**Part 1: Scenario**

**Actors:** Site User

**Pre-Conditions:** The user is trying to schedule a reservation for 3 pm 2 days from now.

**Scenario:**

1. The user searches for the site
2. The user clicks the “make reservation” button
3. The system displays the online form with a calendar. Any date on the calendar will begin a form for a reservation on that date
4. After the user clicks the date they wish, they are taken to a page that asks for the number of people, the name of the person making a reservation, the time they would like to reserve, and their email.
   1. If the required information is missing, it will highlight the missing portions
   2. If the time isn’t available it will be greyed out so that it cannot be chosen.
   3. If the headcount is over 40, the maximum capacity of the restaurant, the reservation will highlight and display an error
5. When the user clicks submit, the system will check if the email is a valid email, if not they will be returned to the reservation screen
   1. If the email is valid, the calendar will update on the server, marking the time as unavailable if no tables are left available, and save the name and party size on the manager interface
6. If they exit out of the page, the information will not be saved and the reservation won’t be made.

**Post-Conditions:** The user will be sent an email verifying their reservation. The restaurant reservations software will save the reservation and update it for the employees to view.

**Part 2: Attacks**

* **UI Attack #8** is testing invalid outputs to be generated. For example, the attacks test the reservation’s page to random user’s inputs. The user could input the incorrect date, time, name, phone number, and email address. We want to test how the reservation modules handle incorrect inputs.
* **UI Attack #7** is testing different outputs to be generated for each input. For example, the attacks test all valid inputs that the user could enter for the reservation page. We want to test if the user could input numbers or different characters in the name box. These types of inputs could confuse the staff and reservation system.
* **UI Attack #4** is testing the ability to save names that are too long. For example, putting a name that is over 255 characters long. We want to be able to handle this without crashing the site and the reservation software that the restaurant would use.

**Part 3: Test Matrices**

Tables 1 and 2 are test matrices for testing the reservation functionality.

**Requirements to be tested:** As a customer, I want to create a reservation so that I can have a table when I come to the restaurant. Tests for the availability of tables for party sizes of 1 or more with a max capacity of 40 people total in the restaurant.

**Table 1: Reservation Test cases for**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case (Input)** | **Basis: Contains Boundary Values** | | | **Expected Result** |
| **With negative** | **With Zero** | **Positive** |
| **(-3 persons)** | **x** |  |  | **Invalid Amount** |
| **(0 persons)** |  | **x** |  | **Invalid Amount** |
| **(5 persons)** |  |  | **x** | **Submitted** |
| **(41 persons)** |  |  | **x** | **Invalid Amount** |

**Table 2: Reservation Test cases for Table Size & Number of Customers**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case (Input)** | **Basis: Table Size & Number of Customers** | | | | **Expected Result** |
| **Minimum** | **Middle** | **Max** | **Too Many Customers** |
| **1 customer, table capacity 4** | x |  |  |  | **Reservation Success** |
| **2 customers, table capacity 4** |  | x |  |  | **Reservation Success** |
| **4 customers, table capacity 4** |  |  | x |  | **Reservation Success** |
| **5 customers, table capacity 4** |  |  |  | x | **Reservation Error** |