



BOOKDOWN STYLER

A SHINY APP FOR RENDERING & CUSTOMIZING
BOOKDOWN DOCUMENTS



Pachhai, Siddhartha

Table of Contents

Introduction	2
Methods	3
1. User credentials.....	4
2. Database	5
3. Using JavaScript.....	6
4. The book-down library	7
5. Data.....	8
Results	8
1. User registration.....	9
2. New Project	11
3. Customization and Downloading.....	17
4. Example 2	21
Conclusion & Future works	24

Introduction

Book down is an open source R package that simplifies the process of creating professional quality books and documents in multiple formats including HTML, PDF and ePUB. It uses R markdown as a foundation, and compiles it into a book like document, even being able to knit together multiple markdown documents. It has many built in features like automatic numbering and organization of file structure that allows easy uploading to the internet specially with HTML markdown document.

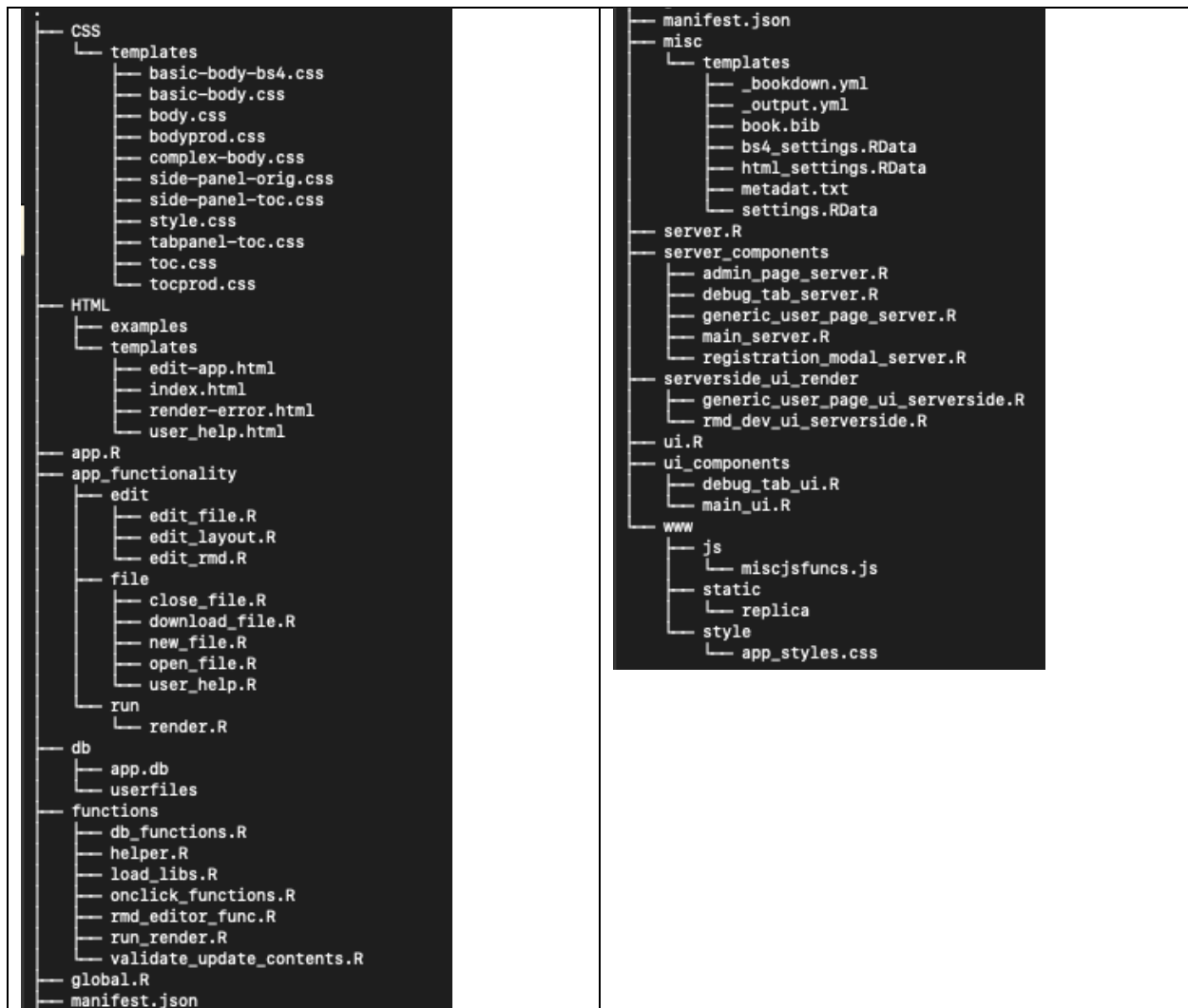
R Shiny is a package for building web application using R. It allows developers to build applications that allow the capabilities of R in regards to data management and statistical analysis to be integrated into a HTML application with custom user interface for purposes of automation and dashboarding.

I was mainly inspired by Yihui Xies's example in GitHub [bookdown.org/yihui/bookdown-demo] of a book-down document creation process and wanted to implement an automated version that would hide the complexities required to program and debug the process which I believe would be cumbersome to someone in a hurry. Additionally I learned how you can leverage Cascading Style Sheets (CSS) to customize the document, so I thought of creating a shiny application that would allow an easy way to customize the appearance and creation process of these book-down documents in HTML form. The choice of HTML document is mainly due to CSS being mostly applied to HTML documents which was a large part of my work.

R-markdown is a crucial part of Statistical analysis now a days. Being able to neatly document our analysis is very helpful to our work. My aim is to help improve the quality and group statistical documents in a more modern way allowing for a better user experience, with well divided sections and simplicity of information through use of hyperlinks, drop downs and other HTML artifacts.

Methods

The main packages that I used for this project were R Shiny and R Shiny-JS. Additionally, packages like book-down, rvest, and xml2 were also used for the purpose of handling the book-down documents. RSQLite was used for the purpose of storing data to be used by the application and save projects. The tidyverse suite was used to manage and transform data in between processes. A extensive list of packages used can be found in the shiny application package.



Below is the folder structure for the application. The choice for such folder structure is from my own experience working with shiny and java script-based web applications. Currently frameworks such as Golem are popular for creating production grade shiny applications

with a more robust folder structure design, but in my opinion having a more flexible structure like that I have chosen is more convenient for non-traditional shiny applications. Golem in my opinion is more suited for traditional shiny applications like dashboards or simpler tools.

