November 2024

R Shiny Masterclass Series - Advanced

Session 7

User authentication, permissions and management in R Shiny

EPI-interactive

Agenda

Session 1 | October 30th | Introduction, recap and responsive interfaces in R Shiny

Session 2 | November 1st | Advanced reactivity and UX considerations

Session 3 | November 5th | Useful R packages to extend core Shiny functionality

Session 4 | November 6th | Managing complexity: modularizing with the module pattern

Session 5 | November 8th | Advanced data sources and processing

Session 6 | November 12th | Automated report generation

Session 7 | November 13th | User authentication, Extended exercise

Session 8 | November 15th | AI Tools, Programming sins and how to avoid them

Today

- User authentication
- User management and permissions
- Extended exercise

User authentication

Authentication considerations

- Security
- How big is the user base?
  - Smaller apps
    - IP restrictions or host restrictions
  - Larger apps
    - Authentication service
    - Individual user management
- Can create your own or use existing services

Authentication options:

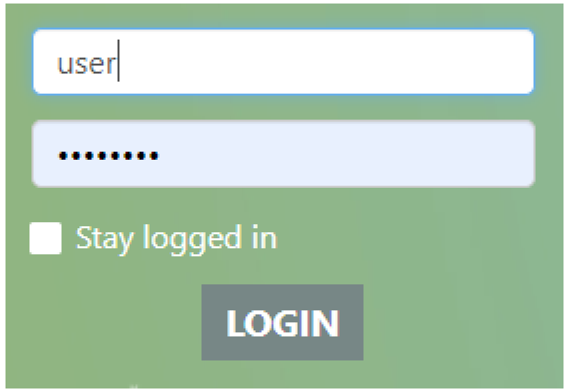
- Built-in
  - **DIY *(today’s topic)***
  - Shinymanager
  - Shinyauthr

- Shinyapps.io
  - Google authentication
  - Github authentication

- Posit Professional Products

Design considerations

- Show user feedback for the different states:
  - Success
  - Failure
  - Processing
- How many fail attempts will the user get?
- Where do we want to put the login?
  - Before application
  - Within application
- Password strength
  - Length
  - Specific characters



EPI-interactive

User management and permissions

Database functions

- dbGetQuery
  - Returns the result of a query as a data frame
  - Used for SELECT
- dbExecute(conn, statement)
  - Executes a statement and returns the number of rows affected
  - Used for INSERT, UPDATE, DELETE

Creating a user

- Check that username is not already taken

```
SELECT uid FROM usermgmt.users WHERE username
= ‘admin’
```

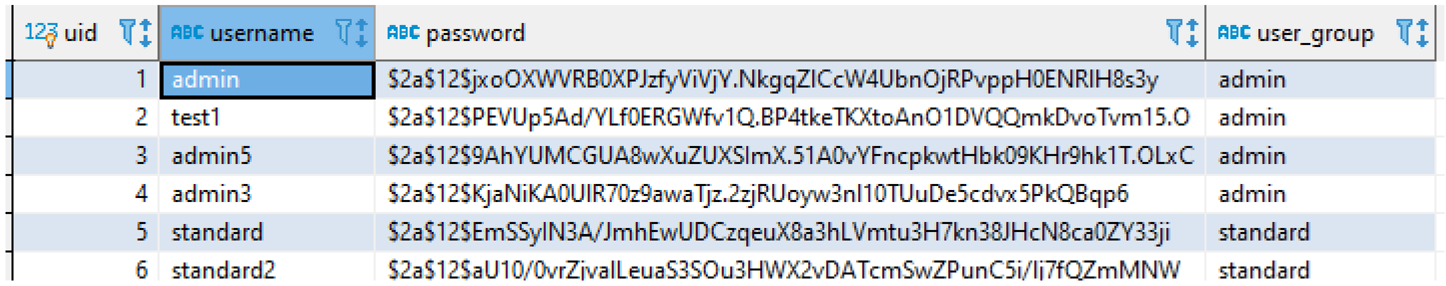
- If not, add user

```
INSERT INTO usermgmt.users(username, password,
user_group)
values('admin', 'password', 'admin’)
```

- If it is taken, display error

Never save passwords in plain text!

