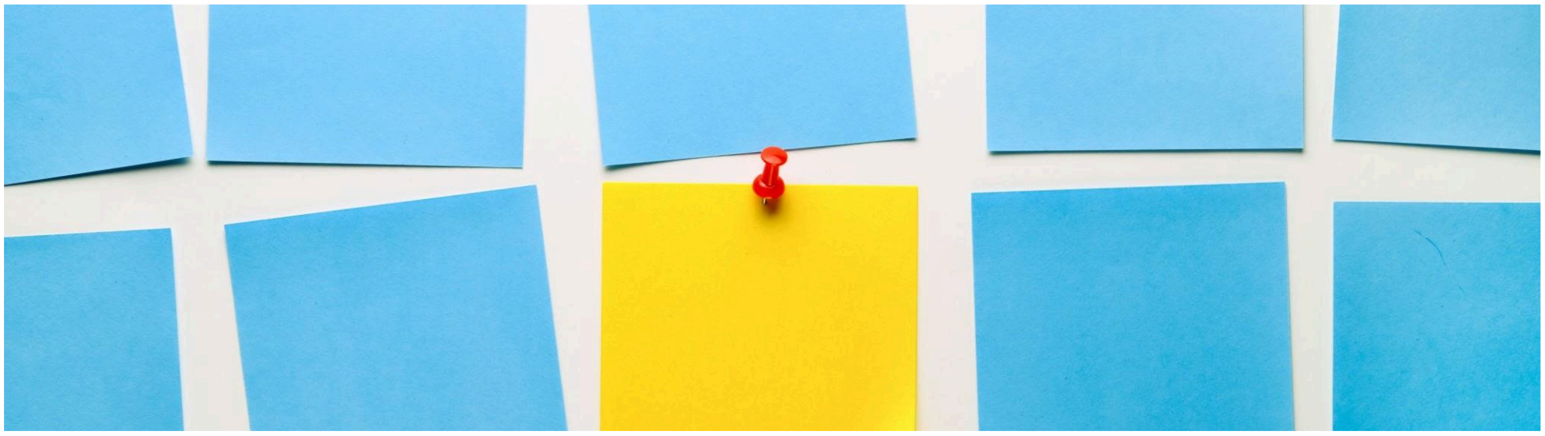

TEST PLAN DOCUMENT

Overview

Project Scope Summary: QA for "ShopEz" Payment Gateway

The goal of this project is to ensure the successful launch of "ShopEz's" new payment gateway by validating its functionality, security, reliability, and user-friendliness across supported payment methods: credit/debit cards, digital wallets, and bank transfers. The QA process encompasses the following:

Functional Validation:

- Verify that all core functionalities, including payment initiation, processing, and confirmation, perform as expected.

- Ensure Compatibility with multiple payment options (credit/debit cards,digital wallets,bank transfers).

User Experience (UX) Testing:

- Assess the user-friendliness and accessibility of the payment gateway on both desktop and mobile platforms.
- Validate intuitive navigation and clear error messages for users.

Security Assurance:

- Confirm compliance with industry standards (e.g., PCI DSS) and encryption protocols for safeguarding user data.
- Test measures against vulnerabilities such as transaction fraud and data breaches.

Performance and Reliability Testing:

- Evaluate the system's ability to handle high transaction volumes without performance degradation.
- Simulate different network conditions to verify gateway responsiveness and stability.

Edge Case and Error Handling:

- Identify and test edge cases, such as incorrect card details, expired cards, or interrupted transactions.
- Verify proper system recovery and user notifications for failure scenarios.

Objective: Ensure the payment gateway performs reliably, securely, and is user-friendly for all payment methods (credit/debit cards, digital wallets, bank transfers).

Test Design

Test Design Objectives ;

- Ensure all payment methods work
- Validate taxes ,discounts and shipping costs are calculated correctly

- Simulate situations like cancelled transactions andre network failures.
- Identify scenarios where transactions might fail and describe the expected system response
- Analyze error messages for clarity and usefulness.
- Consider potential risk such as data breaches or sensitive data exposure
- Ensure the gateway follows the best practices such as encrypting credit card information and compiling with PCI DSS standards.
- Ensure the site is user friendly .

