Vrije Universiteit Amsterdam

Bachelor Thesis

Automating Front-End Development: The Impact of Large Language Models in UI Component Generation

Author: Zefan Morsen (2691045)

1st supervisor: Dr. Sieuwert van Otterloo
2nd reader: supervisor name

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science

June 28, 2025

Contents

1

Introduction

The software development field is continuously changing, due to the rapid improvement in generative artificial intelligence. Traditional development workflows are changing due to the rapid innovations of large language models (LLMs) like GPT-4 and Gemini 2.5 via the automation of code generation from natural language prompts (1). This thesis explores the practical use of these large language models into front-end web development. This way we can analyze how the AI operates in a domain where efficiency, complexity, and user experience is top priority. This research aims to offer a clear understanding of the benefits and challenges that comes to today's developers by using large language models (LLMs) for the generation of common user interface (UI) components.

1.1 Background and Context

In the past, software development has been framed as a tool that is only used by entitled people because of its high complexity and high entry barriers that required much time and special technical skills to use. In response to these challenges, no-code and low-code development platforms rose and they were designed to make software creation accessible by using graphical interfaces and reusable components to abstract the underlying code (2) (3). Research on platforms such as Triggre has shown that this method reduces the barrier and complexity for users, enabling them to easily build functional applications (4) (5).

Generative AI really is the next step in the evolution process of automation and simplification. A direct route of turning an idea into a creation is made possible due to the fact that large language models (LLMs) can interpret human natural language and turn that into functional code. Even though this seems new, this isn't really a new concept. Early experiments like Sketch2Code showed that AI could convert visual designs into code (6).

Today, the potential of the new AI models are being realized on a scale that is never seen before.

This research will specifically focus on front-end development using React, which was ranked by the development community as the most popular and widely-used framework in the past years (7). It has a component based architecture which makes an ideal environment to test the Ai in the creation of individual user interface components.

1.2 Investigated Problem, Motivation, and Scientific Contribution

The main motivation for including LLMs in front-end development is their capability to dramatically enhance accessibility and efficiency (1). Any developer would ideally be able to speed up UI implementation, reduce repetitive tasks, and pay more attention to architecture and design challenges. However, there is a huge gap between this promise and the practical reality of using current AI tools.

The main problem is that the output of LLMs are unreliable and inconsistent. Research was done in this area and this is what has been found:

- **Accuracy and Correctness:** A comprehensive benchmark, "Design2Code," discovered that even state-of-the-art multimodal LLMs struggle with remembering visual elements with accuracy and structuring complex layouts correctly (1).
- **Need for Human Intervention:** While AI can speed up the initial design of components, comparisons of AI-assisted development with traditional ones show that advanced designs will require human intervention by a developer all the time to check the correctness, adherence to best practices, and ultimately usability (8).
- **Quality and Maintainability:** Beyond functionality, there are still doubts about the quality and long-term maintainability of code composed with AI. The code may execute, but it may not prove efficient, readable, or easily update-able.

In response to these challenges, organized prompting techniques have emerged as a central way of optimizing and improving the output. Research shows that adjusting the prompt requirements of LLMs improves their performance substantially in practical applications (9) (10).

This brings the central concern investigated within this thesis: we have no strict, empirical evidence that shows the real-world impact LLMs has applied on typical, real-world front-end development projects within mainstream frameworks like React.

The science of this research lies in bridging this void with a systematic evaluation. In this thesis, we will generate quantitative and qualitative information regarding time efficiency and quality of the ai generated components, providing a clear view of how far we've come with LLMs today and generating insights for both academic and industrial communities.

1.3 Research Question

This study's fundamental aim is to evaluate and compare UI component generation with LLM-assisted methods and traditional manual coding practices within the React framework. The comparison will essentially be based on time efficiency, quality, and maintainability of the components.

This aim can be outlined as the following main research question:

How does the integration of Large Language Models (LLMs) affect the speed and quality of creating UI components in React compared with traditional front-end development strategies?

Sub-division of Research Question

In an effort to thoroughly address the main research question, the research itself is guided by two central sub-questions that address the performance of the resulting code as well as how it is generated:

1. What are the time savings of using LLMs for ui component generation compared with manual coding?
2. What is the quality and maintainability of UI components written by AI compared with manually written components?

2

Related Work

This chapter provides a review of published literature relevant to the interaction of artificial intelligence, no-code/low-code systems, and front-end software development. The goal is to situate this research within a broader academic and technical framework, highlight key themes and advances, and therefore clarify the specific void this thesis attempts to fill.

Software Development: From No-Code to AI

This idea of opening software development to non-programmers is not a new idea. The rise of no-code and low-code systems was a huge step toward this direction, with an aim of "democratizing software creation" (2). These platforms employ visual interfaces and pre-built, reusable components, which eliminate the classical code complexity (3). This has shown to reduce the barrier to entry, reduce development expenses, and speed up building of commercial applications (11).

