Testowanie

Maciej Bajer
Pawet Janduta
Kirill Vereshchako
Yevhen Savchuk
15 June 2025

I. Wstęp

1. Cele

Celem testowania jest zapewnienie wysokiej jakości aplikacji do zarządzania napiwkami w restauracjach, która będzie działać niezawodnie i bez błędów uniemożliwiających pracę. Chcemy osiągnąć działającą, stabilną aplikację, która będzie możliwa do zastosowania w rzeczywistym środowisku pracy w restauracjach.

Testowanie projektu ma na celu sprawdzenie:

1. czy aplikacja realizuje wszystkie założenia określone w dokumentacji,
2. czy wszystkie przewidziane funkcje działają zgodnie z oczekiwaniami,
3. czy system prawidłowo reaguje na nietypowe dane wejściowe oraz nieprzewidywalne działania użytkownika,
4. czy nie występują błędy wynikające z pomyłek programistycznych,
5. czy interfejs użytkownika oraz sposób działania programu są intuicyjne i nie budzą wątpliwości,
6. czy w różnych scenariuszach użytkowania aplikacja zachowuje się w sposób logiczny i przewidywalny.

2. Plan testów

Proces testowania rozpoczęliśmy od dokładnego sprawdzenia poszczególnych komponentów aplikacji za pomocą testów jednostkowych. Ich celem było upewnienie się, że każda funkcja i klasa działa poprawnie w izolacji, zgodnie z wymaganiami. Po zakończeniu testów jednostkowych przeszliśmy do testów systemowych, które objęły całą aplikację jako całość. Dzięki nim mogliśmy zweryfikować poprawność współpracy między komponentami oraz ocenić zachowanie systemu w rzeczywistych scenariuszach użytkowania.

II. Testy jednostkowe

1. Wprowadzenie

Testy jednostkowe zostały napisane w języku C++ z wykorzystaniem frameworka Qt Test, który jest integralną częścią środowiska Qt i umożliwia łatwe tworzenie oraz uruchamianie testów automatycznych. Każda testowana klasa otrzymała własny zestaw testów, w których sprawdzano poprawność działania jej metod w różnych warunkach, zarówno typowych, jak i skrajnych. Dla każdej klasy testowej utworzono osobny plik z klasą testującą dziedziczącą po QObject, a poszczególne przypadki testowe zdefiniowano jako prywatne sloty oznaczone makrem Q_SLOT lub private slots:.

Do wykonywania testów wykorzystywaliśmy makra dostarczane przez Qt, takie jak QCOMPARE(), QVERIFY(), QFAIL(), czy QTEST(), które pozwalają w przejrzysty sposób sprawdzać oczekiwane wyniki i zgłaszać błędy w przypadku niezgodności. Testy uruchamialiśmy lokalnie w środowisku Qt Creator, co pozwalało na bieżące monitorowanie wyników i szybkie wykrywanie błędów. Testy były uruchamiane w ramach oddzielnych plików wykonywalnych, które można łatwo kompilować i uruchamiać, dzięki czemu można było skupić się na poszczególnych komponentach systemu.

Przy testowaniu wykorzystywano podejście white-box, co oznacza, że testerzy mieli pełen wgląd w strukturę i implementację testowanych klas. Pozwoliło to na szczegółowe pokrycie kodu testami. Dla klas o kluczowym znaczeniu, takich jak UserManager, TipManager, DatabaseService, czy BackupService, stworzono wiele przypadków testowych obejmujących różne scenariusze użycia i błędne dane wejściowe.

Kod testów jednostkowych został przechowywany w repozytorium Git, w osobnym folderze tests, a każda zmiana wprowadzana do głównego kodu aplikacji była poprzedzana jego weryfikacją i uruchomieniem testów. Dzięki temu mogliśmy szybko wykrywać potencjalne regresje oraz zapewnić większą stabilność i niezawodność aplikacji na dalszych etapach rozwoju.

