Development of A Web Application for MUBI Movie Lovers --Part 3: Web App Design

Team

Jie Zhen, jz67@iu.edu

Xuemei Hu, xh18@iu.edu

Xin Tu, xintu@iu.edu

Summary

Previously, we created the MUBI relational database model with optimized tables and relationships in MySQL workbench using Structured Query Language (SQL). In this part, we will design a web app demo with defined architecture and basic functions. Firstly, we will design the web app architecture: a mock view for front-end and schema for back-end architecture. Next, we will create the backend RSQLite database and import tables by connecting to the database in MySQL workbench. We will build the frontend web application demo with Shiny in R and create several Shiny app demos for different functions such as: search, click, and select to view different results from the database. Finally, we will design the web app layout with basic functions with Shiny in R.

1. Web App Architecture

1.1 Web App Architecture Design

The web application architecture describes the interactions between applications, databases, and middleware systems on the web. It ensures that multiple applications work simultaneously. Our web app is a typical web application which follows MVC schema. Model: The backend MUBI database that contains all the data logic. View: The frontend web or graphical user interface (GUI). Controller: The brains of the application that controls how data is displayed. In our web app architecture (**Figure 1**), the backend was the RSQLite database, and the frontend was the Shiny web app. As shown in Figure 1, the web was connected to the backend RSQLite database through the Shiny server. Users can access the frontend web and input values. Shiny app will process input values to produce the output results by querying from the backend database.

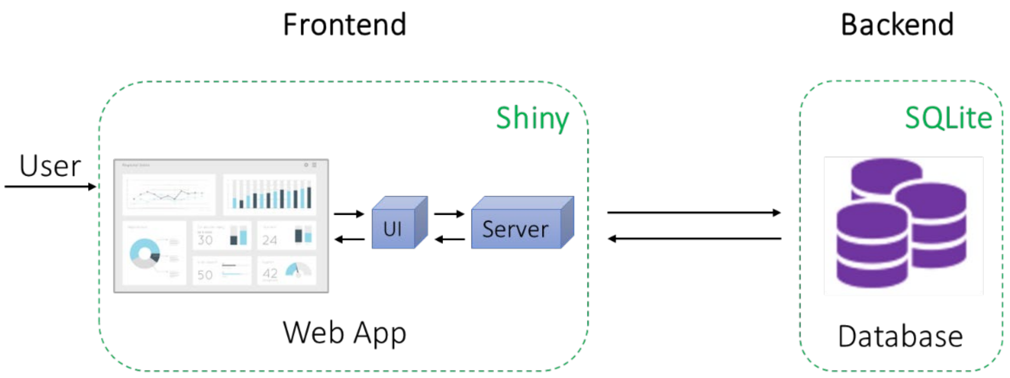
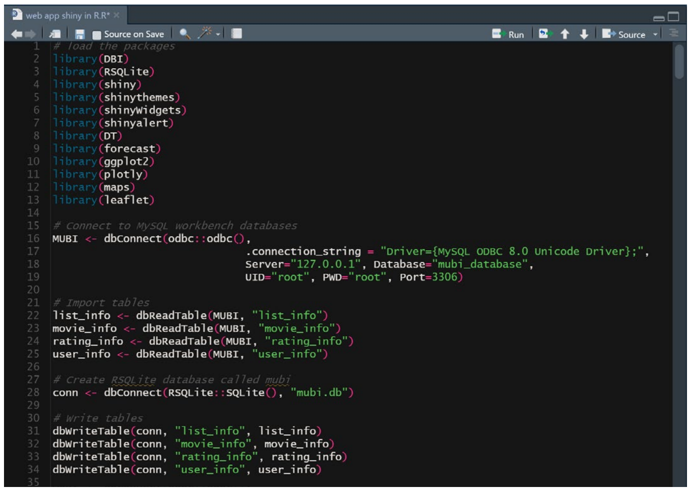


Figure 1. Web app architecture

1.2 Backend Database

To facilitate interactions between R and SQLite databases, we use the R package RSQLite. It is a powerful tool that embeds the SQLite database engine in R, and that leverages all of the database interaction functionalities also found in other packages like RMySQL and ROracle. Previously, we created the MUBI relational database model with optimized tables and relationships in MySQL workbench using Structured Query Language (SQL). To import the MUBI relational database, we connected to the MySQL database server using the dbConnect function in RStudio (**Figure 2**). Next, we created a RSQLite database and wrote the MUBI tables into the RSQLite database in RStudio (**Figure 2**). All data was stored, managed, and updated in the RSQLite database. We then created a connection using the dbConnect function to access the RSQLite database. To secure the database, we set up credentials for this connection.

