Fachhochschule Kiel
University of Applied Sciences

Data Visualization and Visual Analytics
Shiny Apps

Study Program Data Science, Summer Semester 2020
Prof. Dr. Tillmann Schwörer

What is Shiny?

- R package for interactive web apps
- Front-end and back-end are implemented in R
  - **Front end** (User Interface):
    - ✓ dedicated set of R functions that generate HTML, CSS and Java script
  - **Back end** (Server):
    - ✓ Typical R functions used in exploratory data analysis, visualization, and modelling
    - ✓ Dedicated functions for rendering R outputs (text, table, plot, etc.) to HTML (renderText, renderTable, renderPlot, etc.)
  - User-interface and server communicate with each other via lists

Background knowledge: web development

- Web development typically consists of separate front-end and back-end development
- **Front end**: visible part of the web site
  - HTML
    - ✓ Hyper text markup language
    - ✓ Creates and organizes the content of the web site
  - CSS
    - ✓ Cascading style sheet
    - ✓ Defines the style of the web site such as layout, colors, fonts, etc
  - JavaScript:
    - ✓ Make HTML pages more dynamic and interactive: drop downs, hoverinfo, pop-ups, etc.
    - ✓ Mostly run in the client’s browser
    - ✓ No communication with the server, no reload of the page

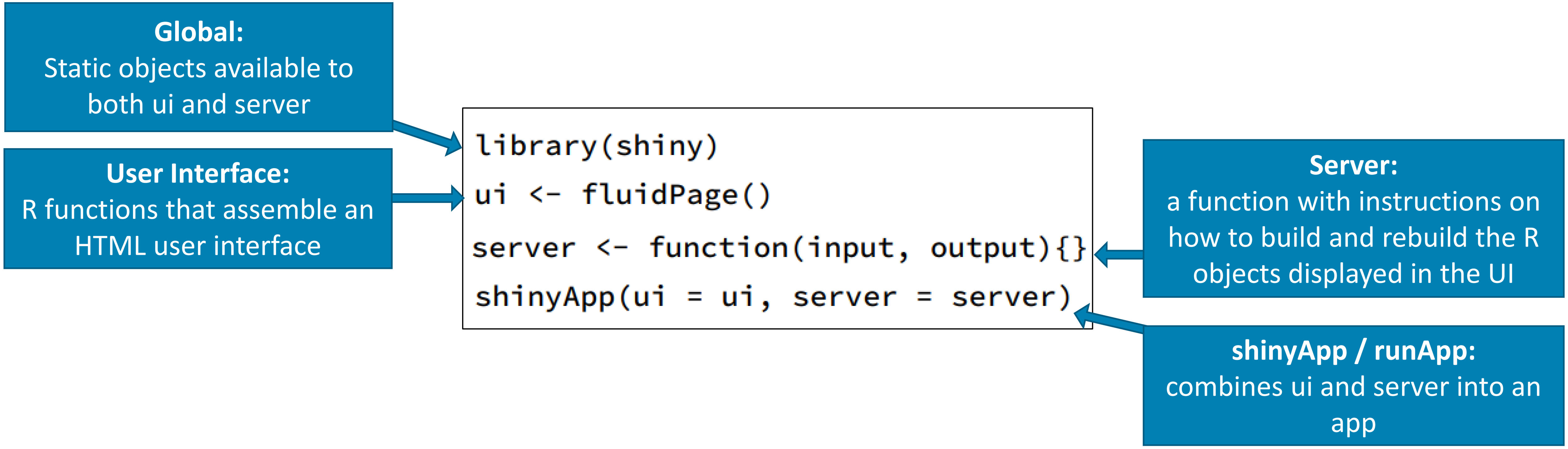
Background knowledge: web development

- **Back end**: invisible part of the web site
  - Retrieve information from databases
  - Implement business logic in code
  - Communicate with front-end: receive inputs, return outputs, manage sessions
  - Languages: node.js, Java, Python, Ruby, PHP, SQL, …

- **Interactivity on web pages** can be implemented …
  - On the client-side via JavaScript elements
  - Or through client-server interaction:
    - ✓ client sends request to server
    - ✓ server retrieves relevant data from database
    - ✓ server implements logic, runs algorithms, etc.
    - ✓ results are rendered to HTML and sent to client for display

Shiny‘s main building blocks

- Either **single file app** (app.R) or **multiple file app** (global.R, ui.R, server.R)
  - Separating into multiple files is useful if app grows larger and more complicated over time
  - The app‘s name is given by the directory name
- Minimal single file shiny app:



Global:
Static objects available to both ui and server

User Interface:
R functions that assemble an HTML user interface

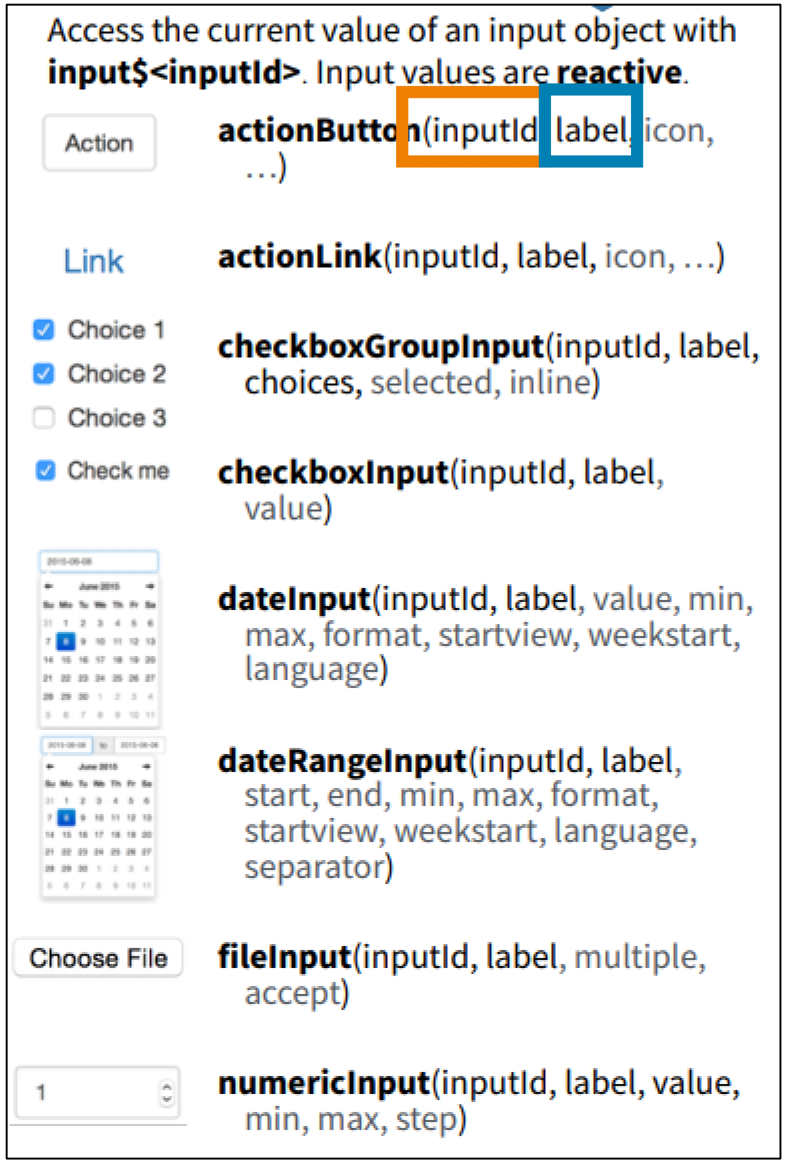
```r
library(shiny)
ui <- fluidPage()
server <- function(input, output){}
shinyApp(ui = ui, server = server)
```

Server:
a function with instructions on how to build and rebuild the R objects displayed in the UI

shinyApp / runApp:
combines ui and server into an app

Inputs

- **User interface**:
  - Add **input controls** to the User Interface
  - Set **inputId** and **label** plus optional parameters
- **Server**:
  - Can access the current value of an input object with **input$<inputId>**
- Gallery of available input controls with code:
  - Default widgets: https://shiny.rstudio.com/gallery/widget-gallery.html
  - shinyWidgets package: http://shinyapps.dreamrs.fr/shinyWidgets/



Access the current value of an input object with **input$<inputId>**. Input values are **reactive**.

