The Complete Guide to Apiros3's Website Template

Apiros3

2025-09-08

Abstract

This comprehensive guide covers the complete architecture, build pipeline, and maintenance procedures for my academic portfolio website. I detail every component, from the TeX-to-HTML conversion system to the metafile configuration system, providing both high-level overviews and implementation specifics. This document serves as both documentation for the current system and a reference for anyone wishing to understand or modify the website's functionality.

1 Introduction

This website represents a sophisticated academic portfolio system built around a flexible, metadata-driven architecture. The system combines TeX document processing, Python-based site generation, and modern web technologies to create a maintainable, professional academic presence.

The core philosophy is *separation of concerns*: content (TeX files), configuration (JSON metafiles), and presentation (HTML/CSS) are kept distinct, allowing for easy maintenance and updates without touching the underlying codebase.

2 System Architecture Overview

2.1 High-Level Architecture

The website follows a three-tier architecture:

1. **Content Layer**: TeX source files, JSON metadata files, and static assets
2. **Processing Layer**: Python scripts, build tools, and conversion pipelines
3. **Presentation Layer**: Generated HTML, CSS, and JavaScript

2.2 Core Components

The system consists of several key components:

- **TeX Processing Pipeline**: Converts academic papers and blog posts from TeX to HTML
- **Metafile Configuration System**: Manages site-wide settings and content metadata
- **Page Generation Engine**: Creates HTML pages from templates and data
- **Publication Management**: Handles academic publications and talks
- **Notes Organization**: Manages academic notes and paper summaries
- **Reading List System**: Tracks books, papers, and resources

3 Directory Structure and File Organization

3.1 Root Directory Structure

```
Apiros3.github.io/
|-- index.html                # Main homepage (GitHub Pages entry)
|-- site.meta.json           # Site configuration metafile
|-- posts/                    # Blog posts and TeX sources
|   |-- index.html           # Blog listing page
|   |-- [yyyy]-[mm]-[dd]-[title].tex  # TeX source files
|   ‘-- [title]/             # Generated blog post directories
|-- publications/             # Publications and talks
|   |-- index.html
|   |-- data/                # Publication metadata
|   ‘-- scripts/             # Generation scripts
|-- Notes/                   # Academic notes
|-- templates/               # HTML templates
|-- script/                  # Python generation system
|-- css/                     # Stylesheets
|-- images/                  # Static assets
|-- build_html.sh           # TeX conversion script
|-- Makefile                # Build system
‘-- README.md               # Documentation
```

3.2 Key Configuration Files

Definition 3.1 (Metafile System)**.** A metafile is a JSON configuration file that contains structured data about content, settings, or metadata. The system uses multiple metafiles to manage different aspects of the website.

The system uses several metafiles:

- `site.meta.json`: Main site configuration
- `notes.meta.json`: Academic notes metadata
- `reading-list.meta.json`: Reading list data
- `publications/data/*.meta.json`: Publication metadata
- `posts/*.meta.json`: Blog post metadata

4 The TeX Processing Pipeline

4.1 TeX to HTML Conversion

The TeX processing pipeline converts academic documents to web-friendly HTML:

Listing 1: TeX conversion process

```
# TeX file naming convention
posts/YYYY-MM-DD-title.tex

# Conversion command (simplified)
pandoc input.tex -o output.html --mathjax --standalone
```

4.2 Build Script Architecture

The `build_html.sh` script handles the conversion process:

1. Scans the `posts/` directory for TeX files
2. Extracts metadata from TeX headers
3. Converts each file using Pandoc with appropriate settings
4. Generates corresponding metadata files
5. Creates individual blog post directories
6. Updates the blog listing page

4.3 Pandoc Configuration

The system uses Pandoc with specific settings optimized for academic content:

- Math rendering via MathJax
- Syntax highlighting for code blocks
- Cross-reference support
- Table of contents generation
- Custom CSS integration

