Thomas Redding
Ben Wedin

Phase 2.4: Development Plan

Deliverables

CLIENT SIDE - UI

index.html – the main display of our webapp. This will display the options and input boxes available to the user so that they can request for their data visualization.
style.css – gives the style for index.html (makes it look pretty)

FRONT-END

Drawer.js (view) - this reads the data stored inside index.html and draws to <canvas>

SERVER SIDE - BACK-END

webapp.py - this script takes requests from the client, parses it, calls the appropriate function in DataSource.py, and then sends the response data back to the client
DataSource.py - this class acts as an interface between the server and the database

Note: we will implement everything from the back-end on forward. Thus, we will implement the database first, followed by Datasource.py, followed by webapp.py, etc. This will allow for easier testing.

Drawer.js

parseRequestAndData()

- Parses data and request summary inside index.html to decide which function to call in Drawer.js.
- def parseRequestMakesRightRequest() - this method will assert whether or not the correct calls to drawer.js are made given different requests.

drawBarGraphYearRange(data, statistic,state,startYear,endYear,tick,color1, color2)

- Draws a bar chart with the given data. Figure is labeled with the statistic, year, and state/region, and the two color parameters are used to display the statistic, where color2 represents a high value, and color 1 is a low value. startYear and endYear specify the range of data to be visualized, and tick gives an indication for how frequently data points should be drawn. If only one color is given, the two parameters are identical. Graphic is drawn into index.html.
- Our test for this will involve comparing our visual to an already existing bar chart for a given statistic, which we can draw in Excel.

drawBarGraphNationwide(data, statistic,state,year,sort,color1, color2)

- Draws a bar graph with the given data. Figure is labeled with the year and statistic, and the two color parameters are used to display the statistic, where color2 represents a high value, and color 1 is a low value. If only one color is given, the two parameters are identical. Sort parameter indicates how the graphic will be drawn. Graphic is drawn into index.html.
- Our test for this will involve comparing our visual to an already existing bar chart for a given statistic, which we can draw in Excel.

drawLineChartYearRange(data, statistic,state,startYear,endYear,tick,color1, color2)
- Draws a line chart with the given data. Figure is labeled with the statistic, year, and state/region, and the two color parameters are used to display the statistic, where color2 represents a high value, and color 1 is a low value. startYear and endYear specify the range of data to be visualized, and tick gives an indication for how frequently data points should be drawn. If only one color is given, the two parameters are identical. Graphic is drawn into index.html.
- Our test for this will involve comparing our visual to an already existing bar chart for a given statistic, which we can draw in Excel.

drawColorMap(data,statistic,year,color1,color2)
- Draws a national map with the given data. Figure is labeled with the year and statistic, and the two color parameters are used to display the statistic, where color2 represents a high value, and color 1 is a low value. If only one color is given, the two parameters are identical. Graphic is drawn into index.html.
- Our test for this will involve comparing our visual to an already existing color map for a given statistic.

<u>Webapp.py</u>

main()
- This is the first function called on the server. It calls getCgiParameters() to parse the parameters and then calls parseRequest() with this parameter dictionary

getCgiParameters()
- This function parses the string passed from the server into a dictionary
- This was given to us by Jadrian. We will not test it.

sanitizeUserInput(s)
- This function ensures that the client didn’t pass us any colons or commas/ This is redundant (in a good way), because the client (1) cannot edit the database and (2) is limited by index.html to non-bad parameter values
- This was given to us by Jadrian. We will not test it.

parseRequest(parameters)

- Decides which of the request___() functions to call based on the client-provided parameters' values.
- def parseRequestCallsCorrectRequest() - we will test parseRequest to see that each of the request functions in webapp.py are appropriately called when parseRequest is given a request given by different parameters input.

requestListOfStates(statistic, year)

- Calls getListOfStates(statistic, year) from Datasource.py to receive the list of all states from the database for which a statistic is available (optionally) in a given year
- The function takes no inputs, so to test it we will simply call it and see if it returns the list of 51 states (50 + D.C.)

requestListOfStatisticNames()

