Design Specification Version 1.0

For Project Avocado
Lab 01 Group 05

Erfan Jamalifar
Dakota McKay
Andrew Mitchell
Daniel Wu

Computer Science 2XB3
Computer Science Practice and Experience: Binding Theory to Practice
Department of Computer Science McMaster University

April 15, 2019

1 Revision History

1.1 Revision History

Version	Date	Comments
0.1	March 1st	Initial Design
0.2	March 15th	Friends Added as Feature
0.3	April 1st	Implemented skeleton of GUI
0.4	April 7th	Implemented text UI in case GUI was not finished in time
1.0	April 14th	Finalized Avocado

1.2 Team Members

Name	Student Number	Roles\Responsibilities
Erfan Jamalifar	400027547	Log Admin
Dakota McKay	001421859	Researcher
Andrew Mitchell	001225727	Project Lead
Daniel Wu	001158441	Database Lead

1.3 Attestation of Consent

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.

2 Contribution Page

Name	Role(s)	Contributions	Comments
Erfan Jamalifar	Log Admin	Log and Meeting minutes, UI state chart, GUI research, GUI development, Documentation	
Dakota McKay	Researcher	Enums, basis of User class, score calculation, much documentation	
Andrew Mitchell	Project Lead	Project design, finished user class, contributed to Jdbc class, combined modules into working project	
Daniel Wu	Database Lead	Database research, set up database, connected Java to database, implemented Jdbc class	

3 Executive Summary

Avocado is an application meant to help its users pay back their student debt faster by means of gamification and competition. Avocado is intended to help Canadian former students who recently graduated from a post secondary institution. This is accomplished by collecting data from users and combining it with median demographic income from statistics Canada to provide each user a score on a level playing field. This score can then be used in either an optional global leader board or with a user constructed leader board of friends, to provide both accountability and light competition.

4 Class\Module Description

4.1 Class\Module Explanations

4.1.1 EducationLevel

The EducationLevel Module is an enum that represents the education levels represented in our dataset.

4.1.2 FieldOfStudy

The FieldOfStudy Module is an enum that represents the fields of study represented in our dataset.

4.1.3 Jdbc

Establish connection with local database and facilitate data manipulation in specific datatypes.

4.1.4 Location

The Location Module is an enum that represents the locations represented in our dataset.

4.1.5 MergeSort

The MergeSort module contains the functions used to sort data. Specifically the Module implements a mergesort algorithm and can accept Arrays of any object that implements compareable.

4.1.6 NotCurrentUser

NotCurrentUser is a class meant to store information about other users while still in a session for a user. This is done as loading a full user object takes more memory and time as well as exposes information to uneccessary risk.

4.1.7 Session

This class is the Controler of the application when Avocado is viewed in the Model View Controler scheme. Running this class runs the application as this class is mostly a main function that calls everything needed to run the application.

4.1.8 Sex

The Sex Module is an enum that represents the sexes represented in our dataset.

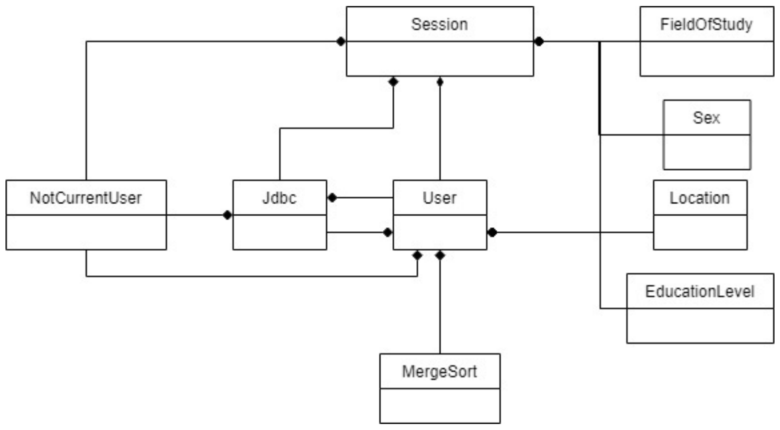
4.1.9 User

The User class stores all of the information about the user whose session is active. This class is a local copy of all of the user information from the database. This object is where nearly all information about the user will come from for the duration of a session after logging in.

4.2 Reason For Particular Decomposition

The reason that this particular breakdown into these classes is a desire to keep the cohesion of the modules and classes high. However while it did lead to good results in coheision of modules, it did cause high coupling. Another reason for the particular breakdown is to keep the concerns separate.

