DASHBOARDS

Shiny from RStudio™

OUTLINE

- Dashboards
  - What is in a dashboard?
  - Server
    - reactiveFileReader
    - reactivePoll
  - UI
    - Static vs. dynamic dashboards
    - flexdashboard
    - Shiny pre-rendered
  - shinydashboard
    - Body
    - Menu
    - Header

DASHBOARDS

Shiny from RStudio™

What is in a dashboard?

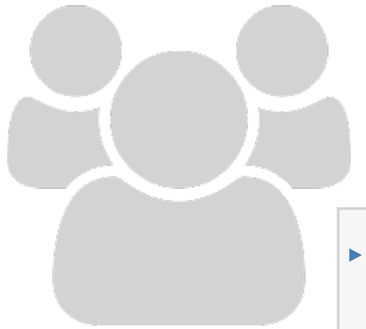
DASHBOARDS

- Automatically updating
  - Not just based on user gestures
  - But also when data source changes
- Many viewers looking at the same data
- May or may not be interactive

Server

MOTIVATION

- You have new data coming in — constantly, continuously, or on a schedule
- When new data comes in, it's automatically received, and transformed, aggregated, summarized, etc.
- May want to call attention to exceptional results



EXERCISE

- Why might this not be a good idea?

```
dataset <- reactive({
  result <- read.csv(“data.csv")
  invalidateLater(5000)
  result
})

output$plot <- renderPlot({
  plot(dataset()) # or whatever
})
```

SOLUTION

Lots of overhead!

reactiveFileReader

REACTIVEFILEREADER

- Reads the given file (`"data.csv"`) using the given function (`read.csv`)
- Periodically reads the last-modified time of the file
- If the timestamp changes, then (and only then) re-reads the file

Single file, on disk
(not database or web API)

```
dataset <- reactiveFileReader(
  intervalMillis = 1000,
  session = session,
  filePath = "data.csv",
  readFunc = read.csv
)

output$plot <- renderPlot({
  plot(dataset()) # or whatever
})
```

Must have data path as first argument

REACTIVEFILEREADER

```
dataset <- reactiveFileReader(
  intervalMillis = 1000,
  session = session,
  filePath = "data.csv",
  readFunc = read.csv,
  stringsAsFactors = FALSE
)

output$plot <- renderPlot({
  plot(dataset()) # or whatever
})
```

Add any named arguments

reactivePoll

REACTIVEPOLL

- `reactiveFileReader` is limited to files on disk. It doesn't work for non-file-based data sources like databases or web APIs
- `reactivePoll` is a generalization of reactiveFileReader
  - `checkFunc`: A function that can execute quickly, and merely determine if anything has changed
    - Should be fast as it will block the R process while it runs! The slower it is, the greater you should make the polling interval.
    - Should not return TRUE or FALSE for changed/unchanged. Instead, just return a value (like the timestamp, or the count); it's `reactivePoll`'s job, not yours, to keep track of whether that value is the same as the previous value or not.
  - `valueFunc`: A function with the (potentially expensive) logic for actually reading the data

EXERCISE

- When might we want to use reactivePoll on dashboards?

SOLUTION

When we are pulling from a database or Web API!

```
QueriedData <- reactivePoll(30000, session,
   # This function checks the rows and when the rows are higher than previously, in those cases it reads the table
  checkFunc = function(){
    # connect
    con <- poolCheckout(mysqldb)
    # Return the current numbers of rows in mysqltable
    rowcount <- dbGetQuery(con, "SHOW TABLE STATUS;") %>% filter(Name == "mysqltable") %>% pull(Rows)
 # disconnect database
 poolReturn(con
    },
  valueFunc = function() {
 # connect
    con <- poolCheckout(mysqldb)
    test_db <- dbReadTable(con, “mysqltable”)
   })
output$mytable  <- DT::renderDT({
    test_db <- QueriedData() %>% as.data.frame()
    DT::datatable(test_db)
  })
```

Static vs. dynamic dashboards

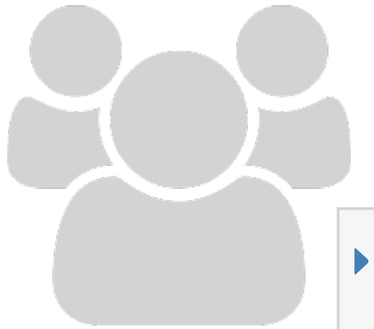
STATIC VS. DYNAMIC

- Static:
  - R code runs once and generates an HTML page
  - Generation of this HTML can be scheduled
- Dynamic:
  - Client web browser connects to an R session running on server
  - User input causes server to do things and send information back to client
  - Interactivity can be on client and server
  - Can update data in real time
  - User potentially can do anything that R can do

