DIJEMSS: Dinasti International Journal of Education Management and Social Science
E-ISSN: 2686-6331
P-ISSN: 2686-6358
https://dinastipub.org/DIJEMSS dinasti.info@gmail.com +62 811 7404 455
DOI: https://doi.org/10.38035/dijemss.v7i1

Automating Mobile WEB App Prototyping from Wireframe Sketches

Fajrul Hidayat[1], Darlis Herumurti[2]
[1]Department of Informatics Institut Teknologi Sepuluh Nopember Surabaya, Indonesia, 6025221030@student.its.ac.id
[2]Department of Informatics Institut Teknologi Sepuluh Nopember Surabaya, Indonesia, darlis@if.its.ac.id

Corresponding Author: 6025221030@student.its.ac.id[1]

Abstract: Manual development of mobile web app prototypes is time-consuming, costly, and error-prone, particularly in arranging UI elements. This study aims to develop an automated system based on Deep Learning that can directly convert wireframe sketches into HTML and Bootstrap CSS code. The system employs the YOLOv11 algorithm to detect UI elements in the wireframe sketches, which are then translated into a hierarchical structure using a Domain-Specific Language (DSL) as an intermediary between detection results and code generation. The UI elements arranged within the DSL structure are subsequently converted into HTML and Bootstrap CSS code using a template-based approach with the Bootstrap 5 framework, ensuring layout consistency with the original design. The results show that the system is capable of generating prototypes with an element conversion success rate exceeding 85%. Evaluations conducted with several respondents indicate that the system can accelerate the early design process of web applications, reduce manual errors, and optimize resource usage in software development. These findings highlight the potential of integrating DSL and Deep Learning to automate the generation of accurate and efficient web UI prototypes from wireframe sketches.

Keywords: Sketch to Code, YOLOv11, Wireframe, Deep Learning.

INTRODUCTION

The software development process is a complex activity that requires a systematic and structured approach to ensure the successful creation of high-quality systems that meet user needs. One of the most widely used frameworks in software development is the Software Development Life Cycle (SDLC). SDLC is a methodological framework that defines the stages involved in software development, from requirements analysis and design to implementation, testing, and post-deployment maintenance. The main goal of this approach is to create software systems that not only function according to technical specifications but also optimally fulfill end-user expectations and needs.

Within the SDLC, prototyping plays a highly important and strategic role. A prototype is an early representation of the system to be developed, whether in the form of user interface

sketches, limited functional models, or simulations of business processes to be automated. According to Iglesias (2023), prototypes serve as effective visual tools to bridge communication between developers and stakeholders, such as clients, end users, and project managers. Through prototypes, stakeholders can better understand and evaluate the system concept, even before any actual code is written.

Integrating prototyping into the SDLC offers many strategic advantages. This process not only enhances communication and collaboration but also speeds up decision-making, reduces the risk of errors, and increases user satisfaction. In today's software development landscape, where speed, flexibility, and user-centered design are emphasized, prototyping has become indispensable in delivering innovative and impactful software solutions.

However, despite the many benefits of prototyping such as improved communication, faster feedback loops, and facilitated design iterations there are also limitations that cannot be overlooked. One major challenge is the significant time and resources required when prototypes are created manually. According to Yang et al. (2020), the manual prototyping process demands active involvement from both design and development teams, who must create interface elements, map user interactions, and repeatedly revise designs based on stakeholder feedback. This process consumes considerable time and effort, particularly during the early development stages.

Prototyping often requires substantial time and energy investment, particularly during the initial stages of development. Development teams need to allocate extra time to design, assemble, and validate the prototype with various stakeholders before the main system is built. If this process is not efficiently planned, the time intended for core system development may be drained by prototyping alone, potentially delaying the entire project timeline—especially in projects with tight deadlines.

Advancements in software engineering have led to automated prototyping approaches, offering significant advantages over traditional manual methods, particularly in terms of time and cost efficiency. According to Li et al. (2010), using automatically generated prototypes can drastically accelerate the design process, as interface elements and layout structures can be created instantly by the system without complex manual intervention. This results in high efficiency, both in terms of development time and the number of human resources required.

