



Jakub Kalinowski

Atlassian **Jira Server & Data Center**

Programowanie rozwiązań
w projektach biznesowych

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Szymon Sz wajger

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

Copyright © Helion S.A. 2023

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

	Od autora	13
	Wprowadzenie	15
	CZĘŚĆ I. PRZYGOTOWANIE ŚRODOWISKA PRACY	19
ROZDZIAŁ 1.	Instalacja własnej instancji Jiry	21
	1.1. Warunki początkowe	21
	1.2. Przygotowanie systemu operacyjnego	22
	1.3. Instalacja bazy danych	24
	1.4. Instalacja Jira Software Server/Data Center	25
	1.5. Pierwsze uruchomienie Jiry	27
	CZĘŚĆ II. DOKUMENTACJA I NARZĘDZIA DEWELOPERA	31
ROZDZIAŁ 2.	Składnia Groovy'ego	33
	2.1. Uwagi dla programistów Javy	33
	2.2. Import bibliotek	34
	2.3. Deklaracja i inicjalizacja zmiennych	35
	2.4. Tworzenie funkcji	36
	2.5. Wykorzystanie klas	38
	2.6. Instrukcje sterujące i pętle	40
	2.6.1. Instrukcja if... else...	41
	2.6.2. Pętla for in	41
	2.6.3. Pętle each() i eachWithIndex()	42
	2.6.4. Metody find() i findAll()	43
ROZDZIAŁ 3.	Atlassian Java APIs	44
	3.1. Accessors (akcesory)	44
	3.1.1. ComponentAccessor	44
	3.1.2. PluginAccessor	45
	3.2. Managers (menedżery)	45
	3.2.1. AttachmentManager	45
	3.2.2. ChangeHistoryManager	46
	3.2.3. CommentManager	46
	3.2.4. CustomFieldManager	47
	3.2.5. GroupManager	48
	3.2.6. IssueLinkManager	48
	3.2.7. IssueManager	49
	3.2.8. LabelManager	49
	3.2.9. OptionsManager	50
	3.2.10. PriorityManager	51

3.2.11. ProjectComponentManager	51
3.2.12. ProjectManager	52
3.2.13. ProjectRoleManager	52
3.2.14. SubTaskManager	53
3.2.15. UserManager	53
3.2.16. VersionManager	54
3.3. Services (serwisy)	55
3.3.1. IssueService	55
3.3.2. IssueIndexingService	56
3.3.3. ProjectService	57
3.3.4. SearchService	57
3.3.5. UserService	58
ROZDZIAŁ 4. Dokumentacja w sieci	59
4.1. Społeczność	60
4.2. Biblioteki Javy dla produktów Atlassian	60
4.3. Ogólne zagadnienia z programowania w Javie i Groovym	60
4.4. Dokumentacja REST API	60
4.5. Dokumentacja pluginów	61
4.6. Pozostałe	61
ROZDZIAŁ 5. Testowanie zmian	62
5.1. Logi	62
5.1.1. Application logs (logi aplikacyjne)	62
5.1.2. System logs (logi systemowe)	64
5.2. Loggers (loggery)	65
5.3. Audit log (analizator dzienników)	66
5.4. Support zip (plik dla działu wsparcia)	67
5.5. Testowanie uprawnień użytkowników	68
5.5.1. Permission helper	69
5.5.2. Switch to a different user	69
5.6. Sposoby testowania skryptów	70
CZĘŚĆ III. TWORZENIE SKRYPTÓW ZA POMOCĄ PLUGINÓW	73
ROZDZIAŁ 6. Wybór pluginów	75
6.1. Określenie aktualnych i przyszłych potrzeb	75
6.2. Porównanie najpopularniejszych pluginów	76
6.2.1. ScriptRunner	76
6.2.2. Jira Misc Workflow Extensions (JMWE)	77
6.2.3. JSU Automation Suite for Jira Workflows	78
6.3. Decyzja i jej konsekwencje	78
6.4. Utrzymanie rozwiązań w pluginach	79

