****

CRM Accounting Invoice Quote

**Testing Manual**

# Table of Contents

[Table of Contents 2](#_hkg590vvsisn)

[Introduction 3](#_2i2jrd6rm7lf)

[Project & System Overview 3](#_gu9vbpwxik5a)

[Testing Strategy & Approach 4](#_vx84gxxrjc3r)

[1. Functional Testing 4](#_ctpb0tkq6ls8)

[2. Integration Testing 4](#_hthuy9jyplh5)

[3. User Interface (UI) Testing 4](#_cq341lkl8z0h)

[4. Role-Based Access Testing 4](#_dbjevyp3nlis)

[5. Data Validation & CRUD Testing 5](#_zelhgnjogreo)

[6. Regression Testing 5](#_s1k7ce4lysoq)

[7. Usability Testing 5](#_22mxp2e8rnld)

[8. Compatibility Testing 5](#_44ntse7po4yv)

[9. Exploratory Testing 5](#_enh7oi2q3u43)

[10. White Box Testing 5](#_7njqjbbw0u79)

[Testing Approach 7](#_2cyvi9b0xvqz)

[Test Planning 7](#_3htxbr29olf0)

[Test Case Design 7](#_g93pm2evz9rj)

[Test Execution 7](#_eyonkbb7bwze)

[Defect Logging & Tracking 7](#_xrik37j77m0t)

[Test Reporting 7](#_o8j97f9z5ax2)

[Test Cycles 8](#_d6c422t8m5j5)

[Environment Details 8](#_89y2bkx6yn8n)

[Test Case Documentation 9](#_vywdt2ix1er7)

[Validation Checklist 9](#_m5z1zvu501xz)

[Test Techniques Applied 9](#_i0mifgdz3ecu)

[Test Schedule 12](#_3g7d2qmtpssf)

[Test Summary Report 13](#_rk9b6u6qqjhx)

# Introduction

**Purpose**Define the testing scope, objectives, and guidelines to assure IDURAR meets functional and quality benchmarks.

**Scope**Features under test cover Invoice, Payments, Customer, Quote

**Audience**QA team, developers, project stakeholders.

# Project & System Overview

**Description**  
The iDurar ERP is a modular, MERN-stack-based Enterprise Resource Planning system designed to streamline business operations across finance, sales, human resources, inventory, and customer management. Built using React.js (frontend), Node.js and Express.js (backend), and MongoDB (database), the system supports JWT-based authentication and RESTful APIs for communication between modules.

The objective of this testing manual is to ensure the stability, reliability, and usability of the system before it goes live or is delivered to stakeholders. Testing will cover all functional modules, integration points, and user interactions to validate both frontend behavior and backend processes.

**Features/Modules**  
Invoice, Quote, Payment, Customer, Inventory

**Tech Stack**  
 Frontend: React.js + Ant Design + Redux  
 Backend: Node.js + Express + MongoDB

# Testing Strategy & Approach

The testing process for iDurar ERP is structured to ensure complete functional validation, data accuracy, workflow reliability, and a secure user experience across all modules. Given the modular nature of ERP systems and the variety of user roles and transactions involved, a combination of manual and systematic functional testing techniques will be used.Our goal is to deliver a robust and user-validated ERP system that performs consistently under realistic business scenarios. The following testing strategies will be applied:

#### **1. Functional Testing**

* Focuses on validating individual modules such as Invoices, Quotes, Customers, Payments, and Reports.
* Ensures that each function works according to business rules and acceptance criteria.
* Includes positive, negative, and boundary tests to verify field-level and workflow-level behavior.

#### **2. Integration Testing**

* Ensures that modules communicate and operate smoothly with each other.  
  Examples include:  
  + A Quote being successfully converted to an Invoice.
  + Payments being linked and reflected in Invoices.
  + Customers being shared between modules like Quotes, Invoices, and Payments.

We didn't perform this as there is no service that has been integrated with the system.

#### **3. User Interface (UI) Testing**

Confirms that:

* + Pages load correctly in supported browsers (Chrome, Firefox, Edge).
  + Buttons, dropdowns, and forms are functional and intuitive.
  + Labels, validations, and notifications are accurate and user-friendly.

#### **4. Role-Based Access Testing**

* Validates that each user role (e.g., Admin, Accountant) only has access to the modules and data relevant to them.
* Tests include login scenarios, restricted module access, and CRUD permission enforcement.

#### **5. Data Validation & CRUD Testing**

* Ensures that data creation, updating, and deletion across forms (Customers, Invoices, Products, etc.) are functioning correctly.
* Includes database integrity checks and verification of auto-calculated fields (e.g., tax, totals).

#### **6. Regression Testing**

* Re-runs test cases after system updates, patches, or new feature deployments.
* Prevents re-introduction of previously resolved issues or workflow disruptions.

Note: Once the bugs are fixed Regression testing will be performed.

#### **7. Usability Testing**

* Performed from the end-user’s point of view to ensure ease of use, clarity of actions, and logical layout of screens.
* Ensures that workflows mimic real-world tasks (e.g., creating an invoice from a customer view).

#### **8. Compatibility Testing**

* Ensures application performance and layout consistency across:  
  + Browsers (Chrome, Firefox, Edge).
  + Devices (Desktop, Laptop, Tablet).

#### **9. Exploratory Testing**

* Performed by testers outside the scripted test cases to uncover hidden issues or usability gaps.

