Tool for automating template changes of static web pages and sending notifications via email

Miloš Milošević[1][0009-0009-7182-3518] and Marija Punt[1][0000-0002-1944-7086]

[1] Faculty of Electrical Engineering, University of Belgrade, Belgrade, Serbia

Abstract. This paper aims to describe the tool created to alleviate and speed up the process of updating static web pages of the courses held by the Department of Computer Science and Information Technology of the School of Electrical Engineering, University of Belgrade. The paper gives an overview of the current solutions and their shortcomings. The end result of this paper is a tool that automates a large portion of the process of updating static web pages and reduces the amount of user input.

Keywords: static web pages, automation, templates, HTML, static site generators, Python

1 Introduction – Why focus on static web pages

The web pages of the courses held by Department of Computer Science and Information Technology of the School of Electrical Engineering, University of Belgrade, are implemented as static web pages, a type of implementation that comes with its own advantages and disadvantages. These web pages display content to the users, without the ability to change the content displayed on the page through their actions. Unlike dynamic web pages, static web pages do not rely on scripting languages to generate content during use; instead, they consist solely of HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) code, which makes them simpler to create.

An additional advantage of this approach to web development is the ease of page loading. When displaying static web pages, the web browser does not need to execute potentially large amounts of poorly written or unoptimized script code in order to generate the content to be displayed. As a result, access time is shorter and availability is broader. This makes this method of web page development a good candidate for educational web pages, such as course pages with announcements and teaching materials.

However, the aforementioned advantages of static web pages do not come without a cost, which in this case is reflected in the difficulty of modifying or updating the pages. Any change requires editing the source HTML code of the page located on a remote server. Simplifying and speeding up the process of updating source code of the web pages would eliminate this disadvantage while keeping all the previously mentioned advantages of using static web pages for providing educational content.

This paper is split into several chapters. Second chapter will provide overview of current solutions, their advantages and disadvantages. Third chapter will focus on the proposal of the new solution. Chapter number four fill feature overview of the technologies used to create the tool. Fifth chapter will describe tool's system architecture. Finaly, the sixth chapter will provide conclusion that contins tool's review, its advantages and disadvantages comapred to other solutions, as well as potential ways to further upgrade presented tool.

2 Overview of current maintenance techniques

This chapter will provide overview of current solutions commonly used for maintaining and updating static web pages. Each solution will have a brief description as well as most notable advantages and disadvantages of said solution.

2.1 Manual source code updates

The simplest, but also most tedious, way to update static web pages is through manual editing of the HTML files [1]. A HTML file, containing all of the web pages content, is a text file at its core, highly specialized form of text designed to be rendered by web browsers into visual representation but still, just a text file. As such, it can be opened and its content modified using any of the numerous text editors.

This method of updating static web pages allows for any change of structure, appearance, or content of the web pages, but at the cost of a large amount of manual input, the need for knowledge of HTML and the structure of the web page in order to avoid making potential mistakes during the update. Any mistake that breaks the HTML structure could cause web page to not be able to load and display content, requiring maintainers to exercise extra caution.

The obstacle present in this solution is the fact that HTML files containing the source code are often located on remote servers, making access more difficult. The files first need to be downloaded from the server using one of the network file transfer protocols. Once downloaded to a local device, the HTML files can be easily edited, and after the changes are made, they must be uploaded back to the server using the same network protocols.

2.2 Static site generators

Static site generators [2] are software tools designed to create static web pages. The focus of this chapter will not be on the details of any specific static site generator, but on the general process of generating static web page shared by all static site generators, with some occasional generator specific detail used to expand the context. Some of the more notable static site generators include Next.js, Hugo and Gatsby among many others.

Using markup languages, commonly Markdown, to describe page content in separate files and HTML files to provide layout that content should be presented in, static site generators combine the two in order to fill the defined template with desired content. As this process involves parsing of markup files and generating final HTML file, creation of static web page is not instantaneous and requires certain build time. Build times can vary depending on the number of factors including number of content files and template complexity, with simpler pages requiring seconds to build, while more complex projects could take significantly longer to build. Notably, Hugo requires less time to build static web pages due to being written in more performant Go language, compared to more commonly used JavaScript.