- **`bcrypt::hashpw(input$password)`**
- Creates a hashed version of the plain string
- Even if people get access to your userbase, signing in with your user is more difficult



uid	username	password	user_group
1	admin	$2a$12$jxoOXWVRB0XPJzfyViVjY.NkgqZICcW4UbnOjRPvppH0ENRIH8s3y	admin
2	test1	$2a$12$PEVUp5Ad/YLf0ERGWfv1Q.BP4tkeTKXtoAnO1DVQQmkDvoTvm15.O	admin
3	admin5	$2a$12$9AhYUMCGUA8wXuZUXSImX.51A0vYFncpkwtHbk09KHr9hk1T.OLxC	admin
4	admin3	$2a$12$KjaNiKA0UIR70z9awaTjz.2zjRUoyw3nI10TUuDe5cdvx5PkQBqp6	admin
5	standard	$2a$12$EmSSyIN3A/JmhEwUDCzqeuX8a3hLVmtu3H7kn38JHcN8ca0ZY33ji	standard
6	standard2	$2a$12$aU10/0vrZjvalLeuaS3SOu3HWX2vDATcmSwZPunC5i/Ij7fQZmMNW	standard

EPI-interactive

Logging in

- Check that the username exists in the database

```
input$username %in% allUsers()$username
```

- Check the plain text password entered in the form against the hashed version in the database

```
bcrypt::checkpw(
    input$password,
    getPasswordForUser(input$username)
)
```

- getPasswordForUser(username)

```
paste0("select password  from usermgmt.users
where username = '",username,"'")
```

EPI-interactive

Being logged in

- Can create a reactiveVal to keep track of whether a user is logged in
  - This allows us to display different outputs depending on log in status
  - Only log the user in if the password check is successful
  - Remember if the user clicks “log out”, to change the status
- If there is a user logged in, we want to know who they are and their settings
  - Keep track of the user with a unique identifier

User groups

- Design different permissions for different ‘groups’
  - E.g. ‘Admin’ vs ‘User’ vs ‘Guest’ permissions
- Assign users to a User group to grant permissions in the Shiny app
  - Conditional areas of the app (E.g. admin page for Admin users only)
  - Restrict the ways for users to interact with the app
  - Limit the data user groups have access to

- Assigning permissions to a group instead of an individual

Updating and deleting

- Who has permission to do this?
  - User groups – should restrict access to higher levels for Update / Delete actions
- Be specific
  - Only update/delete one user at a time

- Prompt before saving change
- In some cases, allow the user to undo the change

Updating and deleting: SQL

- Update one user at a time using a unique identifier

```
UPDATE usermgmt.users SET
user_group = ‘standard’
WHERE uid = 1
```

- ***Do not delete without including a WHERE clause!***

```
DELETE FROM usermgmt.users WHERE uid = 1
```

Staying logged in

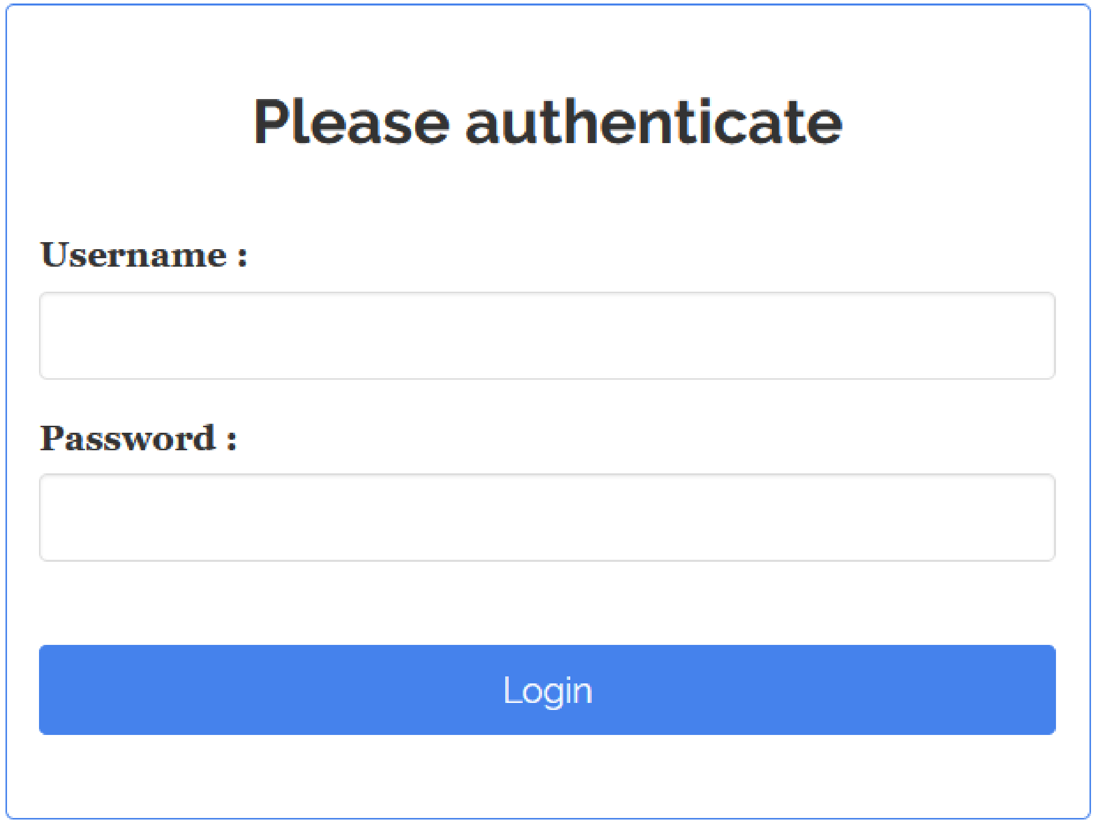
- Initialize a shinyinput to a cookie token
  - For examples sake, we can just use the hashed username as a token and tomorrow's date as the expiry
- Once the user clicks the login button, update the cookie to match the current user
- In an observe function, check if the user is already logged in
  - Can test by refreshing app
- If the user logs out, clear the cookie

