

# Module Interface Specification for Grocery Spending Tracker

Team 1, JARS

Jason Nam

Allan Fang

Ryan Yeh

Sawyer Tang

July 18, 2024

# 1 Revision History

Date	Version	Notes
17/01/2024	1.0	Initial version of MIS
01/04/2024	1.1	Converted Syntax to proper format for definition of variables and their types and fixed multi-character variables to use proper text format instead of italics
02/04/2024	1.2	Specified Module Type for each modules
03/04/2024	1.3	Added definitions for Optional and Object <sup>n</sup> in Notations section, clarification for Section 7.4 assumption

## 2 Symbols, Abbreviations and Acronyms

See SRS Documentation at <https://github.com/r-yeh/grocery-spending-tracker/blob/master/docs/SRS/SRS.pdf>

---

symbol	description
SRS	Software Requirements Specification
MIS	Module Interface Specification
SKU	Stock Keeping Unit: ID of Product Used by Retailers
UPC	Universal Product Code: widely used barcode symbology for tracking trade items in stores
JWT	JSON Web Token

---

# Contents

<b>1</b>	<b>Revision History</b>	<b>i</b>
<b>2</b>	<b>Symbols, Abbreviations and Acronyms</b>	<b>ii</b>
<b>3</b>	<b>Introduction</b>	<b>1</b>
<b>4</b>	<b>Notation</b>	<b>1</b>
<b>5</b>	<b>Module Decomposition</b>	<b>2</b>
<b>6</b>	<b>MIS of Receipt Vision Module</b>	<b>3</b>
6.1	Module . . . . .	3
6.1.1	Module Type . . . . .	3
6.2	Uses . . . . .	3
6.3	Syntax . . . . .	3
6.3.1	Exported Constants . . . . .	3
6.3.2	Exported Types . . . . .	3
6.3.3	Exported Access Programs . . . . .	3
6.4	Semantics . . . . .	3
6.4.1	State Variables . . . . .	3
6.4.2	Environment Variables . . . . .	4
6.4.3	Assumptions . . . . .	4
6.4.4	Access Routine Semantics . . . . .	4
6.4.5	Local Functions . . . . .	4
<b>7</b>	<b>MIS of Receipt Extraction Module</b>	<b>5</b>
7.1	Module . . . . .	5
7.1.1	Module Type . . . . .	5
7.2	Uses . . . . .	5
7.3	Syntax . . . . .	5
7.3.1	Exported Constants . . . . .	5
7.3.2	Exported Types . . . . .	5
7.3.3	Exported Access Programs . . . . .	6
7.4	Semantics . . . . .	7
7.4.1	State Variables . . . . .	7
7.4.2	Environment Variables . . . . .	7
7.4.3	Assumptions . . . . .	7
7.4.4	Access Routine Semantics . . . . .	7
7.4.5	Local Functions . . . . .	8

<b>8</b>	<b>MIS of Location Management Module</b>	<b>9</b>
8.1	Module . . . . .	9
8.1.1	Module Type . . . . .	9
8.2	Uses . . . . .	9
8.3	Syntax . . . . .	9
8.3.1	Exported Constants . . . . .	9
8.3.2	Exported Types . . . . .	9
8.3.3	Exported Access Programs . . . . .	9
8.4	Semantics . . . . .	9
8.4.1	State Variables . . . . .	9
8.4.2	Environment Variables . . . . .	9
8.4.3	Assumptions . . . . .	10
8.4.4	Access Routine Semantics . . . . .	10
8.4.5	Local Functions . . . . .	10
<b>9</b>	<b>MIS of User Analytics Module</b>	<b>11</b>
9.1	Module . . . . .	11
9.1.1	Module Type . . . . .	11
9.2	Uses . . . . .	11
9.3	Syntax . . . . .	11
9.3.1	Exported Constants . . . . .	11
9.3.2	Exported Types . . . . .	11
9.3.3	Exported Access Programs . . . . .	11
9.4	Semantics . . . . .	12
9.4.1	State Variables . . . . .	12
9.4.2	Environment Variables . . . . .	12
9.4.3	Assumptions . . . . .	12
9.4.4	Access Routine Semantics . . . . .	12
9.4.5	Local Functions . . . . .	13
<b>10</b>	<b>MIS of Users Module</b>	<b>14</b>
10.1	Module . . . . .	14
10.1.1	Module Type . . . . .	14
10.2	Uses . . . . .	14
10.3	Syntax . . . . .	14
10.3.1	Exported Constants . . . . .	14
10.3.2	Exported Types . . . . .	14
10.3.3	Exported Access Programs . . . . .	15
10.4	Semantics . . . . .	15
10.4.1	State Variables . . . . .	15
10.4.2	Environment Variables . . . . .	15
10.4.3	Assumptions . . . . .	15
10.4.4	Access Routine Semantics . . . . .	15

