Application layout guide

LAST UPDATED: 01 FEB 2021
BY: JJ ALLAIRE

Overview

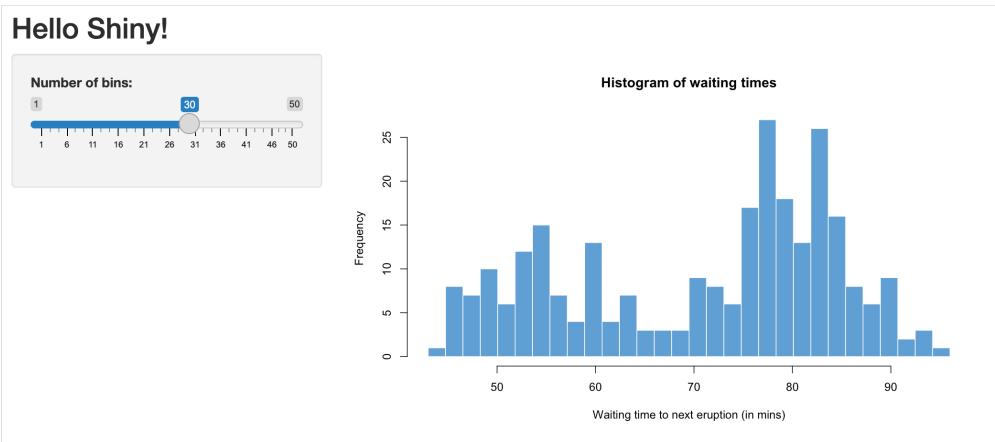
Shiny includes a number of facilities for laying out the components of an application. This guide describes the following application layout features:

1. A `sidebarLayout()` : for placing a `sidebarPanel()` of inputs alongside a `mainPanel()` output content.
2. Custom layouts using Shiny's grid layout system (i.e., `fluidRow()` & `column()`).
3. Segmenting layouts using the `tabsetPanel()` and `navlistPanel()` functions.
4. Creating applications with multiple top-level components using the `navbarPage()` function.

All these features are made available via Bootstrap (https://getbootstrap.com/), an extremely popular HTML/CSS framework (though no prior experience with Bootstrap is assumed). Shiny currently defaults to Bootstrap 3. To upgrade to a more recent version and/or implement custom Bootstrap themes, check out the application themes article (themes.html).

Sidebar Layout

The sidebar layout is a useful starting point for most applications. This layout provides a sidebar for inputs and a large main area for output:



Here's the code used to create this layout:

```
ui <- fluidPage(
  titlePanel("Hello Shiny!"),

  sidebarLayout(

    sidebarPanel(
      sliderInput(
        "bins", label = "Number of bins:",
        min = 1, value = 30, max = 50
      )
    ),

    mainPanel(
      plotOutput("distPlot")
    )
  )
)
```


Note that the sidebar can be positioned to the left (the default) or right of the main area. For example, to position the sidebar to the right you would use this code:

```
sidebarLayout(position = "right",

  sidebarPanel(
    # Inputs excluded for brevity
  ),
  mainPanel(
    # Outputs excluded for brevity
  )
)
```

