PharmaSUG China 2025 - Paper AA-125

AMAT - Designing for Flexibility in SAS TLF Automation - A React Case Study

Yong Cao, Phastar

ABSTRACT

As professionals from diverse backgrounds enter the field of statistical programming, and generative AI reshapes the landscape of software development, there has been a surge in tools built with various tech stacks and frameworks. However, many presentations focus solely on the final product, overlooking the development journey—where critical decisions, challenges, and learning moments reside. This paper argues that understanding the rationale behind application design is essential for appreciating its value and for inspiring others that are looking to build similar tools.

Using the development of the AMAT web application—a web-based application designed to automate the generation of TLF (Tables, Listings, and Figures) programs —as a case study, this paper explores the motivations, architectural decisions, and trade-offs made throughout the process. Built with React and React-Router, the application emphasizes flexibility, usability, and user control. Additionally, this paper explores how generative AI facilitated the development process through code completion, debugging assistance, and learning support. Finally, it reflects the evolving role of SAS programming in a rapidly changing industry and encourages programmers to embrace new technologies while leveraging their domain expertise.

Keywords: Web development, React, SAS, TLF automation, AMAT, application design, AI

INTRODUCTION

This paper outlines the development journey of a web-based application that is designed for TLF automation, highlighting key design decisions and trade-offs. By sharing this experience, the author hopes to encourage and inform others to pursue similar initiatives, particularly those that bridge statistical programming knowledge with emerging technologies.

BACKGROUND AND MOTIVATION

Despite the growing importance of R in statistical programming, within a CRO environment, SAS continues to play a crucial and arguably dominant role. Nearly all our projects still heavily rely on SAS for tasks such as TLF programming. Therefore, it remains essential to continue investing in SAS capabilities to maintain efficiency and quality.

AMAT (a modular approach to TLF) is a SAS macro suite designed to simplify and streamline table programming. Initially inspired by the author's former work experience PPD, AMAT addresses common challenges faced by statistical programmers in TLF programming. Key features of AMAT macros include:

- **Modularity**. Each macro is dedicated to a single function in table programming. Calculation and presentation are separate concerns and are covered by different macros.
- **Consistency**. The macros follow standardized naming conventions and parameter structures, reducing the learning curve and minimizing errors.
- **Flexibility**. No restrictive assumptions are made about input (ADaM) datasets. Adaptable to various table structures and reporting needs. Customization (e.g., percentage formats, descriptive statistics layout, rounding rules) is natively supported and intuitive and straightforward.
- **Datasets flow through macros**. Output from one macro seamlessly serves as input to the next, ensuring a predictable and maintainable process for all tables.
- **Powerful default system**. A lot of macro parameters are set to configurable default values through global macro variables. The study team can then set these default values at study level, which can vary among studies. These defaults are the ones that control percentages format, descriptive statistics layout etc. The individual programs inherit these defaults without needing to

explicitly set them and study leads can centrally change them without needing to touch individual programs.

In essence, AMAT offers a complete workflow for table programming. It can be viewed as a set of high-level APIs for generating tables efficiently, as well as a scaffold for TLF development that abstracts away low-level implementation details. This allows statistical programmers to focus on what matters most: defining the input logic and interpreting the output, rather than getting bogged down in repetitive coding tasks.

Below are the diagrams of how AMAT macros produce frequency counts, percentages, and descriptive statistics.

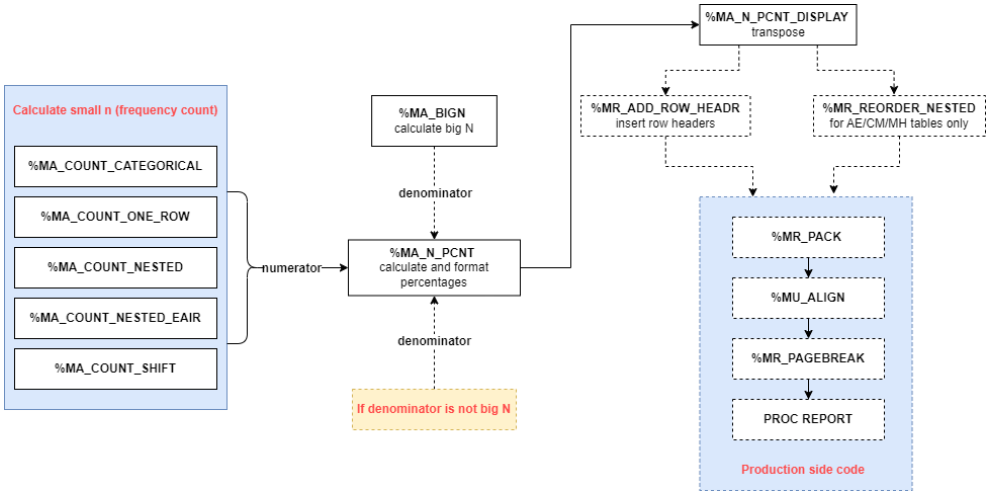

Figure 1 AMAT process flow for frequency count and percentage

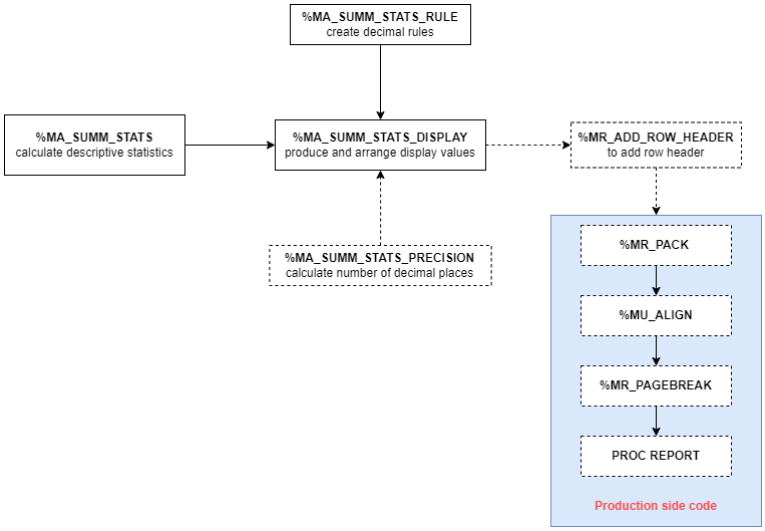

Figure 2 AMAT process flow for descriptive statistics

Despite its advantages, the primary barrier to widespread adoption of the AMAT macro suite within our company is its steep learning curve. Programmers without prior experience often find themselves spending significant time understanding its design principles and master individual macros—potentially offsetting the efficiency gains it promises.

