

Design Specifications

YouRecommend

Version 1

April 12, 2020

Group Number: 02

Lab Section: L01

Course: SE 2XB3

McMaster University - Department of Computing and Software

Team Members	Student Emails
Kabishan Suvendran	suvendrk@mcmaster.ca
Franklin Tian	tiany38@mcmaster.ca
Jiawei Yu	yuj130@mcmaster.ca
Bowen Zhang	zhangb82@mcmaster.ca


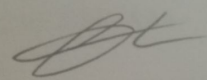

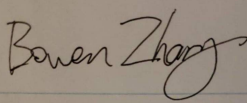
Revision Page

Revision History

Initially, the group wanted to create a web application that would suggest YouTube channels based on the user's channel preference. While this goal was achieved, there were some aspects of the project that were overly ambitious. For example, displaying the suggested YouTube channels using a visual representation. For example, a bar-graph could be used to display the top ten suggested YouTube channels. This, however, ended up being computationally demanding. Instead, implementing a geographical suggestion option, which would suggest other channels of the same geographical location and channel genre as the top YouTube channel for each category, seemed to be a more feasible option. For example, for the 'Gaming' category, if you select to get suggestions based on followers, the top YouTube channel would be VanossGaming, who is from Canada. Now with the geographical suggestion option enabled, the user would receive YouTube 'Gaming' channel suggestions from Canada.

Attestation and Consent

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

Team Members	Student Numbers	Roles	e-signature
Kabishan Suvendran	400196622	Front-End Developer	
Franklin Tian	400171067	Back-End Developer	
Jiawei Yu	400152646	Back-End Developer	
Bowen Zhang	400168985	Back-End Developer	

Contributions

Name	Role	Contributions	Comments
Kabishan Suvendran	Front-End Developer	Constructed the front-end aspect of the project by self-learning Apache Tomcat, Java Dynamic Web Applications, Servlets and Java Enterprise Edition.	
Franklin Tian	Back-End Developer	Developed the back-end implementation of reading dataset class (readCSV), search class, some parts of Main class back-end API.	
Jiawei Yu	Back-End Developer	Wrote the implementation of YouTuber class, graph part, depth-first search, and some parts of Main class.	
Bowen Zhang	Back-End Developer	Implement the Sort Module, some parts of the Main Module and the TestMain JUnit tests.	

Table of Contents

Executive Summary	6
Description of Classes/Modules	6
Reason Behind the Classes/Modules	8
Application UML diagram	9
Interface (Public Entities), Semantics and Syntax Specifications	10
Implementation (Private Entities)	33
UML State Machine Diagrams	37
Evaluation/Review	38

Executive Summary

When using YouTube nowadays, controversies commenced as many individuals are recommended with undesired videos and channels. This causes many clients with unnecessary content popping up on their feed which brought inconvenience. The solution is a web application that provides recommended YouTube channels by directly asking users for their intended channel preferences. This solution will filter out all of the channels that are irrelevant, which is determined by a category identifier. By choosing the most efficient algorithms, this problem could be solved effectively and efficiently. The database of channels consists of more than 100,000 entries to support the product. Every piece of data indicates a channel and the statistics of each channel will be used to conduct algorithmic manipulations that will solve the complication. The testing will be conducted on smaller portions of the dataset within the database for efficiency. By comparing the recommendations made by the application with YouTube's recommendations, the accuracy could be determined and unique features of the product will be revealed.

Description of Classes/Modules

Read File Module

The Read File module is of top priority when initiating the project, as almost all of the other modules use the reading dataset function. The Read module builds the foundation for starting this project as it achieves the function of being able to iterate through the entire dataset. Other modules such as Graph must be able to store the data read from the Read File module into a graph data structure of its own.

Search Module

The Search module will use the Read File module to extract the entire dataset. This module's main purpose is to provide the client with the feature of looking for a YouTube category as a string value and providing all the recommended channels related to this YouTube category.

Sort Module

The Sort module takes an arraylist of YouTubers as input, and converts the input to a sorted arraylist based on the followers or video uploads. If the input arraylist consists of over a thousand elements, quicksort will be used to minimize the processing time. Otherwise, mergesort will be applied because mergesort performs better when there are fewer elements in the arraylist.

