Form Auto Generation: An Analysis of GUI Generation

Jedadiah McFarland

University of Nebraska at Omaha
College of Information Science & Technology
Department of Computer Science
Supervisor: Dr. Harvey Siy

Honors Capstone Report

in partial fulfillment for the degree
Bachelor of Science in Computer Science (Honors Distinction)
in Spring 2023

Form Auto Generation

—

An Analysis of GUI Generation

Submitted by:

Jedadiah McFarland
E-Mail: jedadiahmcfarland@unomaha.edu
B.S. Computer Science

Submission date: May 2023

Abstract

Graphical User Interfaces (GUIs) have transformed how we interact with computers, offering visually appealing and intuitive systems. This paper explores the origins and evolution of GUIs, explicitly focusing on form auto-generation in modern GUI-driven environments. Form auto-generation has emerged as a prominent practice, enabling automatic form creation based on predefined models. To better understand form auto-generation, I investigate SurveyJS, an open-source form auto-generation library known for its active development and support. This investigation aims to understand how SurveyJS recognizes and renders objects from a JSON model. The methodology involves a trial and error examination of the library, exploring its live demos and source code. Different approaches to form auto-generation, including template-based methods, database-driven approaches, web form builders, document automation platforms, and custom programming, are explored. This research enhances our understanding of this crucial aspect of modern computing by delving into GUI origins, investigating SurveyJS, and examining various forms of auto-generation approaches. The findings suggest that form auto-generation will likely grow in popularity due to its accessibility, productivity benefits, dynamic personalization, integration with emerging technologies, and cross-platform compatibility. Organizations can leverage form auto-generation to optimize form creation processes, improve user experiences, and boost productivity across various domains.

Contents

List of Figures

1 Introduction

Graphical User Interfaces (GUIs) have played a pivotal role in shaping how we interact with computers. From the early days of text-based interfaces to the present era of visually appealing and intuitive systems, GUIs have revolutionized the user experience (UX). This paper delves into the origins and evolution of GUIs, specifically focusing on generating forms in modern GUI-driven environments.

The journey of GUIs began with computers such as the Xerox Alto and Apple's Lisa, which introduced GUIs to the personal computer market. As technology progressed, GUIs became the norm for personal computers, embodied by Apple's release of Mac OS X and its GUI-centric nature. GUIs have become essential for enhancing user understanding and productivity, particularly in the information-driven era we live in today. In addition to operating systems, the evolution of GUIs has extended to web development. Advanced webpages now offer dynamic UX through the utilization of versatile languages such as JavaScript (JS), Hyper Text Markup Language (HTML), and Cascading Style Sheets (CSS). Within this context, form auto-generation has emerged as a prominent practice, allowing for automatic form creation based on predefined models.

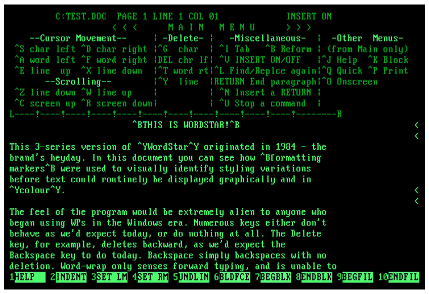
This paper focuses on exploring form auto-generation, with a particular emphasis on SurveyJS—an open-source form auto-generation library known for its active development community and strong developer support. The investigation of SurveyJS aims to understand how the library recognizes an object in a JSON model and how it renders that object while maintaining basic HTML functionality. The methodology employed in this study involves a trial-and-error examination of the SurveyJS library, including its provided examples and source code. A deeper understanding of form auto-generation techniques can be learned by gaining insights into these mechanisms.

Furthermore, this paper explores different approaches to form auto-generation, including template-based methods, database-driven approaches, web form builders, document automation platforms, and custom programming. Each approach caters to specific use cases, offering distinct strengths and capabilities. By examining the origins of GUIs, investigating the SurveyJS library, and exploring different approaches to form auto-generation, this research seeks to enhance our understanding of this important aspect of modern computing.