To mitigate this challenge, template programs for common table types have been developed and provided to programmers. This has proven effective, as template programs allow users to learn the macros in a practical context rather than starting from scratch. However, since templates are static, they cannot accommodate every study's unique table requirement. Programmers often need to modify these

templates, which again can be difficult without some understanding of the macros, leading to frustration and inefficiencies.

To fundamentally address these adoption challenges, the author began exploring an innovative approach. The goal soon became to develop a web-based application that dynamically generates TLF programs. This solution should empower programmers to:

- Configure tables interactively through an intuitive point-and-click interface
- Automatically generate SAS code leveraging the AMAT macro suite
- Support diverse table types by providing great flexibility

CRITICAL DECISIONS

In the realm of TLF (program) automation, numerous approaches have emerged. Some emphasize metadata-driven design, while others aim to implement standards such as ARS (Analysis Results Standard). There are also tools that rely on predefined TLF templates, and newer solutions are beginning to explore the use of generative AI for accelerated development.

Each of these methods has its merits, but the author believes that effective TLF automation must begin with a solid foundation — one built on well-defined macros, consistent workflows, and deep domain expertise. Only with this kind of stable infrastructure can automation be scalable, maintainable, and truly impactful.

While the overall architecture of the AMAT web app may appear straightforward, several critical design decisions have had a profound impact on its usability, scalability, and long-term value. These choices span from the fundamental approach to table building, to how programs are generated, whether to include data processing logic, and even which technologies to use on the frontend.

Each of these decision points involves trade-offs and reflects a deliberate balance between flexibility, simplicity, and maintainability. The following sections explore these key decisions in detail, offering insight into the rationale behind the current implementation and the direction for future development.

TABLE BUILDING APPROACH

There are two primary approaches to building tables in the AMAT web app. The first is based on standardized TLF shell templates, which most companies already maintain. Since each template corresponds closely to a specific output format, very few customization options are needed.

For example, Figure 3 illustrates a change-from-baseline table design. In this case, the only dynamic element is the subtitle, which may or may not be included and can represent parameters or other variable(s). This approach offers several clear advantages:

- **Development and Testing Efficiency**: There is nearly a one-to-one mapping between templates and SAS programs, making both development and validation straightforward.
- **Simplified User Interface**: With minimal dynamic components, the UI for each template remains clean and easy to use.
- **User-Friendly Experience**: Users simply select the appropriate template. Beyond that, little additional input is required.

By leveraging existing templates, this method ensures consistency with organizational standards while minimizing complexity for both developers and end users.

Parameter: xxxxxxxxxxx

		Actual value								Change from baseline							
Treatment Group	Analysis Visit	n	Mean	Min	Q1	SD	Median	Q3	Max	n	Mean	Min	Q1	SD	Median	Q3	Max
Active Drug (N=XXX)	Analysis Visit 1	xx	xx.xx	xx.x	xx.xx	xx.xxx	xx.xx	xx.xx	xx.x	xx	xx.xx	xx.x	xx.xx	xx.xxx	xx.xx	xx.xx	xx.x
	Analysis Visit 2	xx	xx.xx	xx.x	xx.xx	xx.xxx	xx.xx	xx.xx	xx.x	xx	xx.xx	xx.x	xx.xx	xx.xxx	xx.xx	xx.xx	xx.x
	Repeat Analysis Visit	xx	xx.xx	xx.x	xx.xx	xx.xxx	xx.xx	xx.xx	xx.x	xx	xx.xx	xx.x	xx.xx	xx.xxx	xx.xx	xx.xx	xx.x
Placebo (N=XXX)	Analysis Visit 1	xx	xx.xx	xx.x	xx.xx	xx.xxx	xx.xx	xx.xx	xx.x	xx	xx.xx	xx.x	xx.xx	xx.xxx	xx.xx	xx.xx	xx.x
	Analysis Visit 2	xx	xx.xx	xx.x	xx.xx	xx.xxx	xx.xx	xx.xx	xx.x	xx	xx.xx	xx.x	xx.xx	xx.xxx	xx.xx	xx.xx	xx.x
	Repeat Analysis Visit	xx	xx.xx	xx.x	xx.xx	xx.xxx	xx.xx	xx.xx	xx.x	xx	xx.xx	xx.x	xx.xx	xx.xxx	xx.xx	xx.xx	xx.x

Figure 3 An example of change from baseline table

However, this approach also comes with several notable drawbacks.

First, organizations — especially CROs working with multiple clients — often maintain a very long list of standard templates. Converting each of these into a dedicated interface not only requires significant development effort but also places the burden on users to be familiar with the template library to select the correct one. When similar templates exist, distinguishing between them can become confusing and frustrating for end users.

Second, TLF shells represent specific ways to present a summary, but there can be multiple variations of the same underlying summary. Figure 4 illustrates three different variations of change from baseline table. Compared to Figure 3, the first two variations in Figure 4 differ only in the layout structure. A very large portion of the SAS programs to generate them are the same. In such cases, treating them as separate templates within a UI-driven automation tool is inefficient. Instead, it makes more sense to offer different layout options within a single, flexible interface.

Third, note that the only difference between the last two designs in Figure 4 is how descriptive statistics are organized - a variation that commonly occurs across all summaries of continuous variables. If we need to support multiple styles (as is often the case in CRO settings), creating a separate template for each would be impractical. Since these differences are unrelated to what is being summarized (e.g., AVAL and CHG), they should instead be abstracted as configurable options within the interface.

