Jakub Kalinowski

Atlassian Jira Server & Data Center

Programowanie rozwiązań w projektach biznesowych

Helion

Redaktor prowadzący: Szymon Szwajger

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki
Grafika na okładce została wykorzystana za zgodą Shutterstock.com.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/atjise
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Materiały do książki znaleźć można pod adresem:
https://ftp.helion.pl/przyklady/atjise.zip

ISBN: 978-83-283-9784-2

Printed in Poland.

Spis treści

ROZDZIAŁ 5. Testowanie zmian

W rozdziale zostaną omówione sposoby logowania zdarzeń i błędów w instancji Jiry, wykorzystanie wbudowanych narzędzi do audytu i zestaw praktycznych porad związanych z weryfikacją poprawności działania skryptów języka Groovy.

5.1. Logi

Uruchomiona Jira generuje dwa podstawowe grupy logów, tj. aplikacyjne i systemowe. Pierwsze z nich zawierają zapis zdarzeń będących efektem pracy samej instancji Jiry, drugie zaś logi środowiska uruchomieniowego, na które składa się system operacyjny, serwer bazy danych i serwer Tomcat[1]. Logi można przeglądać przez zwykłą edycję pliku lub w trybie na żywo po użyciu polecenia uniksowego `tail`[2]. Oba przykłady zostały przedstawione w listingu 5.1.

LISTING 5.1. Przegląd logów aplikacyjnych z poziomu konsoli Linuksa

```
// 1. Otwarcie konsoli ssh i połączenie z serwerem wirtualnym.
// 2. Przeglądanie aktualnie nadpisanego pliku logów aplikacyjnych.
sudo su
cd /var/atlassian/application-data/jira/log
nano atlassian-jira.log
// 3. Wyświetlanie logów aplikacyjnych na żywo.
// Wyjście z tail jest możliwe np. za pomocą kombinacji klawiszy CTRL + c.
sudo su
cd /var/atlassian/application-data/jira/log
tail -f atlassian-jira.log
```

5.1.1. Application logs (logi aplikacyjne)

W skład logów aplikacyjnych wchodzą pliki zlokalizowane domyślnie w katalogu */var/atlassian/application-data/jira/log*. Na poniższej liście zostały wymienione logi szczególnie przydatne w trakcie testowania zmian i diagnozowania problemów. Zestaw wszystkich pozostałych logów został opisany w dokumentacji Atlassian[3].

- *atlassian-jira.log* — podstawowy plik logów, zawierający większość zdarzeń związanych z pracą aplikacji, w tym informacje o błędach poszczególnych modułów.

---

[1] *Apache Tomcat*, *https://tomcat.apache.org/* [dostęp: 17.07.2022].

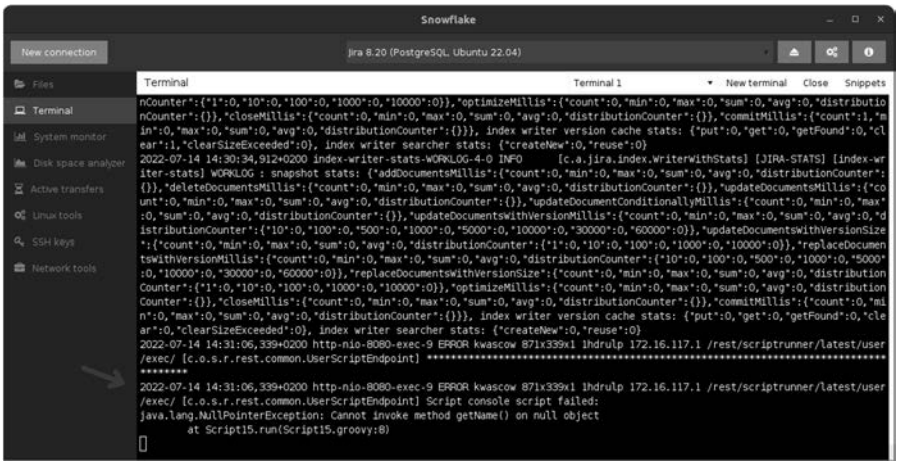
[2] *Tail*, *http://www.polarhome.com/service/man/?qf=tail&tf=2&of=OpenIndiana&sf=1* [dostęp: 17.07.2022].

