

# Automatyzacja Atlassian dla skalowalnych projektów

Aleksander Kopyto                      Maksymilian Walicki  
[aleks@student.agh.edu.pl](mailto:aleks@student.agh.edu.pl)                      [mwalicki@student.agh.edu.pl](mailto:mwalicki@student.agh.edu.pl)

Michał Zychowicz  
[zychowiczm@student.agh.edu.pl](mailto:zychowiczm@student.agh.edu.pl)

Styczeń 2024

## Spis treści

<b>Wstęp</b>	<b>2</b>
<b>1 Genius commity</b>	<b>2</b>
1.1 Dlaczego warto używać genius commitów?	2
1.2 Instalacja	3
1.3 Opis dodawania Genius Commitów	4
1.3.1 Przykładowa konfiguracja komendy @comment	5
1.4 Spis dostępnych szablonowych commitów	6
<b>2 Powiadomienia</b>	<b>8</b>
2.1 E-mail	8
2.1.1 Przykład automatyzacji z wysłaniem e-maila	8
2.2 Smart values	9
2.3 Wiadomość Microsoft Teams	10
2.3.1 Webhook - medium transmisyjne?	10
2.3.2 Przykład automatyzacji z wysłaniem wiadomości MS Teams	11
<b>3 Script Runner</b>	<b>12</b>
3.1 Co to Script Runner?	12
3.2 Instalacja	12
3.3 Przykładowe Skrypty	13
3.3.1 Wstęp	13
3.3.2 Dodawanie komentarza do nowo utworzonych skryptów	14
3.3.3 Automatyczne przypisywanie zadań autorowi issue	15
3.3.4 Automatyczne przypisywanie zadań na podstawie tytułów	15
3.3.5 Łączność z Microsoft Teams	16
<b>Bibliografia</b>	<b>16</b>

# Wstęp

W dynamicznym krajobrazie nowoczesnego zarządzania projektami zdolność do płynnego kierowania i skalowania przepływów pracy jest kluczowa. Organizacje borykają się z coraz bardziej skomplikowanymi projektami i potrzebą efektywnej współpracy, a rola solidnych narzędzi do zarządzania projektami nie może być przeceniana. Wśród liderów w tej dziedzinie znajduje się Atlassian, wiodący dostawca rozwiązań oprogramowania, który umożliwia zespołom planowanie, śledzenie i dostarczanie projektów z precyzją.

Ten artykuł zagłębia się w dziedzinę zdolności automatyzacji Atlassiana i ich kluczowej roli w ułatwianiu skalowalności różnorodnych projektów. Pakiet narzędzi Atlassiana oferuje kompleksowy ekosystem, który usprawnia przepływy pracy projektowej i zwiększa produktywność zespołu.

## Radzenie sobie ze złożonością

W współczesnym krajobrazie biznesowym projekty często wykraczają poza proste listy zadań; obejmują złożone procesy, wieloaspektową współpracę i potrzebę dostosowania w czasie rzeczywistym. Czy to zarządzanie zadaniami w Jirze, czy dokumentowanie wiedzy w Confluence, Atlassian jest dostosowany do uwzględnienia niuansów współczesnych środowisk projektowych.

## Uwolnienie Mocy Automatyzacji

Kluczowym czynnikiem wpływającym na skalowalność projektów jest jego silna zdolność do automatyzacji. Automatyzacja nie tylko przyspiesza powtarzalne zadania, ale także zapewnia spójność i dokładność w realizacji projektu. Funkcje automatyzacji Atlassiana pozwalają zespołom skupić się na kreatywnym rozwiązywaniu problemów i strategicznym podejmowaniu decyzji, zamiast być przytłoczonymi manualnymi, czasochłonnymi procesami.

W trakcie tej eksploracji automatyzacji dla projektów skalowalnych, zagłębimy się w najlepsze praktyki i historie sukcesu. Od optymalizacji współpracy po przyspieszanie dostarczania projektu, narzędzia automatyzacji Atlassian stają się niezbędnymi zasobami dla zespołów radzących sobie z złożonościami dzisiejszych projektów.

Dołącz do nas w tej podróży przez skrzyżowanie nowoczesnej technologii i doskonałości w zarządzaniu projektami, gdy odkrywamy niewykorzystany potencjał automatyzacji Atlassiana w napędzaniu skalowalności, wspieraniu innowacji i osiąganiu sukcesu projektowego.

## 1 Genius commity

### 1.1 Dlaczego warto używać genius commitów?

Genius Commits pozwalają na automatyzację procesów poprzez dodawanie poleceń do wiadomości commitów w systemach Git, Subversion i Mercurial. Przykładowo, commit o treści:

```
1 || Fix the FOO-1 bug. @comment Bug fixed! @status Done
```

automatycznie dodaje komentarz i aktualizuje status związany z zadaniem FOO-1.

Narzędzie Genius Commit'ów jest niesamowicie konfigurowalne, umożliwiając deweloperom uruchamianiem dowolnych działań, takich jak wpływanie na zadania jirowe, rejestrowanie czasu pracy, uruchamianie skryptów, wysyłanie zapytań HTTP, uruchamianie narzędzi testowania kodu czy też uruchamianie całych procesów CI/CD - bezpośrednio poprzez dedykowany trigger Genius Commit Created.

W przeciwieństwie do "Smart Commits" w Bitbucket, Genius Commits są bardziej zaawansowane polegając na dowolnej liczbie dostosowywalnych poleceń.

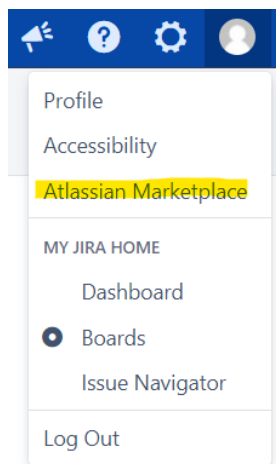
Genius Commity integrują się z procesami deweloperskimi, umożliwiając automatyzację skomplikowanych operacji przy zachowaniu maksymalnej prostoty użycia.

