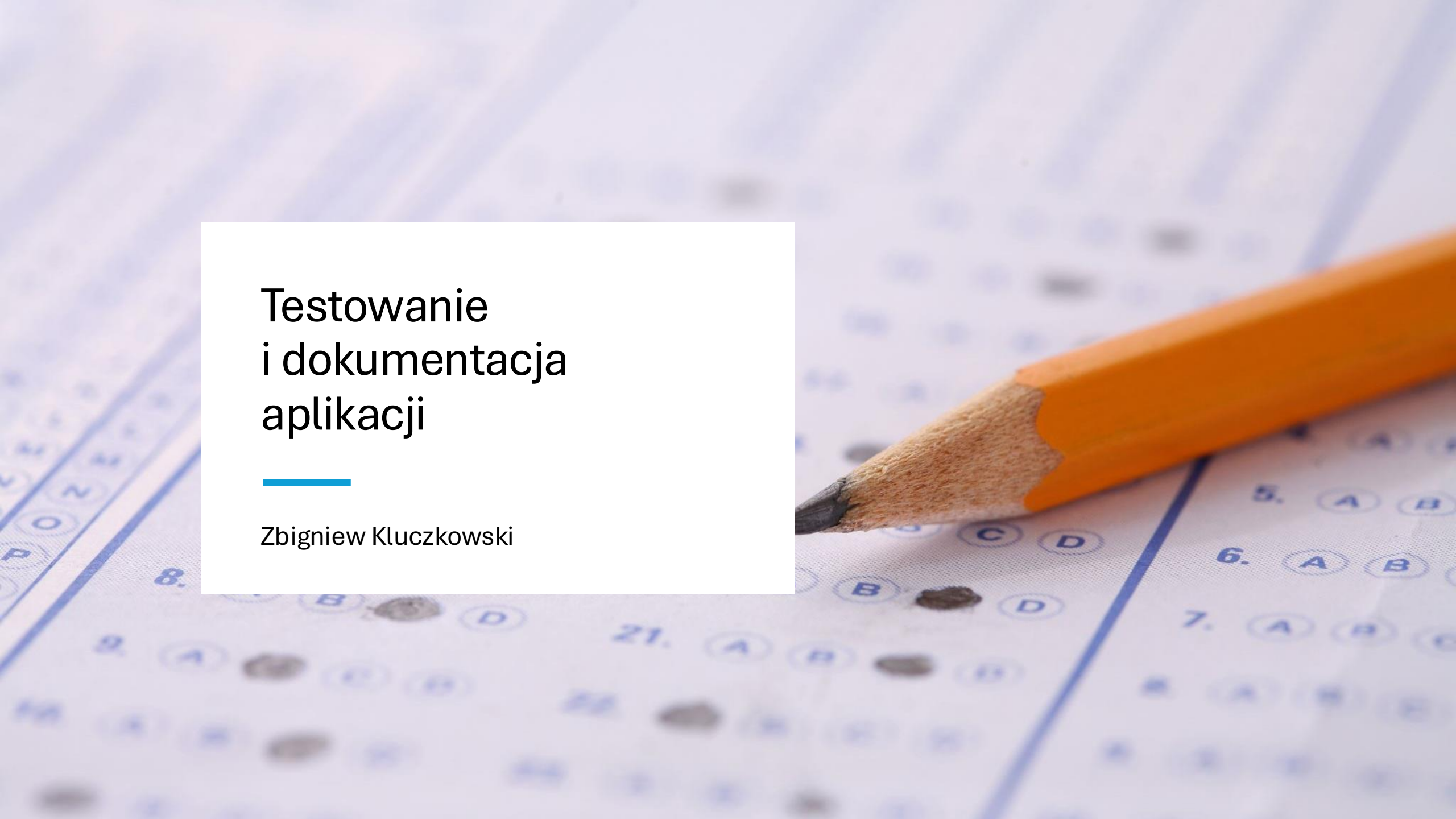
Testowanie i dokumentacja aplikacji

Zbigniew Kluczkowski



Spis treści

Wprowadzenie

1

Testowanie aplikacji to systematyczny proces sprawdzania, czy oprogramowanie działa zgodnie z założeniami oraz jest wolne od błędów, które mogą powodować nieprawidłowe działanie lub awarie.

2

Dokumentacja aplikacji to zorganizowany zbiór informacji o aplikacji — jak jest zbudowana, jak ją użytkować, jak ją rozwijać i naprawiać. To „mapa" dla programistów, testerów i użytkowników.

3

Przykład:
W dużych projektach informatycznych bez testowania aplikacji często wdrażane są wersje z poważnymi błędami, które zniechęcają użytkowników. Bez dokumentacji trudno utrzymać i rozwijać aplikację, zwłaszcza gdy zmienia się zespół.

Dlaczego to takie ważne?

Stabilność i bezpieczeństwo:
Testowanie zapobiega awariom i podatnościom na ataki (np. SQL Injection).

Wykrywanie błędów:
Błędy w kodzie mogą powodować np. utratę danych, błędne działanie funkcji, problemy z kompatybilnością.

Użyteczność:
Testy pozwalają sprawdzić, czy interfejs jest intuicyjny i czy aplikacja spełnia oczekiwania użytkowników.

Oszczędność kosztów:
Naprawa błędów w fazie testów jest wielokrotnie tańsza niż po wdrożeniu na produkcję.

Przykład:
Facebook regularnie testuje swoje funkcje przed wdrożeniem, aby zapobiec awariom, które mogłyby wpłynąć na miliony użytkowników.

Rodzaje testów

- **Testy jednostkowe (Unit tests)**

  Testują najmniejsze fragmenty kodu, np. pojedynczą funkcję.
  Przykład: Sprawdzenie, czy funkcja obliczPodatek(kwota) zwraca prawidłową wartość.

- **Testy integracyjne**

  Sprawdzają, czy różne moduły działają poprawnie razem.
  Przykład: Testowanie współpracy modułu logowania z modułem bazy danych.

- **Testy systemowe**

  Testują całą aplikację jako całość, łącznie z interfejsem użytkownika.
  Przykład: Sprawdzenie, czy użytkownik może zarejestrować się, zalogować i dokonać zakupu w aplikacji.

- **Testy akceptacyjne (UAT)**

  Wykonywane przez użytkowników końcowych, by sprawdzić, czy aplikacja spełnia ich potrzeby.
  Przykład: Testy beta nowej wersji aplikacji bankowej, gdzie klienci zgłaszają uwagi.

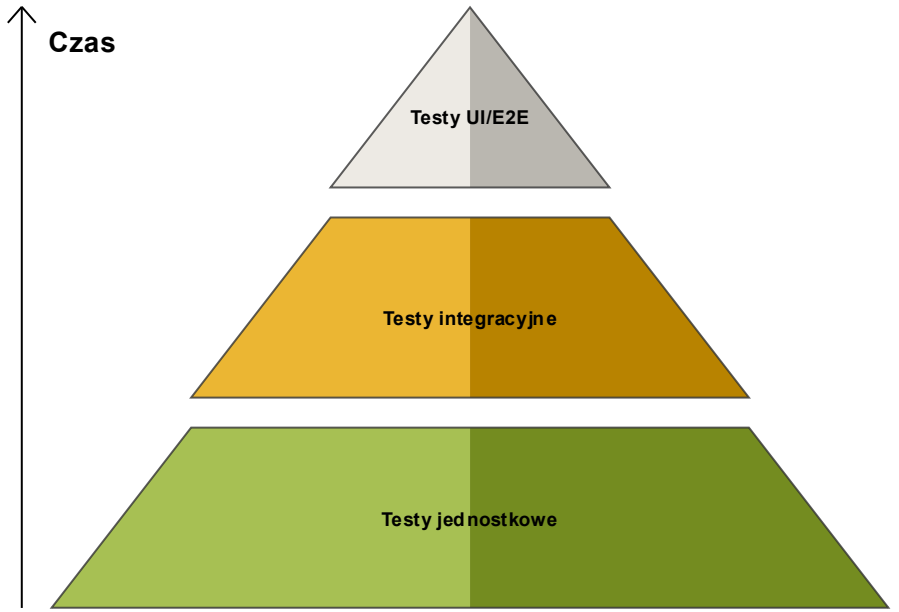
- **Testy regresyjne**

  Po każdej zmianie w kodzie testują, czy nic innego nie zostało zepsute.
  Przykład: Po dodaniu nowej funkcji koszyka na stronie e-commerce, testy regresyjne sprawdzają, czy nie popsuto procesu płatności.

Test jednostkowy

```
# Przykład funkcji do przetestowania
def oblicz_podatek(kwota):
    if kwota <= 0:
        return 0
    return kwota * 0.19  # 19% VAT

# Test jednostkowy z użyciem unittest
import unittest

class TestPodatek(unittest.TestCase):
    def test_podatek_dla_100(self):
        self.assertEqual(oblicz_podatek(100), 19)
    def test_podatek_dla_0(self):
        self.assertEqual(oblicz_podatek(0), 0)
    def test_podatek_dla_ujemnej(self):
        self.assertEqual(oblicz_podatek(-50), 0)

if __name__ == '__main__':
    unittest.main()
```

Dla pojedynczej metody lub funkcji. Test w formie klasy, która sprawdza poprawność napisanej funkcji.

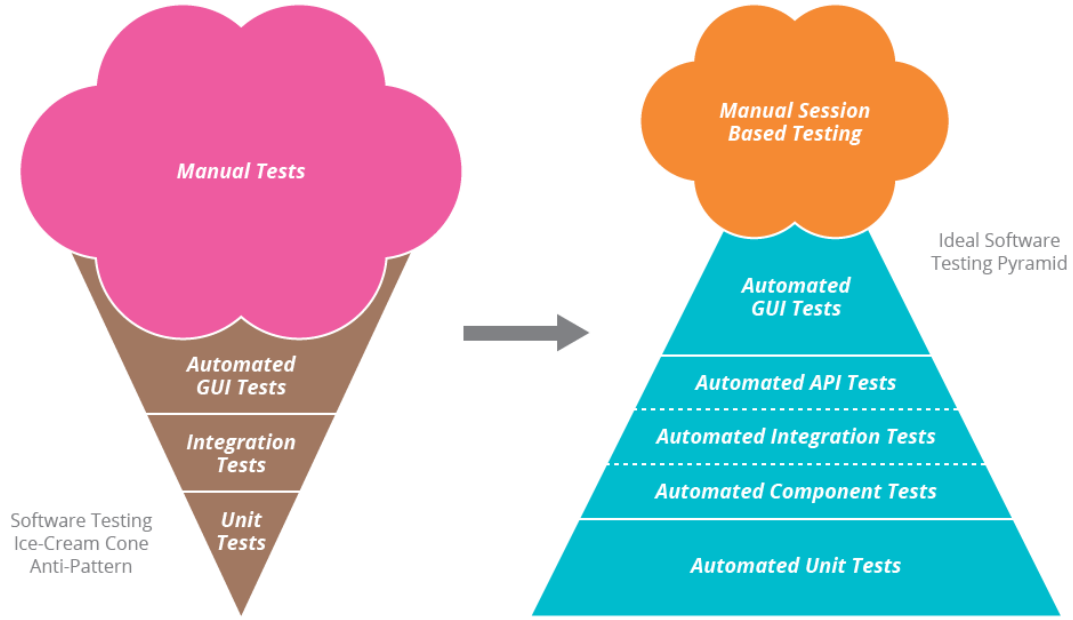


Piramida testów

Testy integracyjne

```
# Przykład uproszczonej integracji z bazą danych (symulacja)

class BazaDanych:
    def pobierz_cene(self, produkt_id):
        return 100  # symulacja pobrania ceny z bazy

def oblicz_cene_z_podatkiem(produkt_id, baza):
    cena = baza.pobierz_cene(produkt_id)
    return cena + oblicz_podatek(cena)

import unittest
class TestIntegracja(unittest.TestCase):
    def test_cena_z_podatkiem(self):
        baza = BazaDanych()
        wynik = oblicz_cene_z_podatkiem(1, baza)
        self.assertEqual(wynik, 119)

if __name__ == '__main__':
    unittest.main()
```

Sprawdza czy kod jest powiązany z innymi zasobami, np. z bazą danych.



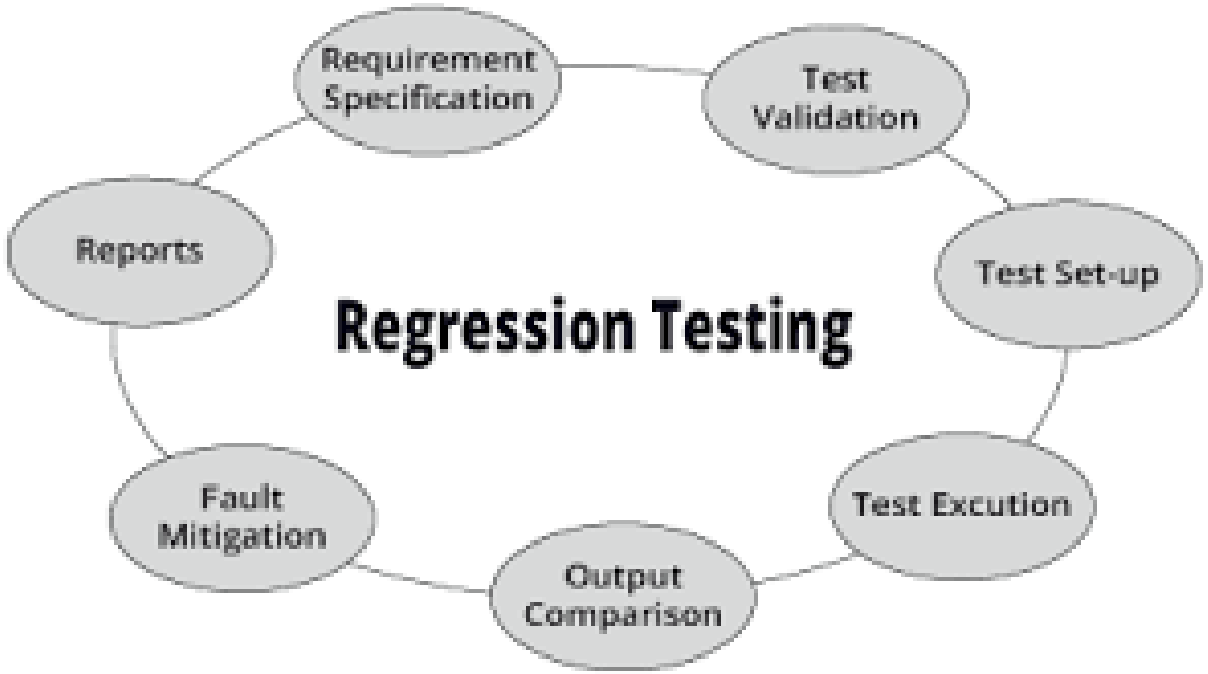
Testy systemowe

```
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.get("https://example.com/login")

# Wypełnij pola formularza i zaloguj się
driver.find_element(By.ID, "username").send_keys("testuser")
driver.find_element(By.ID, "password").send_keys("password123")
driver.find_element(By.ID, "login-button").click()

# Sprawdź, czy po zalogowaniu pojawiła się strona użytkownika
assert "Dashboard" in driver.title

driver.quit()
```

Dotyczą obsługi interfejsu (UI) – np. Selenium w Python.

Testy regresyjne

```
# Załóżmy, że mamy zestaw testów jednostkowych w pliku testy.py
# Testy regresyjne to po prostu ich ponowne uruchomienie

import unittest

def uruchom_testy_regresyjne():
    loader = unittest.TestLoader()
    suite = loader.discover('.')
    runner = unittest.TextTestRunner()
    runner.run(suite)

if __name__ == "__main__":
    uruchom_testy_regresyjne()
```



Testy bezpieczeństwa

1. Testy podatności (Vulnerability Assessment)

Automatyczne lub ręczne skanowanie aplikacji w poszukiwaniu znanych luk bezpieczeństwa.

Wykrywanie problemów takich jak brak aktualizacji bibliotek, błędy w konfiguracji serwera, niepoprawne ustawienia nagłówków HTTP.

2. Testy penetracyjne (Penetration Testing)

Symulacja rzeczywistego ataku na aplikację.

Sprawdzenie, czy można uzyskać dostęp do danych, ominąć mechanizmy logowania, przejąć sesję użytkownika itp.

Wykonywane ręcznie przez pentesterów lub półautomatycznie.

