**Record of change**

\*A - Added M - Modified D - Deleted

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Effective Date** | **Changed Items** | **A\* M, D** | **Change Description** | **New Version** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**SIGNATURE PAGE**

**ORIGINATOR:** <Name> <Date>

< Role>

**REVIEWERS:** <Name> <Date>

< Role >

<Name, if it’s needed> <Date>

< Role >

**APPROVAL:** <Name> <Date>

< Role >

**TABLE OF CONTENTS**

[1](#_heading=h.jyvebp8hlgu1) INTRODUCTION 6

[1.1 Purpose 6](#_heading=h.am7o8ocz3w58)

[1.2 Definitions, Acronyms, and Abbreviations 6](#_heading=h.les9y9vyg01i)

[1.3 References 7](#_heading=h.z3couw3upomm)

[1.4 Background information 7](#_heading=h.lqgtobaktlz7)

[1.5 Scope of testing 7](#_heading=h.xzywy3p42zdk)

[1.6 Constraints 8](#_heading=h.8orsiev2y66)

[1.7 Risk list 8](#_heading=h.lagnqjpve3p4)

[1.8 Training needs 8](#_heading=h.2opl7ddnujsb)

[2](#_heading=h.1vfou92zmved) Requirements for Test 9

[2.1 Test items 9](#_heading=h.mcbhwshtimu3)

[2.2 Acceptance Test Criteria 9](#_heading=h.21lj9uqso9jf)

[3](#_heading=h.29y6nc1u05su) TEST STRATEGY 10

[3.1 Test types 10](#_heading=h.wgreld8jotxy)

[3.1.1 Function Testing 10](#_heading=h.nyoxzj9vb50v)

[3.1.1.1 Function Testing 10](#_heading=h.g2clypvxkogq)

[3.1.1.2 Business Cycle Testing 11](#_heading=h.qcxcrn5zb51s)

[3.1.2 User Interface Testing 11](#_heading=h.tbn2l1g3nw2x)

[3.1.3 Data and Database Integrity Testing 12](#_heading=h.ywsuvjgwrqmb)

[3.1.4 Performance testing 13](#_heading=h.en44i46864dw)

[3.1.4.1 Performance testing 13](#_heading=h.sxti6jwekpp9)

[3.1.4.2 Load Testing 14](#_heading=h.o48jz8x9nozj)

[3.1.4.3 Stress Testing 15](#_heading=h.2d16jj2u391r)

[3.1.4.4 Volume Testing 16](#_heading=h.2d16jj2u391r)

[3.1.5 Security and Access Control Testing 17](#_heading=h.evvz2812fgkp)

[3.1.6 Regression Testing 18](#_heading=h.dooio6ifqq0t)

[3.2 Test stages 18](#_heading=h.t2rrrc1v6w5y)

[4](#_heading=h.qlil1z2hooc2) RESOURCE 20

[4.1 Human Resource 20](#_heading=h.m75k7yg8ngpd)

[4.2 Test management 20](#_heading=h.k4dzux8ap2or)

[5](#_heading=h.cqylh34te23i) Test environment 21

[5.1 Hardware 21](#_heading=h.4o55oamt39d2)

[5.2 Software 21](#_heading=h.vclh1g1xzddd)

[5.3 Infrastructure 21](#_heading=h.8j0y7af5ond7)

[6](#_heading=h.ogb6pg2ia2e0) TEST MILESTONES 22

[7](#_heading=h.8cr3hqzhk0rx) DELIVERABLES 23

# 

# INTRODUCTION

## Purpose

The purpose of this document is to outline the Test Strategy/Methodology and Quality Assurance process for fast food delivery. This document will establish a System test plan for fast food delivery website. This plan will allow the development team, business analysts and project managers to coordinate their efforts and effectively manage the testing of the website. The Quality Assurance (QA) process outlined in this System Test Plan will ensure that a quality fast food delivery website is deployed successfully and on schedule.

