**Test Plan Template:**

Car Bot Search

**Prepared by:**

Dehod Vitalii

15.03.2024

**TABLE OF CONTENTS**

1.0 INTRODUCTION

1. OBJECTIVES AND TASKS
   1. Objectives
   2. Tasks

3.0 SCOPE

1. Testing Strategy
   1. Alpha Testing (Unit Testing)
   2. System and Integration Testing
   3. Performance and Stress Testing
   4. User Acceptance Testing
   5. Batch Testing
   6. Automated Regression Testing
   7. Beta Testing

5.0 Hardware Requirements

1. Environment Requirements
   1. Main Frame
   2. Workstation

7.0 Test Schedule

8.0 Control Procedures

9.0 Features to Be Tested

10.0 Features Not to Be Tested

11.0 Resources/Roles & Responsibilities

12.0 Schedules

13.0 Significantly Impacted Departments (SIDs)

14.0 Dependencies

15.0 Risks/Assumptions

16.0 Tools

* 1. Approvals

1. **INTRODUCTION**

This bot helps users find the best deals on cars by:

* **Tracking new car sale ads:** The bot monitors popular websites like Auto.ria for new listings.
* **Matching user criteria:** Users can set filters (make, model, year, etc.) to only receive notifications for relevant cars.
* **Sending notifications:** Users are alerted via Telegram whenever a new matching ad is posted.

This saves users time and effort by eliminating the need to constantly search for cars themselves.

**Additional Notes:**

* The bot does not facilitate the car buying process itself (purchase, financing, etc.).
* This is a basic version, with functionalities like price negotiation assistance being considered for future updates.

1. **OBJECTIVES AND TASKS**

Bot will target several key objectives:

* **Defining Tasks and Responsibilities:** The plan will clearly outline the testing process, including:
  + Specific test cases to be executed at different levels (e.g., unit testing, integration testing).
  + Roles and responsibilities for each testing phase (who tests what).
  + Resources needed for each testing activity (e.g., testing tools, sample data).
* **Communication Vehicle:** The plan will serve as the central document for communication between all stakeholders involved in testing, including developers, testers, and product managers. This ensures everyone is on the same page regarding testing goals, strategies, and timelines.
* **Test Management:** The Master Test Plan will establish the framework for managing the entire testing lifecycle. This includes defining:
  + Entry and exit criteria for each testing phase.
  + Defect tracking and reporting procedures.
  + Risk management strategies to identify and mitigate potential issues.
* **Not a Service Level Agreement (SLA):** While the Master Test Plan establishes expectations for testing quality, it typically wouldn't function as a formal SLA. SLAs often focus on service delivery metrics after launch, whereas the Test Plan is centered on ensuring quality during the development phase.

In summary, the Master Test Plan acts as a roadmap for achieving a high-quality car deal finder bot. It defines tasks, promotes communication, and manages the testing process, without directly dictating post-launch service level agreements.

**Pre-Testing Tasks:**

* **Defining Test Scope:** Determine which functionalities and features will be tested based on the product requirements and specifications.
* **Test Environment Setup:** Set up a testing environment that mirrors the production environment as closely as possible, including data sources (e.g., mock Auto.ria connection) and notification channels (e.g., test Telegram group).
* **Test Data Preparation:** Create or acquire test data (sample car listings) that covers various filter combinations and scenarios.
* **Test Case Design:** Develop detailed test cases that outline specific user actions, expected outcomes, and pass/fail criteria for each functionality.

**Testing Tasks:**

* **Unit Testing:** Test individual bot components (modules) to ensure they function as designed (e.g., data parsing, filter logic, notification generation).
* **Integration Testing:** Verify how different bot components work together (e.g., data retrieval from Auto.ria, filtering based on user criteria, sending notifications to Telegram).
* **Functional Testing:** Test the overall bot functionality from the user's perspective (e.g., user sets filters, receives notifications for matching ads).
* **Non-Functional Testing:** Assess non-functional aspects like performance (speed), reliability (uptime), and security (data protection).

**Post-Testing Tasks:**

* **Defect Reporting and Tracking:** Identify and document any issues encountered during testing (bugs, errors) using a defect tracking system.
* **Test Result Reporting:** Summarize test execution results, including pass/fail rates and encountered problems.
* **Regression Testing:** Re-run critical test cases after bug fixes or code changes to ensure they haven't introduced new issues.
* **Test Data Cleanup:** Remove any temporary test data used during testing.

**Additional Tasks:**

* **Usability Testing:** (Optional) Conduct user testing sessions to evaluate how easy and intuitive the bot is to use for real users.
* **Documentation Updates:** Update the Master Test Plan and other relevant documents to reflect any changes or learnings from the testing process.

