Team: Imposters 4156 T5

Name	Yumeng Luo	Kevin Hwang	Sang Jun Chun	Shengtan Mao
UNI	yl4655	jch2169	sc4658	sm4954

NOTE: We are currently still under development for extra features. Commits after “fc70882ad97a06b12d305d492b38c3525d715fab” (Dec 3rd) are done after the completion of this document. For coverage and build report, please refer to the link given in part 3&4.

Part 1:

User Stories

USId1. (NearBy) As a user who **wants to purchase items nearby**, I want to be able to **search for items near me and see where they are on a map**, so that I know **where I can purchase cheaper items I need**. My conditions of satisfaction are :

a. The tool correctly identifies my current location
b. The items searched are near me
c. The tool correctly represents the searched items’s price and location on the map

USId2. (Alternative) As **a user looking for cheaper alternatives or closer alternatives or specific items but in different locations**, I want to **specify what type of alternatives I am looking for** so that **I can purchase better alternative items**. My conditions of satisfaction are:

a. I can customize the type of alternatives Im looking for
b. The tool proposes the alternatives satisfying my need

USId3. (Saving) As **a user with a tight monthly budget**, I want to **know and choose cheaper alternatives for daily expenses** so that **I can stay under budget and save money**. My condition of satisfactions are:

a. The tool can find information about item I want
b. The tool proposes a cheaper option
c. I can choose cheaper options
d. The tool correctly records my savings

Part 2:

Link to test suite:
https://github.com/yumeng-luo/COMS4156/tree/main/src/test/java/savings/tracker

Class Name: Target API Helper Methods

Method name: getZip()

1. Input: double lat, double lon

2. Equivalence Classes:

#	Test Data		Expected Outcome	Test Cases: *Implemented is highlighted in green*
	Latitude	Longitude		

1) **Valid** Input partition with all valid inputs	[-90, 90]	[-180, 180]	Accept	{LAT: 0, LON: 0}
				{LAT: 37.4215 LON: -122.0837}
				{LAT: -80, LON: 150}

2) **Invalid** Input partition with all values below the lower limit	(-inf, -90)	(-inf, -180)	Return -1	{LAT: -91, LON: 0}
				{LAT: 0, LON: -181}
				{LAT: -91, LON: -181}

3) **Invalid** Input partition with all values above the lower limit	(90, inf)	(180, inf)	Return -1	{LAT: 91, LON: 0}
				{LAT: 0, LON: 181}
				{LAT: 9, LON: 181}

3. Test names:
    a. Valid: validCoords()
    b. Invalid: invalidCoords()

Method name: getStoreIdList()

1. Input: int zip code
2. Equivalence Classes

#	Test Data	Expected Outcome	Test Cases: *Implemented is highlighted in green*
	Zip code		
1) **Valid** Input partition with all valid inputs	5 digit Positive integer	Accept	{Zipcode: 10025}
2) **Invalid** Input partition with all values below the lower limit	Non positive integer	Return null	{Zipcode: -5}

3) **Invalid** Input partition with digits more than expected number of digits	6+ digit integer	Return null	{Zipcode: 111111}

3. Test names:
    a. Valid: validCoords()
    b. Invalid: invalidCoords()

Method name: getSecStoreIdList()

1. Input: int zip code
2. Equivalence Classes

#	Test Data	Expected Outcome	Test Cases: *Implemented is highlighted in green*
	Zip code		
1) **Valid** Input partition with all valid inputs	5 digit Positive integer	Accept	{Zipcode: 10025}
2) **Invalid** Input partition with all values below the lower limit	Non positive integer	Return null	{Zipcode: -10}
3) **Invalid** Input partition with digits more than expected number of digits	6+ digit integer	Return null	{Zipcode: 111111}

3. Test names:
    a. Valid: validZipSecList()
    b. Invalid: invalidZipSecList()

Method name: getItem()

1. Input: int store id , string tcin
2. Equivalence Classes:

	Test Data	Expected	

#	Store id	tcin	Outcome	Test Cases: *Implemented is highlighted in green*
1) **Valid** Input partition with all valid inputs	Positive integer	8 digit string with only numbers	Accept	{store id: 911 tcin: 54191097}
2) **Invalid** Input partition with all values below the lower limit	Non positive integer	Non positive integer	Return null	{store id: -1263, tcin: -06}
3) **Invalid** Input partition with all values above the lower limit	Positive integer larger than 100000	9+ digit positive integer	Return null	{store id :999999999, tcin: 999999999}

3. Test names:
    a. Valid: validTcin()
    b. Invalid: invalidTcin()

Method name: getItemList()

