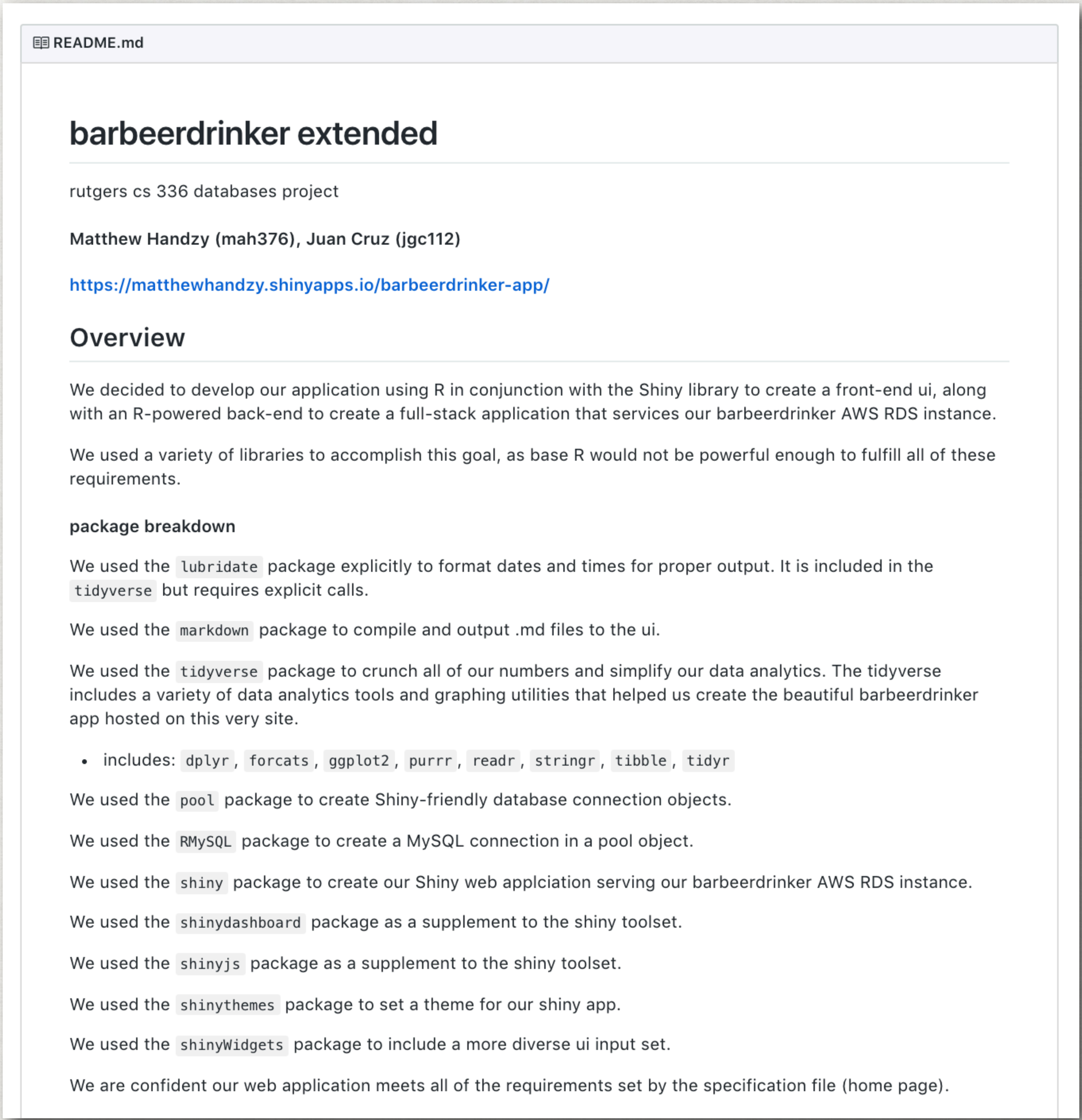
MATTHEW HANDZY & JUAN CRUZ PRESENT:

BARBEERDRINKER EXTENDED

QUICK NOTE

- this presentation is derived from the README.md / README.pdf files you will find in the root directory of our project. It is also located in /barbeerdrinker-app/www/README.* in the .md, .pdf, and .html formats (html format has <head> tag stripped for proper shiny app compatibility, may look strange in browser)
- please view this presentation by reading the README file
- this presentation is made up of screenshots from the README (no, that is not a joke)
- please, please, please read the readme file instead of a presentation about our already presentable front-end ui application that we spent so much time on perfecting
- please read the readme.



