



wydział  
elektrotechniki  
elektroniki  
informatyki  
i automatyki



# METODY TESTOWANIA OPROGRAMOWANIA

*LABORATORIUM*

*Behavior Driven Development*

*Jakość testu*

wersja 1.0

przygotował:  
dr inż. Radosław Adamus

## Efekty:

Po ukończeniu laboratorium będziesz:

1. Potrafił opracować kod testu strony internetowej z wykorzystaniem wzorca PageObject
2. Potrafił wykorzystać narzędzia z bibliotek Selenium oraz Serenity do zbudowania warstw testu
3. Potrafił wygenerować raport z testu

## Wymagania wstępne:

1. Posiadanie konta na platformie Github.

## Narzędzia:

1. Eclipse IDE
2. Cucumber JVM Eclipse Plugin
3. Git
4. Nodejs

## Reguły wykonywania ćwiczeń laboratoryjnych:

1. Zmiany należy zatwierdzać często. Zatwierdzenie zbiorcze zmian na koniec laboratorium równoważne jest z jego niezaliczeniem.
2. Zmiany zatwierdzone w repozytorium kontroli wersji muszą posiadać znaczące komentarze.
3. Git powinien być tak skonfigurowany, aby zatwierdzane zmiany były identyfikowane danymi studenta (adres email oraz nazwisko).
4. W przypadku nieukończenia zadania w trakcie zajęć rezultat pośredni powinien być, po zatwierdzeniu w repozytorium lokalnym wypchnięty do macierzystego repozytorium na GitHub.


## Opis laboratorium:

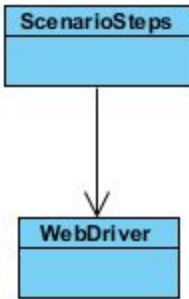
### 1. Wprowadzenie

W ramach laboratorium wykorzystasz dwa projekty:

- Pierwszy z nich (flying-high-pol) reprezentuje testowaną aplikację i znajduje się w repozytorium: <https://github.com/mto-lab/flying-high-pol>. Projekt wykorzystuje platformę nodejs. Uruchomienie aplikacji wymaga istnienia polecenia http-server (instalacja polecenia w konsoli: `npm install -g http-server`). Komenda uruchamiająca (wykonana z poziomu katalogu projektu): `http-server app`.
- Drugi projekt (flying-high-ui-test) implementuje testy webowe i jest aplikacją Java i znajduje się w repozytorium <https://github.com/mto-lab/flying-high-ui-tests>. W projekcie flying-high-ui-test jest zaimplementowany scenariusz dla cechy systemu reprezentującej.

Warstwa testów akceptacyjnych odwołuje się bezpośrednio do struktury strony webowej (wykorzystując WebDriver z biblioteki Selenium). Takie rozwiązanie powoduje, że kod testów akceptacyjnych jest wrażliwy na każdą zmianę struktury strony i jako taki, musi być przy każdej takiej zmianie również zmodyfikowany.

 feature



Rys.1 Bezpośrednia zależność scenariusza od warstwy technicznej

## 1. Zadania


1.0 Zmiany zatwierdzaj do gałęzi 'solution'. W trakcie wykonywania zadań weź pod uwagę tylko cechę `user_authentication.feature` oraz klasę kroków `UserAuthenticationSteps`. Uruchamianie testów dla tej cechy odbywa się za pośrednictwem klasy `UserAuthenticationTests`.

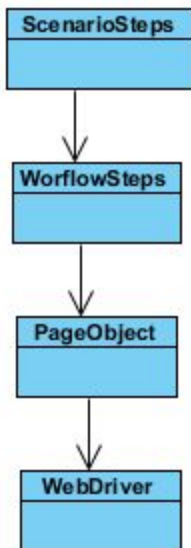
Pozostałe cechy oraz definicje kroków nie są zaimplementowane.

1.1 Wykorzystaj typ wyliczeniowy `edu.iis.mto.bdd.model.FrequentFlyerMember` w definicji kroków scenariusza jako parametr metod w miejsce parametru `user`. Wykorzystaj instancje w kodzie metod. Sprawdź jak zachowuje się test przy zmianie imienia w specyfikacji.

1.2 Dokonaj refaktoryzacji kodu testów poprzez wydzielenie warstwy technicznej i uniezależnienie implementacji wykonywalnych specyfikacji od szczegółów implementacyjnych dotyczących struktury strony webowej. Jako interfejs warstwy technicznej wykorzystaj wzorzec `PageObject`. Dla scenariusza autoryzacji powinny istnieć dwie klasy `PageObject`. Jedna dla strony logowania, druga dla strony domowej zalogowanego użytkownika.

1.3 Aby uniknąć bezpośredniego operowania pojęciami z warstwy technicznej na poziomie wykonywalnych specyfikacji zaimplementuj dodatkową warstwę przepływ prac (workflow), która będzie reprezentowała logikę działań jakie musi wykonać użytkownik w trakcie realizacji scenariusza biznesowego. Utwórz klasę reprezentującą tę warstwę i nazwij ją `'AuthenticationWorkFlowSteps'`. Umieść klasę w odpowiednim pakiecie Dla przykładowego scenariusza kroki przepływu pracy powinny reprezentować wpisanie danych uwierzytelniających oraz weryfikację istnienia napisu powitalnego.

 feature



Rys 2. Wydzielone warstwy separujące scenariusz od szczegółów technicznych

#### 1.4 Generowanie raportów z testów akceptacyjnych

Umieść adnotacje `@Steps` oraz `@Step` z biblioteki Serenity w odpowiednich miejscach implementacji aby wprowadzić informacje pozwalające na automatyczną generację raportów. Wygenerowanie raportów wymaga zmiany runnera Junit z Cucumber na CucumberWithSerenity.

### Przypisy

[1] John Ferguson Smart, BDD w działaniu. Sterowanie zachowaniem w rozwoju aplikacji, Helion 2016.