## Definitions, Acronyms, and Abbreviations

*<This subsection provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the Test Plan>*

|  |  |  |
| --- | --- | --- |
| **Abbreviations** | **Description** | **Note** |
| AT | Acceptance test |  |
| B Voucher | Bug voucher |  |
| DMS | Defect Management System (Fsoft tool) |  |
| ES | Enhance Specification |  |
| IT | Integration test |  |
| PM | Project Manager |  |
| PTL | Project Technical Leader |  |
| PT/TT | Program test/ Total test |  |
| P Voucher | Program voucher |  |
| QA | Quality Assurance |  |
| QUP | Quality up |  |
| SRS | Software Requirement Specification |  |
| ST | System test |  |
| TP | Test Plan |  |
| TC | Test Case |  |
| TR | Test Report |  |
| UAT | User Acceptance test |  |
| UT | Unit test |  |

## References

| **#No** | **Title/File name** |
| --- | --- |
| **1** | Fast Food Delivery\_Software Requirements Document |
| **2** |  |
| **3** |  |

## Background information

The objective of this test is a Full Stack Food Ordering Web Application built using React JS, Node.js (Express), MongoDB and Stripe. The application allows users to register/login, browse the menu, add products to the cart, place orders and complete online payments. The application also provides an admin dashboard to manage products and order status.

The system follows the MERN model: React handles the front-end and user interface, Express and Node.js provide the back-end API and business logic, MongoDB stores the application data and Stripe enables secure payment processing. The project aims to provide a responsive, scalable and user-friendly food delivery platform that supports both customers and administrators.

## Scope of testing

***1.5.1. Testing phases***

The following table lists the various phases of International-kids.com application testing and the team responsible for it.

|  |  |
| --- | --- |
| Phase | Teams Responsible |
| Unit Testing | Development team |
| Integration Testing | Testing team |
| System Testing | Testing team |
| User Acceptance Testing | Fastfooddelivery User Representatives |

***1.5.2. Testing Types***

|  |  |
| --- | --- |
| Activity | Teams Responsible |
| Functionality Testing | * Performed by testing team during Integration/System testing phase to meet agreed upon functional requirements of fast food delivery website. * All the features put under test are mentioned in brief under “Features to be tested” section in this plan. * documents and Test case documents. On completion of every single functional area, test scenario and test case documents will be delivered |
| Database Testing | * Performed by testing team during Integration/System testing phase to qualify database which houses the content that the fast food delivery application manages, run queries and fulfill user requests for data storage |
| Security Testing | * Performed by testing team during Integration/System testing phase to meet agreed upon Security requirements of fast food delivery application |
| GUI and Usability Testing | * Performed by testing team during Integration/System testing phase |
| Performance and Load /Volume Testing | * Performed by testing team during System Testing phase. Automation testing will be performed to carry out these types of testing. * (Tool name to be decided/updated) tool will be used to perform these tests.Various Reports that are part of fast food delivery Application will be one of the main areas while performing load/volume testing Performance test methodology. |
| Code Testing | * Performed by development team during Unit Testing phase at every method level. |
| Smoke Testing | * Performed by development team during Unit Testing phase for qualifying the build for releasing it to Testing team. Performed by Testing team during Integration/System phase for qualifying the build for further tests. |
| Regression Testing | * Performed by testing team during Integration/System testing phase for re- testing an entire or partial system after a modification has been made to ensure that no unwanted changes were introduced to the system. |
| Defect fix verification testing/Defect validation testing) | * Performed by testing team during Integration/System testing phase for verifying the defect fixes |
| Compatibility Testing | * Performed by Development team during Unit testing phase Performed by testing team during Integration/System testing phase and Performed by International-kids.com team during UAT phase in order to have a complete test. PA testing team will be responsible for testing this functionality by accessing Fast Food Delivery -QA environment |
| Interface Testing | * Performed by Testing team during Integration/System testing phase to test the Navigations that are unusual and Negative scenarios within and across the components. |

***1.5.3. Not in Scope***

## Constraints

*1.6.1 Test Environment Constraints*

* *The system integration with the Stripe payment gateway and email notification service is performed using sandbox mode, without real financial transactions.*

*1.6.2 Tool and Resource Constraints*

* *Due to limited hardware and network resources, the system may respond slowly when multiple users access it simultaneously.*
* *Test data is generated manually, as there are no automated data generation tools available.*

