Project Design Document

Group 3:

Ashraya Regmi, Jesse Cruse, Tyler Roland, Matthew Stevenson

University of Maryland University College

Revision 1.05

June 12, 2017

# **Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Editor** | **Description of Changes** |
| 1.00 | 06/12/2017 | Matthew Stevenson | Original Document   * Added Items to Class Design Section |
| 1.01 | 06/13/2017 | Tyler Roland | * Minor text fixes * Added pseudocode to make sure user is logged in for user pages |
| 1.02 | 06/14/17 | Jesse Cruse | * Added Event-Trace scenario one |
| 1.03 | 06/15/17 | Ashraya Regmi | * Added unresolved risk and possible mitigation |
| 1.04 | 6/17/17 | Jesse Cruse | * Added Algorithm Pseudocode * Added Event-trace scenarios 1-3 * Added notable project plan adjustments |
| 1.05 | 6/18/17 | Jesse Cruse | * Added Event-trace scenario 4 * Updated pseudocode with error handling |

System Functional Requirements:

|  |  |
| --- | --- |
| **Requirement #** | **Description** |
| 1 | This program will have a user interface to interact with the web application. The user interface will include a submission form for additional item entries and will also output a shopping list. |
| 2 | The program tracks and stores what the user has bought over time. If an item has already been stored in the database (product ID, description, price), then only new purchase dates will be entered for that item. No other details are required to be stored. |
| 3 | This program will allow the user to generate a shopping list for regular interval trips as well as special trips. The user interface screen will have a button that will generate a list of items that the algorithm deems ought to be bought at that point in time. The shopping list will have check boxes next to each item so that purchase dates are recorded only for items purchased. If an item is displayed on the list but is not needed, the user simply skips checking it and no purchase date will be recorded. |
| 4 | This program will require a backend database to store purchase and user details. The database will store a product ID and product description for each distinct product. |
| 5 | This program will enable the user to email lists to themselves from the website or print the list. |

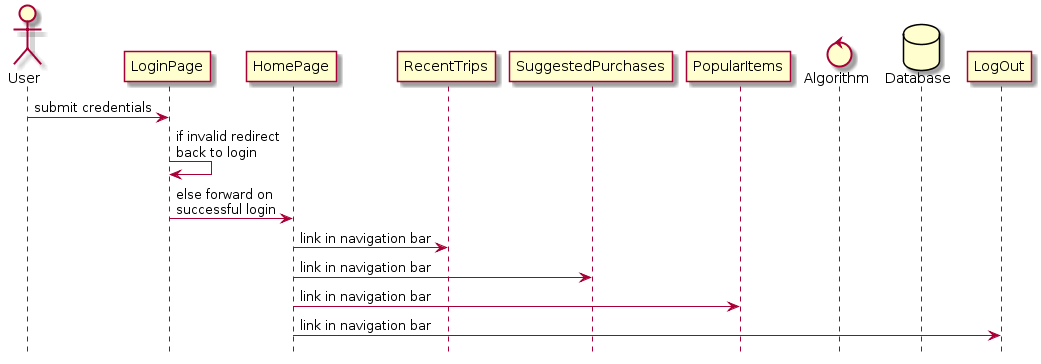
1. **Event Trace Diagram**

**Scenario 1: Start Up**

**Description:** Start up. The user opens the app and logs in.

**Precondition**: Online account is created and accessible. Database is populated with some shopping items to show on a shopping list.

