PROJECT PLAN

FUND REQUESTS MANAGEMENT

FUNDIT BY DIVPLUSPLUS

COMPREHENSIVE TESTING PLAN REPORT

PREPARED BY DIVPLUSPLUS

TEAM:

ISMAAEEL SHAIKH EBRAHIM – 2435817

YONATAN AZARAF – 2460755

ROHAN CHHIKA – 2543404

NAFTALI DINER – 2144598

SIZWE SITHOLE – 2393358

INTRODUCTION:

In the development of the SDProject (Node Express backend) and SD_REACT_CHHIKA (React frontend), adopting a Test-Driven Development (TDD) methodology is imperative to ensure that both applications not only meet the functional requirements but also maintain a high standard of quality and robustness. This document outlines a detailed testing plan that describes the strategies and methodologies to be employed throughout the development lifecycle of both projects.

OBJECTIVES

The primary objective of this testing plan is to ensure the integrity, reliability, and performance of all features implemented in the SDProject and SD_REACT_CHHIKA. By incorporating TDD, the development teams are expected to improve design, catch defects early, and facilitate refactoring which leads to a more flexible and bug-free codebase.

TESTING METHODOLOGY

Test Driven Development is at the heart of our testing strategy. TDD is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved to pass the new tests. The process follows three main steps:

1. **Write a Failing Test:** Every new feature begins with writing tests that describe expected behavior, which fail initially since the feature isn't implemented yet.
2. **Make the Test Pass:** The next step involves writing code that makes these tests pass. This ensures that the software only contains code that is necessary to pass the test, thereby avoiding over-engineering.
3. **Refactor:** Once the tests pass, the next iteration involves cleaning up the code without changing its functionality. The tests are run again to ensure that refactoring has not introduced any new bugs.

TOOLS AND FRAMEWORKS

For the backend (SDProject), we use Mocha and Chai for writing comprehensive unit and integration tests, Supertest for HTTP assertions, and Sinon for creating spies, mocks, and stubs. For the frontend (SD_REACT_CHHIKA), Jest is utilized as the primary testing framework along with the React Testing Library to handle component-level testing and Mock Service Worker (MSW) for handling network request simulations.

DETAILED TEST SCENARIOS

For the backend, the tests cover a variety of scenarios including user authentication processes like registration and login, CRUD operations on fund management, and handling of manager requests. The frontend tests ensure that all components render correctly, user interactions such as clicks and form submissions result in the expected behavior, and state management changes trigger the correct application responses.

EXECUTION

Testing is integrated into the daily development process with automated test suites running as part of the continuous integration (CI) pipeline. Before any release, a comprehensive test run is mandatory to ensure all parts of the application work harmoniously together. Post-release, smoke tests are conducted in the production environment to ensure the deployed features function as expected.

REPORTING AND FEEDBACK

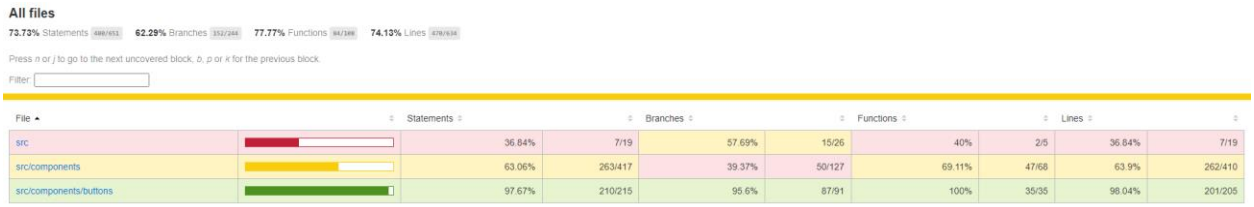
Test results are documented and made accessible to all team members via the CI pipeline's reporting tools. Issues identified during testing are logged into the project's issue tracker with appropriate tags for priority and severity. Furthermore, a quarterly review of the testing strategies is conducted to ensure the methodologies remain effective and incorporate improvements based on feedback from the developers, testers, and stakeholders.

CONCLUSION

Adhering to this structured testing plan ensures that the SDProject and SD_REACT_CHHIKA are developed with a focus on quality from the outset. By integrating TDD into our development practices, we aim to deliver applications that not only meet the project's requirements but are also scalable, maintainable, and robust. This approach underscores our commitment to quality and our proactive stance on issue resolution, making our software development process both efficient and effective.

RESULTS:

Frontend



File		Statements		Branches		Functions		Lines	
src		36.84%	7/19	57.69%	15/26	40%	2/5	36.84%	7/19
src/components		63.06%	263/417	39.37%	50/127	69.11%	47/68	63.9%	262/410
src/components/buttons		97.67%	210/215	95.6%	87/91	100%	35/35	98.04%	201/205

Backend

File		Statements		Branches		Functions		Lines	
SDProject		78.57%	143/182	67.24%	39/58	88.88%	16/18	78.57%	143/182
SDProject/config		87.5%	7/8	100%	0/0	100%	1/1	87.5%	7/8
SDProject/models		100%	20/20	100%	0/0	100%	0/0	100%	20/20

Throughout the testing process, we achieved an average test coverage of 77.34%, with the backend (SDProject) reaching 80.95% coverage, reflecting a thorough validation of our server-side logic. The frontend (SD_REACT_CHHIKA), although slightly lower, achieved a respectable

73.73% coverage, indicating a solid effort in verifying user interactions and visual components. This balanced approach ensures that both the server and client-side functionalities are rigorously tested, providing a stable and reliable software solution.