The same principle applies to which variables are included in the summary. Whether a table summarizes only AVA or also includes percentage change from baseline (PCHG), these variations should not require separate templates. Should we treat them differently just because of the number of variables being summarized? What about adding percent change from baseline as an optional output? These decisions highlight the limitations of a rigid template-based approach and reinforce the need for a more dynamic and orthogonal design.

Given the limitations of a template-driven approach, the author chooses a more flexible and scalable solution: a category-based design for table automation. In this model, a category represents a high-level abstraction of common table types. For example, in earlier examples, both the change-from-baseline table (regardless of whether percentage change is included) and the summary of AVAL fall under the "Summary" category. The defining characteristic of this category is that it represents tables summarizing one or more continuous variables.

Within this category, aspects such as:

- By-variables (e.g., analysis visit and parameter),
- Layout of descriptive statistics, and
- Orientation of treatment groups and summary variables (vertical vs. horizontal)

are treated as configurable options — not separate templates. This modular approach ensures flexibility without unnecessary complexity. Other categories include:

- “Nested” which corresponds but not limited to frequently seen TEAE tables, concomitant medications and medical history tables. Commonality is summary of one or several variables with hierarchy.

- “Shift” which corresponds to shift table. Commonality is summary of two categorical variables that corresponds to baseline and post-baseline status.
- “Generic” which corresponds to summary of mixed things. Commonality of this type of summary is they are made up of sections where each section is either summary of continuous variables, or a frequency counts type of summary.

For each category, a set of predefined options covers common variations in table layout and content. These options align directly with the capabilities of the underlying AMAT SAS macros, ensuring seamless integration between UI choices and generated code.

One of the key advantages of this approach is scalability:

- New options can be added to support additional table variations.
- However, categories remain stable, growing only infrequently as new use cases emerge.

From a user experience perspective, this means users always interact with a familiar interface. Over time, the interface evolves by adding visual controls that expand coverage to more table types — without fundamentally changing how users work.

Parameter: xxxxxxxxxxx

		Drug X (N=XXX)		Placebo (N=XXX)	
Analysis Visit	Statistics	Actual value	Change from baseline	Actual value	Change from baseline
Analysis Visit 1	n	xx	xx	xx	xx
	Mean	xx.xx	xx.xx	xx.xx	xx.xx
	SD	xx.xxx	xx.xxx	xx.xxx	xx.xxx
	Min	xx.x	xx.x	xx.x	xx.x
	Q1	xx.xx	xx.xx	xx.xx	xx.xx
	Median	xx.xx	xx.xx	xx.xx	xx.xx
	Q3	xx.xx	xx.xx	xx.xx	xx.xx
	Max	xx.x	xx.x	xx.x	xx.x
Analysis Visit 2	n	xx	xx	xx	xx
	Mean	xx.xx	xx.xx	xx.xx	xx.xx
	SD	xx.xxx	xx.xxx	xx.xxx	xx.xxx
	Min	xx.x	xx.x	xx.x	xx.x
	Q1	xx.xx	xx.xx	xx.xx	xx.xx
	Median	xx.xx	xx.xx	xx.xx	xx.xx
	Q3	xx.xx	xx.xx	xx.xx	xx.xx
	Max	xx.x	xx.x	xx.x	xx.x

Parameter: xxxxxxxxxxx

Analysis Visit	Variable	Statistics	Drug X (N=XXX)	Placebo (N=XXX)
Analysis Visit 1	Actual value	n	xx	xx
		Mean	xx.xx	xx.xx
		SD	xx.xxx	xx.xxx
		Min	xx.x	xx.x
		Q1	xx.xx	xx.xx
		Median	xx.xx	xx.xx
		Q3	xx.xx	xx.xx
		Max	xx.x	xx.x
	Change from baseline	n	xx	xx
		Mean	xx.xx	xx.xx
		SD	xx.xxx	xx.xxx
		Min	xx.x	xx.x
		Q1	xx.xx	xx.xx
		Median	xx.xx	xx.xx
		Q3	xx.xx	xx.xx
		Max	xx.x	xx.x

Parameter: xxxxxxxxxxx

Analysis Visit	Variable	Statistics	Drug X (N=XXX)	Placebo (N=XXX)
Analysis Visit 1	Actual value	n	xx	xx
		Mean (SD)	xx.xx (xx.xxx)	xx.xx (xx.xxx)
		SE	xx.xxx	xx.xxx
		Median	xx.xx	xx.xx
		Q1, Q3	xx.xx, xx.xx	xx.xx, xx.xx
		Min, Max	xx.x, xx.x	xx.x, xx.x
	Change from baseline	n	xx	xx
		Mean (SD)	xx.xx (xx.xxx)	xx.xx (xx.xxx)
		SE	xx.xxx	xx.xxx
		Median	xx.xx	xx.xx
		Q1, Q3	xx.xx, xx.xx	xx.xx, xx.xx
		Min, Max	xx.x, xx.x	xx.x, xx.x

Figure 4 Different variations of change from baseline table

READY-TO-RUN PROGRAM?

While it might seem ideal to generate completely automated, ready-to-run SAS programs, the AMAT web app intentionally focuses solely on table generation rather than data processing. This deliberate design choice stems from several key considerations:

- Data processing falls outside of the original goal, which is to lower the adoption barriers of AMAT macros. It's something programmers are already proficient with. This also simplifies the web app by not having a complex interface to cover edge cases. For example, occasionally some tables may summarize things that are not so-called "one proc away" from ADaM. In this case, it's to manually code the data processing part, but it's extremely hard to do it on the interface.
- Repeated tables. Most programmers are more used to the workflow of creating a macro to cover all repeated tables. These macros typically expose parameters for different subset conditions or analysis flag etc. Exporting a program that skips the data processing part is good enough and flexible enough for programmers to transform them into macros. With the power of AMAT SAS macros, things like scaling up a table to add variables only take one line of code.
- Skip the data processing clearly sets the boundary between what the web app does and what its users need to do. It also clearly makes its users the owners of the programs.
- In the future, once the app is integrated with ADaM datasets and/or specification documents, implementing a dataset filter interface will become more straightforward and user-friendly. This would enable the automation of the data-processing steps for most tables, further reducing manual coding effort. However, this functionality is not considered a priority.