2. Zaimplementowane testy:

BackupServiceTests:

PASS : BackupServiceTest::initTestCase()
PASS : BackupServiceTest::testCreateBackup_successful()
PASS : BackupServiceTest::testCreateBackup_emptyPath()
PASS : BackupServiceTest::testCreateBackup_generatesUniqueFile()
PASS : BackupServiceTest::testBackupContentMatchesOriginal()
PASS : BackupServiceTest::testCreateBackup_invalidDestination()
PASS : BackupServiceTest::testCreateBackup_filenameFormat()
PASS : BackupServiceTest::testCreateBackup_notEmptyFile()
PASS : BackupServiceTest::cleanupTestCase()
Totals: 9 passed, 0 failed, 0 skipped, 0 blacklisted, 1042ms

BalanceCalculatorTests:

PASS : BalanceCalculatorTest::initTestCase()
PASS : BalanceCalculatorTest::testGetBalanceInRange(waiter1_full_access)
PASS : BalanceCalculatorTest::testGetBalanceInRange(waiter2_full_access)
PASS : BalanceCalculatorTest::testGetBalanceInRange(waiter1_blocked_from_waiter2)
PASS : BalanceCalculatorTest::testGetBalanceInRange(manager_for_waiter1)
PASS : BalanceCalculatorTest::testGetBalanceInRange(manager_for_waiter2)
PASS : BalanceCalculatorTest::testGetBalanceInRange(empty_date_range)
PASS : BalanceCalculatorTest::testGetTodayBalance_accessControl()
PASS : BalanceCalculatorTest::testGetMonthlyBalance()
PASS : BalanceCalculatorTest::cleanupTestCase()
Totals: 10 passed, 0 failed, 0 skipped, 0 blacklisted, 27ms

DatabaseServiceTest:

PASS : DatabaseServiceTest::initTestCase()
PASS : DatabaseServiceTest::test_ConnectToNewFile_ShouldCreateDb()
PASS : DatabaseServiceTest::test_ConnectToMemoryDatabase()
PASS : DatabaseServiceTest::test_ConnectToInvalidPath_ShouldFail()
PASS : DatabaseServiceTest::test_ConnectionOpensDatabase()
PASS : DatabaseServiceTest::test_ConnectToReadOnlyLocation_ShouldFail()

PASS : DatabaseServiceTest::test_FileActuallyCreated()
PASS : DatabaseServiceTest::test_ReconnectDoesNotDropTables()
PASS : DatabaseServiceTest::test_DefaultPathUsedIfNoneProvided()
PASS : DatabaseServiceTest::test_DatabaseConnectionIsValid()
PASS : DatabaseServiceTest::cleanupTestCase()
Totals: 12 passed, 0 failed, 0 skipped, 0 blacklisted, 42ms

PerformanceServiceTests:

PASS : PerformanceServiceTest::initTestCase()
PASS : PerformanceServiceTest::testManagerCanAccessAnyWaiter()
PASS : PerformanceServiceTest::testWaiterCanAccessSelf()
PASS : PerformanceServiceTest::testWaiterCannotAccessOthers()
PASS : PerformanceServiceTest::testUnauthorizedRoleDenied()
PASS : PerformanceServiceTest::testRepositoryCalledWithCorrectDates()
PASS : PerformanceServiceTest::cleanupTestCase()
Totals: 7 passed, 0 failed, 0 skipped, 0 blacklisted, 6ms

ReportGeneratorTests:

PASS : ReportGeneratorTest::initTestCase()
PASS : ReportGeneratorTest::test_generateReports_validData()
PASS : ReportGeneratorTest::test_generateReports_noTipsInRange()
PASS : ReportGeneratorTest::test_generateReports_invalidOutputDir()
PASS : ReportGeneratorTest::cleanupTestCase()
Totals: 5 passed, 0 failed, 0 skipped, 0 blacklisted, 93ms

SqlTipMetricsRepositoryTests:

PASS : SqlTipMetricsRepositoryTest::initTestCase()
PASS : SqlTipMetricsRepositoryTest::test_SingleTip_FullAmountToWaiter1()
PASS : SqlTipMetricsRepositoryTest::test_SingleTip_SplitBetweenWaiters()
PASS : SqlTipMetricsRepositoryTest::test_MixedTips_MultipleCalculations()
PASS : SqlTipMetricsRepositoryTest::test_NoTips_ReturnsZero()
PASS : SqlTipMetricsRepositoryTest::test_TipAssignedOnlyAsWaiter2()
PASS : SqlTipMetricsRepositoryTest::test_TipsOutsideDateRange_Ignored()
PASS : SqlTipMetricsRepositoryTest::test_NullSecondWaiterCountedCorrectly()

PASS : SqlTipMetricsRepositoryTest::test_NonParticipantWaiter_ZeroMetrics()
PASS : SqlTipMetricsRepositoryTest::test_WaiterAsBothWaiters()
PASS : SqlTipMetricsRepositoryTest::test_MassiveTipsCount()
PASS : SqlTipMetricsRepositoryTest::test_AverageAmountCalculationPrecision()
PASS : SqlTipMetricsRepositoryTest::test_BrokenTable_ReturnsUnsuccessful()
PASS : SqlTipMetricsRepositoryTest::cleanupTestCase()
Totals: 14 passed, 0 failed, 0 skipped, 0 blacklisted, 183ms