5 The Metafile Configuration System

5.1 Site Configuration (`site.meta.json`)

The main site configuration controls the homepage and global settings:

```
{
  "site": {
    "title": "Apiros3",
    "description": "Academic Portfolio",
    "author": "Apiros3"
  },
  "about": {
    "title": "About Me",
    "content": "Multi-paragraph content...",
    "profile_picture": "images/profile.jpg"
  },
  "contact": {
    "email": "email@institution.edu",
    "institution": "University Name",
    "location": "City, Country"
  },
  "navigation": {
    "brand": "Apiros3",
    "items": [...]
  }
}
```

5.2 Multiple Paragraph Support

The system supports multiple paragraphs in the about section using double line breaks:

```
{
  "about": {
    "content": "First paragraph.\n\nSecond paragraph.\n\nThird paragraph."
  }
}
```

This is processed by the `format_about_content()` function in `script/page_generators.py`.

6 The Python Generation System

6.1 Core Modules

The Python system consists of several modules:

- `config.py`: Configuration loading and management
- `page_generators.py`: HTML page generation logic
- `template_engine.py`: Template rendering functions
- `data_loader.py`: Data loading and processing
- `generate_site_new.py`: Main orchestration script

6.2 Page Generation Process

The page generation follows this workflow:

1. Load configuration from metafiles
2. Process TeX files and extract metadata
3. Load publication and talk data
4. Generate HTML pages using templates
5. Apply styling and JavaScript
6. Output final HTML files

6.3 Template System

The template system uses Python string formatting with custom functions:

Listing 2: Template function example

```python
def generate_html_head(title: str, base_path: str = "") -> str:
    """Generate HTML head section with CSS and metadata."""
    return f"""<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8"/>
  <meta name="viewport" content="width=device-width,initial-scale
      =1"/>
  <title>{title}</title>
  <link rel="stylesheet" href="{base_path}css/main.css">
</head>"""
```

7 Publication Management System

7.1 Publication Data Structure

Publications are managed through JSON metadata files:

```json
{
  "title": "Paper Title",
  "authors": ["Author One", "Author Two"],
  "conference": "Conference Name",
  "year": "2025",
  "abstract": "Abstract text...",
  "arxiv": "https://arxiv.org/abs/...",
  "doi": "https://doi.org/...",
  "code": "https://github.com/...",
  "venue": "CONF 2025",
  "pages": "1-10"
}
```

7.2 Talk Management

Talks follow a similar structure with additional fields:

```json
{
  "title": "Talk Title",
  "type": "workshop / seminar / conference / invited",
  "venue": "Venue Name",
  "location": "City, Country",
  "date": "2025-09-08",
  "year": "2025",
  "slides": "https://slides.com/...",
  "video": "https://youtube.com/...",
  "abstract": "Talk abstract...",
  "coauthors": ["Author One", "Author Two"]
}
```

8 Notes and Reading List Systems

8.1 Notes Organization

Academic notes are organized by subject with metadata:

```
{
  "title": "Subject Name",
  "description": "Brief description...",
  "pdf_file": "notes.pdf",
  "pdf_files": ["lec1.pdf", "lec2.pdf"],
  "tags": ["mathematics", "analysis"]
}
```

8.2 Reading List Management

The reading list tracks academic resources:

```
{
  "title": "Book/Paper Title",
  "author": "Author Name",
  "year": "2025",
  "type": "book / paper / article",
  "status": "reading / completed / planned",
  "description": "Brief description..."
}
```

9 Build System and Automation

9.1 Makefile Targets

The Makefile provides several build targets:

Listing 3: Makefile targets

```
all: clean blog main pub # Full build
blog: # Blog posts only
main: # Main pages only
pub: # Publications only
clean: # Clean generated files
help: # Show help
```

9.2 Cross-Platform Support

The system supports multiple platforms:

- **Linux/macOS**: Native Make support
- **Windows**: Batch files and PowerShell scripts
- **WSL**: Full Linux compatibility on Windows

10 CSS and Styling System

10.1 CSS Architecture

The styling system uses a modular CSS approach:

- `base.css`: Core variables, typography, and base styles
- `layout.css`: Layout components and responsive design
- `components.css`: Reusable UI components
- `pages.css`: Page-specific styles
- `markdown.css`: Content styling for blog posts
- `main.css`: Main stylesheet that imports all others