README.md

barbeerdrinker extended

rutgers cs 336 databases project

Matthew Handzy (mah376), Juan Cruz (jgc112)

https://matthewhandzy.shinyapps.io/barbeerdrinker-app/

Overview

We decided to develop our application using R in conjunction with the Shiny library to create a front-end ui, along with an R-powered back-end to create a full-stack application that services our barbeerdrinker AWS RDS instance.

We used a variety of libraries to accomplish this goal, as base R would not be powerful enough to fulfill all of these requirements.

package breakdown

We used the `lubridate` package explicitly to format dates and times for proper output. It is included in the `tidyverse` but requires explicit calls.

We used the `markdown` package to compile and output .md files to the ui.

We used the `tidyverse` package to crunch all of our numbers and simplify our data analytics. The tidyverse includes a variety of data analytics tools and graphing utilities that helped us create the beautiful barbeerdrinker app hosted on this very site.

- includes: `dplyr`, `forcats`, `ggplot2`, `purrr`, `readr`, `stringr`, `tibble`, `tidyr`

We used the `pool` package to create Shiny-friendly database connection objects.

We used the `RMySQL` package to create a MySQL connection in a pool object.

We used the `shiny` package to create our Shiny web applciation serving our barbeerdrinker AWS RDS instance.

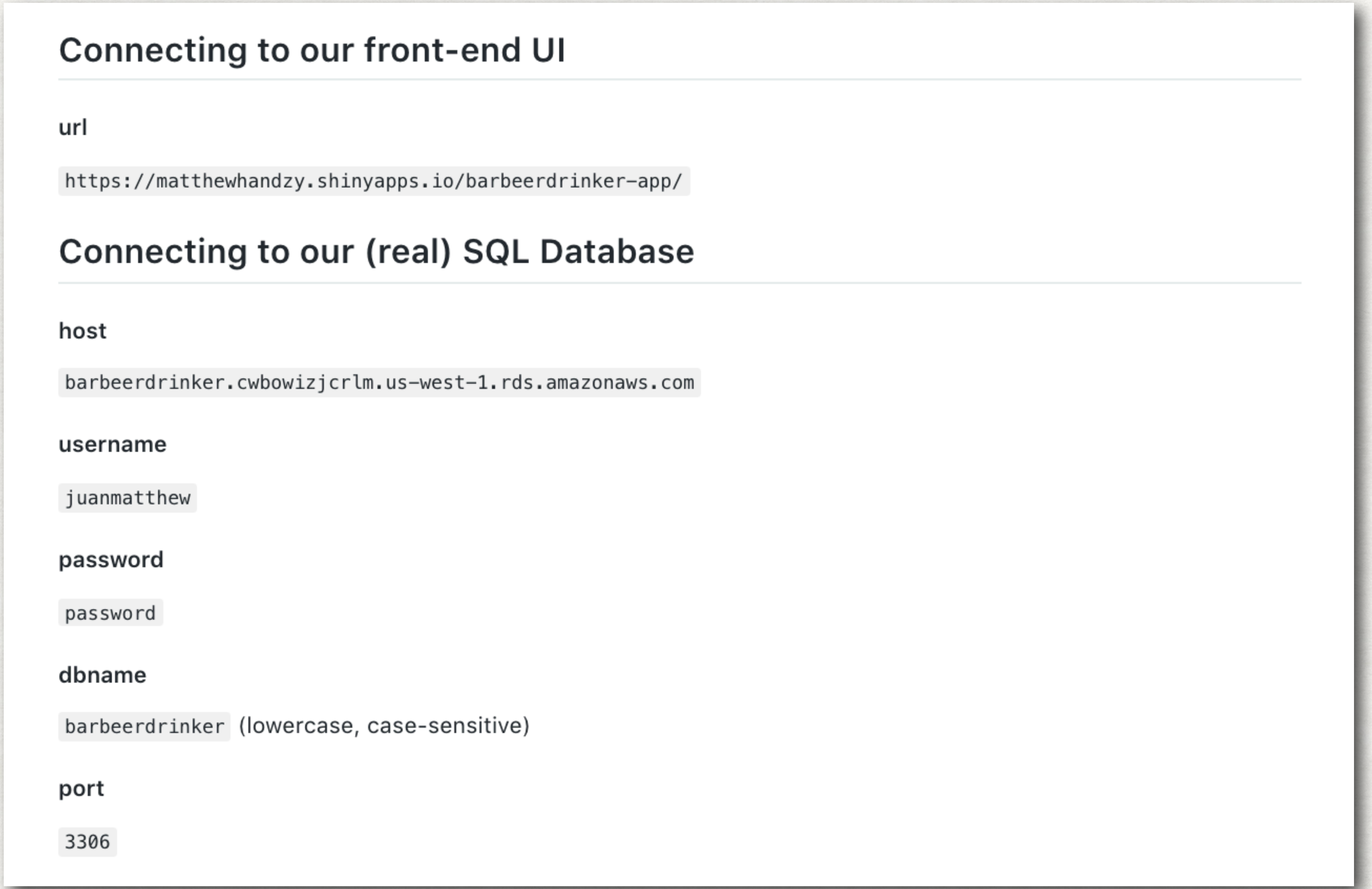
We used the `shinydashboard` package as a supplement to the shiny toolset.