ROZDZIAŁ 7.	Przegląd sposobów automatyzacji	80
7.1.	Post functions (post funkcje)	80
7.2.	Conditions (warunki)	82
7.3.	Validators (walidatory)	85
7.4.	Operacje cykliczne	87
7.4.1.	ScriptRunner	88
7.4.2.	JMWE	89
7.5.	Skrypty wyzwalane przez listenery	90
7.5.1.	ScriptRunner	90
7.5.2.	JMWE	91
7.6.	Behaviours	92
7.7.	Fragments (fragmenty)	94
7.8.	REST Endpoint	95
7.9.	Aktualizacja system fields i custom fields	96
7.10.	Wykonywanie przejść w workflows	96
7.11.	Integracja Jiry i Confluence	97
7.12.	Integracja Jiry z Insight - Asset Management	99

CZĘŚĆ IV. TWORZENIE PLUGINÓW ZA POMOCĄ ATLASSIAN SDK 101

ROZDZIAŁ 8.	Instalacja Atlassian SDK	103
8.1.	Środowisko Windows	103
8.1.1.	Weryfikacja JDK	104
8.1.2.	Instalacja JDK	104
8.1.3.	Instalacja Atlassian SDK	107
8.2.	Środowisko Linux	107
8.2.1.	Instalacja JDK	108
8.2.2.	Instalacja Atlassian SDK	109
8.3.	Praca w Eclipse	110
8.3.1.	Konfiguracja projektu w Eclipse	110
8.3.2.	Konfiguracja projektu w IntelliJ IDEA	112
ROZDZIAŁ 9.	Generowanie szkieletu projektu	114
9.1.	Tworzenie szkieletu pluginu	116
9.2.	Tworzenie modułów pluginu	117
9.3.	Budowanie paczek za pomocą Mavena i ich instalacja	118
ROZDZIAŁ 10.	Przykłady własnych pluginów i modułów	120
10.1.	Uwagi techniczne	120
10.1.1.	Plik pom.xml	120
10.1.2.	Plik konfiguracyjny atlassian-plugin.xml	121
10.1.3.	Zmiana logo pluginu	122
10.1.4.	Wstrzykiwanie zależności	122
10.2.	Dodatkowy element w menu głównym Jiry	123

10.3. Post functions (post funkcje)	126
10.3.1. Wymagania biznesowe	128
10.3.2. Utworzenie klasy obsługującej skrypt post funkcji	128
10.3.3. Utworzenie serwisu obsługującego parametry post funkcji	130
10.3.4. Przygotowanie szablonów Velocity	132
10.4. Conditions (warunki przejścia)	135
10.4.1. Wymagania biznesowe	136
10.4.2. Utworzenie klasy obsługującej skrypt warunku przejścia	136
10.4.3. Utworzenie serwisu obsługującego parametry warunku przejścia	137
10.4.4. Przygotowanie szablonów Velocity	139
10.5. Validators (walidatory)	140
10.5.1. Wymagania biznesowe	141
10.5.2. Utworzenie klasy obsługującej skrypt walidatora	142
10.5.3. Utworzenie serwisu obsługującego parametry walidatora	143
10.5.4. Przygotowanie szablonów Velocity	145
10.6. REST Endpoint	147
10.6.1. Wymagania biznesowe	148
10.6.2. Utworzenie modelu danych JSON	148
10.6.3. Utworzenie kontrolera dla REST Endpointu	150

CZĘŚĆ V. SYSTEM FIELDS I CUSTOM FIELDS 153

ROZDZIAŁ 11. Praca z polami	155
11.1. Sposób wykorzystania system fieldów i custom fieldów	155
11.2. Dobre praktyki stosowania custom fieldów	156
11.2.1. Nazewnictwo custom fieldów	157
11.2.2. Stosowanie kontekstów w custom fieldach	158
11.2.3. Używanie identyfikatorów custom fieldów zamiast nazw	160
11.2.4. Dobór odpowiedniego typu custom fieldu	160
ROZDZIAŁ 12. Dane z system fieldów i custom fieldów	164
12.1. System fields	164
12.1.1. Warunki początkowe	164
12.1.2. Pobieranie wartości	164
12.2. Text fields i number fields	165
12.2.1. Warunki początkowe	165
12.2.2. Pobieranie wartości	166
12.2.3. Przetwarzanie wartości	167
12.3. Select list (single choice)	168
12.3.1. Warunki początkowe	168
12.3.2. Pobieranie wartości	169
12.3.3. Przetwarzanie wartości	169
12.3.4. Dodawanie, usuwanie i dezaktywacja opcji	171

