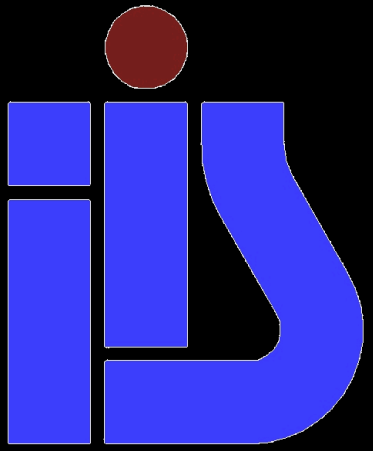


Metody testowania oprogramowania

Jakość testu akceptacyjnego

PRZYJAZNY DLA UŻYTKOWNIKA, rysował J.D. „Illiad" Frazer

Legenda

- Cechy dobrego testu akceptacyjnego
- Zmienność a kod testu akceptacyjnego
- Dane dla testu
- Architektura warstwowa testu
- Testy webowe
  - Wzorzec PageObject
  - Wykorzystanie narzędzi
  - Problemy z testami webowymi

Dobre automatyczne testy akceptacyjne

- Powinny:
  - wyraźnie komunikować swój zamiar
  - dostarczać sensownego sprzężenia zwrotnego
  - być wiarygodne
  - być łatwe w utrzymaniu

Konfiguracja testu

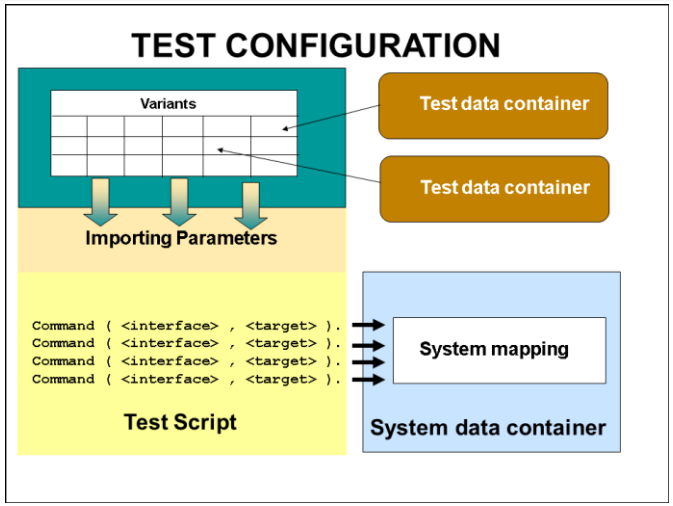
- System musi być w prawidłowym i dobrze znanym stanie początkowym
  - Potrzeba inicjalizacji usług, zasobów, baz danych, systemów plików
- To też należy automatyzować.
- Praktyki
  - Dedykowana instancja bazy danych
  - Wykorzystanie narzędzi automatyzacji (np. Puppet, Chef) do tworzenia środowisk wirtualnych dla potrzeb testów

„Wrażliwy” test

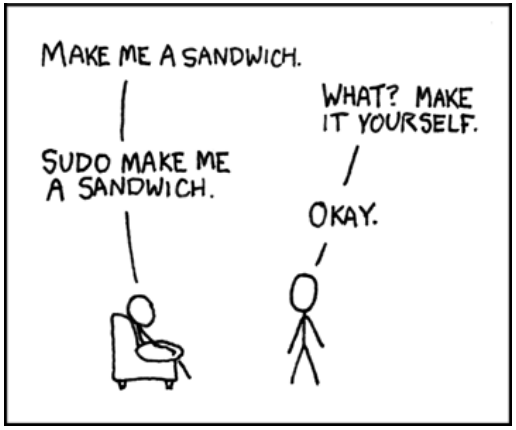
```
@When("^(.*) authenticates with a valid and password$")
public void whenJaneAuthenticatesWithAValidEmailAddressAndPassword(String userEmail) {
    driver.get("http://localhost:8080/#/welcome");
    driver.findElement(By.name("email")).sendKeys(userEmail);
    driver.findElement(By.name("password")).sendKeys("s3cr3t");
    driver.findElement(By.name("signin")).click();

}

@Then("^(.*) should be given access to (?:her|his) account$")
public void thenTheUserShouldBeGivenAccessToAccount(String userName) {
    assertThat(driver.findElement(By.id("welcome-message")).getText(), equalTo("Witaj " + userName));

}

@Given("^(.*) has logged on$")
public void aUserHasLoggedOnAs(String userEmail) {
    driver.get("http://localhost:8080/#/welcome");
    driver.findElement(By.name("email")).sendKeys(userEmail);
    driver.findElement(By.name("password")).sendKeys("s3cr3t");
    driver.findElement(By.name("signin")).click();
}
```