Grid Layout

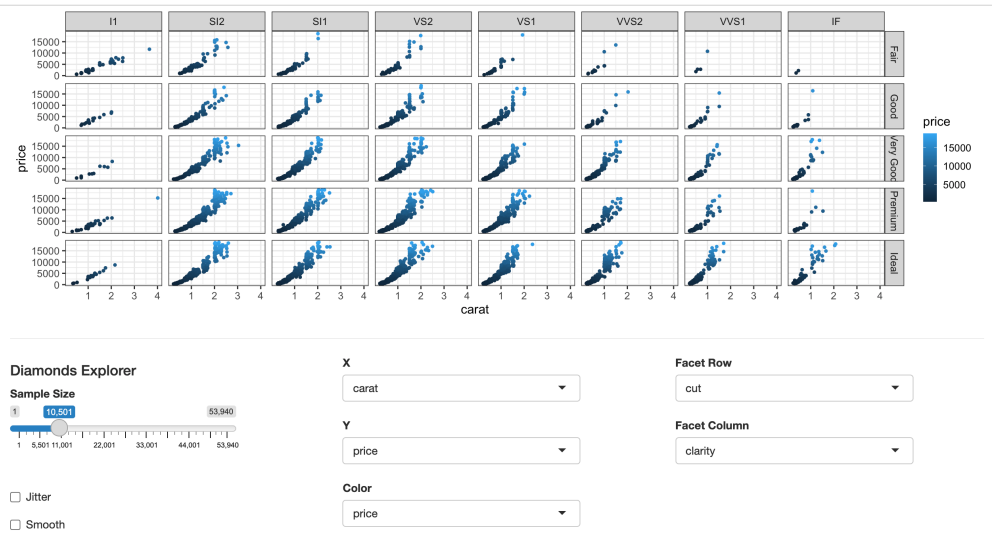
The familiar `sidebarLayout()` described above makes use of Shiny's lower-level grid layout functions. Rows are created by the `fluidRow()` function and include columns defined by the `column()` function. Column widths are based on the Bootstrap 12-wide grid system, so should add up to 12 within a `fluidRow()` container.

To illustrate, here's the sidebar layout implemented using the `fluidRow()`, `column()` and `wellPanel()` functions:

```
fluidPage(
  titlePanel("Hello Shiny!"),

  fluidRow(

    column(4,
      wellPanel(
        sliderInput(
          "bins", label = "Number of bins:",
          min = 1, value = 30, max = 50
        )
      )
    ),

    column(8,
      plotOutput("distPlot")
    )
  )
)
```

The first parameter to the `column()` function is it's width (out of a total of 12 columns). It's also possible to offset the position of columns to achieve more precise control over the location of UI elements. You can move columns to the right by adding the `offset` parameter to the `column()` function. Each unit of offset increases the left-margin of a column by a whole column.

Here's an example of a UI with a plot at the top and three columns at the bottom that contain the inputs that drive the plot:



The code required to implement this UI is as follows:

```
library(shiny)
library(ggplot2)
dataset <- diamonds

ui <- fluidPage(

  title = "Diamonds Explorer",

  plotOutput('plot'),

  hr(),

  fluidRow(
    column(3,
      h4("Diamonds Explorer"),
      sliderInput('sampleSize', 'Sample Size',
                  min=1, max=nrow(dataset), value=min(1000,
nrow(dataset)),
                  step=500, round=0),
      br(),
      checkboxInput('jitter', 'Jitter'),
      checkboxInput('smooth', 'Smooth')
    ),
    column(4, offset = 1,
      selectInput('x', 'X', names(dataset)),
      selectInput('y', 'Y', names(dataset), names(dataset)[[2]]),
      selectInput('color', 'Color', c('None', names(dataset)))
    ),
    column(4,
      selectInput('facet_row', 'Facet Row', c(None='.', names(dataset))),
      selectInput('facet_col', 'Facet Column', c(None='.',
names(dataset)))
    )
  )
)
```

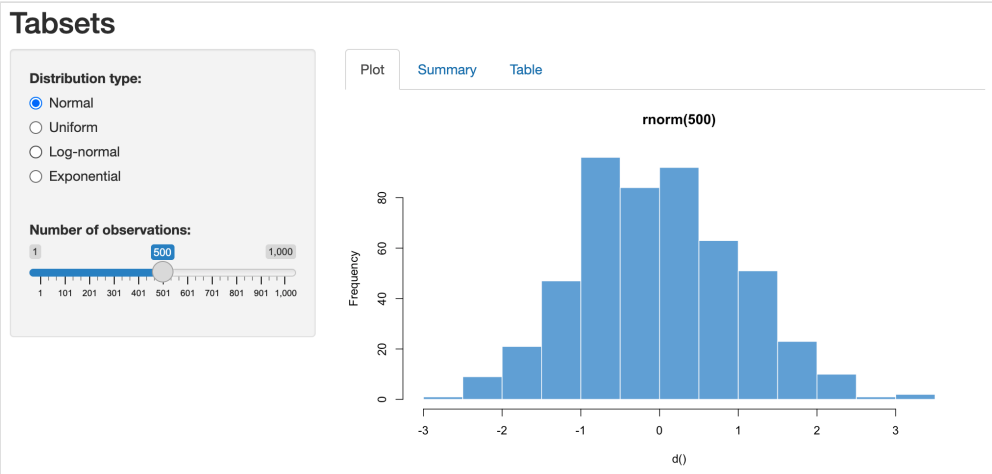
There are a few important things to note here:

1. The inputs are at the bottom and broken into three columns of varying widths.
2. The `offset` parameter is used on the center input column to provide custom spacing between the first and second columns.
3. The page doesn't include a `titlePanel()` so the title is specified as an explicit argument to `fluidPage()`.

Grid layouts can be used anywhere within a `fluidPage()` and can even be nested within each other. You can find out more about grid layouts in the Grid Layouts in Depth section below.

Tabsets

Often applications need to subdivide their user-interface into discrete sections. This can be accomplished using the `tabsetPanel()` function. For example:



The code required to create this UI is:

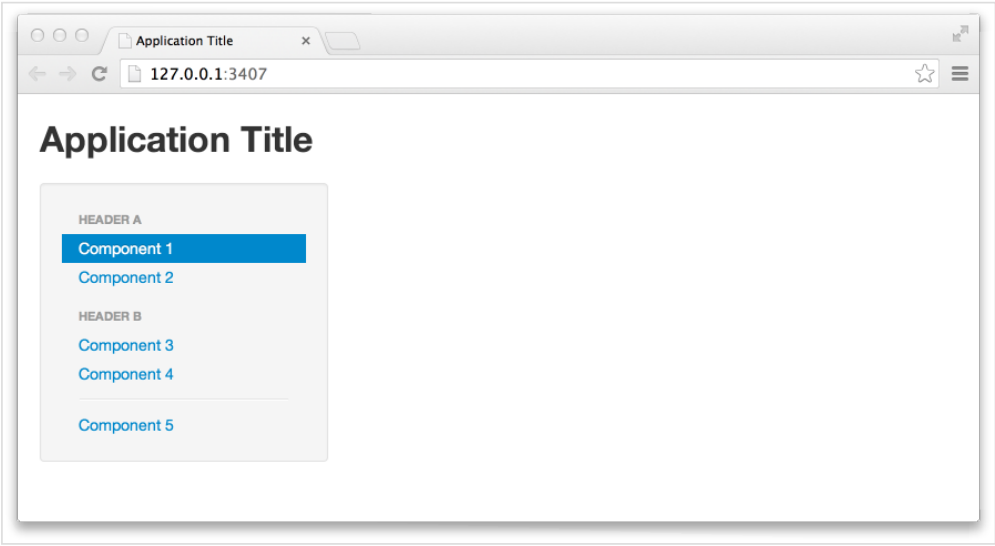
```
ui <- fluidPage(

  titlePanel("Tabsets"),

  sidebarLayout(

    sidebarPanel(
      # Inputs excluded for brevity
    ),

    mainPanel(
      tabsetPanel(
        tabPanel("Plot", plotOutput("plot")),
        tabPanel("Summary", verbatimTextOutput("summary")),
        tabPanel("Table", tableOutput("table"))
      )
    )
  )
)
```

Tabs can be located above (the default), below, left, or to the right of tab content. For example, to position the tabs below the tab content you would use this code:

```
tabsetPanel(position = "below",
  tabPanel("Plot", plotOutput("plot")),
  tabPanel("Summary", verbatimTextOutput("summary")),
  tabPanel("Table", tableOutput("table"))
)
```