This focused approach ensures the web app remains lightweight and maintainable while delivering maximum value where it matters most - in simplifying the creation of table programs that utilize AMAT SAS macros. The separation of concerns between data preparation and table generation ultimately provides both greater flexibility and better long-term sustainability.

REAL-TIME RENDERING SHELLS

One of AMAT’s most praised features is its real-time shell rendering - displaying the table structure visually as users configure it. During development and testing, it probably costs 10x development effort than the module for generating SAS programs. Why is it spending so much time on this feature that seems not relevant to SAS program generation?

- As previously mentioned, table building in AMAT web app isn’t linked to standard template of shells. Instead, various options are provided in each category to generate various tables. It’s often difficult to just rely on textual descriptions of these options and users’ imagination to understand how final output looks. This not only eliminates guesswork but also saves time and frustration by reducing trail-and-errors.
- True WYSIWYG (What You See Is What You Get) Experience. Instead of only having an interface with a bunch of textboxes and dropdown lists of which the sole purpose is to be transcribed to SAS programs, having a shell rendering in real time that provides user instant visual feedback is what makes AMAT web app unique and attractive to its users. It’s often the case that how something works is just as important as what it does.
- Since the purpose of the web app is to simplify the use of the AMAT SAS macros, it's essential that the interface itself doesn't become a new obstacle. A steep learning curve would defeat the very purpose of the tool. With intuitive design and real-time preview capabilities, the app ensures that users can start building tables effectively — often without needing any formal training at all.
- Critical for Future Development. In the future, the web app will support figure generation. Configuring visual elements without a preview is nearly impossible. In addition, this feature lays out the foundation of a TLF shell application that the author is planning to develop.

This real-time rendering feature is more than just a "nice-to-have”. It fundamentally transforms the application from a basic code generation tool into a true productivity booster. By leveraging the power of modern web technologies, it elevates the user experience, making the application feel polished, responsive, and truly contemporary.

CHOICE OF DEVELOPMENT STACK

Choosing a web-based interface for this tool was a natural and strategic decision, driven by several key advantages:

- Rich Ecosystem for Dynamic Interfaces: Compared to desktop applications, web technologies offer a mature and flexible ecosystem for building complex, dynamic user interfaces. Modern frameworks and libraries make it easier than ever to create responsive, interactive experiences tailored to specific workflows.
- Easier Deployment and Maintenance: Unlike desktop applications that require installation on individual machines, web apps can be deployed centrally and accessed through any modern browser. This makes updates, bug fixes, and feature rollouts much easier to manage — ensuring all users are always working with the latest version.
- Alternative Solutions Have Limitations: While tools like R Shiny or Python Dash allow developers to build interactive web applications without deep knowledge of HTML, JavaScript, or CSS, they are better suited for data visualization and analytics dashboards rather than domain-specific tools like TLF automation.

To fully leverage the capabilities of modern web technologies, the author opted for a native web development approach. This ensures long-term maintainability, scalability, and performance, while avoiding potential technical debt that could arise from using less flexible or less extensible frameworks. React and Vue are the two leading front-end frameworks in modern web development. Prior to AMAT web app development, the author didn't have any experience in either framework. React was chosen for two main reasons:

- Developer Familiarity. The JSX/TSX approach to React aligns perfectly with the author's existing JavaScript/TypeScript expertise (for non-web development), reducing the learning curve. Its API surface is smaller and more intuitive for developers coming from non-web backgrounds.
- Ecosystem Strength. React has a vast collection of mature libraries and components that gave us confidence we could find solutions for any specialized UI challenges that might arise.