1. Input: List<Store> storeList, List<Item> orgList
2. Equivalence Classes:

#	Test Data		Expected Outcome	Test Cases: *Implemented is highlighted in green*
	storeList	orgList		
1) **Valid** Input partition with all valid inputs	Contains at least 1 store	Contains at least 1 item	Accept	{storeList: <store 1, store 2, store 3> orgList: <item1, item3, item4>}
2) **Invalid** Input partition with all values below the lower limit	Null / empty	Null / empty	Return null	{storeList: <> orgList: <>}

3. Test names:
    a. Valid: getItemList()

b. Invalid: getEmptyItemList()

Method name: getTargetAlternatives()

1. Input: int zip, List<Item> orgList
2. Equivalence Classes:

#	Test Data		Expected Outcome	Test Cases: *Implemented is highlighted in green*
	zip	orgList		
1) **Valid** Input partition with all valid inputs	5 digit Positive integer	Contains at least 1 item	Accept	{zip: 10025 orgList: <item1>}
2) **Invalid** Input partition with all values below the lower limit	Non positive number	Null / empty	Return null	{zip: -10 orgList: <>}

3. Test names:
   a. Valid: getValidAlternativeList()
   b. Invalid: getInvalidAlternativeList()

Class Name: Wegman API Helper Methods

Method name: getItems()

1. Input: String name, double lat, double lon
2. Equivalence Classes:

#	Test Data			Expected Outcome	Test Cases: *Implemented is highlighted in green*
	name	lat	lon		
1) **Valid** Input partition with all valid inputs	Valid string with letters	[-90, 90]	[-180, 180]	Accept	{name: “whole milk” lat: 43.663 lon: -72.368}
2) **Invalid** Input partition with	String with purely	(-inf, -90)	(-inf, -180)	Return null	{name: “!!&[]” lat: -99 lon: -190}

all values below the lower limit	special characters				

3. Test names:
    a. Valid: testGetItems()
    b. Invalid: testInvalidGetItems()

<u>Method name: getDistance()</u>

1. double lat1, double lon1, double lat2, double lon2
2. Equivalence Classes:

#	Test Data				Expected Outcome	Test Cases: *Implemented is highlighted in green*
	lat1	lon1	lat2	lon2		
1) **Valid** Input partition with all valid inputs	[-90, 90]	[-180, 180]	[-90, 90]	[-180, 180]	Accept	{lat1: 43.663, lon1: -72.368, lat2: 37.7510, lon2: -97.8220}
2) **Invalid** Input partition with all values below the lower limit	(-inf, -90)	(-inf, -180)	(-inf, -90)	(-inf, -180)	Return -1	{lat1: -999 lon1: -999, lat2: -100, lon2: -200}

3. Test names:
    a. Valid: testDistanceCalc()
    b. Invalid: testInvalidDistanceCalc()

<u>Method name: getNearestStore()</u>

1. Input: double lat, double lon, String type
2. Equivalence Classes:

#	Test Data			Expected Outcome	Test Cases: *Implemented is highlighted in green*
	type	lat	lon		

1) **Valid** Input partition with all valid inputs	Valid string with letters	[-90, 90]	[-180, 180]	Accept	{type: “Wegmans Store” lat: 37.7510, lon: -97.8220}
2) **Invalid** Input partition with all values below the lower limit	String with purely special characters	(-inf, -90)	(-inf, -180)	Return null	{type: “!!!” lat: -99 lon: -999}

3. Test names:
    a. Valid: testGetNearestStore()
    b. Invalid: testGetInvalidNearestStore()

Method name: getSecNearestStore()

1. Input: double lat, double lon, String type
2. Equivalence Classes:

#	Test Data			Expected Outcome	Test Cases: *Implemented is highlighted in green*
	type	lat	lon		
1) **Valid** Input partition with all valid inputs	Valid string with letters	[-90, 90]	[-180, 180]	Accept	{type: “Wegmans Store” lat: 37.7510, lon: -97.8220}
2) **Invalid** Input partition with all values below the lower limit	String with purely special characters	(-inf, -90)	(-inf, -180)	Return null	{type: “!!!” lat: -99 lon: -999}

3. Test names:
    a. Valid: testGetSecondNearestStore()

b. Invalid: testGetInvalidSecNearestStore()

Method name: getAlternativeItems()

1. Input: String name, double lat, double lon, double initialPrice
2. Equivalence Classes:

#	Test Data				Expected Outcome	Test Cases: *Implemented is highlighted in green*
	name	lat	lon	initialPrice		
1) **Valid** Input partition with all valid inputs	Valid string with letters	[-90, 90]	[-180, 180]	Non negative double	Accept	{name: “whole milk" lat: 43.663, lon: -72.368 initialPrice: 20}
2) **Invalid** Input partition with all values below the lower limit	String with purely special characters	(-inf, -90)	(-inf, -180)	Negative values	Return null	{name: “!!!” lat: -99 lon: -999 initialPrice: -1}