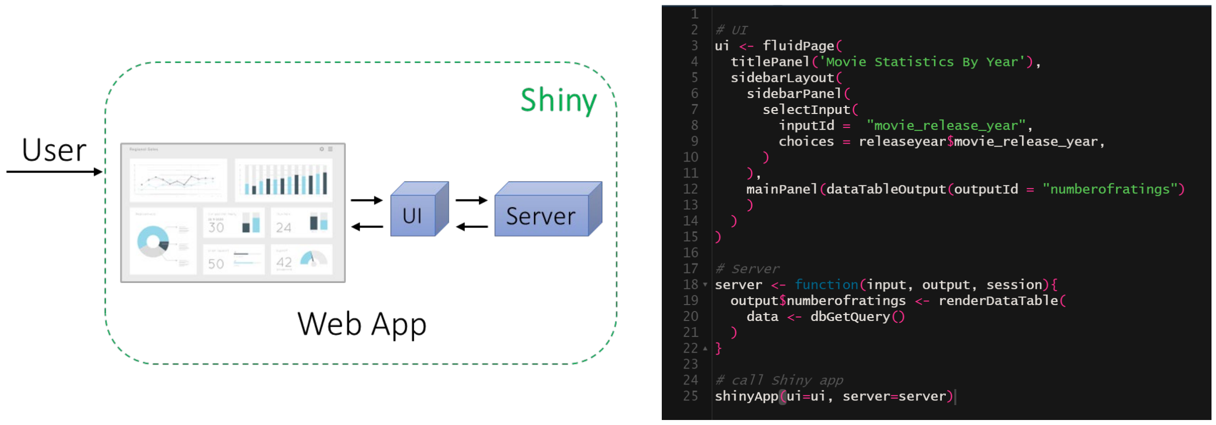


```r
# load the packages
library(DBI)
library(RSQLite)
library(shiny)
library(shinythemes)
library(shinyWidgets)
library(shinyalert)
library(DT)
library(forecast)
library(ggplot2)
library(plotly)
library(maps)
library(leaflet)

# Connect to MySQL workbench databases
MUBI <- dbConnect(odbc::odbc(),
                              .connection_string = "Driver={MySQL ODBC 8.0 Unicode Driver};",
                              Server="127.0.0.1", Database="mubi_database",
                              UID="root", PWD="root", Port=3306)

# Import tables
list_info <- dbReadTable(MUBI, "list_info")
movie_info <- dbReadTable(MUBI, "movie_info")
rating_info <- dbReadTable(MUBI, "rating_info")
user_info <- dbReadTable(MUBI, "user_info")

# Create RSQLite database called mubi
conn <- dbConnect(RSQLite::SQLite(), "mubi.db")

# Write tables
dbWriteTable(conn, "list_info", list_info)
dbWriteTable(conn, "movie_info", movie_info)
dbWriteTable(conn, "rating_info", rating_info)
dbWriteTable(conn, "user_info", user_info)
```

Figure 2. Create the RSQLite database in R

1.3 Frontend Web App

Next, we built the frontend web application demo with Shiny in R. Shiny allows us to easily build interactive web applications in RStudio without knowledge of web development such as CSS, JavaScript and PHP. Shiny app has three components (**Figure 3. left**). A user interface (UI): the frontend that accepts user input values. A server function (server): the backend that processes input values to produce the output results. A call to the Shiny app function: the app itself that combines the UI and the server. The basic codes were used to create the Shiny web app in RStudio (**Figure 4. right**).

