[“Meow” application] \*мені подобається назва

**Test Strategy**

**Revision History**

| Date | Version | Author | Description |
| --- | --- | --- | --- |
| 26.04.2023 | 2 | Yuliia Huzieieva | An application that allows you to share photos of cats. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[**1. Scope 3**](#_heading=h.30j0zll)

[**2. Test Approach**](#_heading=h.1fob9te) **4**

**3. Test Environment 6**

[**4. Testing Tools 9**](#_heading=h.2et92p0)

[**5. Release Control 10**](#_heading=h.tyjcwt)

[**6. Risk Analysis 12**](#_heading=h.3dy6vkm)

[**7. Review and Approvals 14**](#_heading=h.1t3h5sf)

# Scope

Reviewer– Oksana Revvo

Approver - Oksana Revvo

Testing activities carried out with timelines:

**Unit testing:** This is the earliest form of testing and is carried out by developers themselves. The timeline for this testing is **1 day.**

**Integration testing:** Integration testing is carried out once the individual units of the software have been tested. The timeline for this testing is **1 week.**

**System testing:** This is carried out after integration testing, and it checks the complete system against the requirements. The timeline for system testing is **1 month.**

**Acceptance testing:** This is the final stage of testing and is done to ensure that the software meets the customer's requirements. The timeline for this testing activity **2 weeks. (серйозне прийомочне, зазвичай його або нема, або воно коротке у часі)**

**Regression testing:** This testing activity is carried out after any changes are made to the software to ensure that the changes have not impacted the previously working functionality. The timeline for regression testing is **4 days. (а це зазвичай найдовша активність, хоча якщо є автоматизація, то це полегшує справу)**

**Performance testing:** This testing activity checks how the software performs under different loads and conditions. The timeline for performance testing **3 days.**

# Test Approach

It defines

**Process of testing:**

Testing process: The testing process will follow the Agile methodology and will include the following phases:

Unit testing: Developers will test their code using unit testing frameworks.

Integration testing: Testers will test the integration of individual components.

System testing: Testers will test the entire system against the requirements.

Acceptance testing: Business users will test the system to ensure it meets their requirements.

Regression testing: Testing will be conducted on each iteration to ensure that changes have not affected existing functionality.

**Testing levels:**

Testing levels: The testing levels will include unit testing, integration testing, system testing, and acceptance testing, regression testing, performance testing. The objectives and criteria for each level will be defined in the test plan.

**Roles and responsibilities of each team member:**

The roles and responsibilities of each team member involved in testing will be defined in the test plan. The testing team will report issues to the project manager, who will escalate them to the development team if necessary.

**Types of Testing:**

The types of testing to be conducted will include functional testing, performance testing, security testing, and usability testing.

**Testing approach & automation tool if applicable:**

Testing will be conducted using a combination of manual and automated testing. Test management tools such as TestRail and test automation tools such as Selenium will be used.

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

Defects will be reported using JIRA, and defects will be triaged by the project manager and development team. Regression testing will be conducted after each defect is fixed, and the test sign off criteria will be defined in the test plan.

# Test Environment

**Hardware and Software Requirements:**

The test environment should include a variety of devices with different screen sizes, operating systems, and processing power. This could include devices such as iPhones, Android phones, tablets, and desktop computers running different versions of Windows, macOS, and Linux. The environment should also include various web browsers and their respective versions, such as Chrome, Firefox, Safari, and Edge.

**Scalability:**

The environment should be able to handle a large volume of traffic. Load testing should be performed using tools such as Apache JMeter or Gatling to simulate a large number of concurrent users and measure the app's performance under different load conditions. The test environment should also be configured to scale horizontally to handle additional load as needed.

**Security:**

This is a social media app that stores sensitive user data, so the test environment must be secure and comply with data privacy and security regulations. The environment should be configured with firewalls, intrusion detection and prevention systems, and other security measures to prevent unauthorized access. Test data should be masked to ensure data privacy, and the test environment should be isolated from the production environment to prevent accidental data leaks.

