



Alchemist

NSERC Grant Search and Forecasting Tools Design Specification Document

Version 1.1.0
April 3, 2016
Group Number: 29

Danish Khan: 1217176
Haris Khan: 1413511
Michael Bitzos: 1405050
Junhao Wang: 1215428

SFWR ENG 2XB3

Course: Software Engineering Binding Theory to Practice
Department of Computing & Software
McMaster University

Revision History

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.

Version 1.0.0

- Document created March 19th, 2015. Inserted title page, cover page, application icon, introduction, revision history page, contribution page and modular decomposition.

Version 1.0.1

- Completed MIS/MID for Model and Controller.
- Completed Application Logo, revised MIS/MID methods.
- Added private implementation documentation for all variables.
- Edited Contribution Page

Version 1.0.2

- Added executive summary, and table of contents.
- Revised contribution table.
- Added all referenced books and libraries, download links to libraries and library descriptions.
- Added a project log.
- Revised sections 5.1.1, 6.1.2, 5.1.3, 5.1.4, 6.1.1, 6.1.2, 6.1.3, 6.1.4

Version 1.0.3

- Completed Introduction and Architecture
- Added all online references
- Revised introduction to include the three functional features of the toolset: forecasting, data visualization and searching.
- Inserted modular guide and software architecture diagrams in sections 4.0 and 3.0.
- Updated table of contents to reflect changes in references section.

Version 1.0.4

- Updated UML figures, and Uses Relationship figures.
- Revised introduction.
- Added **Table 1: Grant Columns are associated with a character.**
- Added server client architecture.
- Added state diagram, state machine architecture.

Version 1.0.5

- Added sections 2.1 Overview, 2.2 Client, 2.2.1 Model, 2.2.1.1 Data Structures, 2.2.1.2 Searching & Sorting, 2.2.1.3 Algorithmic Challenges, 2.2.2 View, and 2.2.3 Controller and 2.3 Server.
- Revised MIS variables in Section 6.1.1, 6.1.2, 6.1.3, 6.1.4, 6.1.5, 6.1.6, 6.1.7, and 6.1.8.

Version 1.1.0

- Completed functional and nonfunctional requirements and trace to requirements.
- Edited section 2.2.1.1 Data structures to include List data structures used within the program.
- Added **Figure 2: Server-Client overview** and added architecture overview and description.
- Revised for spelling, grammar, format and syntax errors throughout the document.

Contribution Page

Each row is assigned to one member of the group. This table will list contributions made to the project by each individual member of the group. You cannot repeat a contribution item for another member of a group. You should breakdown the contributions such that each contribution falls into exactly one row. You can add a comment column to clarify the contributions if it is necessary. The contributions listed in this table should be consistent with the project log.

Name	Role(s)	Contributions	Comments
Danish Khan	i. Team Leader ii. Programmer iii. Documenter	<i>Manage project, meet all milestones, and produce the prototype.</i>	Detailed list of tasks can be found below.
Haris Khan	i. Designer ii. Programmer iii. Tester iv. Documenter	<i>Primary Software Architect and toolset Designer.</i>	Detailed list of tasks can be found below.
Michael Bitzos	i. Programmer ii. Log Administrator iii. Documenter	<i>Document all progress, meeting minutes, decisions and progress.</i>	Detailed list of tasks can be found below.
Junhao Wang	i. Programmer ii. Tester iii. Documenter	<i>Primary Test Plan Designer</i>	Detailed list of tasks can be found below.

Danish

- Project compilation, revision, appendix, table of contents, revision and technical writing.
- Executive summary.
- MIS / MID Sections 5.1.1, 5.1.2, 5.1.3, 5.1.4, 5.1.5, 5.1.6, 5.1.7, 5.1.8, 6.1.1, 6.1.2, 6.1.3, 6.1.4, 6.1.5, 6.1.6, 6.1.7, 6.1.8 and private implementation
- Trace to Requirements: requirements
- Conclusion

Haris

- Introduction and Architecture
- Modular Decomposition and Hierarchy: all except 2.3.1 Model
- Modular Guide
- MIS / MID - variable descriptions contributions
- Trace to Requirements: module and explanations
- Uses relationship
- Anticipated changes and discussion

Micheal

- Modular Decomposition Hierarchy: 2.3.1 Model
- MIS / MID Sections 5.1.9, 5.1.10, 6.1.9, 6.1.10

- Test plan / Design [GrantGraph]
- Project Log
- Section 2.3.5.1 Whitebox Testing

Junhao

- ADT Design
- Black-box testing for functionality
- Basic comparator design
- Trace to Requirements in Implementation and Testing(7.2)
- Test plan

Table of Contents

1. Executive Summary.....	8
Figure 1: Alchemist running on Ionic Lab iOS and Android Simulator.....	9
2. Introduction and Architecture.....	10
2.1 Overview.....	10
Figure 2: Server-Client Module Overview.....	10
Figure 3: Server-Client Composition.....	10
2.2 Client.....	11
2.2.1 View.....	11
Table 1: State Descriptions.....	11
Figure 4: State Diagram.....	12
2.3 Server.....	12
Table 2: Grant Columns are associated with a character.....	12
2.3.1 Model.....	13
2.3.1.1 Data Structures.....	13
2.3.1.2 Searching & Sorting.....	13
2.3.1.3 Algorithmic Challenges.....	13
Figure 5: Graph Searching Scheme.....	14
2.3.3 Controller.....	14
2.3.3.1 HTTP Server.....	14
2.3.4 Utility.....	15
2.4. Modular Decomposition and Hierarchy.....	16
2.4.1 Uses Relationship.....	16
Figure 6: Uses Relationship.....	16
2.4.2 UML Diagram.....	17
Figure 7: UML Module Diagram.....	17
5. MIS.....	

	18
5.1 Model.....	18
5.1.1 AlchemistModel.....	18
5.1.2 Grant.....	18
5.1.3 GrantColumn.....	19
5.1.4 GrantDatabase.....	19
5.1.5 Province.....	20
5.1.6 Series.....	20
5.1.7 TimSort.....	21
5.1.8 FileUtils.....	21
5.1.9 GrantGraph.....	21
5.1.10 BreadthFirstPaths.....	22
5.2 Controller.....	22
5.2.1 AlchemistController.....	22
6. MID.....	23
6.1 Model.....	23
6.1.1 AlchemistModel.....	23
6.1.2 Grant.....	23
6.1.3 GrantColumn.....	25
Figure 8: UML State Diagram for GrantColumn.....	26
6.1.4 GrantDatabase.....	26
6.1.5 Province.....	27
6.1.6 Series.....	27
6.1.7 TimSort.....	28
6.1.8 FileUtils.....	31
6.1.9 GrantGraph.....	31
Figure 9: UML State diagram for GrantGraph().....	32
6.1.10 BreadthFirstPaths.....	32
6.2 Controller.....	33

6.2.1 AlchemistController.....	33
7. Trace to Requirements.....	34
7.1 Trace to Requirements.....	34
Table 3: Trace to Requirements.....	34
7.2 Trace to Requirements as it pertains to Implementation and Testing.....	39
9. Internal Review and Analysis of Design.....	40
9.1 Anticipated Change.....	40
9.2 Pros of Design.....	40
9.3 Cons of Design.....	41
10. Test Plan/Design.....	41
10.1 Whitebox Testing.....	41
10.2 Black Box Testing.....	43
10.2.1 Functionality Testing.....	43
11. Conclusion.....	46
Figure 10: Alchemist Toolset	46
12. Appendix.....	48
12.1 List of Figures.....	48
12.2 List of Tables.....	48
13. References.....	49
13.1 Libraries.....	49
13.1.1 Gson.....	49
13.1.2 Spark.....	49
13.1.3 OpenForecast.....	50
13.1.4 Ionic Framework.....	50
13.1.4 Highcharts.js.....	50
13.2 Books.....	50
13.3 Websites.....	51
13.4 Code/Modules.....	51
13.4.1 TimSort.java.....	51

14. Project Log.....	52
14.1 List of Task IDs.....	52
14.2 Project Log.....	53

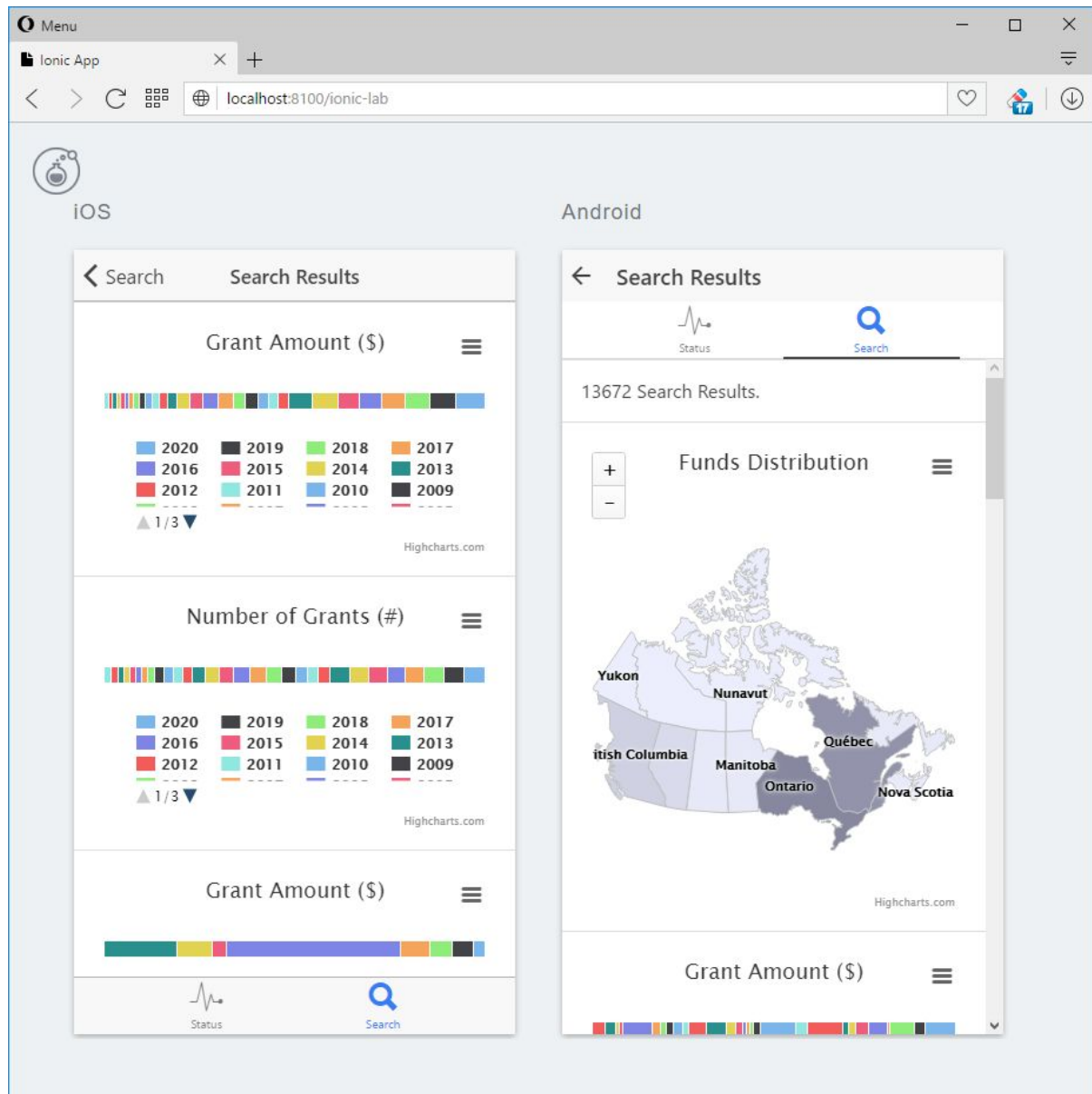
1. Executive Summary

Resource allocation is a very large problem, it is a very difficult problem and it is present in hierarchies across society because it is the consequence of economy and markets.

The Alchemist toolset is an attempt at solving a very particular portion of the problem of resource allocation which is projection or forecasting within the context of the financing of scientific research in Canada. The toolset uses the NSERC (Natural Sciences and Engineering Research Council) grant datasets from 1991 to 2014 to forecast the financial size of a field of research up to five years into the future.

The NSERC is the largest institution funding research in Canada and frequently partners with industry to finance research in University and research institutions across the nation. The dataset is over 0.5 million nodes in size and contains a record of every grant awarded to a scientist working within a Canadian institution.