*1.6.3 Human Resource Constraints*

* *There is no dedicated QA or security testing team, and advanced testing such as performance or usability testing is not covered extensively.*

*1.6.4 Time Constraints*

* *Some extended features, such as real GPS-based delivery tracking or external email notifications, are simulated rather than fully implemented.*

## Risk list

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *No.* | *Risk Description* | *Impact* | *Mitigation Strategy* | *Contingency Plan* |
| *1* | ***Test environment instability****-The local server or database may crash during testing.* | *High* | *Regularly back up the database and configuration files. Use lightweight datasets during test execution.* | *Reinstall the local environment or restore from the latest backup.* |
| *2* | ***Third-party service unavailability*** *– Stripe payment API or email service (e.g., SendGrid) may be unavailable during testing.* | *Medium* | *Use sandbox endpoints and mock services for integration tests.* | *Delay related test cases and perform re-testing once the service is restored* |
| *3* | ***Limited testing tools****-Lack of advanced automation or load testing tools may reduce test coverage.* | *Medium* | *Focus on manual functional testing of key features. Use open-source tools where possible.* | *Perform partial automation later when tools or time become available.* |
| *4* | ***Insufficient test data*** *– Lack of realistic data may lead to incomplete test scenarios.* | *Medium* | *Create representative sample datasets that cover all functional flows* | *Generate additional mock data if new cases arise during testing.* |
| *5* | ***Human resource limitation*** *– Only one tester (student) available to execute all test cases.* | *High* | *Prioritize critical functionalities (order processing, payment, authentication).* | *Extend testing time frame slightly if needed or request peer review.* |
| *6* | ***Tight project schedule*** *– Short timeline within the semester may restrict full regression testing* | *High* | *Plan and execute testing in small iterations (unit → integration → system)* | *Reduce the scope of lower-priority tests if the deadline is near.* |
| *7* | ***Tool or software compatibility issues*** *– Version mismatches between Node.js, database, or libraries may cause failures.* | *Medium* | *Document software versions and lock dependencies (package-lock.json).* | *Reinstall dependencies or rollback to previous working versions* |
| *8* | ***Security configuration risks*** *– Improper handling of JWT tokens or bcrypt setup may expose vulnerabilities.* | *Medium* | *Follow standard security configurations; perform manual validation of token and password flows.* | *Disable risky features temporarily and fix in the next release iteration.* |
| *9* | *Network connectivity issues – Internet connection interruptions may affect testing of online API calls.* | *Low* | *Conduct as much testing as possible in offline/local mode.* | *Resume testing once the network is restored* |

## Training needs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *No.* | *Training Topic* | *Purpose / Description* | *Target Participants* | *Training Type* |
| *1* | *Software Testing Fundamentals* | *Understand core testing concepts such as test levels, test types, and test design techniques (black-box, white-box).* | *All team members* | *Self-study / Online course* |
| *2* | *Test Plan and Test Case Design* | *Learn how to design structured test cases and maintain traceability to requirements.* | *QA role / Student tester* | *Workshop / Guided learning* |
| *3* | *Manual Testing using Postman* | *Practice sending API requests, validating responses, and verifying integration between backend and frontend* | *Student tester* | *Hands-on practice* |
| *4* | *Automation Testing with Selenium (basic)* | *Learn to automate browser-based tests for key website functionalities such as login and order placement* | *Student tester* | *Tutorial / Demo session* |
| *5* | *API Testing and Validation* | *Understand how to test RESTful APIs using Postman or Newman, and validate request-response structures* | *Student tester* | *Self-practice* |
| *6* | *Performance Testing using JMeter (intro)* | *Learn how to simulate multiple users and evaluate website performance under load.* | *Student tester* | *Guided demo / Experiment* |
| *7* | *Version Control (Git & GitHub)* | *Manage source code, test scripts, and track changes through Git version control* | *All team members* | *Self-learning via documentation* |
| *8* | *Defect Management Process* | *Learn how to log, track, and report defects using a simple issue-tracking tool (e.g., GitHub Issues or Excel)* | *QA / Developer* | *Instructor-led / Self-practice* |
| *9* | *Security Basics (JWT & bcrypt)* | *Understand how authentication and password encryption are handled and tested* | *Developer & Tester* | *Reading / Discussion* |
| *10* | *CI/CD and Test Automation Integration (optional)* | *Gain an overview of how automated testing can be integrated in the CI/CD pipeline using GitHub Actions* | *Developer / Tester* | *Self-learning tutorial* |