The next logical step of this process is the rise of powerful LLMs. AI-based modelling and automation can decrease development even more by translating natural language descriptions directly into usable software components. Studies about introducing no-code systems with tools like ChatGPT have resulted in substantial increases in development speed and customization capabilities (10)(5). These studies suggest that AI can easily automate workflow as well as data model construction, bridging the gap between requirements and technical implementation (4)(12).

AI-Assisted Code Generation and Evaluation

While AI integration into no-code systems is about code abstraction, a second huge area of research considers direct code production by LLMs. Initial attempts, such as Sketch2Code

(6), showed how UI code can be generated from visual specifications. In contemporary LLMs, this capability has improved immensely.

Their performance, however, remains a subject of ongoing research. Si et al. (2024) evaluated multimodal LLMs using their "Design2Code" benchmark and observed significant struggle with model output at correctly understanding visual designs and generating correspondingly suitable layout structures. This is a fundamental gap between understanding a design and generating perfect functional code.

Additional comparative research has analyzed the performance of other AI-driven development tools. (13)Siam compared ChatGPT, Gemini, and GitHub Copilot and determined considerable differences between their usability and accuracy of code. Correspondingly, Hosseini and Rydén(8) directly compared classical front-end development with front-end development assisted by AI. From their findings, even if AI can enormously speed up the implementation of UIs, manual human effort remains almost always unavoidable in terms of producing the final code correct, usable, and conforming to project requirements. This indicates the fact that LLMs at this point mainly function as a "tool" of programmers rather than as a "sole programmer."

The Importance of Prompt Engineering

A returning theme within the writings is the important role of prompt engineering. The output of an LLM depends directly and heavily on prompt input quality. Work by Martins, Branco, and Mamede and Castelberg and Flury(10)(9) places greatest emphasis on use of structured, iterative, and detailed prompting as a key to unlocking maximum capability of an LLM within development applications. This is repeated within the case applications by de Schiffart (4)and Şahin(5), with considerable effort having been dedicated within those applications to prompt engineering capable of the correct AI interpretation to generate desired components and workflow production. This would suggest the interaction with an LLM as a skill itself and one of greatest importance within any successful application.

Identifying the Research Gap

The writings confirm that LLMs are a transformative force within software development, both on an abstraction level of no-code systems and in direct code development. There exists a specific void, however, which this thesis attempts to fill. While many papers have examined AI within the scope of a no-code framework (12)(5) or have benchmarked AI

on typical coding tasks at large (13), comparatively far less exists on a focused, empirical comparison as a way of building common UI components within a mainstream platform such as React.

Further, papers like Hosseini and Rydén(8) state quality, yet a more detailed study is needed. One that combines quantitative estimates of development time with developer-side qualitative comparisons of code maintainability and quality. This thesis improves on existing research by developing strictly and systematically a standardized set of UI components manually and with the help of LLMs, and comparing results critically on development speed as well as quality as a practitioner would assess.

3

Methodology

In order to provide answers to research questions, this research applied a comparative, structured research approach designed to measure differences between developing React UI components with traditional manual construction and with LLMs. Research was carried out by setting up a set of standard components, having them built with two alternate approaches, and comparing outcomes based on predetermined evaluation criteria.

The approach was specifically designed with obtaining data with which to provide answers to the following two sub-questions:

- What are the time savings of using LLMs for UI component generation compared with manual coding?
- What is the quality and maintainability of UI components written by AI compared with manually written components?

3.1 Research Approach

Overall approach was a multi-step process ranging from scope specification of the experiment all the way through final comparative analysis.

UI Components Identification and Specification

To create a standardized basis for comparison, a set of five typical UI components was selected. The selection was based on exhaustive online sources cataloging common UI components used in modern day web design (14)(15). These components included:

- Button (submit)

- Text Input (Search Bar)
- Form (Login Form)
- Table (salary payout)
- Navigation bar

A detailed specification sheet was created where requirements for each component were listed (see Appendix A). This document included exact functional as well as styling specifications to ensure that both the manual developers and the AI prompts would all strive toward an identical, measurable output. All the components needed to be built in React and styled with Tailwind CSS.

3.2 Component Implementation

The two methods that were used in building the five components were AI generation and manual coding.

Manual Implementation

Five junior-front-end developers were approached to participate in this study. These five developers were handed the specification sheet with a request to implement each of these five components manually. These developers were instructed to use a stopwatch during their active development time of the components. They had to time how long it took in order to create a single component. The time included all relevant activities, such as coding and even searching on the internet for documentation or a solution. In the end they all had 5 different times for the 5 different components.

AI-assisted Implementation

The same five components have been built using two of the highest-ranking LLMs: Google's Gemini (model 2.5 Pro) and OpenAI's ChatGPT (model 4.1). These implementations have utilized two different prompting strategies:

Individual Detailed Prompts: These individual prompts, designed per component, were finely tuned based on established prompt patterns (16) and contained all of the function and styling requirements of the specification sheet. An example of an individual detailed prompt for the Button component is as follows:

"Create a standard form submission button as a React component, styled with Tailwind CSS. Functional Requirements: The button must have type='submit' and its visible text must be 'Submit'. Styling Requirements: The button must, in default state, have a blue background color with white-colored text in the standard sans-serif font. On hover, the background color should change to baby blue. The button must have rounded corners on all sides."