The toolset provides forecasting, data visualization and searching services through a web based app that can be accessed by any modern browser running Javascript and HTML. The toolset contextualizes all queries through geographic graph data and visualizing other metrics to allow the user to visually experience the result of the query.

Figure 1: Alchemist running on Ionic Lab iOS and Android Simulator

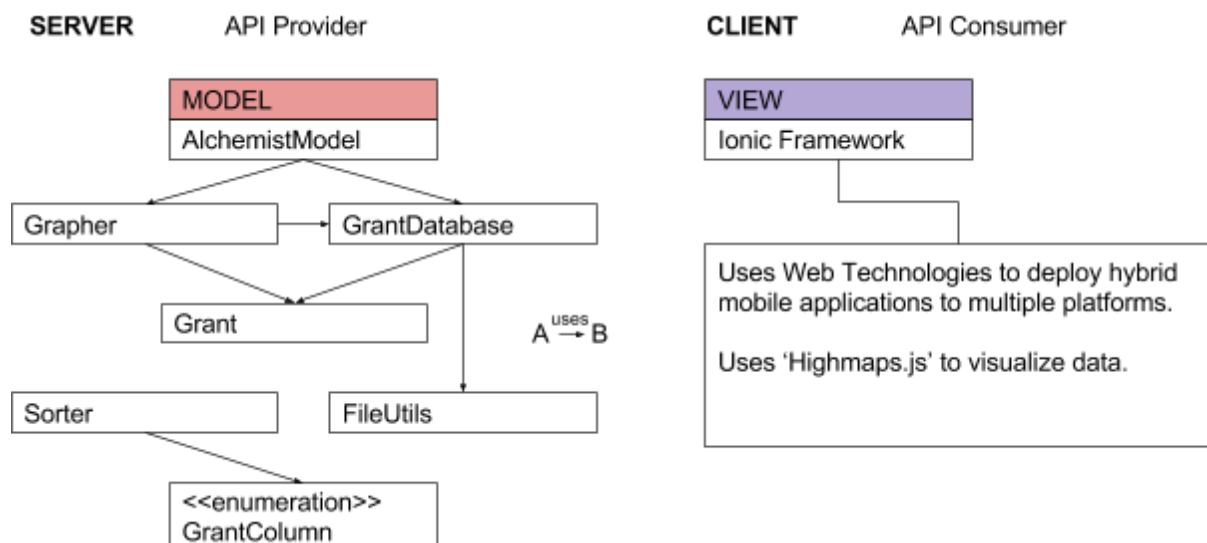
2. Introduction and Architecture

2.1 Overview

This software toolset is developed with the Model-View-Controller paradigm. The business logic and data is contained within a module called “Model.” The model contains the procedures for loading and processing the dataset. The “Controller” module acts as a director for the view; it allowed the view to connect to the model. Finally, the “View” module contains all the necessary classes which allows data to be viewed visually, through an effective UI.

Although the toolset is developed under the Model-View-Controller design, it uses server-client architecture. Behind the scenes, the “Model” and “Controller” modules are what make up the server, while the “View” is what makes the client. The figure below shows the brief overview of the server and client. In the following sections, they are described in detail.

Figure 2: Server-Client Module Overview



The following figure illustrates the compounded dual layered architecture that composes the toolset. On the surface, the architecture can be decomposed to a server client model; The server is then further decoupled to model and controller modules.

Figure 3: Server-Client Composition

Alchemist NSERC Toolset Architecture Breakdown		
SERVER		CLIENT
Model	Controller	View
NSERC Datasets		

2.2 Client

The client uses the highly developed, complex web of modules that is the Ionic Framework, and it is difficult to describe its Uses relationship. The Ionic Framework's documentation can be found online at: <http://ionicframework.com/docs/>.

The following will attempt to describe the client.

2.2.1 View

The client is broken down into 2 tabs.

- The “Server Tab” is responsible for
 - Showing the status of the connection to server.
 - Allowing the user to change connection settings.
- The “Search Tab” is responsible for
 - Providing an interface through which the user can search for grants.
 - Transitioning into the search results state.

The search results screen is a separate off-screen tab which stacks on top of the “Search Tab” after a valid search is conducted. All these tabs (on-screen and off-screen) are managed using a state machine which the Ionic Framework automatically handles.

Table 1: State Descriptions

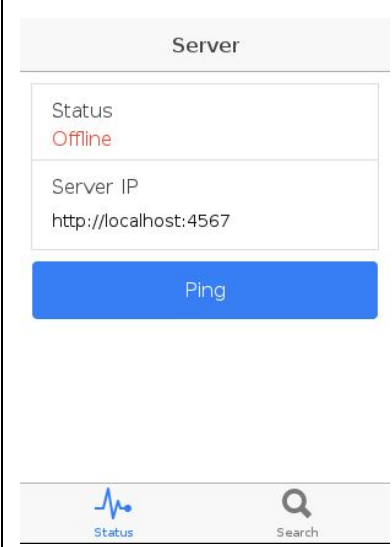
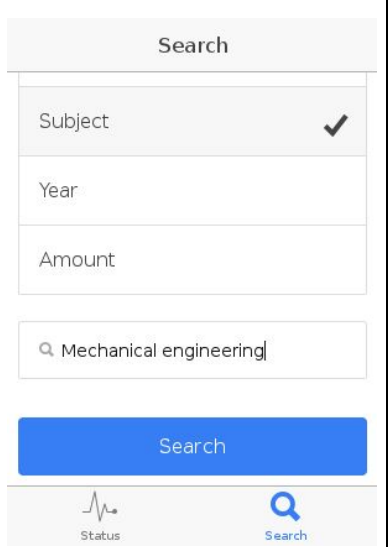

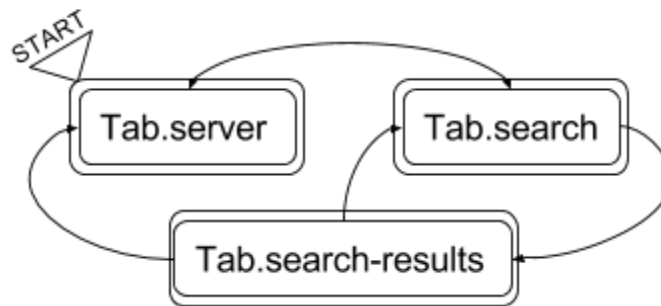
Tab.server	Tab.search	Tab.search-results
		
Manages the server connection. User is able to change the Server IP and ping the server.	Acts as the search interface; transitions to Tab.search-results when user searches.	Shows the results geographically and through bar graphs. All visuals are interactive.

Figure 4: State Diagram

The user is able to exit the app at any time without consequences

2.3 Server

The server is responsible for doing all the calculations and manipulations of the datasets, and serve JSON data to the clients upon requests. In order to simplify the problem of categorizing the data on the server, the following grant column schema was devised.

Table 2: Grant Columns are associated with a character.

Grant Column	Character abbreviation	Example
Id (A unique number identifier)	i	20481
Field of Research	f	Mechanical engineering
Organization / Institution	o	McMaster University
Professor's name	n	"Fox Kristi"
Province	p	Ontario
Subject	s	Communications systems
Year	y	2003
Amount (Currency: CAD)	a	10000
Number (Number of grants received)	z	100

This allowed grants to be classified based on specific grant columns.

In terms of the technology stack used, the server is written in Java and uses gradle to manage its dependencies which include:

- GSON used to convert Java objects to JSON strings.
- Spark used to manage an HTTP server.
- OpenForecast used to find trends in data and predict the future

The server fulfills the following functions:

1. Loads the raw database csv files into RAM
2. Parses and sanitizes the data into objects
3. Converts the sanitized raw data into an optimized JSON format
4. Saves optimized data to a flat file (and loads from this optimized file in the future)
5. Creates a local HTTP server which exposes API to clients.

The server is divided into four modules:

- Model Module Contains business logic classes: cohesive, and hierarchical
- Controller Module Contains AlchemistController (Program entry point; boots up server)
- Utility Module Contains all the utility classes (independent units)
- Test Module Contains all tests

2.3.1 Model

Refer to Figure 6 section 2.4 for an overview of the model and sections 5 and 6 for a details on implementation.

2.3.1.1 Data Structures

Lists - Lists were mainly used to locally store all the grants in the database in AlchemistController and GrantDatabase. The reason being this is that adding to an array would take a lot of array resizings due to the large amounts of grants being transferred to the client side. In addition the list data structures were used when reading from a file as it allows to quickly add the the data being inputted line by line in FileUtils.

Enum - An enum data type [in Provinces] was additionally used to hold the names of provinces in Canada. An enum was optimal because of the fact that the provinces were all static and it helped keep everything consistent in terms of retrieving information.

HashMap - Finally for the GrantGraph, we had used a HashMap data type to hold the adjacency list for the tree. The reasoning for this was that it allowed us to store edges that were not index based but rather key based. Since we couldn't use an array-like data type using a HashMap with $O(1)$ complexity for value access was crucial due to the sheer size of the graph.

2.3.1.2 Searching & Sorting

Searching- For searching, Alchemist uses a binary search to search for individual grants in the GrantDatabase. Since the GrantDatabase is sorted upon initialization of the program, binary search was a perfect choice to find the grants with $O(\log(N))$ in the worst case.

Sorting - For sorting, Alchemist uses Timsort to sort the list of grants in GrantDatabase. There were several reasons for choosing Timsort over mergesort/quicksort. The first reason was due to its $O(N \log(N))$ in the worst and average case, this offers the fastest complexity in terms of compare based sorting. Another reason was that it's best case (pre-sorted database) was only $O(N)$ complexity. With the chance of having a pre-sorted database taken from the NSERC being quite high, Timsort was an essential sorting algorithm to have in Alchemist.

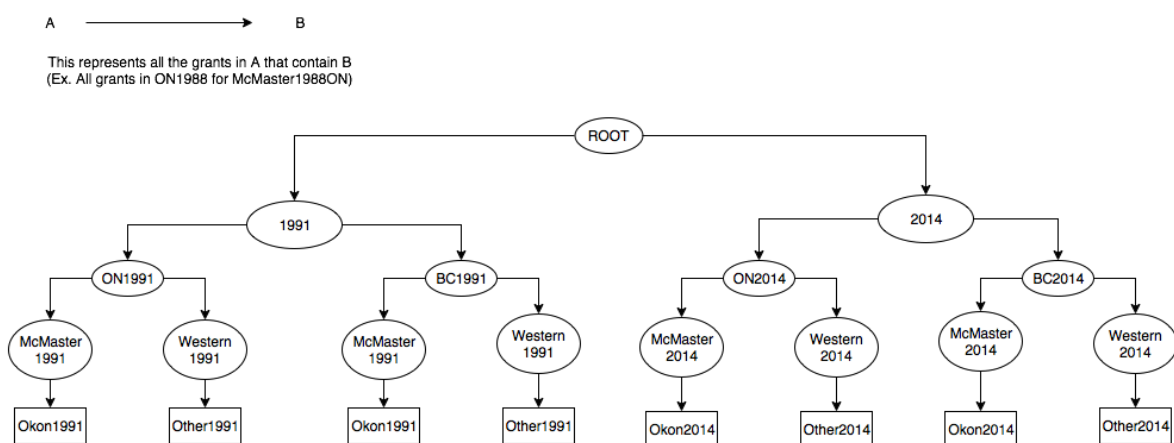
2.3.1.3 Algorithmic Challenges

Since part of Alchemist's functionality includes pulling all the large amount of data on the client side for graphical representation, having to find the absolute best algorithm to optimize for time was a

huge factor in development. Choosing a search algorithm with the best complexity of $O(\log(n))$ and a sorting algorithm with the best complexity of $O(N\log(N))$ was one of the development team's top priority.

For the graphing, one of our challenges was to form the data into a graph that we could use and would help. Our decision for the data was to form it into a tree structure [see Fig. 5]. We had created a directed weighted graph where the edges of the tree represented the sum of the grants for the subtree. For the tree we had decided to use the BFS algorithm to traverse the tree to find a specific path from the root to a node. This was used to help search for a specific edge weight in the tree which in parallel works to find the sum of the grants for that sub tree. We had used BFS to find a path for the tree as the space and time complexity of it being only $O(V+E)$ made it a fast algorithm for the tree.