## 1.2 Instalacja

Aby rozpocząć pracę z Genius Commitami potrzebujemy dwóch, **w pełni darmowych**, dodatków do naszego systemu Atlassian:

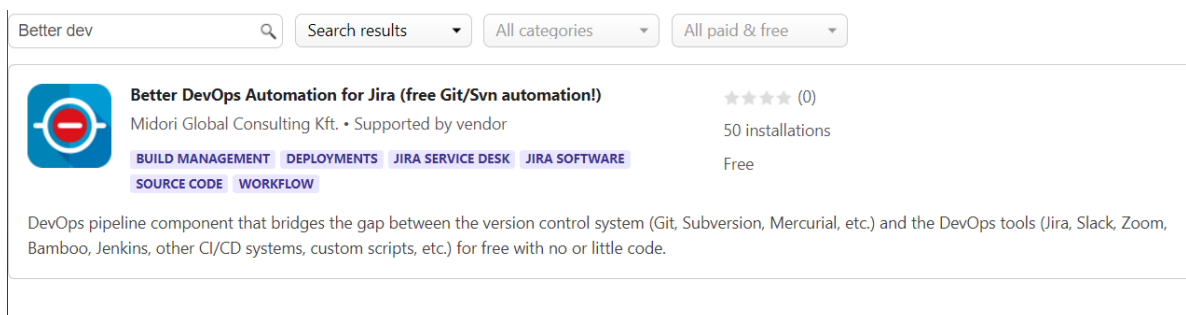
„Better Commit Policy for Jira (Git, Subversion and more!)” oraz „Better DevOps Automation for Jira (free Git/Svn automation!)”

1. Instalację rozpoczynamy od kliknięcia w awatar naszego użytkownika w prawym górnym rogu oraz przejście do „Atlassian Marketplace”.

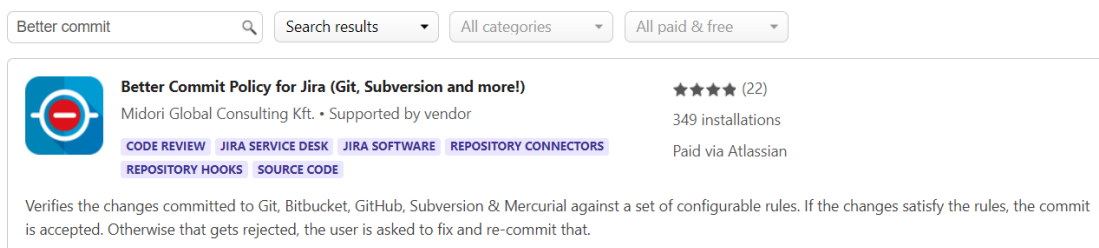


Rysunek 1:

2. Następnie wyszukujemy potrzebne dla nas dodatki i instalujemy je, bądź tworzymy zapytanie o instalację administratora naszego systemu Atlassianowego.

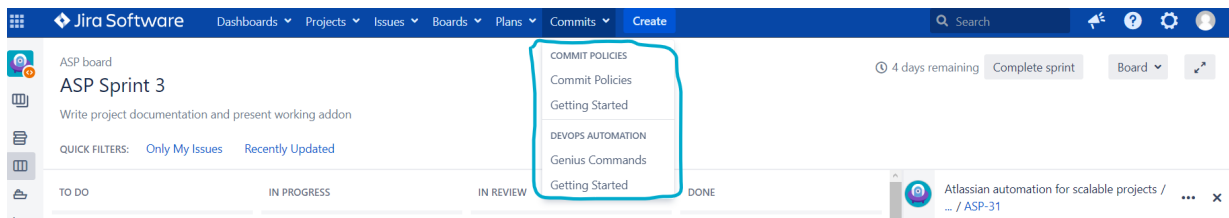


Rysunek 2:



Rysunek 3:

3. Po prawidłowym zainstalowaniu dodatków pojawi się nowa zakładka w naszym menu Jiry.

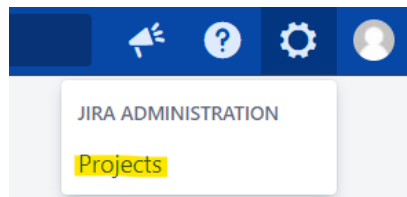


Rysunek 4:

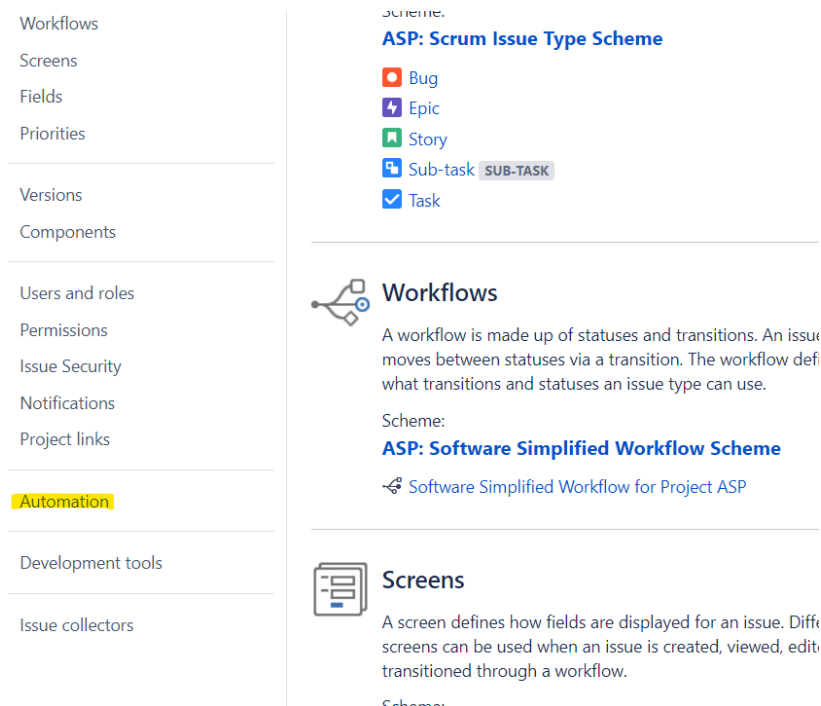
### 1.3 Opis dodawania Genius Commitów

Aby dodać swój pierwszy Genius Commit rozpocznij od:

1. Zaloguj się do Jiry jako administrator, przejdź do Administration → System → Automation rules.



Rysunek 5:



Rysunek 6:

2. Kliknij Create rule.

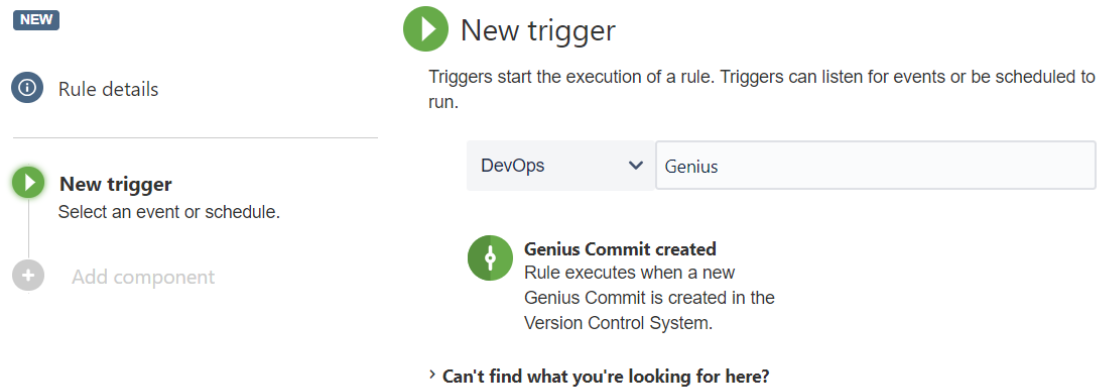
#### Automation

Automation rules allow you to automate repetitive tasks based on criteria that you set. Here you can manage existing rules and create new ones. [Learn more about automation](#)