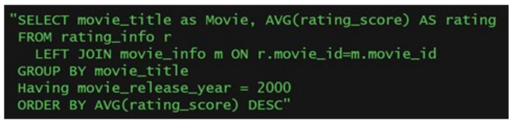


```r

# UI
ui <- fluidPage(
  titlePanel('Movie Statistics By Year'),
  sidebarLayout(
    sidebarPanel(
      selectInput(
        inputId =  "movie_release_year",
        choices = releaseyear$movie_release_year,
      )
    ),
    mainPanel(dataTableOutput(outputId = "numberofratings")
    )
  )
)

# Server
server <- function(input, output, session){
  output$numberofratings <- renderDataTable(
    data <- dbGetQuery()
  )
}

# call Shiny app
shinyApp(ui=ui, server=server)
```

Figure 3. Frontend web app: (**left**) schematic illustration of how a Shiny web app works, (**right**) sample codes for Shiny app

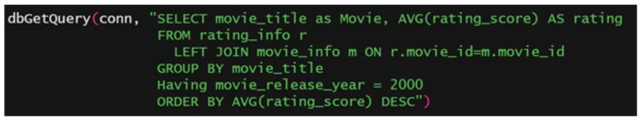
We created several Shiny app demos for different functions such as: search, click, and select to view different results from the database. There are three steps in the process: 1) compiling a database query with SQL, 2) issuing that query with RSQLite, and 3) embedding that query in Shiny app (**Figure 4**). As shown in Figure 4, we made several SQL queries to extract useful information (**Figure 4A**). Next, we executed these queries in the RSQLite database (**Figure 4B**). We embedded these queries in the Shiny app server to get output from the RSQLite database and return the output to the UI (**Figure 4C**). Results were returned to users on the Shiny web app page (**Figure 5**).

A



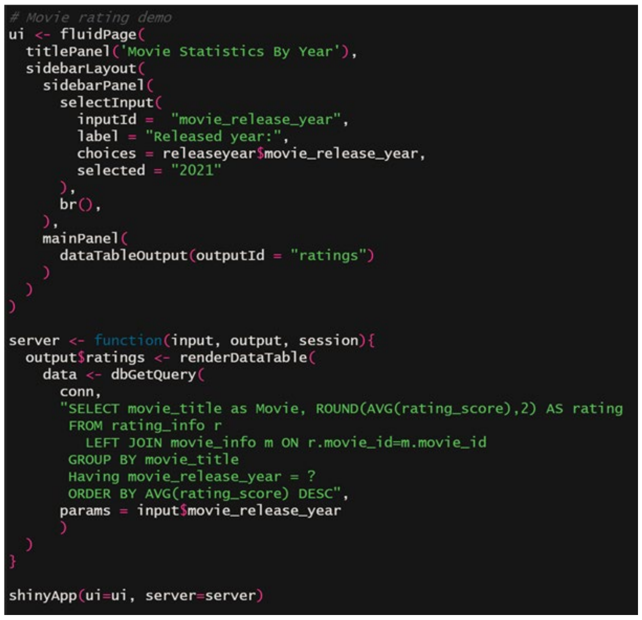
```
"SELECT movie_title as Movie, AVG(rating_score) AS rating
 FROM rating_info r
   LEFT JOIN movie_info m ON r.movie_id=m.movie_id
 GROUP BY movie_title
 Having movie_release_year = 2000
 ORDER BY AVG(rating_score) DESC"
```

B



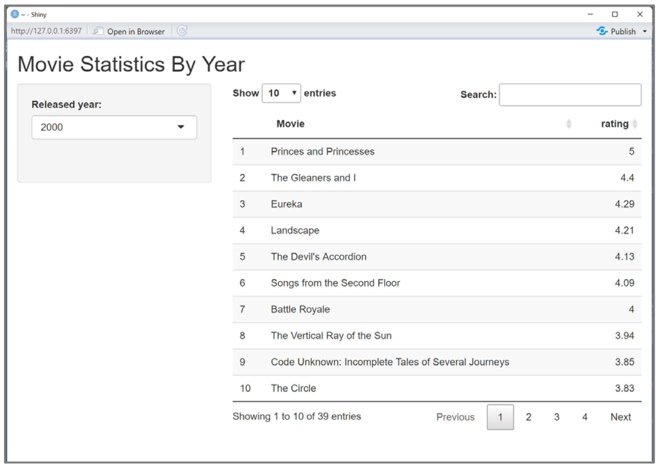
```
dbGetQuery(conn, "SELECT movie_title as Movie, AVG(rating_score) AS rating
                  FROM rating_info r
                    LEFT JOIN movie_info m ON r.movie_id=m.movie_id
                  GROUP BY movie_title
                  Having movie_release_year = 2000
                  ORDER BY AVG(rating_score) DESC")
```