[3] *Logging and profiling*, *https://confluence.atlassian.com/adminjiraserver/logging-and-profiling-938847671.html* [dostęp: 14.07.2022].

- *atlassian-jira-slow-queries.log* — zapis zapytań JQL, których czas wykonywania był dłuższy niż zdefiniowano w ustawieniach. Domyślną wartością jest 400 ms.
- *atlassian-jira-sql.log* — zapis logów połączeń z bazą danych.
- **logi pluginów** — podstawowe logi zainstalowanych pluginów, w tym pluginu *Insight - Asset Management*[4], *ScriptRunner*[5] i innych.

Sposób działania logów można przetestować na prostym przykładzie.

1. Należy połączyć się z serwerem aplikacji za pomocą klienta `ssh` i uruchomić komendę `tail` na pliku *atlassian-jira.log*, tak jak to przedstawiono w listingu 5.2. Punkt ten można oczywiście pominąć i otworzyć plik logu w dowolnym edytorze po wykonaniu operacji z punktu 3.
2. W oknie konsoli `ssh` można zauważyć nowe logi, powstające w trakcie wykonywania operacji w Jirze.
3. W przeglądarce należy zalogować się do Jiry i przejść do konsoli Groovy'ego (*Administration/Manage apps/ScriptRunner/Console*), a następnie wykonać skrypt, który będzie próbował pobrać *username* z pustego obiektu użytkownika, tak jak to przedstawiono w listingu 5.2.
4. Po wykonaniu skryptu z listingu 5.2 w oknie konsoli `ssh` zostanie wyświetlony nowy błąd, dotyczący tej operacji, tak jak to przedstawiono na rysunku 5.1.



RYSUNEK 5.1. Widok błędu w logu atlassian-jira.log

[4] *Insight - Asset Management*, *https://marketplace.atlassian.com/apps/1212137/insight-asset-management?tab=overview&hosting=datacenter* [dostęp: 17.07.2022].

[5] *ScriptRunner for Jira*, https://marketplace.atlassian.com/apps/6820/scriptrunner-for-jira?tab=overview&hosting=server [dostęp: 17.07.2022].

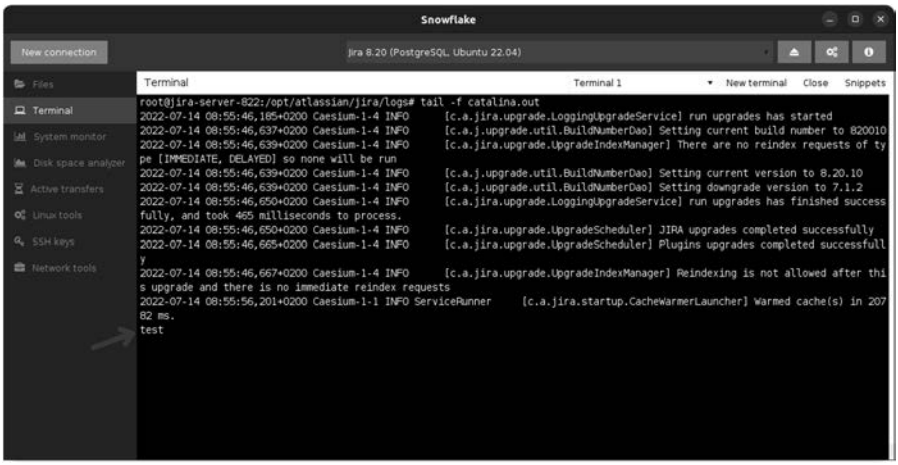
Listing 5.2. Wykonanie błędnego kodu w konsoli Groovy'ego

```
import com.atlassian.jira.bc.user.UserService;
import com.atlassian.jira.component.ComponentAccessor;
// 1. Wstrzyknięcie komponentu UserService.
def userService = ComponentAccessor.getComponent(UserService);
// 2. Próba pobrania obiektu typu interfejsowego ApplicationUser nieistniejącego użytkownika.
def userToChanger = ComponentAccessor
  .getUserManager().getUserByName("mkopernik1234567890").getName();
```