FINAL PRODUCT

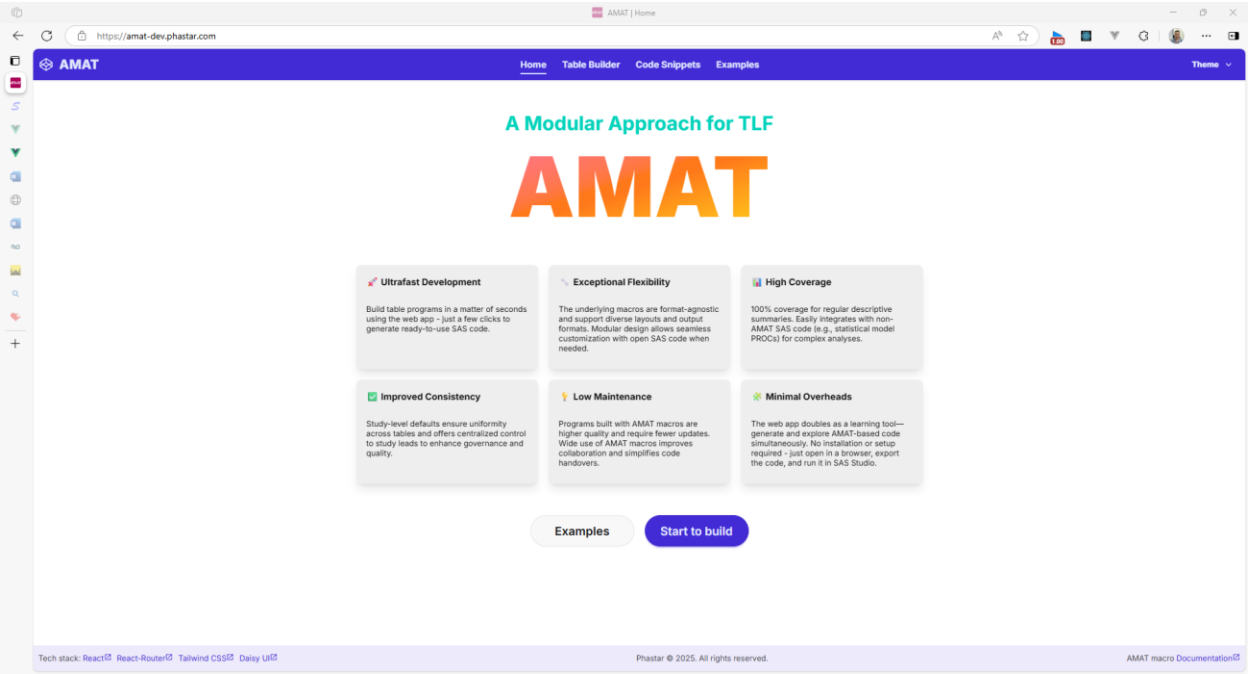

Figure 5 Lading page of AMAT web app

The final product of the AMAT web application is a front-end-only web app, designed to streamline TLF automation through an intuitive interface. It is built using the following modern web technologies:

- **React** – core frontend library.
- **React-Router (v7)** – routing, data loading, and app structure.
- **Tailwind CSS and Daisy UI** – modern, responsive UI design.
- **Vitest and Testing Library** –automated unit testing to ensure code quality and maintainability.
- **"Backend" -** Node.js (TypeScript) for logic and utilities.

WHY MAKING IT FRONT-END ONLY

The primary function of the app is to generate SAS programs based on user input from the interface. Currently, the tool does not fully automate end-to-end TLF generation; instead, it produces SAS code that users can further customize — particularly for the data preparation steps, which are often study-specific. Given this use case, a traditional backend server (typically used for database interactions) is unnecessary. All intended functionality — including real-time table rendering and exporting the shell to Word/DOCX format — runs efficiently within the browser.

By adopting a **front-end-only architecture**, several key benefits are realized:

- **Simplified Deployment**: Using React Router's build command, a production-ready bundle is generated that can be easily deployed via a static file server such as Nginx.
- **Accelerated Development Lifecycle**: In CRO environments where time and workforce resources are critical, eliminating backend dependencies significantly reduces the time required to move from development to deployment.
- **Future-Proof Flexibility**: Thanks to React Router's support for Server-Side Rendering (SSR), transitioning to a backend-integrated architecture in the future is straightforward and requires minimal refactoring.

GENERAL WORKFLOW

The AMAT web app is primarily used during the TLF development phase. As previously discussed, the app adopts a category-based approach to table building. Once programmers become familiar with the four core categories and their associated configuration options, they can navigate to the corresponding interface and use the available settings to define the structure of a table.

As users make selections, the app provides an interactive, real-time rendering of the table layout. Once satisfied with the visual representation, users can export a SAS program directly from the app.

This generated program is then placed into the study-specific folder and opened in SAS for further customization — typically involving data-processing steps that are unique to the study or sponsor requirements.

It's important to note that despite the app's rich set of configuration options, there may be cases where the desired output cannot be fully achieved through the UI alone. In such situations, users need to leverage their knowledge of the underlying AMAT SAS macros to manually refine the generated code. While this scenario occurs infrequently — thanks to the comprehensive set of built-in options — its inclusion in the workflow serves an important purpose:

- It sets clear expectations, reducing confusion when edge cases arise.
- It reflects one of the core philosophies of the app: prioritize practicality and efficiency over perfection, with the understanding that perfection is an ongoing process that evolves with user feedback and feature enhancements.

By combining guided automation with the flexibility of manual override, the AMAT web app strikes a balance between usability and control — empowering statistical programmers without constraining their expertise.

USER INTERFACE

At the heart of the app is the table builder Interface, which is organized into five main tabs:

Tab	Purpose and Scope
Generic	A flexible tab for section-based tables, where each section either presents frequency counts or descriptive statistics. Commonly used for Demographics, Disposition, Exposure Summary, etc. This is the most versatile type as it covers a wide variety of tables.
Nested	Designed for hierarchical summaries such as Adverse Events, Concomitant Medications, or Medical History tables. Also supports multi-level breakdowns such as summary of number of subjects by country and site.
Shift	Handles both single-directional and bi-directional shift tables, commonly used in laboratory data analysis.
Summary	Intended for summary of one or multiple numeric variables. Typical example includes summary of analysis value over time and change from baseline table.
Study Default	Allows users to define default layouts for treatment groups, descriptive statistics, and frequency counts/percentages. Multiple layout options can be defined per category, with one marked as the default to apply across all tables unless overridden.

Table 1 Tabs in table builder interface

Each table category offers a rich set of configuration options through intuitive UI elements.

Category	Configuration Options
Generic	• Add multiple sections with frequency or descriptive statistics • Define decimal places by statistical type (mean, median, etc.) • Customize percentage calculation methods (denominator) • Aggregate columns
Nested	• Add one or multiple nested variables (e.g. AEBODSYS, AEDEOCD) • Include sub-nested variables like AETOXGR (which can be presented horizontally or vertically) • Custom sorting per nesting level • Support EAIR (exposure-adjusted incidence rate) • Apply percentage cutoffs (e.g., only show PTs >5%) • Aggregate columns
Shift	• Add one or more summary variables (AVAL, CHG, PCHG) • Set treatment groups, summary variable and descriptive statistics as horizontal or vertical independently • Aggregate columns
Summary	• Configure values of shift from/to variables • Set treatment group, shift from/to variables as horizontal or vertical independently • Aggregate columns
Across all	• Add BY variables that are either presented as a column in tables or as subtitle • Real-time rendering example tables (shells) • Choose treatment group layout • Choose frequency count and percentage layout

	• Choose statistics layout • Export configuration as a JSON file which can be imported later

Table 2 Configuration options of each category

Here are a few screenshots of the web app.

