**Document history**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Author | Status | Comments |
| 1.0 | 05-20-2023 | Kha nguyen | Draft | Ready for review |

**References**

|  |  |  |  |
| --- | --- | --- | --- |
| Reference | Title | Author | Version |
| **1** | GeoComply - CaseStudy.pdf | **GeoComply** | **1.0** |

Table of contents

[1. Introduction 2](#_Toc135589378)

[2. Test strategy 2](#_Toc135589379)

[3. Automation Architeture 3](#_Toc135589380)

[4. Schedules 4](#_Toc135589381)

[4.1 Test implementation 4](#_Toc135589382)

[4.1.1 UI automation test implementation 4](#_Toc135589383)

[4.2 Test execution 4](#_Toc135589384)

[4.3 Test report 4](#_Toc135589385)

[5. Dependencies 4](#_Toc135589386)

[6. Risks and Assumption 4](#_Toc135589387)

[6.1 Risk 4](#_Toc135589388)

[6.2 Asumsion 5](#_Toc135589389)

[7. Tools 5](#_Toc135589390)

[8. Resouces 5](#_Toc135589391)

[8.1 QC resources 5](#_Toc135589392)

[8.2 Test environment 5](#_Toc135589393)

# Introduction

The following document outlines the test plan for the file uploading feature of GeoComply. This test plan is designed to ensure the quality and reliability of the file uploading functionality, identifying any potential issues or defects that may arise during its usage. The file uploading feature plays a critical role in the GeoComply system, allowing users to securely upload files and facilitating various important operations.

The objective of this test plan is to thoroughly test the file uploading feature across different scenarios and validate its functionality, performance, and security. The testing activities will encompass both positive and negative test cases, covering a range of file types, sizes, and potential error conditions. The primary focus will be on verifying that files are successfully uploaded, stored securely, and can be accessed and manipulated as intended.

This test plan will involve a comprehensive set of test cases that will be executed systematically, ensuring proper coverage of all relevant aspects of the file uploading feature. The tests will be conducted using appropriate test data, encompassing different file formats, large files, and edge cases to assess the system's capabilities and limitations.

The testing process will be carried out by a dedicated QA team, utilizing a combination of manual and automated testing techniques. Test environments will be set up to closely mimic the production environment, ensuring realistic testing conditions. Test data will be generated or obtained from various sources to simulate real-world scenarios and evaluate the system's performance and stability.

The test plan will also include the generation of detailed test reports, documenting the test results, any issues or defects encountered, and their respective severity levels. The reports will be regularly communicated to the development team, stakeholders, and project management to facilitate effective collaboration and decision-making.

By following this test plan, we aim to identify and address any potential issues or shortcomings in the file uploading feature of GeoComply, ensuring its reliability, security, and overall functionality. The successful execution of this test plan will contribute to delivering a high-quality product that meets the expectations and requirements of our users.

Note: This is a generic introduction and should be tailored to reflect the specific details and requirements of the GeoComply file uploading feature.

# Test strategy

1. Test Scope:
   1. Focus on automated functional and regression testing of the uploading feature.
   2. Cover various scenarios, such as file selection, uploading, error handling, and user experience.
2. Test Environment:
   1. Utilize Serenity Test Framework for efficient test management and reporting.
   2. Support multiple browsers, including Chrome, Firefox, and Safari.
   3. Test on different platforms, such as Windows, macOS, and Linux.
3. Test Approach:
   1. Develop automated test scripts using the Page Object Model (POM) design pattern.
   2. Leverage Serenity's capabilities to create reusable test steps and data-driven tests.
   3. Implement behavior-driven development (BDD) approach using Gherkin syntax for test scenarios.
   4. Prioritize test scenarios based on risk analysis and business requirements.
4. Test Coverage:
   1. Verify file selection functionality, ensuring correct files can be chosen for upload.
   2. Validate successful file upload and confirmation messages.
   3. Test error handling for invalid file types, large file sizes, or network interruptions.
   4. Verify the display of uploaded files and associated metadata.
   5. Evaluate performance by testing the feature with varying file sizes and concurrent uploads.
   6. Include security testing to ensure data protection during the upload process.
5. Test Data:
   1. Utilize a combination of valid and invalid test data to cover different scenarios.
   2. Include edge cases such as empty files, excessively large files, and special characters in file names.
6. Test Execution and Reporting:
   1. Run automated tests as part of the continuous integration (CI) process.
   2. Monitor test execution, track results, and analyze test coverage using Serenity's reporting capabilities.
   3. Generate comprehensive test reports highlighting test status, coverage, and any defects identified.
7. Test Maintenance:
   1. Regularly review and update test scripts to align with application changes and new requirements.
   2. Perform regression testing to ensure existing functionalities are not impacted by updates.
   3. Continuously enhance the test suite to cover additional scenarios and improve overall test coverage.
8. Collaboration and Communication:
   1. Collaborate with developers, business analysts, and stakeholders to clarify requirements and address any issues.
   2. Maintain open communication channels to report test results, defects, and progress updates.