Navlists

When you have more than a handful of tabPanels the `navlistPanel()` may be a good alternative to `tabsetPanel()`. A navlist presents the various components as a sidebar list rather than using tabs. It also supports section heading and separators for longer lists. Here's an example of a `navlistPanel()`:

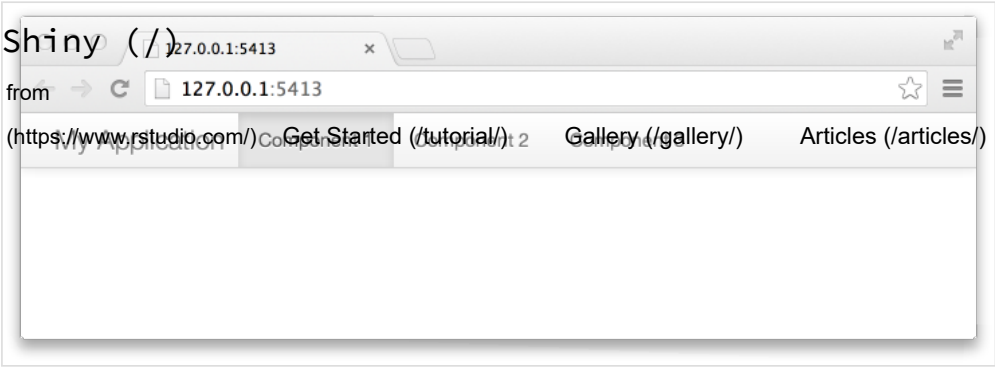


The code required to implement this is as follows (note that the tabPanels are empty to keep the example uncluttered, typically they'd include additional UI elements):

```
ui <- fluidPage(

  titlePanel("Application Title"),

  navlistPanel(
    "Header A",
    tabPanel("Component 1"),
    tabPanel("Component 2"),
    "Header B",
    tabPanel("Component 3"),
    tabPanel("Component 4"),
    "-----",
    tabPanel("Component 5")
  )
)
```

Navbar Pages

You may want to create a Shiny application that consists of multiple distinct sub-components (each with their own sidebar, tabsets, or other layout constructs). The `navbarPage()` function creates an application with a standard Bootstrap Navbar (http://getbootstrap.com/components/#navbar) at the top. For example:

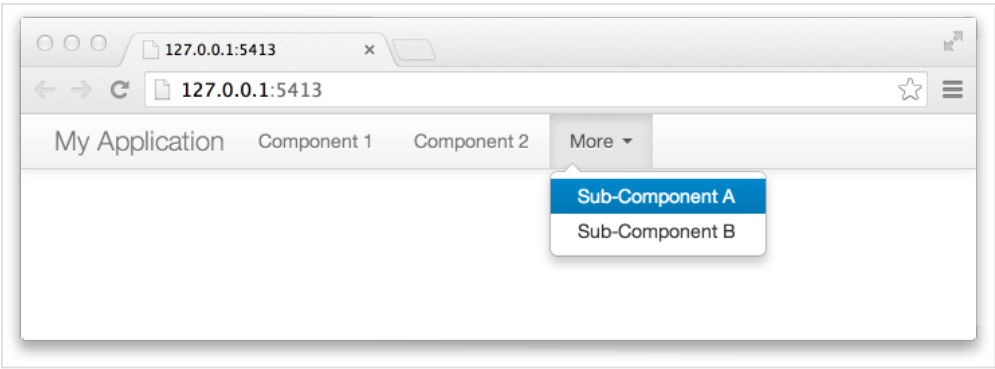



```
ui <- navbarPage("My Application",
  tabPanel("Component 1"),
  tabPanel("Component 2"),
  tabPanel("Component 3")
)
```

Note that the Shiny `tabPanel()` is used to specify the navigable components.