12.4. Select list (multiple choices)	177
12.4.1. Warunki początkowe	177
12.4.2. Pobieranie wartości	178
12.4.3. Przetwarzanie wartości	179
12.4.4. Dodawanie, usuwanie i dezaktywacja opcji	180
12.5. Checkboxes, radio buttons	181
12.6. Date pickers, date time pickers	181
12.6.1. Warunki początkowe	181
12.6.2. Pobieranie wartości	181
12.6.3. Przetwarzanie wartości	183
12.7. User pickers i group pickers	190
12.7.1. Warunki początkowe	190
12.7.2. Pobieranie wartości	191
12.7.3. Przetwarzanie wartości	193
12.8. Scripted fields (pola skryptowe)	196
12.8.1. Definiowanie scripted fieldów	197
12.8.2. Text field (multi-line)	198
12.8.3. Data, data time	202
12.8.4. Number fields	205
12.8.5. User picker, group picker	208

CZĘŚĆ VI. OPERACJE NA ZGŁOSZENIACH I PROJEKTACH 211

ROZDZIAŁ 13. Aktualizacja i tworzenie zgłoszeń	213
13.1. Aktualizacja tasków i sub-tasków	213
13.1.1. Warunki początkowe	214
13.1.2. Aktualizacja system fieldów	215
13.1.3. Aktualizacja pojedynczego custom fieldu w zgłoszeniu	217
13.1.4. Aktualizacja wielu custom fieldów i system fieldów w zgłoszeniu	219
13.2. Tworzenie tasków (samodzielnych zgłoszeń)	222
13.2.1. Tworzenie zgłoszenia z podstawowymi parametrami	223
13.2.2. Tworzenie zgłoszenia z ustawieniem wartości wybranych custom fieldów	226
13.3. Tworzenie sub-tasków (podzadań)	231
13.3.1. Warunki początkowe	232
13.3.2. Wymagania biznesowe	232
13.3.3. Utworzenie skryptu realizującego wymagania biznesowe	232
13.4. Walidacja zgłoszeń	236
13.4.1. Warunki początkowe	236
13.4.2. Weryfikacja błędów w trakcie walidacji	236
13.5. Operacje na załącznikach (attachments)	241
13.5.1. Warunki początkowe	241
13.5.2. Pobieranie załączników ze zgłoszenia	242
13.5.3. Tworzenie załączników w zgłoszeniach	243

ROZDZIAŁ 14.	Linked issues (zgłoszenia powiązane)	245
14.1.	Rodzaje linked issues	245
14.2.	Pobieranie danych	247
14.2.1.	Warunki początkowe	247
14.2.2.	Pobranie danych z linków zgłoszenia	247
14.2.3.	Przetwarzanie danych	249
14.3.	Tworzenie linked issues	253
14.3.1.	Warunki początkowe	253
14.3.2.	Wymagania biznesowe	253
14.3.3.	Realizacja wymagań biznesowych	253
14.4.	Powiązania issues in epic	255
14.5.	Struktury hierarchiczne powyżej trzech poziomów	256
ROZDZIAŁ 15.	Praca z danymi projektu	257
15.1.	Components (komponenty)	257
15.1.1.	Warunki początkowe	257
15.1.2.	Pobieranie danych	257
15.1.3.	Tworzenie i edycja komponentów	258
15.1.4.	Dodawanie zgłoszeń do komponentów	261
15.2.	Versions (wersje)	262
15.2.1.	Warunki początkowe	262
15.2.2.	Pobieranie danych	262
15.2.3.	Tworzenie i edycja wersji	263
15.2.4.	Dodawanie zgłoszeń do wersji	265
15.3.	Project category (kategoria projektu)	266
15.3.1.	Warunki początkowe	266
15.3.2.	Dodanie projektu do kategorii	266
15.3.3.	Utworzenie i edycja kategorii projektu	267
15.4.	Projekty	269
15.4.1.	Warunki początkowe	269
15.4.2.	Tworzenie projektu z domyślną konfiguracją	269
15.4.3.	Klonowanie projektów	270
15.4.4.	Pobieranie danych i edycja projektu	273
ROZDZIAŁ 16.	Automatyzacja przejść w workflow	275
16.1.	Zmienna transientVars	276
16.2.	Wykonanie prostego przejścia	279
16.2.1.	Wykonanie przejścia za pomocą WorkflowTransitionUtil	279
16.2.2.	Wykonanie przejścia za pomocą IssueService	280
16.3.	Złożone przejścia	280
16.3.1.	Wykonanie złożonego przejścia za pomocą WorkflowTransitionUtil	281
16.3.2.	Wykonanie złożonego przejścia za pomocą komponentu IssueService	282