3. Testy funkcjonalne bezpieczeństwa

Weryfikacja mechanizmów uwierzytelniania i autoryzacji (np. czy użytkownik z niższymi uprawnieniami nie może dostać się do danych administratora).

Sprawdzenie poprawności przechowywania i szyfrowania haseł.

Kontrola nadawania ról i uprawnień w systemie.

4. Testy odporności na ataki webowe

Najczęściej zgodne z listą **OWASP Top 10**, np.:

SQL Injection (wstrzykiwanie zapytań do bazy danych),

XSS (Cross-Site Scripting),

CSRF (Cross-Site Request Forgery),

brak walidacji danych wejściowych,

ataki typu brute force na logowanie.

5. Testy bezpieczeństwa danych

Czy dane wrażliwe (np. hasła, numery kart, PESEL) są szyfrowane i bezpiecznie przesyłane (np. HTTPS, TLS).

Analiza logów – czy nie wyciekają w nich poufne informacje.

6. Testy wydajności i odporności (Security Stress Testing)

Sprawdzenie, jak aplikacja zachowuje się przy próbie ataku typu DoS/DDoS.

Testy ograniczeń liczby prób logowania.

Typowa lista do sprawdzenia

1. Uwierzytelnianie (Authentication)

Hasła są przechowywane z użyciem **hashowania + soli** (np. bcrypt, Argon2).

Hasła mają wymogi co do długości i złożoności.

Istnieje mechanizm blokady konta / CAPTCHA po wielu nieudanych próbach logowania.

Sesje wygasają po określonym czasie nieaktywności.

Logowanie odbywa się zawsze przez **HTTPS**.

2. Autoryzacja (Authorization)

Użytkownik nie ma dostępu do danych i funkcji, do których nie ma uprawnień (test tzw. **Privilege Escalation**).

Role i uprawnienia są jasno zdefiniowane.

API i endpointy są chronione – nawet jeśli nie są wyświetlane w interfejsie.

3. Walidacja i filtrowanie danych (Input Validation)

Wszystkie dane wejściowe są walidowane po stronie serwera.

Aplikacja jest odporna na **SQL Injection**.

Aplikacja jest odporna na **XSS** (Cross-Site Scripting).

Dane przesyłane przez formularze i API są ograniczane do dozwolonych wartości.

4. Bezpieczeństwo komunikacji

Wymuszony jest protokół **HTTPS / TLS 1.2+**.

Certyfikat SSL jest poprawny i aktualny.

Nagłówki bezpieczeństwa (np. Content-Security-Policy, X-Frame-Options, Strict-Transport-Security) są ustawione.

5. Przechowywanie danych

Dane wrażliwe (PESEL, numery kart, adresy) są szyfrowane w bazie.

Kopie zapasowe są szyfrowane i chronione.

Logi nie zawierają haseł ani danych poufnych.

6. Sesje i ciasteczka

Tokeny sesji są trudne do odgadnięcia i generowane losowo.

Ciasteczka mają ustawione flagi HttpOnly, Secure i SameSite.

Sesje wygasają poprawnie po wylogowaniu.

7. Odporność na ataki

Ochrona przed **CSRF** (tokeny, nagłówki).

Mechanizmy rate limiting / throttling (np. ograniczenie liczby zapytań).

Aplikacja jest odporna na DoS/DDoS (np. testy obciążeniowe).

8. Aktualizacje i konfiguracja

Wszystkie biblioteki i frameworki są **aktualne**.

Nie ma włączonych zbędnych portów i usług.

Tryb debugowania/logowania jest wyłączony w środowisku produkcyjnym.

9. Testy mobilne (jeśli aplikacja mobilna)

Brak przechowywania haseł w pamięci urządzenia w formie jawnej.

Brak wrażliwych danych w logach systemowych.

API jest zabezpieczone (tokeny, OAuth2).

Aplikacja weryfikuje certyfikat serwera (SSL Pinning).

10. Testy końcowe (penetracyjne)

Symulacja prób ataku SQL Injection.

Symulacja prób XSS.

Próby obejścia logowania.

Test przejęcia sesji.

Próby dostępu do API bez autoryzacji.

Biblioteki do testowania

Python:

- **unittest –** wbudowana biblioteka do testów jednostkowych.
- **pytest –** popularny framework testowy, prostszy i bardziej elastyczny niż unittest.
- **nose2 –** rozszerzenie unittest, dodatkowe raporty i pluginy.
- **doctest –** pozwala pisać testy w docstringach funkcji.
- **hypothesis –** testy oparte na generowaniu danych wejściowych.
- **selenium –** testowanie aplikacji webowych (automatyzacja przeglądarki).
- **requests + responses –** testowanie zapytań HTTP i API.

Biblioteki do testowania

Java:

- **JUnit** – podstawowa biblioteka do testów jednostkowych.
- **TestNG** – alternatywa dla JUnit z bardziej rozbudowanymi funkcjami.
- **Mockito** – biblioteka do tworzenia atrap (mocków) w testach.
- **Espresso** – testy UI aplikacji Android.

JavaScript / TypeScript:

- **Jest** – popularny framework testowy dla frontendu i backendu.
- **Mocha** + **Chai** – testy jednostkowe i asercje.
- **Cypress** – testy end-to-end aplikacji webowych.
- **Puppeteer / Playwright** – automatyzacja i testowanie interfejsu w Chrome/Edge.

Biblioteki do testowania

Inne:

- **Postman** – testowanie API (HTTP, REST, GraphQL).
- **SoapUI** – testy usług webowych SOAP/REST.
- **JMeter** – testy wydajnościowe i obciążeniowe.
- **OWASP ZAP** – testy bezpieczeństwa aplikacji webowych.

Przykłady zastosowania

JUnit

Calculator.java

```
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int divide(int a, int b) {
        return a / b;
    }
}
```

CalculatorTest.java

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class CalculatorTest {

    @Test
    void testAdd() {
        Calculator calc = new Calculator();
        assertEquals(5, calc.add(2, 3));   // ✅ 2+3 = 5
        assertEquals(0, calc.add(-1, 1));  // ✅ -1+1 = 0
    }

    @Test
    void testDivide() {
        Calculator calc = new Calculator();
        assertEquals(5, calc.divide(10, 2));  // ✅ 10/2 = 5
        assertThrows(ArithmeticException.class, () ->
calc.divide(10, 0)); // ❌ dzielenie przez 0
    }
}
```

Przykłady zastosowania

Unittest

calculator.py

```
def add(a, b):
    return a + b

def divide(a, b):
    return a / b
```

```
python -m unittest
test_calculator.py
```

test_calculator.py

```
import unittest
from calculator import add, divide

class TestCalculator(unittest.TestCase):
    def test_add(self):
        self.assertEqual(add(2, 3), 5)      # ✅ 2+3 = 5
        self.assertEqual(add(-1, 1), 0)     # ✅ -1+1 = 0

    def test_divide(self):
        self.assertEqual(divide(10, 2), 5)  # ✅ 10/2 = 5
        with self.assertRaises(ZeroDivisionError):  # ❌ 10/0 = błąd
            divide(10, 0)

if __name__ == '__main__':
    unittest.main()
```

Przykłady zastosowania

Testy kodu JavaScript (Jest)

calculator.js

```
function add(a, b) {
  return a + b;
}

function divide(a, b) {
  return a / b;
}

module.exports = { add, divide };
```

calculator.test.js

```
const { add, divide } = require("./calculator");

test("dodawanie liczb", () => {
  expect(add(2, 3)).toBe(5);    // ✅
  expect(add(-1, 1)).toBe(0);   // ✅
});

test("dzielenie liczb", () => {
  expect(divide(10, 2)).toBe(5); // ✅
  expect(() => divide(10, 0)).toThrow(); // ❌ dzielenie
przez 0
});
```

npx jest

Narzędzia do testowania

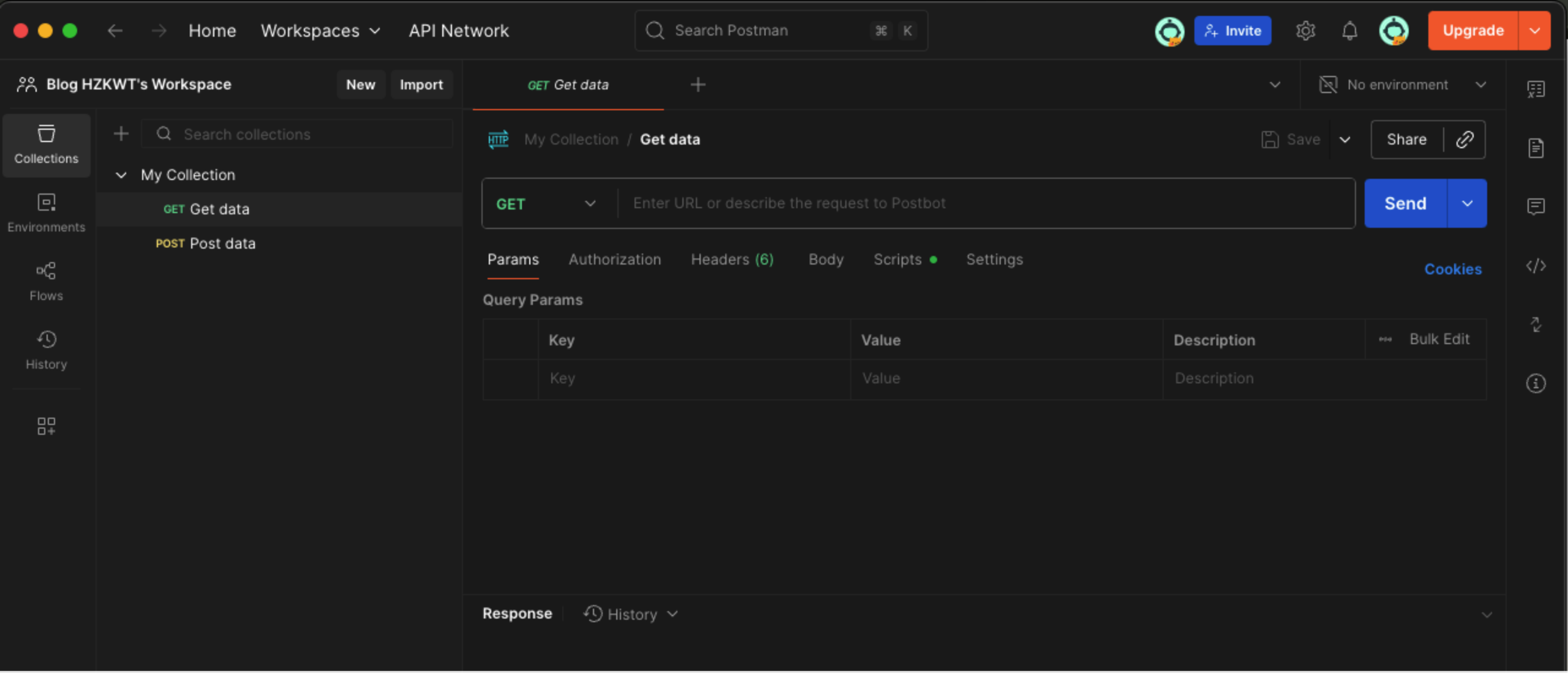
JUnit (Java), NUnit (.NET) – frameworki do pisania testów jednostkowych i integracyjnych.
Przykład: Automatyczny test, który codziennie sprawdza działanie funkcji kalkulującej ceny.

Selenium – automatyzacja testów UI w przeglądarce.
Przykład: Automatyczne klikanie i wypełnianie formularzy na stronie internetowej, aby zweryfikować poprawność działania.

Postman – testowanie i automatyzacja testów API (interfejsów do komunikacji między systemami).
Przykład: Sprawdzenie, czy API zwraca poprawne dane podczas zapytania o profil użytkownika.

Jenkins – narzędzie do ciągłej integracji (CI), które automatycznie uruchamia testy po każdej zmianie w repozytorium kodu.
Przykład: Po wgraniu nowego kodu do GitHub Jenkins uruchamia wszystkie testy, a wynik jest wysyłany do zespołu.

SonarQube – narzędzie do analizy jakości kodu i wykrywania potencjalnych problemów, takich jak błędy, luki bezpieczeństwa czy złe praktyki programistyczne.
Przykład: Raport SonarQube pokazujący, które fragmenty kodu wymagają refaktoryzacji.


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Blog HZKWT's Workspace
New
Import
GET Get data
No environment
Collections
Environments
Flows
History
Search collections
My Collection
GET Get data
POST Post data
My Collection / Get data
Save
Share
GET
Enter URL or describe the request to Postbot
Send
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Response
History
Enter the URL and click Send to get a response

POSTMAN

Testowanie API

1. Tworzenie żądania

Załóżmy, że testujemy API do zarządzania użytkownikami, które ma endpoint do pobierania informacji o użytkowniku:

- **Metoda:** GET
- **URL:** https://api.example.com/users/1

2. Wysyłanie żądania

W Postmanie:

- Otwórz nową kartę.
- Wybierz metodę GET.
- Wprowadź URL: https://api.example.com/users/1.
- Kliknij przycisk "Send".

3. Analiza odpowiedzi

Po wysłaniu żądania, Postman wyświetli odpowiedź. Przykładowa odpowiedź może wyglądać tak:

json5 lines
Click to expand

```
{
"id": 1,
...
```

4. Dodawanie testów

W zakładce "Tests" w Postmanie możesz dodać skrypty testowe, aby zweryfikować odpowiedź. Oto kilka przykładów testów:

javascript18 lines
Click to close

```
// Sprawdzenie statusu odpowiedzi
pm.test("Status code is 200",
function () {
...
```

5. Uruchamianie testów

Po dodaniu testów, kliknij przycisk "Send" ponownie, aby wysłać żądanie. Postman automatycznie uruchomi testy i wyświetli wyniki w zakładce "Test Results".

6. Analiza wyników testów

W zakładce "Test Results" zobaczysz, które testy przeszły pomyślnie, a które nie. Możesz również zobaczyć szczegóły błędów, jeśli jakieś testy nie powiodły się.

Po sprawdzeniu ...

```
// Sprawdzenie statusu odpowiedzi
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

// Sprawdzenie, czy odpowiedź zawiera właściwości
pm.test("Response has id, name, and email", function () {
    const responseJson = pm.response.json();
    pm.expect(responseJson).to.have.property("id");
    pm.expect(responseJson).to.have.property("name");
    pm.expect(responseJson).to.have.property("email");
});

// Sprawdzenie wartości pola name
pm.test("Name is John Doe", function () {
    const responseJson = pm.response.json();
    pm.expect(responseJson.name).to.eql("John Doe");
});
```

Testowanie API w Postmanie jest proste i intuicyjne. Dzięki możliwości dodawania testów w JavaScript, możesz automatycznie weryfikować odpowiedzi i upewnić się, że Twoje API działa zgodnie z oczekiwaniami.

SELENIUM

Czym jest Selenium?

- Selenium to popularne narzędzie do automatyzacji testów aplikacji webowych.
- Umożliwia symulację interakcji użytkownika z przeglądarką.

Dlaczego warto używać Selenium?

- Wspiera wiele przeglądarek (Chrome, Firefox, Safari, Edge).
- Obsługuje różne języki programowania (Java, C#, Python, Ruby, JavaScript).
- Umożliwia testowanie aplikacji w różnych środowiskach.

- Selenium Webdriver – wykorzystuje API (ang. Application Programming Interface) przeglądarek internetowych. Umożliwia symulowanie obsługi przeglądarki dokładnie w taki sposób, w jaki zrobiłby to człowiek.
- Selenium Grid – wykorzystuje przypadki testowe stworzone przy użyciu Selenium Webdriver. Pozwala na ich uruchomienie na różnych platformach sprzętowych.
- Selenium IDE – to wtyczka do przeglądarki. Można z jej pomocą nagrać wykonywane działania, a później je odtworzyć.
-

Przykład zastosowania

```
from selenium import webdriver

# Inicjalizacja WebDriver
driver = webdriver.Chrome()

# Otwórz stronę
driver.get("https://example.com")

# Znajdź element i wykonaj akcję
element = driver.find_element_by_name("q")
element.send_keys("Selenium")
element.submit()