3. Test names:
   a. Valid: testGetAlternatives()
   b. Invalid: testGetInvalidAlternatives()

Class Name: SendGrid API Helper Methods

Method name: Send()

1. Mail input

Mail (mail object)	Result	Test value
Not null	Accept	Empty mail object
Null	Return false	Null mail object

2. API key input

Key (string)	Result	Test value
Empty	Return false	“”
First two letters are ‘SG.’	Accept	“SG.38487ddsaf”
First two letters not ‘SG.’	Return false	“123343rgasd”
Long string	Return false	string.len() > 50

3. Test names:
    a. Valid: sendEmailTest, sendEmailInputNoNullTest, sendEmailAPIKeyValidTest
    b. Invalid: sendEmailInputNullTest, sendEmailAPIKeyEmptyTest, sendEmailAPIKeyInvalidTest, sendEmailAPIKeyLongTest

Method name: buildDynamicTemplate()

1. Email address input

Key (string)	Result	Test value
Empty	Return false	“”
Contains ‘@’ symbol	Accept (potentially)	“abc@abc.com”
Short string	Return false	string.len() ∈ [1,5)
Valid string size	Accept (potentially)	string.len() ∈ [5,254] “a@a.a” would be shortest possible
Long string	Return false	string.len() > 254

2. Test names:
    a. Valid: buildDynamicTemplateEmailValidTest
    b. Invalid: buildDynamicTemplateEmailEmptyTest, buildDynamicTemplateEmailShortTest, buildDynamicTemplateEmailLongTest

Method name: sendDynamicEmail()

1. Email address input

Key (string)	Result	Test value
Empty	Return false	“”
Contains ‘@’ symbol	Accept (potentially)	“abc@abc.com”
Short string	Return false	string.len() ∈ [1,5)
Valid string size	Accept (potentially)	string.len() ∈ [5,254] “a@a.a” would be shortest possible
Long string	Return false	string.len() > 254

2. Test names:
    a. Valid: sendDynamicEmailEmailValidTest
    b. Invalid: sendDynamicEmailEmailEmptyTest, sendDynamicEmailEmailShortTest, sendDynamicEmailEmailLongTest

Class Name: Controller

Method name: POST /search?item=NAME

1. Equivalence Classes:

Item name (NAME)	Result	Test value
Empty string	NULL	“”
Long string	NULL	string.len() > 50
Short string	NULL	string.len() ∈ (0,2]
Valid string	Accept	string.len() ∈ (2, 50]

2. Test names:
   a. Valid: postSelectIValidName
   b. Invalid: postSearchLongName

Method name: POST /select_item?item_number=NUMBER

1. Equivalence Classes:

Item number (NUMBER)	Result	Test value
Non-integer	NULL	String, char, etc.
Invalid int (too big)	NULL	int n > # of items returned*
Invalid int (negative)	NULL	int n < 0
Valid int	Accept	0 < n < # of items returned*

*We keep track of how many items we provide to users as search results.

2. Test names:
   a. Valid: postSelectItemValid
   b. Invalid: postSelectItemInvaid

Method name: POST /alternatives?CHEAPER=TRUE?CLOSER=TRUE?SAME=TRUE

1. Cheaper (CHEAPER) input

CHEAPER	Result	Test value
Boolean values (int)	Accept	CHEAPER == 0, 1

Non-boolean values	Return NULL	CHEAPER != 0, 1

2. Closer (CLOSER) input

CHEAPER	Result	Test value
Boolean values (int)	Accept	CHEAPER == 0, 1
Non-boolean values	Return NULL	CHEAPER != 0, 1

3. Same (SAME) input

CHEAPER	Result	Test value
Boolean values (int)	Accept	CHEAPER == 0, 1
Non-boolean values	Return NULL	CHEAPER != 0, 1

4. Test names:
   a. Valid: postalternativesValid
   b. Invalid: postalternativesInvalid

Method name: POST /select_purchase?upc=ITEM_NUMBER&lat=LATITUDE&lon=LONGITUDE

1. Equivalence Classes:

LATITUDE	LONGITUDE	Result	Test value

[-90, 90]	[-180, 180]	Accept	{LAT: 0, LON: 0}
			{LAT: -90, LON: -180}
			{LAT: 90, LON: -180}
			{LAT: -90, LON: 180}
			{LAT: 90, LON: 180}

(-inf, -90)	(-inf, -180)	Return -1	{LAT: -91, LON: 0}
			{LAT: 0, LON: -181}
			{LAT: -91, LON: -181}