TipManagerTests:

PASS : TipManagerTest::initTestCase()
PASS : TipManagerTest::testAddTip_waiterOnly()
PASS : TipManagerTest::testAddTip_withSecondWaiter()
PASS : TipManagerTest::testAddTip_zeroAmount()
PASS : TipManagerTest::testAddTip_negativeAmount()
PASS : TipManagerTest::testAddTip_largeAmount()
PASS : TipManagerTest::testAddTip_checkDate()
PASS : TipManagerTest::testAddTip_nullSecondWaiterExplicit()
PASS : TipManagerTest::testAddTip_multipleTimes()
PASS : TipManagerTest::testAddTip_extremelyLargeAmount()
PASS : TipManagerTest::testAddTip_floatingPointPrecision()
PASS : TipManagerTest::testAddTip_invalidWaiterId()
PASS : TipManagerTest::testAddTip_withSameWaiters()
PASS : TipManagerTest::testAddTip_failsOnMissingTable()
PASS : TipManagerTest::test_NoTips_ReturnsFalse()
PASS : TipManagerTest::test_ManagerCanCorrectLastTip()
PASS : TipManagerTest::test_WaiterCanCorrectOwnTip()
PASS : TipManagerTest::test_WaiterCannotCorrectOthersTip()
PASS : TipManagerTest::test_InvalidRoleDenied()
PASS : TipManagerTest::test_UpdateFails_ReturnsFalse()
PASS : TipManagerTest::cleanupTestCase()
Totals: 21 passed, 0 failed, 0 skipped, 0 blacklisted, 41ms

UserManagerTests:

PASS : UserManagerTest::initTestCase()
PASS : UserManagerTest::testHashPassword()

PASS : UserManagerTest::testLogin(alice correct)
PASS : UserManagerTest::testLogin(admin correct)
PASS : UserManagerTest::testLogin(kelner correct)
PASS : UserManagerTest::testLogin(friend correct)
PASS : UserManagerTest::testLogin(uppercase correct)
PASS : UserManagerTest::testLogin(emoji correct)
PASS : UserManagerTest::testLogin(alice wrong password)
PASS : UserManagerTest::testLogin(admin wrong)
PASS : UserManagerTest::testLogin(unknown user)
PASS : UserManagerTest::testLogin(alise with space)
PASS : UserManagerTest::testLogin(space with alice)
PASS : UserManagerTest::testLogin(sql inject username)
PASS : UserManagerTest::testLogin(sql inject password)
PASS : UserManagerTest::testLogin(sql inject both)
PASS : UserManagerTest::testLogin(empty username)
PASS : UserManagerTest::testLogin(empty password)
PASS : UserManagerTest::testLogin(empty both)
PASS : UserManagerTest::testLogin(case mismatch)
PASS : UserManagerTest::testLogin(case mismatch password)
PASS : UserManagerTest::testLogin(long username)
PASS : UserManagerTest::testLogin(long password)
PASS : UserManagerTest::testLogin(long both)
PASS : UserManagerTest::testGetRole(manager user)
PASS : UserManagerTest::testGetRole(admin user)
PASS : UserManagerTest::testGetRole(kelner user)
PASS : UserManagerTest::testGetRole(friend user)
PASS : UserManagerTest::testGetRole(emoji user)
PASS : UserManagerTest::testGetRole(uppercase name)
PASS : UserManagerTest::testGetRole(nonexistent)
PASS : UserManagerTest::testGetRole(empty login)
PASS : UserManagerTest::testGetRole(wrong case)
PASS : UserManagerTest::testGetRole(injected input)
PASS : UserManagerTest::testGetRole(long input)
PASS : UserManagerTest::testGetUserId(valid alice)
PASS : UserManagerTest::testGetUserId(valid admin)
PASS : UserManagerTest::testGetUserId(valid kelner1)
PASS : UserManagerTest::testGetUserId(unicode name)
PASS : UserManagerTest::testGetUserId(emoji name)
PASS : UserManagerTest::testGetUserId(nonexistent user)

