Test Approach

Initial Questions

It is essential that our team understands the requirements for the Loan for Debt Consolidation feature.

The requirements must be clear and unambiguous, preferably in the form of acceptance criteria that can be referenced at any time to ensure that all of the requirements have been tested successfully.

We need to have a clear understanding of the validation rules for each step within the Search and Compare process. Which fields are mandatory, conditional, or optional?

We need to understand how the list of search results is generated? Are we calling a back-end API? What information needs to be submitted to the API? What are the expected responses from the API (included error status codes)?

We need to understand the intended audience for this feature. This will help us to create user journeys for the site.

Hopefully, we already have metrics for which browsers and devices are used to access the site. This will help us to devote the appropriate amount of testing time to each of them.

Finally, what are the benchmarks of the performance tests for this site?

Components to be Tested

I am going to assume that the header and footer components of the site have already been tested. The following is what I would be looking to test:

- Each of the five steps of the Search and Compare process
- Loan Results page, including:
  - Successful results
  - No results
  - Results after updating loan amount and/or loan term
  - View & Apply button functionality
- Calls from the site to back-end API(s), if any
- Accessibility testing on each page within this feature
- Back-end API endpoints
- UI & API performance tests
- Security tests (including static, dynamic, and pen tests)

Test Plan

Unit & Functional Tests

I would expect the unit tests to be written by the developers (using NUnit or similar), and the functional tests to be written by the testers (using Specflow or similar). Therefore, the developers and testers should agree which test scenarios will be covered by unit tests, and which will be covered by functional tests, in order to avoid a duplication of work.

The functional tests for the UI should call a mocked version of the back-end API.

The functional tests for the API should call mocked versions of any third-party services.

In the interest of time, I will give a brief overview of the unit/functional tests that are required for the site.

The following tests are required for each field within the Search and Compare steps:

- Mandatory check
- Data type check
- Entered field value is within the valid range
- Invalid field values will be highlighted with an appropriate error message
- Conditional fields are only displayed when required
- Conditional fields are only validated when they are displayed
- Search boxes will display matching values only after the third letter is entered
    - If there are no matching values, then “No matches found” is displayed in the drop down list
- Help text is displayed with the appropriate wording when the user clicks on a Field Help icon

The following tests are required for the buttons within the Search and Compare steps:

- Continue button will go to the next step, if – and only if – all of the field values in the current step are valid
- See Your Results button will initiate the search, if – and only if – all 5 steps have been validated successfully
- See Your Results button will initiate the search by submitting the information entered by the user to the back-end API

The following tests are required on the Loan Results page:

- The back-end API will return the search results, which will be displayed Eligibility order by default
- The user can sort the results into APR Representative order
- The the user has the ability to update the entered information and re-submit the search
- If the back-end API returns an error status, then an appropriate error message is displayed
- The View & Apply button takes the user to appropriate Lender's page

The following tests are required by the back-end API:

- Authentication and/or Authorization checks
- Mandatory field checks on the information submitted (as there is no guarantee that the information has come directly from the UI)
- Search results checks based on the information submitted (not sure how this will be done)

Exploratory Tests

I would use exploratory testing for the scenarios that cannot be automated easily. This would include usability testing, unexpected scenarios (e.g. the back-end API is unavailable), cross-browser testing, device testing, and general sanity-checks (e.g. spelling mistakes).

Integration & End-to-End Tests

The purpose of the integration tests is to ensure that the different components of the site are "talking" to each other successfully. Therefore, we need to ensure that the UI is calling the back-end API successfully. The information returned by the API must be displayed (and sorted) successfully in the UI. If the API is using any third-party services, then we have to ensure that it is integrating successfully with them, etc.

The purpose of the end-to-end tests is to ensure that each user journey from entering the required information, receiving the results, and initiating the loan application can be completed successfully.

Regression & Smoke Tests

If we automate as many tests as possible then they can be used as regression tests by default. Every time a new piece of work is completed, it must pass the new tests written to ensure that it meets its requirements, plus it must pass previously written tests to ensure that it has not broken any other requirements.

Smoke tests can be scheduled to checkout the latest develop branch, and run the appropriate automated tests against it overnight. We don't have to run all of the automated tests, just a subset that proves that the Loan for Debt Consolidation functionality has not been broken. If the smoke tests pass, then we know that our code is in a releasable state. If they fail, then we have to fix any issues as a matter of priority.

