PlotMe

Interactive Data Visualizing Application

Design Specification Document

Friday, April 10, 2015

Computer Science – Comp Sci 2XB3

Group 18

Saim Malik, Ahmed Khan and Zayan Imtiaz

Team Roles

Name	**Student Number**	**Roles**
Ahmed Khan	1324501	Project Leader, Programmer, Back-end, Web Development
Zayan Imtiaz	1152665	Researcher, Programmer, Log Admin, Back-end
Saim Malik	1321796	Programmer, Tested, Front-end, UI, Web Development and Design

“By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through [CS] or [SE]-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.”

Name	**Role(s)**	**Contributions**	**Comments**
Saim	Web Design	Website's user interface	Designed the UI for all the web pages using HTML5, CSS (Bootstrap), JavaScript (JQuery)
Ahmed	Java Programmer	Implemented sorting algorithms	Created an optimized sorting algorithm that uses both merge and insertion sort based on the size of the data
Zayan	Java Programmer	CSV file parser	Wrote a parser that extracts data from a given CSV file and returns it in a format usable by the program
Ahmed	Web Developer	Graphs using d3js	Made visually appealing graphs using d3js library and JavaScript
Saim	Web Developer	Wrote the controller	Combined the back-end of the application with the front-end using the Grails Framework (and MVC design)
Zayan	Documentation	Requirements and Specification Document	Wrote the Requirements and Specification Document
Saim	Documentation	Design Specification Document	Wrote the Design Specification Document
Saim	Documentation	Presentation Slides	Made PowerPoint presentation slides
Zayan	Log Admin	Group log file	Updated the group's progress in the log file
Ahmed	Java Programmer	Implemented analyze functionality	Programmed an algorithm to analyze a given dataset and present the 'outliers' in the data in graph format
Zayan	Java Programmer	Implemented search algorithms	Designed a search algorithm that users a BST to efficiently search through the dataset for a given input
Saim	Tester	Tested the application	Ran the application with several input files and different graph settings to assure the robustness of the application

Abstract

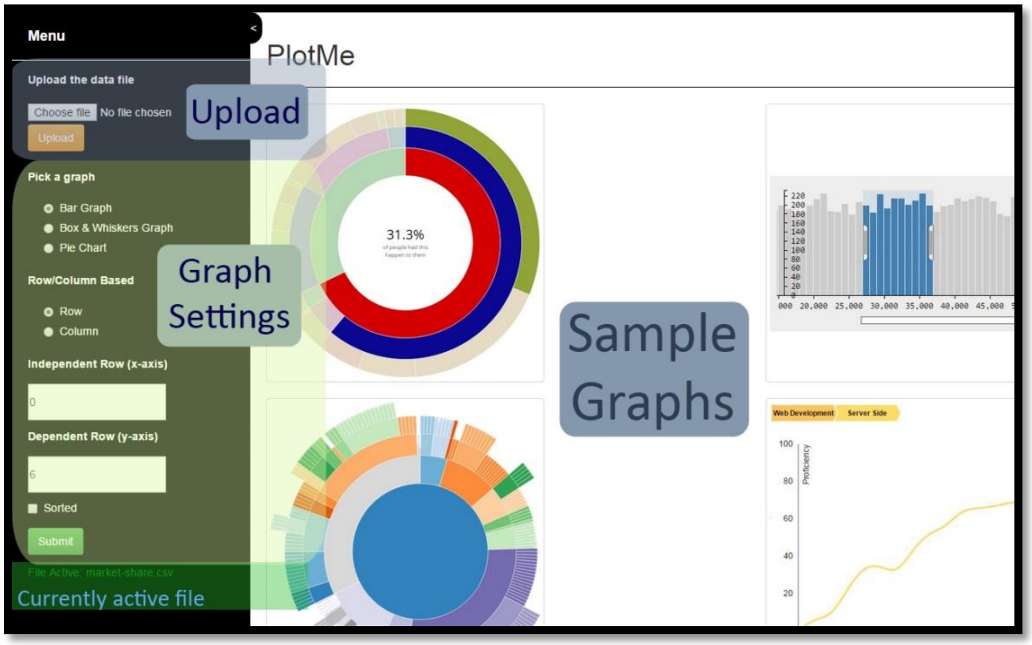
PlotMe is a web-based application that allows its users to visualize data files as interactive graphs. The application allows the user to choose from several graph settings to account for the structure the data could be in. For instance, some data sets have headings in the first row and others might have headings in the first column; the graph settings in this application account for both such conditions and several others. Aside from the functionality, PlotMe has an interface that is both visually appealing and easy to use. The graphs generated include Bar Graph, Box and Whiskers Graph and Pie Chart.