4.3 Diagram of Uses Relationships



4.4 Class Interfaces

4.4.1 EducationLevel

public class	EducationLevel	
int	toIncomeVal()	Returns part of the income co-ordinate the enum this is called on
int	toDebtVal()	Returns part of the debt co-ordinate the enum this is called on

4.4.2 FieldOfStudy

public class	FieldOfStudy	
int	toVal()	Returns part of the income co-ordinate the enum this is called on

4.4.3 Jdbc

public class	Jdbc	
boolean	validLogin(String username, String password)	Runs ValidateSQL query return the boolean value that represents the validity of their user login
boolean	userExsits(String user)	Return the ExistSQL query and returns boolean that represents either user exist or not
User	loadUser(String username)	Return username's attributes in the hard coded order
void	saveUser(User U)	Update the the LOGIN_TB after the user is registered in createUser() function as an OPTED user, and the calls setRanked() to ranked the users in ascending order
void	createUser(String username, String password)	new users by providing username and password and prepare all the attributes for saveUser() to set the data
void	addFriend(int YOU, int ME)	Takes ME friend code and You friend code add freidn(s) to each user
boolean	friendCodeExists(int friendID)	Takes friendID return a boolean representation of friend(s) existence
boolean	userExists(String username)	Takes a username and returns a boolean value which represents the existing of user
void	createTables()	Create Debt table, income table, user table, and friend table
void	createFriendTable()	creating the Friend table if the table does not exist
void	createUserTable()	Runs the sql query to create login table (LOGIN_TB)
void	createDebtTable()	Create the debt table (debt_min)
void createIncomeTable()	Create the income table (income_min)	
NotCurrentUser	getLeaderboard(int r)	Takes rank(r) returns OPTED user at rank, else "NOT FOUND"
void	setRanks()	Runs queryGetRank query which outputs USERNAME, and descending order of SCORE, and update the LOGIN_TB with a rank wiht each user
double	getDebtData(String coord)	Takes coord (coordinate) to run the DebtSelect query to return float of DE_VALUE
double	getIncomeData(String coord)	Takes coord (DE_COORDINATE) to run the IncomeSelect query to return float of IN_VALUE

ArrayList <NotCurrentUser>	getFriends(int code)	Constructs a NotCurrentUser for each friend of the user with friendCode code and returns them as an ArrayList
ArrayList <String>	getSuggested(int code, int howmany)	Finds howmany users connected to the user with friendCode code, and returns their usernames in an ArrayList.
Connection	getConnection()	Using the database driver to facilitate the connection using localhost, username, and password in order for the java implementation to run queries to manipulate all existing data.

4.4.4 Location

public class	Location	
int	toVal	Returns part of the dataset co-ordinate the enum this is called on

4.4.5 MergeSort

public class	MergeSort	
return type	method signature	method explanation

4.4.6 NotCurrentUser

public class	NotCurrentUser	
	NotCurrentUser (String Username, double Score)	Creates a NotCurrentUser object with the specified username and score
String	getUsername()	Returns the username given in the constructor
double	getScore()	Returns the score given in the constructor
int	compareTo(NotCurrentUser that)	returns 1 if the of that is less than the score of the this, 0 if the score of this is equal to the score of that, and -1 if the score of that is less than the score of this.

4.4.7 Session

public class	Session	
	Session()	Creates a session object which runs all user interaction through terminal for the operation of the application

4.4.8 Sex

public class	Sex	
int	toVal()	Returns part of the dataset co-ordinate the enum this is called on

4.4.9 User

public class	User	
	user()	constructor meant to be used on account creation
	user()	constructor meant to be used on loading an existing account
double	getMedianDemographicIncome()	Returns a double representing the user's median demographic income
double	getMedianDemographicDebt()	Returns a double representing the user's median demographic debt
double	getScore()	Returns the user's calculated score
String	getUsername()	Returns the user's username
int	getRank()	Returns the number of user's who have a higher score than the current user + 1
double	getDebtAtGrad()	Returns the user's debt at graduation
double	getCurrentDebt()	Returns the user's current debt
void	setCurrentDebt(double debt)	Sets the user's current debt to the given double
double	getInterestRate()	Returns the interest rate on the user's debt
String	getGradDateS()	Returns the graduation date of the current user as a string
String	getDateOfBirthS()	Returns the user's Date of Birth as a String
int	getFriendCode()	Returns the user's friend code
boolean	optedIn()	Returns true if the user has opted into the leaderboard and false otherwise
void	optIn()	Opts the user into the leaderboard
void	optOut()	Opts the user out of the leaderboard
String	getSexS()	Returns the user's sex as a string
String	getLocationS()	Returns the User's location as a string
String	getFieldOfStudyS()	Returns the user's field of study as a string
String	getEducationLevelS()	Returns the user's education level as a string
int	getAgeAtGrad()	Calculates the user's age at their graduation date in years
int	getAge()	Calculates the user's current age in years and returns it
ArrayList <NotCurrentUser>	getFriends()	Returns a list of friends including the current user in order of score
ArrayList<String>	getSuggested()	Returns an array list of the usernames of possible friends for the current user
String	getDebtCoordinate()	Returns the co-ordinate for the debt dataset which details the key for the table
String	getIncomeCoordinate()	Returns the co-ordinate for the income dataset which details the key for the table
void	scoreCalc()	Calculates and stores the user's score
ArrayList <NotCurrentUser>	getFriendLeaderboard()	Returns an array list of the user's friends and them sorted by their score.

