**Naviria**

**План тестування**

**(Test Plan)**

**Вурсія 1.0**

**(Version 1.0)**

Історія змін документа

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Версія | Опис | Автор |
| 2025-05-10 | 1.0 | Початкове створення тестового плану | Маріам Алмакадма |

Зміст

[1. Вступ 5](#_Toc199020631)

[1.1 Члени команди 5](#_Toc199020632)

[2. Обсяг тестування 5](#_Toc199020633)

[2.1 Функціонал, що підлягає тестуванню 5](#_Toc199020634)

[2.2 Функціонал, що не тестується 6](#_Toc199020635)

[3. Цілі якості 6](#_Toc199020636)

[3.1 Основні цілі 6](#_Toc199020637)

[3.2 Додаткові цілі 7](#_Toc199020638)

[4. Припущення та ризики 7](#_Toc199020639)

[4.1 Припущення 7](#_Toc199020640)

[4.2 Ризики 7](#_Toc199020641)

[5. Підхід до тестування 8](#_Toc199020642)

[5.1 Ручне тестування 9](#_Toc199020643)

[5.2 Автоматизоване тестування 9](#_Toc199020644)

[6. Критерії початку та завершення тестування 9](#_Toc199020645)

[6.1 Критерії початку тестування 9](#_Toc199020646)

[6.2 Критерії завершення тестування 10](#_Toc199020647)

[7. Критерії призупинення та відновлення тестування 10](#_Toc199020648)

[7.1 Критерії призупинення тестування 10](#_Toc199020649)

[7.2 Критерії відновлення тестування 11](#_Toc199020650)

[8. Стратегія тестування 11](#_Toc199020651)

[8.1 Роль тестувальника у процесі тестування 11](#_Toc199020652)

[8.2 Життєвий цикл дефекту 12](#_Toc199020653)

[8.3 Типи тестування 13](#_Toc199020654)

[8.4 Визначення рівнів серйозності та пріоритетності дефектів 15](#_Toc199020655)

[9. Потреби в ресурсах та середовищі тестування 17](#_Toc199020656)

[9.1 Інструменти тестування 17](#_Toc199020657)

[9.2 Управління конфігурацією 18](#_Toc199020658)

[9.3 Тестове середовище 18](#_Toc199020659)

[10. Графік тестування 19](#_Toc199020660)

1. Вступ

Цей Тестовий План описує стратегію забезпечення якості для проєкту Naviria, включаючи серверний API (NaviriaAPI на C#), вебінтерфейс (React) і мобільний застосунок для Android. Містить цілі тестування, підходи, ролі, середовища та графіки, щоб забезпечити стабільну роботу всіх компонентів системи відповідно до вимог користувачів.

* 1. Члени команди

У таблиці 1.1 наведено список членів команди та їх ролі.

Таблиця 1.1 – Таблиця членів команди

|  |  |
| --- | --- |
| Ім’я та Прізвище | Посада |
| Маріам Алмакадма | QA Engineer |
| Анастасія Червенко | C# Developer |
| Надія Дашко | Frontend Developer |
| Єлизавета Попова | Mobile Developer (Android) |

1. Обсяг тестування
   1. Функціонал, що підлягає тестуванню

До функціоналу, який підлягає тестуванню в межах проєкту «Naviria», належать як функціональні, так і нефункціональні вимоги. Зокрема, буде перевірено наступне:

Функціональні вимоги:

* реєстрація та вхід користувачів (auth flow);
* управління профілем (досягнення, статистика);
* управління цілями (завдання);
* аналітика та відстеження прогресу (графіки);
* гейміфікація (бали, рівні, таблиця лідерів);
* сповіщення;
* чат зі ШІ;
* соціальні функції (друзі, виклики);
* робота з базою даних (інтеграція з MongoDB);
* API-функціональність (валідація відповідей);
* контроль доступу (перевірка автентифікації);
* обробка помилок (400, 404, 500).

Нефункціональні вимоги:

* продуктивність;
* масштабованість;
* стабільність роботи та відсутність критичних збоїв;
* зручність користування інтерфейсом у браузері та на мобільному пристрої;
* правильна реалізація авторизації, захист персональних даних;
* поведінка під навантаженням.
  1. Функціонал, що не тестується
* все, що не зазначено у пункті 2.1.

