DSCI-D 590 Final Project Part 3
LE Lee | Andrew Mouat | Nick Perry

Part 1 - Web App Architecture

Data Storage

Our data is currently stored locally but we are exploring options for storing it remotely via MongoDB or MySQL as both of these options offer an associated R Package (mongolite and RMySQL, respectively). However, due to the complexity of continually updating our data utilizing our custom-built Zillow scraper, local storage makes the most sense for the time being. In the future, if our team elects to store the data remotely, we will publish a static copy of the dataset at that time as deployment of the custom web scraper is outside the scope of this project.

Languages for Back End

We intend to use R to build the back-end of our app as this is what we have been utilizing thus far to generate the data pre-processing, modeling and forecasting workflow. As mentioned in previous submissions, the custom-built Zillow scraper was programmed in Python, but is only used to generate the input dataset for our R Script. As outlined above, for the purposes of this project, our team will assume a static dataset as input and not incorporate dataset updating into our back-end code. Using R as the back-end programming language will also allow us to easily build an interactive web page in Shiny (see "Languages for Front End" section below). We will be using an extensive list of packages and libraries, including but not limited to: fable, forecast, tidyr, imputeTS and ggplot2.

Data access

Currently, our app is connected to the data via CSV upload – this is an administrative privilege as the files are currently housed locally and require username and password authentication to access. Upon deployment (see "Deployment Location" section below), the CSV file will be uploaded to the deployment server along with the application file. Only data uploaded with the application is available to the application and the deployment server allows for server administrative controls to be implemented. Thus, users will not have access to upload their own datasets. Since the data is freely and publicly available, there will not be user logins. This will ensure the data's integrity and will avoid issues if the original data source goes offline. As we add to the data set, we can also maintain versions of the CSV files as a form of version control. If remote storage utilizing MongoDB or MySQL is implemented, our team will investigate proper means of data security for these services at that time.

Languages for Front-End

The front-end of our app will be built utilizing Shiny R functions. More specifically, our team will first utilize built-in "layout functions" to provide the high-level visual structure of a multi-page application (title panel, sidebars, tabsets, etc.). After the high-level structure is built, our team will utilize several other Shiny R functions to generate Shiny widgets that allow for user input including drop-down menus, sliders, and buttons. Additionally, our team will implement interactivity into our displayed plots. Plots in Shiny can respond to four different events: click, double click, hover and rectangular selection. Our team will utilize these features to enhance the usability of our application. As needed, our team will incorporate CSS themes and HTML widgets, if they are applicable and useful to our user interface above what is readily available in Shiny. Some possibilities would be to add time series charting or interactive graphics from the dygraphs and Plotly HTML widget libraries.

Deployment Location

Our application will be deployed on a Shiny server. We will most likely publish to shinyapps.io, RStudio's hosting service for Shiny Apps, since it provides free hosting and allows for file upload directly from an RStudio session. Shinyapps.io also will not require any maintenance of a physical or virtual server. The security of the hosting site is also not a concern, as we are not using any sensitive data that is not already publicly available.

Interactivity

As mentioned in the "Languages for Front-End" section above, our application will provide several means of interactivity. For example, users will be able to input parameter values of interest pertaining to homes – e.g. zip code, square footage, number of bedrooms, and/or number of bathrooms via drop down menus or input fields. They will also be able to specify a certain amount of time to forecast the price of a home that matches the parameters specified and change the model used for forecasting. Furthermore, the graphics generated following the user input (historical data and forecasts) will offer interactive features such as hovering (tool tips), zooming via rectangular selection, etc.

Web App Architecture

Figure 1 below provides a depiction of the web app architecture that our team will utilize.