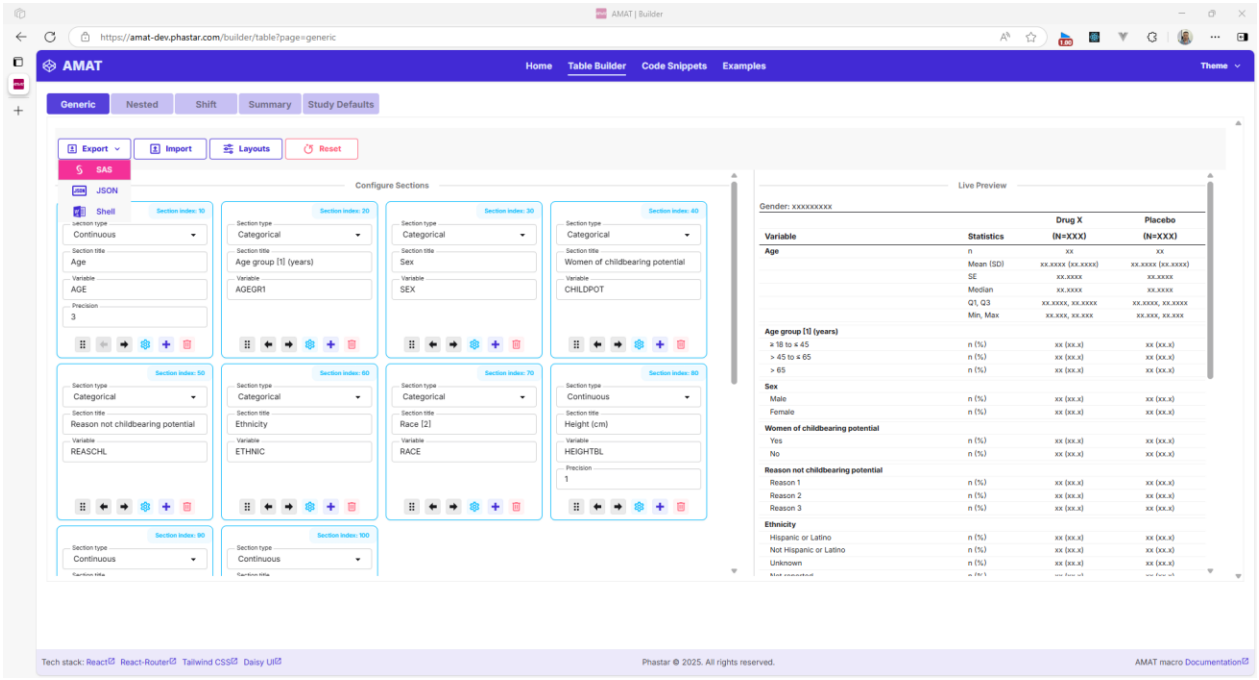

Figure 6 Generic category (demographic table)

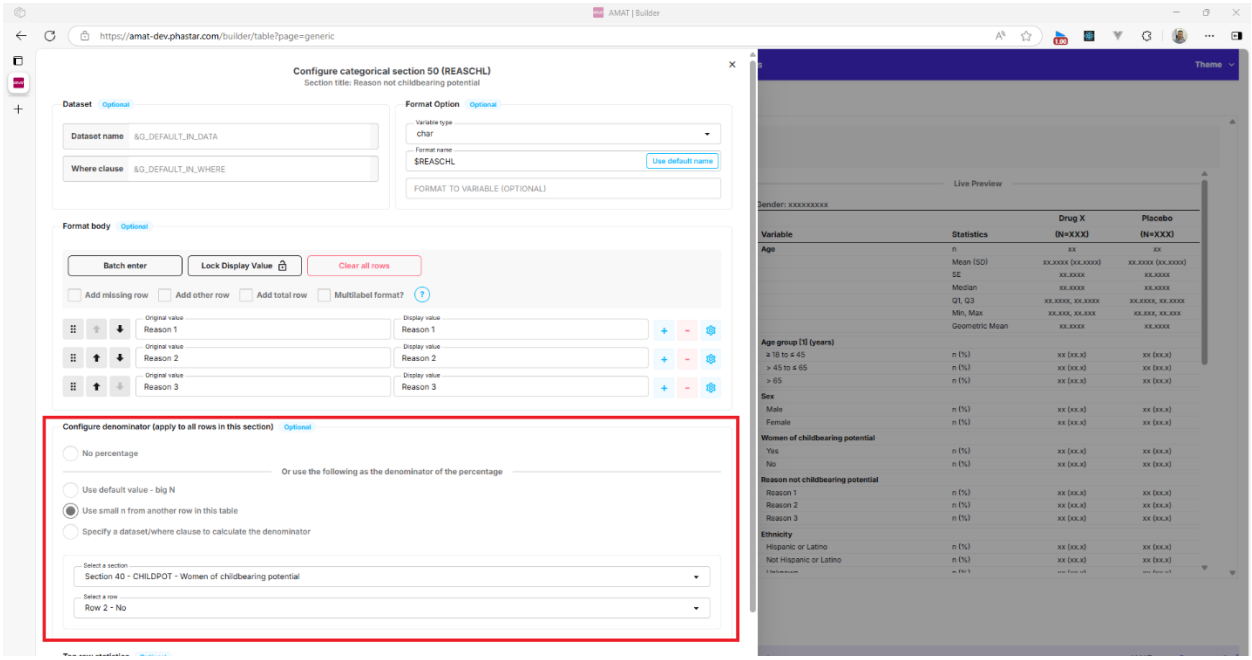

Figure 7 Set denominator of reason not childbearing to women of not childbearing potentials

