, обеспечивающий интеграцию шаблонизатора Jinja2 с веб-фреймворком aiohttp;
- beautifulsoup4, предоставляющий инструменты для парсинга и обработки HTML;
- geocoder, обеспечивающий функционал для работы с геолокацией;pycountry, предоставляющий базу данных о странах и их атрибутах;
- cryptography, обеспечивающий криптографические операции и безопасное хранение данных;
- opencv-python-headless, предоставляющий инструменты для обработки изображений без графического интерфейса.

Для разработки и тестирования фреймворка используются следующие зависимости:

- pytest, предоставляющий фреймворк для написания и запуска тестов;
- pytest-asyncio, обеспечивающий поддержку тестирования асинхронного кода;
- pytest-cov, предоставляющий инструменты для измерения покрытия кода тестами;
- black, обеспечивающий автоматическое форматирование кода по стандартам PEP 8;

- isort, предоставляющий функционал для автоматической сортировки импортов;mypy, обеспечивающий статическую проверку типов в коде.

Poetry значительно упрощает процесс разработки и поддержки проекта, обеспечивая надежное управление зависимостями и воспроизводимые сборки, поэтому перечень библиотек и их минимальные версии указаны в файле pyproject.toml.

2.3.1 Структура фреймворка

Фреймворк Staticflow будет реализован как Python-проект с модульной архитектурой. Основные компоненты системы будут организованы в следующие директории:

- core/ - ядро фреймворка, содержащее основные классы и интерфейсы,
- parsers/ - модули для обработки различных форматов контента,
- admin/ - компоненты административной панели,
- plugins/ - система плагинов и встроенные плагины,
- utils/ - вспомогательные утилиты и общие функции,
- deploy/ - модуль развертывания,
- cli/ - интерфейс командной строки,
- templates/ - система шаблонов.

2.3.2 Реализация ядра фреймворка

Начнем с реализации основных компонентов фреймворка. Первым шагом создадим ключевой класс Config для управления конфигурацией. Этот класс

обеспечивает загрузку и сохранение настроек из файлов формата TOML, управляет многоязычностью, а также валидирует конфигурацию. Класс конфигурации обеспечивает удобный интерфейс для доступа к настройкам через методы get и set, а также обеспечивает их сохранение в файл.

На рисунке 2.1 показаны функции для управления конфигурационным файлом.

```python
def load(self, config_path: Path) -> None:
    """Load configuration from file."""
    if not config_path.exists():
        raise FileNotFoundError(f"Config file not found: {config_path}")

    suffix = config_path.suffix.lower()
    with config_path.open("r", encoding="utf-8") as f:
        if suffix == ".toml":
            loaded_config = toml.load(f)
        else:
            raise ValueError(f"Unsupported config format: {suffix}")

        if loaded_config and isinstance(loaded_config, dict):
            self.config.update(loaded_config)
        else:
            raise ValueError(
                f"Invalid configuration format in {config_path}. "
                "Configuration must be a dictionary."
            )

def get(self, key: str, default: Any = None) -> Any:
    """Get configuration value."""
    return self.config.get(key, default)

def set(self, key: str, value: Any) -> None:
    """Set configuration value."""
    self.config[key] = value

def save(self, config_path: Optional[Path] = None) -> None:
    """Save configuration to file."""
    save_path = config_path or self.config_path
    if not save_path:
        raise RuntimeError("No config file path set")

    suffix = save_path.suffix.lower()
    with save_path.open("w", encoding="utf-8") as f:
        if suffix == ".toml":
            toml.dump(self.config, f)
        else:
            raise ValueError(f"Unsupported config format: {suffix}")
```

Рисунок 2.1. Функции управления конфигурационным файлом класса Config

За процесс генерации статического сайта отвечает класс Engine. Он координирует работу всех остальных компонентов системы и управляет процессом сборки. На рисунках 2.2, 2.3, 2.4 и 2.5 представлены функции для процесса сборки сайтов, обработку страниц, управления статическими файлами и управления плагинами соответственно.

```python
def build(self) -> None:
    """Build the site."""

    if self.site.output_dir:
        self.site.output_dir.mkdir(parents=True, exist_ok=True)

    for plugin in self.plugins:
        if hasattr(plugin, 'pre_build'):
            plugin.pre_build(self.site)

    self.site.clear()
    self.site.load_pages()
    self._process_pages()

    for plugin in self.plugins:
        if hasattr(plugin, 'post_build'):
            plugin.post_build(self.site)

    try:
        from ..admin import AdminPanel
        admin = AdminPanel(self.config, self)
        admin.copy_static_to_public()
    except Exception as e:
        print(f"Error copying admin static files: {e}")

    self._copy_static_files()
```