Od wrażliwego do odpornego testu

- Konfigurowanie danych

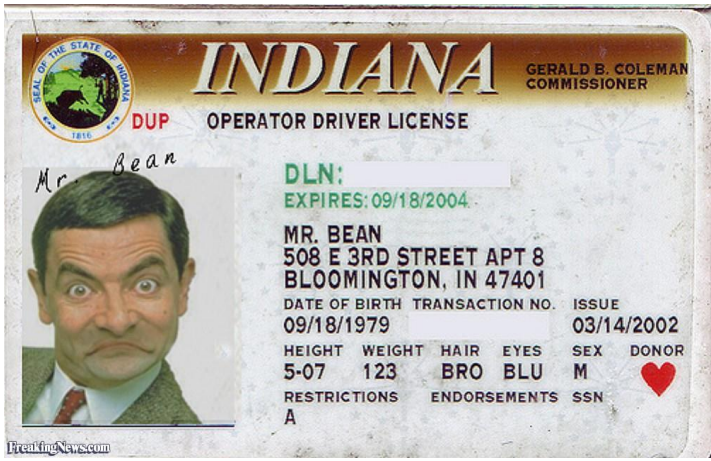


- Separacja „co” od „jak”
  - Deklaratywny -> imperatywny



Konfigurowanie danych specyficznych dla scenariuszy

- Osoby / znane encje
  - Technika konfiguracji danych dla scenariuszy
    - Fikcyjne postacie reprezentujące różne typy użytkowników w systemie
    - Szczegółowo opisane
    - Posiadające dobrze znaną nazwę



Wykorzystanie znanych encji w cucumber-jvm

```
public enum FrequentFlyerMember {

    Jane("janina.kowalska@acme.com", "Janina", "Kowalska", "s3cr3t"),
    Joe("janusz.bloger@acme.com", "Janusz", "Bloger", "s3cr3t2");
```