**3.0**

## Scope

The Used and New Car Deal Finder Bot test plan will encompass the following functionalities:

* Data Acquisition: Ability to retrieve new car listings from the data source (e.g., Auto.ria).
* Filtering: Functionality to process user-defined filters (make, model, year, etc.) and identify matching ads.
* Notification System: Capability to send notifications to users via Telegram about new matching listings.
* User Interface (Optional): If the bot interacts with a user interface (e.g., basic configuration screen), its functionality will also be tested.
* Integration: The interaction between the bot's components (data retrieval, filtering, notification) will be examined.

## Tactics

**Data Acquisition**

* Test Cases: Develop test cases simulating successful data retrieval (valid website connection) and error handling scenarios (website unavailable, invalid data format).
* Test Data: Mock data resembling real car listings will be used to test parsing and processing capabilities.

**Filtering**

* Test Cases: Design test cases covering various filter combinations (make, model, year range, price range) and ensure accurate ad matching.
* Test Data: Prepare sample car listings with diverse features to test all filtering options.

**Notification System**

* Test Cases: Create test cases to verify notification delivery via Telegram for different scenarios (new matching ad, existing user, notification format).
* Test Environment: Set up a test Telegram group to receive and verify bot notifications.

**User Interface (if applicable)**

* Test Cases: Design test cases to validate user interaction with the interface (setting filters, saving preferences) and ensure proper functionality.
* Test Data: Utilize various user inputs to test interface responsiveness and data handling.

**Integration**

* Test Cases: Develop test cases to verify seamless interaction between bot components. This includes data flow from retrieval to filtering and notification generation.
* Test Environment: Utilize the mock testing environment (data source simulation, test Telegram group) to assess overall integration.

**Communication and Coordination**

* Stakeholders (developers, product managers) will be informed about testing activities and timelines.
* Collaboration with developers will be established for setting up the test environment and providing test data.
* A defect tracking system will be used to report and manage any issues identified during testing.

**4.0 TESTING STRATEGY**

**1. Unit Testing:**

* **Approach:** This focuses on testing individual modules of the bot in isolation (e.g., data parsing module, filter logic module, notification generation module).
* **Ensuring Functionality:** Unit testing uses mock data and predefined scenarios to verify each module performs its designated task correctly.

**2. Integration Testing:**

* **Approach:** This assesses how different bot components work together when data flows between them.
* **Ensuring Functionality:** Integration tests simulate user actions and data flow, verifying components like data retrieval, filtering, and notification generation work seamlessly.

**3. Functional Testing:**

* **Approach:** This evaluates the bot's functionality from the user's perspective.
* **Ensuring Functionality:** Users (or testers mimicking user behavior) interact with the bot to set filters, trigger notifications, and confirm they receive alerts for matching car listings based on their criteria.

**4. Non-Functional Testing:**

* **Approach:** This goes beyond core functionality and evaluates aspects like performance (speed), reliability (uptime under load), and security (data protection).
* **Ensuring Functionality:** Specific tools and techniques are used to measure response times, simulate high user volume, and probe for vulnerabilities to ensure the bot performs efficiently and securely.

**5. Usability Testing (Optional):**

* **Approach:** This involves real users interacting with the bot to assess its ease of use and intuitiveness.
* **Ensuring Functionality:** Observations and user feedback are collected to identify any usability issues and refine the user interface or interaction flow for a smooth user experience.

1. **Unit Testing**

**Definition and Comprehensiveness**

* **Desired Comprehensiveness:** We aim for high unit test coverage, striving to test all critical code paths and functionalities within each bot module.
* **Techniques for Measuring Comprehensiveness:**
  + **Statement Coverage:** We will target a high percentage (e.g., 80%+) of code statements being executed at least once during unit tests.
  + **Branch Coverage:** We will aim for coverage of a significant portion (e.g., 70%+) of conditional branches (if/else statements) to ensure various code execution paths are tested.
* **Additional Completion Criteria:**
  + Low Error Frequency: We aim for a low number of errors identified during unit testing, indicating a solid foundation for further testing stages.

**Tracing Requirements**

* Unit tests will be traced back to specific functional requirements documented for the bot's functionalities (data parsing, filtering logic, notification generation). This ensures each requirement has corresponding unit tests to verify its implementation.

**Participants**

* **Primary Responsibility:** Developers will have the primary responsibility for writing and executing unit tests for the code they develop.
* **Secondary Responsibility:** Testers may collaborate with developers to ensure test cases effectively cover all functionalities and edge cases.