Advantage of using static site generators compared to manual updates is reduction in required user input, which is now used only for wring content, not encapsulating HTML code, making it significantly easier to approach by non-developers. However, this also comes with a drawback of having to rebuild the page every time a change needs to be made and waiting out the previously mentioned build time before the page is ready. Additionally, this means that static site generators cannot be used to update existing static web pages, but only those that were created using static site generators.

3 Proposal of a new solution

The solutions presented in the previous chapters have their shortcomings that make the process of maintaining and updating static web pages more difficult than it needs to be. The focus of this paper is to provide users with a solution that combines the advantages of both previously described approaches, while also introducing new features and eliminating most of the drawbacks. As a platform-independent tool, the new solution offers users:

- the ability to automatically generate new HTML code based on user requirements, simplifying the process of adding new content,
- a preview and the ability to edit key information extracted from the page [3],
- the option to send notifications via email, either independently or when a page is updated,
- fast and efficient operation.
- intuitive GUI (Graphical User Interface) made to improve user experience compared to command line tools

Additionally, this solution maintains compatibility with the existing method of updating pages. A drawback of the proposed solution is its current limitation to working exclusively with a single web page template used for course pages.

Where this solution significantly improves user experience for updating static web pages is with its templates system. As there are lots of instances of adding content pieces that differs very slightly compared to previously written ones, templates offer a way to significantly reduce amount of user input required to add new content to the web page, by requiring user to add only the information that varies between similar content pieces. The information that template requires can also be configured to take one of several shapes like preconfigured options selectable from a dropdown, text that can be input freely or files that will be uploaded to the server and linked to in the content piece.

4 Used technologies

This chapter will go over the technologies and libraries used during the development of the application, with brief details about them and their role within the application.

4.1 Python

The tool was implemented using the Python [4] programming language, specifically version 3.12. This language allows for easy handling of large amounts of data as well as string literals, which makes it an ideal candidate for developing this tool, as most of the work involves parsing existing HTML code and generating new HTML code.

As one of the most popular programming languages, Python offers a large number of community-developed libraries that implement various functionalities. The libraries used for the development of this tool include: colorama, paramiko, BeautifulSoup, darkdetect, and PySide6.

colorama. The colorama library [5] was used to change the text color in the terminal output. This library defines a set of symbols that translate into ANSI (American National Standards Institute) character sequences. Using these symbols makes terminal interaction easier and reduces the chances of errors when outputting sequences.

paramiko. The paramiko library [6] is an implementation of the SSHv2 (Secure Shell version 2) protocol for communication between devices. SFTP is a component of the SSHv2 protocol that enables secure file transfer between connected devices — in this case, for transferring files containing HTML source code or other important files between the server and the user's device.

BeautifulSoup. The BeautifulSoup library [7] is used for parsing HTML code. With it, a parse tree can be generated from the code, allowing easier searching, removal of existing parts of the code, or insertion of new ones. Once the code is regenerated from the parse tree, all changes to the tree will be reflected in the updated HTML code. This significantly simplifies the process of modifying HTML code, which is now represented as a data structure (a tree) rather than plain text.

darkdetect. The darkdetect library [8] provides functionality for detecting the system theme. Since different systems store theme settings in different ways and locations, retrieving this information is not as trivial as it may initially seem. darkdetect offers a platform-independent solution that ensures code portability, while handling platform-specific complexities internally.

This information is important when the application's style is inherited from the operating system. In such cases, most UI elements have a default appearance unless the developer chooses to manually define their look. Knowing whether the system is in "light" or "dark" mode allows for consistent styling of elements like background colors, text, and icons, aligning them with the rest of the application.

PySide6. The PySide6 library [9] allows the integration of Python applications with the Qt development framework used for creating graphical user interfaces. The basic building blocks of Qt graphical applications are components (widgets), which are combined to form the desired interface layout. Users can create their own components or use predefined ones such as labels, buttons, text input fields, and many others.