Рисунок 2.2 Функция сборки сайта

```python
def _process_page(self, page: Page) -> None:
    if page.source_path.suffix.lower() == '.md':
        content_html = self.markdown.convert(page.content)
    else:
        content_html = page.content

    for plugin in self.plugins:
        content_html = plugin.process_content(content_html)
```

Рисунок 2.3 Функция обработки страницы

```python
def _copy_static_files(self) -> None:
    """Copy static files to the output directory."""
    if not self.site.output_dir:
        return

    static_dir = self.config.get("static_dir", "static")
    if not isinstance(static_dir, Path):
        static_dir = Path(static_dir)

    if static_dir.exists():
        output_static = self.site.output_dir / "static"
        if output_static.exists():
            shutil.rmtree(output_static)
        shutil.copytree(static_dir, output_static)
```

Рисунок 2.4 Функция обработки статических файлов

```python
def add_plugin(self, plugin: Plugin,
               config: Optional[Dict[str, Any]] = None) -> None:
    """Add a plugin to the engine with optional configuration."""
    plugin.engine = self
    if config:
        plugin.config = config
    plugin.initialize()
    self.plugins.append(plugin)

def get_plugin(self, name: str) -> Optional[Plugin]:
    """Get a plugin by its name."""
    for plugin in self.plugins:
        if hasattr(plugin, 'metadata') and plugin.metadata.name == name:
            return plugin
    return None
```

Рисунок 2.5 Функции управления плагинами

Следующий фундаментальный компонент фреймворка - класс Page, который предоставляет отдельную страницу сайта и управляет ее содержимым и метаданными. Функции обработки страницы представлены на рисунке 2.6.

```python
@classmethod
def from_file(cls, path: Path, default_lang: str = "en") -> "Page":
    """Create a Page instance from a file."""
    if not path.exists():
        raise FileNotFoundError(f"Page source not found: {path}")

    content = ""
    metadata = {}

    raw_content = path.read_text(encoding="utf-8")

    if raw_content.startswith("---"):
        parts = raw_content.split("---", 2)
        if len(parts) >= 3:
            try:
                metadata = yaml.safe_load(parts[1])
                part = parts[2]
                if isinstance(part, Path):
                    part = str(part)
                content = part.strip()
            except yaml.YAMLError as e:
                raise ValueError(f"Invalid front matter in {path}: {e}")
    else:
        content = raw_content

    page = cls(path, content, metadata, default_lang)
    page.modified = path.stat().st_mtime
    return page

@property
def title(self) -> str:
    """Get the page title."""
    return self.metadata.get("title", self.source_path.stem)

@property
def url(self) -> str:
    """Get the page URL."""
    if self.output_path:
        return str(self.output_path.relative_to(
            self.output_path.parent.parent))
    return ""
```

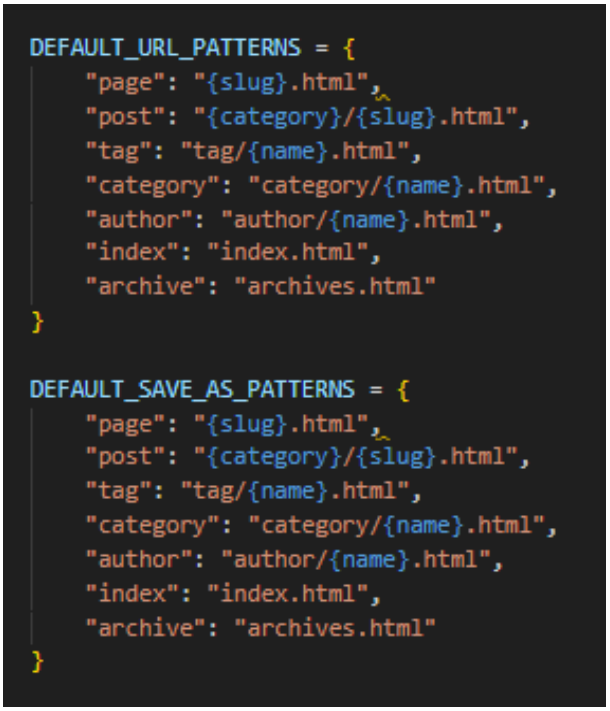
Рисунок 2.6 Функции управления страницей

Класс Router представляет собой компонент системы маршрутизации фреймворка StaticFlow, реализующий паттерн "URL Router". Он обеспечивает абстракцию между структурой контента и его URL-представлением через систему шаблонов, где каждый тип контента имеет предопределенный паттерн URL.