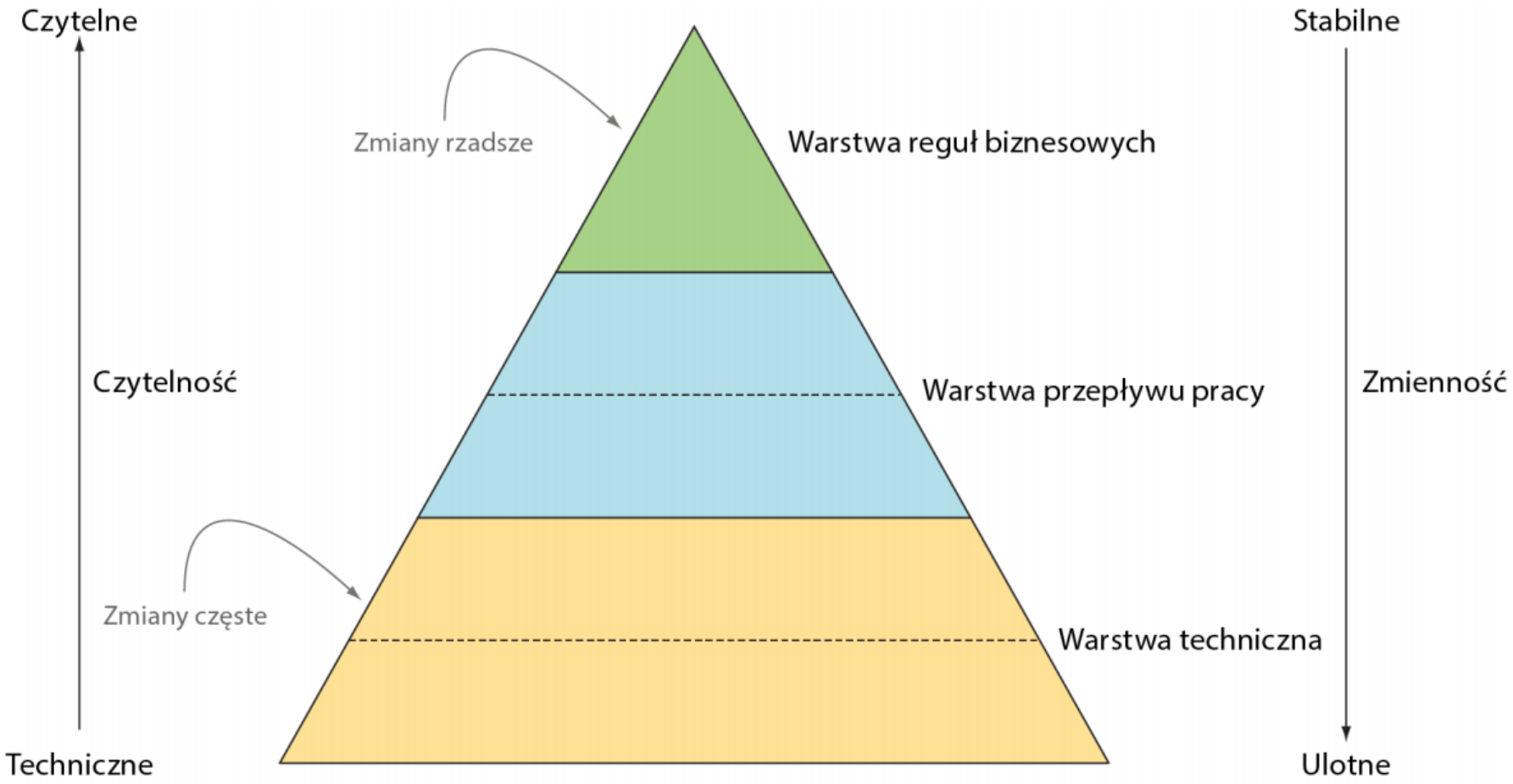
```
Scenario: Successful authentication
  Given Jane is a registered Frequent Flyer
  When Jane authenticates with a valid email address and password
  Then Jane should be given access to her account
```

```
    @When("^(.*) authenticates with a valid email address and password$")
    public void whenJaneAuthenticatesWithAValidEmailAddressAndPassword(FrequentFlyerMember user) {
        driver.get("http://localhost:8080/#/welcome");
        driver.findElement(By.name("email")).sendKeys(user.getEmail());
        driver.findElement(By.name("password")).sendKeys(user.getPassword());
        driver.findElement(By.name("signin")).click();

    }

    @Then("^(.*) should be given access to (?:her|his) account$")
    public void thenTheUserShouldBeGivenAccessToAccount(FrequentFlyerMember user) {
        assertThat(driver.findElement(By.id("welcome-message")).getText(), equalTo("Witaj " + user.getFirstName()));

    }
```

Oddzielenie warstwy „co” od warstwy „jak”

- Cel
  - minimalizacja wpływu zmian technicznych na automatyczne kryteria akceptacji
  - jasne, zrozumiałe testy
- Metoda: Izolacja
  - niskopoziomowych szczegółów implementacji testów (np. układ ekranu, nazwy pól, nazwy bibliotek)
  - od
    - Wysokopoziomowych reguł biznesowych
- Sposób
  - Wprowadzenie warstw abstrakcji

