Welcome to shiny.fluent

Shiny.fluent ports Fluent UI to R using shiny.react machinery, giving powerful, professional, and user-friendly interfaces to the Shiny community. In this tutorial we will walk you through how to create a dashboard for a Fluent UI app to improve upon a simple sales analysis application. If you haven't already, we recommend you start your shiny.fluent adventure here to familiarize yourself with using Fluent components – the tutorial below will build on what you learn there.

Getting Started

Shiny.react was created with ease-of-use in mind for R developers. Inputs are as close as possible to the Shiny API and the react package(s) documentation is provided inside the R documentation system. To begin use of shiny.fluent we will install the shiny.react and shiny.fluent packages.

Installation

To install the packages, run:

Prerequisites

With shiny.react and shiny.fluent packages installed, let's load the libraries needed for this example.

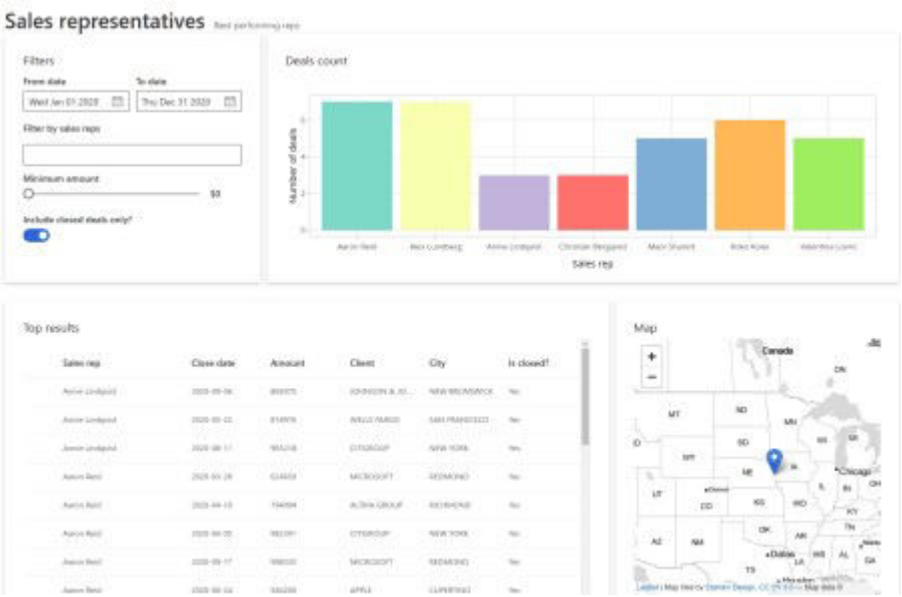
Creating a shiny.fluent dashboard

In this tutorial we will walk through how to build a dashboard UI.

Single Page Layout

As a next step, let's add a title and subtitle to our current app. We'll create a helper function and call it makePage, so that it is easy to add more pages in the same fashion.

We can now take our entire UI built so far and put it in a "page" layout, giving it a helpful title and subtitle.



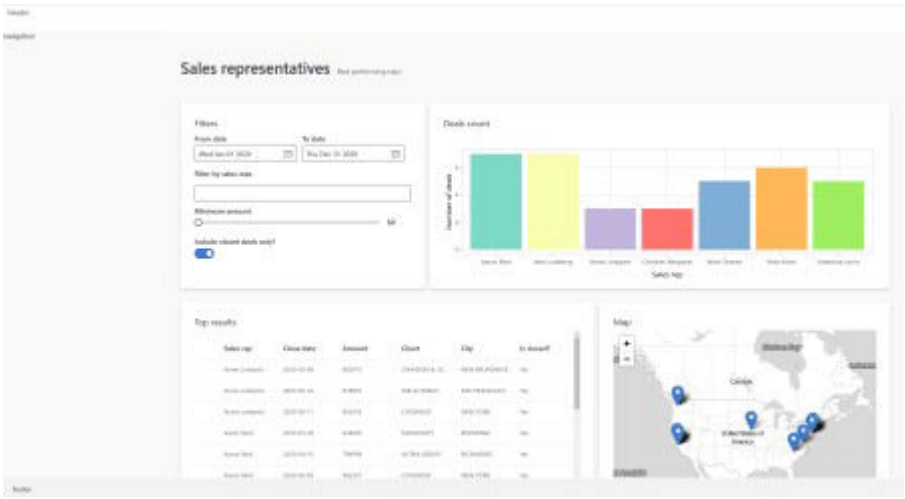
Dashboard Layout

It's time to create a place for our header, navigation sidebar, and footer. We'll use CSS grid for that. It's a modern, flexible and straightforward way to achieve such a layout.
We start by creating divs for each of the areas, with placeholder texts that we will later replace.

Now it's time to tell the browser using CSS how to arrange these areas. To define how our areas should be laid out on the page, let's put the following rules in www/style.css:

We can also use this opportunity to add some additional styling for the entire page, and add the following rules to the same file:

Now we only need to update our UI definition to load styles from www/style.css and use the new layout.