We used the `shinyjs` package as a supplement to the shiny toolset.

We used the `shinythemes` package to set a theme for our shiny app.

We used the `shinyWidgets` package to include a more diverse ui input set.

We are confident our web application meets all of the requirements set by the specification file (home page).



Connecting to our front-end UI

url

`https://matthewhandzy.shinyapps.io/barbeerdrinker-app/`

Connecting to our (real) SQL Database

host

`barbeerdrinker.cwbowizjcrlm.us-west-1.rds.amazonaws.com`

username

`juanmatthew`

password

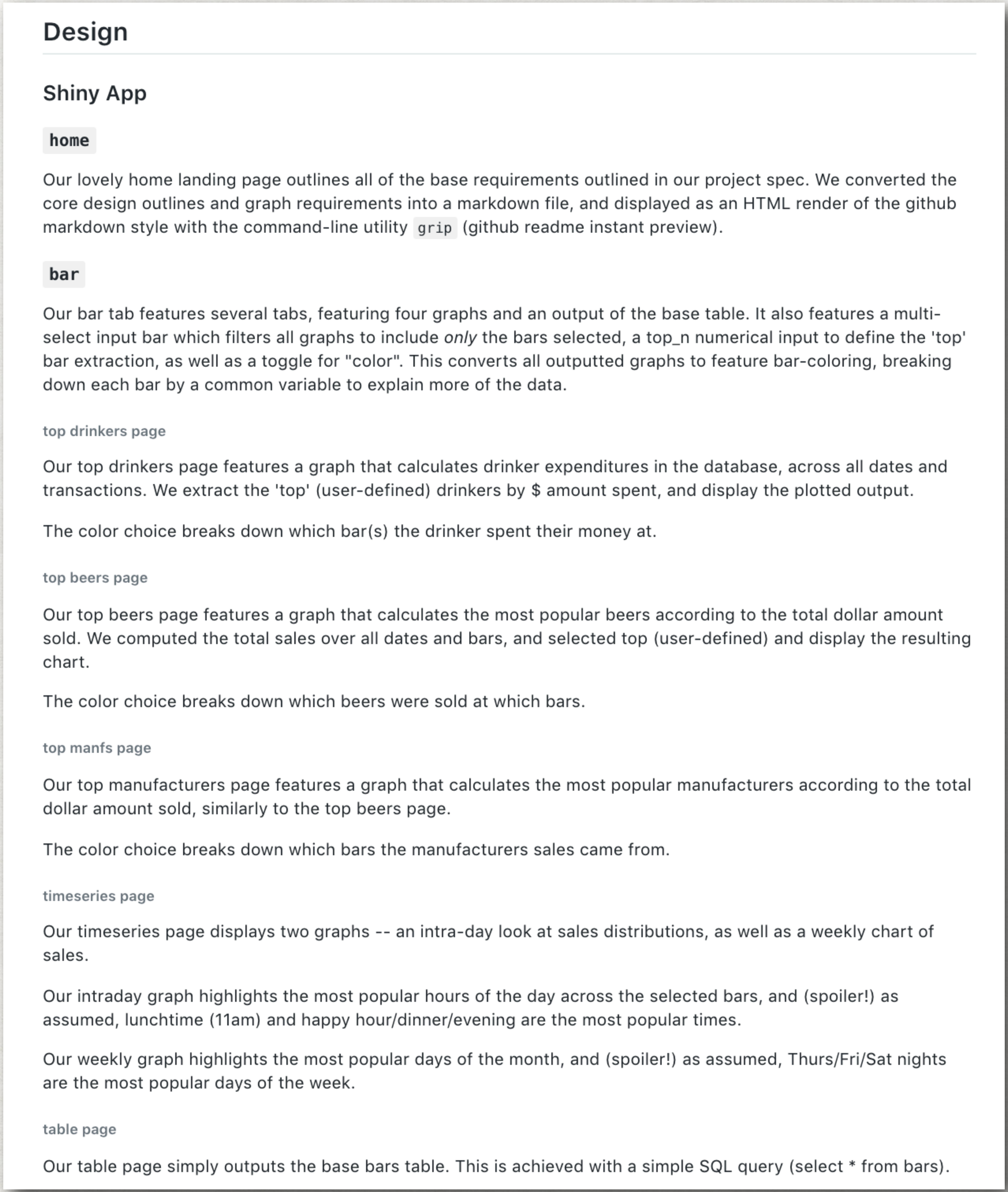
`password`

dbname

`barbeerdrinker` (lowercase, case-sensitive)

port

`3306`



Design

Shiny App

`home`

Our lovely home landing page outlines all of the base requirements outlined in our project spec. We converted the core design outlines and graph requirements into a markdown file, and displayed as an HTML render of the github markdown style with the command-line utility `grip` (github readme instant preview).