- Calls getListOfStatisticNames() from Datasource.py to receive a list of all statistics names from the database.
- This function takes no inputs, so to test it we will simply call it and see if it returns the list of statistic names in the database.

requestDataForYearRange(statistic, start_year, end_year, state)

- Calls getDataForYearRange() from Datasource.py to get data from some statistic between two years for a specific state. These last three parameters are optional.
- We will not unit test this function, because it is trivial.

requestDataForSpecificYear(statistic, single_year, state)

- Calls getDataForSpecificYear() from Datasource.py to receive the data for some statistic in some year for some state (the last parameter is optional).
- We will not unit test this function, because it is trivial.

requestMaxYearRange(statistic, state)

- Calls getMaxYearRange() from Datasource.py to receive the first and last year for which a statistic is available for some state (this last parameter is optional).
- We will not unit test this function, because it is trivial.

<u>Datasource.py</u>

class DataSource:

getDataForYearRange(self, statistic, start_year, end_year, state)

- Grabs data for some statistic between two years for some state (the last three parameters are optional).

getDataForSpecificYear(self, statistic, single_year, state)

- Grabs data for some statistic at some year for(optionally) some state.

getMaxYearRange(self, statistic, state)

- Grabs the first and last year for which (optionally) some statistic is available in (optionally) some state.
- def parseRequestGetsCorrectYearRange() - this test will check to see if the correct year ranges are given for a given statistic and state

getListOfStatisticNames(self)

- Grabs a list of all the statistic-names in the database.
- This function takes no inputs, so to test it we will simply call it and see if it returns the list of statistic-names in the database.

getListOfStates(self, statistic, year)

- Grabs a list of all the state-names in the database.
- def parseRequestMakesRightRequest() - this test will check to see if the correct list of states are given for a given statistic and year (statistics that have all 50+1 states, statistics that have less, statistic that vary over time)