2 Background

2.1 Origins of GUI

In the early age of computers, GUIs rarely existed; everything with a computer was done in a terminal, either through a command line interface or a text-based user interface (TUI). TUIs are fullscreen applications that allow some freedom for user navigation via arrow keys. Such interfaces for applications emerged ever since computer monitors became common modes of display in the 1960s. Implementation details vary, but by the 1970s, UNIX systems have developed portable ways of displaying text user interfaces (Arnold and Amir, 1977). By the 1980s these were the prevalent types of application interfaces on terminals. The availability of TUIs for business applications contributed to the popularity of personal computers (Figure 1).



```
     C:TEST.DOC  PAGE 1 LINE 1 COL 01            INSERT ON
                <<<    M A I N   M E N U    >>>
   --Cursor Movement--     | -Delete- |   -Miscellaneous-   |  -Other  Menus-
 ^S char left ^D char right |^G  char  | ^I Tab    ^B Reform | (from Main only)
 ^A word left ^F word right |DEL chr lf| ^V INSERT ON/OFF    |^J Help  ^K Block
 ^E line  up  ^X line down  |^T word rt|^L Find/Replce again|^Q Quick ^P Print
      --Scrolling--         |^Y  line  |RETURN End paragraph|^O Onscreen
 ^Z line down ^W line up    |          | ^N Insert a RETURN |
 ^C screen up ^R screen down|          | ^U Stop a command  |
L----!----!----!----!----!----!----!----!----!----!----!--------R
                         ^BTHIS IS WORDSTAR!^B                              <
                                                                            <
This 3-series version of ^YWordStar^Y originated in 1984 - the
brand's heyday. In this document you can see how ^Bformatting
markers^B were used to visually identify styling variations
before text could routinely be displayed graphically and in
^Ycolour^Y.                                                                 <
                                                                            <
The feel of the program would be extremely alien to anyone who
began using WPs in the Windows era. Numerous keys either don't
behave as we'd expect today, or do nothing at all. The Delete
key, for example, deletes backward, as we'd expect the
Backspace key to do today. Backspace simply backspaces with no
deletion. Word-wrap only senses forward typing, and is unable to
1HELP   2INDENT 3SET LM 4SET RM 5UNDLIN 6BLDFCE 7BEGBLK 8ENDBLK 9BEGFIL 10ENDFIL
```

Figure 1: Text-based user interface (WordStar)

Source: (Leggitt, 2022)

However, with the increasing capabilities of computer and monitor hardware, graphical user interfaces (GUIs) have emerged as the standard for new application development. This research focuses on personal computers and how their GUIs came to be. The Xerox Alto and Apple's Lisa are two of the first computers that implemented a GUI design on a commercial computer (Computer History Museum, 1983; Raymond and Landley, 2004). Xerox Alto was coined as the first commercial computer with a GUI as it was released in 1973, and we can see an example of this GUI in Figure 2. It created these GUIs using Basic Combined Programming

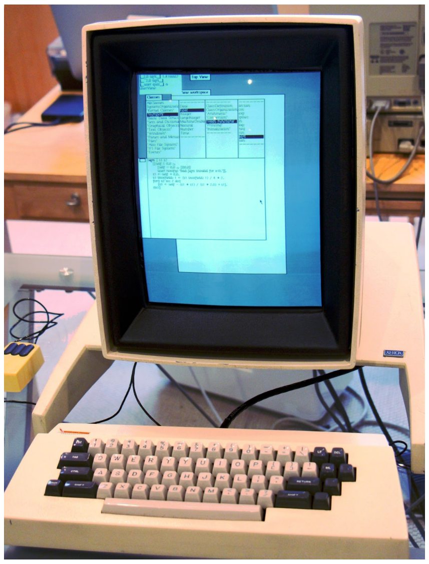


Figure 2: Xerox Alto GUI

Source: (Shirriff, 2017)

Language (BCPL) and Mesa programming languages,
with a new concept of bit block transfer (bit blit) operation which assisted greatly in creating the GUIs (Xerox, 1976). While the languages are outdated and replaced, the bit blit operations are still used throughout computer graphics today.