Система маршрутизации основана на принципе "Convention over Configuration", предоставляя стандартные шаблоны для различных типов контента (страницы, посты, теги и т.д.), которые могут быть переопределены через конфигурацию. Паттерны URL реализуют механизм подстановки переменных, позволяющий формировать структурированные URL-адреса на основе метаданных контента.

Класс Router поддерживает многоязычность через систему префиксов языков и обеспечивает разделение логического представления контента (URL_PATTERNS) от его физического хранения (SAVE_AS_PATTERNS). Это позволяет гибко настраивать структуру URL без изменения базовой логики маршрутизации, сохраняя при этом слабую связанность компонентов системы.

На рисунках 2.7, 2.8, 2.9 изображены следующие ключевые части кода класса Router: определение паттернов URL_PATTERNS и SAVE_AS_PATTERNS, основной метод генерации URL, метод форматирования шаблонов.



```
DEFAULT_URL_PATTERNS = {
    "page": "{slug}.html",
    "post": "{category}/{slug}.html",
    "tag": "tag/{name}.html",
    "category": "category/{name}.html",
    "author": "author/{name}.html",
    "index": "index.html",
    "archive": "archives.html"
}

DEFAULT_SAVE_AS_PATTERNS = {
    "page": "{slug}.html",
    "post": "{category}/{slug}.html",
    "tag": "tag/{name}.html",
    "category": "category/{name}.html",
    "author": "author/{name}.html",
    "index": "index.html",
    "archive": "archives.html"
}
```

Рисунок 2.7 Определение паттернов URL_PATTERNS и SAVE_AS_PATTERNS

```python
def get_url(self, content_type: str, metadata: Dict[str, Any]) -> str:
    """Получить URL для контента на основе типа и метаданных."""
    if 'url' in metadata:
        return metadata['url']

    pattern = self.url_patterns.get(content_type)
    if not pattern:
        if 'slug' in metadata:
            return f"{metadata['slug']}.html"
        return ""

    url = self._format_pattern(pattern, metadata)

    if self.use_clean_urls and url.endswith('.html'):
        url = url[:-5]

    if self.use_language_prefixes and 'language' in metadata:
        language = metadata['language']
        if language != self.default_language or not self.exclude_default_lang_prefix:
            if url == "index.html" or (self.use_clean_urls and url == "index"):
                if self.use_clean_urls:
                    url = f"{language}/"
                else:
                    url = f"{language}/index.html"
            else:
                url = f"{language}/{url}"

    return url
```

Рисунок 2.8 Основной метод генерации URL

```python
def _format_pattern(self, pattern: str, metadata: Dict[str, Any]) -> str:
    """Заменить все переменные в шаблоне значениями из метаданных."""
    result = pattern

    for match in re.finditer(r'\{([^}]+)\}', pattern):
        key = match.group(1)

        if key == 'category' and 'category' in metadata:
            category = metadata['category']
            if isinstance(category, list) and category:
                replacement = category[0]
            else:
                replacement = str(category)
        elif key in ('year', 'month', 'day') and 'date' in metadata:
            if key == 'year':
                replacement = self._format_date(metadata['date'], '%Y')
            elif key == 'month':
                replacement = self._format_date(metadata['date'], '%m')
            elif key == 'day':
                replacement = self._format_date(metadata['date'], '%d')
            else:
                replacement = ''
        else:
            replacement = str(metadata.get(key, ''))

        pattern_to_replace = '{' + key + '}'
        result = result.replace(pattern_to_replace, replacement)

    return self._normalize_path(result)
```

Рисунок 2.9 Метод форматирования шаблонов

Класс Server представляет собой ключевой компонент фреймворка StaticFlow, реализующий функциональность веб-сервера для разработки и обслуживания статических сайтов. В его основе лежит асинхронный веб-фреймворк aiohttp, который обеспечивает эффективную обработку HTTP-запросов.

Основная задача класса Server заключается в организации взаимодействия между различными компонентами системы: маршрутизацией, шаблонизацией, обработкой статических файлов и административной панелью. Особое внимание уделено режиму разработки, который включает автоматическую валидацию структуры проекта и пересборку сайта при изменениях.