1. Цілі якості
   1. Основні цілі

Основною метою забезпечення якості в межах проєкту «Naviria» є гарантування того, що всі функціональні та нефункціональні вимоги до системи виконуються повною мірою.

Особлива увага приділяється стабільності та масштабованості серверної частини і вебінтерфейсу при роботі під навантаженням, що дозволяє забезпечити надійну роботу системи в умовах високої інтенсивності користувацьких запитів.

Не менш важливим є забезпечення безперебійної інтеграції між серверною частиною, вебінтерфейсом та мобільним застосунком, що дозволяє досягти узгодженості функціонування всіх компонентів. Крім того, кінцевою метою є підтвердження готовності програмного забезпечення до випуску шляхом досягнення відповідності всім вимогам і очікуванням зацікавлених сторін.

* 1. Додаткові цілі

Серед другорядних, але важливих завдань є раннє виявлення дефектів на етапах розробки, що дозволяє мінімізувати витрати на їх усунення. Ефективна комунікація в межах команди, зокрема оперативна передача інформації про дефекти, сприяє підвищенню швидкості реакції на виявлені проблеми. Також важливим є забезпечення широкого покриття тестами як основного функціоналу, так і граничних сценаріїв використання, що дозволяє досягти більшої впевненості в надійності програмного продукту.

1. Припущення та ризики
   1. Припущення

* код буде структурованим та викладеним у GitHub;
* основне тестування проводиться з використанням реальної бази даних, за винятком деяких випадків, де застосовуються ізольовані тестові дані для запобігання впливу на продуктивне середовище;
* API задокументовано у Swagger.
  1. Ризики

Було визначено ризики та відповідні дії для пом’якшення їхнього впливу на проєкт. Вплив ризику визначається тим, як проєкт постраждає у разі його спрацювання. Тригером є те, яка віха або подія призведе до того, що ризик стане проблемою, яку потрібно вирішити (див. табл. 4.1).

Таблиця 4.1 – Таблиця ризиків

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Ризик | Вплив | Тригер | План пом’якшення |
| 1 | Зміни в контракті API під час тестування | Високий | Помилки тестів або необхідність їх переписування | Заморозити контракт API до початку етапу тестування |
| 2 | Зміни у функціоналі можуть зробити недійсними вже створені тест-кейси | Високий | Втрата раніше написаних тест-кейсів | Експортувати тестові дані до оновлення, за потреби адаптувати та імпортувати повторно |
| 3 | Обмежене тестове покриття через нестачу часу | Високий | Тиск через стислі терміни | Визначити пріоритетність і зосередити ресурси на критичних тест-кейсах |

1. Підхід до тестування

Підхід до тестування ґрунтується на вимогах і є сумісним із методологією Agile. Тестування виконується ітеративно відповідно до спринтів, що дає змогу виявляти проблеми на ранніх етапах розробки. Процес тестування передбачає поєднання ручного, автоматизованого та навантажувального тестування.

5.1 Ручне тестування

* тестування API через Swagger UI: виконується для більшості кінцевих точок з метою перевірки правильності відповіді, формату та обробки крайових випадків;
* поверхневе тестування вебінтерфейсу та мобільних застосунків: зосереджене на компонентах, які зазнали змін або є новими;
* валідація UI/UX взаємодії: базове тестування інтерфейсу з метою перевірки візуальної узгодженості та навігації у змінених частинах системи.

5.2 Автоматизоване тестування

* логіка серверної частини та валідація: автоматизується за допомогою фреймворків NUnit і Moq для перевірки бізнес-логіки та серверної обробки даних;
* інтеграційне тестування з MongoDB: охоплює сценарії взаємодії з базою даних для перевірки узгодженості та правильності збереження даних;
* повторювані UI-тести для вебплатформ: автоматизовані тести з використанням Cypress (для веб) для перевірки валідності форм, навігаційних переходів і статичних очікуваних результатів.