2.2 Middle era GUIs

The mid-2000s brought many new advancements in GUIs and their generation. Video games are becoming more common. Apple has released Mac OS X, which only solidified the need and suitability for GUIs as the default for operating systems (OS) and applications. At this time, terminal-based personal computers are almost completely gone but let us look at how Apple created its operating system with GUIs in mind. Mac OS was built from Unix and used common OS programming languages like C, C++, Swift, Assembly, and Objective-C. More interestingly, Mac OS uses Aqua, shown in Figure 3, as the default user interface, which defines the design language and visual theme of the OS, also written in C++.

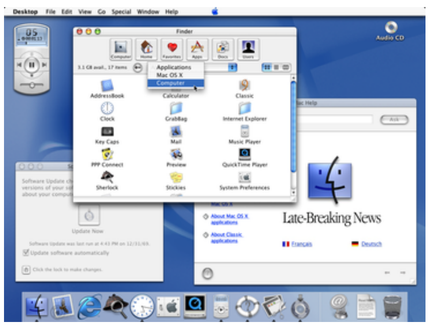


Figure 3: Mac OS X Aqua theme

Source: (Wikipedia, 2023)

2.3 Present GUIs

From the beginning of using the terminal to move files around, personal computers have now evolved to allow the user to see all their files and where they are going, all without commands. This evolution allows less experienced users to understand computers much easier and use them more effectively, which is necessary in today's information era. Operating systems have become very advanced and exclude GUI based; these GUIs are created using multiple languages but still most commonly C-based languages. Additionally, there is tremendous support for third-party applications to run on the OS without fail.

Webpages have also become highly advanced; some high-end websites will have each object act dynamically to the user providing the best user experience (UX). The generation of said webpages relies heavily on versatile languages such as JavaScript (JS), Hyper Text Markup Language (HTML), and Cascading Style Sheets (CSS). These are the most dominant languages in the web development market that can create extremely simple or advanced web applications. In terms of automatically creating forms, multiple libraries or plugins for the above languages will use a model to repeatedly create the same form, which can then be modified in a defined manner. This process is known as form auto-generation and will serve as the topic of this research.