5.1.2. System logs (logi systemowe)

Drugim ważnym źródłem informacji o pracy instancji Jiry są logi systemowe, które przechowują np. zdarzenia wywołane w środowisku uruchomieniowym Jiry, a więc m.in. w serwerze Tomcat[6]. Oprócz nich znajdują się też logi dostępowe, wpisy związane z pracą aplikacji *garbage collector*[7] i inne. Pliki zlokalizowane są domyślnie w katalogu */opt/atlassian/jira/logs*. Na liście poniżej wymieniono najbardziej przydatne logi z punktu widzenia programisty i administratora. Zestaw wszystkich pozostałych został opisany w dokumentacji Atlassian[8].

- *catalina.out* — logi serwera Tomcat.
- *access_log.<data>* — rejestracja dostępów do Jiry.
- *atlassian-jira-gc.<data>* — logi wirtualnej maszyny Javy (JVM) rejestrujące ilość pamięci zajętej przez nieużywane obiekty, które nie zostały jeszcze zutylizowane przez aplikację *garbage collector*.



RYSUNEK 5.2. Widok wpisu do logu catalina.out

[6] *Apache Tomcat*, *https://tomcat.apache.org/* [dostęp: 17.07.2022].

[7] *Garbage Collector Performance Issues*, *https://confluence.atlassian.com/doc/garbage-collector-performance-issues-200707997.html* [dostęp: 17.07.2022].

[8] *Logging and profiling*, *https://confluence.atlassian.com/adminjiraserver/logging-and-profiling-938847671.html* [dostęp: 14.07.2022].

Po uruchomieniu skryptu z listingu 5.2, przedstawionego w poprzednim podrozdziale, można zauważyć, że do logu *catalina.out* nie zostanie zapisany żaden błąd związany z tą operacją. Dzieje się tak dlatego, że działanie skryptu było zdarzeniem aplikacyjnym, niewpływającym w żaden sposób na działanie systemu. Jeżeli natomiast w skrypcie zostanie wykonana operacja typu `println "test";`, czyli wyświetlenie jakiejś linii tekstu w konsoli, to taki wpis już się pojawi w logu *catalina.out*, tak jak to przedstawiono na rysunku 5.2.

5.2. Loggers (loggery)

Logowanie zdarzeń na poziomie instancji Jiry jest realizowane za pomocą modułu `log4j`[9]. Konfiguracja poszczególnych loggerów jest możliwa za pomocą sekcji *Administration/System/Logging and profiling*, w której można włączyć logowanie dla zapytań HTTP, bazy danych, a także skonfigurować loggery dla poszczególnych pakietów, w tym dla pluginów. Umożliwia to precyzyjne tworzenie logów dla wybranych elementów systemu i wskazanie poziomu logowania, czyli sytuacji, w których dane zdarzenie zostanie odnotowane, tj.: `trace`, `debug`, `info`, `warn`, `error`, `fatal` lub *wyłączenie logowania*[10]. W tabeli 5.1 przedstawiono schemat zasięgu logowania dla poszczególnych poziomów ustawionych w sekcji *Logging and profiling*.