# Sprawdź tytuł strony
assert "Selenium" in driver.title

# Zamknij przeglądarkę
driver.quit()
```

Najlepsze praktyki:

- **Struktura testów**
- Organizuj testy w klasy i metody.
- Używaj wzorców projektowych, takich jak Page Object Model.
- **Zarządzanie danymi testowymi**
- Używaj plików konfiguracyjnych lub baz danych do przechowywania danych testowych.
- **Raportowanie**
- Używaj narzędzi do generowania raportów (np. Allure, ExtentReports).

Przykładowe zadania

Testy jednostkowe (Unit Tests):

- Sprawdź, czy funkcja obliczPole(3,4) zwraca poprawną wartość 12.
- Zweryfikuj, czy metoda dodajUżytkownika() dodaje użytkownika do bazy danych.
- Upewnij się, że metoda zaloguj("admin","1234") zwraca true tylko dla poprawnych danych logowania.
- Testuj, czy metoda dzielenie(10,0) zgłasza wyjątek zamiast zwracać błędny wynik.

Przykładowe zadania

Testy systemowe (System Tests):

- Zweryfikuj, czy użytkownik może się **zarejestrować, zalogować, a następnie wylogować** poprawnie.
- Sprawdź, czy po dodaniu produktu do koszyka wyświetla się poprawna suma zamówienia.
- Upewnij się, że przy próbie rejestracji z istniejącym mailem system zwraca komunikat o błędzie.
- Sprawdź, czy aplikacja działa poprawnie w różnych przeglądarkach (Chrome, Firefox, Edge).

Przykładowe zadania

Testy regresyjne (Regression Tests):

- Po aktualizacji modułu płatności sprawdź, czy nadal można zapłacić kartą i PayPalem.
- Po dodaniu nowej walidacji haseł upewnij się, że logowanie i rejestracja wciąż działają poprawnie.
- Po poprawce błędu w module koszyka sprawdź, czy dodawanie i usuwanie produktów nadal działa.
- Po wdrożeniu nowego UI zweryfikuj, czy linki w menu prowadzą do odpowiednich stron.

Przykładowe zadania

Testy bezpieczeństwa (Security Tests):

- Spróbuj wprowadzić w polu logowania dane "' OR '1'='1" i upewnij się, że system jest odporny na **SQL Injection**.
- Sprawdź, czy sesja użytkownika wygasa po 15 minutach bezczynności.
- Zweryfikuj, czy ciasteczka mają ustawione flagi HttpOnly i Secure.
- Przetestuj, czy użytkownik z rolą „user” nie ma dostępu do panelu administratora.
- Spróbuj wysłać zapytanie POST bez tokena CSRF i sprawdź, czy system je odrzuca.

Przykład testu zgodnego z egzaminem INF.04

- **Nazwa projektu:** System rejestracji użytkowników
- **Autor:** Jan Kowalski
- **Data:** 19.08.2025
- **Cel testów:** Sprawdzenie poprawności działania aplikacji oraz jej bezpieczeństwa.
- **2. Plan testów**
- Rodzaje testów, które zostaną przeprowadzone:
- **Testy jednostkowe** – sprawdzenie poprawności działania pojedynczych funkcji.
- **Testy systemowe** – weryfikacja działania całej aplikacji jako całości.
- **Testy regresyjne** – upewnienie się, że nowe zmiany nie psują istniejących funkcji.
- **Testy bezpieczeństwa** – sprawdzenie odporności aplikacji na ataki i nieautoryzowany dostęp.

Przykład testu zgodnego z egzaminem INF.04

ID testu	Opis testu	Dane wejściowe	Oczekiwany wynik	Wynik rzeczywisty	Status
T1	Test jednostkowy funkcji dodaj()	2, 3	5	5	✅ OK
T2	Rejestracja nowego użytkownika	login: adam, hasło: 1234	„Rejestracja udana”	„Rejestracja udana”	✅ OK
T3	Rejestracja istniejącego użytkownika	login: adam, hasło: 1111	„Użytkownik już istnieje”	„Użytkownik już istnieje”	✅ OK
T4	Test logowania (regresja)	login: admin, hasło: 1234	Dostęp przyznany	Dostęp przyznany	✅ OK
T5	Test bezpieczeństwa – SQL Injection	login: "' OR '1'='1", hasło: "' OR '1'='1"	Odmowa logowania	Dostęp przyznany ❌	❌ Błąd
T6	Sesja użytkownika po 15 min braku aktywności	brak działań użytkownika	Sesja wygasła	Sesja wygasła	✅ OK

Przykład testu zgodnego z egzaminem INF.04

Wnioski:

- Funkcje aplikacji działają poprawnie (T1–T4).
- Zidentyfikowano **lukę bezpieczeństwa** związaną z podatnością na SQL Injection (T5).
- Mechanizm sesji działa poprawnie (T6).

Zalecenia:

- Zaimplementować parametryzowane zapytania SQL, aby usunąć podatność na SQL Injection.
- Wprowadzić dodatkową walidację danych wejściowych.

Podsumowanie - pytania

1. Kto wykonuje testy akceptacyjne?
2. Na czym polegają testy integracyjne?
3. Do czego służy POSTMAN?
4. Na czym polegają testy bezpieczeństwa?
5. Co jest sprawdzane podczas testów uwierzytelniania?
6. Co to jest SQL Injection?
7. Do czego służy Jenkins?

Zadania do samodzielnego wykonania

Zadanie 1

Korzystając z języka Python oraz biblioteki Unittest wykonaj test jednostkowy dla poniższych programów:

```
def poleprostokata(a, b):
    return a*b
```

```
def dodaj(a, b):
    return a + b
```

```
def oblicz(a, b, c):
    return a + b – c
```

Zadania do samodzielnego wykonania

Zadanie 2

Wykonaj test systemowy dla poniższego kodu:

```
class Aplikacja:
    def __init__(self):
        self.uzytkownicy = {}

    def rejestracja(self, login, haslo):
        if login in self.uzytkownicy:
            return "Użytkownik już istnieje"
        self.uzytkownicy[login] = haslo
        return "Rejestracja udana"
```

Zadania do samodzielnego wykonania

Zadanie 3

Wykonaj test regresyjny dla poniższego kodu:

```
class Aplikacja:
    def __init__(self):
        self.uzytkownicy = {"admin": "1234"}

    def logowanie(self, login, haslo):
        return self.uzytkownicy.get(login) == haslo
```

Zadania do samodzielnego wykonania

Zadanie 4 (na 5)

Wykonaj test bezpieczeństwa wstrzykując do programu następujące zapytanie:

```
def logowanie_sql(login, haslo):
    zapytanie = f"SELECT * FROM users WHERE login='{login}' AND haslo='{haslo}'"
    return zapytanie
```

Zadania do samodzielnego wykonania

Zadanie 5 (na 5)

```
import tkinter as tk
from tkinter import messagebox

# ======== Baza użytkowników (słownik login:hasło) ========
users = {}

# ======== Funkcja rejestracji ========
def register():
    login = entry_login.get()
    password = entry_password.get()

    # Zadanie 4: Walidacja pustych pól
    if login == "" or password == "":
        messagebox.showerror("Błąd", "Login i hasło nie mogą być puste!")
        return

    # Zadanie 5: Sprawdzenie czy login istnieje
    if login in users:
        messagebox.showerror("Błąd", "Użytkownik już istnieje!")
        return

    # Rejestracja
    users[login] = password
    messagebox.showinfo("Sukces", f"Zarejestrowano użytkownika: {login}")

# ======== Funkcja logowania ========
def login():
    login = entry_login.get()
    password = entry_password.get()

    # Zadanie 4: Walidacja pustych pól
    if login == "" or password == "":
        messagebox.showerror("Błąd", "Login i hasło nie mogą być puste!")
        return

    # Logowanie
    if login in users and users[login] == password:
        messagebox.showinfo("Sukces", f"Zalogowano jako: {login}")
    else:
        messagebox.showerror("Błąd", "Niepoprawny login lub hasło!")

# ======== GUI ========
root = tk.Tk()
root.title("Test systemowy – Zadanie 4 i 5")
root.geometry("300x200")

# Login
tk.Label(root, text="Login:").pack(pady=5)
entry_login = tk.Entry(root)
entry_login.pack()

# Hasło (Zadanie 4 – maskowanie hasła)
tk.Label(root, text="Hasło:").pack(pady=5)
entry_password = tk.Entry(root, show="*")
entry_password.pack()

# Przyciski
tk.Button(root, text="Rejestracja", command=register).pack(pady=5)
tk.Button(root, text="Logowanie", command=login).pack(pady=5)

root.mainloop()
```

Zadania do samodzielnego wykonania

Na podstawie kodu z poprzedniej strony wykonaj test bezpieczeństwa sprawdzający czy hasło jest ukrywane i nie można zalogować się bez wpisania hasła.

Sprawdź kod na przynajmniej dwóch systemach operacyjnych i uruchom go przynajmniej 10 razy. Obserwacje wypisz w tabeli według wzoru.

Numer próby	Obserwacje – OS 1	Obserwacje – OS 2
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

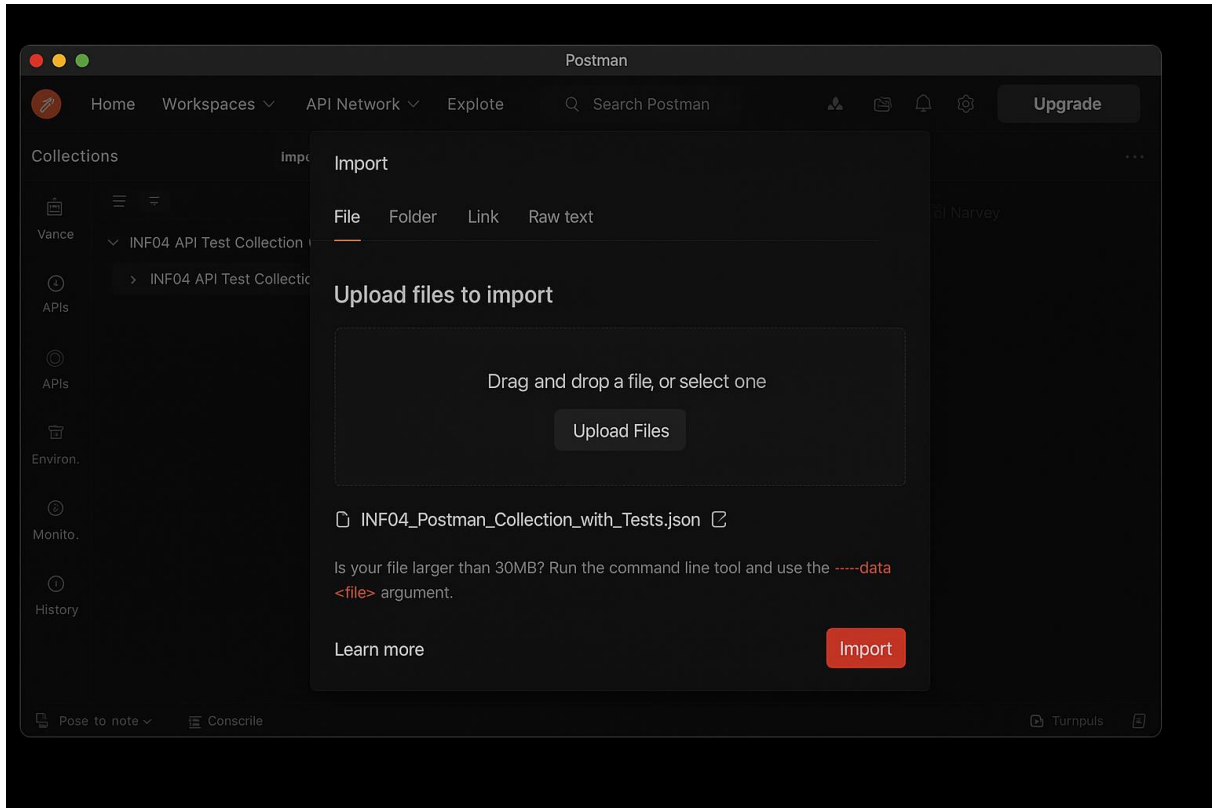
Zadania do samodzielnego wykonania


```
{
"info": {
"name": "INF04 API Test Collection",
"schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json",
"_postman_id": "12345678-abcd-efgh-ijkl-987654321000"
},
"item": [
{
"name": "Get Users",
"request": {
"method": "GET",
"header": [],
"url": {
"raw": "http://localhost:5000/users",
"protocol": "http",
"host": [
"localhost"
],
"port": "5000",
"path": [
"users"
]
}
}
},
{
"name": "Create User",
"request": {
"method": "POST",
"header": [
{
"key": "Content-Type",
"value": "application/json"
}
],
"body": {
"mode": "raw",
"raw": "{\"name\": \"Jan Kowalski\", \"email\": \"jan.kowalski@example.com\"}"
},
"url": {
"raw": "http://localhost:5000/users",
"protocol": "http",
"host": [
"localhost"
],
"port": "5000",
"path": [
"users"
]
}
}
},
{
"name": "Get User by ID",
"request": {
"method": "GET",
"header": [],
"url": {
"raw": "http://localhost:5000/users/1",
"protocol": "http",
"host": [
"localhost"
],
"port": "5000",
"path": [
"users",
"1"
]
}
}
},
{
"name": "Delete User",
"request": {
"method": "DELETE",
"header": [],
"url": {
"raw": "http://localhost:5000/users/1",
"protocol": "http",
"host": [
"localhost"
],
"port": "5000",
"path": [
"users",
"1"
]
}
}
}
]
}
```

Zadania do samodzielnego wykonania

Zadanie 6 (na 5)

Za pomocą POSTMAN wczytaj kolekcję danych z poprzedniej strony. Następnie wykonaj wszystkie możliwe testy tej kolekcji. Po weryfikacji kolekcji wskaż błędy w postaci zrzutu ekranu. Zapisz wynik w pliku nazwanym swoim imieniem i numerem z dziennika.



Zadania do samodzielnego wykonania

Zadanie 7 (na 5)

Poniższe kod zawiera istotne błędy:

Wykonaj test jednostkowy, regresyjny oraz bezpieczeństwa dla tego kodu. Wskaż istotne błędy oraz zaproponuj ich rozwiązanie.

```
class User:
    def __init__(self, username, password):
        self.username = username
        self.password = password

    def greet(self):
        return "Witaj " + self.username

    def divide_numbers(self, a, b):
        return a / b

    def get_password(self):
        return self.password
```

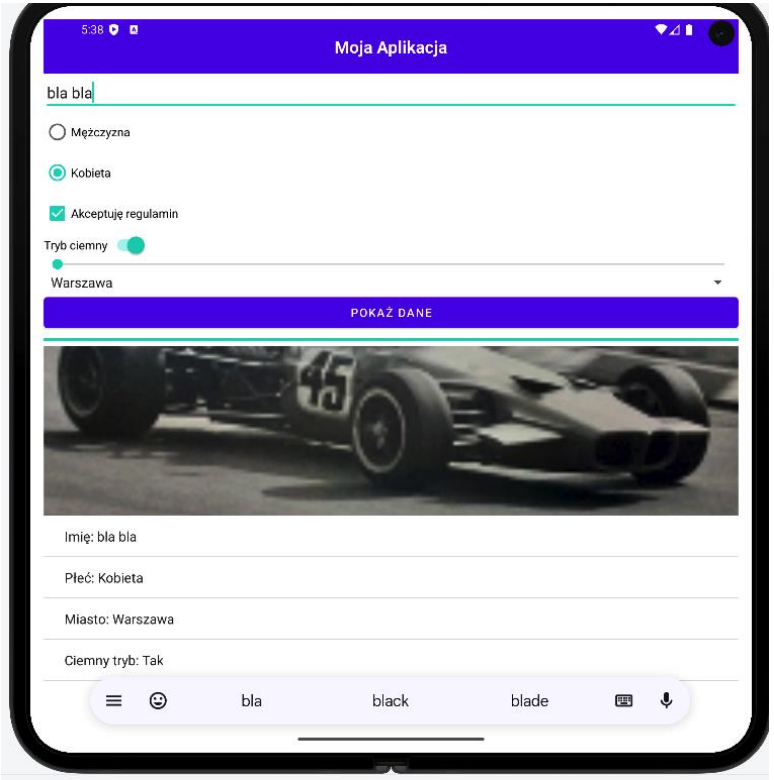
Zadania do samodzielnego wykonania

Zadanie 8 (na 5)

Wykonaj aplikację mobilną według wzoru (całość znajdziesz w prezentacji o języku JAVA I Android Studio).

Następnie wykonaj testy, w których sprawdzisz czy aplikacja działa na API 28, API 30, API 32 i najnowszym API 36.

Obserwacje zapis w tabeli.



Zadania do samodzielnego wykonania

Zadanie 9

calculator.js

