POLITECNICO DI TORINO
Repository ISTITUZIONALE

Leveraging Large Language Models for End User Website Generation

Original
Leveraging Large Language Models for End User Website Generation / Calò, T., De Russis, L.. - ELETTRONICO. - (2023), pp. 52-61. (IS-EUD: the 9th International Symposium on End-User Development Cagliari (Italy) 06-08 June 2023) [10.1007/978-3-031-34433-6_4].

Availability:
This version is available at: 11583/2978033 since: 2025-08-04T15:59:41Z

Publisher:
Springer

Published
DOI:10.1007/978-3-031-34433-6_4

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
Springer postprint/Author's Accepted Manuscript (book chapters)

This is a post-peer-review, pre-copyedit version of a book chapter published in End-User Development. The final authenticated version is available online at: http://dx.doi.org/10.1007/978-3-031-34433-6_4

(Article begins on next page)

28 November 2026

Leveraging Large Language Models for End-User Website Generation

Tommaso Calò[0000−0002−3200−2348] and Luigi De Russis[0000−0001−7647−6652]

Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Turin, Italy
{tommaso.calo, luigi.derussis}@polito.it

Abstract. This work introduces an innovative approach that harnesses the power of large language models (LLMs) to facilitate the creation of websites by end users through natural language specifications. Our key contribution lies in a user-oriented method that utilizes prompt engineering, compelling the LLM response to adhere to a specific template, which in turn enables direct parsing of the model's responses, allowing users to focus on refining the generated website without concerning themselves with the underlying code. The engineered prompt ensures model efficiency by implementing a modification strategy that preserves context and tokens generated in the LLM responses, updating only specific parts of the code rather than rewriting the entire document, thereby minimizing unnecessary code revisions. Moreover, our approach empowers LLMs to generate multiple documents, augmenting the user experience. We showcase a proof-of-concept implementation where users submit textual descriptions of their desired website features, prompting the LLM to produce corresponding HTML and CSS code. This paper underscores the potential of our approach to democratize web development and enhance its accessibility for non-technical users. Future research will focus on conducting user studies to ascertain the efficacy of our method within existing low-code/no-code platforms, ultimately extending its benefits to a broader audience.

1 Introduction

The proliferation of the Internet has fundamentally changed the way we live, work, and communicate, leading to a substantial demand for website development. Traditional website development typically necessitates technical expertise, which can be a barrier for many individuals without these skills. In response, researchers and practitioners have sought to develop tools and approaches to enable end-users to create websites without coding. End-User Development (EUD) [2, 21, 16, 8] has emerged as a popular approach to enable non-technical users to create websites [13, 17]. Low-code/no-code tools have been developed to support EUD by offering increased ease of use and flexibility [23, 26, 7, 9, 28]. These tools enable users to create websites by visually arranging pre-built components, such as buttons, images, and forms, with the underlying code generated automatically. Although these tools have simplified website development, they can be limited

in flexibility and may be challenging to use for more complex websites [1, 11, 25, 12]. One of the primary limitations of low-code/no-code tools is the steep learning curve associated with their usage. Users often need to invest significant time learning how to use these tools effectively, which can be a substantial barrier for non-technical users [18].

Recently, there has been growing interest in leveraging artificial intelligence (AI) to enable end users to create websites [22]. Large language models (LLMs) have emerged as a promising approach for generating code based on natural language descriptions provided by end users. LLMs are trained on massive amounts of data and can generate text that closely matches human language, making them well-suited for generating code from natural language input [4, 29, 6]. However, these approaches have certain limitations, such as not allowing users to refine the output of the LLM with subsequent input or generating multiple pages. These limitations can hinder users from creating websites tailored to their specific needs and preferences.