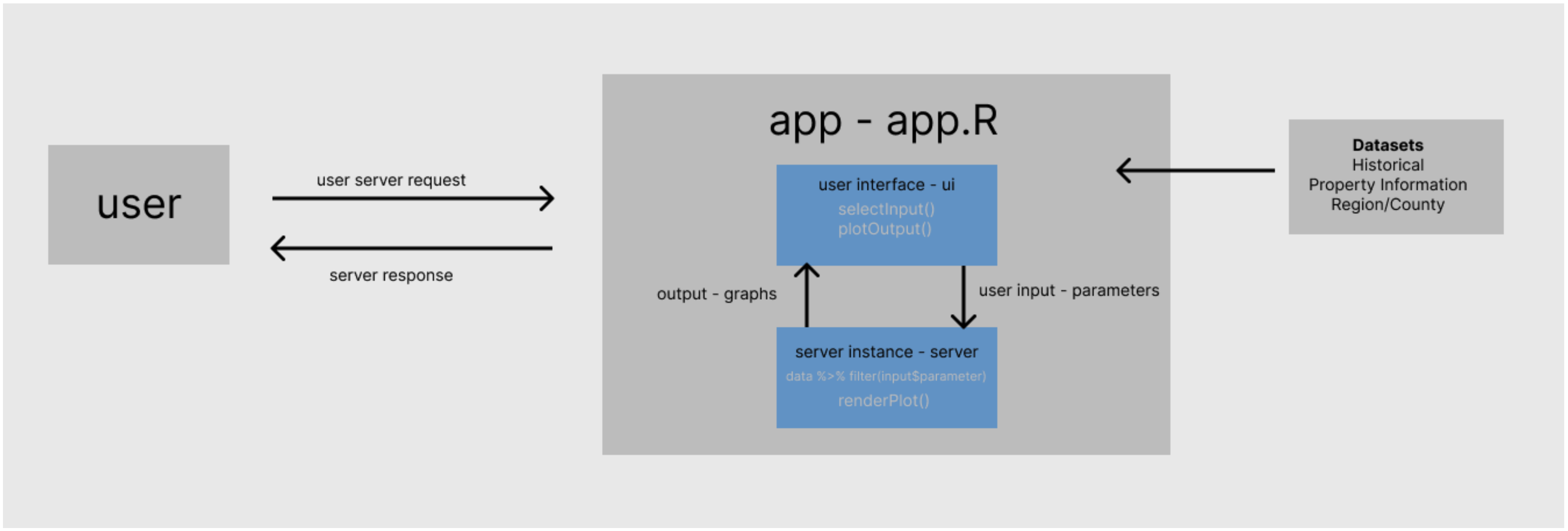


Figure 1: Web App Architecture Depiction

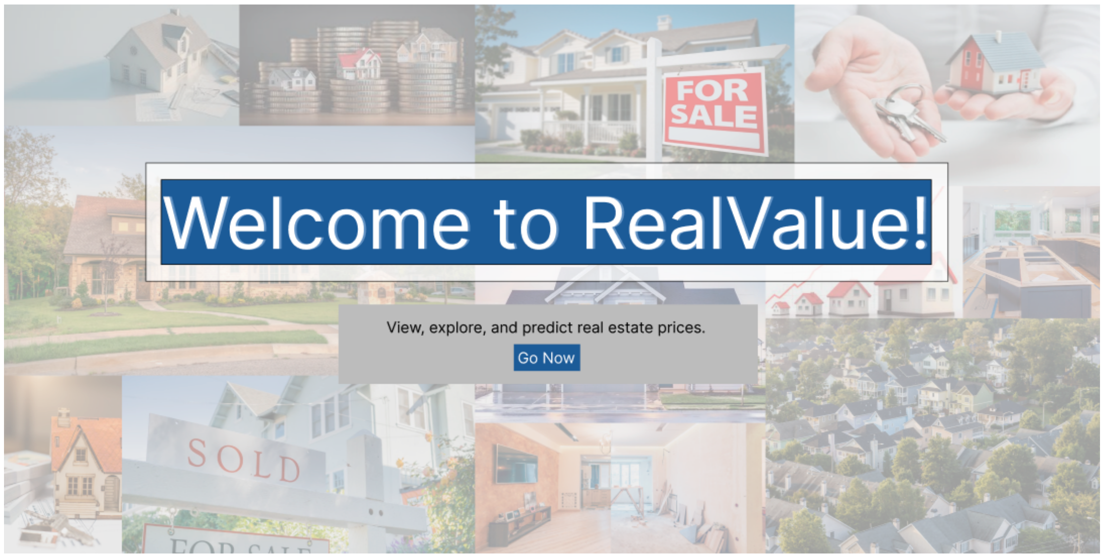
The datasets generated by our team's custom-built web scraper are initially fed into R as CSV files. As previously mentioned, these CSV files will either be uploaded to the Shiny server along with the application file, or the data will be remotely stored (MongoDB, MySQL, etc.). The user's server request (via navigation to the associated URL or interactions) requests the application to run. The user's input, the relevant parameters for the time series analysis, is fed into the R application via the user interface and run through the server instance as a Shiny function. The output graphics are then returned to the user. The user can enter a cycle of inputting parameters, returning an output graph, and then interfacing with the graph, as mentioned in the "Interactivity" section above.

Part 2 - Web App Layout

At a high-level, our team plans to create a two-paged web application, excluding the landing page that the user will first see. Specific details about these pages ("Historical View" and "Forecasted Value View") are provided below. Navigation between the two pages will be accomplished through the use of tabs or automatic navigation on button-clicks. For both pages, the layout will include a menu or control panel on the left-hand side of the page along with a main viewing panel on the right-hand side of the page. Initially, the overall color scheme will be a combination of blues, greys, and white. However, our team will evaluate other color schemes utilizing ColorBrewer2to ensure our web application is accessible and print friendly.