What sets this framework apart is the signals and slots system [10], which enables communication between components. Components define signals that are triggered by certain events (e.g., button clicks, text input), which may be of interest to other components. These signals can be connected to slots — functions that respond to the signals by executing when a signal is triggered. Since slots can be methods of other components, these methods will be executed upon signal activation, enabling communication between components without violating encapsulation. Signals can also be defined to pass parameters to connected slots. This system enables the graphical interface to easily detect user interactions and/or system changes and respond accordingly.

4.2 JSON

JSON (JavaScript Object Notation) [11][12] is a text-based format primarily designed for data exchange over the internet. It is named after the JavaScript programming language because it is based on the object notation used in JavaScript, which involves storing key-value pairs.

The most common use of JSON, for which it was originally designed, is communication between client and server components of a system. However, other uses for JSON were quickly recognized, such as data storage (e.g., configuration files or non-relational databases). Today, all major programming languages support working with JSON format/files.

4.3 HTML

HTML is a markup language specifically designed for displaying web pages in a browser. Web browsers receive HTML documents from storage servers and render them for the user. An HTML document contains a description of a webpage's layout, which browsers interpret and render as visual elements on the screen.

5 System architecture

The system consists of two components: a storage server (hereinafter referred to as the server) and a client application (hereinafter referred to as the application). The server is used for storing web pages. Each page has its own directory containing all relevant files, such as HTML, CSS, and other style files necessary for rendering the page, as well as documents that can be downloaded from the page.

The application is a tool designed to facilitate easier manipulation of these page-related files. The application communicates with the server via the SSHv2 protocol. Upon establishing an SSHv2 session, the application gains access rights to the server's content via SFTP (Secure File Transfer Protocol) [13]. This functionality is handled by the application's communication layer. Two additional layers can be found within the application. Application layer is designed to handle logic and processing functionality, while the presentation layer is responsible for GUI and user interaction. Inside the application data is exchanged between neighboring layers so that each layer can successfully perform its tasks (see Fig. 1).

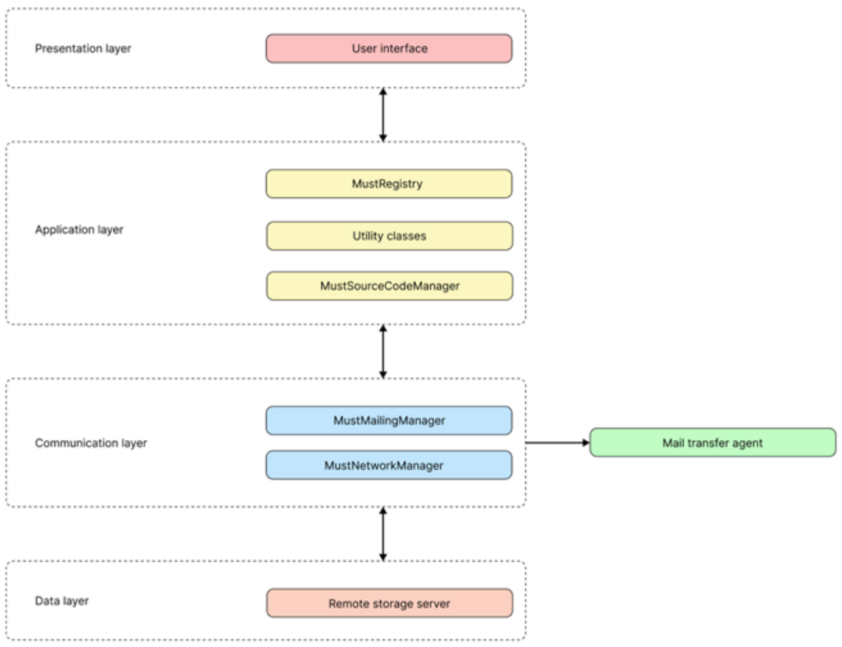


Fig. 1. Schematic representation of the system architecture with its main components.

