DCSI-D523 Applied Database Technologies

Spring 2023

Final Project, Part 3

Date: 15 April 2023

Team: **Alex Bordanca and David Thiriot**

Title

Web App Design for our Medical Practice Database

Introduction

This report describes the web app design for our database, designed to support the data needs of a small medical practice. The report will discuss the web app architecture, the web app layout, and include reflections from each team member (Alex Bordanca and David Thiriot).

The data in our database is a set of 1000 patients and 10 doctors for which we simulated realistic names, addresses, personal information, health information, and financial information. We created this simulated dataset using R (1) and R packages including ‘randomNames’ (2) and ‘charlatan‘ (3). The setting for our medical practice is an area comprising 6 zip codes within New York City.

Web App Architecture

The front-end user interface (ui) and “controller” (or server) functions of our web app are accomplished using R Shiny (4). The back-end database uses RSQLite (5). Connections between the Shiny App and the RSQLite database (commands including dbConnect, dbExecute and dbGetQuery) are facilitated using the R package ‘DBI’ (6).

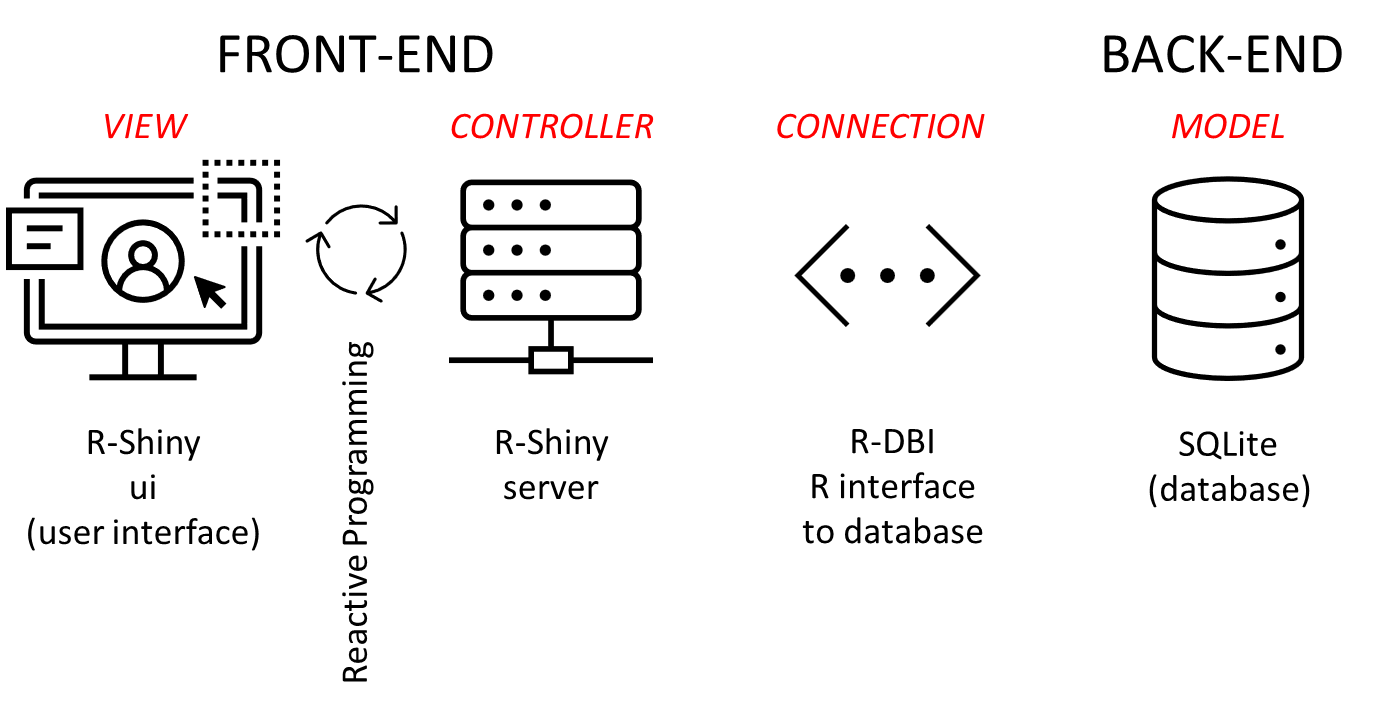
We selected SQLite as a database because of its ease of implementation, being largely self-contained and not requiring third party software or server connections to run. While we simulated our data and ultimately created our web app using R, we also demonstrated database functionality in our Project Part 2 submission using Python. Both R and Python have effective interfaces to SQLite databases.