PASS : UserManagerTest::testGetUserId(nonexistent user with space)
PASS : UserManagerTest::testGetUserId(nonexistent user with space at start)
PASS : UserManagerTest::testGetUserId(empty username)
PASS : UserManagerTest::testGetUserId(empty username with space)
PASS : UserManagerTest::testGetUserId(wrong case)
PASS : UserManagerTest::testGetUserId(sql injection)
PASS : UserManagerTest::testGetUserId(long input)
PASS : UserManagerTest::cleanupTestCase()
Totals: 49 passed, 0 failed, 0 skipped, 0 blacklisted, 38ms

UserRepositoryTests:

PASS : UserRepositoryTest::initTestCase()
PASS : UserRepositoryTest::test_insertUser_validRoles()
PASS : UserRepositoryTest::test_insertUser_missingFields()
PASS : UserRepositoryTest::test_insertUser_invalidRoles()
PASS : UserRepositoryTest::test_insertUser_duplicateUsername()
PASS : UserRepositoryTest::test_deleteUser_valid()
PASS : UserRepositoryTest::test_deleteUser_nonExistent()
PASS : UserRepositoryTest::test_deleteUser_invalidId()
PASS : UserRepositoryTest::cleanupTestCase()
Totals: 9 passed, 0 failed, 0 skipped, 0 blacklisted, 26ms

III. Testy systemowe

W tej części dokumentacji przedstawiono strategię, zakres, harmonogram i zasoby wymagane do przeprowadzenia testów systemowych aplikacji. Celem jest weryfikacja kluczowych funkcji oprogramowania do zarządzania napiwkami, a także sprawdzenie jego wydajności, bezpieczeństwa i użyteczności

System testowany: Aplikacja System Zarządzania Napiwkami v. 1.0 uruchamiana w środowisku Windows 11 Pro

Tworzenie przypadków testowych:

Testy funkcjonalne:

Test 1: Logowanie

Kroki: 1. Wprowadzamy poprawny login hasło. 2. Klikamy "Zaloguj".

Oczekiwany rezultat: Pomyślne zalogowanie do systemu i wyświetlenie głównego interfejsu.

Rezultat: Zgodny z oczekiwanym rezultatem SUKCES

Test 2: Standardowy podział napiwku bez dzielenia

Kroki: 1. Klikamy pole ,,Dodaj napiwek", wpisujemy w ,,Kwota napiwku:" kwotę 42,02 i klikamy OK, a następnie sprawdzamy wynik klikając w ,,Sprawdź swoje saldo". Ten krok powtarzamy jedynie zmieniając kwotę napiwku na 62,87

Oczekiwany rezultat: W ,,Sprawdź swoje saldo" najpierw pojawi się kwota 42,02, a następnie 104,89.

Rezultat: Zgodny z oczekiwanym rezultatem SUKCES

Test 3: Użycie opcji ,,ID drugiego kelnera"

Kroki: Klikamy pole ,,Dodaj napiwek" wpisujemy w ,,Kwota napiwku:" kwotę 57,02, a w ,,ID drugiego kelnera" wpisujemy ID 2 – innego kelnera którego stworzyliśmy na potrzebę testu

Oczekiwany rezultat: W ,,Sprawdź swoje saldo" kelnera, w którym testujemy pojawi się kwota 28,51, a u kelnera o ID 2 również 28,51.

Rezultat: Zgodny z oczekiwanym rezultatem SUKCES

Test 4: Użycie opcji ,,Edytuj ostatni napiwek"

Kroki: 1. Dodajemy napiwek o wartości 50,00. 2. Sprawdzamy saldo (powinno wynosić 50,00). 3. Klikamy opcję "Edytuj ostatni napiwek" i zmieniamy kwotę na 40,00. 4. Sprawdamy ponownie saldo.

Oczekiwany rezultat: Saldo po edycji powinno wynosić 40,00.

Rezultat: Zgodny z oczekiwanym rezultatem SUKCES

Test 5: Próba wprowadzenia niepoprawnych danych.

Kroki: 1. W polu "Dodaj napiwek" w kwocie spróbujemy wpisać tekst (np. "abc"). 2. Spróbujemy dodać napiwek z ujemną kwotą (np. -10,00). 3. Spróbujemy podzielić napiwek z ID kelnera, które nie istnieje.

Oczekiwany rezultat: System nie powinien pozwolić na wpisywanie danych dla każdego z kroków i nie powinien pozwolić na dodanie niepoprawnego wpisu. Saldo nie powinno ulec zmianie.

Rezultat: Zgodny z oczekiwanym rezultatem SUKCES

Test 6: Próba wygenerowania kopii zapasowej.

Kroki: 1. Po zalogowaniu jako manager klikamy ,,Utwórz kopię zapasową". Następnie wybieramy lokalizacje pliku do zapisania i klikamy ,,Wybierz folder".