**Post-condition:** User is taken to home page.

****

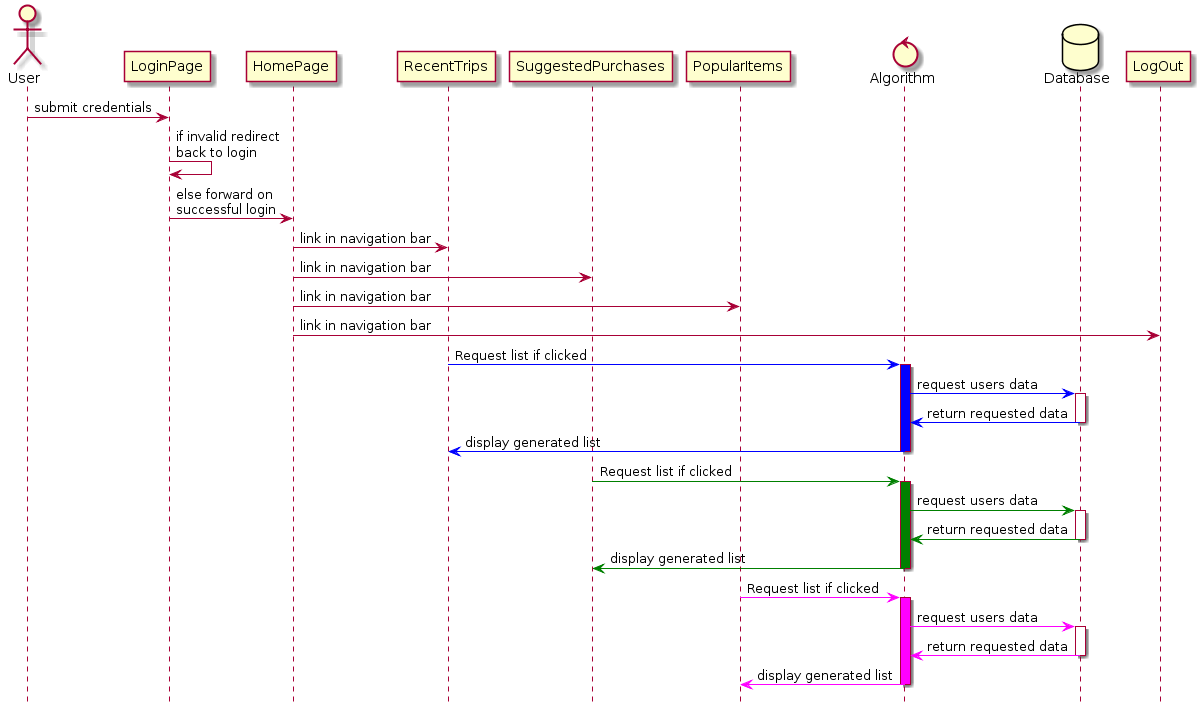
**Figure 1: Scenario 1 - Start Up**

**Scenario 2: Normal Operation**

**Description:** The user logs in and is forwarded to the home page. The user selects ‘Recent Trips’, ‘Suggested Purchases’, or ‘Popular items’. Selected content is generated by the back-end algorithm and is fed to the front-end for display. Links to program functionality are displayed in the navigation bar.

**Precondition**: Online account is created and accessible. Database is populated with some shopping items to show on a shopping list.

**Post-condition:** User shopping list is displayed if there are products that should be purchased based on logic of the algorithm.