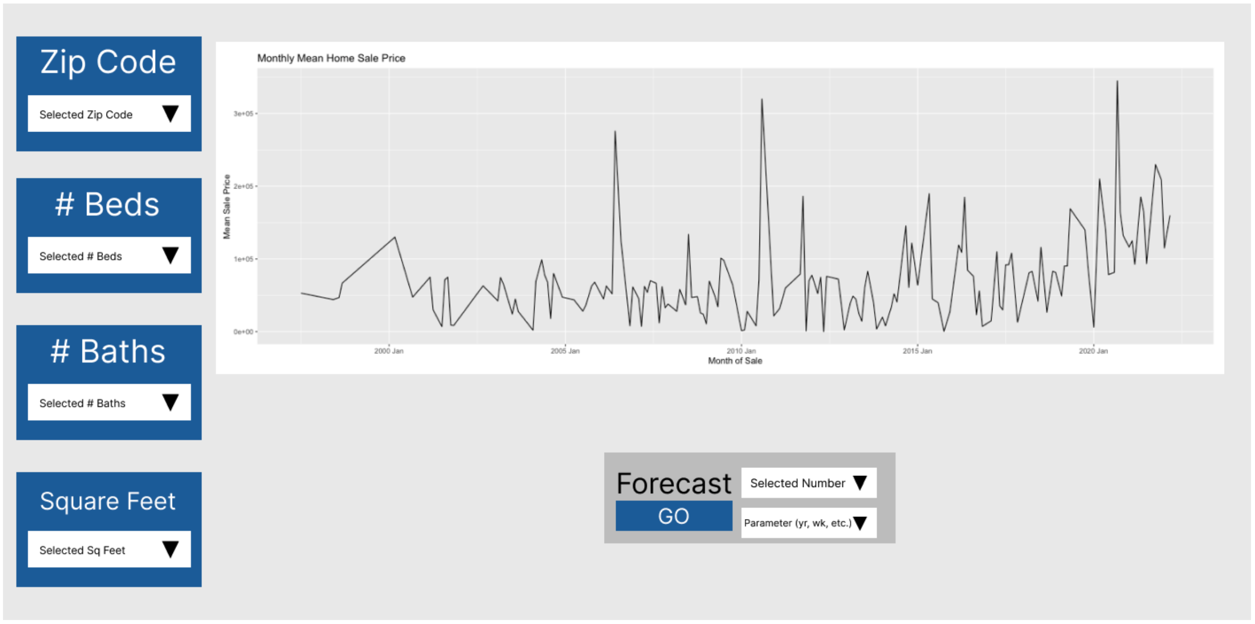
Welcome Page

This page serves as a welcome page for the user. Clicking the "Go Now" button takes the user to the "Historical View" page.



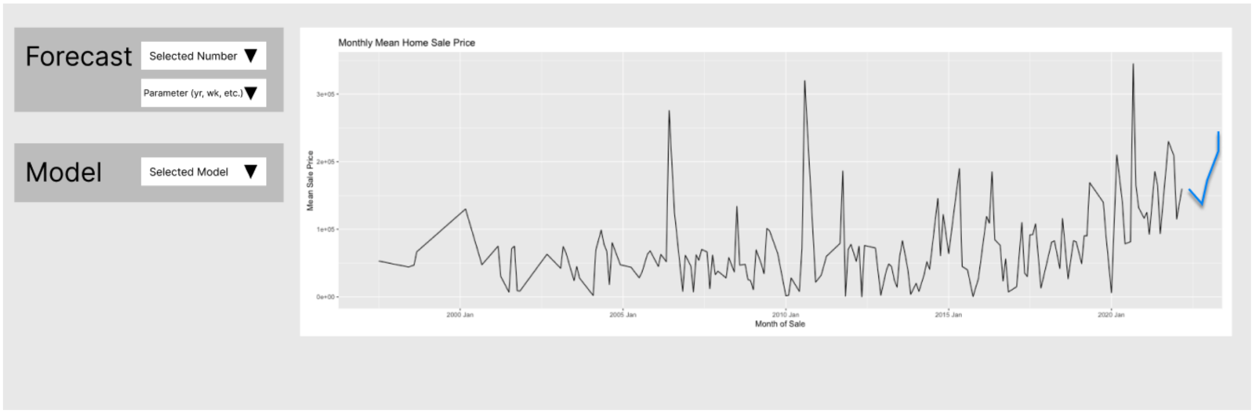
Historical View

On this page, users can input & select their parameters of interest among zip codes, number of beds, number of baths, and square footage. These parameters are preliminary, and we expect to incorporate additional options. After users input these parameters, a plot of historical monthly mean home sale prices matching those parameters will be shown. Users then have the option to forecast sale prices for homes matching these parameters by utilizing the forecast box. Pressing "Go" will take them to the forecast page.



Forecasted Value View – Model Selection

On this page, users will be able to edit the parameters of how far ahead they'd like to forecast, as well as what model they would like to use to forecast sale prices.



Part 3 - Teamwork

Work Planning and Distribution

Our team has reflected on the remaining work to be completed and delegated primary responsibility for each item as listed below. However, our team intends to continue working collectively on each project deliverable.

LE Lee:

- Web Scraper Administration and Final Input Data Set Generation
- Final Data Set Cleaning
- Historical Data Visualization Workflow Finalization

Andrew Mouat:

- Web Application Build
- Data/File Storage and Access
- Final Technical Documentation

Nicholas Perry:

- Time Series Modeling Workflow Finalization
- Time Series Forecasting Workflow Finalization
- Final Application Deployment