Język Kotlin na platformie Spring

Programowanie aplikacji internetowych

Miloš Vasić

Helion

Tytuł oryginału: Building Applications with Spring 5 and Kotlin

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-5183-7

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Dodatkowe materiały do książki można znaleźć pod adresem: *ftp://ftp.helion.pl/przyklady/jkotsp.zip*

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

Printed in Poland.

Spis treści

ROZDZIAŁ 2

Pierwsze kroki z platformą Spring

W poprzednim rozdziale skonfigurowałeś środowisko programistyczne. Teraz jesteś gotów do utworzenia swojego pierwszego projektu w języku Kotlin opartego na platformie Spring. Zanim to jednak zrobisz, musisz poznać podstawy tej platformy i dowiedzieć się, co można za jej pomocą uzyskać. Wtedy będziesz mógł wystartować z projektem. Przygotuj się!

W tym rozdziale opisane są następujące zagadnienia:

- czym jest platforma Spring,
- najważniejsze funkcjonalności platformy Spring,
- obsługa języka Kotlin w platformie Spring,
- generowanie projektów,
- tworzenie projektu w środowisku IntelliJ IDEA.

Czym jest platforma Spring?

Spring jest platformą programistyczną opracowaną przez firmę Pivotal. Platforma oferuje funkcjonalności do tworzenia interfejsu użytkownika i modeli konfiguracyjnych nowoczesnych aplikacji. Dzięki tym funkcjonalnościom programiści mogą skupić się na implementowaniu algorytmu aplikacji. Dzisiaj niemal wszystkie nowoczesne aplikacje tworzy się przy użyciu platformy Spring. Jest to jedna z najczęściej stosowanych i najpopularniejszych platform programistycznych.

Jakie funkcjonalności oferuje platforma Spring?

Platforma Spring jest wyposażona w mnóstwo nowoczesnych funkcjonalności niezbędnych każdemu programiście. W tej części rozdziału opisanych zostało kilka najważniejszych.

Wstrzykiwanie zależności

Pierwsza funkcjonalność, o której należy wspomnieć i która jest wykorzystywana w codziennym programowaniu, to wstrzykiwanie zależności (ang. *dependency injection*). Każdy dobry programista stara się tworzyć klasy jak najbardziej uniezależnione od siebie. Są to tzw. klasy luźno sprzężone, czyli „nieświadome" istnienia innych klas. Wstrzykiwanie zależności służy wiązaniu klas ze sobą z zachowaniem niezależności.

Jak to się dzieje?

Wstrzykiwanie zależności to technika nowoczesnego programowania polegająca na udostępnianiu obiektu A obiektowi B. Zależnością jest obiekt A, natomiast obiekt B wykorzystuje wstrzykniętą zależność. Dobrym przykładem jest udostępnianie zależności w testach jednostkowych.

Zależność najczęściej definiujemy, tworząc **instancję zależności** lub **obiekt fabrykujący** generujący instancję. Dzięki wstrzykiwaniu zależności instancja jest przekazywana zewnętrznemu **klientowi**. Sposób wykorzystania instancji to problem zewnętrznego kodu, a nie Twojego! Co to jest ów zewnętrzny kod? Jest nim na przykład:

- obiekt znajdujący się wyżej w hierarchii zależności,
- wstrzykiwarka zależności (platforma) generująca graf zależności.

Poniżej znajduje się kilka przykładów wstrzykiwania zależności. Wykorzystane są w nich konstruktor i obiekt fabrykujący. W obu przypadkach rzecz się komplikuje, jeżeli trzeba dokładnie przetestować kod. Jednak dzięki platformie Spring jest to bardzo łatwe! W tej książce dowiesz się, jak to się robi.

