[name of project]

**Test Strategy**

**Revision History**

| Date | Version | Author | Description |
| --- | --- | --- | --- |
| 10.02 | 1.0 | Yehor |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[**1. Scope 3**](#_gjdgxs)

[**2. Test Approach 4**](#_30j0zll)

[**3. Test Environment 5**](#_1fob9te)

[**4. Testing Tools 6**](#_3znysh7)

[**5. Release Control 7**](#_2et92p0)

[**6. Risk Analysis 8**](#_tyjcwt)

[**7. Review and Approvals 9**](#_3dy6vkm)

# Scope

It defines parameters like

* Who will review the document?

Depending on the purpose and content the document may be reviewed by the team leader by a group of testers or subject matter experts it may also be checked by editor legal/compliance team or clients/customers for quality accuracy or compliance with standards and requirements

* Who will approve this document?

Depending on the nature and importance of the document it is approved by persons such as supervisors, department heads, managers or project managers

* Testing activities carried out with timelines

Test work starting with test planning and test case development and then test environment preparation тest execution including reporting and defect tracking with continuous monitoring and problem resolution finally acceptance testing is performed followed by test closure activities summarising results and drawing conclusions

# Test Approach

It defines

* Process of testing

1. test planning
2. test case development
3. test execution
4. defect reporting and test completion.

* Testing levels

Levels of testing such as unit testing, integration testing, system testing and acceptance testing should be ensured.

* Roles and responsibilities of each team member

**Team leader by a group** **of testers** will be responsible for planning **Test analyst** will be responsible for test case development

**Testing engineer** will be responsible for test execution defect tracking and reporting.

* Types of Testing

The following types of testing should be provided on the project

load testing

Security testing

Functional testing

Regression testing

Usability Testing

* Testing approach & automation tool if applicable

Automation tools

Selenium for web application testing

JMeter for performance testing

Appium for mobile application testing.

* Adding new defects, re-testing, Defect triage, Regression Testing and test sign off

documentation of new defects, defect removal, retesting of fixed defects and regression testing. This ensures that identified issues are fixed

# Test Environment

* Define number of requirement and setup required for each environment

Number of environments and setup requirements:

We must use the following environments:

A. Development environment. It can include local development settings on individual development machines or on a shared development server.

b. Test environment. Ideally, it should reflect the production environment as closely as possible in terms of hardware, software configurations, and data.

C. Production Environment: This is the live environment in which the application or system is available to end users. It should be very stable, secure and scalable.

Each environment must meet the following requirement:

Technical characteristics of the equipment:

CPU: Multi-core processor (such as Intel Core i7 or similar).

RAM: minimum 8 GB RAM

Storage: Sufficient capacity, preferably SSD for faster read/write operations.

Software dependencies:

Operating system: Linux based OS

Web server: Nginx.

Database: MySQL.

Application platform: Node.js.

Cloud platform: Microsoft Azure.

* Define backup of test data and restore strategy

Backup and recovery strategy:

a) Regular backups:

Schedule daily backups of application data, including images and database content.

b) Automation of the backup process:

Use the automatic backup tools provided by Microsoft Azure cloud storage to optimize the backup process.

c) Backup storage:

Store backups on a secure and redundant storage system separate from your main Azure Blob Storage environment.

d) Backup storage policy:

Backups should be kept for two weeks before overwriting. The team lead can identify a copy containing significant changes to be archived

f) Recovery strategy:

- checking the integrity of the backup copy

- validation checks.

f) Testing backup and recovery procedures:

Regularly test backup and recovery procedures to ensure functionality and reliability by simulating various failure scenarios.

# Testing Tools

* Automation and Test management tools needed for test execution

To effectively run tests, we will use JEST for web applications. For test management - TestLink is open source.

Jest: A versatile JavaScript testing framework that provides an easy-to-use API and a wide range of features such as inline snapshots and mocks.

By assessing the number of concurrent users accessing these tools, we will select and ensure seamless integration with the existing development ecosystem to maximize efficiency.

* Figure out number of open-source as well as commercial tools required, and determine how many users are supported on it and plan accordingly

1. **Functional Testing:**

**Selenium WebDriver**: To automate functional testing of a web application. Selenium WebDriver allows you to run tests in different browsers, which is important for checking cross-browser compatibility.

2. **Load testing**:

**Apache JMeter**: Used to create and run load and stress tests on an application. JMeter allows you to simulate a large number of virtual users to evaluate system performance under different loads.