Piramida testu akceptacyjnego



Źródło: John Ferguson Smart, BDD w działaniu, Helion 2016

Warstwa reguł biznesowych

- Opisuje testowane wymagania w wysokopoziomowych kategoriach biznesowych.
  - W BDD przyjmuje formę scenariusza
    - Koncentruje się na rezultatach biznesowych
    - Nie zajmuje się sposobem dostarczenia rezultatów
  - Niewielkie prawdopodobieństwo zmiany

```
Scenariusz: Rejestrowanie online nowego konta uczestnika programu Frequent Flyer
Zakładając Janina nie jest uczestnikiem programu Frequent Flyer
Gdy Janina zarejestruje się w systemie w celu uzyskania nowego konta
Wtedy do Janina powinien być wysłany e-mail z potwierdzeniem zawierający jej numer
I Janina powinna uzyskać 500 punktów premii
```

```
Scenario: Successful authentication
  Given Jane is a registered Frequent Flyer
  When Jane authenticates with a valid email address and password
  Then Jane should be given access to her account
```

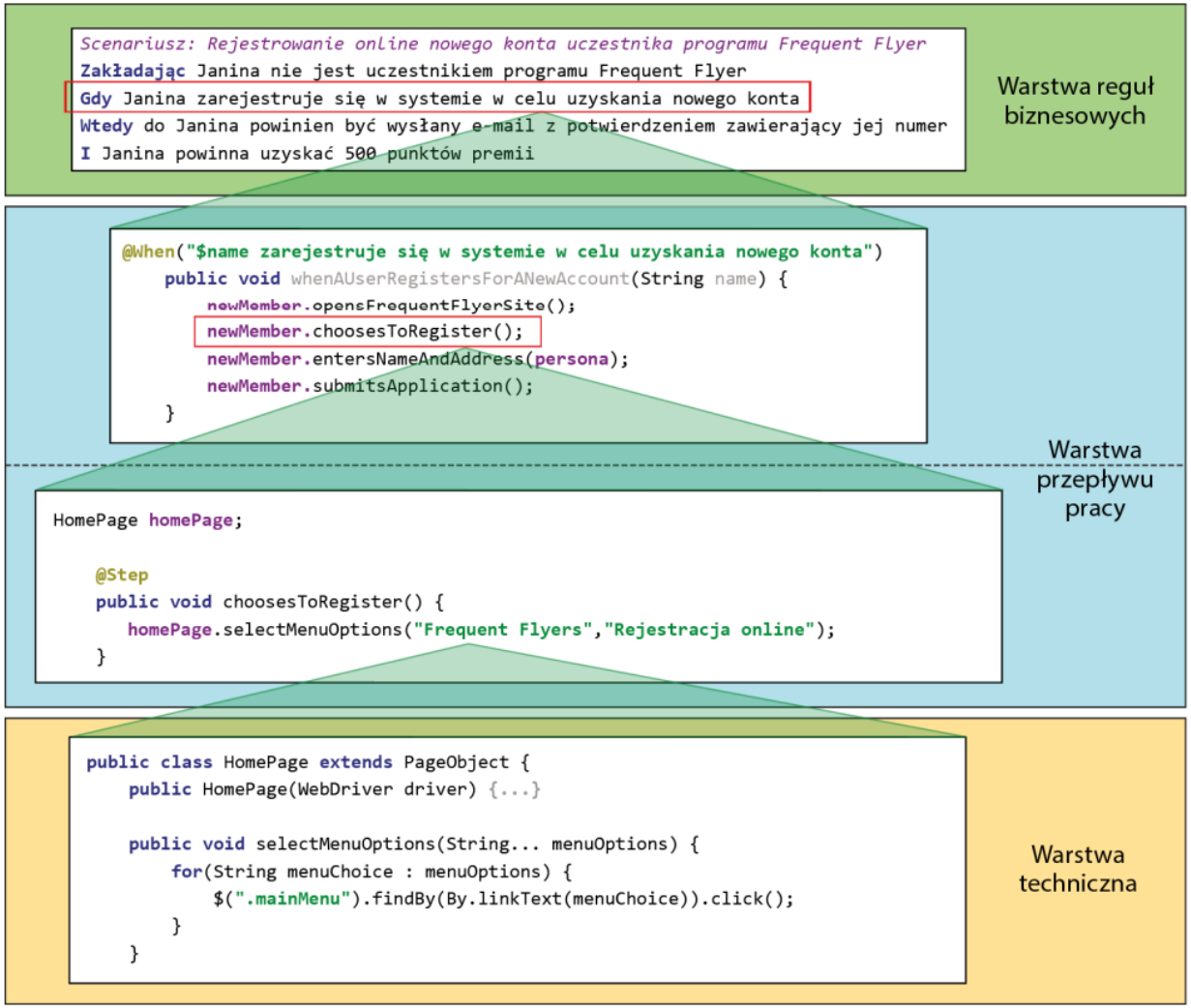
Warstwa przepływu pracy (ang. workflow)

