

RestAssured (Java)
automatyzacja testów funkcjonalnych serwisów REST API

Termin: 22-24.11.2023

Prowadzi:
Jakub Szewczyk

15 lat doświadczenia

● **949** współpracujących firm

● **98%** zadowolonych klientów

● **330** współpracujących trenerów

Główne obszary działania



| Szkolenia

Oferujemy szeroki katalog szkoleń z technologii mainstreamowych i specjalistycznych, wschodzących i legacy. Zajęcia prowadzimy w trybie warsztatowym, a programy są oparte o praktyczne know-how. Specjalizujemy się w prowadzeniu dedykowanych szkoleń technologicznych, których agendę dostosowujemy do potrzeb naszych klientów i oczekiwania uczestników.

| Kursy rozwojowe

Posiadamy kursy otwarte (Kodołamacz.pl) i dedykowane akademie dla firm, pozwalające na zdobycie nowych kompetencji w ramach kompleksowych programów rozwojowych dla pracowników. Oferujemy również wsparcie w rekrutacji i edukacji przyszłych pracowników naszych klientów.

| E-learning połączony z warsztatami

Jako uzupełnienie szkoleń tradycyjnych oraz formę nauki samodzielnej proponujemy kursy e-learningowe. Aktualnie w naszej ofercie dostępne są szkolenia typu masterclass, rozumiane jako szkolenia wideo, uzupełnione o spotkania/warsztaty na żywo (zdalnie) z autorem kursu.

| Wydarzenia IT

Inspirujemy i szerzymy wiedzę o technologiach z różnych obszarów na kilkugodzinnych, praktycznych warsztatach w ramach naszej inicjatywy Stacja IT. Organizujemy konferencje m.in. AI & NLP Day, Testaton. Występujemy na konferencjach wewnętrznych naszych klientów np. Orange Developer Day.

| Studia podyplomowe

Współpracujemy z uczelniami wyższymi wspierając realizację zaawansowanych kierunków studiów z zakresu specjalistycznej IT. Jesteśmy partnerami studiów podyplomowych:

na Politechnice Warszawskiej:

Data Science: Algorytmy, narzędzia i aplikacje dla problemów typu Big Data
Big Data: Przetwarzanie i analiza dużych zbiorów danych
Wizualna analityka danych.

na Akademii Leona Koźmińskiego:

Data Science i Big Data w zarządzaniu
Chmura obliczeniowa w zarządzaniu projektami i organizacją.



Cześć!

Jestem Kuba

- Nauczę was tworzenia automatycznych testów
- jakub.szewczyk@hotmail.com
- [LinkedIn](#)



Agenda

Etap 1: Wprowadzenie

- Przygotowanie środowiska

Etap 2: Testy automatyczne

- RestAssured
- Dobre praktyki

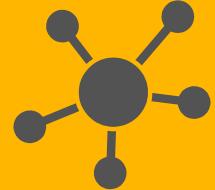
Etap 3: Zaawansowane techniki

- Serializacja / deserializacja
- Dane testowe
- Zmienne środowiskowe
- Testy kontraktowe ([pack.io](#))



Zasady

- **Szkolenie jest dla Ciebie**
 - pytaj
 - dociekaj
 - próbuj
- **Live Coding**
 - twórz a nie kopuj
- **Rób notatki**

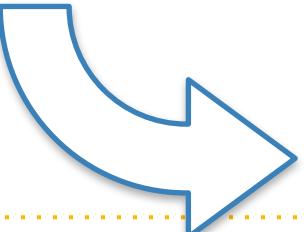


Testy automatyczne

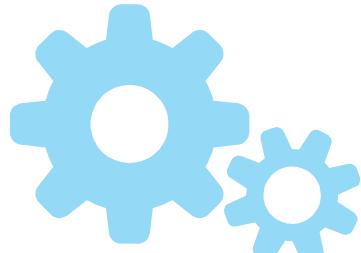
Przygotowanie środowiska



DEVELOPMENT



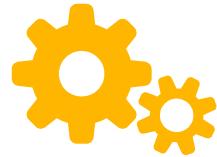
**max
30 min**



BUILD / DEPLOY



TEST





RestAssured



- ekosystem JVM
- automatyzacja komunikacji http
- Głównie do testowania / tworzenia Rest API
- Wieloplatformowy
 - Win, OSX, Linux, Web
- open source
- [strona projektu](#)
- [dokumentacja](#)



Hello world

Skorzystamy z <http://numbersapi.com/>

Kroki do wykonania:

1. Stworzenie projektu Maven
2. Dodanie biblioteki: [rest-assured](#)
3. Dodanie testu `helloWorld` do `AppTest.java`
4. Wykonanie requestu

GET <http://numbersapi.com/1410/year>

Hello world



```
@Test  
public void helloWorld() {  
    RestAssured  
        .given()  
            .baseUri("http://numbersapi.com")  
            .log().all()  
        .when()  
            .get("314/year")  
        .then().log().all();  
}
```