TEST CASES DOCUMENT:

Below is the excel document on the various possible test cases based on priority or severity .
Navigate to the link below:

TEST CASE EXEL DOCUMENT

Test Cases:

Test Case 1: Credit/Debit Card Payment (Valid Transaction)

- **Objective**: Verify that a valid card payment is processed successfully.

- **Pre-condition**: User has a valid credit/debit card.
- **Steps**:
  - Select credit/debit card as payment method.
  - Enter valid card number, expiry date, CVV, and billing address.
  - Click the "Pay" button.
- **Expected Result**:
  - Payment is processed successfully.
  - User receives confirmation of payment.
  - Transaction ID is generated.
- **Post-condition**: Payment successfully reflected in user's order status.

Test Case 2: Credit/Debit Card Payment (Invalid Card Number)

- **Objective**: Verify that invalid card details are handled correctly.
- **Pre-condition**: User attempts payment with an invalid credit/debit card number.
- **Steps**:
  - Select credit/debit card as payment method.
  - Enter an invalid card number (e.g., incorrect digits).
  - Enter valid expiry date and CVV.

  - Click "Pay" button.
- **Expected Result**:
  - Payment fails with an appropriate error message (e.g., "Invalid card number").
  - No charge is made to the user’s account.
  - The system prompts the user to re-enter valid card details.

Test Case 3: Digital Wallet Payment (Valid Transaction)

- **Objective**: Verify that a valid digital wallet transaction is successful.
- **Pre-condition**: User has a valid account in a supported digital wallet (e.g., PayPal, Google Pay).
- **Steps**:
  - Select digital wallet as payment method.
  - Authenticate with the wallet account (e.g., sign in to PayPal).
  - Confirm the payment.
- **Expected Result**:
  - Payment is processed successfully.
  - Transaction confirmation is displayed to the user.
  - Transaction ID is generated.

Test Case 4: Bank Transfer Payment (Successful Transfer)

- **Objective**: Verify successful bank transfer payment.
- **Pre-condition**: User selects bank transfer as a payment method.
- **Steps**:
  - Select bank transfer as payment method.
  - Provide bank details .
  - Click "Confirm Transfer.
  - Receive bank account confirmation with the payment details.
- **Expected Result**:
  - Bank transfer is processed successfully.
  - User receives confirmation for the successful transfer.
  - Transaction ID is generated.

Test Case 5: Security Test (SQL Injection)

- **Objective**: Test for SQL injection vulnerabilities in the payment gateway.
- **Pre-condition**: User is on the payment page.
- **Steps**:
  - Enter a malicious SQL query
  - Submit the payment request.
- **Expected Result**:

    - The system should prevent the query from executing and return an error message.
    - No unauthorized access or data manipulation should occur.

Test Case 6: Security Test (Cross-Site Scripting – XSS)

- **Objective**: Test for XSS vulnerabilities in the payment page.
- **Pre-condition**: User is on the payment page.
- **Steps**:
    - Enter a malicious JavaScript snippet into any input field.
    - Submit the payment request.
- **Expected Result**:
    - The system should sanitize inputs and not execute any script.
    - No alert box or unusual behavior should be triggered.

Test Case 7: Performance Test (Load Testing)

- **Objective**: Verify how the payment gateway performs under heavy load.

- **Pre-condition**: The system is prepared to simulate multiple users (e.g., 50 concurrent users).
- **Steps**:
  - Simulate 50 concurrent payment transactions using a load testing tool.
  - Monitor response times, system resource usage, and transaction success rate.
- **Expected Result**:
  - The system should handle the load without significant slowdowns or failures.
  - Response time should stay within acceptable limits

Test Case 8: Payment Retry After Failure

- **Objective**: Test the system's ability to handle payment retry after a failed transaction.
- **Pre-condition**: User attempts to make a payment, but the transaction fails (e.g., insufficient funds).
- **Steps**:
  - Select credit/debit card as payment method.
  - Attempt a payment with insufficient funds or incorrect card details.
  - Receive a failure message with retry options.
  - Retry payment with valid details.