#### **10. White Box Testing**

* This strategy focuses on validating the internal logic, structure, and code execution paths within the ERP modules.
* Typically performed by developers or QA engineers with access to the source code.
* Examples include:  
  + Ensuring conditional branches (e.g., discount logic in invoices) behave correctly.
  + Verifying backend function output through unit tests (e.g., invoice total calculation, payment status updates).
  + Testing API-level logic and backend validation rules.

### 

### 

### 

### 

### 

### 

### 

### **Testing Approach**

The approach will be **iterative and module-driven**, matching the delivery pipeline of the ERP features. Here’s how the testing will be planned and executed:

#### **Test Planning**

* Define the test scope, modules covered, resources, and timelines.
* Prioritize modules based on business criticality and usage frequency.

#### **Test Case Design**

* Prepare detailed test cases for each module with:  
  + Test scenario ID
  + Pre-conditions
  + Steps to execute
  + Expected results
  + Actual results
  + Pass/Fail status

#### **Test Execution**

* Execute test cases manually on the **staging/test environment**.
* Record all results and capture evidence (screenshots, logs) for failures.

#### **Defect Logging & Tracking**

* Defects will be logged in a shared platform like Azure DevOps or GitHub Issues.
* Each bug will be assigned severity, priority, and ownership for tracking.

#### **Test Reporting**

* Daily/weekly summary reports will be shared with the project team.
* Metrics to track include: test case coverage, pass/fail rate, open/closed defects.

#### **Test Cycles**

* Multiple test cycles will be conducted per module:  
  1. Initial Testing
  2. Fix Verification
  3. Regression Cycle
  4. Final UAT (User Acceptance Testing)

#### **Environment Details**

* Testing will be performed in a dedicated staging environment.
* The backend will connect to a seeded MongoDB test database.
* Separate test accounts with different roles will be used to simulate real-world scenarios.

**Tools/Environment**:

* Jest/Mocha + Supertest (API)
* Cypress or Selenium (UI)
* MongoDB test DB

# 

# 

# 

# 

# 

# Test Case Documentation

All functional and integration scenarios for the iDurar ERP have been documented as structured test cases. Each case includes detailed steps, expected outcomes, and actual results to ensure consistent test execution and traceability.



# Validation Checklist

A comprehensive validation checklist has been prepared to confirm successful testing of each module, covering UI, logic, data flow, and user permissions.



# Test Techniques Applied

**Boundary Value Analysis**:

|  |  |  |
| --- | --- | --- |
| Field Name | Validation Technique | Description |
| Price | BVA | Checked values at boundaries (e.g., 0, 0.01, 999999.99, and negative input rejection). |
| Name | BVA | Tested min/max length of product name (e.g., 1, 50, 51 characters). |
| Quantity | BVA | Validate 0, 1, and upper threshold (e.g. 99999) — and out-of-range values. |
| Tax Rate | BVA | Validate 0, 0.01, 100, and 100+ to test edge boundaries. |
| Password | BVA | Test lengths: min allowed (e.g., 8), max (e.g., 32), and above/below. |

**Decision Table Testing**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Condition / Rule** | Rule 1 | Rule 2 | Rule 3 | Rule 4 |
| Product Name is entered | YES | YES | NO | YES |
| Product Category is selected | YES | NO | YES | NO |
| Price is greater than 0 | YES | YES | YES | YES |
| Image is uploaded | YES | YES | YES | YES |
| **Decision / Outcome** | Submit | Not Submit | Not Submit | Not Submit |
| **System Message** | Product is added | Select product category | Show "Enter name" | Show "Invalid price" |

* **State Transition**:

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | **Current State** | | **Trigger (Event/Input)** | **Next State** |
| Draft | Click “Submit” | Pending |
| Pending | Click “Send for Approval” | Sent |
| Sent | Reviewer clicks “Accept” | Accepted |
| Sent | Reviewer clicks “Decline” | Declined |
| Declined | User clicks “Resubmit” | Pending |

* **Error Guessing**: Missing required fields, invalid input.

# Test Schedule

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Phase** | **Start Date** | **End Date** | **Tasks** | **Responsibility** |
| Test Planning | 01‑Jun‑2025 | 03‑Jun‑2025 | Analyzing BRD, Draft of Testing Manual | QA |
| Testing Setup | 03‑Jun‑2025 | 04‑Jun‑2025 | Assigning Tasks, Ask for Ready to Test Build | QA + DEV |
| Test Case Writing | 05‑Jun‑2025 | 10‑Jun‑2025 | Document Test cases and draft Test Summary Report | QA |
| White Box Testing | 11‑Jun‑2025 | 14‑Jun‑2025 | The project was installed following the provided instructions. The database schema and data in MongoDB Atlas were thoroughly tested and validated. Methods defined in the code but never invoked were identified, along with segments of unused code that were still committed to the repository. Additionally, unit tests were not found, making it difficult to assess test coverage. | DEV |
| UI Testing and Compatibility Testing | 15‑Jun‑2025 | 21‑Jun‑2025 | Check the User Interface and Usability of the system | QA |
| Functional Testing | 15‑Jun‑2025 | 10‑July‑2025 | Ensure the functionality of all the features | QA |

# Test Summary Report

* **Total Test Cases**: 53
* **Passed**: 49 | **Failed**: 4
* **Critical Issues**: 3 open
* **Lessons Learned**:  
  + Perform smoke tests whenever the build has been released.
  + Test case status should be maintained and updated.
* **Future Improvements**:  
  + The error content should be concise and clear.
  + The leads, product and invoice pages have to be fixed.