Small Bank Management System – API testing

Files involved:

Program under test: bank\_API\_impl.js

Test case input module: APItesting\_testcases.js

Test driver: APItesting\_driver.js

Test oracle: APItesting\_oracle.js

Database test data population script: todo

Purpose: The purpose of this document is to show the derivation of the test cases for each API function. The method to be used to determine the test cases was Functionality-Based Input Domain Modeling. The basic principles are; for each function identifies the parameters and characteristics of interest, then identify blocks where the partitions of the characteristics are equivalence classes and characteristic complete. The goal is to create test frames with the blocks by following the all combination criterion (all combinations of blocks from all characteristics must be used, excluding [single] and [error]). Then test cases are recorded in the Test plan table.

Special cases:

[single]: Special or redundant conditions that do not have to be combined with all possible choices. It is tested with one test case.

[error]: Assumed that if the parameter has this particular value, any call of the function using that choice will result in the same error. Don’t have to be combined with other blocks because the other blocks don’t matter in this case.

General Specification of “bank\_API\_impl.js”: The javaScript file’s purpose is to utilize nodeJS to interface with an SQL database, “bankdb” in MySQL workbench, to perform specific SQL queries on the tables of “customers” and “accounts”.

General special tests cases are the column related attributes like datatype and max size, after confirming that an error is returned once for each characteristic in all columns it does not have to be tested again in every other API function.

Since this API has no business logic and is purely making queries to the database the main concern is the correctness of the query’s response and modifications on the tables. For example in the client program, a customer’s pin can only be 4 integers or it will give a warning and re-prompt, but in the current API it does not check therefore if the parameters are the valid datatype and size the query should return a non-error response related to the query.

The response varies depending on the query, for example “selecting” will return a JSON object while removing a row will return an integer of rows affected.

API Testing Functions

1. createCustomer (name: string, pin: int, date: string, num\_acc: int, num\_trans: int, callback: JSON)

Purpose: The purpose of this function is to create a new customer in the “customers” table, it performs multiple queries:

The first query to the database is to select all the primary keys, “id\_cust”, from the “customers” table to determine a new non conflicting primary key for the new customer.

The second is the do the insertion to the “customers” table.

The third and final is to select the newly created customer to return to the client.

Domain modelling

1.1. Characteristics of interest:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameters: | Name: string | Pin: int | Date: string | Num\_acc: int | Num\_trans: int |
| Characteristics: | width | width, datatypes | width | width, datatypes | width, datatypes |

1.2. Blocks:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameters | Characteristics | Blocks | Values | Constraints | Label |
| Name: string | width | Minimum(0) | [0] | [single] | B1 |
| Nominal(8) | (0, 50) |  | B2 |
| Maximum(50) | [50] | [single] | B3 |
| Beyond(51) | (50, ∞) | [error] | B4 |
| Pin: int | width | Minimum(0) | [0] | [single] | B5 |
| Nominal(4) | (0, 11) |  | B6 |
| Maximum(5) | [5] | [single] | B7 |
| Beyond(6) | (6, ∞) | [error] | B8 |
| datatypes | Contains a non-int | !int | [error] | B9 |
| All integers | int |  | B10 |
| Date: string | width | Minimum(0) | [0] | [single] | B11 |
| Nominal(8) | (0, 25) |  | B12 |
| Maximum(25) | [25] | [single] | B13 |
| Beyond(26) | (25, ∞) | [error] | B14 |
| Num\_acc: int | width | Minimum(0) | [0] | [single] | B15 |
| Nominal(4) | (0, 11) |  | B16 |
| Maximum(11) | [11] | [single] | B17 |
| Beyond(12) | (11, ∞) | [error] | B18 |
| datatypes | Contains a non-int (string) | !int | [error] | B19 |
| All integers | int |  | B20 |
| Num\_trans: int | width | Minimum(0) | [0] | [single] | B21 |
| Nominal(4) | (0, 11) |  | B22 |
| Maximum(11) | [11] | [single] | B23 |
| Beyond(12) | (11, ∞) | [error] | B24 |
| datatypes | Contains a non-int (string) | !int | [error] | B25 |
| All integers | int |  | B26 |

There are special test cases that can be tested once for the entirety of this API because they are related to the database’s tables’ column properties.