- **Expected Result**:
  - System allows retry with valid payment information.
  - Payment is successfully processed after retry.
  - The user receives an updated confirmation.

BUG REPORT DOCUMENT

1. **Bug: Payment Gateway Unavailable**
   - **Test Case ID**: TC013
   - **Description**: The third-party payment gateway fails to respond or returns an error, causing the transaction to be unsuccessful.
   - **Steps to Reproduce**: Simulate third-party gateway failure (e.g., PayPal or Stripe downtime).
   - **Expected Behavior**: The system should show an appropriate error message like "Payment Gateway Unavailable" and allow the user to try another payment method or retry later.
   - **Bug**: The system does not display the correct error message, and the user is left with a blank screen or an unclear error message.
2. **Bug: SQL Injection Vulnerability**

- **Test Case ID**: TC005
- **Description**: The payment form is vulnerable to SQL injection.
- **Steps to Reproduce**: Enter SQL injection code into the credit card number field.
- **Expected Behavior**: The input should be sanitized, and the transaction should be rejected with an error message like "Invalid input."
- **Bug**: SQL code is executed, and the database may be manipulated or exposed.

3. **Bug: Invalid Card Number Handling**
   - **Test Case ID**: TC002
   - **Description**: The system does not reject an invalid card number correctly.
   - **Steps to Reproduce**: Enter an invalid card number (e.g., incorrect number of digits or non-existent number).
   - **Expected Behavior**: Payment should be rejected with an error message like "Invalid card number."
   - **Bug**: The system accepts the invalid card number and proceeds to the next step, potentially causing further errors or incorrect transaction data.
4. **Bug: Cross-Site Scripting (XSS) Vulnerability**

- **Test Case ID**: TC006
- **Description**: The payment page is vulnerable to XSS attacks.
- **Steps to Reproduce**: Enter JavaScript code in the credit card or CVV field and submit the payment.
- **Expected Behavior**: The JavaScript should be sanitized, and no script should execute.
- **Bug**: The script executes, causing a potential security breach or unexpected behavior on the payment page.

5. **Bug: Incorrect Error Handling for Payment Retry**
    - **Test Case ID**: TC008
    - **Description**: The retry option does not work properly after a failed transaction.
    - **Steps to Reproduce**: Enter incorrect card details to simulate a payment failure. Click "Retry Payment" after receiving an error message and re-enter correct card details.
    - **Expected Behavior**: The system should allow the user to retry the payment with correct details and successfully process the transaction.

- **Bug**: The "Retry Payment" option does not allow retry or reverts to the previous error message without allowing the user to proceed with the transaction.

6. **Bug: Email/SMS Confirmation Not Received**
    - **Test Case ID**: TC009
    - **Description**: The user does not receive an email or SMS confirmation after a successful payment.
    - **Steps to Reproduce**: Complete a valid transaction using any payment method and wait for the email/SMS confirmation.
    - **Expected Behavior**: The user should receive an email or SMS confirmation within 5 minutes.
    - **Bug**: No confirmation email/SMS is received even after the transaction is completed successfully.
7. **Bug: Payment Retry Does Not Handle Insufficient Funds Properly**
    - **Test Case ID**: TC008
    - **Description**: The system does not provide a proper error message or retry mechanism for transactions that fail due to insufficient funds.
    - **Steps to Reproduce**: Attempt a payment with a credit card that has insufficient funds.

- **Expected Behavior**: The user should receive an error message like "Insufficient funds" and be prompted to retry or choose another payment method.
- **Bug**: The system throws a generic error or allows the user to proceed without proper messaging.

8. **Bug: Currency Conversion Error**
   - **Test Case ID**: TC012
   - **Description**: The system incorrectly converts the amount when switching currencies.
   - **Steps to Reproduce**: Select a different currency (e.g., EUR or GBP) for a transaction, then proceed with the payment.
   - **Expected Behavior**: The transaction amount should correctly reflect the selected currency based on current exchange rates.
   - **Bug**: The system either shows an incorrect converted amount or does not convert the amount at all.
9. **Bug: Poor User Interface (UI) on Mobile Devices**
   - **Test Case ID**: TC011
   - **Description**: The payment page is not responsive on mobile devices, causing elements to be misaligned or unreadable.

