### **Test Plan**

| **Module** | **Responsible Person** | **Testing Type** |
| --- | --- | --- |
| UI Testing (forms, responsiveness) | Zarah Hassan | Functional Testing & Test Automation |
| Backend & Database Testing | Tayaba | Structural Testing |
| Integration Testing (payment + booking flow) | Sanjna | Functional & Security Testing |
| Overall System Testing | Nabiha | Regression Testing & Blackbox Testing |

#### **Introduction**

### This test plan defines the testing strategy and deliverables for the registration and login functionality in the "Adventour" website. This page allows users to register, log in, and interact with a form to manage their accounts.

### **Registration**

##### **Scope:**

* **UI Testing**: Verify the form's layout and the visibility of fields such as full name, email, password, contact number, and address.
* **Backend Testing**: Ensure the data is correctly processed and stored in the database. Test user registration logic, including email validation, password hashing, and preventing duplicate emails.
* **Integrations Testing**: Validate the form submission with the backend, ensuring proper error handling and data storage.
* **Security Testing**: Test input validation to prevent SQL injection and ensure password security.

##### **Objectives:**

* Ensure that all fields on the form are correctly validated.
* Verify that the user can successfully register and that data is saved in the database.
* Ensure the form correctly displays error messages for invalid inputs.
* Confirm that password visibility toggle works as expected.

**Test Strategy**

##### **Test Types & Levels:**

* **Unit Testing**: Test individual components like form validation, backend logic for user registration, and password hashing.
* **Integration Testing**: Verify the interaction between the front-end form and the back-end database, ensuring data is properly processed.
* **System Testing**: Test the entire registration flow, including user inputs, database interaction, and redirection after successful registration.
* **Acceptance Testing**: Verify that the registration process meets the business requirements.

#### **Test Deliverables**

* **Test Cases**: A set of test cases for validating user registration and login form functionality.
* **Test Reports**: Detailed results of test execution, including passed/failed tests.
* **Defect Reports**: A record of defects found during testing, including severity, steps to reproduce, and fixes.

#### **Risk & Mitigation**

##### **Potential Risks:**

* **Database Connection Issues**: If there is a problem with the database connection, the form may not be able to register users.
* **Duplicate Registration**: If the form doesn't correctly handle duplicate email registrations, users may be able to register multiple times with the same email.
* **Security Vulnerabilities**: SQL injection attacks, especially if the form inputs are not properly sanitized.
* **Incorrect Field Validation**: Users may enter incorrect data, and the form may not handle these scenarios gracefully.

##### **Mitigation Strategies:**

* **Error Handling**: Ensure all database errors are handled correctly with appropriate error messages.
* **Prepared Statements**: Use prepared statements for database queries to prevent SQL injection.
* **Data Validation**: Implement proper client-side and server-side validation for all fields.
* **Thorough Testing**: Test the registration process with both valid and invalid data to ensure the system behaves as expected.

#### **Design Test Scenarios**

1. **Test Case 1**: Test registration with valid data.
   * **Description**: Ensure the user can register with a valid name, email, password, phone, and address.
   * **Expected Result**: User is successfully registered and redirected.
2. **Test Case 2**: Test registration with missing required fields.
   * **Description**: Attempt to register with missing data (e.g., name or email).
   * **Expected Result**: The system should prompt the user for missing fields.
3. **Test Case 3**: Test registration with an already registered email.
   * **Description**: Attempt to register with an email that's already in the database.
   * **Expected Result**: The system should show an error message indicating the email is already registered.
4. **Test Case 4**: Test registration with invalid email format.
   * **Description**: Attempt to register with an invalid email format.  
     **Expected Result**: The system should display a validation error for the email.
5. **Test Case 5**: Test password visibility toggle.
   * **Description**: Verify that clicking the password visibility toggle shows/hides the password correctly.
   * **Expected Result**: The password visibility should toggle between text and password.
6. **Test Case 6**: Test login form functionality.
   * **Description**: Ensure the login button switches the form to the login view.
   * **Expected Result**: The form should display only email and password fields when switching to login.

#### **Prepare Test Data**

* **Valid Data**:
  + Name: Nabeeha
  + Email: nabeeha@example.com
  + Password: Password123!
  + Contact Number: 1234567890
  + Address: 123 Main St, City, Country
* **Invalid Data**:
  + Invalid Email: Nabeeha@domain
  + Invalid Phone: 12345
  + Missing Fields: Leave one or more fields empty

**2. Home**

#### **Scope**:

* + Testing the **UI**, **backend**, and **integrations** of the travel website.
  + Ensuring that pages like **home**, **about**, **book**, **package**, and **tourguide** are functioning correctly.
  + Verifying all forms, such as the **search form** and **registration form**, are properly working.
  + Testing navigation links to ensure they direct to the correct pages.
* **Objectives**:
  + Verify that each component is working as expected (UI elements, buttons, forms).
  + Ensure all back-end processes (e.g., data handling, booking functionality) are seamless.
  + Validate integrations (e.g., database for user registration and bookings).

