**Brand Engagement Network Inc. Coding Test**

Part 1: Test Case Writing Scenario: A new feature in a web application allows users to register an account. The registration form includes fields for username, password, email, and a "Subscribe to newsletter" checkbox.

Task: Write a set of test cases covering functional, boundary, and edge cases.

Ans. Functional Testcases:

1. Username:

* Valid username with alphanumeric characters (e.g., "user123")
* Username with allowed special characters (e.g., "user\_name")
* Username at minimum length requirement
* Username at maximum length requirement
* Username with mixed case (e.g., "UserName123")
* Empty username field
* Username with disallowed special characters (e.g., "@#$%")
* Username with spaces
* Username shorter than minimum length
* Username longer than maximum length
* Username containing only numbers
* Duplicate username (already registered)

1. Password:

* Password meeting minimum complexity requirements
* Password with special characters, numbers, and mixed case
* Password at minimum length requirement
* Password at maximum length requirement
* Empty password field
* Password shorter than minimum length
* Password longer than maximum length
* Password without required character types (missing numbers/special chars)
* Password with spaces
* Password matching username

1. Email:

* Standard email format ([user@domain.com](mailto:user@domain.com))
* Email with subdomains ([user@sub.domain.com](mailto:user@sub.domain.com))
* Email with numbers and special characters
* Empty email field
* Invalid email format (missing @ symbol)
* Invalid email format (missing domain)
* Invalid email format (multiple @ symbols)
* Invalid domain format
* Duplicate email (already registered)

1. Newsletter Subscription:
   * Checkbox checked
   * Checkbox unchecked
   * Toggle checkbox multiple times before submission

Edge Test Cases:

* Username with Special Characters
* Password with Special Characters
* Email with Special Characters
* HTML/JavaScript Injection
* Very Long Inputs
* Empty Checkbox State

Part 2: Bug Identification and Reporting Scenario: having the page hosted in this server: http://13.209.85.69/ Task: Identify, document, and report the bugs. Emphasize clarity and detail in the bug report, including steps to reproduce, expected vs. actual results, and severity assessment.

Ans. Bugs found:

1. Username with only numbers is accepted

Description - If you provide username with only numbers it is getting accepted which shouldn’t be allowed as it is invalid.

Actual- Username containing only numbers is accepted as a username

Expected – Only valid username should be allowed like containing alphanumeric characters or some allowed special characters.

Severity – Medium

1. Username with only special characters is getting accepted e.g ‘12345’

Description - If you provide username with only special characters it is getting accepted which shouldn’t be allowed as it is invalid.

Actual- Username with only numbers is accepted as a username like ‘!@#$%’

Expected – Only valid username should be allowed like containing alphanumeric characters or some allowed special characters. E.g (user123, user\_123, user1)

Severity – High

1. Invalid email format is getting accepted

Description – Invalid email format for e.g [richa@+gmail.com](mailto:richa@+gmail.com), richa@123 are getting accepted

Actual – Invalid email formats are getting accepted

Expected - Email validation should be in place which should allow only valid emails like [richa@gmail.com](mailto:richa@gmail.com), [richa123@domain.com](mailto:richa123@domain.com)

Severity - High

1. No maximum length check on password, username and email field

Description – Very long inputs on password, username and email field are accepted

Actual - Very long inputs on password, username and email field are accepted

Expected – There should be maximum length defined for all the input fields

Severity – high

1. Password doesn’t have any complexity requirements

Description – Any password which is longer than 8 characters is accepted as in if you only provide numbers or only alphabets or just the special characters as long as it is longer than 8 characters it is accepted which is wrong, password should have some complexity requirements.

Actual - Any password which is longer than 8 characters is accepted e.g 12345678, abcdefgh, !@#$%^&

Expected – Password should have some complexity requirements.

Part 3: Automated Testing Challenge (optional writing the full script) Scenario: having the page hosted in this server: http://13.209.85.69/ Task: Write a script that automates the test for the page hosted above.