**Methodology**

* **Test Script Development:** Developers will write unit test scripts using a suitable unit testing framework (e.g., Python unittest, JUnit).
* **Test Execution Sequence:**
  + **Individual Module Testing:** Unit tests will be executed for each isolated bot module (data parsing, filtering, notification).
  + **Integration-oriented Unit Tests:** After individual module testing, some unit tests can focus on interactions between closely related modules to identify potential integration issues early on.
* **Testing Activities:**
  + Developers will run unit tests frequently during the development process to catch regressions and ensure code changes don't introduce new issues.
  + A testing tool or framework will be used to automate unit test execution and reporting.
  + Code coverage reports will be generated to identify any untested sections of code and guide further test development.

1. **System and Integration Testing**

**Definition**

* **System Testing:** This phase will assess the overall functionality of the bot as a complete system, simulating real-world user interactions.
* **Integration Testing:** This focuses on verifying how individual bot components interact and exchange data with each other to achieve the desired outcome (delivering car deal notifications).

**Participants**

* **Primary Responsibility:** Testers will take the lead in designing and executing system and integration tests.
* **Secondary Responsibility:** Developers may be involved in providing technical expertise and troubleshooting any integration issues identified during testing.

**1. System Testing**

* **Test Scripts:** Testers will develop test cases simulating user actions and scenarios (setting filters, triggering notifications for matching ads).
* **Sequence of Events:**
  1. **Test Environment Setup:** A testing environment replicating the production setup (data source simulation, test Telegram group) will be established.
  2. **Test Case Execution:** Testers will interact with the bot as a user, following the test scripts to set filters, trigger notifications, and verify the system delivers alerts for relevant car listings.
  3. **Defect Reporting:** Any discrepancies between expected and actual behavior will be documented as defects for further investigation and rectification.
* **Testing Activities:** System testing will be conducted after successful completion of unit testing. It may be performed iteratively throughout the development process.

**2. Integration Testing**

* **Test Scripts:** Testers will design test cases focusing on data flow and interaction between bot modules (data retrieval, filtering, notification generation).
* **Sequence of Events:**
  1. **Mocking External Dependencies:** For integration testing, external dependencies (like the real Auto.ria website) may be mocked or simulated to isolate and test internal bot interactions.
  2. **Test Case Execution:** Testers will execute test scripts that simulate data flow between modules, verifying data is processed correctly through each stage.
  3. **Error Identification:** Any errors in data exchange or processing between modules will be identified and reported for developers to fix.
* **Testing Activities:** Integration testing will be conducted after successful unit testing of individual modules. It can be performed iteratively as new functionalities are integrated.

1. **Performance and Stress Testing**

**Definition**

Stress testing for the bot will involve pushing its functionalities beyond normal operating conditions to assess its stability and performance under heavy load. This helps identify potential weaknesses before real-world usage.

**Participants**

* **Primary Responsibility:** Testers will take the lead in designing and executing stress tests.
* **Secondary Responsibility:** Developers may be involved in analyzing test results, identifying performance bottlenecks, and optimizing the code for better scalability.

**Methodology**

* **Test Scripts:** Testers will develop scripts simulating a high volume of user activity (concurrent filter requests, notification triggers).
* **Sequence of Events:**
  1. **Test Environment Setup:** The testing environment will be configured to handle increased load (e.g., increased virtual users, simulated data source).
  2. **Stress Test Execution:** The test scripts will be executed, gradually increasing the load on the bot to simulate heavy user traffic.
  3. **Performance Monitoring:** System performance metrics (response times, resource utilization) will be monitored during the test.
  4. **Identifying Breaking Points:** The test will continue until the bot exhibits performance degradation or crashes, indicating its stress threshold.
* **Testing Activities:** Stress testing may be conducted after core functionalities are validated through system and integration testing. It can be repeated throughout development to assess improvements after code optimization.

**Additional Considerations**

* **Scalability Testing:** As part of stress testing, scalability testing can be conducted to evaluate how the bot scales horizontally (adding more servers) to handle increased user load.

1. **User Acceptance Testing**

**Definition**

User Acceptance Testing (UAT) aims to verify if the car deal finder bot meets the needs and expectations of its end-users (potential car buyers). This involves real users interacting with the bot and comparing its functionality to the initial requirements.

**Participants**

* **Primary Responsibility:** A representative group of target users (potential car buyers) will be recruited to participate in UAT.
* **Secondary Responsibility:** Testers will facilitate the testing process, guide users through test scenarios, and document their feedback.
* **Developers:** Developers may be involved in observing UAT sessions to gain valuable user insights and address any critical usability issues identified.