16.4. Ignorowanie walidatorów i warunków	283
16.4.1. Ignorowanie warunków w przejściach (WorkflowTransitionUtil)	283
16.4.2. Ignorowanie warunków w przejściach (IssueService) ...	284
16.5. Wykorzystanie danych z transientVars	285
ROZDZIAŁ 17. ScriptRunner Behaviours	286
17.1. Pobieranie i ustawianie wartości pól formularza	286
17.2. Ustawianie wartości różnych typów pól	287
17.2.1. Select lists, radio buttons, checkboxes	288
17.2.2. Pickers (pikery)	288
17.2.3. Date i date time pickers	289
17.3. Pozostałe operacje z wykorzystaniem Behaviours	291
CZĘŚĆ VII. KOMUNIKACJA PRZEZ REST API	293
ROZDZIAŁ 18. Charakterystyka Jira REST API	295
18.1. URI zapytania	295
18.2. Uwierzytelnienie i autoryzacja	296
18.2.1. Basic authentication	296
18.2.2. OAuth	297
18.3. Dokumentacja Jira REST API	299
18.4. Wykorzystanie Jira REST API w projektach biznesowych	300
ROZDZIAŁ 19. Operacje przez Jira REST API	302
19.1. Pobieranie danych przez Jira REST API	304
19.1.1. Plik CSV z listą zgłoszeń	304
19.1.2. Zestawienie custom fieldów, które nie występują na screenach	305
19.1.3. Pobranie użytkowników z danej grupy	306
19.1.4. Zwrocenie listy zarchiwizowanych projektów	306
19.2. Tworzenie obiektów w Jirze	307
19.2.1. Tworzenie zgłoszeń z podstawowymi danymi	307
19.2.2. Tworzenie zgłoszeń z wartościami custom fieldów	308
19.2.3. Tworzenie projektów	310
19.3. Edycja zgłoszeń w Jirze	311
ROZDZIAŁ 20. Własne REST API	313
20.1. Utworzenie REST Endpoint	313
20.2. Strony WWW oparte na REST Endpointach	315
20.3. Wystawienie danych w postaci obiektu JSON	316
20.4. Przetworzenie żądania POST	318
ROZDZIAŁ 21. Komunikacja z Jira REST API przez aplikacje zewnętrzne	320
21.1. Aplikacje Javy	320
21.2. JavaScript	323
21.3. PHP	325

21.4. Python	327
21.5. Excel/Power Query	328
21.5.1. Visual Basic for Application (VBA)	329
21.5.2. Power Query	331

CZĘŚĆ VIII. INTERAKCJA JIRY Z CONFLUENCE 333

ROZDZIAŁ 22. Operacje na przestrzeniach i stronach	335
22.1. Połączenie instancji Jiry i Confluence	335
22.2. Pobieranie zawartości strony	336
22.3. Tworzenie przestrzeni	337
22.4. Aktualizacja stron w przestrzeni	339
22.4.1. Utworzenie przestrzeni z modyfikacją strony głównej	339
22.4.2. Aktualizacja wybranej strony Confluence	341
22.5. Tworzenie stron i przestrzeni	342
22.6. Tworzenie podstron	343
22.7. Dodanie etykiet (labels) do stron i podstron	344
22.8. Tworzenie stron z szablonu	346
ROZDZIAŁ 23. Zarządzanie uprawnieniami do przestrzeni	348

