TEST PLAN

Type of document: Test Plan (combining the general testing/regression aspects in reference to the 'Like' Functionality)
Version: 1.9
Creation date: 22.01.2025
Update date: 25.01.2025
Prepared by: ZBIGNIEW BERNACKI software tester

Spis treści

1. INTRODUCTION

1.1 Document purpose and interpretation method

This document is a test plan covering:

1. General rules and procedures aimed at minimizing the risk of regression in the project (including unit, integration, module, sanity, smoke, full tests, etc.).
2. The concept of tests for the ‘Like’ Functionality in the daycare’s web application and the mobile application for parents, including acceptance criteria and test scenarios required by the client.

   [*assumption: the mobile application is for parents, and the web application is for the daycare staff]

This Test Plan should be considered the main document describing how to plan, perform, and report tests that ensure high software quality and meet the client’s requirements (including no regression in already functioning areas).

1.2 Definition of regression

The phenomenon of regression involves the appearance of new defects in parts of the application that were not directly modified in the course of development or bug fixes. Therefore, it is crucial to test existing functionalities (both automated and manual) in a comprehensive and regular manner to detect unwanted changes in the code or application behavior in time.

---

2. PRODUCT DEFINITION

The application is a system enabling daycares to share various kinds of information and observations about children (photos, descriptions of activities, comments, progress assessments, etc.) with parents. Parents have access to a mobile application (Android/iOS), while daycare staff have access to a web application.

‘Like’ Functionality:

- Allows parents and daycare staff to like (heart icon) a given child’s observation/activity.
- Each observation has a ‘Like’ counter.

- It is possible to display a list of people who have liked the observation.
- Staff members are additionally labeled with “(Nursery or other name)”.

[*More details in section no. 9 Tests of the ‘Like’ Functionality.]

---

3. TECHNOLOGIES USED OR PLANNED IN THE PROJECT

- **Web application** (for daycare staff):
  a) Software development – React/Angular/Vue framework, backend (Java/Spring, .NET, Node.js), database, Git.
  b) Software testing – Postman, Playwright, TestRail, Jenkins, Cucumber, Git, Jira for respective tasks.
- **Mobile application** (for parents):
  native technologies (Kotlin/Swift) or hybrid (Flutter, React Native).
- **Project management** (tools for the project team):
  Jira, Azure Microsoft, Confluence, SharePoint

3.1 Main tools for automated tests

- **Playwright** – we designate Playwright as the main testing tool (especially E2E tests in the web application) due to:
  1. Support for multiple browsers (Chromium, Firefox, WebKit, Edge, Safari, etc.),
  2. Test stability and speed,
  3. Easy integration with CI/CD,
  4. A rich API for simulating user behaviors (clicking, typing, copying, inputting content, and many other user-like actions).
- **Postman** – because of the need to test the API (REST) used in the application, as well as verify correct communication between the frontend and backend.
  Postman enables:
  1. Creating and running automated API tests,
  2. Controlling flows and parameters in requests,
  3. Easily sharing test collections within the team,

4. Quick verification of server responses and regression tests in the communication layer.

---

4. TEST APPROACH AND REGRESSION

4.1 The regression testing process

Due to the complexity and size of the application, verification of whether a regression has occurred should be continuous:

- On the developers environment – developers check if new code does not cause degradation (unit tests, code review).
- On testing/QA environments – testers verify the application's behavior in business processes and critical functionalities (functional tests, smoke, sanity).
- On the pre-production environment – the client may have access and, together with the test team, they jointly check if the application behaves correctly before going live.

4.2 Documentation and risk analysis

In order to effectively perform regression tests, it is necessary to:

- Maintain correct and up-to-date documentation (architecture, flow diagrams, test scenarios).
- Continuously assess risk (functionality priority, potential failure consequences).
- Regularly update test cases and test data to avoid the pesticide paradox [explanation in later sections].

4.3 The test pyramid structure

1. **Base** – unit tests, written and maintained by developers, covering key parts of the code (recommended ≥85% coverage).
2. **Higher level** – module, integration, sanity, smoke tests (both automated and manual).
3. **Top level** – end-to-end tests (full tests) and acceptance tests (conducted with the client), which verify complete business processes.

