**Technical Challenge: Software Engineer in Test**

*This should take no longer than 2.5 hours max.*

**The Challenge**

SnapTravel is a conversational commerce startup, and our chatbot operates on top of existing messaging platforms to help users find, book and receive support for great hotel deals.

Your goal is to write an e2e test in any language/framework (preferably Python) for a complete search and booking flow for SnapTravel. You will then submit your test, along with a brief outline of your testing plan and approach, via a Github/Gitlab repo or snippet.

**Requirements**:

1. Write out your testing plan
2. Write the e2e test in code
3. Run the e2e test and output the results of the test
4. Briefly describe tradeoffs you made between rigidity and brittleness when writing the tests
5. Briefly describe what other test cases you would write in order of importance from a regression testing perspective
6. Push your code to a repo or snippet on GitHub/GitLab and submit your code/snippet

**Extra Info for your test**:

* Start with <https://www.snaptravel.com/search?encrypted_user_id=5xqebwRCiWusH08KS2yJKA> and run a search for Paris
* Choose any hotel
* Choose any room for the given hotel
* Pretend to make a booking
  + Use any guest name, any phone number
  + test@snaptravel.com for the email
  + 4111 1111 1111 1111 for the credit card number
  + any CVV and expiry date and any billing address
* **You do not need to complete booking, you can end test here**

**Note:***Please submit your finished work to* [***brett@snaptravel.com***](mailto:brett@snaptravel.com)*, and let me know if you have any questions/concerns. If sharing a link to your personal Github/Gitlab, please ensure I have auth to view :)*

# Test Plan

* I’ve created a high level test plan for pages and areas surrounding the scope of the automation scripts written.

|  |  |
| --- | --- |
| Test Set | Test Case |
| Search Results Page | Verify Hotels are selectable and Details of Each hotel are correct |
|  | Verify Search can be performed (no errors returned) |
|  | Check that Search results match search criteria |
|  | Check that UI shows search results matching with the API response |
|  | Validate that UI elements are as expected |
|  | Make sure that search results are not cached (test for live updates) |
|  |  |
| Search Page | Test text box input validation (strings cannot be too long, empty searches should not be able to be performed, boundary testing) |
|  | Make sure that when a search is conducted the request is properly sent to the Backend |
|  | Date Validation Tests – make sure user cannot search in the past (dates), as well as any other date limits determined by the product requirements. |
|  |  |
| Hotel Details Page | Verify UI elements are as expected |
|  | Verify that results/rates from the API, UI and DB are consistent |
|  | Verify that hotel prices get updated (prices do not get cached, and are reactive to new price updates upon refresh) – can mock API if necessary |
|  | Navigation testing – verify that navigation elements work as expected. |
|  |  |
| Hotel Booking Page | Text/field validation testing (make sure that field requirements are followed, ie. No letters or symbols in a credit card field) |
|  | Test that the API is storing/sending the right information upon clicking the submit button. |
|  | Refresh page/navigate away from then back to page, test that any values within the fields are NOT cached. |
|  |  |

# Rigidity vs. Brittleness Trade off

I decided to use a Page Object model framework for this task, as it provides the most reusability in the future. It required a bit more time to setup, but once it was setup, it makes the original test class very simple and easy to read. It also makes developing any future tests much more fluid. This model also makes it so that if anything changes in the UI which would break a number of different tests, there is a central location to update/accommodate those changes that would fix all the tests at once (vs. fixing them one at a time). The trade-off is that this type of framework requires a bit more maintenance, as it needs to be updated anytime that UI elements are changed to ensure that the page classes are up to date.

# Additional Regression Testing

1. Basic Functionality/ Happy Paths (common/necessity paths that the user uses; ie. Ability to search and book hotels)
2. Business Critical Functionalities
3. Integrations with platforms like messenger, whatsApp, hotels etc.
4. High Risk areas (or areas where there have been bugs before)