10.4.5	Local Functions . . . . .	16
<b>11</b>	<b>MIS of Authentication Module</b>	<b>18</b>
11.1	Module . . . . .	18
11.1.1	Module Type . . . . .	18
11.2	Uses . . . . .	18
11.3	Syntax . . . . .	18
11.3.1	Exported Constants . . . . .	18
11.3.2	Exported Types . . . . .	18
11.3.3	Exported Access Programs . . . . .	18
11.4	Semantics . . . . .	18
11.4.1	State Variables . . . . .	18
11.4.2	Environment Variables . . . . .	18
11.4.3	Assumptions . . . . .	19
11.4.4	Access Routine Semantics . . . . .	19
11.4.5	Local Functions . . . . .	19
<b>12</b>	<b>MIS of Recommendation Module</b>	<b>20</b>
12.1	Module . . . . .	20
12.1.1	Module Type . . . . .	20
12.2	Uses . . . . .	20
12.3	Syntax . . . . .	20
12.3.1	Exported Constants . . . . .	20
12.3.2	Exported Types . . . . .	20
12.3.3	Exported Access Programs . . . . .	20
12.4	Semantics . . . . .	20
12.4.1	State Variables . . . . .	20
12.4.2	Environment Variables . . . . .	20
12.4.3	Assumptions . . . . .	21
12.4.4	Access Routine Semantics . . . . .	21
12.4.5	Local Functions . . . . .	21
<b>13</b>	<b>MIS of Classification Module</b>	<b>22</b>
13.1	Module . . . . .	22
13.1.1	Module Type . . . . .	22
13.2	Uses . . . . .	22
13.3	Syntax . . . . .	22
13.3.1	Exported Constants . . . . .	22
13.3.2	Exported Types . . . . .	22
13.3.3	Exported Access Programs . . . . .	23
13.4	Semantics . . . . .	23
13.4.1	State Variables . . . . .	23
13.4.2	Environment Variables . . . . .	23

13.4.3	Assumptions . . . . .	23
13.4.4	Access Routine Semantics . . . . .	23
13.4.5	Local Functions . . . . .	24
<b>14</b>	<b>MIS of Database Driver Module</b>	<b>25</b>
14.1	Module . . . . .	25
14.1.1	Module Type . . . . .	25
14.2	Uses . . . . .	25
14.3	Syntax . . . . .	25
14.3.1	Exported Constants . . . . .	25
14.3.2	Exported Types . . . . .	25
14.3.3	Exported Access Programs . . . . .	25
14.4	Semantics . . . . .	25
14.4.1	State Variables . . . . .	25
14.4.2	Environment Variables . . . . .	25
14.4.3	Assumptions . . . . .	26
14.4.4	Access Routine Semantics . . . . .	26
14.4.5	Local Functions . . . . .	26
<b>15</b>	<b>Appendix</b>	<b>28</b>
15.1	Database Schema and Tables . . . . .	28
15.1.1	rawItems . . . . .	28
15.1.2	classifiedItems . . . . .	28
15.1.3	transaction . . . . .	29
15.1.4	users . . . . .	29
15.1.5	goals . . . . .	29

### 3 Introduction

The following document details the Module Interface Specifications for the Grocery Spending Tracker. The Grocery Spending Tracker is a mobile application designed to help households face the ever rising cost of groceries in Canada. To accomplish this, users can easily scan their grocery receipts with their device camera allowing the application to record and track user spending. Analytics are then produced with this data to help users better visualize their spending habits. They can also set budgets and spending goals for themselves allowing the application to suggest cheaper alternatives based on recently purchased grocery items and location preferences. Overall, these features should help users stay more informed on their grocery spending leading to smarter financial decisions and reduced spending over time.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at <https://github.com/r-yeh/grocery-spending-tracker>.

### 4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol  $:=$  is used for a multiple assignment statement and conditional rules follow the form  $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$ .

The following table summarizes the primitive data types used by Grocery Spending Tracker.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in $(-\infty, \infty)$
natural number	$\mathbb{N}$	a number without a fractional component in $[1, \infty)$
real	$\mathbb{R}$	any number in $(-\infty, \infty)$
real positive	$\mathbb{R}^+$	any number in $[0, \infty)$
boolean	$\mathbb{B}$	a truth value, resolving to true or false

The specification of Grocery Spending Tracker uses some derived data types: lists, strings, maps and tuples. Lists are data types that store an array of a set data type, formatted as List<Object>. Strings are sequences of characters. Maps are a list of key value pairs of two identical or different data types formatted as Map<Key, Value>. Tuples contain a series of values, potentially of different types. Furthermore, some exported data types using a combination of the above primitive and derived data types will be defined as required



in certain MIS sections. In addition, Grocery Spending Tracker uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification. Some additional notations that are used in this specification are `Optional<Object>` and `Objectn`. `Optional<Object>` means the field is of type `Object` but it can be left empty. In other words, a user can choose whether to include the field or not. `Objectn` means that the field is comprised of  $n$  instances of type `Object`. As an example, if  $n = 2$ , then the field would consist of 2 `Objects`.