Wstrzykiwanie zależności za pomocą konstruktora wygląda następująco:

```
class MyExampleClass(val parameter: Any){
    private val dependency: Any
    init {
        dependency = parameter
        // Kod wykorzystujący zależność.
    }
}
```

Wstrzykiwanie zależności za pomocą obiektu fabrykującego wygląda następująco:

```
class MyExampleClass2 {
    private val dependency = Factory().create()
}
class Factory {
    fun create(): Any {
```

```
            // Kod tworzący instancję zależności.
            return Any()
        }
    }
```

Jak wspomniałem, dobrym przykładem wstrzykiwania zależności są testy kodu. Dzięki zależnościom testowanie jest o wiele prostsze! Jaki problem pojawia się w powyższych dwóch przykładach? Załóżmy, że chcemy przetestować klasę `MyExampleClass2`. Potrzebna jest jej imitacja (ang. *mock*) lub atrapa (ang. *stub*). Sprawę dodatkowo komplikuje to, że nie mamy wpływu na klasę `Factory`! Prostszym sposobem byłoby wstrzykiwanie konstruktora, ale nie rozwiązałoby to problemu. Dlaczego? Wzajemne uzależnianie tworzonych obiektów jest w praktyce nie do przyjęcia. Tego problemu nie ma w przypadku wstrzykiwania zależności za pomocą setterów. Wkrótce przekonasz się jednak, że takie podejście ma więcej wad niż zalet.

Odwrócenie sterowania

Najważniejszą techniką luźnego sprzęgania klas jest odwrócenie sterowania (ang. *Inversion of Control* — **IoC**). Polega ono na udostępnianiu zależności przez obiekt podrzędny zamiast tworzenia lub szukania obiektów zależnych. Sprzęganie obiektów następuje w trakcie działania kodu. Celem zarówno odwrócenia sterowania, jak i wstrzykiwania zależności jest redukowanie liczby zależności w kodzie.

Przeanalizujmy kilka przykładów ilustrujących powyższą technikę. Wyobraź sobie, że tworzysz aplikację do odtwarzania muzyki i potrzebny jest mechanizm do sterowania głośnością. Początkowy kod wygląda następująco:

```
class VolumeControl
class MusicPlayer {
  val volumeControl = VolumeControl()
}
```

Teraz zastosujmy odwrócenie sterowania:

```
abstract class VolumeControlAbstract
class MusicPlayerIOC(
  private val volumeControl: VolumeControlAbstract
)
```

W pierwszym kodzie tworzona jest instancja klasy:

```
val volumeControl = VolumeControl()
```

Zatem klasa `MusicPlayer` jest bezpośrednio uzależniona od klasy `VolumeControl`!

Jak to wygląda w drugim kodzie? Tworzona jest w nim klasa abstrakcyjna. W sygnaturze konstruktora klasy `MusicPlayer` zdefiniowana jest klasa zależna `VolumeControl`. Nie jest jednak tworzona jej instancja. Zależność jest tworzona, a następnie przekazywana klasie `MusicPlayer` w następujący sposób:

```
// Zainicjowanie zależności.
val vc = VolumeControlImpl()
```

```
// Przekazanie zależności.
val player = MusicPlayerIOC(vc)
```

W tym przypadku kod kliencki tworzący instancję klasy `MusicPlayer` ma kontrolę nad implementacją klasy `VolumeControl`. Jak widać, zależność jest wstrzykiwana w sygnaturze klasy `MusicPlayer`!

Programowanie aspektowe

Platforma Spring oferuje jeszcze jedną ważną funkcjonalność: programowanie aspektowe (ang. *aspect oriented programming* — **AOP**). Programowanie aspektowe stosuje się do tworzenia obiektowych interfejsów API. Jest to nowy styl tworzenia struktury kodu. W programowaniu obiektowym podstawową jednostką jest klasa. Natomiast w programowaniu AOP jest to aspekt. Wstrzykiwanie zależności ma na celu rozprzęganie obiektów tworzących aplikację. Natomiast programowanie AOP pozwala rozdzielać zagadnienia przekrojowe (ang. *cross-cutting concerns*) od obiektów, których te zagadnienia dotyczą. Zagadnienie przekrojowe jest to funkcjonalność wpływająca na wiele obszarów aplikacji. Dobrym przykładem jest zabezpieczanie kodu. Dlaczego? Ponieważ w wielu metodach może być wymagane zaimplementowane reguł bezpieczeństwa.

W programowaniu aspektowym każdy moduł implementuje aspekty. W następnych rozdziałach zastosujesz ten styl programowania w praktyce.