Rysunek 7:

3. Wybierz trigger "Genius Commit created" (z kategorii DevOps).

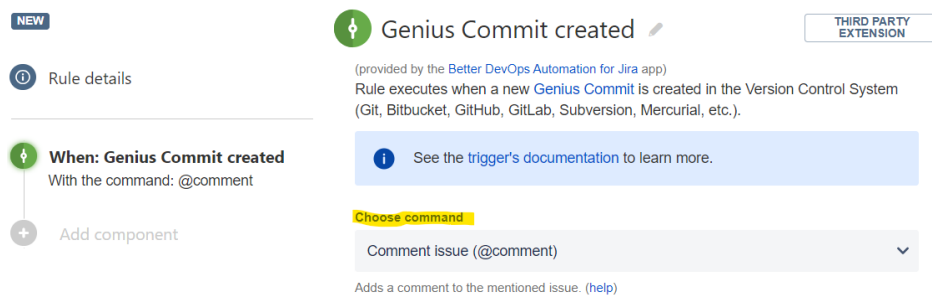


Rysunek 8:

### 1.3.1 Przykładowa konfiguracja komendy @comment

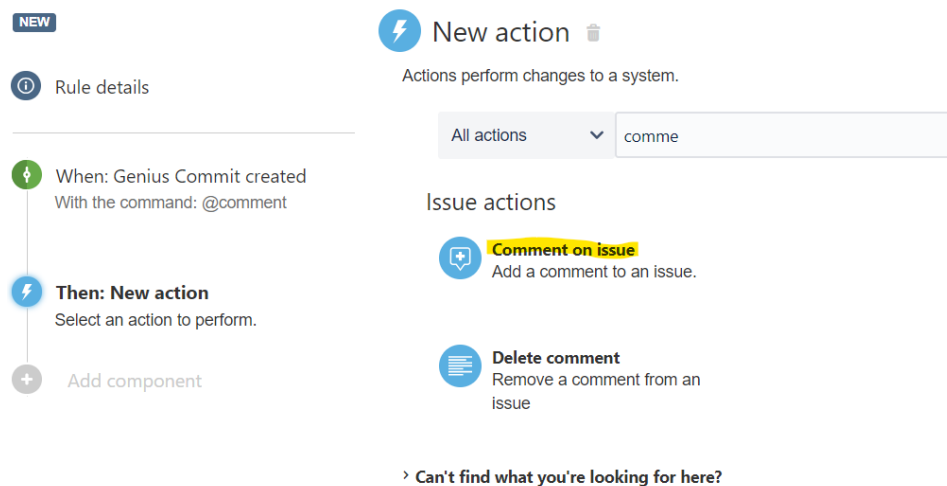
Aby skonfigurować automatyczne dodawanie komentarzy do issue w Jirze:

1. Wybierz z listy wielu szablonowych Genius Commit'ów polecenie "Comment issue" i kliknij Save.



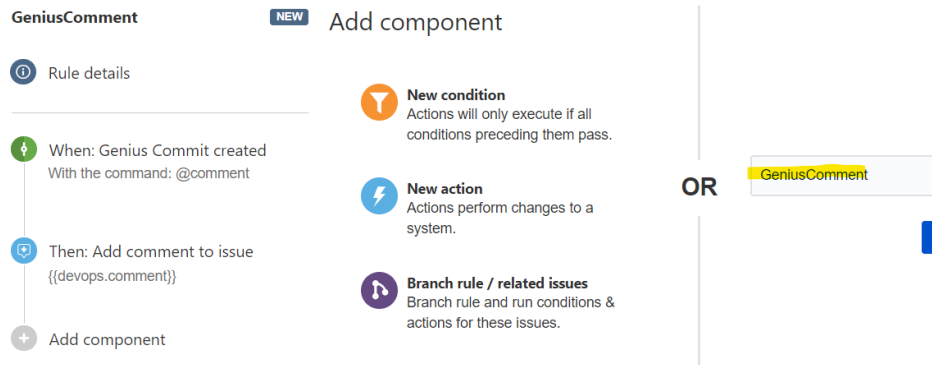
Rysunek 9:

2. Kliknij New action i wybierz akcję "Comment on issue".



Rysunek 10:

3. Wprowadź smart value "{devops.comment}" do pola opisu.
4. Kliknij Save.
5. Nadaj swojej regule automatyzacji intuicyjną nazwę i ją włącz.



Rysunek 11:

Jak widać, Genius Commity pozwalają na dodawanie nieskończenie dużego zestawu reguł i intuicyjnych akcji.  
**Wiadomość commita w jednej linijce:**

```
1 || Protect against the null value that caused the FOO-1 bug. @comment Bug fixed!
```

Komentarz ten zostanie dodany do zadania FOO-1: *Bug fixed!*

**Wiadomość commita w wielu liniach:**

```
1 || Protect against the null value that caused the FOO-1 bug.
2 || @comment Bug fixed!
3 || (It was caused by an unexpected null value.)
```

Komentarz ten zostanie dodany do zadania FOO-1: *Bug fixed!*

*(It was caused by an unexpected null value.)*

**Wiadomość commita w wielu liniach z formatowaniem:**

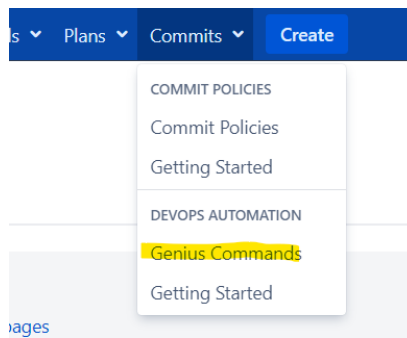
```
1 || Protect against the null value that caused the FOO-1 bug.
2 || @comment Affected input fields in the [Acme sign-up form|https://www.acme.com/
3 ||   sign-up]:
4 || * First name
5 || * Last name
6 || * Age
```

Komentarz ten zostanie dodany do zadania FOO-1: *Affected input fields in the Acme sign-up form:*

- *First name*
- *Last name*
- *Age...where "Acme sign-up form" is a link with list items below it!*

## 1.4 Spis dostępnych szablonowych commitów

Spis wszystkich podstawowych komend dostępnych znajduje się z zakładce Commits → Genius Commands.