The three main files are Server.R, Ui.R and Global.R, these files orchestrate the backend, frontend and global variables respectively. The www folder holds static files like HTML templates and images, which can be referenced while the app is running. Files outside the www folder cannot be linked to the HTML page. It is possible to load HTML in the server component but any file that must be linked, needs to be in www folder.

Explaining each and every file will take a long time, so I have highlighted some approaches to building the app which was challenging, novel and crucial.

1. User credentials.

```
#creates an authorization method using the user_base dataframe.
credentials <- shinyauthr::loginServer(
  id = "login",
  data = user_base,
  user_col = user,
  pwd_col = password,
  sodium_hashed = TRUE,
  log_out = reactive(logout_init())
)

# Logout to hide
logout_init <- shinyauthr::logoutServer(
  id = "logout",
  active = reactive(credentials()$user_auth)
)
```

The shinyauthr library makes it very easy to build user login and password management. The data parameter of the loginServer function holds a dataframe, like the one shown below which allows the shiny application to keep track of users and logins.

	username	password	role
1	shiny_manager	apple	admin
2	test_user_1	apple	user
3	user	apple	user

```

#renders the admin page
output$admin_ui <- renderUI({
  req(credentials())$user_auth)
  #require admin role.
  req(credentials())$info[["permissions"]] == "admin")

  fluidRow(
    div(id = "user_table_admin",
      dataTableOutput("user_table"))
  )
})

```

The code snippet above shows how the credentials created right before can be used to render or stop rendering of UI element, or backend process. The credentials object can also be used to differentiate user type.

2. Database

```

#This functions modifies a table, takes in a SQL command string.
sqlite_dml <- function(dml,params=NULL){
  #Initialize the users table
  conn <- dbConnect(RSQLite::SQLite(),
    paste0(fileLoc_runApp,"/bookdownstyler/db/app.db"))

  # Set the encryption key
  dbExecute(conn, "PRAGMA key = 'defenceoftheancients';")

  dbExecute(conn, dml, params=params)

  dbDisconnect(conn)
}

```

For the database creation process, I used the RSQLite and DBI packages. These packages allowed me to create tables that stored information such as user logins, project that they could save to the application and project details which allowed the application to be more dynamic. Wrapping the database process into a function starting with connect and ending

with disconnect as shown above was helpful and made for cleaner code as huge chunks didn't have to be repeated. An example of this can be seen below.

```
sqlite_dml(
"UPDATE projects SET type = 'bs4_book' WHERE project_name = ? AND username = ?",
  params = list(project, user))
```

3. Using JavaScript.

Using JavaScript allowed the app to be dynamic.

```
runjs('document.getElementById("main_tab_text").innerText = "Login";');
  runjs('document.getElementById("main_tab_icon").classList.remove("fa", "fa-
file");');
  runjs('document.getElementById("main_tab_icon").classList.add("fa-solid", "fa-
arrow-right-to-bracket");');
```

For example using runjs from the ShinyJS library and knowledge of javascript and html documents allowed to dynamically change the contents of the ui. An example of this shown above where I can replace the text and change the attributes of certain front end HTML elements from calling runjs in the server side.

```
runjs("Shiny.setInputValue('update_rmd_trigger', 1, {priority: \"event\"});")

#every time the edit rmd button is either opened or updated, run this code to list all
rmd files, if they exist (file > 0)
observeEvent(input$upload_rmd_submit | input$edit_rmd | input$update_rmd_trigger,{

  output$edit_rmd_file_ui <- renderUI({

    if (!is.null(input$user_open_file_project_selected)){
      if (input$user_open_file_project_selected != ""){

        print ("UI Rendered First")

        user = credentials()$info[["user"]]
        project = input$user_open_file_project_selected
        validation_query = sqlite_dql(paste0("SELECT * from CONTENTS where username
= '',user,'" and project_name = '',project,'" ORDER BY run_order;"))
        if (nrow(validation_query) > 0){
```

Using JavaScript also allowed me to create my own triggers to perform certain actions. The `Shiny.setInputValue` function is available when running the Shiny application and be called using JavaScript. This allows us to create our own variable which can be added to ``input$``, An example of this is shown above, where I create a variable to be observed which then sends a UI element to the front end.

4. The book-down library

```
x = readLines(paste0(location, '/', contents_loc, '/index.rmd'))

if (applycss){
  cat(
    'bookdown::', fmt, ':\n', '  css: [' , style_list, ']\n',
    sep = ' ', file = paste0(location, '/', contents_loc, '/_output.yml'),
    append = TRUE
  )
}

bookdown::render_book(paste0(location, '/', contents_loc, '/'),
  paste0('bookdown::', fmt),
  quiet = TRUE, output_dir = out_dir,
  config_file = paste0(location, '/', contents_loc, '/_bookdown.yml'))
```

The book-down generation process is fairly simple in terms of R code. The `render book` function takes in arguments like the location of where to find the `.Rmd` files, what format to use and where to output the book. Book-down uses a location and reads a file called `index` as the first file, then proceeds to read other files sequentially based on how they are named. This involves a lot of R code to copy files from one place to another and renaming them using the ``file.copy()`` function in R. There is one more way of doing this, which is using the `bookdown.yml` file to edit the run-order, but I found it easier to just rename the files. If a `bookdown.yml` file is to be used we can pass it into the `config_file` parameter of the `render book` function.

The `render book` function works by creating a `_main.Rmd` file and stitching all the contents together into it. So in the code above, the styling and formatting options are read from a file called `_output.Rmd`, which needs to be placed in the same folder as where you have the R-markdown documents. As shown above, the style document (CSS) can be added here. As mentioned since all the documents are stitched together into one, the run order is

important, if the files are not labeled correctly, variables declared can be missing if declaration markdown document is later than the document that calls it.