Kontener

Ważną cechą platformy Spring jest możliwość tworzenia obiektów aplikacyjnych i zarządzania ich cyklem życia oraz konfiguracją. Do tego celu służy interfejs `org.springframework.context.ApplicationContext`, który jest odpowiedzialny za tworzenie instancji klas, ich konfigurowanie i kompletowanie. Platforma Spring oferuje kilka gotowych implementacji interfejsu `ApplicationContext`.

Wzorzec MVC

Platforma Spring pozwala tworzyć aplikacje zgodnie ze wzorcem **MVC** (ang. *Model-View-Controller* — model-widok-kontroler). Co to oznacza? We wzorcu MVC również stosuje się interfejsy i różne technologie tworzenia widoków. Wzorzec ten wykorzystuje się do pisania interfejsów użytkownika. Dzięki niemu powstały kod jest prosty i czytelny. Zgodnie z tym wzorcem kod aplikacji dzieli się na trzy części: model, widok i kontroler.

Model

Model to centralna część aplikacji odpowiedzialna za jej działanie. Jest niezależny od widoku i ma bezpośredni wpływ na dane, algorytm i stosowane zasady.

Widok

Widok służy do prezentowania informacji przetwarzanych przez aplikację. Przykładem widoku jest strona HTML tworzona po wysłaniu zapytania do interfejsu API. Aplikacja nie musi mieć jednego widoku. We wzorcu MVC można prezentować tę samą informację w wielu widokach.

Kontroler

Kontroler odbiera dane wejściowe i przetwarza je na polecenia dla modelu lub widoku. Jak widać, jest to kolejny przykład rozprzęgania klas w platformie Spring!

Zarządzanie transakcjami

Platforma Spring oferuje ogólną warstwę abstrakcyjną do zarządzania transakcjami. Zarządzanie to obejmuje nie tylko środowisko J2EE. W odróżnieniu od mechanizmu EJB CMT, powiązanego z interfejsem JTA, deklaratywne zarządzanie transakcjami w platformie Spring można stosować w każdym środowisku. Zarządzać można transakcjami JTA, jak również transakcjami lokalnymi, jeżeli zajdzie taka potrzeba, przy użyciu następujących technik:

- JDBC,
- JPA,
- Hibernate,
- JDO.

Wystarczy jedynie odpowiednio zmienić pliki konfiguracyjne!

Inne funkcjonalności

Wyjątki w warstwie abstrakcji JDBC w platformie Spring tworzą hierarchiczną strukturę, dzięki której kodowanie obsługi błędów jest łatwiejsze.

Platforma Spring umożliwia tworzenie komercyjnych aplikacji za pomocą obiektów **POJO** (ang. *Plain Old Java Object* — zwykły, stary obiekt Java). Co to oznacza? Że nie jest potrzebny kontener EJB, na przykład serwer aplikacji. Programista może użyć nawet podstawowego kontenera serwletów, na przykład Tomcata lub innego komercyjnego produktu. Jednym z możliwych rozwiązań jest wykorzystanie środowiska AWS.

Warto zaznaczyć, że platforma Spring ma modułową budowę. Oznacza to, że w aplikacji można wykorzystywać jedynie niezbędne pakiety, a inne pominąć.

Razem z platformą można stosować inne popularne technologie, na przykład platformę ORM, platformy logujące, środowisko JEE, czasomierze Quartz i JDK.

Jak wygląda testowanie aplikacji? Dzięki platformie Spring testy przeprowadza się szybko i łatwo. Kod uzależniony od otaczającego go środowiska przenosi się do platformy. Za pomocą obiektów POJO można łatwo wstrzykiwać zależności i dane potrzebne do wykonania testów.

Platforma Spring oferuje doskonały interfejs API do przekształcania wyjątków charakterystycznych dla różnych technologii na spójne wyjątki niekontrolowane.

Kontenery IoC w platformie Spring są bardzo małe. Pod tym względem znacznie różnią się od kontenerów EJB i w mniejszym stopniu obciążają pamięć i procesor komputera.

To są najpopularniejsze funkcjonalności platformy Spring. Wykorzystasz je podczas pracy z tą książką. Spotkasz się również z innymi funkcjonalnościami, które nie zostały tu wymienione. Gdy dogłębnie poznasz platformę Spring, na pewno przez długi, bardzo długi czas, a być może już nigdy, nie skorzystasz z innych platform!