Ans. Python file will be uploaded in the github

Part 4: Performance Testing Scenario Scenario: You are given a basic cloud service that is the entry point for our mobile app. This service is responsible for signing up and signing in users and related helper functionalities. This application is expected to handle up to 10,000 concurrent users during peak hours. Task: Develop a performance testing plan outlining the following aspects: 1. Objectives: Define what you aim to achieve with this performance test (e.g., maximum load handling, response time under load). 2. Tools: Choose a performance testing tool (e.g., JMeter, LoadRunner) and justify your choice. 3. Test Cases: Describe at least three test cases that you would execute (e.g., searching for books, viewing book details, adding books to the cart). 4. Metrics: List the key metrics that you would monitor (e.g., response time, error rate, throughput). 5. Test Environment: Describe the setup of your test environment (e.g., server configuration, network settings). 6. Analysis: Explain how you would analyze the results of the performance test.

### Performance Testing Plan for Cloud Service

1. Objectives:

* Verify the system can handle 10,000 concurrent users during peak load
* Measure and ensure that the response time for key operations (e.g., sign-up, sign-in) remains within acceptable limits (e.g., under 2 seconds) under peak load.
* Verify that the system remains stable and recovers gracefully after peak load conditions.
* Assess the system's ability to scale horizontally or vertically to handle increased load.

1. Tools:

Selected Tool – Apache JMeter

Justification:

* Open-source with extensive plugin ecosystem
* Strong support for REST API testing
* Built-in reporting and monitoring capabilities
* Distributed testing capability for high concurrent user simulation
* Active community support and documentation

1. Test Cases:

* **User Sign-Up**:

Description: Simulate multiple users signing up simultaneously.

Objective: Measure the response time and success rate of the sign-up process under load.

Parameters: Number of concurrent users, input data (e.g., username, password, email).

* **User Sign-In**:

Description: Simulate multiple users signing in simultaneously.

Objective: Measure the response time and success rate of the sign-in process under load.

Parameters: Number of concurrent users, input data (e.g., username, password).

* **Password Reset**:

Description: Simulate multiple users requesting a password reset simultaneously.

Objective: Measure the response time and success rate of the password reset functionality under load.

Parameters: Number of concurrent users, input data (e.g., email).

1. Metrics

* **Response Time**: The time taken for the server to respond to a request. Target: < 2 seconds for key operations.
* **Error Rate**: The percentage of requests that result in an error. Target: < 1%.
* **Throughput**: The number of requests processed per unit of time. Target: High throughput to handle 10,000 concurrent users.
* **CPU and Memory Usage**: Monitor server resource utilization to ensure it remains within acceptable limits.
* **Latency**: The time taken for a request to travel from the client to the server and back.
* **Concurrent Users**: The number of users actively using the system at the same time.
* **Request Success Rate**: The percentage of successful requests out of the total requests made.

1. Test Environment

* **Server Configuration**:
  + **Hardware**: Use servers with specifications similar to production (e.g., 8-core CPU, 16GB RAM).
  + **Software**: Ensure the same OS, middleware, and database versions as in production.
  + **Load Balancer**: If applicable, configure a load balancer similar to the production environment.
* **Network Settings**:
  + **Bandwidth**: Simulate network conditions similar to production (e.g., 100 Mbps).
  + **Latency**: Introduce network latency if necessary to mimic real-world conditions.
* **Database**: Use a database with a similar size and configuration as the production database.
* **Test Data**: Generate realistic test data (e.g., user profiles, credentials) to simulate real-world usage.

#### Analysis

* **Data Collection**: Collect metrics from JMeter, server logs, and monitoring tools (e.g., New Relic, Prometheus).
* **Response Time Analysis**: Compare the response times against the target (e.g., < 2 seconds). Identify any operations that exceed the target and investigate the cause.
* **Error Rate Analysis**: Analyze the error rate to identify any patterns or specific conditions that cause errors. Investigate and resolve any issues.
* **Resource Utilization**: Review CPU, memory, and disk usage to identify any bottlenecks. Consider scaling options if resources are maxed out.
* **Throughput Analysis**: Ensure that the system can handle the expected throughput. If not, investigate potential optimizations or scaling strategies.
* **Scalability Assessment**: Determine if the system can scale horizontally (adding more servers) or vertically (upgrading server resources) to handle increased load.
* **Report Generation**: Compile a detailed report with findings, including graphs, tables, and recommendations for improvements.