The adoption of automation technology in prototyping not only enhances technical efficiency but also fosters dynamic and productive collaboration. This approach is particularly relevant for addressing the fast-paced, accurate, and adaptive demands of modern software development. Deep Learning, a major branch of Artificial Intelligence, leverages deep neural networks to automatically extract features and patterns from raw data without manual engineering. It has significantly advanced fields like computer vision, natural language processing, and speech recognition by learning complex data representations. In computer vision, Deep Learning especially through Convolutional Neural Networks (CNNs) effectively identifies visual patterns and object features. Among object detection models, YOLO (You Only Look Once) stands out for its single-stage detection method, which performs object localization and classification in one pass. This makes YOLO highly efficient and suitable for real-time applications, including surveillance systems, autonomous vehicles, and medical imaging.

Deep Learning models like YOLO rely heavily on high-quality, diverse datasets to perform well, especially in object detection tasks. These models are essential for identifying and locating interface elements such as buttons or input fields in hand-drawn wireframes. By using object detection, systems can convert sketches into digital prototypes automatically, accelerating UI development. However, success depends on the model's ability to handle visual inconsistencies in sketches, which requires well-designed, representative training data. Overall, this approach enhances design efficiency and supports automation in interface prototyping.

Object detection is one of the key achievements in the development of computer vision, thanks to the rapid advancements of Deep Learning technology, which enables systems to efficiently recognize complex visual patterns. As explained by Zhao et al. (2018), the success of object detection heavily depends on the ability of Deep Learning models to handle variations in scale, rotation, and illumination within an image. The main goal of object detection is to determine whether an object of a certain class is present in an image, and if so, to return its spatial information in the form of a bounding box, which indicates the object's position and coverage in two-dimensional coordinates (Everingham et al., 2010; Russakovsky et al., 2015). In addition to object classification, object detection also requires high spatial localization precision to ensure the validity of the output data.

The application of Deep Learning-based object detectors, particularly YOLO, provides an effective and efficient solution for automating the identification of interface elements from hand-drawn wireframes. This approach not only increases speed and accuracy in the development of user interface prototypes but also opens up significant opportunities for the development of adaptive and scalable Sketch-to-Code systems.

Deep Learning technology, especially through object detection approaches such as YOLO (You Only Look Once), offers great potential in the task of detecting interface elements in manually drawn wireframe sketches. Models like YOLO are designed to perform object detection quickly and efficiently, making them ideal for applications that require real-time image processing, including automated user interface design systems. However, the optimal performance of such Deep Learning models greatly depends on the availability of adequate and representative datasets. Datasets rich in variation in shape, size, and layout of wireframe elements are crucial to enable the model to learn and recognize visual patterns with high accuracy. Without relevant and high-quality training data, the model will struggle to generalize to new sketches that are not part of the training data.

Considering these various aspects, integrating Deep Learning models such as YOLO with sketch-based wireframe approaches presents a significant opportunity for automating user interface creation. This not only enhances design process efficiency but also facilitates technology adoption in resource-limited environments, while encouraging the development of faster, more cost-effective, and user-adaptive interface development systems.

For example, the RICO dataset (Deka et al., 2017), one of the largest collections of Android app user interfaces, includes over 9,300 apps with thousands of screen views and metadata. This dataset provides visual information in the form of screenshots, component hierarchy structures, text, and interactive properties of user interface elements. Although highly useful for analyzing modern user interfaces and studying application design, RICO does not include wireframe sketches or data representing the early design process based on hand-drawn images. This results in a significant data gap for research aimed at automating sketch-to-code conversion.

Therefore, the development of a more comprehensive, open, and standardized wireframe sketch dataset becomes an important and urgent step to advance the field of automated interface design. Such initiatives will not only strengthen research foundations but also enable the development of more accurate, reliable models that are ready for real-world application.

Unlike conventional RNNs that struggle to retain contextual information over long time sequences, LSTM is equipped with more complex memory control mechanisms, such as forget gates, input gates, and output gates. These mechanisms allow LSTM to dynamically choose which information to store, update, or remove from the cell memory. This feature is crucial in many tasks that depend on sequential dependencies, such as natural language modeling, handwriting recognition, and particularly in the context of this study, generating web page structures from wireframe sketches.

LSTM and its variant BiLSTM play a key role in building efficient Sketch-to-Code systems by processing sequential data, effectively bridging visual designs and functional web

code. This research aims to simplify web mobile UI prototyping using Deep Learning, addressing the inefficiencies of manual prototyping, which is time-consuming and costly.

To solve this, the proposed system, Generate One Page Application, automatically converts hand-drawn wireframes into HTML and CSS prototypes. These outputs serve both as realistic visuals and as starting frameworks for further development. By automating UI element detection and code generation, the system speeds up the prototyping process, enhances collaboration between designers and developers, and reduces repetitive manual work.