#### **Test Strategy:**

* **Test Types**:
  + **Unit Testing**: Testing individual components like the registration form, search function, and navigation.
  + **Integration Testing**: Verifying interactions between the front-end and back-end (e.g., submitting forms, fetching data from the database).
  + **System Testing**: Full end-to-end testing of all modules together, including UI, functionality, and integration.
  + **Acceptance Testing**: Ensure that the website meets user expectations and performs correctly.
* **Test Levels**:
  + **Unit Testing**: Testing individual functions like form validations, search functionality, etc.
  + **Integration Testing**: Testing the registration and login system, along with backend database integration.
  + **System Testing**: Ensure that all modules (like home, packages, about) integrate correctly and there are no broken links.
  + **Acceptance Testing**: Verify that the website is user-friendly and meets the requirements.

#### **Test Deliverables:**

* **Test Cases**: Documented test cases based on functionality (e.g., user registration, search form).
* **Test Reports**: Summary of the testing progress, issues, and results.
* **Defect Reports**: Detailing defects found during testing (e.g., broken links, form validation errors).
* **Test Data**: Sample data for testing various form submissions (e.g., valid/invalid email, password combinations).

#### **Risk and Mitigation:**

* **Risks**:
  + **Missing Requirements**: Unclear or missing specifications for some website features.
  + **Incomplete Data**: Insufficient data for testing backend functionalities like registration and search.
  + **Integration Issues**: Potential issues when connecting front-end with back-end (e.g., form submission failures).
* **Mitigation Strategies**:
  + **Clear Requirements**: Ensure that all features are clearly documented and any uncertainties are clarified.
  + **Test Data Preparation**: Create comprehensive test data before testing, covering all possible scenarios (e.g., valid, invalid inputs).
  + **Frequent Communication**: Keep regular communication between front-end and back-end developers to address integration concerns early.

### **Test Scenarios**

1. **Home Page**:
   * Test the navigation links to ensure they direct users to the correct pages.
   * Verify the responsiveness of the homepage across different devices.
   * Ensure the image slider works correctly.
2. **Registration Form**:
   * Verify that all fields (full name, email, password, contact, address) are correctly validated.
   * Ensure that the password visibility toggle works.
   * Check if duplicate emails are handled (show proper error message).
3. **Search Form**:
   * Ensure that the search functionality works with various inputs.
   * Verify that no broken links appear when submitting search data.
4. **Booking Page**:
   * Test form submission with sample booking data.
   * Verify backend booking logic (whether the correct data is being inserted into the database).
5. **Package Page**:
   * Verify that the "Book Now" button leads to the correct booking form.
   * Test the functionality of "Load More" for displaying additional packages.

### **Test Data**

* **Valid Registration Data**:
  + Full Name: Tayaba
  + Email: tayaba@example.com
  + Password: StrongPassword123
  + Contact: 1234567890
  + Address: 123 Street, City, Country
* **Invalid Registration Data**:  
  + Invalid Email: Tayaba.com
  + Invalid Password: 12345
* **Sample Search Queries**:  
  + "adventure"
  + "tour packages"

**3. Booking**

* **Scope**:
  + **UI Testing**: Ensure that the booking form appears as expected on various devices and browsers, maintaining a responsive layout.
  + **Backend Testing**: Verify that the booking data is correctly processed by the backend (e.g., through the book\_form.php script).
  + **Integration Testing**: Ensure smooth interaction between the front-end form and the backend processing, especially form submission and database integration.
* **Objectives**:
  + Test the form's functionality (input fields, validation, submission).
  + Test form handling in PHP (e.g., capturing form data).
  + Test backend integration with the database for saving bookings.
  + Verify that the website's appearance is consistent across different devices (desktop, tablet, mobile).

#### **Test Strategy**

* **Unit Testing**:
  + Validate each individual component such as form fields and buttons.
  + Ensure that form elements are working (e.g., the "Submit" button is clickable, fields accept correct input types).
* **Integration Testing**:
  + Ensure that data entered into the form is correctly passed to book\_form.php and saved to the backend database.
  + Check that any error messages or success confirmation are correctly displayed after form submission.
* **System Testing**:
  + Ensure that the entire process from booking form input to successful submission works seamlessly. This includes checking for data validation and proper user interface behavior.
* **Acceptance Testing**:
  + Verify that the form meets the business requirements for booking trips and is ready for production use.
  + Test that the website meets accessibility and user experience standards.

#### **Test Deliverables**

* **Test Cases**:
  + Detailed test cases for each part of the form and its functionality.
  + Validation checks for each field in the form (e.g., name, email, phone number).
* **Test Reports**:  
  + Summary reports of each test case's execution, with clear pass/fail status and detailed descriptions of any issues found.
* **Defect Reports**:  
  + A log of any issues or defects found during testing, including steps to reproduce, severity, and recommended fixes.

