Date: 9th October 2015

Time: 10.30am – 12.30am (Code Review)

12.30-1.30pm (Review of Iteration)

Venue: SOA SR 2-2

Attendees: All

Agenda:

* Code Review
* Code Sharing

The following was decided after much discussion:

## **Basic App Functionality:**

Breakdown by usage time category :

* take in startdate and enddate**: retrieveAppByStartEndDate**
* Get the app usages from AppDAO given a start date and end date
* **calculateAppUsageTime (App1, App 2)**: takes in the first app usage and the second app usage
* If different app user, check the next day and see if its above 120 seconds, check if they are in the same day as well
* (i+1) < appList.size() ensures that there is no index out of bounds, with reference to the last app
* Return type: HashMap<String, Integer> : user, usage time
* **retrieveAppByStartEndDate**, passes into the **retrieveUSageCategory** which calculate and categorises. Both of which are called in the **ViewController**

Breakdown by Usage Time category and Demographics: **retrieveAppUsageByDemographicBreakdown()**

* App: Sort by year, gender and school
* App -> breakdown by year, then gender, and further breaks down into school : using a **recursion**. Consist of three methods
  + sortByGender
  + sortByYear
  + sortBySchool
* Basic App Usage Object: contains an ArrayList of BAU (which one consists of its particular breakdown). Other attributes(variable) include:
  + Percentage
  + Count
  + List<App>
  + Category (sort by year, gender, school)
  + sortCategory: possible **values** of category
* Subdividing apps into the further breakdown using the recursion method
* Reuse **retrieveAppByStartEndDate** to get the appList
* totalUniqueMacAddress: global variable
* currLevel: the current level to be reached in the hierarchy
* maxLevel: the max level to be reached
* When currLevel and maxLevel is the same: calculate usage intensity, we store it at that level of **categoryListResult**: List of linkedHashMap (an attribute of BAU)
* Each BAU contains its own list of BAU (by the characteristic based on choice selection from the user in the UI)
* For example: After breaking down sort by gender, the BAU will consist BAU objects of both male and female
* After going through the first sorting, currLevel++ (increase the level by one) and goes on to the next level of sorting
* Breakdown by category: breakdown based on the last list of BAU
* **basicAppUsage.getBasicAppUsageList() == null**: check if we are at the first level
* ArrayListy<LinkedHashMap<String, Integer>: parameter of category list result

Breakdown by app category:

* **retrieveAppUsageByCategory:** between a startDate and EndDate
* Uses HashMap<String, String> for dynamic display: show 0 entries for particular category as well
* Reusing the same logic for **calculateAppUsageTime (firstApp, nextApp)**

Diurnal pattern of app usage time

* Gist of this function: By each hour, what is the usage?
* currDate would be the cursor between start date and end date
* currCalendar.add(GregorianCalendar.MINUTE, 59), currCalendar.add(GregorianCalendar.SECOND, 59)
  + Both methods ensures there is no overlap of dates (Wiki requirements)
  + E.g. 12AM (inclusive) – 1AM (exclusive)

## **Top-K App Functionality:**

Retrieve Top-K App:

* Retrieve app and rank from AppDAO. CalculateAppDuration method to get the usage time. It also check by school via the field email in demographics.
* After retrieval, it sorts the app based on their duration by descending. It will then check for the duration time, if the duration time are the same for the 2 records, it will be assigned the same rank.
* Return the List of Top-K app

Retrieve by Top-K Student:

* Do not check the categories from the hql, only check for categories inside controller.
* Do not filter by categories first, as the usage time will be different, so have to sort by the usage time first. Return the linkedhashmap after it is sorted

Retrieve by Top-K School:

* Sort by school, Assign it in the order then pass it to the UI

JSONController:

* Check for the common validator, and invalid inputs. List will be put inside the node so that the success message will be displayed. This is store in the TopKApp class for JSON output.

## **Smartphone Overuse Functionality:**

* Only for one user so have to take in the email. It is to calculate the average usage time.
* 3 different methods to calculate the average daily smartphone, average daily gaming duration and smartphone access frequency.
* For game duration, we have to retrieve the game category first.
* Access frequency is by hours instead of days. It will add the number if it is more than 120 minutes or it is in 2 different hours as it may be different applications. Assign light = 0, moderate = 1, severe = 2.

JSONController:

There is a SmartphoneOveruse class to store the JSON output. This also check for the common validator and invalid inputs. Then, store the overuse index here. And then have a node to store the categories.

SOP for Integration:

* Naming Convention (Done during SD discussion)
* Schedule a PP session to do code clean-up
* Regression Testing split into separate tasks, allocate more hours

**Iteration 2 Review**

Coding: Be sure of your own functions and other function (sharing of code knowledge at the start during planning)

Coding tasks to be at least 6 hours and above. Each PP session should not be in one hour, minimum 2 hours.

Test cases should be done by everyone, during the planning phase and should be completed before the any PP starts

Research at least 10 hours, SD assign at least 3 hours, test case 2 hours

Research in groups and minutes attached for the group meeting on research

During integration testing, if there is bug found in your function (“White-label error”), categorise it as TEN and solve it immediately.

Team finds it okay to lump JSON with non-JSON functionalities in one task

We should include buffer time in between PP sessions to prevent mass schedule changing for the time.

Regression testing should be the last task for the iteration, but in the situations of bugs found, log it into bug log and decide if there is a need for subsequent debugging sessions.

Entity type of angular brackets <> should be stated for JSP pages. Openshift will crash if absent.

The meeting was adjourned at 1.30 PM.

Prepared by,

TANG SHING HEI

Vetted and edited by,

REMY NG

**NOTE:** Updated diagrams will be submitted by the end of today.