1. Imagine the following situation. You need to establish a QA process in a cross-functional team. The team builds a front-end application using REST APIs.

1. Where would you start? What would be your first steps?

Start with the test strategy:

- What functional features will be done for this application (logins, search, get information, etc.)?
- What are the requirements for these features (responses expected from the features, successful messages, error messages, information)?
- What are the non-functional requirements (UI mockups, text, font, font size, font color, background colors, error messages wording, etc.)?
- How to handle the defects, move to the backlog or fix during the current sprint?
- Environments and browsers which will be required for testing (mobile?).

Come up with the test scenarios for API and UI based on the expected results from the requirements and then create test cases for positive and negative scenarios (validate expected errors).

Let the developers know to test their code with unit test and also help the QA team with having unique names or ID's for objects in the UI for automation.

Create the test cases in the Testing Management tool that is being used by the project and decide which will be the test cases that can be automated for this application.

This is assuming there is a development environment already setup for testing.

2. Which process would you establish around testing new functionality? How would you want the features to be tested?

The testing of the new features will be done first in the development environment.

Assuming the APIs will be created by the team, first thing I would do is start testing the new feature with functional/manual testing on each APIs endpoint through an API testing tool (Ready API or Postman for example) to validate the expected response.

After APIs are tested successfully we can automate the different positive and negative test cases that had been created and add them to a Test Suite in our API testing tool. Once the APIs are working as expected the next step would be to test the functionality on the UI.

We could do a functional smoke test to the UI once the APIs have been implemented to the front end. Once the validation is successful we could automate all the positive and negative scenarios that were planned for the feature's UI validations based on the functional and nonfunctional requirements.

We would create an automation Regression Test suite for when there is a feature change or new feature implementation.

3. Which tools would you suggest using to help your team with a daily work?

- Slack for team communication, it has been very useful in the last projects I have been on, it’s instant.
- Any email service for communication too, this is mostly used for communication with clients but Slack can also work if the client also uses it.
- Webex for team meetings, you can call through the internet or phone, share your screen, chat conversation, control someone else screen, record meeting, schedule meetings, among other things, it’s the tool I’ve been using for the past years for daily meetings and it has been better than Skype for Business calls.
- Ready API for testing REST APIs, you can create test suites and add your test cases for each REST service and you can combine more than one web service in the same test cases, connect to databases, configure environment you want to test, etc. In my current project we have test suites automated for each of the applications we are currently working making it easier to run and check the service when there is a change.
- Sourcetree is a Git GUI, been using Sourcetree for committing changes to projects for the past years, it’s very easy to use.
- JIRA works well with scrum, good for project management, issue (tasks) tracking, bug tracking, I have worked with JIRA for many years now.
- TestRail for testing management, you can add test cases, results, the effort of your testing and it integrates with JIRA easily, you can add the User Story as reference as well as the bugs found. I have used TestRail for the last 2 years and had a good experience, before I used ALM but TestRail is much easier to use.

4. If you would do a test automation which techniques or best practices would you use the application?

We would divide work with the team, better knowledge in a scripting language would work with automating scripts and other team members can work on the test cases and data creation.

We would first have to decide which tests can be automated by selecting the ones that are repetitive, test where we know that the response/results won’t change, where we can get a human error(like validating a long text), manual test that take long time (this can be regression tests).

I would like to have an automated smoke test suite per module/feature of the application just to have a quick check when there is a small change (both at API and UI level).

There would also be an automated regression test suite for full checks (both at API and UI level).

Would be nice to have an automated nightly run test for the APIs so we know if any of them starts failing. We can also add nightly run with the smoke test suite of the UI of the application if necessary.

2. How would you test search UI functionality of your favourite website (e.g. https://medium.com, https://www.google.de)?

1. Choose your favourite website which has search functionality.

Amazon.com

2. Document several test cases.

- Basic search, since Amazon has many products and returns even if it's not exactly what you're looking, it will return any related item you searched for, it almost always has a response. We will verify that the entered product name is at least mentioned once in the search results.
- Enter special character search, it is not common to find products with special character search thus giving us a message of product not found, I would validate the message given for when no products are found.
- Pagination test, Amazon has many products which gives you pages to go over, validate the pagination works for when there are many search results.
- Sort order test, the search results are ordered by default by Relevance, verify the sorting order options for search results.
- Filters test, every time you search Amazon gives you the option to filter by displaying the filter options on the left side of the screen, validate the filter options from the side bar.
- Run mentioned test cases for Logged in member and for not logged member.
- Run in different browsers

3. Implement one or two automated tests based on the test cases.

```
import unittest
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time


class SearchTest(unittest.TestCase):
    def setUp(self):
        self.driver =
webdriver.Chrome(executable_path='C:\\Users\\joser.sanchez\\PycharmProjects\\chro
medriver')
        self.driver.get('https://www.amazon.com')
        self.driver.maximize_window()

    #Basic Amazon search
    def test_amazon(self):
        search_input = 'playstation 4'
        search_id = 'twotabsearchtextbox'

        search = self.driver.find_element_by_id(search_id)
        search.send_keys(search_input)
        search.send_keys(Keys.ENTER)
        time.sleep(3)
        assert search_input in self.driver.page_source
        print('Basic search successful.')

    #Search results with next and previous page(pagination)
    def test_pagination(self):
        search_input = 'Wonder Woman'
        search_id = 'twotabsearchtextbox'
        nextpage_id = 'pagnNextString'
        prevpage_id= 'pagnPrevString'

        search = self.driver.find_element_by_id(search_id)
        search.send_keys(search_input)
        search.send_keys(Keys.ENTER)
        time.sleep(3)

        #Search results page 1
        assert search_input in self.driver.page_source
        next_page = self.driver.find_element_by_tag_name('body')
        next_page.send_keys(Keys.END)
        time.sleep(3)

        # Search results page 2
        self.driver.find_element_by_id(nextpage_id).click()
        next_page2 = self.driver.find_element_by_tag_name('body')
        next_page2.send_keys(Keys.END)
        time.sleep(3)

        # Search results page 3 and validate they are still Wonder Woman results
        self.driver.find_element_by_id(nextpage_id).click()
        prev_page = self.driver.find_element_by_tag_name('body')
        prev_page.send_keys(Keys.END)
        time.sleep(3)
        assert search_input in self.driver.page_source

        # Previous search results (page 2) and validation for still Wonder
Woman/Amazon
        self.driver.find_element_by_id(prevpage_id).click()
        assert search_input in self.driver.page_source
        assert 'Amazon' in self.driver.title

        print('Successful pagination search')

    def tearDown(self):
        self.driver.quit()


if __name__ == '__main__':
    unittest.main
```