#### **Risk & Mitigation**

* **Potential Risks**:
  + **Missing Requirements**: If the form doesn't capture all necessary information (e.g., missing required fields or incorrect field types).
  + **Form Validation Errors**: If the form doesn’t properly validate user input (e.g., missing fields or invalid email).
  + **Database Integration Issues**: If data is not correctly saved to the database or any errors occur when processing bookings.
* **Mitigation Strategies**:
  + **Missing Requirements**: Review the form fields with the business team to ensure all necessary data is being captured.
  + **Form Validation Errors**: Implement robust client-side (JavaScript) and server-side (PHP) validation. Use error messages to guide the user.
  + **Database Integration Issues**: Test with different datasets and ensure proper error handling is in place (e.g., preventing SQL injection).

#### **Test Scenarios**

* **Form Field Validation**:
  + Check for required fields (e.g., name, email, phone).
  + Ensure proper validation for the email and phone number fields (correct format).
  + Test the "guests" field for numeric input only.
* **Form Submission**:
  + Test submitting the form with valid data and verify that the submission is successful.
  + Check for errors when submitting incomplete or incorrect data.
* **Responsive Design**:  
  + Test the booking page on various devices (mobile, tablet, desktop) to ensure that the form is responsive and user-friendly.
* **Database Integration**:  
  + Verify that the booking data is properly saved to the database once the form is submitted.
  + Check that all fields (e.g., name, email, address) are correctly stored in the database.

#### 

#### **Prepare Test Data**

* **Valid Data**:
  + Name:Zarah
  + Email: Zarah@example.com
  + Phone: 1234567890
  + Address: 123 Main St
  + Location: Lahore
  + Guests: 3
  + Arrival Date: 2025-06-15
  + Leaving Date: 2025-06-22
* **Invalid Data**:
  + Name: (empty)
  + Email: invalid-email
  + Phone: 12345 (short length)
  + Guests: -5 (negative number)

**4. BOOKING\_FORM**

**Scope & Objectives**:

* **What will be tested**:
  + UI: Form fields on the booking page.
  + Backend: PHP code handling form submission, database interaction, and redirect logic.
  + Database: Ensure correct insertion of booking data into the book\_form table.
  + Functional: Proper calculation of the total amount based on the number of guests and redirection to the payment page.

**Test Strategy**:

* **Test Types**:
  + **Unit Testing**: Test individual functions, especially for form processing and database operations.
  + **Integration Testing**: Ensure the form correctly integrates with the database and passes the expected data to the next page (pay.php).
  + **System Testing**: Test the entire booking system from form submission to payment redirection.
  + **Acceptance Testing**: Confirm that the system meets the business requirements (e.g., successful booking process, correct total price calculation).
* **Test Levels**:
  + **Unit Tests**: Test PHP functions for handling input validation, database insertion, etc.
  + **Integration Tests**: Test data flow between the form, PHP backend, and database.
  + **System Tests**: Ensure that all components (UI, backend, database) work together correctly.
  + **Acceptance Tests**: Validate the complete user flow, from form submission to payment page redirection.

**Test Deliverables**:

* Test cases (functional and non-functional).
* Test execution reports (including pass/fail results).
* Defect reports for any issues encountered during testing.
* Test summary report.

**Risk & Mitigation**:

* **Risk 1**: Incorrect data insertion into the database (e.g., SQL injection).
  + **Mitigation**: Use parameterized queries to prevent SQL injection.
* **Risk 2**: Missing or invalid data in form fields.
  + **Mitigation**: Validate form input on the server side to ensure required fields are not empty and data is in the correct format.
* **Risk 3**: Incorrect total amount calculation.  
  + **Mitigation**: Ensure the totalAmount calculation is tested with a variety of inputs.

**Test Scenarios**:

1. **Scenario 1**: User submits a booking form with valid input.
   * **Expected Outcome**: Data is inserted into the database, and the user is redirected to the payment page with the correct total amount.
2. **Scenario 2**: User submits a booking form with missing required fields (e.g., email, phone).
   * **Expected Outcome**: Form submission fails, and an error message is displayed.
3. **Scenario 3**: User submits a booking form with invalid data (e.g., non-numeric guests).
   * **Expected Outcome**: Form submission fails, and a validation error is displayed.
4. **Scenario 4**: User submits a booking form with a large number of guests.
   * **Expected Outcome**: Total amount calculation is accurate and handles large inputs without errors.
5. **Scenario 5**: SQL injection attempt in form fields.
   * **Expected Outcome**: The system prevents SQL injection and does not execute malicious SQL code.

#### **Prepare Test Data**

* **Test Data Examples**:  
  + Valid inputs:
    - Name: "sanjana"
    - Email: "sanjana@example.com"
    - Phone: "1234567890"
    - Address: "123 Main St"
    - Location: "Karachi"
    - Guests: 3
    - Arrival Date (date): "2025-05-01"
    - Departure Date (date): "2025-05-07"
  + Invalid inputs:  
    - Name: "sanjana"  
      Email: "sanjana@example"
    - Guests: "abc" (non-numeric value)

**5. Package**

* **Scope**:
  + **UI Testing**: Ensure that the layout and design of the "Packages" page render correctly across various devices and browsers.
  + **Backend Testing**: Verify that data from the database (travel packages) is correctly fetched and displayed on the page.
  + **Integration Testing**: Test the interaction between the front-end and the back-end, ensuring that the data flow from the database to the UI works without issues.
  + **Search Functionality**: Verify the behavior of the search bar, ensuring that it correctly filters packages based on user input.
* **Objectives**:
  + Ensure that all travel packages are displayed correctly with their respective images, names, descriptions, and prices.
  + Test the functionality of the "Book Now" and "Details" buttons, ensuring correct data handling (e.g., passing package price and package ID).
  + Verify that the database query correctly fetches all the necessary package information.
  + Test the search functionality to filter packages based on user input.