Part 5: Problem-Solving Questions

Scenario 1: During a release cycle, you find that the number of bugs reported in the new features is significantly higher than in the previous releases. The development team claims that the features were developed according to specifications. Task: Write a response detailing how you would handle this situation, including steps to identify the root cause and possible solutions to reduce the number of bugs in future releases.

### Gather Data and Analyze the Bugs

* **Collect Bug Reports**: Compile all bug reports related to the new features, including details such as severity, frequency, and the specific areas of the code or functionality affected.
* **Categorize Bugs**: Group the bugs into categories (e.g., functional, UI/UX, performance, integration, etc.) to identify patterns or common themes.
* **Compare with Previous Releases**: Analyze the bug trends from previous releases to understand if there are specific areas where the current release deviates.

### Review the Development Process

* **Verify Specifications**: Cross-check the implemented features against the original specifications to ensure alignment. If discrepancies are found, determine whether they stem from unclear requirements or misinterpretation.
* **Assess Code Quality**: Conduct a code review to identify potential issues such as poor coding practices, lack of modularity, or insufficient error handling.
* **Evaluate Testing Processes**:
  + Review the test cases to ensure they cover all aspects of the new features.
  + Check if the testing environment accurately reflects the production environment.
  + Assess whether automated tests were sufficient or if additional manual testing is needed.

### Engage the Development Team

* **Collaborate with Developers:** Work closely with the development team to understand their perspective and challenges during the development process.
* **Identify Gaps**: Discuss potential gaps in the development lifecycle, such as inadequate unit testing, insufficient peer reviews, or time constraints that may have impacted quality.
* **Encourage Ownership**: Foster a culture of accountability where developers take ownership of both the code they write and the bugs that arise from it.

### Identify the Root Cause

* Conduct a Root Cause Analysis (RCA): Use techniques like the *5 Whys* or *Fishbone Diagram* to drill down into the underlying reasons for the increased bug count.
* Common Root Causes:
* Incomplete or ambiguous requirements.
* Insufficient testing coverage.
* Lack of communication between teams (e.g., development, QA, and product management).
* Tight deadlines leading to rushed development or testing.
* Technical debt or legacy code issues.

1. Implement Solutions

### Based on the root cause analysis, implement the following solutions to reduce bugs in future releases:

* Improve Requirements Gathering**:**
  + Ensure requirements are clear, detailed, and reviewed by all stakeholders before development begins.
  + Use prototypes or mockups to validate requirements with stakeholders early in the process.
* Enhance Testing Processes:
  + Increase test coverage, including edge cases and negative scenarios.
  + Implement automated regression testing to catch issues early.
  + Conduct exploratory testing to uncover unexpected issues.
* Strengthen Code Quality Practices:
  + Enforce coding standards and conduct regular code reviews.
  + Use static code analysis tools to identify potential issues before deployment.
  + Refactor code to reduce technical debt and improve maintainability.
* Improve Collaboration:
  + Foster better communication between development, QA, and product teams.
  + Conduct regular sync meetings to address issues and align on priorities.
* Allocate Adequate Time:
  + Ensure realistic timelines for development and testing to avoid rushed work.
  + Include buffer time for bug fixes and regression testing.
* Provide Training:
  + Offer training sessions for developers on best practices for writing clean, maintainable code.
  + Train QA teams on advanced testing techniques and tools.

### **6.** Monitor and Measure Progress

* Track Bug Metrics: Monitor the number, severity, and resolution time of bugs in subsequent releases to assess the effectiveness of the implemented solutions.
* Conduct Retrospectives: After each release, hold a retrospective to identify what went well and what needs improvement.
* Iterate and Improve: Continuously refine processes based on feedback and metrics to ensure sustained improvement.