Figure 5: Graph Searching Scheme



2.3.3 Controller

The controller is made of just a single class: AlchemistController. Using the Spark library, the controller boots an HTTP server, which is specified below.

2.3.3.1 HTTP Server

The HTTP Server responds to client requests in JSON, and exposes the following API:

- **/api/ping**
Responds with "**Success**". (Used to test client's connection with the server)
- **/api/get/:column/:object**
Responds with a list of grants which have **:column** equal to **:object**
 - **:column** Character from Table 1 (Column to search)
 - **:object** The object which all returned grants will contain (see Table 1 for a list of appropriate column characters)

I.e. <http://localhost/api/get/p/Ontario> will get grants where province = Ontario.
- **/api/search/:column/:object/generate/:x/:y**
 - **:column** Character from Table 1 (Column to search)
 - **:object** The object which all returned grants will contain
 - **:x** Character from Table 1 (X-axis of series of data)

- **:y** Character from Table 1 (Y-axis of series of data)
Responds with a series of data which can be plotted to a graph.
I.e. <http://localhost/api/search/p/Ontario/generate/y/a> will generate a series of data for all grants in ontario, with the x-axis as the year, and the y-axis as the grant amount.

2.3.4 Utility

The Utility module was made up of classes which were independent units and could be replaced individually. Currently, the Utility module consists of:

- **TimSort** Exposes a generic sort method for any List<T>.
- **FileUtils** Exposes a method to read files, returning a list of lines.
- **Series & SeriesData** A data structure to help manage data relationships.

TimSort and FileUtils are straight-forward. SeriesData is a dependency of the Series utility class. The Series class is a data structure which helps package time-series data into packages which can be forecasted or optimized for better performance on the client-side.

Forecasting in the Series class is done using the OpenForecast library, which simply takes all the points in the series, finds the best forecasting model, and projects the next few points in the series. Documentation of the OpenForecast library can be found [here](#). Further details on the description and features of this library are found in section 13.1.3.

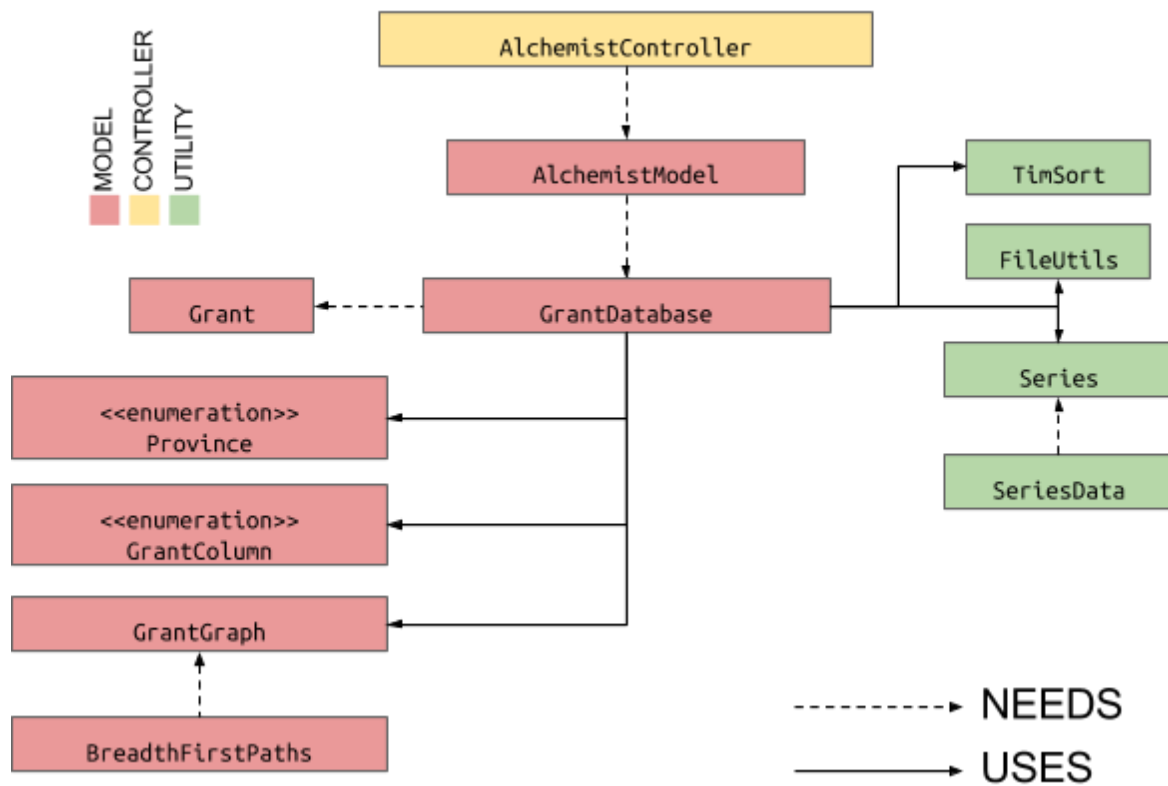
Optimizing is a procedure where data that is insignificant is removed and packaged under the “Other” label.

2.4. Modular Decomposition and Hierarchy

In the following sections, Uses-Relationship charts and UML diagrams for the server and client are presented. The Test module is omitted from the diagrams.

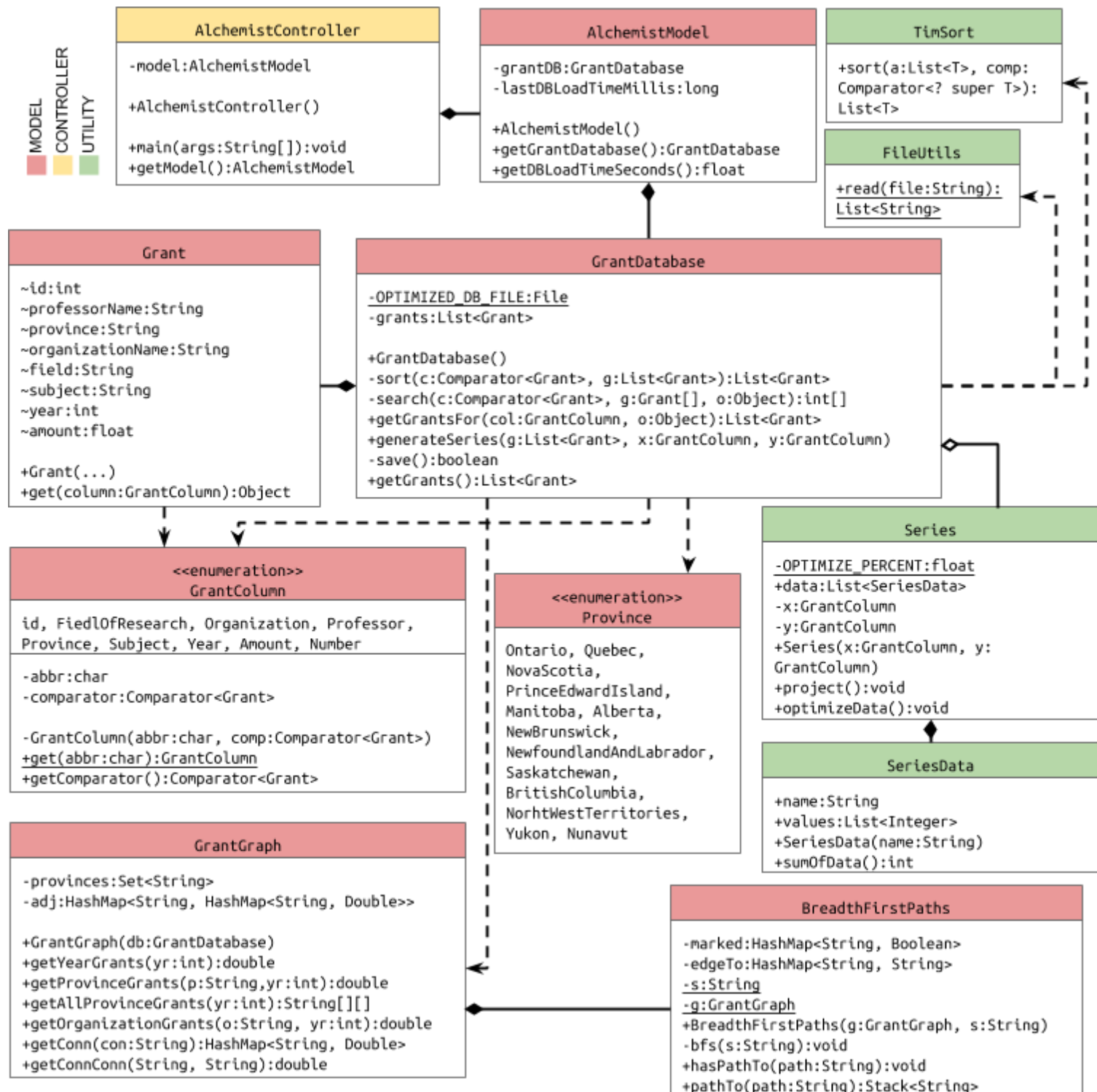
2.4.1 Uses Relationship

Figure 6: Uses Relationship



2.4.2 UML Diagram

Figure 7: UML Module Diagram



The UML diagram above shows that the architecture is decoupled and very modular. There are no cyclic or unnecessary dependencies between the classes.

5. MIS

5.1 Model

5.1.1 AlchemistModel

Class AlchemistModel.java

Package xb3_Final_Project.model

Uses

GrantDatabase

Type

None

Access Programs

AlchemistModel(): AlchemistModel

Creates the model of this web application and initializes a GrantDatabase object

getGrantDatabase(): GrantDatabase

Returns an entire GrantDatabase object

getDBLoadTimeSeconds(): float

Returns the time taken to create and initialize the grantdatabase object in second

5.1.2 Grant

Class Grant.java

Package xb3_Final_Project.model

Uses

None

Type

None

Access Programs

Grant(): Grant

Initializes a new Grant object with a default ID tag of -1 and empty uninitialized parameters.

Grant(id:int, field:String, institutionName:String, professorName:String, province:String, researchSubject:String, year:int, amount:float): Grant

Initializes a new Grant object with the following input parameters id, field, institution name, professor name, province name, research subject name, year, amount awarded.

getID(): int

Returns the integer id of this grant object

getProfessorName(): String

Returns the professor name string parameter of this grant object

getProvince(): String

Returns the province string parameter of this grant object

getOrganizationName(): String

Returns the organization name string parameter of this grant object

getSubject(): String

Returns the subject name string parameter of this grant object

getField(): String

Returns the field of research string parameter of this grant object

getAmount(): float

Returns the amount awarded float parameter of this grant object

getYear(): int

Returns the integer year in which this grant was awarded

get(x:GrantColumn): Object

Returns the province string parameter of this grant object

5.1.3 GrantColumn

Class GrantColumn.java

Package xb3_Final_Project.model

Uses

Comparator<Grant>

Type

enum

Access Programs

get(char column): GrantColumn

Returns a GrantColumn object based on the character abbreviation input.

The valid character denotations for this method are the following:

input ' i ' returns the column id GrantColumn object

input ' f ' returns the field of research GrantColumn object

input ' o ' returns the organization GrantColumn object

input ' n ' returns the professor GrantColumn object

input ' p ' returns the province GrantColumn object

input ' s ' returns the subject of research GrantColumn object

input ' y ' returns the year awarded GrantColumn object

input ' a ' returns the amount GrantColumn object

input ' z ' returns the number GrantColumn object

getComparator(): Comparator<Grant>

Returns the appropriate comparator for a column.

5.1.4 GrantDatabase

Class GrantDatabase.java

Package xb3_Final_Project.model

Uses

Gson

Type

None

Access Programs

GrantDatabase(boolean useBackup)

Standard constructor which loads and sanitizes the database files, then save them in an efficient format for faster loading in the future.

sort(Comparator<Grant> comparator, List<Grant> grants): List<Grant>

Sorts the given list of grants based on a comparator.

search(GrantColumn column, Grant[] array, Object searchFor): int[]

Searches an array of grants in a specific column for a specific grant.

getGrantsFor(GrantColumn column, Object searchFor): List<Grant>

Obtains grants for a given column based on a search criteria.

getTotalGrantAmount(List<Grant> grants): float

Returns total grant amount for grants

getTotalNumberOfGrantsAwardedFor(List<Grant> grants): int

Computes the total number of grants awarded for in a List<Grant> object.

getAverageGrantAmountFor(List<Grant> grants): float

Returns average grant amount for list of grants given

getProjectedGrantAmountFor(GrantColumn column, Object key): float

Returns projected grant amount for a specific column and key.

save(): boolean

Saves the database in an efficient JSON format to a flat file.

getGrants(): List<Grant>

Accesses all the grants in the database.

forecastFor(GrantColumn x, GrantColumn y): Series

Forecasts grants for the given columns.

generateSeries(List<Grant> grants, GrantColumn x, GrantColumn y): Series

Generates a series of grants based on a given X, Y column scheme. These coordinate can then be used to generate a graph of the this list.