Thus, this research is expected to provide a real contribution in simplifying and accelerating the process of designing mobile web application interfaces, as well as supporting the development of software that is more efficient, cost-effective, and easily accessible to various developer groups.

METHOD

Research Methodology

This research uses a Deep Learning-based software engineering methodology to automate the prototyping of mobile web interfaces from hand-drawn wireframe sketches. The methodology consists of three main stages: system design, implementation, and evaluation.

In the design phase, the dataset of wireframe sketches is prepared and annotated using tools like Roboflow to label interface elements such as buttons, text inputs, and images. The labeled data is used to train an object detection model based on YOLOv11. The detection output is then converted into a Domain Specific Language (DSL) format, which serves as an intermediate representation before generating HTML and CSS code.

The implementation phase focuses on developing a web-based user interface that allows users to upload sketch images. These images are processed through a system pipeline—including normalization, object detection, DSL conversion, and automatic HTML/CSS generation. The final output includes a prototype preview and source code.

In the evaluation phase, the system is tested for functionality, object detection accuracy, and prototype quality. User testing is conducted to assess usability and alignment with expected designs. The results are used to evaluate the system's effectiveness and future development potential.

This methodology aims to ensure accurate interface element detection and generate usable HTML/CSS prototypes for mobile web application development.

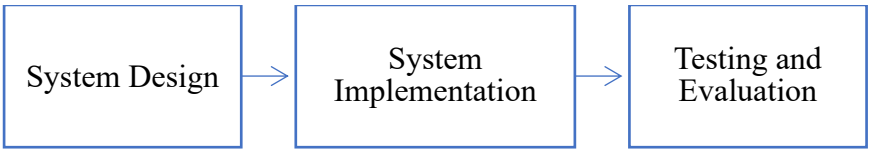

Figure 1. Web Application Development Process

System Design

The system design stage provides an overview of the steps involved in creating the proposed system, as shown in Figure 1. In this study, a system design is carried out to automatically generate HTML code that includes 1 page or more, based on the number of Wireframe sketches entered and can be used as a prototype or framework in the application development process. This study applies the YOLOv11 object detection method to identify UI elements from Wireframe sketches, then processed into DSL using potential parent and children searches for each element using the coordinate points and dimensions of the detected objects using YOLOv11, then converting the DSL hierarchy into HTML code that has a position and size close to the input Wireframe sketch.

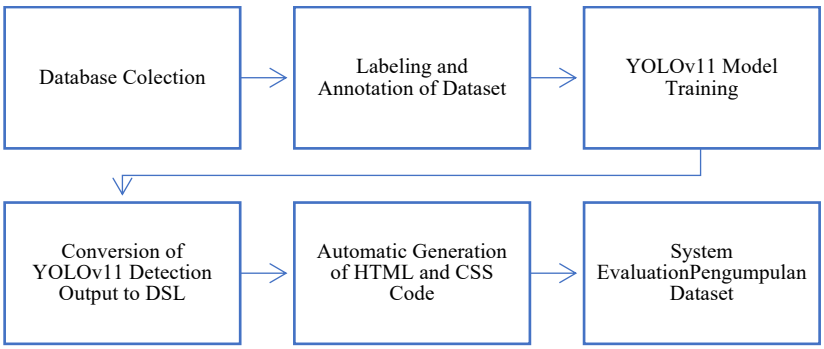

Figure 2. One Page Application Generate System Design

System Implementation

System implementation is a crucial stage in realizing the design into a functional prototype. The primary objective of this stage is to integrate all designed components, from image input and interface element detection to HTML and CSS code generation, into a single, automated and efficient workflow. This system is implemented to accept wireframe sketch images as input and generate a ready-to-use web interface prototype end-to-end.

Figure 2 illustrates the system implementation flowchart, outlining the main steps for transforming a wireframe sketch into a web interface previewable through HTML and CSS. These implementation stages are designed to be automated and systematic, from the input process to the final output, a preview of the interface. Each system component is built to communicate with each other through a uniform data format, such as JSON for detection results and DSL code for abstracting UI structures. With this approach, the system is expected to not only help accelerate the interface design process but also provide flexibility for users to develop the generated code.