Shiny manager

Shinymanager

- Handles authentication and login for us
- Saves login info in SQLite file on the server
- Doesn’t support remote SQL databases by default
- Avoids loading the rest of the UI until authenticated
- Works across multiple applications (if SQLite file is shared)
- Can have an admin section for managing users and logs

Shinymanager


Please authenticate

Username :

Password :

Database Features

- The newest version has built in SQL database features
- These are not yet on CRAN
- Balance that with security requirements about packages that are still in development and not available on CRAN

https://datastorm-open.github.io/shinymanager/

https://github.com/datastorm-open/shinymanager/

Shinymanager

global.R

```
credentials <- data.frame(
  user = c('Test_User'),
  password = c('L3tM3In'),
  admin = c(FALSE),
  stringsAsFactors = FALSE
)
create_db(
  credentials_data = credentials,
  sqlite_path = "auth_db.sqlite",
  passphrase = 'sqlite_db_password'
)
```

server.R

```
shinyServer(function(input, output, session) {

  res_auth <- secure_server(check_credentials = check_credentials(
    'auth_db.sqlite',
    passphrase = 'sqlite_db_password'
  ))
  output$auth_output <- renderPrint({
    reactiveValuesToList(res_auth)
  })

  # Normal server functionality here
  ...
})
```

ui.R

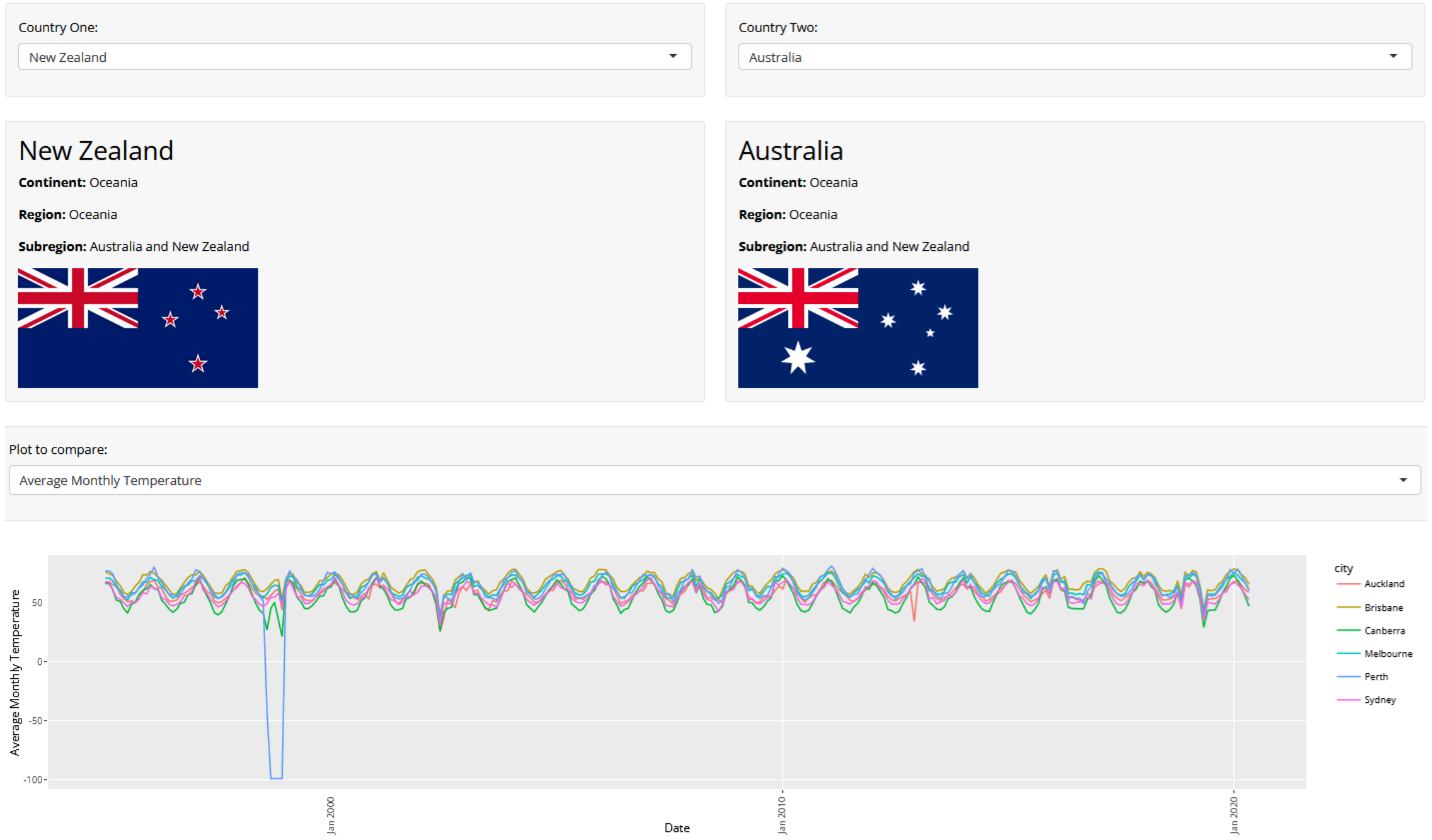
```
ui <- fluidPage(
  ...
)

secure_app(ui)
```