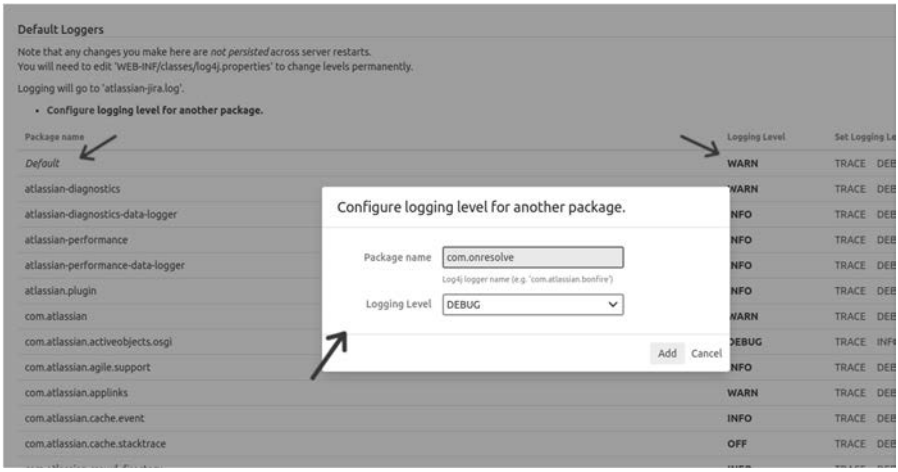
TABELA 5.1. Zasięg poziomów logowania w Jirze

POZIOM LOGOWANIA	TYP LOGGERA					
	FATAL	ERROR	WARN	INFO	DEBUG	TRACE
OFF						
FATAL	X					
ERROR	X	X				
WARN	X	X	X			
INFO	X	X	X	X		
DEBUG	X	X	X	X	X	
TRACE	X	X	X	X	X	X

W przypadku braku zdefiniowanego loggera dla któregoś z komponentów systemu korzysta on z domyślnego poziomu logowania, określonego w sekcji *Logging and profiling*. W celu dodania loggera dla pakietu należy odczytać początek nazwy *App key* z danych pluginu (*Administration/Manage apps/Manage apps/<rozwinięcie wiersza właściwego pluginu>*) i zdefiniować logger, tak jak to przedstawiono na rysunku 5.3.

[9] *Apache Log4j 2*, *https://logging.apache.org/log4j/2.x/* [dostęp: 14.07.2022].

[10] *Class Level*, *https://logging.apache.org/log4j/2.x/log4j-api/apidocs/index.html* [dostęp: 14.07.2022].



RYSUNEK 5.3. Dodanie loggera dla pluginu ScriptRunner[11]

Logowanie w skryptach wykonuje się za pomocą domyślnie zainicjalizowanego obiektu loggera, dostępnego pod zmienną `log`. Dobrą praktyką jest stosowanie loggerów w kluczowych operacjach skryptu, których wpisy umożliwią wyszukanie informacji o wykonaniu danego fragmentu kodu w pliku *atlassian-jira.log*. W listingu 5.3 został przedstawiony sposób wywołania loggerów w skryptach Groovy'ego.

LISTING 5.3. Użycie loggera w skryptach Groovy'ego

```
// 1. Wywołanie loggerów przy ustawieniu poziomu DEBUG w ustawieniach Logging and profiling.
log.fatal("fatal test");
log.error("error test");
log.warn("warn test");
log.info ("info test");
log.debug("debug test");
log.trace("trace test");
// 2. Wyświetlenie wyniku.
/*
2022-07-14 19:41:47,476 FATAL [runner.ScriptBindingsManager]: fatal test
2022-07-14 19:41:47,476 ERROR [runner.ScriptBindingsManager]: error test
2022-07-14 19:41:47,476 WARN [runner.ScriptBindingsManager]: warn test
2022-07-14 19:41:47,476 INFO [runner.ScriptBindingsManager]: info test
*/
```

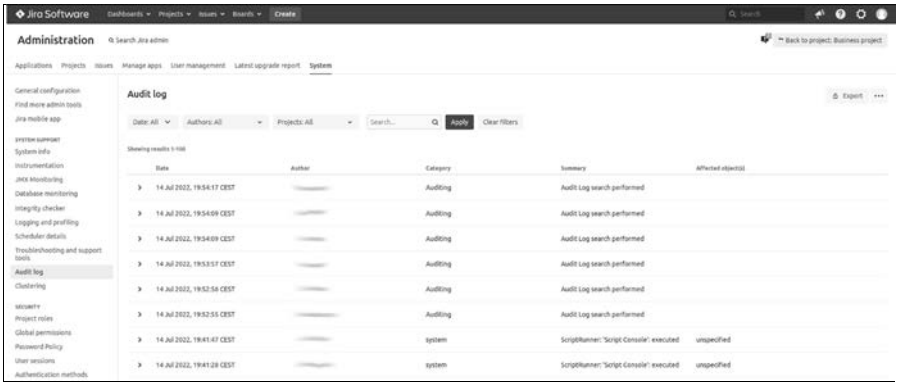
5.3. Audit log (analizator dzienników)