## 5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	Level 2
Hardware-Hiding	
	Receipt Vision Module (OCR)
	Receipt Extraction Module
	Location Management Module
Behaviour-Hiding	User Analytics Module
	Users Module
	Authentication Module
Software Decision	Recommendation Engine
	Classification Engine
	Database Driver Module

Table 1: Module Hierarchy

## 6 MIS of Receipt Vision Module

The Receipt Vision Module is responsible for providing methods and logic related to image capture of the receipt. This includes responsibilities like handling device permissions and camera access.

### 6.1 Module

receiptVision

#### 6.1.1 Module Type

Library Module

### 6.2 Uses

None

### 6.3 Syntax

#### 6.3.1 Exported Constants

None

#### 6.3.2 Exported Types

None

#### 6.3.3 Exported Access Programs

Name	In	Out	Exceptions
requestCameraPermission	-	-	-
didChangeAppLifeCycle	appLifecycle	-	-
scanReceipt	-	String	noPermissionException

### 6.4 Semantics

#### 6.4.1 State Variables

appLifecycle : **App Lifecycle**  $\rightarrow$  Tracks module activity on the user's device.

isPermissionGranted :  $\mathbb{B} \rightarrow$  Stores camera permission status.

selectedCamera : **Device Camera**  $\rightarrow$  Stores data on which camera to use.

### 6.4.2 Environment Variables

User Camera : **Device Camera**

User Display : **2D Sequence of Pixels**

### 6.4.3 Assumptions

- Application will always be used on a device with a camera.

### 6.4.4 Access Routine Semantics

requestCameraPermission():

- transition: Request camera permission and update *isPermissionGranted* accordingly.
- output: none
- exception: none

didChangeAppLifeCycle(*appLifecycle*):

- transition: ( $\neg isPermissionGranted$ )  $\Rightarrow$  no action is performed |  
( $isPermissionGranted \wedge appLifecycle = active$ )  $\Rightarrow$  Turn on device camera |  
( $isPermissionGranted \wedge appLifecycle = inactive$ )  $\Rightarrow$  Turn off device camera
- output: none
- exception: none

scanReceipt():

- transition: Display camera preview and interface for image capture.
- output: *out* := *isPermissionGranted*  $\Rightarrow$  take picture with camera and parse text
- exception: *exc* := ( $\neg isPermissionGranted$ )  $\Rightarrow$  noPermissionException

### 6.4.5 Local Functions

initializeCamera(availableCameras):

- transition: Let *findBackFacingCamera* =  $\forall cameras (\exists camera \in cameras : camera.direction = back)$   
( $\neg isPermissionGranted$ )  $\Rightarrow$  no action is performed |  
*isPermissionGranted*  $\Rightarrow$  *findBackFacingCamera*(*availableCameras*)  $\Rightarrow$   
*selectedCamera* = *camera*

## 7 MIS of Receipt Extraction Module

The Receipt Extraction Module contains methods related to extracting relevant data such as grocery items, prices, and date of transaction from a parsed receipt.

### 7.1 Module

receiptExtraction

#### 7.1.1 Module Type

Library Module

### 7.2 Uses

Database Driver Module (Section 14)

### 7.3 Syntax

#### 7.3.1 Exported Constants

None

#### 7.3.2 Exported Types

**Item**

Output Name	Output Type	Description
itemKey	String	SKU of the purchased item
itemDesc	String	Name of the product on the receipt
price	$\mathbb{R}$	Purchase price of the product
taxed	$\mathbb{B}$	Whether an item is taxed (Some groceries in Canada aren't taxed)

**UpdatedItem**

Output Name	Output Type	Description
itemKey	Optional<String>	SKU of the purchased item
itemDesc	Optional<String>	Name of the product on the receipt
price	Optional< $\mathbb{R}$ >	Purchase price of the product
taxed	Optional< $\mathbb{B}$ >	Whether an item is taxed (Some groceries in Canada aren't taxed)

### GroceryTrip

Output Name	Output Type	Description
userId	String	ID of the user
dateTime	$\mathbb{N}$	Date and time of transaction as Unix timestamp
location	String	Address of the grocery store location
items	List<Item>	List of purchased grocery items from receipt
subtotal	$\mathbb{R}$	Total spent on grocery trip before tax
total	$\mathbb{R}$	Total spent on grocery trip
tripDesc	Optional<String>	Notes made by the user