4.5 Class\Module Implementation Descriptions

4.5.1 EducationLevel

Class Variables
None
Class Private Functions
None
Implementation Description
Education level was designed as an enumerated type and as such it has no state variables. The values for this enum were taken from the levels of higher education that Statistics Canada recognizes. There is a public function for ease in converting the enum into the integer portion of a co-ordinate for determining a user's demographic data.

4.5.2 FieldOfStudy

Class Variables
None
Class Private Functions
None
Implementation Description
Field of Study was designed as an enumerated type and as such it has no state variables. The values for this enum were taken from the fields of study in higher education that Statistics Canada recognizes. There is a public function for ease in converting the enum into the integer portion of a co-ordinate for determining a user's demographic data.

4.5.3 Jdbc

Class Variables
None
Class Private Functions

- createFriendTable()
- createUserTable()
- createDebtTable()
- createIncomeTable()
- setRanks()
- ArrayList¡Integer¿ getFriendsCodes(int code)
- ArrayList¡Integer¿ friendsOfFriends(ArrayList¡Integer¿ inputCodes)
- ArrayList¡String¿ getDatabseInfo()

- Connection getConnection()

Implementation Description
Jdbc(Java Database Connection) was designed to have stable connection with database and Jdbc class. The class create many tables, store(incomes, debts, user login, and friend data), delete, update data with designated return types with levels of security that prevented SQL injection attack.

4.5.4 Location

Class Variables
None
Class Private Functions
None
Implementation Description
Location was designed as an enumerated type and as such it has no state variables. The values for this enum were taken from the location levels that Statistics Canada provided in the datasets. There is a public function for ease in converting the enum into the integer portion of a co-ordinate for determining a user's demographic data.

4.5.5 MergeSort

Class Variables
None
Class Private Functions

- sort(Comparable[] a, Comparable[] aux, int lo, int hi)
- merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)

Implementation Description

4.5.6 NotCurrentUser

Class Variables

- String Username
- double Score

Class Private Functions
None
Implementation Description
This class takes a score and username upon creation and assigns them to the class variables. These variables are not allowed to change as they are private variables and there are no functions that modify them

4.5.7 Session

Class Variables

- User user

Class Private Functions

- Start(Scanner sc)
- login()
- signup()
- homepage(Scanner sc)
- friends(Scanner sc)
- addFriend(Scanner sc)
- leaderboard(Scanner sc)
- update(Scanner sc)
- debtupdate(Scanner sc)

Implementation Description

This is the basic text UI. This class controls a session of Avocado, and every other class is called by this. Each function represents a UI state, and the session moves from state to state by calling each state's function.

UML State Machine

4.5.8 Sex

Class Variables

None

Class Private Functions

None

Implementation Description

Sex was designed as an enumerated type and as such it has no state variables. The values for this enum were taken from the sexes that Statistics Canada recognizes. There is a public function for ease in converting the enum into the integer portion of a co-ordinate for determining a user's demographic data.

4.5.9 User

Class Variables

- double medianDemographicIncome
- double medianDemographicDebt
- double score
- String username
- int rank
- double debtAtGrad
- double currentDebt
- double interestRate
- Date gradDate
- Date dateOfBirth
- ArrayList¡NotCurrentUser¿ friends
- ArrayList¡String¿ suggestedFriends
- boolean optedIn
- Location geoLocation
- EducationLevel educationLevel
- FieldOfStudy fieldOfStudy
- Sex sex

Class Private Functions

- generateFriendCode()
- getMedianData()
- findFriends()
- sortRank(ArrayList¡NotCurrentUser¿ list)

Implementation Description
Custom data type containing all of a users information, and all relevant methods for working with the users information
UML State Machine

5 Internal Review of Design

5.1 Efficiency

Description
MergeSort being an example of divide-and-conquer paradigm is an extremely efficient algorithm to implement in context of sorting all your friends' by score when User.java calls MergeSort.java.

JDBC is efficient because most of the sorting and searching is dealt through native SQL queries, and these functions are efficiently written, fast, and minimizes errors.

5.2 Session

Description
Originally we wanted to implement the GUI, but due to the time restriction of the semester, we were only able to complete 3/4 of the GUI; therefore we opted to a simpler text User Interface.

5.3 NotCurrentUser

Description
We wanted to implement a comparable data type that contain a username and score, so they can be sorted, and this was the best way to handle the task.

5.4 FieldOfStudy

Description
This is dealt using enum, and we be lived that this is the most effective way to handle when the class is called.