FLEX VS. SHINY DASHBOARD

flexdashboard	shinydashboard
R Markdown	Shiny UI code
Super easy	Not quite as easy
Static or dynamic	Dynamic
CSS flexbox layout	Bootstrap grid layout

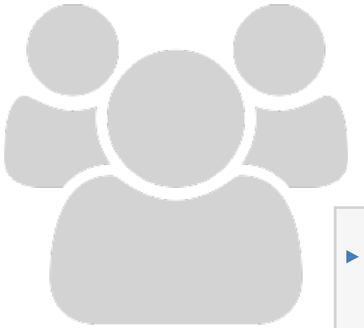
flexdashboard



EXERCISE

- `library(flexdashboard)`
- File → New file → R Markdown → From Template
- Create three plots that go in each of the panes using built-in R datasets or any data we have used in the worksho (or your own data)

3m 00s



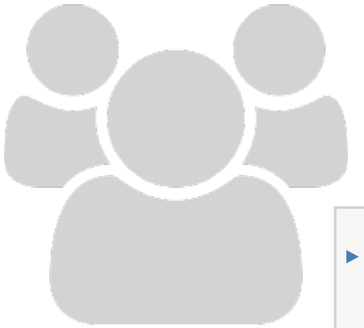
EXERCISE

- Open `apps/flexdashboard_01.Rmd`
- How is it different than Shiny apps we have been building so far, how is it similar?
- Make a change to the layout of the dashboard, see http://rmarkdown.rstudio.com/flexdashboard/using.html#layout for help
- Change the theme of the dashboard, see http://rmarkdown.rstudio.com/flexdashboard/using.html#appearance for help

5m 00s

SHINY DOCUMENTS

- Add runtime: shiny to header.
- Add `inputs` in code chunks.
- Add `renderXyz` functions in code chunks.
  - No need for `output$x <-` assignment, or for `xyzOutput` functions.



EXERCISE

- Continue working on `apps/dashboards/flexdashboard_01.Rmd`
- Add another UI widget, a radioButton, that allows the user to select whether the plot used to visualize the distribution of weight should be histogram or a violin plot

3m 00s

SOLUTION

Sample solution at `apps/flexdashboard_02.Rmd`

SHINY DOCUMENT DRAWBACKS

- Start-up time: knits document every time someone visits it
- Resizing can trigger re-knit
- Auto-reconnection doesn’t work (i.e. client browsers cannot automatically reconnect afer being disconnected due to network problems)

- The solution: Pre-rendered Shiny Documents

Shiny

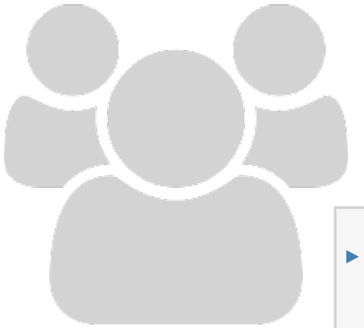
pre-rendered

SHINY PRE_RENDERED

- Rendering phase: UI code (and select other code) is run once, before users connect.
- Serving phase: Server code is run once for each user session.
- Each phase is run in a separate R sessions and can’t access variables from the other phase.

CONTEXTS FOR SHINY_PRERENDERED

- `"render"`: Runs in rendering phase (like `ui`)
- `"server"`: Runs in serving phase (like `server`)
- Additional contexts:
  - `"setup"`: Runs in both phases (like `global.R`)
  - `"data"`: Runs in rendering phase (any variables are saved to a file, and available to serving phase, useful for data preprocessing)
  - `"server-start"`: Runs once in serving phase, when the Shiny document is first run and is not re-executed for each new user of the document, appropriate for
    - establishing shared connections to remote servers (e.g. databases, Spark contexts, etc.)
    - creating reactive values to be shared across sessions (e.g. with `reactivePoll`, `reactiveFileReader`)



EXERCISE

- Start with `apps/flexdashboard_02.Rmd`
- Turn your document into `runtime: shiny_prerendered`
- Note: You will need to use `output$x <-` assignment and `xyzOutput` functions

