**Software Testing Strategy**

We will be using black box testing to test the three use cases. The login page of the application will be tested for seamless user login. The login screen has 2 fields, the username and password as the inputs and the output will be access/denial to the system. The test will not consider the specification of code, only the validity of login credentials for an account. The test will check if either the user logged in with correct username and password, or receives an error message when wrong username/password is inputted. This will be done through manual or automated UI testing to check for client side and documented testing for server side. Equivalence portioning technique will be used for this to check if number of characters, valid type of non-alphanumerical characters is not violated. State transition technique will be used to block user from access for few minutes.

The setting ride page from rider side also has 2 fields: location and start time. the page will be tested for its usability, performance and data load. The system will check the validity of location and date/time. The locations (starting and destination) and pick-up time are the inputs and the output is succession of proceeding to send a ride request. Boundary analysis will be used for time test and state transition will be required for location check. Users have a choice of using current location or search for a specific location. If pick-up location searched by user is not available then by default current location will be used. Decision table testing will be done to check if passenger and/or driver are present or absent at the agreed location on time; which will then lead to driver facing a consequence and passenger eligible for refund, also to be tested by same technique. The same method will be used to confirm ride requests acceptance and cancellation.

**Testing for use case 1: log-in**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test case** | **Description** | **Steps** | **Data** | **Expected Results** | **Type** |
|  | Check customer log-in credentials with valid databases | 1. Go to application 2. Enter username 3. Enter password 4. Enter sign-in | Username: Ishrat  Password: Ishrat123 | User should log-in to system | Functionality |
|  | Check customer log-in credentials with invalid databases (not username) | 1. Go to application 2. Enter username 3. Enter password 4. Enter sign-in | Username: Ishrta  Password: Ishrat123 | User should not log-in to system | Functionality |
|  | Check customer log-in credentials with invalid databases (wrong password) | 1. Go to application 2. Enter username 3. Enter password 4. Enter sign-in | Username: Ishrat  Password: Ishrat124 | User should not log-in to system | Functionality |
|  | User passes all 4 chances to log-in and username is entered wrong | 1. Go to application 2. Enter username 3. Enter password 4. Enter sign-in | Username: Ishrts  Password: Ishrat123 | User should not log-in to system and is blocked out for 5 mins | Functionality |
|  | User is redirected to password reset page | 1. Enter email 2. Receive password reset link in email 3. Enter username and new password 4. Confirm new password 5. Enter submit | Email: ishrat@mail.ca  Username: Ishrat  Password: Ishrat111 | New credentials are saved | Functionality |
|  | User passes all 4 chances to log-in and password is entered wrong | 1. Go to application 2. Enter username 3. Enter password 4. Enter sign-in | Username: Ishrat  Password: Ishrat125 | User should not log-in to system and is blocked out for 5 mins | Functionality |
|  | User is redirected to password reset page | 1. Enter email 2. Receive password reset link in email 3. Enter username and new password 4. Confirm new password 5. Enter submit | Email: ishrat@mail.ca  Username: Ishrat  Password: Ishrat111 | New credentials are saved | Functionality |
|  | Check customer log-in credentials with invalid databases (blank field) | 1. Go to application 2. Enter username 3. Enter password 4. Enter sign-in | Username: Ishrat  Password: | Message displayed to enter data in blank field | Functionality |
|  | Check customer log-in credentials with invalid databases (blank field) | 1. Go to application 2. Enter username 3. Enter password 4. Enter sign-in | Username:  Password: Ishrat123 | Message displayed to enter information in blank field | Functionality |
|  | Check if user cannot enter characters past the valid range for any field | 1. Go to application 2. Enter username 3. Enter password 4. Enter sign-in | Username: Ishrat1234567890  Password: Ishrat123 | Message displayed to enter a username between 8-15 characters | Security |
|  | Check if user cannot enter characters past the valid range for any field | 1. Go to application 2. Enter username 3. Enter password 4. Enter sign-in | Username: Ishrat123  Password: Ishrat!@#%^\*$  ^&\*()\_+\_) | Message displayed to enter a password between 8-15 characters | security |
|  | Check if a user shouldn’t be allowed to log-in with same credentials from different devices simultaneously | 1. Open application in one mobile device 2. Enter username & password 3. Enter sign-in 4. Open application in another device 5. Enter username & password 6. Enter sign-in | Username: Ishrat  Password: Ishrat123 | Message displayed to warn current user is active in a device | security |
|  | Check time taken to log-in with valid credentials | 1. Go to application 2. Enter username 3. Enter password 4. Enter sign-in | Username: Ishrat  Password: Ishrat123 | User should log-in to system within max 15 seconds | performance |
|  | Check if the ‘Cancel’ button works to erase texts entered | 1. Go to application 2. Enter email/password/username 3. Press ‘Cancel’ | Email: ishrat@mail.ca  Username: Ishrat  Password: Ishrat111 | Text fields should become blank | usability |
|  | Check if ‘Enter’ key of keyboard works correctly | 1. Go to application 2. Enter email/password/username 3. Press ‘Enter’ | Email: ishrat@mail.ca  Username: Ishrat  Password: Ishrat111 | Pressing Enter should allow user to log in | functionality |
|  | Check text fonts, size and color coding | n/a | n/a | Text alignment and size should appear in a particular way for web browsers an in different way for mobile device | UI |
|  | Check page controls in different OS and web browsers | n/a | n/a | Control buttons redirect to appropriate pages without error | OS/browser compatibility |

