**Requirements For Para Bank & Restful Booker**

**1. Introduction**

This document outlines the functional and non-functional requirements for testing the Para Bank and Restful Booker applications. It includes user stories, use cases, and expected system behavior.

**2. Functional Requirements**

**2.1 Para Bank Functional Features**

1. **User Authentication & Authorization**
   * Customer login and authentication
2. **Customer Features**
   * Open a new account
   * View accounts overview
   * Transfer funds between accounts
   * Pay bills
   * Find and view transaction history
   * Update contact information
   * Request a loan
   * Log in / Log out
3. **Admin Features**
   * Database management (initialize and clear database)
   * Data access mode selection (SOAP, REST, JDBC)
   * Web service configuration (WSDL, WADL, OpenAPI)
   * Application settings (account, loan, and threshold configurations)

**2.2 Restful Booker API Features**

1. **Booking Management Endpoints**
   * Create a new booking
   * Retrieve an existing booking
   * Update booking details
   * Delete a booking
2. **Authentication & Security**
   * Supports token-based authentication for API access
3. **Negative Testing Cases**
   * Handle invalid or missing API request parameters
   * Validate proper error handling
4. **Performance & Security Testing**
   * API response time measurement
   * System performance can be evaluated under simulated load conditions

**3. Non-Functional Requirements**

1. **Performance**
   * API should respond within 200ms for valid requests
2. **Usability**
   * System should be accessible from multiple devices
   * API documentation should be clear and easy to follow
3. **Security  
   ParaBank:**
   * System should properly handle invalid inputs and prevent unauthorized access through basic validation.

**Restful Booker API:**

* + Token-based authentication must be implemented to restrict access.
  + API should return proper authorization error codes (401 for unauthorized, 403 for forbidden).
  + User sessions should be securely managed.

**4. User Stories & Use Cases**

**4.1 User Stories**

**As a Customer:**

* I want to log into my account so that I can manage my finances.
* I want to transfer funds between my accounts so that I can manage my money efficiently.
* I want to view my transaction history so that I can track my expenses.
* I want to update my contact information so that my details remain current.
* I want to request a loan so that I can meet my financial needs.

**As an Admin:**

* I want to manage the database so that I can maintain system integrity.
* I want to configure loan and account settings so that I can define banking rules.

**4.2 Use Cases**

**Use Case: Customer Login**

* **Actors**: Customer
* **Preconditions**: Customer is registered in the system.
* **Steps**:
  1. Customer enters username and password.
  2. System validates credentials.
  3. If valid, system grants access.
  4. If invalid, system displays an error message.

**Use Case: Transfer Funds**

* **Actors**: Customer
* **Preconditions**: Customer has an active account.
* **Steps**:
  1. Customer selects the transfer funds option.
  2. Customer enters the amount, source, and destination accounts.
  3. System validates the request.
  4. If valid, funds are transferred.
  5. Confirmation message is displayed.

**Use Case: API Booking Creation**

* **Actors**: External API consumer
* **Preconditions**: Valid API authentication.
* **Steps**:
  1. API consumer sends a request to create a booking.
  2. System validates request parameters.
  3. If valid, booking is created and ID is returned.
  4. If invalid, system returns an error message.

**5. Expected Behavior**

* The system should prevent invalid user actions with proper error messages.
* The API should return appropriate status codes (200 for success, 400 for invalid requests, 401 for unauthorized access, etc.).
* All user inputs should be validated based on defined constraints.
* The UI should be responsive and load within 3 seconds.