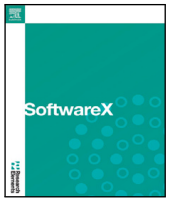
SoftwareX 27 (2024) 101809

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original software publication

EmintWeb: Creation of embedded web applications in C++ for specific systems

Juan Domingo *, Jose Ignacio Panach, Esther Dura

Department of Informatics, University of Valencia, Avda. de la Universidad, s/n, 46100 Burjasot, Valencia, Spain

ARTICLE INFO

Keywords:
Embedded web applications
Web development
High load web applications
Web application security

ABSTRACT

Most contemporary web applications are primarily coded in interpreted languages (JavaScript, PHP, Python...) and are initiated by the web server. This requires solving the persistence issue: HTTP/HTTPS is a stateless protocol but user identity and computational state across consecutive requests must be preserved, typically using cookies and/or backend database servers.

This work develops embedded web applications: single compiled executable programs that encapsulate a web server. These applications are coded in a compiled language (in our case, C++). They initiate a separate thread for each session to establish optionally encrypted communication with the client and the HTML5 code for each page is dynamically generated at runtime.

This approach offers several advantages: it enhances security on both the server and client sides. On the server side there is only one file on disk (the executable) that can be altered. On the client side cookies are not needed and client-side code execution can be eliminated. Also, the use of compiled code enhances speed and faster application performance compared to interpreted languages.

This methodology is realized through a framework named EmintWeb (Embedded Interactive Web Development) which comprises a C++ code generator to create the HTML5 pages at runtime and link them with the business logic code of the application. Subsequently, it generates the executable that serves the application. An example of the same application developed using EmintWeb and PHP is provided to evaluate the speed and robustness of both implementations. This approach is not a replacement for current web frameworks but a software system to build web applications using C++ for systems that require the specific characteristics mentioned before.

Code metadata

Current code version	v1.0
Permanent link to repository for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00180
Permanent link to Reproducible Capsule	
Legal Code License	GNU General Public License v. 3
Code versioning system used	git
Software code languages, tools, and services used	C++
Compilation requirements, operating environments & dependencies	Any Linux distribution, g++ compiler. Depends on: gumbo parser https://github.com/google/gumbo-parser/ Qt5 libraries https://www.qt.io/ Generated code uses: Poco C++ libraries https://pocoproject.org/
If available Link to developer documentation/manual	
Support email for questions	Juan.Domingo@uv.es

* Corresponding author.
E-mail addresses: Juan.Domingo@uv.es (Juan Domingo), joigpana@uv.es (Jose Ignacio Panach), Esther.Dura@uv.es (Esther Dura).

Available online 4 July 2024
https://doi.org/10.1016/j.softx.2024.101809

Received 20 March 2024; Received in revised form 12 June 2024; Accepted 21 June 2024

1. Motivation and significance

The arising of the Internet made it possible to execute interactive programs in two different locations: the server handles the business logic, while the client takes charge of the presentation/GUI [1]. The use of a web browser, due to its ubiquity, has become widely accepted as the default solution to run web systems. States are presented as web pages written in HTML, and the business logic is executed partly by the server in the form of interpreted code in PHP or a similar language, and partly by the client, mostly as JavaScript snippets of code. However, this approach brings about some obvious problems: slowness (interpreted vs. compiled code) and security concerns. The client is required to execute code (JavaScript), which could be malicious, and is obligated to accept and provide information to account for the computation state (cookies, habitually used for other, spurious purposes) [2]. Additionally, flaws in the server code can be exploited through the introduction of treacherous data, as seen in SQL injection attacks [3].

Additionally, from the developer's perspective, a significant challenge arises due to the stateless nature of the HTTP/HTTPS protocols that underpin web navigation [4]. Consequently, adequate measures must be implemented to maintain continuity in computation between successive states. Regarding efficiency, multithreaded servers are employed to handle simultaneous requests, aiding in balancing the computational load [5]. In terms of security, a server attack with privilege escalation can modify web pages (HTML or PHP code), potentially affecting all users and sessions [6].