1.3. All combination coverage:

|  |  |
| --- | --- |
| Test Frames | Test Cases |
| B1 ( or B1, B6, B10, B12, B16, B22, B26) – special | Name = “”,  pin = 1234,  date = “Wed May 22 12:07:41 2019”,  num\_acc = 5,  num\_trans = 20 |
| B2, B6, B10, B12, B16, B20, B22, B26 | Name = “Jonathan”,  pin = 1234,  date = “Wed May 22 12:07:41 2019”,  num\_acc = 5,  num\_trans = 20 |
| B3 ( or B3, B6, B10, B12, B16, B22, B26) - single | Name = “JonathanJonathanJonathanJonathanJonathanJonathanNN”, pin = 1234,  date = “Wed May 22 12:07:41 2019”,  num\_acc = 5,  num\_trans = 20 |
| B4 ( or B4, B6, B10, B12, B16, B22, B26) - error | Name = “JonathanJonathanJonathanJonathanJonathanJonathanNNNN”,  pin = 1234,  date = “Wed May 22 12:07:41 2019”,  num\_acc = 5,  num\_trans = 20 |
| B5 ( or B2, B5, B11, B15, B21) – other min singles. Not good practice, just to cut down on test cases here. | Name = “Jonathan”,  pin = “”,  date = “”,  num\_acc = “”,  num\_trans = “”.  Start with an empty string and then parseInt |
| B7 ( or B2, B7, B10, B13, B17, B20, B23, B26) – other max singles | Name = “JonathanJonathanJonathanJonathanJonathanJonathanNN”,  pin = 12341,  date = “Wed May 22 12:07:41 2019999999”,  num\_acc = 55555555555,  num\_trans = 20202020202 |
| B8 ( or B2, B10, B12, B16,B20, B22, B26) - error | Name = “Jonathan”,  pin = 123412,  date = “Wed May 22 12:07:41 2019”,  num\_acc = 5,  num\_trans = 20 |
| B9 ( or B2, B6, B9, B12, B16, B20, B22, B26) - error | Name = “Jonathan”,  pin = “12e4”,  date = “Wed May 22 12:07:41 2019”,  num\_acc = 5,  num\_trans = 20 |
| B14 ( or B2, B6, B10, B14, B16, B20, B22, B26) - error | Name = “Jonathan”,  pin = 1234,  date = “Wed May 22 12:07:41 2019 Wed May 22 12:07:41 2019”,  num\_acc = 5,  num\_trans = 20 |
| B18 ( or B2, B6, B10, B12, B18, B20, B22, B26) - error | Name = “Jonathan”,  pin = 1234,  date = “Wed May 22 12:07:41 2019”,  num\_acc = 555555555555,  num\_trans = 20 |
| B19 ( or B2, B6, B10, B12, B16, B19, B22, B26) - error | Name = “Jonathan”,  pin = 1234,  date = “Wed May 22 12:07:41 2019”,  num\_acc = “5e10”,  num\_trans = 20 |
| B24 ( or B2, B6, B10, B12, B16, B20, B22, B24, B26) - error | Name = “Jonathan”,  pin = 1234,  date = “Wed May 22 12:07:41 2019”,  num\_acc = 5,  num\_trans = 202020202020 |
| B25 ( or B2, B6, B10, B12, B16, B20, B22, B25) - error | Name = “Jonathan”,  pin = 1234,  date = “Wed May 22 12:07:41 2019”,  num\_acc = 5,  num\_trans = “20e10” |

2. getCustomers (id: int, callback: JSON)

Purpose: The purpose for this function is to query the database for the rows in the “customers” table. If the id input is the value of -2 then it will query for all rows. Otherwise it will query for the specific row indicated with the id.

Domain modelling

2.1. Characteristics of interest: There is no need to test the “id\_cust” column’s width since it was done once before in API test function 1.

|  |  |
| --- | --- |
| Parameters: | id: int |
| Characteristics: | Value |

2.2. Blocks:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameters | Characteristics | Blocks | Values | Constraints | Label |
| Id: int | Value | Non-int | !int | [error] | B1 |
| Less than -2 | (∞, -2) | [single] | B2 |
| -2 for all rows | [-2] | [single] | B3 |
| (-2, 0) | (-2, 0) | [single] | B4 |
| Nominal(4) | [0, 2147483647) |  | B5 |
| Maximum(2147483647) – signed int | [2147483647, ∞) | [single] | B6 |

An environment variable that the tests need to consider is whether the row with the specified id exists in the “customers” table.

|  |  |  |  |
| --- | --- | --- | --- |
| Environment Variable | Characteristics | Blocks | Label |
| Customer row in table | ID exists | Exists | B8 |
| Does not exist | B9 |

2.3. All combination coverage:

|  |  |
| --- | --- |
| Test Frames | Test Cases |
| B1 | Id = “1e” |
| B2 | Id = -3 |
| B3 | Id = -2 |
| B4 | Id = -1 |
| B5, B8 | Id = 1  Id exists = true (make sure in database) |
| B5, B9 | Id = 1  Id exists = false (make sure in database) |
| B6 | Id = 2147483647 – since max width is 5, out of range |

3. updateCustomer (id: int, name: string, pin: int, date: string, num\_acc: int, num\_trans: int, callback: JSON)

Purpose: The purpose of this function is to update all columns in a specified customer row in the “customers” table indicated by the input parameter “id.” It is the client application’s responsibility to provide even unchanged information in the packet sent to the server. Not the most elegant way of handling updates, but for now works.

Domain modelling

3.1. Characteristics of interest: Width of the integer columns don’t need to be manually tested again, but for the string columns they are present to define test blocks.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Parameters: | Id: int | Name: string | Pin: int | Date: string | Num\_acc: int | Num\_trans: int |
| Characteristics: | value | width | datatypes | width | datatypes | datatypes |