Rysunek 12:

- **@affectsversion**: Ustawia wersję issue.
- **@analyze**: Rozpoczyna analizę kodu (z wykorzystaniem skonfigurowanych narzędzi).
- **@assign**: Przypisuje issue do użytkownika.
- **@build**: Wyzwalacz buildowania w systemie CI/CD.
- **@call**: Wysyła żądanie HTTP do (REST) API.
- **@comment**: Komentuje issue.
- **@fixversion**: Ustawia wersję poprawki issue.
- **@issuetype**: Ustawia typ issue.
- **@label**: Ustawia etykiety issue.
- **@priority**: Ustawia priorytet issue.
- **@run**: Uruchamia skrypt/program.
- **@status**: Ustawia status issue.
- **@time**: Rejestruje czas pracy nad issue.

Jeśli chcemy natomiast utworzyć własny, niestandardowy Genius Commit, to możemy to zrobić na końcu listy klikając w przycisk "Add command".

 A screenshot of the 'Add command' form in the application. The form is divided into three sections, each with a title, a command placeholder, and a description. 
   
1. The first section is titled 'Set issue status' and has a command placeholder: `@status {<status>[^\|]*}({:\|/{<resolution>[^\|]*})?`. The description says: 'Sets the status and optionally the resolution of the mentioned issue. (help)'. It includes a table of smart values for the '@status' command and example commit messages.
   
2. The second section is titled 'Set issue type' and has a command placeholder: `@issuetype {<issuetype>[a-zA-Z]}`. The description says: 'Sets the issue type of the mentioned issue. (help)'. It includes a table of smart values for the '@issuetype' command and an example commit message.
   
3. The third section is titled 'Transition issue' and has a command placeholder: `@transition {<transition>[^\|]*}({:\|/{<resolution>[^\|]*})?`. The description says: 'Transitions the mentioned issue and optionally updates the issue's resolution. (help)'. It includes a table of smart values for the '@transition' command and example commit messages.
   
At the bottom of the form, there is a '+ Add command' button and a 'Reset to default' link.

Rysunek 13:

## 2 Powiadomienia

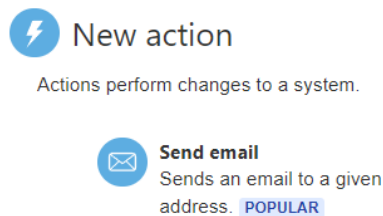
Wysyłanie powiadomień w systemie Jira stanowi niezbędny element sprawnego zarządzania projektami. To narzędzie umożliwia szybką i efektywną alokację zadań, przypisywanie ich do odpowiednich członków zespołu oraz monitorowanie postępu ich realizacji. Powiadomienia działają jak skuteczne ostrzeżenia, informując zespół o nowych zadaniach, zmianach w harmonogramie lub ważnych aktualizacjach dotyczących projektu.

Główne korzyści z wysyłania zautomatyzowanych powiadomień:

1. **Natychmiastowa informacja o zadaniach:** Automatyczne powiadomienia umożliwiają natychmiastowe powiadamianie członków zespołu o nowych zadaniach, zmianach w harmonogramie lub innych istotnych aktualizacjach. To eliminuje potrzebę manualnego śledzenia zmian i pozwala na szybkie reagowanie na nowe wyzwania.
2. **Zachęcanie do odpowiedzialności:** Automatyczne przypominanie o przypisanych zadaniach skutecznie zachęca członków zespołu do odpowiedzialnego podejścia do swoich obowiązków. To pomaga utrzymać dyscyplinę w realizacji zadań i przyspiesza tempo pracy.
3. **Szybsze reagowanie na zmiany:** Powiadomienia umożliwiają szybkie reagowanie na zmiany w projekcie, co jest istotne w dynamicznym środowisku biznesowym. Bezpośrednie informowanie zespołu o zmianach w wymaganiach, terminach czy priorytetach pozwala na dostosowywanie się do nowych warunków w czasie rzeczywistym.
4. **Optymalizacja czasu:** Dzięki automatyzacji procesu wysyłania powiadomień, zespół może skupić się na istotnych zadaniach bez konieczności ręcznego monitorowania zmian. To optymalizuje czas pracy i pozwala na skupienie się na kluczowych aspektach projektu.

### 2.1 E-mail

Aby stworzyć automatyzację należy wejść w zakładkę **Project settings**, **Automation** oraz stworzyć nową "zasadę". Będziemy się posługiwać akcją wysłania wiadomości e-mail [3].



Rysunek 14: Przycisk akcji: Wyślij e-mail

#### 2.1.1 Przykład automatyzacji z wysłaniem e-maila

Na potrzeby demonstracji możliwości możemy stworzyć automatyzację, która poinformuje naszego członka zespołu, że został właśnie przypisany do konkretnego zadania.



### Send email when issue is assigned

When: Issue assigned  
Rule is run when an issue is assigned to a user.

If: Assignee is not empty

**Then: Send email**  
Assignee  
Issue `*{{issue.key}}*` has been assigned to you.

### Send email

To  
x Assignee x

Cc Bcc

Subject  
Issue `*{{issue.key}}*` has been assigned to you.

Content  

```

<h2>Hi {{assignee.displayName}}, </h2>

<h5>a new issue "<strong>{{issue.summary}}</strong>" has been just assigned to you.

<a href="{{issue.url}}">You can check it here and start work right away!</a>

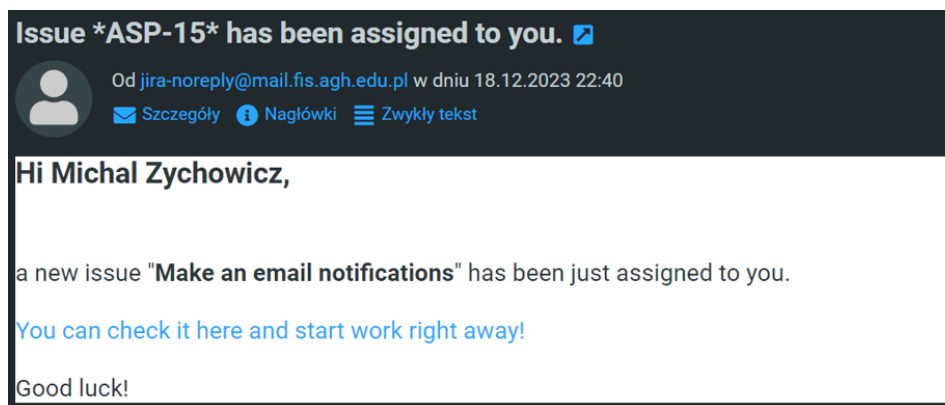
Good luck!</h5>

```

Rysunek 15: Informowanie o przydzieleniu użytkownika do "tasku" w systemie Jira

W Rys. 15 warunek `if` jest nam potrzebny, aby Jira nie wysyłała błędu autorowi automatyzacji, gdy żaden użytkownik nie jest przypisany jako wykonawca taska [4]. Dodatkowo konstruując wzór wiadomości e-mail korzystamy z HTML'a oraz mamy do dyspozycji tak zwane **Smart values**, które jedynie nam pomagają oraz wzbogacają wiadomość.