3. **security testing**:

**OWASP ZAP** (Zed Attack Proxy): It is used to perform security tests on web applications. ZAP helps to identify vulnerabilities such as cross-site scripting (XSS) and SQL injections.

4. **API Testing:**

**Postman**: It is used to test and debug APIs. Postman allows you to create queries, automate testing, and generate reports.

5. **Test Management**:

**TestRail** (Cloud or Open Source version): Used to manage test cases, test plans and reports. TestRail provides centralised storage of test documentation and test results.

# Release Control

* Release management plan with appropriate version history that will make sure test execution for all modification in that release

**Release Management Plan**:

This plan outlines our process for managing software releases, ensuring proper versioning and coordination with test execution for all modifications.

1. **Versioning**:

- We'll use Semantic Versioning (SemVer) with MAJOR.MINOR.PATCH format.

- MAJOR for incompatible API changes, MINOR for new features, PATCH for bug fixes.

2. **Release Process**:

- **Feature Development**:

- Developers work on features/bug fixes on separate branches.

- Code reviews ensure quality and standards.

- **Integration Testing**:

- Feature branches merge into development for tests.

- **Release Candidate**:

- RC branch created after integration testing.

- **QA Testing**:

- QA team tests RC for functionality, performance, security.

- **Staging Deployment**:

- RC deployed for user acceptance testing (UAT).

- **UAT and Bug Fixes**:

- Users provide feedback in UAT, bugs fixed promptly.

- **Release**:

- RC merged into master, tagged as new version.

3. **Test Execution**:

- Automated tests run in CI after code merges.

- Manual testing by QA for new features and regressions.

4. **Release Documentation**:

- Release notes document modifications, known issues, and instructions.

- Updated documentation reflects changes.

This streamlined plan ensures efficient release management while maintaining quality and stability.

# Risk Analysis

* List all risks that you can estimate
* Give a clear plan to mitigate the risks also a contingency plan

**Risk Analysis:**

**The identified risks are summarised below, along with mitigation plans and contingency plans**:

**Risk Mitigation Plan**:

1. **Technology Risks**:

Conduct thorough compatibility testing during the development and integration phases.

Keep up to date with the latest patches and updates for all technology components.

**Contingency Plan**:

Maintain a rollback plan to revert to previous stable versions in case of compatibility issues.

Prepare alternate technology options as backups if necessary.

2. **Performance Risks**:

Conduct load testing using tools such as Apache JMeter to simulate expected traffic.

Track performance metrics during testing and production.

**Contingency plan**:

Scale resources on the Azure platform to handle increased load.

Implement caching mechanisms to optimise performance.

3. **Security risks**:

Conduct regular security audits and code reviews.

Implement security best practices such as encryption, secure authentication mechanisms, and input validation.

**Contingency plan**:

Fix vulnerabilities as soon as they are discovered.

Activate security protocols to isolate affected components and limit potential damage.

4. **Deployment Risks**:

Automate the deployment process with tools such as Azure DevOps.

Conduct thorough testing in intermediate environments before deploying to a production environment.

**Contingency plan**:

Maintain backup servers or environments to quickly switch over in the event of a deployment failure.

Have a rollback plan to revert to previous stable versions.

5. **Team risks**:

Cross-train team members to ensure knowledge redundancy.

Maintain thorough documentation and knowledge sharing practices.

**Contingency Plan**:

Implement a succession plan to ensure continuity of operations.

If necessary, use outside resources or consultants to fill gaps.

6. **Compliance Risks**:

Stay abreast of relevant regulations (e.g. GDPR, HIPAA) and ensure compliance throughout the development and operational process.

Implement data encryption, access controls, and audit mechanisms as required by regulations.

**Contingency Plan**:

Develop a response plan to promptly resolve compliance issues.

If necessary, engage legal counsel or compliance experts to resolve regulatory issues.

By identifying and addressing these risks in advance, we can mitigate their impact on project success and keep your application running smoothly.

# Review and Approvals

1. Reviewers: All documents will be reviewed and analysed by a cross-functional team including developers, testers, project managers and business representatives.

2. Review Process: Each document will be sent to a version control system (e.g. Git) for review. The team will then have a certain period of time to conduct the review.

3. Criteria for Review: The review will be based on correct formatting, compliance with project requirements and company standards, and the adequacy and completeness of the information provided.

4. Approval Authority: Upon completion of the review, the document will be sent for approval to the responsible person who has the authority to make the final decision to accept the document.

5. Approval Process: Approval will be done by signature or approval in the version control system, after which the document will be considered approved and ready for further use in the application development and testing process.