Scenario 2: You are part of a QA team that is transitioning from manual to automated testing. However, some team members are resistant to this change, citing a lack of experience in programming. Task: Describe how you would address this challenge to ensure a smooth transition to automated testing while keeping the team motivated and engaged.

Ans To ensure a smooth transition to automated testing I will try to do the following things:

### 1. **Acknowledge Concerns and Build Trust**

* **Listen to their concerns:** Hold a team meeting to openly discuss their fears and reservations about the transition. Acknowledge that learning new skills can be intimidating.
* **Highlight the benefits:** Explain how automated testing can reduce repetitive tasks, improve efficiency, and allow them to focus on more strategic and creative aspects of testing.
* **Emphasize support:** Assure them that they will not be left to figure things out on their own and that the transition will be gradual.

### Provide training and resources

* **Offer beginner-friendly training: Start with basic programming and automation concepts, using tools that are easy to learn (e.g., Selenium, Cypress, or Katalon). Provide hands-on workshops or online courses tailored to their skill levels.**
* **Pair programming and mentoring: Pair less experienced team members with those who have programming knowledge to foster collaboration and knowledge sharing.**
* **Create a learning roadmap: Break down the learning process into manageable steps, setting achievable milestones to build confidence.**

### **Start Small and Scale Gradually**

* **Pilot projects:** Begin with small, low-risk projects to allow the team to practice and gain confidence in automated testing.
* **Incremental adoption:** Gradually introduce automation into existing workflows, allowing team members to see the benefits firsthand without feeling overwhelmed.
* **Celebrate small wins:** Recognize and celebrate progress, no matter how small, to keep morale high.

### **Foster a Collaborative Environment**

* **Encourage teamwork:** Promote a culture where team members help each other and share knowledge. Create a safe space for asking questions and making mistakes.
* **Involve the team in tool selection:** Let them participate in choosing the automation tools and frameworks, ensuring they feel ownership over the process.

### 5. **Address Skill Gaps with External Support**

* **Hire or consult experts:** If feasible, bring in automation experts to guide the team and provide advanced training.
* **Leverage external resources:** Use online communities, forums, and tutorials to supplement internal training.

### 6. **Showcase the Impact of Automation**

* **Demonstrate ROI:** Share metrics and success stories from other teams or organizations that have successfully adopted automated testing.
* **Highlight personal growth:** Emphasize how learning automation can enhance their skill sets and career prospects.

Part 6: Tool Proficiency Task 1 (JIRA): Given a scenario where multiple high-priority bugs have been reported in a new feature, draft a bug report for one of these bugs in JIRA. Include fields like Summary, Description, Steps to Reproduce, Expected Result, Actual Result, Priority, and Attachments (if applicable).

JIRA BUG REPORT

* **Summary:**  
  Critical UI Crash on Checkout Page When Applying Promo Code
* **Description:**  
  When users attempt to apply a promo code during checkout, the page crashes, preventing them from completing their purchase. This issue affects multiple users across different browsers and devices, making it a high-priority bug.
* **Steps to Reproduce:**
  1. Go to the e-commerce website and log in.
  2. Add any product to the cart and proceed to checkout.
  3. Enter a valid promo code in the "Apply Promo Code" field.
  4. Click the "Apply" button.
* **Expected Result:**  
  The promo code should be successfully applied, and the discount should be reflected in the total price.
* **Actual Result:**  
  The page crashes with a blank screen, and users are unable to proceed with the checkout. Console logs show a JavaScript error:  
  "TypeError: Cannot read properties of null (reading 'discount')".
* **Priority:**  
  **High** (Blocking users from completing purchases)
* **Attachments:**
  1. Screenshot of the error message
  2. Console log output
  3. Screen recording of the bug reproduction
* **Environment:**
  1. OS: Windows 11, macOS Ventura
  2. Browsers: Chrome (v133.0), Firefox (v134.0.2), Edge (v118.0)

Task 2 (Postman): Create a collection in Postman to test a simple REST API (e.g., a CRUD API for users in our app). The collection should include requests for creating a new user, retrieving information about a user, updating a user, and deleting a user. Include tests in each request to validate the response status code and response body.

Ans. Will attach the postman collection file in the github