`bar`

Our bar tab features several tabs, featuring four graphs and an output of the base table. It also features a multi-select input bar which filters all graphs to include *only* the bars selected, a top_n numerical input to define the 'top' bar extraction, as well as a toggle for "color". This converts all outputted graphs to feature bar-coloring, breaking down each bar by a common variable to explain more of the data.

top drinkers page

Our top drinkers page features a graph that calculates drinker expenditures in the database, across all dates and transactions. We extract the 'top' (user-defined) drinkers by $ amount spent, and display the plotted output.

The color choice breaks down which bar(s) the drinker spent their money at.

top beers page

Our top beers page features a graph that calculates the most popular beers according to the total dollar amount sold. We computed the total sales over all dates and bars, and selected top (user-defined) and display the resulting chart.

The color choice breaks down which beers were sold at which bars.

top manfs page

Our top manufacturers page features a graph that calculates the most popular manufacturers according to the total dollar amount sold, similarly to the top beers page.

The color choice breaks down which bars the manufacturers sales came from.

timeseries page

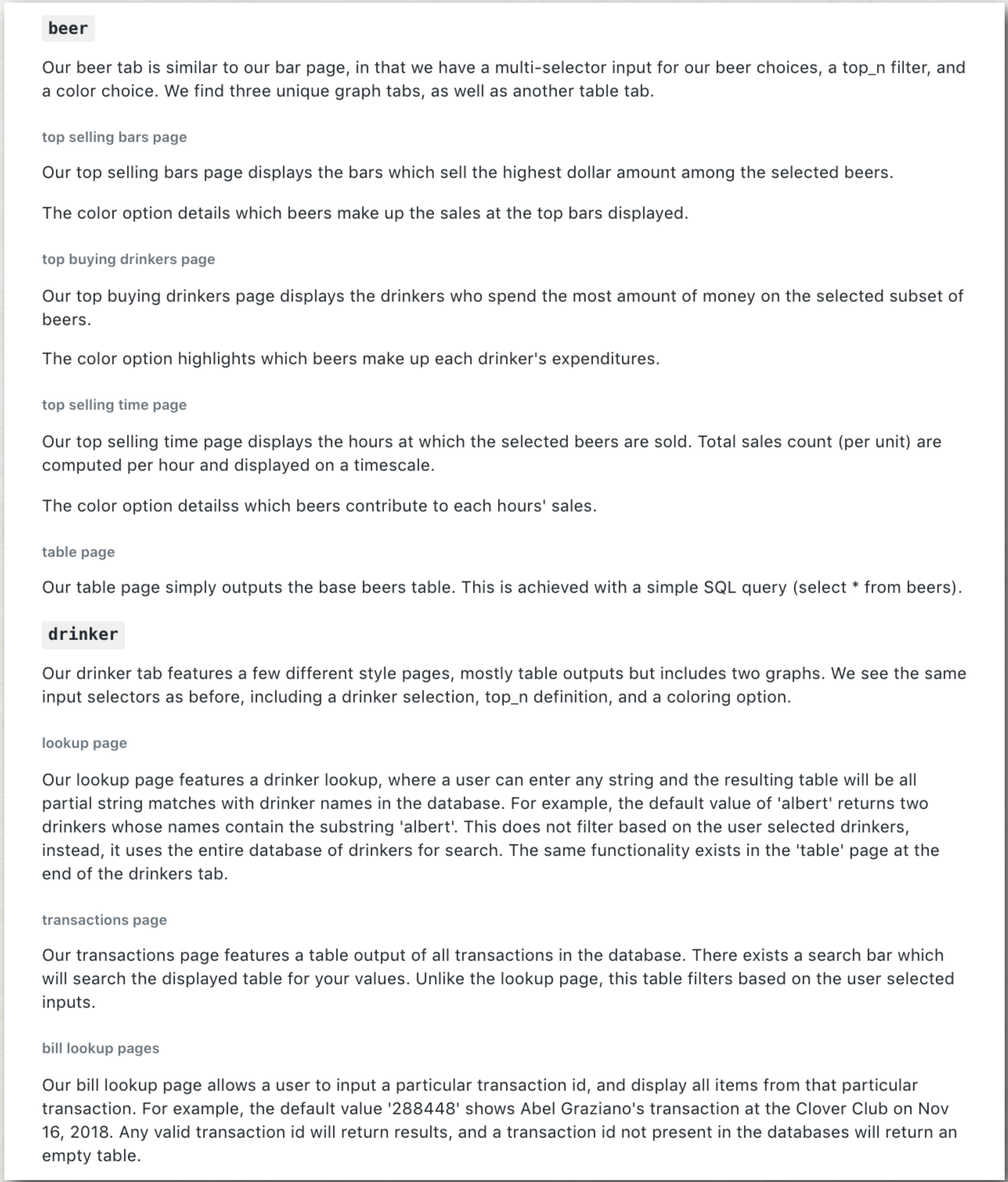
Our timeseries page displays two graphs -- an intra-day look at sales distributions, as well as a weekly chart of sales.