To address these limitations, we propose a novel approach for leveraging LLMs for EUD using natural language processing to generate code from specifications. This approach is familiar and intuitive for most people, as it utilizes natural language communication [19, 30]. Our method is centered around prompt engineering, which constrains the LLM response to follow a predefined template, facilitating the direct parsing of the model's output. This enables users to concentrate on refining their generated websites without the need to delve into the underlying code. Our technique allows users to iteratively adjust the LLM output and create multiple pages, offering greater adaptability and command over the generated code. With minimal technical expertise required, our approach bypasses the necessity to master programming language syntax, structure, or web development tools, considerably reducing the learning curve typically associated with website development. In addition, the approach incorporates an efficient prompting strategy to interact with external LLM APIs [15], which enables generating code that benefits from a larger context window[1] thus enabling refinements that reference earlier parts of the conversation and earlier generated documents. By maintaining a longer context, users can create more complex websites that refer to multiple pages, enhancing the overall functionality and richness of the generated content [15]. The engineered prompt ensures model efficiency by implementing a modification strategy that preserves context and tokens generated in the LLM responses, updating only specific parts of the code rather than rewriting the entire document, thereby minimizing unnecessary code revisions. This method also results in fewer tokens being generated, leading to cost savings, as the cost of API usage is related to the number of generated tokens.

To demonstrate the feasibility of this approach, we present a proof-of-concept implementation where users input textual descriptions of their website require-

[1] The context window refers to the amount of information an LLM can process at once. Preserving context is essential for interactions with LLMs, as it allows user to reference earlier parts of the conversation within the same generation process.

ments, and the LLM processes this input, generating HTML and CSS code to construct the desired website. This proof-of-concept showcases LLMs' potential in automating website development, reducing the time and effort required. In summary, our work builds on the emerging areas of EUD and LLMs, aiming to address existing limitations by empowering users to refine the LLM output with subsequent input and generate multiple pages. Our approach has the potential to democratize website development and significantly help bridge the digital divide.

2 Background and Related Works

End-user development (EUD) has attracted significant attention in recent years, as researchers and practitioners strive to make website development more accessible to non-technical users [2, 13, 16, 8, 27, 21]. Low-code/no-code tools have emerged as a leading approach to EUD, offering users the ability to create websites without coding expertise [23, 26, 7, 1, 9, 28]. Despite their popularity, these tools often present limited flexibility and can be challenging to use for developing complex websites [18, 11, 25, 12].

Interestingly, an early prototype of an AI-assisted EUD solution was presented to the AI and HCI community over 50 years ago, referred to as retrieval by reformulation, and had a system called RABBIT as its primary example [24]. This work, although significant, did not gain widespread adoption outside of the research community. Nonetheless, it is important to acknowledge its contribution to the field as it laid the foundation for future developments.

In recent times, large language models (LLMs) have been identified as a promising avenue for EUD [4, 29]. LLMs, trained on extensive data, can generate natural language text that closely resembles human language. This capability makes them well-suited for generating code based on textual descriptions provided by end-users.

Several studies have explored the use of LLMs for EUD. Huang et al. [10], for example, proposed a framework that automatically generates website layouts from textual descriptions using LLMs. Chen et al. [6], instead, developed a method for generating code snippets from natural language queries using LLMs. These studies underscore the potential of LLMs for EUD and provide a strong foundation for our research. Our work is unique in its focus on a user-oriented approach that allows end users to refine the LLM output with subsequent input iteratively. This feature empowers users to have greater control over the generated code, ensuring it meets their specific requirements. This refinement process is a critical aspect that differentiates our work from earlier studies on EUD using LLMs. While previous research in EUD using LLMs primarily focused on generating single pages, our work expands this research scope by enabling LLMs to generate multiple pages. It highlights the importance of designing AI systems that ensure high control and high automation, qualities essential for systems interacting with humans.

In addition to the existing body of research on EUD and LLMs, it is essential to consider the relationship between our approach and AI tools that

convert hand-drawn website designs into complete systems (e.g., [3, 5]). These tools provide an intuitive way for non-technical users to create websites without requiring knowledge of specific website structures and terms. In contrast, our LLM-based approach might still necessitate that users are familiar with technical terms to effectively communicate their design intentions, as illustrated by the use of "navbar" in Figure 3.

AI services that generate websites based on drawings can offer a more accessible alternative for users who lack knowledge of website terminology. However, our proposed LLM-based approach presents several advantages. By utilizing natural language input, the method encourages iterative refinement of generated code, providing users with greater control and customization options. Moreover, our approach can potentially accommodate a wider range of user preferences and design complexities, as it is not limited to interpreting visual representations.