5m 00s

SOLUTION

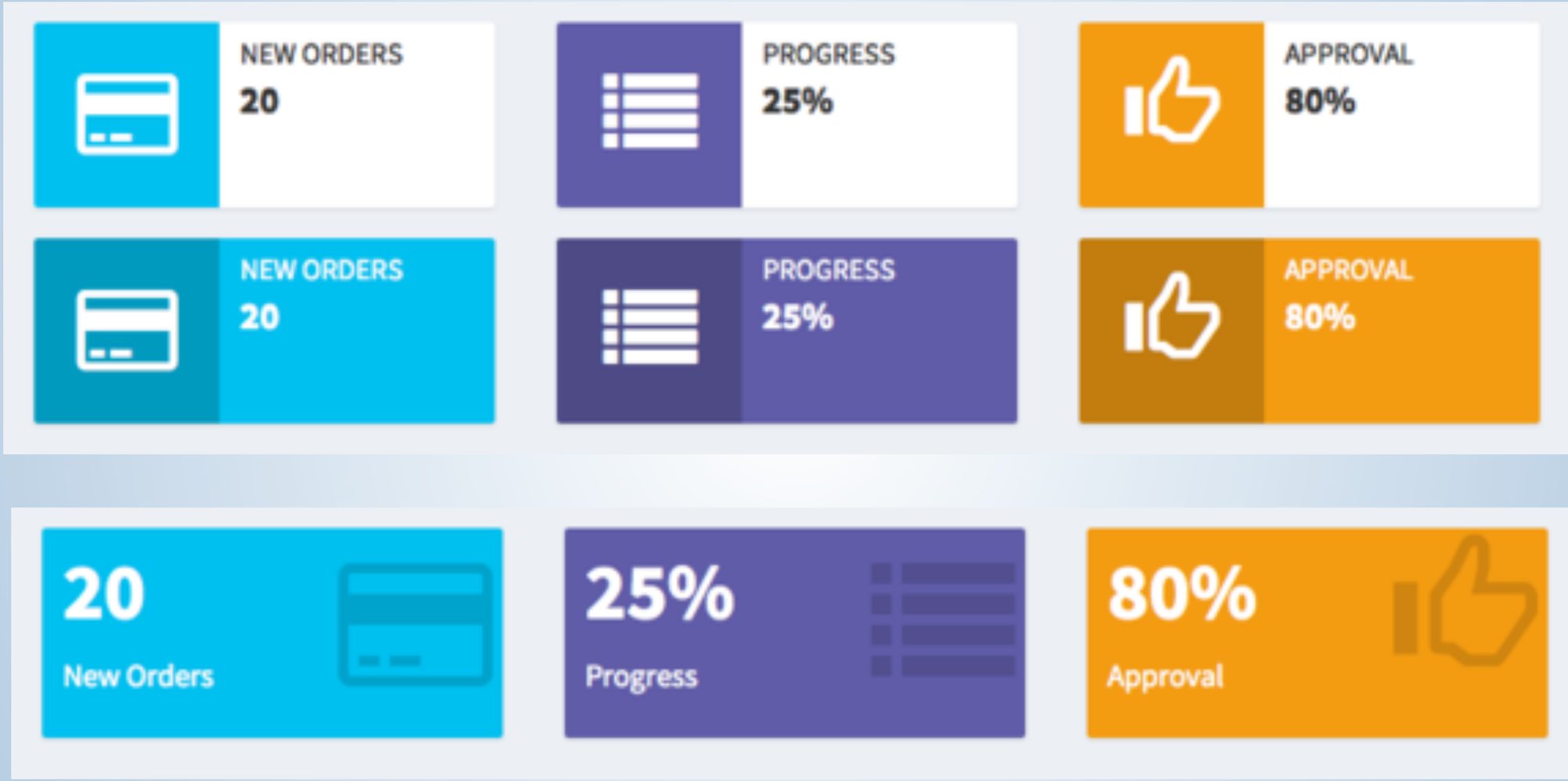
Sample solution at `apps/flexdashboard_03.Rmd`

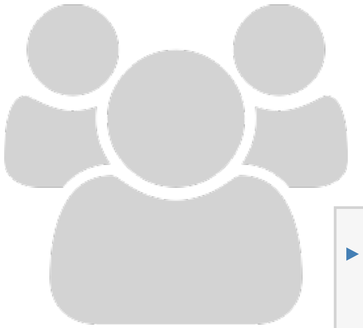
shinydashboard

FORMAT

- shinydashboard is an advanced layout of a typical shiny app
- The ui has more arguments
  - header
  - sidebarMenu
  - body (similar to fluid pages)
  - title
  - skin (color of the page)

Body


NEW ORDERS
20
PROGRESS
25%
APPROVAL
80%
NEW ORDERS
20
PROGRESS
25%
APPROVAL
80%
20
New Orders
25%
Progress
80%
Approval