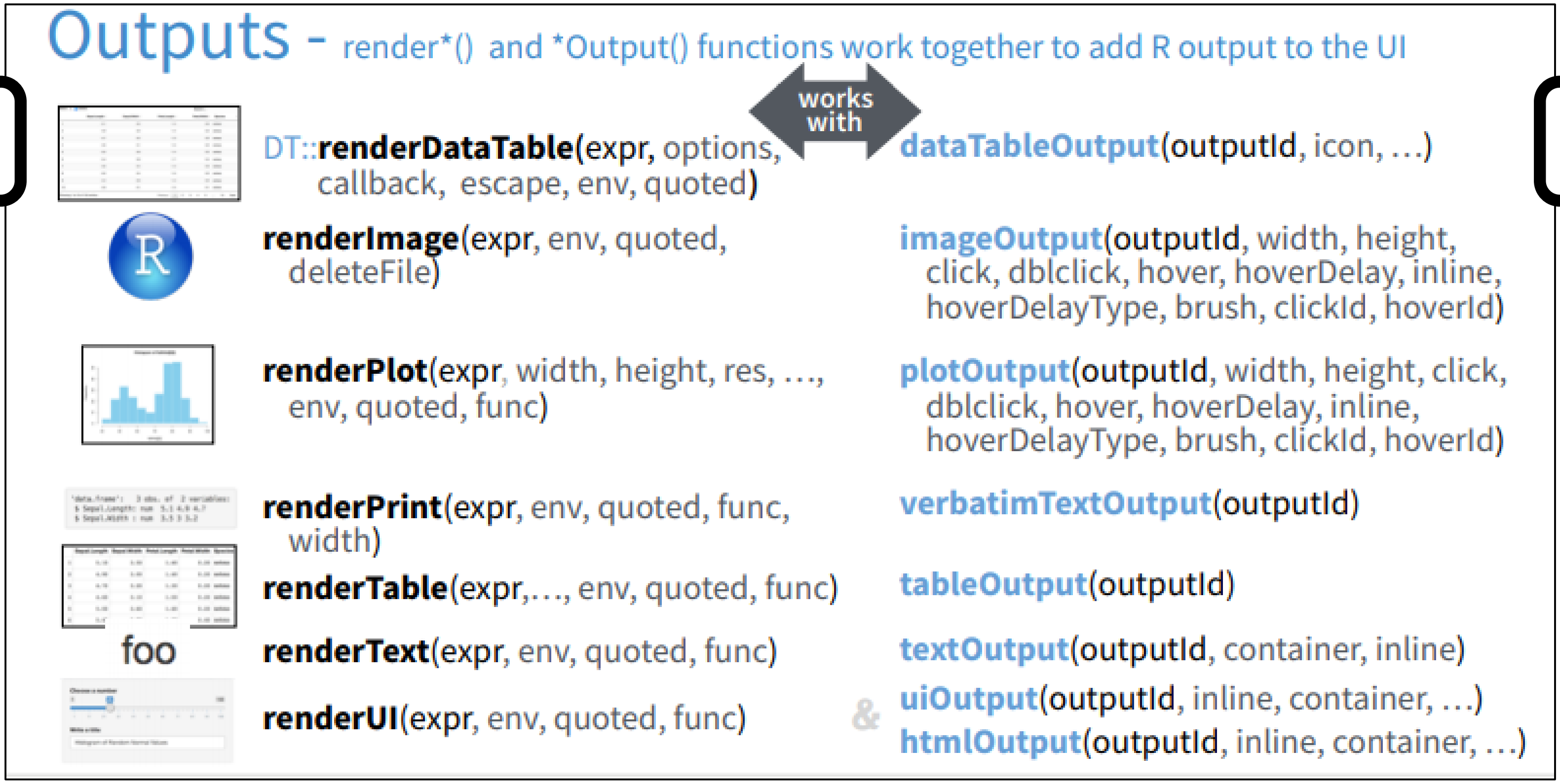
Action	**actionButton**(inputId, label, icon, …)
Link	**actionLink**(inputId, label, icon, …)
Choice 1 / Choice 2 / Choice 3	**checkboxGroupInput**(inputId, label, choices, selected, inline)
Check me	**checkboxInput**(inputId, label, value)
	dateInput(inputId, label, value, min, max, format, startview, weekstart, language)
	dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)
Choose File	**fileInput**(inputId, label, multiple, accept)
1	**numericInput**(inputId, label, value, min, max, step)

Outputs

- **Server**: creates HTML via **render<…>** commands (renderTable, renderPlot, …) depending on the type of R object and sets an **outputId**
- **UI**: can access the current output object via a corresponding **<…>Output** command (tableOutput, plotOutput, …) and a reference to the corresponding **outputId**

Server

UI

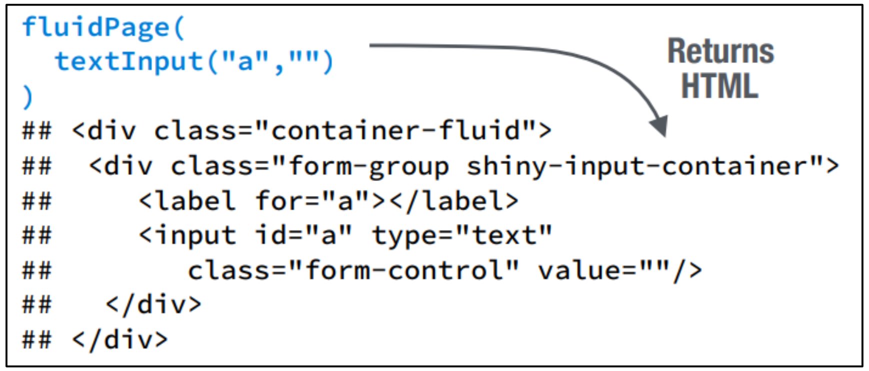


Outputs - render*() and *Output() functions work together to add R output to the UI