Table of Contents

User Interface Guide (How to Use)

Top

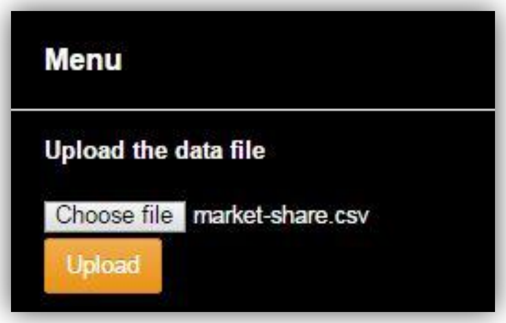
Open a web browser and go to http://plotme.cfapps.io (or http://plotme.cfapps.io/main if the first link doesn't work)

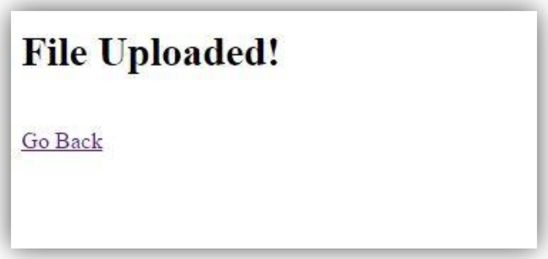


Click on the pictures of the sample graphs to view them (most of them are interactive – i.e. you can click on the graph and it will do something)

To upload your own dataset to be graphed, expand the collapsible menu (if it isn't already) and follow these steps:

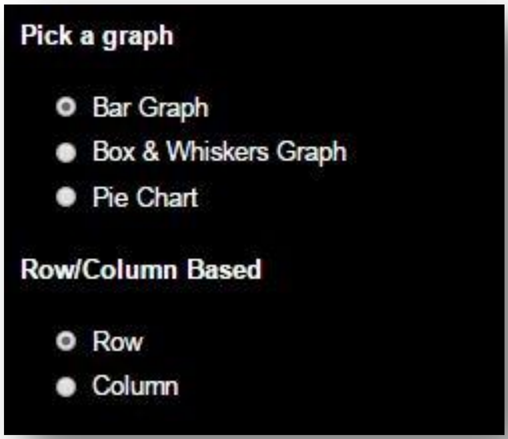
- Click Choose File
- Go to the "sample-data" folder (provided with the submission) and pick one of the CSV files
- Click Upload; it should open a new window that says "File Uploaded" with a "Go Back" link

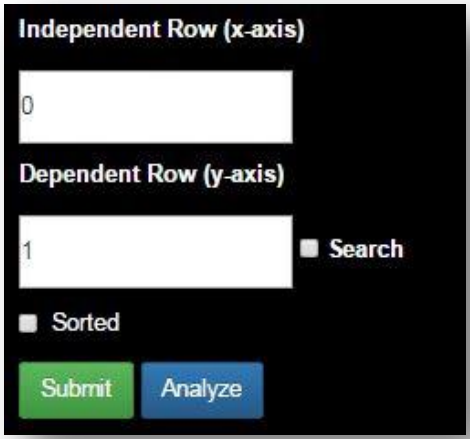



File Uploaded!