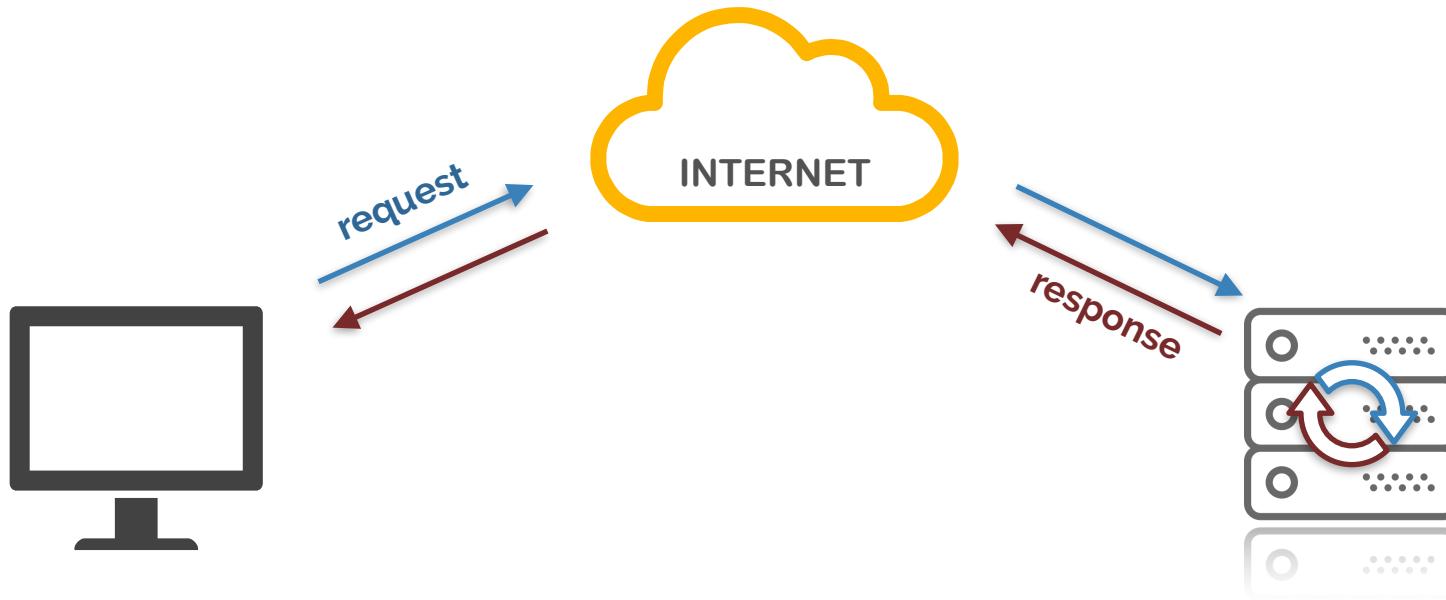


Interface RestAPI

Trochę teorii



Komunikacja HTTP





Request HTTP

GET http://numbersapi.com/314/year

Typ request'u

Metoda

GET, POST....

Protokół

http, https

Adres serwera

host:port

ip:port

Path / ścieżka

Określa konkretny zasób
na serwerze



Request HTTP

GET http://numbersapi.com/314/year

Host: numbersapi.com

User-Agent: curl/7.64.1

Accept: */*

Opcjonalna
wiadomość do serwera



Request HTTP

GET http://numbersapi.com/314/year

Host: numbersapi.com



Nagłówki requestu:

User-Agent: curl/7.64.1

Miejsce na dodatkowe informacje przekazywane pomiędzy klientem a serwerem.

Accept: */*

**Opcjonalna
wiadomość do serwera**

Nagłówki to para
klucz : wartość



Request HTTP

GET `http://numbersapi.com/314/year`

`Host: numbersapi.com`

Wiadomość do serwera

`User-Agent: curl/7.64.1`

Wiadomością może być:

`Accept: */*`

- XML

- JSON

- Tekst

- ...

Opcjonalna
wiadomość do serwera



To co wyślemy opisujemy w
nagłówku **Content-Type**



Response HTTP

HTTP/1.1 200 OK

Server: nginx/1.4.6
(Ubuntu)

Content-Type: text/html;
charset=utf-8

**314 is the year that
Alexander becomes
Bishop of Byzantium.**



Response HTTP

HTTP/1.1 200 OK ←

Server: nginx/1.4.6
(Ubuntu)

Content-Type: text/html;
charset=utf-8

**314 is the year that
Alexander becomes
Bishop of Byzantium.**

Status odpowiedzi:

2xx - Success

200 OK

204 No Content

4xx - Client errors

401 Unauthorized

403 Forbiden

404 Not Found

5xx - Server errors

500 Internal Server Error

503 Service Unavailable



Response HTTP

HTTP/1.1 200 OK

Server: nginx/1.4.6
(Ubuntu)

Content-Type: text/html;
charset=utf-8

**314 is the year that
Alexander becomes
Bishop of Byzantium.**

Nagłówki odpowiedzi:

Miejsce na dodatkowe informacje przekazywane pomiędzy klientem a serwerem.

Nagłówki to para
klucz : wartość



Response HTTP

HTTP/1.1 200 OK

Server: nginx/1.4.6
(Ubuntu)

Content-Type: text/html;
charset=utf-8

**314 is the year that
Alexander becomes ←
Bishop of Byzantium.**

Wiadomość zwrotna z serwera

Wiadomością może być:

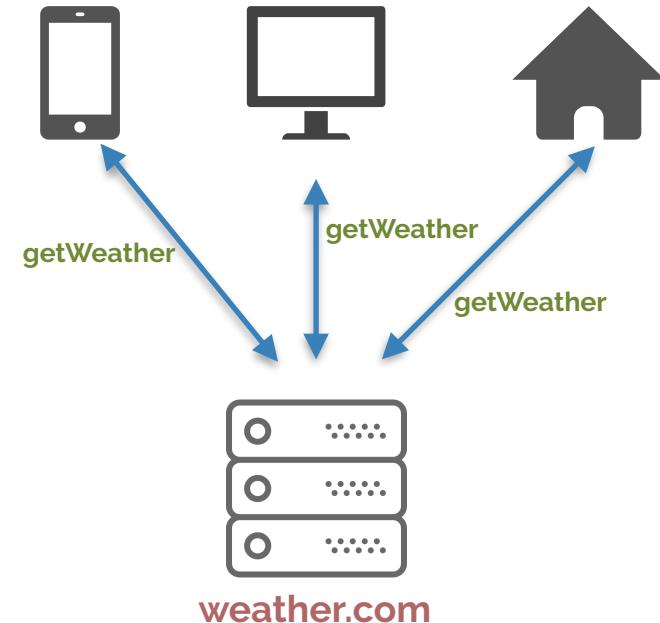
- obrazek
- HTML
- XML
- JSON
- Tekst
- CSV
- i wiele innych

Rest API



Interfejs do komunikacji z serwerem

- pozwalający na zarządzanie danymi (CRUD)
 - Create
 - Read
 - Update
 - Delete
- pozwala na odseparowanie warstwy UI