**Testing for use case 2: driver receives the registered passenger**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test case** | **Description** | **Steps** | **Data** | **Expected Results** | **Type** |
|  | Check Passenger got in car | Driver confirms Passenger is present | Passenger Ride history | Passenger Ride History should be updated with “+1” | Functionality |
|  | Check Passenger is absent | Driver confirms Passenger is NOT present | Passenger Ride History | Passenger Unsuccessful Ride History should be updated with “+1” | Functionality |
|  | Check Driver picked up Passenger | Passenger confirms Driver is present | Driver Ride History | Driver Ride History should be updated with “+1” | Functionality |
|  | Check Driver is absent | Passenger confirms Driver is NOT present | Driver Ride History | Driver Unsuccessful Ride History should be updated with “+1” | Functionality |
|  | Check Refund if Driver is absent | Driver doesn’t arrive.  Passenger receives refund | refund amount | Passenger receives a refund | Functionality |
|  | Check Rating and Review | Upon ending the drive, Passenger gives the Driver a star rating and a review | Star rating  And review | Driver profile is updated with the new rating  And review | Functionality |
|  | Check Valid Driver Information:  Current Location | Driver inputs Current Location into profile | Valid location | Driver’s profile is updated with the current location | Functionality |
|  | Check Invalid Driver Information:  Current Location | Driver inputs invalid Current Location into profile | Invalid Location | Error msg displayed:  “Please input valid location” | Functionality |
|  | Check Driver Info:  Destination | Driver inputs Destination into profile | Valid Destination | Driver profile is updated with the destination | Functionality |
|  | Check invalid Driver Info:  Destination | Driver inputs invalid destination into profile | Invalid Destination | Error msg displayed:  “Please input valid destination” | Functionality |
|  | Check Driver info: valid date and time | Driver inputs valid date and time into profile | Valid date and time | Driver profile is updated with the date and time | Functionality |
|  | Check Driver Info:  Invalid Date and Time | Driver inputs an invalid date and time into their profile | Invalid Date and Time  Ex. Yesterday’s date | Error msg displayed:  “Please input valid date and time” | Functionality |
|  | Check that the Driver can see their Passenger’s profile | 1. Driver logs in. 2. Driver checks who they will be picking up. 3. Driver checks their Passenger’s profile | n/a | Driver should see their passenger’s profile to know their name | performance |
|  | Check that the Passenger can cancel a ride | 1. Passenger logins 2. Passenger cancels ride 3. Driver logins 4. Driver sees that their ride has been canceled | n/a | Driver should be able to see that the Passenger has canceled and the Driver no longer needs to drive | usability |
|  | Check that the Driver can cancel a ride | 1. Driver logins 2. Driver cancels ride 3. Passenger logins 4. Passenger sees that ride has been canceled | n/a | Passenger should be able to see that the Driver will not be arriving | functionality |