На рисунках 2.10 и 2.11 представлены настройка маршрутизации и обработка запросов соответственно.

```python
def setup_routes(self):
    """Setup server routes."""
    # Admin routes
    self.app.router.add_get('/admin', self.admin_handler)
    self.app.router.add_get('/admin/{tail:.*}', self.admin_handler)
    self.app.router.add_post('/admin/api/{tail:.*}', self.admin_handler)
    self.app.router.add_post('/admin/{tail:.*}', self.admin_handler)

    # Static files
    static_url = self.config.get('static_url', '/static')
    static_dir = self.config.get('static_dir', 'static')
    if not isinstance(static_dir, Path):
        static_path = Path(static_dir)
    else:
        static_path = static_dir
    self.app.router.add_static(static_url, static_path)

    # All other routes
    self.app.router.add_get('/{tail:.*}', self.handle_request)
```

Рисунок 2.10 Метод настройки маршрутизации

```python
async def handle_request(self, request):
    path = request.path

    if path == '/':
        path = '/index.html'

    output_dir = self.config.get('output_dir', 'public')
    if not isinstance(output_dir, Path):
        output_path = Path(output_dir)
    else:
        output_path = output_dir

    file_path = output_path / path.lstrip('/')

    if not file_path.exists():
        potential_index = file_path / 'index.html'
        if potential_index.exists() and potential_index.is_file():
            file_path = potential_index
        else:
            if self.dev_mode:
                return web.Response(status=404, text="Not Found")
            else:
                raise web.HTTPNotFound()

    return web.FileResponse(
        file_path,
        headers={"Content-Type": content_type}
    )
```

Рисунок 2.11 Метод обработка запросов

2.4 Тестирование

2.5 Разработка документации

Вывод по главе 2

В этой главе были выявлены функциональные и нефункциональные требования к продукту. Это дало понять, как именно компоненты должны находится в фреймворке. С учетом всех выявленных требований к программному продукту, был успешно разработан фреймворк для генерации статических сайтов, о чём свидетельствуют рисунки…, демонстрирующие ожидаемый результат. Полный код продукта доступен в приложении А, а ссылка на документацию представлена в приложении В.

ЗАКЛЮЧЕНИЕ

В ходе проектирования и разработки фреймворка для генерации статических сайтов были выполнены следующие задачи:

1) Проведен анализ существующих решений для генерации статических сайтов, что позволило выявить ключевые недостатки и определить направления для улучшения.
2) Разработана гибкая архитектура фреймворка StaticFlow.
3) Реализован прототип фреймворка с базовыми функциями.
4) Разработана система плагинов, позволяющая расширять функциональность фреймворка.
5) Обеспечено полное покрытие тестами.
6) Разработана документация.

В результате проведения работы были достигнуты все поставленные задачи, и был успешно разработан фреймворк для генерации статических сайтов. Фреймворк Staticflow показал себя как эффективный способ разработки статических сайтов для разработчиков с разным опытом владения языка программирования Python.

БИБЛИОГРАФИЯ

1) Pelican Documentation [Электронный ресурс]. – Режим доступа: https://docs.getpelican.com/en/stable/ (дата обращения: 13.02.2024)
2) MkDocs Documentation [Электронный ресурс]. – Режим доступа: https://www.mkdocs.org/ (дата обращения: 13.02.2024)
3) Nikola Documentation [Электронный ресурс]. – Режим доступа: https://docs.getnikola.com/en/stable/nikola.html (дата обращения: 13.02.2024)
4) Sphinx Documentation [Электронный ресурс]. – Режим доступа: https://www.sphinx-doc.org/en/master/ (дата обращения: 13.02.2024)
5) Lektor Documentation [Электронный ресурс]. – Режим доступа: https://www.getlektor.com/docs/ (дата обращения: 13.02.2024)
6) SSG vs SSR vs ISR: что выбрать для вашего проекта? [Электронный ресурс] / LightNode. – Режим доступа: https://go.lightnode.com/ru/tech/ssg-vs-ssr-vs-is (дата обращения: 01.02.2024)
7) SEO-оптимизация сайта: полное руководство [Электронный ресурс] / Kokoc. – Режим доступа: https://kokoc.com/blog/seo-optimization/ (дата обращения: 24.03.2024)
8) Что такое sitemap.xml и чем этот файл помогает в продвижении [Электронный ресурс] / Timeweb. – Режим доступа: https://timeweb.com/ru/community/articles/chto-takoe-sitemap-xml-i-chem-etot-fayl-pomogaet-v-prodvizhenii (дата обращения: 22.04.2024)
9) Что такое минификация CSS [Электронный ресурс] / Timeweb. – Режим доступа: https://timeweb.com/ru/community/articles/chto-takoe-minifikaciya-css (дата обращения: 14.03.2024)
10) Оптимизация RSS-ленты [Электронный ресурс] / Devaka. – Режим доступа: https://devaka.info/articles/rss-feed-optimization (дата обращения: 15.04.2024)

ПРИЛОЖЕНИЕ А

Репозиторий готового продукта

Ссылка: https://github.com/nestessia/StaticFlow-diploma

ПРИЛОЖЕНИЕ Б

Документация фреймворка staticflow

ПРИЛОЖЕНИЕ В

Ссылка на PyPi