# Requirements for Test

## Test items

*<The listing below identifies those items (use cases, functional requirements, non-functional requirements) that have been identified as targets for testing. This list represents what will be tested and the number of test case estimated for testing each item.*

*Enter a high level list of features and functions to be tested/not tested. Refer to the sample in Guideline Test Plan>.*

*The table below can be used if suitable.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Name of features and functions** | **Outline of features and functions** | **Number of Test case(Estimate)** | **Note** |
|  | Menu Catalog Management | Allows customers to browse food items by category, filter, and view (name, description, image, price, availability). Admin can add, update, or delete menu items | 10 | Includes CRUD operations and display checks. |
|  | Shopping Cart | Customers can add, remove, or update items in their cart. The total amount automatically updates, including delivery fee and discounts. | 8 | Focus on dynamic updates and price recalculation. |
| 3. | Payment Process | System validates order data, integrates with Stripe (sandbox) for payment | 7 | Uses mock Stripe API |
| 4. | Order Management | Stores confirmed orders, allows customers to view order history, and enables Admin to update order statuses (Pending → Preparing → Delivering → Completed) | 9 | Includes both user-side and admin-side verification. |
| 5. | Delivery Tracking | Displays real-time order progress. Status updates are triggered by Admin or delivery staff. | 5 | Simulated GPS updates in test environment |
| 6. | Access Control (Authentication & Authorization) | Allows login/logout for both Customer and Admin. Redirects users to correct page based on role. Uses JWT tokens and bcrypt for security. | 6 | Includes both valid and invalid login scenarios. |
| 7. | Admin Dashboard   |  | | --- | |  |  |  | | --- | |  | | Provides interface for Admin to manage menus, users, and orders. Includes data summary views. | 6 | Focus on data accuracy and role-based access. |
| 8. | System Usability and UI Testing | Checks layout consistency, button functions, and user navigation flow. | 4 | Includes cross-browser verification. |
| 9. | Non-Functional Testing (Performance & Security) | Verifies response time under load and basic token-based security handling. | 4 | Performed at basic level due to student environment |

## Acceptance Test Criteria

|  |  |  |
| --- | --- | --- |
| **No** | **Criteria** | **Description / Target Level** |
| 1. | Test Coverage | At least **95%** of all functional and non-functional requirements (ordering, payment, delivery tracking, admin dashboard) must be covered by corresponding test cases |
| 2. | Successful Test Execution Rate | A minimum of **90%** of executed test cases must pass without any **Critical** or **High-severity** defects remaining open. |
| 3. | Defect Density | The number of high-severity defects must not exceed **2 defects/KLOC** (per thousand lines of code). |
| 4. | Defect Closure Rate | At least **95%** of all reported defects must be resolved and verified before proceeding to UAT |
| 5. | Statement Coverage (Unit Test) | Minimum **85%** statement coverage should be achieved for core backend modules (Order Management, Payment API, Authentication) |
| 6. | Branch Coverage (Unit Test) | Minimum **80%** branch coverage for critical business logic such as payment confirmation and order status update |
| 7. | Integration Test Coverage | All critical integrations (e.g., Payment Gateway, GPS Delivery Tracking, and Database) must be tested with **100% of integration scenarios** executed successfully. |
| 8. | System Stability | The system must operate continuously under normal user load (e.g., 50 concurrent users) for **at least 24 hours** without downtime or crash. |
| 9. | Performance Acceptance | Response time for key user actions (place order, update status, view delivery) must be **≤ 3 seconds** under normal load |
| 10. | User Acceptance (UAT) | The final release must be approved by stakeholders or the supervising instructor, confirming that all core functionalities meet expected quality and usability standards |

## 

# TEST STRATEGY

*<The Test Strategy presents the recommended approach to the testing of the target-of-test. Outline for test strategy, refer to Guideline Test Plan. >*