**Methodology**

* **Test Script Development:** Testers will develop UAT test cases based on user stories and acceptance criteria defined in the initial project requirements. These scenarios will cover core functionalities (setting filters, receiving notifications) and user experience aspects (ease of use, intuitiveness).
* **Sequence of Events:**
  1. **User Recruitment:** A diverse group of potential car buyers will be recruited to participate in UAT.
  2. **UAT Training:** Testers will provide a training session for users, familiarizing them with the bot's functionalities and UAT testing procedures.
  3. **Test Case Execution:** Users will interact with the bot following the UAT test scripts, providing feedback on its effectiveness in meeting their needs.
  4. **Feedback Collection:** Testers will document user feedback (positive experiences, usability issues, suggestions for improvement) throughout the UAT process.
* **Testing Activities:** UAT will be conducted after successful completion of system and integration testing. It ensures the bot is user-friendly and delivers value to its target audience.

**Additional Considerations**

* **Usability Testing Tools:** Screen recording tools can be used to capture user interactions with the bot, providing valuable insights into user behavior and potential usability issues.
* **Iterative Process:** UAT findings may lead to further refinements of the bot's functionalities or user interface to enhance user experience.

1. **Automated Regression Testing**

**Definition**

Regression testing is crucial for our car deal finder bot. It involves re-running a set of automated tests after code modifications or bug fixes to ensure these changes haven't introduced new issues and the core functionalities remain intact.

**Participants**

* **Test Automation Engineer/Developer:** This role (or a team) will be responsible for developing, maintaining, and executing the automated regression test suite.
* **Testers:** Testers may collaborate with the test automation specialist to define the scope of automated regression testing and ensure the test suite covers critical functionalities.

**Methodology**

1. **Test Suite Development:**
   * The test automation engineer will develop a suite of automated tests covering core functionalities (data acquisition, filtering, notification delivery). These tests will be designed to be repeatable and executable through a testing framework (e.g., Selenium, Cypress).
   * The test suite should be designed to be modular and allow for easy maintenance as the bot evolves.
2. **Test Execution Integration:**
   * The automated regression tests will be integrated into the development process (e.g., continuous integration/continuous delivery pipeline). This allows for automatic test execution after code changes, providing rapid feedback on potential regressions.
   * Testers can also manually execute the regression test suite periodically or after specific bug fixes to ensure overall functionality.
3. **Test Result Analysis:**
   * The testing framework will generate reports summarizing the test execution results (passed/failed tests).
   * Testers will analyze these reports to identify any failing tests that might indicate regressions caused by code changes.
   * Failed tests will require investigation by developers to diagnose the root cause and fix any regressions before deployment.

**Benefits of Automated Regression Testing**

* **Reduced Testing Time:** Automating repetitive tests saves time and resources compared to manual regression testing.
* **Improved Efficiency:** Automated tests can be executed more frequently, enabling faster detection and resolution of regressions.
* **Increased Test Coverage:** A wider range of test cases can be included in the automated suite, leading to more comprehensive regression testing.
* **Improved Consistency:** Automated tests ensure consistent execution and reduce the risk of human error during manual testing.

**4.7 Beta Testing**

**Participants**

* **Target Users:** A group of representative users (potential car buyers) will be recruited to participate in the beta test. This should be a diverse group reflecting the target audience for the bot.
* **Testers:** The testing team will be responsible for managing the beta testing process, including user recruitment, communication, and feedback collection.
* **Developers:** Developers may be involved in addressing critical bugs or usability issues identified during beta testing to improve the bot before public release.

**Methodology**

1. **Beta Test Planning:**
   * Define the goals of beta testing (identify bugs, gather user feedback, refine functionalities).
   * Determine the size and demographics of the target user group for the beta test.
   * Develop a communication plan to keep beta testers informed about the testing process and how to provide feedback.
2. **Beta Tester Recruitment:**
   * Utilize various methods to recruit potential car buyers for the beta test (social media ads, online forums, website signup).
   * Provide clear instructions and expectations for beta testers, including the testing period and feedback channels.
3. **Beta Testing Process:**
   * Grant beta testers access to the bot through a dedicated channel (beta version of the app/website).
   * Encourage testers to explore the bot's functionalities, set filters for car searches, and interact with the notification system.
   * Facilitate communication channels (forum, surveys) for beta testers to report bugs, provide feedback on usability, and suggest improvements.
   * The testing team will collect and analyze user feedback to identify recurring issues and areas for improvement.
4. **Post-Beta Testing:**
   * Based on beta test results, developers will prioritize and address critical bugs and usability issues before public release.
   * The testing team will prepare a report summarizing the beta testing findings and recommendations for improvement.
   * Beta testers may be offered gratitude for their participation and potential early access to the publicly released bot.