Single Combined Prompt: A single combined prompt was created that contained all of the specifications of all five UI components, requesting the LLM to generate all of them in one response.

The elapsed time of AI-assisted components was computed from when the prompt was initially written to when the last functional code was received from the LLM.

3.3 Data Analysis

The data collected from this implementation phase was directly applied toward giving answers to research sub-questions.

Time Savings

A comparative investigation was conducted with a quantitative approach. The average of each of the five components from the manual developers development time was directly compared with each of the two AI prompt generation processes' corresponding time with each of the two language models.

Quality and Maintainability

A two-part analysis was carried out. During the first part, both prompt types of the AI-generated code are being checked of there functional completeness with respect to specification sheet. In the second part, we performed a qualitative assessment on developer view of code. The five participating developers were shown the final, functional UI components generated by the LLM and asked to qualitatively compare this version with their own. This developer preference served as a proxy for perceived code quality, readability, and maintainability.

4

Results

This chapter presents results of the comparative study. The results are structured according to the two research sub-questions, initially on the quantitative time savings, and then on the qualitative assessment of code quality and maintainability.

4.1 Time Savings

The first sub-question aimed at identifying time saved when using LLMs for creating UI components as compared to manual development. Time taken on development was captured on all techniques: manual coding as well as AI-generation using ChatGPT and Gemini. Average time taken on manual development and time taken on AI techniques can be seen from Table 4.1.

Table 4.1: Comparison of Development Times (in seconds)

UI Component	**Avg. Manual Dev. Time**	**AI Time (ChatGPT - Detailed)**	**AI Time (Gemini - Detailed)**
Button	298	123	121
Text Input	498	137	123
Form	745	232	231
Table	883	295	248
Navbar	1529	450	285
Total Individual Time	**3953**	**1237**	**1008**

Further, the single combined prompt approach was also evaluated, which consumed a combined time of 839 seconds by ChatGPT and 753 seconds by Gemini.

The following figures graphically show data from Table 4.1, providing a clear comparison of the development time under the different methods.

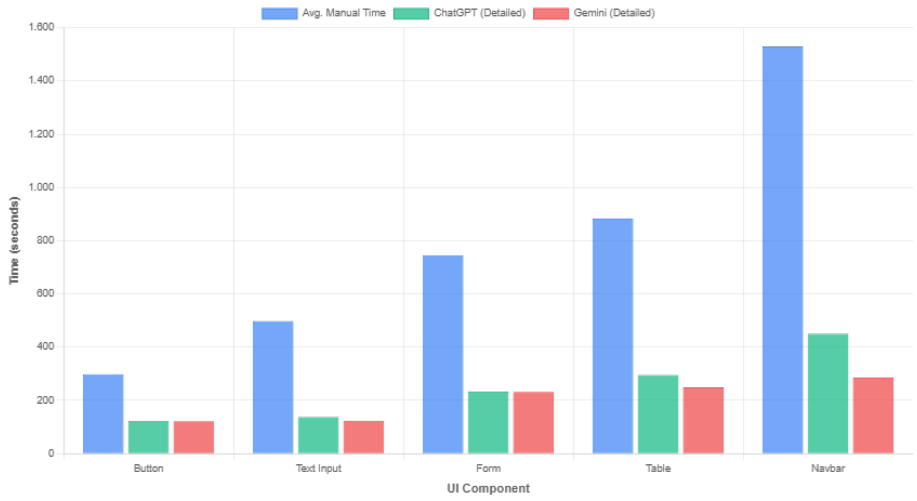

Figure 4.1: Development Time per Component: Manual vs. AI (Individual Prompts)

It is clear from data that all development with help of AI took incredibly less time with respect to manual development. As seen in Figure 4.1, for every individual component, Gemini and ChatGPT took substantially less time compared with each individual manual developer. In cumulative time per prompt basis, ChatGPT (1237 seconds) and Gemini (1008 seconds) took approximately 69% and 74% less time compared with manual code writing (3953 seconds).