5. Data

For this project there aren't any datasets in the traditional sense, but I used some self-generated R-markdown files for purposes of testing and developing the app. They have been submitted as part of the submission package.

```
.
├── default
│   ├── 01-intro.Rmd
│   ├── 02-literature.Rmd
│   ├── 03-method.Rmd
│   ├── 04-application.Rmd
│   ├── 05-summary.Rmd
│   ├── 06-references.Rmd
│   └── index.Rmd
├── hw
│   ├── index.Rmd
│   ├── page1.Rmd
│   ├── page2.Rmd
│   ├── page3.Rmd
│   └── page4.Rmd
├── long_error
│   └── long_error.Rmd
├── long_correct
│   ├── long_correct.Rmd
│   └── long_correct_2.Rmd
```

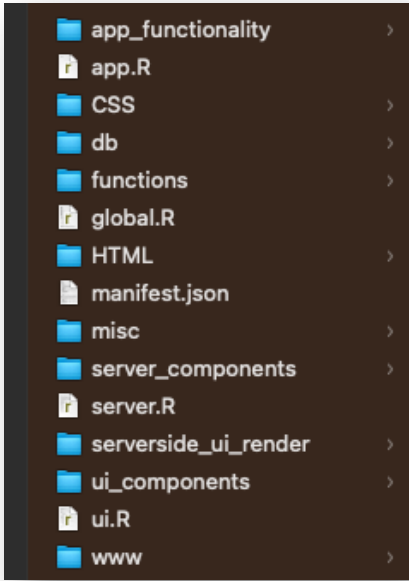
The default folder files, are from by Yihui Xies's original example, while the other files are from my UFL assignments.

Running the application with these files observing the errors and developing around the files was analogous to using a dataset.

Results

The Bookdownstyler app can be run in either of two ways.

The first one is using R-studio.

	<p>A zip file has been provided, once unzipped, double clicking the app.R file from where it has been unzipped and running the entire code chunk will start the process. It is important that you open the file directly as this effects the working directory which is important for the application.</p> <p>The libraries needed can be located in global.R file. This file has a lot of libraries listed, but not all of them are required. I have loaded more than required set of packages as the .Rmd files may use them, but the core libraries are dplyr, shiny, shinyauthr, DBI, RSQLite, shiny.router, shinyjs, shiny, purr, shinyFiles, htmlwidgets, shinyalert, readr, rvest, Xml2, fs, colourpicker, R.utils, bookdown, rstudioapi, downlit.</p> <p>There might be some errors while running the program due to missing libraries, but a simple fix to this is either installing them or commenting them out from the global.R file.</p>
---	---

Note: On mac devices you might have to change the access requirements for the folder where the app is located in using ``chmod -R a+w <folder>`` command. If you encounter this problem it should mention it in the logs that certain folders were unable to be created. If so delete the entire folder, copy over the files again and change the access prior to running.

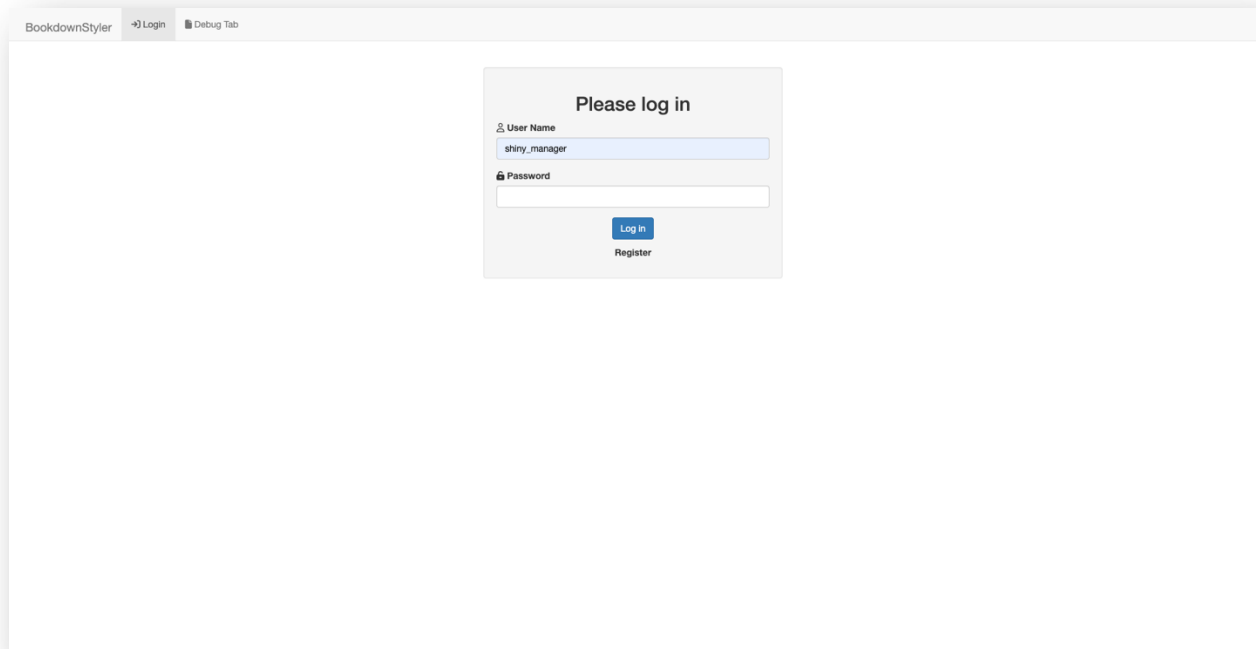
The second one which is recommended is using the following link

<https://bookdownstyler.shinyapps.io/bookdownstyler/>

The application has been set up and configured to run on ``shinyapps.io``. The only issue with this method is that memory is ephemeral, so saved progresses are lost, but for convience I am using this method, future changes are mentioned in the last section of the report. There is an idle time of 1 hour, but to be on the safe side it is recommended to not let the instance run idle for more than 30 minutes.

Please use a chrome browser for the following section.