3 Synthesis and Discussion

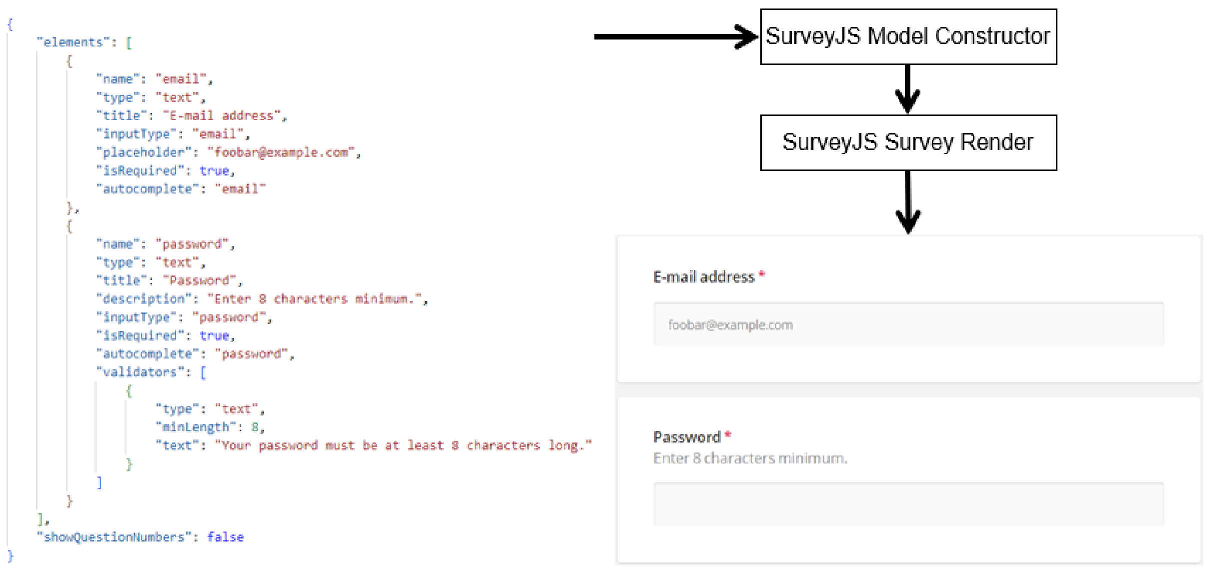
3.1 Looking into a particular library

SurveyJS is an open-source form auto-generation library with an active development team and great support for developers using the library (Devsoft Baltic OÜ, 2020). So, to better understand form auto-generation, I did a deep dive into SurveyJS to understand how the library takes a JavaScript Object Notation (JSON) model and renders it into a form. The two main goals of this venture are to understand how it recognizes an object in a JSON model and how it renders that object while maintaining basic HTML functionality. To do this, I used an intuitive trial-and-error methodology that allowed me to understand each goal.

3.2 Methodology

Firstly, I went to the main website for SurveyJS to play around with the examples they provide to demo their library. Understanding these demos gave me a baseline for how the app worked and what specific codes or objects translated to in the resultant form. Using this newfound knowledge, I went to the library's source code, where after some trial and error, I found a folder full of HTML component definitions. Here I was able to understand how the objects read from the model were translated to fit the format of an HTML element while retaining its functionality. These components define HTML objects like buttons, panels, pages, images, etc. Now that I have found how the objects are defined, I need to understand how the library interprets the model. To do this, I returned to the demo applications to find the commonly defined structure among each. The idea is that each demo passes the model through a specific entry point to the library. I found this to be accurate and discovered that the Model constructor acted as the entry point, with the Survey object normalizing and rendering the model before being passed to the React renderer, creating the form on the HTML Document Object Model (DOM).

This process can be seen simply through the diagram in Figure 4 and the sequence diagram in Figure 5.



```
{
    "elements": [
        {
            "name": "email",
            "type": "text",
            "title": "E-mail address",
            "inputType": "email",
            "placeholder": "foobar@example.com",
            "isRequired": true,
            "autocomplete": "email"
        },
        {
            "name": "password",
            "type": "text",
            "title": "Password",
            "description": "Enter 8 characters minimum.",
            "inputType": "password",
            "isRequired": true,
            "autocomplete": "password",
            "validators": [
                {
                    "type": "text",
                    "minLength": 8,
                    "text": "Your password must be at least 8 characters long."
                }
            ]
        }
    ],
    "showQuestionNumbers": false
}
```

Figure 4: High-level depiction of how a form is created from a JSON model
Source: Own Illustration

3.3 Results

Through the exploration of SurveyJS, it became evident that the library employs keyword and object-matching techniques to interpret the JSON objects provided. This finding aligns with the expected behavior of the library, as it defines a specific format that the JSON file must adhere to for compatibility. Any unrecognized keywords or naming conventions within the JSON file would throw an error. From the methodology described earlier, it was observed that SurveyJS achieves the conversion of JSON objects to HTML elements by utilizing multiple component files. Where within a specific component file, multiple variables and functions would correspond to that of standard HTML elements.