That said, a discussion comparing the intuitiveness and accessibility of our LLM-based approach and AI tools that convert drawings into websites is warranted. Future research could explore ways to combine the strengths of both methods, creating a more comprehensive and user-friendly solution for EUD. Integrating hand-drawn elements with natural language input could potentially lead to a more intuitive and powerful tool that caters to users with varying levels of technical knowledge and design skills.

To conclude, our research makes a contribution to the EUD and LLM domains by presenting an user-oriented approach that places end-users at the forefront of the website development process. By offering the ability to refine LLM outputs with subsequent input and generate multiple pages, we provide users with a high level of control and customization options [20], thus creating websites tailored to their specific needs and preferences.

3 Methodology

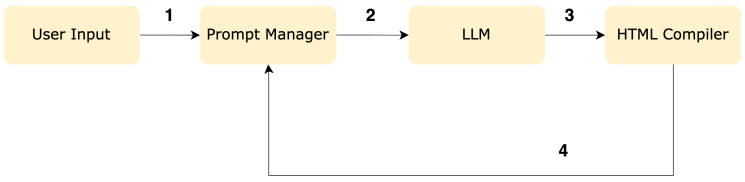


Fig. 1. The interactive website development process: (1) User provides input, (2) Prompt manager processes input, (3) LLM generates HTML code, (4) HTML compiler checks for errors, and if any are detected, the process loops back to the prompt manager for refinement.

In our approach, we utilize a technique that forces the LLM to follow a specific response template, as outlined by the given prompt in Figure 2. The

primary goal is to ensure that the generated code is structured and adheres to the user's specifications. The LLM is guided by a set of rules that dictate the format of the generated responses. By leveraging this template-based approach, end users can focus on their desired website functionality and design without concerning themselves with the underlying code, as the requests and responses are parsed in the specified format and rendered as HTML by the system.

When creating a new document, the LLM follows the response structure: `new, ⟨document name⟩, ⟨code⟩`. In cases where a document requires modifications, the LLM adheres to a response format that avoids outputting the entire code, opting for a more efficient strategy: `⟨document name⟩; ⟨add or replace⟩, ⟨n1-n2 range of lines if replace, n1 if add⟩,⟨code⟩; ... ⟨add or replace⟩, ⟨n1-n2 range of lines if replace, n1 if add⟩ ; ⟨code⟩`. This technique ensures that only necessary modifications are made, preserving the original code and avoiding unnecessary API responses. The LLM takes the user's request as input, with the format `request:⟨request⟩`, and generates responses according to the aforementioned template. By adhering to the specified format, the LLM efficiently modifies existing documents, only reporting the modification lines and modifications as needed, while leaving unmodified parts of the document untouched.

Furthermore, the technique facilitates error detection and resolution. The structured response format allows the parser to identify any errors in the generated code, providing the user with the necessary information to address these issues. In cases where the LLM generates code with compiling errors, the system can rectify the problem directly by prompting the LLM with the error and the code to correct, as illustrated in Figure 1. This streamlined approach to error detection and resolution saves time and effort, making the web development process more accessible and efficient for non-technical users.

3.1 Iterative Refinement and Multiple Pages Generation

The methodology allows users to refine the output of the LLM with subsequent input, providing them with greater flexibility and control over the generated code. This iterative process ensures that the final website design closely matches the users' requirements and preferences. Users can provide feedback and request changes in real-time, allowing them to actively shape the development process and avoid time-consuming revisions after the website has been generated. Moreover, the approach enables the generation of multiple pages, further enhancing the user experience and providing a more comprehensive website development solution. Users can create interconnected pages with varying designs and content, allowing for the development of complex and feature-rich websites without needing extensive technical expertise.