Audit log jest narzędziem administracyjnym Jiry umożliwiającym przegląd niektórych aktywności użytkowników w instancji. Z uwagi na ograniczony zakres przechowywanych danych stosuje się go najczęściej do szybkiej weryfikacji zmian w następujących zakresach:

[11] *ScriptRunner for Jira, https://marketplace.atlassian.com/apps/6820/scriptrunner-for-jira?tab=overview&hosting=server* [dostęp: 17.07.2022].

- *Global permissions* — ustawienia globalne modyfikowane przez administratora.
- Zmiany w bazie użytkowników i grupach.
- Modyfikacja *permissions schemes* dla projektów.
- Niektóre operacje wykonywane na projektach, np. usunięcie i dodanie projektu.
- Niektóre działania pluginów.

W celu użycia narzędzia należy przejść do sekcji *Admin/System/Audit log* i wyszukać dane za pomocą podręcznych filtrów, tak jak to przedstawiono na rysunku 5.4. Więcej informacji o analizatorze dzienników można znaleźć na stronie *Atlassian Support*[12].



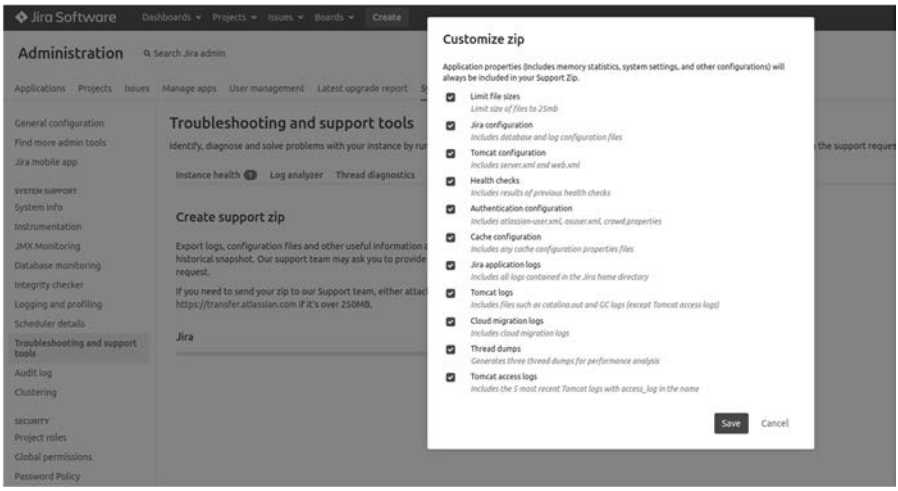
RYSUNEK 5.4. Widok narzędzia Audit log

5.4. Support zip (plik dla działu wsparcia)

Funkcjonalność tworzenia paczki *support zip* została utworzona w celu diagnozowania błędów przez zespół wsparcia Atlassian. Osoby posiadające aktywną usługę `maintenance` przy zgłaszaniu problemów do zespołu wsparcia są zwykle proszone o wygenerowanie aktualnego *support zip* i dołączenie go do zgłoszenia. Plik ten może mieć też inne zastosowanie. Zdarzają się sytuacje, w których programista pracujący w danej instancji jest administratorem Jiry, ale nie ma dostępu do serwera aplikacji. W takim przypadku plik dla działu wsparcia jest jedynym źródłem informacji o logach systemowych i aplikacyjnych.

W celu utworzenia pliku *support zip* należy przejść do sekcji *Administration/System Troubleshooting and support tools/Create support zip/Customize zip* i skonfigurować dane, które mają zostać pobrane, np. tak jak to przedstawiono na rysunku 5.5.