Graph Module

The Graph module creates a graph that connects YouTubers who are under the same video category. The module uses an undirected graph and the goal is to ease the implementation of the Search module and make the module run faster.

YouTuber Module

The YouTuber module is integral, as all other modules are based on the YouTuber module. This module is used to create YouTuber objects to store data provided in the CSV dataset. This module can make the program more understandable, maintainable, and easier to implement.

Main Module

The Main module is very important in terms of implementing the program. This module acts as the back-end application programming interface that uses all of the search, sort and graph features to produce the recommended result.

FrontEndServlet

The FrontEndServlet module is used to display the generated list of YouTubers based on what users search on the screen. It is necessary for the front-end to use this module. The module is the connection between the user interface and the underlying implementations.

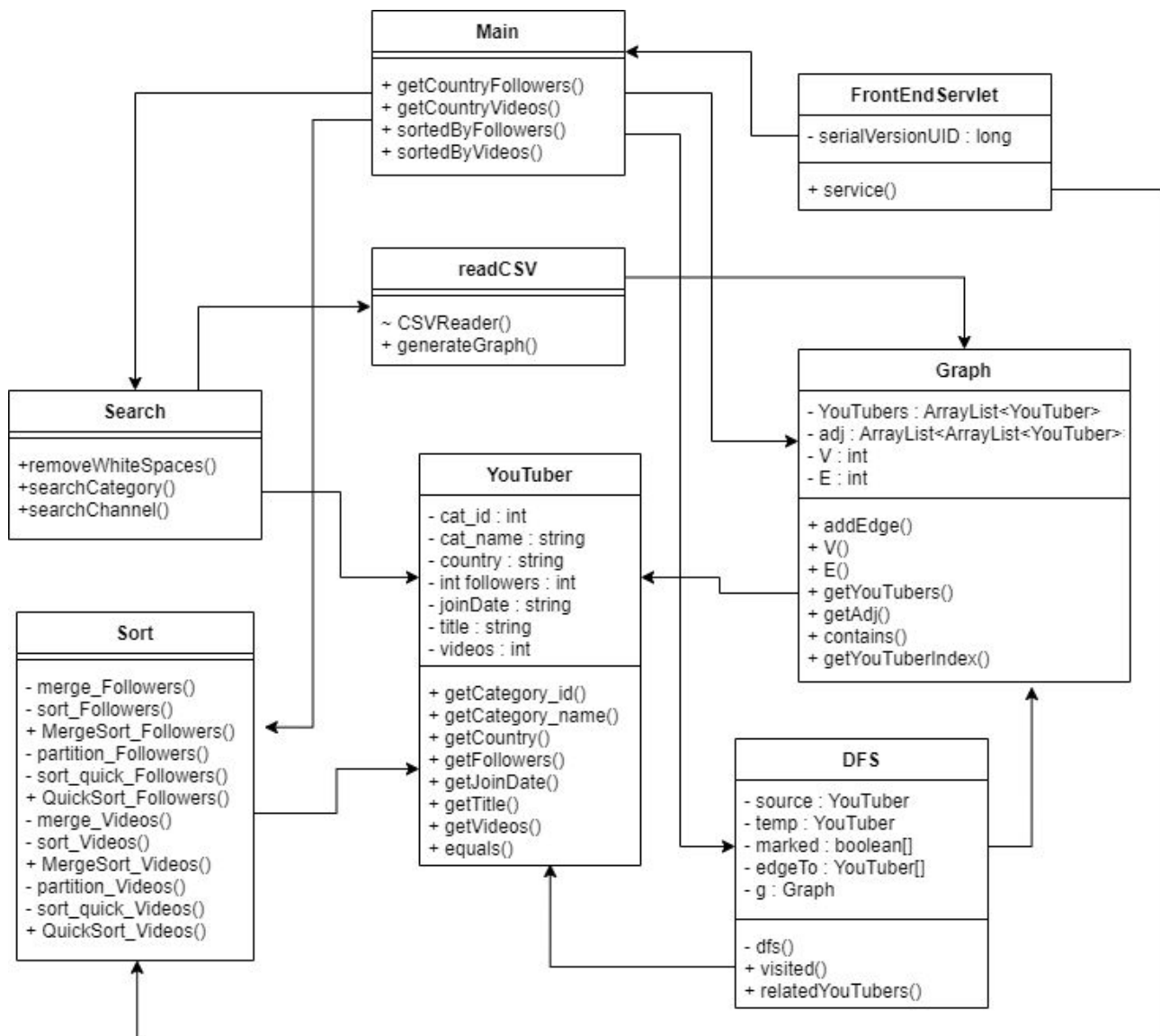
Reason Behind the Classes/Modules

To begin with, since the main requirement for the project is to utilize a dataset containing over 100,000 data entries and to manipulate the data, it is required to develop a class, `readCSV`, that takes in the data and stores them in an organized manner. Which is a 2D array. Since each channel has various attributes, `YouTube` class was designed as an abstract data type to store each channel and their corresponding information so other classes could perform operations on it.

Another requirement of this project is to use a searching algorithm, sorting algorithm and a graph data structure. Hence, the classes `Search`, `Sort`, `Graph` along with `DFS` was built. The main feature of `YouRecommend` is to take in a category and provide specific channel recommendations with the option of choosing from a specific country. Either displayed by the amount of the channel's subscribers or their video count. To accomplish these tasks, advantages of the project requirements were taken and this is where the algorithms behind the `Search`, `Sort` and `Graph` classes have taken place.