CZĘŚĆ IX. INTERAKCJA Z INSIGHT - ASSET MANAGEMENT 351

ROZDZIAŁ 24. Wprowadzenie do baz Insight - Asset Management	353
24.1. Tworzenie schematów obiektów (object schemas)	353
24.1.1. Ustawienia statusów	355
24.1.2. Ustawienia ról	356
24.2. Tworzenie typów obiektów	357
24.3. Object attributes (atrybuty obiektów)	358
24.4. Atrybuty referencyjne	359
ROZDZIAŁ 25. Pola Insighta w Jirze	361
25.1. Typy pól Insighta w Jirze	362
25.1.1. Insight object/s	362
25.1.2. Insight referenced object/s	364
25.1.3. Pozostałe typy custom fieldów	366
25.2. Obiekty Insighta w Javie/Groovym	366
25.2.1. Poziom pierwszy — ObjectBean	367
25.2.2. Poziom drugi — ObjectAttributeBean	368
25.2.3. Poziom trzeci — ObjectAttributeValueBean	369
25.3. Pobieranie danych z obiektów Insighta	370
25.3.1. Warunki początkowe	370
25.3.2. Pobranie atrybutów typu default z obiektów Insighta	371
25.3.3. Pobieranie atrybutów typu object	372

25.4. Aktualizacja wartości custom fieldów typu Insight object/s	373
25.4.1. Obiekty Insighta pobrane przez identyfikator	374
25.4.2. Obiekty Insighta wybrane przez zapytanie IQL	375
25.5. Operacje za pomocą ScriptRunner Behaviours	376
ROZDZIAŁ 26. Operacje na schematach obiektów Insighta	377
26.1. Zmiana wartości atrybutu typu default	377
26.2. Zmiana wartości atrybutu typu object	379
26.3. Tworzenie obiektów Insighta	381
26.4. Usuwanie obiektów z bazy Insight	384

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 Tomcat¹. 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 wchodzi 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 Atlassian³.

- *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.

```

Snowflake
Jira 8.20 (PostgreSQL Ubuntu 22.04)

Terminal
nCounter:{"1":0,"10":0,"100":0,"1000":0,"10000":0},"optimizeMillis":{"count":0,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{"count":0,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{}}},"commitMillis":{"count":1,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{}}}, index writer version cache stats: {"put":0,"get":0,"getFound":0,"clear":1,"clearSizeExceeded":0}, index writer searcher stats: {"createNew":0,"reuse":0}
2022-07-14 14:30:34,912+0200 index-writer-stats-WORKLOG-4-0 INFO [c.a.jira.index.writerWithStats] [JIRA-STATS] [index-writer-stats] WORKLOG : snapshot stats: {"addDocumentsMillis":{"count":0,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{}},"deleteDocumentsMillis":{"count":0,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{}},"updateDocumentsMillis":{"count":0,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{}},"updateDocumentConditionallyMillis":{"count":0,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{}},"updateDocumentsWithVersionMillis":{"count":0,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{"1":0,"10":0,"100":0,"1000":0,"10000":0,"30000":0,"60000":0}},"updateDocumentsWithVersionSize":{"count":0,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{"1":0,"10":0,"100":0,"1000":0,"10000":0}},"replaceDocumentsWithVersionMillis":{"count":0,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{"1":0,"10":0,"100":0,"1000":0,"10000":0}},"replaceDocumentsWithVersionSize":{"count":0,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{"1":0,"10":0,"100":0,"1000":0,"10000":0}},"optimizeMillis":{"count":0,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{"1":0,"10":0,"100":0,"1000":0,"10000":0}},"closeMillis":{"count":0,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{"1":0,"10":0,"100":0,"1000":0,"10000":0}},"commitMillis":{"count":0,"min":0,"max":0,"sum":0,"avg":0,"distributionCounter":{}}}, index writer version cache stats: {"put":0,"get":0,"getFound":0,"clear":0,"clearSizeExceeded":0}, index writer searcher stats: {"createNew":0,"reuse":0}
2022-07-14 14:31:06,339+0200 http-nio-8080-exec-9 ERROR kwasow 871x339v1 ihdrulp 172.16.117.1 /rest/scriptrunner/latest/user/exec/ [c.o.s.r.rest.common.UserScriptEndpoint] *****
2022-07-14 14:31:06,339+0200 http-nio-8080-exec-9 ERROR kwasow 871x339v1 ihdrulp 172.16.117.1 /rest/scriptrunner/latest/user/exec/ [c.o.s.r.rest.common.UserScriptEndpoint] Script console script failed:
java.lang.NullPointerException: Cannot invoke method getName() on null object
    at Script15.run(Script15.groovy:8)