W efekcie możemy się spodziewać takiego o to rezultatu:



Rysunek 16: Efekt wykonania automatyzacji w systemie Jira z Rys. 15

## 2.2 Smart values

Termin ten odnosi się często do dynamicznych, zmiennych elementów używanych w różnych systemach i narzędziach, w tym w treści wiadomości e-mail. To zazwyczaj dane, które mogą być automatycznie wstawiane w treść wiadomości w zależności od pewnych warunków lub zdarzeń. Poniżej przedstawiamy kilka przykładów **smart values** wykorzystywanych w treściach powiadomień [5]:

- `{{issue.summary}}` – wypisze nam streszczenie danego taska;
- `{{issue.key}}` – wypisze nam klucz danego taska;

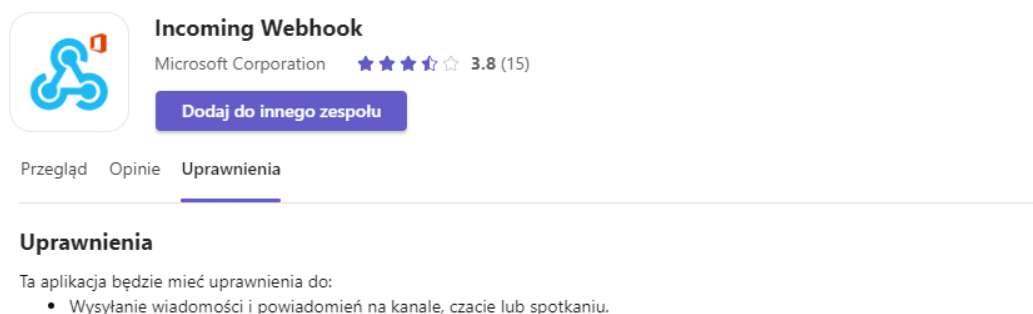
- `{{issue.url}}` – wypisze nam link danego taska;
- `{{assignee.displayName}}` – wypisze nam imię i nazwisko osoby wykonującej task;
- `{{issue.Reporter.emailAddress}}` – wypisze nam imię i nazwisko osoby wykonującej task;

Przykłady te ilustrują, jak "smart values" mogą być wykorzystywane w treści e-maili, aby dostarczać bardziej spersonalizowane, dynamiczne i skuteczne komunikaty. Warto jednak pamiętać o zgodności z przepisami dotyczącymi prywatności, takimi jak RODO, i unikaniu zbierania oraz wykorzystywania danych w sposób niezgodny z prawem.

## 2.3 Wiadomość Microsoft Teams

### 2.3.1 Webhook - medium transmisyjne?

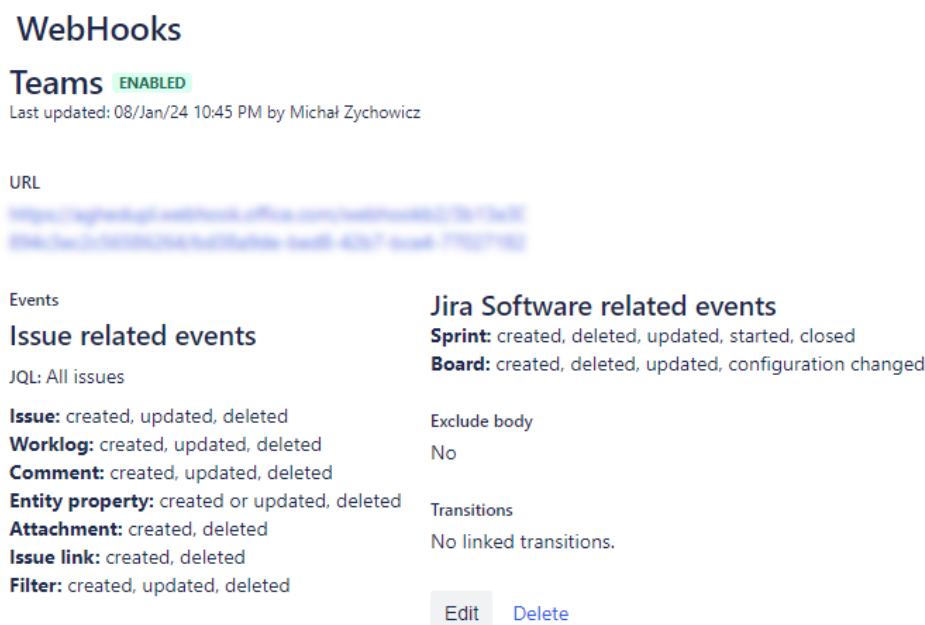
Mamy również możliwość wysyłania powiadomień na kanale w zespole. Aby się do tego przygotować należy posiadać zespół, którego jesteśmy właścicielami, następnie zainstalować aplikację **Incoming Webhook** prezentowaną na Rys. 17.



Rysunek 17: Aplikacja Incoming Webhook dostępna w MS Teams

Następnie należy przejść w: **Zarządzanie zespołem, Aplikacje**, kliknąć w powyższą aplikację oraz Skonfigurować łącznik podając *Nazwę* oraz *obraz* naszego "asystenta". Podany kod kopiujemy oraz trzymamy na później [7].

Logujemy się jako administrator oraz wchodzimy w **Ustawienia, System**, w sekcji *Advanced* w **WebHooks**. Tworząc nowy Webhook wybieramy *nazwę*, wklejamy nasz *adres URL* oraz wybieramy interesujące nas opcje:



Rysunek 18: Ustawienie Webhooka w **Ustawieniach** systemu Jira

### 2.3.2 Przykład automatyzacji z wysłaniem wiadomości MS Teams

Po tym etapie nie potrzebujemy już "boskich" uprawnień i jako szary użytkownik (z uprawnieniami do tworzenia automatyzacji) możemy uruchomić naszego bota wysyłającego powiadomienia na kanał jak pokazano na Rys. 19.

Teams

ENABLED

When: Issue transitioned

FROM

In Progress

TO

Done

Then: Send Microsoft Teams message

Thanks to {{assignee.displayName}} another issue has been completed.