Next, the decision was to create a Main back-end API to combine the algorithms and provide a more straightforward interface to connect to the `FrontEndServlet` class. Finally, a web user interface is constructed for the client and it requires a way to communicate to the back-end APIs. This is why `FrontEndServlet` must be constructed as it acts as the communicator or bridge between the front-end and the back-end.

Application UML diagram



Search Module

Module

Search

Uses

YouTuber, readCSV

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
removeWhiteSpaces	String	String	
searchCategory	String	sequence of YouTuber	
searchChannel	String	sequence of YouTuber	

Semantics

State Variables

None

State Invariant

None

Access Routine Semantics

removeWhiteSpaces(input)

- output: $\text{out} \models \text{input.replaceAll}(<\textit{WhiteSpaces}>)$
- exception: none

searchCategory(category)

- transition: $\forall i : N | i \in [1..|a| - 1] : (\text{a.get}(i).\text{getCategory_name()} == \text{category}) \Rightarrow \text{channelInfo.add}(\text{a.get}(i))$
- output: $\text{out} \models \text{channelInfo}$
- exception: none

searchChannel(channel)

- transition: $\forall i : N | i \in [1..|a| - 1] : (\text{a.get}(i).\text{getChannel_name()} == \text{channel}) \Rightarrow \text{channelInfo.add}(\text{a.get}(i))$
- output: $\text{out} \models \text{channelInfo}$
- exception: none

Graph Module

Template Module

Graph(N)

Uses

YouTuber

Syntax

Exported Constants

None

Exported Types

Graph = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Graph	N, sequence of YouTuber	Graph	
V		N	
E		N	
addEdge	YouTuber, YouTuber		

getYouTubers		Seq of YouTuber	
getAdj		Seq of seq of YouTuber	
contains		\mathbb{B}	

Semantics

State Variables

V: N

E: N

YouTubers: sequence of YouTubers

adj: sequence of sequences of YouTuber

State Invariant

None

Assumptions

The constructor Graph should be called before calling any other methods.

Access Routine Semantics

new Graph()

- transition: $V, E := 0, 0$

- output: $\text{out} \models \text{self}$
- exception: none

$V()$

- output: $\text{out} \models V$
- exception: none

$E()$

- output: $\text{out} \models E$
- exception: none

$\text{addEdge}(v1, v2)$

- transition: $(v1, v2 : \text{YouTuber} \mid \neg \text{contains}(v1) \Rightarrow \text{YouTubers.add}(v1) : \neg \text{contains}(v2) \Rightarrow \text{YouTubers.add}(v2) : \text{adj.get}(\text{index of } v1).\text{add}(v2) \wedge \text{adj.get}(\text{index of } v2).\text{add}(v1))$
- exception: none