Our intraday graph highlights the most popular hours of the day across the selected bars, and (spoiler!) as assumed, lunchtime (11am) and happy hour/dinner/evening are the most popular times.

Our weekly graph highlights the most popular days of the month, and (spoiler!) as assumed, Thurs/Fri/Sat nights are the most popular days of the week.

table page

Our table page simply outputs the base bars table. This is achieved with a simple SQL query (select * from bars).



`beer`

Our beer tab is similar to our bar page, in that we have a multi-selector input for our beer choices, a top_n filter, and a color choice. We find three unique graph tabs, as well as another table tab.

top selling bars page

Our top selling bars page displays the bars which sell the highest dollar amount among the selected beers.

The color option details which beers make up the sales at the top bars displayed.

top buying drinkers page

Our top buying drinkers page displays the drinkers who spend the most amount of money on the selected subset of beers.

The color option highlights which beers make up each drinker's expenditures.

top selling time page

Our top selling time page displays the hours at which the selected beers are sold. Total sales count (per unit) are computed per hour and displayed on a timescale.

The color option detailss which beers contribute to each hours' sales.

table page

Our table page simply outputs the base beers table. This is achieved with a simple SQL query (select * from beers).

`drinker`

Our drinker tab features a few different style pages, mostly table outputs but includes two graphs. We see the same input selectors as before, including a drinker selection, top_n definition, and a coloring option.

lookup page

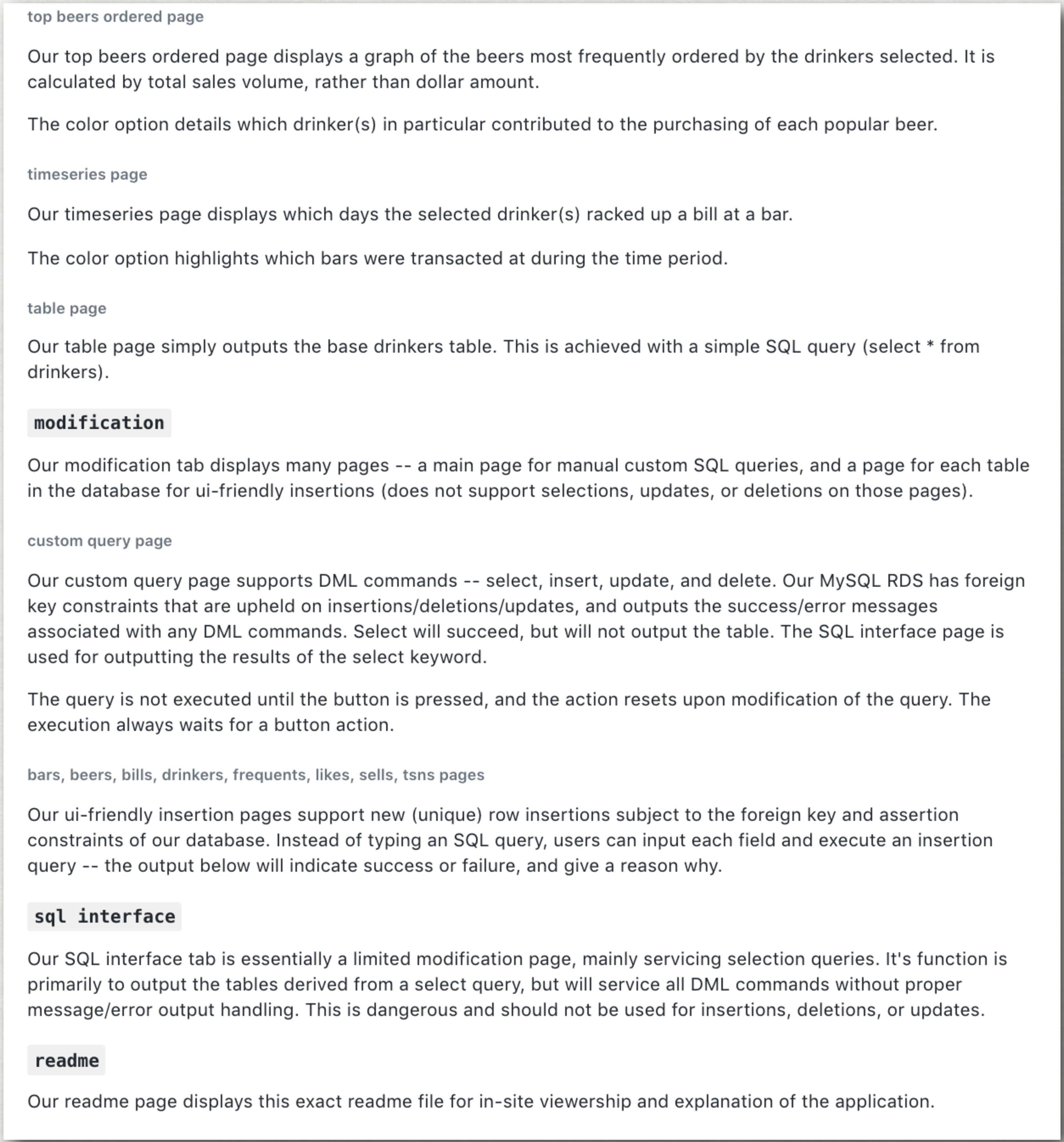
Our lookup page features a drinker lookup, where a user can enter any string and the resulting table will be all partial string matches with drinker names in the database. For example, the default value of 'albert' returns two drinkers whose names contain the substring 'albert'. This does not filter based on the user selected drinkers, instead, it uses the entire database of drinkers for search. The same functionality exists in the 'table' page at the end of the drinkers tab.