## Test types

### Function Testing

#### Function Testing

|  |  |
| --- | --- |
| **Test Objective:** | Ensure proper backend functionality of the Fast\_Food\_Delivery system, including API navigation, data entry, processing, and retrieval across modules such as User, Food, Cart, and Order. The objective is to verify that each API endpoint works according to its business rules and requirements. |
| **Technique:** | * The expected results occur when valid data is used (e.g., successful login, order creation). * The appropriate error or warning messages are displayed when invalid data is used (e.g., missing fields, invalid token). * Each business rule is properly applied (e.g., cannot order without items, cannot register with duplicate email). |
| **Completion Criteria:** | * All identified defects have been logged, fixed, re-tested, and closed. * All main user flows (login, food listing, cart management, order creation) function correctly without blocking defects. |
| **Special Considerations:** | * Authentication tests depend on valid JWT tokens generated by the /login API. * File upload tests (for food images) require correct configuration of the uploads/ directory and middleware. * Some routes are protected by middleware and require proper authorization headers. |

#### Business Cycle Testing

*<Outline for Business Cycle Testing. Refer to the sample in Guideline Test Plan>*

|  |  |
| --- | --- |
| **Test Objective** | Ensure that all backend processes and business operations of the **Fast\_Food\_Delivery** system function correctly according to the defined business models and schedules. This includes verifying that order processing, payment flow, and background job executions (such as order auto-updates or data cleanup) occur properly throughout a full business cycle. |
| **Technique:** | * Modify or enhance existing functional test cases to repeatedly execute key business flows (e.g., user registration → order placement → payment → order completion) to simulate multiple concurrent users over a specified duration. * Execute all time- or date-sensitive operations using valid and invalid inputs (e.g., expired promo codes, outdated orders, expired authentication tokens) to ensure correct system handling. * Test all processes that occur on a periodic schedule, ensuring they trigger and execute as expected (e.g., scheduled order cleanup or report generation). * Validate with both valid and invalid data to confirm that * Expected results occur with valid data. * Appropriate error or warning messages are displayed with invalid data. * All business rules (e.g., cannot reorder canceled items, must authenticate before placing an order) are correctly enforced. |
| **Completion Criteria:** | * + All identified issues or defects have been documented, addressed, and verified as fixed.   + The system demonstrates stable operation through multiple business cycles without functional or data integrity errors |
| **Special Considerations:** | * Some date/time-sensitive operations may need simulated clock adjustments or test data manipulation. * Test environment must support running multiple simulated users and scheduled background services concurrently. * Dependency on external services (e.g., payment gateway, email API) should be mocked or isolated to prevent false failures during cycle simulation. |

### User Interface Testing

|  |  |
| --- | --- |
| **Test Objective:** | Verify that the user interface of the Fast\_Food\_Delivery system accurately supports all business functions and provides a consistent and user-friendly experience. Ensure that navigation, layout, and visual components operate as expected and conform to established UI/UX standards. Specifically verify that:   * Navigation through the application properly reflects business requirements, including transitions between pages, input fields, and menus. * Window objects and characteristics such as menus, size, position, state, and focus conform to design standards. * Access methods (mouse clicks, tab keys, keyboard shortcuts) function properly and consistently across all screens. |
| **Technique:** | * Check navigation between main pages (Home, Menu, Cart, Checkout, Admin Dashboard). * Verify all buttons, input fields, and links perform the expected actions. * Confirm that pop-ups, modals, and notifications display and close correctly. * Ensure responsive design works properly on various screen sizes (desktop, tablet, mobile). * Testing will be performed manually using browser tools and visually inspected for alignment, colors, and layout compliance. Automated UI testing tools (e.g., Selenium or Cypress) may be used where applicable. |
| **Completion Criteria:** | * Navigation, window states, and interactions remain consistent with the design specifications * No critical UI defects remain unresolved. |
| **Special Considerations:** | * UI rendering may vary slightly across different browsers and operating systems; visual consistency tolerance should be defined. * Color contrast and accessibility should be reviewed to ensure compliance with usability standards (e.g., WCAG). |