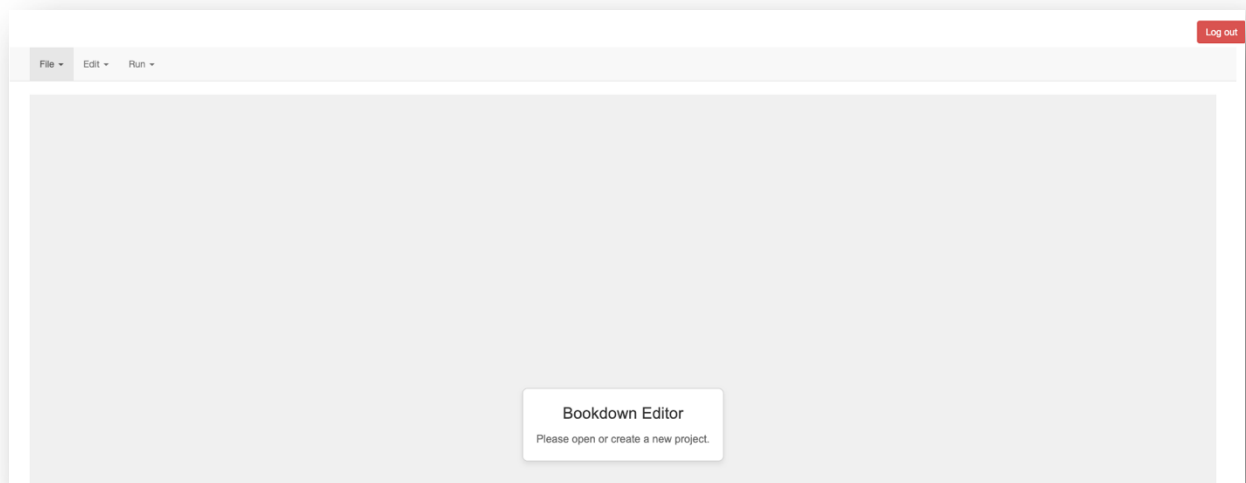
1. User registration.



When you enter the application you first have to register as a new user, there is also a user called user available with the password apple, but let's go over from step 1.

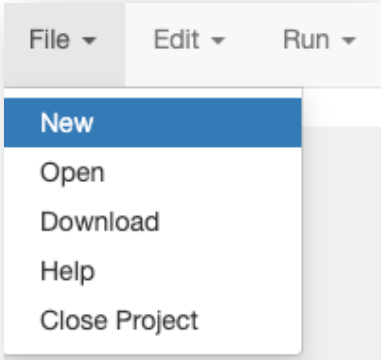


A screenshot of a registration modal window titled "Please enter your information". It contains three input fields labeled "Username", "Password", and "Retype Password". At the bottom right of the modal are two buttons: "Submit" and "cancel".	<p>The next step is to create a user. A modal will pop up when you click on register.</p>
--	---

<div><div>Please enter your information</div><div><div>Username</div><div><input type="text" value="student"/></div></div><div><div>Password need to mach</div><div><input type="password" value="*****"/></div></div><div><div>Retype Password</div><div><input type="password" value="*****"/></div></div><div><div>Submitcancel</div></div></div>	<p>The user registration does some basic checks like if the user already exists or if the passwords match. Once the registration is completed the modal will disappear and you will be in the log-in screen.</p>
---	--

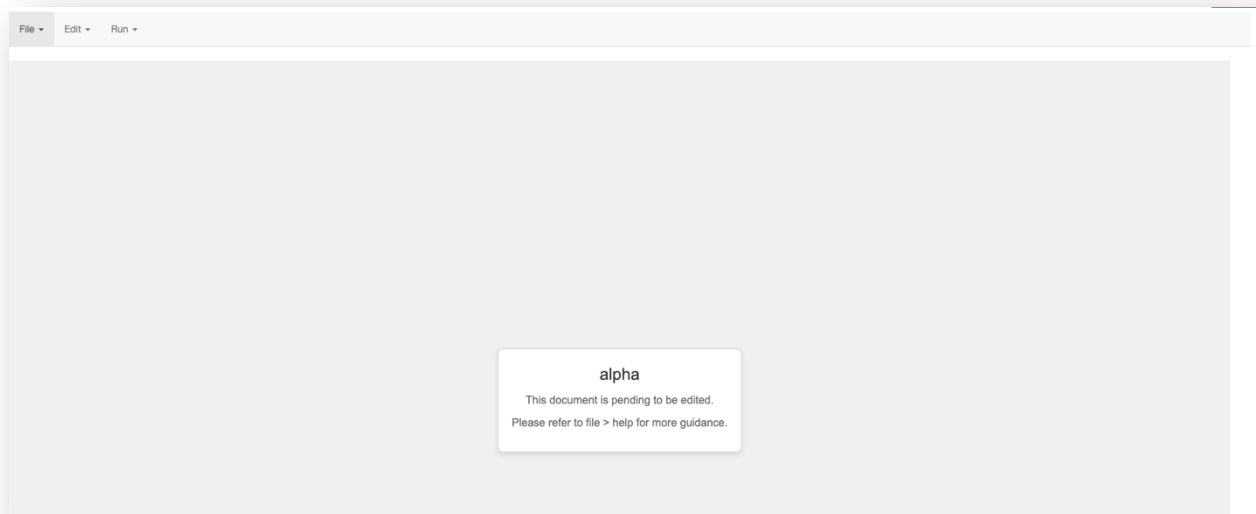


2. New Project

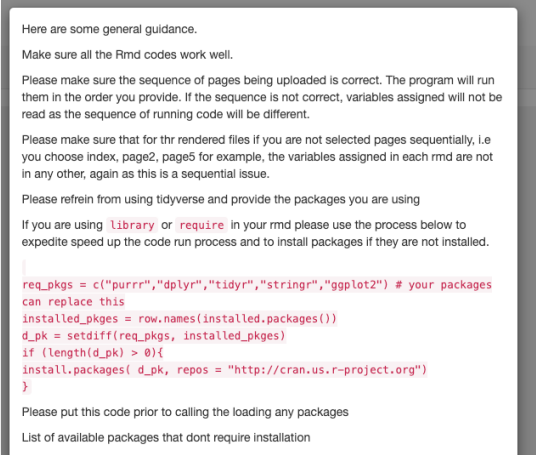
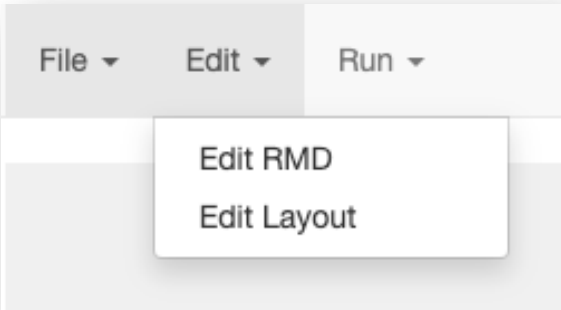
Once logged in you will see the view as seen above. The logout button will log you out. Next lets create a new project.