### UpdatedGroceryTrip

Output Name	Output Type	Description
dateTime	Optional< $\mathbb{N}$ >	Date and time of transaction as Unix timestamp
location	Optional<String>	Address of the grocery store location
items	Optional<List<UpdatedItem>>	List of purchased grocery items from receipt
subtotal	Optional< $\mathbb{R}$ >	Total spent on grocery trip before tax
total	Optional< $\mathbb{R}$ >	Total spent on grocery trip
tripDesc	Optional<String>	Notes made by the user

#### 7.3.3 Exported Access Programs

This module takes as input User as described in Section 10.

Name	In	Out	Exceptions
createGroceryTrip	User, String	GroceryTrip	-
updateGroceryTrip	UpdatedGroceryTrip	GroceryTrip	-
createItem	String	Item	-
updateItem	UpdatedItem	Item	-

## 7.4 Semantics

### 7.4.1 State Variables

None

### 7.4.2 Environment Variables

User Display : **2D Sequence of Pixels**

### 7.4.3 Assumptions

- *createItem* will always be working with strings of the format "*itemKey itemDesc Optional<H> price*" where "H" is a character on the receipt indicating whether an item is taxed or not.

### 7.4.4 Access Routine Semantics

*createGroceryTrip*(*user*, *receiptData*):

- transition: Extracts *GroceryTrip* data, redirects user to new page where confirmation of extracted data can be performed.
- output: *out* := new *GroceryTrip*(*user.userId*, toUnixTimestamp(getDateTime(*receiptData*)), getLocation(*receiptData*), getItems(*receiptData*), getSubtotal(*receiptData*), getTotal(*receiptData*), "" )
- exception: none

*updateGroceryTrip*(*updatedTrip*):

- transition: Updates *GroceryTrip*, reloads confirmation page.
- output: *out* := Update all fields of the *GroceryTrip* object with non-empty fields of *updatedTrip*.
- exception: none

*createItem*(*itemString*):

- transition: none
- output: Let *itemComponents* = *itemString*.split(" ")  
*out* :=  $\exists taxed \in itemString : taxed \in L(H) \Rightarrow$   
new Item(*itemComponents*[0], *itemComponents*[1], *itemComponents*[3], True) |  
 $\neg(\exists taxed \in itemString : taxed \in L(H)) \Rightarrow$   
new Item(*itemComponents*[0], *itemComponents*[1], *itemComponents*[2], False)

- exception: none

updateItem(*updatedItem*):

- transition: none
- output: *out* := Update all fields of the *Item* object with non-empty fields of *updatedItem*.
- exception: none

#### 7.4.5 Local Functions

getDateTime(*receiptData*):

- output: Let  $A = \{x : 0 \leq x \leq 9\}, B = \{x : 0 \leq x \leq 2\}, C = \{x : 0 \leq x \leq 5\}$   
 Let  $containsDateTime = \exists data \in receiptData : data$   
 $\in L((A|AA)(/|-)(A|AA)(/|-)(AAAA) (BA):(CA):(CA))$   
 $out := containsDateTime(receiptData) \Rightarrow data| \neg containsDateTime(receiptData) \Rightarrow null$

toUnixTimestamp(*dateTime*):

- output: takes String *dateTime* and converts it to a Unix timestamp.

getLocation(*receiptData*):

- output: Let  $A = \{x : 0 \leq x \leq 9\}, B = \{a, b, \dots, y, z\} \cup \{A, B, \dots, Y, Z\} \cup \{ \}$   
 Let  $containsLocation = \exists data \in receiptData : data \in L(A^* B^*)$   
 $out := containsLocation(receiptData) \Rightarrow data| \neg containsLocation(receiptData) \Rightarrow null$

getItems(*receiptData*):

- output: Let  $A = \{x : 0 \leq x \leq 9\}, B = \{a, b, \dots, y, z\} \cup \{A, B, \dots, Y, Z\} \cup \{ \}$   
 Let  $getItems = \exists data : data \in L(B^* (\$|\epsilon)A^*.AA), itemList = []$   
 $out := itemList \text{ s.t. } \forall text (getItems(text) \wedge text \in receiptData) : itemList.append(createItem(text))$

getSubtotal(*receiptData*):

- output: Let  $A = \{x : 0 \leq x \leq 9\}$   
 Let  $containsSubtotal = \exists data \in receiptData : data \in L((SUBTOTAL|Subtotal) \$A^*.AA)$   
 $out := containsSubtotal(receiptData) \Rightarrow data| \neg containsSubtotal(receiptData) \Rightarrow null$

getTotal(*receiptData*):

- output: Let  $A = \{x : 0 \leq x \leq 9\}$   
 Let  $containsTotal = \exists data \in receiptData : data \in L((TOTAL|Total) \$A^*.AA)$   
 $out := containsTotal(receiptData) \Rightarrow data| \neg containsTotal(receiptData) \Rightarrow null$

## 8 MIS of Location Management Module

The Location Management Module manages user location data and outputs relevant resources based the available user location data.