3.2 Variety of Design Options

The proposed approach offers users the possibility to choose between a variety of design options in the generated documents. By providing diverse design alterna-

Prompt:
You have been asked to create HTML and CSS code based on the user's specifications. You can create multiple HTML documents, but only one CSS document which will contain the page's style. I will tell you the format of needed responses, you must strictly follow the following response format, and you must not output other words that are not contained in the formats.
If you need to create a new document, your response must be in the form of: new, ⟨document name⟩, ⟨code⟩.
If you need to modify a document already generated in another response, you must not output the whole code, even if the modification is large, you must use the following format for your response:
⟨document name⟩; ⟨add or replace⟩, ⟨n1-n2 range of lines if replace, n1 if add⟩,⟨new line⟩; ... ⟨add or replace⟩, ⟨n1-n2 range of lines if replace, n1 if add⟩; ⟨new line⟩.
If no changes are required to a given document you must not output nothing.
You need to specify ¡add¿ if the line must be added to the specified line number while ¡replace¿ if the line at the specified range must be replaced to accomplish the modification.
Note that the user's request will be inputted as "request:⟨request⟩".
Also, if you need to modify an existing document, please only report the modification lines and modification in the format specified above, as efficiently as possible. In order to not rewriting unmodified parts of the document.

Fig. 2. The prompt engineered to create and modify HTML and CSS documents based on natural language specifications, with strict response formats for creating new documents and updating existing ones.

tives, users can explore different aesthetics and layouts for their website, ensuring that the final product aligns with their desired look and feel. This feature adds an extra layer of customization and adaptability to the website development process, empowering users to create a unique and personalized online presence.

To facilitate the selection of design options, the methodology can incorporate predefined templates or design components that the LLM can use as a starting point. Users can then refine and customize these templates based on their preferences, allowing them to quickly create visually appealing websites without starting from scratch. The LLM can also learn from user feedback during the iterative refinement process, further improving the quality of the generated design options and adapting to the users' specific needs.

3.3 Efficient Prompting Strategy

The efficient prompting strategy involves asking the LLM only to respond with the number of lines to modify and the specific modifications, instead of rewriting the entire document. This approach enables the generation of code that benefits from a larger context window, as fewer interactions and tokens are generated. As

a result, the LLM can accommodate a more extended sequence of refinements, leading to better model performance and improved user experience.

One of the core challenges in leveraging LLMs for website development is managing the limitations imposed by the maximum token length of the model. Other approaches often involve generating the entire codebase in one go, which may result in exceeding the token limit. By using the efficient prompting strategy, the methodology allows the model to focus on the most relevant portions of the code, thus reducing the likelihood of exceeding the token limit. Minimizing the number of generated tokens is essential for reducing the cost of API usage, as the cost is directly related to the number of tokens generated. This not only makes the methodology more affordable for end-users but also enables more extensive usage of LLM resources for website development.

Additionally, a larger context window allows the model to process longer sequences of text, enhancing its understanding of the user's requirements and improving its ability to generate accurate and contextually relevant code. Maximizing the context window also helps the LLM to maintain coherence across the generated code, ensuring that the resulting website maintains a consistent design and structure. As the LLM has access to more contextual information, it can make better-informed decisions when generating code, improving the overall quality of the generated website.

By generating fewer tokens, the proposed approach leads to cost savings and better resource usage, as discussed in the previous sections. This not only makes the methodology more affordable for end users but also enables more extensive usage of LLM resources for website development. The efficient use of API resources can also lead to faster response times and a more seamless experience for users when interacting with the LLM.

3.4 Proof of Concept

To demonstrate the effectiveness of our approach, we have developed a proof-of-concept implementation using GPT-4 [14], showcasing the technique's practical application. Figure 3 illustrates the sequential website development natural language instructions with the GPT-4 model. The figure highlights the interaction format, enabling readers to understand the structured responses and the methodology applied. Although the actual code in the responses is not shown, the reported responses provide sufficient information to comprehend the template and the format used in the GPT-4 interaction. This proof-of-concept serves as a tangible example of how our approach can be employed in real-world scenarios to create and refine websites using natural language specifications and the power of LLMs, ultimately streamlining the web development process for end users.

4 Conclusion

The proposed methodology consists of an efficient prompting strategy for interacting with external LLM APIs, which optimizes resource usage and enhances