transactions page

Our transactions page features a table output of all transactions in the database. There exists a search bar which will search the displayed table for your values. Unlike the lookup page, this table filters based on the user selected inputs.

bill lookup pages

Our bill lookup page allows a user to input a particular transaction id, and display all items from that particular transaction. For example, the default value '288448' shows Abel Graziano's transaction at the Clover Club on Nov 16, 2018. Any valid transaction id will return results, and a transaction id not present in the databases will return an empty table.



top beers ordered page

Our top beers ordered page displays a graph of the beers most frequently ordered by the drinkers selected. It is calculated by total sales volume, rather than dollar amount.

The color option details which drinker(s) in particular contributed to the purchasing of each popular beer.

timeseries page

Our timeseries page displays which days the selected drinker(s) racked up a bill at a bar.

The color option highlights which bars were transacted at during the time period.

table page

Our table page simply outputs the base drinkers table. This is achieved with a simple SQL query (select * from drinkers).

`modification`

Our modification tab displays many pages -- a main page for manual custom SQL queries, and a page for each table in the database for ui-friendly insertions (does not support selections, updates, or deletions on those pages).

custom query page

Our custom query page supports DML commands -- select, insert, update, and delete. Our MySQL RDS has foreign key constraints that are upheld on insertions/deletions/updates, and outputs the success/error messages associated with any DML commands. Select will succeed, but will not output the table. The SQL interface page is used for outputting the results of the select keyword.

The query is not executed until the button is pressed, and the action resets upon modification of the query. The execution always waits for a button action.