4.4 Test automation – definition, purpose, maintenance

1. **What is test automation?**
   Test automation involves creating and running test scripts (e.g., in Playwright, Postman) to check the application's behavior without having to perform all actions manually. These scripts replicate real user actions or service/API communication.
2. **Why perform test automation?**
   - Shortening the testing time of repetitive scenarios,
   - Detecting regression in various areas of the application after making changes,
   - Improving quality by running tests frequently and quickly,
   - Saving costs in the long-term development of the project.
3. **How to maintain test automation?**
   - Regularly updating the test scripts whenever the application logic changes,
   - Version control (e.g., Git) – storing scripts in the same repository as the application code, using a new branch per new implementation,
   - Integration with CI/CD – tests run continuously, reporting results,
   - Ongoing reviews (code review) of automated tests to avoid outdated scripts and minimize false positives.
4. **Who handles test automation?**
   - Test automation is performed by testers with programming skills, enabling them to create and maintain scripts in languages appropriate to the chosen tool (e.g., TypeScript/JavaScript for Playwright, Postman scripts, etc.).
5. **Why are automation testers and automated tests needed?**
   - Speed and efficiency – allows running a large number of tests in a short time,
   - Early defect detection – CI/CD integration enables constant quality monitoring,
   - Extended test coverage – it is not always possible to manually cover many browsers, data variants, and paths in a short time,
   - Consistent quality – automated tests operate identically each time, eliminating the risk of human error.

---

5. RESOURCES AND ROLES

5.1 Teams

- **Development Team (Dev Team)**
    - Maintaining code quality (unit tests, code review).
    - Regularly running lower-level functional tests.
    - Fixing defects reported by testers and the client.
- **Testing Team (Test Team)**
    - Designing and executing manual and automated tests on QA and pre-production environments.
    - Preparing test scenarios, reporting results.
    - Verifying new modifications, retesting critical areas.
- **Client**
    - Providing the specification, business requirements, acceptance criteria.
    - Performing retests and accepting the application on the pre-production environment – if so agreed by both parties.

5.2 Types of tests

- **Unit tests** – verify a single piece of code (Dev Team).
- **Module tests** – smallest functional module tests (Dev Team / Test Team).
- **Integration tests** – verify cooperation between modules (Dev Team / Test Team).
- **Sanity tests** – check key business functions after changes (quick scanning of functionality).
- **Smoke tests** – quick overview of the overall application stability.
- **Full tests** – a comprehensive verification of business processes (Test Team / Client).
- **Non-functional tests** – performance, security tests (require specialized approach).

(Automated tests can apply to both the API integration level (Postman) and the graphical interface (Playwright), providing broader test coverage. Playwright can combine tests from Postman with UI tests.)

---

6. TEST ENVIRONMENTS

1. **Development** – used by developers for continuous code changes.
2. **QA/Demo/Test** – environment for quality verification by testers (as close to production as possible).
3. **Pre-production** – the target environment for acceptance tests (with client involvement), nearly identical to production (sensitive data replaced with fake data or protected in the same way as in Production).
4. **Production** – the environment used by end users (parents, staff).

(It is important to keep environments independent so as not to affect test quality.)

---

7. SCHEDULE

- **Start of tests:** 01.02.2025
- **Testing period:** continuous testing in parallel with software development
- **Regression tests (fixes):** before each major Deployment + possible regression testing in case of urgent hotfixes
- **End of tests:** final handover of the software to the client

The schedule may change depending on environment availability, fixes, or business priorities.

---

8. ENTRY AND EXIT CRITERIA

8.1 Entry criteria

1. A provided version of the application (web, mobile) with the ‘Like’ feature implemented.
2. Access to test environments with appropriate configuration.
3. Test accounts (parent, staff).
4. At least one child observation in the system or the ability to add one.
5. Complete (at the given time) requirements and specifications (or user stories).