5.1.5 Province

The province enumeration provides a validation service. Used primarily to sanitize the database before unloading to the server.

Class Province.java

Package xb3_Final_Project.model

Uses

None

Type

enum

Access Programs

isProvince(String name): boolean

Returns true is the string name is a valid registered province in the database.

5.1.6 Series

This object provides column and data correlational services, and plotting ready data formats.

Class Series.java

Package xb3_Final_Project.model

Uses

None

Type

None

Access Programs

Series(GrantColumn x, GrantColumn y)

Constructs a series object which relates GrantColumn object X to GrantColumn object Y. The series object simplifies plotting services for the server.

optimizeData(): void

Optimizes the data in this series if optimization options are available for this run.

project(): void

Forecasts the next few years in the series of the x-column == GrantColumn.Year

5.1.7 TimSort

This object provides sorting services used primarily by the model.

Class TimSort.java

Package xb3_Final_Project.model

Uses

Comparator<Object>

Type

None

Access Programs

sort(List<T> list, Comparator<? super T> c): <T>List<T>:

Sorts the given list list via the Comparator.

5.1.8 FileUtils

This object provides file reading, file writing and file availability checking services that are accessed by the model.

Class FileUtils.java

Package xb3_Final_Project.model

Uses

None

Type

None

Access Programs

read(String file): public static List<String>

Returns the line of a specified file as a List<String> object.

5.1.9 GrantGraph

Forms the grantData into a tree using hashmaps.

Class GrantGraph

Package xb3_Final_Project.model

Uses

GrantDatabase

Grant

Type

getAllProvinceGrants()

getOrganizationGrants()

getProvinceGrants()

getYearGrants()

Access Programs

GrantGraph(GrantDatabase:db): public GrantGraph

Initializes and creates the graph with the GrantDatabase object passed through.

Puts each individual grant in the tree's appropriate branch and keeps track of the sums by keeping the edges of the graph up to date when a new Grant is added

getYearGrants(int:year): public double

Returns the total grant amount for that year.

getProvinceGrants(String:province, int:year): public double

Returns the total grant amount for that province for that year specified if it exists.

getAllProvinceGrants(int:year):String[][]

Returns a list of all the provinces with all their total grants for the specified year.

getOrganizationGrants(String:organization, int:year): public double

Returns the total grant amount for that organization for the specified year.

getConn(String:Location): public HashMap<String,Double

Returns the branches of the tree specified at the node.

getConnConn(String:adjLocation, String:location):public double

Returns the grant amount of the tree's branches connection specified at the nodes.

5.1.10 BreadthFirstPaths

Traverses the graph by performing the BFS algorithm which gives a path to all nodes in the graph.

Class BreadthFirstPaths

Package xb3_Final_Project.model

Uses

None

Type

pathTo(String:v):Stack<String>

Access Programs

BreadthFirstPaths(GantGraph:g, String:s): public BreadthFirstPaths

Initializes the BFS path at a node specified.

hasPathTo(String:v):public boolean

Returns if there is a path from the starting node in the BFS to the node specified.

pathTo(String v):public Stack<String>

Returns the path from the starting node to v in a stack.

5.2 Controller

5.2.1 AlchemistController

This controller controls the startup of the toolset. The controller handles and manages all resource and dataset acquisition procedures, server and client side building and communication bridging and data pipelining to the view.

Class AlchemistController.java

Package xb3_Final_Project.model

Uses

AlchemistModel

Type

None

Access Programs

AlchemistController()

Constructor for the Alchemist Controller. This constructor boots up a RESTful server, which acts as the API provider.

6. MID

6.1 Model

6.1.1 AlchemistModel

Class AlchemistModel.java

Package xb3_Final_Project.model

Variables

grantDB: GrantDatabase

Database object used for query and sorting services. Holds all available NSERC datasets. This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor.

long lastDBLoadTimeMillis: long

Used in testing and validating requirements specification.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor and getDBLoadTimeSeconds().

Access Programs

AlchemistModel(): AlchemistModel

Creates the model of this web application and initializes a GrantDatabase object

getGrantDatabase(): GrantDatabase

Returns an entire GrantDatabase object

getDBLoadTimeSeconds(): float

Returns the time taken to create and initialize the grantdatabase object in seconds

6.1.2 Grant

Class Grant.java

Package xb3_Final_Project.model

Variables

id: int

Denotes the id of this grant object

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor.

professorName: String

Denotes the the name of the professor this grant was awarded to.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor.

province: String

Denotes the name of the province in which this grant was awarded.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor.

organizationName: String

Denotes the name of the professor's affiliated organization.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor.

field: String

Denotes the name of the field of research this grant funded.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor.

subject: String

Denotes the name of the subject of research this grant funded.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor.

year: int

Denotes the year in which this grant was awarded.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor.

amount: float

Denotes the amount awarded in CAD.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor.

Access Programs

Grant(): Grant

Initializes a new Grant object with a default ID tag of -1 and empty uninitialized parameters.

Grant(id:int, field:String, institutionName:String, professorName:String, province:String, researchSubject:String, year:int, amount:float): Grant

Initializes a new Grant object with the following input parameters id, field, institution name, professor name, province name, research subject name, year, amount awarded.

getID(): int

Returns the integer id of this grant object

getProfessorName(): String

Returns the professor name string parameter of this grant object

getProvince(): String

Returns the province string parameter of this grant object

getOrganizationName(): String

Returns the organization name string parameter of this grant object

getSubject(): String

Returns the subject name string parameter of this grant object

getField(): String

Returns the field of research string parameter of this grant object

getAmount(): float

Returns the amount awarded float parameter of this grant object

getYear(): int

Returns the integer year in which this grant was awarded

get(x:GrantColumn): Object

Returns the province string parameter of this grant object.

6.1.3 GrantColumn

Class GrantColumn.java

Package xb3_Final_Project.model

Variables

char: abbr

Holds the character abbreviation of the GrantColumn so that columns can be represented by characters, making messaging between server and client more efficient and error free.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor and the getChar() method.

comparator: Comparator<Grant>

The comparator for the specific column; used in sorting and searching.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor.

Access Programs

get(char column): GrantColumn

Returns a GrantColumn object based on the character abbreviation input.

The valid character denotations for this method are the following:

input ' i ' returns the column id GrantColumn object

input ' f ' returns the field of research GrantColumn object

input ' o ' returns the organization GrantColumn object

input ' n ' returns the professor GrantColumn object

input ' p ' returns the province GrantColumn object

input ' s ' returns the subject of research GrantColumn object

input ' y ' returns the year awarded GrantColumn object

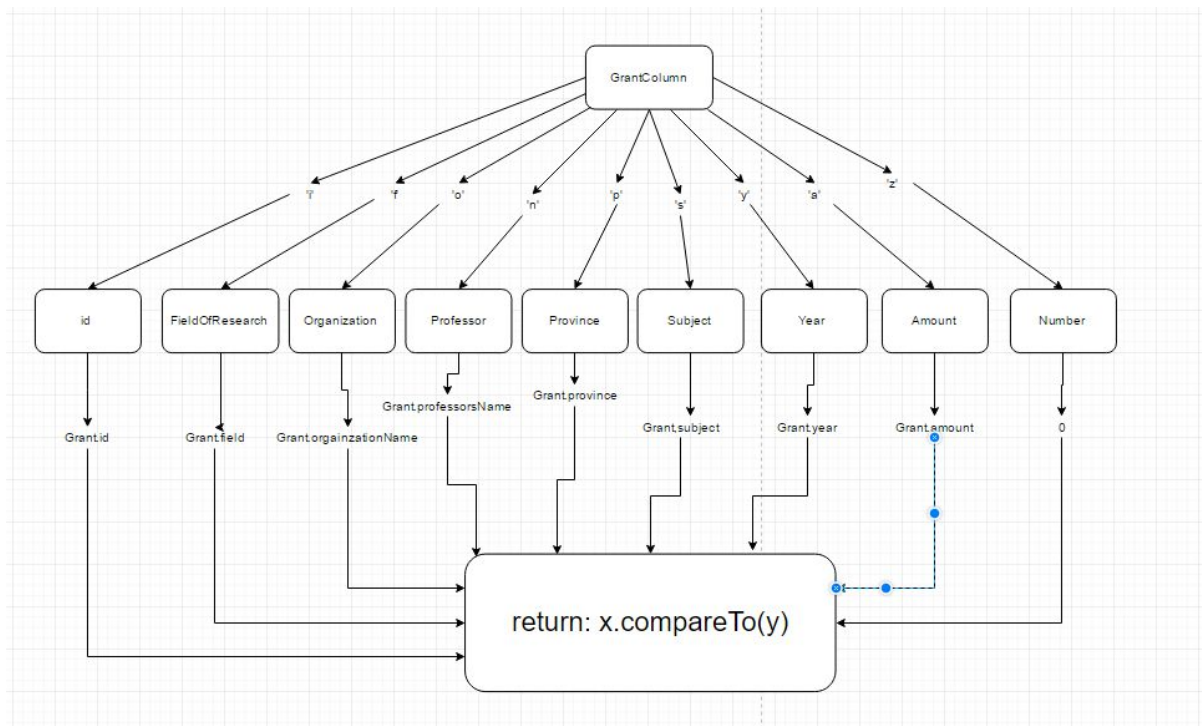
input ' a ' returns the amount GrantColumn object

input ' z ' returns the number GrantColumn object

getComparator(): Comparator<Grant>

Returns the appropriate comparator for a column. [See Figure 8]

Figure 8: UML State Diagram for GrantColumn



6.1.4 GrantDatabase

Class GrantDatabase.java

Package xb3_Final_Project.model

Variables

TEMPLATE: String (final)

Database CSV filename template which is used to open the csv files.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor and the getChar() method.

grants: List<Grant>

Holds the grants after they are parsed and created from the database.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor, sort(), search(), getGrantsFor(), getGrants(), and the generateSeries() methods.

Access Programs

GrantDatabase(): GrantDatabase

Initializes an empty GrantDatabase object

GrantDatabase(boolean useBackup): GrantDatabase

Standard constructor which loads and sanitizes the database files, then save them in an efficient format for faster loading in the future.

sort(Comparator<Grant> comparator, List<Grant> grants): List<Grant>

Sorts the given list of grants based on a comparator.

search(Comparator<Grant> grantComparator, Grant[] array, Grant key): int[]

The following function sorts a grant column based on a grant search, then returns the start and end indices of all grants that match that search in the following format;

Returns an integer array [A,B] where;

A - denotes the start index

B - denotes the end index

search(GrantColumn column, Grant[] array, Object searchFor): int[]

Searches an array of grants in a specific column for a specific grant.

getGrantsFor(GrantColumn column, Object searchFor): List<Grant>

Obtains grants for a given column based on a search criteria.

getTotalGrantAmount(List<Grant> grants): float

Returns total grant amount for grants

getTotalNumberOfGrantsAwardedFor(List<Grant> grants): int

Computes the total number of grants awarded for in a List<Grant> object.

getAverageGrantAmountFor(List<Grant> grants): float

Returns average grant amount for list of grants given

getProjectedGrantAmountFor(GrantColumn column, Object key): float

Returns projected grant amount for a specific column and key.

save(): boolean

Saves the database in an efficient JSON format to a flat file.

getGrants(): List<Grant>

Accesses all the grants in the database.

forecastFor(GrantColumn x, GrantColumn y): Series

Forecasts grants for the given columns.

generateSeries(List<Grant> grants, GrantColumn x, GrantColumn y): Series

Generates a series of grants based on a given X, Y column scheme. These coordinate can then be used to generate a graph of the this list.

6.1.5 Province

Class Province.java

Package xb3_Final_Project.model

Variables

name: String

The name of the province to examine or set.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor.

Access Programs

Province(String name)

Creates a new Province enumeration object with a specified String name.

isProvince(String name): boolean

Returns true is the string name is a valid registered province in the database.