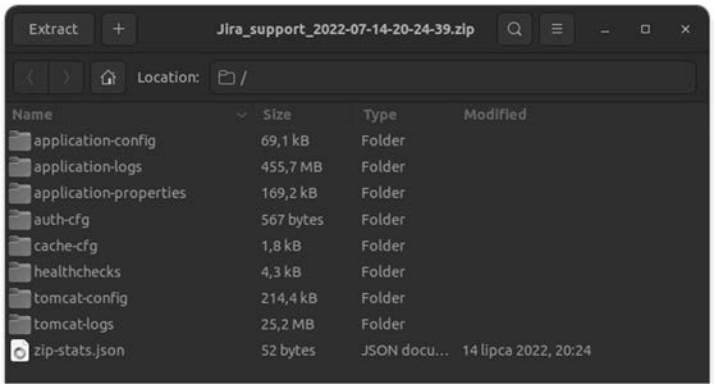
[12] *Auditing in Jira*, *https://confluence.atlassian.com/adminjiraserver/auditing-in-jira-938847740.html* [dostęp: 14.07.2022].



RYSUNEK 5.5. Konfiguracja pliku support zip

Po skonfigurowaniu zakresu paczki wystarczy kliknąć przycisk *Create zip*, aby ją pobrać. Przykładowa zawartość rozpakowanego katalogu została przedstawiona na rysunku 5.6.

RYSUNEK 5.6. Zawartość pliku support zip



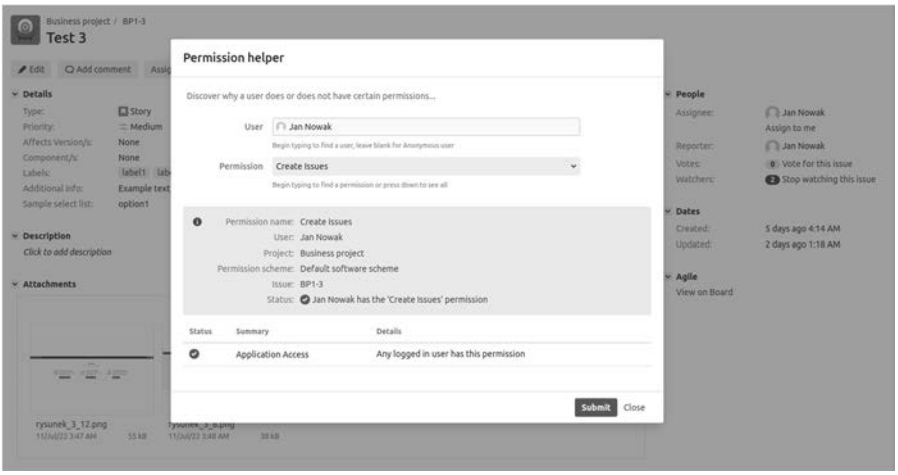
5.5. Testowanie uprawnień użytkowników

Dość częstą przyczyną błędów w tworzonych rozwiązaniach skryptowych jest brak uprawnień użytkownika wykonującego daną operację do projektu, zgłoszenia lub brak któregoś z uprawnień globalnych. Jira i plugin ScriptRunner[13] oferują kilka szybkich sposobów weryfikacji uprawnień, co może pomóc w ustaleniu przyczyny błędu.

[13] *ScriptRunner for Jira*, *https://marketplace.atlassian.com/apps/6820/scriptrunner-for-jira?tab=overview&hosting=server* [dostęp: 17.07.2022].

5.5.1. Permission helper

Pierwszym omawianym narzędziem jest *permission helper* — natywne rozwiązanie Jiry, które umożliwia zbadanie uprawnień konkretnego użytkownika. Funkcjonalność jest dostępna z poziomu każdego zgłoszenia (*Issue/Admin* [menu pod podsumowaniem zgłoszenia (*Summary*)]/*Permission helper*) i w panelu administracyjnym (*Administration/System/Admin helper/Permission helper*). Przykładowy widok weryfikacji został przedstawiony na rysunku 5.7.