Go Back

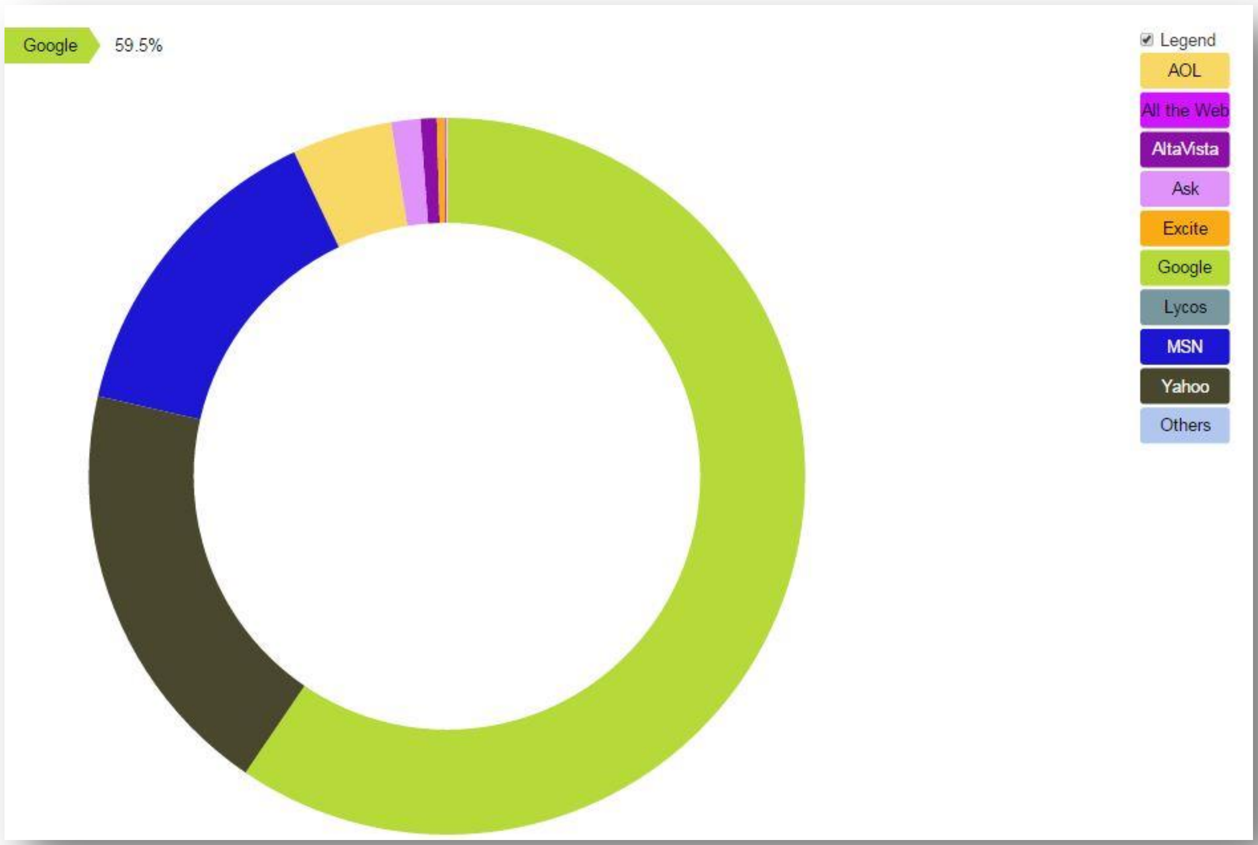
- Click "Go Back"; the sidebar menu should now have different settings visible to graph the data
- Pick one of the three available graphs (Bar Graph, Box and Whiskers or Pie Chart)
- Choose whether the dataset is "Row Based" or "Column Based"
    - Row Based datasets are those in which the headings are the **first row** and **each row after** that has values with respect to its appropriate column
    - Column Based datasets are those in which the headings are the **first column** and **each column after** that has values with respect to its appropriate row





- In the first textbox, write the **row (if Row Based)/column (if Column Based) number** for the x-axis (usually the first row/column with all the headings) – **<u>NOTE</u>**: The numbers start with 0; so for the first row/column, input 0; for the second row/column, input 1, etc.
- In the second textbox, you have two options:
    - Search is NOT checked
        - Write the **row (if Row Based)/column (if Column Based) number** with the data for the y-axis (can't be the same as the x-axis)
    - Search is checked
        - Write the **column number,<search input> (if Row Based)/row number,<search input> (if Column Based)** and the program will automatically find the row/column with the searched text (an example for the "market-share.csv" dataset (a Row Based dataset) would be, "0,1/1/2005" – where "0" is the "Date" column and the "1/1/2005" is the Date being searched for)
- Check the Sorted checkbox if you want the graphed data to be sorted
- Finally, click the Submit button (that should now be enabled)

- A new window should open with a graph based on the provided settings
- To change any settings or to upload a new dataset, “Go Back” and follow the same process as described above



Pie Chart for market-share.csv dataset

Classes/Modules Description

Top

Parser.groovy

This class is used to parse the given data sets and store the results in terms of points containing independent variables (x-axis) and dependent variables (y-axis).

Point.groovy

This class creates an object Point which contains the independent variable, x axis stored as a String and the dependent variable y axis stored as a double. Storing it in this method makes access of required pairs convenient.