6.1.6 Series

Class Series.java

Package xb3_Final_Project.model

Variables

data: List<SeriesData> Contains the grant data for this Series object.

x: GrantColumn - GrantColumn object with x coordinate data used in search.

y: GrantColumn - GrantColumn object with y coordinate data used in search.

optimizable: boolean - initialized to true if the constructor detects optimizability in input data used to initializing a Series object.

Access Programs

Province(String name)

Creates a new Province enumeration object with a specified String name.

isProvince(String name): boolean

Returns true if the string name is a valid registered province in the database.

6.1.7 TimSort

Class TimSort.java

Package *xb3_Final_Project.model*

Variables

MIN_MEGE: int = 32

This is the minimum sized sequence that will be merged. Shorter sequence will be lengthened by calling binarySort. If the entire array is less than this length, no merges will be performed. This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor, sort(), and minRunLength() methods.

MIN_GALLOP: int = 7

During galloping mode, we stay there until both runs win less often than MIN_GALLOP consecutive times.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor, and mergeLo() methods.

INITIAL_TMP_STORAGE_LENGTH: int = 256

This variable stores the maximum size of the temporary array used in merging. The temporary array is grown to accommodate demand. 256 is the standard maximum size that this array will reach whilst performing storage for merging services in this module.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor.

minGallop: int

This variable controls when the galloping mode of TimSort is entered. The mergeLo and mergeHi methods sets this variable higher for random data, and lower for highly structured data.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor, mergeLo(), and mergeHi() methods.

tmp: T[]

Temporary storage array used within merging services.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor, mergeAt(), gallopLeft(), gallopRight(), mergeLo(), mergeHi(), and the ensureCapacity() method.

a: T[]

This array serves as a pointer for the array to be sorted in the format of a specified array of T type objects.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor.

c: Comparator<? super T>

The specified comparator to be used to sort the given input array of elements.

stackSize: int

Denotes the number of pending runs yet to be merged on a stack.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor, sort(), pushRun(), mergeCollapse(), mergeForceCollapse(), and the mergeAt() method.

runBase: int[]

Used to store the initial base addresses of a current run.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor, pushRun(), and the mergeAt() methods.

runLen: int[]

Used to store the lengths of a current run.

This variable is kept private to reduce coupling between modules, and implement information hiding. It is maintained by the constructor, sort(), and the pushRun() methods.

Access Programs**TimSort(T[] a, Comparator<? super T> c)**

Creates a new TimSort instance to maintain the state of an ongoing sort.

sort(List<T> aList, Comparator<? super T> c): <T> List<T>

This method, which is packaged, private and static, constitutes the API of this class.

Each of these methods obeys the contract of the public method with the same signature in java.util.Arrays.

sort(T[] a, Comparator<? super T> c): void

This method, which is packaged, private and static, constitutes the API of this class.

Each of these methods obeys the contract of the public method with the same signature in java.util.Arrays.

sort(T[] a, int lo, int hi, Comparator<? super T> c): void

This method, which is packaged, private and static, constitutes the API of this class.

Each of these methods obeys the contract of the public method with the same signature in java.util.Arrays.

binarySort(T[] a, int lo, int hi, int start, Comparator<? super T> c): void

Sorts a specified portion of a specified array using the binary insertion sort algorithm.

The binary insertion sort method is the best method for sorting small arrays.

It requires a $O(n \log(n))$ compares but only $O(n^2)$ element exchanges in the worst case.

countRunAndMakeAscending(T[] a, int lo, int hi, Comparator<? super T> c) : int

Returns the length of the run beginning at the specified position in the specified array and reverses the run if it is descending (ensuring that the run will always be ascending when the method returns).

A run is the longest ascending sequence with:

$$a[lo] \leq a[lo + 1] \leq a[lo + 2] \leq \dots$$

or the longest descending sequence with:

$$a[lo] > a[lo + 1] > a[lo + 2] > \dots$$

For its intended use in a stable mergesort, the strictness of the definition of "descending" is needed so that the call can safely reverse a descending sequence without violating stability.

reverseRange(Object[] a, int lo, int hi): void

Reverses the specified range of elements within the specified array.

minRunLength(int n): int

Returns the minimum acceptable run length for an array of the specified length.

Natural runs shorter than this will be extended with

Roughly speaking, the computation is:

If $n < MIN_MERGE$, return n (it's too small to bother with fancy stuff).

Else if n is an exact power of 2, return $MIN_MERGE/2$.

Else return an int k , $MIN_MERGE/2 \leq k \leq MIN_MERGE$, such that n/k is close to, but strictly less than, an exact power of 2.

pushRun(int runBase, int runLen): void

Pushes the specified run onto the pending-run stack.

mergeCollapse(): void

Examines the stack of runs waiting to be merged and merges adjacent runs until the stack invariants are reestablished:

1. $runLen[i - 3] > runLen[i - 2] + runLen[i - 1]$

2. $runLen[i - 2] > runLen[i - 1]$

This method is called each time a new run is pushed onto the stack, so the invariants are guaranteed to hold for $i < stackSize$ upon entry to the method.

mergeForceCollapse(): void

Merges all runs on the stack until only one remains. This method is called once, to complete the sort.

mergeAt(int i): void

Merges the two runs at stack indices i and $i+1$. Run i must be the penultimate or antepenultimate run on the stack. In other words, i must be equal to $stackSize-2$ or $stackSize-3$.

gallopLeft(T key, T[] a, int base, int len, int hint, Comparator<? super T> c): int

Locates the position at which to insert the specified key into the specified sorted range; if the range contains an element equal to key, returns the index of the leftmost equal element.

gallopRight(T key, T[] a, int base, int len, int hint, Comparator<? super T> c): int

Like `gallopLeft`, except that if the range contains an element equal to key, `gallopRight` returns the index after the rightmost equal element.

mergeLo(int base1, int len1, int base2, int len2): void

Merges two adjacent runs in place, in a stable fashion. The first element of the first run must be greater than the first element of the second run ($a[base1] > a[base2]$), and the last element of the first run ($a[base1 + len1 - 1]$) must be greater than all elements of the second run.

For performance, this method should be called only when $len1 \leq len2$; its twin, `mergeHi` should be called if $len1 \geq len2$. (Either method may be called if $len1 == len2$.)

mergeHi(int base1, int len1, int base2, int len2): void

Like `mergeLo`, except that this method should be called only if $len1 \geq len2$; `mergeLo` should be called if $len1 \leq len2$. (Either method may be called if $len1 == len2$.)

ensureCapacity(int minCapacity): T*[] (Note: where T is a generic object)

Ensures that the external array `tmp` has at least the specified number of elements, increasing its size if necessary. The size increases exponentially to ensure amortized linear time complexity.

rangeCheck(int arrayLen, int fromIndex, int toIndex): void

Checks that fromIndex and toIndex are in range, and throws an appropriate exception if they aren't.

6.1.8 FileUtils

Class FileUtils.java

Package xb3_Final_Project.model

Variables

None

Access Programs

read(String file): public static List<String>

Returns the line of a specified file as a List<String> object.

6.1.9 GrantGraph

Class GrantGraph

Package xb3_Final_Project.model

Variables

Provinces: Set<String>

Holds the unique list of provinces in the database. Maintained in getAllProvinceGrants to loop through all the unique provinces. Held private to reduce code coupling and encapsulation.

Adj: HashMap<String,HashMap<String,Double>>

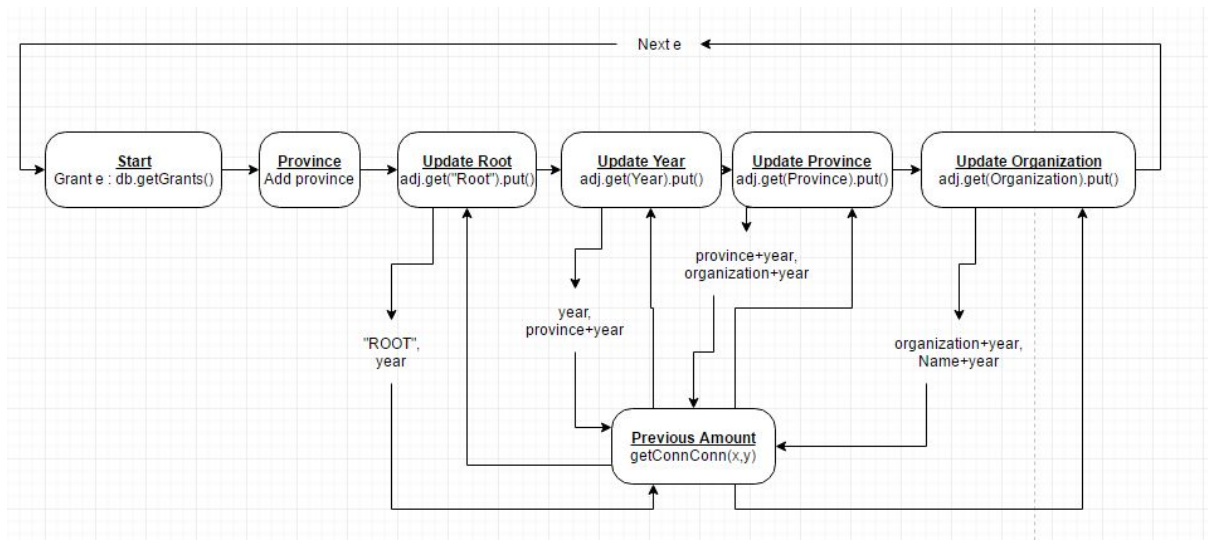
Holds the list of edges of the graph, Instead of holding an edge class as the object the second inner HashMap acts as an edge between the key of the first HashMap and the key of the second, where the value in the second hashMap (type: double) is the weight of the graph. Held private to reduce code coupling and encapsulation. Maintained in GrantGraph(), getConnConn() and getConn().

Access Programs

GrantGraph(GrantDatabase db): GrantGraph

The constructor for the class that initializes provinces, and adj. It first starts by adding the root of the graph by creating a new edge from "Root" to nothing then adding the adj. Then it loops through each grant in db.getGrants() adding the grants to the tree in their appropriate placement, whilst adding their province to the provinces set.

To maintain the structure of the weighted tree, it will first pull the previous value of the tree at that edge (using getConnConn()) and then override it with the oldAmount + the current grants.getAmount() by adding to adj. It will do this 4 times where it will first add it to the "ROOT", the year, the province under the year, the institute for that specific year and province, then finally add the professor's name (which symbolizes the individual grant) [See Figure 8].

Figure 9: UML State diagram for GrantGraph()**getYearGrants(int:year):double**

Returns the edge weight from ROOT to the year by calling adj.get(ROOT).get(year).

getProvinceGrants(String:province, int: year):double

Returns the edge weight from year to province only if it exists in the adj HashMap.

Does this by called adj.get(year).get(provinceYear)

getAllProvinceGrants(int: year):String[][]

Loops through the set of provinces and keeps track of the province in a 2d array where the first dimension is the province and the second dimension is the weight of edge from year to that province. This is done by looping through each province and calling getProvinceGrants(province,year).

getOrganizationGrants(String:organization, int:year):double

Loops through all provinces in adj.get(year).keyset() to search for the organization.

Does this by returning adj.get(province).get(organization) if it is contained in the hashmap.

getConn(String:location): HashMap<String,Double>

If the adj.get(location) does not exists, then it adds it to the adj hashMap with an empty hashMap as the value.Returns adj.get(location)

getConnConn(String: adjLocation, String:location):double

Gets the edge for adjLocation by calling getConn(adjLocation).

If the edge is not connected to location then add the location to it with no weight

Returns the edge at the location.

6.1.10 BreadthFirstPaths

Name BreadthFirstPaths

Package xb3_Final_Project.model

Uses

GrantGraph

Variables

marked: HashMap<String,boolean>

Holds whether this node has been visited by the BFS. Held private as other classes do not need access, and to reduce coupling. Maintained in bfs() and hasPathTo(), BreadthFirstPaths().

edgeTo: HashMap<String,String>

Holds the information for the edge in the BFS path from node A to node . Held private as pathTo is responsible for building a path in the BFS and other modules do not need access to edges. Maintained by: bfs(), pathTo(), BreadthFirstPaths().

S: final String

Holds the starting node name for the bfs to start searching from. Held private as there is no reason for other modules to access this and it is set as constant so other modules can not modify even if public. Maintained in BreadthFirstPaths() and pathTo().

