CSCI 3060U Course Project

Winter 2022

Phase #1 - Front End Requirements

February 22nd, 2022

Test Plan Document

Group members:

Ben Tsoumagas 100751395

Thinh Le 100741899

Muaz Rehman

Cody Malcolm 100753739

Test plan:

On the following page, there is a consolidated list of requirements (original requirements updated with clarifications/revisions from the client). These requirements contain codes such as “CR02” to identify the tests that are used to verify and/or validate the requirement - all tests are named and described in a table in test_cases_document.pdf.

Each test is contained in a single directory and the test directories are grouped by transaction being evaluated by the test. For example, the test files associated with the test “cannot_delete_admin”, which ensures that admin accounts cannot be deleted with the delete transaction, are located in the directory tests/delete/cannot_delete_admin. All test locations can easily be identified by the descriptive prefix in their test code - for example, the above test has the code DL05, and all other delete test codes also start with the prefix “DL”.

Within the directory of an individual test will be a minimum of three files: input.txt, exp_output.txt, and exp_daily_transaction_file.txt. The input.txt file contains a sequential series of inputs (separated by newlines) that will evaluate the functionality being verified and/or validated by the test. The exp_output.txt file will contain the exact system output expected to result from the inputs in input.txt. The exp_daily_transaction_file.txt will contain the exact Daily Transaction File that should be generated by the series of inputs in input.txt.

Additionally, these directories may optionally contain a current_user_accounts.txt file and/or an available_rental_units.txt file. Should either of these files be present, they will be used as the input current_user_accounts.txt/available_rental_units.txt files upon application startup. If these files are not present, then the test will use default available_rental_units.txt/ current_user_accounts.txt files which are found in the root ‘test’ directory.

To run the tests, a shell script will traverse all subdirectories and for each tail directory containing an input.txt file, execute the application, using the inputs in the input.txt file. The script will compare the generated outputs (both system output and Daily Transaction file) against the respective expected output files and identify any discrepancies. Results will be aggregated in a file with filename format ‘results_yyyymmddnn.txt’ which will show, on each line, the name of the test (identified by the path containing the input.txt file) and whether the test passed or failed. In the case of failure, below the line indicating the failure the script will print the expected and actual output, in that order.

Consolidated list of requirements:

General:

- When invalid input is entered, the user will be re-prompted until the input is valid, subject to noted exceptions. Ways in which input may be invalid (username not found, alphanumeric input where numeric was expected, etc) will be noted in the corresponding sections. One broad requirement is that the system needs to be able to handle “bad input of all kinds”, we have often made assumptions about how this is to take place. Where assumptions have been made in the following requirements, it is mentioned, and there is a consolidated list of every assumption after the table of tests.
- In all cases, the application will ignore leading and trailing whitespace
- If a transaction is expected (login, logout, create, delete, post, search, rent, or quit) and any other input is entered, the system will prompt for a valid input (GN01)
- The current user account file and available tickets files are read into memory when the front end first loads
- Internally, the program should utilize a ‘rented’ flag to keep track of which units have been rented during the front-end session
- For all transactions except ‘login’ and ‘quit’, user must be logged in to perform the transaction (LO01, CR01, DL01, PT01, SR01, RT01)

Login:

- Asks for the username (LI01)
    - Could require reprompting when username entered is not found (LI02)
- Can only login when no user is currently logged in (LI03)
- Leading and trailing whitespace is ignored (LI04)

Logout:

- Transactions records queued to be written to daily transaction file are written when the user logs out (LO02)
- Subsequent logouts during the same front-end session should write to the same daily transaction file (LO03)
- Users cannot logout during other transactions (LO04, LO05, LO06, LO07, LO08)

Create:

- Only admin users may execute a create transaction (CR02)
- Asks for a new username (CR03)
    - Username should be at most 15 characters (CR04)
    - Assumed: Username cannot be an empty string (CR05)
    - Username can contain letters, numbers, dashes, underscores, and spaces (CR06)
    - Usernames are case sensitive (CR07)
    - Usernames are unique (CR08)
    - Violation of any of these constraints will require reprompting
- After accepting a username, asks for a user type (CR09)

    - Valid user types are “AA”, “FS”, “RS”, “PS” - all other inputs require prompting (CR10)
    - No constraints on whether these types should be case-sensitive
        - For a better user experience, lowercase and mixed-case inputs will also be accepted, but will be saved as upper case (CR10)
- This transaction is saved in the daily transaction file in the following format: (CR10)

```
01 <username      > TT<47 spaces>
```