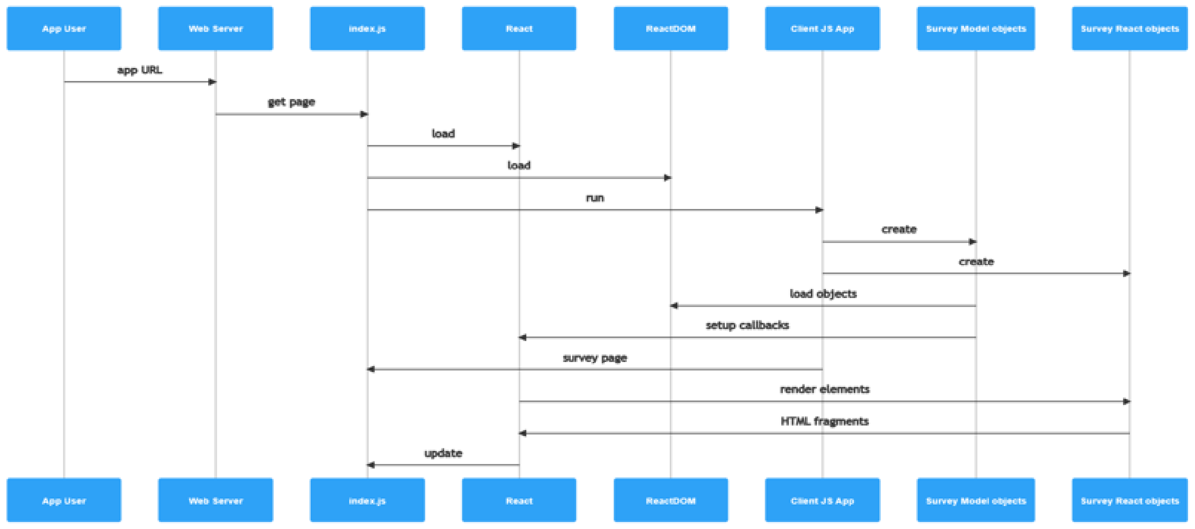


Figure 5: Sequence diagram of SurveryJS rendering a form from JSON model
Source: Dr. Harvey Siy

3.4 Different form generation approaches

Within form auto-generation, five main approaches offer distinct ways to create and generate forms. These different approaches to form auto-generation provide options for various scenarios and user preferences. Let us explore these approaches in more detail.

3.4.1 Template-Based Method

The template-based method relies on predefined templates that contain placeholders. When the templates are loaded, the placeholders are automatically populated with real data from a data source. This approach allows for efficient form creation by reusing templates and dynamically filling them with the necessary information.

3.4.2 Database-Driven Approach

In the database-driven approach, the form structure and data are stored in a database. The database holds the definitions of form fields and properties. When generating a form, the relevant data is queried from the database, and the form is dynamically created based on the retrieved information (Rahman and Nandi, 2019). This approach is advantageous when dealing with large volumes of data and dynamic form requirements.

3.4.3 Web Form Builders

Web form builders provide a visual interface that enables users to create forms by dragging and dropping elements. These tools offer a user-friendly way to design forms, allowing non-technical users to create forms without writing code. Many web form builders also support the generation of forms based on predefined templates or imported data, further simplifying the form creation process.

3.4.4 Document Automation Platforms

Document automation platforms facilitate the automatic generation of many documents or forms. These platforms typically provide a user interface that allows users to create and customize forms and form templates (Memon et al., 2001). These templates can be designed easily; once completed, the user can merge the template with provided data to create a form.

3.4.5 Custom Programming

Custom programming allows for highly customized form generation to meet specific requirements. With custom programming, developers have full control and flexibility over the form creation process. They can utilize programming languages, libraries, and frameworks to build forms tailored to their unique needs. JavaScript libraries, like SurveyJS, are often employed to generate forms programmatically.

3.5 Prediction on where its headed and future usability

Form auto-generation is bound to grow in popularity due to several key factors. Firstly, it simplifies form creation by empowering non-technical users to create forms without extensive programming knowledge. This accessibility opens form development to a broader range of individuals and organizations. Secondly, it enhances productivity by automating the form generation process, saving time and effort. Additionally, form auto-generation allows for dynamic and personalized forms tailored to individual users or specific data sources, improving the UX and data collection. Integrating emerging technologies, such as artificial intelligence, further automates the process and increases accuracy. Lastly, its cross-platform compatibility ensures consistent form generation across different operating systems and devices. With these advantages in mind, form auto-generation will likely continue gaining popularity as organizations seek efficient and user-friendly solutions for form creation.