G: final GrantGraph

Holds the graph being searched. Held private to reduce coupling and maintain encapsulation. Maintained in BreadthFirstPaths(), bfs().

Access Programs

BreadthFirstPaths(GrantGraph: g, String: s):BreadthFirstPaths

Initializes the marked, edgeTo HashMaps, and s and g. Calls bfs(s).

bfs(String:a)

Marks the node a as visited, and adds it to the queue.

While the queue is not empty:

- It will retrieve the top of the queue by called queue.removeFirst().
- Loops through all the connections that v has by looping through g.getconn(v).keySet()
- If the the marked hashMap does not contain information about it, initialize it with a false value.
- if the node has not been visited yet, add it to the edgeTo hashmap from connection to the top of the queue called earlier with removeFirst(). Additionally set the value of its marked HashMap to true. Finally add the connection to the queue.

hasPathTo(String:v):boolean

Returns marked.get(v)

pathTo(String:v):Stack<String>

- If (hasPathTo(v) is false, return null)
- else loop through edgeTo hashmap starting at v, and follow the connections until s (starting point) is reached. Add the node attached along the path to the Stack
- Finally add s to the stack and return the stack

NOTE: the stack holds the reverse order of the path so you must loop through the stack and get the values via popping.

6.2 Controller

6.2.1 AlchemistController

Class AlchemistController.java

Package xb3_Final_Project.model

Variables

SERVER: boolean

Set to true if a server connection was found and validated.

DEBUG: boolean

Sets the toolset to debug mode that prints out program behaviour to console.

model: AlchemistModel

Used to access all of data maintenance and logic services.

gson: Gson

Used for bridging dataset and toolset, and dataset management.

Access Programs

AlchemistController()

Constructor for the Alchemist Controller. This constructor boots up a RESTful server, which acts as the API provider.

enableCORS(String origin, final String methods, final String headers): void

Modifies CORS rules. (Cross-Origin-Requests). Cross Origin Requests refer to requests that are outside of the server's domain. i.e. if the server is running on localhost, then requests that do not originate from localhost are either allowed or rejected, based on the CORS rules

7. Trace to Requirements

The table below lists the assignment's requirements, and the modules that fulfil those requirements.

7.1 Trace to Requirements

Table 3: Trace to Requirements

Requirements	Modules	How is it achieved
Functional Requirements		
0. The toolset must contain two visually distinct spaces of operation; a query page from which the user will be able to conduct a query and a data presentation/visualization page where the user will be able to see the result of a query. The user must be able to go back to the query page from the presentation/visualization page to conduct another query.	/client/www/js/app.js	The Ionic Framework creates tabs using states. Thus, states were defined, which created the views which are required.
1. The program must forecast the financial size of a valid field of research as categorized by the NSERC grant datasets and produce a forecasted financial size of a field for every year from 2015 to 2020.	/server/src Classes Series, GrantDatabase	The appropriate grants are pulled from the grant database. Then, time-series data is formed from this data which the OpenForecast library projects forward.

2. The program must forecast the number of grants awarded for a valid field of research as categorized by the NSERC grant datasets and produce a forecasted number of grants awarded to that field for every year ranging from 2015 to 2020.	/server/src Classes Series, GrantDatabase	The appropriate grants are pulled from the grant database. Then, time-series data is formed from this data which the OpenForecast library projects forward.
3. The program must present a visual depicting the result of requirements 1 and 2 in the form of a bar graph showing the change in the size of the field of research and number of graphs awarded from 2015 to 2020.	/server/src Classes Series, GrantDatabase	The appropriate grants are pulled from the grant database. Then, time-series data is formed from this data which the OpenForecast library projects forward.
4. The program must allow users to plot the total amount of grant dollars awarded to a field of research on a provincial distribution map of Canada for a given field of research as categorized by the NSERC grant datasets.	/server/src Classes Series, GrantDatabase	The appropriate grants are pulled from the grant database. Then, time-series data is formed from this data which the OpenForecast library projects forward.
5. The program must allow users to plot the total amount of grant dollars awarded to a field of research on a distribution bar graph listing of all of the most funded institutions for a given field of research as categorized by the NSERC grant datasets.	/server/src Classes Series, GrantDatabase	The appropriate grants are pulled from the grant database. Then, time-series data is formed from this data which the OpenForecast library projects forward.
6. The program must allow users to plot the total number of grants awarded to a field of research on a distribution bar graph listing of institutions for a given field of research as categorized by the NSERC grant datasets.	/client/www/templates/t ab-search-results.html	Using the Ionic Framework and and Highcharts.js, the appropriate graph is displayed for the given GrantColumn.
7. The program must allow users to plot the total amount of grant dollars awarded to a field of research on a distribution bar graph listing of all the most funded subjects within the field of research as categorized by the NSERC datasets.	/client/www/templates/t ab-search-results.html	Using the Ionic Framework and and Highcharts.js, the appropriate graph is displayed for the given GrantColumn.
8. The program must allow users to plot the total number of grants awarded to a field of research on a distribution bar graph listing of all the most awarded subjects within the	/client/www/templates/t ab-search-results.html	Using the Ionic Framework and and Highcharts.js, the appropriate graph is displayed for the given GrantColumn.

field of research as categorized by the NSERC datasets.		
9. The program must allow users to plot the total number of grants awarded to a field of research on a distribution bar graph listing of all of the Canadian provinces and territories for a valid field of research as categorized by the NSERC grant datasets.	/client/www/templates/t ab-search-results.html	Using the Ionic Framework and Highcharts.js, the appropriate graph is displayed for the given GrantColumn.
10. The program must allow users to plot the total number of grants awarded to a field of research on a distribution bar graph listing of all of the Canadian provinces and territories for a valid field of research as categorized by the NSERC grant datasets.	/client/www/templates/t ab-search-results.html	Using the Ionic Framework and Highcharts.js, the appropriate graph is displayed for the given GrantColumn.
11. The program must provide a visual distribution of all the entire financial history of a given field of research as categorized by the NSERC grant datasets.	/client/www/templates/t ab-search-results.html	Using the Ionic Framework and Highcharts.js, the map of Canada is displayed with the visual of the entire financial history for the given search.
12. The program must provide a visual distribution of all the entire history of the number of awarded grants to a given field of research as categorized by the NSERC grant datasets.	/client/www/templates/t ab-search-results.html	Using the Ionic Framework and Highcharts.js, the map of Canada is displayed with the visual of the entire financial history for the given search.
13. The program must list professors, alumni, or researchers who've received a grant whose subject of interest is within the valid field of research as categorized by the NSERC grant datasets.	/client/www/templates/t ab-search-results.html	Using the Ionic Framework, the details of grants are listed under search results, including what was required.
14. All queries made within the toolset should return a result within 1.5 seconds.	/client/www/js/services .js /client/www/templates/ search-results.html	Results from the server take less than 0.5 seconds. They are obtained via JSON from the server.
15. The toolset server must be able to handle a client load maximum of 100 queries per second.	/server/src Class AlchemistController	The Spark server is very efficient and can easily handle thousands of requests per second.
16. The toolset must be accessible from a modern browser running HTML5 and with Javascript enabled regardless of the operating system.	Ionic Framework	The Ionic Framework packages the app which can be deployed to iOS, Android and even run as a web app in browsers.

17. The user must be able to search the NSERC datasets for other than forecasting purposes. For example, the user should be able to look for the number of grants awarded to a specific professor, institution, subject or province, organization, or search for any professor to whom was granted an x amount of grant dollars.	/server/src Class GrantDatabase AlchemistController /client/www/js/services.js	The server exposes API which can be used to search the datasets generally. The Client uses this API, and displays appropriate results in the search.
18. The user must be able to select from a list of the following grant column options when performing a query: 1. Field of Research 2. Organization 3. Province 4. Professor 5. Subject, 6. Year awarded 7. Amount awarded	/client/www/templates/t ab-search.html	The client allows the user to search by selecting any of the grant columns and a specific keyword.
19. The query page must contain a “search for” button that returns the result of the query and displays it in the visualization page.	/client/www/templates/t ab-search.html	The client allows the user to search by selecting any of the grant columns and a specific keyword using a “search for” button. Then it transitions to a search results page.
20. The query page must show the number of results that were retrieved for a specific query.	/client/www/templates/t ab-search-results.html	The search results page displays the search results by counting the number of grants the query pulled.
21. The user must be able to download any bar graphs or maps presented in the visualization page as a .jpeg, .png, .pdf, .svg format directly within the toolset.	/client/www/templates/t ab-search-results.html	The Highcharts.js library allows users to download charts in many formats including those mentioned.
22. The user must be able to print any visual directly within the toolset.	/client/www/templates/t ab-search-results.html	There is a button to easily print the visuals in the results page.
23. The client side interface must be able to function within and be accessible through any mobile phone running Android 3.0 Honeycomb or later and iOS 7.0 or later.	Ionic Framework	The Ionic Framework deploys to many platforms including those mentioned. Moreover, all our architecture works with these devices.

24. The application must be accessible through the following web browsers; FireFox, Opera, Chrome, Safari, and Internet Explorer.	Ionic Framework	The Ionic Framework allows the app to be locally accessed via web browsers.
25. The application must not require the installation or any additional software downloads to function on the client side.	Ionic Framework	The Ionic Framework packages the app thus everything required is already inside the app.
Non-functional Requirements		
1. The toolset must be easy to use and pleasing to the eyes. The colour scheme of the application must be universal.	/client/www/css/style.css	The app uses the default Ionic css theme, and it is very pleasant.
2. The toolset must present all graphs with multiple colours to allow users to be able to easily discern between the entries.	/client/www/templates/tab-search-results.html	The highcharts library allows us to colour the charts.
3. A search query must be easily conducted without the use of a user manual or a visual instruction.	/client/www/templates/tab-search.html	The search interface is very intuitive and straightforward.
4. A search query must be able to predict what information the user is looking for and suggest visualizations for the search query automatically.	/client/www/templates/tab-search-results.html	The search results display appropriate graphs according to the search query.
5. The app must show all results within a single web page or scrollable frame.	/client/www/index.html	The index.html is the main frame of the app, while all other templates are loaded inside of it.
6. The frame on which data is presented must be scrollable via a mouse connected to a desktop or through a touch screen device.	/client/www/index.html	The main frame is scrollable by touch and the mouse.
7. The user must be able to see the online or offline status of server.	/client/www/templates/tab-server.html	The status tab shows the server status.
8. The user must be able to hover through the bars in the bar graph and see a numerical representation of its magnitude in percent and respective unit.	/client/www/templates/tab-search-results.html	The Highcharts.js library allows us to display graphs in an interactive manner showing various information about each portion of the graph.
9. The transition from the query to the data visualization page must be smooth and visually appealing.	Ionic Framework	Transitions between tabs is handled by the framework, and it is indeed smooth and appealing.

10. The application must be advertisement free.	*	There are no advertisement messages.
---	---	--------------------------------------

7.2 Trace to Requirements as it pertains to Implementation and Testing

Format: **Requirement**
 Implementation
 Testing

The user should be able to search for a grant by any field

This requirement is fulfilled at several stages in the program. First the Grant ADT is designed to contain a variable for every field, making sure that each field is always available for each grant. For every field or “column” in the Grant ADT, there exists a comparator that can evaluate whether or not the field in the grant is the same as the query. Finally, a breadth-first search(BFS) was implemented to find the result the user is looking for.

Testing for this requirement is done via the GUI manually. The user entered queries known to exist in the database for various fields and obtained results matching their query.

The search result should be sorted in some tangible manner

This requirement is fulfilled by the TimSort implementation in the program. The BFS is first used to generate the list of top N results and the TimSort is applied to the results to shown them ordered by the specified field.

Testing for this requirement is done via JUnit and the GUI manually. For the JUnit, a random integer array was sorted by the TimSort algorithm to ensure algorithmic correctness. For the GUI test, the user entered queries known to generate multiple results and verified that the results are sorted.

The results of the user’s query should be displayed in an easy to understand format

For this requirement, an external library Ionic Framework was used to process the raw results of the user’s query into a more visual form. Unlike the query itself, the aspect of representing the results in a visual form (map, graph...etc) is done on the client end of the application in order to ensure a degree of user interactivity.

The testing for this requirement was done through a user on the GUI manually. The test simply verified that the visual elements are indeed working. The correctness of the displays were roughly checked, but not exhaustively due to time constraints.