Performance Tests

Performance testing will consist of UI and API performance tests to prove that the site is meeting a set of pre-defined benchmarks. We must resolve each issue where we are failing to meet the required standards of performance.

Security & Accessibility Tests

Security tests can be executed to ensure that the code, the UI, and the API do not have any vulnerabilities. Accessibility tests can be executed to ensure that the site is usable by all types of customers regardless of any disability.

Team Approach

I am assuming that the team will be using the Agile methodology of breaking down the Loan for Debt Consolidation feature into user stories. Each user story will contain clear and unambiguous acceptance criteria. The team will agree up-front on the Definition of Done, so that we are absolutely clear about when each story is complete and ready to be released to the Live environment.

Definition of Done

The following is a brief outline of what is required for a user story to be ready for release:

- Code development is complete
  - Unit tests are passing
  - Code has been peer-reviewed and any issues raised have been resolved
  - Code has been security checked and any issues raised have been resolved
  - Code has been deployed to the Test environment
- Testing is complete
  - Automated functional tests have been written and are passing
  - Exploratory tests (including cross-browser and usability tests) have been completed
  - If required, automated integration and end-to-end test have been written and are passing
  - If required, performance tests are written
  - Accessibility tests are passing
  - Security tests are executed and any issues raised are triaged
  - All tests are reviewed by team members including BA/PO and any issues raised have been resolved
  - Pull request is raised and any issues raised have been resolved
  - Code has been deployed to the CAT environment
- Partner Acceptance Testing is complete
  - Current story is demonstrated to BA/PO to ensure that all of the acceptance criteria have been met
  - Any issues raised will be resolved and re-demonstrated
  - BA/PO has agreed that the story is ready for release
  - Pull request is merged back to the develop branch
  - Develop branch is deployed to Pre-Live environment
- Pre-Live Testing is complete
  - Performance tests are executed and any issues raised are triaged
  - Develop branch is merged to the release branch, which is then deployed to Live

- Live Environment
    - Release branch has been on the Live environment for at least 1 day without any issues

Cross Browser and Device Testing Coverage

I am assuming that we have metrics on what browsers and devices are used to access the site. Therefore, I would ensure that the automated tests are running against the most popular browser and/or device. The other browsers and devices can be tested manually as basic usability tests. Alternatively, the can be tested by an automated cross-browser testing tool.

Other Considerations

Dynamic security testing and penetration testing should be scheduled to run as frequently as possible, and any issues raised must be triaged.

It makes sense to execute the performance tests against the Pre-Live environment, as I am assuming that this environment matches the set-up of the Live environment.

Depending on the story, the smoke tests may need to be updated/extended to include the new functionality.

We will use the appropriate reporting tools to document the results of the tests (see Testing Tools section).

Automated Testing vs Manual Testing

Ideally, we would want to automate as many tests as possible. However, manual testing is still required when it comes to exploratory and usability testing. Additionally, it may not be possible to automate certain test scenarios due to feasibility or complexity.

Testing Tools

Unit Tests

NUnit is a typical tool that developers use to write unit tests

Automated Tests

Specflow is a good tool for writing and executing automated tests – including functional, integration, end-to-end and smoke tests. Specflow also has a plug-in called Living Docs, which can be used to generate documentation for the executed tests including the date and time of execution as well as the pass and failure status of each test.

Performance Tests

JMeter is a good performance testing tool for APIs. It has a basic reporting capability, so we may need to write an application to create a more user-friendly version of the report that is generated.

I cannot recommend a performance testing tool for UIs because I have yet to use one.

Security Tests

Veracode is a good security testing tool. It can execute a static security scan on the development code (for both the UI and API) to check for any vulnerabilities. It can also execute a dynamic security scan on your web site by trawling through the pages in the site checking for vulnerabilities.

I cannot recommend a penetration testing tool as I have yet to use one.

Cross-Browser Tests

Applitools is an automated cross-browser testing tool that could be useful. I have seen it work in theory, but I haven't used it myself. It can validate a whole web page with a single screenshot assertion, and it does cover many versions of different browsers and devices.

Accessibility Tests

Selenium Axe is an automated testing tool that can be executed on an “AfterStep” hook in a SpecFlow test. It will analyse a web page for any accessibility issues and report them in a spreadsheet. You can set up your automated tests to fail, if Selenium Axe detects any serious or critical accessibility issues.

Final Thoughts

This may be way more detail than you required for this technical test, and I definitely spent more time on this than I anticipated, but I wanted to give a full answer so that I can learn from anything that I have missed.