### 8.1 Module

locationManagement

#### 8.1.1 Module Type

Library Module

### 8.2 Uses

Database Driver Module (Section 14)

### 8.3 Syntax

#### 8.3.1 Exported Constants

None

#### 8.3.2 Exported Types

None

#### 8.3.3 Exported Access Programs

Name	In	Out	Exceptions
getLastLocation	-	String	noLocationSavedException
setLastLocation	String	-	invalidLocationException
getNearbyStores	-	List<String>	-
setRadius	N	-	-

### 8.4 Semantics

#### 8.4.1 State Variables

None

#### 8.4.2 Environment Variables

None



### 8.4.3 Assumptions

None

### 8.4.4 Access Routine Semantics

getLastLocation():

- transition: none
- output:  $out := \neg location = null \Rightarrow location$ , where  $location$  is the user location stored in the database (Section 14)
- exception:  $exc := location = null \Rightarrow noLocationSavedException$ , where  $location$  is the user location stored in the database (Section 14)

setLastLocation( $location$ ):

- transition: Set user location in the database (Section 14) to  $location$
- output: none
- exception:  $exc := invalidLocationException$ , if  $location$  is not a valid location in the places dependency

getNearbyStores():

- transition: none
- output:  $out := locations$ , where  $locations$  is a List of Strings representing relevant stores specified by the user's preferences stored in the database (Section 14)
- exception: none

setRadius( $R$ ):

- transition: Set user location radius in the database (Section 14) to  $R$
- output: none
- exception: none

### 8.4.5 Local Functions

None

## 9 MIS of User Analytics Module

The User Analytics Module provides analytics and feedback on user purchasing behaviour, offering insight on recent and future purchasing for individual users.

### 9.1 Module

userAnalytics

#### 9.1.1 Module Type

Library Module

### 9.2 Uses

Database Driver Module (Section 14)

### 9.3 Syntax

#### 9.3.1 Exported Constants

None

#### 9.3.2 Exported Types

**Item** as defined in Section 7

#### 9.3.3 Exported Access Programs

Name	In	Out	Exceptions
getPurchaseHistoryByTrip	$\mathbb{N}$	List<List<Item>>	-
getSpendingByCategory	$\mathbb{N}^2$	Map<String, List<Item>>	-
getSpendingByTime	$\mathbb{N}^2$	Map<String, $\mathbb{R}^+$ >	-
createBudgetGoals	$\mathbb{N}^3$ , String	-	-
updateBudgetGoals	$\mathbb{N}^4$ , String	-	-
deleteBudgetGoals	$\mathbb{N}$ , String	-	-
getBudgetGoals	-	List<( $\mathbb{N}^3$ , String)>	-

## 9.4 Semantics

### 9.4.1 State Variables

$pageSize : \mathbb{N} \rightarrow$  number of trips in a page of shopping trips

### 9.4.2 Environment Variables

**GroceryTrip** (Section 7)

### 9.4.3 Assumptions

None

### 9.4.4 Access Routine Semantics

$getPurchaseHistoryByTrip(page)$ :

- transition: none
- output:  $out := [GroceryTrip_1, GroceryTrip_2, \dots, GroceryTrip_{pageSize}]$  s.t.  $GroceryTrip_i.dateTime > GroceryTrip_{i+1}.dateTime$ , where GroceryTrip are extracted from (Section 14)
- exception: none

$getSpendingByCategory(startDate, endDate)$ :

- transition: none
- output:  $out := spendingList$ , where  $spendingList$  is a List of user purchases in the time period specified by  $startDate$  and  $endDate$  grouped by product category stored in the database (Section 14)
- exception: none

$setSpendingByTime(startDate, endDate)$ :

- transition: none
- output:  $out := spendingList$ , where  $spendingList$  is a List of user spending amount by day within the specified  $startDate$  and  $endDate$  stored in the database (Section 14)
- exception: none

$createBudgetGoals(startDate, endDate, budget, category)$ :

- transition: Create budget goal for the user in the Goals table of the database (Section 14)

- output: none
- exception: none

updateBudgetGoals(*budgetId*, *startDate*, *endDate*, *budget*, *category*):

- transition: Update budget goal for the user in the Goals table of the database (Section [14](#))
- output: none
- exception: none

deleteBudgetGoals(*budgetId*):

- transition: Delete budget goal for the user by budget goal ID (*budgetId*) in the Goals table of the database (Section [14](#))
- output: none
- exception: none

getBudgetGoals():

- transition: none
- output: *out* := *budgetGoals* Where *budgetGoals* is the List of budget goals of the user in the Goals table of the database (Section [14](#))
- exception: none

#### 9.4.5 Local Functions

None

## 10 MIS of Users Module

The Users Module exists to handle CRUD operations for operations related to users and their grocery trips.