The system should use a graph to optimize query processing

This requirement is fulfilled by the GrantGraph module which organizes the grants in the database into a tree like structure to narrow down the range of a user’s query. Specifically, the tree narrows down the scope of searching through the database via categorizing the grants by fields which helps reduce the number of elements the searching algorithm has to crawl through.

Testing for this requirement is done in detail in the TestGrantGraph JUnit test module.

The program should be able to predict within an accepted margin of error

This requirement is implemented through an external library OpenForecast which uses statistical models as well as other prediction algorithms to approximate future performance. This function is made accessible to the user through GUI interactions.

Testing for this requirement is done through manual testing by using a segment of historical data to predict grant amounts for recent years. Several considerations had to be made to allow for this testing to be reasonable within the constraints of our program:

- The program should not be asked to predict grants near or at events that significantly disrupts economic stability. This is because our algorithm cannot predict shift in the global market.
- The program should not use historical data from too far back (i.e. several decades ago). This is to prevent the non-linear inflation of currency from becoming a significant factor in our predictions.
- The results may require manual verification due to market inflation/deflation or failure to avoid scenarios described in the first bullet point.

9. Internal Review and Analysis of Design

Using this decomposition gives the architecture the flexibility to produce optimal results, whilst keeping maintenance cost low. This is due to the fact that the classes within the modules ensure that they hide information, and swapping modules out with others will not affect the rest of the software. Other than ensuring modularity, the UML diagram in further sections shows that it is decoupled as well. In fact, there are no cycles in the class hierarchy and it is relatively easy to navigate from one class to another. Thus, the benefit of such a decomposition is that it promotes performance, lowers maintenance costs, and allows a developer to quickly pick up on the intuition behind the design. In the following subsections, the pros and cons of the design choices are discussed, and anticipated changes are listed.

9.1 Anticipated Change

- AC1: Modifying the database files in anyway, including headers, file names, data, etc.
Affected Classes: GrantDatabase
- AC2: Switching from Server-client model to a networkless model. This is easily achieved by packaging the server and client software together; however this will result in loss of portability.
Affected Classes: AlchemistController
- AC3: Adding different graphs and visualizations is easily done by modifying appropriate templates in the client.
- AC4: Changing the supported devices. Affected Classes: None; only the API consumer needs to be changed.

9.2 Pros of Design

- Server Client Architecture
 - A centralized server means centralized data; when data needs to be updated or changed, clients do not.
 - Clients do not need to be high-end devices; the server does the heavy lifting of processing and manipulating the large dataset.
 - Only the server needs to be upgraded (in terms of RAM, processing power, etc).

- A client for all platforms can be made simply by consuming the api using a compatible framework.
- Can easily be deployed as a package, thus alleviating the requirement of hosting the server.
- MVC Design Pattern
 - Decouples logic and data from view, lowering maintenance costs.
 - Relatively easier to test.
 - Separation of concerns.
 - Information hiding.
 - Less coupling.
 - Code-reusability.

9.3 Cons of Design

- Server Client Architecture
 - Server needs to be hosted somewhere, which means there is a higher maintenance cost. However, this can be overlooked as we can provide the end-user with both the server and client, thus it becomes the user's problem.
- Ionic Framework
 - Lower performance compared to creating a native application for iOS/Android.
 - Large overhead.

10. Test Plan/Design

10.1 Whitebox Testing

Type: Whitebox- JUnit Module Testing

Module Tested: GrantGraph

Test: getYearGrants
Method: getYearGrants(Int:year):double
Description: This method was one of the methods tested as it was one of the most important methods that the GrantGraph class had. While this gave a specific function/result from the graph, testing this method would essentially test if the first “year” layer of the tree was functional. As well as testing if the summation of the grants for the edge weights were being properly computed
Case 1: Legal input <pre> For (i =2000 ; i<2011; i++) Expected output = getYearGrants(i) </pre> Case Description: Tested 10 different years in the database (2000-2010)

Margin of Error: 5%

Expected Output:

451159747,43411630,7417146970,477141549,526133116,537934460,554668585,614020200,695
509580,763406550,809510608

Actual Output:

451159747,434116337,417146970,477141539,526133116,537934451,554668568,614020215,695
509585,763406534,809510608

Grade: PASS

Case 2: Edge Case

```
getYearGrants(1920)
```

Case Description: Tested a year that is not included in the database (1920)

Margin of Error: 0%

Expected Output:

0

Actual Output:

0

Grade: PASS

Test: getProvinceGrants

Method: getProvinceGrants(String:province, Int:year):double

Description: This method was one of the methods tested as it too was one of the most important methods. It was used for the visual aspect of the map of Canada showing the amount of grants for each country. Again, testing this method would essentially test if the first “year” layer of the tree was functional. As well as testing if the summation of the grants for the edge weights were being properly computed

Case 1: Legal input

```
For (String e in
    {"Ontario", "Quebec", "Alberta", "Manitoba", "British
    Columbia", "Nova Scotia"})
    Expected output = getProvinceGrants(e, 2014);
```

Case Description: Tested 6 different provinces for the year 2014

Margin of Error: 5%

Expected Output:

266318650,153739004,64879675,15013923,102110100,18845821

Actual Output:

266318645,153739008,64879685,15013926,102110102,18845819

Grade: PASS

Case 2: Edge Case

```
getProvinceGrants("NFL", 2001)
```

Case Description: Tested a province's shorthand version (Newfoundland) in 2001

Margin of Error: 0%

Expected Output:

null

Actual Output:

null

Grade: PASS

10.2 Black Box Testing

10.2.1 Functionality Testing

Grant ADT Functionality Test:

Method Tested: Grant.get

This test evaluates the performance of the get method across all fields of the grant ADT. A Grant object was constructed during the setup phase of the JUnit test module with generic information passed into the fields. The get method is then called upon each "column" or field of the Grant object and an assertion is made to ensure the correct information is indeed retrieved. Since the get method returns a generic Object, a Try-Catch block was also used to ensure that there was no errors in casting the objects back to their original types. Doing this also verifies that the type returned by the get method is indeed compatible to the original type intended for the field.

Comparators Functionality Test:

Methods/Modules Tested: Comparators for each field of the Grant ADT

This test evaluates the accuracy of the Comparators in terms of yielding true results. In the setup phase of the JUnit module, two Grant objects were created. The first instance of Grant was the object used in the ADT Functionality Test. The second Grant object was created such that every field of this Grant would be lesser (whether alphabetically or numerically) than the corresponding field of the first Grant. For each comparator, a total of three tests were performed:

1. The comparator should return $\text{Grant1} > \text{Grant2}$
2. The comparator should return $\text{Grant1} = \text{Grant1}$
3. The comparator should return $\text{Grant2} < \text{Grant1}$

Only two Grant objects were used for these comparisons because statement 1 above implies statement 3, and statement 2 is obvious. Thus, if the comparator asserts statement 1 to be true, it MUST assert the other two statements to be true as well for it to be considered a functional comparator.

TimSort Functionality Test:

Methods/Modules Tested: TimSort.sort

This test evaluates the correctness of the implementation of TimSort in our program. During the setup phase, a random list of random integers ranging from 0-1000000 of length 30-130 was generated. An Integer comparator was used in conjunction with this list as the parameters for the sort method. The test involved asserting that all i-th element in the resulting list was lesser than or equal to the next (i+1) element. This ensures that the sorting algorithm does in fact perform the sorting correctly and according to the order provided by the comparator.

Test: testGrant()
Input: new Grant(1234, "Science", "McMaster", "John Doe", "Ontario", "Algorithm Design",2016, 35000);
Expected Output: all fields match input when get is called on the corresponding column
Result: all assertion statements passed. (output from get method == input)

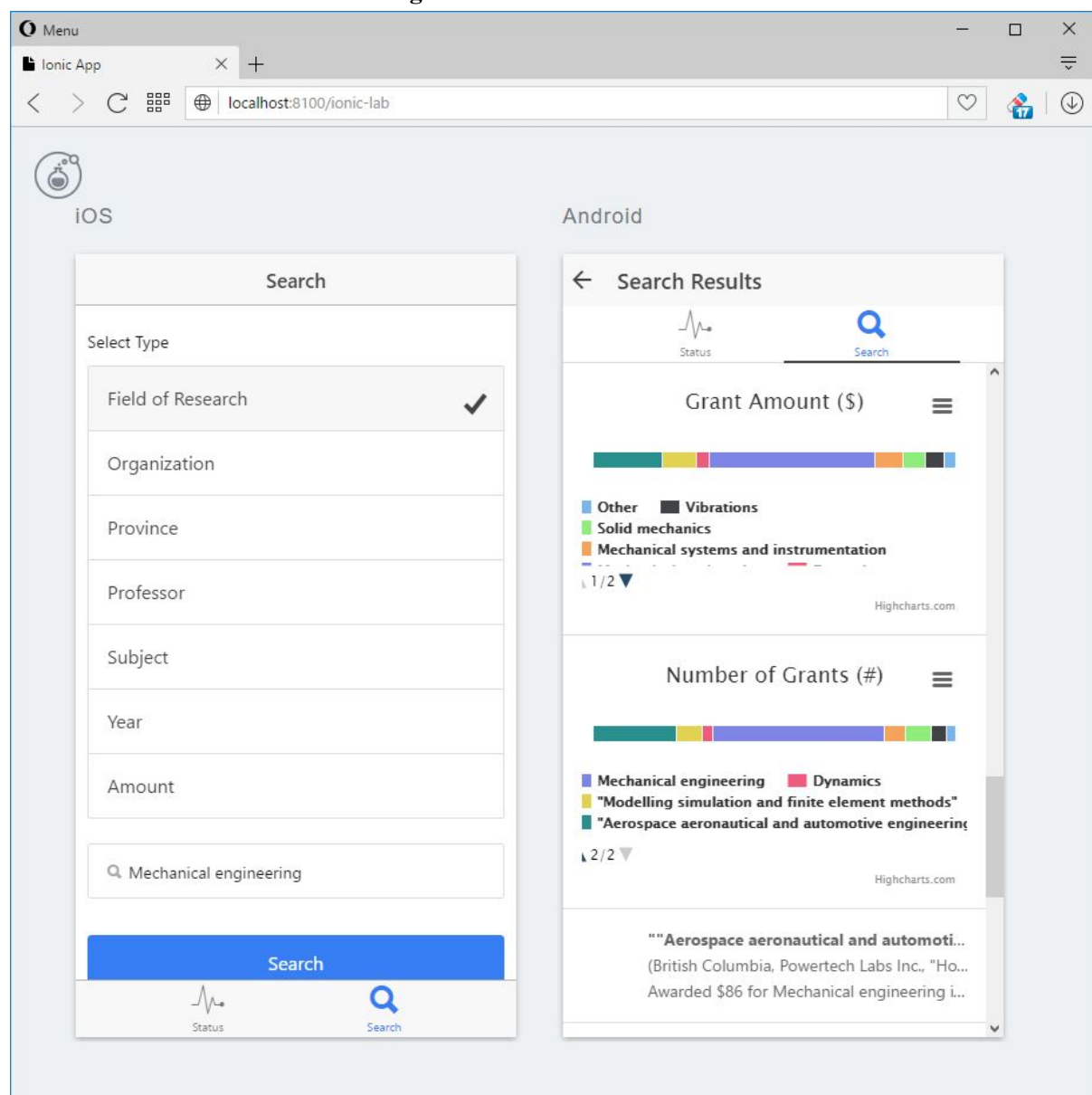
Test: testComparators()
Input: Grant grant = new Grant(1234, "Science", "McMaster", "John Doe", "Ontario", "Algorithm Design",2016, 35000); Grant grant2 = new Grant(1233, "AScience", "AMcMaster", "AJohn Doe", "AOntario", "AAlgorithm Design",2015, 34000);
Expected Output: for all fields the following must hold: <ol style="list-style-type: none"> 1. grant > grant2 2. grant = grant 3. grant2 < grant
Result: all assertion statements passed. (the three statements hold across all fields)

Test: testTimSort()
Input: Integer array of random length 30-130, with random elements from 0-1000000 Comparator 1 which returns -1 for lesser than, 0 for equal and 1 for greater than Comparator 2 which returns 1 for lesser than, 0 for equal and -1 for greater than
Expected Output: For the result of comparator 1: All i-th element should be lesser than the next element For the result of comparator 2: All i-th element should be greater than the next element
Result: all assertion statements passed. (TimSort successfully sorts according to the order dictated by the comparator passed into the method)

11. Conclusion

The Alchemist toolset is an iOS, Android, and web browser enabled application. It's primary use is to determine approximately the financial size of field of research in Canada through the use of NSERC datasets. NSERC is the largest financier of research in Canada and more than half of their grants are co-funded by private entities. Aside from providing forecasting services, the Alchemist toolset also provides searching and data visualization services that allow users to interact and view data more critically and easily.

