Requirements Specifications Document

Group Number: 02

Lab Section: L01

Team Members: Kabishan Suvendran, Franklin Tian, Jiawei Yu, Bowen Zhang

The domain of our application consists of the back-end, which is comprised of the Read, Sort, Search, Graph and YouTuber modules, and the front-end, which is comprised of the Display module. By enabling low coupling and high cohesion between these modules, we will be able to achieve an application domain in which operations can be executed in a manner that does not affect other modules. In creating all of these modules and organizing them into a USES hierarchy, we wish to be able to sort, search and display YouTube channels that correspond to the channel preference of the user. The stakeholders of this application are solely those seeking entertainment from YouTube. It is also possible for the YouTube corporation to be involved in this application, as they may see some benefits in our product and may choose to modify their algorithm to resemble our solution. This is highly unlikely, however. The goal of this application is to provide a sequence or list of YouTube channels that correspond to the client's preference. Once this sequence has been generated, it is also possible to display a visual representation that orders the channels based on followers. Ordering and searching for these channels will be handled by the Read, Sort, Search and YouTuber modules. Displaying the representation will be added into the module interface specification (MIS) in the future. The Graph module will use searching algorithms to provide further channel suggestions based on the client's channel preferences. As of now, we expect for all of these functionalities to be incorporated in the final product, but we will update this document in the future. Once the client inputs their channel preference, the front-end will communicate with the back-end modules, which will do most of the heavy lifting. The product will be designed in a manner that enables information hiding and encapsulation. The client will only see a list of YouTube channels and a visual representation. The relationships between the modules can be found in the MIS below. We wish to make life easier for the stakeholders of our product by providing accurate recommendations. These recommendations will be checked against YouTube recommendations.

For the functional requirements, please see the MIS below.

In terms of the non-functional requirements, we will discuss reliability, accuracy of the results, performance, human-computer interface issues and portability issues. In terms of reliability, we will host our solution on a trusted cloud application platform like Netlify or Heroku. These domains provide HTTPS encryption, host the website 24/7/365 and do not log user information. Since there is no login or registration feature, encrypting passwords is not a problem. In terms of the accuracy of the results, we can verify the results of our program with channel recommendations from YouTube itself. To mitigate human-computer interface issues, we will design a webpage that is easy to use, as it will have a huge font size, an input box and button. Lastly, since this is a web application, we will try our best to ensure that it is compatible with most mainstream browsers, but Google Chrome and Mozilla Firefox are our primary concerns.

We will be using Git for version control and will ensure the quality of our product by occasionally conducting JUnit tests on a subset of our sample data to ensure that certain features are working properly. We give a high priority to the searching, sorting and graphing algorithms because they perform the bulk of the operations and are integral in completing the application. The front-end has a lower priority in comparison to the back-end. We do not anticipate any likely changes to the system maintenance procedures at the moment. This document will be updated regularly.

Search Module

Module

Search

Uses

YouTuber, Graph

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
Search_id	String, sequence of sequences of YouTuber	sequence of YouTuber	
displayStats	String, sequence of YouTuber	String Country, String Category_name, String JoinDate, ℕ Followers ℕ Videos	NotFoundException

Semantics

State Variables

name: String

youtuberList: sequence of YouTuber

State Invariant

None

Access Routine Semantics

Search_id(n, s)

- transition: name, youtuberList ≔ n, s
- output: out ≔ L : sequence of YouTuber where (x : YouTuber | x.getCategory_name() = s ⇒ L.add(x))
- exception: none

displayStats(s : String, l : sequence)

- output: out ≔ <Country, Category_name, JoinDate, Followers, Videos>
- exception: (s ∉ l ⇒ NotFoundException)

Graph Module

Template Module

Graph(ℕ)

Uses

YouTuber, Read

Syntax

Exported Constants

None

Exported Types

Graph = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Graph	ℕ, sequence of YouTuber	Graph	
Vertices		ℕ	
Edges		ℕ	
addEdge	YouTuber, YouTuber		
adj	YouTuber	YouTuber	

Semantics

State Variables

vertices: ℕ

edges: ℕ

AdjList: 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(V, L)

