💻 Code Samples Showcase

This document demonstrates the syntax highlighting capabilities across various programming languages.

JavaScript

```javascript
// Modern JavaScript with ES6+ features
class MarkdownViewer {
  constructor(config) {
    this.config = { ...this.defaultConfig, ...config };
    this.files = new Map();
    this.activeTab = null;
  }

  async loadFile(fileName) {
    try {
      const response = await fetch(`items/${fileName}`);
      if (!response.ok) throw new Error(`HTTP ${response.status}`);

      const content = await response.text();
      this.displayContent(fileName, content);
    } catch (error) {
      console.error('Failed to load file:', error);
    }
  }

  displayContent(fileName, content) {
    const html = marked.parse(content);
    document.getElementById('content').innerHTML = html;

    // Highlight code blocks
    Prism.highlightAll();
  }
}

// Usage
const viewer = new MarkdownViewer({
  theme: 'dark',
  syntax: 'github'
});
```

Python

```python
# Python class for file processing
import asyncio
import aiohttp
from pathlib import Path
from typing import List, Dict, Optional

class MarkdownProcessor:
    """Advanced markdown processing with async support."""

    def __init__(self, base_path: Path):
        self.base_path = base_path
        self.files: Dict[str, str] = {}
        self.session: Optional[aiohttp.ClientSession] = None

    async def __aenter__(self):
        self.session = aiohttp.ClientSession()
        return self

    async def __aexit__(self, exc_type, exc_val, exc_tb):
        if self.session:
            await self.session.close()

    async def load_files(self, file_list: List[str]) -> Dict[str, str]:
        """Load multiple files concurrently."""
        tasks = [self.load_file(filename) for filename in file_list]
        results = await asyncio.gather(*tasks, return_exceptions=True)

        return {
            filename: result
            for filename, result in zip(file_list, results)
            if not isinstance(result, Exception)
        }

    async def load_file(self, filename: str) -> str:
        """Load a single file with error handling."""
        file_path = self.base_path / filename

        if file_path.exists():
            return file_path.read_text(encoding='utf-8')
        else:
            raise FileNotFoundError(f"File not found: {filename}")

# Usage example
async def main():
    async with MarkdownProcessor(Path('./items')) as processor:
        files = await processor.load_files(['README.md', 'guide.md'])
        for name, content in files.items():
            print(f"Loaded {name}: {len(content)} characters")

if __name__ == "__main__":
    asyncio.run(main())
```

TypeScript

```typescript
// TypeScript interfaces and generics
interface ViewerConfig {
  theme: 'light' | 'dark';
  mode: 'replace' | 'tabs';
  syntax: boolean;
  math: boolean;
}

interface FileItem {
  name: string;
  path: string;
  type: 'markdown' | 'html';
  size?: number;
  lastModified?: Date;
}

type ViewMode = 'replace' | 'tabs';
type Theme = 'light' | 'dark';

class TypedMarkdownViewer<T extends ViewerConfig = ViewerConfig> {
  private config: T;
  private files: Map<string, FileItem> = new Map();
  private activeTab: string | null = null;

  constructor(config: T) {
    this.config = config;
  }

  public async loadFiles<K extends keyof FileItem>(
    fileNames: string[],
    fields?: K[]
  ): Promise<Pick<FileItem, K>[]> {
    const loadPromises = fileNames.map(async (name): Promise<FileItem> => {
      const response = await fetch(`items/${name}`);
      const content = await response.text();

      return {
        name,
        path: `items/${name}`,
        type: name.endsWith('.md') ? 'markdown' : 'html',
        size: content.length,
        lastModified: new Date()
      };
    });

    const results = await Promise.all(loadPromises);
    return fields ? results.map(item => this.pickFields(item, fields)) : results;
  }

  private pickFields<T, K extends keyof T>(obj: T, fields: K[]): Pick<T, K> {
    const picked = {} as Pick<T, K>;
    fields.forEach(field => {
      picked[field] = obj[field];
    });
    return picked;
  }
}

// Generic usage
const viewer = new TypedMarkdownViewer({
  theme: 'dark',
  mode: 'tabs',
  syntax: true,
  math: true
});
```

