Testplan

**­­­­­­­­­­­**



Student name: Rowan van der Weel

Student number: 4924576

Class: DB-04

Teacher: Tim Kurvers

Version: 1.0

Date: 19/01/2024

Table of contents

[**Introduction** 3](#_Toc154240026)

[**Results** 3](#_Toc154240027)

[**Resolution** 3](#_Toc154240028)

[**References** 3](#_Toc154240029)

[**Version History** 3](#_Toc154240030)

# **Introduction**

This document serves as a guide for outlining and executing the testing procedures for GamifyWork. It encompasses the test strategy, elucidating the rationale behind specific tests and the intended outcomes. Following the strategy, User Acceptance Tests (UATs) will be enumerated. It's important to note that this document is dedicated to detailing the types of tests to be conducted and is not intended for documenting test results.

## Test Environment

For GamifyWork, the chosen technology stack will be .NET 6.0. Anticipating the project's expansion, I have opted for a MySQL database, as it offers efficient data management capabilities. The backend will be developed using C# with ASP.NET, complemented by a frontend built on the React framework. Version control will be facilitated through GitHub, and the CI/CD pipeline will be managed using GitHub Actions.

## Test strategy

**Unit Test**

Unit tests will be continuously executed to assess the functionality of specific features or modules in the code. These tests, integrated into the CI pipeline, run automatically on each GIT push to verify that small code segments operate as intended. Employing Mockito, dependencies are mocked to isolate the code being tested, ensuring precision in evaluating individual components.

**Integration Test**

Integration tests amalgamate the small units tested in unit testing to guarantee the seamless functioning of new and updated features within the overall system. These tests, executed by the CI pipeline, ascertain that these changes integrate appropriately with the application.

**Acceptance Test**

Acceptance tests, exclusively manual, are designed to validate that all acceptance criteria outlined in the backlog are met. Each test includes details such as the user story, preconditions (if applicable), test date (if needed), and the expected result. Successful execution of these tests, aligning with expected results, ensures the fulfilment of all acceptance criteria.

**End-to-End Test**

End-to-end tests, automated via the pipeline, evaluate the application's functionality from start to finish. Like user acceptance testing, these tests scrutinize the entire application after successful system testing. Using Cypress as the testing tool, end-to-end tests validate the integration of frontend and backend components, providing a comprehensive assessment of the application's performance.

**Utilizing SonarQube for Static Code Analysis**  
Ensuring Code Quality and Compliance with Standards. Additionally, verifying extensive code coverage through testing.