- Reprezentacja akcji wykonywanych przez użytkownika w celu realizacji celu biznesowego (proces):
  - Np.: rejestracja nowego konta
    - Wejście na stronę
    - Wybranie opcji rejestracji nowego konta
    - Wprowadzenie danych rejestracyjnych
    - Zatwierdzenie wniosku
  - Zestaw kroków prowadzących do realizacji celu biznesowego ma większe prawdopodobieństwo zmiany niż sama reguła biznesowa

Warstwa techniczna

- Szczegółowo reprezentuje sposób interakcji z systemem
  - Np. jak wejść na stronę, jak zidentyfikować potrzebne pola
  - Wymagana wiedza na temat implementacji systemu (np. struktura strony, dostępność usług)
- Dobrze opracowany interfejs warstwy technicznej:
  - uniezależnia warstwy wyższe od zmian na niskim poziomie.
  - Umożliwia implementację reguł biznesowych oraz przepływu prac przed implementacją interfejsu użytkownika.

Architektura warstwowa



Źródło: John Ferguson Smart, BDD w działaniu, Helion 2016

Wzorzec PageObject

- Izoluje kodu testów od technicznych aspektów implementacji strony webowej,
  - Tworzy API aplikacji – ukrywa API dotyczące HTML.
- Należy do warstwy technicznej dostarczając wygodnych usług dla warstwy przepływu pracy.
- Pomimo nazwy klasa PageObject nie musi reprezentować pojedynczej strony
  - Aplikacje typu SPA – PageObject na stan strony
  - Reprezentacja ważnych części ekranu (np. główny pasek menu)

Wykorzystanie wzorca PageObject

```java
@When("^(.*) authenticates with a valid email address and password$")
public void whenJaneAuthenticatesWithAValidEmailAddressAndPassword(FrequentFlyerMember user) {
    LoginPage loginPage = new LoginPage(driver);
    loginPage.open();
    loginPage.signinWithCredentials(user.getEmail(), user.getPassword());

}

@Then("^(.*) should be given access to (?:her|his) account$")
public void thenTheUserShouldBeGivenAccessToAccount(FrequentFlyerMember user) {
    HomePage homepage = new HomePage(driver);

    assertThat(homepage.getWelcomeMessage(), equalTo("Witaj " + user.getFirstName()));

}
```

```java
public class LoginPage {

    private final WebDriver driver;

    public LoginPage(WebDriver driver) {
        this.driver = driver;
    }

    public void open() {
        driver.get("http://localhost:8080/#/welcome");
    }

    public void signinWithCredentials(String userEmail, String userPassword) {
        driver.findElement(By.name("email")).sendKeys(userEmail);
        driver.findElement(By.name("password")).sendKeys(userPassword);
        driver.findElement(By.name("signin")).click();
    }
}
```

Wsparcie Page Object w Selenium/WebDriver

```
public class LoginPage  {

    @FindBy(name="email")
    private WebElement email;
    private WebElement password;

    private WebElement signin;

    private final WebDriver driver;

    public LoginPage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }

    public void open() {
        driver.get("http://localhost:8080/#/welcome");
    }

    public void signinWithCredentials(String userEmail, String userPassword) {
        email.sendKeys(userEmail);
        password.sendKeys(userPassword);
        signin.click();
    }

}
```