Java

```java
// Java with Spring Boot annotations
package com.example.markdown;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.*;
import org.springframework.http.ResponseEntity;
import java.util.*;
import java.util.concurrent.CompletableFuture;
import java.util.stream.Collectors;

@SpringBootApplication
@RestController
@RequestMapping("/api/markdown")
public class MarkdownViewerApplication {

    private final MarkdownService markdownService;

    public MarkdownViewerApplication(MarkdownService markdownService) {
        this.markdownService = markdownService;
    }

    @GetMapping("/files")
    public ResponseEntity<List<FileInfo>> getFiles() {
        List<FileInfo> files = markdownService.discoverFiles()
            .stream()
            .sorted(Comparator.comparing(FileInfo::getName))
            .collect(Collectors.toList());

        return ResponseEntity.ok(files);
    }

    @GetMapping("/content/{filename}")
    public CompletableFuture<ResponseEntity<String>> getContent(
            @PathVariable String filename) {

        return markdownService.loadFileAsync(filename)
            .thenApply(content -> ResponseEntity.ok(content))
            .exceptionally(ex -> {
                return ResponseEntity.notFound().build();
            });
    }

    @PostMapping("/render")
    public ResponseEntity<RenderedContent> renderMarkdown(
            @RequestBody RenderRequest request) {

        try {
            String html = markdownService.renderToHtml(
                request.getContent(),
                request.getOptions()
            );

            return ResponseEntity.ok(new RenderedContent(html));
        } catch (Exception e) {
            return ResponseEntity.badRequest().build();
        }
    }

    public static void main(String[] args) {
        SpringApplication.run(MarkdownViewerApplication.class, args);
    }
}

// Data classes
record FileInfo(String name, String path, FileType type, long size) {}
record RenderRequest(String content, RenderOptions options) {}
record RenderedContent(String html) {}

enum FileType {
    MARKDOWN("md"), HTML("html");

    private final String extension;

    FileType(String extension) {
        this.extension = extension;
    }

    public String getExtension() {
        return extension;
    }
}
```

C

```
// C# with async/await and LINQ
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;

namespace MarkdownViewer.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class MarkdownController : ControllerBase
    {
        private readonly ILogger<MarkdownController> _logger;
        private readonly IMarkdownService _markdownService;

        public MarkdownController(
            ILogger<MarkdownController> logger,
            IMarkdownService markdownService)
        {
            _logger = logger;
            _markdownService = markdownService;
        }

        [HttpGet("files")]
        public async Task<ActionResult<IEnumerable<FileInfoDto>>> GetFilesAsync()
        {
            try
            {
                var files = await _markdownService.DiscoverFilesAsync();
                var dtos = files.Select(f => new FileInfoDto
                {
                    Name = f.Name,
                    Path = f.FullName,
                    Type = Path.GetExtension(f.Name).ToLower() switch
                    {
                        ".md" => FileType.Markdown,
                        ".html" => FileType.Html,
                        _ => FileType.Unknown
                    },
                    Size = f.Length,
                    LastModified = f.LastWriteTime
                }).OrderBy(f => f.Name);

                return Ok(dtos);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error discovering files");
                return StatusCode(500, "Internal server error");
            }
        }

        [HttpGet("content/{filename}")]
        public async Task<ActionResult<string>> GetContentAsync(string filename)
        {
            if (string.IsNullOrWhiteSpace(filename))
                return BadRequest("Filename is required");

            try
            {
                var content = await _markdownService.LoadFileAsync(filename);
                return Ok(content);
            }
            catch (FileNotFoundException)
            {
                return NotFound($"File '{filename}' not found");
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error loading file {Filename}", filename);
                return StatusCode(500, "Internal server error");
            }
        }
    }

    public record FileInfoDto
    {
        public string Name { get; init; } = string.Empty;
        public string Path { get; init; } = string.Empty;
        public FileType Type { get; init; }
        public long Size { get; init; }
        public DateTime LastModified { get; init; }
    }

    public enum FileType
    {
        Unknown,
        Markdown,
        Html
    }
}
```