CRUD w RestAPI

Create

Tworzenie
POST /todos

Read

Pobieranie
GET /todos
GET /todos/{id}

Update

Zmiana
PUT /todos/{id}

Delete

Kasowanie
DELETE /todos/{id}



Konstrukcja path'a

GET /todos/1 ←
DELETE /todos/3 ← path param
numbersapi.com/number/type

DELETE /todos/1?archive=true ← query param
numbersapi.com/random?min=10&max=20&json



Format danych

- Rest API nie narzuca formatu danych
- Najczęściej używany jest **JSON**
 - format tekstowy (dobrze się kompresuje)
 - prosta składnia
 - mały narzut składniowy
 - jest to format "internetu 2.0" **J**ava **S**cript **O**bject **N**otation



Obiekt JSON

```
{  
  "id": 234,  
  "application": "Postman",  
  "free_plan": true,  
  "supported_os": ["Linux", "Windows", "OSX"]  
  "urls": {  
    "homepage": "https://www.postman.com",  
    "documentation": "https://learning.postman.com/docs"  
  }  
}
```



Tablica JSON

```
[  
  {  
    "id": 1,  
    "title": "Przygotuj szkolenie Postman",  
    "completed": true  
  },  
  {  
    "id": 2,  
    "title": "Przeprowadź szkolenie Postman",  
    "completed": false  
  }]  
]
```



Wyzwania

- **Tworzenie requestów**
 - definiowanie metod
 - autoryzacja
 - definiowanie danych testowych
- **Testowanie responsów**
 - Statusy odpowiedzi
 - poprawność nagłówków
 - poprawność wiadomości
 - czasy odpowiedzi



Pierwszy scenariusz

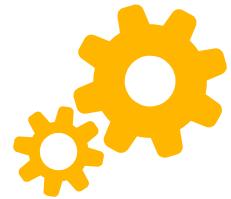
Zaczynamy zabawę!



Nasza piaskownica

- todoist.com - Menadżer zadań
 - Posiada interfejs webowy
 - Udostępnia prosty interfejs Rest API.
 - [Dokumentacja](#)
-
- Zdanie dla Ciebie: Proszę załóż sobie konto.

Scenariusz testowy



Użytkownik może stworzyć nowy projekt

1. Adam loguje się do todoist.com
2. Adam tworzy nowy projekt
3. Adam sprawdza czy projekt został utworzony
4. Adam sprawdza czy projekt jest na liście wszystkich projektów



Organizacja testów

- Do testów wykorzystamy bibliotekę jUnit
- Jako runner testów: [maven-surefire-plugin](#)
- [Domyślne wymagania](#) nazewnictwa klas z testami



Krok: logowanie

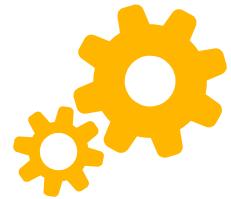
1. Adam loguje się do todoist.com

Autoryzacja w interfejsach Rest API

- Token - najczęściej spotykana forma autoryzacji
- Dokumentacja autoryzacji w todoist.com:
<https://developer.todoist.com/rest/v1/#authorization>
- Token będziemy dodawać do każdego requestu:

Header: "Authorization: Bearer \$token"

Krok: tworzenie projektu



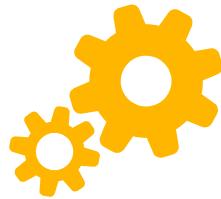
2. Adam tworzy nowy projekt

```
POST https://api.todoist.com/rest/v2/projects
```

```
Headers: "Authorization: Bearer $token"
```

```
"Content-Type: application/json"
```

```
{  
  "name": "Szkolenie Rest API"  
}
```



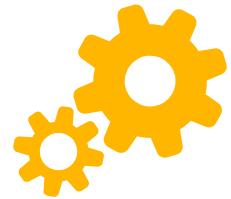
Krok: tworzenie projektu

```
RestAssured  
    .given()  
  
    .when()  
  
    .then();
```

Podstawowa klasa RestAssured

- gotowa do użycia
- nie trzeba tworzyć obiektu
- wszystko co potrzeba wystawia jako metody statyczne
- NIE jest thread safe!

Krok: tworzenie projektu



```
RestAssured  
    .given()  
  
    .when()  
  
    .then();
```

`given()`

definiujemy request...

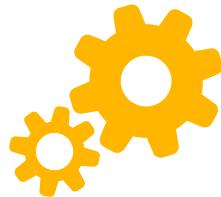
`when()`

...wysyłamy go...

`then()`

...przetwarzamy response

Tworzymy...



Nagłówki

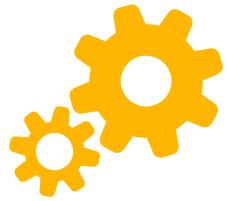
```
.given()
  .baseUri("https://api.todoist.com")
  .header("Authorization", "Bearer token")
  .header("Content-Type", "application/json")

  .body("{ \"name\" = \"RestAssured Course\" }")
```

Adres serwera

Treść

...wysyłamy...



RestAssured

```
.given()  
.when()  
.get("/path")
```

RestAssured

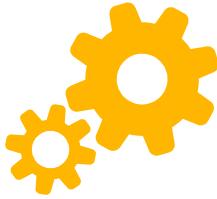
```
.given()  
.when()  
.put("/path")
```

RestAssured

```
.given()  
.when()  
.post("/path")
```

RestAssured

```
.given()  
.when()  
.delete("/path")
```



...i sprawdzamy

funkcja "grupująca" asercje

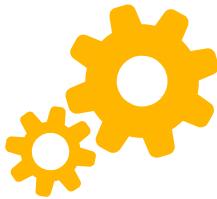
```
.then()  
    .assertThat()  
        .statusCode(200)  
        .body("jsonKey", Matchers.equalTo("expectedValue"))  
        .header("key", "expectedValue")  
        .header("key", Matchers.contains("expected"))
```

Spodziewana wartość pola w JSON

status response'a

wymagane nagłówki

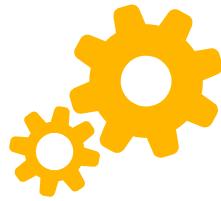
...i sprawdzamy



Biblioteka: [Hamcrest](#)