Opcjonalne jeżeli nazwa pola klasy odpowiada atrybutowi ‘id’ lub ‘name’ elementu na stronie

Serenity

- Biblioteka dla środowiska Java
- Ułatwia implementacje kodu testów
- Udostępnia mechanizmy automatycznego generowania raportów

PageObject z wykorzystaniem Serenity

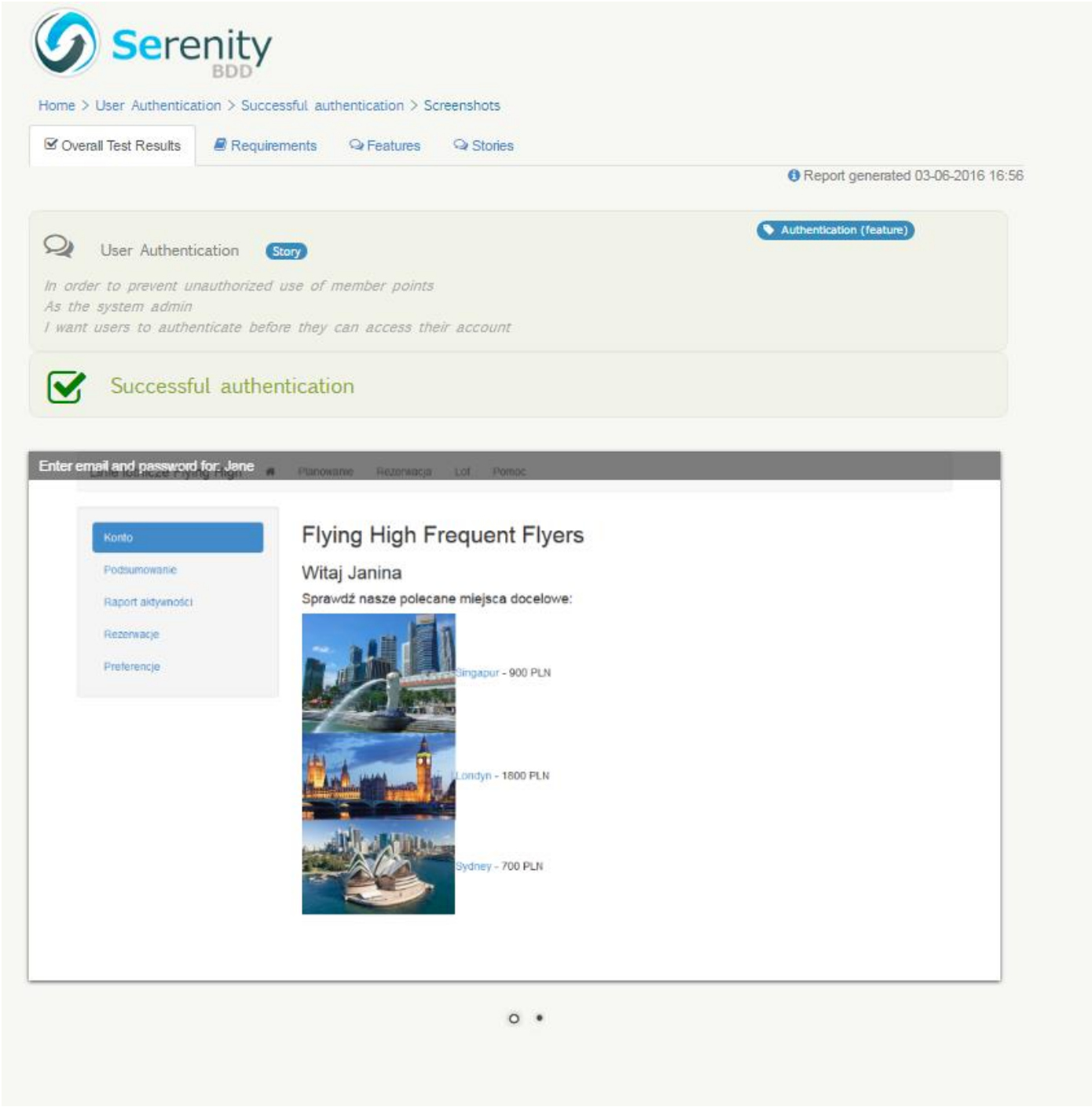
```
@DefaultUrl("http://localhost:8080/#/welcome")
public class LoginPageSerenity extends PageObject {

    private WebElement email;
    private WebElement password;

    private WebElement signin;

    public void signinWithCredentials(String userEmail, String userPassword) {
        email.sendKeys(userEmail);
        password.sendKeys(userPassword);
        signin.click();
    }

}
```