#### **Test Strategy**

* **Unit Testing**:
  + Validate individual components like search input, database connection, and form actions.
  + Ensure each package's data (name, description, price) is fetched and displayed correctly.
* **Integration Testing**:
  + Test how data from the database is passed and displayed on the website.
  + Ensure that clicking the "Book Now" button passes the correct price to the book.php page and redirects properly.
  + Ensure the "Details" button passes the correct package ID to the details.php page.
* **System Testing**:
  + Test the page as a whole to ensure that the entire process from fetching package data, displaying it, and user interaction (search, booking, details) works smoothly.
  + Test the layout of the page on different screen sizes to ensure it's responsive.
* **Acceptance Testing**:
  + Verify that the functionality matches the project requirements (correct display of package information, functionality of buttons, search capability).
  + Confirm that the user experience aligns with expectations (easy to use, intuitive).

#### **Test Deliverables**

* **Test Cases**:  
  + Detailed test scripts for each feature, including the search functionality, package display, and button actions.
* **Test Reports**:  
  + A report detailing the results of each test, including any failed tests, errors found, and descriptions of how to reproduce them.
* **Defect Reports**:  
  + A log of any issues identified during testing, including steps to reproduce, severity, and any workarounds or fixes.

#### **Risk & Mitigation**

* **Potential Risks**:
  + **Data Fetching Errors**: Database issues could prevent package data from being fetched.
  + **Incorrect Data Display**: Package details might not be rendered correctly (e.g., missing images, wrong descriptions, etc.).
  + **Search Malfunction**: The search feature might fail to filter packages correctly or might not work at all.
* **Mitigation Strategies**:  
  + **Data Fetching Errors**: Check for database connection issues and verify the query logic to ensure data is being pulled correctly.
  + **Incorrect Data Display**: Double-check the PHP logic that handles package data and ensure that all HTML tags are closed properly and data is being injected into the page correctly.
  + **Search Malfunction**: Test the search functionality with various inputs to ensure that it returns results and handles edge cases (e.g., no results, empty input).

#### **Test Scenarios**

* **Package Display**:  
  + Verify that the correct package data is shown for each package (name, description, price, and image).
* **Search Functionality**:  
  + Test the search input to ensure that when a user types a destination or keyword, the correct packages are filtered and displayed.
* **Booking Button**:  
  + Verify that the "Book Now" button correctly passes the package price to book.php through the GET method.
* **Details Button**:  
  + Ensure the "Details" button correctly passes the package ID to details.php through the GET method.
* **Responsive Layout**:  
  + Test the page on various screen sizes to ensure proper layout and responsiveness.

#### **Prepare Test Data**

* **Valid Data**:
  + Package Name: "Paris Adventure"
  + Description: "A luxurious tour to the beautiful city of Paris."
  + Price: $500
  + Image: Base64-encoded image data.
* **Invalid Data**:
  + Search: "xyz" (no matching results)
  + Package Name: (empty or missing)
  + Price: Negative value (e.g., -$100).

**7. Payment**

* **Scope**:
  + **UI Testing**: Validate the user interface of the payment page, including the form fields, buttons, error messages, and layout.
  + **Backend Testing**: Ensure the correct processing of the total payment amount and the successful redirection upon payment.
  + **Integration Testing**: Ensure the form's data validation works as expected and integrates with the backend logic (e.g., payment processing).
  + **Security Testing**: Test input validation to ensure that there are no security vulnerabilities such as injection attacks in the form fields.
* **Objectives**:
  + Verify the accurate display of the payment amount.
  + Test the form for required fields and proper validation (card number, expiration date, CVV).
  + Ensure the expiration date is checked correctly and prevents expired cards from being processed.
  + Test the successful redirection to ordersuccess.php upon payment submission.

#### **Test Strategy**

* **Unit Testing**:  
  + Test individual form fields (card number, expiration date, CVV) for correctness and validation.
  + Verify that the correct amount is displayed on the page.
  + Check if form submission redirects to ordersuccess.php after payment is processed.
* **Integration Testing**:
  + Test the interaction between the front-end (form submission) and back-end (payment processing logic).
  + Verify that the correct total amount is passed via the GET parameter and used in the form.
  + Test form validation logic to ensure invalid inputs trigger appropriate error messages.
* **System Testing**:
  + Test the overall workflow from loading the payment page to submitting the form and redirecting to ordersuccess.php.
  + Ensure that the form is responsive and functions across different devices and browsers.
* **Acceptance Testing**:  
  + Verify that the functionality matches user requirements (correct display of amount, correct form validation, successful redirection).
  + Ensure the page meets security and usability standards.