The EmintWeb software aims to address these concerns by developing embedded web applications. An embedded web application is an executable program, entirely written in a high-level language (in our case, C++), encompassing a simple web server and the business logic. It dynamically generates HTML5 code sent to the client at each moment. An embedded web application initiates a session with a client and exclusively handles its requests. Similarly to a traditional graphical program, it generates graphical input/output screens and navigates through them based on user responses. The key distinction is that these screens are not graphical pixel screens but HTML5 pages with forms. Like in most servers, we utilize multiple threads, but each one is associated with a specific user and terminates when the interaction with that user ends—either voluntarily or due to the expiration of an inactivity period. According to the OWASP top ten security ranking [7] the developed software directly solve the following risks: Broken access control; Vulnerable and Outdated Components; Security logging and monitoring failures; Server-Side Request Forgery. Additionally, there are other risks that are not directly addressed but can be easily mitigated: Injection; Security Misconfiguration; Identification and Authentication Failures; Software and data integrity failures. Finally, the risks of Cryptographic failures and Insecure design are still present. The main benefits of using EmintWeb rely on its security characteristics and efficiency. Since the web application is in binary, it should be faster than one written in an interpreted language.

Our application starts from a workflow of the task that includes nodes representing the different graphical outputs/forms to be shown (states) and arches between them which represent the data processing done when moving form one state to another (transitions). Obviously, depending on the data introduced by the user in the former state the destination state of a transition may vary.

Similar concepts to those previously described have indeed been explored in the past. The software engineering community has been dealing with the concept of generating web systems from conceptual models for many years. One notable contribution in this area has been made by Brambilla et al. [8], who present a set of models for generating various components of web systems. Another approach, as suggested by Zafar et al. [9], involves the generation of web services from Business Process Modeling Notation (BPMN). Li et al. [10] present a method for creating an executable model to build web systems. These proposals aim to generate web systems or tests using languages specifically tailored for them, requiring execution on a web server. While the use of models enhances the abstraction layer for analysts, it is important to note that the code running on the server remains specific to each web technology.

There are other works that generate web systems from a workflow. One of these works is the one developed by Haller et al. [11]. This work makes the integration of XPDL (a process model) with BPMN to use workflows to abstractly represent choreography interfaces of web systems. The work of Li et al. [12] is called AFlow, an automated service combination system combining artificial intelligence and workflow techniques to build web services. Guerrero-García [13] suggests leveraging workflows to generate web user interfaces. structure, workflow, process, and tasks. All these works use workflow models to help in code generation. However, they focus on specific elements, i.e., web services or user interfaces. There is not a holistic method that generates fully functional web systems.

There are other works aimed at developing web systems using C++. Lima and Eler [14] have defined a C++ Web Framework, which combines C++ with Qt and a tag library called CSTL. Okamoto and Kohana [15] developed a C++ library that serves as an API to build systems in Node.js and Express.js. Szabó and Nehéz [16] defined the Emscripter compiler, which translates C++ code into JavaScript. There are other languages that aim to imitate C++ for the web development. For example, the Rust language, authored by Anderson et al. [17]. Rust is a language that works for a specific browser named Servo. All these works use C++ as source code, but they translate C++ into other languages specific to web systems. Moreover, evaluations have not considered performance tests to study how several clients in several threads may affect the execution of the web system.

After analyzing the related works, we can conclude that the idea of constructing web systems without specific languages for this context has emerged in recent years. This idea aims to assist in the development of web systems for experts in desktop systems without the need to learn a plethora of languages, each specific to a particular context. There are two perspectives: the generation of web code from models and the generation of web code from C++. However, to the best of the authors' knowledge, there are no software packages that generate a web system in native C++ without the use of a web server and without generating code for a web environment. The primary contribution of this paper is to address this gap by proposing EmintWeb, a method to implement web systems in C++ with the assistance of a workflow model. EmintWeb is not a new paradigm to change how web applications are currently developed, but rather as a new framework to be used by web systems that require some of the characteristics it offers.