EXERCISE

- Open starwars_01.R
  - Add an info or value box counting for mass and height respectively (lines 120 or 125)
    - Hint: First run the app to figure out what measurements might make sense
      - Stretch goal: Create the other kind of box

5m 00s

SOLUTION

```
See starwars_02.R
```

Tab1 Tab2 **First tabBox**

First tab content

Tab3 Tab2 Tab1

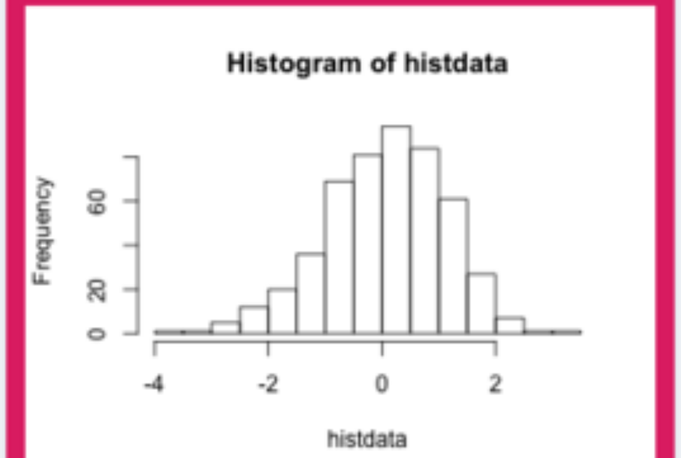
Note that when side=right, the tab order is reversed.

Tab1 Tab2 **tabBox status**

Currently selected tab from first box:

Tab1

Histogram



Inputs

Box content here

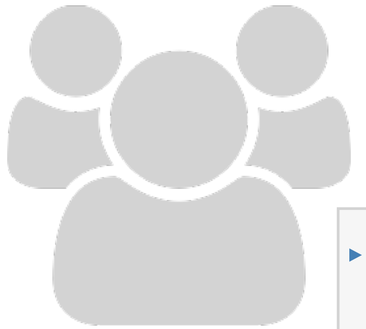
More box content

Slider input:

50

1 11 21 31 41 51 61 71 81 91 100

Text input:



EXERCISE

- Open starwars_02.R
  - Add a tabBox in the body that holds the output of both the plots for mass and height.
    - What arguments do you need to pass to the box so the table fits?
  - Stretch goal: Give the box a title