```
function add(a, b) {
  return a - b;
}

function multiply(a, b) {
  return a + b;
}

function divide(a, b) {
  if (b === 0) {
    return 0;
  }
  return a / b;
}

module.exports = { add, multiply, divide };
```

Za pomocą Jest wykonaj test poprawności kodu umieszczonego obok.

Instalacja:

```
npm init -y
npm install --save-dev jest
```

W package.json dodaj:

```
"scripts": {
  "test": "jest"
}
```

Uruchom:

```
npm test
```

Zadania do samodzielnego wykonania

Zadanie 10

Wykonaj aplikację konsolową w języku Java, której zadaniem będzie sortowanie bąbelkowe i kubełkowe liczb wprowadzonych przez użytkownika.

Użytkownik może wybrać jeden lub drugi sposób sortowania a następnie wprowadza dowolną ilość liczb. Po wykonaniu programu wykonaj test poprawności kodu za pomocą testu jednostkowego.

Zadanie 11

Wykonaj aplikację konsolową w języku Python, której zadaniem będzie sortowanie szybkie i przez wstawianie liczb wprowadzonych przez użytkownika.

Użytkownik może wybrać jeden lub drugi sposób sortowania a następnie wprowadza dowolną ilość liczb. Po wykonaniu programu wykonaj test poprawności kodu za pomocą testu jednostkowego.

W jakim celu tworzy się dokumentację?

Zrozumienie i utrzymanie:
Nowy programista łatwiej zrozumie działanie aplikacji dzięki szczegółowym opisom.

Wprowadzanie zmian:
Dokumentacja wskazuje, gdzie i jak zmieniać kod, by nie powodować błędów.

Szkolenie użytkowników:
Instrukcje pozwalają użytkownikom szybko nauczyć się korzystać z aplikacji.

Przejrzystość zespołu:
Ułatwia komunikację i współpracę między programistami, testerami i menadżerami.

Przykład:
Firma Microsoft udostępnia dokumentację API dla programistów tworzących aplikacje korzystające z Windows.

Rodzaje dokumentacji

Dokumentacja techniczna:

- Opis architektury systemu
- Schematy bazy danych
- API (interfejsy programistyczne)
- Instrukcje konfiguracji środowiska
  Przykład: Swagger – narzędzie do tworzenia dokumentacji API w formie interaktywnej strony.

Dokumentacja użytkownika:

- Instrukcje obsługi programu
- FAQ (Najczęściej zadawane pytania)
- Tutoriale i poradniki
  Przykład: Dokumentacja Office 365 dostępna na stronie Microsoft, opisująca jak korzystać z aplikacji.

Dokumentacja projektowa:

- Specyfikacja wymagań funkcjonalnych
- Plany testów i scenariusze testowe
- Raporty z błędów i decyzji projektowych
  Przykład: Jira i Confluence do zarządzania dokumentacją i śledzenia błędów w projekcie.

Dobre praktyki

- **Dokumentuj na bieżąco:**
  Unikaj sytuacji, że dokumentacja jest tworzona dopiero po zakończeniu prac — wtedy może być niekompletna lub błędna.
- **Jasny i prosty język:**
  Unikaj zbyt technicznego żargonu tam, gdzie to nie jest konieczne.
- **Standardy i szablony:**
  Używaj spójnych formatów, aby dokumentacja była czytelna i ułatwiała nawigację.
- **Aktualizuj dokumentację:**
  Dokumentacja powinna odzwierciedlać aktualny stan projektu.
- **Narzędzia:**
  - Markdown (łatwa składnia do formatowania tekstu)
  - Wiki firmowe (np. Confluence)
  - Systemy kontroli wersji (np. Git) do śledzenia zmian dokumentów
- **Przykład:**
  Wiele zespołów programistycznych korzysta z GitHub Wiki lub Notion do tworzenia i przechowywania dokumentacji projektowej.

Przykład dokumentacji wewnątrz kodu

```
def oblicz_podatek(kwota):
    """
    Oblicza podatek VAT (19%) od podanej kwoty.

    Parametry:
    kwota (float): Kwota netto, od której ma być obliczony podatek.

    Zwraca:
    float: Wartość podatku VAT.

    Przykład użycia:
    >>> oblicz_podatek(100)
    19.0
    >>> oblicz_podatek(-10)
    0
    """
    if kwota <= 0:
        return 0
    return kwota * 0.19
```

Wewnątrz funkcji wstawiono opis działania i zastosowane wartości. Proste?

Dokumentacja w formie dokumentu zewnętrznego

- **Moduł obliczeń podatku VAT**
- **Cel modułu**
- Moduł odpowiedzialny za obliczanie wartości podatku VAT od kwoty netto.
- **Funkcje**
- **oblicz_podatek(kwota: float) -> float**
  Oblicza podatek VAT (19%) dla podanej kwoty netto.
  - **Parametry:**
    - kwota: liczba zmiennoprzecinkowa — wartość netto.
  - **Zwraca:**
    - podatek VAT jako liczba zmiennoprzecinkowa.
  - **Zachowanie:**
    - Dla wartości ujemnych i zerowych zwraca 0.

```
from podatek import oblicz_podatek

kwota = 200.0
podatek = oblicz_podatek(kwota)
print(f"Podatek VAT od {kwota} wynosi {podatek}")
```

Scenariusz użytkowania

Scenariusz 1: Obliczenie podatku VAT dla poprawnej kwoty

- **Cel:** Użytkownik chce obliczyć podatek VAT od kwoty netto.
- **Kroki:**

1. Użytkownik uruchamia aplikację.
2. Wprowadza kwotę netto (np. 100).
3. Kliknięciem potwierdza obliczenie.
4. System oblicza podatek VAT (19%) i wyświetla wynik (19).
5. Użytkownik odczytuje wynik i kończy działanie.

- **Rezultat:** Wyświetlona jest poprawna wartość podatku VAT.

Scenariusz 2: Próba obliczenia podatku VAT dla wartości ujemnej

- **Cel:** Użytkownik sprawdza, co się stanie, gdy poda wartość ujemną.
- **Kroki:**

1. Użytkownik uruchamia aplikację.
2. Wprowadza kwotę netto ujemną (np. -50).
3. Kliknięciem potwierdza obliczenie.
4. System wykrywa niepoprawną wartość i zwraca podatek równy 0.
5. System wyświetla komunikat „Podaj wartość dodatnią” (opcjonalnie).

- **Rezultat:** Aplikacja nie wykonuje obliczeń dla wartości ujemnych i informuje użytkownika o błędzie.

Scenariusz 3: Wprowadzenie wartości zerowej

- **Cel:** Użytkownik wprowadza 0 jako kwotę netto.
- **Kroki:**

1. Użytkownik uruchamia aplikację.
2. Wprowadza kwotę 0.
3. Potwierdza obliczenie.
4. System zwraca wartość podatku równą 0.
5. Wyświetla wynik.

- **Rezultat:** Podatek VAT wynosi 0, co jest poprawnym wynikiem.

Scenariusz 4: Zapis wyniku do pliku (rozszerzenie funkcjonalności)

- **Cel:** Użytkownik chce zapisać wynik obliczenia do pliku tekstowego.
- **Kroki:**

1. Użytkownik uruchamia aplikację.
2. Wprowadza kwotę netto.
3. Potwierdza obliczenie.
4. System wyświetla wynik i pyta o zapis.
5. Użytkownik wybiera opcję zapisu.
6. System tworzy plik tekstowy z wynikami.
7. Użytkownik kończy pracę.

- **Rezultat:** Wynik jest zachowany w pliku, do późniejszego odczytu.

Dokumentacja z wykonania zgodna z INF.04

1. Informacje ogólne

Nazwa projektu: System rejestracji użytkowników

Autor: Jan Kowalski

Data: 19.08.2025

Cel zadania: Stworzenie aplikacji umożliwiającej rejestrację i logowanie użytkowników oraz bezpieczne przechowywanie danych.

2. Opis funkcjonalności

Rejestracja użytkownika

- Formularz zawiera pola: login, hasło, powtórzenie hasła.
- Sprawdzenie, czy login jest unikalny.
- Hasła są szyfrowane (hash + sól) przed zapisaniem w bazie danych.

Logowanie

- Użytkownik loguje się poprzez podanie loginu i hasła.
- System weryfikuje zgodność danych z bazą.
- Sesja użytkownika wygasa po 15 minutach bezczynności.

Bezpieczeństwo

- Dane przesyłane są przez HTTPS.
- Mechanizmy ochrony przed SQL Injection i XSS.
- Role użytkowników: zwykły użytkownik i administrator (odmienne uprawnienia).

Interfejs użytkownika

- Prostota i intuicyjność.
- Komunikaty błędów przy niepoprawnych danych.
- Responsywny wygląd, działający w różnych przeglądarkach.

3. Technologie użyte w projekcie

Backend: Python (Flask)

Frontend: HTML, CSS, JavaScript

Baza danych: SQLite

Dodatkowe biblioteki: bcrypt (szyfrowanie haseł), Flask-Login (sesje)

4. Etapy wykonania

- Utworzenie struktury projektu (foldery: templates, static, app).
- Implementacja logiki rejestracji i logowania.
- Wdrożenie mechanizmów bezpieczeństwa (hashowanie haseł, walidacja danych, ochrona przed SQL Injection).
- Zaprojektowanie interfejsu użytkownika.
- Testy funkcjonalne ręczne (sprawdzenie rejestracji, logowania, sesji).
- Finalne wdrożenie i dokumentacja.

5. Wnioski

- Aplikacja spełnia założenia zadania egzaminacyjnego.
- Wszystkie podstawowe funkcje działają poprawnie.
- Dodatkowe zabezpieczenia poprawiają bezpieczeństwo użytkowników.

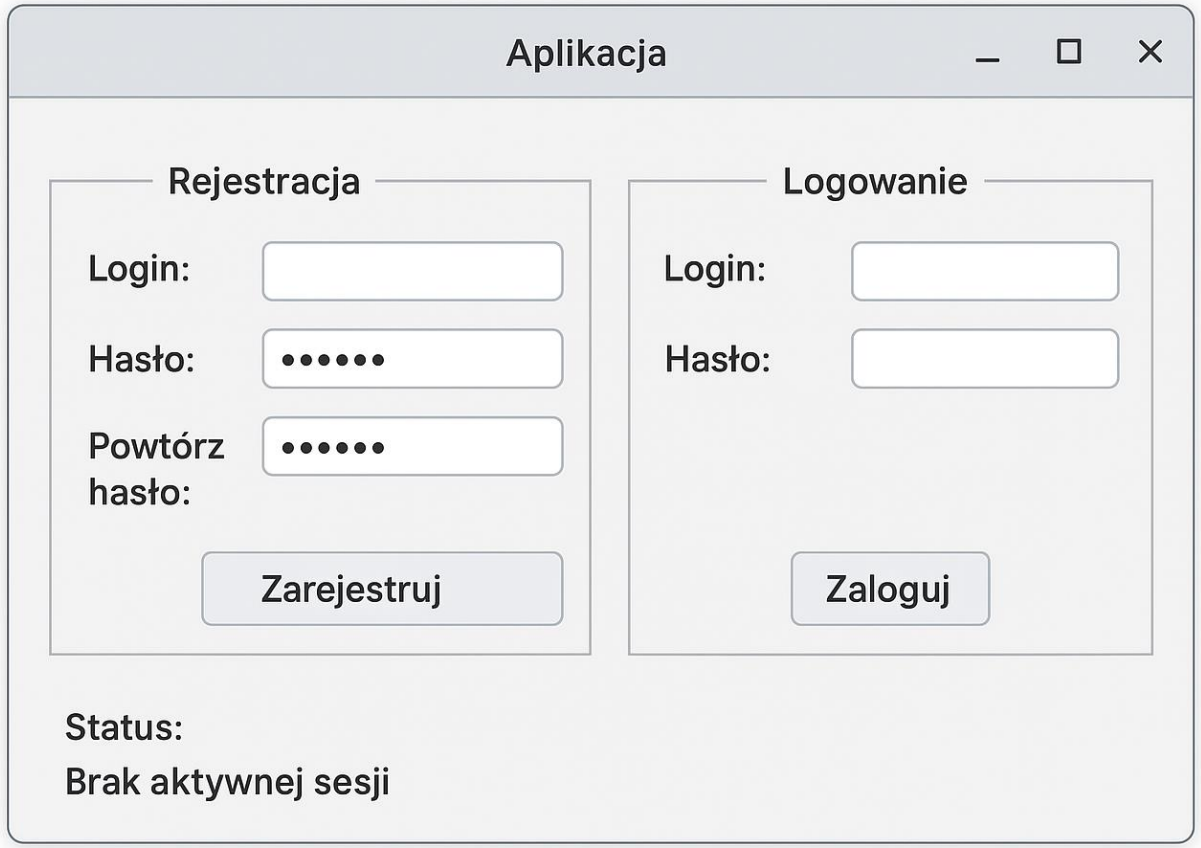
Dokumentacja z wykonania zgodna z INF.04

```
# ============================
# Dokumentacja wykonania zadania – INF.04
# ============================
# Projekt: System rejestracji użytkowników
# Autor: Jan Kowalski
# Data: 19.08.2025
# Cel: Stworzenie aplikacji umożliwiającej rejestrację i logowanie użytkowników,
#      zapewniając bezpieczne przechowywanie danych.

# ============================
# 1. Opis funkcjonalności
# ============================
# Rejestracja użytkownika:
# - Formularz z polami: login, hasło, powtórzenie hasła
# - Sprawdzenie unikalności loginu
# - Hasła szyfrowane przy użyciu hash + sól

# Logowanie:
# - Weryfikacja loginu i hasła z bazą danych
# - Sesja użytkownika wygasa po 15 minutach bezczynności

# Bezpieczeństwo:
# - Dane przesyłane przez HTTPS
# - Ochrona przed SQL Injection i XSS
# - Role użytkowników: zwykły użytkownik i administrator

# Interfejs użytkownika:
# - Intuicyjny i prosty w obsłudze
# - Wyświetlanie komunikatów błędów
# - Responsywny wygląd, kompatybilność z różnymi przeglądarkami

# ============================
# 2. Technologie użyte
# ============================
# - Backend: Python (Flask)
# - Frontend: HTML, CSS, JavaScript
# - Baza danych: SQLite
# - Biblioteki dodatkowe: bcrypt (hashowanie haseł), Flask-Login (zarządzanie sesjami)

# ============================
# 3. Etapy wykonania
# ============================
# 1. Utworzenie struktury projektu (foldery: templates, static, app)
# 2. Implementacja logiki rejestracji i logowania
# 3. Wdrożenie mechanizmów bezpieczeństwa
# 4. Projektowanie interfejsu użytkownika
# 5. Testy funkcjonalne ręczne (rejestracja, logowanie, sesje)
# 6. Finalne wdrożenie i dokumentacja

# ============================
# 4. Wnioski
# ============================
# - Aplikacja spełnia założenia zadania egzaminacyjnego
# - Wszystkie podstawowe funkcje działają poprawnie
# - Mechanizmy bezpieczeństwa zapewniają ochronę danych użytkowników

# ============================
# 5. Przykładowa implementacja funkcji rejestracji
# ============================
import bcrypt

uzytkownicy = {}

def rejestracja(login, haslo):
    """
    Rejestracja nowego użytkownika.
    Sprawdza unikalność loginu i zapisuje hasło w formie zaszyfrowanej.
    """
    if login in uzytkownicy:
        return "Użytkownik już istnieje"
    hashed = bcrypt.hashpw(haslo.encode('utf-8'), bcrypt.gensalt())
    uzytkownicy[login] = hashed
    return "Rejestracja udana"
```

Dokumentacja z wykonania zgodna z INF.04