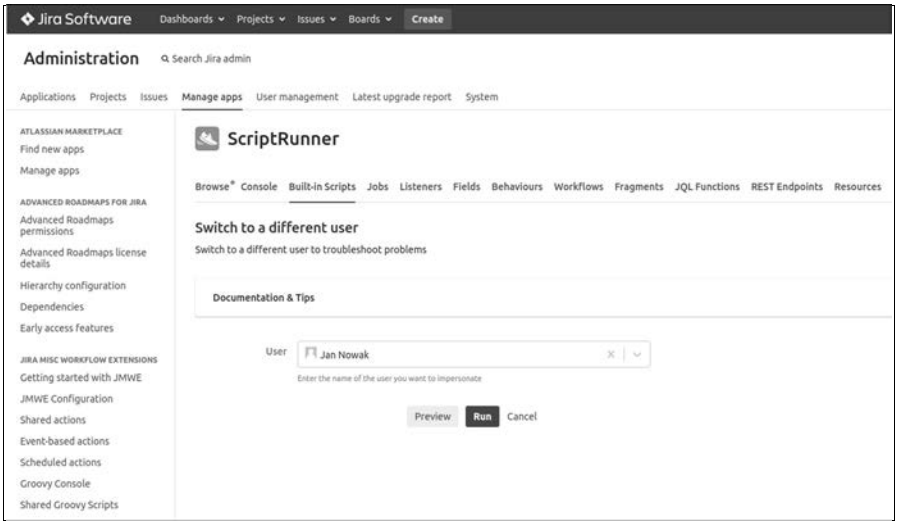
RYSUNEK 5.7. Weryfikacja uprawnień użytkownika za pomocą permission helper

5.5.2. Switch to a different user

Drugim sposobem weryfikacji uprawnień użytkownika jest zalogowanie się na jego konto za pomocą funkcjonalności *Switch to a different user* dostarczanej przez plugin ScriptRunner[14]. Rozwiązanie to jest o tyle wszechstronniejsze niż *permission helper*, że umożliwia przetestowanie przygotowanego rozwiązania tak, jakby to robił użytkownik końcowy.

W celu przełączenia użytkownika najlepiej otworzyć nową sesję Jiry w trybie *incognito* przeglądarki, aby uniknąć wylogowania z naszego konta, i przejść do sekcji *Administration/Manage apps/ScriptRunner/Built-in Scripts/Switch to a different user*. Na rysunku 5.8 został przedstawiony widok ekranu przełączania użytkownika.

[14] *ScriptRunner for Jira*, *https://marketplace.atlassian.com/apps/6820/scriptrunner-for-jira?tab=overview&hosting=server* [dostęp: 17.07.2022].