Jak jest obsługiwany język Kotlin?

Począwszy od wersji 5 platforma Spring oficjalnie obsługuje język Kotlin. To świetna wiadomość! Jedną z największych zalet Kotlina jest jego doskonała kompatybilność z bibliotekami języka Java. Ale to nie wszystko! Tworząc aplikacje przy użyciu platformy Spring, można w pełni korzystać z zalet składni języka Kotlin. Dzięki temu osiąga się zupełnie inny poziom wydajności i elastyczności kodu! Dlatego począwszy od wersji 5 platformy Spring język Kotlin jest oficjalnie obsługiwany.

Podstawową funkcjonalnością języka Kotlin, umożliwiającą stosowanie go razem z platformą Spring, są rozszerzenia. Dzięki nim można rozbudowywać istniejące klasy platformy, co wprowadza najnowszą wersję platformy w zupełnie nowy wymiar!

Należy podkreślić, że rozszerzenia w języku Kotlin są rozwiązywane statycznie. Oznacza to, że trzeba je importować.

Poniżej wymienione są najważniejsze korzyści płynące ze stosowania platformy Spring i języka Kotlin:

- Platforma Spring może wykorzystywać charakterystyczne dla języka Kotlin zabezpieczenia przed zerowymi wskaźnikami.
- Do kierowania zapytań można wykorzystywać język DSL.
- Dzięki urzeczowionym parametrom typowanym można uniknąć problemów wynikających z wymazywania typów w maszynie JVM.
- Narzędzie Gradle można konfigurować w języku Kotlin.
- Można stosować szablony skryptów w języku Kotlin.
- Stosując platformę Spring z językiem Kotlin, można tworzyć krótszy, prostszy, bardziej efektywny i czytelny kod.

Język Kotlin to przyszłościowe rozwiązanie dla wszystkich programistów korzystających z platformy Spring!

Tworzenie projektu Spring

Przyszedł czas na utworzenie i uruchomienie pierwszego projektu opartego na platformie Spring. Twoim podstawowym językiem programowania będzie oczywiście Kotlin. Do zainicjowania projektu wykorzystasz narzędzie Spring Initializr.

Czym jest narzędzie Spring Initializr?

W skrócie: Spring Initializr jest internetowym generatorem projektów opartych na platformie Spring. Narzędzie to jest dostępne pod adresem *https://start.spring.io*. Ponieważ jest to otwarty projekt, jego kod znajduje się w repozytorium GitHub pod adresem *https://github.com/spring-io/initializr*.

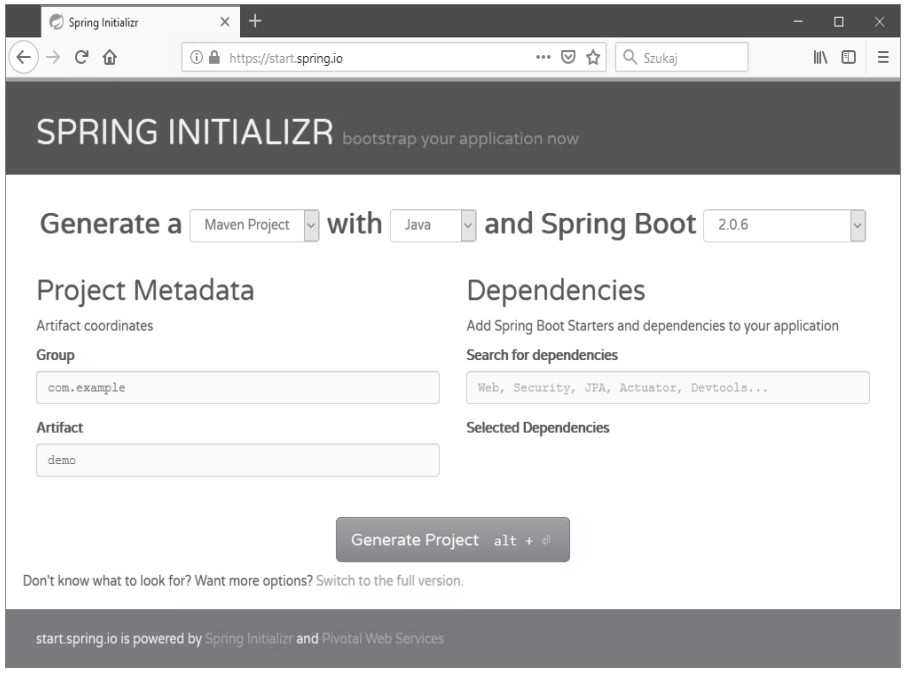
Spring Initializr to narzędzie udostępniające interfejs API do szybkiego generowania projektów opartych na platformie Spring. Możesz użyć jego domyślnej instancji o podanym wyżej adresie lub sklonować jego repozytorium w serwisie GitHub i utworzyć własną instancję.