```
# ============================
# Dokumentacja wykonania zadania – INF.04
# ============================
# Projekt: System rejestracji użytkowników
# Autor: Jan Kowalski
# Data: 19.08.2025
# Cel: Stworzenie aplikacji umożliwiającej rejestrację i logowanie użytkowników,
#      zapewniając bezpieczne przechowywanie danych.
# Technologie: Python, Flask, SQLite, bcrypt
# ============================
# Importy bibliotek
# ============================
import bcrypt  # Biblioteka do szyfrowania haseł
import time    # Do obsługi czasu sesji użytkownika
# ============================
# Inicjalizacja danych
# ============================
uzytkownicy = {}  # Słownik przechowujący użytkowników i zaszyfrowane hasła
sesje = {}       # Słownik przechowujący sesje użytkowników
# ============================
# Funkcja rejestracji użytkownika
# ============================
def rejestracja(login, haslo):
    """
    Rejestracja nowego użytkownika.
    - Sprawdzenie unikalności loginu
    - Hashowanie hasła
    - Zapis do słownika użytkowników
    """
    if login in uzytkownicy:  # Jeśli login już istnieje
        return "Użytkownik już istnieje"  # Zwróć komunikat o błędzie
    # Szyfrowanie hasła z użyciem bcrypt
    hashed = bcrypt.hashpw(haslo.encode('utf-8'), bcrypt.gensalt())
    uzytkownicy[login] = hashed  # Zapisanie loginu i zaszyfrowanego hasła
    return "Rejestracja udana"  # Zwróć komunikat o powodzeniu
# ============================
# Funkcja logowania użytkownika
# ============================
def logowanie(login, haslo):
    """
    Logowanie użytkownika.
    - Sprawdzenie poprawności loginu i hasła
    - Utworzenie sesji użytkownika
    """
    if login not in uzytkownicy:  # Jeśli użytkownik nie istnieje
        return "Nieprawidłowy login"  # Zwróć komunikat błędu
    # Sprawdzenie poprawności hasła
    if bcrypt.checkpw(haslo.encode('utf-8'), uzytkownicy[login]):
        sesje[login] = time.time()  # Utworzenie sesji (czas logowania)
        return "Logowanie udane"  # Zwróć komunikat powodzenia
    else:
        return "Nieprawidłowe hasło"  # Zwróć komunikat o błędzie
# ============================
# Funkcja sprawdzania aktywnej sesji
# ============================
def sesja_aktywna(login):
    """
    Sprawdzenie czy sesja użytkownika jest aktywna.
    - Sesja wygasa po 15 minutach bezczynności
    """
    if login not in sesje:  # Jeśli brak sesji dla loginu
        return False  # Sesja nieaktywna
    if time.time() - sesje[login] > 15 * 60:  # Jeśli minęło 15 minut
        del sesje[login]  # Usuń sesję
        return False  # Sesja wygasła
    return True  # Sesja jest aktywna
# ============================
# Przykładowe użycie funkcji
# ============================
print(rejestracja("adam", "1234"))  # Rejestracja nowego użytkownika
print(rejestracja("adam", "1234"))  # Próba rejestracji istniejącego użytkownika
print(logowanie("adam", "1234"))    # Logowanie poprawnym hasłem
print(logowanie("adam", "1111"))    # Logowanie niepoprawnym hasłem
print(sesja_aktywna("adam"))       # Sprawdzenie sesji po logowaniu
```

Dokumentacja z wykonania zgodna z INF.04



W dokumentacji w formie dokumentu tekstowego załączamy zrzuty ekranu działającego programu. Każdy z nich powinien pokazywać jeden etap obsługi – proces logowania, zapisu itd..

JIRA i Confluence

JIRA to narzędzie do zarządzania projektami i zadaniami używane głównie w metodykach **Agile (Scrum, Kanban)**.

Pozwala tworzyć:

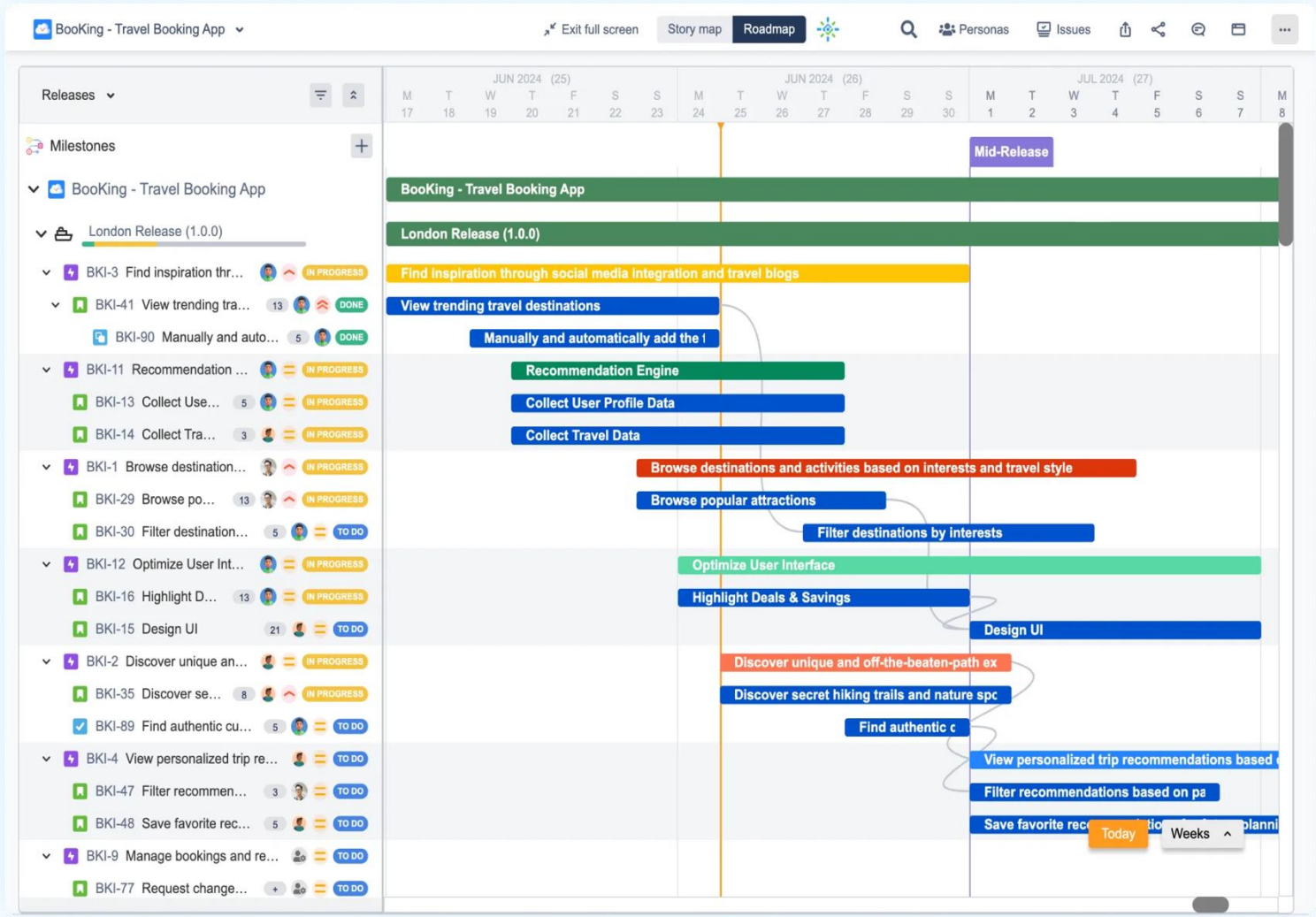
- backlog (lista zadań/projektów do zrobienia),
- sprinty (iteracje pracy),
- tablice Kanban i Scrum,
- zgłoszenia typu **task, bug, epic, story**.

W praktyce programiści, testerzy i managerowie korzystają z JIRA, aby śledzić postęp prac, raportować błędy, planować kolejne etapy projektu.

Przykład użycia JIRA:

Tester znajduje błąd → zakłada **ticket (bug)** w JIRA → przypisuje go programiście → programista naprawia i zmienia status zadania → tester weryfikuje.

JIRA


BooKing - Travel Booking App
Exit full screen
Story map
Roadmap
Personas
Issues
Releases
Milestones
BooKing - Travel Booking App
London Release (1.0.0)
JUN 2024 (25)
JUN 2024 (26)
JUL 2024 (27)
Mid-Release
BooKing - Travel Booking App
London Release (1.0.0)
BKI-3 Find inspiration thr...
Find inspiration through social media integration and travel blogs
BKI-41 View trending tra...
View trending travel destinations
BKI-90 Manually and auto...
Manually and automatically add the
BKI-11 Recommendation ...
Recommendation Engine
BKI-13 Collect Use...
Collect User Profile Data
BKI-14 Collect Tra...
Collect Travel Data
BKI-1 Browse destination...
Browse destinations and activities based on interests and travel style
BKI-29 Browse po...
Browse popular attractions
BKI-30 Filter destination...
Filter destinations by interests
BKI-12 Optimize User Int...
Optimize User Interface
BKI-16 Highlight D...
Highlight Deals & Savings
BKI-15 Design UI
Design UI
BKI-2 Discover unique an...
Discover unique and off-the-beaten-path ex
BKI-35 Discover se...
Discover secret hiking trails and nature spc
BKI-89 Find authentic cu...
Find authentic c
BKI-4 View personalized trip re...
View personalized trip recommendations based
BKI-47 Filter recommen...
Filter recommendations based on pa
BKI-48 Save favorite rec...
Save favorite rec
BKI-9 Manage bookings and re...
BKI-77 Request change...
Today
Weeks

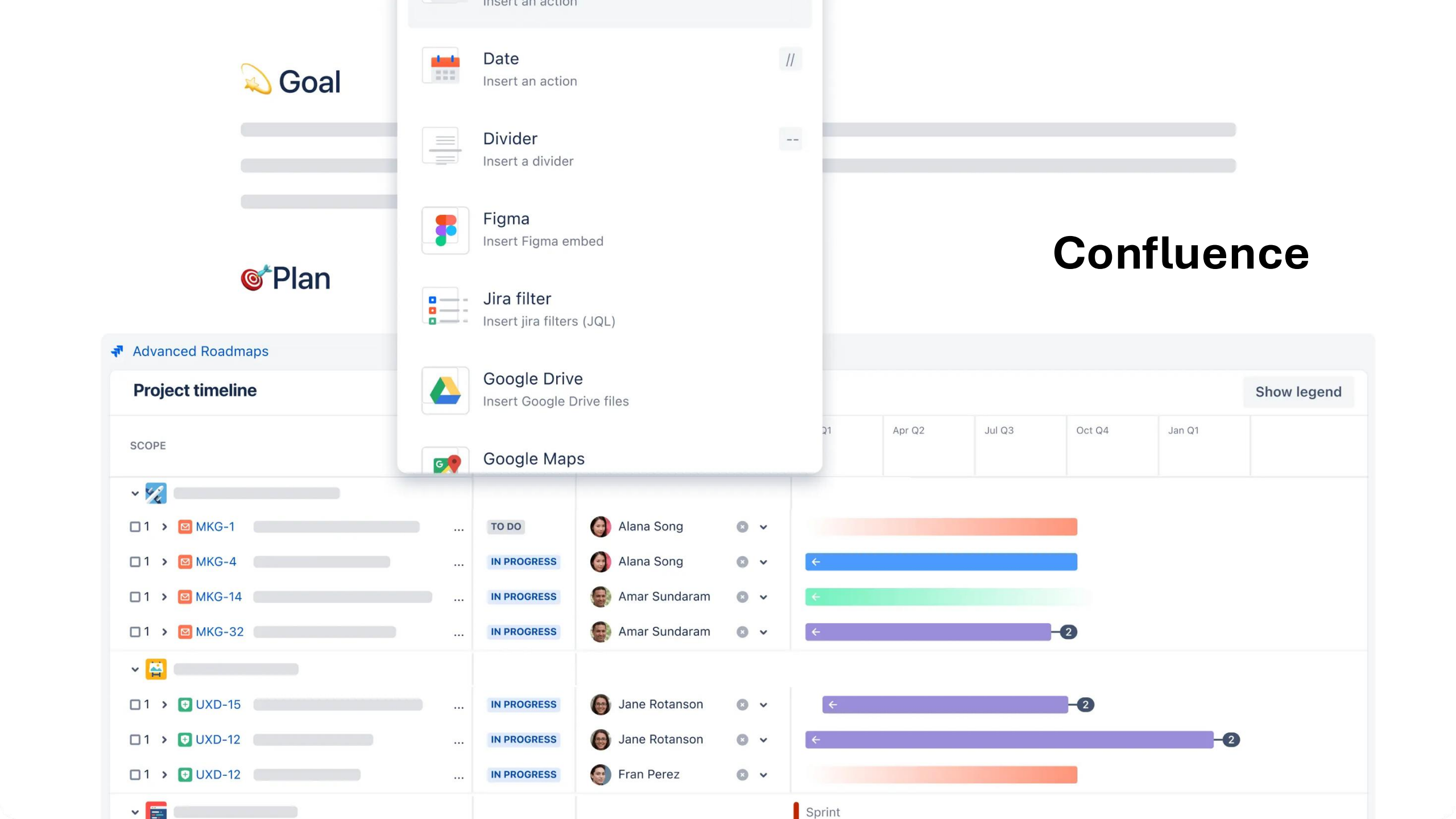
JIRA i Confluence

Confluence to **narzędzie do dokumentacji i współpracy zespołowej**.

- Działa jak firmowa **wiki** – można tam pisać instrukcje, opisy architektury, podręczniki użytkownika, procedury testowe.