Within the directory assigned to a web page, one can find the index.html file, which defines the structural layout of the homepage, and a number of subdirectories used for categorizing and organizing files. Among them, the most notable are the info and style directories. The style directory contains all the style files that determine the final appearance of the page's structural components. The obavestenja.html file, located in the info directory, contains announcements displayed on the website. This content is embedded into the homepage during the loading process, before the page is rendered. Other potentially present directories serve to store various documents. Additionally, the page directory may contain metadata files, including templates that users can use to create announcements or send emails.

The process the application executes in order to update a web page is following:

1. Connect to the host server
2. Fetch web page metadata and source files
3. Create page parsing tree by parsing source files
4. Make desired changes
5. Update parsing tree and regenerate source code from it
6. Push source files with updated code back to the server and optionally send accompanying email notification
7. Disconnect from the server

Only user input required during the process occurs in steps 1 and 4, with input in step 1 being credentials to access the server, while all other steps are automated. While making changes users have options of adding new, modifying or deleting existing content trough intuitive and user friendly GUI (see Fig. 2 and 3).

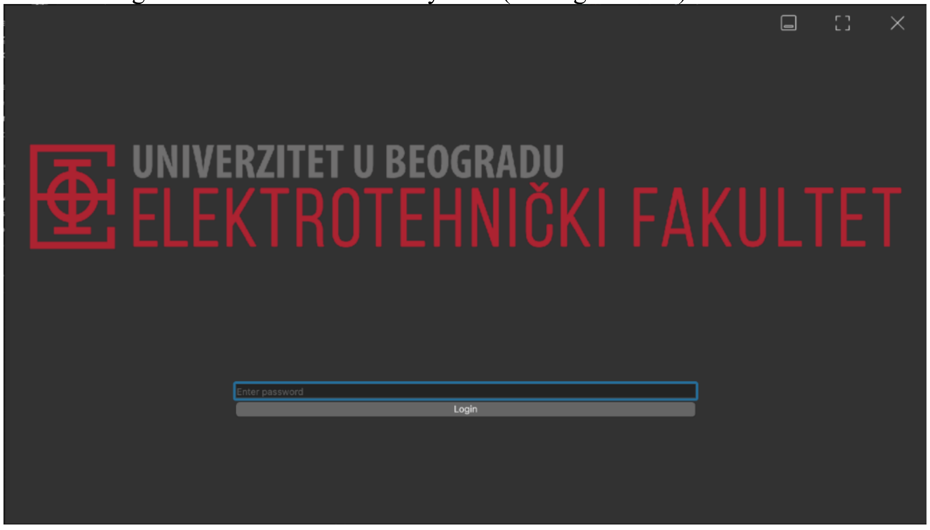
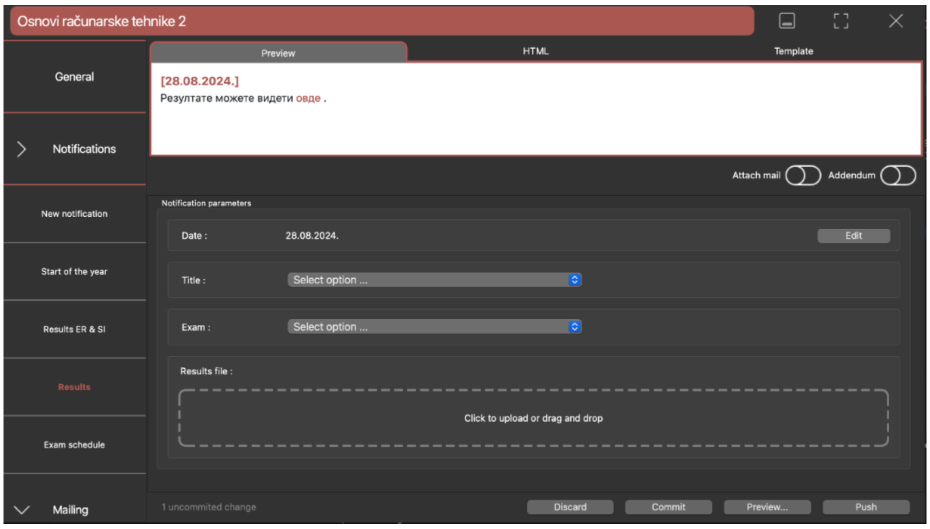


Fig. 2. Login screen used for authentication and connection to the server.