Narzędzie Spring Initializr ma kilka opcji konfiguracyjnych, za pomocą których definiuje się różne aspekty tworzonego projektu, między innymi:

- listę zależności,
- wersję języka Java,
- wersję języka Kotlin,
- wersję platformy Spring Boot.

Utworzenie projektu

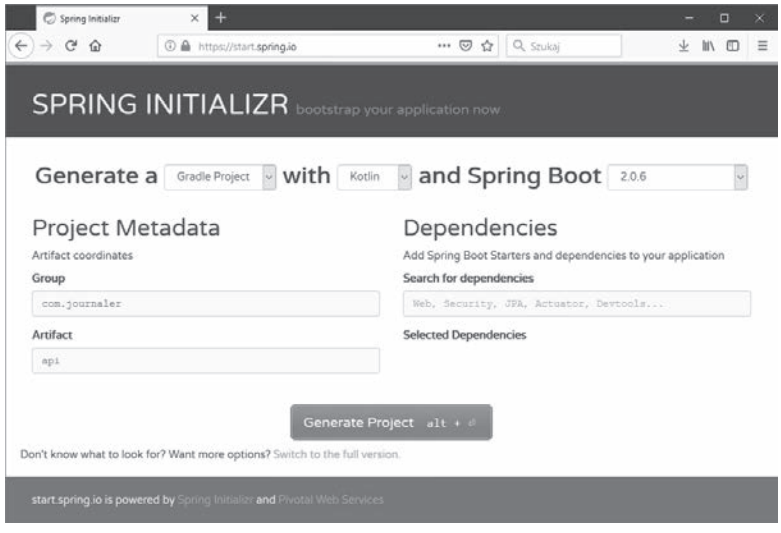
Otwórz stronę narzędzia Spring Initializr: *https://start.spring.io*. Pojawi się kreator projektów, jak na poniższym rysunku:



Wybierz następujące ustawienia projektu:

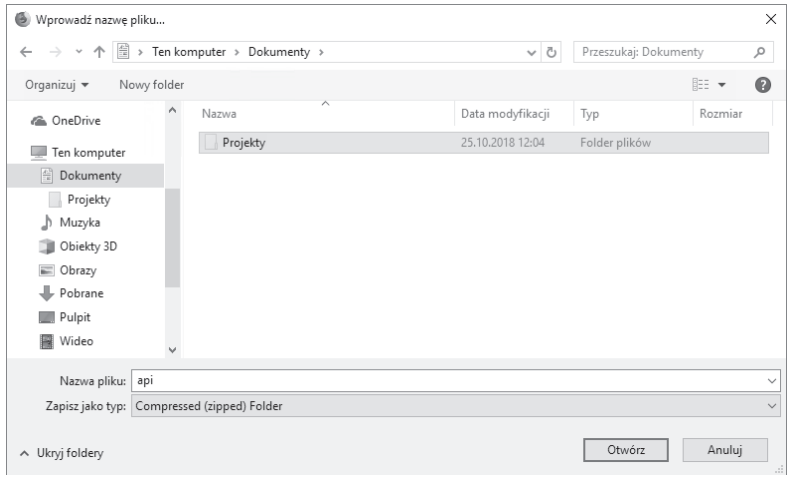
- Wybierz *Generate a Gradle Project with Kotlin and Spring Boot 2.0.6* (wygeneruj projekt Gradle z językiem Kotlin i platformą Spring Boot 2.0.6).
- W polu *Group* wpisz *com.journaler*.
- W polu *Artifact* wpisz *api*.