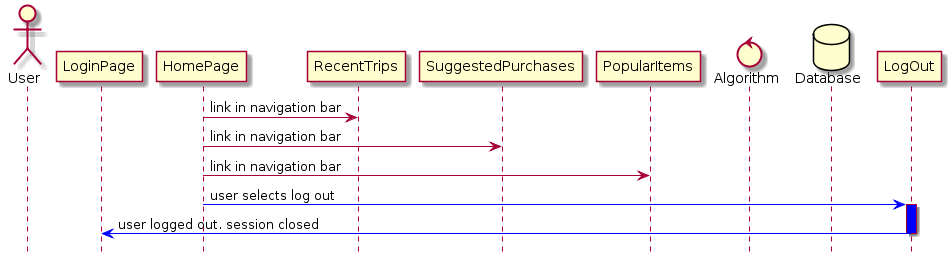
****

**Figure 2: Scenario 2 - Normal Operation**

**Scenario 3: Shut Down**

**Description:** The user is finished using the app and logs out.

**Precondition**: User is currently logged in.

**Post-condition:** User is successfully logged out, session is closed, and context is returned to user.

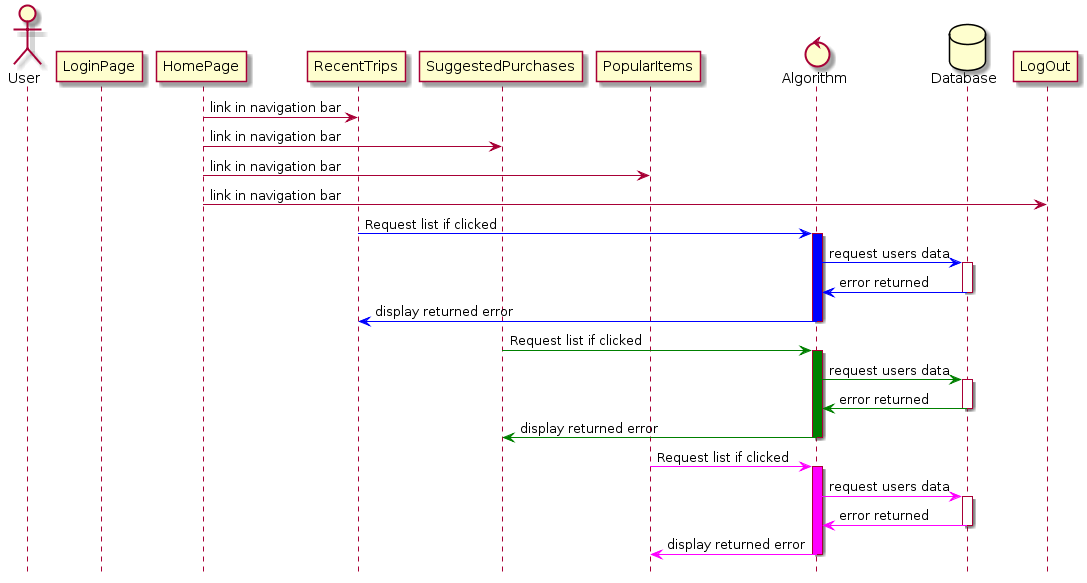
**Figure 3: Scenario 3 - Shut Down**

**Scenario 4: Error Handling**

**Description:** The user logs in and is forwarded to the home page. The user selects ‘Recent Trips’, ‘Suggested Purchases’, or ‘Popular items’. The back-end algorithm detects that there is a database connectivity issue. Program displays error on screen.

**Precondition**: Online account is created and accessible. Database is populated with some shopping items to show on a shopping list. User is already logged in.

**Post-condition:** Error message is displayed to user.

****

**Figure 4: Scenario 4 - Error Handling**

1. **Class Design**

In this section, we will list pseudocode of webpages, php, and database connections. There is a page for each column in the sequence diagrams and all actions are accounted for via webpages.

1. Login Page

HTML

Head

Title Predictive Shopping App End Title

End Head

Body

Create form

Username

Password

Submit

Reset

PHP

Connect to Database

If(account in database) {

Get user credentials from form;

Verify Credentials;

If (verified){

Go to homepage;

} else {

Error message;

}

}

End PHP

End Body

End HTML

1. Header Page

HTML

Head

Title End Title

End Head

Body

Create Unsorted List

List Items:

1. Recent Trips
2. Suggested Purchases
3. Popular Items
4. Logout

End Unsorted List

End Body

End HTML

1. Footer Page

HTML

Head

Title End Title

End Head

Body

Link to print current page End Link

End Body

End HTML

1. Home Page

PHP

Redirect if not logged in

End PHP

HTML

Head

Title Home Page End Title

End Head

Body

Import header page

Import footer page

Unsorted List

List Items

1. Add/Edit Shopping List
2. Remove Shopping List

End Unsorted List

PHP

If (user has shopping list){

Display shopping list as unsorted list links

} else if {

No Shopping Lists

} else {

Error Connecting to Database

}

End PHP

End Body

End HTML

1. Add/Edit Shopping List

PHP

Redirect if not logged in

End PHP

HTML

Head

Title Add/Edit Shopping List End Title

End Head

Body

Import header page

Import footer page

Unsorted List

List Items

1. Add Item to List
2. Remove Item from List

End Unsorted List

PHP

If (user has shopping list) {

Display contents of shopping list

} else if {

No Shopping Lists found

} else if {

No Items on Shopping List

} else {

Error Connecting to Database

}

End PHP

End Body

End HTML

1. Add Item to List

PHP

Redirect if not logged in

End PHP

HTML

Head

Title Add Item to List End Title

End Head

Body

Import header page

Import footer page

Create Form

1. Item Name
2. Item Quantity
3. Is Item Recurring?
4. Reset button
5. Submit Button

PHP

Connect to Database

If (db exists) {

Add item to database item

} else if (db does not exist) {

Create db item

Add item to db item

} else {

Cannot connect to db

}

End PHP

End Body

End HTML

1. Remove Item from List

PHP

Redirect if not logged in

End PHP

HTML

Head

Title Remove item from List End Title

End Head

Body

Import header page

Import footer page

Create Form

1. Item to Remove
2. Remove all instances of item
3. Reset button
4. Submit Button

PHP

Connect to Database

If (db exists) {

Remove db item;