    - Where ‘TT’ is the user type as described above
- For all inputs, leading and trailing whitespace is ignored (CR11)

Delete:

- Only admin users may execute a delete transaction (DL02)
- Asks for the username (DL03)
- Valid delete transactions are written to the daily transaction file in the following format (DL03):

```
02 <username      > TT<47 spaces>
```

- In the case of invalid input, the delete transaction terminates and is not written to the daily transaction file. Cases:
    - The user performing the delete transaction may not delete themself (DL04)
    - Admin users may not be deleted (DL05)
    - The username entered is not found (DL06)
- Leading and trailing whitespace is ignored (DL07)

Post:

- Rent-standard users may not execute post transactions (PT02)
- Asks for the city (PT03)
    - Cities should be at most 25 characters (PT04)
    - Blank inputs not accepted (PT05)
    - Cities can contain letters, spaces, and dashes (PT06)
    - City names are case sensitive (PT07)
    - Any violation of the above constraints requires reprompting
- Once a city name has been accepted, asks for a rental price (per night) (PT03)
    - A rental price can be at most 999.99 (PT08)
    - Assumed requirement: Rental prices should be greater than 0 (PT09)
    - Rental prices should be numeric (PT10)
    - Any violation of the above constraints requires prompting
- Once a price has been accepted, asks for the number of bedrooms (PT03)
    - The maximum number of bedrooms is 9 (PT11)
    - Assumed requirement: The number of bedrooms must be at least 0 (PT12)
    - The input should be an integer in numeric format (PT13)
    - Any violation of the above constraints requires prompting
- The system should generate an alphanumeric rental ID, further details were not provided (PT03)
- Leading and trailing whitespace is ignored (PT14)

- Valid transactions are written to the daily transaction file in the following format: (PT03)
  `03 <username > TT <unitID> <city (width=25)> B DDD.CC< >`
- Where 'DDD.CC' is the price in dollars and cents, B is the number of bedrooms

Search:

- Asks for the city (SR02)
  - Filters results based on city (SR02)
  - '*' acts as a wildcard value (SR03)
- Asks for the maximum rental price (SR04)
  - Filters results based on maximum rental price (SR04)
  - Input must be numeric or '*' (SR05)
  - Reprompt if input is not numeric or '*'
  - If input is '*', use the maximum possible rental price (999.99) (SR06)
- After the maximum rental price is accepted, asks for the minimum number of bedrooms (SR07)
  - Filters results based on minimum number of bedrooms (SR07)
  - Input must be numeric or '*' (SR08)
    - Assumption: Float allowed, in which case round up (SR09)
  - Reprompt if input is not numeric or '*'
  - If input is '*' or a negative number, use the minimum possible number of bedrooms (0) (SR10, SR11)
- Units with rented flag set to 'T' do not appear in searches (SR12)
- Assumption: Searches with no results state an appropriate message (SR13)
- Searches are logged in the daily transaction file in the following format (SR02):
  `04 <username > TT <spaces> <city (width=25)> B DDD.CC< >`
  - Assumption: Since no validation is performed on the user input for city, in the event the user enters more than 25 characters for the city, only the first 25 characters of the search will be recorded (SR14)
- Assumptions made: We can use the wildcard for all input options
- Leading and trailing whitespace is ignored (SR15)

Rent:

- Post-standard users may not perform rent transactions (RT02)
- Assumption: If no rentals are available, reject the rent transaction. Two cases:
  - There are no rentals available (RT03)
  - The only rentals available belong to the logged in user (RT04)
- Asks for a rental ID (RT05)
  - Rental ID must belong to an existing rental available in the system (RT06)
  - Rental ID must correspond to an available rental in the system (RT07)
  - Rental ID must not correspond to one of the users' own listings (RT08)
  - Any violation of the above requirements requires prompting
- Once a rental ID has been accepted, asks for the number of nights (RT05)
  - Units can be rented for a maximum of 14 days (RT09)
  - Input should be numeric (RT10)

    - Assumption: number of days should be an integer (RT11)
    - Assumption: number of days should be greater than 0 (RT12)
    - Any violation of the above constraints requires reprompting
- Once a number of nights has been accepted, the cost both per day and in total should be displayed to the user, and the user should be prompted for ‘yes’ to confirm (RT05) or ‘no’ (RT13) to cancel
    - Reprompt until input is either ‘yes’ or ‘no’ -> Assumption: not case sensitive
- Assumption: Only confirmed rentals are written to the daily transaction file (RT13)
- Leading and trailing whitespace is ignored (RT14)
- Confirmed rentals are written to the daily transaction file in the following format: (RT05)
  `05 <username      > TT <unitID> <city (width=25)> B DDD.CC NN`
    - Where NN is the number of nights