$\text{getYouTubers}()$

- output: $\text{out} \models \text{YouTubers}$
- exception: none

$\text{getAdj}()$

- output: $\text{out} \models \text{adj}$
- exception: none

$\text{contains}(\text{YouTuber } u)$

- output: $\text{out} \models (u : \text{YouTuber} \mid u \in \text{YouTubers})$

- exception: none

`getYouTuberIndex(YouTuber u)`

- output: $\text{out} \models (i : \mathbb{N} \mid i \in [0..|\text{YouTubers}| - 1] : \text{YouTubers}[i] = u)$
- exception: none

YouTuber Module

Template Module

YouTuber

Uses

None

Syntax

Exported Constants

None

Exported Types

YouTuber = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new YouTuber	N, String, String, N, String, String, N	YouTuber	
getCategory_id		N	
getCategory_name		String	
getCountry		String	

getFollowers		N	
getJoinDate		String	
getTitle		String	
getVideos		N	

Semantics

State Variables

cat_id: N

cat_name: String

country: String

followers: N

join_date: String

title: String

videos: String

State Invariant

None

Assumptions

The constructor “YouTuber” should be called before calling any other methods.

Access Routine Semantics

new YouTuber(id, n, c, f, j, t, v)

- transition: cat_id, cat_name, country, followers, join_date, title, videos := id, n, c, f, j, t, v
- output: out := self
- exception: none

getCategory_id()

- output: out := cat_id
- exception: none

getCategory_name()

- output: out := cat_name
- exception: none

getCountry()

- output: out := country
- exception: none

getFollowers()

- output: out := followers
- exception: none

getJoinDate()

- output: out := join_date
- exception: none

getTitle()

- output: $\text{out} \models \text{title}$
- exception: none

getVideos()

- output: $\text{out} \models \text{videos}$
- exception: none

equals(YouTuber that)

- output: $\text{out} \models (\text{this.cat_id} = \text{that.cat_id} \wedge \text{this.cat_name} = \text{that.cat_name} \wedge \text{this.country} = \text{that.country} \wedge \text{this.followers} = \text{that.followers} \wedge \text{this.joinDate} = \text{that.joinDate} \wedge \text{this.title} = \text{that.title} \wedge \text{this.videos} = \text{that.videos})$
- exception: none

Read CSV File Module

Module

readCSV

Uses

YouTuber, Graph

Syntax

Exported Constants

None

Exported Types

Read = ?

Exported Access Programs

Routine name	In	Out	Exceptions
CSVReader	String	sequence of YouTuber	FileNotFoundException
generateGraph		Graph	FileNotFoundException

Semantics

State Variable

None

State Invariant

None

Environment Variables

filepath: the path of the dataset

Access Routine Semantics

CSVReader(filepath: String)

- output: $out \models$ Store the data in the csv file searched by filepath as a 2D array
- exception: $exc := \neg (filepath.exists()) \Rightarrow FileNotFoundException$

generateGraph()

- output: $out \models$ Graph such that vertices of the same countries connect to each other and it does not contain YouTubers without available country information.
- exception: $exc := \neg (filepath.exists()) \Rightarrow FileNotFoundException$

Sort Module

Module

Sort

Uses

YouTuber

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
MergeSort_Followers	sequence of YouTuber	sequence of YouTuber	
MergeSort_Videos	sequence of YouTuber	sequence of YouTuber	
QuickSort_Followers	sequence of YouTuber	sequence of YouTuber	
QuickSort_Videos	sequence of YouTuber	sequence of YouTuber	

Semantics

State Variables

None

State Invariant

None

Access Routine Semantics

MergeSort_Followers(arr)

- output: out \models sequence of YouTuber(sort_Followers(arr, 0, arr.size()-1))
- exception: none

MergeSort_Videos(arr)

- output: out \models sequence of YouTuber(sort_Videos(arr, 0, arr.size()-1))
- exception: none

QuickSort_Followers(arr)

- output: out \models sequence of YouTuber(sort_quick_Followers(arr, 0, arr.size()-1))
- exception: none

QuickSort_Videos(arr)

- output: out \models sequence of YouTuber(sort_quick_Videos(arr, 0, arr.size()-1))
- exception: none

