Test Plan Report

for

EasySearch

Prepared by:

Dinçer İnce

Halil Mert Güler

Süleyman Samed Çalışkan

14.05.2023

Table of Contents

Introduction

Purpose of the Test Plan Report

The main goals of this report are as follows:

1. The report helps in communicating the testing approach and strategy to all involved in the project. Ensuring that everyone understands the objectives of the testing effort.

2. The report serves as a reference for the team, providing them with instructions and guidelines on how to perform the tests, from the types of tests to methodologies to be used for effective testing for future reference.

3. The report helps identifying the potential risks and issues related to the testing process. It helps minimizing the impact of risks and ensures that testing activities proceed smoothly.

Overview of the Project

EasySearch is a project that aims to help users store, manage, analyse, and search through various documents all at once in a neat package. The project has two main systems which are the web application and the API. Through which the API can be used to extend the functionality of an already existing system, and the web application can be used as a standalone solution for document management needs.

The project is aimed at two different audiences one of which is a general consumer who uses the project as a cloud document management system. Where the user can perform various search techniques to find and reach the data it stored effectively, the user can use various search techniques in order to find the exact or data similar to the query itself. The user only needs to use the web application in order to manage, view, edit, and add new documents, and the documents can be reached from anywhere.

The other audience is the developers who need natural language processing functionality in a simple and effective manner. Natural language processing techniques and procedures are tedious and complex to implement and integrate. EasySearch offers an API solution in order to provide natural language processing activities for developers to integrate to their already existing system. The main usecase for this audience is the various search tecniques, in order to help their users find the document they are looking for and recommend similar documents to enhance their experience.

Test Objectives

High-Level Goals of the Testing Process

1. Ensuring quality is the main goal of this testing process. The system is tested thoroughly in order to veify that the project's specifications are being met. The project functions as expected, in a manner that satisies the requirements, and is free from errors.

2. Detecting and fixing the errors within the project, identifying the root causes, and changing the implementation where necessary to stop the error from occuring, with it the reliability of the project can be dramatically better. By identifying these problems in the testing process, the problems can be easier to sort and fix.

3. Performance monitoring of the various critical and high-frequency use components of the project, in order to figure out if there is a need for optimization and if so, finding more optimized solutions for those components.

4. Identiying and mitigating the points where the security can be compromised, and finding out if there is any vulnerability possible under various scenarios. By identifying the possible spots where the security may become vulnerable ensures better reliability in future maintenance.

Specific Objectives of Each Testing Phase

1. Unit Testing

   In this project unit tests are performed in order to verify the functionality of specific components, by themselves. Tests are done in order to validate whether the component by itself works as expected, without any outside interference.

2. Integration Testing

   This phase is used to test how components integrate and work with each other. The components that interact with eachother are tested as a whole to see if they function as they do in the unit tests, and spot discrepencies if exists.

3. System Testing

   In this phase the whole system is tested as a complete package. The components that integrate with each other interact with each other and the system testing is done to see how the complete application behaves and spot the discrepencies that arrive due to the interactions of integrated components. As a result it is possible to determine whether the system meets the requirements as specified.

Test Approach

Testing Methodologies and Techniques

1. Unit Testing

   This focuses on testing individual units or components of the API in isolation, such as functions, methods, or classes. Unit tests help ensure that each unit works correctly and identify any bugs or issues at an early stage.

2. Integration Testing

   This type of testing verifies the interaction between different components or modules of the API. Integration tests check if the API functions correctly when its components are combined. In the case of Easy Search, you may want to test how it integrates with other APIs or systems.

3. System Testing

   This type of testing verifies the system as a whole to see if it functions as intended. In EasySearch's case this verifies whether the various API calls manipulates the data in a reliable manner, whether the authentication with JWT from the web application functions correctly as well as the API key authentication for different requests.

Test Levels

1. Unit Testing

   Unit testing will be an integral part of the Easy Search API project. It involves testing individual functions or methods within the API's codebase. By breaking down the API into smaller units, we can ensure the correctness of the logic and behavior of each component.

   During unit testing, various test cases will be designed to cover different scenarios, including edge cases and boundary conditions. This ensures that the units handle a wide range of inputs and produce the expected outputs. Unit tests will be written using a suitable testing framework like XUnit to automate the testing process and facilitate future maintenance.