Javadoc: [Matchers](#)

```
Matchers.equalTo("expected");  
  
Matchers.equalToIgnoringCase("EXPected");  
  
Matchers.containsString("exp");  
  
Matchers.not(Matchers.containsString("exp"));
```



Parę usprawnień

stały "prefix" do path'a

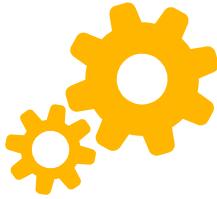
dla wszystkich requestów

możliwość definicji
wielu nagłówków
w jednej funkcji

```
.given()
    .baseUri("https://api.todoist.com")
    .basePath("/rest/v1")

    .headers("Authorization", "Bearer token",
             "Content-Type", "application/json")

    .body("{ \"name\" = \"RestAssured Course\" }")
```



Parę usprawnień

Wykluczamy możliwość pomyłki

w definicji "Content-Type"

```
.given()
    .baseUri("https://api.todoist.com")
    .basePath("/rest/v1")

    .header("Authorization", "Bearer token")
    .contentType(ContentType.JSON)

    .body("{ \"name\" = \"RestAssured Course\" }")
```



Zadanie 1

Zaimplementuj krok:

3. Adam sprawdza czy projekt został utworzony

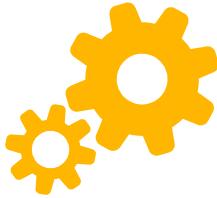
Podpowiedź:

użyj requestu:

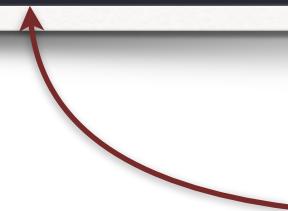
```
GET https://api.todoist.com/rest/v2/projects/ID_PROJEKTU
Headers: "Authorization: Bearer $token"
```

ID projektu znajdziesz w odpowiedzi na poprzedni request

Pobieranie danych

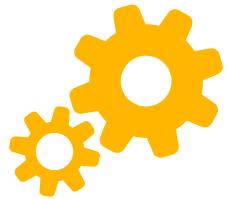


```
.then()  
    .assertThat()  
        .statusCode(200)  
        .body("name", Matchers.equalTo("Project name"))  
  
.and()  
    .extract().path("id");
```



To jest "return" z naszej funkcji.

Może występować tylko na końcu!



Path params

```
.when()  
    .get("/projects/" + projectId)
```

Prosta konkatenacja

```
.when()  
    .get(String.format("/projects/%d", projectId))
```

formatowanie

uproszczone formatowanie
z RestAssured

```
.given()  
    .pathParam("id", projectId)  
.when()  
    .get("/projects/{id}")
```



Zadanie 2

Zaimplementuj krok (bez asercji):

4. Adam sprawdza czy projekt jest na liście wszystkich projektów

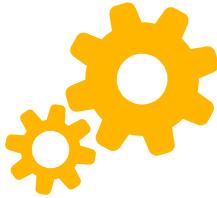
Podpowiedź

Wyślij request:

GET <https://api.todoist.com/rest/v2/projects>

Headers: "Authorization: Bearer \$token"

Filtrowanie danych

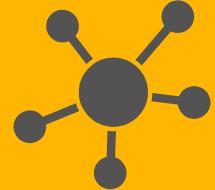


```
.then()  
    .assertThat()  
        .body("find{ it.id == 1234 }.name",  
              Matchers.equalTo("Project name"))  
.and()  
    .extract().path("find{ it.id == 1234 }");
```

meta język do filtrowania danych z tablic

[XML Path](#)

[JSON Path](#)



Refaktor

Jak oszlifować diament



Refaktor

Działający kod > ładny kod

- Pierwsze rozwiązanie może być proste (prostackie)
- ważne że działa
-i że działa poprawnie



Refaktor c.d.

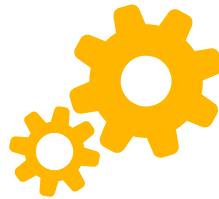
działający kod + ładny kod > działający kod

- Większość rozwiązań można poprawić
- Lepiej poznaliśmy problem do rozwiązania
- Optymalizacja kodu



Obszary do poprawienia

- Poprawienie kodu
 - zasada DRY
 - optymalizacje
- Wszelkie wartości przetrzymujemy w zmiennych
 - teksty
 - liczby
- Dodajemy dokumentację
 - komentarze
 - lepsze nazewnictwo

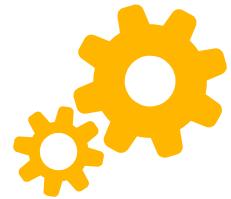


DRY: nie powtarzaj się

Tak oznaczona funkcja uruchomi się przed każdym testem

```
@Before  
public void setup() {  
    RestAssured.baseURI = "https://api.todoist.com";  
    RestAssuredbasePath = "/rest/v1";  
  
    RestAssured.requestSpecification =  
        RestAssured.given()  
            .header("Authorization", "Bearer token")  
            .contentType(MediaType.JSON);  
}
```

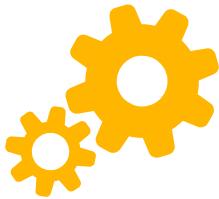
Request builder



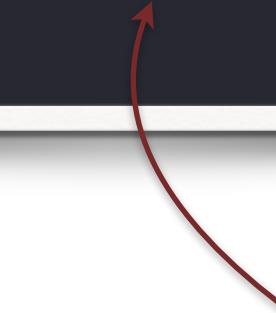
Bardziej "eleganckie" rozwiązanie - pełna, bazowa definicja
requestu jest w jednym miejscu

```
RequestSpecBuilder builder = new RequestSpecBuilder();  
  
RestAssured.requestSpecification = builder  
    .setBaseUri("https://api.todoist.com")  
    .setBasePath("/rest/v1")  
    .addHeader("Authorization", authValue)  
    .setContentType(MediaType.JSON)  
    .build();
```

Optymalizacja



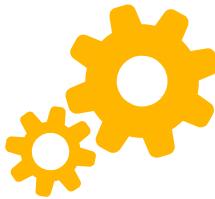
```
@Before  
public void setup() {  
  
    RestAssured.enableLoggingOfRequestAndResponseIfValidationFails();  
  
}
```



Jak wszystko działa to po co nam logi?