} else if (db does not exist) {

Error db does not exist;

Exit to Homepage;

} else {

Error Cannot connect to db;

}

End PHP

End Body

End HTML

1. Remove Shopping List

PHP

Redirect if not logged in

End PHP

HTML

Head

Title Remove Shopping list End Title

End Head

Body

Import header page

Import footer page

Create Form

1. Shopping List to Remove
2. Reset button
3. Submit Button

PHP

Connect to Database

If (db exists) {

Remove db item;

} else if (db does not exist) {

Error db does not exist;

Exit to Homepage;

} else {

Error Cannot connect to db;

}

End PHP

End Body

End HTML

1. Recent Trips

PHP

Redirect if not logged in

End PHP

HTML

Head

Title Recent Trips End Title

End Head

Body

Import header page

Import footer page

PHP

Connect to Database

If (db exists) {

If (user has record of shopping trips) {

Output to screen recent shopping trip

Allow user to clink to view previous trips

} else if {

No trips

} else {

Error Message

}

} else if (db does not exist) {

Error db does not exist;

Exit to Homepage;

} else {

Error Cannot connect to db;

}

End PHP

End Body

End HTML

1. Suggested Purchases

PHP

Redirect if not logged in

End PHP

HTML

Head

Title Suggested Purchases End Title

End Head

Body

Import header page

Import footer page

PHP

Connect to Database

If (db exists) {

If (user == user in db) {

If (user has purchases) {

If (purchase expiring) {

Consider stocking up;

}

If (product == another product type) {

You might be interested;

}

} else {

No purchases to display;

}

} else {

Error Message

}

} else if (db does not exist) {

Error db does not exist;

Exit to Homepage;

} else {

Error Cannot connect to db;

}

End PHP

End Body

End HTML

1. Popular Items

PHP

Redirect if not logged in

End PHP

HTML

Head

Title Popular Items End Title

End Head

Body

Import header page

Import footer page

PHP

Connect to Database

If (db exists) {

If (items in db == top 10 items) {

Output items to screen as list with potential to be added to list;

}

} else if (db does not exist) {

Error db does not exist;

Exit to Homepage;

} else {

Error Cannot connect to db;

}

End PHP

End Body

End HTML

**Algorithm Pseudocode**

/\*

\* Author: Jesse Cruse

\* Date: 14 June 2017

\* Program: Predictive Shopping App

\* File Desscription: Shopping app algorithm pseudocode

\*/

**Class** ShoppingList

//Declare class variables

**Declare** Private Numeric currentShoppingList //Inital empty array of items to buy

//Constructor

**Constructor** **Method** makeShoppingList**()**

int[] itemIDs = getAllItemIDs**()** //get item ID array

currentShoppingList = array**()** //Initialize as empty

**End** **Method**

//Method to test for successful database connectivity

**Method** Boolean testDatabase**()**

if **(**connection successful**)**

return true

else

return false

**End** **Method**

//Returns the top ten purchased items

**Method** String[] itemPopularityDensity**()**

int[] itemIDs = getAllItemIDs**()** //get item ID array

String[] occuranceCountArray = array**()** //array of items

foreach**(**itemsIDs as item**)**

int occuranceCount = //MySQL statement to count occurances of itemIDs

itemIDs[item] = occuranceCount //Add occurance count to array

sort**(**occuranceCountArray**)** //sort array from highest to lowest

int[] popularItemArray = array**()** //create empty output array

return array\_slice**(**occuranceCountArray, 0, 10**)** //return top 10 purchased items

**End** **Method**

//Method to return a list of recent purchases (within the last two weeks)

**Method** String[] recentPurchases**()**

int[] itemIDs = getAllItemIDs**()** //get item ID array

String[] recentPurchases = array**()** //initialize enmpty output array

foreach**(**itemIDs as item**)**

if **(**item purchased within last 14 days**)** //MySQL select statement

array\_push**(**recentPurchases, item**)** // add item to array

return recentPurchases

**End** **Method**

//Method to test each item in array for eligibility to be added to shopping list

**Method** Numeric generateList**(**itemIDs**)**

foreach**(**itemIDs as item**)**

//Construct array of previous purchase dates for item via MySQL

String[] itemPurchaseDateArray = getSortedPurchaseDateArray**(**item**)**

if **(**eligible**(**itemPurchaseDateArray**))**

//If item is eligible, add to current shopping list

array\_push**(**currentShoppingList, item**)**

return currentShoppingList

**End** **Method**

//Method to determine eligibility to be added to current shopping list

**Method** Boolean eligible**(**Numeric[] itemPurchaseDateArray**)**

if**(**count**(**dayIntervals**)** < 5**)**

return true

if **(**daySinceLastPurchase**(**itemPurchaseDateArray**)** >=

averagePurchaseInterval**(**itemPurchaseDateArray**)** -

stats\_standard\_deviation**(**getDayIntervalArray**(**itemPurchaseDateArray**))** - 5**)**

