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

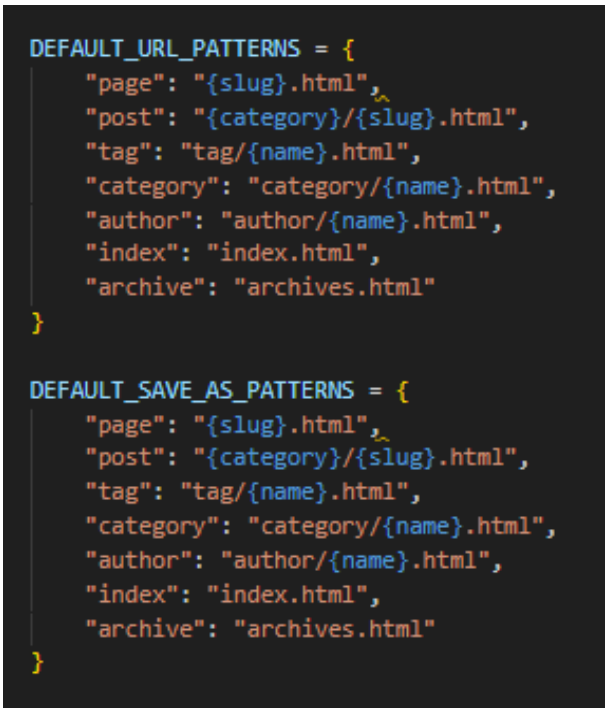
Класс Router представляет собой компонент системы маршрутизации фреймворка StaticFlow, реализующий паттерн "URL Router". Он обеспечивает абстракцию между структурой контента и его URL-представлением через систему шаблонов, где каждый тип контента имеет предопределенный паттерн URL.

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