### Data and Database Integrity Testing

|  |  |
| --- | --- |
| **Test Objective:** | Ensure that all database access methods, queries, and processes in the Fast\_Food\_Delivery system function correctly, maintain data integrity, and prevent data corruption. Verify that CRUD (Create, Read, Update, Delete) operations work as expected across all collections, including Users, Foods, Carts, and Orders. |
| **Technique:** | * Verifying data insertion for valid records (e.g., new user registration, order creation). * Attempting to insert invalid or incomplete data to confirm appropriate validation and error handling. * Executing read operations to ensure correct data retrieval for the right users and queries. * Performing update and delete operations to validate that data changes are accurately reflected. * Inspecting the MongoDB database directly to ensure data integrity and that no duplicate, missing, or corrupted records exist. * Monitoring Mongoose middleware (e.g., pre-save, post-update hooks) to verify expected behavior. |
| **Completion Criteria:** | * No data corruption, duplication, or loss occurs during or after testing. * All CRUD operations return accurate and consistent results. |
| **Special Considerations:** | * Some processes may need to be executed manually or with automated scripts to simulate real transactions. * Testing should be conducted on a small dataset to easily detect anomalies or data inconsistencies. * Backup and restore operations should be tested separately to verify data recovery capability. |

### Performance testing

#### Performance testing

*<Performance test is to measure and evaluate response times, transaction rates, and other time-sensitive requirements. It includes Load test, Stress test, Volume test...*

*For more details, refer to the sample in Guideline Test Plan >*

|  |  |
| --- | --- |
| **Test Objective:** | Verify the performance behavior of the **Fast\_Food\_Delivery** system under various workload conditions. Ensure that response times, transaction rates, and system throughput meet performance requirements during normal and peak usage scenarios. Performance testing will validate system behavior under both normal anticipated workload and worst-case workload. |
| **Technique:** | * Measuring response time for key transactions such as login, food listing retrieval, order placement, and payment processing under normal user load. * Increasing the number of transactions or iterations in test scripts to simulate peak traffic and concurrent user activity. * Executing test scripts on a single client to establish a performance benchmark, then repeating tests with multiple concurrent virtual users to assess scalability. * Monitoring server metrics such as CPU usage, memory consumption, and database query latency throughout testing. |
| **Completion Criteria:** | * Single transaction or single user: Successful completion of test scripts without failure and within expected time limits per transaction. * Multiple transactions or multiple users: Successful completion of test scripts without system crashes or timeouts and within acceptable response time thresholds. * Performance metrics must remain within predefined limits (e.g., 95% of requests completed under 2 seconds) |
| **Special Considerations:** | * Background server workload should be simulated to reflect realistic operating conditions. * Multiple testing methods may be applied, including: * Driving transactions directly to the backend through API calls. * Creating virtual users to simulate hundreds of concurrent clients. * Running scripts from multiple physical or virtual machines to distribute system load. * The database used for performance testing should be representative of the production size or proportionally scaled. * Network latency and bandwidth should be monitored to isolate system performance from external factors. |

#### Load Testing

*<Load testing is a performance test which subjects the target-of-test to varying workloads to measure and evaluate the performance behaviors and ability of the target-of-test to continue to function properly under these different workloads. For more details, refer to the sample in Guideline Test Plan >*

|  |  |
| --- | --- |
| **Test Objective:** | Verify the performance behavior and response time of the **Fast\_Food\_Delivery** system under varying workload conditions. Ensure that the application continues to function correctly and efficiently as the number of concurrent users and transactions increases. |
| **Technique:** | * Gradually increasing the number of simulated users or transactions to measure system performance and identify potential bottlenecks. * Modifying test data and scripts to perform repeated execution of key operations such as login, food listing, adding to cart, and order placement. * Monitoring system resources (CPU, memory, I/O, and database query time) to determine the system’s ability to maintain acceptable performance under increasing load. |
| **Completion Criteria:** | * Response times remain within acceptable limits defined by performance requirements. * The system demonstrates stable behavior with consistent throughput under sustained load conditions. |
| **Special Considerations:** | * The test database should be either of actual production size or proportionally scaled to simulate realistic conditions. * Network configuration and bandwidth should be consistent during testing to avoid skewed results. * Load testing results should be analyzed alongside server monitoring data to identify resource utilization patterns and performance constraints. |