Webhook URL \*

https://graph.microsoft.com/webhooks.office.com/webhooks/27617ba288-42

Message Title \*

Congratulations!


Message \*

Thanks to {{assignee.displayName}} another issue has been completed.


☒ Include issue summary in message

Rysunek 19: Automatyzacja z wykorzystaniem Wysyłania wiadomości MS Teams

Powyższa przykładowa automatyzacja wysyła podziękowanie za ukończenie konkretnego zadania co pokazuje, że również w tej formie powiadomień możemy korzystać z **smart values** [6]. Efekt przy zmianie stanu zadania (z "In Progress" na "Done") pokazano na Rys. 20.



J.A.R.V.I.S. 01:52 AM



**Congratulations!**

Thanks to Michał Zychowicz another issue has been completed.

AAF-1 Zadanie

Stan

Done


Typ zgłoszenia

Task

Przypisane do

Michał Zychowicz

[Wyświetl w Jira](#)

 Odpowiedz

Rysunek 20: Automatyzacja informująca o zrealizowaniu konkretnego zadania przez użytkownika

## 3 Script Runner

### 3.1 Co to Script Runner?

Jest to dodatek do Jiry (darmowy do 10 urządzeń) pozwalający na użycie skryptów Groovy wyzwalanych np. wydarzeniami czy czasem. Do komunikacji z Jirą wykorzystywane jest API REST [2] i żądania HTTP. Przykładowe [1] żądanie (w Groovy) wygląda tak:

```
1 def result = put("/rest/api/2/issue/${issueKey}")
2   .header('Content-Type', 'application/json')
3   .body([
4       fields: [
5           summary: newSummary
6       ]
7   ])
8   .asString()
```

Do zmiennej result typu string przekazujemy wynik żądania [8]. put() jako argument pobiera "endpoint", w tym wypadku jest to nasz issue. Metoda Put służy do modyfikacji zasobów na serwerze. W nagłówku .header() podajemy typ zawartości, jest nim JSON. W ciele .body() żądania opisujemy jakie dane chcemy pobrać/zmodyfikować, .asString() wykonuje żądanie i zwraca rezultat w stringu. W dodatku, jeśli wykonanie było poprawne result.status powinno wynosić 204. Kod 204 oznacza sukces, ale bez zwrotu danych, nasz kod i tak ich nie oczekuje. W skrócie, kody odpowiedzi z zakresu 100 - 199 są informacyjne, często specyficzne dla danego programu, 200 - 209 oznaczają sukces, 300-399 przekierowania zasobów, 400-499 błędy po stronie klienta, a 500-599 błędy po stronie serwera.

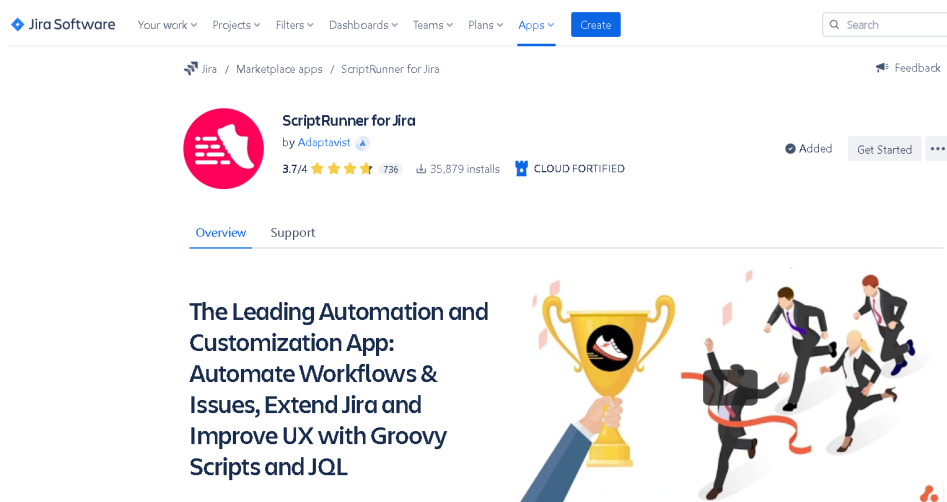
GET	Pobiera dane z zasobu, bez modyfikacji
PUT	Dokonuje całkowitej zmiany w zasobie, kolejne wykonania mają każdy ten sam efekt
POST	Wysyła dane na serwer, kolejne wykonania POST mogą mieć wzajemnie różne efekty.
DELETE	Usuwa zasób

Tabela 1: Porównanie metod HTTP, używanych często w skryptach

Metodę POST wykorzystuje się często np. do dodawania nowych użytkowników czy sklejania baz danych - operacji w których poprzednie dane dalej trwają. PUT - gdy zostają całkowicie zmienione, np. zamiana użytkownika.

### 3.2 Instalacja

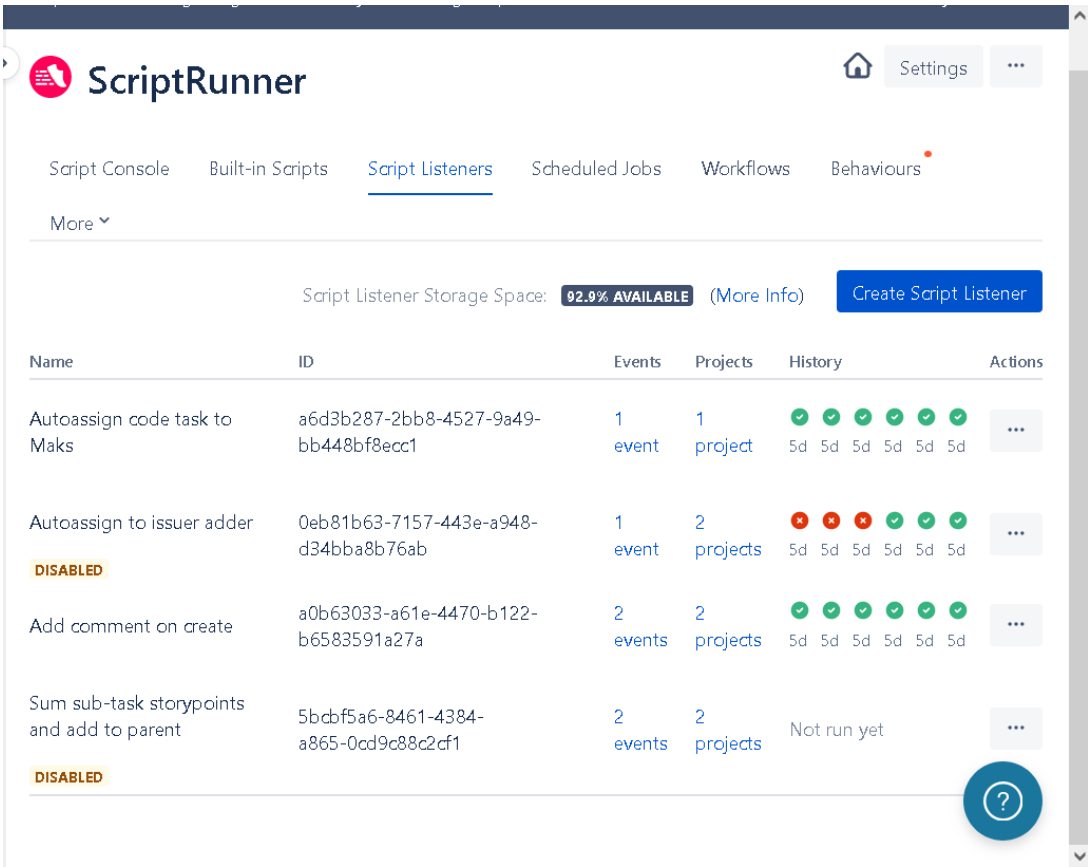
Aplikacja dostępna w sekcji Apps na Jira Cloud, jak również jako plik .jar do instalacji na własnej instancji.



