

Mesop Visual Editor v2





Overview

Provide a way to intuitively edit Mesop UI through a combination of WYSIWYG editing and AI prompting.

This doc doesn't get into the specifics of how we would call an LLM, but presumably we could eventually integrate with Gemini API and call a model like Gemini Flash (with sufficient prompt context about Mesop) to do this.

Design

The visual editor will be composed of the following components:

1. **Floating toolbar** (*new*) - This would be floating in the main app frame and include the following commands:
 - a.  **Single select** - select a single component
 - b.  **Multi select** - select multiple components
 - c.  **Prompt** - you can type a prompt to modify the app (not component-specific)
 - d.  **Add component** - add a new component onto the page.
 - i. Open question of where the component gets added. You could put it at the bottom of the page, but it might look funky.
2. **Editor sidenav** - This is the existing right-hand pane, updated with more functionalities.
3. **Component overlay** - This exists and is the box highlighting the selected component.
 - a. Functionalities:
 - i. **Add component** - either as a child for content components or as a sibling for all components.
 - ii. **Delete component.**
4. **Command palette** - It'll also be nice to have a command palette (e.g. command-K/P) which allows you to activate any of these commands as well as do some of the [heuristic commands](#) mentioned below.
5. **Diff viewer** - Once we have AI integration, being able to view & approve the changes by the model in a diff viewer will be important.

- a. It'll also be nice to have a way to “undo” a change, essentially having revisions that you can toggle between. I’m not sure if this would be native integration with a SCM (e.g. git) or a built-in version control done in Mesop Editor.

Screenshot of today’s component overlay:



User journeys

Inserting content

When you’re first creating an app or adding new functionality to an existing app, there’s typically one of three kinds of things that you will want to add in the UI, from largest to smallest building block:

- **Page templates**
 - Examples:
 - Common layouts:
 - header + sidebar + footer
 - Two columns
 - Use-case based:
 - Home page (e.g. starter kit / DuoChat codelab)
 - Chat/conversation page
- **Pattern**
 - Examples:
 - Grid table
 - Dialog
 - Toast
- **Component**

- Examples:
 - Web components wrapping third-party JS libraries (e.g. Plotly)
 - chat component callsite / boilerplate

Registry

It will be good to have some kind of registry for these different kinds of building blocks. This way you can add your own components to the registry and use it from the visual editor. I think there will be a few tiers:

- **Official**
 - These are building blocks in the main Mesop repo and will be available by default in the Visual Editor.
- **Community** (published)
 - These are blocks that the Mesop core team has seen (but does not directly maintain) and will be searchable from the Visual Editor. You can select a block, which will pull down the block into your local workspace.
- **Personal** (unpublished)
 - These are blocks that may be in your personal repos. You should be able to add them in (e.g. configure in Visual Editor settings like a JSON/py file) which will make them available in the visual editor.

Polishing existing content

The other kind of user journey is updating and iterating on an existing UI:

- **Layout & spacing**
 - Adding padding and margin
 - Center a component (vertically, horizontally)
 - Make a set of components into a row
- **Styling**
 - Change the font size, font family
 - Add box shadow to a container

LLM vs. Heuristics

The visual editor should be useful without AI because there will be use cases where an LLM cannot be used feasibly (e.g. data privacy, costs, etc.) and we'd like to provide as much core functionality for this offline use case.

For example, things like adding padding and margin can be done without LLMs and we can make it easy to space things out nicely (e.g. make adding `me.Padding.all(16)` a simple click in the UI).

Open questions

So far the visual editor has focused on the UI-heavy aspects of development and left out the business logic / wiring pieces, such as writing event handlers and state classes. I think these won't benefit too much from a visual editor, but we could imagine having a simple text editor to make quick changes for these pieces, assisted by AI.