### 10.1 Module

users

#### 10.1.1 Module Type

Library Module

### 10.2 Uses

Database Driver Module (Section 14)

### 10.3 Syntax

#### 10.3.1 Exported Constants

None

#### 10.3.2 Exported Types

**Item** as defined in Section 7

**GroceryTrip** as defined in Section 7

**User**

Output Name	Output Type	Description
userId	String	Unique user id number
email	String	Email of user
password	String	Password of user (Encrypted)
firstName	String	First name of User
lastName	String	Last name of User
location	String	Location of user's home-base. (Coordinates)

### 10.3.3 Exported Access Programs

Name	In	Out	Exceptions
newUser	User	-	userExistsException
getUser	String	User	userDneException
updateUser	User	-	userDneException
deleteUser	String	-	userDneException
addTrip	GroceryTrip	-	userDneException
getTrip	$\mathbb{N}$	GroceryTrip	-

## 10.4 Semantics

### 10.4.1 State Variables

None

### 10.4.2 Environment Variables

None

### 10.4.3 Assumptions

None

### 10.4.4 Access Routine Semantics

newUser(*user*):

- transition:  $\neg userExists(user.userId) \Rightarrow addUserInDB(user)$
- output: none
- exception  $exc := userExists(user.userId) \Rightarrow userExistsException$

getUser(*userId*):

- transition: none
- output:  $out := userExists(user.userId) \Rightarrow getUserInDB(userId)$
- exception  $exc := \neg userExists(user.userId) \Rightarrow userDneException$

updateUser(*user*):

- transition:  $userExists(user.userId) \Rightarrow updateUser(user)$
- output: none
- exception  $exc := \neg userExists(user.userId) \Rightarrow userDneException$

deleteUser(*user*):

- transition:  $userExists(user.userId) \Rightarrow deleteUserInDB(user)$
- output: none
- exception  $exc := \neg userExists(user.userId) \Rightarrow userDneException$

addTrip(*groceryTrip*):

- transition:  $userExists(groceryTrip.userId) \Rightarrow addTripInDB(groceryTrip)$
- output: none
- exception  $exc := \neg userExists(user.userId) \Rightarrow userDneException$

getTrip(*dateTime*):

- transition: none
- output:  $out := getTripFromDateTime(dateTime)$

#### 10.4.5 Local Functions

addUserInDB(*user*)

- transition: Add tuple to User database (Section 14) using fields from *user*.

getUserInDB(*userId*)

- output: Query User database (Section 14) where *userId* and map to fields in *user*. Output *user*.

updateUserInDB(*user*)

- output: Replace tuple with key *user.userId* in User database (Section 14) with *user* by mapping fields.

deleteUserInDB(*user*)

- transition: Remove tuple to User database (Section 14) using fields from primary key *user.userId*.

*userExists(groceryTrip.userId)*

- output: Query User database (Section 14) and output True if a tuple with *userId = groceryTrip.userId* exists in the table.

*addTripInDB(groceryTrip)*

- transition: Add tuple to Trip database (Section 14) using fields from *groceryTrip*.

*getTripFromDateTime(dateTime)*

- output: Query Trip database (Section 14) and output the *GroceryTrip* tuple that is closest to *dateTime*.



## 11 MIS of Authentication Module

The Authentication module acts to "log-in" the user by providing the client with a JWT token.

### 11.1 Module

authentication

#### 11.1.1 Module Type

Library Module

### 11.2 Uses

Database Driver Module (Section 14)

### 11.3 Syntax

#### 11.3.1 Exported Constants

None

#### 11.3.2 Exported Types

None

#### 11.3.3 Exported Access Programs

Name	In	Out	Exceptions
getToken	String <sup>2</sup>	String	invalidCredentialsException
verifyToken	String	String	invalidTokenException

### 11.4 Semantics

#### 11.4.1 State Variables

None

#### 11.4.2 Environment Variables

PrivateKey : **String** → used for JSON Web Tokens (JWT)

PublicKey : **String** → used for JSON Web Tokens (JWT)

### 11.4.3 Assumptions

None

### 11.4.4 Access Routine Semantics

$\text{getToken}(\text{email}, \text{passwordEncrypted})$ :