SQL

```sql
-- Advanced SQL with CTEs and window functions
WITH file_stats AS (
  SELECT
    filename,
    file_type,
    file_size,
    created_date,
    ROW_NUMBER() OVER (PARTITION BY file_type ORDER BY created_date DESC) as rn,
    AVG(file_size) OVER (PARTITION BY file_type) as avg_size_by_type
  FROM markdown_files
  WHERE status = 'active'
),
recent_files AS (
  SELECT
    filename,
    file_type,
    file_size,
    created_date,
    CASE
      WHEN file_size > avg_size_by_type THEN 'large'
      WHEN file_size < avg_size_by_type * 0.5 THEN 'small'
      ELSE 'medium'
    END as size_category
  FROM file_stats
  WHERE rn <= 10
)
SELECT
  f.filename,
  f.file_type,
  f.size_category,
  v.view_count,
  v.last_viewed,
  COALESCE(r.rating, 0) as avg_rating
FROM recent_files f
LEFT JOIN file_views v ON f.filename = v.filename
LEFT JOIN (
  SELECT
    filename,
    ROUND(AVG(rating), 2) as rating
  FROM file_ratings
  GROUP BY filename
  HAVING COUNT(*) >= 3
) r ON f.filename = r.filename
ORDER BY v.view_count DESC NULLS LAST, f.created_date DESC;

-- Stored procedure example
DELIMITER //
CREATE PROCEDURE GetMarkdownFilesByCategory(
  IN category_filter VARCHAR(50),
  IN limit_count INT DEFAULT 20
)
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE file_cursor CURSOR FOR
    SELECT filename, file_size, created_date
    FROM markdown_files
    WHERE file_type = category_filter
    ORDER BY created_date DESC
    LIMIT limit_count;

  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
  CREATE TEMPORARY TABLE temp_results (
    filename VARCHAR(255),
    file_size BIGINT,
    created_date DATETIME,
    size_rank INT
  );

  OPEN file_cursor;

  read_loop: LOOP
    FETCH file_cursor INTO @filename, @file_size, @created_date;
    IF done THEN
      LEAVE read_loop;
    END IF;

    INSERT INTO temp_results (filename, file_size, created_date)
    VALUES (@filename, @file_size, @created_date);
  END LOOP;

  CLOSE file_cursor;

  -- Add ranking
  UPDATE temp_results tr
  SET size_rank = (
    SELECT COUNT(*) + 1
    FROM temp_results tr2
    WHERE tr2.file_size > tr.file_size
  );

  SELECT * FROM temp_results ORDER BY size_rank;

  DROP TEMPORARY TABLE temp_results;
END //
DELIMITER ;
```

CSS

```css
/* Modern CSS with custom properties and grid */
:root {
  --primary-color: #007bff;
  --secondary-color: #6c757d;
  --success-color: #28a745;
  --danger-color: #dc3545;
  --warning-color: #ffc107;
  --info-color: #17a2b8;

  --font-family-sans-serif: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto;
  --font-family-monospace: "SF Mono", Monaco, "Cascadia Code", "Roboto Mono";

  --border-radius: 0.375rem;
  --box-shadow: 0 0.125rem 0.25rem rgba(0, 0, 0, 0.075);
  --transition: all 0.15s ease-in-out;
}

/* CSS Grid layout */
.markdown-viewer {
  display: grid;
  grid-template-areas:
    "sidebar header"
    "sidebar main";
  grid-template-columns: 300px 1fr;
  grid-template-rows: auto 1fr;
  height: 100vh;
  overflow: hidden;
}

.sidebar {
  grid-area: sidebar;
  background: var(--bs-light);
  border-right: 1px solid var(--bs-border-color);
  overflow-y: auto;
}

.header {
  grid-area: header;
  background: white;
  border-bottom: 1px solid var(--bs-border-color);
  padding: 1rem;
}

.main-content {
  grid-area: main;
  overflow: auto;
  padding: 2rem;
}

/* Flexbox utilities */
.d-flex {
  display: flex !important;
}

.flex-column {
  flex-direction: column !important;
}

.justify-content-between {
  justify-content: space-between !important;
}
```

```css
.align-items-center {
  align-items: center !important;
}

/* Custom components */
.file-item {
  display: flex;
  align-items: center;
  padding: 0.75rem 1rem;
  border-radius: var(--border-radius);
  transition: var(--transition);
  cursor: pointer;
  border: 1px solid transparent;
}

.file-item:hover {
  background-color: var(--bs-light);
  border-color: var(--bs-border-color);
  transform: translateX(4px);
}

.file-item.active {
  background-color: var(--primary-color);
  color: white;
  box-shadow: var(--box-shadow);
}

/* Responsive design */
@media (max-width: 768px) {
  .markdown-viewer {
    grid-template-areas:
      "header"
      "sidebar"
      "main";
    grid-template-columns: 1fr;
    grid-template-rows: auto auto 1fr;
  }

  .sidebar {
    max-height: 40vh;
  }
}

/* Dark theme */
@media (prefers-color-scheme: dark) {
  :root {
    --bs-body-bg: #212529;
    --bs-body-color: #dee2e6;
    --bs-light: #343a40;
    --bs-border-color: #495057;
  }
}

.dark-theme {
  --bs-body-bg: #0d1117;
  --bs-body-color: #c9d1d9;
  --bs-light: #161b22;
  --bs-border-color: #30363d;
  --primary-color: #58a6ff;
}

/* Animations */
@keyframes slideIn {
  from {
    opacity: 0;
    transform: translateX(-20px);
  }
  to {
    opacity: 1;
    transform: translateX(0);
  }
}

.file-item {
  animation: slideIn 0.3s ease-out;
}

/* Print styles */
@media print {
  .sidebar,
  .header {
    display: none !important;
  }

  .main-content {
    grid-area: auto;
    padding: 0;
    max-width: none;
  }
}
```