**Additional Considerations**

* **Confidentiality Agreements:** Beta testers may be required to sign confidentiality agreements to protect unreleased features of the bot.
* **Beta Testing Duration:** The beta testing period should be long enough to gather sufficient user feedback but not so long that user interest wanes.
* **Iterative Process:** Insights from beta testing may inform further refinement of the bot's functionalities before and even after its public release.

**5.0 HARDWARE REQUIREMENTS**

1. **Development Environment:** The specifications needed for the developers to build and test the bot.
2. **Deployment Environment:** The hardware needed to run the bot in production for real-world usage.

Here's a breakdown of potential hardware needs for each environment:

**Development Environment:**

* **Computers:**
  + Developers will likely need personal computers with standard specifications for software development. This could include:
    - Operating System: Windows 10/11, macOS, or Linux (depending on developer preference and development tools)
    - Processor: Mid-range processor (e.g., Intel Core i5 or equivalent)
    - Memory (RAM): 8GB or more
    - Storage: Sufficient storage space for development tools, codebase, and test data.
* **Modems:**
  + A standard internet connection modem is generally sufficient for developers to access necessary resources (code repositories, testing platforms) and collaborate online.

**Deployment Environment:**

* **Computers:**
  + The bot itself might run on a server-grade computer depending on the chosen deployment method. Here are two options:
    - **Cloud Hosting:** Popular cloud platforms (AWS, Google Cloud Platform, Microsoft Azure) offer various virtual server configurations. The specific requirements will depend on the bot's expected traffic and processing needs. You can start with a lower-tier configuration and scale up as needed.
    - **On-Premise Server:** If you choose to host the bot on your own server hardware, the specifications will depend on the bot's complexity and user traffic. You'll need a server with a reliable processor, sufficient RAM, and storage space to handle data processing, user requests, and communication with Telegram.
* **Modems:**
  + A reliable internet connection with a business-grade modem might be required for the server, especially if you anticipate high user traffic or real-time data updates.

**Additional Considerations:**

* **Databases:** Depending on the bot's design, it might require a separate database server to store user data, filter preferences, and potentially car listing information (if temporarily cached). The database server's hardware requirements will depend on the chosen database solution and data volume.
* **Scalability:** It's wise to choose hardware that can be easily scaled up if the bot experiences a surge in users or requires additional processing power in the future. Cloud platforms offer easy scaling options, while on-premise servers might require hardware upgrades.

1. **ENVIRONMENT REQUIREMENTS**

**Hardware:**

* **Development Environment:** As mentioned earlier, developers will likely need personal computers with standard development specifications.
* **Deployment Environment:** There are two main options:
  + **Cloud-Based Testing:** Utilize a cloud platform (AWS, Google Cloud Platform, Microsoft Azure) to set up a virtual server environment mirroring the production setup (operating system, database).
  + **On-Premise Testing Server:** If using an on-premise server for deployment, a dedicated server with sufficient processing power, RAM, and storage should be allocated for testing purposes. This server should mimic the production server's specifications.

**Communications and System Software:**

* **Operating System:** The chosen operating system should be compatible with the bot's development environment and deployment plans (e.g., Windows, Linux, macOS for development; Linux for cloud-based deployment).
* **Database Software:** If the bot utilizes a database (user data, filter preferences, car listings), a compatible database management system (DBMS) needs to be installed in the test environment (e.g., MySQL, PostgreSQL).
* **Development Tools:** Developers will require their usual development tools and libraries for building and testing the bot (e.g., version control system, programming language IDE, testing frameworks).
* **Telegram Integration:** A test Telegram group or account will be needed to simulate notifications sent by the bot during testing.

**Mode of Usage:**

* **Stand-alone (Isolated):** The test environment should be isolated from the production environment to prevent any interference or data contamination between testing and real-world usage.

**Security:**

* The test environment should implement appropriate security measures to protect the bot's code, data (user information, potentially car listing data), and communication channels (Telegram integration). This may involve:
  + User access controls to restrict access to the testing environment and bot code.
  + Data encryption for sensitive information (if applicable).
  + Secure communication protocols for data exchange between the bot and external services (Telegram API).

**Special Test Tools:**

* **Unit Testing Framework:** A suitable unit testing framework (e.g., Python unittest, JUnit) for developers to write and execute unit tests for individual bot modules.
* **Test Automation Framework:** A framework (e.g., Selenium, Cypress) to automate repetitive test cases, particularly during regression testing.
* **Performance Monitoring Tools:** Tools to monitor system performance metrics (response times, resource utilization) during stress testing.