**Test Data Management:**

The test environment should have a comprehensive test data management system to ensure that the test data is accurate, complete, and up-to-date. Test data should be generated using tools such as Faker or DataFactory and should include a variety of scenarios, such as empty profiles, profiles with large amounts of data, and profiles with various combinations of media and interactions. Test data should be stored in a database or other data storage system that is separate from the production environment.

**Network Configuration:**

The test environment should simulate different network conditions, including high latency and low bandwidth, to test the app's performance under different network conditions. Network emulation tools such as WANem or Clumsy can be used to simulate different network conditions. Additionally, the network configuration should be in sync with the production environment to ensure that the app behaves consistently.

**Integration with other systems:**

The app integrates with various third-party services. The test environment should be configured to support these integrations and ensure that they work correctly. This may require setting up test accounts on these services and configuring the test environment to use these accounts.

**Availability and Uptime:**

The test environment should be available and up-to-date as per the testing schedule. The app is available 24/7, so the test environment should also be available 24/7. Scheduled downtime for maintenance or updates should be outside peak testing periods. Additionally, monitoring tools such as Nagios or Zabbix should be used to monitor the availability and performance of the test environment and alert the testing team in case of any issues.

**Test Environment Management:**

The test environment should be managed throughout the testing process. The test environment management plan should define processes and tools required to manage the test environment effectively. Version control systems such as Git should be used to manage changes to the test environment, and configuration management tools such as Ansible or Puppet should be used to automate the deployment of updates to the environment. Additionally, containerization technologies such as Docker or Kubernetes can be used to ensure consistent and reproducible test environments.

**Test Execution:**

The test environment should be configured to support the execution of various types of tests, such as functional, performance, security, and compatibility testing. Automated testing tools such as Selenium, Appium, or Cypress can be used to automate test execution and provide reliable and repeatable test results. Test management tools such as TestRail or Zephyr can be used to manage test cases, track defects, and provide test metrics and reports.

**Test Environment Monitoring:**

The test environment should be monitored continuously to ensure that it is stable, available, and meets the testing requirements. Monitoring tools such as New Relic or Datadog can be used to monitor the system resources, network performance, and application behavior. Additionally, logging tools such as Logstash or Graylog can be used to collect and analyze log data to troubleshoot issues and optimize performance.

\*Дуже грунтовно по даним пунктам, хороша і містка робота!

# Testing Tools

**Git:** A version control system used for managing source code. Git allows you to track changes to your source code, collaborate with other developers, and roll back changes if necessary.

Postman: A REST API testing tool used for testing APIs. Postman provides a user-friendly interface for creating and executing API tests.

**SoapUI:** A testing tool used for testing SOAP and REST web services. SoapUI provides a variety of features for creating and executing web service tests.

Jenkins: An open-source automation server used for automating software builds and testing. Jenkins provides a web-based interface for configuring and executing automated tests.

**Docker:** A containerization platform used for packaging and deploying applications. Docker allows you to package your application and its dependencies into a container, which can be easily deployed to any environment.

**Selenium WebDriver:** A browser automation tool used for automating web browsers. Selenium WebDriver allows you to automate browser actions and verify web page behavior.

**Appium:** An open-source testing tool used for automating mobile applications on iOS and Android platforms. Appium allows you to automate mobile actions and verify mobile app behavior.

Visual Studio Code: A source code editor used for writing and editing code. Visual Studio Code provides a variety of features for debugging and testing code.

**TestRail:** TestRail for test case creation, test case tracking, and manual test case execution. TestRail provides a centralized location for managing test cases and test runs, and allows for easy tracking of test execution results.

**Jira:** Jira for defect management and test reporting. Jira provides a comprehensive defect management system that allows you to track defects from creation to resolution. Jira also provides reporting capabilities that allow you to generate reports on test execution results and defect metrics.

**PDF:** PDF for generating test reports. You can export test execution results from TestRail or Jira in PDF format to share with stakeholders.