**Testing for use case 3: set a ride on passenger side**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test case** | **Description** | **Steps** | **Data** | **Expected Results** | **Type** |
| 1. | Passenger sets the needed location for the ride | 1. Go to the application 2. Set the current location or desired location to be picked up 3. Set the destination | Valid locations | Passenger’s current location should be automatically determined by the app, and the passenger should have the ability to modify it if necessary. | Functionality |
| 2. | Passenger can set and change the timing | 1. Set a pickup time for later | Valid date and time | Passenger should set a time for pickup either for now or in the future | Functionality |
| 3. | Passenger sees the ride price based on how many people want to share the ride with | 1. Select how many people want to share the ride with 2. See the price upfront and knows how much it would be to the destination in the selected time | Calculating ride price | Show the ride price based on location, drive time, traffic and etc | Functionality |
| 4. | Passenger sets a ride or requests an existing ride | 1. Passenger choose the type of ride (e.g. Individual or group) | Choose the type of ride | Based on the passenger desire they can choose if they want to share a ride with other passengers or ride to the destination alone | Functionality |
| 5. | Driver checks for passengers on their route | 1. Driver checks near areas for passengers 2. Choose a passenger based on location 3. Driver chooses the passenger 4. Sees the passenger information | Valid location | The driver must have a chance to view passengers’ location and destination so they can decide on whether or not they want to accept the request for a ride from a passenger | Functionality |
| 6. | Accepted passenger(s) see the driver’s information | 1. Passenger must be shown the driver’s information such as name, car model, licence plate number, location | Valid information | Passenger must have a clear understanding of what car is designed to pick them up | Functionality |
| 7. | Ride cancellation | 1. Driver can cancel the ride if necessary 2. Passenger can cancel the ride if necessary | Order cancellation | Both sides should be able to cancel their ride based on different situations | Functionality |
| 8. | The ride (trip) itself | 1. Provide an access to the road map for both sides | Valid location | Driver should see the map to drive, and passenger should see the map to know where the driver is going and make suggestions if needed | Functionality |
| 9. | Driver ends the ride at the destination | 1. Driver should end the trip on the app | End Trip | When the driver has dropped off the passenger, they must end the trip so they can have other activities on the app | Functionality |
| 10. | Passenger’s account is going to be charged with the ride’s price | 1. After driver ended the trip, passenger will be charged | Payment | The price of the trip will be charged to the passengers’ bank accounts that have been provided before | Functionality |
| 11. | Driver’s fee will go to their account | 1. After each trip the driver’s fee will be transferred to their account | Payment | The fee for each trip will be transferred to the driver’s bank accounts that have been provided before | Functionality |
| 12. | Review driver/passenger | 1. After each trip, driver can provide a review on the passenger or skip it 2. After each ride, passenger can provide a review on the driver or skip it | Review/ Feedback | Passenger and driver have the choice to rate and submit a review after the ride. This system of checks and balances maintains a high-level service and helps avoid any misunderstandings | Functionality |
| 13. | Start to search for another ride or passenger | 1. Driver searches for other passengers if wants to 2. The passenger can start another ride after the driver ends the ride | Start another ride | Driver and passenger can look for other rides with their desired location | Functionality |
| 14. | Application security | 1. Save and protect passenger and driver information | Protect information | The app must protect all of the rides, locations, and payments to itself and prevent them from leaking to anyone else who is not involved | Security |
| 15. | Maintain discipline in the application | 1. Prevent the app from crashing, getting attacks, leaks | Maintain security and functionality | The application must always work at its best and if necessary, let the users know about the upcoming updates and security concerns | Security |

**Software Testing Tools**

One tool we would like to use is Katalon Studio. Katalon Studio is a web and mobile automation framework which is built on top of Selenium and Appium. This tool is also a cross-platform tool to transform manual testing into automation testing, which is a time-saving and strong feature in this tool. The cross-platform enables the tool to require minimum program background, which makes it a lot easier to set up.

Another one we would also like to use is SmartBear ReadyAPI. It is a tool for testing APIs. Since the project will have interfaces for adding different kinds of functions, and the development group might also need to develop their own interfaces, so a tool for testing APIs is necessary. A wide variety of API testing capabilities are available and continuous integration also serves as a strong feature in this tool.

The project might also use open-source programs and interfaces, so corresponding tools for them are also needed. BlazeMeter is a great choice for this. It is open source-based performance tester as well as offers scalable load testing. It also offers real-time reporting and mobile capture, which are very useful since the project might have a great potential as an app. It mainly tests scalable load testing.