- transition:  $\text{signedJwtToken}$  produced using  $\text{PrivateKey}$
- output:  $\text{out} := \text{verifyCredentials}(\text{email}, \text{passwordEncrypted}) \Rightarrow \text{signedJwtToken}$
- exception  $\text{exc} := \neg \text{verifyCredentials}(\text{email}, \text{passwordEncrypted}) \Rightarrow \text{invalidCredentialsException}$

$\text{verifyJwtToken}(\text{jwtToken})$ :

- transition: none
- output:  $\text{out} := \text{verifyToken}(\text{jwtToken})$
- exception:  $\text{exc} := \neg \text{verifyToken}(\text{jwtToken}) \Rightarrow \text{invalidTokenException}$ , if the provided JWT token does not evaluate true in comparison to the  $\text{PublicKey}$

### 11.4.5 Local Functions

$\text{verifyCredentials}(\text{email}, \text{passwordEncrypted})$

- output:  $\text{out} := \text{True}$  if the password is deemed correct for the given email using the database (Section 14), else  $\text{False}$

$\text{verifyToken}(\text{jwtToken})$

- output:  $\text{out} := \text{True}$  if the JWT token is deemed correct using the  $\text{PublicKey}$  and  $\text{JWTLibrary}$ , else  $\text{False}$

## 12 MIS of Recommendation Module

Module for recommending user purchases.

### 12.1 Module

recommendationModule

#### 12.1.1 Module Type

Library Module

### 12.2 Uses

Database Driver Module (Section [14](#))

### 12.3 Syntax

#### 12.3.1 Exported Constants

None

#### 12.3.2 Exported Types

**Item** as defined in Section [7](#)

#### 12.3.3 Exported Access Programs

Name	In	Out	Exceptions
collectItemData	Item	-	InvalidDataFormatException
getCheaperAlternatives	String	List<String>	-
getRecommendationDetails	String	Item	-

### 12.4 Semantics

#### 12.4.1 State Variables

None

#### 12.4.2 Environment Variables

None

### 12.4.3 Assumptions

None

### 12.4.4 Access Routine Semantics

collectItemData(*Item*):

- transition: Updates the module's internal data structure with the new *Item* data.
- output: none
- exception: *exc* := *InvalidDataFormatException*, if *Item* does not adhere to the expected format.

getCheaperAlternatives(*itemId*):

- transition: none
- output: *out* := *getCheaperAlternatives(itemId)*
- exception: none

getRecommendationDetails(*itemId*):

- transition: none
- output: *out* := *getItemDetails(itemId)*
- exception: none

### 12.4.5 Local Functions

getCheaperAlternatives(*itemId*):

- output: *out* := list of item IDs that are cheaper alternatives to the item identified by, *itemId*

getItemDetails(*itemId*):

- output: *out* := detailed information of the item identified by, *itemId*

## 13 MIS of Classification Module

The Classification Module is responsible for interpreting and identifying recognized items from receipt data.

### 13.1 Module

classificationModule

#### 13.1.1 Module Type

Library Module

### 13.2 Uses

Database Driver Module (Section [14](#))

### 13.3 Syntax

#### 13.3.1 Exported Constants

None

#### 13.3.2 Exported Types

**ClassifiedItem**

Column Name	Type	Description
itemId	String	Unique id of this instance of this item
productKey	String	Universal item identifier (UPC)
dateTime	$\mathbb{N}$	Date and time of purchase
price	$\mathbb{R}$	Price of item
location	String	Location of purchase
userId	String	ID of user
itemDesc	String	Item description
isTaxed	$\mathbb{B}$	Describes if the item is taxed
category	String	Describes the product category the item belongs to

### 13.3.3 Exported Access Programs

This module uses *Item* as an input, as described in Section 7

Name	In	Out	Exceptions
setItemData	Item	-	invalidDataException
getClassifiedItem	String	ClassifiedItem	-

## 13.4 Semantics

### 13.4.1 State Variables

None

### 13.4.2 Environment Variables

None

### 13.4.3 Assumptions

None

### 13.4.4 Access Routine Semantics

setItemData(*Item*):

- transition: Set Item data in the database (Section 14) to *Classified Item*
- output: none
- exception: *exc* := *invalidDataException*, if *Item* does not adhere to expected format.

getClassifiedItem(*itemId*):

- transition: none
- output: *out* := *itemExists(itemId)*  $\Rightarrow$  *getClassifiedItemInDB(itemId)*
- exception: none

### 13.4.5 Local Functions

`itemExists(itemId):`

- output: *out* := (*itemId* is present in the database)

`getClassifiedItemInDB(itemId):`

- output: *out* := *itemExists(itemId)*  $\Rightarrow$  *retrieveClassifiedItem(itemId)*
- exception: *exc* := *itemNotFoundException*, if *itemId* does not exist in the database.