#### 

#### Volume Testing

*< Volume Testing subjects the target-of-test to large amounts of data to determine if limits are reached that cause the software to fail. For more details, refer to the sample in Guideline Test Plan >*

|  |  |  |
| --- | --- | --- |
| **Test Objective:** | |  | | --- | | The goal is to ensure that the application and its database can handle the following high-volume scenarios without degradation or failure: |  * Maximum number of concurrent clients connected or simulated, all performing high-frequency business functions (e.g., placing or viewing orders, updating menus). * Maximum database size (actual or scaled) has been reached, and multiple concurrent queries or reports are executed simultaneously over an extended period. |
| **Technique:** | * Simulating a high number of users accessing the system simultaneously using automated load generation tools (e.g., k6, Artillery, or JMeter). * Creating or importing a large dataset in MongoDB (e.g., tens of thousands of menu items, users, and orders). * Executing complex queries, reporting operations, and data aggregation tasks repeatedly under heavy load. * Running concurrent database write/read operations for an extended duration to assess long-term stability. * Monitoring system metrics such as response time, memory usage, database query latency, and CPU load throughout the test period. |
| **Completion Criteria:** | * The system remains operational under heavy data and transaction volumes. * No data corruption, query failure, or service unavailability occurs. * Performance degradation remains within acceptable thresholds defined in the project requirements. * Any limits or breaking points are clearly identified and documented |
| **Special Considerations:** | * The acceptable time period for continuous high-volume testing should simulate expected peak usage (e.g., several hours of simulated real-world traffic). * Database indexing, caching strategies, and query optimization should be observed for effectiveness under load. * Consider testing with replica sets or sharded clusters if MongoDB scaling is expected in production. * Resource monitoring tools (e.g., PM2, MongoDB Profiler, or system dashboards) should be used to record performance metrics throughout the test cycle. |

### 

### Security and Access Control Testing

|  |  |
| --- | --- |
| **Test Objective:** | Verify that the Fast\_Food\_Delivery system properly enforces security policies and access controls for all user types and system components.  The goal is to ensure that application-level and system-level security mechanisms prevent unauthorized access while allowing legitimate operations: |
| **Technique:** | Security testing will validate role-based access control across all controllers and middleware components.  Key test procedures include:   * Testing each user type with assigned controllers and data access permissions * Verifying permission boundaries and role-based restrictions * Testing authentication and authorization scenarios (JWT validation via auth.js) * Validating system-level access controls and API endpoint security |
| **Completion Criteria:** | * All security test cases have been executed successfully across all user roles * Unauthorized access attempts are properly blocked with appropriate error responses * Users can only access authorized functions in cartController.js, foodController.js, orderController.js, and userController.js |
| **Special Considerations:** | * Security testing requires coordination for database security (db.js) and file upload directory protections * Testing should validate both authenticated routes and public accessibility of system resources |

### Regression Testing

*<Regression testing is a necessary maintenance activity aimed at showing that code has not been adversely affected by changes. For more details, refer to the sample in Guideline Test Plan >*

|  |  |
| --- | --- |
| **Test Objective:** | Verify that modifications and enhancements to the Fast\_Food\_Delivery system do not adversely affect existing functionality or introduce new defects.  The goal is to ensure that code changes, bug fixes, and new features maintain system stability and do not cause regression in previously working components: |
| **Technique:** | Regression testing will reuse and extend existing test suites from previous testing phases.  Key test procedures include:   * Selecting 80% of test cases randomly from existing functional, integration, and system test suites * Creating automated test scripts for critical business workflows using testing tools * Building program-analysis infrastructure to identify regression test scope based on code changes * Executing tests against modified modules and their dependencies |
| **Completion Criteria:** | * All selected regression test cases have been executed and passed successfully * Modified modules function as intended without impacting existing functionality * Automated regression suite provides comprehensive coverage of critical system components |
| **Special Considerations:** | * Regression testing should be performed after each code modification or release cycle * Test environment must closely mirror production to ensure accurate validation * Balance between comprehensive coverage and testing efficiency must be maintained |