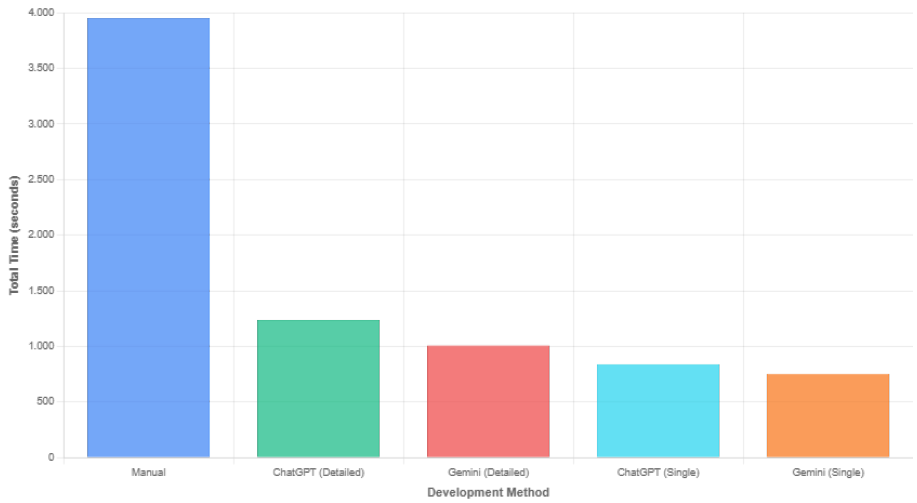

Figure 4.2: Total Development Time: Manual vs. AI (Individual and Combined Prompts)

Figure 4.2 reveals a comparison of all combined development times. This chart best illustrates how much time is saved with an approach of a single combined prompt. Gemini, at a combined 753 seconds, was approximately 81% quicker than coding by hand. ChatGPT at 839 seconds was approximately 79% quicker. In each of the AI- assisted methods, combining their entire task as a single prompt saw a faster result over creating them individually.

4.2 Quality and Maintainability

The second sub-question established AI code quality and maintainability. It was initially established based on how well the code generated met its stated requirements, and then developer preference as a proxy for maintainability.

The functional completeness of a component designed with both the detailed prompts and the single combined prompt was checked of Gemini and ChatGPT. The results in Table 4.2 show that even with detailed prompts, the LLMs occasionally failed to implement all requirements perfectly.

Table 4.2: Functional Completeness of AI-Generated Components

UI Component	**Reqs.**	**ChatGPT (Detailed)**	**ChatGPT (Single)**	**Gemini (Detailed)**	**Gemini (Single)**
Navbar	5	5 of 5	5 of 5	5 of 5	5 of 5
Button	5	4 of 5	4 of 5	5 of 5	5 of 5
Text Input	3	3 of 3	3 of 3	3 of 3	3 of 3
Form	5	5 of 5	5 of 5	5 of 5	4 of 5
Table	5	4 of 5	4 of 5	5 of 5	4 of 5

No single AI method was flawless on every test. Gemini, however, when presented with the detailed individual instruction, was the only one to achieve a perfect score, with all requirements having been applied on all parts. ChatGPT was unsuccessful on one requirement on every test on the Button and Table components, regardless of whether the prompt was individual or combined. Gemini was also faulty on a limited scale, with one requirement missing on the Form and the Table component with the single combined prompt. However, in all instances where a requirement was missed, the issue was resolved instantly by providing the AI with a single, refined follow-up prompt.

Besides ranking code quality and maintainability, all five programmers participating were also asked comparatively, on a qualitative level, to assess their own manually written implementations against the corresponding AI-programmed implementations. Overall, programmers had a strong preference for the corresponding AI-programmed implementations. Primary reasons were because the AI-programmed implementations regularly had clearer and shorter code. Programmers also noted that the AI-programmed

5

Discussion

This chapter interprets the results presented in the previous section, connecting them to the research questions and the broader context of the related literature. The chapter will initially evaluate the main findings on time efficiency and code quality before considering broader implications across front-end development processes.

This study's results present a balanced view of how LLMs may operate within front-end development. These results confirm development time may be reduced substantially with AI and, within this study, generated code was also of qualitatively preferred by developers. But the results also show how even with exact guidance, output completion is never guaranteed, reinforcing human supervisory control.

The Paradigm Shift of Efficiency: From Automating Tasks to Consolidating Workflows

The best result of this research is the spectacular time savings of these AI-assisted methods. While these individual prompt methods were significantly faster on their own compared with their respective manual counterparts, the one combined prompt is an altogether different development approach. By producing all five desired components with one step (as fast as 753 seconds with Gemini), it was over 5 times as fast as the combined manual development time (3953 seconds).

This opens a window into the real strength of LLMs during development: it isn't a simple case of automating one, discrete task, but bundling an end-to-end workflow. The developer's task list is replaced with "describe the entire required UI and generate it" as opposed to "build A, then B, then C." This aligns with AI's promise of satisfying multi-part, tougher-to-specify requirements rather than a simple "copilot" functionality (8) and a broader automation of the early development phase.

Code Quality and the Efficiency of Iteration

In particular, this study discovered developers on the whole preferred the AI-code rather than their own efforts. They constantly referred to their AI code as readable and concise and noted the resulting AI components appeared more professional. This suggests at least at a junior level, LLMs can act as a productive tool, as a teaching tool and compliance tool with regards to best practices of code stylism and organization. The AI, trained on vast datasets of existing code, may produce solutions that are more aligned with conventional standards than what a developer with less experience might create on their own.