Quit:

- Can not be used when any user is logged in (QT01)
- Writes ‘00’ to the daily transaction file (QT02)
- Users cannot quit during other transactions (QT03, QT04, QT05, QT06, QT07)
- Terminates the program

Multi-function requirements:

Deleted users:

- Once a user has been deleted, they can no longer log in (MT01)
- Rental units of deleted users do not appear in searches (MT02)
- Rental units of deleted users can not be rented (MT03)

New listings:

- New listings should not appear in searches until the next user session (MT04)
- New listings should not be rentable until the next user session (MT05)

Rented units:

- Rented units should no longer appear in searches (MT06)
- Rented units should no longer be rentable (MT07)

Consolidated list of assumptions/requirements problems:

For usernames, the client was specifically and directly asked about what characters are permissible and if there were any sort of “minimum requirements” for usernames, such as a need to contain letters. The client did not specify any requirements beyond disallowing symbols. Therefore there is no username validation to protect against usernames such as “100_001” or “-_-” which contain only allowed characters but might normally be expected to be disallowed.

For the post transaction, no minimums were specified for the price or number of bedrooms. We have assumed that rentals cannot be free (price > 0) and that rentals may have 0 bedrooms (eg. like a bachelor apartment).

For the search transaction, assumptions have been made about how tolerant to be of imperfect input. For example, for the city name, the user may user characters that would not be permitted to be part of a city name. However, we will permit this to happen as it will simply result in no searches being found. Likewise with the number of bedrooms, when posting a rental only non-negative integers are permissible. However, if during a search a user enters a fraction, or negative number, it is not an issue to treat these as valid inputs. Speaking of the case where there are no results in a search, as a matter of user experience, we have assumed that when there are no results for a search, an appropriate message should state such rather than saying something along the lines of “here are your results” and displaying an empty table. Finally, it was specified that ‘*’ should be a valid wildcard, but it was not specified for which fields this wildcard was intended and a question about exactly how it should work for the city field was unanswered by the client. We have assumed it is to be implemented for all search fields, and quite robustly for the city field.

There is a general requirement that transactions started must be finished, and for the rent transaction there is also a requirement that users cannot rent their own units. In addition, there is always the possibility that there are simply no units available to rent. In such a situation, if a user starts a rent transaction they will become “stuck” in this transaction and be unable to exit the program, potentially causing loss of data and other issues. As a result, we have assumed a requirement to reject rent transaction requests when there are no possible valid rentals. Also for the rent transaction, we have assumed that the number of nights can only accept positive integers as inputs - unlike with the search transaction, we cannot make any assumptions about the user’s intention if they enter a negative integer or a fraction. Finally, as the user has the opportunity to confirm or cancel their rental transaction, we have assumed that only confirmed rentals are written to the daily transaction file - based on the fact that this file contains no way to differentiate between confirmed and canceled rentals.

The client did not answer a question about any preferred format for rental IDs beyond that they are alphanumeric and 8 characters long. We will use the format ‘A0000000’, incrementing the 7 digit suffix each rental.

In the latest correspondence, the client stated that "the total number of characters per line in the daily transaction file is the sum of the length of each field", but to fulfil this requirement we would no longer have spaces between fields in the line (aside from padding spaces where a field is not using all the space its permitted). It's highly probable the client meant "sum of the length of each field plus one space between each field", consistent with the requirements document. As such, we are ignoring this clarification.

The client recommended adding a quit transaction. Doing this created some ambiguity in how to define a ‘session’ - is a session from when a user logs in until when the user logs out, or is a session from when the application is started until the quit command is entered and the application terminates? Which interpretation makes sense largely depends on the context of the original requirements, so we have handled this on a case-by-case basis, with both “user sessions” and “the front-end session” as the case may be. For example, the original requirements stated that ‘00’ should be written out to the daily transaction file at “end of session.” Since there is nothing written to the daily transaction file upon login, and also because this seems a sort of null termination indicator, we deemed this requirement to apply to the front-end session, and not each user session. In the requirements above, where the definition of a session is relevant, we have stated whether we are working with a user session or front-end session - for example the requirement associated with test MT04 is “New listings should not appear in searches until the next **user** session” (emphasis added).

Finally, the client left it up to our discretion whether we needed a test to explicitly prevent users from deleting their own account, since this is already prevented by the fact that only admins can delete accounts and admin accounts can’t be deleted. We have opted to implement the redundant requirements in case the program is later changed to allow admin accounts to be deleted.