**Other Testing Needs:**

* **Office Space (Optional):** Depending on your team structure, dedicated office space for testers and developers collaborating on testing activities might be beneficial.
* **Documentation:** Access to relevant project documentation (functional specifications, user stories) will be crucial for testers to design and execute effective test cases.

**Resource Acquisition:**

1. Most hardware and software resources can likely be obtained from existing IT infrastructure or through cloud platform subscriptions.
2. Any additional resources not currently available (specific testing tools) can be procured through dedicated software licenses or open-source options.

**7.0 TEST SCHEDULE**

The overall testing duration is estimated to be around 10.5 weeks, with the following breakdown:

* Weeks 1-2: Pre-Testing & Unit Testing
* Weeks 3-4.5: Integration Testing
* Weeks 5-6.5: System Testing
* Week 7: Stress Testing
* Week 8: User Acceptance Testing
* Throughout Development: Regression Testing

**8.0 CONTROL PROCEDURES**

**. Incident Identification:**

* Testers or developers encountering an issue during testing will document the incident details as soon as possible.

**2. Incident Documentation:**

* A standardized bug tracking system (e.g., Jira, Bugzilla) will be used to log and track all incidents.
* The following information will be captured for each incident:
  + Summary: A brief description of the problem encountered.
  + Steps to Reproduce: Detailed steps to consistently reproduce the issue.
  + Expected Behavior: What the bot should be doing according to the requirements.
  + Actual Behavior: The observed behavior that deviates from expectations.
  + Severity:\*\* Classification of the issue's severity (critical, major, minor).
  + Priority:\*\* Prioritization of the issue based on urgency and impact (high, medium, low).
  + Screenshots/Recordings (optional): Visual evidence of the issue if applicable.
  + Assignee: Assigned developer or team responsible for investigating and fixing the issue.

**3. Communication and Resolution:**

* The documented incident will be submitted to the bug tracking system.
* Testers will communicate the issue to developers, providing clear explanations and any relevant attachments.
* Developers will investigate the reported incident, diagnose the cause, and develop a fix.
* Developers will keep testers informed of the progress and estimated resolution timeframe.
* Upon fixing the issue, developers will provide a resolution summary and regression test the affected area to ensure the fix is effective.
* Testers will verify the fix by re-running the test case that exposed the issue.
* If the issue is verified as fixed, the incident will be marked as resolved in the bug tracking system.

**Change Requests**

**1. Change Request Initiation:**

* Changes to the bot's functionalities or requirements can be initiated by various stakeholders:
  + Testers identifying usability issues or functionality gaps during testing.
  + Developers proposing improvements or optimizations based on technical considerations.
  + Project stakeholders suggesting new features or modifications based on business needs.

**2. Change Request Documentation:**

* All proposed changes will be documented in a change request form within the project management tool.
* The form should capture details like:
  + Description of the proposed change.
  + Rationale for the change (justification for its implementation).
  + Impact assessment (potential effects on existing functionalities, schedule, resources).
  + Estimated effort required for implementing the change.

**3. Change Request Approval:**

* A designated Change Control Board (CCB) will review all submitted change requests.
* The CCB will consist of representatives from development, testing, and project management.
* The CCB will evaluate each request based on:
  + Alignment with project requirements and objectives.
  + Feasibility and technical complexity of implementation.
  + Impact on project schedule and resource allocation.
  + Potential benefits and risks associated with the change.
* The CCB will approve or reject the change request based on the evaluation.

**4. Change Implementation:**

* Approved change requests will be incorporated into the project plan.
* The development team will implement the approved changes, following established development and testing procedures.
* Regression testing will be conducted to ensure existing functionalities remain intact after the change implementation.

**5. Change Management:**

* A log of all change requests will be maintained, documenting the submitted request, CCB decision, and implementation status.
* This log will provide a clear record of changes made throughout the project lifecycle.

**9.0 FEATURES TO BE TESTED**

**1.Core Functionalities:**

* **Feature:** User Input for Search Criteria (Car Make, Model, Year Range, Price Range)
  + **Testing Combinations:**
    - Valid inputs within specified ranges (e.g., existing car makes, model years within range).
    - Invalid inputs (e.g., non-existent car makes, unrealistic price ranges).
    - Empty or missing input fields.
* **Feature:** Data Retrieval from External Source (Auto.ria Website)
  + **Testing Combinations:**
    - Simulate successful data retrieval for matching car listings.
    - Simulate data retrieval errors (website unavailable, API issues).
    - Test handling of empty or limited search results.
* **Feature:** Filtering Search Results based on User Criteria
  + **Testing Combinations:**
    - Test filtering by individual criteria (make, model, year, price).
    - Test combinations of multiple filtering criteria.
    - Test handling of edge cases (e.g., filtering for a very specific car model).
* **Feature:** Notification Delivery via Telegram Integration
  + **Testing Combinations:**
    - Test successful notification delivery for matching car listings.
    - Test notification delivery failure due to Telegram API issues.
    - Test notification content and formatting (includes relevant car details).