**Microsoft Excel:** Microsoft Excel for creating checklists. Excel provides a simple way to create and manage checklists. +

# Release Control

**Release objectives:**

Deliver new features, such as a new search functionality or a new payment method

Fix critical bugs that affect the user experience

Improve performance and stability of the application

Release updates to comply with new regulations or security standards

**Release schedule:**

Release date is set for the end of the sprint or the end of the month

Duration of the release is one week, including deployment and testing

Timeline for each phase of the release is defined, such as testing, bug fixing, and deployment

**Release criteria:**

The application must pass a set of critical test cases with no major defects

The application must comply with performance and security standards

The application must be approved by the product owner and the business users before release

**Release procedures:**

Deploy the application to a staging environment for testing and approval

Roll back the release if critical defects are found after deployment

Communicate with stakeholders, such as developers and business users, to keep them informed of the release progress

**Release documentation:**

Release notes that include a summary of new features and bug fixes

User manuals and installation guides for the updated application

A list of known issues and workarounds for the users

**Release testing:**

Functional testing to ensure that new features work as expected and critical bugs are fixed

Performance testing to ensure that the application can handle expected user load

Security testing to ensure that the application is protected against common vulnerabilities

**Release stakeholders:**

Developers responsible for coding and testing new features and bug fixes

Testers responsible for ensuring the quality of the application

Project managers responsible for overseeing the release process

Business users responsible for approving the release and testing new features from a user perspective +

# Risk Analysis

**Inadequate Test Coverage:**

Inadequate test coverage may occur when the testing team fails to cover all critical and high-risk areas of the application. This may happen due to insufficient resources, time constraints, or a lack of understanding of the application's requirements. To mitigate this risk, the testing team should prioritize the testing based on criticality and risk, and ensure that all critical and high-risk areas of the application are covered by testing.

**Poor Test Planning:**

Poor test planning may occur when the testing team does not plan the testing process properly. This may happen due to insufficient resources, time constraints, or a lack of understanding of the testing requirements. To mitigate this risk, the testing team should create a detailed test plan that includes test objectives, scope, resources, timelines, and risks, and ensure that the test plan is reviewed and approved by all stakeholders.

**Inefficient Test Execution:**

Inefficient test execution may occur when the testing team does not execute the test cases properly. This may happen due to inadequate training, lack of attention to detail, or a lack of understanding of the test cases. To mitigate this risk, the testing team should ensure that all team members are adequately trained on the testing process, and that the test cases are reviewed and approved by all stakeholders before execution.

**Inadequate Test Data:**

Inadequate test data may occur when the testing team does not have access to adequate test data. This may happen due to data privacy concerns, data availability issues, or a lack of understanding of the data requirements. To mitigate this risk, the testing team should work with the development team to ensure that adequate test data is available, and that the data is properly sanitized and secured before use.

**Testing Environment Issues:**

Testing environment issues may occur when the testing environment is not set up properly. This may happen due to hardware or software issues, configuration issues, or a lack of understanding of the testing requirements. To mitigate this risk, the testing team should ensure that the testing environment is properly set up, configured, and validated before testing.

**Inadequate Test Automation:**

Inadequate test automation may occur when the testing team does not leverage test automation effectively. This may happen due to a lack of expertise, inadequate resources, or a lack of understanding of the benefits of test automation. To mitigate this risk, the testing team should identify areas that can be automated, and develop a test automation strategy that includes tools, frameworks, and resources.

**Poor Communication:**

Poor communication may occur when there is a lack of communication between the testing team and other project stakeholders. This may happen due to cultural differences, language barriers, or a lack of understanding of the communication requirements. To mitigate this risk, the testing team should establish clear communication channels, define communication protocols, and ensure that all stakeholders are aware of the testing progress and issues.

\*Гарний аналіз ризиків

# Review and Approvals

|  | Project Manager | QA Lead |
| --- | --- | --- |
| Name | Oksana Revvo | Oksana Revvo |
| Signature |  |  |

\*Стратегія складена дуже добре, робота на відмінно