**Test Deliverables**

* **Test Cases**:  
  + Detailed test cases for each feature (form validation, error messages, redirection).
* **Test Reports**:  
  + A report detailing the results of each test, including the pass/fail status and any defects found.
* **Defect Reports**:  
  + Log of defects, including severity, steps to reproduce, and workarounds/fixes.

#### **Risk & Mitigation**

* **Potential Risks**:
  + **Form Validation Failure**: Incorrect or missing validation for card details, leading to incorrect payments or failed submissions.
  + **Security Issues**: Risk of injection attacks or improper handling of sensitive card data.
  + **Payment Processing**: Issues with redirection or data not being correctly passed (e.g., total amount not properly displayed).
* **Mitigation Strategies**:
  + **Form Validation Failure**: Ensure proper client-side validation of card details (expiration date, CVV) before submission.
  + **Security Issues**: Implement input sanitization to avoid injection attacks and ensure sensitive data like card details is not stored or logged improperly.
  + **Payment Processing**: Double-check that the payment process redirects to the success page and the correct total is passed.

#### **Test Scenarios**

* **Test Scenario 1**: **Card Number Field Validation**
  + **High-level Description**: Test that the card number input accepts only valid credit card numbers.
  + **Expected Outcome**: The form should not accept invalid card numbers (e.g., incorrect format or length).
* **Test Scenario 2**: **Expiration Date Validation**
  + **High-level Description**: Test that the expiration date input checks whether the card has expired.
  + **Expected Outcome**: If the expiration date is in the past, display an error message and prevent submission.
* **Test Scenario 3**: **CVV Field Validation**
  + **High-level Description**: Test that the CVV field accepts only valid 3-digit numbers.
  + **Expected Outcome**: The form should reject any CVV input that is not a valid 3-digit number.
* **Test Scenario 4**: **Form Submission and Redirection**
  + **High-level Description**: Test that the form redirects to ordersuccess.php upon successful payment submission.
  + **Expected Outcome**: After clicking the "Pay Now" button, the page should redirect to ordersuccess.php.
* **Test Scenario 5**: **Amount Display**
  + **High-level Description**: Test that the correct total amount is displayed on the page.
  + **Expected Outcome**: The total amount shown should match the amount passed via the GET parameter.
* **Test Scenario 6**: **Responsive Layout**
  + **High-level Description**: Test the payment page on different screen sizes and devices to ensure it is responsive.
  + **Expected Outcome**: The page should be properly displayed on mobile, tablet, and desktop.

#### **Prepare Test Data**

* **Valid Data**:
  + Total Amount: $100
  + Card Number: 4111111111111111 (valid test card)
  + Expiration Date: 12/25
  + CVV: 123
* **Invalid Data**:
  + Total Amount: $100
  + Card Number: 12345678901234 (invalid card number)
  + Expiration Date: 01/20 (expired card)
  + CVV: abc (non-numeric CVV)

**7. Tour Guide**

* **Scope:**
  + Test the functionality and user experience of the "Tour Guides" page, which displays information about available tour guides, including their name, email, phone, about, and image.
  + Ensure the correct display of guide details fetched from the database.
  + Test the search functionality (if implemented in search\_dest.php) for accurate results.
* **Objectives:**
  + Validate data retrieval from the database.
  + Ensure proper display of tour guides' information (name, email, phone, image).
  + Ensure that the page layout is responsive and correctly displays on different devices.
  + Test the search feature for correct behavior when searching for guides by name or other criteria.
  + Verify security aspects such as SQL injection vulnerabilities.

#### **Test Strategy:**

* **Manual Testing:**
  + The focus will be on manual testing of UI components such as the search functionality, card display of guides, and overall user interaction.
* **Automation Testing:**
  + Where feasible, we will automate certain UI aspects using tools like Selenium for repeated testing of the page, search functionality, and form validations.

#### **Testing Levels:**

* **Unit Testing:**
  + Test individual components like the database connection and query execution.
* **Integration Testing:**
  + Ensure that the frontend (HTML and CSS) integrates properly with the backend (PHP and MySQL) and displays the correct data.
* **System Testing:**
  + Test the entire system, ensuring that data flows from the database to the webpage correctly, and the page functions as expected.
* **Acceptance Testing:**
  + Verify that the application meets user requirements, focusing on the search functionality and overall usability.

#### **Testing Types:**

* **Functional Testing:**
  + Ensure that the guide details (name, email, phone, image) are fetched and displayed correctly.
  + Validate the search functionality by entering different inputs (e.g., name or partial name of guides).
* **Structural Testing:**
  + Test how the database queries work and whether they are secure (e.g., ensure no SQL injection vulnerabilities).
* **Automation Testing:**
  + Automate UI validation for guide display and search functionality.