But this qualitative preference must also be balanced against the quantitative data on functional completeness. The outcomes of Table 4.2 indicate that even when a preference for a code existed, on first try it was never flawless. ChatGPT as well as Gemini had continual slips, missing instancing of low level requirements on certain components. This outcome leans against promised outright automation and favors a persisting requirement on human inspection.

Notably, this exploration also uncovered these lesser errors never became large obstacles. In every case, one iteration of prompt refinement was sufficient to catch and correct the error outright. This reveals yet another critical aspect of AI-assisted development. Not just the speed of initial generation, but the speed of debugging and correction. Unlike manual development, where a developer can devote a lot of time figuring out and fixing a bug, the human-AI collaboration loop ensures correction occurs at effectively near-instantaneous rates, adding yet one further reduction of development time. The role of the human developer shifts from pure implementation to one of verification and rapid, high-level refinement.

Finally, this experiment underlines as central the importance of prompt engineering, a point driven repeatedly in the literature (9)(10). That a prompt with considerable detail produced nearly-complete code shows this point: producing high-quality output from an LLM demands a developer expend effort on providing a prompt with direction, specificity, and thoroughness. Quality and speed of AI pay back their dividends only when their output produces code with at least a minimum amount of heavy manual rework.

5.1 Limitations and Future Work

This study makes relevant comments about how LLMs can be of assistance during front-end development, but like all research, it has shortcomings which can inform research directions into the future.

This study's greatest limitation is having a limited and homogeneous sample of developers. The five participants were all junior developers. While this guarantees a consistent base case, their performance will not generalize to the broader developer population. There is a possibility that senior developers, with greater experience, would be far faster at manual coding and reduce or even reverse the time saving afforded by AI. A critical task for a study of the future would be a larger and more diverse sample of developers, including mid- and senior level developers, as a way of looking at how level of experience mediates AI tool use's efficiency- and quality-based trade-offs.

Second, this research was narrowly scoped to a single front-end framework, React. While React is a popular framework, results may not generalize to other frameworks like Angular, Vue, or Svelte, each of which has a distinct syntax, state management paradigm, and best practices. This study was also done with the use of tailwind css for styling, but there are far more styling frameworks that can be used. An interesting direction for future research would be a replication of this study on multiple frameworks to assess whether the performance and quality of AI-generated code remain consistent.

Finally, the complexity of the tasks was constrained to individual UI components. While this was necessary for a controlled study, this does not account for all of front-end development. Further research will require exploring the capability of LLMs on larger projects such as developing fully interactive multi-page applications on the web or even an entire mobile app. Extending the scope further into including backend logic and even database schema generation would provide a better picture of what LLMs can do to redefine the software development lifecycle as a whole.

6

Conclusion

This research evaluated the influence Large Language Models has on the efficiency and quality of front-end UI component generation. The findings indicate Large Language Models like ChatGPT and Gemini offer a transformative speed advantage compared to traditional manual coding. Maximum productivity advantage was obtained with one single detailed prompt executing numerous components at once, a procedure nearly six times quicker compared to manual code creation. There is a suggestion of a paradigm shift from simple task automation toward a wider workflow unification.

Qualitatively, AI-written code was always preferred by junior programmers as clear, concise, and professionally looking, an indication that LLMs can indeed serve as a good tool for exercising best practices. The paper, however, also shows that AI is hardly a one-shot, perfect solution. Minor errors in the output code even now underscore a human review as a verification and fine-tuning process. Encouragingly, these refinements were found to be quick and simple, pointing to an efficient human-AI collaboration model where the developer's role shifts from line-by-line coding to high-level direction and quality assurance.

In short, this thesis concludes that LLMs can be a highly practical and efficient front-end development tool. Prompted with well-engineered text, they can exponentially accelerate high-quality UI component construction, freeing developer time to devote to tougher architectural and UX issues.

References

[1] Chenglei Si, Yanzhao Zhang, Ru-Yuan Li, Ziyang Yang, Ran Liu, and Diyi Yang. **Design2Code: Benchmarking Multimodal Code Generation for Automated Front-End Engineering**. *arXiv preprint arXiv:2403.03163*, 2024. 3, 4

[2] Karlis Rokis and Marite Kirikova. **No-code development and democratization of software creation**, 2023. Incomplete citation. Please find the journal, conference, or book where this was published. 3, 6

[3] Stijn van Otterloo. **Onderzoek naar Low Code Development Platforms**. Technical report, ICT Institute, 2020. Available at: `https://softwarezaken.nl/2020/09/low-code-development-platforms/`. 3, 6

[4] Joost de Schiffart. **Using AI to Support No-code Software Creation: A Case Study**, 2024. 3, 6, 7

[5] Şeyma Şahin. **Transforming Workflow Designs in No-Code Platforms Through ChatGPT**, 2024. 3, 6, 7

