Test Plan Document for InterviewReady

This test plan is designed for InterviewReady, a platform which simplifies System Design by providing and explaining the examples along with live code as well as visual animations. The test plan includes various types of testing such as Functional Testing, Performance testing and Interface testing. Although, as of now, our main focus would be on the following scenarios:

1. Test if the videos in the course are visible to users
2. Test if the APIs are fetching the content properly
3. Test the User Interface correctness

1. Testing the visibility of the videos in the course to the given users:

- **Test Case Description:** Checking if the videos of the course are visible to the user as per the requirement specification.

- **Assumptions**: It is assumed that the user is authenticated to the platform and has access to the course.

- **Steps to be executed:**
  - Log in to InterviewReady with your credentials
  - Navigate to the course section
  - Click on any of the video

- **Expected Result:** The video is shown to the user in the Vimeo player inside the browser with the right dimensions of the player, i.e., width=640 and height=320

This test case scenario can be tested by automation, using the selenium framework along with any language of choice.

For that, the user needs to be logged in to his account, move to the 'Course Preview' section, scroll down to the video, and check the availability of the player along with its dimensions.

In the code given below the video player in the "Course Preview" section is being checked.

(Note: The 'Course Preview' section is being used due to the lack of access to the full course as of now)

Link to the code: https://github.com/harsiddhdave44/InterviewReady-Task/blob/master/chekcVideoVisibility.py

2. **Test if the APIs are fetching the content properly**

The APIs can be checked either by the process of manual testing, or by using automation scripts or tools, such as Postman, JMeter, or a script which validates the necessary metrics as well as the contents of the API response.

In order to check if the APIs are working as expected, the following things must be checked:

- **The purpose of the API must be fulfilled.**

  The APIs must accept and return only the data which they're supposed to.
  For example, the "FetchCoursePlaylist" API should return the list of all the videos available in the given course along with their titles, video number, links, etc.

  The API(s) would have a certain "Content-Type" that they're supposed to accept and return, for example, in JSON format. So, in that case, the API must be receiving only that certain types of Content-Types and must also be checked against the response Content Type.
- **The status code of the API Response.**

  The most common way of verifying if the API is returning the appropriate data is by checking its status codes. But although it's the most common way to check the API, it does not cover all the possible scenarios.

  For an instance, it is possible that the API is returning the status code 200 OK, but the content is empty or not as desired. This may indicate that the API is working properly, but the business logic behind it is not working as it should. Also, as the status codes are set by the development team, it should also be made sure that the correct status codes are sent as per the API's result.

Alongside all of this, two types of test cases must be prepared for testing the APIs, Positive Tests as well as Negative Tests.

The Positive Tests would contain the test data which, when provided to the API, would return the output as specified in the requirement. Whereas the Negative Tests would contain the test data which would not return the output as expected, but the input would be as expected.

The Positive and Negative tests would ensure that the API works in both the scenarios, when the data returns expected response, and when the data doesn’t return the response as expected.

When it comes to Automation Testing, the most popular and commonly used tools are Postman, JMeter, or simply using a script, such as Python with the ”pytest” and ”request” modules.

3. Test the User Interface correctness

User Interface testing has always a broad scope when it comes to testing, but here are some of the essential scenarios that need to be covered when it comes to testing for correctness in the User Interface.

- **Field Validations**
  It should be ensured that the fields present in a certain page follows the right set of rules in the way it is intended to be.

  For example, an Email Address field must only allow the user to write a valid Email
  Address before the form is submitted to the server. Likewise, the password field must mask the characters being typed.

  It should also be validated that the field is not breaking the limit of the number of elements present in it. For example, a field that only allows up to 30 characters in a field must not allow more than 30 characters to be submitted to the server.

- **Dimensions of the fields/elements**
  The fields or the elements must have the correct dimensions, i.e., the width and height so that it can fit in the input being typed into that field, or it can show the text or any other content it is supposed to show.

  For an instance, an Email Address field must have appropriate dimensions so that the user can clearly see the input being typed. Likewise, when it comes to an element, such as a Button, or an image, it must be ensured that it has the right dimensions to fit in their content and keep it visible.

- **Correct use of colors and elements**
  Colors are one of the primary ways we perceive things in real world, which makes it important to use the colors the right way.

  For example, the red color must be used when something's being deleted, or when there's an error, etc. Likewise, the green color is supposed to be used when some task is successful, or when we have done something the right way.

  Furthermore, the different types of elements available in HTML are just as useful to convey the purpose or type of operation being performed.

  For example, for submitting a form, it would be a bad approach to use a checkbox or a link. For selecting a single option from a set of options, radio buttons are the way to go instead of using checkboxes.

- **Progress bars**
  Progress bars are a way of showing that the page is being loaded, or the operation is in progress. Adding progress bars at the right places helps users know when to wait for the operation to complete and when to continue their tasks.

- **Performance of the HTML, JS and CSS**
  In addition to the correct usage of elements, it is just as necessary that the elements are loaded as quickly as they can be loaded. For this, it is required that the CSS and JS files are minified and optimized well so that it doesn't take long to load in the DOM and be rendered by the browser.

- **Error Logging in the Developer Console**

There's always going to be a situation when the code doesn't work as expected and generates some errors, in which case the errors should be logged correctly in the browser console, and if needed, the logs should be sent back to the server as well.

The UI correctness check would generally involve performing a series of steps in order to open up the pages or the forms which need to be tested, which can be performed manually or can be automated as well.

For example, in order to check the Login Page, the following steps need to be performed in sequence along with a series of test data.

1. Open up the Login Page
2. Enter the Email Address
3. Enter the Password
4. Click on the Submit button

Another example can be of checking to see if the video player is working as it should.
For example, the video player should be visible and have proper height and width, it should have the player elements such as Pause/Play, Video Seeker, Volume changer, Full Screen/Restore buttons, Settings, Closed Captions(if available), etc. Furthermore, it should also support play/pause or changing to full screen by clicking on the video screen twice for full screen, or once for play or pause.

In the below example, for the sake of simplicity, the Login page is being checked for correctness by validating various attributes as well as the tag names, if necessary, in order to check if the elements are validating the inputs correctly and the right tags are being used.

Link to the code: https://github.com/harsiddhdave44/InterviewReady-Task/blob/master/checkUICorrectness.py