Filling All the Areas

Moving forward, we may now begin adding a Header, Footer, and Sidebar to the empty areas of our dashboard.

Header

Let's replace the previous header definition with:

As you can see, we're using CommandBar and CommandBarItem from shiny.fluent. We also need to add a bit of styling to our CSS file:

Navigation in the Sidebar

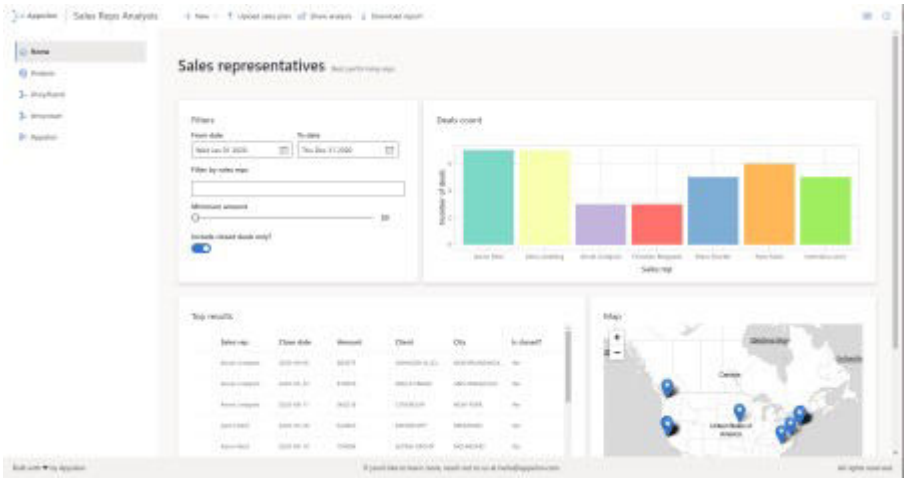
Nav is a very powerful component from Fluent UI. It has very rich configuration options, but we will use it to show just a couple of links:

Footer

Footer is relatively straightforward – we can put anything we want there. Here we use Text for

typography (setting uniform font styling). We also use Stack to arrange elements horizontally and with bigger gaps.

Let's see how this looks together.



Additional Pages

The final step is to add pages and enable navigation between them.

Home Page

Let's make a home page, consisting of two cards with some welcome text.

If we replace analysis_page with home_page in our ui, we can see this page. However, there's one problem: we don't have a way to switch between pages! This is where so-called page routing comes into play.

Adding shiny.router

To enable switching between pages we will use the shiny.router package. This way we will also have shareable URLs to individual pages.

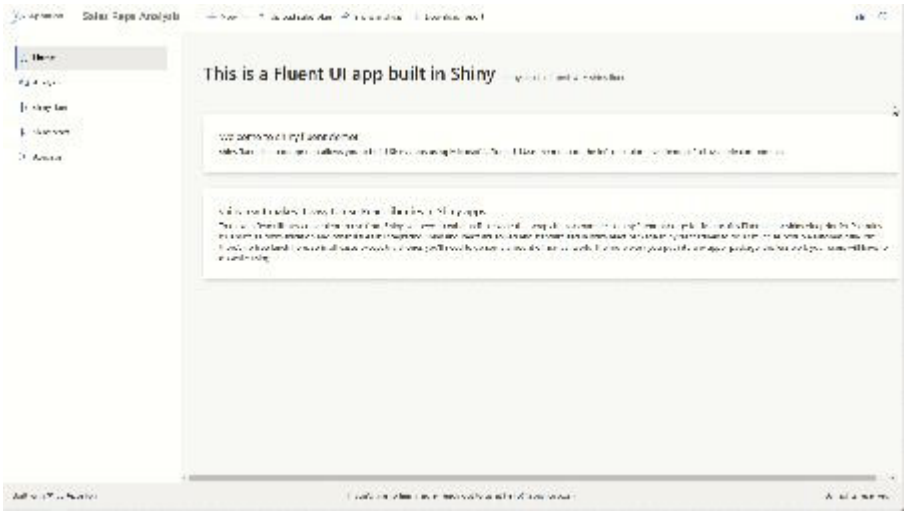
The first step is to define the available routes:

Now, we need to put router$ui in the place where we want the selected page to appear. (Currently, we also need to manually include router's JavaScript dependencies, because shiny.fluent is not compatible with the way shiny.router loads this dependency. We expect this to be resolved in future versions of shiny.router).

One final step is to add this single line to our server function, which otherwise remains untouched from the Part 1 of the tutorial.

That's it!

And there you go! We now have styled a shiny.fluent app into a solid dashboard layout. Here's the final result:



Conclusion

The speed at which you can now add user-friendly and ubiquitous Microsoft product elements to your project is quite impressive. In a matter of minutes, we were able to create a functional, professional-looking Shiny dashboard with shiny.fluent. And if your users are already familiar with Microsoft products, adoption can be seamless.

There is more complexity over something like shiny.semantic. However, an intermediate level of knowledge of Shiny is sufficient to begin. Component examples are provided and more comprehensive documentation can be found in the official Fluent UI docs or by typing shiny.fluent:: (e.g., '?shiny.fluent::MyComponentName').