Bash

```bash
#!/bin/bash
# Advanced bash script for markdown processing

set -euo pipefail  # Exit on error, undefined vars, pipe failures

# Configuration
readonly SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
readonly ITEMS_DIR="${SCRIPT_DIR}/items"
readonly OUTPUT_DIR="${SCRIPT_DIR}/dist"
readonly MANIFEST_FILE="${ITEMS_DIR}/manifest.json"

# Colors for output
readonly RED='\033[0;31m'
readonly GREEN='\033[0;32m'
readonly YELLOW='\033[1;33m'
readonly BLUE='\033[0;34m'
readonly NC='\033[0m' # No Color

# Logging functions
log_info() {
    echo -e "${BLUE}[INFO]${NC} $*" >&2
}

log_warn() {
    echo -e "${YELLOW}[WARN]${NC} $*" >&2
}

log_error() {
    echo -e "${RED}[ERROR]${NC} $*" >&2
}

log_success() {
    echo -e "${GREEN}[SUCCESS]${NC} $*" >&2
}

# Error handling
handle_error() {
    local exit_code=$?
    log_error "Script failed on line $1 with exit code $exit_code"
    cleanup
    exit $exit_code
}

trap 'handle_error $LINENO' ERR

# Cleanup function
cleanup() {
    log_info "Performing cleanup..."
    # Remove temporary files if they exist
    [[ -f "$temp_manifest" ]] && rm -f "$temp_manifest"
}

# Function to check dependencies
check_dependencies() {
    local deps=("jq" "find" "sort")

    for cmd in "${deps[@]}"; do
        if ! command -v "$cmd" &> /dev/null; then
            log_error "Required command '$cmd' not found"
            exit 1
        fi
    done
    log_success "All dependencies satisfied"
}

# Function to discover markdown files
discover_files() {
    local files=()

    log_info "Discovering markdown and HTML files in $ITEMS_DIR"

    # Find files and sort them
    while IFS= read -r -d '' file; do
        # Get relative path from items directory
        local rel_path="${file#$ITEMS_DIR/}"
        files+=("$rel_path")
    done < <(find "$ITEMS_DIR" -type f \( -name "*.md" -o -name "*.html" -o -name "*.ht
m" \) -print0 | sort -z)

    printf '%s\n' "${files[@]}"
}

# Function to generate manifest
generate_manifest() {
    local temp_manifest
    temp_manifest=$(mktemp)

    log_info "Generating manifest file"

    # Create JSON array of files
    {
        echo '{'
        echo '  "files": ['

        local files=()
        mapfile -t files < <(discover_files)

        for i in "${!files[@]}"; do
            local file="${files[i]}"
            local comma=""
            [[ $i -lt $((${#files[@]} - 1)) ]] && comma=","

            echo "    \"$file\"$comma"
        done

        echo '  ],'
        echo "  \"generated\": \"$(date -Iseconds)\","
        echo "  \"count\": ${#files[@]}"
        echo '}'
    } > "$temp_manifest"

    # Validate JSON
    if jq empty "$temp_manifest" 2>/dev/null; then
        mv "$temp_manifest" "$MANIFEST_FILE"
        log_success "Manifest generated with ${#files[@]} files"
    else
        log_error "Generated invalid JSON"
        cat "$temp_manifest"
        rm -f "$temp_manifest"
        exit 1
    fi
}

# Function to validate files
```

```
validate_files() {
    log_info "Validating files in manifest"

    if [[ ! -f "$MANIFEST_FILE" ]]; then
        log_error "Manifest file not found: $MANIFEST_FILE"
        return 1
    fi

    local files
    files=$(jq -r '.files[]' "$MANIFEST_FILE")
    local error_count=0

    while IFS= read -r file; do
        local full_path="$ITEMS_DIR/$file"
        if [[ ! -f "$full_path" ]]; then
            log_error "File not found: $file"
            ((error_count++))
        else
            log_info "✓ $file"
        fi
    done <<< "$files"

    if [[ $error_count -gt 0 ]]; then
        log_error "Found $error_count missing files"
        return 1
    fi

    log_success "All files validated successfully"
}

# Function to show statistics
show_stats() {
    if [[ ! -f "$MANIFEST_FILE" ]]; then
        log_warn "No manifest file found"
        return
    fi

    local total_files
    total_files=$(jq -r '.count' "$MANIFEST_FILE")

    local md_count
    md_count=$(jq -r '.files[] | select(endswith(".md"))' "$MANIFEST_FILE" | wc -l)

    local html_count
    html_count=$(jq -r '.files[] | select(endswith(".html") or endswith(".htm"))' "$MAN
IFEST_FILE" | wc -l)

    echo
    log_info "📊 File Statistics:"
    echo "  Total files: $total_files"
    echo "  Markdown files: $md_count"
    echo "  HTML files: $html_count"
    echo
}

# Main function
main() {
    local command="${1:-help}"

    case "$command" in
        "generate"|"gen")
            check_dependencies
            mkdir -p "$ITEMS_DIR"
            generate_manifest
            show_stats
            ;;
        "validate"|"val")
            check_dependencies
            validate_files
            ;;
        "stats"|"stat")
            show_stats
            ;;
        "clean")
            log_info "Cleaning generated files"
            [[ -f "$MANIFEST_FILE" ]] && rm -f "$MANIFEST_FILE"
            log_success "Cleanup complete"
            ;;
        "help"|"-h"|"--help")
            cat << EOF
Markdown Viewer Build Script

Usage: $0 [COMMAND]

Commands:
  generate, gen    Generate manifest.json from discovered files
  validate, val    Validate all files in manifest exist
  stats, stat      Show file statistics
  clean           Remove generated files
  help            Show this help message

Examples:
  $0 generate      # Create manifest from files in items/
  $0 validate      # Check all manifest files exist
  $0 stats         # Show file counts and statistics

EOF
            ;;
        *)
            log_error "Unknown command: $command"
            echo "Use '$0 help' for usage information"
            exit 1
            ;;
    esac
}

# Run main function with all arguments
main "$@"
```