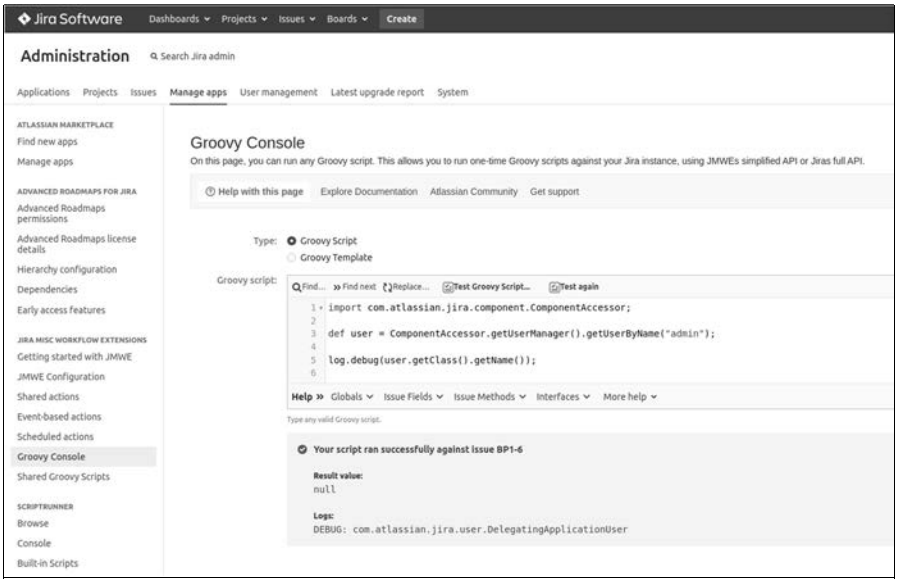


RYSUNEK 5.8. Przełączanie użytkownika za pomocą ScriptRunnera

5.6. Sposoby testowania skryptów

Pisanie skryptów automatyzujących instancję Jiry jest o tyle specyficzne, że twórca nie dysponuje typowym środowiskiem programistycznym z zestawem przydatnych narzędzi takich jak zarządzanie zależnościami, konsola, uruchomienie w trybie debugowania, różnego rodzaju generatory itp. Jedyne, na co może liczyć, to lepiej lub gorzej działający mechanizm podpowiedzi składni w konsoli i błędy zwracane przez kompilator przy uruchamianiu skryptu.

Aby w jakiś sposób kontrolować wartości zmiennych i móc na bieżąco sprawdzać poprawność działania kodu, należy utworzyć coś w rodzaju własnego debugowania opartego na logach lub wartościach zwracanych przez skrypt. Dane takie można następnie odczytać w logach pluginu, logach aplikacyjnych i systemowych lub zwrócić je jako wynik za pomocą słowa kluczowego `return`. Przykłady takich rozwiązań zostały przedstawione na rysunkach 5.9 i 5.10.



RYSUNEK 5.9. Sprawdzenie typu obiektu przechowywanego w zmiennej user za pomocą logów w JMWE[15]

RYSUNEK 5.10. Sprawdzenie typu obiektu przechowywanego w zmiennej user za pomocą logów w ScriptRunnerze[16]

[15] *Jira Misc Workflow Extensions (JMWE), https://marketplace.atlassian.com/apps/292/jira-misc-workflow-extensions-jmwe?tab=overview&hosting=server* [dostęp: 17.07.2022].

[16] *ScriptRunner for Jira, https://marketplace.atlassian.com/apps/6820/scriptrunner-for-jira?tab=overview&hosting=server* [dostęp: 17.07.2022].

By szybciej, sprawniej i skuteczniej zarządzać projektami

Znalezienie odpowiedzi na zaprzątające umysł pytania z zakresu automatyzacji zwykle wiąże się ze spędzeniem długich godzin na wertowaniu niespójnej dokumentacji lub przeszukiwaniu forum Atlassian Community, gdzie swoimi problemami (a także autorskimi pomysłami na ich rozwiązanie) dzielą się inni użytkownicy Jiry. To jednak często bezproduktywne działanie, a jedynym skutecznym — albo i nieskutecznym — sposobem, by pokonać przeszkodę, jest metoda prób i błędów. W szczególności dotyczy to klas, metod i interfejsów, o których możemy jedynie powiedzieć, że istnieją i przyjmują określone typy parametrów.

Książka Jakuba Kalinowskiego ma w zamyśle oszczędzić czytelnikom żmudnego procesu pozyskiwania wiedzy i dać wskazówki, jak rozwiązać konkretne problemy, z którymi mogą się oni zetknąć podczas pracy z Atlassian Jira Server. Znalazło się tu omówienie takich kwestii jak *custom fields*, *workflows*, integracje z API, funkcjonalności dostarczane przez pluginy, a także tego, co niesłusznie jest uznawane za *terra incognita*, czyli tworzenia własnych pluginów. Autor uzupełnił treść o kilka zagadnień związanych z integracją Jiry i Confluence, jak również z bazami Insight – Asset Management, ponieważ obie aplikacje często są obecne w projektach biznesowych, a ich automatyzacja zazwyczaj przysparza wielu problemów.

To książka adresowana przede wszystkim do czytelników, którzy mają już doświadczenie w zarządzaniu projektami i instancjami Jiry, jednak do tej pory nie zdecydowali się na pisanie skryptów automatyzujących.

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-9784-2

9 788328 397842

Cena: 69,00 zł