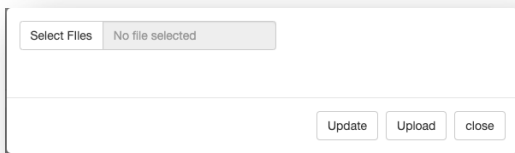
	<p>When you click on file you will see various options, lets first click on open.</p>
	<p>We haven't created any new files so you will see the following message. Next lets create a new file.</p>
	<p>Let's create a project called alpha and submit it.</p>

Once the new project has been created the name will appear on the center, indicating a new project has been created.



Let's revisit the help button in the file tab to understand the application in more depth.

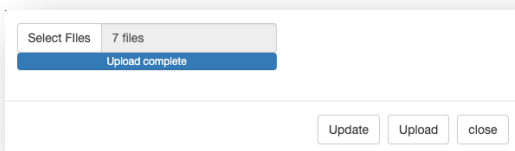
 A screenshot of a help page. It contains several paragraphs of text providing general guidance. The first paragraph says 'Here are some general guidance.' followed by 'Make sure all the Rmd codes work well.' The second paragraph discusses the sequence of pages and code execution. The third paragraph talks about rendering files sequentially. The fourth paragraph advises on using 'library' or 'require' and provides a code snippet for installing packages. The code snippet is: <pre>req_pkgs = c("purrr","dplyr","tidyr","stringr","ggplot2") # your packages can replace this installed_pkgs = row.names(installed.packages()) d_pk = setdiff(req_pkgs, installed_pkgs) if (length(d_pk) > 0){ install.packages(d_pk, repos = "http://cran.us.r-project.org") }</pre> The final paragraph says 'Please put this code prior to calling the loading any packages' and 'List of available packages that dont require installation'.	<p>The help page will give some helpful tips and some reasons why there might be an error in the app, as well as the packages that are available to the user. We can close the help modal by scrolling to the bottom of the modal and clicking on cancel.</p>
 A screenshot of the application's menu system. The 'File', 'Edit', and 'Run' menus are visible at the top. The 'Edit' menu is open, showing two options: 'Edit RMD' and 'Edit Layout'. The 'Edit RMD' option is highlighted with a white background and a gray border.	<p>Next let's click on the edit button to edit the document. Let's first upload markdown documents by clicking on Edit RMD.</p>



To upload markdown files first click on select files. Examples have been submitted with the project which can be used to understand the app.

Name	Size
> bad	--
> long_correct	--
> long_error	--
> hw	--
▼ default	--
index.Rmd	831 bytes
06-references.Rmd	55 bytes
05-summary.Rmd	45 bytes
04-application.Rmd	114 bytes
03-method.Rmd	629 bytes
02-literature.Rmd	52 bytes
01-intro.Rmd	1 KB

Lets look at the documents in the default folder. These are the original examples shared by Yihui Xies's. Lets select all the files and click on open or similar button depending on what os you are using.



Once the files are ready to be uploaded the blue bar will show up. Next click on upload to upload the R-markdown files.

File Name	Run Order	Index Page	render File	Delete File
Index.Rmd	1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
06-references.Rmd	2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
05-summary.Rmd	3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
04-application.Rmd	4	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
03-method.Rmd	4	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
02-literature.Rmd	3	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
01-intro.Rmd	2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

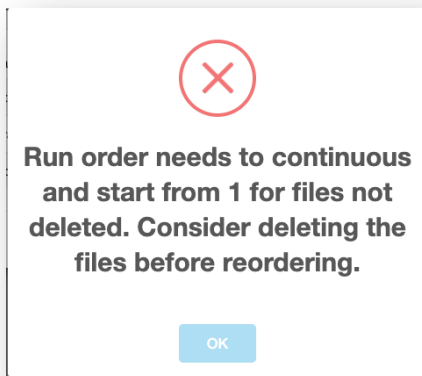
Once the files are uploaded you will see the following. The delete file column will select the file that needs to be deleted. The index page checkbox denotes the first page of the markdown document. The render file check box is an option to display the rendered document for preview, up to 4 can be previewed. Only 4 can be selected for preview as to not overcrowd the screen.

Here are some rules that we must follow. The index page always needs to be rendered. So as you can see in our case index.Rmd is the index page or beginning of the book, the Rmd file doesn't necessarily have to be named

index. Index page document should always have run order number as 1.

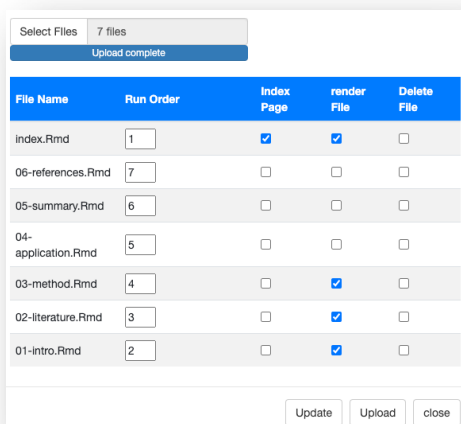
Next let's click on index page checkbox to mark the index page. We will select index.Rmd, 01-intro.Rmd, 02-literature.Rmd, 03-method.Rmd for rendering and also update their run orders to be 1,2,3,4 respectively.

The order in which files are loaded may differ. So for demo purposes match the run order to the screen shot you see above.



You probably would have gotten the following error.

Files to be ordered with unique numbers, start from 1, the index page should also always be ordered.

A modal window titled "Select Files" showing a list of 7 files. At the top, it says "Upload complete". The table has columns: File Name, Run Order, Index Page, render File, and Delete File. The files are listed with their current run orders and checkboxes for selection. At the bottom are buttons for "Update", "Upload", and "close".

File Name	Run Order	Index Page	render File	Delete File
index.Rmd	1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
06-references.Rmd	7	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
05-summary.Rmd	6	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
04-application.Rmd	5	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
03-method.Rmd	4	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
02-literature.Rmd	3	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
01-intro.Rmd	2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

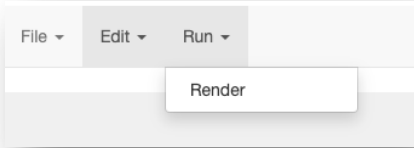
Let's reorder the pages correctly and hit update.