[6] Tony Beltramelli. **Sketch2Code: Creating UIs from Hand-Drawn Sketches Using AI**. *arXiv preprint arXiv:1808.09100*, 2018. 3, 7

[7] State of JavaScript. **Front-end Frameworks**, 2024. 4

[8] Reza Hosseini and Pontus Rydén. **Generative AI in Frontend Web Development: A Comparison Between AI as a Tool and as a Sole Programmer**, 2024. 4, 7, 8, 16

[9] David Castelberg and Luca Flury. **LLM-Assisted Development**. Technical report, OST – Eastern Switzerland University of Applied Sciences, 2024. Available at: `https://www.ost.ch/research/publications/llm-assisted-development`. 4, 7, 17

[10] José Martins, Frederico Branco, and Henrique Mamede. **Combining Low-Code Development with ChatGPT to Novel No-Code Approaches: A Focus-group Study**. *Intelligent Systems with Applications*, **20**:200289, 2023. 4, 6, 7, 17

[11] Iqbal H. Sarker. **AI-Based Modeling: Techniques, Applications and Research Issues Towards Automation, Intelligent and Smart Systems**. *SN Computer Science*, **3**(2):158, 2022. 6

[12] Samruddhi Jare. **Evaluating LLMs: Translating Business Needs in NLP into Functional Components in a No-Code Platform**, 2024. 6, 7

[13] Md Kamrul Siam, Huanying Gu, and Jerry Q. Cheng. **Programming with AI: Evaluating ChatGPT, Gemini, AlphaCode, and GitHub Copilot for Programmers**. *ICCA '24: Proceedings of the International Conference on Communications and computer Applications*, 2024. Preprint or conference paper, full details pending publication. 7, 8

[14] Olivia Arizona. **List of UI Components**, 2023. Medium. 9

[15] UXCEL. **Common UI Components**. No date available on source. 9

[16] Bamidele Opeyemi. **A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT**. *arXiv preprint arXiv:2302.11382*, 2023. 10

Appendix

Appendix A - The UI Component Specification Sheet

This document outlines the requirements for five distinct UI components. Each component must be built using React and styled with Tailwind CSS. All components should be centered to screen horizontally. Everything can be made in one project since you will need the other components to create the navbar component.

Keep track of the time it takes to code an individual component. So in the end you should have 5 different time durations it took to code the components. Only keep track when you are actively working on a component (e.g. coding the component, searching web etc). It would be best to code a single component without breaks but if you take a break stop the time and continue when ur getting back to it.

1. Navigation Component

A navigation bar to switch between the component showcases.

Functional Requirements

- **Links:** The bar must contain four links, one for each component: "Button", "Search Bar", "Login Form", and "Salary Table".
- **Interactivity:** Clicking on a link should display the corresponding component and hide the others. Only one component should be visible at a time.

Styling Requirements

- **Layout:** The navigation bar should be positioned at the top of the page. The links within the bar should be horizontally centered on the page.
- **Active Link State (The currently selected link):**

  - Background Color: Blue.
  - Text Color: White.
- **Inactive Link State:**
  - Text Color: Dark Grey.
- **Hover State:** Background color should change to light grey.
- **General:** The bar should have a subtle shadow underneath to separate it from the content (e.g., grey border under).

2. Button Component

A standard form submission button.

Functional Requirements

- **Type:** Must be a submit button.
- **Text:** The button's text must explicitly say "Submit".

Styling Requirements

- **Default State:**
  - Background Color: Blue.
  - Text Color: White.
  - Font: Arial (or a standard sans-serif equivalent like Tailwind's font-sans).
- **Hover State:** Background Color: Baby Blue.
- **Borders:** All corners must be rounded.

3. Text Input: Search Bar

A simple search input field.

Styling Requirements

- **Icon:** A magnifying glass icon must be visible at the start (left side) of the input field.
- **Shape:** The input field must have fully rounded corners (pill shape).
- **Placeholder:** Should contain placeholder text (e.g., "Search...").

4. Form Component: Login

A user login form.

Component Structure

- **Email Field:** An input field for the user's email. A corresponding `<label>` that includes an email icon and the text "Email".
- **Password Field:** An input field for the user's password. A corresponding `<label>` that includes a lock icon and the text "Password".
- **Security:** Text entered into this field must be hidden (e.g., using `type="password"`).
- **Submit Button:** A button with the text "Submit". This button must follow all the requirements outlined in Component 2 (Button Component).

Styling Requirements

Both the email and password input fields must have rounded corners.

5. Table Component: Salary

A table displaying employee salary details.

Data Requirements

- **Columns:** The table must have four columns in this order: Employee, Hours worked, Bonus, Hour rate.
- **Rows:** The table must be populated with the following five rows of data:
  - Alice Johnson, 160, $500, $25
  - Bob Williams, 152, $400, $23

- Charlie Brown, 165, $650, $28
- Diana Miller, 140, $300, $22
- Ethan Davis, 180, $700, $30

Styling Requirements

- **Layout:** The entire table component must be centered on the screen. The outer container of the table must have rounded corners.
- **Header Row (Title Row):** The background color of the header row must be light grey. The text color in the header row must be dark grey.
- **Borders:** There must be a grey border (horizontal line) separating each row in the table body. There should be no vertical borders between columns.