3.6 Best use case for each scenario

The five form auto-generation methods cater to different use cases based on their strengths and capabilities. The template-based approach is valuable in industries like HR, legal, or finance, where standardized forms with variable data must be generated. The database-driven approach is suitable when there is a need for dynamically generating forms based on large volumes of data stored in a database, commonly seen in CRM or CMS applications. Web form builders are ideal for non-technical users who require quick and easy form creation, making them useful in web development projects, surveys, and event registrations. Document automation platforms excel in organizations dealing with complex document generation, such as insurance companies or legal firms, offering advanced features for template design, data integration, and workflow automation. Lastly, custom programming provides the utmost flexibility for unique requirements, system integration, and fine-grained control over form generation.

4 Conclusion

Graphical User Interfaces (GUIs) have evolved significantly, shaping how we interact with computers and enhancing UX. This paper explored the origins and evolution of GUIs, specifically focusing on form auto-generation in modern GUI-driven environments. We delved into the history of GUIs, highlighting key advancements from early computers like the Xerox Alto to the widespread adoption of GUI-centric operating systems like Mac OS X. The SurveyJS library was examined as an example of an open-source form auto-generation tool. Through a trial-and-error methodology, the investigation uncovered how SurveyJS recognizes and renders objects from a JSON model while maintaining basic HTML functionality. It was observed that the library utilizes keyword and object-matching techniques to interpret the provided JSON objects. While multiple component files redefine standard HTML tags, facilitating the seamless conversion of JSON objects into corresponding HTML elements.

Furthermore, we discussed the different approaches to form auto-generation, including template-based methods, database-driven approaches, web form builders, document automation platforms, and custom programming. Each approach is biased toward specialized use cases and offers distinct strengths and capabilities, where organizations can choose an approach to best fit their needs. The synthesis and discussion highlighted the growth potential of form auto-generation. Accessibility, productivity gains, dynamic personalization, integration with emerging technologies, and cross-platform compatibility make form auto-generation an attractive solution for efficient and user-friendly form creation. As organizations seek streamlined and intuitive form development, the popularity of form auto-generation is expected to rise.

In conclusion, exploring GUI origins, investigating the SurveyJS library, and examining different form generation approaches contribute to a deeper understanding of this critical aspect of modern computing. By leveraging form auto-generation techniques and selecting the appropriate approach, organizations can optimize form-creation processes, improve user experiences, and enhance overall productivity in various domains.

References

Arnold, K. C. R. C. and Amir, E. (1977). Screen updating and cursor movement optimization: A library package. Technical report, University of California, Berkeley.

Computer History Museum (1983). 1983 | Timeline of Computer History. `https://www.computerhistory.org/timeline/1983/`.

Devsoft Baltic OÜ (2020). SurveyJS - JavaScript libraries for surveys and forms. `https://surveyjs.io/`.

Leggitt, B. (2022). Word processing software: Revolution pending? `https://popzazzle.blogspot.com/2022/05/word-processing-software-revolution-pending.html`.

Memon, A., Pollack, M., and Soffa, M. (2001). Hierarchical GUI test case generation using automated planning. *IEEE Transactions on Software Engineering*, 27(2):144–155.

Rahman, P. and Nandi, A. (2019). Transformer: a database-driven approach to generating forms for constrained interaction. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, IUI '19, pages 485–496. Association for Computing Machinery.

Raymond, E. and Landley, R. W. (2004). The art of Unix usability: The first GUIs. `http://www.catb.org/~esr/writings/taouu/html/ch02s05.html`.

Shirriff, K. (2017). The Xerox Alto, Smalltalk, and rewriting a running GUI. `http://www.righto.com/2017/10/the-xerox-alto-smalltalk-and-rewriting.html`.

Wikipedia (2023). Aqua (user interface). `https://en.wikipedia.org/w/index.php?title=Aqua_(user_interface)&oldid=1149287464`. Page Version ID: 1149287464.

Xerox (1976). ALTO: A personal computer system hardware manual. `https://web.archive.org/web/20170904111228/http://bitsavers.org/pdf/xerox/alto/Alto_Hardware_Manual_Aug76.pdf`.