**2. Additional Functionalities (if applicable):**

* **Feature:** User Account Management (Optional)
  + **Testing Combinations:**
    - User registration, login, and profile management.
    - Password reset functionality.
    - User data privacy and security.
* **Feature:** Saved Searches and Alerts (Optional)
  + **Testing Combinations:**
    - Saving user search criteria for future reference.
    - Setting up alerts for new listings matching saved criteria.
    - Managing and deleting saved searches and alerts.

**3. Combinations of Features:**

* Test scenarios that involve combinations of core functionalities to simulate real-world usage.
  + Example: User sets search filters, bot retrieves data, filters results, and sends notifications for relevant listings.

**10.0 FEATURES NOT TO BE TESTED**

**1. Excluded Features:**

* **Advanced User Management (if not planned):** If the bot doesn't have functionalities requiring user accounts (saved searches, purchase history), user management features (registration, login, profile management) won't be a testing focus.
* **Payment Processing (if not applicable):** If the bot doesn't directly facilitate car purchases, payment processing features wouldn't be included in testing.

**2. Non-Significant Feature Combinations:**

* **Redundant or Trivial Combinations:** Testing every possible combination of filtering criteria might be unnecessary. We can focus on core combinations (filtering by make, model, year, price) and edge cases (highly specific filters).
* **Unrealistic or Destructive Combinations:** There's no need to test combinations that wouldn't occur in real use (e.g., setting a price range exceeding market value).

**3. Reasons for Exclusion:**

* **Project Scope and Time Constraints:** Focusing on core functionalities and critical combinations ensures efficient use of testing resources within the project timeframe.
* **Limited Development Resources:** Depending on team size and expertise, testing highly specialized features or exhaustive combinations might not be feasible.
* **Focus on User-Facing Functionalities:** The primary concern is ensuring the bot interacts effectively with users, so internal system functionalities not directly impacting user experience might receive less rigorous testing.

**4. Alternative Testing Approaches:**

* **Code Reviews:** Code reviews by developers can identify potential issues early in the development process, potentially reducing the need for extensive testing of specific features or combinations.
* **Static Code Analysis Tools:** These tools can automatically identify potential coding errors and vulnerabilities, reducing the workload for manual testing.

**5. Importance of Re-Evaluation:**

* As the project progresses, new features or functionalities might be added. It's crucial to re-evaluate the testing scope to ensure all relevant aspects are covered before deployment.
* Unexpected issues encountered during testing might necessitate expanding the scope to include additional features or combinations for a more comprehensive evaluation.

**11.0 RESOURCES/ROLES & RESPONSIBILITIES**

**Testing Team:**

* **Test Lead:**
  + Responsible for overall testing strategy, planning, and execution.
  + Manages the testing team, assigns tasks, and tracks progress.
  + Oversees the creation and maintenance of the test plan and test cases.
  + Reports testing results and identifies areas for improvement.
* **QA:**
  + Design, develop, and execute test cases for all testing phases (unit, integration, system, regression, user acceptance).
  + Identify and document defects encountered during testing.
  + Participate in user acceptance testing sessions with potential users.
  + Collaborate with developers to troubleshoot and resolve identified issues.

**Development Team:**

* **Developers:**
  + Collaborate with testers during pre-testing to ensure a suitable testing environment is established.
  + Write unit tests for their own code modules.
  + Work with testers to troubleshoot and fix defects identified during testing.
  + May be involved in user acceptance testing to observe user interaction and address usability issues.

**14.0 DEPENDENCIES**

**1. Test-Item Availability:**

* **External Data Source:** The bot relies on data retrieval from the Auto.ria website.
  + Limitations on the website's API or data availability could restrict testing scenarios (e.g., limited car listings for specific criteria).
  + We might need to develop workarounds (simulated data) to ensure comprehensive testing of filtering and notification functionalities.
* **Target Users (UAT):** Recruiting a sufficient number of representative users for User Acceptance Testing (UAT) could be a challenge. We might need to leverage alternative methods (usability testing tools) to gather user feedback if finding enough participants proves difficult.