Figure 10: Alchemist Toolset



This application is designed to describe the relationship between research grant awards and the specific field of research to which the grants were awarded. The application analyzes the

relationship between the number of grants awarded in a field of study over time. By comparing trends of the fields of research and grants awarded to these fields of research over time, the application produces a projected investment amount specific to that field of research that the user is interested in. The overall goal of this application is to reduce the financial waste incurred through poor investment decisions and inform investors of the potential strengths of the field of which they are interested in investing in as well as provide the students, faculty, and staff an overview of the metrics present within the NSERC grant datasets.

12. Appendix

12.1 List of Figures

Figure 1: Alchemist running on Ionic Lab iOS and Android Simulator.....	9
Figure 2: Server-Client Module Overview.....	10
Figure 3: Server-Client Composition.....	10
Figure 4: State Diagram.....	12
Figure 5: Graph Searching Scheme.....	14
Figure 6: Uses Relationship.....	16
Figure 7: UML Module Diagram.....	17
Figure 8: UML State Diagram for GrantColumn.....	26
Figure 9: UML State diagram for GrantGraph.....	32
Figure 10: Alchemist Toolset.....	46

12.2 List of Tables

Table 1: State Descriptions.....	11
Table 2: Grant Columns are associated with a character.....	12
Table 3: Trace to Requirements.....	34

13. References

13.1 Libraries

13.1.1 Gson

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

There are a few open-source projects that can convert Java objects to JSON. However, most of them require that you place Java annotations in your classes; something that you can not do if you do not have access to the source-code. Most also do not fully support the use of Java Generics. Gson considers both of these as very important design goals.

Features

- Provide simple toJson() and fromJson() methods to convert Java objects to JSON and vice-versa
- Allow pre-existing unmodifiable objects to be converted to and from JSON
- Extensive support of Java Generics
- Allow custom representations for objects
- Support arbitrarily complex objects (with deep inheritance hierarchies and extensive use of generic types)

Authors

Google Inc. (2008) Gson (Version 2.0) [Library].

Download

Available at <https://github.com/google/gson> (Accessed March 20th 2016)

13.1.2 Spark

Spark Framework is a simple and lightweight Java web framework built for rapid development. Spark's intention isn't to compete with Sinatra, or the dozen of similar web frameworks in different languages, but to provide a pure Java alternative for developers that want to (or are required to), develop their web application in Java. Spark is built around Java 8's lambda philosophy, which makes a typical Spark application a lot less verbose than most application written in other Java web frameworks.

Spark focuses on being as simple and straight-forward as possible, without the need for cumbersome (XML) configuration, to enable very fast web application development in pure Java with minimal effort. It's a totally different paradigm when compared to the overuse of annotations for accomplishing pretty trivial stuff seen in other web frameworks.

Features

- Built for productivity
- Allows for the direct use of the Java Virtual Machine
- Many microservices available

Authors

Spark (2016) MLlib (Version 1.6.1) [Library].

Download

Available at <http://spark.apache.org/mllib/> (Accessed March 26th 2016)

13.1.3 OpenForecast

OpenForecast is a package of general purpose, forecasting models written in Java that can be applied to any data series. One of the design goals was/is to make it easy for a developer to use in an application even if they do not understand, or care to understand, the differences between the different forecasting models available.

Features

- Can create models using a variety of methods:
 - single variable linear regression
 - single variable polynomial regression
 - multivariable linear regression
 - moving average
- Accuracy increased with large dataset

Authors

Gould, S. ,(2002-2011) *OpenForecast* (Version 0.4) [Computer Program].

Download

Available at <http://openforecast.sourceforge.net/docs/> (Accessed March 13th 2016)

13.1.4 Ionic Framework

Ionic is a powerful HTML5 SDK that helps you build native-feeling mobile apps using web technologies like HTML, CSS, and Javascript. The ionic framework focuses on look and feel, and UI interaction of the Alchemist toolset. It is not a replacement for a Javascript framework, instead it simplifies a large portion of the front end of the toolset. The framework requires AngularJS for UI interaction, gestures, animations and other visual features.

Features

- Simplifies front end javascript code.
- Provides interfacing tools and look and feel for the web applet.

Authors

The ionic framework was built by the makers of Codiqa and Jetstrap

Download

Available at <http://ionicframework.com/getting-started/>

13.1.4 Highcharts.js

Javascript library which creates interactive graphs.

Features

- Displays time-series data in a visual way.
- Draws interactive maps.

Authors

Highsoft AS

Download

Available at <http://www.highcharts.com/docs/getting-started/installation/>

13.2 Books

1. Sedgewick, Robert. *Algorithms*. Reading, MA: Addison-Wesley, 1988. Print.

13.3 Websites

1. Riedl, B. M. (n.d.). Top 10 Examples of Government Waste. Retrieved March 18, 2016, from <http://www.heritage.org/research/reports/2005/04/top-10-examples-of-government-waste>
2. Broad, W. J. (2014). Billionaires With Big Ideas Are Privatizing American Science. Retrieved March 19, 2016, from http://www.nytimes.com/2014/03/16/science/billionaires-with-big-ideas-are-privatizing-american-science.html?_r=0

13.4 Code/Modules

13.4.1 TimSort.java

The Model uses the services of TimSort.java to initially sort the data set inputs before forming the graph structure for the server.

Download:

http://cr.openjdk.java.net/~martin/webrevs/openjdk7/timsort/raw_files/new/src/share/classes/java/util/TimSort.java

14. Project Log

Team Name: Team Alchemist

Team Topic: Forecasting and Data Visualization

Team Members:

Name:	Roles:
Danish Khan: 1217176	Team Leader, Designer, Implementer, Documenter
Haris Khan: 1413511	Software Architect, Designer Implementer, Documenter
Michael Bitzos: 1405050	Log Administrator, Implementer, Tester, Documenter
Junhao Wang: 1215428	Implementer, Tester, Documenter

14.1 List of Task IDs

- 1.0 Database selection
- 2.0.0 Project name & brand selection
- 3.0 Project Timeline
- 3.1 Group Roles
- 4.0.0 Requirements and specification
- 4.0.1 Introduction to design specification document
- 5.0 BitBucket
- 6.0 Database extraction
- 7.0 ADT
- 8.0 Predicting
- 9.0 Graphing Implementation
- 9.1 Geolocations
- 10.0 Presentation
- 11.0 Referencing
- 1.0.0 Design Specification Document
- 12.0 Requirements Specification
- 13.0 Labeling Tables/Figures

14.2 Project Log

Timestamp	Originator	Type	Task ID	Status	Comments	Supporting Documents
160129T1600	All	Decision	1.0	Completed	Chosen database for final project.	2XB3_FinalProject MeetingTemplates.docx - Jan 29th
160205T1530	All	Decision	2.0	Not-Completed	Came up with project name	Meeting minutes 2.pdf
160222T1555	All	Decision	3.0	Completed	Divided up the work of the project	2XB3_FinalProject Templates.docx-Feb 22
160304T1600	Danish/Michael	Decision	3.1	Not-Completed	Assigned Group Roles (2/4)	Meeting Minutes-Mar 4.docx
160305T1400	All	Task	4.0.0	WIP	Requirements Specifications document	Meeting Minutes Mar 5.docx
160305T1400	All	Decision	2.1.0	Completed	Made new team name	Meeting minutes Mar 5
160306T2005	Danish	Decision	2.1.1	Completed	Completed Project App Icon	Requirements Specification.pdf
160306T2005	Danish	Task	4.0.1	Completed	Completed design document introduction	Requirements Specification.pdf
160306T2005	Danish	Task	4.0.2	Completed	Completed Requirement specification revision and submission.	Requirements Specification.pdf
160305T1400	Haris	Task	5.0	Completed	Created BitBucket for group project, added everyone to the team, and set up repo.	Meeting Minutes Mar 5.docx
160311T1500	Haris	Task	6.0	Completed	Extracted Database from website	Meeting Minutes Mar 11.docx
160311T1500	Danish	Task	3.2	Completed	Made project Timeline	2XB3 Final Project Bulletin.pdf
160306T1200	Junhao	Task	7.0	Completed	Developed ADT	Meeting Minutes Mar 13.docx
160313T1340	Haris & Danish	Decision	8.0	Completed	Found predicting library	Meeting Minutes Mar 13.docx

160314T1820	Danish	Task	1.0.0	Completed	Created design specification document in Google Docs.	Design Specification Document.pdf
160314T1821	Danish	Task	1.0.0	Completed	Added proposed an App Logo Design to Specification document.	Design Specification Document.pdf
160314T1321	Danish	Task	1.0.0	Completed	Title page, cover page, application icon, introduction, revision history page, contribution page and modular decomposition.	Design Specification Document.pdf
160330T2200	Haris/Michael	Idea	9.0	Complete	Thought of graph implementation	Meeting minutes Mar 30.docx
160330T2200	Danish	Task	1.0.1	Work in Progress	Started MIS/MID Documentation.	Design Specification Document
160331T1900	Michael	Task	9.1	Complete	Found geolocations of the institutes/organizations	Meeting minutes Apr 2.docx
160331T1900	Michael	Task	9.0	Complete	Made data into graph	Meeting minutes Apr 2.docx
160403T1400	ALL	Task	10.0	Complete	Finished Presentation Slides	Meeting Minutes Apr 3.docx
160406T2100	Michael	Task	11.0	Complete	References / Cited all external libraries used	
160410T1100	Michael	Task	1.0.1	Complete	MIS/MID for graph	Final Design Specification Document.pdf
160410T1420	Michael	Task	1.0.6	Complete	Completed Testing documentations	Final Design Specification Document.pdf
160410T1007	Danish	Task	1.0.1	Complete	Completed MIS/MID for Model	Final Design Specification Document.pdf
160410T1007	Danish	Task	1.0.1	Complete	Completed MIS/MID for Controller	Final Design Specification Document.pdf

160410T1650	Danish	Task	1.0.1	Complete	Completed Design Specification format.	Final Design Specification Document.pdf
160410T1650	Danish	Task	1.0.2	Complete	Added Executive Summary	Final Design Specification Document.pdf
160410T1650	Danish	Task	1.0.3	Complete	Added References section	Final Design Specification Document.pdf
160410T1650	Danish	Task	1.0.3	Complete	Added Project log	Final Design Specification Document.pdf
160410T1650	Danish	Task	1.0.3	Complete	Added Citations for online resources	Final Design Specification Document.pdf
160410T1650	Danish	Task	1.0.3	Complete	Added library overviews and features.	Final Design Specification Document.pdf
160410T180	Danish	Edit	1.0.3	Complete	Revised the introduction to include the three basic functions of the app.	Final Design Specification Document.pdf
160410T180	Danish	Edit	1.0.3	Complete	Revised the introduction to include the three basic functions of the app.	Final Design Specification Document.pdf
160410T180	Danish	Edit	1.0.4	Complete	Edited Executive Summary.	Final Design Specification Document.pdf
160410T1080	Haris & Michael	Task	1.0.5	Complete	Added sections 2.1 Overview, 2.2 Client, 2.2.1 Model, 2.2.1.1 Data Structures, 2.2.1.2 Searching & Sorting, 2.2.1.3 Algorithmic Challenges, 2.2.2 View, and 2.2.3 Controller and 2.3 Server.	Final Design Specification Document.pdf
160410T180	Danish	Task	1.1.0	Complete	Added Trace to requirements functional and nonfunctional requirements.	Final Design Specification Document.pdf

160411T0930	Danish	Task	12.0	Addition	Added more requirements to requirements specification.	Final Requirements Specification Document.pdf
160410T0930	Danish	Task	13.0	Addition	Labeled all tables and figures.	Requirements Specification Document.pdf
160410T0930	Danish	Task	13.1	Addition	<ul style="list-style-type: none"> - Added application icon, edited title page to meet requirements. Added precise list of functional requirements - Added precise list of non-functional requirements - Edited introduction to include domain, stakeholders, knowledge of domain that informs the requirements. 	Final Requirements Specification Document.pdf
160413T1900	ALL	Task	13.2	Addition	Finished Final Requirements Specifications Documentation	Final Requirements Specifications Document.pdf