- Łatwo integruje się z JIRA (np. można w dokumentacji wstawić linki do zadań z JIRA).

Przykład użycia Confluence:

- Tester tworzy stronę z **planem testów** i podlinkowuje do konkretnych zgłoszeń błędów w JIRA.
- Programista opisuje **architekturę systemu** lub API.
- Project Manager przygotowuje **raport ze sprintu** dostępny dla całego zespołu.


Goal
Plan
Confluence
Insert an action
Date
Insert an action
//
Divider
Insert a divider
--
Figma
Insert Figma embed
Jira filter
Insert jira filters (JQL)
Google Drive
Insert Google Drive files
Google Maps
Advanced Roadmaps
Project timeline
Show legend
SCOPE
Q1
Apr Q2
Jul Q3
Oct Q4
Jan Q1
MKG-1
TO DO
Alana Song
MKG-4
IN PROGRESS
Alana Song
MKG-14
IN PROGRESS
Amar Sundaram
MKG-32
IN PROGRESS
Amar Sundaram
UXD-15
IN PROGRESS
Jane Rotanson
UXD-12
IN PROGRESS
Jane Rotanson
UXD-12
IN PROGRESS
Fran Perez
Sprint

Podsumowanie - pytania

1. Co powinien zawierać nagłówek dokumentacji w formie komentarzy?
2. W jakim celu dodaje się zdjęcia do dokumentacji?
3. Do jakiego rodzaju dokumentacji należą tutoriale i instrukcje obsługi?
4. W jakim celu tworzy się scenariusze użytkowania?
5. Co to jest JIRA?
6. Wymień cechy Confluence?

Zadania do samodzielnego wykonania

Zadanie 1

Napisz aplikację konsolową, która wykonuje szyfrowanie za pomocą szyfru Cezara zdania wpisanego przez użytkownika. Szyfr Cezara wykorzystuje przesunięcie +3 dla każdego znaku.

Następnie za pomocą komentarzy wykonaj dokumentację projektu wewnątrz kodu.

Możesz użyć dowolnego języka programowania.

Zadanie 2

Napisz aplikację konsolową, której zadaniem będzie losowanie kilku zakładów gry LOTTO. Użytkownik wybiera 6 liczb z zakresu od 1 do 49. Następnie program losuje swoje liczby i porównuje z typowanymi przez użytkownika.

Następnie za pomocą komentarzy wykonaj dokumentację projektu wewnątrz kodu.

Możesz użyć dowolnego języka programowania.

Zadania do samodzielnego wykonania

Zadanie 3 (zadanie na 5)

Napisz aplikację konsolową w dowolnym języku, której zadaniem będzie sortowanie bąbelkowe tablicy 6-elementowej zawierającej następujące liczby (9,15,2,10,23,29).

Następnie wykonaj dokumentację projektu w formie dokumentu tekstowego w formacie DOC lub PDF. Dokumentacja powinna zawierać przynajmniej 3 zrzuty ekranu.

Zadanie 4

Napisz aplikacje konsolową, której zadaniem będzie wprowadzanie do pliku danych wprowadzonych przez użytkownika: Imię, Nazwisko, Stanowisko pracy, Telefon, PESEL i Numer stanowiska.

Następnie za pomocą komentarzy wykonaj dokumentację wewnątrz pliku z kodem. Po zakończeniu plik nazwij swoim imieniem i numerem w dzienniku.

Zadania do samodzielnego wykonania

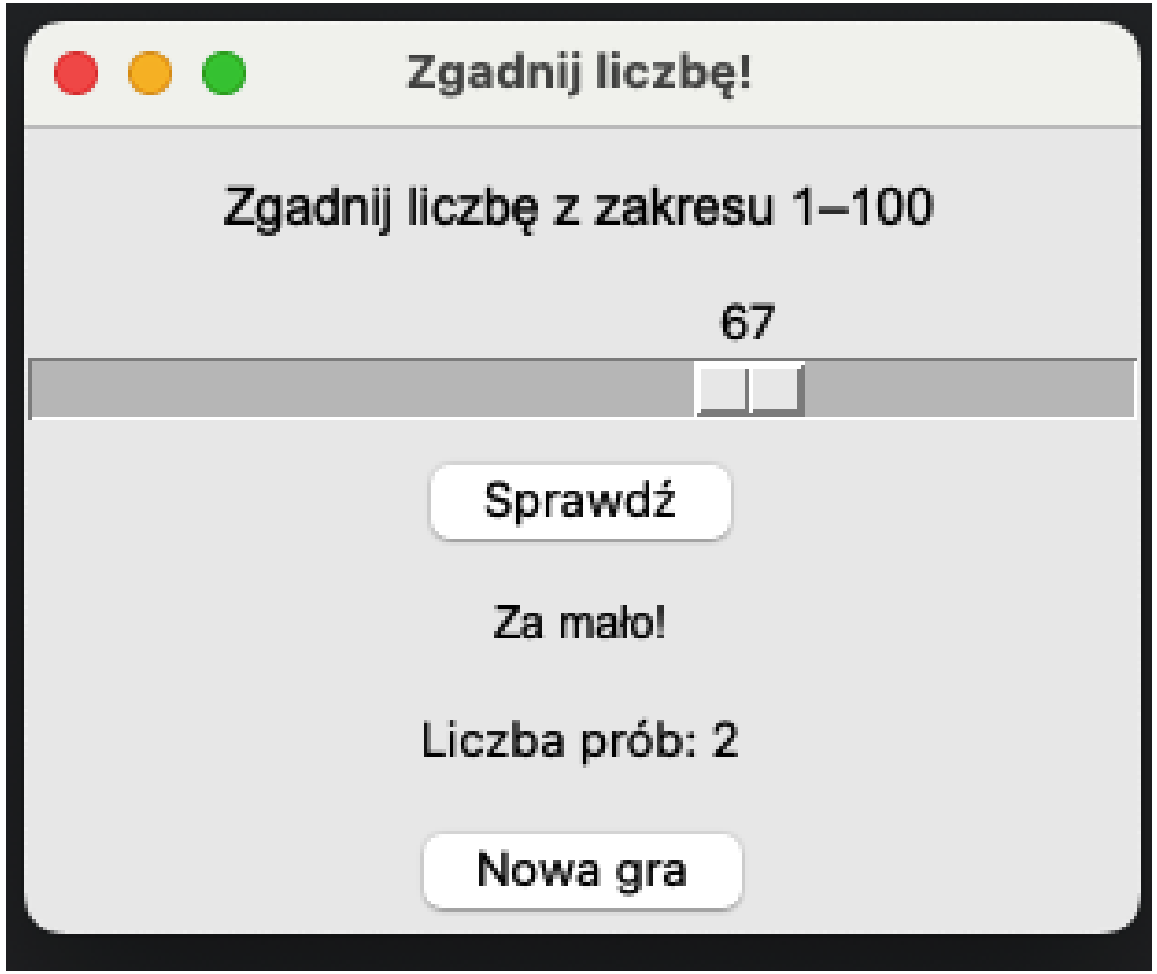
Zadanie 5

Utwórz prostą aplikację kalkulatora
w dowolnym języku
a następnie wykonaj do niej
dokumentację w postaci komentarzy
umieszczonych w kodzie.

Zadania do samodzielnego wykonania

Zadanie 6

Wykonaj aplikację w dowolnym języku według załączonego wzoru. Następnie dołącz do niej dokumentację w postaci pliku WORD lub PDF .

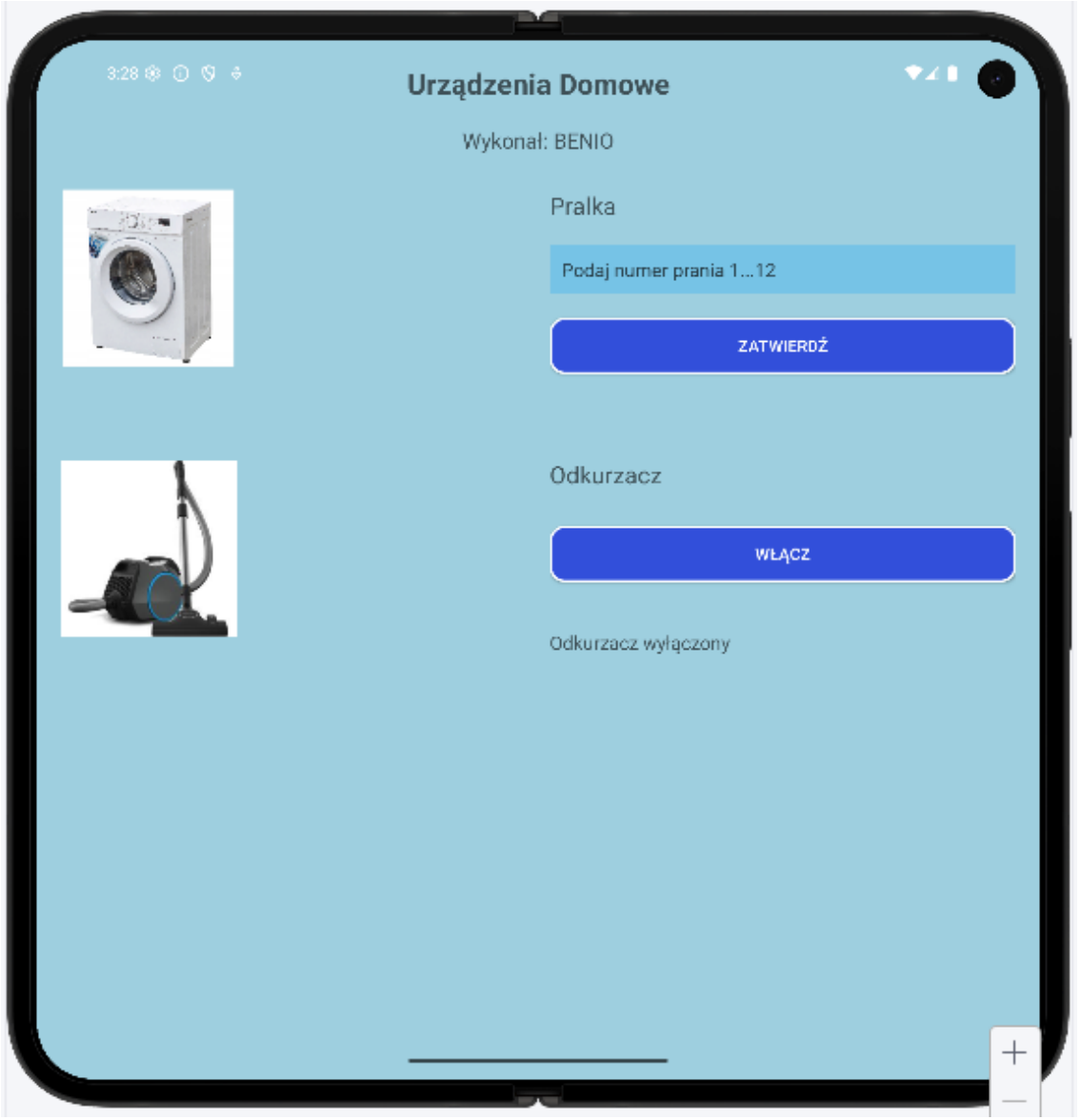


Zadania do samodzielnego wykonania

Zadanie 7 (zadanie na 5)

Po wykonaniu aplikacji Android w języku Java napisz dokumentację projektu w postaci pliku DOC lub PDF. W dokumentacji dołącz przynajmniej 3 zrzuty ekranu oraz.

Całość – projekt, dokumentację oraz wygenerowany plik APK zapisz w postaci archiwum i zapisz podając w tytule swoje imię i nazwisko.

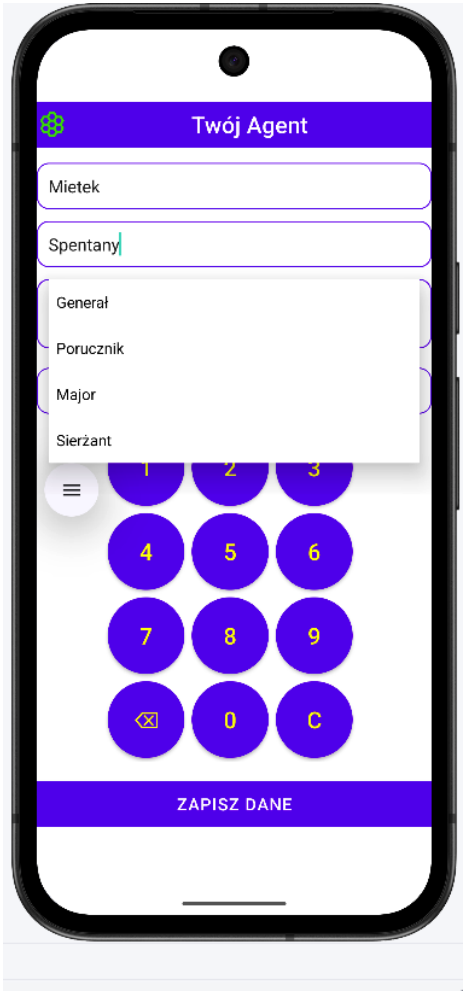


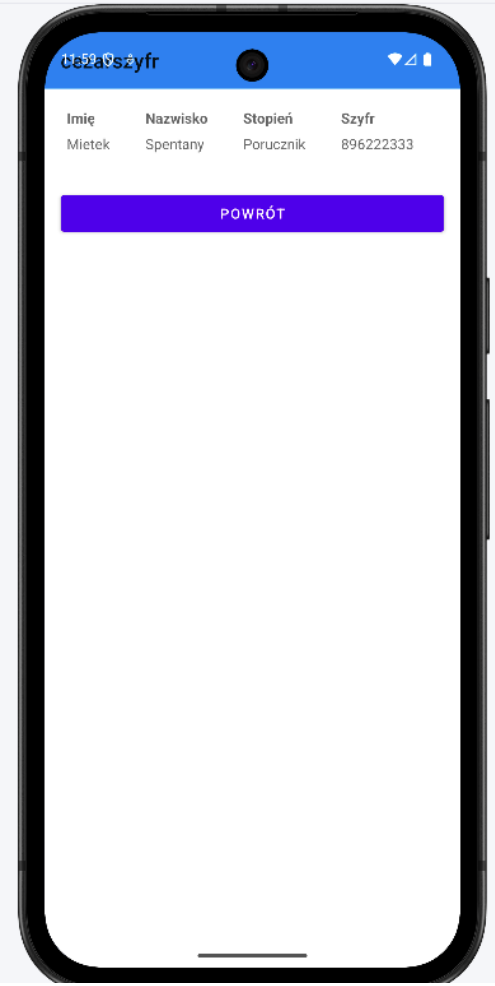
Zadania do samodzielnego wykonania

Zadanie 8 (na 5)

Utwórz aplikację Android według wzoru. Użytkownik wprowadza imię, nazwisko, stopień oraz numer telefonu. Numer telefonu zostaje zaszyfrowany szyfrem Cezara (+3).

Następnie wykonaj do niej dokumentację w formie dokumentu WORD lub PDF z instrukcją obsługi dla użytkownika



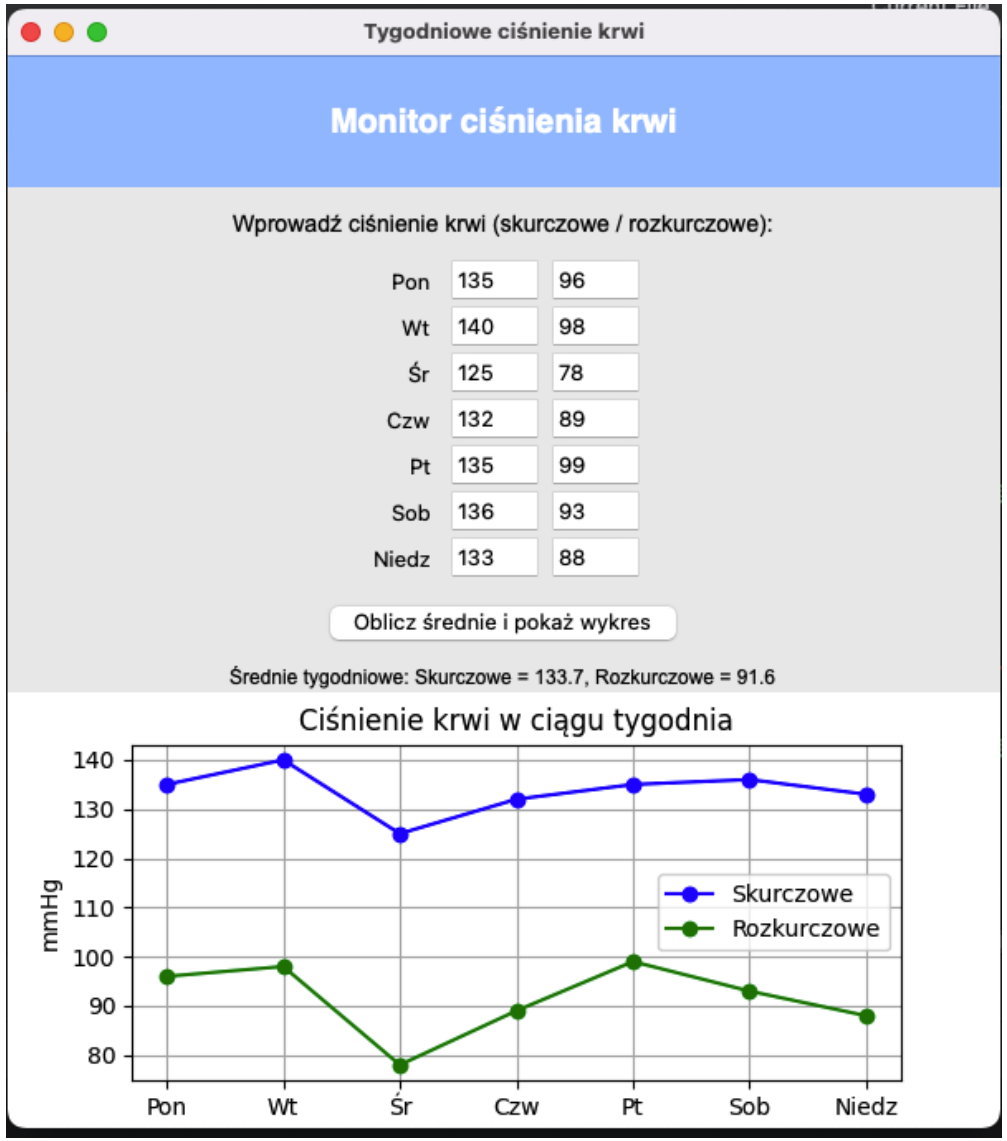


Zadania do samodzielnego wykonania

Zadanie 9 (na 6)

Utwórz w dolownym języku aplikację do monitorowania ciśnienia. Nastepnie wykonaj testy działania na systemie operacyjnym Windows oraz Linux, ChromeOS lub OS i porównaj wyniki.

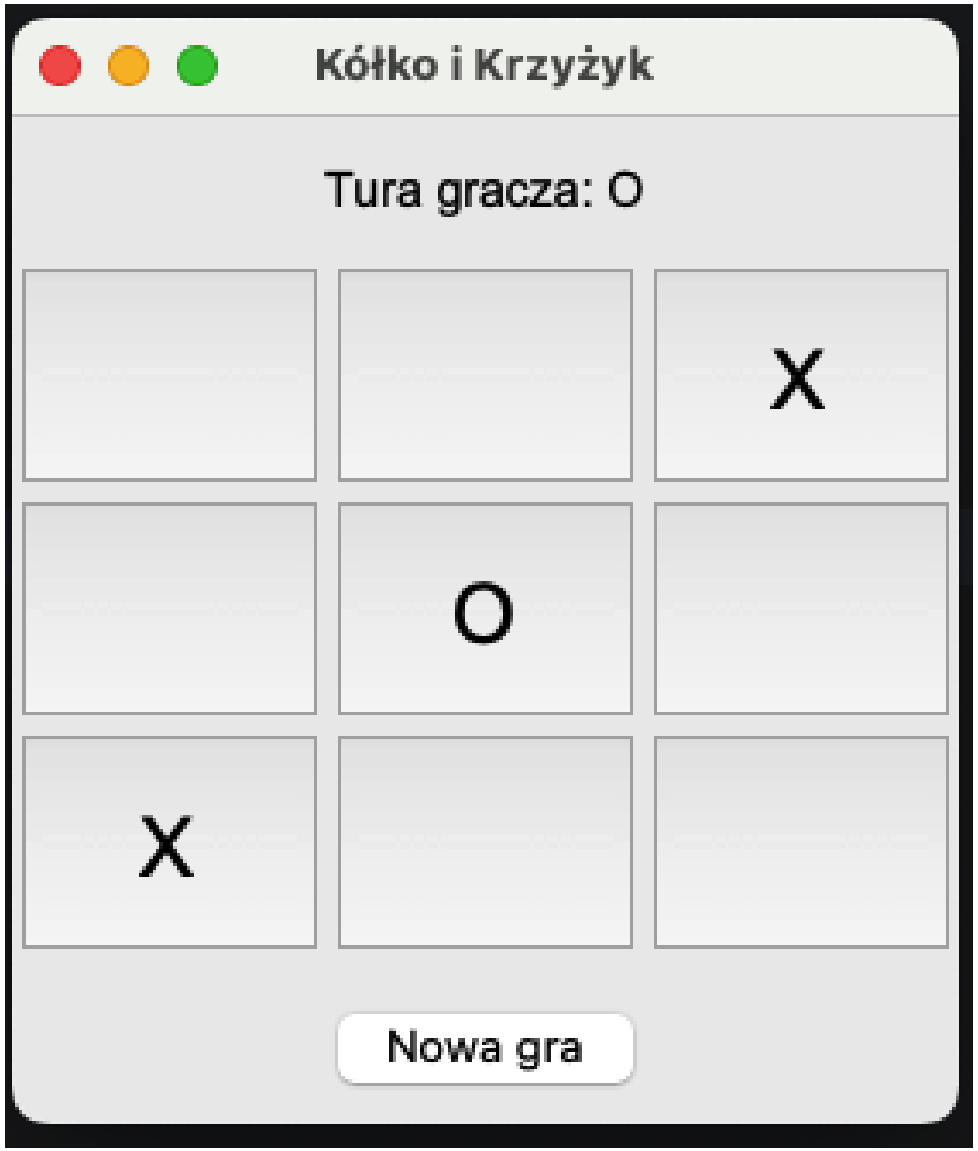
Dla potrzeb użytkownika przygotuj instrukcję obsługi w formie broszury (skorzystaj np. z Canva).



Zadania do samodzielnego wykonania

Zadanie 10 (na 5)

Wykonaj grę w dowolnym języku programowania według załączonego wzoru. Następnie przygotuj w formie komentarzy dokumentację projektu zgodną ze wzorem egzaminacyjnym.



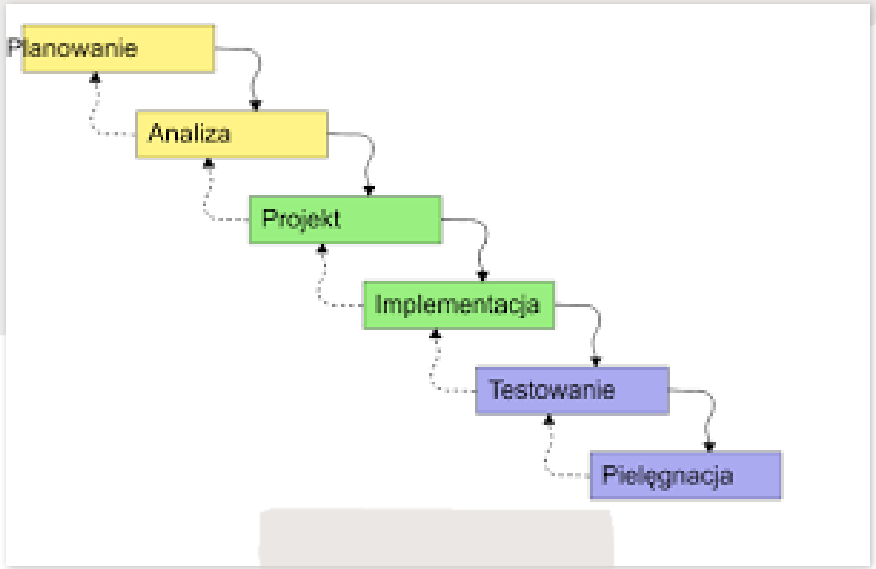
Metodyki programowania

- **Czym są metodyki programowania?**
  Metodyki programowania to zestawy zasad, praktyk i procesów, które definiują, jak powinien przebiegać proces tworzenia oprogramowania od planowania do wdrożenia i utrzymania.
- **Dlaczego warto stosować metodyki?**
- Zapewniają organizację pracy zespołu.
- Pomagają w planowaniu, kontrolowaniu i monitorowaniu postępu.
- Ułatwiają wykrywanie i szybkie reagowanie na błędy oraz zmiany wymagań.
- Podnoszą jakość końcowego produktu.
- **Rola metodyk**
  Dzięki metodykom możliwa jest standaryzacja procesów, co ułatwia współpracę w zespole i dostarczenie produktu spełniającego oczekiwania klienta.

Klasyfikacja metodyk

- **Metodyki tradycyjne (kaskadowe)** – proces przebiega liniowo, etap po etapie, od analizy wymagań do wdrożenia.
- **Metodyki zwinne (Agile)** – proces jest iteracyjny, elastyczny, z ciągłym dostarczaniem działających fragmentów produktu i adaptacją do zmian.
- **Metodyki hybrydowe** – łączą elementy obu podejść, by dopasować proces do specyficznych potrzeb projektu i zespołu.

Metodyka kaskadowa