(90, inf)	(180, inf)	Return -1	{LAT: 91, LON: 0}
			{LAT: 0, LON: 181}
			{LAT: 91, LON: 181}

2. Test names (Tested all boundary conditions):
   a. Valid:
      i. postSelectPurchaseValidBoundaryCondition1
         1. Input: ({LAT: 0, LON: 0})

ii. postSelectPurchaseValidBoundaryCondition2
    1. Input {LAT: -90, LON: -180}

iii. postSelectPurchaseValidBoundaryCondition3
    1. Input {LAT: 90, LON: -180}

iv. postSelectPurchaseValidBoundaryCondition4
    1. Input {LAT: -90, LON: 180}

v. postSelectPurchaseValidBoundaryCondition5
    1. Input {LAT: 90, LON: 180}

b. Invalid:

i. postSelectPurchaseInvalidBoundaryCondition1
    1. Input {LAT: -91, LON: 0}

ii. postSelectPurchaseInvalidBoundaryCondition2
    1. Input {LAT: 0, LON: -181}

iii. postSelectPurchaseInvalidBoundaryCondition3
    1. Input {LAT: -91, LON: -181}

iv. postSelectPurchaseInvalidBoundaryCondition4
    1. Input {LAT: 91, LON: 0}

v. postSelectPurchaseInvalidBoundaryCondition5
    1. Input {LAT: 0, LON: 181}

vi. postSelectPurchaseInvalidBoundaryCondition6
    1. Input {LAT: 91, LON: 181}

vii. postSelectPurchaseInvalidBoundaryCondition7
    1. Input {LAT: -900, LON: -900}

3. Boundary Conditions:

Test name	Condition	Test Data	Result
postSelectPurchaseValidBoundaryCondition1	Typical valid	{LAT: 0, LON: 0}	Code message 200 (accept)
postSelectPurchaseValidBoundaryCondition2	Boundary valid	{LAT: -90, LON: -180}	Code message 200 (accept)
postSelectPurchaseValidBoundaryCondition3	Boundary valid	{LAT: 90, LON: -180}	Code message 200 (accept)
postSelectPurchaseValidBoundaryCondition4	Boundary valid	{LAT: -90, LON: 180}	Code message 200 (accept)
postSelectPurchaseValidBoundaryCondition5	Boundary valid	{LAT: 90, LON: 180}	Code message 200 (accept)
postSelectPurchaseInvalidBoundaryCondition1	Boundary invalid	{LAT: -91, LON: 0}	Code message 400 (invalid)
postSelectPurchaseInvalidBoundaryCondition2	Boundary invalid	{LAT: 0, LON: -181}	Code message 400 (invalid)

postSelectPurchaseInvalidBoundaryCondition3	Boundary invalid	{LAT: -91, LON: -181}	Code message 400 (invalid)
postSelectPurchaseInvalidBoundaryCondition4	Boundary invalid	{LAT: 91, LON: 0}	Code message 400 (invalid)
postSelectPurchaseInvalidBoundaryCondition5	Boundary invalid	{LAT: 0, LON: 181}	Code message 400 (invalid)
postSelectPurchaseInvalidBoundaryCondition6	Boundary invalid	{LAT: 91, LON: 181}	Code message 400 (invalid)
postSelectPurchaseInvalidBoundaryCondition7	Typical invalid	{LAT: -900, LON: -900}	Code message 400 (invalid)

Part 3:

Branch Coverage: 74.39%
Uncovered Branch:
DatabaseJdbc.java:
Line 36-38

```
...} catch (Exception e) {
     System.err.println(e.getClass().getName() + ": " + e.getMessage());
     return null;
   }
```

Line 76-77

```
} catch (SQLException e) {
     e.printStackTrace();

...
```

Reason: The branches are dealing with exceptions in sql execution or with database connections. To ensure the crash safe feature of our tool, we save every single operation in the database and retrieve from it whenever we reconnect. There are around 300 lines of code that are not tested because they are either dealing with sql exceptions or database connection errors. Creating a test case for each of these instances will need at least 60 test cases and we do not have the bandwidth to do so.
Other than that, our code coverage is around 91.3% (1319 + 300 / 1773)
Link to coverage test reports: https://codecov.io/gh/yumeng-luo/COMS4156 (Up to data)
https://codecov.io/gh/yumeng-luo/COMS4156/commit/fc70882ad97a06b12d305d492b38c3525d715fab (By the time this document is written)

Part 4:

Link to CI configuration:

https://github.com/yumeng-luo/COMS4156/blob/main/.travis.yml
Link to CI report: https://travis-ci.org/github/yumeng-luo/COMS4156/builds/ (Up to data)
https://travis-ci.org/github/yumeng-luo/COMS4156/builds/747590777 (By the time this document is written)