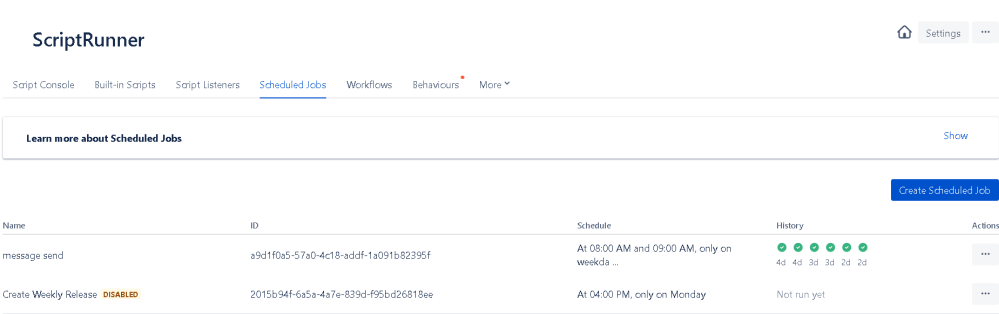
Rysunek 21: Instalacja

### 3.3 Przykładowe Skrypty

#### 3.3.1 Wstęp



Rysunek 22: Menu Script Listeners, widoczna historia wykonywania skryptu

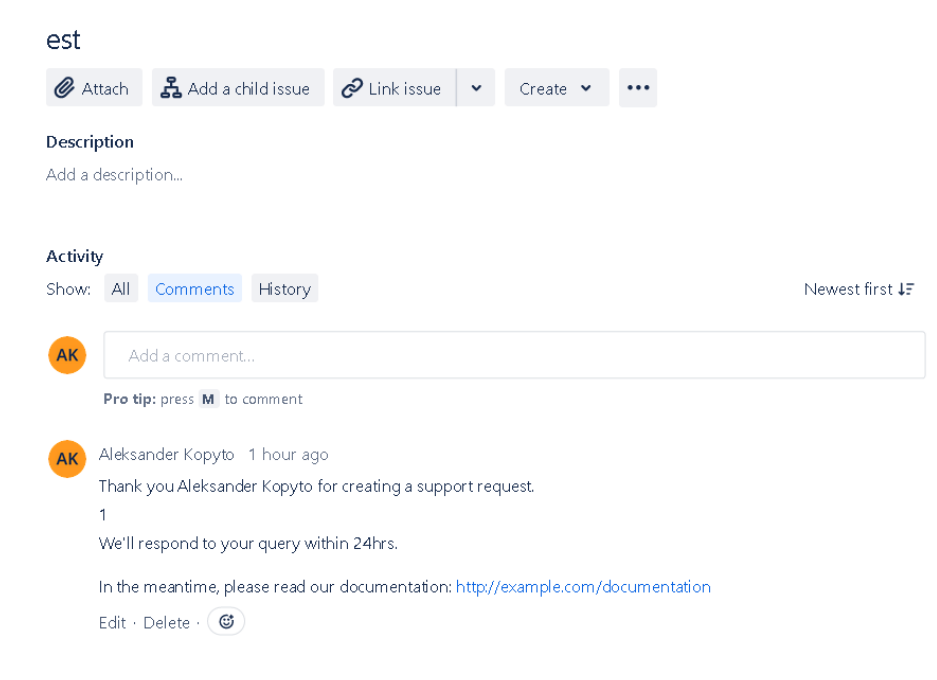


Rysunek 23: Menu Scheduled Jobs, Skrypt będzie wykonany pomiędzy 8 a 9, ale nie mamy gwarancji dokładnej chwili wykonania, jedynie przedział.

### 3.3.2 Dodawanie komentarza do nowo utworzonych skryptów

```
1 def projectKey = 'EDP'
2
3 // Restrict this script to only run against in one project
4 if (issue.fields.project.key != projectKey) {
5     return
6 }
7
8 def author = issue.fields.creator.displayName
9
10 // Add a plain-text comment, see https://docs.atlassian.com/jira/REST/cloud/#api/2/issue/{issueIdOrKey}/comment-addComment for more details
11
12 def commentResp = post("/rest/api/2/issue/${issue.key}/comment$")
13     .header('Content-Type', 'application/json')
14     .body([
15         body: """Komentarz"""
16     ])
17     .asObject(Map)
18
19 assert commentResp.status == 201
```

Skrypt ten automatyzuje dodawanie komentarza do zadania, ale tylko wtedy, gdy zadanie należy do określonego projektu o kluczu 'EDP'. Komentarz jest zdefiniowany jako "Komentarz" w treści skryptu.