```

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

```

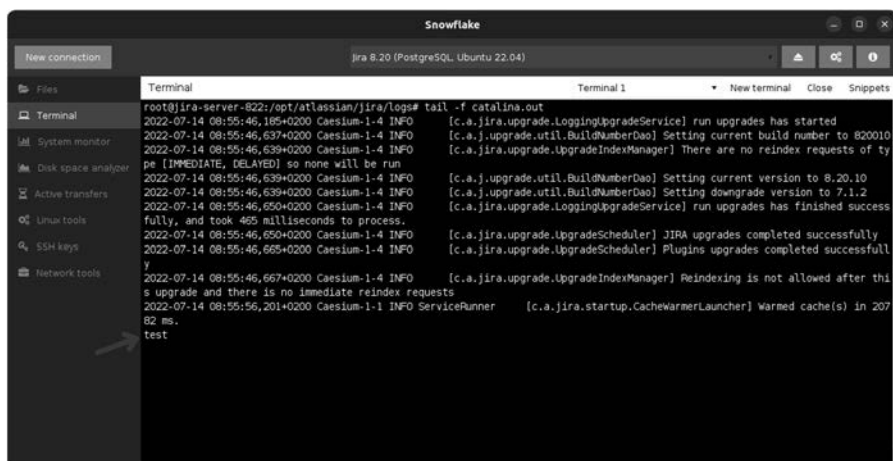
import com.atlassian.jira.bc.user.UserService;
import com.atlassian.jira.component.ComponentAccessor;
// 1. Wstrzyknięcie komponentu UserService.
def userService = ComponentAccessor.getComponent(UserService);
// 2. Próba pobrania obiektu typu interfejsowego ApplicationUser nieistniejącego użytkownika.
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 Tomcat⁶. 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 Atlassian⁸.

- *catalina.out* — logi serwera Tomcat.
- *access_log.<data>* — rejestracja dostępu 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*.

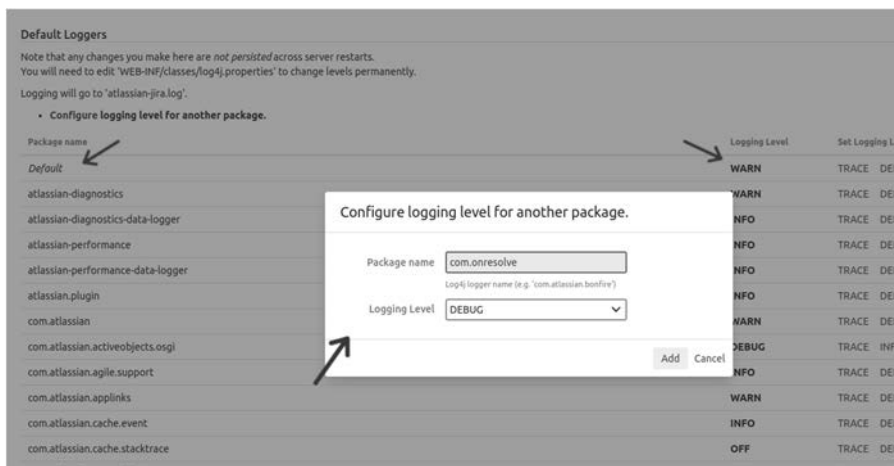
TABELA 5.1. Zasięg poziomów logowania w Jirze

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/<rozwińnięcie 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 ScriptRunner¹¹

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 wyszukiwanie 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.

Date	Author	Category	Summary	Affected object(s)
14 Jul 2022, 19:56:17 CEST		Auditing	Audit Log search performed	