Data.groovy

This Data class is used to store the ArrayList containing the pairs of points for the independent and dependent variables as well as other relevant information. This is converted later into a Json object for the d3 library.

RedBlackBST.java

This class was taken from the 2XB3/2CO3 class textbook website:
http://algs4.cs.princeton.edu/33balanced/RedBlackBST.java.html

It is used to search through a given row or column in an ArrayList containing a parsed dataset (with rows corresponding to lines) and to return the corresponding column or row values. For these purposes only the put and get methods were utilized.

OptimalSort.groovy

This class sorts the given array using merge sort and when the array gets to small it sorts it using insertion sort because insertion sort is faster for smaller inputs
Combines merge sort and insertion sort for more efficiency

Insertion.groovy

This class gets called by the OptimalSort.groovy. It sorts small arrays. In the given test cases that we did we found at 15 the insertion sort was the fastest.

UploadedFile.groovy

Used as a table in the database to store the uploaded file name and path. The class itself has no methods or constructor, just instance variables.

pieChart.js

This class draws pie chart on the screen. It takes a json object given by the server and displays the chart as a pie chart in the current html page.

Public Entities

Top

Parser.groovy

public static ArrayList<String[]**> getArray(**String S**)**

Input: Name of dataset csv file to be parsed taken as a string S.
Output: Returns an ArrayList containing rows of all the lines in the dataset as a list of strings.

public static Data getPairCol(ArrayList<String[]**>** s, int x, int y**)**

Input:
An ArrayList containing rows of all the lines in the dataset as a list of strings s.
The independent column (x) and dependent column (y)

Output:
Returns a Data object containing an ArrayList which includes the list of points with each point containing an independent variable (String) and dependent variable (double).

public static Data getPairRow(ArrayList<String[]> s, int x, int y**)**

Input:
An ArrayList containing rows of all the lines in the dataset as a list of strings s.
The independent row (x) and dependent row (y)

Output:
Returns a Data object containing an ArrayList which includes the list of points with each point containing an independent variable (String) and dependent variable (double).

public static Data getPair(ArrayList<String[]> s, int x, int y, String u**)**

Input:
An ArrayList containing rows of all the lines in the dataset as a list of strings s. The independent row (x) and dependent row (y). A String u which specifies is the user wants to create a pair using rows (“row”) or columns (“col”).

Output:
Returns a Data object containing an ArrayList which includes the list of points with each point

containing an independent variable (String) and dependent variable (double).

*This method works by calling either getPairCol or getPairRow depending on what is specified in String u.

public static Data getSetofPairs (ArrayList<String[]> s, int x, int y, String u**)**

Input:
An ArrayList containing rows of all the lines in the dataset as a list of strings s. The independent row (x) and dependent row (y). A String u which specifies is the user wants to create a pair using rows (“row”) or columns (“col”).

Output:
Returns a Data object containing an ArrayList which includes the list of points that include <u>all</u> possible dependent variables for a given independent variable with each point containing an independent variable (String) and dependent variable (double).

public static RedBlackBST <String, Integer**> getBSTRow (**ArrayList<String[]> a, int x**)**

Input:
An ArrayList containing rows of all the lines in the dataset as a list of strings a.
The row to be stored as an int x.

Output:
A RedBackBST that contains the row elements as strings and their corresponding column numbers.

*This is utilized in the UI for the user's convenience so they can can search for a column number by just knowing the String they desire as opposed to the column number where it's located.

public static RedBlackBST <String, Integer> getBSTCol (ArrayList<String[]> a, int x)

Input:
An ArrayList containing rows of all the lines in the dataset as a list of strings a.
The column to be stored as an int x.

Output:
A RedBackBST that contains the column elements as strings and their corresponding row numbers.

*This is utilized in the UI for the user's convenience so they can can search for a row number by just knowing the String they desire as opposed to the row number where it's located.

public static RedBlackBST <String, Integer**> getBST (**ArrayList<String[]> a, int x, String okay**)**

Input:
An ArrayList containing rows of all the lines in the dataset as a list of strings a.
The column or row to be stored as an int x.
A String Okay containing “row” or “col” to determine which is being stored.

Output:
A RedBackBST that contains the column/row elements as strings and their corresponding row/column numbers.

*this function calls either getBSTCol or getBSTRow to create the BST depending on what the user specifies for string Okay.

