# Markdown Rendering Feature - Complete Guide

## Overview

This document explains how your project converts Markdown to HTML with sanitization, HTTP caching, and content negotiation. You'll understand the rendering pipeline, security measures, and why each design decision was made.

# 1. What is Markdown Rendering?

Markdown is a lightweight markup language that uses plain text formatting. It's easy to write and read, and can be converted to HTML for display in browsers.

## Example Markdown:

```
# My Note Title
This is **bold** and *italic*.
- Item 1
- Item 2
```

## Rendered as HTML:

```
<h1>My Note Title</h1>
<p>This is <strong>bold</strong> and <em>italic</em>.</p>
<ul><li>Item 1</li><li>Item 2</li></ul>
```

# 2. The MarkdownService Class

MarkdownService handles all markdown-to-HTML conversion with three main responsibilities: render to HTML, generate cache keys (ETags), and combine rendering with caching.

## Allowed HTML Tags

The ALLOWED_TAGS list defines which HTML tags are permitted in the output. This is crucial for security - any tag not in this list is removed.

### Categories of Allowed Tags:

• Text Formatting: p, br, strong, em, u, del, ins

• Headers: h1, h2, h3, h4, h5, h6

• Lists: ul, ol, li

• Code: code, pre

• Links and Images: a, img

• Tables: table, thead, tbody, tr, th, td

• Other: blockquote, hr, div, span

## Allowed Attributes

Even allowed tags can have dangerous attributes. The ALLOWED_ATTRIBUTES dictionary specifies which attributes are safe for each tag.

```
ALLOWED_ATTRIBUTES = {
    'a': ['href', 'title', 'target', 'rel'],
    'img': ['src', 'alt', 'title', 'width', 'height'],
    'code': ['class'],
    'pre': ['class'],
    'div': ['class'],
    'span': ['class']
}
```

# 3. The Three Main Functions

## Function 1: render_to_html()

Converts Markdown text to sanitized HTML using the markdown library and bleach sanitizer.

```
    def render_to_html(markdown_text: str) -> str:
        # Step 1: Convert Markdown to HTML
        html = markdown.markdown(
            markdown_text,
            extensions=['extra', 'codehilite', 'toc'],
            output_format='html5'
        )

        # Step 2: Sanitize HTML
        sanitized_html = bleach.clean(
            html,
            tags=ALLOWED_TAGS,
            attributes=ALLOWED_ATTRIBUTES,
            strip=True
        )

        return sanitized_html
```

### Step 1: markdown.markdown()

The markdown library converts Markdown text to HTML. It applies extensions like tables, code highlighting, and table of contents.

### Step 2: bleach.clean()

Bleach is a security library that sanitizes HTML. It removes anything not explicitly allowed. This prevents XSS (Cross-Site Scripting) attacks.

# 4. HTML Sanitization (Security)

## Why Sanitize?

Users could enter malicious Markdown that generates dangerous HTML. Without sanitization, attackers could steal data, hijack accounts, or deface the application.

## Example Attack (XSS):

```
User writes:
<script>alert('Hacked!')</script>

Without sanitization:
Browser executes the script!

With sanitization:
<script> tag is completely removed
```

## What Gets Removed:

• script tags - JavaScript execution

• iframe tags - Embedded malicious sites

• onclick, onload - Event handler attributes

• javascript: links - JavaScript in href

• object, embed tags - Plugin content

# 5. HTTP Caching with ETag

## What is an ETag?

ETag (Entity Tag) is an HTTP header used for caching. It's a unique identifier (hash) for specific content. If content hasn't changed, the ETag stays the same.

## Function 2: generate_etag()

```
def generate_etag(content: str) -> str:
    # Create MD5 hash of content
    content_hash = hashlib.md5(
        content.encode('utf-8')
    ).hexdigest()

    # Wrap in quotes (HTTP standard)
    return f'"{content_hash}"'
```

## How HTTP Caching Works:

### First Request:

```
Client → GET /notes/123/render
Server → Renders HTML, generates ETag
Server → Response:
    Status: 200 OK
    ETag: "a3f5c8b2e9d1"
    Content: <html>...
```

### Second Request:

```
Client → GET /notes/123/render
    If-None-Match: "a3f5c8b2e9d1"

Server → Renders, compares ETags
Server → If match:
    Status: 304 Not Modified
    ETag: "a3f5c8b2e9d1"
    (No content body)

Client → Uses cached content
```

## Benefits:

• Bandwidth Savings - No full HTML if unchanged

• Faster Load - 304 response is smaller

• Better UX - Pages load instantly from cache

• Mobile Friendly - Saves data on mobile networks

# 6. Content Negotiation

## What is Content Negotiation?

Content negotiation allows clients to request different formats of the same resource. The client sends an Accept header, and the server responds accordingly.

