Agile manual testing lab practice questions based on real-life scenarios:

Scenario: Your Agile team is developing a new feature for a mobile application that requires users to enter their personal information. How would you approach testing the input fields to ensure that the data is being validated and stored correctly?

Here is an approach that can be taken to test the input fields for personal information:

1. Identify the input fields that require personal information: First, identify all the input fields that require users to enter their personal information. For example, fields such as name, address, phone number, email address, date of birth, etc.

2. Define validation rules: Once the input fields have been identified, define the validation rules for each field. This includes checking the data type, format, length, and range of values. For example, the name field should only accept alphabetic characters, the email field should contain an '@' symbol, and the phone number field should only accept numeric characters.

3. Test boundary values: Test the boundary values of each input field. This involves entering the minimum and maximum values that are allowed for each field to ensure that the system can handle extreme values. For example, the date of birth field should not accept dates outside the expected range of values, such as dates in the future.

4. Test error messages: Test the error messages that are displayed when invalid data is entered into each field. This includes testing that the error messages are displayed correctly and that they provide clear instructions on how to correct the error.

5. Test data storage: Test that the data entered into each field is being stored correctly in the database. This includes checking that the data is being stored in the correct format, such as a date being stored as a date object and not as a string.

6. Regression testing: As new features are added to the application, it's important to perform regression testing on the input fields to ensure that changes have not affected the validation and storage of personal information.

7. Cross-device and cross-browser testing: Test the input fields on different devices and browsers to ensure that the user experience is consistent and that the input fields are working as expected on all platforms.

Scenario: Your Agile team is working on a project that requires the use of a chatbot. How would you test the chatbot to ensure that it is providing the correct responses to user queries?

Testing a chatbot involves several steps, including testing its functionality, accuracy, and performance. Here are some approaches that can be taken to test a chatbot:

1. Create test cases: Develop a set of test cases that cover the chatbot's expected behavior in different scenarios. These scenarios can be based on common user queries or issues that are likely to arise. The test cases should cover both positive and negative scenarios, such as valid and invalid user inputs.

2. Use automated testing tools: There are many automated testing tools available that can help test chatbots. These tools can simulate user input and validate the chatbot's responses. Some popular testing tools for chatbots include Botium, Chatbase, and Dialogflow.

3. Conduct manual testing: Manual testing involves using the chatbot as a user would and testing it in real-time. This approach can help uncover issues that automated testing may not identify. Testers can also provide feedback on the chatbot's responses and suggest improvements.

4. Collaborate with subject matter experts: Chatbots often require subject matter experts to help with the responses to specific user queries. Collaborating with these experts can help ensure that the chatbot is providing accurate and relevant responses.

5. Analyze performance metrics: It is important to analyze performance metrics such as response time, latency, and throughput to ensure that the chatbot is providing a good user experience. These metrics can help identify issues that need to be addressed.

6. Conduct user acceptance testing: Once the chatbot has been tested thoroughly, it is important to conduct user acceptance testing. This involves giving the chatbot to real users and getting their feedback. This feedback can be used to improve the chatbot and ensure that it meets the user's needs.

Scenario: Your Agile team is developing a new feature for a web application that requires the use of a file upload system. How would you test the file upload system to ensure that files are being uploaded correctly and that the correct file types are being accepted?

To test the file upload system for a web application, the following approach can be taken:

1. Test File Upload Functionality: Ensure that the file upload functionality is working as intended. This can be done by uploading a file of different types and sizes and checking that the files are successfully uploaded.

2. Test File Type Validation: Ensure that the file upload system only accepts the correct file types. This can be done by attempting to upload files of different types and checking that only the allowed file types are accepted. For example, if only .jpg and .png files are allowed, attempting to upload a .txt file should result in an error.

3. Test File Size Validation: Ensure that the file upload system only accepts files of the correct size. This can be done by attempting to upload files of different sizes and checking that only files within the allowed size range are accepted. For example, if the maximum file size allowed is 5MB, attempting to upload a file larger than 5MB should result in an error.

4. Test File Naming Conventions: Ensure that the file upload system uses a consistent naming convention for uploaded files. This can be done by uploading files with different names and checking that the files are renamed correctly.

5. Test Concurrent Uploads: Ensure that the file upload system can handle concurrent uploads without errors. This can be done by uploading multiple files at the same time and checking that all files are uploaded successfully.

Scenario: Your Agile team is working on a project that requires the use of a payment gateway that has specific requirements for payment information. How would you approach testing the payment information to ensure that it is being collected and processed correctly?

When testing the payment information for a project that requires the use of a payment gateway, here are some steps that can be taken to ensure that the payment information is being collected and processed correctly:

1. Review the payment gateway documentation to understand the requirements for collecting and processing payment information.

2. Create a test environment that simulates the production environment where payments will be processed.

3. Use test credit card numbers, bank account information, or other payment methods provided by the payment gateway to simulate transactions.

4. Test the system with both valid and invalid payment information to ensure that the system correctly handles errors and exceptions.

5. Verify that the payment information is being stored securely and in compliance with any relevant regulations.

6. Test the system under load to ensure that it can handle a large number of transactions and that payments are processed quickly and accurately.

7. Ensure that appropriate error messages are displayed to the user when there is an issue with the payment information.

8. Monitor the system for any errors or issues that may arise during real-world usage and address them as necessary.

Scenario: Your Agile team is developing a new feature for a web application that requires the use of location data. How would you test the location data to ensure that it is being retrieved correctly and accurately?

To test the location data in a web application, the following approach can be taken:

1. Use real and simulated location data: To test the location data, you can use real data by physically moving around and checking if the application updates the location correctly. Alternatively, you can simulate location data using tools like MockLocation or GPX simulators.