Appendix B - Prompts

Individual Detailed Prompts

1. **Navigation Component**

   Create a UI component using React and Tailwind CSS that functions as a navigation bar for switching between different component showcases. This component should be positioned at the top of the page and centered horizontally. It must contain exactly four links with the text "Button", "Search Bar", "Login Form", and "Salary Table". You need to implement the interactivity so that clicking a link displays its corresponding component while hiding the others, ensuring only one showcase is visible at a time. For styling, the links within the bar must be horizontally centered, and the bar should have a subtle shadow or a grey border underneath to separate it from the page content. The active link must have a blue background with white text, while inactive links should have dark grey text and change to a light grey background on hover.

2. **Button Component**

   Create a standard form submission button component using React and Tailwind CSS that is centered horizontally on the screen. Functionally, the button must be a submit type, and the text displayed on it must be

exactly "Submit". For its styling, the default state should have a blue background, white text, and use an Arial or a standard sans-serif font. When a user hovers over the button, the background color should change to baby blue. All corners of the button must be rounded.

3. **Search Bar Component**

   Create a simple search input field component using React and Tailwind CSS, ensuring the component is centered horizontally on the screen. The styling requires a magnifying glass icon to be visible and positioned at the start (left side) of the input field. The input field itself must have fully rounded corners to give it a "pill shape" and should contain placeholder text that says "Search...".

4. **Login Form Component**

   Create a user login form component using React and Tailwind CSS, with the entire form centered horizontally on the screen. The form's structure must contain three elements in order: first, an email input field with a corresponding `<label>` that includes an email icon and the text "Email"; second, a password input field where the entered text is hidden, with its own `<label>` containing a lock icon and the text "Password". Both of these input fields must have rounded corners. Finally, the form must include a submit button with the text "Submit" that is built to follow all the functional and styling requirements specified for the Button Component.

5. **Salary Table Component**

   Create a table component using React and Tailwind CSS to display employee salary details, ensuring the entire component is centered horizontally on the screen. The table must have exactly four columns in the specific order: Employee, Hours worked, Bonus, and Hour rate. It needs to be populated with five rows of data: Alice Johnson, 160, $500, $25; Bob Williams, 152, $400, $23; Charlie Brown, 165, $650, $28; Diana Miller, 140, $300, $22; and Ethan Davis, 180, $700, $30. For styling, the outer container of the table must have rounded corners. The header row's background should be light grey with dark grey text. A grey horizontal border must

be present to separate each row in the table body, and there should be no vertical borders between the columns.

Single Combined Prompt

Create a single-page application using React and Tailwind CSS that showcases four distinct UI components, with all content centered horizontally on the screen. The application's structure will be controlled by a primary navigation component positioned at the top of the page. This navigation bar must contain four links with the text "Button", "Search Bar", "Login Form", and "Salary Table". The interactivity should be implemented so that clicking a link displays only the corresponding component showcase, hiding all others. For its styling, the links within the bar should be horizontally centered. The active link must have a blue background with white text , while inactive links should have dark grey text. When hovering over an inactive link, its background should change to light grey. The entire navigation bar needs a subtle shadow or a grey border underneath it for separation.

When the "Button" link is active, a standard form submission button component should be displayed. Functionally, this must be a submit button with the explicit text "Submit". Its default styling should be a blue background with white text and an Arial or standard sans-serif font , changing to a baby blue background on hover. All corners of this button must be rounded.

The "Search Bar" link should display a simple search input field. This input must have a magnifying glass icon at the start (left side) and be styled with fully rounded corners to create a "pill shape". It should also contain placeholder text like "Search...".

The "Login Form" link will show a user login form component. This form must contain an email input field with a corresponding label that includes an email icon and the text "Email" , followed by a password input field where entered text is hidden , which also has a label with a lock icon and the text "Password". The form is to be submitted with a button that has the text "Submit" and adheres to all functional and styling requirements previously outlined for the Button Component. Additionally, both the email and password input fields must have rounded corners.

Finally, the "Salary Table" link will display a table for employee salary details. This table must be centered and have an outer container with rounded corners. It requires four columns in the specific order: Employee, Hours worked, Bonus, and Hour rate. The table must be populated with five specific rows of data: Alice Johnson, 160, $500, $25; Bob Williams, 152, $400, $23; Charlie Brown, 165, $650, $28; Diana Miller, 140, $300, $22; and Ethan Davis, 180, $700, $30. For styling, the header row must have a light grey background with dark grey text. There must be a grey horizontal border separating each row in the table body, and no vertical borders should be present between columns.

Appendix C - Developer Manual Implementation Times

Table 6.1: Time taken per developer to manually create each component (seconds).