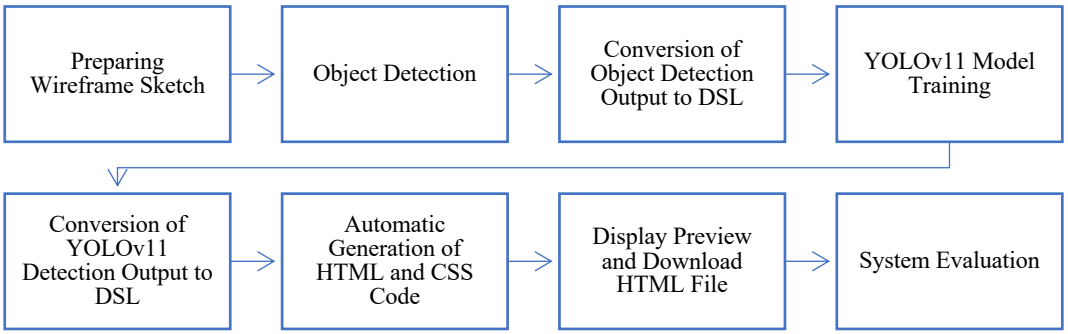

Figure 3. System Implementation

RESULT AND DISCUSSION

UI Element Detection Results

To produce an optimal object detection model, this research conducted a series of experiments using the YOLOv11 architecture, specifically the lightweight variant YOLO11n.pt. After testing various configurations, the best training results were obtained using a set of adjusted parameters. The model was trained for 50 epochs with an image size of 640 × 640 pixels, using a batch size of 5 and enabling cache storage on disk for efficient dataset loading.

The initial learning rate (lr0) was set at 0.01 with a final learning rate (lrf) also at 0.01, and the optimizer was chosen automatically. Data augmentation was applied with parameters mosaic=1.0 and fliplr=0.5 to enhance model generalization. Additionally, the model was trained using a mask ratio of 4 and the overlap_mask feature enabled (overlap_mask=true), which helps in detecting UI elements with overlapping contours, such as adjacent text boxes

or buttons. A momentum value of 0.937 and weight decay of 0.0005 were applied to maintain stability and regularization during training.

Validation was performed at each epoch with an IoU threshold set at 0.7, and model performance was measured using the mAP metric. The best results showed that the model achieved an mAP50 of 91.7% and an mAP50–95 of 77.4%, indicating the model's ability to accurately recognize various user interface elements from wireframe sketches. The use of this training strategy demonstrates that selecting appropriate parameters, along with suitable augmentation and optimization, plays a significant role in improving the performance of Deep Learning-based object detection models for UI element detection tasks from wireframe images. Table 1 below shows detection performance per UI element class:

Table 1. Detection Performance per UI Class

Class	Box(P)	Box(R)	mAP50	mAP50-95
Button	0.996	0.957	0.985	0.897
Carousel	1	0.9	0.988	0.91
Check Box	0.895	0.923	0.965	0.756
Container	0.892	0.818	0.748	0.742
Heading	0.869	0.839	0.822	0.65
Image	0.973	0.973	0.989	0.921
Label	0.991	0.959	0.987	0.748
Paragraph	0.925	0.929	0.975	0.796
Radio Button	0.798	0.78	0.812	0.627
Select	0.855	1	0.989	0.862
Textarea	0.984	1	0.995	0.941
Textbox	0.821	1	0.972	0.808

Table 1 presents the evaluation results of the object detection model's performance across various UI element classes, such as Button, Image, Textbox, and others. The evaluation uses several common metrics in object detection, namely:

1. **Box(P) (Precision)** — measures the model's accuracy in detecting correct elements (the higher, the fewer false positive detections).
2. **Box(R) (Recall)** — measures the model's ability to find all correct elements (the higher, the fewer missed elements).
3. **mAP50 (mean Average Precision at IoU 0.5)** — the average precision value for all classes with an Intersection over Union (IoU) threshold of 0.5.
4. **mAP50-95** — the average precision across all classes with IoU ranging from 0.5 to 0.95 (a stricter step, indicating the overall model capability based on the COCO evaluation standard).

DSL to HTML and CSS Conversion Results

After the UI elements are successfully detected, the system forms a hierarchical structure in DSL format, which serves as a bridge between visual detection and code conversion. The generated DSL contains information about the position, size, and type of each element, which is then converted using a template-based approach into HTML and CSS files using the Bootstrap 5 framework.