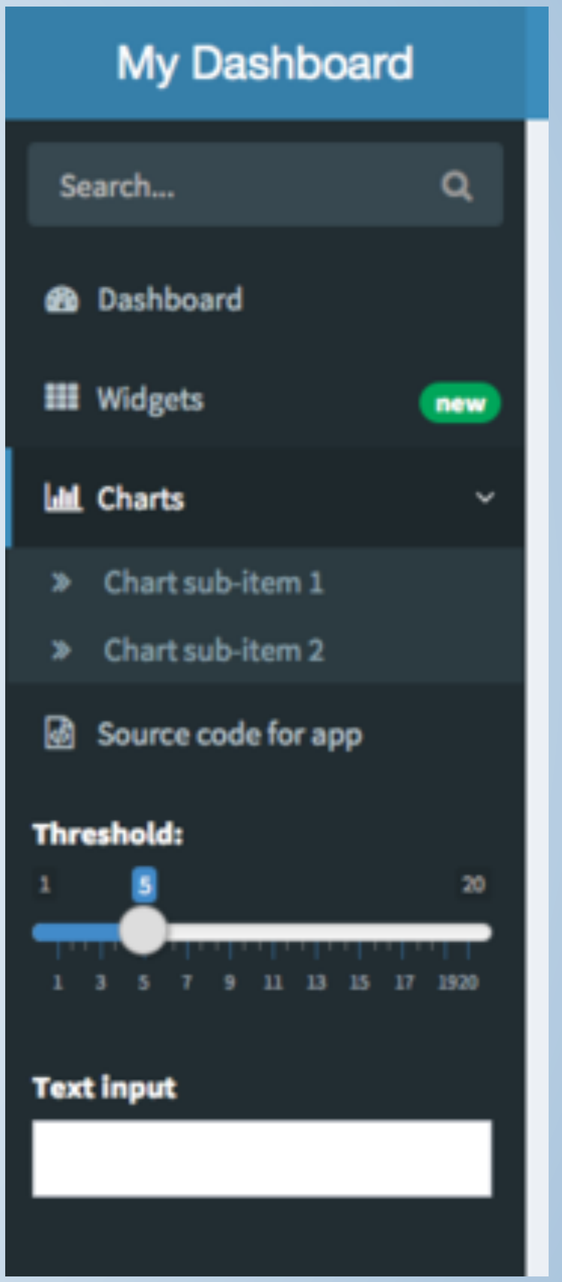
5m 00s

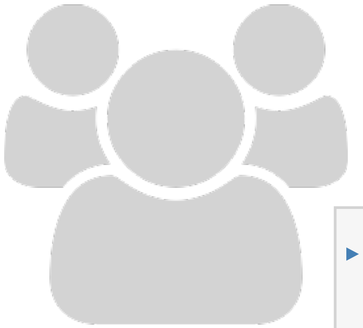
SOLUTION

```
See starwars_03.R
```

Menu


Simple tabs
Dashboard
Widgets
new
Widgets tab content


My Dashboard
Search...
Dashboard
Widgets
new
Charts
Chart sub-item 1
Chart sub-item 2
Source code for app
Threshold:
1
5
20
1 3 5 7 9 11 13 15 17 1920
Text Input



EXERCISE

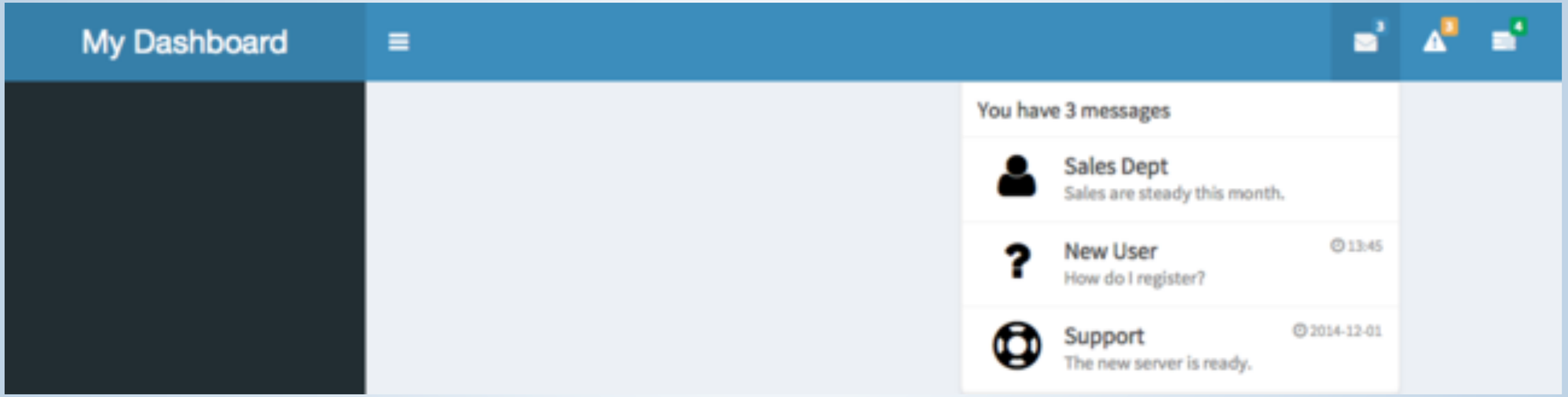
- Open starwars_03.R
  - Add a new menu item that allows users to access the table page

5m 00s

SOLUTION

```
See starwars_04.R
```

Header


My Dashboard
You have 3 messages
Sales Dept
Sales are steady this month.
New User
13:45
How do I register?
Support
2014-12-01
The new server is ready.

HEADER

- Headers have three types of information that can be displayed
  - messageItem - text information along with date/time information
  - notificationItem - basic text information
  - taskItem - show progress towards a goal
- All of these items can be dynamically updated and rendered in the server function
  - For examples see the shinydashboard docs

DEMO

`starwars_04.R`

DASHBOARDS

Shiny from RStudio™

HOMEWORK

Project 1

HOMEWORK

Project 1

Due Date: 9/27

Creating multiple types of visuals from the same data is an important way to convey information to application users. Students will create a Dashboard using a static download of an Open Data or a Dataset from their own place of employment (make sure you have permission to use it for this assignment first!)

Students may make their application in either flexdashboard or shinydashboard layouts and deploy on shinyapps.io.

Directions:

- Include at least:
  - Three (3) input/filters
  - Three (3) single numeric based boxes/gauges
  - One (1) datatable
  - Three (3) interactive and reactively responsive charts. (use ggplot2 for now)
    - These elements should be places throughout a dashboard with at least three (3) pages or tabs with an analytical themes or question about the data.
  - On the server side your plots and tables must utilize the reactive function for any and all datasets.
- Your final app must work when deployed to shinyapps.io.