6. Критерії початку та завершення тестування

6.1 Критерії початку тестування

Тестування може бути розпочато лише за умови дотримання таких передумов:

* вихідний код проєкту розміщено у репозиторії GitHub;
* усі необхідні програмні засоби, включаючи інструменти тестування, успішно встановлено та перевірено на працездатність;
* тестове середовище налаштовано й готове до використання (серверна частина, база даних, мобільний емулятор);
* надано повну документацію до вимог, а також специфікацію API у форматі Swagger;
* підготовлено тестові дані відповідно до очікуваних сценаріїв;
* тестувальник повністю ознайомлена з вимогами та логікою функціонування системи.

6.2 Критерії завершення тестування

Тестування вважається завершеним, якщо виконано такі умови:

* усі критичні тест-кейси були виконані;
* не залишилося жодних відкритих дефектів із високим або блокуючим рівнем серйозності;
* досягнуто охоплення вимог тестами на рівні не менше ніж 95%;
* підготовлено та переглянуто фінальний звіт про результати тестування.

7. Критерії призупинення та відновлення тестування

7.1 Критерії призупинення тестування

До критеріїв призупинення тестування відноситься:

* невдала збірка застосунку або наявність критичних дефектів, що блокують подальше тестування;
* відсутність доступу до тестового середовища, що унеможливлює виконання запланованих перевірок;
* суттєві зміни у вимогах під час активного спринту, що призводять до необхідності перегляду наявних тест-кейсів або тестової стратегії.

7.2 Критерії відновлення тестування

До критеріїв відновлення тестування відноситься повне усунення блокуючих дефектів та стабілізації тестового середовища.

8. Стратегія тестування

Процес тестування в межах проєкту «Naviria» реалізується одним спеціалістом із забезпечення якості, який виконує повний цикл тестування – від аналізу вимог до складання підсумкової звітності.

Стратегія передбачає комплексну діяльність, спрямовану на забезпечення якості розроблюваного програмного забезпечення.

8.1 Роль тестувальника у процесі тестування

Тестувальник повинен виконати наступні дії:

* аналіз вимог: тестувальник ознайомлюється з функціональними вимогами, що надаються розробниками, зокрема у вигляді специфікацій або Swagger-документації. На основі цього виконується повне розуміння очікуваного функціоналу;
* підготовка тест-кейсів: тест-кейси формуються на основі аналізу вимог, а також результатів попереднього дослідницького тестування. Усі можливі сценарії, що охоплюють як позитивні, так і негативні випадки, документуються у структурованому вигляді;
* створення тестових даних: тестувальник самостійно генерує тестові дані відповідно до обраних сценаріїв та умов тестування, використовуючи розроблене середовище (локальне або тестове);
* виконання тестів: тестування проводиться вручну та за допомогою автоматизованих засобів (NUnit, Cypress, тощо) відповідно до тест-кейсів. Після кожного запуску фіксуються фактичні результати виконання (успішно/неуспішно) у відповідній документації;
* реєстрація дефектів: усі виявлені дефекти документуються у системі відстеження із зазначенням умов відтворення, очікуваного та фактичного результату. Про виявлені дефекти повідомляється відповідальному розробнику;
* повторне тестування і регресія: після усунення дефектів розробником виконується повторне тестування (retesting). У разі, якщо зміни можуть вплинути на інші частини системи, додатково проводиться регресійне тестування для перевірки суміжного функціоналу;
* звітність та передача результатів: після завершення всіх етапів тестування формується підсумковий звіт про якість із деталізацією тест-кейсів, покриття вимог, кількості виявлених і виправлених дефектів. Звіт зберігається локально та, за потреби, надсилається керівникові проєкту;

8.2 Життєвий цикл дефекту

Усі проблеми, виявлені під час тестування, будуть записані в документ Word.

Рисунок 8.1 ілюструє життєвий цикл дефекту в процесі тестування програмного забезпечення. Дефект проходить кілька етапів – від створення та підтвердження, через призначення розробнику, виправлення та перевірку до остаточного закриття. Якщо рішення не задовольняє QA, дефект може бути повторно відкрит для доопрацювання.