#### **Test Deliverables:**

* **Test Cases:**
  + A detailed list of all test cases for guide display and search functionality.
* **Test Reports:**
  + Reports that document whether tests pass or fail.
* **Defect Logs:**
  + Logs that track any defects found during testing, including their severity and status.

#### **Risks & Mitigation:**

* **Risk:** Lack of sample guide data in the database.
  + **Mitigation:** Ensure a set of predefined tour guide records in the database before starting testing.
* **Risk:** Database connection issues.
  + **Mitigation:** Ensure that the server and database are properly configured and accessible for testing.
* **Risk:** Inadequate test data for search.
  + **Mitigation:** Prepare test cases with diverse search queries and guide data for thorough validation.

#### **Test Scenarios:**

* **Scenario 1: Database Connection**
  + **Objective:** Verify that the system can successfully connect to the database.
  + **Expected Behavior:** The system should connect to the database without errors.
* **Scenario 2: Guide Display**
  + **Objective:** Ensure that the guide's details (name, email, phone, about, image) are displayed correctly on the webpage.
  + **Expected Behavior:** Each guide's name, email, phone, about, and image should be fetched and displayed correctly.
* **Scenario 3: Search Functionality**
  + **Objective:** Verify the search functionality works as intended, returning accurate results.
  + **Expected Behavior:** The search bar should display guides that match the search term (name or part of the name).

#### **Test Data:**

* **Sample Guide Data:**
  + Name: John Doe, Email: johndoe@gmail.com, Phone: +1234567890, About: "Experienced guide", Image: [image of a guide]
  + Name: Jane Smith, Email: janesmith@example.com, Phone: +9876543210, About: "Local guide with deep knowledge of the city", Image: [image of a guide]

#### **Test Execution Reports:**

* **Report 1: Database Connection Test**
  + **Pass/Fail:** Pass
  + **Issues:** None
  + **Notes:** Connection successful
* **Report 2: Guide Information Display**
  + **Pass/Fail:** Pass
  + **Issues:** None
  + **Notes:** All guides' information is displayed correctly.
* **Report 3: Search Functionality**
  + **Pass/Fail:** Fail
  + **Issues:** Search did not return expected results.
  + **Notes:** Need to check for issues in the search query.
* **Report 4: SQL Injection Test**
  + **Pass/Fail:** Pass
  + **Issues:** None
  + **Notes:** No SQL injection vulnerabilities found.
* **Report 5: Responsive Design Test**
  + **Pass/Fail:** Pass
  + **Issues:** None
  + **Notes:** Page displayed correctly across all devices.

**8. Search Destination  
Scope**:  
 Test the search functionality of search\_dest.php, including:

* + Search query input
  + Backend query handling (MySQL)
  + Proper rendering of results (text and image)
  + Functionality of "Book Now" and "Details" buttons
* **Objectives**:
  + Validate accurate search results based on user input.
  + Ensure UI elements display consistently and are responsive.
  + Test for broken links and invalid input handling.
  + Ensure data is securely retrieved and displayed.

### **Test Strategy**

* **Approach**:  
   Manual Testing across:
  + Desktop and mobile browsers
  + Multiple input cases (valid, invalid, SQL injection, empty)
  + Cross-browser testing: Chrome, Firefox, Edge
* **Why Manual**:  
   Ideal for short-term projects with dynamic content and lower complexity.

### **Testing Levels**

| **Level** | **Description** |
| --- | --- |
| Unit Testing | Test searchPackages() PHP function logic. |
| Integration Testing | Validate database and UI integration. |
| System Testing | Full end-to-end functionality (input ➝ output) |
| Acceptance Testing | Verify that the page meets user expectations. |

### **Testing Types**

| **Type** | **Description** |
| --- | --- |
| **Functional Testing** | Input handling, button actions, database results. |
| **Structural Testing** | HTML rendering, broken links, correct structure. |
| **UI/UX Testing** | Layout consistency, responsive behavior. |
| **Negative Testing** | Empty fields, special characters, SQL edge cases. |
| **Security Testing** | SQL injection attempt resistance. |

#### **Test Deliverables:**

* **Test Cases:**
  + A detailed list of all test cases for guide display and search functionality.
* **Test Reports:**
  + Reports that document whether tests pass or fail.
* **Defect Logs:**

Logs that track any defects found during testing, including their severity and status.

### **Risks & Mitigation**

| **Risk** | **Impact** | **Mitigation** |
| --- | --- | --- |
| No result on valid search | High | Ensure proper SQL LIKE query and collation |
| Broken image rendering | Medium | Validate that images are correctly base64 encoded |
| SQL injection attempt | High | Use mysqli\_real\_escape\_string() and parameterized queries |
| Styling mismatch on different screens | Low | Responsive testing on mobile/tablet/desktop |
| Case-sensitive search failure | Medium | Use LOWER() in SQL query and convert input |

### **Test Scenarios**

| **ID** | **Scenario** | **Expected Result** |
| --- | --- | --- |
| TS01 | Search with valid destination name | Matching records are shown |
| TS02 | Search with special characters (e.g., @!#$) | "No results found" message |
| TS03 | Submit empty search | Message shown or all packages displayed |
| TS04 | Click "Book Now" from a result | Redirects to book.php |
| TS05 | Click "Details" button | Redirects to details.php |
| TS06 | Submit SQL injection string | No query executed, page remains safe |