2. Software description

EmintWeb is a framework to generate embedded web applications. Several libraries are combined and a graphical interface has been programmed to integrate and facilitate the development process. Its main components are:

1. A routine for generating C++ code from HTML5. This includes a HTML5 parser built with the Gumbo libraries and a library (`libhtcpp`) that produces a C++ function (source code) for the state represented by the HTML page. The C++ function takes inputs from the .html file as parameters and returns a string containing the web page (written in HTML) that will be sent to the client. A separate C++ function is generated for each state, and all of them are compiled and integrated into the embedded application.
2. A simple server, either HTTP or HTTPS (at the developer's choice), built with the Poco libraries. This server receives variables instantiated by the client in the response URL, executes the required process with them, and sends the generated page for the next state to the client.

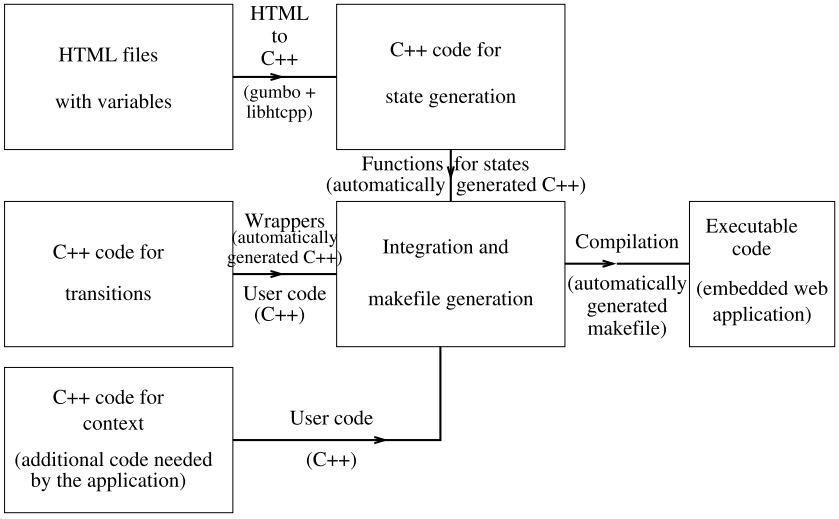


Fig. 1. Architecture of the development framework.

3. Snippets of code, referred to as wrappers, that connect the aforementioned parts with the code written by the developer to implement their business logic.
4. A metaserver, also built with the Poco libraries, that receives the initial request from a client, chooses a port, and launches an instance of the application to serve it. These instances are distinct threads that naturally end when the client voluntarily closes the session or automatically expire after a timeout if no client activity is received.
5. A graphical user interface (GUI) written using the Qt5 libraries to facilitate the drawing of the workflow diagram, automate code generation, and compile it using the Make utility. A screenshot of the GUI is shown in Fig. 2. Alternatively, a command-line interface with the same functionality is also provided.

Automated generation of C++ code covers points 1 to 4, facilitated by a Makefile for compiling both the final server and the metaserver. Users are tasked with crafting HTML pages for individual states (in HTML5) and writing C++ code for transitions between states. Nonetheless, automatic generation also provides prototypes for transition functions and an initial code framework. These components and their interconnections are depicted in the architecture diagram shown in Fig. 1. Steps 1 to 4 can be executed using a command line application (`eiwcli`) so strictly only a text editor, a C++ compiler and the Make utility are needed. Despite this, the use of the GUI (point 5) is very helpful because it opens the HTML and C++ editors for each state/transition and helps in organizing the flow of the application and forcing the developer to think on it exactly the same way as for classical graphical applications: a succession of states and transitions.

The deployment process for the embedded application differs from traditional applications. The sole prerequisites entail installing the generated executable (the metaserver) under a non-privileged user and executing it either via the command line or as a scheduled task. EmintWeb-generated embedded applications operate independently, devoid of any necessity for a global web server or additional services to function. EmintWeb works with several threads, each one for a user. When an error arises in one threat, we capture the signals and process them at the beginning of each thread, before any other code susceptible to failures is executed. This is done by the automatic code generator. The default action for most signals is simply to exit gracefully of the thread and let the other threads go on.