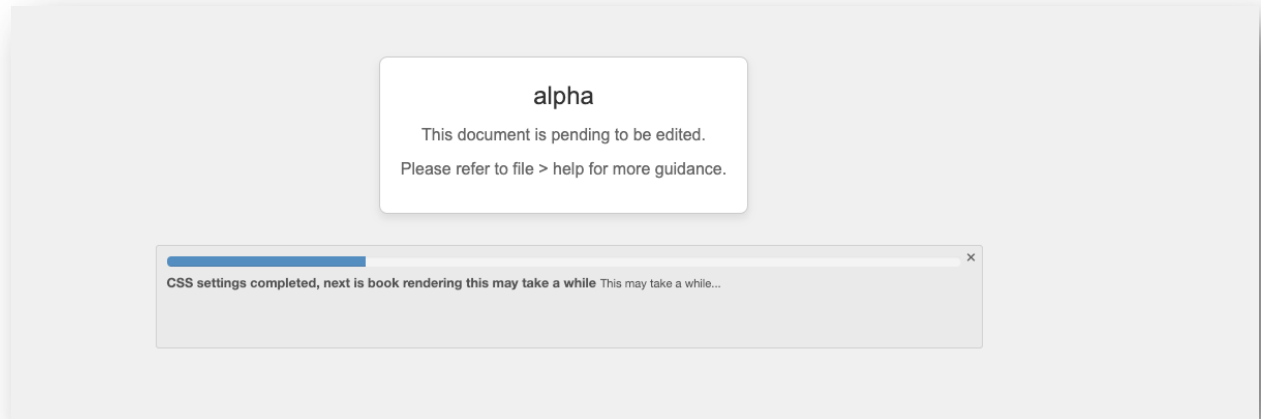
This should also re-order the layout of the files.

Now that the orders have been updated we can close the modal.

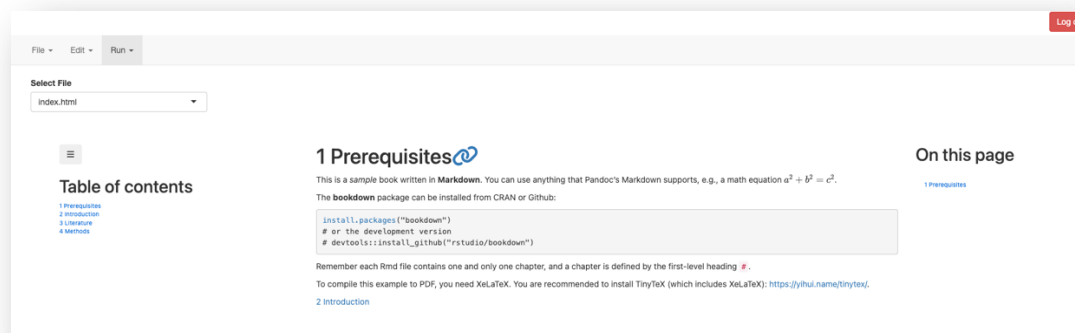
It is always a good idea to hit update before doing anything.

If you encounter errors at this point, please create a new project and after uploading the documents, match the run order and checkbox selection as in the screen shot to the right and continue.

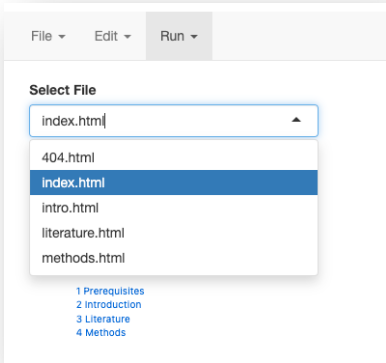
	<p>Next lets go to the run tab and click on render.</p>
---	---



A progress bar will indicate when the rendering is completed.



A preview of the output will be loaded in the first page of the project.



The other documents rendered for preview along with the index page can also be viewed.



2 Introduction

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 2. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter 4.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