Shinymanager Gotchas

- There is a frequent connection between the user and the shiny server to ensure the user is authenticated, which means we can’t use our normal loading bar (because it shows every second or two)
- The username/password ‘admin’ and ‘admin’ don’t work
- The rest of the UI doesn’t load so if you have inputs that are used on the server, they should have a req() around them
- Must run in browser (the popup closes when you get the correct password)

Exercise



Country One:

New Zealand

Country Two:

Australia

New Zealand

Continent: Oceania

Region: Oceania

Subregion: Australia and New Zealand

Australia

Continent: Oceania

Region: Oceania

Subregion: Australia and New Zealand

Plot to compare:

Average Monthly Temperature

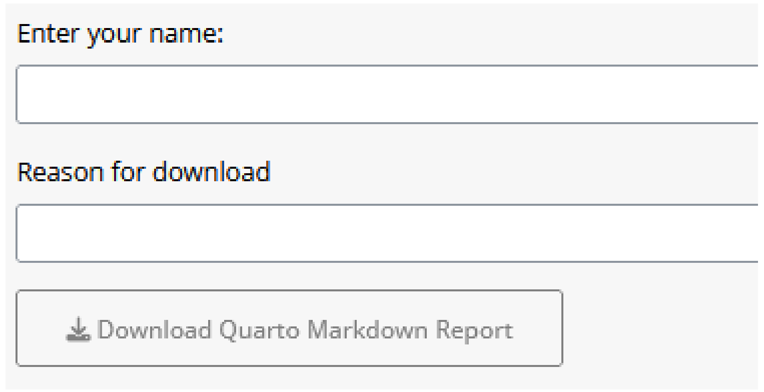
EPI-interactive

Exercise (using /stage1)

- In *compare-country-page-module.R*:
  - Create the layout for the page using bootstap layout (in output$content)
  - Create two selectInputs with all valid countries as the choices.
    - Set appropriate choices and selected values
  - Use *output$countries* to display the summary of these countries
    - Use the compare_country function, passing appropriate parameters
  - Create a selectInput to choose which plot to display
    - Plot for each of: avg_monthly_temp, pop, popDensity, lifeExp, gdpPercap, area_km2
  - Use *output$data_output* to display the selected plot
    - Complete *output$compare_bar* and *output$compare_line* to generate the appropriate plots

Exercise (using /stage2)

- In *quarto-module.R:*
  - *A*dd two textInputs for the user to enter their name and reason for downloading.
  - Make the download quarto button disabled if either textInput is empty
    - See shinyjs::enable(), shinyjs::disable() for enabling/disabling elements, and shinyjs::disabled() for creating an already disabled element
  - Use DBI::sqlInterpolate() and DBI::dbExecute() to insert the name and reason into the 'data_downloads' table in the database
    - Example SQL: "INSERT INTO data_downloads (user_name, reason) VALUES ('Ben', 'Example reason for downloading the data');"
- Add this module to the bottom of you compare country module