# **Results**

## Unit tests

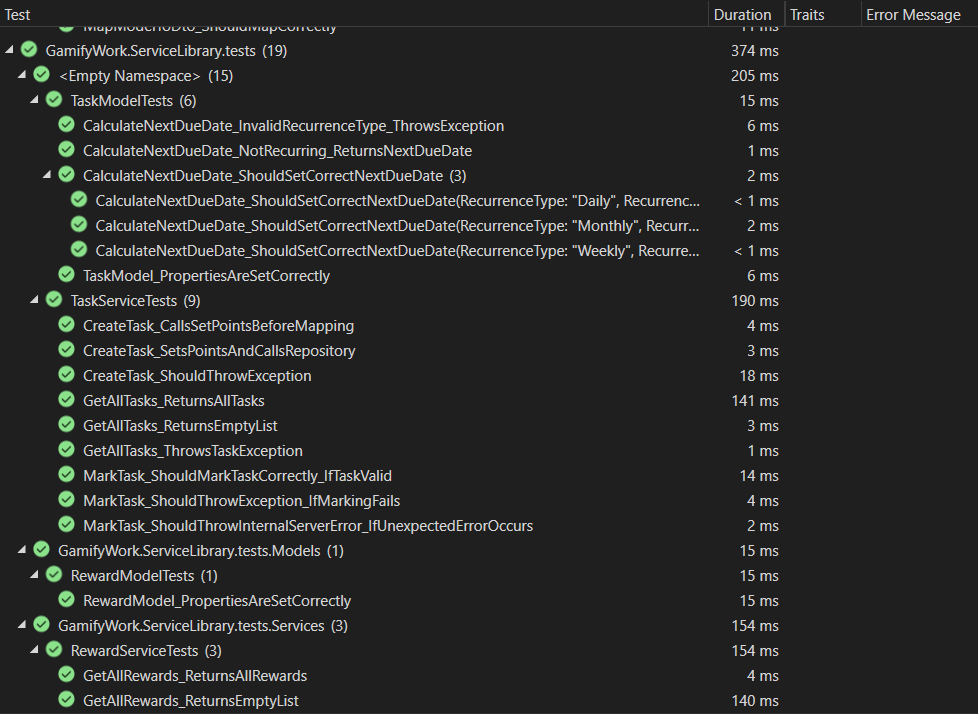
**Backend**

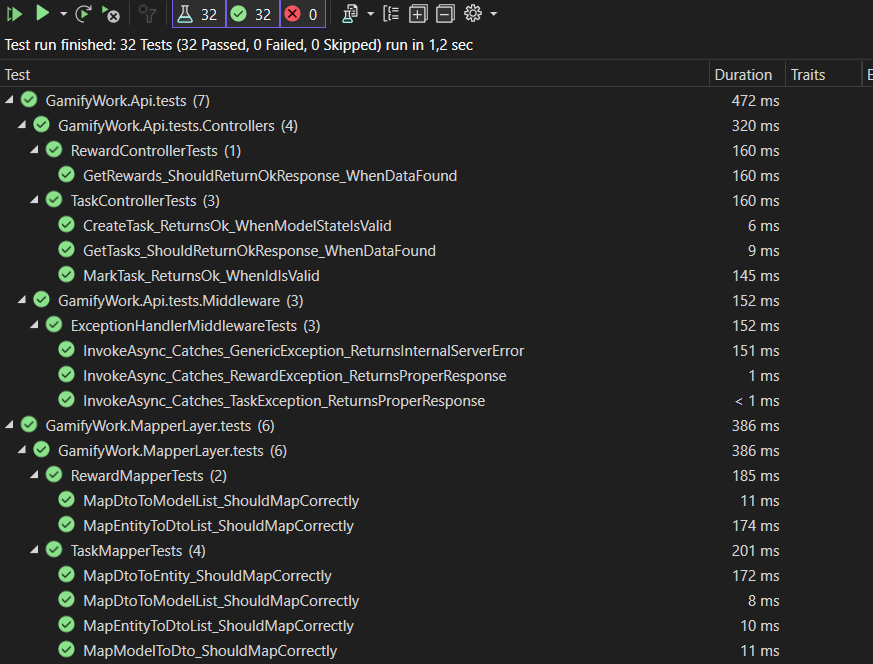
Unit testing in the backend involves the examination of individual components or functions to ensure they operate as intended. This process comprises both the "happy flow," where the system performs under normal conditions, and the "sad flow," which evaluates how the system handles unexpected or erroneous scenarios.

In the happy flow, unit tests assess the expected behavior of backend functionalities when input data is correct, and the system is functioning optimally. This validation ensures that core features and modules perform their designated tasks accurately, contributing to the overall reliability of the backend.

Conversely, the sad flow focuses on testing the backend's resilience and error-handling capabilities. Unit tests in the sad flow simulate various adverse conditions, such as incorrect inputs, unexpected data, or system failures. The objective is to verify that the backend gracefully handles such scenarios, providing meaningful error messages and maintaining stability even in challenging circumstances.

By thoroughly testing both the happy and sad flows, backend unit testing helps identify and rectify potential issues early in the development process, enhancing the robustness and dependability of the overall backend system. Below are the images depicting the backend unit tests.



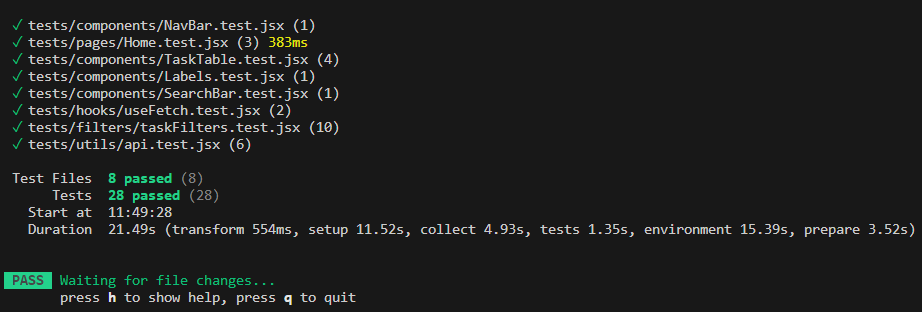


**Frontend**

Frontend testing focuses on validating the functionality and user interface of the application from the end user's perspective. Vitest, a popular testing framework, is employed to conduct these tests efficiently.