Point.groovy

public Point (String n, double v**)**

Constructor that initializes name and value

public void setName (String a**)**

Input: String containing name
Output: sets Name

public String getName ()

Output: returns name

public void setValue (Double a**)**

Input: Double containing value
Output: sets Value

public double getValue ()
Output: returns Value

@Override **public String toString()**{
Output: Prints out current objects of class, used for testing purposes

Data.groovy

public Data (String N, String x, String y, ArrayList<Point> d**)**

Constructor used to initialize the Data object.

public String getName()

Output: return Name;

public String getXaxis ()
Output: return x-axis;

public String getYaxis ()
Output: return y-axis;

public ArrayList<Point>**> getData ()**
Output: return data;

RedBlackBST.java

public void put(Key key, Value val**)**

Input: Key (String name) and value (corresponding row or column number) to be stored.

public Value get(Key key**)**

Input: The string name (key) one is searching for.

Output: Return the corresponding row or column number.

OptimalSort.groovy

enum type

This is used to define what type the user wants the data sorted in. The possible values are INCREASING AND DECREASING.

public static void sort(T[] comparable**)**

Input:

- an array that has a type, which implements the comparable interface.

- Sorts the array in increasing order.

Output:

- Returns nothing

public static void sort(T[] comparable, type sortType**)**

Input:

- comparable = an array of a type that extends the comparable interface,
- sortType = to tell the function how to sort the array. type.INCREASING will sort it in an increasing manor. Type.DECREASING will sort in a decreasing manor.

Output:

- returns nothing

public static void sort (T[] comparables, int min, int max**)**

Input:

- comparables = is an array of a type that extends the comparable interface,
- min = is the minimum index in the range of things that need to be sorted
- max = is the maximum index in the range of things that need to be sorted

Output:

- returns nothing

public static void sort (T[] comparables, int min, int max, type sortType**)**

Input

- comparables = is an array of a type that extends the comparable interface,
- min = is the minimum index in the range of things that need to be sorted
- max = is the maximum index in the range of things that need to be sorted
- sortType = to tell the function how to sort the array. type.INCREASING will sort it in an increasing manor. Type.DECREASING will sort in a Output:
- returns nothing

public static Boolean sorted(T[] comparables**)**

Input:

- comparables = is an array of a type that extends the comparable interface,

Output:

- returns a boolean value that represents whether the array is sorted or not.

public static Boolean sorted(T[] comparables, boolean sortType**)**

Input:

- comparables = is an array of a type that extends the comparable interface
- sortType = to tell the function how to sort the array. type.INCREASING will sort it in an increasing manor. Type.DECREASING will sort in a

Output:

- returns a boolean value that represents whether the array is sorted or not.

boolean sorted(T[] commparables, int min, int max, type OptimalSort.type sortType**)**

Input:

- comparables = is an array of a type that extends the comparable interface,
- min = is the minimum index in the range of things that need to be sorted
- max = is the maximum index in the range of things that need to be sorted
- sortType = to tell the function how to sort the array. type.INCREASING will sort it in an increasing manor. Type.DECREASING will sort in a

Output:

- returns a boolean value that represents weather the array is sorted or not.

void printArray(Object[] words**)**

This function prints the array in the console. It is used for debugging purposes.

Input:

- words = an array

Output:

- returns nothing

Insertion.groovy

void show(Object[] words**)**

This function prints the array in the console. It is used for debugging purposes.

Input:

- words = an array

Output:

- returns nothing,

public static void sort(T[] comparable**)**

Input:

- an array that has a type, which implements the comparable interface.
- Sorts the array in increasing order.

Output:

- Returns nothing

public static void sort(T[] comparable, type sortType**)**

input:

- comparable = an array of a type that extends the comparable interface,
- sortType = to tell the function how to sort the array. type.INCREASING will sort it in an increasing manor. Type.DECREASING will sort in a decreasing manor.