## How It Works:

```
    accept_header = request.headers.get("Accept", "text/html")

    if "application/json" in accept_header:
        return JSONResponse({
            "note_id": str(note_id),
            "title": note.title,
            "html": html_content,
            "markdown": markdown_content
        })
    else:
        return HTMLResponse(full_html_document)
```

## Accept Header Examples:

• text/html → Full HTML document (for browsers)

• application/json → JSON with html field (for APIs)

• */* → HTML by default

# 7. The Two Render Endpoints

## Endpoint 1: /notes/{note_id}/render

Renders note as a complete, styled HTML page OR as JSON. Supports caching and content negotiation.

### Key Features:

✓ Verifies user ownership

✓ Renders Markdown to HTML

✓ Generates ETag for caching

✓ Checks If-None-Match header

✓ Content negotiation (HTML or JSON)

✓ Wraps HTML in styled document

## Endpoint 2: /notes/{note_id}/render/raw

Returns only the rendered HTML without the document wrapper. Useful for embedding in other pages.

### Differences:

• No HTML wrapper - Just rendered content

• No CSS styling - Pure content HTML

• Same security - Still sanitized

• Same caching - ETag support included

• HTML only - No JSON option

# 8. Complete Rendering Flow

1. User requests: GET /notes/123/render
2. FastAPI authenticates with JWT token
3. Query database for note
4. Verify ownership (note.owner_id == user.id)
5. Combine title and content as Markdown
6. Call MarkdownService.render_with_etag()
7. markdown.markdown() converts to HTML
8. bleach.clean() sanitizes HTML
9. hashlib.md5() generates ETag
10. Check If-None-Match header
11. If ETag matches → 304 Not Modified
12. Check Accept header
13. Return JSON or HTML with ETag header

# 9. Supervisor Questions & Answers

## Q1: How does Markdown rendering work?

Answer: I use the Python markdown library to convert Markdown to HTML with extensions for tables and code highlighting. After conversion, I use bleach to sanitize the HTML by removing dangerous tags and attributes. This ensures safe display of user content.

## Q2: Why sanitize HTML output?

Answer: Security. Without sanitization, users could inject malicious JavaScript that executes in other users' browsers (XSS attacks). Bleach removes all tags not in my whitelist and strips dangerous attributes, making output safe.

## Q3: What is an ETag?

Answer: ETag is an HTTP caching mechanism. I generate an MD5 hash of the rendered HTML and return it as an ETag header. If content hasn't changed (same hash), I return 304 Not Modified instead of full content, saving bandwidth.

## Q4: What is content negotiation?

Answer: Content negotiation allows the same endpoint to return different formats. If a browser requests it (Accept: text/html), I return styled HTML. If an app requests it (Accept: application/json), I return JSON. One endpoint, multiple formats.

## Q5: Why have two endpoints (/render and /render/raw)?

Answer: /render returns a complete HTML document with CSS - perfect for browsers. /render/raw returns just the rendered HTML - useful for embedding or when clients want their own styling. Different use cases need different outputs.

## Q6: How do you prevent XSS attacks?

Answer: Three layers: SQLAlchemy prevents SQL injection. Bleach sanitizes HTML output by removing script tags and event handlers. Content-Security-Policy headers add another layer. Even if malicious code reaches the database, it can't execute.

# 10. Key Takeaways

■ Markdown to HTML: Python markdown library with extensions

■ Security First: Bleach sanitizes all HTML output

■ HTTP Caching: ETag/If-None-Match for bandwidth savings

■ Content Negotiation: HTML or JSON based on Accept header

■ Two Endpoints: Full page vs raw HTML

■ XSS Prevention: Whitelist approach to tags/attributes

■ Professional Output: Styled HTML with modern CSS

■ Performance: MD5 hashing and browser caching