Klasyczna metodyka, gdzie proces tworzenia oprogramowania jest podzielony na sztywne fazy:

1. Analiza wymagań
2. Projektowanie
3. Implementacja
4. Testowanie
5. Wdrożenie
6. Utrzymanie

Zalety:

- Prosta i łatwa do zarządzania.
- Każdy etap ma jasno określony cel i dokumentację.

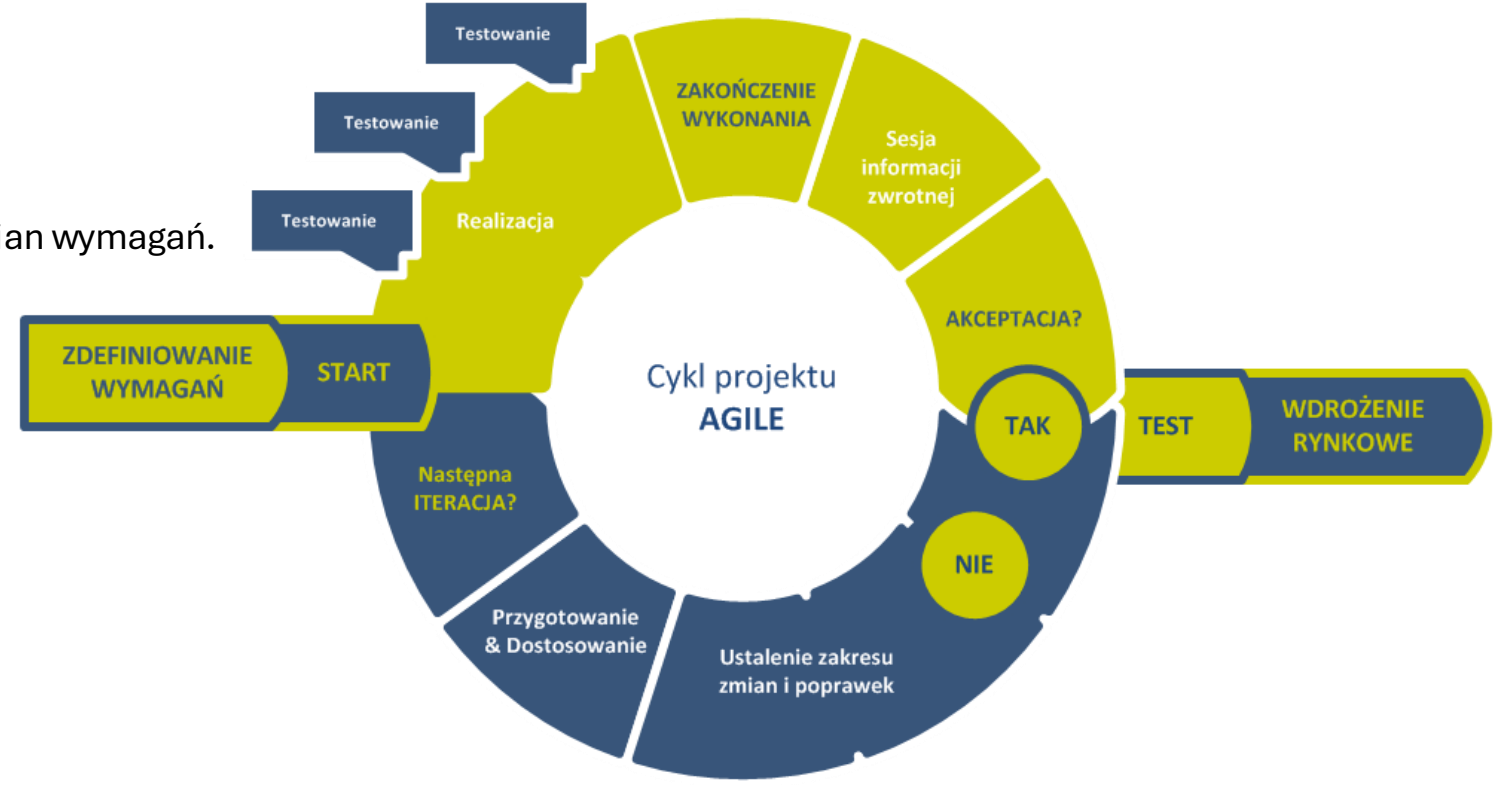
Wady:

- Brak elastyczności — zmiany w wymaganiach są trudne do wprowadzenia po zakończeniu fazy analizy.
- Problemy z przewidzeniem wszystkich wymagań na początku projektu.
- **Przykład zastosowania:**
  Systemy wbudowane, gdzie wymagania są stabilne i niezmienne (np. oprogramowanie do sterowania urządzeniami przemysłowymi).

AGILE – przykład metodyki zwinnej

Agile to filozofia zarządzania projektami, która kładzie nacisk na:

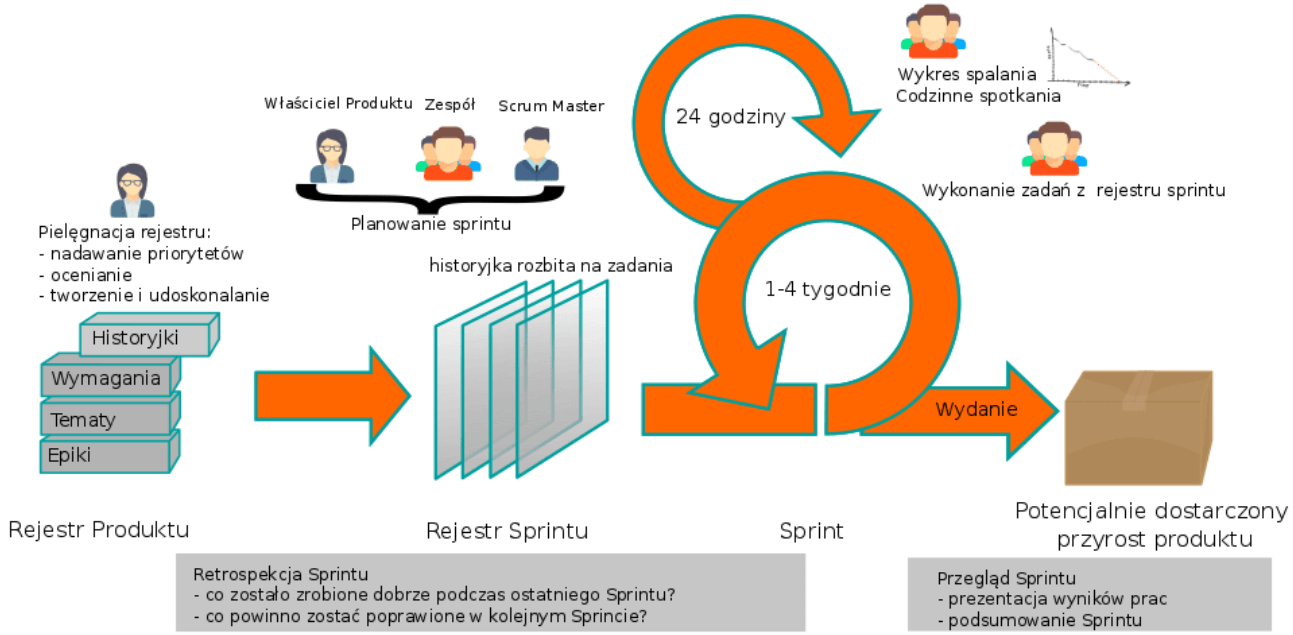
- Elastyczne reagowanie na zmiany.
- Ciągłe dostarczanie działającego oprogramowania w krótkich cyklach.
- Bliską współpracę z klientem.
- Samoorganizujące się zespoły.
- **Zalety:**
- Możliwość szybkiego dostosowania się do zmian wymagań.
- Lepsza komunikacja i transparentność.
- Częste testowanie i walidacja produktu.
- **Przykłady metodyk Agile:**
- **Scrum**
- **Kanban**
- **Extreme Programming (XP)**



SCRUM

Podstawy:

- Praca podzielona na sprinty — krótkie, powtarzalne cykle (zwykle 2-4 tygodnie).
- Zespół ma jasno określone role:
  - **Product Owner** – odpowiada za backlog i priorytety funkcji.
  - **Scrum Master** – dba o proces i usuwa przeszkody.
  - **Zespół Developerski** – tworzy oprogramowanie.
- **Ceremonie:**
- **Planowanie sprintu** – ustalenie, co będzie zrobione.
- **Daily stand-up** – codzienne, krótkie spotkanie zespołu.
- **Przegląd sprintu** – pokazanie efektów klientowi.
- **Retrospektywa** – analiza i usprawnienie procesu.
- **Narzędzia:**
- Backlog (lista wymagań)
- Tablica Scrum (np. Kanban board)
- Burndown chart (wykres postępu pracy)
- **Przykład:**
  Zespół tworzy aplikację webową, dostarczając nowe funkcje co sprint, zbiera feedback i na bieżąco poprawia produkt.



Kanban

Opis:

- Wizualizacja procesu pracy na tablicy Kanban, gdzie karty reprezentują zadania.
- Ograniczenie liczby zadań realizowanych jednocześnie (WIP — Work In Progress).
- Ciągłe doskonalenie procesu.

Zalety:

- Przejrzystość i kontrola nad procesem.
- Elastyczność — brak sztywnych iteracji.
- Łatwość wdrożenia.

Przykład:
Zespół wsparcia technicznego zarządza zgłoszeniami klientów, monitorując statusy i priorytety na tablicy Kanban.

Extreme Programming

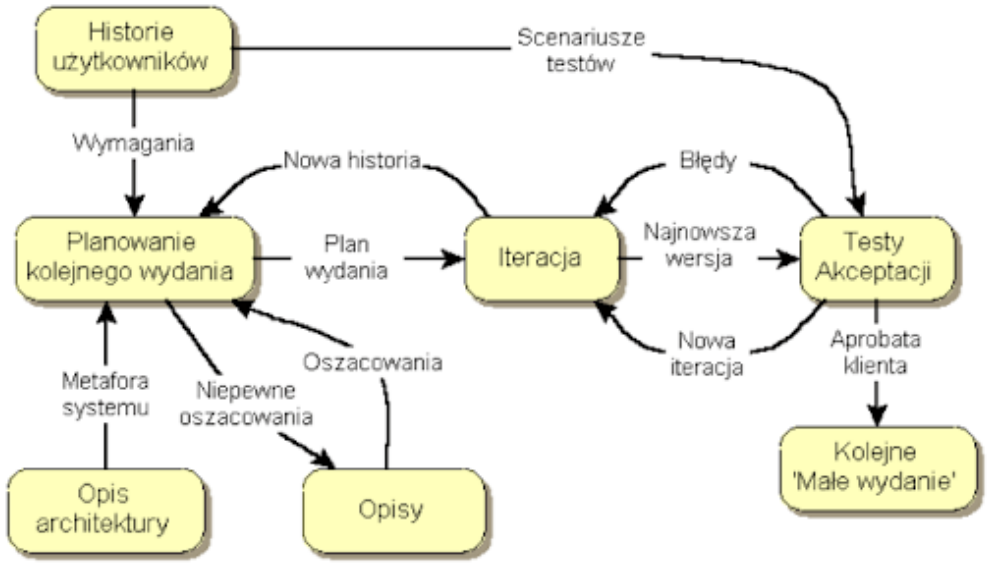
Charakterystyka:

- Nacisk na wysoką jakość kodu i praktyki inżynierskie.
- **TDD (Test Driven Development)** – pisanie testów przed kodem.
- Programowanie w parach (pair programming) – dwóch programistów współpracuje przy jednym komputerze.
- Ciągła integracja i refaktoryzacja kodu.

Zalety:

- Bardzo wysoka jakość i stabilność kodu.
- Szybsze wykrywanie błędów.

Przykład:
Projektowanie systemu bankowego, gdzie bezpieczeństwo i jakość są krytyczne.



Rys. 2.22 Schemat XP

Metodyki hybrydowe

Opis:

- Łączenie zalet metodyk tradycyjnych i zwinnych.
- Pozwala na formalizację i jednoczesną elastyczność.
- **Przykłady:**
- **Scrumban** — połączenie Scrum i Kanban.
- **Agile-Waterfall** — formalne etapy kaskadowe uzupełniane o iteracje Agile.
- **Zastosowanie:**
  Duże organizacje z regulacjami prawnymi, które wymagają dokumentacji i kontrolowanego procesu, ale potrzebują też elastyczności.

Przykłady zastosowania

Metodyka	Charakterystyka	Przykładowe zastosowania	Przykładowy projekt / branża
Waterfall (Kaskadowa)	Liniowy, sekwencyjny proces, brak elastyczności	Projekty ze stabilnymi wymaganiami, gdzie zmiany są minimalne	Systemy wbudowane, oprogramowanie dla przemysłu, systemy bankowe z sztywnymi wymaganiami
Scrum (Agile)	Iteracyjny, praca w sprintach, szybkie dostarczanie wartości	Projekty, gdzie wymagania szybko się zmieniają i jest silna współpraca z klientem	Aplikacje webowe, startupy, projekty IT z dynamicznym rozwojem
Kanban (Agile)	Wizualizacja pracy, ograniczanie WIP, ciągłe usprawnienia	Zespoły wsparcia technicznego, zarządzanie zgłoszeniami, prace utrzymaniowe	Helpdesk, zespoły DevOps, zespoły odpowiedzialne za utrzymanie systemów
Extreme Programming (XP)	Test Driven Development, programowanie w parach, wysoka jakość kodu	Projekty wymagające bardzo wysokiej jakości i niezawodności kodu	Systemy finansowe, aplikacje medyczne, krytyczne systemy bezpieczeństwa
Metodyki hybrydowe	Połączenie elementów tradycyjnych i zwinnych	Projekty dużych organizacji wymagające formalnej dokumentacji i jednocześnie elastyczności	Projekty rządowe, korporacyjne projekty IT, rozwój oprogramowania na potrzeby dużych firm

Podsumowanie - pytania

1. Czym jest projekt?
2. Podaj znaną Tobie klasyfikację metodyk.
3. Wymień wady metodyki kaskadowej.
4. Wymień najważniejsze cechy metodyki AGILE.
5. Wymień cechy Extreme Programming.
6. Wymień przykłady zastosowania SCRUM i Kanban.

Testowanie i dokumentacja na egzaminie INF.04

Część III. Dokumentacja aplikacji konsolowej

Wykonaj dokumentację do aplikacji utworzonych na egzaminie. W kodzie źródłowym aplikacji konsolowej za

pomocą komentarza utwórz nagłówek wybranej funkcji (metody) według wzoru z listingu 1. Komentarz

powinien znaleźć się nad lub pod nazwą funkcji. W miejscu nawiasów <> należy podać odpowiednie opisy.

UWAGA: Dokumentację należy umieścić w komentarzu (wieloliniowym lub kilku jednoliniowych). Znajdujący

się w listingu 1 wzór dokumentacji jest bez znaków początku i końca komentarza, gdyż te są różne dla

różnych języków programowania.

Listing 1. Wzór dokumentacji funkcji (liczba gwiazdek dowolna)