The generated prototype in the form of an HTML file can be directly rendered in a browser and represents the layout according to the given sketch. Elements such as buttons, textboxes, and images are well arranged within the Bootstrap 12-column grid system. Figure 4 shows a comparison between the wireframe sketch and the resulting HTML output:

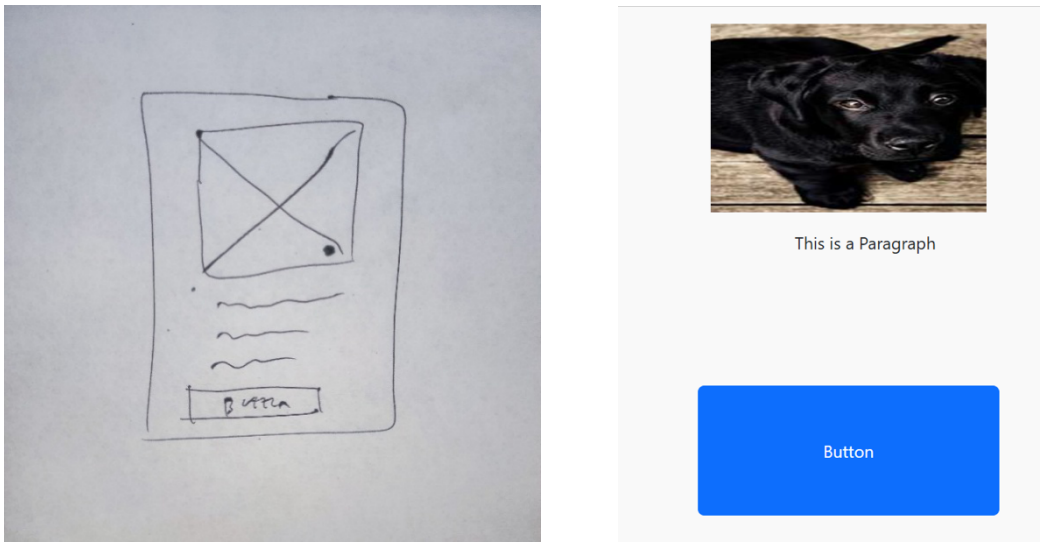

Figure 4. Wireframe Sketch and Automatic code generation results from wireframe sketch

Visual and Code Evaluation

The evaluation focused on two main aspects: the visual accuracy of the layout and the quality of the HTML and CSS code. The test involved 33 respondents, including students and software development professionals. Each respondent was asked to rate the prototype based on four indicators:

1. **Similarity of the generated HTML to the original wireframe sketch**

   The third figure shows the results of the 33 respondents' evaluation of the similarity of the system-generated HTML to the original wireframe sketch. Ratings were given on a scale of 1 to 5, with 1 representing very similar and 5 representing very similar. The survey results indicate that most respondents felt the resulting appearance adequately reflected the original sketch, but not perfectly. Eleven respondents (33.3%) gave a score of 4, indicating that the majority considered the system's output to be quite similar to the reference wireframe sketch.

   However, only five respondents (15.2%) gave a score of 5, indicating that a small percentage of users actually felt the final appearance closely matched the sketch. On the other hand, 7 respondents (21.2%) gave a score of 2, and the same number gave a score of 3, indicating a perception that some results may not fully resemble the sketch or in certain details. 3 respondents (9.1%) even gave a score of 1, indicating high dissatisfaction with the match between the sketch and the resulting HTML output.

   Overall, the distribution of scores was fairly even but tended toward the upper middle, with a score of 4 being the dominant score. This indicates that the system performed reasonably well in representing the visual appearance of the wireframe in HTML, but further improvements in visual fidelity are needed to achieve more consistent results and satisfy all users.

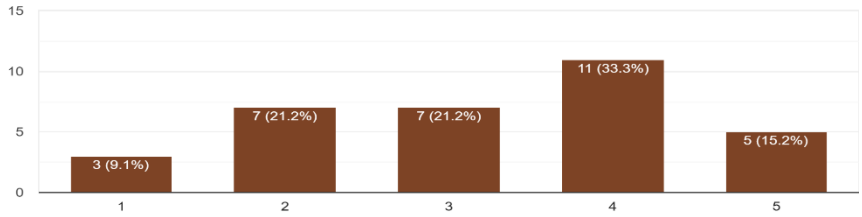

Figure 5. The generated HTML output looks similar to the original wireframe sketch.

2. The generated HTML code is neat and easy to understand.

A total of 33 respondents assessed the quality of the HTML code structure generated by the system, focusing on neatness and ease of understanding. Ratings were given on a scale of 1 to 5, where 1 means very messy and confusing, and 5 means very neat and easy to understand. The evaluation results indicate that the majority of respondents had a fairly positive perception of the quality of the generated code. The highest scores were given to 3 and 4, by 11 respondents (33.3%), respectively. This indicates that more than half of the respondents considered the code to be fairly good and fairly neat, although not yet completely perfect.