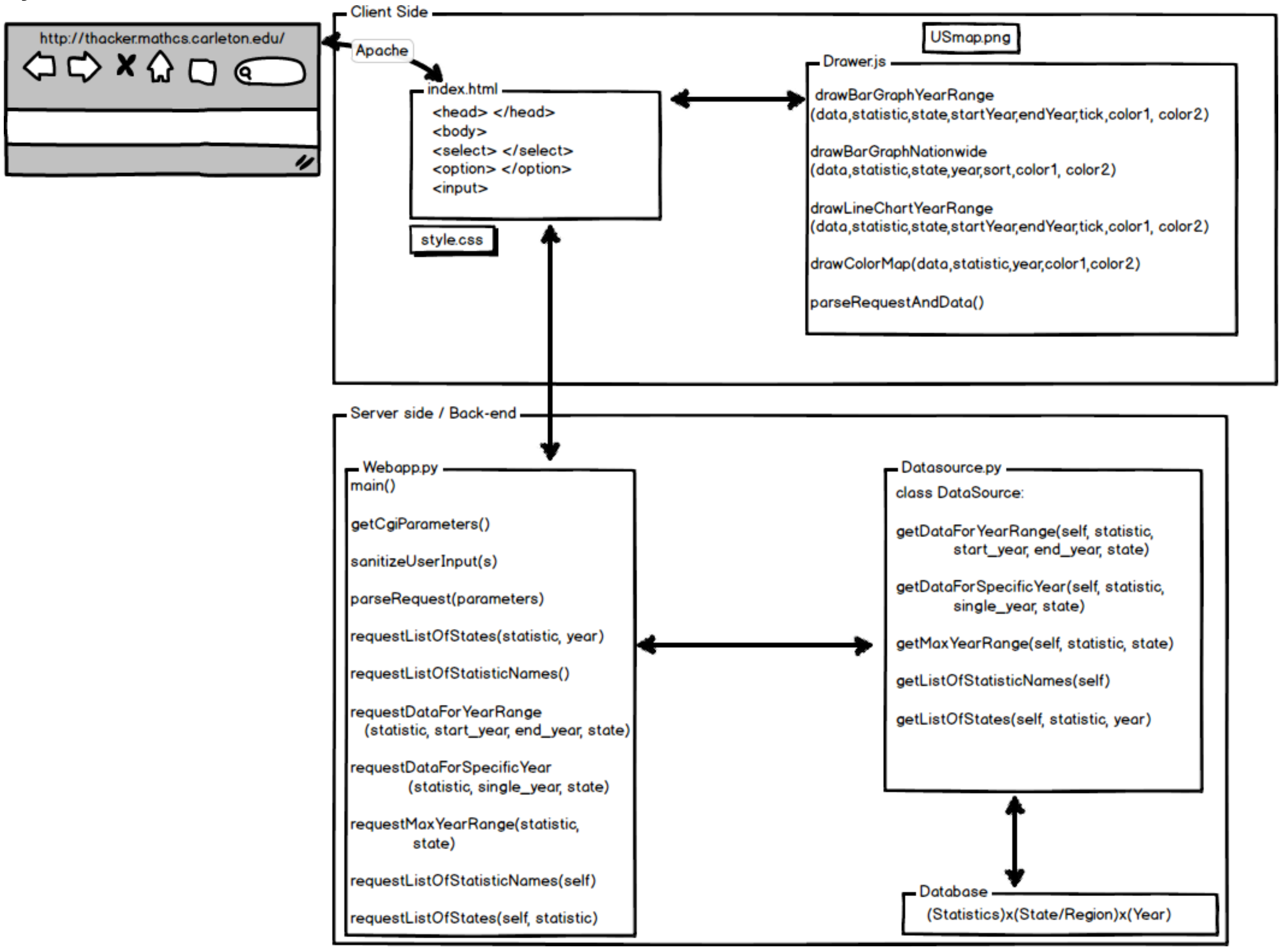
Database

(Statistics)x(State/Region)x(Year)

- The database will store the data as a 3d array (really a dictionary) with the following dimensions:
  - statistic-name
  - state/region-name
  - year
- The statistic name and state/region-name will be stored as ints within this array, and will be accessed by another lookup table that holds the name for each corresponding int
- Data from the database will then be sent back including the request information so that it can be entered back into index.html and eventually accessed by Drawer, in a format like this:

```
“requestType:getData,statistic:Population,state:ND,summary:national,startYear:1
988,endYear:1992
Year, Population
1988,512341
1989,518365
1990,523412
1991,528921
1992,533521”
```

System Architecture



http://thacker.mathcs.carleton.edu/
Client Side
Apache
index.html
<head> </head>
<body>
<select> </select>
<option> </option>
<input>
style.css
USmap.png
Drawer.js
drawBarGraphYearRange
(data,statistic,state,startYear,endYear,tick,color1, color2)
drawBarGraphNationwide
(data,statistic,state,year,sort,color1, color2)
drawLineChartYearRange
(data,statistic,state,startYear,endYear,tick,color1, color2)
drawColorMap(data,statistic,year,color1,color2)
parseRequestAndData()
Server side / Back-end
Webapp.py
main()
getCgiParameters()
sanitizeUserInput(s)
parseRequest(parameters)
requestListOfStates(statistic, year)
requestListOfStatisticNames()
requestDataForYearRange
(statistic, start_year, end_year, state)
requestDataForSpecificYear
(statistic, single_year, state)
requestMaxYearRange(statistic,
state)
requestListOfStatisticNames(self)
requestListOfStates(self, statistic)
Datasource.py
class DataSource:
getDataForYearRange(self, statistic,
start_year, end_year, state)
getDataForSpecificYear(self, statistic,
single_year, state)
getMaxYearRange(self, statistic, state)
getListOfStatisticNames(self)
getListOfStates(self, statistic, year)
Database
(Statistics)x(State/Region)x(Year)

Development Schedule

May 5
All data sources are tracked down (actual links/files).
DataSource and webapp methods are all functional for a single dummy statistic.

May 6
Unit tests written for all methods.

May 7
Phase 5 is done (database has all data).

May 9
Client software is written and has basic visualizations for dummy statistics from a dummy server.

May 10
All unit tests pass and webapp is fully integrated.

May 11
Version 1 is done.

May 15
User testing has been completed.

May 18
Final version, updated after user feedback, is turned in.