Local Functions

merge_Followers: sequence of YouTuber \rightarrow sequence of YouTuber

$\text{merge_Followers}(\text{arr}, l, m, r) = \text{arr}[l..r]$ where $\forall i : N | i \in [l..r-1] :$
 $\text{arr}[i].\text{getFollowers}() \geq \text{arr}[i+1].\text{getFollowers}()$

sort_Followers: sequence of YouTuber \rightarrow sequence of YouTuber

$\text{sort_Followers}(\text{arr}, l, r) = \text{arr}[l..r]$ where $\forall i : N | i \in [l..r-1] :$
 $\text{arr}[i].\text{getFollowers}() \geq \text{arr}[i+1].\text{getFollowers}()$

merge_Videos: sequence of YouTuber \rightarrow sequence of YouTuber

$\text{merge_Videos}(\text{arr}, l, m, r) = \text{arr}[l..r]$ where $\forall i : N | i \in [l..r-1] :$
 $\text{arr}[i].\text{getVideos}() \geq \text{arr}[i+1].\text{getVideos}()$

sort_Videos: sequence of YouTuber \rightarrow sequence of YouTuber

$\text{sort_Videos}(\text{arr}, l, r) = \text{arr}[l..r]$ where $\forall i : N | i \in [l..r-1] :$ $\text{arr}[i].\text{getVideos}() \geq$
 $\text{arr}[i+1].\text{getVideos}()$

partition_Followers: sequence of YouTuber \rightarrow sequence of YouTuber

$\text{partition_Followers}(\text{arr}, \text{low}, \text{high}) = \text{arr}[\text{low}..\text{high}]$ where $(\forall i : N | i \in$
 $[\text{low}..\text{pivot}] : \text{arr}[i].\text{getFollowers}() \geq \text{arr}[\text{pivot}].\text{getFollowers}()) \wedge (\forall j : N | j \in$
 $[\text{pivot}+1..\text{high}] : \text{arr}[\text{pivot}].\text{getFollowers}() \geq \text{arr}[j].\text{getFollowers}())$

sort_quick_Followers: sequence of YouTuber \rightarrow sequence of YouTuber

$\text{sort_quick_Followers}(\text{arr}, \text{low}, \text{high}) = \text{arr}[\text{low}..\text{high}]$ where $\forall i : N | i \in$
 $[\text{low}..\text{high}-1] : \text{arr}[i].\text{getFollowers}() \geq \text{arr}[i+1].\text{getFollowers}()$

partition_Videos: sequence of YouTuber \rightarrow sequence of YouTuber

$\text{partition_Videos}(\text{arr}, \text{low}, \text{high}) = \text{arr}[\text{low}..\text{high}]$ where $(\forall i : N | i \in$
 $[\text{low}..\text{pivot}] : \text{arr}[i].\text{getVideos}() \geq \text{arr}[\text{pivot}].\text{getVideos}()) \wedge (\forall j : N | j \in$
 $[\text{pivot}+1..\text{high}] : \text{arr}[\text{pivot}].\text{getVideos}() \geq \text{arr}[j].\text{getVideos}())$

sort_quick_Videos: sequence of YouTuber \rightarrow sequence of YouTuber

$\text{sort_quick_Videos}(\text{arr}, \text{low}, \text{high}) = \text{arr}[\text{low}..\text{high}]$ where $\forall i : \mathbb{N} | i \in [\text{low}..\text{high}-1] : \text{arr}[i].\text{getVideos}() \geq \text{arr}[i+1].\text{getVideos}()$

Main Module

Module

Main

Uses

Graph, YouTuber, Search, Sort, DFS

Syntax

Exported Constants

None

Exported Access Program

Routine name	In	Out	Exception
getCountryFollowers	YouTuber	sequence of YouTuber	
getCountryVideos	YouTuber	sequence of YouTuber	
sortedByFollowers	String	sequence of YouTuber	
sortedByFollowers	String	sequence of YouTuber	

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

getCountryFollowers(source: YouTuber)

- output: out := Sort.MergeSort_Followers(DFS(readCSV.generateGraph(), source).relatedYouTubers())
- exception: none

getCountryVideos(source: YouTuber)

- output: out := Sort.QuickSort_Videos(DFS(readCSV.generateGraph(), source).relatedYouTubers())
- exception: none

sortedByFollowers(category: String)

- output: out := Sort.MergeSort_Followers(Search.searchCategory(category))
- exception: none

sortedByFollowers(category: String)

- output: out := Sort.QuickSort_Videos(Search.searchCategory(category))
- exception: none