Furthermore, 8 respondents (24.2%) gave a score of 5, indicating that nearly a quarter of respondents were very satisfied with the structure of the generated HTML code—both in terms of neatness and readability. On the other hand, only a few respondents gave low scores: 2 respondents (6.1%) gave a score of 2, and only 1 respondent (3%) gave a score of 1. This indicates a very low level of dissatisfaction with the code structure.

Overall, the distribution of scores skews toward the upper middle, indicating that the system successfully generates HTML code that is clean and understandable to most users. However, with scores of 3 and 4 dominating, there is still room for improvement, particularly in terms of writing consistency and code readability, to ensure a greater number of users are fully satisfied.

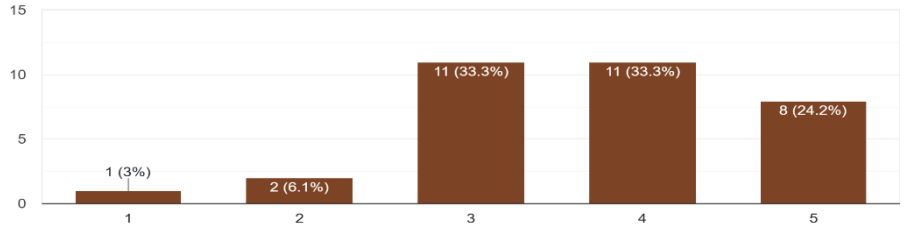

Figure 6. The generated HTML code diagram is neat and easy to understand.

3. The generated code is ready for further development.

A total of 33 respondents assessed the readiness of the system-generated HTML code for further development. The scale used was 1 to 5, with 1 indicating very unready and 5 indicating very ready for development. The evaluation results showed that the majority of respondents considered the generated code to be of high quality and worthy of development. Thirteen respondents (39.4%) gave a score of 4, and 11 respondents (33.3%) gave a score of 5. This means that more than 72% of respondents felt that the generated code was ready or very ready for use in further development.

Meanwhile, 5 respondents (15.2%) gave a score of 3, indicating that the code was considered quite good but may still require some improvements before it can be optimally developed. On the other hand, low scores were relatively small: only 4 respondents (12.1%) gave a score of 2, and none gave a score of 1 (0%), indicating a very poor assessment of code readiness.

Overall, the score distribution shows a positive trend. The system is capable of producing code that is not only clean and easy to understand, but also sufficiently ready to be integrated, modified, or further developed, as required for more complex software development.

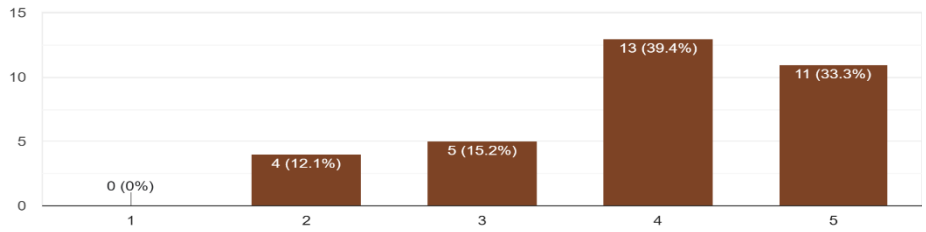

Figure 7. The generated code is ready for further development.

4. I found the automation of converting wireframes to HTML helpful.

A total of 33 respondents rated the extent to which they found the automation feature from wireframes to HTML helpful. The ratings were based on a scale of 1 to 5, with 1 indicating not helpful at all and 5 indicating very helpful. The survey results showed that the majority of respondents gave this feature a very positive rating. Fourteen respondents (42.4%) gave a score of 4, and 11 respondents (33.3%) gave a score of 5, indicating that over 75% of respondents felt a real benefit from this automation.

Six respondents (18.2%) gave a score of 3, indicating that while they found it helpful, they may still encounter limitations or need further improvements in the system. Meanwhile, only one respondent each (3%) gave a score of 1 and 2, indicating very low and insignificant levels of dissatisfaction.

Overall, the distribution of scores shows a very positive and supportive response, indicating that the wireframe-to-HTML automation feature has high utility and is perceived as beneficial by the majority of users in the interface development process. This is an important indicator that the system has successfully met user needs in terms of efficiency and ease of UI design.