- transition: vertices, edges, AdjList ≔ V, 0, L
- output: out ≔ self
- exception: none

Vertices()

- output: out ≔ vertices
- exception: none

Edges()

- output: out ≔ edges
- exception: none

addEdge(v, w)

- transition: (v, w : YouTuber | L : sequence of YouTuber : (v =∈ L[0] ∧ L ∈ AdjList ⇒ L.add(w)) ∨ (w ∈ L[0] ∧ L ∈ AdjList ⇒ L.add(v)))
- exception: none

adj(Y)

- output: out ≔ (adjacency : YouTuber | Y: YouTuber, L : sequence of YouTuber : (Y = L[0] ∧ L ∈ AdjList ⇒ adjacency = L[1..|L| - 1])
- 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: out := title

- exception: none

getVideos()

- output: out := videos
- exception: none

Read File Module

Module

Read

Uses

YouTuber

Syntax

Exported Constants

None

Exported Types

Read = ?

Exported Access Programs

Routine name	In	Out	Exceptions
read	String	sequence of YouTuber	FileNotFoundException

Semantics

State Invariant

None

Access Routine Semantics

read(f)

- output: out := sequence of YouTuber in f
- exception: ¬ (f.exists()) ⇒ FileNotFoundException

Sort Module

Module

Sort

Uses

YouTuber, Search

Syntax

Exported Constants

None

Exported Access Programs

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

Semantics

State Variables

s: sequence of YouTuber

State Invariant

None

Access Routine Semantics

MergeSort_Followers()

- output: out ≔ sequence of YouTuber where ∀ i : N|i ∈ [0..|s| - 2] : s[i].getFollowers() ⩾ s[i + 1].getFollowers()
- exception: none

MergeSort_Videos()

- output: out ≔ sequence of YouTuber where ∀ i : N|i ∈ [0..|s| - 2] : s[i].getVideos() ⩾ s[i + 1].get Videos()
- exception: none

QuickSort_Followers()

- output: out ≔ sequence of YouTuber where ∀ i : N|i ∈ [0..|s| - 2] : s[i].getFollowers() ⩾ s[i + 1].getFollowers()
- exception: none

QuickSort_Videos()

- output: out ≔ sequence of YouTuber where ∀ i : N|i ∈ [0..|s| - 2] : s[i].getVideos() ⩾ s[i + 1].get Videos()
- exception: none

Main Module

Module

Main

Uses

Graph, YouTuber, Search, Sort

Syntax

Exported Constants
None

Exported Access Program

Routine name	In	Out	Exception
init			
creatingGraph			
afterSearching	String	sequence of YouTuber	
afterSorting		sequence of YouTuber	

Semantics

State Variables

g: Graph
list: a sequence of YouTuber
list2: a sequence of sequences of YouTuber
list3: a sequence of YouTuber
list4: a sequence of YouTuber

State Invariant

N = 110000

Assumptions

init should be called before calling any other methods. Then, creatingGraph should be called.

Access Routine Semantics

init()

- transition: g := new Graph(N)
- exception: none

creatingGraph()

- transition: i : $\mathbb{N}$ | i ∈ [0..|list - 1|] : (j : $\mathbb{N}$ | j ∈ [0..|list - 1|] : list[i] = list [j] ⇒ list2[i].add(list[j])

- exception: none

afterSearching(x : String):

- transition: list3 ≔ (i ∈ [0..|list2| - 1] ∧ x = list2[i][0].getCategory_name() ⇒ list2[i])
- output: out ≔ list3
- exception: none

afterSorting():

- transition: list4 ≔ list3
- output: out ≔ (x = “videos” ⇒ list.MergeSort_videos() ∨ list4.QuickSort_videos() | x = “Followers” ⇒ list4.MergeSort_Followers() ∨ list4.QuickSort_Followers())
- exception: none

Display Module

Module

Display

Uses

YouTuber, Search, Graph

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
User_Input	String	sequence of YouTuber	EmptyException

Semantics

State Variables

pref: String

s: sequence of sequences of YouTuber

State Invariant

None

Access Routine Semantics

User_Input(pref)

- output: out := Search_id(pref, s)
- exception: (|pref| = 0) ⇒ EmptyException