14 Jul 2022, 19:56:09 CEST		Auditing	Audit Log search performed	
14 Jul 2022, 19:54:09 CEST		Auditing	Audit Log search performed	
14 Jul 2022, 19:53:57 CEST		Auditing	Audit Log search performed	
14 Jul 2022, 19:52:54 CEST		Auditing	Audit Log search performed	
14 Jul 2022, 19:52:55 CEST		Auditing	Audit Log search performed	
14 Jul 2022, 19:41:47 CEST		system	ScriptRunner: Script Console: executed	unspecified
14 Jul 2022, 19:41:28 CEST		system	ScriptRunner: Script Console: executed	unspecified

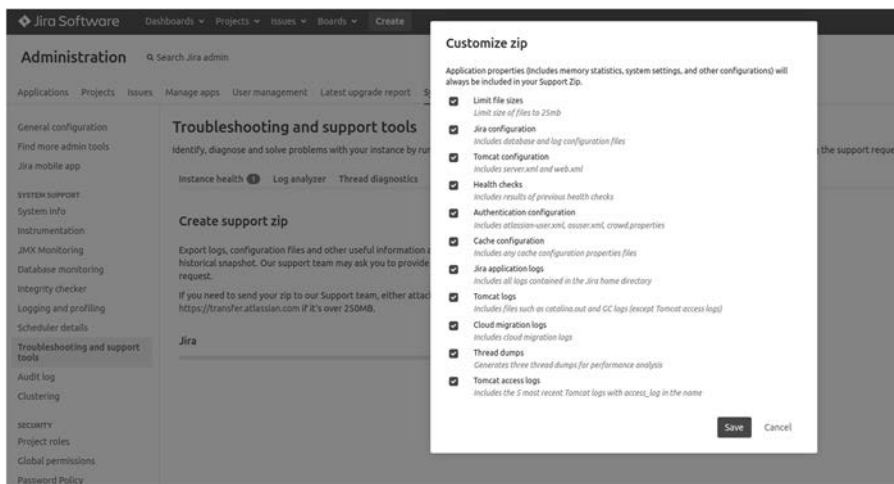
RYSunek 5.4. Widok narzędzia Audit log

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.

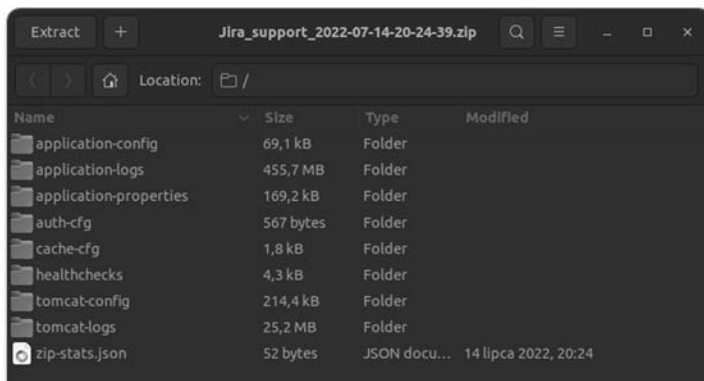
¹² Auditing in Jira, <https://confluence.atlassian.com/adminjiraserver/auditing-in-jira-938847740.html> [dostęp: 14.07.2022].



RYСУNEK 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.

RYСУNEK 5.6.
Zawartość pliku
support zip



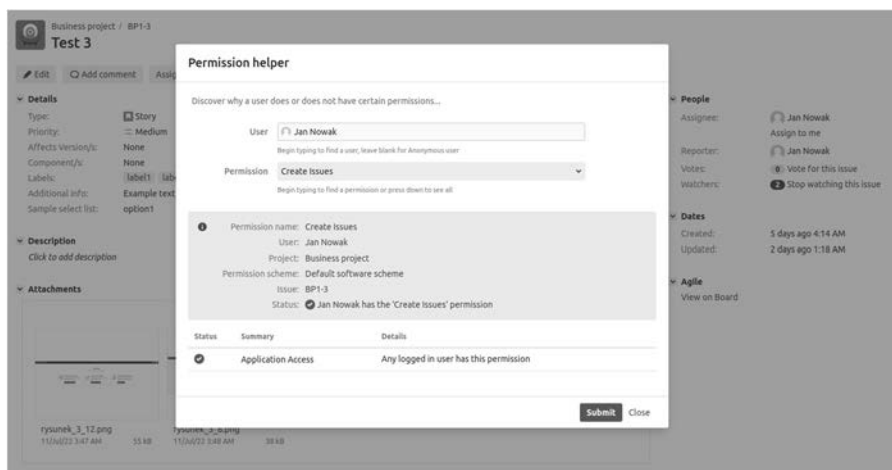
5.5. Testowanie uprawnień użytkowników

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 ScriptRunner¹³ 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.



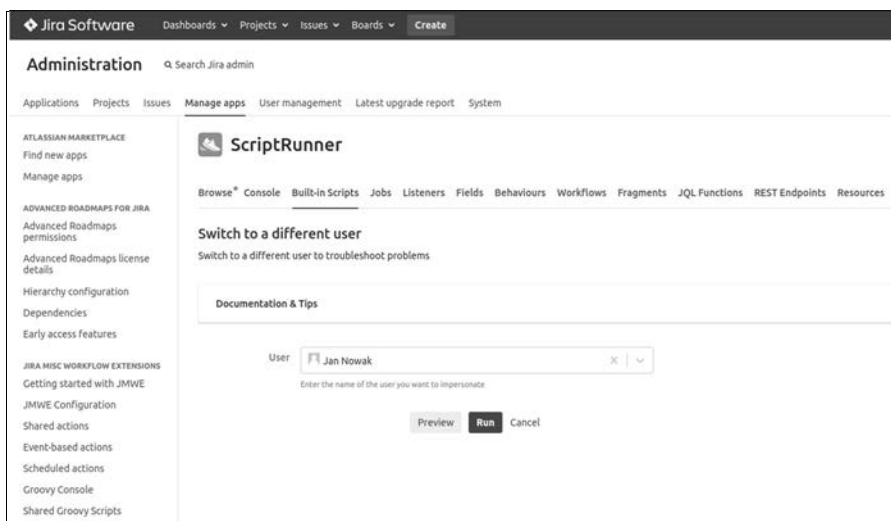