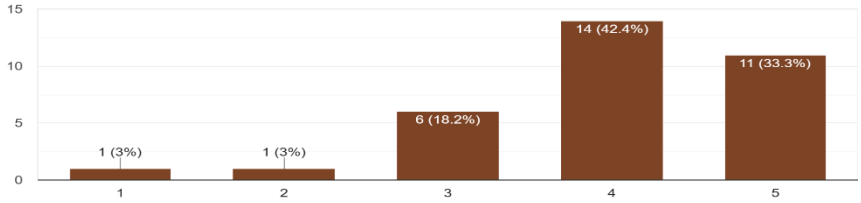

Figure 8. I found it helpful to automate the conversion of wireframes into HTML code.

System Limitations

Several system limitations were discovered during the evaluation:

a. The system does not yet support element content text detection, only UI structure.
b. It does not support real-time detection.
c. It can only process images as input.
d. The responsiveness of the HTML output is limited to mobile dimensions; it does not yet support tablets or desktops.

Results Summary

The system achieved an mAP50 of 91.7%, indicating high detection performance. HTML and CSS prototypes were successfully generated and can be displayed in a browser. User evaluations showed a score of 3.7 on several evaluation aspects. The system can be used as a framework to accelerate the initial interface design process in application development.

CONCLUSION

This research has successfully developed an automated system for creating mobile web app prototypes from wireframe sketches using Deep Learning technology, specifically the YOLOv11 algorithm, and a Domain Specific Language (DSL)-based approach. This system is capable of detecting UI elements from wireframe sketches, organizing them into a DSL structure, and automatically converting them into HTML and Bootstrap CSS code.

From the results of the research implementation and evaluation, the following conclusions can be drawn.

1. The YOLOv11-based object detection system, fine-tuned on a UI-specific dataset, achieved an mAP50 score of 91.7%, demonstrating high accuracy in detecting UI elements such as buttons, text boxes, images, and others.
2. The DSL structure was successfully constructed as an effective intermediary between the element detection results and the code conversion system, enabling a systematically interpretable hierarchical representation of the visual layout.
3. The converted HTML and Bootstrap CSS code from this system was able to produce a prototype that matched the initial sketch, both in terms of structure and layout. The prototype can be directly displayed in a browser and used as a basis for further development.
4. Based on an evaluation of 30 respondents, the system received an average score of 3.2 for visual accuracy and 3.8 for HTML code quality, indicating that the system was well-received and considered useful in the context of web-based application development.
5. This system demonstrates significant potential for reducing time and costs in the early stages of software development, particularly in the user interface design process.

REFERENCES

Abdelhamid, A. A., Alotaibi, S. R., & Mousa, A. (2020). *Deep Learning* -based prototyping of android gui from hand-drawn mockups. *IET Software*, *14*(7), 816–824. https://doi.org/10.1049/iet-sen.2019.0378

Adarsh, S., Harish, D., Balaganapathy, K., Venkatachalapaty, R., Abishiek, E., & Nagarajan, M. (2017). Improved Software Quality and Design Standards Based on Customer Preferences by Applying Evolutionary Prototyping Software Development Model. *International Journal of New Technology and Research*, *3*(5). www.ijntr.org

Adefris, B. B., Habtie, A. B., & Taye, Y. G. (2022). Automatic Code Generation From Low Fidelity Graphical User Interface Sketches Using *Deep Learning . 2022 International Conference on Information and Communication Technology for Development for Africa, ICT4DA 2022*, 47–52. https://doi.org/10.1109/ICT4DA56482.2022.9971204

Aleksić, D., Savić, D., Vlajić, S., Rodrigues Da Silva, A., Stanojević, V., Antović, I., & Milić, M. (2016). *Generate User Interface Using Xtext Framework*.

Beltramelli, T. (2017). *pix2code: Generating Code from a Graphical User Interface Screenshot*. http://arxiv.org/abs/1705.07962

Carlos, J., Ferreira, S., & Ferreira, H. S. (2019). *Live Web Prototypes from Hand-Drawn Mockups*.