Kto sprawdza logi działających testów?

Czytelność kodu

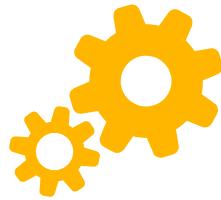


- ▶ Przenieśmy poszczególne kroki do funkcji
- ▶ Nadajmy im znaczące nazwy



```
@Test  
public void userCanCreateProject() {  
  
    long projectId = createProject("RestAssured Course");  
    checkProjectDetails(projectId, "RestAssured Course");  
    checkIfProjectCanBeListed(projectId, "RestAssured Course");  
  
}
```

Ostatnie szlify



```
@Test  
public void userCanCreateProject() {  
  
    String projectName = "RestAssured Course";  
  
    long projectId = createProject(projectName);  
    checkProjectDetails(projectId, projectName);  
    checkIfProjectCanBeListed(projectId, projectName);  
}
```



Zadanie 3

Użytkownik może dodać zadanie do projektu

Warunek wstępny: Adam posiada projekt w todoist

1. Adam dodaje zadanie do projektu
2. Adam sprawdza czy zadanie zostało stworzone
3. Adam sprawdza czy zadanie jest na liście wszystkich zadań



Zadanie 3

1)

```
POST https://api.todoist.com/rest/v2/tasks  
Header: "Authorization: Bearer $token"  
{  
    "content": "Master assertions",  
    "project_id": "ID_PROJEKTU"  
}
```

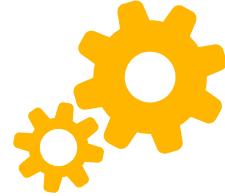
2)

```
GET https://api.todoist.com/rest/v2/tasks/1234
```

3)

```
GET https://api.todoist.com/rest/v2/tasks
```

Re używanie kodu

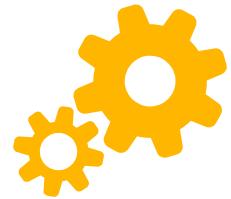


```
ProjectSteps project = new ProjectSteps();
ProjectVerification check = new ProjectVerification();

@Test
public void userCanCreateProject() {
    String projectName = "RestAssured Course";
    long projectId = project.create(projectName);
    check.projectDetails(projectId, projectName);
    check.projectCanBeListed(projectId, projectName);
}
```

...i refaktor

- szukamy duplikacji kodu
- hardcodowanych danych
- optymalizacje





Raporty

Moc Serenity



SerenityAssured



- Automated Acceptance Testing with Style
- Wrapper do RestAssured
 - Dodane raportowanie
 - Jest thread safe!
- **open source**
- [**strona** projektu](#)
- [**dokumentacja**](#)



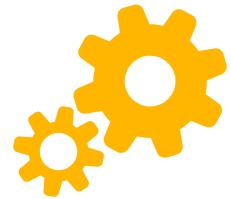
Konfiguracja projektu

Kroki do wykonania:

1. Dodanie biblioteki: [serenity-junit5](#)
2. Dodanie pluginu do raportów:
[źródło](#)

```
<plugin>
  <groupId>net.serenity-bdd.maven.plugins</groupId>
  <artifactId>serenity-maven-plugin</artifactId>
  <version>2.3.12</version>
  <executions>
    <execution>
      <id>serenity-reports</id>
      <phase>post-integration-test</phase>
      <goals>
        <goal>aggregate</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Używamy Serenity



```
@ExtendWith(SerenityJUnit5Extension.class)  
public abstract class BaseTestClass {
```

Do każdej suity musimy dodać
Serenity Runnera

```
SerenityRest  
    .given()  
        .body(payload)  
    .when()  
        .post("/projects")  
    .then()  
        .assertThat()  
            .statusCode(HttpStatus.OK_200)
```

Podmieniamy RestAssured
na wrapper SerenityRest



Odpalamy Serenity

- Uruchomienie wszystkich testów:

```
mvn clean test
```

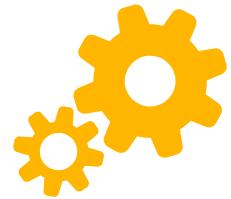
- Wygenerowanie samego raportu

```
mvn serenity:aggregate
```

- Testy + generowanie raportu (jeżeli wszystkie przejdą)

```
mvn clean verify
```

Dependency injection



```
@Steps  
ProjectSteps project;
```

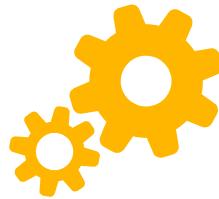
```
@Steps  
ProjectVerification check;
```

Tworzenie obiektów z krokami testów
oddelegujmy do mechanizmu DI

Oznaczmy kroki, które mają
znaleźć się w raporcie

```
@Step  
public long create(String projectName) {
```

Lepsze opisy



```
@Step("User creates project with '{0}' name")
public long create(String projectName) {
```

Do raportu możemy dodać argumenty funkcji

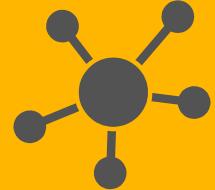
```
@Step("User checks project {0} details. Expected name: '{1}'")
public void projectDetails(long projectId, String projectName) {
```



Zadanie 4

Dodaj moc Serenity do scenariusza:

“Użytkownik może dodać zadanie do projektu”



Dane testowe

Jak robić to dobrze....

Zmienne środowiskowe



- Wyodrębnienie zmiennych środowiskowych per środowisko
 - Wszelkie adresy
 - Dane autoryzujące (np. token)
 - itp....
- Jeden test - wiele środowisk



Pliki .properties

```
baseURI = https://api.todoist.com  
basePath = /rest/v1  
token = moj_token
```

```
InputStream stream =  
    this.getClass().getResourceAsStream("/test.properties");  
Properties env = new Properties();  
env.load(stream);
```



Wybór środowiska

ustawianie zmiennej systemowej:

```
mvn clean test -Denv=prod
```

```
String filename = String.format("%s.properties",  
    System.getProperty("env") );
```



Reprezentacja obiektu

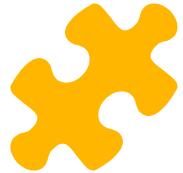
```
public class Project {  
  
    private String name;  
    private String secretData;  
  
    public String getName() { return name; }  
    public void setName(String name) { this.name = name; }  
}
```



Serializacja do JSON

```
public long create(Project project) {  
  
    return SerenityRest  
        .given()  
            .body(project)  
        .when()  
            .post("/projects")
```

Jako body wrzucamy teraz nasz obiekt reprezentujący projekt



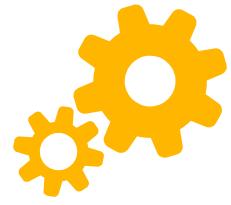
Deserializacja z JSON

```
.then()
    .assertThat()
        .statusCode(200)
        .body("name", Matchers.equalTo(project.getName()))
    .and()
        .extract().body().as(Project.class);

.statusCode(200)
.body(String.format("find{ it.id == %d}.name", this.id), Matchers.equalTo(this.name))
.and().extract().body().jsonPath().getList(".");
```

Na podstawie otrzymanego responsu
możemy sobie zbudować obiekt klasy Project

Zadanie 5



Zbuduj obiekt reprezentujący zadanie i zastosuj go w teście



Dane losowe

Eliminujemy efekt "paradoksu pestycydów"

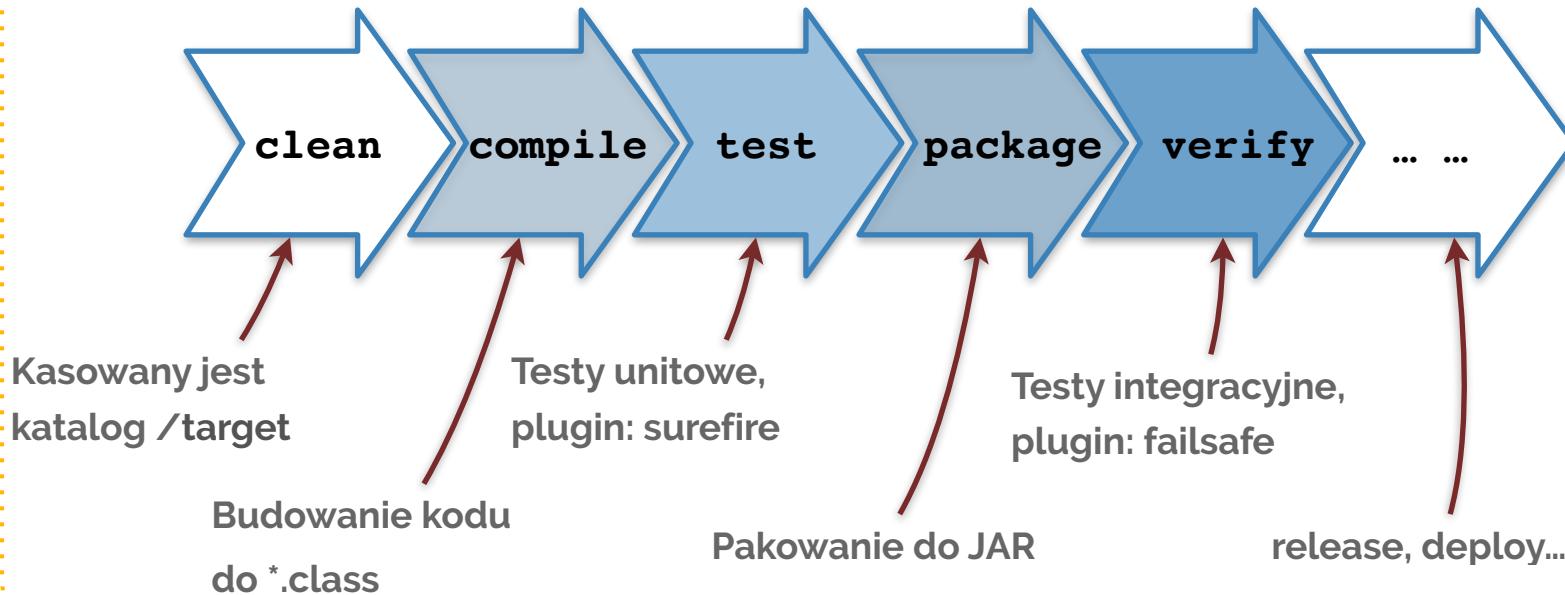
- [java-faker](#): generator danych losowych
- [JavaDoc](#)

```
Faker faker = new Faker();
Project testProject = new Project(faker.commerce().productName());
```

Maven



Maven goals





Maven profiles

Umożliwiają stworzenie profili zawierających pełną konfigurację dla danego środowiska

```
<profile>
    <id>uat</id>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <configuration>
                    <systemPropertyVariables>
                        <env>uat_env</env>
                        <td>static</td>
                    </systemPropertyVariables>
                </configuration>
            </plugin>
        </plugins>
    </build>
</profile>
```



Maven profiles

Umożliwiają stworzenie profili zawierających konfigurację testów do wykonania