A schema of the application running is depicted in Fig. 3. Each new client initiates a request to the metaserver to open a new thread for it and is redirected to the assigned address and port. This thread remains active as long as the client does not exit from the application or ceases interaction for a while.

Regarding additional content, it depends on the application. If it requires access to additional files (such as images or multimedia content to be served), they must be placed in any pre-configured directory. If it needs a database to manage the data provided by the users, it would require a database server, and communication with it can be established either through local ports or any other method the database accepts. The code to implement these additions is part of the so-called context and must be written by the developer of the application.

The concrete details of building an interactive web application using our framework are outside the limited extension of this text, but a video has been included with exemplifies step by step the development of a simple application, in the style of a “Hello, world!” for programming languages. Please, see section “Additional Material”.

3. Example test

To test the feasibility and capability of the framework, a simple web application was developed in EmintWeb and rewritten with the same appearance and functionality in PHP. This fictitious system manages bus transport routes, storing names and identifiers of cities, buses, passengers, reviewers, and reviews in a database. A bus connects two cities, and at any given time, a generic person (the manager) can board or disembark a passenger from any bus. Additionally, the manager can launch a review of a bus to obtain the list of passengers currently on board. Finally, the manager can retrieve the list of all passengers on board of all non-empty buses. The manager represents a user making these requests to the system, and multiple users are assumed to be performing these actions simultaneously. In the test, this is referred to as the ’Number of clients.’

Experiments were conducted using JMeter [18] to simulate an increasing number of simultaneous clients, ranging from 98 to 20,006. For each request, the response time is measured for those requests answered by the system. Additionally, a record is kept of requests that were not answered (failures).

Table 1 shows the mean and median time in ms. as function of the number of clients, as long as the percentage of attended requests. Each client makes several requests, as needed to complete its task. The same results are shown graphically as boxplots in Fig. 4. A more detailed analysis is included as Additional Material.

The response time for EmintWeb is superior until reaching 12 500 clients. Beyond this threshold, PHP exhibits faster performance. However, this increased velocity is attributed to PHP ignoring 86% of the