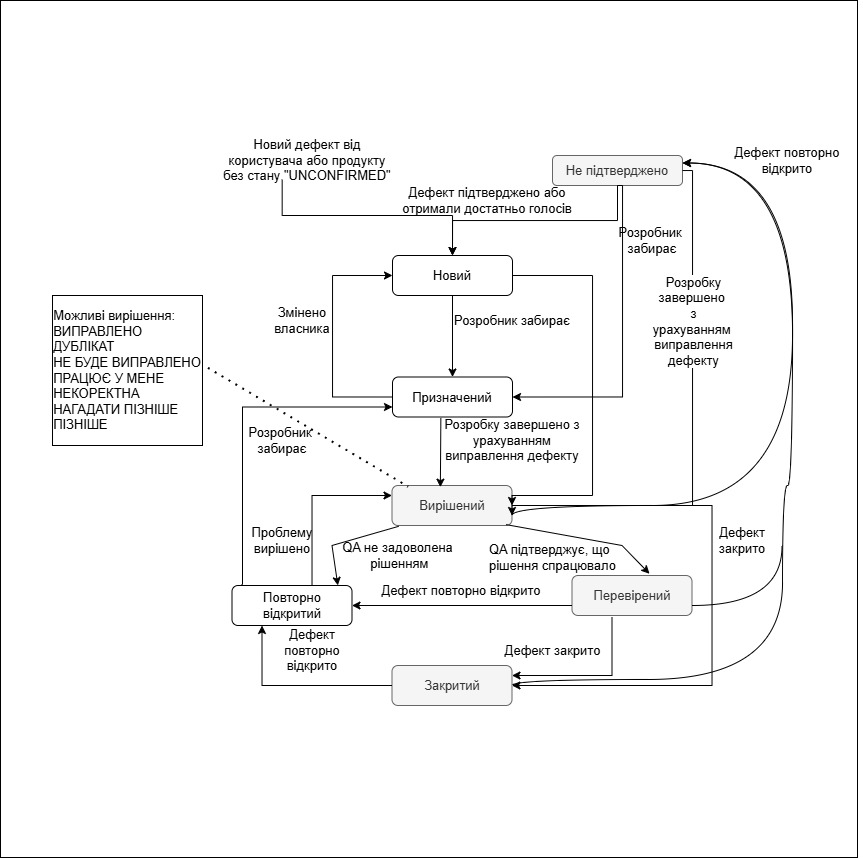


Рисунок 8.1 – Схема життєвого циклу дефекту

8.3 Типи тестування

У межах проєкту «Naviria» повинні застосовуватись декілька типів тестування, які забезпечують комплексну перевірку як функціональних, так і нефункціональних вимог до системи. Кожен з типів спрямований на виявлення дефектів на певному рівні реалізації або взаємодії компонентів системи.

Тестування «чорного ящика» (Black-box Testing) використовується для перевірки функціональності веб- та мобільного інтерфейсу з позиції кінцевого користувача, без доступу до внутрішньої логіки коду. Тестувальник перевіряє відповідність фактичної поведінки системи очікуваним результатам згідно зі специфікаціями.

Тестування «білого» та «сірого» ящика (White-box та Gray-box Testing). Для API застосовується тестування з частковим або повним знанням внутрішньої логіки системи. Це дозволяє перевірити коректність реалізації контролерів, обробку вхідних параметрів, валідацію, бізнес-логіку та обробку виняткових ситуацій на рівні коду.

Інтеграційне тестування (Integration Testing) проводиться для перевірки взаємодії між різними компонентами системи: серверною частиною (API), базою даних MongoDB, фронтендом та мобільним клієнтом. Мета – виявити помилки, що виникають при обміні даними між модулями.

Функціональне тестування (Functional Testing) спрямоване на верифікацію реалізації функціональних вимог: автентифікації, керування цілями, досягненнями, сповіщеннями, соціальними взаємодіями тощо. Тестування виконується відповідно до тест-кейсів, побудованих на основі специфікацій і сценаріїв користування.