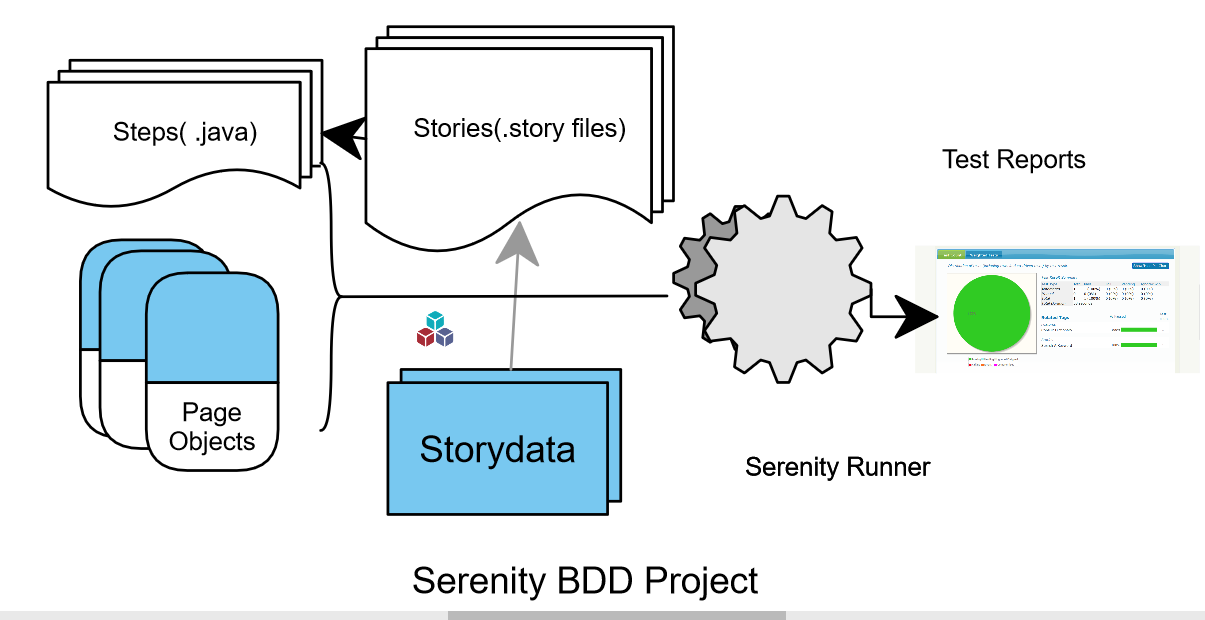
By following this test strategy, we aim to ensure thorough automated testing of the uploading feature, providing confidence in its functionality, performance, and reliability.

# Automation Architecture

The automation framework we have implemented follows a robust architecture, combining Selenium WebDriver, Serenity, and the Page Object Model (POM) design pattern. The architecture consists of several key components:

1. Selenium WebDriver: This serves as the foundation for interacting with web browsers, allowing us to perform actions like clicking buttons, filling forms, and verifying page elements.
2. Serenity: It acts as a layer on top of Selenium, providing enhanced features for test management, reporting, and result analysis. Serenity integrates seamlessly with Selenium and simplifies the test execution process.
3. Page Object Model (POM): Following the POM design pattern, we organize our test code into separate classes called Page Objects. Each Page Object represents a web page or a component of the application, encapsulating the related web elements and their corresponding actions. This promotes code reusability, readability, and maintainability.

By leveraging the Selenium WebDriver, Serenity, and the Page Object Model, our automation framework ensures a structured and scalable approach to test development. It enables clear separation of concerns, facilitates efficient test maintenance, and provides comprehensive reporting capabilities. This architecture empowers our team to deliver high-quality automated tests with ease and confidence.



# Schedules

## Test implementation

### UI automation test implementation

Provided as part of the given case study.

## Test execution

All test cases described in Scope section will be executed via CI/CD server

## Test report

Test repports will be available after execution.

# Dependencies

TBD

# Risks and Assumption

## Risk

TBD

## Asumsion

1. Test scenarios are ready and reviewed.
2. Environment in test shoud be ready.
3. Ensure availability of source control and CI/CD server.
4. The data model should be prepared.
5. Contact points of different stackholders are provided.

# Tools

1. IntelliJ
2. Serenity BDD

# Resouces

## QC resources

1. TBD

## Test environment

The test execution will be executed on QA automation environment.

QA automation environment should be ready and support by DevOps team.

# Development roadmap

1. Enhance Test Coverage: Continuously expand the test coverage by identifying additional test scenarios and implementing them in the automation framework. This ensures comprehensive validation of the application's functionality.
2. Improve Test Efficiency: Explore ways to optimize test execution time by leveraging parallel execution, efficient test data management, and optimized test case design.
3. Integration with Jira: Integrate the automation framework with Jira to establish seamless collaboration between development and testing teams. This integration allows for efficient bug tracking, test case management, and traceability.
4. Implement Chatbot Integration: Integrate a chatbot into the automation framework to provide real-time test status updates, answer frequently asked questions, and assist with test execution. This improves communication and enhances the efficiency of the testing process.
5. Continuous Integration and Deployment: Set up CI/CD pipelines to automate the build, test, and deployment processes. This ensures that the automation tests are executed as part of the overall software delivery pipeline, providing faster feedback on the application's quality.
6. Test Data Management: Implement robust and flexible test data management strategies, such as data generation, data masking, or data provisioning, to ensure the availability of reliable and representative test data for various test scenarios.
7. Reporting and Analytics: Enhance the reporting capabilities of the automation framework to provide insightful and actionable test metrics, including test execution status, defect trends, and overall test coverage. Leverage visualization tools to present test data in a clear and concise manner.
8. Integration with Test Management Tools: Explore integrations with popular test management tools to streamline test case management, test execution, and reporting. This integration improves efficiency, traceability, and collaboration among testing teams.
9. Continuous Learning and Adoption of New Technologies: Stay updated with the latest trends and advancements in the automation testing landscape. Continuously evaluate and adopt new tools, frameworks, and techniques that can further enhance the effectiveness and efficiency of the automation framework.
10. Maintainable and Scalable Framework: Regularly review and refactor the automation framework to ensure maintainability, scalability, and extensibility. Follow best practices such as modular design, code reusability, and abstraction to create a robust and future-proof automation solution.