bars, beers, bills, drinkers, frequents, likes, sells, tsns pages

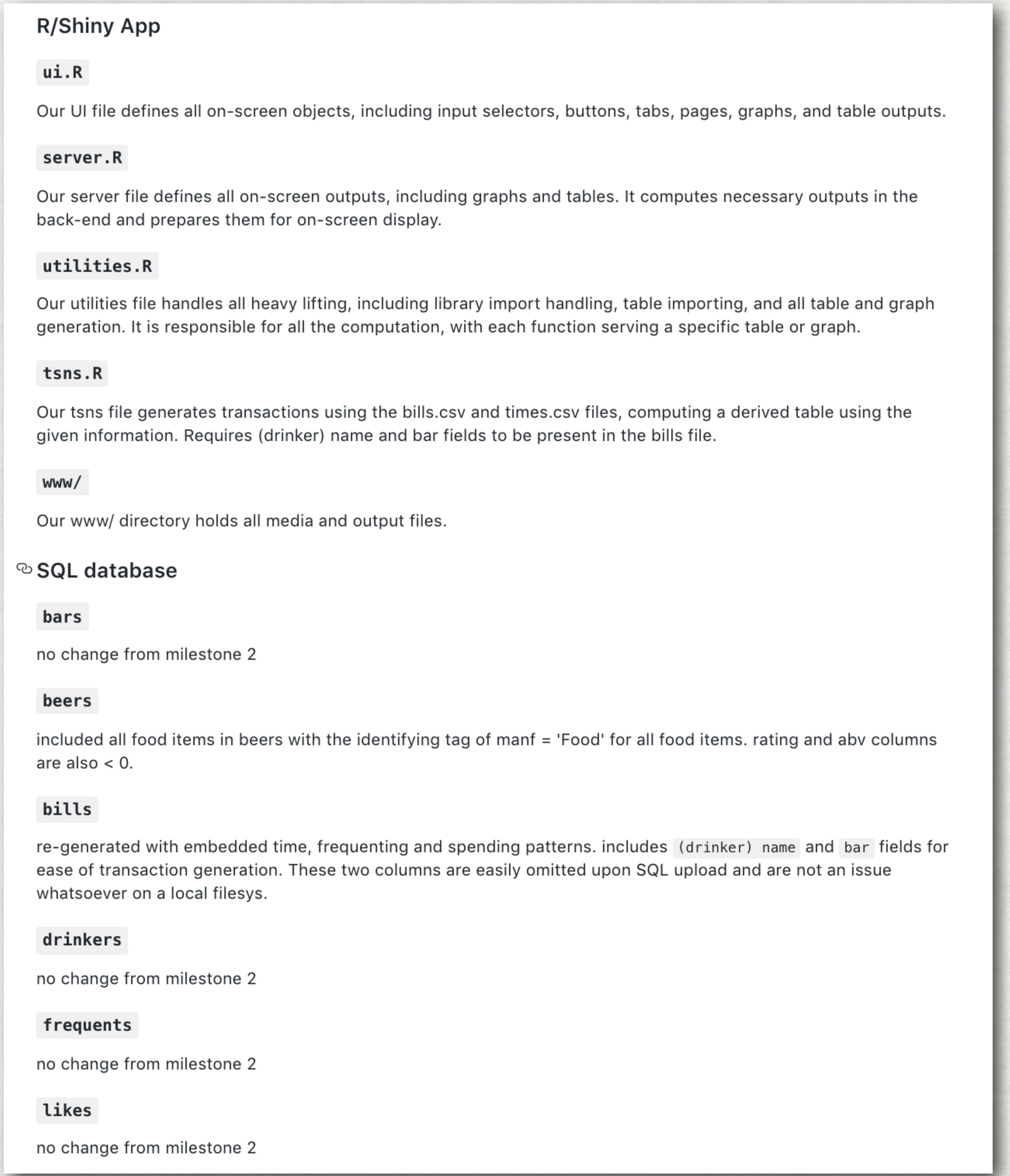
Our ui-friendly insertion pages support new (unique) row insertions subject to the foreign key and assertion constraints of our database. Instead of typing an SQL query, users can input each field and execute an insertion query -- the output below will indicate success or failure, and give a reason why.

`sql interface`

Our SQL interface tab is essentially a limited modification page, mainly servicing selection queries. It's function is primarily to output the tables derived from a select query, but will service all DML commands without proper message/error output handling. This is dangerous and should not be used for insertions, deletions, or updates.

`readme`

Our readme page displays this exact readme file for in-site viewership and explanation of the application.



R/Shiny App

`ui.R`

Our UI file defines all on-screen objects, including input selectors, buttons, tabs, pages, graphs, and table outputs.

`server.R`

Our server file defines all on-screen outputs, including graphs and tables. It computes necessary outputs in the back-end and prepares them for on-screen display.

`utilities.R`

Our utilities file handles all heavy lifting, including library import handling, table importing, and all table and graph generation. It is responsible for all the computation, with each function serving a specific table or graph.

`tsns.R`

Our tsns file generates transactions using the bills.csv and times.csv files, computing a derived table using the given information. Requires (drinker) name and bar fields to be present in the bills file.

`www/`

Our www/ directory holds all media and output files.

SQL database

`bars`

no change from milestone 2

`beers`

included all food items in beers with the identifying tag of manf = 'Food' for all food items. rating and abv columns are also < 0.

`bills`

re-generated with embedded time, frequenting and spending patterns. includes `(drinker) name` and `bar` fields for ease of transaction generation. These two columns are easily omitted upon SQL upload and are not an issue whatsoever on a local filesys.