Poniżej przedstawiony jest widok ustawień:

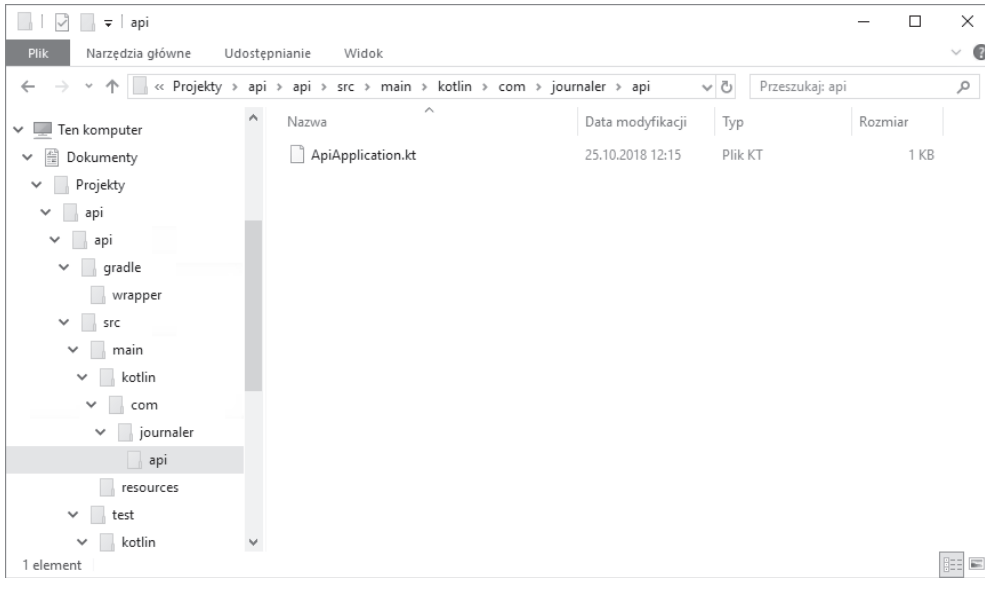


Pierwsza konfiguracja będzie prosta, ale później w miarę swoich postępów będziesz ją rozszerzał o dodatkowe zależności.

Kliknij przycisk *Generate Project* (wygeneruj projekt). Pojawi się okno, jak na poniższym rysunku, z pytaniem o folder, w którym ma być zapisany plik z projektem:



Zapisz plik *api.zip* w preferowanym folderze, a następnie rozpakuj go. Poniższy rysunek przedstawia zawartość projektu:



Jest to standardowy projekt aplikacji w języku Kotlin. Przyjrzyjmy się dokładniej jego najważniejszym częściom. Otwórz plik konfiguracyjny *build.gradle*. Jego zawartość powinna być podobna do tej niżej przedstawionej:

```
buildscript {
  ext {
    kotlinVersion = '1.2.51'
    springBootVersion = '2.0.6.RELEASE'
  }
  repositories {
    mavenCentral()
  }
  dependencies {
    classpath("org.springframework.boot:spring-boot-gradle-plugin:$
    ↪{springBootVersion}")
    classpath("org.jetbrains.kotlin:kotlin-gradle-plugin:${kotlinVersion}")
    classpath("org.jetbrains.kotlin:kotlin-allopen:${kotlinVersion}")
  }
}

apply plugin: 'kotlin'
apply plugin: 'kotlin-spring'
apply plugin: 'eclipse'
apply plugin: 'org.springframework.boot'
apply plugin: 'io.spring.dependency-management'
```

```
group = 'com.journaler'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = 1.8
compileKotlin {
  kotlinOptions {
    freeCompilerArgs = ["-Xjsr305=strict"]
    jvmTarget = "1.8"
  }
}
compileTestKotlin {
  kotlinOptions {
    freeCompilerArgs = ["-Xjsr305=strict"]
    jvmTarget = "1.8"
  }
}

repositories {
  mavenCentral()
}

dependencies {
  implementation('org.springframework.boot:spring-boot-starter')
  implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8")
  implementation("org.jetbrains.kotlin:kotlin-reflect")
  testImplementation('org.springframework.boot:spring-boot-starter-test')
}
```