### **Test Data**

| **Input** | **Description** |
| --- | --- |
| Hunza | Existing destination |
| Skardu | Another known destination |
| Islamabad | Non-listed city (should show nothing) |
| ; DROP TABLE packages;-- | SQL injection string |
| ###$$@! | Special characters |
| (space) | Blank input |

### **Test Execution Report**

| **TC ID** | **Result** | **Notes** |
| --- | --- | --- |
| TC01 | ✅ Pass | Valid destination found |
| TC02 | ✅ Pass | Empty search handled |
| TC03 | ✅ Pass | SQL injection escaped |
| TC04 | ✅ Pass | Special characters don’t break functionality |
| TC05 | ✅ Pass | Base64 image shown correctly |
| TC06 | ✅ Pass | Navigation works |
| TC07 | ✅ Pass | Navigation works |
| TC08 | ✅ Pass | Case-insensitive search works |

**9. Order Success**

**Scope and Objectives**:

* + Define the areas that will be tested, such as specific modules or features (e.g., the booking system, user authentication, or payment gateway in your travel website).
  + The objectives should focus on verifying the functionality, performance, and security of the system as a whole.

**Test Strategy**:

* + **Manual Testing**: To verify functionality and usability, including UI tests, regression tests, and exploratory tests.
  + **Automation**: Use tools like Selenium for regression and UI testing, especially for frequently changing components or complex workflows.
  + **Security Testing**: If applicable, ensure data protection (e.g., through penetration testing or testing for SQL injection).

**Testing Levels**:

* + **Unit Testing**: Individual components or functions will be tested. For example, testing PHP scripts like the booking form or login functionality.
  + **Integration Testing**: Tests the interaction between different modules. For instance, checking the interaction between the front-end UI and the back-end database for booking details.
  + **System Testing**: Tests the entire system, including workflows like booking a tour or making a payment.
  + **Acceptance Testing**: Ensures that the system meets business requirements. It can be done by the client or the testing team to validate the project.

**Testing Types**:

* + **Functional Testing**: Focuses on testing specific functions and features like user login, booking, search functionalities, etc.
  + **Structural Testing**: Involves testing the internal structure of the application (e.g., code coverage, testing the code’s logic, data flow, and control flow).
  + **Test Automation**: Automating repetitive test cases such as UI checks, form submissions, and database verifications using Selenium, JUnit, etc.

**Test Deliverables**:

* + A **Test Plan Document** detailing the strategy, scope, and objectives.
  + **Test Scenarios and Test Cases** that will be executed during the testing phase.
  + **Test Execution Reports** showing the results and status (pass/fail) of each test.

**Risks and Mitigation Strategies**:

* + **Risk**: Limited time for thorough testing due to project deadlines.
    - **Mitigation**: Prioritize testing critical features first, and automate high-priority tests.
  + **Risk**: Lack of resources for testing.
    - **Mitigation**: Use available team members efficiently and consider crowd-sourced testing if feasible.
  + **Risk**: Changes in requirements or scope mid-project.
    - **Mitigation**: Implement agile testing practices to handle iterative development and changing requirements.

**Test Scenarios**:

* + High-level descriptions of what needs to be tested, such as:
    - "Verify the user is able to complete the booking process."
    - "Check that the system handles invalid payment details gracefully."

**Test Data**:

* + Sample data like user profiles, booking details, and payment information will be needed to execute tests.
  + Example: For the booking system, you might need test data for a valid user, a valid package, and valid payment information.

**Test Execution Reports**:

* + After running the tests, you'll need to document the results.
  + The report should include:
    - A summary of all executed test cases.
    - The results (pass/fail).
    - Any defects or issues found, including descriptions, severity, and status (open/closed).
    - Recommendations or follow-up actions if necessary.

**10.About**

**Scope and Objectives:**

* + **Scope:** Define which parts of the system will be tested. For example, the scope could include the booking system, payment gateway, user authentication, and the review section of the website.
  + **Objectives:** Outline the specific goals of testing, such as:
    - Verify that the booking process works end-to-end.
    - Ensure that user input is validated correctly.
    - Confirm the system handles invalid payment details gracefully.

**Test Strategy:**

* + **Manual Testing:** Conduct manual testing for the user interface (UI) and user experience (UX) on different devices to ensure everything looks good and functions properly.
  + **Automated Testing:** Use tools like **Selenium** or **JUnit** to automate repetitive tasks, especially for regression testing. Automation is ideal for tests such as form submissions, booking workflows, and verifying expected behavior in multiple browsers.
  + **Security Testing:** Perform penetration testing to check for vulnerabilities, especially in areas like user authentication, payment gateways, and database queries.