**2. Testing-Resource Availability:**

* **Testers:** The number of testers available can limit the scope and depth of testing. Prioritization of critical functionalities and risk-based testing might be necessary.
* **Automation Engineer (Optional):** If an automation engineer isn't available, the extent of automated regression testing might be limited. Manual regression testing would require more resources and time.

**3. Deadlines:**

* **Project Time Constraints:** Tight deadlines can pressure testers to rush through test execution, potentially compromising test coverage and thoroughness.
  + Clear communication with stakeholders and managing expectations regarding achievable testing goals within the timeframe is crucial.

**15.0 RISKS/ASSUMPTIONS**

**Assumption 1: Reliable External Data Source**

* **High Risk:** The Auto.ria website's API or data availability might be limited or experience outages during testing. This could restrict testing scenarios, especially regarding filtering and notification functionalities that rely on retrieved car listings.
* **Contingency Plan:**
  + Develop a mock data generation tool to simulate car listings with various criteria. This allows testing filtering and notification logic without relying solely on external data.
  + Schedule testing sessions during known website maintenance windows to minimize disruptions.
  + Implement monitoring tools to track the external API's availability and performance. If outages occur, adjust testing priorities to focus on functionalities not reliant on real-time data retrieval.

**Assumption 2: Sufficient Target Users for UAT**

* **High Risk:** Recruiting a sufficient number of representative users for User Acceptance Testing (UAT) might be challenging. This could limit the scope of user feedback and potential identification of usability issues.
* **Contingency Plan:**
  + Utilize online recruitment platforms and user testing communities to find potential participants.
  + Offer incentives (e.g., gift cards) to encourage participation in UAT sessions.
  + Consider alternative user feedback methods:
    - Conduct remote usability testing sessions with a smaller group of users using online testing tools.
    - Gather feedback through surveys distributed to potential users after showcasing the bot's functionalities.

**Assumption 3: Availability of Testers**

* **High Risk:** Limited tester availability could restrict the comprehensiveness of testing, potentially leading to missed defects or insufficient test coverage.
* **Contingency Plan:**
  + Prioritize critical functionalities and high-risk areas for thorough testing.
  + Explore options for expanding the testing team:
    - Train developers on basic testing principles to assist with test execution.
    - Consider outsourcing specific testing tasks (e.g., smoke testing) to external testing services if budget allows.
  + Leverage test automation to free up testers' time for more complex, exploratory testing.

**Assumption 4: Meeting Tight Deadlines**

* **High Risk:** Rushing through testing due to tight project deadlines could lead to inadequate test coverage and potential bugs escaping detection.
* **Contingency Plan:**
  + Manage stakeholder expectations by clearly communicating potential risks associated with compressed testing timelines.
  + Focus on risk-based testing, prioritizing functionalities with the highest user impact or potential for critical issues.
  + Explore options for optimizing the testing process:
    - Utilize parallel testing strategies where possible to run multiple tests concurrently.
    - Implement continuous integration and continuous delivery (CI/CD) pipelines to automate repetitive tasks and expedite feedback loops.

**16.0 TOOLS**

**Automation Tools:**

* **Unit Testing:**
  + Python: unittest framework is a popular option for writing unit tests in Python.
  + Java: JUnit is a widely used framework for unit testing in Java.
* **Integration Testing:** These tools can automate interactions between different bot components during integration testing.
  + Selenium: A popular open-source framework for automating web browser interactions. It can be used to simulate user actions within the Telegram interface for testing bot functionalities.
  + Cypress.io: Another popular option for web UI automation testing, offering a good user experience for writing and running tests.
* **Regression Testing:** These tools can automate repetitive test cases, particularly during regression testing after code changes.
  + Selenium can also be used to automate regression testing of the bot's functionalities. Other testing frameworks like Katalon Studio or frameworks designed specifically for Telegram bot testing could also be explored depending on the chosen development environment and programming languages.

**Bug Tracking Tool:**

* **Jira:** A popular project management and bug tracking tool that offers features for issue tracking, assignment, prioritization, and reporting. It integrates well with development workflows and facilitates communication between testers and developers.
* **Bugzilla:** Another open-source option for bug tracking, offering similar functionalities to Jira for managing and tracking identified issues throughout the testing process.

**Choosing the Right Tools:**

The specific tools selected will depend on several factors, including:

* Programming languages used for bot development
* Development and testing team preferences
* Project budget (open-source vs. commercial tools)
* Integration with existing development workflows

**17.0 APPROVALS**

| # | Name | Role | Date | Signature |
| --- | --- | --- | --- | --- |
| 1 | Vitaliy Dorosh | Sponsor |  |  |
| 2 | Tetiana Sozanska | Team Lead |  |  |