## Test stages

*<Clearly state the stage in which the test will be executed. Identified below are the stages in which common test are executed>*

| **Type of Tests** | **Stage of Test** | | | |
| --- | --- | --- | --- | --- |
| **Unit** | **Integration** | **System** | **Acceptance** |
| <Function Test > | X | X | X | X |
| <User Interface test> | X |  | X |  |
| <Performance Tests  <Performance profiles of individual components> | X | X |  |  |
| <Load, Stress, Volume test> |  |  | X | X |
| <Security test> | X |  | X |  |
| <Date integrity test> |  | X | X |  |
| Database Testing |  | x | x |  |
| Business Cycle Testing |  |  | x |  |

# 

# RESOURCE

## Human Resource

|  |  |  |  |
| --- | --- | --- | --- |
| **Worker/Doer** | **Role** | **Specific Responsibilities/Comments** | **Location** |
| Đỗ Nguyễn Nhật Trường | <Test Leader> | • Manage test resources and assign test tasks  • Create and maintain Test Plan  • Review Test Cases  • Create Test Reports  • Coordinate with development team |  |
| Nguyễn Thanh Tịnh | <Tester > | • Create TC for User Management & Authentication modules  • Execute functional and integration tests  • Report test results and defects  • Perform security testing |  |
| Nguyễn Tấn Thành | <Tester > | • Create TC for Food & Cart Management modules  • Execute test cases  • Report test results  • Perform UI/UX testing |  |
| Nguyễn Thị Diệu Linh | <Tester > | • Create TC for Order & Payment modules  • Execute system and regression tests  • Report test results  • Perform performance testing |  |

## Test management

*<Define about the following items’ method:*

*Test management (Test planning and tracking, Communication)*

*Defect management (Defect management tool and defect process follow……)*

*Refer to the sample in Guideline Test Plan>*

# Test environment

*<List of required hardware and software resources. Describe equipments which needs for testing such as USB Memory, Memory card…etc. Describe the number of equipments need and the duration using these equipments. >*

## Hardware

*<Describe the hardware which projects will use for testing*

*Refer to the sample in Guideline Test Plan>*

## Software

*<Describe the software which projects will use for testing*

*Refer to the sample in Guideline Test Plan>*

## Infrastructure

*<List tools will be employed for this project.*

*Refer to the sample in Guideline Test Plan>*

|  |  |  |  |
| --- | --- | --- | --- |
| **Purpose** | **Tool** | **Vendor/In-house** | **Version** |
| <Defect log> | <DMS2> | <Fsoft tool> | <1.4> |
| <Test effort> | <Timesheet> | <Fsoft tool> | <3.8.4> |

# 

# TEST MILESTONES

*<Separate test milestones, which should be identified to communicate project status accomplishments>*

|  |  |  |  |
| --- | --- | --- | --- |
| **Milestone Task** | **Effort (pd)** | **Start Date** | **End Date** |
| <Create Unit Test Plan> |  | <YY-MM-DD> | <YY-MM-DD> |
| <Review & update UTP> |  |  |  |
| <Create Unit Test case> |  |  |  |
| <Review & update UTC> |  |  |  |
| <Create Test Plan> |  |  |  |
| <Review & update TP> |  |  |  |
| <Create Integration Test case> |  |  |  |
| Review & Update Integration TC> |  |  |  |
| <Create System Test case> |  |  |  |
| <Review & Update System TC> |  |  |  |
| <Execute Unit Test> |  |  |  |
| <Execute Integration test> |  |  |  |
| <Execute System test> |  |  |  |

# 

# DELIVERABLES

*<Define delivery of Testing. Refer to Project Plan for more details.>.*

| **No** | **Deliverables** | **Language** | **Delivered Date** |
| --- | --- | --- | --- |
|  | <Unit Test Plan> | <Japanese> | <YY-MM-DD>\* |
|  | <Test Plan> |  |  |
|  | <Unit Test cases> |  |  |
|  | <Integration Test Cases> |  |  |
|  | <System Test cases> |  |  |
|  | <Defect log> |  |  |
|  | <Test reports> |  |  |

*Note: \* <YY-MM-DD> is only used when customer is Japanese.*

*Content guideline in <> must be deleted when finish test plan.*