Системне тестування (System Testing) охоплює повністю інтегровану систему та перевіряє її відповідність встановленим вимогам у межах повноцінних наскрізних сценаріїв. Наприклад: створення облікового запису, додавання цілі, отримання сповіщення, взаємодія з іншими користувачами, чат з ШІ

Регресійне тестування (Regression Testing) проводиться після внесення змін у код або виправлення дефектів. Мета – переконатися, що нововведення не порушили стабільну роботу існуючого функціоналу. Включає автоматизовані та ручні перевірки критичних користувацьких сценаріїв.

Навантажувальне тестування (Performance Testing) використовується для перевірки поведінки системи в умовах високого навантаження. За допомогою Apache JMeter тестується стійкість серверної частини й інтерфейсів при обробці великої кількості одночасних запитів, зокрема вимірюється час відповіді, швидкість обробки та стійкість до збоїв.

Приймальне тестування (User Acceptance Testing, UAT) проводиться на завершальному етапі перед випуском системи та виконується внутрішнім користувачем (представником команди), щоб оцінити загальну зручність, відповідність бізнес-вимогам і готовність системи до впровадження.

8.4 Визначення рівнів серйозності та пріоритетності дефектів

Поля серйозність (Severity) (див. табл. 8.1) та пріоритет (Priority) (див. табл. 8.2) є критично важливими для класифікації дефектів і визначення черговості їх усунення. Вони дозволяють ефективно організувати процес усунення помилок, враховуючи як технічну критичність, так і вплив на користувача або бізнес-цілі проєкту «Naviria».

Таблиця 8.1 – Класифікація серйозності дефектів

|  |  |  |
| --- | --- | --- |
| ID | Рівень серйозності | Опис |
| 1 | Критична | Дефект призводить до аварійного завершення роботи системи, втрати даних, пошкодження БД або файлів, або блокує роботу всієї системи. Необхідне негайне усунення. |
| 2 | Висока | Основна функціональність недоступна або працює некоректно. Дефект серйозно впливає на користувача або блокує тестування інших модулів. Можливе існування складного або неінтуїтивного обходу. |

Кінець таблиці 8.1

|  |  |  |
| --- | --- | --- |
| ID | Рівень серйозності | Опис |
| 3 | Середня | Некритичний дефект, який не впливає на основний функціонал, має простий обхідний шлях. Виникає у другорядних сценаріях або окремих модулях. |
| 4 | Низька | Орфографічні або візуальні помилки, недоліки в документації або вже підтверджені незначні дефекти, що не потребують термінового виправлення. |

Таблиця 8.2 – Класифікація пріоритетності дефектів

|  |  |  |
| --- | --- | --- |
| ID | Рівень пріоритету | Опис |
| 1 | Обов’язково виправити | Дефект має бути усунутий негайно. Продукт не може бути переданий користувачам у наявності цієї помилки. |
| 2 | Бажано виправити | Дефект суттєвий та повинен бути усуненим у найкоротший термін. Її наявність може негативно вплинути на репутацію продукту. |
| 3 | Виправити за можливості | Виправлення не є критичним для поточного релізу, але бажане за наявності ресурсів. Якщо виправлення не впливає на строки випуску – виконати. |

Кінець таблиці 8.2

|  |  |  |
| --- | --- | --- |
| ID | Рівень пріоритету | Опис |
| 4 | Не критично | Виправлення може бути відкладене на післярелізний період. Часто це незначні вдосконалення або додаткові функції, що виходять за межі поточних цілей. |

9. Потреби в ресурсах та середовищі тестування

9.1 Інструменти тестування

У таблиці 9.1 наведено використання інструментів під час певних процесів.

Таблиця 9.1 – Таблиця процеси та їх інструменти

|  |  |
| --- | --- |
| **Процес** | **Інструмент** |
| Розробка ручних тест-кейсів | Microsoft Excel |
| Зберігання та відстеження тест-кейсів | Microsoft Excel |
| Виконання ручних тестів | Manual Testing, Excel |
| API-тестування | Swagger UI, Postman |
| Юніт-тестування (Unit testing) | NUnit, Moq |
| Автоматизоване UI-тестування (веб) | Cypress |
| Моніторинг навантаження та памʼяті | Android Studio |
| Навантажувальне тестування | Apache JMeter |
| Відстеження дефектів | Microsoft Word |
| Формування звітів про тестування | Microsoft Word, PDF |

9.2 Управління конфігурацією

Код CM: Git

9.3 Тестове середовище

Для проведення тестування системи «Naviria» повинно бути використано мультиплатформне середовище, що охоплює сучасні браузери, емулятори, реальні мобільні пристрої та локальний сервер для запуску серверної частини застосунку.