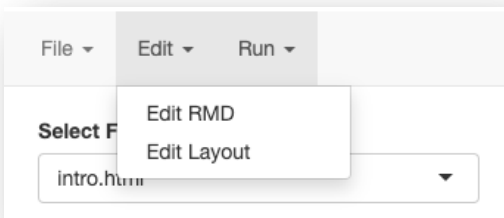
```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

pressure

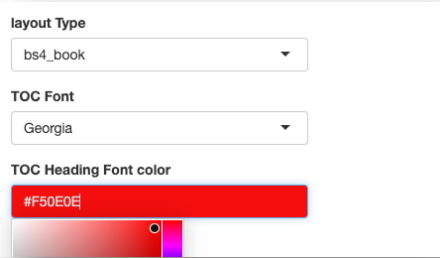
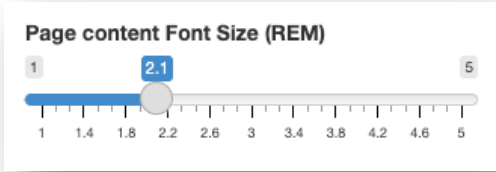
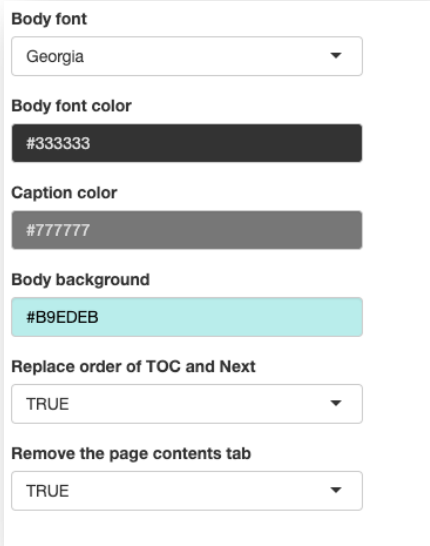
On this page

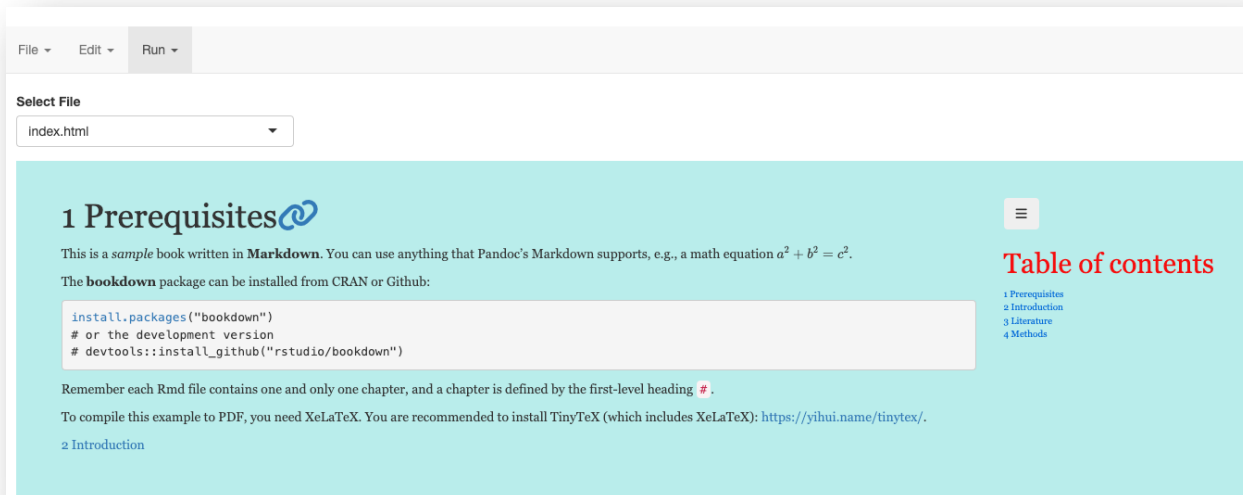
2 introduction

3. Customization and Downloading



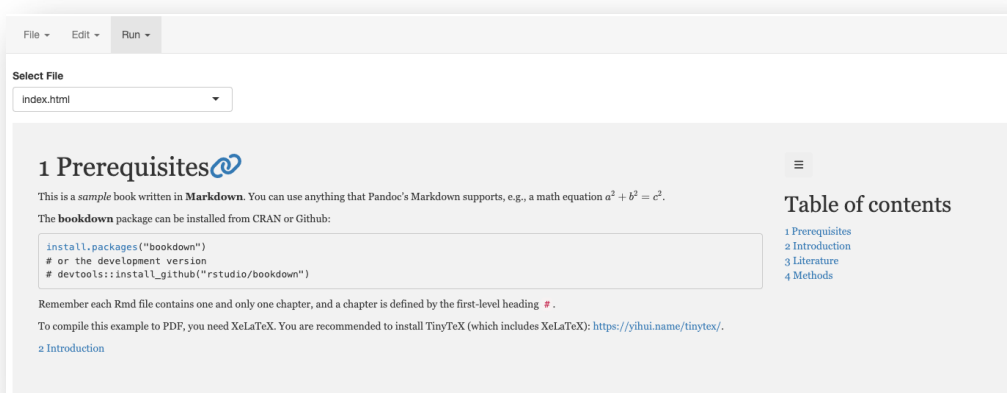
Next let's click on Edit layout to make changes to the style of the output.

	<p>Let's change the font of the TOC to Georgia and the heading color of the TOC to a reddish hue.</p>
	<p>Let's update the page font size to 2.1 rem.</p>
	<p>Let's set the body font to Georgia, the body background to a light blue hue.</p> <p>The last two options are TRUE by default, let's see what they do. Next click on update at the bottom of the modal and let's click render again.</p>

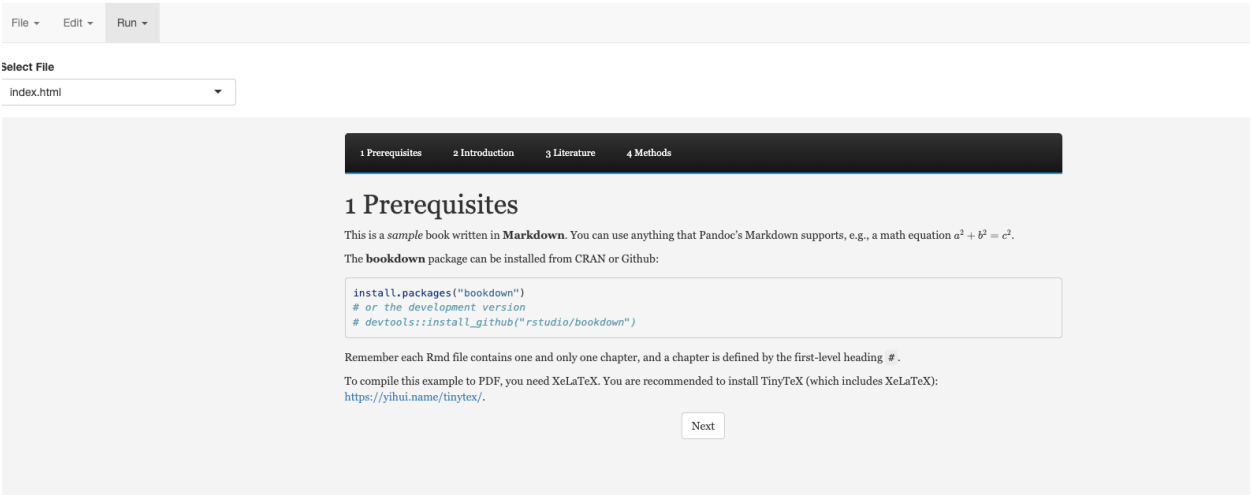


Now the layout of the book has been updated. Because we also selected the last two options, the TOC is now on the right and the other column is removed, if we want it the way it was before, we need to set both options to FALSE.

	<p>Let's update the layout again. By default, the layout type is bs4_book, let's change it to `html_book`. You will notice the options change as well. Let's put it as default, update and render again.</p>
--	---



You will notice the color has changed but nothing too much drastically different from what we originally had. This is due to some background refreshing process that doesn't catch up with the rendering process. **Toggle between different page in the select files drop down and see the update take place.**



The TOC is now a tab on the top and the layout has completely changed.

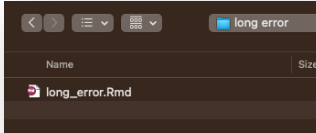

You might have also noticed the hyperlinks inside the rendered content don't work. Next lets down the file. Go to the file tab and click on download.

	<p>A zipped file will be downloaded to your computer.</p>
	<p>Inside it if you open HTML folder you will find the fully rendered book with all pages. You will also notice a folder called style, this is the formatting that the html files used. The path is configured to read from the style folder, so it is highly recommended the folder structure not be altered.</p> <p>The pages are now ready and can be hosted in ones website.</p>

4. Example 2

Let me walk through one more example.

First let's start with a start by creating a new project called "beta"

	<p>Let's upload a R-markdown document called `long_error` from the long error folder, update the page so the index page is rendered then render it.</p>
<div data-bbox="235 806 599 1188"><p>Your file was unable to be rendered. It might be due to the following reasons</p><p>There is something wrong with the rmd file, please make sure it runs.</p><p>Please make sure the sequence of pages being uploaded is correct. The program will run them in the order you provide. If the sequence is not correct, variables assigned will not be read as the sequence of running code will be different.</p><p>Please make sure that for the rendered files if you are not selected pages sequentially, i.e. you choose index, page2, page5 for example, the variables assigned in each rmd are not in any other, again as this is a sequential issue.</p><p>Please refrain from using tidyverse and provide the packages you are using</p><p>If you are using <code>library</code> or <code>require</code> in your rmd please use the process below to expedite speed up the code run process and to install packages if they are not installed.</p><pre>req_pkgs = c("purrr","dplyr","tidyr","stringr","ggplot2") # your packages can_replace_this installed_pkgs = row.names(installed.packages()) d_pk = setdiff(req_pkgs, installed_pkgs) if (length(d_pk) > 0){ install.packages(d_pk, repos = "http://cran.us.r-project.org") }</pre><p>Please put this code prior to calling the loading any packages</p><p>A list of available packages that don't require installation can be found in the help tab</p><p>cancel</p></div> <div data-bbox="235 1329 599 1587"><p></p><p>Rmd files should have at least 1 h1 header</p><p>OK</p></div>	<p>We get either of these error message with details on why the program didn't run so let's open the document in R-Studio to see if any of the warnings apply to our document.</p>