   The goal of unit testing is to catch bugs or issues at an early stage, allowing developers to fix them before they propagate to other parts of the system. It also helps improve code quality, maintainability, and readability. By thoroughly testing individual units, we can build confidence in the reliability and stability of the API's core functionality.

   By conducting performance testing, we can ensure that the Easy Search API can handle the expected user load without degradation in performance. It helps us optimize

the API's performance, enhance its scalability, and deliver a responsive and efficient search experience.

2. Integration Testing

Integration testing will be performed to validate the interaction between different components or modules of the Easy Search API. This testing level ensures that the API functions correctly when its components are combined.

Integration tests will focus on testing the API's integration with external dependencies such as databases, caching systems, or third-party services. By simulating real-world scenarios, we can verify that data flows seamlessly between the API and its dependencies. These tests will cover different integration points and validate the communication protocols and data formats used.

By conducting thorough integration testing, we can identify any compatibility issues or miscommunications between components. This ensures that the API operates as expected when interacting with external systems, providing a smooth and reliable experience for end-users.

3. System Testing

System testing will be performed to validate the behavior of the system as a whole, and see how the integrated components interact with each other. When every part of the system is integrated, it is common to see various problems that weren't detected in the integration testing.

The problems that can be spotted by the system testing, if the other testing phases are performed correctly, are logical problems. These problems can be very hard to solve as the root of the problem already integrated with various parts of the system.

By conducting system testing, it allows us to spot these problems and fix them while still in active development, so it is crucial for a project's success and one of the determining factors of reliability and maintainability of the project in general.

Test Data and Environment

- Test Data: Test data is a crucial component of the testing process. It is used to validate the functionality, performance, and security of the Easy Search API. The test data should cover a wide range of scenarios, including both typical and edge cases, to ensure comprehensive testing coverage.
  Here are some considerations for test data:

- Typical Data: Test data should include representative examples of typical search queries, data sets, and expected search results. This helps verify that the API functions correctly under normal usage scenarios.

- Edge Cases: It's important to include test data that represents extreme or uncommon scenarios. This can involve testing the API's behavior with large data sets, complex search queries, or unusual input values. Edge cases help uncover potential issues or limitations in the API.

- Invalid Inputs: Test data should include cases where invalid or malformed inputs are provided. This helps ensure that the API handles such inputs gracefully, providing appropriate error responses and maintaining data integrity.

- Performance Data: In performance testing, realistic data sets should be used to simulate expected user loads accurately. This includes generating a sufficient amount of test data to evaluate the API's performance under different scenarios and data volumes.

It's important to carefully select and generate test data to cover a wide range of scenarios, allowing for comprehensive testing of the Easy Search API's functionality, performance, and security.

- Testing Environment: The testing environment for the Easy Search API project refers to the setup and configuration of the infrastructure, tools, and software required to conduct testing.
  Here are some considerations for the testing environment:

  - Development and Staging Environments: It is recommended to have separate environments for development and testing. The development environment allows developers to work on new features and conduct unit testing. The staging environment closely resembles the production environment and is used for integration, functional, and performance testing.

  - Infrastructure Setup: The testing environment should mirror the production environment as closely as possible. This includes configuring the necessary hardware, networking, and software components. It may involve setting up servers, databases, caching systems, and other dependencies that the Easy Search API relies on.

  - Test Environment Isolation: To ensure accurate testing results, it's important to isolate the testing environment from the production environment. This prevents any interference or impact on the live system during testing activities.

  - Test Data Management: Test data should be properly managed within the testing environment. This involves creating mechanisms to generate, load, and refresh test data as needed. It may include tools or scripts to automate data generation or restore databases to a known state for each test run.

- Test Environment Configuration: The testing environment should be configured to mimic real-world scenarios. This includes replicating network conditions, load balancers, security configurations, and other relevant aspects that the API will encounter in the production environment.

- Test Environment Monitoring: Monitoring tools can be utilized to capture and analyze metrics during testing. This includes monitoring resource utilization, response times, error rates, and other performance indicators. Monitoring helps identify bottlenecks, performance issues, or anomalies during testing.