- **Steps to Reproduce**: Open the payment page on a mobile device and attempt to complete a transaction.
- **Expected Behavior**: The payment page should be fully responsive, and all elements (buttons, text fields) should be clearly visible and functional.
- **Bug**: The page layout is broken, and users cannot easily enter payment details or navigate through the process.

10. **Bug: Concurrent Payment Race Conditions**
    - **Test Case ID**: TC014
    - **Description**: Multiple users performing transactions simultaneously causes data corruption or race conditions.
    - **Steps to Reproduce**: Simulate multiple concurrent transactions (e.g., 50 users making payments at the same time).
    - **Expected Behavior**: The system should handle concurrent payments without data corruption or inconsistencies.
    - **Bug**: Race conditions occur, resulting in duplicate transaction IDs or data inconsistencies.

Bug Report Template for Bug:TC005

Field	Details
Bug ID	TC-005
Title	SQL Injection Vulnerability In Payment Form
Description	The payment form is vulnerable to SQL injection, allowing attackers to input malicious SQL code and gain unauthorized access to the database.
Repro Steps	1. Navigate to the payment page. 2. Enter SQL injection payload (e.g., `' OR 1=1 --`) in the card number field. 3. Submit the payment form. 4. Observe that the form processes the input without validation.
Priority	High
Severity	Critical

Repro Rate	100
State	Active
Assigned To	Quality Owner
Created On Behalf Of	Faith Kimani
Date Reported	12/03/2024

SECURITY AND USABILITY ANALYSIS

Security and Usability Analysis with Recommendations for Improvement

---

1. Security Analysis:

Key Focus Areas:

- **Data Protection**
- **Encryption and Tokenization**
- **Compliance with PCI DSS Standards**
- **Vulnerability Testing**

Current Security Practices:

1. **Sensitive Data Encryption**:

- **Analysis**: The payment gateway may be encrypting sensitive data (e.g., credit card details) using HTTPS/TLS during transmission. However, ensuring that the encryption is strong (e.g., AES-256) is essential to prevent data breaches.
- **Recommendation**: Conduct regular security audits and penetration tests to ensure data encryption methods are up to industry standards. Ensure the payment gateway uses **strong encryption algorithms** (e.g., AES) for storing and transmitting sensitive data.

2. **Encryption of Payment Information**:

    - **Analysis**: If credit card information is not stored or transmitted securely, there is a significant risk of data theft, especially in the case of network breaches.
    - **Recommendation**: Implement **tokenization** for credit card information, where card details are replaced with tokens to prevent storing raw card data. Ensure that any stored payment information is **fully encrypted** and adheres to PCI DSS requirements.

3. **PCI DSS Compliance**:

    - **Analysis**: The gateway must meet all **PCI DSS standards**, including encryption of payment data, minimal storage of sensitive information, and strong access control.
    - **Recommendation**: Conduct a **PCI DSS compliance audit** regularly to ensure the payment gateway meets all necessary requirements. If not, address the gaps in access controls, encryption, and logging to meet these standards.

4. **Vulnerability to Attacks (SQL Injection, XSS, CSRF)**:

    - **Analysis**: The payment gateway may be vulnerable to common attacks such as SQL Injection (via improperly sanitized inputs) and Cross-Site Scripting (XSS) attacks.
    - **Recommendation**: Implement **input validation and sanitization** throughout the application, especially in payment fields. Use **prepared statements** for SQL queries to prevent SQL injection and ensure that input fields are validated before processing.

Additionally, implement **Content Security Policy (CSP)** to prevent XSS attacks and CSRF protections.

5. **Session Management**:

    - **Analysis**: A lack of proper session management could allow attackers to hijack user sessions and access sensitive information.
    - **Recommendation**: Implement **secure session management** practices, including using **secure cookies**, setting session timeouts, and enforcing **multi-factor authentication (MFA)** where feasible, especially for higher-value transactions.

6. **Authentication & Authorization**:

    - **Analysis**: Insufficient authentication measures could allow unauthorized users to make changes to payment data.
    - **Recommendation**: Ensure strong user authentication, including **MFA** for high-value transactions, and **role-based access controls** (RBAC) for system administrators and users.