* Операційне середовище:

1. операційна система: Windows 11 Home (x64);
2. браузери для вебтестування: Google Chrome (остання стабільна версія) та Microsoft Edge (остання стабільна версія).

* Мобільні пристрої для тестування:

1. Samsung Galaxy A52 (Android 13).

* Серверне середовище (локальна машина):

1. операційна система: Windows 11 Home;
2. процесор: AMD Ryzen 7 5700U with Radeon Graphics, 1.80 GHz;
3. оперативна памʼять: 16 GB;
4. диск: SSD 512 GB;
5. мережеве підключення: локальна мережа;
6. середовище запуску серверної частини: Visual Studio 2022;
7. API-документація: Swagger UI локально.

10. Графік тестування

У таблиці 10.1 наведено графік тестування.

Таблиця 10.1 – Графік тестування

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва етапу** | **Початок** | **Завершення** | **Коментарі** |
| Планування тестування | 08.04.2025 | 11.04.2025 | Організація процесу, уточнення обсягів |
| Аналіз вимог / Ознайомлення з документацією | 10.04.2025 | 16.04.2025 | Перегляд технічної документації |
| Перше розгортання до тестового середовища | 10.04.2025 | | Доступ до QA-середовища |
| Функціональне тестування ітерація 1 | 10.04.2025 | 21.04.2025 | Відповідно до переліку функціоналу |
| Розгортання ітерації 2 до QA-середовища | 21.04.2025 | | Нові зміни для перевірки |
| Функціональне тестування ітерація 2 | 21.04.2025 | 28.04.2025 | Розширене покриття функціоналу |
| Розгортання ітерації 3 до QA-середовища | 28.04.2025 | |  |

Продовження таблиці 10.1

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва етапу** | **Початок** | **Завершення** | **Коментарі** |
| Регресійне тестування | 28.04.2025 | 29.04.2025 |  |
| Системне тестування | 29.04.2025 | 04.05.2025 | Тестування наскрізних сценаріїв |
| Функціональне тестування ітерація 3 | 29.04.2025 | 05.05.2025 |  |
| Розгортання ітерації 4 до QA-середовища | 05.05.2025 | |  |
| Системне тестування | 05.05.2025 | 07.05.2025 |  |
| Функціональне тестування ітерація 4 | 05.05.2025 | 12.05.2025 |  |
| Розгортання ітерації 5 до QA-середовища | 12.05.2025 | |  |
| Регресійне тестування | 12.05.2025 | 14.05.2025 |  |
| Системне тестування | 12.05.2025 | 19.05.2025 |  |
| Функціональне тестування ітерація 5 | 14.05.2025 | 19.05.2025 |  |

Кінець таблиці 10.1

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва етапу** | **Початок** | | **Завершення** |
| Розгортання ітерації 6 до QA-середовища | 19.05.2025 | |  |
| Регресійне тестування | 19.05.2025 | 20.05.2025 |  |
| Функціональне тестування ітерація 6 | 20.05.2025 | 28.05.2025 |  |
| Навантажувальне тестування | 25.05.2025 | 30.05.2025 | Apache JMeter |
| Системне тестування | 25.05.2025 | 30.05.2025 |  |
| Приймальне тестування (UAT) | 26.05.2025 | 30.05.2025 | Внутрішнє тестування користувачами |
| Усунення критичних дефектів / фінальні перевірки | 27.05.2025 | 31.05.2025 | Перед стабільним релізом |

Терміни та абревіатури

|  |  |
| --- | --- |
| Термін / Абревіатура | Визначення |
| API | Application Programming Interface – інтерфейс програмування застосунків |
| QA | Quality Assurance – забезпечення якості |
| UAT | User Acceptance Testing – приймальне тестування |
| GUI | Graphical User Interface – графічний інтерфейс користувача |
| DB | Database – база даних |