Test Strategy

Test Coverage

- Backend Components: The backend components of the Easy Search API project encompass the core functionality, data handling, and business logic.
  The test coverage for the backend components should include:

    - API Endpoints: All API endpoints should be thoroughly tested to ensure that they handle requests and responses correctly. This includes testing various HTTP methods, query parameters, and request payloads.

    - Authentication and Authorization: The components responsible for user authentication and authorization should be extensively tested. This involves verifying that users can register, login, and access protected resources based on their roles and permissions.

    - Database Integration: The interactions between the API and the database should be tested. This includes validating data retrieval, storage, updating, and deletion operations. Test coverage should include testing various data types, edge cases, and data validation rules.

    - Error Handling: Error handling components should be tested to ensure that appropriate error codes, messages, and responses are returned for various error scenarios. This includes testing error scenarios such as invalid requests, unauthorized access attempts, or database connection failures.

- User Data Components: The components responsible for managing and storing user data within the Easy Search API project should also be included in the test coverage.
  This typically involves:

    - User Creation and Management: Test coverage should include testing user registration, updating user profiles, and handling user-related operations. This ensures that user data is correctly stored, retrieved, and modified.

    - Data Validation: Test coverage should include validating user input data for accuracy, completeness, and adherence to predefined rules. This helps ensure that user data is valid and reliable.

    - Data Privacy and Security: Components related to data privacy and security, such as encryption, hashing, and access control, should be tested. This ensures that user data remains confidential and protected against unauthorized access.

Defect Reporting and Tracking Process

- Defect Reporting: Defect reporting involves identifying and documenting any issues or bugs discovered during testing. When a defect is identified, it should be reported in a structured and consistent manner.
  Here are the key steps involved in defect reporting:

    - Reproduce the Defect: The tester should attempt to reproduce the defect and gather relevant information about the steps to reproduce it, including the test environment, input data, and specific conditions that trigger the issue.

    - Assign Severity and Priority: The defect should be assigned a severity level based on its impact on the system and its importance. Severity levels can range from critical to low. Additionally, a priority level should be assigned to indicate the order in which the defect should be addressed.

- Defect Tracking: Defect tracking involves monitoring and managing the reported defects throughout their lifecycle until they are resolved.
  Here are the key steps involved in defect tracking:

    - Assigning and Tracking Defects: Each reported defect should be assigned a unique identifier and allocated to the appropriate developer or team responsible for its resolution. The defect tracking system should maintain a centralized repository of all reported defects, enabling easy tracking and management.

    - Monitoring Progress: The defect tracking system should provide a status field to track the progress of each defect. This allows stakeholders to monitor the defect's lifecycle, including its current status (e.g., open, in progress, resolved, verified).

    - Defect Resolution: Developers work on resolving the reported defects based on the assigned priorities. Once a defect is addressed, the developer updates its status in the defect tracking system to indicate that it has been fixed.

    - Verification and Closure: Testers verify the fixed defects to ensure that they have been correctly resolved. If the fix is verified, the defect can be closed in the tracking system. If the fix is not satisfactory or further issues are discovered, the defect may be reopened or assigned back to the developer for further investigation.

    - Reporting and Metrics: The defect tracking system should provide reporting capabilities to generate metrics and reports on defect status, trends, and resolution time. These reports help identify patterns, assess the quality of the software, and support decision-making in future development cycles.

Risk management

Identification of Potential Risks and Issues

1. Performance Issues: The system may experience performance degradation when handling a large number of documents and concurrent search requests, leading to slow response times and decreased user satisfaction.

2. Data Security Breaches: Unauthorized access to documents or sensitive user information can result in data breaches, compromising the privacy and confidentiality of users and their documents.

3. Compatibility Challenges: Different document formats and search query types may pose challenges in accurately processing and retrieving documents, potentially leading to incorrect search results or incomplete document indexing.

4. System Downtime: Unforeseen hardware or software failures, network issues, or other technical problems can result in system downtime, causing disruptions in document storage and search operations.

5. Scalability Limitations: If the system is not designed to handle increasing volumes of documents and user requests, it may struggle to scale effectively, impacting its performance and overall user experience.

Mitigation Strategies

1. Performance Issues:

    - Conduct thorough performance testing to identify potential bottlenecks and optimize the system's performance.

    - Implement caching mechanisms to improve response times for frequently accessed documents.