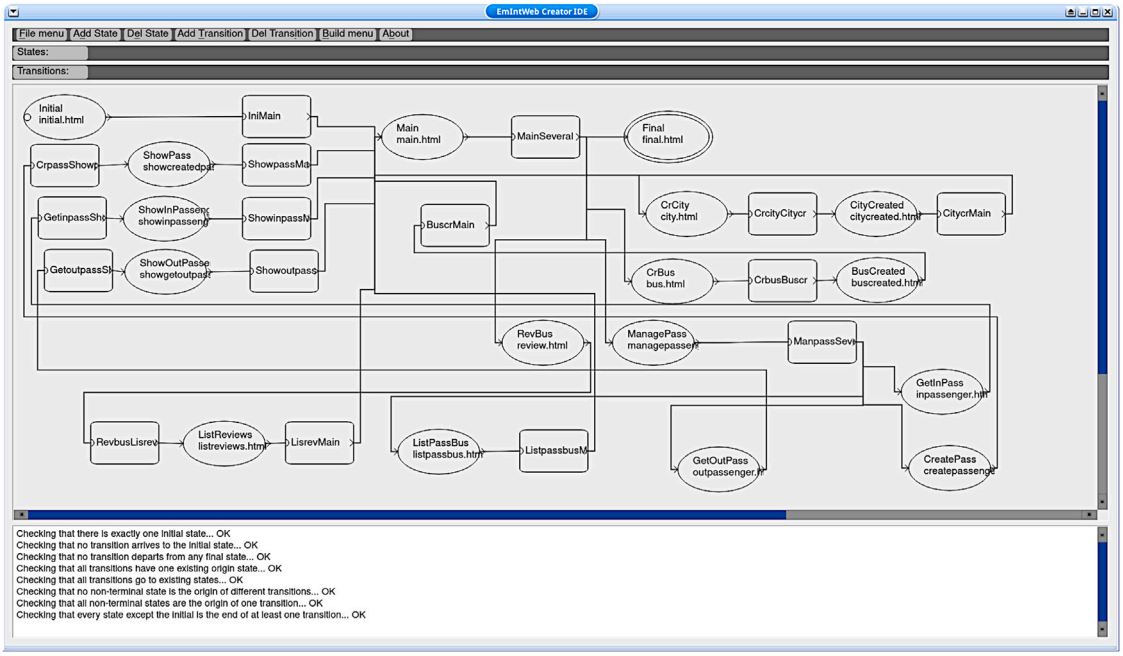


Fig. 2. The GUI with an example project loaded.

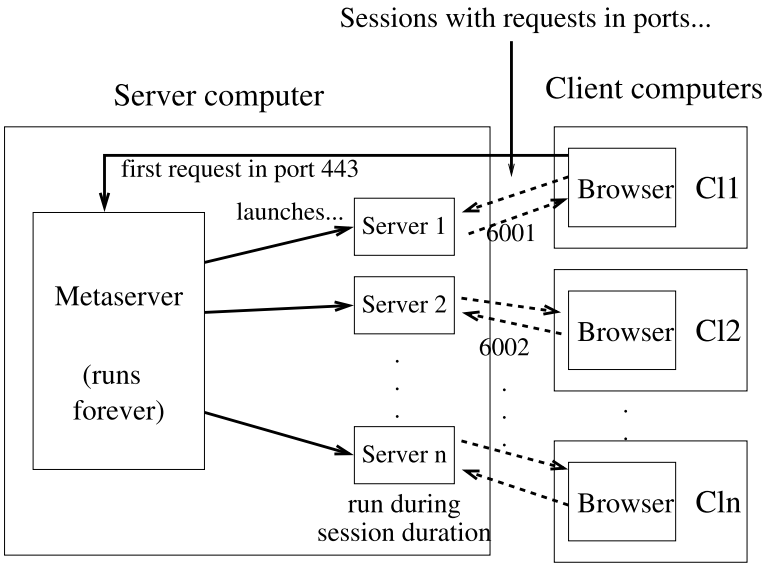


Fig. 3. Schema of an EmintWeb application running.

Table 1
Response time and percentage of failures for EmintWeb and PHP as a function of the number of clients.

	Number of clients									
	98	504	1008	5012	7504	10 010	12 502	15 008	17 500	20 006
EmintWeb										
Mean (ms)	49.49	43.23	48,02	3835.05	6710.60	9183.40	11 881.88	14 266.56	16 534.57	18 760.15
Median (ms)	16.00	16.00	16.00	581.00	850.00	1016.00	1880.00	2485.00	3154.00	3638.00
Att. req. (%)	100.00	100.00	100.00	99.76	99.42	98.66	97.99	97.25	96.44	95.71
PHP										
Mean (ms)	21.88	17.51	22.46	6913.91	11 477.90	17 417.37	1868.65	3071.36	2347.01	2636.07
Median (ms)	13.00	12.00	12.00	6512.00	11 229.00	2902.00	428.00	486.00	610.00	694.00
Att. req. (%)	100.0	99.99	99.99	71.19	46.80	33.23	13.11	12.80	13.24	13.44

requests, whereas EmintWeb successfully responds to 98% of them. Notably, with 20 000 clients, EmintWeb attends to 96% of requests, while PHP only handles 14%. Note that PHP is slightly faster for a small number of connections (under 5.000 connections). This is mainly due to the time needed to negotiate a secure connection every time a new client goes into the system using https. We explicitly check the key files and generate a secure socket with encrypted data for the rest of the interaction, which seems to take some time.