---

2. Usability Analysis:

Key Focus Areas:

- **Ease of Use**
- **Error Message Clarity**
- **Responsive Design**
- **Transaction Feedback**

Current Usability Practices:

1. **Error Messages Clarity**:

    - **Analysis**: Currently, error messages might not provide enough information or actionable steps to resolve issues. For instance, users may encounter generic error messages like “Payment failed” without understanding why or how to fix the issue.
    - **Recommendation**: Error messages should be **clear, specific, and actionable**. For example, if a card is declined, the message should specify the reason, such as "Card declined due to insufficient funds" or "Expiration date invalid". The user should also be provided with steps on how to resolve the issue,

such as "Please check your card details or use another payment method."

2. **User Flow and Experience**:

    - **Analysis**: If users encounter a failure during payment (e.g., network issue, payment method failure), the process can feel frustrating without clear guidance on next steps.
    - **Recommendation**: Ensure that the **payment flow is intuitive and seamless**. For instance, if a payment fails, users should have the ability to **retry** easily or switch payment methods without needing to restart the entire process. Additionally, a **clear progress indicator** (e.g., "Processing Payment") should be displayed so users know the system is working.

3. **Mobile Responsiveness**:

    - **Analysis**: The checkout experience should be optimized for both desktop and mobile devices. If the payment gateway’s UI is not mobile-friendly, it may cause confusion or frustration, particularly in the case of payment fields.

  - **Recommendation**: Ensure the checkout page is **fully responsive** and provides a **mobile-optimized experience**. For example, input fields should be large enough for easy touch input, buttons should be spaced out for easier tapping, and users should be able to complete the payment seamlessly on mobile.

4. **Form Validation**:

   - **Analysis**: Users may make mistakes while entering payment details, such as typing the wrong credit card number, entering a wrong expiration date, or providing a mismatched CVV.
   - **Recommendation**: Implement **real-time form validation** to guide users as they enter their payment details. For example, provide immediate feedback when the card number is invalid or when the expiration date is in the past. This helps users catch errors before submitting the form and enhances the overall user experience.

5. **Payment Method Switching**:

   - **Analysis**: If one payment method fails, users might get stuck or confused if they are not presented with

clear options to switch to an alternative payment method (e.g., from card to PayPal).
   - **Recommendation**: Ensure that the payment gateway allows users to **easily switch between payment methods** if one fails. For example, if a card payment fails, the user should immediately see an option to select PayPal or another payment method without leaving the page.

6. **Transaction Feedback and Confirmation**:

   - **Analysis**: Once a payment is successful, users should receive immediate and clear feedback about the status of their transaction. Lack of confirmation may leave users unsure whether their payment was processed or not.
   - **Recommendation**: Provide **clear transaction feedback** immediately after payment completion. This could include a confirmation screen with the transaction number and an option to **email or print the receipt**. If the payment is unsuccessful, a **clear failure message** should be displayed with next steps.

Security and Usability Summary Recommendations:

Security Recommendations:

1. **Encrypt sensitive data** in transit using TLS and ensure stored data is tokenized and encrypted (AES-256).
2. **Conduct regular security audits** and penetration testing to ensure adherence to PCI DSS standards.
3. **Sanitize inputs** to prevent SQL injection and XSS attacks by using prepared statements and Content Security Policy (CSP).
4. **Strengthen session management** by implementing secure cookies, timeouts, and multi-factor authentication (MFA).
5. **Implement proper access control** and role-based authorization to ensure only authorized personnel can access sensitive information.

Usability Recommendations:

1. **Improve error messages** by making them specific, clear, and actionable to guide users through resolution.
2. **Optimize the checkout flow** to ensure users can easily retry or switch payment methods when a failure occurs.

3. **Enhance mobile responsiveness** so that the payment process is seamless across all devices.
4. **Add real-time form validation** to alert users to errors (e.g., incorrect card number, expired card) before submission.
5. **Provide clear feedback** after a transaction to confirm success or failure and offer the user a clear path to resolve any issues.