Rysunek 24: Wynik działania skryptu

### 3.3.3 Automatyczne przypisywanie zadań autorowi issue

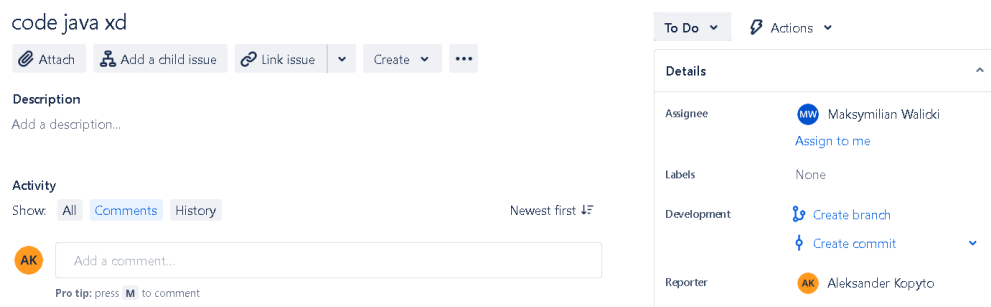
```
1 def projectKey = 'EDP'
2
3 // Ogranicza skrypt dla pol wewnatrz projektu
4
5 if (issue.fields.project.key != projectKey) {
6     return
7 }
8
9 // Pobieramy wlasne id autora, jira cloud operuje na accountId a nie na name
10
11 def authorAccountId = issue.fields.creator.accountId
12
13 // korzystajac z api restowego, metody put ustawiamy assignee, wazne asString – nie
14 // mozemy przekazac id innego typu niz string
15 def response = put("/rest/api/2/issue/${issue.key}/assignee")
16     .header('Content-Type', 'application/json')
17     .body([accountId: authorAccountId])
18     .asString()
```

Skrypt ten automatyzuje proces przypisywania zadań, ustawiając przypisanego użytkownika na tego, który utworzył zadanie, ale tylko w przypadku, gdy zadanie należy do określonego projektu o kluczu 'EDP'.

### 3.3.4 Automatyczne przypisywanie zadań na podstawie tytułów

```
1 def projectKey = 'EDP'
2
3 // Ogranicza skrypt dla p l wewn trz projektu
4 if (issue.fields.project.key != projectKey) {
5     return
6 }
7 def issueName= issue.fields.summary;
8 //proste wyszukiwanie sĆowa "code"
9 def isCode = issueName.toLowerCase().matches(".*code.*")
10 if (isCode){
11     // Z adresu url profilu uzytkownika Maks Walicki pobieramy jego Id
12     //jira cloud operuje na accountId a nie na name
13     def authorAccountId = "712020:0e74deb3-da3b-48d6-b2c8-99b07abb32fd" //Maks
14         Walicki, ew. issue.fields.creator.accountId
15
16     // korzystajac z api restowego, metody put ustawiamy assignee, wazne jest
17     // asString – nie mozemy przekazac id innego typu niz string
18
19     def response = put("/rest/api/2/issue/${issue.key}/assignee")
20         .header('Content-Type', 'application/json')
21         .body([accountId: authorAccountId])
22         .asString()
23 }
```

Skrypt ten automatyzuje przypisanie zadań do określonego użytkownika (w przykładzie Maksowi Walickiemu) na podstawie warunku, że w nazwie zadania występuje słowo "code".



Rysunek 25: Utworzone zadanie zostało przypisane podanemu w kodzie użytkownikowi

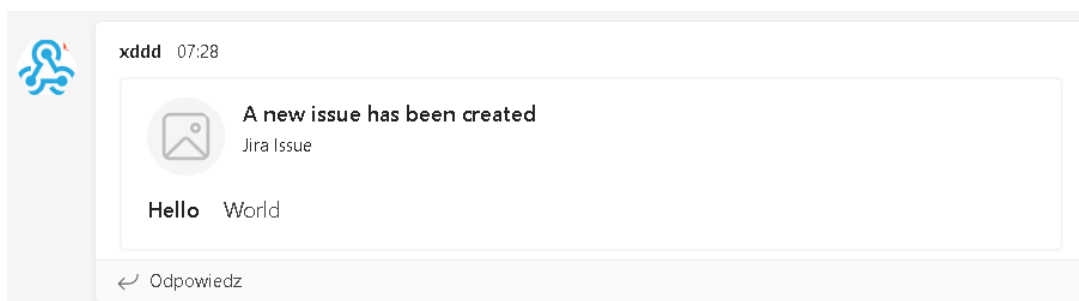
### 3.3.5 Łączność z Microsoft Teams

```

1 def projectKey = 'EDP'
2
3 def webhook = "https://aghedupl.webhook.office.com/webhookb2/3b13e308-93f0-49c3-
4 a184-46293b04d539@80b1033f-21e0-4a82-bbc0-f05fdccd3bc8/IncomingWebhook/5cc7
5 dcc12b9b42fa860364f4f63937e2/3aef51bf-032e-47c2-a003-123529199b49"
6
7 def message = [
8     "@type": "MessageCard",
9     "summary": "Issue Update",
10    "sections": [
11        [
12            "activityTitle": "A new issue has been created",
13            "activitySubtitle": "Jira Issue",
14            "activityImage": "http://www.example.com/images/JiraLogo.png",
15            "facts": [
16                [
17                    "name": "Hello",
18                    "value": "World"
19                ]
20            ],
21            "markdown": true
22        ]
23    ]
24
25 post(webhook).header('Content-Type', 'application/json').body(message).asString()

```

Skrypt ten wysyła komunikat na określony webhook po utworzeniu nowego zadania w projekcie 'EDP'. Treść komunikatu jest dostosowana do formatu obsługiwanego przez usługę, która nasłuchuje na tym webhooku.



Rysunek 26: Wiadomość na teams



## Bibliografia

- [1] Adaptavist. *Dokumentacja ScriptRunner*. Ostatni dostęp: 15.01.2024 r. 2024. URL: <https://docs.adaptavist.com/sr4jc/latest/features/script-console/script-examples>.
- [2] Atlassian. *Dokumentacja REST API Jiry*. Ostatni dostęp: 09.01.2024 r. 2024. URL: <https://developer.atlassian.com/cloud/jira/platform/rest/v2/>.
- [3] Atlassian. *Jira automation actions*. Ostatni dostęp: 09.01.2024 r. 2024. URL: <https://support.atlassian.com/cloud-automation/docs/jira-automation-actions/>.
- [4] Atlassian. *Jira automation conditions*. Ostatni dostęp: 09.01.2024 r. 2024. URL: <https://support.atlassian.com/cloud-automation/docs/jira-automation-conditions/>.
- [5] Atlassian. *Jira smart values - issues*. Ostatni dostęp: 09.01.2024 r. 2024. URL: <https://support.atlassian.com/cloud-automation/docs/jira-smart-values-issues/>.
- [6] Atlassian. *Use automation with Microsoft Teams*. Ostatni dostęp: 09.01.2024 r. 2024. URL: <https://support.atlassian.com/cloud-automation/docs/use-automation-with-microsoft-teams/>.
- [7] Microsoft. *Create Incoming Webhooks*. Ostatni dostęp: 09.01.2024 r. 2023. URL: <https://learn.microsoft.com/en-us/microsoftteams/platform/webhooks-and-connectors/how-to/add-incoming-webhook>.
- [8] Mozilla. *Dokumentacja HTTP*. Ostatni dostęp: 15.01.2024 r. 2024. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP>.