C



```
# Movie rating demo
ui <- fluidPage(
  titlePanel('Movie Statistics By Year'),
  sidebarLayout(
    sidebarPanel(
      selectInput(
        inputId =  "movie_release_year",
        label = "Released year:",
        choices = releaseyear$movie_release_year,
        selected = "2021"
      ),
      br(),
    ),
    mainPanel(
      dataTableOutput(outputId = "ratings")
    )
  )
)

server <- function(input, output, session){
  output$ratings <- renderDataTable(
    data <- dbGetQuery(
      conn,
      "SELECT movie_title as Movie, ROUND(AVG(rating_score),2) AS rating
       FROM rating_info r
         LEFT JOIN movie_info m ON r.movie_id=m.movie_id
       GROUP BY movie_title
       Having movie_release_year = ?
       ORDER BY AVG(rating_score) DESC",
      params = input$movie_release_year
      )
  )
}

shinyApp(ui=ui, server=server)
```

Figure 4. Three steps for creating Shiny app demo pages: (**A**) SQL query, (**B**) query with RSQLite database, (**C**) embedding query in Shiny app demo.


Movie Statistics By Year

Released year: 2000

Show 10 entries　Search:

	Movie	rating
1	Princes and Princesses	5
2	The Gleaners and I	4.4
3	Eureka	4.29
4	Landscape	4.21
5	The Devil's Accordion	4.13
6	Songs from the Second Floor	4.09
7	Battle Royale	4
8	The Vertical Ray of the Sun	3.94
9	Code Unknown: Incomplete Tales of Several Journeys	3.85
10	The Circle	3.83

Showing 1 to 10 of 39 entries　Previous 1 2 3 4 Next

Figure 5. Shiny app demo page.

2. Web App Layout

2.1 Main Layout

We designed the web app layout with basic functions using Shiny in R (**Figure 6**). We used a basic and common color schema which is white background and black text. The initial layout was the home page which contained welcome text and images. We also created several links such as search, find and connect for users to jump to the linked navigation panels with specific functions. The navigation panel served as the menu of this web app. It is located on the top of the web app page. These navigation panels provided users an easy way to access, search and analyze the MUBI movie data.

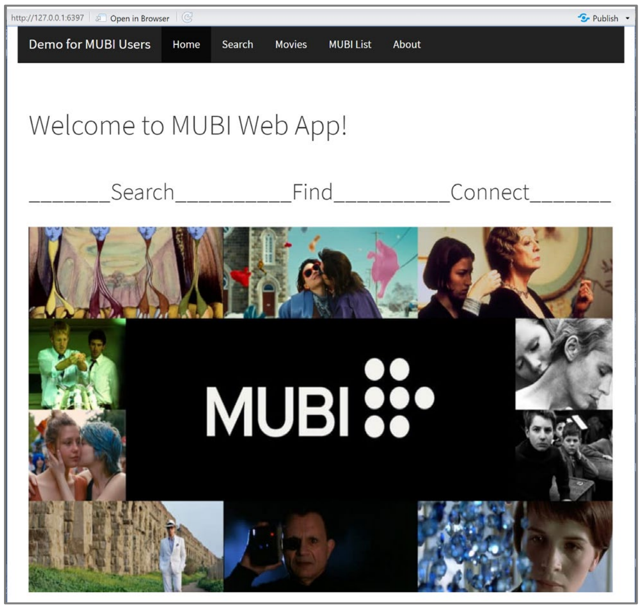
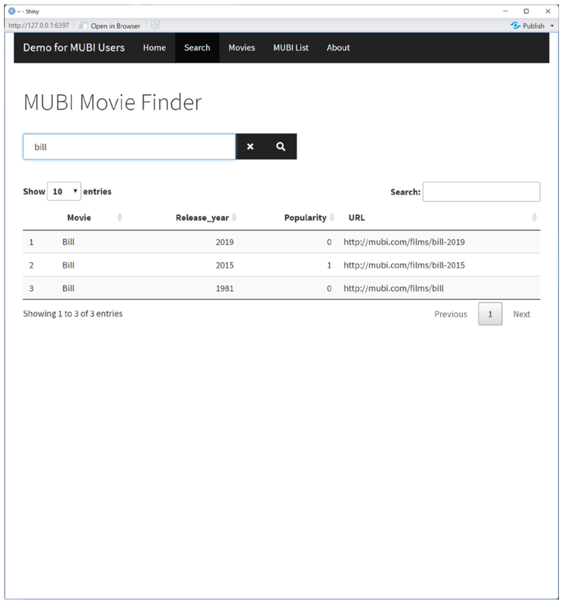