`drinkers`

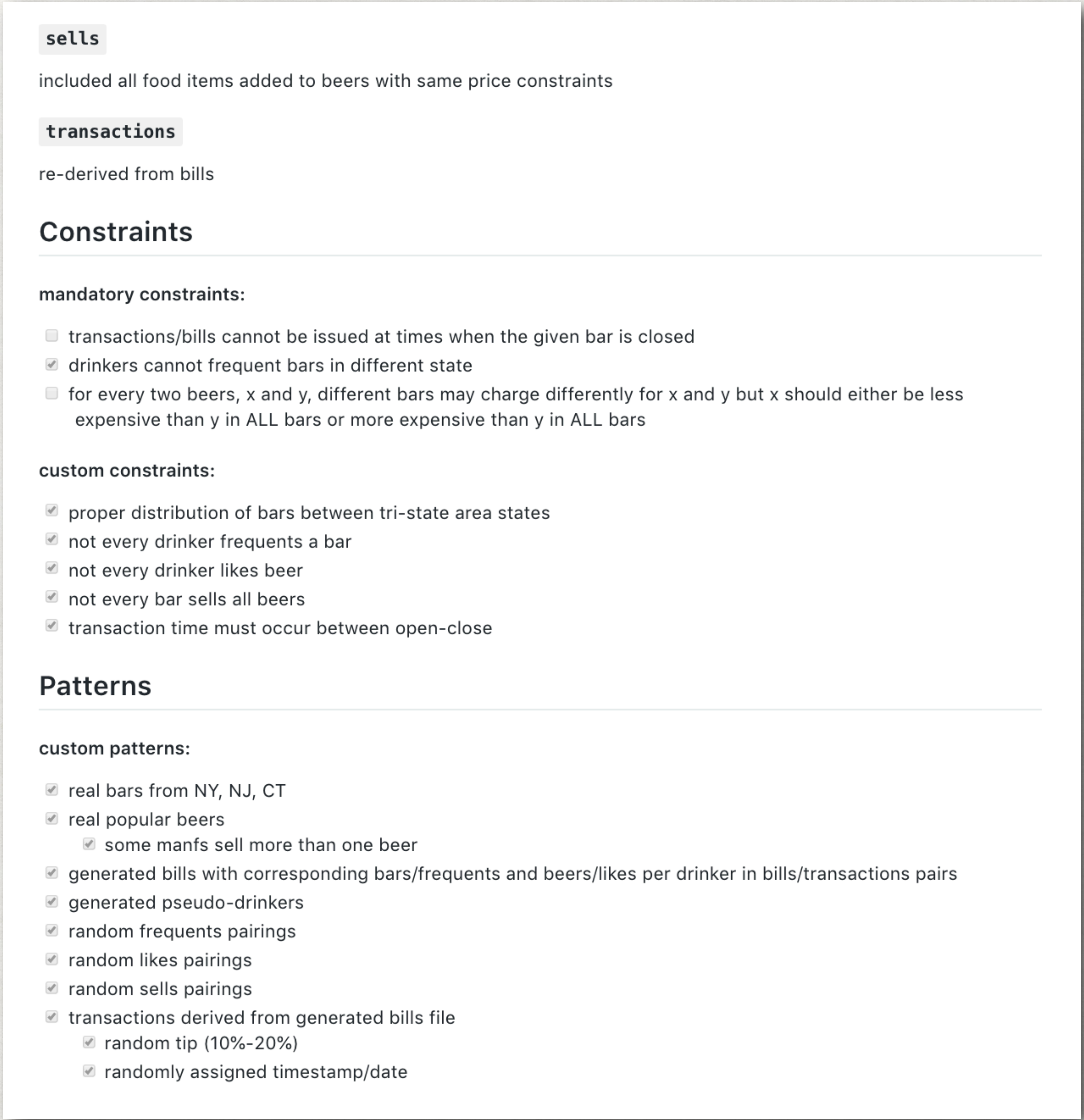
no change from milestone 2

`frequents`

no change from milestone 2

`likes`

no change from milestone 2



`sells`

included all food items added to beers with same price constraints

`transactions`

re-derived from bills

Constraints

mandatory constraints:

- [ ] transactions/bills cannot be issued at times when the given bar is closed
- [x] drinkers cannot frequent bars in different state
- [ ] for every two beers, x and y, different bars may charge differently for x and y but x should either be less expensive than y in ALL bars or more expensive than y in ALL bars

custom constraints:

- [x] proper distribution of bars between tri-state area states
- [x] not every drinker frequents a bar
- [x] not every drinker likes beer
- [x] not every bar sells all beers
- [x] transaction time must occur between open-close

Patterns

custom patterns:

- [x] real bars from NY, NJ, CT
- [x] real popular beers
  - [x] some manfs sell more than one beer
- [x] generated bills with corresponding bars/frequents and beers/likes per drinker in bills/transactions pairs
- [x] generated pseudo-drinkers
- [x] random frequents pairings
- [x] random likes pairings
- [x] random sells pairings
- [x] transactions derived from generated bills file
  - [x] random tip (10%-20%)
  - [x] randomly assigned timestamp/date