Secondary Navigation

You can add a second level of navigation to the page by using the `navbarMenu()` function. This adds a menu to the top level navbar which can in turn refer to additional tabPanels.




```
ui <- navbarPage("My Application",
  tabPanel("Component 1"),
  tabPanel("Component 2"),
  navbarMenu("More",
    tabPanel("Sub-Component A"),
    tabPanel("Sub-Component B"))
)
```

Additional Options

There are several other arguments to `navbarPage()` that provide additional measures of customization:

Argument	Description
header	Tag of list of tags to display as a common header above all tabPanels.
footer	Tag or list of tags to display as a common footer below all tabPanels

Argument	Description
inverse	`TRUE` to use a dark background and light text for the navigation bar
collapsable	`TRUE` to automatically collapse the navigation elements into a menu when the width of the browser is less than 940 pixels (useful for viewing on smaller touchscreen device)

Grid Layouts in Depth

There are two types of Bootstrap grids, fluid and fixed. The examples so far have used the fluid grid system exclusively and that's the system that's recommended for most applications (and the default for Shiny functions like `navbarPage()` and `sidebarLayout()`).

Both grid systems use a flexibly sub-dividable 12-column grid for layout. The fluid system always occupies the full width of the web page and re-sizes it's components dynamically as the size of the page changes. The fixed system occupies a fixed width of 940 pixels by default and may assume other widths when Bootstrap responsive layout kicks in (e.g. when on a tablet).

The following sections are a translation of the official Bootstrap 3 grid system (http://getbootstrap.com/css/#grid) documentation, with HTML code replaced by R code.

Fluid Grid System

The Bootstrap grid system utilizes 12 columns which can be flexibly subdivided into rows and columns. To create a layout based on the fluid system you use the `fluidPage()` function. To create rows within the grid you use the `fluidRow()` function; to create columns within rows you use the `column()` function.

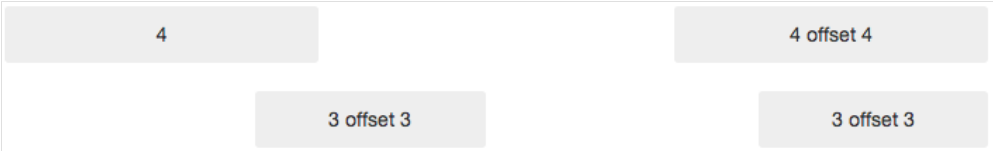
For example, consider this high level page layout (the numbers displayed are columns out of a total of 12):

To create this layout in a Shiny application you'd use the following code (note that the column widths within the fluid row add up to 12):

```
ui <- fluidPage(
  fluidRow(
    column(2,
      "sidebar"
    ),
    column(10,
      "main"
    )
  )
)
```


Column Offsetting

It's also possible to offset the position of columns to achieve more precise control over the location of UI elements. Move columns to the right by adding the `offset` parameter to the `column()` function. Each unit of offset increases the left-margin of a column by a whole column. Consider this layout:

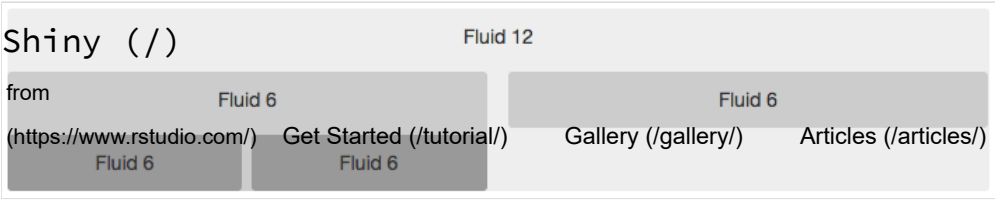


To create this layout in a Shiny application you'd using the following code:

```
ui <- fluidPage(
  fluidRow(
    column(4,
      "4"
    ),
    column(4, offset = 4,
      "4 offset 4"
    )
  ),
  fluidRow(
    column(3, offset = 3,
      "3 offset 3"
    ),
    column(3, offset = 3,
      "3 offset 3"
    )
  )
)
```