Output:

- returns nothing

public static void sort (T[] comparables, int min, int max**)**

Input:

- comparables = is an array of a type that extends the comparable interface,
- min = is the minimum index in the range of things that need to be sorted
- max = is the maximum index in the range of things that need to be sorted

Output:

- returns nothing

public static void sort (T[] comparables, int min, int max, type sortType**)**

Input:

- comparables = is an array of a type that extends the comparable interface,
- min = is the minimum index in the range of things that need to be sorted
- max = is the maximum index in the range of things that need to be sorted
- sortType = to tell the function how to sort the array. type.INCREASING will sort it in an increasing manor. Type.DECREASING will sort in a decreasing manor

Output:

- returns nothing

UploadedFile.groovy

String filePath

- Stores the path to the file that has been uploaded
- Is used by the parser to parse the contents of the file into a data object which then goes to the view to draw

String filename

- Stores the name of the file
- This variable is part of the path, It is used to get the csv file to produce a graph data

Name

- This represents the name of the graph
- We do not currently use this variable but its existence has value because now adding more functionality will be easier

X-axis

- This represents name of the xaxis
- We do not currently use this variable but its existence has value because now adding more functionality will be easier

Y-axis

- This represents the y axis
- We do not currently use this variable but its existence has value because now adding more functionality will be easier

Data

- This represents the data the program will graph
- It has the type of an arraylist.
- The elements of the arraylist are of type Point

pieChart.js

width = represents the width of the area that will display the graph height = represents the height of the area that will display the graph

radius = Represents the radius for the pie chart

x(var input**)** = this variable is a function that maps a given set of inputs into a given outputs. Currently it transforms the points with a square root applied to them. This simulates how in a circle large sets of values appear bigger then small sets of values

for example: If you have a set of input from 0 to 100 and an output of 0 to 500. It will map a given x value to an output.

For example: x(0) → 0

x(100) → 500

x(50) → 353

y(var input**)** = This variable is a function that maps a given set of inputs into a given output. Currently it transforms the points with a square root applied to them. This simulates how in a circle large sets of values appear bigger then small sets of values. This is used position certain inputs to a given y value.

For example: If you have a set of input from 0 to 100 and an output of 0 to 750. It will map a given x value to an output.

For example: y(0) → 0

y(100) → 750

y(50) → 530

b = the bound box, This represents the attributes of the bread crumbs which is the place that tells you the path and its percentage. It tells the width, the height, the spacing, and tip/tail

colors = This represents all the data objects and their respective colors. Currently the program generates a random color which will be displayed on the screen. That random color will get stored in this variable with its name and color respectively. At initialization there is only the root because the root will always exist in all cases

totalSize = total size of all the segments.

vis = This represents the chart itself. It is has the width and the height of the public variables called width and height. It is placed inside the chart div. and it is transformed width/2 to the left and height/2 to the right. This allows the entire graph to be displayed.

Partition = this creates a partition object.

arc = This can be called on a selection. It will configure the selection in a manor that will display it in its correct position using the x and y transformation functions. It will also make sure the values are not outside the bounds of the stage

node = holds the current rote of the program. This is used when the user chooses a certain category and that becomes the current node.

json = represents the json object that the graph needs to draw.

createVisualization()

This function takes in a json object and creates a graph of that json object.

Stash()

- This stores the old x and dx variables. This is used when the graph gets transformed, this allows it to transform back.

arcTweenZoom()

- This function allows the pie chart to grow and shrink. When the user clicks on one of its categories it will make that the center and show the rest of the data with respect to it

mouseover()

- When the user's mouse goes over the graph this function gets called. This allows that current path to be highlighted and everything else to become faded. It also displays the path on the top left and the percentage this piece is taking up. This can be called on an array of collections which will call this on whatever node goes over a mouse.

mouseleave()

- Removes all the effects that were done by the mouseover function

getAncestors()

- gets the path of the current node. Meaning all the objects that are over this one towards the center until it reaches the root node.

initializeBreadcrumbTrail()

- This function displays the trail at the top that shows that path of the current node and its percentage

getRandomColor()

- Returns a random color.

drawLegend()

- draws the legend on the screen.

toggleLegend()

- adds or removes the legend depending on if it was previously there.

buildHierchy2()

- takes in a json object and returns a new json object in the correct format needed to display it on the screen

Private Entities

Top

Point.groovy

This class creates an object Point which contains the independent variable, x axis stored as a String and the dependent variable y axis stored as a double. Storing it in this method makes access of required pairs convenient.