```
*******************************************

nazwa: <tu wstaw nazwę funkcji / metody>

opis: <co wykonuje funkcja / metoda?>

parametry: <opis parametru1, lub „brak”>

<opis parametru2>

...

zwracany typ i opis:<nazwa typu i opis co jest zwracane lub „brak”>

autor: <numer zdającego>

*******************************************
```

Wykonaj zrzuty ekranu dokumentujące uruchomienie aplikacji utworzonych podczas egzaminu. Zrzuty

powinny obejmować cały obszar ekranu monitora z widocznym paskiem zadań. Jeżeli aplikacja uruchamia

się, na zrzucie należy umieścić okno z wynikiem działania programu oraz otwarte środowisko

programistyczne z projektem lub okno terminala z kompilacją projektu. Jeżeli aplikacja nie uruchamia się

z powodu błędów kompilacji, należy na zrzucie umieścić okno ze spisem błędów i widocznym otwartym

środowiskiem programistycznym. Wykonać należy tyle zrzutów, ile interakcji podejmuje aplikacja.

Wymagane zrzuty ekranu:

– Aplikacja konsolowa – dowolna liczba zrzutów nazwanych *konsola1, konsola2,* ... (np. stan początkowy,

błędnie i poprawnie podana liczba kostek, kilka losowań, opcje ‘t’ oraz na końcu opcja ‘n’)

– Aplikacja mobilna – dowolna liczba zrzutów nazwanych *mobile1*, *mobile2,* ... (np. stan początkowy, po

wciśnięciu przycisku RZUĆ KOŚĆMI, po wciśnięciu przycisku „RESETUJ WYNIK”)

W edytorze tekstu pakietu biurowego utwórz plik z dokumentacją i nazwij go *egzamin.* Dokument powinien

zawierać informacje:

– Nazwę systemu operacyjnego, na którym pracował zdający

– Nazwy środowisk programistycznych, z których zdający korzystał na egzaminie

– Nazwę emulatora dla aplikacji mobilnej

– Nazwy użytych języków programowania

Zrzuty ekranu i dokument umieść w podfolderze *dokumentacja*.

UWAGA: Nagraj płytę z rezultatami pracy. W folderze z numerem zdającego powinny się znajdować

podfoldery: dokumentacja, konsolowa, mobilna. W folderze dokumentacja: pliki ze zrzutami oraz plik

egzamin. W folderze konsolowa: spakowany cały projekt aplikacji konsolowej, pliki źródłowe, opcjonalnie plik

wykonywalny. W folderze mobilna: spakowany cały projekt aplikacji mobilnej, pliki z kodem źródłowym

interfejsu i logiki. Po nagraniu płyty sprawdź poprawność nagrania. Opisz płytę numerem zdającego

i pozostaw na stanowisku, zapakowaną w pudełku wraz z arkuszem egzaminacyjnym.

Aplikacja konsolowa

```
/*********************************************
nazwa: rzucKoscmi
opis: Funkcja losuje podaną liczbę kostek i zwraca wyniki rzutów.
parametry: liczbaKostek – liczba kostek do wylosowania
zwracany typ i opis: int[] – tablica wyników rzutów
autor: 12345
*********************************************/
public static int[] rzucKoscmi(int liczbaKostek) {
   Random rnd = new Random();
   int[] wyniki = new int[liczbaKostek];
   for (int i = 0; i < liczbaKostek; i++) {
      wyniki[i] = rnd.nextInt(6) + 1;
   }
   return wyniki;
}
```

Komentarz w postaci nagłówka z opisem jak w przykładzie. W zadaniu może być również określone czy opisać zawartość klasy lub metody. Wtedy powtarzamy opis, parametry i zwracany typ.

Zrzuty ekranu

Musisz zrobić screeny obejmujące **cały ekran** (z paskiem zadań, środowiskiem programistycznym i uruchomioną aplikacją).

- **Aplikacja konsolowa** → nazwy plików konsola1.png, konsola2.png, ...
  - np. początek działania programu
  - błędnie podana liczba kostek
  - poprawnie podana liczba kostek i wynik losowania
  - kilka losowań
  - opcja t (ponowne losowanie)
  - opcja n (zakończenie)
- **Aplikacja mobilna** → nazwy plików mobile1.png, mobile2.png, ...
  - np. ekran startowy
  - po kliknięciu „RZUĆ KOŚĆMI”
  - po kliknięciu „RESETUJ WYNIK”

Dokument w edytorze tekstu

Plik ma zawierać:

- **System operacyjny** – np. *Windows 11 Pro 64-bit*
- **Środowiska programistyczne** – np. *Visual Studio 2022* (dla aplikacji konsolowej) oraz *Android Studio* (dla mobilnej)
- **Emulator dla aplikacji mobilnej** – np. *Pixel 5 API 34 (Android Emulator)*
- **Języki programowania** – np. *C#* (dla konsolowej), *Java* (dla mobilnej)

Przykład dokumentu tekstowego

EGZAMIN – Dokumentacja aplikacji konsolowej i mobilnej

Numer zdającego: ……

1. Informacje ogólne

System operacyjny: Windows 11 Pro 64-bit

Środowiska programistyczne:

- Visual Studio 2022 (aplikacja konsolowa)

- Android Studio Electric Eel (aplikacja mobilna)

Emulator użyty do uruchomienia aplikacji mobilnej: Pixel 5 API 34 (Android Emulator)

Języki programowania:

- C# (aplikacja konsolowa)

- Java (aplikacja mobilna)

2. Dokumentacja kodu

Przykładowy nagłówek funkcji w aplikacji konsolowej:

```
/*********************************************
nazwa: RzucKoscmi
opis: Funkcja losuje podaną liczbę kostek i zwraca wyniki rzutów.
parametry: liczbaKostek – liczba kostek do wylosowania
zwracany typ i opis: int[] – tablica wyników rzutów
autor: [tu wpisz numer zdającego]
*********************************************/
```

3. Zrzuty ekranu

Aplikacja konsolowa

- konsola1 – ekran początkowy aplikacji

- konsola2 – błędnie podana liczba kostek

- konsola3 – poprawnie podana liczba kostek i wynik losowania

- konsola4 – kolejne losowania

- konsola5 – zakończenie programu (opcja 'n')

(w tym miejscu wstaw screeny konsola1.png, konsola2.png, ...)

Aplikacja mobilna

- mobile1 – ekran początkowy

- mobile2 – po kliknięciu przycisku „RZUĆ KOŚĆMI”

- mobile3 – po kliknięciu przycisku „RESETUJ WYNIK”

(w tym miejscu wstaw screeny mobile1.png, mobile2.png, ...)

4. Struktura folderów na płycie

```
12345/
│
├── dokumentacja/
│   ├── egzamin.docx
│   ├── konsola1.png, konsola2.png, ...
│   ├── mobile1.png, mobile2.png, ...
│
├── konsolowa/
│   ├── projekt_konsolowy.zip
│   ├── Program.cs
│   └── Program.exe (opcjonalnie)
│
├── mobilna/
│   ├── projekt_mobilny.zip
│   ├── MainActivity.java
│   └── activity_main.xml
```

5. Uwagi końcowe

Po nagraniu płyty sprawdzono poprawność nagrania.
Płyta została opisana numerem zdającego i pozostawiona na stanowisku w pudełku wraz z arkuszem egzaminacyjnym.

Struktura dokumentów nagranych na płytę

```
12345/                <-- numer zdającego
│
├── dokumentacja/
│   ├── egzamin.docx
│   ├── konsola1.png
│   ├── konsola2.png
│   ├── mobile1.png
│   ├── mobile2.png
│   ...
│
├── konsolowa/
│   ├── projekt_konsolowy.zip  (cały projekt spakowany)
│   ├── Program.cs
│   └── Program.exe   (opcjonalnie)
│
├── mobilna/
│   ├── projekt_mobilny.zip   (cały projekt Android Studio)
│   ├── MainActivity.java
│   └── activity_main.xml
```

Ostatnie kroki:

1. Spakuj projekty i wrzuć w odpowiednie foldery.
2. Dołącz wszystkie screeny i dokument.
3. Nagraj płytę z folderem nazwanym numerem zdającego.
4. Opisz płytę numerem zdającego.
5. Zostaw płytę w pudełku razem z arkuszem.

Test aplikacji na egzaminie

Uruchomienie aplikacji

- sprawdzasz, czy aplikacja kompiluje się bez błędów i startuje.
- robisz zrzut ekranu z okna programu + środowiska programistycznego/terminala.

Sprawdzenie podstawowych funkcji

- np. czy przycisk działa, czy można dodać/usunąć dane, czy formularz zapisuje do pliku/bazy, czy wyświetlają się wyniki obliczeń.
- robisz **kilka screenów** pokazujących różne przypadki.

Test poprawnych danych (scenariusz pozytywny)

- np. wpisujesz prawidłowe dane w formularzu → sprawdzasz, czy wynik jest poprawny.

Test błędnych danych (scenariusz negatywny)

- np. wpisujesz pustą wartość, litery zamiast liczby → aplikacja powinna to obsłużyć (komunikat, brak błędu krytycznego).

Test zamknięcia/zakończenia działania

- sprawdzasz, czy aplikacja kończy pracę zgodnie z założeniem (np. opcja „n” w konsoli).

Test aplikacji na egzaminie

Test aplikacji konsolowej:

- Podano błędne dane (screen konsola2.png) – program poprawnie wyświetlił komunikat o błędzie.

- Podano poprawne dane (screen konsola3.png) – program zwrócił wyniki rzutów kostką.

- Wybrano opcję 't' (screen konsola4.png) – program powtórzył losowanie.

- Wybrano opcję 'n' (screen konsola5.png) – program zakończył działanie.

Test aplikacji mobilnej:

- Stan początkowy aplikacji (screen mobile1.png).

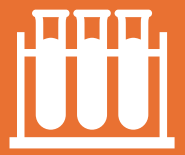
- Po wciśnięciu przycisku „RZUĆ KOŚĆMI” (screen mobile2.png) – wyświetlił się wynik.

- Po wciśnięciu przycisku „RESETUJ WYNIK” (screen mobile3.png) – wynik został wyzerowany.

Tabela testów

Nr testu	Scenariusz testowy	Dane wejściowe	Oczekiwany wynik	Wynik testu
1	Uruchomienie aplikacji	brak	Program uruchamia się bez błędów	✓ działa
2	Podanie błędnej liczby kostek (np. 0)	0	Komunikat o błędzie / ponowne wczytanie danych	✓ działa
3	Podanie poprawnej liczby kostek	3	Wyświetlone 3 wyniki losowania kostką	✓ działa
4	Wybranie opcji „t” (ponowne losowanie)	t	Aplikacja ponawia losowanie	✓ działa
5	Wybranie opcji „n” (zakończenie programu)	n	Program kończy działanie	✓ działa
6	Uruchomienie aplikacji mobilnej	brak	Emulator wyświetla ekran startowy	✓ działa
7	Kliknięcie przycisku „RZUĆ KOŚĆMI”	klik przycisku	Wyświetlenie wyniku losowania	✓ działa
8	Kliknięcie przycisku „RESETUJ WYNIK”	klik przycisku	Wynik zostaje wyzerowany	✓ działa

Dokument tekstowy z załączoną tabelą testów



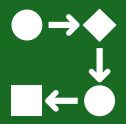
Zwykle testy w postaci tabeli załącza się przed uwagami końcowymi.

Płytę z nagranymi danymi należy sprawdzić po nagraniu. Płyta będzie opisana numerem PESEL oraz nazwą kwalifikacji.

Pamiętaj aby uważnie czytać nie tylko treść zadań, ale również wymagania dotyczące dokumentacji.

Koniec?

Egzamin INF.04 nie należy do łatwych. Należy go dobrze zaplanować aby zmieścić się w czasie. Pamiętaj aby wyraźnie przeczytać polecenia oraz zwrócić uwagę na zrzuty ekranu gotowej aplikacji umieszczone w zadaniach. Często jest tak, że autor nie do końca precyzyjnie opisuje zadanie.



Pamiętaj! Tam, gdzie nie ma dokładnie sprecyzowanego sposobu wykonania, jest możliwość wykorzystania własnej inwencji twórczej. To ważne – możesz zyskać dodatkowe punkty.