```
## title: "IMZ"
## author: "Total points: 100 (5% of final grade)."  
## date: "Due: 11:59 pm, Thursday Sep 26th, 2024"  
## output: html_document  
---  
  
[R setup, include=FALSE]  
knitr::opts_chunk$set(echo = TRUE)  
options(stringsAsFactors = FALSE) ## prevent read in string data to factors.  
library(tidyverse)  
library(ggplot)  
library(plots)
```

Two issues apply in our case. 1 We don't have a level 1 header, second we have loaded tidy-verse these hinder the rendering process so let's make the updates,

```
# Heading 1

## [r setup, include=FALSE]
knitr::opts_chunk$set(echo = TRUE)
options(stringsAsFactors = FALSE) ## prevent read in string data to factors.

req_pkgs = c("purrr", "dplyr", "tidyr", "stringr", "ggplot2", "ggrepel", "ggplots", "magrittr")
installed_pkgs = row.names(installed.packages())
d_pk = setdiff(req_pkgs, installed_pkgs)
if (length(d_pk) > 0){
  install.packages(d_pk, repos = "http://cran.us.r-project.org")
}

library(dplyr)
library(tidyr)
library(stringr)
library(ggplot2)
library(magrittr)
library(ggrepel)
library(ggplots)
```

The following updates should fix the issue. We add a level 1 heading and code snippet for package installation and load the packages individually instead of loading tidy-verse. Tidy-verse loading causes some complications in shinyapps.io, so the following approach is required. Instead of this we can also upload ``long_correct`` from the Long Correct folder from example uploads.

Select Files

long_correct.Rmd

Upload complete

File Name	Run Order	Index Page	render File	Delete File
long_correct.Rmd	<input type="text" value="2"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
long_error.Rmd	<input type="text" value="1"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Update

Upload

close

Select Files

long_correct.Rmd

Upload complete

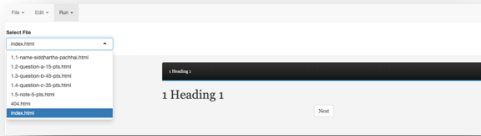
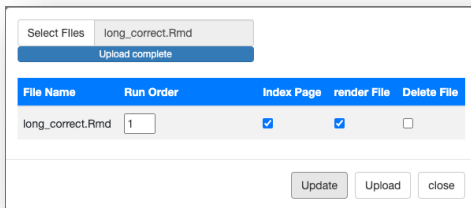
File Name	Run Order	Index Page	render File	Delete File
long_correct.Rmd	<input type="text" value="1"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
long_error.Rmd	<input type="text" value="2"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Update

Upload

close

So we can upload long correct as the first screen shot, change the order of run order index and rendering and lastly delete `long error`. All these steps should be done with making updates at every step.



The one thing worth noticing is that depending on depending on whether you select `bs4_book` or `html_book` it changes the layout of the rendered files.

For `long correct` the html format will create separate pages for different subheadings but for the bs4 format will create a single page.

An identical notebook called `long_correct_2` is also available where there are multiple level 1 headers. If we provide this, the multiple pages are separated for `html_book` format

Conclusion & Future works

This project allowed me to thoroughly explore Shiny application development and understand some of the capabilities and limitations of what can be done using Shiny. I understood how complicated application development can be, and it is hard to program every scenario of how the application can be used and when it will fail. This taught me how I can use tools like Git to plan and revert changes and manage my project. It also taught me time management and gave me a sense of how real-world application development can be and how sometimes you must sacrifice perfection for timely delivery.

I am hoping that this tool will be used in the following ways. Give ability for statisticians to create web books easily. Allow researchers to create and compile statistical documents that allows for easy consumption by targeted audiences. Give statistics students the ability to better organize their work.

In the future there are more features I want to add such as the ability for users to upload their own styles and CSS choices. Additionally, I want to incorporate tests into the development process and iron out some processes and fix potential points of failure.

I also learned the limitations of hosting through Shinyapps. The storage provided is ephemeral, so in the future if I was to use Shiny apps, I would have to rewrite the program so that the storage it uses is not in Shinyapps rather uses cloud services like AWS or google and have functions and API's that transfer data between them. But I think if I had to re do the deployment of this app, I would do it use AWS, host a EC2 instance and deploy the Shiny app there, which I think would give me more control, better performance and give me reliable storage. Also, the development process would be more similar to developing the application in PC, rather than updating the code to match the requirements of Shinyapps.