Figure 6. Initial layout of the MUBI Shiny web app

2.2 Navigation Panels

We created five panel pages for this web app: Home, Search, Movie, List, and About. The home page showed the welcome text, images, and links. The search panel provided the search function for users to search the movie information such as movie name, movie release year, movie popularity and URL link by movie name input (**Figure 7. left**). There was a text input box for users to text movie names for searching. The main panel of the search page will show the search results from the database. The Movie panel was created to find the movie information by selecting the movie release year (**Figure 7. right**). There was a select input which allowed users to select a specific movie release year. On the left side panel, three tabs can display three different outputs: the movie rating number, average rating, and movie popularity (**Figure 7. right**). Users can get the rating information for each movie so that they can choose what movie he/she will watch. List panel can plot some analytic plots by sliding the list movie number. It had a slider input and plot output panel. The last panel displayed the introduction of the designers of the website.



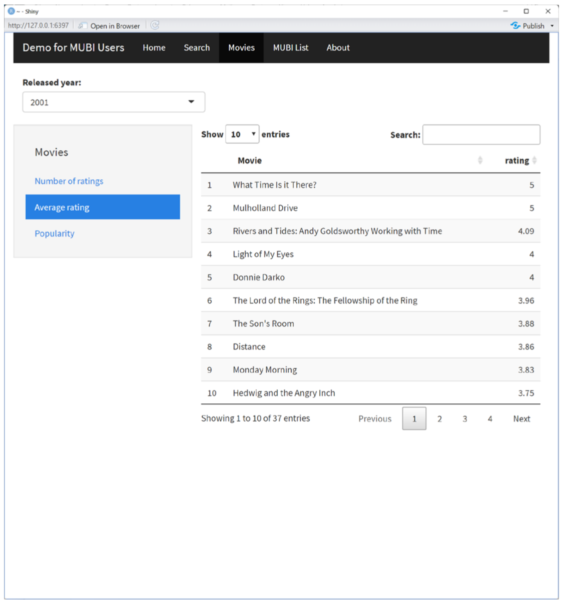
MUBI Movie Finder

bill

Show 10 entries   Search:

	Movie	Release_year	Popularity	URL
1	Bill	2019	0	http://mubi.com/films/bill-2019
2	Bill	2015	1	http://mubi.com/films/bill-2015
3	Bill	1981	0	http://mubi.com/films/bill

Showing 1 to 3 of 3 entries   Previous 1 Next



Released year: 2001

Show 10 entries   Search:

	Movie	rating
1	What Time Is it There?	5
2	Mulholland Drive	5
3	Rivers and Tides: Andy Goldsworthy Working with Time	4.09
4	Light of My Eyes	4
5	Donnie Darko	4
6	The Lord of the Rings: The Fellowship of the Ring	3.96
7	The Son's Room	3.88
8	Distance	3.86
9	Monday Morning	3.83
10	Hedwig and the Angry Inch	3.75

Showing 1 to 10 of 37 entries   Previous 1 2 3 4 Next

Figure 7. The tab panels in the MUBI Shiny web app.

(**left**) the search panel, (**right**) the movie panel.

Next, we will manage UI inputs and outputs to optimize the Shiny web app with full user functionalities.

3. Teamwork

Team member	Task contributions
Jie Zhen	Learn the web app design and web app layout
Xuemei Hu	Describe the web app layout, make queries for the RSQLite database
Xin Tu	Design the web app architecture, design the web app layout with basic functions: search, analysis.

References

https://shanghai.hosting.nyu.edu/data/r/case-3-sql-shiny.html

https://shiny.rstudio.com/gallery/real-estate-investment.html

https://deanattali.com/blog/building-shiny-apps-tutorial/#:~:text=You%20can%20also%20create%20a,R%20%2B%20server.

https://budibase.com/blog/web-application-development/

https://www.kaggle.com/datasets/clementmsika/mubi-sqlite-database-for-movie-lovers?select=mubi_movie_data.csv

https://www.christophenicault.com/post/improve_shiny_ui/