2. Test in various locations: Testing the location data in different locations can help ensure that the application is working as expected in different environments. This can be done by testing in different cities, countries, or even continents.

3. Test in different network conditions: Network conditions can affect the accuracy of location data. Therefore, testing in different network conditions, such as 3G, 4G, or Wi-Fi, can help ensure that the location data is being retrieved correctly.

4. Test with different devices: Different devices can have different GPS sensors, which can affect the accuracy of location data. Therefore, testing with different devices can help ensure that the location data is being retrieved correctly across different platforms.

5. Test with various location-based features: The web application may have different features that rely on location data, such as displaying nearby places or calculating distances. Testing these features can help ensure that the location data is being retrieved accurately and is being used correctly in the application.

6. Perform boundary testing: Boundary testing involves testing the limits of the location data, such as testing the minimum and maximum latitudes and longitudes. This can help ensure that the application can handle location data within the expected range.

7. Perform regression testing: After each change or update to the application, perform regression testing to ensure that the location data is still being retrieved correctly and accurately.

Scenario: Your Agile team is working on a project that requires the use of a database. How would you approach testing the database to ensure that data is being retrieved and stored correctly?

To approach testing the database, here are some steps that can be taken:

1. Create test data: In order to test the database, it's important to have a set of test data that can be used to ensure that the database is functioning as expected. The test data should include a range of values that cover all possible scenarios.

2. Write test cases: Write test cases that cover all the database operations that need to be tested. These test cases should include both positive and negative scenarios.

3. Test database connectivity: Test the database connectivity by connecting to the database and checking if it is able to retrieve data. This ensures that the database is properly configured and accessible.

4. Test database schema: Test the database schema to ensure that it is properly defined and all the tables, columns, and relationships are correctly set up.

5. Test database operations: Test the database operations such as insert, update, and delete to ensure that the data is being stored and retrieved correctly.

6. Test performance: Test the performance of the database by loading a large amount of data and verifying that it is being processed efficiently.

7. Test backups and recovery: Test backups and recovery by simulating a database failure and restoring the database from a backup to ensure that the backup process is working correctly.

8. Conduct integration testing: Conduct integration testing with other parts of the system to ensure that the database is properly integrated and functioning with the rest of the system.

Scenario: Your Agile team is developing a new feature for a mobile application that requires users to navigate through multiple screens. How would you test the navigation to ensure that users can easily move between screens and that the application does not crash?

To test the navigation in the mobile application, you can follow these steps:

1. Create a test plan: Create a test plan that outlines the scenarios that a user might encounter while navigating through the application. This should include the various paths a user can take, as well as any potential errors or edge cases that may occur.

2. Perform manual testing: Start with manual testing by using the application on various mobile devices and operating systems. This can help identify any major issues with the navigation, such as buttons not functioning or screens not displaying correctly.

3. Use automated testing tools: Consider using automated testing tools like Appium, Espresso, or XCUITest to test the application's navigation. These tools can simulate user actions and verify that the application is responding correctly.

4. Conduct usability testing: Conduct usability testing with actual users to gather feedback on the navigation. This can help identify any areas where users may have difficulty navigating or where the application may not be intuitive.

5. Perform load testing: Finally, perform load testing to ensure that the application can handle multiple users navigating through it simultaneously without crashing or experiencing slow response times.

Scenario: Your Agile team is working on a project that requires the use of a search feature. How would you test the search feature to ensure that it is retrieving the correct results and that the results are being displayed correctly?

To test the search feature, I would approach the following steps:

1. Identify the search requirements: The first step is to understand the search requirements, such as the type of search (full-text, keyword, fuzzy, etc.), the search criteria, and the expected results.

2. Test the search functionality: The next step is to test the search functionality to ensure that it is working as expected. This includes testing the search functionality with different search criteria, ensuring that it is returning accurate results, and handling edge cases such as empty or invalid search queries.

3. Test the display of search results: Once the search functionality has been tested, the next step is to test the display of search results. This includes verifying that the search results are displayed correctly, including the order of the results and the formatting of the results.

4. Test the sorting and filtering of search results: Many search features allow users to sort or filter their search results. It is important to test these features to ensure that they are working correctly.

5. Test the performance of the search feature: The search feature should be tested for performance to ensure that it can handle a large number of search requests and return results quickly. Load testing and stress testing can be used to test the performance of the search feature.

6. Test the integration with the rest of the application: Finally, it is important to test the integration of the search feature with the rest of the application. This includes testing how the search feature interacts with other features and how it handles errors or unexpected inputs.

Scenario: Your Agile team is developing a new feature for a web application that requires the use of forms. How would you test the forms to ensure that they are collecting the correct data and that the data is being validated correctly?

To test the forms in a web application, I would approach it in the following way:

1. Test the form fields: First, I would test each field in the form to ensure that it is collecting the correct data type and that it is properly validated. This would include testing required fields to ensure that they cannot be left blank, validating email addresses, phone numbers, and other user inputs.

2. Test form submission: After testing the individual fields, I would test the form submission process to ensure that the data is being correctly processed and stored in the database. This would involve testing the form with both valid and invalid inputs to ensure that the validation is working correctly and that the form is correctly handling errors.

3. Test form response messages: I would also test the response messages displayed to the user after submitting the form to ensure that they are clear and informative. For example, if the user submits an invalid email address, the error message should clearly explain what is wrong with the email address and how to correct it.

4. Test form layout and design: Finally, I would test the form layout and design to ensure that it is user-friendly and easy to navigate. This would include testing the form on different screen sizes and devices to ensure that it is responsive and accessible to all users.