Email Template Editor

A Streamlit-based web application for dynamic HTML email template editing with real-time preview capabilities. This application uses BeautifulSoup4 for HTML parsing and provides a streamlined interface for content management without direct HTML manipulation.

Technical Overview

Core Components

1. **HTML Parser (BeautifulSoup4)**
   - Parses HTML template into navigable data structure
   - Identifies and extracts editable text elements
   - Maintains HTML structure integrity during modifications
   - Handles nested HTML elements and preserves attributes
2. **State Management (Streamlit)**
   - Session state variables:
     - `html_content`: Original template content
     - `modified_html`: Current state of modified template
     - `pending_changes`: Dictionary of uncommitted changes
     - `wellness_url`: Current wellness button URL
     - `signup_url`: Current signup button URL
3. **Text Processing System**
   - Character limit: 128 characters
   - Deduplication algorithm for repeated text
   - Smart text element detection
   - Hierarchical element processing

Technical Requirements

Required Python Packages

```
streamlit>=1.10.0
beautifulsoup4>=4.9.3
base64
re
```

System Requirements

- Python 3.7 or higher
- Modern web browser with JavaScript enabled
- Minimum 512MB RAM
- Write permissions in app directory

Implementation Details

Text Element Detection Algorithm

```
def find_text_elements(soup):
    """
    Algorithm for identifying editable text elements:
    1. Traverses HTML structure using BeautifulSoup
    2. Filters elements based on tag types
    3. Implements deduplication
    4. Handles nested elements
    """
    elements = []
    seen_texts = set()

    for element in soup.find_all(['h1', 'h2', 'h3', 'h4', 'p', 'div',
'span']):
        text_content = element.text.strip()

        # Filtering conditions
        if not text_content or text_content in seen_texts:
            continue

        # Button and special element detection
        if element.find_parent('a', class_='v-button'):
            continue

        seen_texts.add(text_content)
        elements.append(element)

    return elements
```

State Management System

The application implements a two-stage modification system: 1. **Pending Changes Stage** - Changes are stored in `st.session_state.pending_changes` - Format: `{original_text: new_text}` - No modifications to actual HTML

2. **Commit Stage**
   - Triggered by “Save Changes” button
   - Applies all pending changes to HTML
   - Updates preview
   - Writes to file system

HTML Processing Workflow

1. **Template Loading**

```
def load_template():
    with open('template.html', 'r', encoding='utf-8') as file:
        return file.read()
```

2. **Content Modification**
   - Text replacements use direct string replacement
   - URL updates handled separately for special cases
   - Social media link detection using regex patterns
3. **Export Generation**

```
def get_download_link(html_content, filename):
    b64 = base64.b64encode(html_content.encode()).decode()
    return f'<a href="data:text/html;base64,{b64}"
download="{filename}">
            Download Modified Template</a>'
```

Deployment Guide

Local Development

1. Clone repository
2. Install dependencies:

```
pip install -r requirements.txt
```

3. Place template.html in root directory
4. Run application:

```
streamlit run app.py
```

Streamlit Cloud Deployment

1. Push code to GitHub repository
2. Include template.html in repository
3. Connect Streamlit Cloud to repository
4. Deploy application

File Structure Requirements

```
/your-app-directory
├── app.py               # Main application file
├── template.html        # HTML email template
├── requirements.txt     # Dependencies
└── README.md           # Documentation
```

Technical Features

Text Processing

- Character limit enforcement (128 chars)
- Intelligent duplicate detection
- HTML structure preservation
- Nested element handling

URL Management

- Button URL tracking
- Social media link pattern matching
- URL validation and update system
- Link integrity preservation

Preview System

- Real-time HTML rendering
- Dynamic content updates
- Responsive layout handling
- Cross-browser compatibility

Error Handling

1. **File Operations**
   - Template file missing
   - File permission issues
   - Encoding errors
2. **HTML Processing**
   - Malformed HTML handling
   - Invalid element structure
   - Duplicate content management
3. **State Management**
   - Session state corruption
   - Update conflicts
   - Save operation failures

Performance Considerations

- HTML parsing optimization
- State management efficiency
- Memory usage management
- DOM manipulation limitations

Technical Limitations

1. **HTML Parsing**
   - Complex nested structures may not be fully editable
   - Dynamic content limitations
   - JavaScript handling restrictions
2. **Text Processing**
   - 128 character limit on editable text
   - No rich text formatting
   - Limited special character support
3. **State Management**

- Session-based limitations
- Temporary storage constraints
- Multi-user access limitations

Debugging and Development

Common Issues Faced during development of this tool

1. Template loading failures
2. HTML parsing errors
3. State management issues
4. URL update problems