[28.08.2024.]
Резултате можете видети овде .

Date : 28.08.2024.

Fig. 3. Adding new notification to the web page using one of the existing templates.

6 Conclusion

During the daily work of maintaining course web pages at the Department of Computer Science and Information Technology of the School of Electrical Engineering, University of Belgrade, opportunities were identified to improve the workflow by automating parts of the web page update process. These improvements aim to enhance the user experience by reducing the amount of manual input required from users and speeding up the process by automating simple, repetitive actions.

This paper presents an application that implements the identified automation opportunities and facilitates the maintenance of static web pages. Users are no longer required to understand the structure of HTML code or know the exact locations of course pages in order to update them. The application extracts key information from the HTML code and presents it to the user in a much more organized manner, simplifying the process of editing content. When adding new content to a web page, the user no longer has to write the accompanying HTML code, as it is now automatically generated based on templates available within the application. Additionally, the application allows users to send emails either manually or automatically.

A current limitation of this solution is that it can only update pages whose HTML structure conforms to a specific template, which is defined within the application and used for parsing page data.

The application presented in this paper holds potential for upgrades and enhancements. The first improvement would be to expand the set of pages that can be modified using this application by adding new templates for parsing information. Introducing a server-side component would be another direction for improvement. The server-side would handle all functionalities related to file and HTML management, thereby offloading the client side of the application, which would then focus solely on editing the received information. The server-side could also enforce mutual exclusion between users to prevent situations where multiple users edit a page simultaneously, which could result in some changes being lost. These proposed enhancements would further simplify the web page updating process and improve the overall user experience when using the application.

References

1. Chu, J.Y.M., Palya, W.L. & Walter, D.E.: Creating a hypertext markup language document for an information server. Behavior Research Methods, Instruments, & Computers 27, 200–205 (1995). https://doi.org/10.3758/BF03204732
2. Dhillon, V.: Static Site Generators, In: Creating Blogs with Jekyll, Apress, Berkeley, CA, (2016), 21 - 33, https://doi.org/10.1007/978-1-4842-1464-0_3
3. A. Arasu & H. Garcia-Molina: Extracting structured data from Web pages, In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data (SIGMOD '03), Association for Computing Machinery, New York, NY, USA, (2003), 337–348, doi:10.1145/872757.872799
4. Python homepage, https://www.python.org, last accessed 2025/05/28.
5. colorama documentation, https://pypi.org/project/colorama/, last accessed 2025/05/28.
6. paramiko documentation, https://docs.paramiko.org/en/latest/, last accessed 2025/05/28.
7. BeautifulSoup documentation, https://www.crummy.com/software/BeautifulSoup/bs4/doc/ last accessed 2025/05/28.
8. darkdetect documentation, https://pypi.org/project/darkdetect/, last accessed 2025/05/28.
9. PySide6 documentation, https://doc.qt.io/qtforpython-6/index.html, last accessed 2025/05/28.
10. PySide6 signals and slots system in depth overview, https://doc.qt.io/qtforpython-6/tutorials/basictutorial/signals_and_slots.html, last accessed 2025/05/28.
11. JSON homepage, https://www.json.org/json-en.html, last accessed 2025/05/28.
12. Nurseitov, N., Paulson, M., Reynolds, R. & Izurieta, C.: Comparison of JSON and XML data interchange formats: a case study. Caine, 9, 157-162 (2009).
13. S. P. Bryan, P. G. Franklin, & K. J. Qualls.: Secure file transfer and secure file transfer protocol. U.S. Patent No. 8,261,059. (2012).