EPI-interactive

Exercise (using /stage3)

- In *quarto-module.R:*
  - Generate quarto render function with inputs for html download
    - Inputs: filtered world data, filtered temperature data, both country inputs
  - Copy generated file for user to download
- In *quarto.qmd*:
  - Generate a section for each country, displaying the countries name, continent, region, subregion and flag
  - For each displayed plot in compare counties, generate a quarto section with a title and plot

Stage 3

R Shiny Advanced Masterclass

AUTHOR
Epi-interactive

PUBLISHED
March 25, 2024

Session 7

This exercise will allow you to practise the use of Quarto, integrating Quarto with R / R Shiny code, and exporting a PDF from an Quarto Markdown document through a Shiny application.

In separate paragraphs below, please insert the Quarto Markdown text described in the exercise. Feel free to include more additions to this content:

Today's date: 2024-03-25

Countries Summary

Canada

Continent: North America

Region: Americas

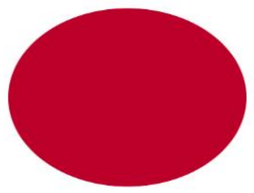
Subregion: Northern America

Japan

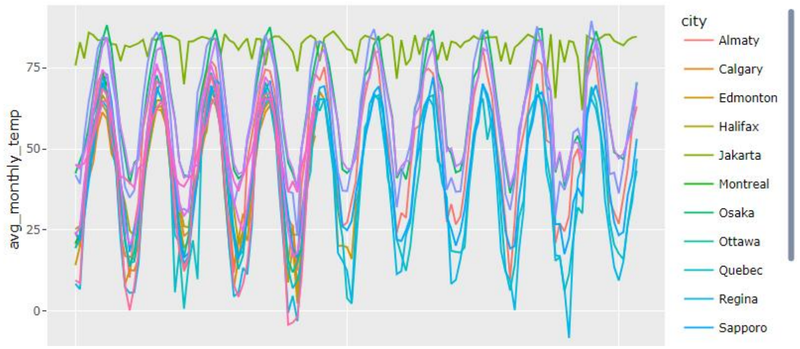
Continent: Asia

Region: Asia

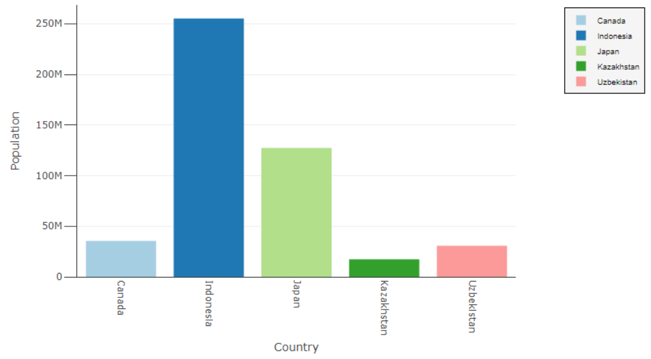
Subregion: Eastern Asia



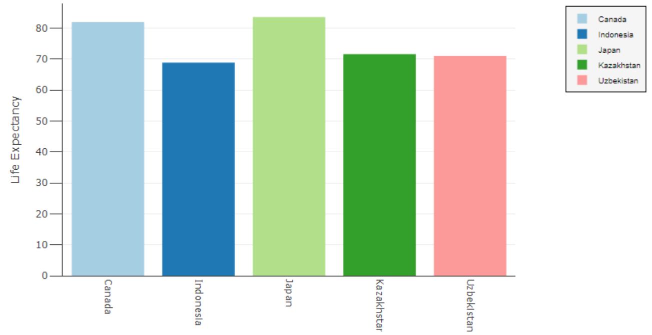
Average Monthly Temperature



Population



Life Expectancy



Exercise: Considerations and hints

- You will need to call a different function in *util.R* for creating the temperature plot (*generate_temperature_plot*) vs the other plots (*generate_compare_plot*)

Next time

- AI Tools discussion
- Programming sins and how to avoid them
- Wrap-up

Challenge:

- Complete the extended Session 7 exercise, and share with us on the forum!
- Using /challenge
    - In *global.R*
        - Create a dataframe with the credentials in it
            - This should use the ADMIN_USER, ADMIN_PW, and AUTH_DB_KEY in your *.env* file
        - Run the create_db() function to initialize the SQLite database
    - In *server.R*
        - Put the credentials check and auth_output at the beginning of the shinyServer function
    - In *ui.R*
        - Wrap the ui in the secure_app() function to ensure it loads the login screen first