    - Scale up hardware resources or consider distributed computing approaches to handle increased loads.

2. Data Security Breaches:

    - Employ strong authentication and access control mechanisms to ensure only authorized users can access and manipulate documents.

- Implement encryption techniques to protect documents and sensitive user information both at rest and in transit.

- Regularly conduct security audits and vulnerability assessments to identify and address potential security gaps.

3. Compatibility Challenges:

   - Perform comprehensive compatibility testing with various document formats and search query types to ensure accurate processing and retrieval.

   - Keep the system updated with the latest libraries and plugins to support a wide range of document formats and search capabilities.

   - Continuously monitor and address compatibility issues reported by users and developers.

4. System Downtime:

   - Implement redundancy and failover mechanisms to minimize single points of failure and ensure high availability.

   - Regularly perform backups of the system and its databases to facilitate quick recovery in the event of data loss or system failures.

   - Monitor the system's health and performance in real-time to proactively detect and address any issues.

5. Scalability Limitations:

   - Design the system with scalability in mind, considering horizontal scaling options such as load balancing and distributed architectures.

   - Continuously monitor the system's performance and load patterns to identify scaling needs and make necessary adjustments.

   - Implement automated scaling mechanisms to dynamically allocate resources based on demand.

By proactively addressing these potential risks and issues through appropriate mitigation strategies, the EasySearch project can enhance its performance, security, compatibility, availability, and scalability, ensuring a robust and reliable document search service for users.

Appendix

Test Environment Configuration

1. Hardware: Set up a server infrastructure with sufficient computing resources to handle the expected load. Consider factors such as CPU, memory, and storage capacity based on anticipated usage patterns.

2. Software: Install and configure the necessary software components, including the operating system, web server, database server, and search engine. Ensure that all dependencies are met and the environment is properly configured.

3. Networking: Configure network settings to enable communication between the EasySearch API, the database, and the client applications. Set up firewalls and security measures to protect the system from unauthorized access.

4. Test Data: Prepare a diverse set of test data, including documents of different formats, sizes, and content. Include various search scenarios to cover different search query types and expected outcomes.

5. Monitoring and Logging: Enable monitoring tools to track system performance, resource utilization, and any errors or exceptions. Configure logging mechanisms to capture relevant information for debugging and analysis purposes.

Sample Test Cases

Sample Test Case 1:

Test Case: Document Upload

Objective: To verify that documents can be successfully uploaded to the EasySearch service.

Steps:

1. Launch the EasySearch application or access the EasySearch API.
2. Navigate to the document upload section.
3. Select a document file to upload.
4. Click on the "Upload" button.
5. Monitor the progress bar or status message to ensure the upload is completed successfully.
6. Verify that the uploaded document appears in the document list or database.
7. Repeat steps 3-6 with multiple document files of different formats and sizes.

8. Verify that all uploaded documents are correctly stored and accessible for search operations.

Expected Results:

- The document upload process completes without any errors or timeouts.
- The progress bar or status message accurately reflects the upload progress.
- All uploaded documents appear in the document list or database.
- Each uploaded document retains its original format and content.
- The uploaded documents can be successfully searched using appropriate search queries.

Sample Test Case 2: Document Search

Objective: To verify that the EasySearch service accurately retrieves documents based on search queries.

Steps:

1. Launch the EasySearch application or access the EasySearch API.
2. Navigate to the search interface or section.
3. Enter a valid search query in the search input field.
4. Click on the "Search" button or press Enter.
5. Review the search results displayed on the screen.
6. Verify that the search results include relevant documents matching the search query.
7. Validate that the search results are displayed in a clear and organized manner.
8. Click on a specific search result to open the document.
9. Ensure that the document content matches the search query and the search terms are highlighted or emphasized.
10. Repeat steps 3-9 with different search queries, including various keywords and combinations.
11. Verify that the search results accurately reflect the documents matching the search criteria.

Expected Results:

- The search functionality executes the search query without any errors.
- The search results include relevant documents matching the search query.
- The search results are displayed in a readable and organized format.
- The document content accurately matches the search query, and search terms are appropriately highlighted.
- Clicking on a search result opens the corresponding document with the expected content.
- Different search queries yield different search results based on the matching documents.