**Testing Levels:**

* + **Unit Testing:** Testing individual components or functions of the code. For instance, validating PHP functions that calculate prices or check availability.
  + **Integration Testing:** Ensure that the booking system integrates correctly with other parts of the application, such as user login and payment processing.
  + **System Testing:** Perform end-to-end tests, including booking a tour package, verifying that payment is processed correctly, and checking email confirmation.
  + **Acceptance Testing:** Conduct acceptance testing with the client to verify that the system meets their requirements and business objectives.

**Testing Types:**

* + **Functional Testing:** Ensure that all features are working as expected, like booking, payment processing, and showing available packages.
  + **Structural Testing:** Test the internal structure of the application, such as checking code quality and testing the system's response to edge cases and unusual inputs.
  + **Test Automation:** Automate repetitive functional tests to save time, such as UI validation using Selenium or API testing.

**Test Deliverables:**

* + **Test Plan Document:** A formal document that outlines the scope, objectives, strategy, and methodologies for testing.
  + **Test Scenarios:** High-level descriptions of the functionality that needs to be tested.
  + **Test Cases:** Detailed scripts for each scenario that describe the steps, expected outcomes, and actual results.
  + **Test Execution Reports:** A summary of test execution results, including the pass/fail status of each test case and defect tracking.

**Risks and Mitigation Strategies:**

* + **Risk 1:** Insufficient time for thorough testing due to project deadlines.
    - **Mitigation:** Prioritize testing critical features first and consider early automation of key scenarios.
  + **Risk 2:** Lack of access to real-world test data (e.g., booking records).
    - **Mitigation:** Use mock data or create test datasets that simulate real-world conditions.
  + **Risk 3:** Frequent changes to the requirements or features during testing.
    - **Mitigation:** Apply agile testing practices to handle ongoing changes and continuously update the test plan.

**Test Scenarios:**

* + These are high-level descriptions of what you intend to test. Example scenarios could include:
    - **Login Flow:** Test if a user can log in successfully with valid credentials.
    - **Booking Flow:** Test if a user can select a package, add it to the cart, and proceed to payment.
    - **Review System:** Verify if users can submit and view reviews for tour packages.

**Test Data:**

* + This includes all the sample data you will use for testing. For instance:
    - **Test Data 1:** Valid user credentials (username and password).
    - **Test Data 2:** Sample tour package details with package name, price, and availability.
    - **Test Data 3:** Payment information for testing successful and failed transactions.

**Test Execution Reports:**

* + Once tests are executed, the results should be documented. This includes:
    - A list of all executed test cases.
    - The status (pass/fail) for each test case.
    - Any defects found (e.g., bugs, UI issues) and their severity.
    - Recommendations for further testing or bug fixes.

**11. Details**

**Scope and Objectives**:

* + **Scope**: Specifies the features and functionalities to be tested, including modules, user stories, and requirements.
  + **Objectives**: Defines what the testing process aims to achieve, such as identifying bugs, validating business logic, or ensuring compatibility.

**Test Strategy**:

* + Describes the testing approach, including the overall testing methodology (e.g., Agile, Waterfall).
  + Specifies the tools, techniques, and processes to be used for testing.

**Testing Levels**:

* + **Unit Testing**: Focuses on testing individual components or functions of the system.
  + **Integration Testing**: Ensures that different system components work together as expected.
  + **System Testing**: Involves testing the complete system to ensure it meets all requirements.
  + **Acceptance Testing**: Validates the system against business requirements and ensures it meets user expectations.

**Testing Types**:

* + **Functional Testing**: Verifies that the system performs as intended, including the validation of inputs, outputs, and expected behaviors.
  + **Structural Testing**: Ensures that the internal structure of the system (e.g., code paths, logic) works as expected.
  + **Test Automation**: Involves automating repetitive tests to save time, especially for regression and load testing.

**Test Deliverables**:

* + **Test Plan Document**: The main document defining the test strategy and schedule.
  + **Test Scenarios**: High-level descriptions of the test cases to be executed.
  + **Test Cases**: Detailed steps and expected results for individual test scenarios.
  + **Test Execution Reports**: Reports that detail the execution status of each test (pass/fail), defects found, and any blockers.

**Risks and Mitigation Strategies**:

* + **Risks**: Potential challenges such as delays in test environment setup, incomplete requirements, or lack of test data.
  + **Mitigation Strategies**: Plans for addressing these risks, such as using mock data, prioritizing test cases, or adjusting the testing timeline.

**Test Scenarios**:

* + High-level descriptions of what needs to be tested, often based on functional requirements.
  + Example: "Test the login functionality for valid and invalid inputs."

**Test Data**:

* + Sample data sets required to execute the test cases. This could include valid and invalid user credentials, form inputs, database records, etc.
  + Example: A list of user credentials for testing login functionality.

**Test Execution Reports**:

* + Reports generated after executing the test cases. They will include:
    - **Pass/Fail Status**: Whether the test case passed or failed.
    - **Defects**: Any issues or bugs identified during testing.
    - **Test Coverage**: The extent to which the application is tested.
    - **Test Logs**: Detailed logs from the test runs for troubleshooting issues.
    - **Test Environment**: Information about the environment used for testing (e.g., browser versions, OS).