RYСУNEK 5.7. Weryfikacja uprawnień użytkownika za pomocą permission helper

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 ScriptRunner¹⁴. 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].

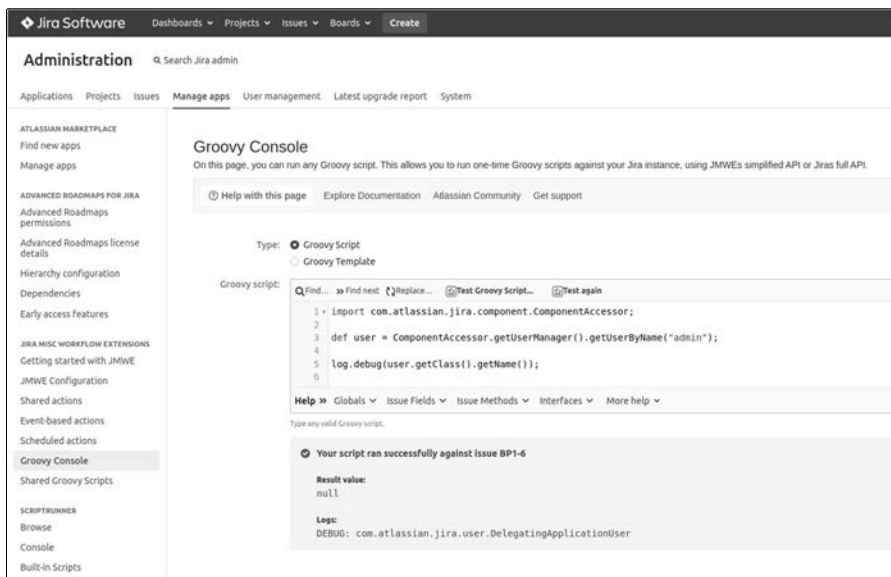


RYСУNEK 5.8. Przełączanie użytkownika za pomocą ScriptRunniera

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.



RYСУNEK 5.9. Sprawdzenie typu obiektu przechowywanego w zmiennej user za pomocą logów w JMWE¹⁵



RYСУNEK 5.10. Sprawdzenie typu obiektu przechowywanego w zmiennej user za pomocą logów w ScriptRunnerze¹⁶

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].

PROGRAM PARTNERSKI

— GRUPY HELION —

- 
1. ZAREJESTRUJ SIĘ
 2. PREZENTUJ KSIĄŻKI
 3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

By szybciej, sprawniej i skuteczniej zarządzać projektami

Znalezienie odpowiedzi na zaprzatające 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 niestety 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.

		KOD KORZYŚCI Sięgnij po więcej! ▶		
	helion.pl			
	HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	ISBN 978-83-283-9784-2		 9 788328 397842
Cena: 69,00 zł				