Depth-First Search Module

Module

DFS

Uses

Graph, YouTuber

Syntax

Exported Constants

None

Exported Access Program

Routine name	In	Out	Exception
new DFS	Graph, YouTuber	DFS	
visited		\mathbb{B}	
relatedYouTubers		sequence of YouTuber	

Semantics

State Variables

source: YouTuber

temp: YouTuber

marked: sequence of \mathbb{B}

edgeTo: sequence of YouTuber

G: Graph

State Invariant

None

Access Routine Semantics

new DFS(g: Graph, source: YouTuber)

- transition: this.source, this.g := source, g
- output: out := self
- exception: none

visited(u: YouTuber)

- output: out := marked[g.getYoutuberIndex(u)]
- exception: none

relatedYouTubers()

- output: out := sequence of YouTubers such that the country and category of these YouTubers are the same
- exception: none

Local Functions

dfs: Graph \times YouTuber

dfs(G, source) \equiv marked[g.getYoTuberIndex(source)] := true \wedge (u: YouTuber
| u \in g.getAdj().get(g.getYoTuberIndex(source)): \neg
marked[g.getYoTuberIndex(u)] \Rightarrow edgeTo(g.getYoTuberIndex(u)) = source
 \wedge dfs(g,u))

FrontEndServlet Module

Module inherits HttpServlet

FrontEndServlet

Uses

Main, Sort, YouTuber

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
service	HttpServletRequest, HttpServletResponse		IOException

Environment Variables

screen: two dimensional sequence of positions on the screen, where each position holds a character

Semantics

Environment Variables

filepath: the path of the dataset

State Variables

serialVersionUID: Z

State Invariant

None

Access Routine Semantics

service(req: HttpServletRequest, res: HttpServletResponse)

- transition: The state of the screen is modified to first show a layout in which the user can select their search criteria. The user must specify their channel preference and whether they want to receive their suggestions sorted by followers or video uploads. It is optional for the user to select the geographical suggestion option. Once the user has made their request, the state of the screen is modified to show a table-like structure, where the left column is the name of the suggested YouTubers and the right column is the number of followers or video uploads.
- $\text{exc} := \neg (\text{filepath.exists()}) \Rightarrow \text{IOException}$

Implementation

YouTuber Class

In the YouTuber Class, there are seven private variables - cat_id, cat_name, country, followers, joinDate, title, and videos. These state variables are used to store the state of a YouTuber. It also has seven getters. These getters have access to the state information of a YouTuber. Besides, there is a method named “equals”. It is used to compare the current YouTuber with another YouTuber. It checks whether the values of the state variables of the two YouTubers are the same. It is very useful for comparing two YouTuber objects.

readCSV Class

To read the input file, BufferedReader is used to read the text from the file path. The most important method, CSVReader, manually sets the dataset array's size to the size based on the dataset. It will go through the dataset and while there still exist entries in the csv, each entry will be split by a comma and be added to the 2D dataset array, which will be returned by this function.

Search Class

The search class's most important method, searchCategory, takes a string (channel category) as a parameter. It will call the readCSV class's CSVReader method to get the dataset stored in a 2D array. A YouTuber arraylist is created and using a for loop, all the data from the 2D array get stored as a YouTuber object. Then, it is added into the YouTuber arraylist. Using another for loop, the channel category is compared with every YouTuber object's category within the YouTuber arraylist. To make the program more compatible with client's input, spaces and capitalization are ignored. If the category of a YouTuber matches, that specific YouTuber is added to another arraylist which will be returned by this function.

Sort Class

The Sort class is used to sort a given arraylist of YouTuber in descending order based on the numbers of followers and videos of the YouTuber. The Sort class has 4 static methods and 8 private methods. The 4 static methods are classified by mergesort and quicksort, and each