Przepływ pracy z adnotacjami kroków Serenity

```java
public class UserAuthenticationStepsSerenity {

    @Steps
    AuthenticationWorkFlowSteps authenticationWorkFlow;

    @When("^(.*) authenticates with a valid email address and password$")
    public void whenJaneAuthenticatesWithAValidEmailAddressAndPassword(FrequentFlyerMember user) {
        authenticationWorkFlow.enterEmailAndPasswordFor(user);
    }

    @Then("^(.*) should be given access to (?:her|his) account$")
    public void thenTheUserShouldBeGivenAccessToAccount(FrequentFlyerMember user) {
        authenticationWorkFlow.verifyWelcomeMessageFor(user);
    }

    @Given("^(.*) has logged on$")
    public void aUserHasLoggedOnAs(FrequentFlyerMember user) {
        authenticationWorkFlow.enterEmailAndPasswordFor(user);
    }
```

```java
public class AuthenticationWorkFlowSteps {
    LoginPage loginPage;
    HomePage homePage;

    @Step
    public void enterEmailAndPasswordFor(FrequentFlyerMember user) {
        loginPage.open();
        loginPage.signinWithCredentials(user.getEmail(), user.getPassword());
    }

    @Step
    public void verifyWelcomeMessageFor(FrequentFlyerMember user) {
        String welcomeMessage = homePage.getWelcomeMessage();
        assertThat(welcomeMessage, equalTo("Witaj " + user.getFirstName()));
    }
}
```

- Raport na podstawie kroków przepływu pracy



Problemy testów webowych

- Drogie w utrzymaniu
  - Wymagają aktualizacji (zmiany w przeglądarkach, w projekcie strony)
  - Wolne działanie
  - Błędy wynikające ze zmian w przeglądarkach
  - Problemy z opóźnieniami w sieci

Testy webowe z przeglądarkami typu headless

- Wysyłanie żądań HTTP bezpośrednio do serwera bez uruchamiania przeglądarki
  - HtmlUnit, PhantomJS
- Zawartość nie jest renderowana tak jak w rzeczywistej przeglądarce

SPA a testy akceptacyjne

- Nowoczesne aplikacje webowe
  - **Aplikacja po stronie klienta** (np. z wykorzystaniem AngularJS)
    - Możliwa zaawansowana logika biznesowa po stronie klienta
    - Dedykowane narzędzia do testowania (np. ProtractorJS dla AnularJS)
  - **Aplikacja po stronie serwera** udostępniająca usługi poprzez API typu RESTfull
    - Testy usług internetowych
    - Narzędzia: RESTAssured (Java)

Podsumowanie

- Automatyczne testy akceptacyjne powinny być niezawodne i łatwe w utrzymaniu
- Testy akceptacyjne powinny być deklaratywne
- Imperatywne szczegóły techniczne powinny być ukryte w dedykowanej warstwie
- Należy wykorzystywać wzorce jak i wsparcie, które dają narzędzia.

Bibliografia

- John Ferguson Smart, BDD w działaniu, Helion 2016
- http://martinfowler.com/bliki/PageObject.html
- http://www.seleniumhq.org/docs/06_test_design_considerations.jsp
- http://www.slideshare.net/abagmar/patterns-in-test-automation