Column Nesting

When you nest columns within a fluid grid, each nested level of columns should add up to 12 columns. This is because the fluid grid uses percentages, not pixels, for setting widths. Consider this page layout:



To create this layout in a Shiny application you'd use the following code:

```
ui <- fluidPage(
  fluidRow(
    column(12,
      "Fluid 12",
      fluidRow(
        column(6,
          "Fluid 6",
          fluidRow(
            column(6,
              "Fluid 6"),
            column(6,
              "Fluid 6")
          )
        ),
        column(width = 6,
          "Fluid 6")
      )
    )
  )
)
```

Note that each time a `fluidRow()` is introduced the columns within the row add up to 12.

Fixed Grid System

The fixed grid system also utilizes 12 columns, and maintains a fixed width of 940 pixels by default. If Bootstrap responsive features are enabled (they are by default in Shiny) then the grid will also adapt to be 724px or 1170px wide depending on your viewport (e.g. when on a tablet).

The main benefit of a fixed grid is that it provides stronger guarantees about how users will see the various elements of your UI laid out (this is because it's not being dynamically laid out according to the width of the browser). The main drawback is that it's a bit more complex to work with. In general we recommend using fluid grids unless you absolutely require the lower level layout control afforded by a fixed grid.

Using Fixed Grids

Using fixed grids in Shiny works almost identically to fluid grids. Here are the differences to keep in mind:

1. You use the `fixedPage()` and `fixedRow()` functions to build the grid.

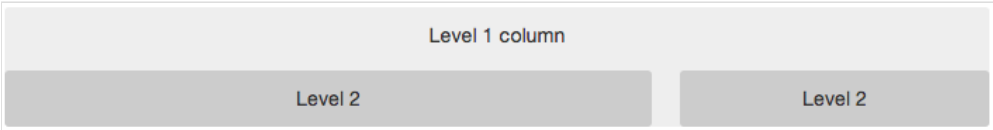
2. Rows can nest, but should always include a set of columns that add up to the number of columns of their parent (rather than resetting to 12 at each nesting level as they do in fluid grids).

Here's the code for a fixed grid version of the simple sidebar layout shown earlier:

```
ui <- fixedPage(
  fixedRow(
    column(2,
      "sidebar"
    ),
    column(10,
      "main"
    )
  )
)
```

Column Nesting

In fixed grids the width of each nested column must add up to the number of columns in their parent. Here's a `fixedRow()` with a 9-wide column that contains two other columns of width 6 and 3:



The create this row within a Shiny application you'd use the following code:

```
fixedRow(
  column(9,
    "Level 1 column",
    fixedRow(
      column(6,
        "Level 2"
      ),
      column(3,
        "Level 2"
      )
    )
  )
)
```

Note that the total size of the nested columns is 9, the same as their parent column.

Responsive Layout

The Bootstrap grid system supports responsive CSS, which enables your application to automatically adapt its layout for viewing on different sized devices. Responsive layout includes the following:

1. Modifying the width of columns in the grid
2. Stack elements instead of float wherever necessary
3. Resize headings and text to be more appropriate for devices

Responsive layout is enabled by default for all Shiny page types. To disable responsive layout you should pass `responsive = FALSE` to the `fluidPage()` or `fixedPage()` function.

Supported Devices

When responsive layout is enabled here is how the Bootstrap grid system adapts to various devices:

	Layout width	Column width	Gutter width
Large display	1200px and up	70px	30px
Default	980px and up	60px	20px
Portrait tablets	768px and above	42px	20px
Phones to tablets	767px and below	Fluid (no fixed widths)	Fluid (no fixed widths)
Phones	480px and below	Fluid (no fixed widths)	Fluid (no fixed widths)

Note that on smaller screen sizes fluid columns widths are used automatically even if the page uses fixed grid layout.

If you have questions about this article or would like to discuss ideas presented here, please post on RStudio Community (https://community.rstudio.com/c/shiny). Our developers monitor these forums and answer questions periodically. See help (/help) for more help with all things Shiny.