De Macedo, G. T., Fontão, A. de L., & Gadelha, B. F. (2023). Prototyping in Software Quality Assurance: A Survey With Software Practitioners. *Proceedings of the XXI Brazilian Symposium on Software Quality*. https://doi.org/10.1145/3571473.3571477

Dehaerne, E., Dey, B., Halder, S., De Gendt, S., & Meert, W. (2022). Code Generation Using Machine Learning: A Systematic Review. Dalam *IEEE Access* (Vol. 10, hlm. 82434–82455). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2022.3196347

Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., Nichols, J., & Kumar, R. (2017). Rico: A mobile app *dataset* for building data-driven design applications. *UIST*

2017 - Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, 845–854. https://doi.org/10.1145/3126594.3126651

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2010). The pascal visual *object* classes (VOC) challenge. *International Journal of Computer Vision*, *88*(2), 303–338. https://doi.org/10.1007/s11263-009-0275-4

Iglesias, M. J. F. (2023). Prototyping. Dalam *Design Thinking for Engineering* (hlm. 91–109). https://doi.org/10.1049/PBME024E_ch6

Kolthoff, K. (2019). Automatic Generation of Graphical User Interface Prototypes from Unrestricted Natural Language Requirements. *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 1234–1237. https://doi.org/10.1109/ASE.2019.00148

Konat, G., Steindorfer, M. J., Erdweg, S., & Visser, E. (2018). PIE: A Domain-Specific Language for Interactive Software Development Pipelines. *Art, Science, and Engineering of Programming*, *2*(3). https://doi.org/10.22152/programming-journal.org/2018/2/9

Le, T. H. M., Chen, H., & Babar, M. A. (2020). *Deep Learning* for Source Code Modeling and Generation: Models, Applications, and Challenges. *ACM Computing Surveys*, *53*(3). https://doi.org/10.1145/3383458

Li, X., Liu, Z., Schäf, M., & Yin, L. (2010). AutoPA: Automatic prototyping from requirements. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6415 LNCS*(PART 1), 609–624. https://doi.org/10.1007/978-3-642-16558-0_49

Liang, J. (2024). A review of the development of *YOLO object detection* algorithm. *Applied and Computational Engineering*, *71*(1), 39–46. https://doi.org/10.54254/2755-2721/71/20241642

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). *Focal Loss for Dense Object Detection* . http://arxiv.org/abs/1708.02002

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2015). *SSD: Single Shot MultiBox Detector*. https://doi.org/10.1007/978-3-319-46448-0_2

Mohian, S., & Csallner, C. (2020). Doodle2App: Native app code by freehand UI sketching. *Proceedings - 2020 IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2020*, 81–84. https://doi.org/10.1145/3387905.3388607

Olmsted-Hawala, E. L., Romano, J. C., & Murphy, E. D. (2009). The use of paper-prototyping in a low-fidelity usability study. *2009 IEEE International Professional Communication Conference*, 1–11. https://doi.org/10.1109/IPCC.2009.5208693

Rahman, M. M., Watanobe, Y., & Nakamura, K. (2021). A bidirectional *LSTM* language model for code evaluation and repair. *Symmetry*, *13*(2), 1–15. https://doi.org/10.3390/sym13020247

Ren, S., He, K., Girshick, R., & Sun, J. (2015). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. http://arxiv.org/abs/1506.01497

Robinson, A. (2019). *Sketch2code: Generating a website from a paper mockup*. http://arxiv.org/abs/1905.13750

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Sonje, S., Dave, H., Pardeshi, J., & Chaudhari, S. (2022). draw2code: AI based Auto Web Page Generation from Hand-drawn Page Mock-up. *2022 IEEE 7th International conference for Convergence in Technology, I2CT 2022*. https://doi.org/10.1109/I2CT54291.2022.9824521

Uzayr, S. bin. (2022). Prototyping. Dalam *Mastering Sketch*. http://dx.doi.org/10.1201/9781003261575-6

Web UI Sketch Dataset . (t.t.). Diambil 24 April 2025, dari https://www.kaggle.com/*dataset*s/alaalsanea/sketch-web-ui-*dataset*

Yang, Y., Li, X., Ke, W., & Liu, Z. (2020). Automated Prototype Generation From Formal Requirements Model. *IEEE Transactions on Reliability*, *69*(2), 632–656. https://doi.org/10.1109/TR.2019.2934348

Zhao, Z.-Q., Zheng, P., Xu, S., & Wu, X. (2018). *Object Detection with Deep Learning : A Review*. http://arxiv.org/abs/1807.05511

Zou, Z., Chen, K., Shi, Z., Guo, Y., & Ye, J. (2023). *Object Detection* in 20 Years: A Survey. *Proceedings of the IEEE*, *111*(3), 257–276. https://doi.org/10.1109/JPROC.2023.3238524