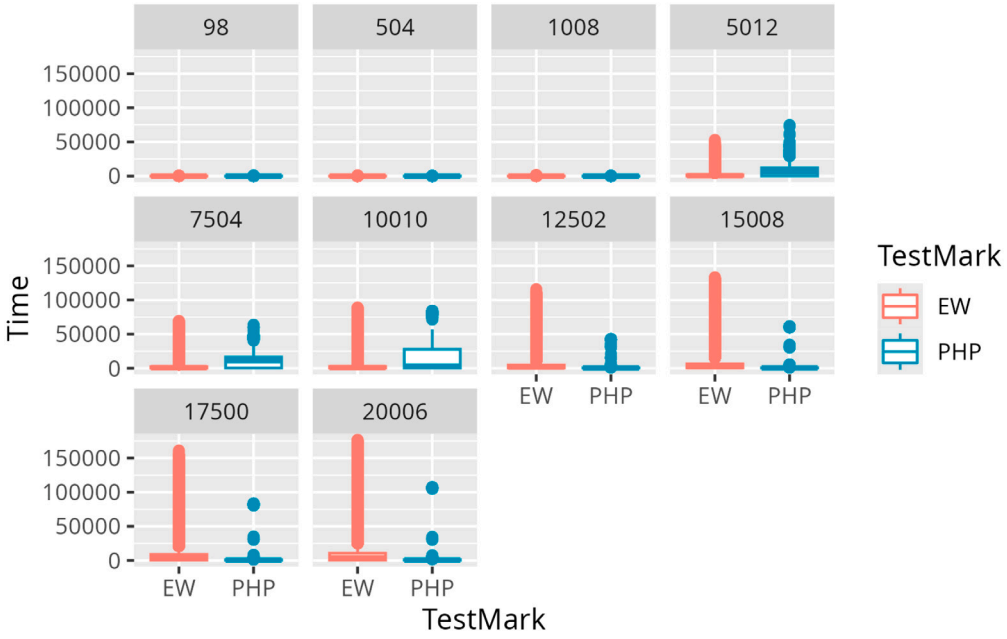


Fig. 4. Box-plots for each number of clients of response time (only for attended requests) of EmintWeb and PHP.

4. Impact

The EmintWeb system is designed to have its main impact in two aspects: security and high availability.

Impact on security:

- The concept of executing each session through a dedicated, individually encrypted channel on a dedicated port for each user enhances security by making attacks more challenging. Impersonation and man-in-the-middle attacks are mitigated through establishing a SSL encrypted connection at all times and checking that client IP does not change during an established session. This makes session cookies unnecessary.
- No code needs to be executed in the client-side, which only renders HTML code (with forms) which constitutes the interface of the application. In fact, JavaScript execution can be disabled, completely avoiding exposure of the server's business logic and enhancing client-side security. Nonetheless, the framework provides the option to include JavaScript in the generated pages if the developer chooses to do so.
- No web pages or server code are present as explicit HTML or JS files in the server so they cannot be altered. They are embedded inside the compiled code (indeed, generated at run time), so their alteration is not directly possible. The attacker would have two options: either build a new executable that reproduces exactly the behavior of the legitimate application and substitute it or alter the executable itself (for instance changing the area of static variables where the javascript code of the page, if any, would be stored as text). To prevent these kind of attacks the application generated by EmintWeb (the executable) should be installed as owned by a special user which has read and execute, but not write permissions. For an even more secure environment the use of tools for file monitoring and integrity checking like AIDE (Advanced Intrusion Detection Environment, [19]) is also advisable.
- The stateless of HTTP/HTTPS does not arise: a single program contains variables, which are in memory during all its execution. No database or additional provisions are needed to keep their values.
- Injection of spurious data by an authenticated, malicious user is still possible, and must be checked by the application's programmer, but it is more likely to provoke a crash of the thread of a particular session and not a global breach.
- There is only one entry point to the server: the URL with variables sent by the client. First, a default check is in place to prevent overflow of the URL+variables string and it is straightforward to check that only the expected variable names and sensible values for them are provided.
- The lines of code automatically generated by EmintWeb have been carefully written to be as robust and safe as possible, incorporating thorough error checks and exception handling [20]. Note that errors are not the only downfall of threads; memory leaks and stack leaks, are prevalent and are not directly dealt with by EmintWeb, even the usual precautions taken by the compilers, namely Address Space Layout Randomization (ASLR) and stack canaries, are applied by default.