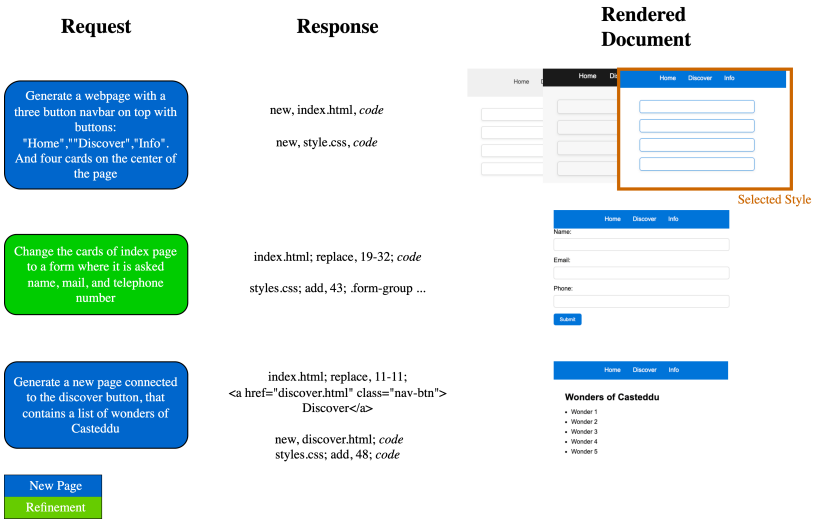


Fig. 3. Sequential website development process visualized in columns and rows: Columns display the Request, Response, and Rendered Page; Rows showcase three requests – the first and third for generating new pages, and the second for refining the existing page. Note that the code in the response is not reported, however, the reported responses should let readers understand the format used to interact with the LLM.

the user experience. By focusing on minimizing the number of generated tokens and maximizing the context window, this approach enables cost savings, better resource usage, improved model performance, and more refined control over the generated code.

The provision of a variety of design options and iterative refinement further adds to the customization and adaptability of the website development process. By addressing the core challenges of LLM-based website development, such as token limitations and contextual understanding the proposed methodology is ready to be integrated with existing low-code/no-code tools to enable a wider audience to benefit from the technology.

As a future work, the approach can be integrated with a low-code/no-code platform to ascertain the efficacy and utility of the methodology, through user studies. This integration can also streamline the development process by providing an interface for users to interact with the LLM, further enhancing the overall user experience. The final goal will always be to democratize website development and make it more accessible to users without technical expertise.

References

1. Alamin, M.A.A., Malakar, S., Uddin, G., Afroz, S., Haider, T., Iqbal, A.: An empirical study of developer discussions on low-code software development challenges. pp. 46–57 (05 2021). https://doi.org/10.1109/MSR52588.2021.00018

2. Barricelli, B.R., Cassano, F., Fogli, D., Piccinno, A.: End-user development, end-user programming and end-user software engineering: A systematic mapping study. Journal of Systems and Software **149**, 101–137 (2019). https://doi.org/https://doi.org/10.1016/j.jss.2018.11.041
3. Beltramelli, T.: Pix2code: Generating code from a graphical user interface screenshot. In: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems. EICS '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3220134.3220135, https://doi.org/10.1145/3220134.3220135
4. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D.: Language models are few-shot learners. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 1877–1901. Curran Associates, Inc. (2020), https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf
5. Calò, T., De Russis, L.: Style-aware sketch-to-code conversion for the web. In: Companion of the 2022 ACM SIGCHI Symposium on Engineering Interactive Computing Systems. p. 44–47. EICS '22 Companion, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3531706.3536462, https://doi.org/10.1145/3531706.3536462
6. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)
7. Di Ruscio, D., Kolovos, D., Lara, J., Pierantonio, A., Tisi, M., Wimmer, M.: Low-code development and model-driven engineering: Two sides of the same coin? Software and Systems Modeling **21** (01 2022). https://doi.org/10.1007/s10270-021-00970-2
8. Ghiani, G., Paternò, F., Spano, L.D., Pintori, G.: An environment for end-user development of web mashups. International Journal of Human-Computer Studies **87**, 38–64 (2016). https://doi.org/https://doi.org/10.1016/j.ijhcs.2015.10.008
9. Gomes, P.M., Brito, M.A.: Low-code development platforms: A descriptive study. In: 2022 17th Iberian Conference on Information Systems and Technologies (CISTI). pp. 1–4 (2022). https://doi.org/10.23919/CISTI54924.2022.9820354
10. Huang, F., Li, G., Zhou, X., Canny, J.F., Li, Y.: Creating user interface mock-ups from high-level text descriptions with deep-learning models. arXiv preprint arXiv:2110.07775 (2021)
11. Käss, S., Strahringer, S., Westner, M.: Drivers and inhibitors of low code development platform adoption. In: 2022 IEEE 24th Conference on Business Informatics (CBI). vol. 01, pp. 196–205 (2022). https://doi.org/10.1109/CBI54897.2022.00028
12. Luo, Y., Liang, P., Wang, C., Shahin, M., Zhan, J.: Characteristics and challenges of low-code development: The practitioners' perspective. In: Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). ESEM '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3475716.3475782
13. Namoun, A., Daskalopoulou, A., Mehandjiev, N., Xun, Z.: Exploring mobile end user development: Existing use and design factors. IEEE Transactions on Software Engineering **42**(10), 960–976 (2016). https://doi.org/10.1109/TSE.2016.2532873