sorting method uses the numbers of followers and videos of the YouTuber to generate a sequence of YouTuber. The static methods only take one parameter which is the input arraylist of YouTuber and call the corresponding private methods to simplify the developers' operations. The mergesort contains two parts, merge and sort. The merge method takes four parameters, input arraylist of YouTuber *arr*, integer *l*, *m*, *r*. The merge method merges two subarrays of *arr*, the first subarray is *arr[l..m]* and the second subarray is *arr[m+1..r]*. The merge method first creates two temp arrays and stores the data from the subarrays to the temp arrays. Then the two subarrays are merged into one that the elements are sorted in descending order. The sort method takes three parameters, input arraylist of YouTuber *arr*, integer *l* and *r*. The method keeps calculating the middle point *m* between *l* and *r*, and dividing *arr* into subarrays using the middle points. Last, the method sorts *arr[l..r]* by calling merge recursively. The quicksort consists of two methods, partition and sort. The partition function takes three parameters, input arraylist of YouTuber *arr*, integer *low* and *high*. The function takes the last element in *arr* as pivot, the pivot element will be placed at the correct position that all the elements on its left are greater than it and all the elements on its right are smaller than the pivot. The sort part of quicksort still takes three parameters, input arraylist of YouTuber *arr*, integer *low* and *high*. The method calls partition and gets the partitioning index of the pivot which is already at the right place. Then the function calls itself recursively to separate *arr* into subarrays using the pivot and sort the elements.

Graph Class

The Graph class is used to provide some operations on a graph. First, it has four state variables - *YouTubers*, *adj*, *V*, *E*. "*YouTubers*" is an ArrayList of YouTuber, and it stores the vertices of the graph object. "*adj*" is a 2D ArrayList. It stores the adjacency list of each vertex in the graph. This class has seven methods and a constructor. The constructor *Graph()* is used to initialize a graph object. The "*addEdge(YouTuber v1, YouTuber v2)*" function requires two parameters of YouTuber and is used to add edges between the two YouTubers. When adding the edge, it first checks whether the ArrayList "*YouTubers*" has already contained the two vertices or not. If it has not contained yet, the YouTuber is added to the ArrayList "*YouTubers*" and an empty ArrayList<YouTuber> is added to *adj*. Also, the number of vertices(*V*) will increase by 1. Then, it uses the method "*getYouTuberIndex()*" to get the

index of a YouTuber in YouTubers. This index is also used for adj. Therefore, it forms the one-to-one corresponding relations between YouTubers and adj. Then, v2 was added to v1's adjacency list and v1 to v2's adjacency's list. The number of edges (E) will increase by 1. These two methods V() and E() are used to get the number of vertices and edges. The two methods getYouTubers and getAdj are used to get the list of vertices and the adjacency list. The method "contains" checks whether an input YouTuber has already been contained in "YouTubers" or not. The last method named getYoutuberIndex() is used to get the index position of a YouTuber in "YouTubers". It uses a for loop to complete.

DFS Class

In the DFS class, there are five private variables - source, temp, marked, edgeTo, g. "g" and "source" is used to store the input YouTuber of the constructor. "temp" is almost the same with "source". The difference is that "temp" points to the address of a YouTuber object in the graph while "source" points to the address of a YouTuber object outside the graph. The constructor DFS initializes the DFS object. It requires two inputs - an undirected graph and a source YouTuber. Inside the constructor, it uses a for loop to find the "temp", which has the same values of state variables with the input "source", in the graph. Then, it initializes the marked array and edgeTo array. Next, it calls the private "dfs()" method to do depth-first search. The "dfs" method first gets the index position of the source YouTuber in YouTubers which is the list of vertices in the graph. Then, it marks the corresponding position of the "marked" array as true. Then it will do dfs recursively in the source vertex's adjacency list. Vertices which are marked in the adjacency list will be skipped. The "visited" method checks whether a YouTuber vertex has been visited or not. It can be directly checked by using the "marked" array. Besides, it has a method named "relatedYouTubers()". It returns an ArrayList of YouTubers that is from the same country and has the same category with the sourcer YouTuber. It uses a for loop to check the "marked" array. It adds the YouTubers that are marked and have the same category with the input source YouTuber to the output ArrayList.

Main Class

The Main class provides comprehensive methods for the front-end module. There are four static methods and the methods combine all the modules together. The `getCountryFollowers` and `getCountryVideos` take a source YouTuber as input, and return a sorted arraylist of YouTuber based on the country of the source YouTuber, and the number of Followers or the videos uploaded. The two methods first call the `readCSV` module to generate a graph. Then DFS is called to get the related YouTubers of the source YouTuber. Last, the Sort Module is used to sort the given arraylist of YouTubers. The other two methods `sortedByFollowers` and `sortedByVideos` take a category as input, and return a sorted arraylist of YouTuber under that category based on the number of Followers and the videos uploaded. The functions first call the Search Module to get the arraylist of YouTuber under the specified category. Then the functions call the Sort Module to sort the provided arraylist of YouTuber.