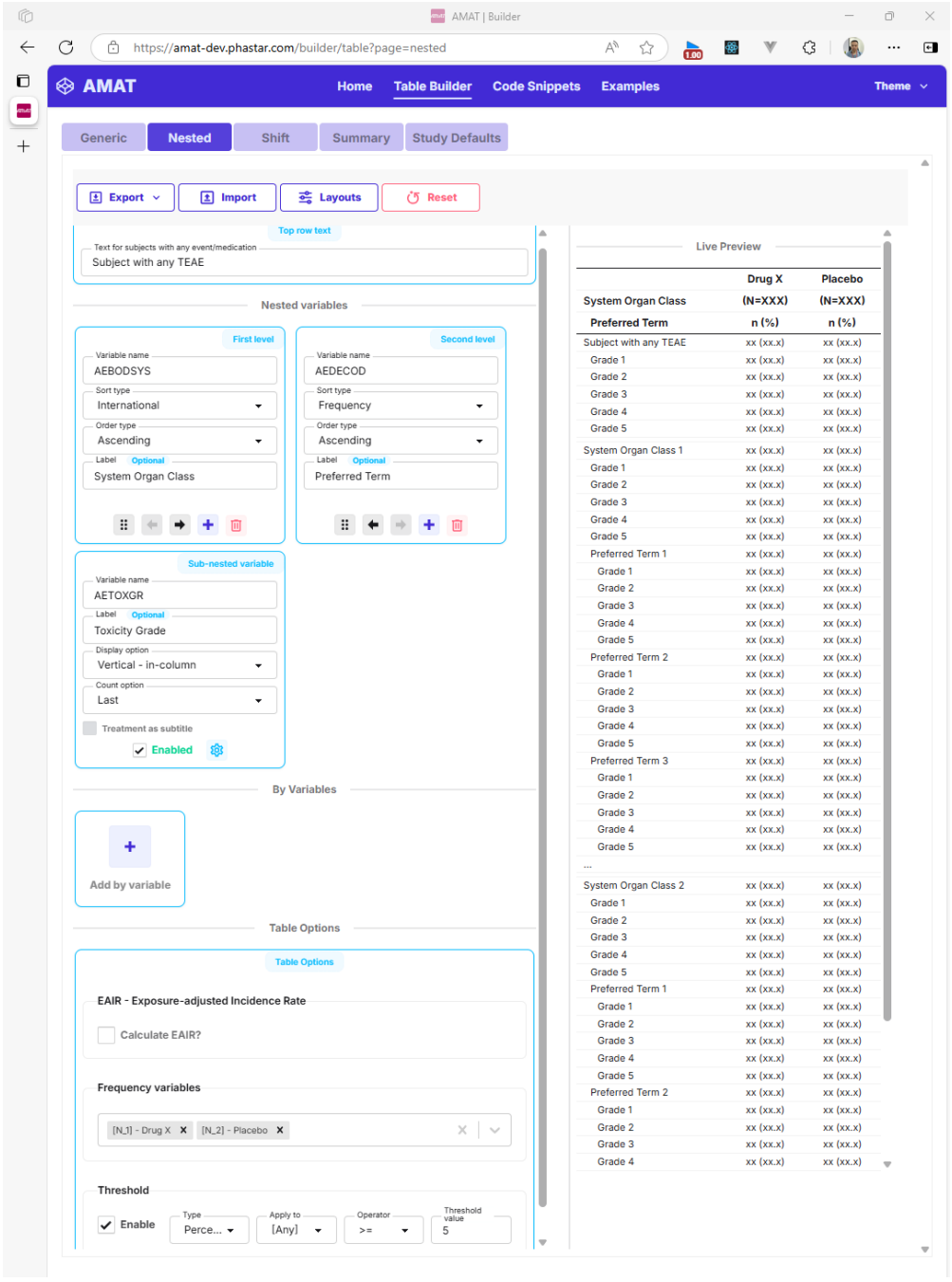

Figure 8 Nested table (TEAE by toxicity grade)

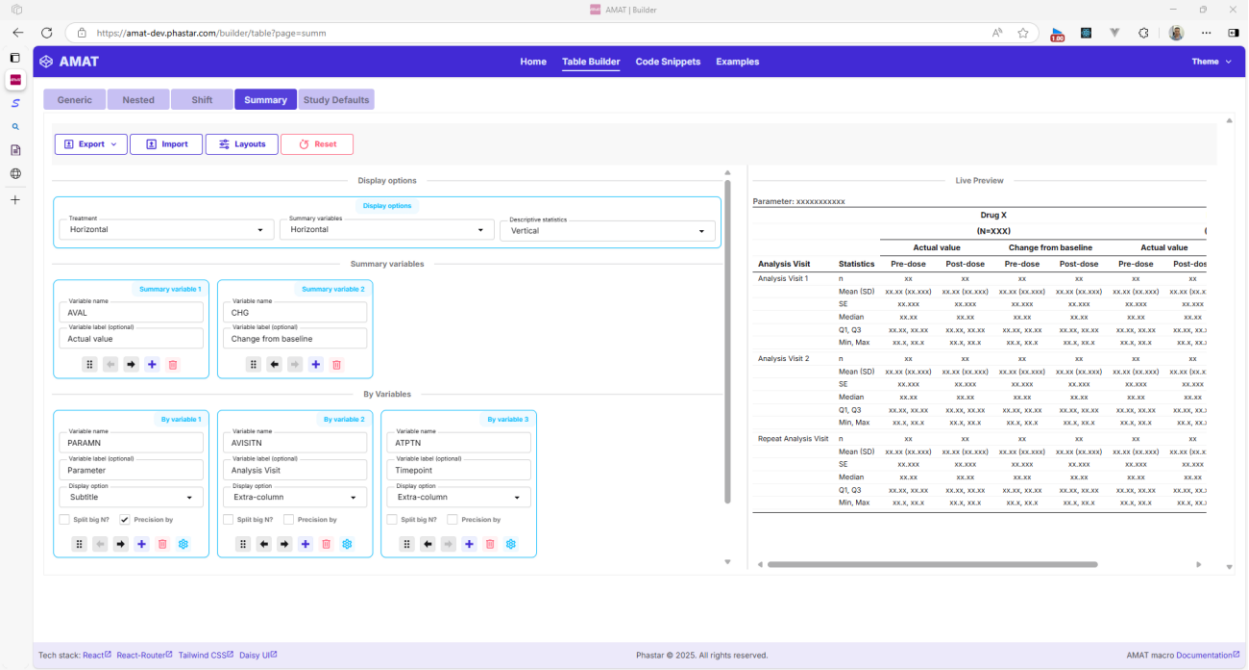

Figure 9 Summary table

Here are a few animations to showcase responsive and dynamic user interfaces:

- Customize denominator (1): https://i.postimg.cc/764Qf67N/customize-denominator-1.gif
- Customize denominator (2): https://i.postimg.cc/C14rDgsp/customize-denominator-2.gif
- Summary table: https://i.postimg.cc/5yN7SRsY/summary-table.gif
- Aggregate columns: https://i.postimg.cc/QM1y85Bt/aggregate.gif

ROADMAP OF FURTHER DEVELOPMENT

The AMAT web app is an ongoing initiative. Below are the main directions for future development:

EXPAND TABLE COVERAGE

Currently, not all table can be fully configured through the user interface. In such cases, users may need to manually change the generated SAS code to achieve the desired output. To address this, the author is continuously expanding the range of configuration options available in the UI. The goal is to support a broader variety of table shapes and layouts directly within the interface - minimizing the need for post-export customization and further streamlining the workflow.

FIGURE AND LISTINGS

At present, the AMAT web app focuses exclusively on tables. However, support for listings and figures is planned for future releases. Figure generation will offer significant efficiency gains for programmers—especially those who are less experienced with figure programming. Not only will it simplify the creation of figures, but it will also serve as an educational platform for learning how to program figures effectively.

PORTING AMAT SAS MACROS TO R

R has become increasingly popular in the pharmaceutical and clinical research industries, with many organizations actively adopting it for statistical analysis and reporting. As previously mentioned, the AMAT SAS macros function as a high-level API for structured table programming. Their design is not inherently tied to SAS. This makes them highly portable and relatively straightforward to implement in R. By developing an R version of these macros, the AMAT web app will be able to generate programs in both SAS and R, significantly broadening its applicability and appeal across different user communities.

EVOLUTION TOWARD A TLF SHELL TOOL

While current version of AMAT web app is not a TLF shell tool, its real-time rendering capability already provides a solid foundation for one. The long-term vision includes extending this functionality to support the creation and management of TLF shells. Imagine being able to define the structure of your deliverables early in the process—before ADaM datasets are even available—while still generating much of the underlying program logic automatically.
This aligns perfectly with AMAT's core philosophy of separating data processing logic from summary and presentation logic, making it ideal for early-stage TLF planning and development.

CONCLUSIONS

The AMAT web app exemplifies the seamless integration of modern web development technologies with traditional SAS macro programming. It goes beyond a basic interface where users merely fill in text boxes or select from drop-down lists. Instead, it leverages the power of reactivity—a key feature of modern web frameworks—to deliver capabilities such as real-time rendering of TLF shells. This results in an unprecedented level of intuitiveness and ease of use.

This dynamic interface not only boosts productivity but also lays a solid foundation for evolving into a comprehensive TLF shell tool in the future. Such a tool would support early-stage planning and iterative development of TLFs, significantly enhancing overall workflow efficiency. The AMAT case study illustrates how thoughtful user interface design can dramatically enhance the effectiveness of a technical tool.

Behind this user-friendly front end, the robust AMAT SAS macros provide a solid framework for automating TLF programming, greatly simplifying the logic required to generate SAS code through the web app. Importantly, these macros are designed with portability in mind. Their underlying architecture allows them to be adapted for use with other statistical programming languages such as R, ensuring that the AMAT Web App is not tied exclusively to SAS. This flexibility ensures its long-term relevance and adaptability across various analytical environments.

In summary, the AMAT web app demonstrates how combining modern web capabilities with proven SAS macro techniques can result in a powerful, forward-looking tool that enhances efficiency, lowers technical barriers, and supports a wide range of users in the clinical and pharmaceutical domains.

AI AND DEVELOPMENT PROCESS

Like many other modern software projects, artificial intelligence has played a supportive role in the development of the AMAT web app. Throughout the project lifecycle, AI tools have been used to enhance productivity and code quality.

During the development process, the author transitioned from VS Code to Trae (a frontend IDE focused on AI integration) to take advantage of improved AI-assisted coding features. This switch significantly enhanced the overall development experience.

Key applications of AI in this project include:

- Code completion: AI-powered suggestions helped accelerate development by reducing repetitive typing and guiding implementation patterns.
- Commit message generation: AI helped in drafting clear, descriptive commit messages, improving version control documentation.

- Potential bug detection: AI was used to find possible logic issues or edge cases early in the development cycle, contributing to more robust code.

By integrating AI into the development workflow, the AMAT project benefits from increased efficiency, cleaner code, and a smoother developer experience - proving how modern tooling can complement traditional programming practices.

THE EVOLVING ROLE OF STATISTICAL PROGRAMMERS IN A CHANGING LANDSCAPE

For a long time, "statistical programmer" has been nearly synonymous with "SAS programmer." However, as the field continues to evolve, more professionals are adopting general-purpose languages like Python and embracing modern software development practices to build tools that go far beyond the scope of traditional SAS workflows.

We now live in an era where generative AI and rapidly advancing web technologies have made tool development more accessible than ever before. At the same time, this progress brings challenges — including information overload and increasing competition from professional developers entering the clinical data space.

In this shifting landscape, the core value of a statistical programmer lies not just in technical ability, but in industry experience, mastery of CDISC standards, and a deep understanding of the clinical trials. While not every programmer needs to become a tool builder, those who do can use their domain knowledge to create meaningful solutions — an advantage that technical skills alone cannot replicate.

Ultimately, this evolution should not be seen as a threat, but as an opportunity. The future belongs to those who are willing to embrace modern technologies, while staying grounded with the domain knowledge that defines the statistical programming profession.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Yong (Ken) Cao
Phastar
Yong.Cao@phastar.com
WeChat: kenwdcao