Impact on availability and speed:

- Compiled code runs faster than current interpreted code so the same server can attend a higher number of simultaneous requests.
- Load distribution within the server is automatically managed by the operating system scheduler, which fairly allocates threads (i.e., clients) among the multiple cores of modern processors. Since in a real application these threads will be most of the time awaiting for user interaction, the hyperthreading capability of some current processors is also beneficial.

Limitations:

- Use of threads, even beneficial for efficiency reasons, opens the door to stack overflow attacks in one thread that could compromise the security of the whole server. The only way to mitigate this would be the use of full processes instead of threads. The changes in the automatically generated code are not drastic and will be introduced in a future version so the user will be allowed to choose between a multi-process and a multi-threaded version.

Summarizing, EmintWeb is suitable for web applications that require the parallel connection of many clients, quick response times, and secure connections to the server. An example of an application with

such requirements is an online banking platform. This type of application handles sensitive financial information, must process transactions quickly, and support access by thousands of users simultaneously. Also, it would be suitable for those clients who prefer not to allow any external code, not even Javascript scripts, to run in their own machines, either by security or by license-related reasons.

5. Conclusions

This work focuses on the development of EmintWeb, a methodology for building web systems using C++ and a workflow model. The core concept revolves around reversing the conventional approach: instead of the server launching the application, the application is written as a traditional, fully sequential program responsible for executing the server. While previous discussions have highlighted the advantages of this approach, such as enhanced security, simplified deployment, and improved execution speed, it is crucial to acknowledge some limitations as well:

- The number of simultaneous clients is limited by the design of the TCP/IP protocol which allows up to 65,535 ports (and the first 1024 privileged ports must be excluded). However, it is possible to instruct the metaserver to redirect new clients to other servers (in different machines) simply by sending to such servers a signal to open a new thread on any desired port. This approach is also a viable method for load balancing, which will be implemented in future work.
- Each thread consumes a certain amount of memory, and having too many threads could lead to memory exhaustion. Developers must exercise discipline and use constructors and destructors of objects judiciously. A future modification is planned to precompile web pages in memory, reducing the need for extensive memory usage, as only string substitution will be required to generate each particular page.
- While it is not possible to alter the web pages or server code, it could be possible for a hacker with sufficient privileges to substitute the executable of the application with a malicious one. However, this risk is inherent to all systems and, to the best of our knowledge, depends on the security measures implemented by the server itself.

CRediT authorship contribution statement

Juan Domingo: Writing – original draft, Validation, Software, Conceptualization. **Jose Ignacio Panach:** Writing – original draft, Validation, Formal analysis, Conceptualization. **Esther Dura:** Writing – review & editing, Validation, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This publication is part of the I+D+i project PGC type B with reference PID2020-117114GB-I00 funded by the MCIU Spanish Ministry of Science, Innovation and Universities, MCIN/AEI/10.13039/501100011033/. This is also supported by the project TENTACLE (GVRTE/2023/4592166) from Generalitat Valenciana, Spain.

Appendix A. Supplementary data

Detailed results are described as the .pdf file *SupplementaryMaterial1.pdf*

A video that explains EmintWeb and builds a simple "Hello, World!" application is available as .mp4 file *EmintWeb.mp4*

A virtual machine with EmintWeb and the example application installed is available in https://johnford.uv.es/EmintWeb.

A video for the setup of that virtual machine is available as .mp4 file *VirtualMachineSetup.mp4*

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.softx.2024.101809.