Our intent for the database in this project assignment is that those who have access to the database (for example, staff at a medical practice) can view patient records and update certain fields within patient records. We did not implement a multi-tiered access scheme for our database. Admittedly, our implementation for this assignment would need more development to serve as a real database supporting a medical practice.

We chose Shiny as a front end because it provides a reactive user interface that is responsive to user input, dynamically showing data and updating user views and providing an interactive experience. Briefly, writing a Shiny app involves writing a user interface “ui” section and a controlling “server” section, then calling those two sections together. The server is “listening” and reacting to input the user provides. User input and server output are stored in list variables “input” and “output”. Server output and user interface output use matched pairs of “render” and “output” functions, render in the server and output in the ui functions. Shiny has functions to handle multiple page views or tabs, and a variety of simple user input devices like selection pulldown menus, calendar date selection tools, radio buttons, and text input boxes. Shiny is a good way for beginners to generate the user-interface webpage displays, without needing to have expert HTML or CSS programming skills. Shiny was originally implemented for R, but a recent implementation for Python (7) is growing in popularity. We wrote our Shiny App in R but given time and resources moving our Shiny App to Python appears to be relatively straightforward to accomplish if it were needed.

We deployed a version of our Shiny App on the service shinyapps.io, and plan to deploy the final version of the App in the future. Shinyapps.io offers a free starter account, but the amount of time an App can run for free on the service is limited to only 5 hours, so (not wanting to exceed that) most of our testing during development was done on local computers.

The schematic below illustrates our web app architecture:



Web App Layout

At present, our App has several data tabs including the following:

* View Patient Information
* Update Health Information
* Update Financial Information
* Update Patient Personal and Contact Information
* Update Patient Vitals (implemented, not shown in figures below)
* Add New Patient
* Visualization dashboard (in development, not shown in figures below)

When executed, the App starts on the View Patient Information tab. The user enters a Patient ID number, and the information about that patient is displayed as shown below.

Graphical user interface, text, application

Description automatically generated

If a new patient joins the practice, they need to be entered into the database. This can be done through the Add New Patient tab. The Shiny interface has some convenient user input functions to help enter the information, as illustrated below.

This image illustrates the textInput , radioButtons and selectInput user interface functions.

Graphical user interface, application, Word

Description automatically generated

The next two images illustrate a dateInput interface, to get a new patient’s date of birth.

Graphical user interface, application

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

A third example from the subtab Update Financial Information illustrates dynamic calculation and display of the patient account balance. After selecting simulated patient Thomas Anderson (Patient ID #45), we see he has an account balance of $3600.00. The medical practice staff adds the information that he incurred a new bill of $40.00, and made a payment of $300. When the actionButton “Click Here to Update Amount Due in Database” is clicked, the new amount due of $3340.00 will be recorded to the SQLite database.

Graphical user interface, text, application, email

Description automatically generated

The initial entry point is the View Patient Information tab, without a patient selected. The user navigates the web app by selecting the appropriate tab at the top of the app page. There are 6 functional tabs now, and a visualization tab in development.

In the tab View Patient Information, the user input is the Patient ID number, and data from the patient from 7 different SQLite tables is displayed. If the Patient ID does not exist, no data is displayed.

In the tab Update Health Information, the user inputs a Patient ID number. At that point, a subset of data about the patient is displayed and the user is asked which health condition (out of 11) needs updating? If a condition needs updating, the Shiny App dynamically adds and input field to accept the updated information. At the bottom of the tab there is an actionButton that when pressed results in the update to the SQLite database being performed.

In the Update Financial Information tab, the user inputs a Patient ID number, and the account balance Amount Due for the patient is displayed. Then the user can enter an amount for a new bill, or a payment made, or both. A new account balance for the patient is calculated. After clicking the actionButton at the bottom of the tab, the update is made to the SQLite database.

In the tab Update Patient Personal and Contact Information, the user inputs a Patient ID number, and certain information about the patient is displayed. The user is asked whether they want to update the name, current patient status, assigned doctor, email, street address, and zip code (within the set of zip codes that this medical practice services). Two items here of particular note are the following. First, rather than fully deleting a patient from the database, we chose to change a flag called Current\_patient from 1 (active patient) to 0 (not a current patient). That way, records are retained, but a query can be performed to exclude patients who are not current patients, if needed. Second, while we originally wanted to offer the option of a patient changing their name, we recognized some complications with that for the coding. After trying unsuccessfully to resolve those, we decided that our current approach would be to provide a message indicating that a name change should be managed by creating a new patient record.

In the tab Update Patient vitals, the user inputs a Patient ID number and information about that patient, including height, weight, heart rate, and blood pressure, is displayed. The user is asked if they want to update these data, and if the answer is yes, inputNumeric fields dynamically appear for data input. After adding necessary data, clicking the actionButton results in updates to the SQLite database.