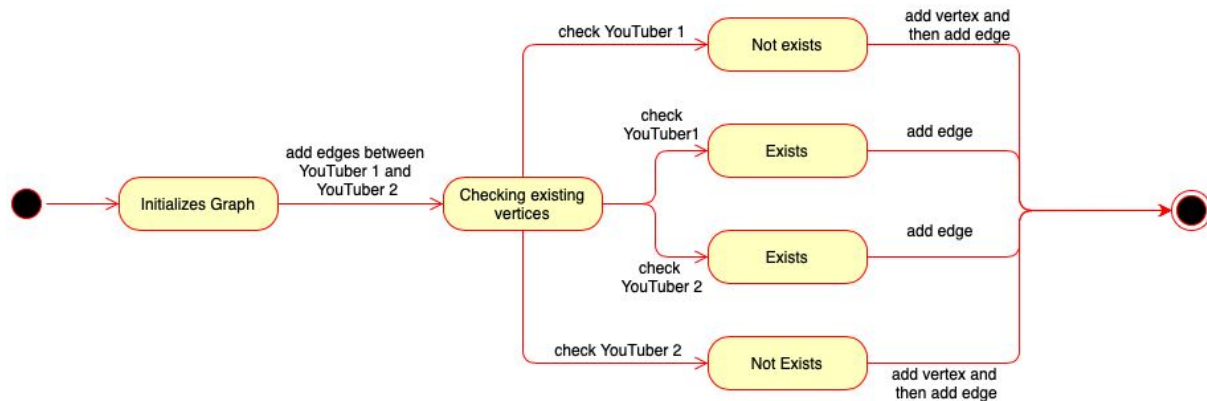
FrontEndServlet Class

The purpose of the `FrontEndServlet` module is to connect with the endpoints on the back-end Main module and act as the view and controller for the project. The module needs to maintain a *serial* state variable, which is used to ensure that the sender and receiver, where the sender is the user and the receiver is the program, have loaded all of the classes for the servlet to function in terms of serialization. More information can be found on the Java documentation on servlets. The `FrontEndServlet` displays two pages. One where the user can decide their response using search criteria and another where the user can view the response to their response. The user can choose to get channel suggestions based on their channel preference and sort their suggestions based on the number of followers or video uploads. The user has an optional feature to choose, which is geographical suggestions, where the program will generate a list of YouTubers from the same geographical location as the top YouTuber. Details on how the geographical suggestions are made are defined above. Once the user has made their criteria clear, the front-end requests the back-end to supply a sorted arraylist of YouTube channels either based on the number of followers or the number of videos uploaded. The front-end requests in addition to the Main module for a geographically sorted arraylist if the user has toggled this option. This option requires graph construction and traversal, which can make the response time rather slow. Nonetheless, once the back-end

returns the results to the front-end, the front-end displays the results on a new page in a table-like structure. The formatting of this second page ensures maximum reading visibility for the user.

UML State Machine Diagrams

Graph Class



FrontEndServlet Class



Evaluation/Review

Nowadays, when people use YouTube, many unrelated videos are presented on the home page. The design brings inconvenience to users and ruins the users' experience. YouRecommend has many advantages to improve those drawbacks. First, the project is practical because it provides users many choices of recommended YouTubers based on their

channel preferences. Second, the project provides users with options to sort the YouTubers based on different standards.

If more time was given, links between the project web page and YouTube could be added to each searched result. The new feature can make the project more convenient for users to get access to the recommended YouTubers, because users only need to click on the name of those recommended YouTubers to directly go to the corresponding YouTube page instead of going to YouTube and searching the YouTuber. Besides, a comprehensive evaluation system could be developed which is used to evaluate the quality of the videos that a YouTuber uploaded, not just the numbers. For example, the quality of a YouTuber can be calculated by dividing the number of followers by the number of videos. Then, YouRecommend can recommend YouTubers based on this system.