/\*

\* Add item to recommended purchase list if the number of days that

\* the item was last purchased is greater than the average purchase

\* interval minus the standard deviation of the purchase interval.

\* Subtracting a further five days helps to reduce shopping trip

\* intervals by looking further into the future.

\*/

return true

return false

**End** **Method**

//Helper method to calculate number of days since last purchase

Private **Method** Numeric daySinceLastPurchase**(**String[] itemPurchaseDateArray**)**

return date\_diff**(**today, itemPurchaseDateArray[0]**)**

**Method**

//Helper method to get average purchase interval

Private **Method** Numeric averagePurchaseInterval**()**

return averagePurchaseInterval**(**dayIntervals**)** / count**(**dayIntervals**)**;

**End** **Method**

//Helper method to retrieve a list of dates for current item, sorted from

//most to least recently

Private **Method** String[] getSortedPurchaseDateArray**(**itemID**)**

//Generate string of item purchase dates from database via MySQL

string[] getitemPurchaseDateArray = sort**(**getPurchaseDateArray**(**itemID**))**;

**End** **Method**

//Helper method to generate an array of differences between purchase dates

Private **Method** Numeric getDayIntervalArray**(**int[] dateArray**)**

dayIntervalArray = array**()**; //Number of days between each purchase date

for**(**i = 1; i < count**(**dateArray**)**; i++**)** {

int prevElement = i - 1;

//Calculate day interval between dates

int dayDifference = date\_diff**(**dateArray[prevElement], dateArray[i]**)**;

//Append the number of days between each date to the dayIntervalArray array

array\_push**(**dayIntervalArray, dayDifference**)**;

}

return dayIntervalArray;

**End** **Method**

//Helper method to determine the average purchase interval for an item

Private **Method** Numeric averagePurchaseInterval**(**int[] dateArray**)**

int sum = 0;

foreach**(**dateArray as value**)** {

sum += value

}

return sum

**End** **Method**

//Helper method to build array of dates for each item in the users database

Private **Method** String[] getPurchaseDateArray**(**int itemID**)**

outputArray = array**()**;

foreach**(**line in database**)** {

if**(**current line itemID == itemID**)**

array\_push**(**outputArray, purchaseDate**)**

return outputArray

**End** **Method**

//Helper Method to get a numeric array of all itemIDs in user database

**Method** Numeric[] getAllItemIDs**()**

return array**()** //MySQL statement getting all item IDs

**End** **Method**

**End** Class

1. **Unresolved Risks and Mitigation**

Risk 1: Password cracking

Possible Mitigation: Password encryption, when user types the password it is not seen visible.

Risk 2: Duplicate product entries within the database, skewing algorithm results.

Possible Mitigation: For the purposes of this project, the database will be preloaded with items that have been filtered of duplicate entries. Although potentially capable, the algorithm design would have to advance beyond the scope of this project to fully implement duplicate entry mitigation techniques.

Risk 3: Newly entered items with too few date entries can provide inaccurate purchase forecasting. This has the potential to render this app pointless if allowed to continue in such a state unaddressed, algorithmically.

Possible Mitigation: For newly entered products, the algorithm makes some assumptions until there is enough purchase history to take over the prediction scheme. The assumption made, for the purposes of this project, is a product purchase interval of five days. Therefore, items that fall under this assumption scheme will continue to populate on shopping lists - although the user will, of course, skip purchasing the item if it is not needed at the time – until there are enough (approximately five) dates in the database to begin predicting a purchase pattern.

**Notable Project Plan Adjustments**

From this point moving forward, the print-list and email functions as described within the project description, plan, and analysis documents will be deleted and no longer included in the intended project functionalities. These features, although likely included in a marketed product, were deemed to inject unjustified complexity to the project, considering that these functions didn’t truly further the overall core intent thereof. Henceforth, as mentioned, the inclusion of these features in any future documentation or developments will be discontinued.