Test Plan Template

1. **Introduction**
   1. Test Plan Objectives:

The project involves creating a package delivery application for a local delivery service using three trucks, each assigned to specific routes. When a shipment is ready, the program determines which truck has the space to handle the package and deliver it closest to the target destination. The key inputs are package weight, size, and destination coordinates. The goal of this test plan is to verify system reliability, performance, and user satisfaction, aiming to identify issues and optimize quality.

1. **Scope**

What to Test:

* + - * + Performance Evaluation: Assess responsiveness by simulating workloads that check if the system can handle shipment allocation efficiently.
        + Functionality Validation: Verify the assignment of packages to trucks, focusing on optimal space usage and route calculation.
        + Capacity Testing: Test truck capacities with various package dimensions, ensuring compliance with size and weight limits (maximum 1000 kg and 36 cubic meters).
        + System Stability: Validate error-handling processes to ensure system resilience.

What Not to Test:

* + - * + External Dependencies: Testing does not cover third-party service integration.
        + Security and Penetration: Penetration tests are not included, though security will be considered.
        + Hardware & Network: The plan excludes hardware and network infrastructure testing, focusing instead on software functionality.

1. **Test Strategy**
   1. **This strategy covers multiple test types:**

3.1. System Test: Ensures that package allocation and route optimization work correctly.

3.2. Performance Test: Checks system efficiency under different loads.

3.3. Security Test: Assesses potential vulnerabilities.

3.4. Automated Testing: Automates repeatable tests using scripts to improve efficiency.

3.5. Stress and Volume Test: Pushes the system to peak capacity for stability evaluation.

3.6. Recovery Test: Verifies error-handling and fault tolerance.

3.7. Documentation Test: Ensures accuracy in user and technical documentation.

3.8. Beta Test: The application is tested by a small group of real users to uncover practical issues, evaluate usability, and gather feedback to improve performance.

3.9. User Acceptance Test: It ensures the application meets all requirements, with users testing key features like route optimization in real scenarios to confirm system reliability and user satisfaction before launch.

* 1. The test design involves understanding requirements, building a traceability matrix, creating comprehensive test cases, and undergoing peer review to ensure complete and accurate testing.

1. **Environment Requirements**
   1. The test environment includes specific hardware and software setups:
      * + - Hardware: Test computers with consistent network access.
          - Software: Compatible operating systems and development tools, including city map simulations for realistic testing.
2. **Execution Strategy**
   1. Entry Criteria
      * + - To begin testing, the following must be completed: understanding requirements, setting up the environment, creating test cases, and preparing test data.

Exit Criteria

* + - * + Testing concludes when all cases are run, critical defects are fixed, and the system meets stability and performance requirements, with documentation prepared for future reference.
  1. Defect Severity Levels:
     1. **critical**: System issues or incorrect results.
     2. **High:** Loss of primary functions, workaround available.
     3. **Medium**: Quality affected; workaround available.
     4. **Low**: Minor issues not affecting function.
     5. **Cosmetic:** Minor user interface issues.
  2. Test Reporting
     1. This Daily Summary Reports:

Recipient: Project Manager, Development Team, Quality Assurance (QA) Team

Content: Summary of tests conducted, including the number of tests passed, failed, and any areas of concern.

Frequency: Daily

Purpose: To provide a general overview of the project’s testing status, enabling managers and team members to monitor progress and prioritize problem-solving efforts.

* + 1. Defect Reports:

Recipient: Development Team, QA Team

Content: Detailed information on newly discovered defects, focusing on high- and medium-priority bugs.

Frequency: Generated upon detection of new defects

Purpose: To ensure the development team has sufficient details to apply, prioritize, and resolve defects effectively.

* 1. Feedback and Communication

Testers will document and report issues via an issue-tracking tool, flagging critical and high-severity bugs for prompt action.

The Project Manager will assign defects to developers based on the priority and urgency of the issue.

Testers will re-test resolved defects to verify fixes and update defect statuses, maintaining clear communication with the development team on ongoing issues.

This structured reporting and communication framework enables swift action on high-priority bugs, helping maintain project timelines and ensuring quality standards are met.

1. **Test Schedule**

**This is a six-week timeline:**

* + - 1. **Week 1: Requirement review, scenario and case identification, environment setup.**
      2. **Week 2: Test case creation, traceability matrix setup.**
      3. **Weeks 3-4: Test execution, defect tracking, regression testing.**
      4. **Week 5: Defect fixing and retesting.**
      5. **Week 6: Final test execution, reporting, and stakeholder review.**

1. **Control Procedures**
   1. 6.1 Reviews: Regular assessment of test plan and cases.  
      6.2 Bug Review Meetings: Review and assign priorities to defects.  
      6.3 Change Request: Procedures for test plan amendments.  
      6.4 Defect Reporting: Outline of defect reporting protocols.
2. **Functions To Be Tested:**

**1. int getNumRows(const struct Map\* map);**

**2. int getNumCols(const struct Map\* map);**

**3. void printMap(const struct Map\* map, const int base1, const int**

**alphaCols);**

**4. struct Map addRoute(const struct Map\* map, const struct Route\* route);**

**5. void addPtToRoute(struct Route\* route, struct Point pt);**

**6. void addPointToRouteIfNot(struct Route\* route, const int row, const int**

**col, const struct Point notThis);**

**7. void addPointToRoute(struct Route\* route, const int row, const int col);**

**8. struct Route getBlueRoute();**

**9. struct Route getGreenRoute();**

**10. struct Route getYellowRoute();**

1. **Resources and Responsibilities**  
   8.1. Resources: Requires QA testers, developers, and access to test environments and equipment for simulations.  
   8.2. Responsibilities: QA team oversees test execution and reporting; developers address identified defects; project manager ensures test progress and resource allocation.
2. **Deliverables**
   * + - * Deliverables include daily test reports, defect logs, final test summaries, and the completion report to be submitted to stakeholders for review.
3. **Suspension / Exit Criteria**
   * + - * Testing may pause if critical issues prevent further progress or if essential resources are unavailable. **Exit**: Testing concludes when all tests pass, critical bugs are resolved, and the application meets acceptance criteria.
4. **Resumption Criteria**
   * + - * **Testing will resume once critical blocking issues are addressed, resources are restored, and the test environment is confirmed ready for continuation.**
5. **Dependencies**  
   12.1 Personnel Dependencies: Requires QA staff, developers, and project managers.  
   12.2 Software Dependencies: Depends on specific IDEs, C/C++ compilers, and testing frameworks.  
   12.3 Hardware Dependencies: Needs workstations with stable network access for map simulations.  
   12.3 Test Data & Database: Includes sample delivery locations, route data, and packages for testing various scenarios.
6. **Risks**  
   13.1. Schedule: Delays from resource constraints or unforeseen complexities.  
   13.2. Technical: Issues with memory management or integration within the C environment.  
   13.3. Management: Risks with coordination, especially if there are changes in requirements.  
   13.4. Personnel: Availability and expertise of team members may impact progress.  
   13.5 Requirements: Unclear requirements or scope changes can impact alignment and testing.
7. **Tools**
   * + - * Tools include IDEs like Visual Studio 2022, Eclipse and Code::Blocks.
         * Testing frameworks, e.g., CUnit or Unity.
         * Version control systems like Git or CVN(Subversion.)
8. **Documentation**
   * + - * Documentation includes test plans, test cases, execution reports, and the final summary, ensuring traceability and clarity for future reference.
9. **Approvals**
   * + - * Final approvals will come from the QA lead, project manager, and relevant stakeholders once the application passes all acceptance tests.