Another functional tab is Add New Patient. The user can provide information about the new patient using the Shiny functions textInput, dateInput, radioButtons, selectInput, numericInput, and checkboxInput. When the new patient record is complete, the user clicks a labeled actionButton to add the new patient to the database. A new unique Patient ID is assigned to the new patient. We also implemented at this stage a check, that a new patient record that has the same Firstname, Lastname and Date of Birth as an existing patient record cannot be created. A message that the database update failed will be returned in such a case, and if that very improbably scenario were to actually occur (two people with the exact same name and birthday), the database could be extended to include a middle name, or some kind of name extension could be used if desired.

The final tab, in development, is a visualization dashboard. The current scope of this tab is to visualize the patient population vital statistics density distribution like height, weight, etc, aggregated by gender. Another functionality that we’re implementing is measuring a patient’s vital statistics in relation to the distribution from his/her gender. We may expand this to other non vital statistic information, time and development skills permitting, though given the simulated and therefore random dataset, we may not uncover any interesting or meaningful trends.

Note that while our App is currently functional, there are things that we recognize could be improved. We hope to add some color and design to make it more visually appealing over the next week. (The current Project Part 3 version is all or mostly black and white.) We could also explore adjustments to layout using for instance the sidePanel function in Shiny. We also note that the two tabs Transaction Confirmation and Transaction Failure are simply messages indicating successful completion of a database update, or failure to update. Ideally, these would be handled in a different way than as a visible tab, but we haven’t figured out how to hide them or change that yet.

Reflections / Work Assessments by Team Members

Alex Bordanca

This project has been both very intriguing and very challenging. I initially thought a good implementation for our goal was a Django web app. In preparation for the CRUD features, I constructed a simplified dummy app. As the scope of the project grew, determined by multiple tables or models, more and more work was needed to be done on the front end in addition to back end challenges stemming from our attempts at foreign key implementations. With that in mind, I was quite happy when David proposed the idea of using R Shiny for our implementation. This was a new framework for me, but with David’s guidance and the documentation, I feel we were able to make significant strides in the development of the web app that far exceeded our at-then present stalling with our Django project. I hope that we are able to make further improvements to both the design and the functionality of the app prior to the delivery date. Overall, we have spent a great deal of time working out the details and collaborating on this project at all stages of development, and in that process overcome many challenges. I would rate myself on the task completion score an 8 out of 10. As David mentions, in terms of making a real life business use case for it, several fundamental changes would need to be made to both the database structure as well as the web app.

Likewise, I have throughout enjoyed working with David, and I feel we make a good team with each of our knowledge or skill gaps being filled by the other’s strengths.

David Thiriot

I have wanted to learn R Shiny for a long time, but always been busy with other priorities. This project finally provided me the opportunity and motivation to spend some time and create a Shiny App. It has been challenging to focus both on core functionality and “nice looks” at the same time, and core functionality has received most of my attention so far. The amount of time I have spent has been more than I expected, and as much time as I have been able to spend. I am satisfied with the effort and output and give myself a task completion score of perhaps 8 out of 10 for the project, but to make this project database really serve for a medical practice, several upgrades would need to be performed that could easily double or triple the amount of time we have for the project. One of the most important upgrades would be to incorporate a way to keep all historical medical data in the database. At present, the database stores just one height, weight, blood pressure, etc. But a real medical database would need to store all that information, over time. That would be a fundamental change and improvement in the database, that we did not plan and build into our work for this project.

I have enjoyed working with my teammate Alex on this and a prior project. I appreciate him and think we work together well.

References

1. R Core Team (2022). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.
2. Damian W. Bertbenner (2021). randomNames: Function for Generating Random Names and a Dataset. URL https://cran.r-project.org/package=randomNames.
3. Chamberlain S, Voytovich K (2022). \_charlatan: Make Fake Data\_. https://docs.ropensci.org/charlatan/ (website) https://github.com/ropensci/charlatan (devel).
4. Chang W, Cheng J, Allaire J, Sievert C, Schloerke B, Xie Y, Allen J, McPherson J, Dipert A, Borges B (2002). \_shiny: Web Application Framework for R\_. https://shiny.rstudio.com/
5. Muller K, Wickham H, James DA, Falcon S (2023). \_RSQLite: SQLite Interface for R\_. https://rwqlite.r-dbi.org, https://github.com/r-dbi/RSQLite.
6. R Special Interest Group on Databases (R-SIG-DB), Wickham H, Muller K (2022). \_DBI: R Database Interface\_. https://dbi.r-dbi.org, https://github.com/r-dbi/DBI
7. Shiny for Python. https://shiny.rstudio.com/py/index.html