10.2 Typography System

The system uses a professional typography stack:

Listing 4: Typography configuration

```
:root {
  --font-family: 'Inter', 'Source Sans Pro', 'Segoe UI', system-ui,
      sans-serif;
  --font-family-mono: 'JetBrains Mono', 'Fira Code', monospace;
  --font-size-base: 16px;
  --line-height-base: 1.7;
}
```

10.3 Responsive Design

The system implements responsive design principles:

- Mobile-first approach
- Flexible grid system
- Adaptive typography
- Touch-friendly navigation

11 Content Management Workflow

11.1 Adding New Blog Posts

To add a new blog post:

1. Create TeX file: `posts/YYYY-MM-DD-title.tex`
2. Write content using LaTeX syntax
3. Optionally create metadata file: `posts/title.meta.json`
4. Run build system: `make all`

11.2 Updating Site Configuration

To update site-wide settings:

1. Edit `site.meta.json`
2. Modify desired configuration sections
3. Run site generation: `python script/generate_site_new.py`
4. Verify changes in generated HTML

11.3 Managing Publications

To add a new publication:

1. Create metadata file: `publications/data/paper.meta.json`
2. Add PDF file if available
3. Run build system to update publications page

12 Deployment and Hosting

12.1 GitHub Pages Integration

The system is designed for GitHub Pages deployment:

1. Push repository to GitHub
2. Enable GitHub Pages in repository settings
3. Set source to "Deploy from a branch"
4. Select main branch and root folder
5. The `index.html` serves as the entry point

12.2 Local Development

For local development:

Listing 5: Local development commands

```
# View main page
start index.html

# View blog
start posts/index.html

# View publications
start publications/index.html
```

13 Advanced Features

13.1 Tag Filtering System

The blog system includes JavaScript-based tag filtering:

- Automatic tag extraction from TeX metadata
- Interactive filter buttons
- Real-time content filtering
- URL-based filter state management

13.2 Math Rendering

Mathematical content is rendered using MathJax:

- Inline math: `$...$`
- Display math: `$$...$$`
- LaTeX environments support
- Cross-reference compatibility

13.3 Code Highlighting

Code blocks are highlighted using Pandoc's syntax highlighting:

- Language detection
- Theme customization
- Copy-to-clipboard functionality
- Mobile-friendly display

14 Maintenance and Troubleshooting

14.1 Common Issues

1. **Pandoc not found**: Install Pandoc and ensure it's in PATH
2. **Python errors**: Check Python installation and dependencies
3. **TeX conversion errors**: Verify LaTeX syntax and packages
4. **Metafile errors**: Validate JSON syntax
5. **Missing files**: Run full build system

14.2 Debugging Workflow

When issues arise:

1. Check build output for error messages
2. Verify all dependencies are installed
3. Ensure you're running from project root
4. Check file permissions and paths
5. Validate JSON syntax in metafiles

15 Performance Considerations

15.1 Optimization Strategies

The system implements several optimization strategies:

- Minified CSS and JavaScript
- Optimized images and assets
- Efficient HTML generation
- Lazy loading for large content
- CDN integration for external resources

15.2 Loading Performance

Key performance metrics:

- Fast initial page load
- Efficient navigation
- Quick content filtering
- Responsive image loading

16 Future Enhancements

16.1 Planned Features

Potential future improvements:

- Dark mode support
- Search functionality
- RSS feed generation
- Comment system integration
- Advanced analytics

16.2 Extensibility

The system is designed for easy extension:

- Modular architecture
- Plugin system potential
- Template customization
- Theme system

17 Conclusion

This academic portfolio website represents a sophisticated, maintainable system for presenting academic work online. The combination of TeX processing, Python generation, and modern web technologies creates a flexible platform that can adapt to changing needs while maintaining professional presentation standards.

The key to the system's success is its *separation of concerns*: content, configuration, and presentation are kept distinct, allowing for easy maintenance and updates. The metafile system provides a user-friendly interface for content management, while the Python generation system ensures consistency and reliability.

For anyone wishing to understand, modify, or extend this system, this document provides the comprehensive technical foundation necessary for effective development and maintenance.