```
<profile>
    <id>projects</id>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <configuration>
                    <includes>
                        <include>**/ProjectCreationTest.java</include>
                    </includes>
                </configuration>
            </plugin>
        </plugins>
    </build>
</profile>
```

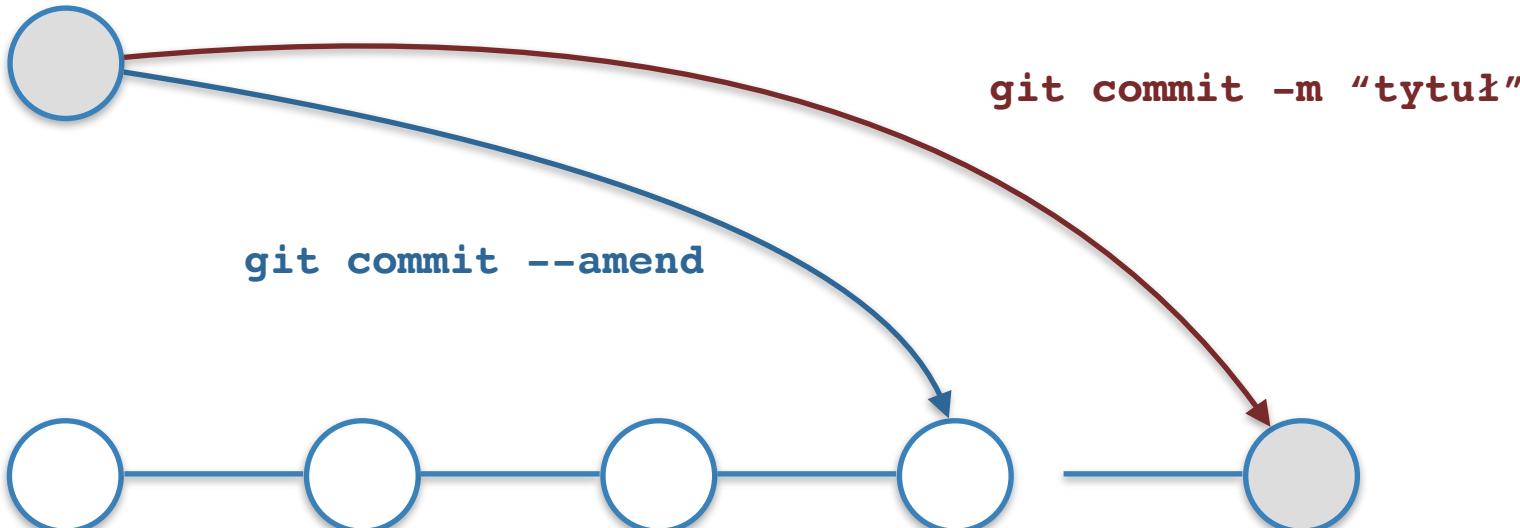
GIT

Jak nie pogubić się w kodzie

git commit



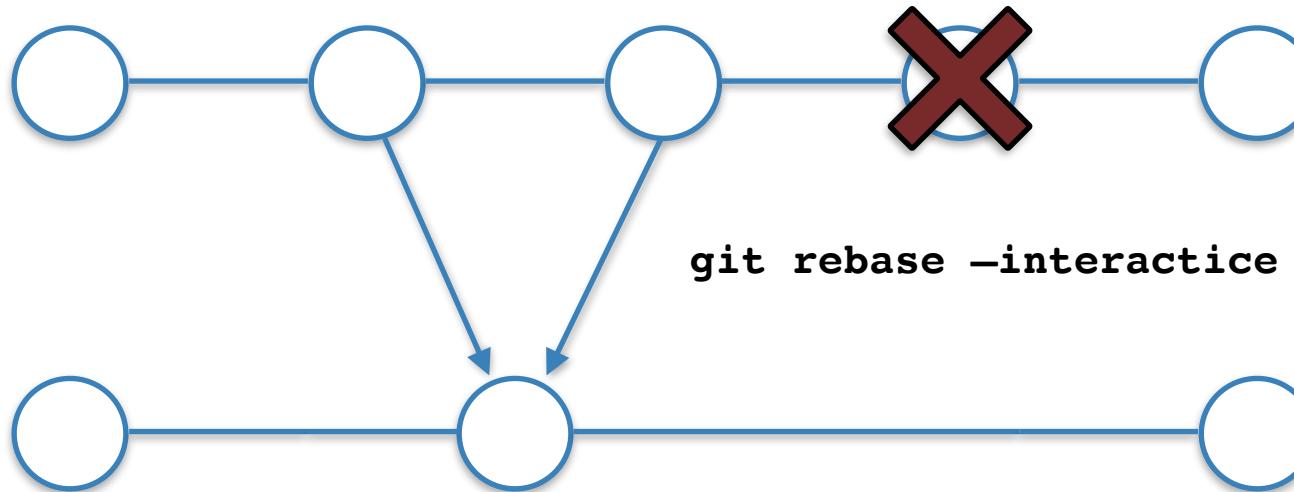
Dodanie zmian do repozytorium





git rebase interactive

Rearanżacja / edycja commitów

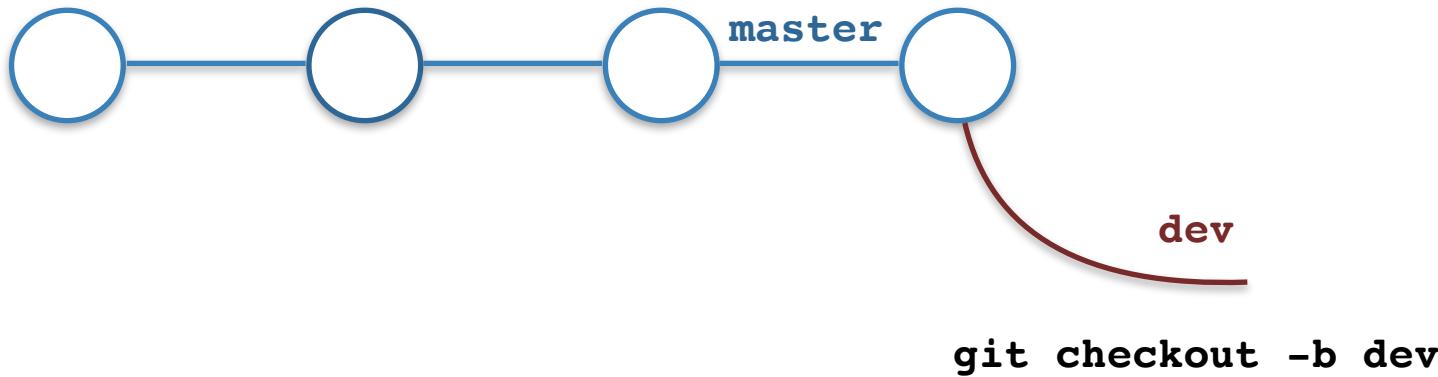


`git rebase -interactice HEAD~5`



git checkout -b

Utworzenie nowej pustej gałęzi kodu



git checkout



Przełączanie się pomiędzy gałęziami
lub cofnięcie się do konkretnego
committa

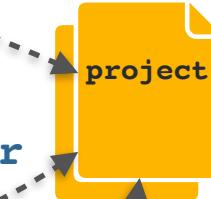
`git checkout #commit-id`

`git checkout master`

`master`

`dev`

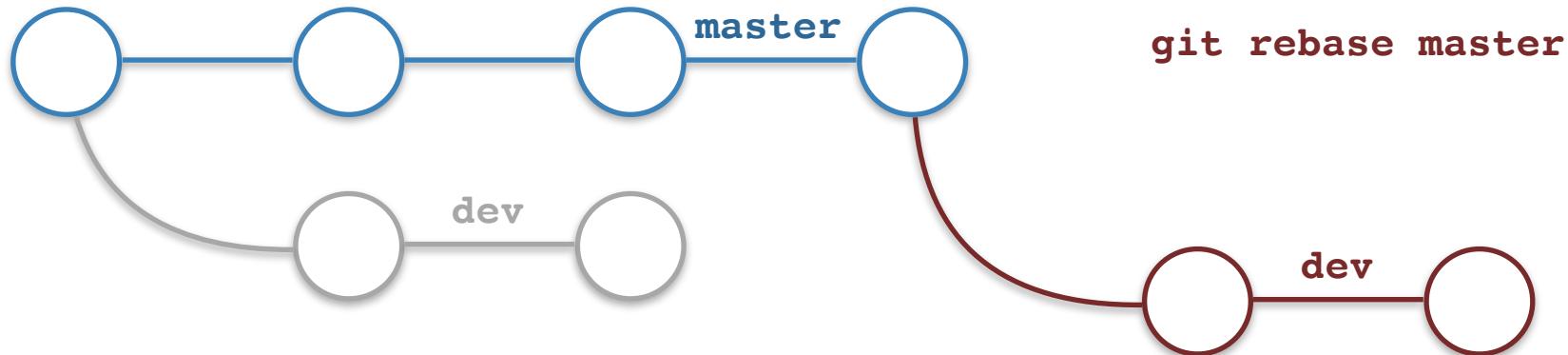
`git checkout dev`





git rebase

Przepięcie gałęzi na najnowszego commita

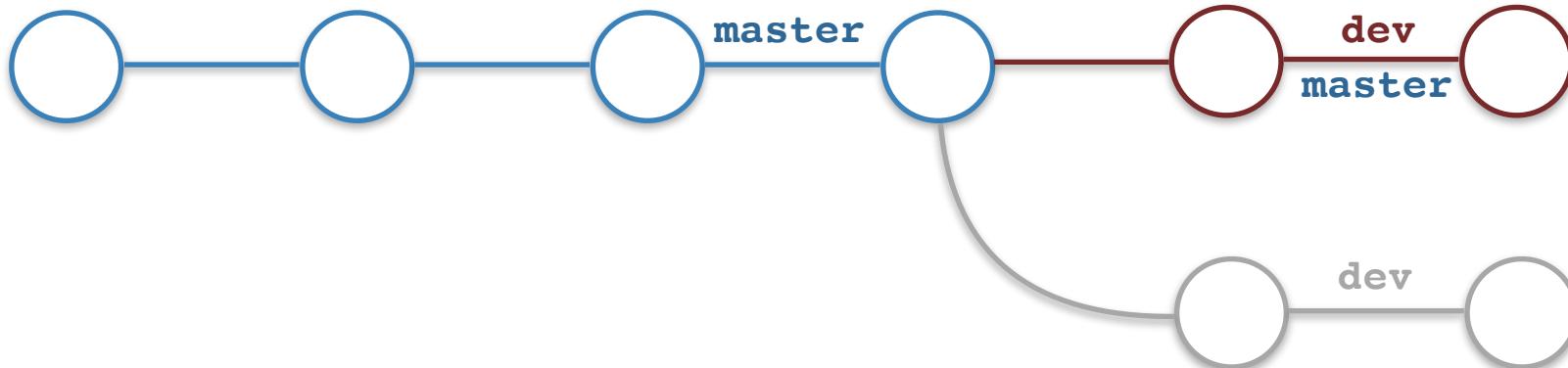


git merge



Połączenie gałęzi

`git merge dev`





Tips and tricks

Przyśpieszamy jeszcze bardziej



Surefire

Uruchomienie wszystkich testów z klasy