private String name;
independent instance variable

private double value;
dependent instance variable

Data.groovy

private String Name;

can be used to store the String name

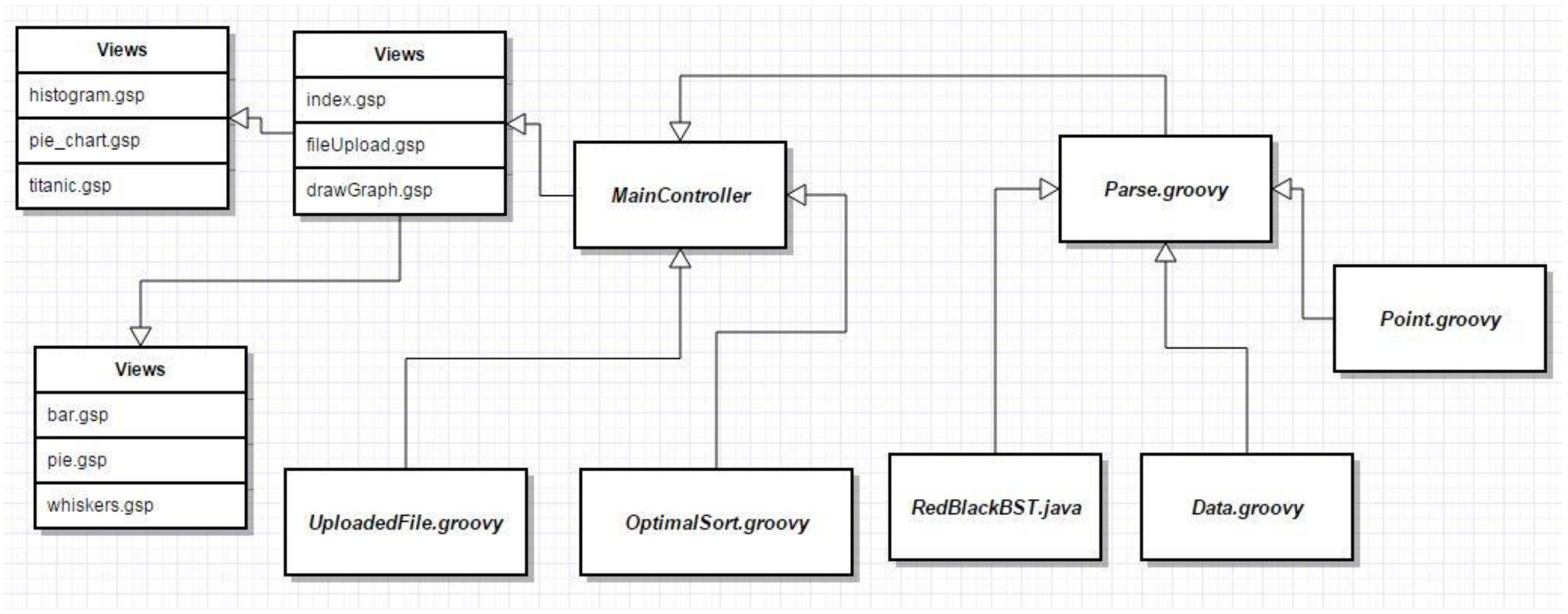
private String xaxis;
Stores the name of the x axis

private String yaxis;
stores the name of the y axis

private ArrayList<Point**> data;**
Stores an ArrayList of point objects for the independent and dependent value pairs.

UML

Top


Views
histogram.gsp
pie_chart.gsp
titanic.gsp
Views
index.gsp
fileUpload.gsp
drawGraph.gsp
MainController
Parse.groovy
Point.groovy
Views
bar.gsp
pie.gsp
whiskers.gsp
UploadedFile.groovy
OptimalSort.groovy
RedBlackBST.java
Data.groovy

Internal Evaluation of Design

Top

Pros	**Cons/areas for improvement**
-utilizes MVC (Model, View Controller) design which allows for more effective group work as the view does not interact with the model and thus when combining the code, knowing the ins and outs of each section are not required -Parser can accommodate a wide variety of data sets in terms of both rows and columns for the independent and dependent variables, can also accommodate multiple dependent variables for the same independent variable -UI is user friendly and aesthetically pleasing, can be used to graph various types of graphs as well as very specific data the user desires -Ability to compare and analyze graphs (display the graph with the weirdest distribution in a given set of points) -Code is well documented and clear instructions are given on how to use the UI -web-page successfully works on various browsers on both Mac and PC as well as cell phones	-could have done better encapsulation of code -organization of code and development could have been better through the use of ADTs for the model -UI could have been even more user friendly and intuitive -could have added successful upload message for datasets on the webpage on the same page as opposed to going to a new web-page -could have added more functionality such as comparing and/or displaying several graphs at once -could have spent more time testing in general -could have made more use of the balanced BST