8.2 Exit criteria

1. All test cases have been executed, with no open critical defects (P1, P2).
2. The client has accepted the test results and confirmed that functional requirements are met.
3. It has been confirmed there is no regression in critical areas and no blocker defects preventing deployment.

---

9. TESTS OF THE ‘Like’ FUNCTIONALITY

This section focuses on the client’s requirements for ‘Like’ and the detailed tests related to this functionality.

9.1 Scope of the ‘Like’ functionality tests

1. Adding/removing 'Like' (heart icon) in observation cards (mobile application, web application).
2. Adding/removing 'Like' on the observation details screen (mobile application).
3. Displaying the 'Like' counter next to the heart icon (hidden when 0).
4. List of users who liked the observation (in both applications), labeling staff with “(Nursery or other name)”.
5. Checking data consistency between the web application and the mobile application.
6. Out of scope – performance tests, security tests (unless the client has specific requirements).

9.2 Test scenarios

Attachment excel file “Functionality of 'Like' - SAMPLE_ID-123**” may contain detailed steps, expected results, and data.

---

10. TEST DATA

- **Test accounts**:
    - Parent account: [e.g., login: rodzic_test1, password: …].
    - Staff account: [e.g., login: personel_test1, password: …].

- **Observations in the system**:
    - “Joe started a day” – 08:30
    - “Joe had a breakfast” – 09:00
    - Or others generated by the system/according to arrangements or others generated by the system/according to arrangements or others generated by the system/according to arrangements
    - The possibility of mocking data through automated tests to verify frontend behavior,

[∗If needed,additional observations or data can be created depending on project requirements]

---

11. TEST RESULT AND DEFECT REPORTING

- All test results will be recorded in JIRA, TestRail.
- Defects will be classified by priority (P1 – critical, P2 – high, P3 – medium, P4 – low) and escalated to the development team.
- After each test session, a summary report will be drawn up, listing the number of defects found and the status of all executed tests.

---

12. DIVISION OF TEAM RESPONSIBILITIES

What?	**Dev Team**	**Test Team**	**Client**
User stories	[N/A]	Creates and consults	Accepts / also creates
Unit tests	Creates	Accepts if needed	[N/A]
Test scenarios	[N/A]	Creates and accepts	[N/A]
Acceptance criteria	[N/A]	Accepts	Creates
Module tests	Creates own	Creates own and executes	[N/A]

What?	Dev Team	Test Team	Client
Integration tests	Creates own	Creates own and executes	[N/A]
Sanity tests	[N/A]	Executes	[N/A]
Smoke tests	[N/A]	Executes	[N/A]
Full tests	[N/A]	Executes	Co-executes (acceptance)

---

13. ATTACHMENTS AND ADDITIONAL MATERIALS

1. **Checklist** – containing the sequence of testing tasks.
    - fillindata–e.g.,link to file, PDF attachment fill in data – e.g., link to file, PDF attachment fill in data – e.g link to file, PDF attachment
2. **Attachment "Scenarios&TestCases-Like_Functionality"** – a detailed table of test cases (steps, data, expected result).
    - excel file "Scenarios&TestCases-Like_Functionality"

---

14. GLOSSARY (DEFINITIONS)

- **Regression** – the appearance of new defects in already functioning areas not part of a given implementation or fixes.
- **Retest** – re-testing a functionality or component where defects were previously discovered.
- **User stories (user story)** – short scenarios describing the user's goals and needs.
- **Pesticide paradox** – repeatedly running the same tests may reduce their effectiveness, so it's important to change data and update tests.
- **Continuous integration (CI)** – the team constantly integrates small code changes, allowing frequent and quick detection of errors.
- **Continuous delivery (CD)** – the process of continuously deploying the application in small increments to the target environment.
- **Smoke test** – a quick overview of the application's stability (running key functions).

- **Sanity test** – checking if the most important use cases work correctly after changes.

---

15. FINAL REMARKS / CONCLUSIONS

- In case of questions regarding requirement details, test data, or environments, please contact the Analyst/PO.
- **Important!** This document must be updated as the software evolves and subsequent versions are implemented, to prevent regression in the future.