JSON Configuration

```json
{
  "name": "advanced-markdown-viewer",
  "version": "2.0.0",
  "description": "A comprehensive markdown viewer for GitHub Pages",
  "config": {
    "viewer": {
      "defaultMode": "replace",
      "themes": ["light", "dark", "auto"],
      "features": {
        "syntaxHighlighting": {
          "enabled": true,
          "languages": [
            "javascript", "typescript", "python", "java",
            "csharp", "sql", "bash", "css", "html", "json",
            "yaml", "xml", "php", "ruby", "go", "rust"
          ],
          "theme": "prism"
        },
        "mathSupport": {
          "enabled": true,
          "engine": "mathjax",
          "delimiters": {
            "inline": ["$", "$"],
            "display": ["$$", "$$"]
          }
        },
        "codeBlocks": {
          "copyButton": true,
          "lineNumbers": false,
          "wrapLines": false,
          "fontControls": true
        }
      }
    },
    "fileDiscovery": {
      "manifestFile": "items/manifest.json",
      "itemsFolder": "items/",
      "supportedExtensions": [".md", ".html", ".htm"],
      "sorting": "alphabetical",
      "recursive": true
    },
    "ui": {
      "sidebar": {
        "width": 300,
        "collapsible": true,
        "showFileIcons": true
      },
      "tabs": {
        "maxOpen": 10,
        "showCloseButton": true,
        "scrollable": true
      },
      "responsive": {
        "breakpoints": {
          "mobile": 768,
          "tablet": 1024
        }
      }
    },
    "performance": {
      "lazyLoading": true,
      "caching": {
        "enabled": true,
        "ttl": 3600
      },
      "compression": "gzip"
    }
  },
  "build": {
    "output": "dist/",
    "minify": true,
    "sourceMaps": false,
    "optimization": {
      "bundleSize": "minimal",
      "treeShaking": true
    }
  },
  "deployment": {
    "platform": "github-pages",
    "customDomain": null,
    "https": true,
    "caching": {
      "static": "1y",
      "html": "1h"
    }
  }
}
```

Interactive code blocks with full syntax highlighting! 🎨

Try the font size, family, and line spacing controls on each code block above.