Vitest allows developers to automate the testing of various frontend components, ensuring that user interactions, visual elements, and overall user experience align with the intended design. By incorporating Vitest into the testing process, developers can create automated scripts to simulate user actions and assess how the frontend responds under different scenarios.

These frontend tests play a crucial role in identifying and rectifying issues related to user interface elements, responsiveness, and overall usability. Ensuring a seamless and error-free frontend experience is essential for delivering a high-quality application to end users. The inclusion of Vitest streamlines the testing process, providing developers with a reliable tool to enhance the frontend's performance and user satisfaction.



## Integration tests

Integration testing in the frontend focuses on evaluating the collaborative behaviour and seamless interaction among various components within the user interface. These tests are designed to ensure that different modules, services, and data sources integrate effectively, contributing to a cohesive and reliable frontend application.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Description** | **Preconditions** | **Expected Result** |
| TC-01 | Render loading state | useFetch hook returns loading state. | The homepage should not render anything except a blue screen with the logo blanking. (indicating the loading screen). |
| TC-02 | Render error state | useFetch hook returns error. | The homepage should render an error component with in that the error message returned from the hook. |
| TC-03 | Render Tasks and rewards | useFetch hook returns tasks and rewards | The homepage should correctly render the fetched tasks and rewards |

## Acceptance tests

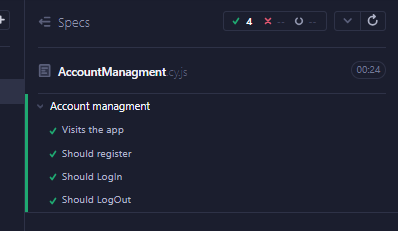
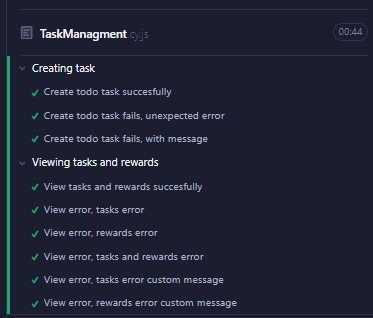
Acceptance tests are a set of systematic evaluations designed to ensure that a software application meets the specified requirements and satisfies user expectations. These tests focus on verifying that the application functions correctly and delivers the intended user experience. In the provided context of GamifyWork, each acceptance test corresponds to a specific user story, detailing the expected behaviour, preconditions, and test data. By executing these tests, developers can validate that the software aligns with user needs, promoting reliability and functionality across various features, from task management to social interactions within the application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **User story** | **Preconditions** | **Test data** | **Expected result** |
| AT-01 | View tasks in homepage | User is logged in | - | The homepage displays a list of tasks, categorizing them as to-dos recurring tasks and rewards. User can filter tasks by “completed”, “Active” and “All”. |
| AT-02 | Create tasks | User is logged in | Task title with a maximum of 50 characters. | Users can create tasks categorized as "daily tasks" and "to-do". Recurring tasks can be set up even after marking as completed. |
| AT-03 | Mark tasks | User is logged in | - | Users can click on a task to mark it as completed. Visible action, like a checkbox, associated with each task. |
| AT-04 | Recurring tasks | User is logged in | - | After a user has marked a recurring task, it will be unmarked after the period set of the task. |
| AT-05 | Create account | User is not registered | User registration details (username, email, password). | Users can successfully register an account with valid credentials. |
| AT-06 | Chat with other users | Usesr is logged in | - | Users can send and receive messages to/from friends within the application. The chat interface allows for a real-time conversation with other users. |

## End-to-End tests

End-to-end (E2E) testing is a comprehensive testing approach that evaluates the entire application workflow from start to finish, ensuring that all components and interactions work seamlessly together. This type of testing aims to simulate real user scenarios and identify potential issues that might arise in a production environment.

Cypress is a popular end-to-end testing framework known for its simplicity and efficiency. It allows developers to write tests using JavaScript or TypeScript, providing a robust testing environment for web applications. Cypress enables the creation of tests that interact with the application's frontend, mimicking user actions and validating expected outcomes.

# **Version History**

|  |  |
| --- | --- |
| **When?** | **What?** |
| 17/01/2024 | Start, unit and integration tests |
| 18/01/2024 | Added acceptance end E-2-E tests |