Developer	Button	Search/Text Input	Form/Login	Table	Navbar
Developer 1	240	300	420	390	1590
Developer 2	420	660	1080	1140	1800
Developer 3	232	751	545	1385	2457
Developer 4	300	480	1080	900	1200
Developer 5	300	300	600	600	1160

Appendix D - Code Snippets for Table Component

Manual Code Implementation

```
import React from 'react';

const Salary = () => {
  return (
  <div className="relative overflow-x-auto mx-auto rounded-lg">
    <table className="min-w-full text-sm text-left text-gray-500 border-
    collapse">
      <thead className="bg-gray-100 text-gray-700 uppercase text-xs">
        <tr>
          <th scope="col" className="px-6 py-3">
            Employee
          </th>
          <th scope="col" className="px-6 py-3">
            Hours worked
```

```
        </th>
        <th scope="col" className="px-6 py-3">
          Bonus
        </th>
        <th scope="col" className="px-6 py-3">
          Hour rate
        </th>
      </tr>
    </thead>
    <tbody>
      <tr className="bg-white border-b border-gray-200">
        <th scope="row" className="px-6 py-4 font-medium text-gray-900
whitespace-nowrap">
          Alice Johnson
        </th>
        <td className="px-6 py-4">160</td>
        <td className="px-6 py-4">$500</td>
        <td className="px-6 py-4">$25</td>
      </tr>
      <tr className="bg-white border-b border-gray-200">
        <th scope="row" className="px-6 py-4 font-medium text-gray-900
whitespace-nowrap">
          Bob Williams
        </th>
        <td className="px-6 py-4">152</td>
        <td className="px-6 py-4">$400</td>
        <td className="px-6 py-4">$23</td>
      </tr>
      <tr className="bg-white border-b border-gray-200">
        <th scope="row" className="px-6 py-4 font-medium text-gray-900
whitespace-nowrap">
          Charlie Brown
        </th>
        <td className="px-6 py-4">165</td>
        <td className="px-6 py-4">$650</td>
        <td className="px-6 py-4">$28</td>
      </tr>
      <tr className="bg-white border-b border-gray-200">
        <th scope="row" className="px-6 py-4 font-medium text-gray-900
whitespace-nowrap">
          Diana Miller
        </th>
        <td className="px-6 py-4">140</td>
        <td className="px-6 py-4">$300</td>
        <td className="px-6 py-4">$22</td>
      </tr>
      <tr className="bg-white border-b border-gray-200">
```

```
57            <th scope="row" className="px-6 py-4 font-medium text-gray-900
    whitespace-nowrap">
58              Ethan Davis
59            </th>
60            <td className="px-6 py-4">180</td>
61            <td className="px-6 py-4">$700</td>
62            <td className="px-6 py-4">$30</td>
63          </tr>
64        </tbody>
65      </table>
66    </div>
67    );
68 };
69
70 export default Salary;
```

Listing 6.1: Manual implementation of the Salary Table component.

AI-Generated Code Implementation

```
1  const SalaryTableComponent = () => {
2      const salaryData = [
3          { id: 1, employee: 'Alice Johnson', hours: 160, bonus: '$500', rate
       : '$25' },
4          { id: 2, employee: 'Bob Williams', hours: 152, bonus: '$400', rate:
        '$23' },
5          { id: 3, employee: 'Charlie Brown', hours: 165, bonus: '$650', rate
       : '$28' },
6          { id: 4, employee: 'Diana Miller', hours: 140, bonus: '$300', rate:
        '$22' },
7          { id: 5, employee: 'Ethan Davis', hours: 180, bonus: '$700', rate:
       '$30' },
8      ];
9
10     return (
11         <div className="w-full max-w-2xl p-8">
12             <div className="overflow-hidden rounded-lg border border-gray
       -200 shadow-md">
13                 <table className="w-full text-left text-sm text-gray-700">
14                     <thead className="bg-gray-100">
15                         <tr>
16                             <th scope="col" className="px-6 py-3 font-
       medium text-gray-600">Employee</th>
17                             <th scope="col" className="px-6 py-3 font-
       medium text-gray-600">Hours worked</th>
18                             <th scope="col" className="px-6 py-3 font-
       medium text-gray-600">Bonus</th>
```

```
                            <th scope="col" className="px-6 py-3 font-
    medium text-gray-600">Hour rate</th>
                        </tr>
                    </thead>
                    <tbody className="bg-white">
                        {salaryData.map((row, index) => (
                            <tr key={row.id} className="border-b border-
    gray-200 last:border-b-0">
                                <td className="px-6 py-4">{row.employee}</
    td>
                                <td className="px-6 py-4">{row.hours}</td>
                                <td className="px-6 py-4">{row.bonus}</td>
                                <td className="px-6 py-4">{row.rate}</td>
                            </tr>
                        ))}
                    </tbody>
                </table>
            </div>
        </div>
    );
};
```

Listing 6.2: AI-generated implementation of the Salary Table component.