Oczekiwany rezultat: Pomyślnie utworzono kopię zapasową

Rezultat: Komunikat ,,Nie udało się utworzyć kopii zapasowej" PORAŻKA

Testy wydajnościowe:

Test 1: Dodanie wielu nowych osób do bazy danych:

Kroki: 1. Po zalogowani jako admin dodajemy bardzo dużo nowych użytkowników do bazy danych. Następnie testujemy działanie aplikacji, logujemy się na konta różnych użytkowników i sprawdzamy, czy wszystko działa bezproblemowo.

Oczekiwany rezultat: Program działa bez żadnych problemów

Rezultat: Zgodny z oczekiwanym rezultatem SUKCES

T**est 2: Czas logowania dla użytkownika przy bazie z 10 rekordami wynosi 0.5s.**

Kroki: 1. Dodajemy do bazy danych 10 użytkowników, a następnie logujemy się na każde pojedyncze konto, czas logowania nie powinien przekroczyć 0.5s.

Oczekiwany rezultat: Czas nie przekroczy 0,5s

Rezultat: Zgodny z oczekiwanym rezultatem SUKCES

Testy bezpieczeństwa:

Test 1: Próba wykradnięcia haseł z bazy danych

Kroki: 1. Spróbujemy otworzyć bazę danych oraz ,,podejrzymy” hasła

Oczekiwany rezultat: Hasła, jako, że są haszowane wyświetlą się zakodowane

Rezultat: Zgodny z oczekiwanym rezultatem SUKCES

Testy użyteczności:

Test 1: Ocena intuicyjności interfejsu

Kroki: 1. Poprosimy nowego użytkownika (który nie zna aplikacji) o wykonanie podstawowych zadań: zalogowanie, dodanie napiwku, podzielenie go i sprawdzenie salda.

Oczekiwany rezultat: Użytkownik jest w stanie wykonać wszystkie zadania bez potrzeby zaglądania do instrukcji. Interfejs jest oceniany jako "łatwy" lub "intuicyjny".

Rezultat: Zgodny z oczekiwanym rezultatem SUKCES

Testy odporności:

Test 1: Odporność na nieoczekiwane zamknięcie

Kroki: 1. W trakcie dodawania nowego napiwku (po wpisaniu kwoty, ale przed kliknięciem "OK") zamkniemy aplikację w sposób nieoczekiwany (np. za pomocą Menedżera Zadań). 2. Uruchomimy aplikację ponownie.

Oczekiwany rezultat: Aplikacja uruchamia się poprawnie. Niezatwierdzony napiwek nie został dodany do salda. Nie ma żadnych uszkodzeń danych.

Rezultat: Zgodny z oczekiwanym rezultatem SUKCES

Testy zgodności:

Test 1: Działanie na różnych rozdzielczościach ekranu

Kroki: 1. Uruchomimy aplikację na systemie Windows 11 Pro. 2. Zmienimy rozdzielczość ekranu na najpopularniejsze wartości (np. 1920x1080, 1366x768).

Oczekiwany rezultat: Interfejs użytkownika skaluje się poprawnie, wszystkie elementy są widoczne i funkcjonalne przy każdej rozdzielczości.

Rezultat: Niezgodny z oczekiwanym rezultatem PORAŻKA

Testy dokumentacji:

Test 1: Weryfikacja zgodności dokumentacji z aplikacją

Kroki: 1. Przejrzymy dostępną dokumentację użytkownik. 2. Krok po kroku wykonamy opisane w niej funkcje.

Oczekiwany rezultat: Wszystkie opisane funkcje działają w aplikacji zgodnie z opisem w dokumentacji. Brak jest nieudokumentowanych funkcji.

Rezultat: Zgodny z oczekiwanym rezultatem SUKCES

Raportowanie:
Po zakończeniu wszystkich testów zostanie sporządzony raport końcowy. Raport będzie zawierał podsumowanie wyników dla każdej kategorii testów, listę znalezionych błędów wraz z ich priorytetem (krytyczny, wysoki, średni, niski) oraz rekomendacje dotyczące ewentualnych poprawek przed wdrożeniem oprogramowania.

Raport podsumowujący testy:

Test 6: Próba wygenerowania kopii zapasowej. – Problem z hardcoded path to a database, został naprawiony poprzez zmianę adresu ścieżki na „tips.db”

Po naprawie testu przeprowadzono testy regresji (każdy test sprawdzony z osobna) i program przeszedł pozytywnie te same testy co wcześniej + Test 6.