14. OpenAI: Gpt-4 technical report (2023)
15. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C.L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al.: Training language models to follow instructions with human feedback. arXiv preprint arXiv:2203.02155 (2022)
16. Rode, J., Rosson, M.B., Qui˜nones, M.A.P.: End User Development of Web Applications, pp. 161–182. Springer Netherlands, Dordrecht (2006). https://doi.org/10.1007/1-4020-5386-X_8
17. Rosson, M.B., Sinha, H., Bhattacharya, M., Zhao, D.: Design planning in end-user web development. In: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007). pp. 189–196 (2007). https://doi.org/10.1109/VLHCC.2007.45
18. Sahay, A., Indamutsa, A., Di Ruscio, D., Pierantonio, A.: Supporting the understanding and comparison of low-code development platforms. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 171–178 (2020). https://doi.org/10.1109/SEAA51224.2020.00036
19. Sales, J.E., Freitas, A., Oliveira, D., Koumpis, A., Handschuh, S.: Revisiting principles and challenges in natural language programming. In: Virvou, M., Nakagawa, H., C. Jain, L. (eds.) Knowledge-Based Software Engineering: 2020. pp. 7–19. Springer International Publishing, Cham (2020)
20. Shneiderman, B.: Human-centered AI. Oxford University Press (2022)
21. Sinha, N., Karim, R., Gupta, M.: Simplifying web programming. In: Proceedings of the 8th India Software Engineering Conference. p. 80–89. ISEC '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2723742.2723750
22. Stocco, A.: How artificial intelligence can improve web development and testing. In: Companion Proceedings of the 3rd International Conference on the Art, Science, and Engineering of Programming. Programming '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3328433.3328447
23. Symmonds, N.: Visual Web Developer (01 2006). https://doi.org/10.1007/978-1-4302-0180-9_5
24. Tou, F.N., Williams, M.D., Fikes, R., Henderson, A., Malone, T.: Rabbit: An intelligent database assistant. In: Proceedings of the Second AAAI Conference on Artificial Intelligence. p. 314–318. AAAI'82, AAAI Press (1982)
25. Tzafilkou, K., Protogeros, N.: Diagnosing user perception and acceptance using eye tracking in web-based end-user development. Computers in Human Behavior **72**, 23–37 (2017). https://doi.org/https://doi.org/10.1016/j.chb.2017.02.035
26. Waszkowski, R.: Low-code platform for automating business processes in manufacturing. IFAC-PapersOnLine **52**(10), 376–381 (2019). https://doi.org/https://doi.org/10.1016/j.ifacol.2019.10.060, 13th IFAC Workshop on Intelligent Manufacturing Systems IMS 2019
27. Wong, J.: Marmite: Towards end-user programming for the web. In: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007). pp. 270–271 (2007). https://doi.org/10.1109/VLHCC.2007.40
28. Woo, M.: The rise of no/low code software development—no experience needed? Engineering **6** (07 2020). https://doi.org/10.1016/j.eng.2020.07.007
29. Wu, C., Yin, S., Qi, W., Wang, X., Tang, Z., Duan, N.: Visual chatgpt: Talking, drawing and editing with visual foundation models (2023). https://doi.org/10.48550/ARXIV.2303.04671, https://arxiv.org/abs/2303.04671
30. Xu, F.F., Vasilescu, B., Neubig, G.: In-ide code generation from natural language: Promise and challenges. ACM Trans. Softw. Eng. Methodol. **31**(2) (mar 2022). https://doi.org/10.1145/3487569