`identifyItemUsingModel(inputItemData):`

- output: *out* := *classifyUsingModel(inputItemData)*
- exception: *exc* := *modelIdentificationException*, if the classification model fails to identify the item.

`classifyUsingModel(inputItemData):`

- output: *out* := the classification result of the model for, *inputItemData*
- exception: *exc* := *classificationException*, if the model encounters an error during the classification process.

`findMatchingProduct(inputItemData):`

- output: *out* := product details from the database that best match, *inputItemData*
- exception: *exc* := *noProductFoundException*, if no product in the database matches *inputItemData*.

## 14 MIS of Database Driver Module

Module for hiding database implementation. Database schema and tables are detailed in Section [15.1](#)

### 14.1 Module

dbModule

#### 14.1.1 Module Type

Library Module

### 14.2 Uses

None

### 14.3 Syntax

#### 14.3.1 Exported Constants

None

#### 14.3.2 Exported Types

None

#### 14.3.3 Exported Access Programs

Name	In	Out	Exceptions
insert	String <sup>2</sup>	-	-
update	String <sup>3</sup>	-	-
delete	String <sup>2</sup>	-	-
select	String <sup>3</sup>	String	-

### 14.4 Semantics

#### 14.4.1 State Variables

None

#### 14.4.2 Environment Variables

db : Relational Database



### 14.4.3 Assumptions

- The schema is set up as described in Section 15.1
- The connection to the database is established with necessary permissions.

### 14.4.4 Access Routine Semantics

$\text{insert}(\text{tableName}, \text{values})$ :

- transition: the target table in  $db$  specified by  $\text{tableName}$  adds  $\text{values}$ .
- output: none
- exception: none

$\text{update}(\text{tableName}, \text{values}, \text{conditions})$ :

- transition: the target items in  $db$  specified by  $\text{tableName}$  and  $\text{conditions}$  is updated with  $\text{values}$ .
- output: none
- exception: none

$\text{delete}(\text{tableName}, \text{conditions})$ :

- transition: the target items in  $db$  specified by  $\text{tableName}$  and  $\text{conditions}$  are deleted.
- output: none
- exception: none

$\text{select}(\text{fields}, \text{tableName}, \text{conditions})$ :

- transition: none
- output:  $\text{out} := \text{data}$ , where  $\text{data}$  describes the fields specified by  $\text{fields}$  of the target items specified by  $\text{tableName}$  and  $\text{conditions}$  in  $db$ .
- exception: none

### 14.4.5 Local Functions

None

## References

- Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.
- Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL <http://citeseer.ist.psu.edu/428727.html>.

## 15 Appendix

### 15.1 Database Schema and Tables

#### 15.1.1 rawItems

Column Name	Type	Description
itemId	String	PRIMARY KEY unique id of this instance of this item
itemKey	String	Item identifier specific to the retailer (usually SKU)
dateTime	$\mathbb{N}$	Date and time of purchase
price	$\mathbb{R}$	Price of item
location	String	Location of purchase
userId	String	FOREIGN KEY ID of user
itemDesc	String	Item description
isTaxed	$\mathbb{B}$	Describes if the item is taxed

#### 15.1.2 classifiedItems

Column Name	Type	Description
itemId	String	PRIMARY KEY unique id of this instance of this item
productKey	String	Universal item identifier (UPC)
dateTime	$\mathbb{N}$	Date and time of purchase
price	$\mathbb{R}$	Price of item
location	String	Location of purchase
userId	String	FOREIGN KEY ID of user
transactionId	String	FOREIGN KEY ID of transaction
itemDesc	String	Item description
isTaxed	$\mathbb{B}$	Describes if the item is taxed
category	String	Describes the product category the item belongs to

### 15.1.3 transaction

Column Name	Type	Description
transactionId	String	PRIMARY KEY ID of transaction
userId	String	FOREIGN KEY ID of user
dateTime	$\mathbb{N}$	Date and time of transaction
location	String	Location of purchase
transactionDesc	String	Transaction description

### 15.1.4 users

Column Name	Type	Description
userId	String	PRIMARY KEY unique id assigned to the user
email	String	User email
password	String	Encrypted user password
firstName	String	User first name
lastName	String	User last name
location	String	Location of the user, usually describes the user's home

### 15.1.5 goals

Column Name	Type	Description
goalId	$\mathbb{N}$	PRIMARY KEY unique id assigned to the goal
userId	String	FOREIGN KEY ID of user
startDate	$\mathbb{N}$	Start date of the goal
endDate	$\mathbb{N}$	End date of the goal
budget	$\mathbb{Z}$	Money allotted for the goal
category	String	The product category of relevant to the goal