References

[1] Oluwatosin Haroon Shakirat. Client–server model. IOSR J Comput Eng 2014;16(1):67–71.
[2] Wang Yao, Cai Wan-dong, Wei Peng-cheng. A deep learning approach for detecting malicious javascript code. Secur Commun Netw 2016;9(11):1520–34.
[3] Sadeghian Amirmohammad, Zamani Mazdak, Abdullah Shahidan M. A taxonomy of sql injection attacks. In: 2013 international conference on informatics and creative multimedia. 2013, p. 269–73.
[4] Ihrig Colin J. HTTP. Berkeley, CA: A Press; 2013, p. 167–88.
[5] Beltran Vicenc, Torres Jordi, Ayguade Eduard. Understanding tuning complexity in multithreaded and hybrid web servers. In: 2008 IEEE international symposium on parallel and distributed processing. 2008, p. 1–12.
[6] Monshizadeh Maliheh, Naldurg Prasad, Venkatakrishnan VN. Mace: Detecting privilege escalation vulnerabilities in web applications. In: Proceedings of the 2014 ACM SIGSAC conference on computer and communications security. New York, NY, USA: Association for Computing Machinery; 2014, p. 690–701.
[7] Open Web Application Security Project. Owasp. 2024, https://owasp.org/. [Online accessed 16 May 2024].
[8] Brambilla Marco, Cabot Jordi, Wimmer Manuel. Model-driven software engineering in practice. Morgan & Claypool Publishers; 2017.
[9] Zafar Iqra, Azam Farooque, Anwar Muhammad Waseem, Maqbool Bilal, Butt Wasi Haider, Nazir Aiman. A novel framework to automatically generate executable web services from bpmn models. IEEE Access 2019;7:93653–77.
[10] Li Liping, Gao Honghao, Shan Tang. An executable model and testing for web software based on live sequence charts. In: 2016 IEEE/aCIS 15th international conference on computer and information science. 2016, p. 1–6.
[11] Haller Armin, Marmolowski Mateusz, Gaaloul Walid, Oren Eyal, Sapkota Brahmanada, Hauswirth Manfred. From workflow models to executable web service interfaces. In: 2009 IEEE international conference on web services. 2009, p. 131–40.
[12] Li Xin, Tang Xinhuai, Song Zhaoteng, Yuan Xiaozhou, Chen Delai. Aflow: An automated web services composition system based on the ai planning and workflow. In: 2010 IEEE international conference on progress in informatics and computing, vol. 2. 2010, p. 1067–71.
[13] Guerrero-García Josefina, González-Calleros Juan Manuel, González-Monfil Adelaida, Pinto David. A method to align user interface to workflow allocation patterns. In: Proceedings of the XVIII international conference on human computer interaction. Association for Computing Machinery; 2017.
[14] Lima Herik, Eler Marcelo Medeiros. C++ web framework: A web framework for web development using c++ and qt. In: Proceedings of the 23rd international conference on enterprise information systems - vol. 2. INSTICC, SciTePress; 2021, p. 76–87.
[15] Okamoto Shusuke, Kohana Masaki. A c++ header library for web applications. In: 2016 19th international conference on network-based information systems. 2016, p. 541–5.
[16] Szabó Martin, Nehéz Károly. C/c++ applications on the web. Prod Syst Inf Eng 2019;8:69–87, Copyright - Copyright University of Miskolc 2019; Última actualización - 2023-12-03.
[17] Anderson Brian, Bergstrom Lars, Goregaokar Manish, Matthews Josh, McAllister Keegan, Moffitt Jack, et al. Engineering the servo web browser engine using rust. In: 2016 IEEE/ACM 38th international conference on software engineering companion. 2016, p. 81–9.
[18] The Apache software foundation. Apache jmeter. 2010-2023, https://jmeter.apache.org/index.html. [Online accessed 16 March 2024].
[19] Lehti R, Virolainen P, Kemelen P, Markley M, Grubb S, van den Berg R, et al. Advanced intrusion detection environment. 2024, https://github.com/aide. [Online accessed 12 June 2024].
[20] Shiina Shumpei, Iwasaki Shintaro, Taura Kenjiro, Balaji Pavan. Lightweight preemptive user-level threads. In: Proceedings of the 26th ACM SIGPLAN symposium on principles and practice of parallel programming. 2021, p. 374–88.