works with

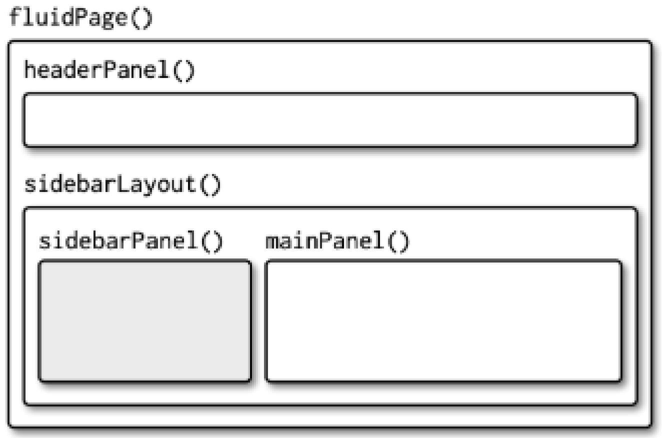
DT::**renderDataTable**(expr, options, callback, escape, env, quoted)	**dataTableOutput**(outputId, icon, …)
renderImage(expr, env, quoted, deleteFile)	**imageOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)
renderPlot(expr, width, height, res, …, env, quoted, func)	**plotOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)
renderPrint(expr, env, quoted, func, width)	**verbatimTextOutput**(outputId)
renderTable(expr,…, env, quoted, func)	**tableOutput**(outputId)
renderText(expr, env, quoted, func)	**textOutput**(outputId, container, inline)
renderUI(expr, env, quoted, func)	**uiOutput**(outputId, inline, container, …) & **htmlOutput**(outputId, inline, container, …)

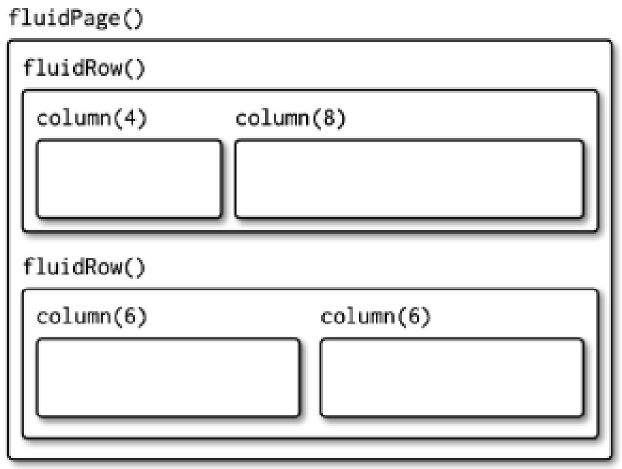
UI Layout



```
fluidPage(
  textInput("a","")
)
## <div class="container-fluid">
##  <div class="form-group shiny-input-container">
##     <label for="a"></label>
##     <input id="a" type="text"
##        class="form-control" value=""/>
##   </div>
## </div>
```

- Basic color themes are available in the **shinythemes** package
- Resources for layout options:
  - Layout guide:://shiny.rstudio.com/articles/layout-guide.html
  - Dashboards: https://rstudio.github.io/shinydashboard/





Best practices

- To make the shiny app **faster and easier to maintain** it is important to implement logic in the right place and to remove duplicated code
- **Static objects** belong in a separate file global.R, e.g.
  - Load packages and datasets
  - Set options
  - Define static lists which are later on referenced by the server or UI
- **Move duplicated server code into reactive expression**
  - Move code into a **reactive function** and assign to variable
  - Example: selected_vars <- reactive({data %>% select(input$vars})
  - This reactive function can then be called at multiple points in the server

Typical errors

- **User Interface:**
  - Parenthesis: check whether opened and closed parenthesis match
  - Commas: elements in the UI are separated by commas
- **Server:**
  - Curly brackets: if server functions span multiple lines we need curly brackets, e.g. renderTable({ <multiple lines of code> })
  - Reactive context: The server can only work with user inputs in a so called reactive context: This can be a **render<…>** function or a **reactive()** function
- **Communication between server and UI**
  - IDs: check whether the inputIds and outputIds of server and UI match
  - Check whether the input type that the UI sends corresponds to the input type that your server function expects

Resources

- **Video tutorial**: https://shiny.rstudio.com/tutorial/
- **Gallery of use cases with code**: https://shiny.rstudio.com/gallery/
- **Links to basic and advanced topics** (including e.g. reactivity, code quality, testing, deployment, etc.): https://shiny.rstudio.com/articles/
- **Cheat sheet**: https://github.com/rstudio/cheatsheets/raw/master/shiny.pdf (maybe a bit overwhelming in the beginning…)