```
mvn test -Dtest=ProjectTest  
mvn test -Dtest=ProjectTest,ProjectTaskTest
```

Uruchomienie pojedynczych testów

```
mvn test -Dtest=ProjectTest#createProject  
mvn test -Dtest=ProjectTest#createProject+deleteProject
```



Tagi

```
@Test  
@WithTagValuesOf({"regression", "smoke"})  
public void userCanCreateProject() {
```

Uruchomienie testów:

```
mvn -Dtags=smoke  
mvn -Dtags=smoke,regression
```



Systemy CI

- Worker CI musi mieć zainstalowanego maven'a
- Musi mieć dostęp do plików projektu
 - systemy kontroli wersji (GIT, SVN...)
- Używamy dokładnie takich samych komend



Wielowątkowość

Ustawienie biblioteki jUnit5 do uruchamiania testów w wielu wątkach:

plik:

src/test/resources/junit-platform.properties

junit.jupiter.execution.parallel.enabled = true
junit.jupiter.execution.parallel.config.fixed.parallelism = 4

liczba wątków

Oznaczenie testów,
które chcemy zrównoleglić

```
@Execution(ExecutionMode.CONCURRENT)
@ExtendWith(SerenityJUnit5Extension.class)
public abstract class BaseSetup {
```



Sprzątamy bałagan

Czyścimy wygenerowane dane

@After test

- Standard zaleca:
 - Warunki początkowe - Test - Sprzątamy
- ...ale nie polecam tego podejścia
- Najważniejsze są dane
 - do analizy
 - do śledztwa
 - do obciążenia
- Sprzątaj, ale raz w tygodniu ;)

Co dalej?

1. Zapoznać się z dokumentacją
2. Rozwijać się jako programista:
 1. [Head First: Design patterns](#)
 2. [Clean Code](#)
3. Darmowe interfejsy do ćwiczeń - [tutaj](#), i [tutaj](#) jeszcze trochę