Kolejne sekcje skryptu, począwszy od samego początku, zawierają następujące ustawienia:

- oznaczenia wersji języka Kotlin i platformy Spring Boot,
- nazwy i ścieżki repozytoriów niezbędnych do skompilowania kodu,
- wtyczki niezbędne do skompilowania i uruchomienia kodu,
- nazwę grupy projektu (`com.journaler`),
- oznaczenie wersji projektu (`0.0.1-SNAPSHOT`),
- wymóg kompatybilności z wersją języka Java 1.8,
- nazwy wykorzystywanych repozytoriów,
- zależności wykorzystywane przez język Kotlin i platformę Spring.

Inną ważną kwestią jest konfiguracja narzędzia Git. Otwórz plik *.gitignore*, zawierający ustawienia wyłączające niepotrzebną kontrolę wersji w przypadku niektórych plików, i przyjrzyj się jego zawartości:

```
.gradle
/build/
!gradle/wrapper/gradle-wrapper.jar

### STS ###
.apt_generated
.classpath
.factorypath
```

```
.project
.settings
.springBeans
.sts4-cache

### IntelliJ IDEA ###
.idea
*.iws
*.iml
*.ipr
/out/

### NetBeans ###
/nbproject/private/
/nbbuild/
/dist/
/nbdist/
/.nb-gradle/
```

Otwórz plik *application.propeties*, który znajduje się w folderze *src\main.resources*. Plik ten zawiera ustawienia aplikacji specyficzne dla wykorzystywanego środowiska. W tej chwili jest on pusty.

I wreszcie: przejrzyj kod! Projekt został utworzony w języku Kotlin, więc przejdź do folderu *src\main\kotlin\com\journaler\api* i otwórz plik *ApiApplication.kt*:

```
package com.journaler.api

import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.runApplication

@SpringBootApplication
class ApiApplication

fun main(args: Array<String>) {
  runApplication<ApiApplication>(*args)
}
```

Kod nie jest długi, więc łatwo go zrozumieć. Zdefiniowana jest w nim klasa aplikacyjna `ApiApplication` opatrzona adnotacją `@SpringBootApplication`. Metoda `main()` uruchamia aplikację wykorzystującą platformę Spring. To wszystko!

Zanim skompilujesz i uruchomisz kod, musisz sprawdzić jedną rzecz. Przejdź do folderu *src\test\kotlin\com\journaler\api* i otwórz plik *ApiApplicationTest.kt*, który zawiera następujący test:

```
package com.journaler.api

import org.junit.Test
import org.junit.runner.RunWith
import org.springframework.boot.test.context.SpringBootTest
import org.springframework.test.context.junit4.SpringRunner
```

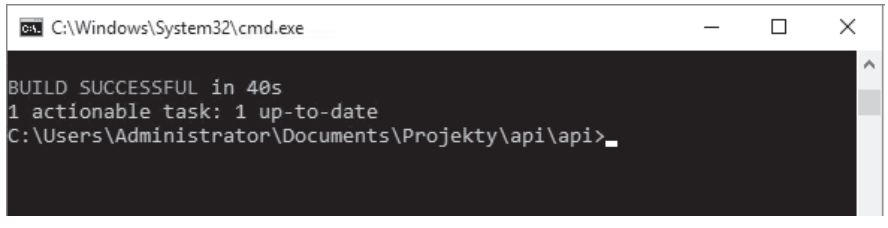
```
@RunWith(SpringRunner::class)
@SpringBootTest
class ApiApplicationTests {

  @Test
  fun contextLoads() {
  }

}
```

Powyższy test nie robi nic. Więcej informacji o testowaniu aplikacji opartych na platformie Spring znajdziesz w następnych rozdziałach książki. Na razie ogólnie zapoznaj się z kodem testu.

Teraz skompiluj i uruchom aplikację. W tym celu otwórz wiersz poleceń, przejdź do głównego folderu projektu i wpisz następujące polecenie:

```
C:\Users\Administrator\Documents\Projekty\api\api>gradlew clean
```

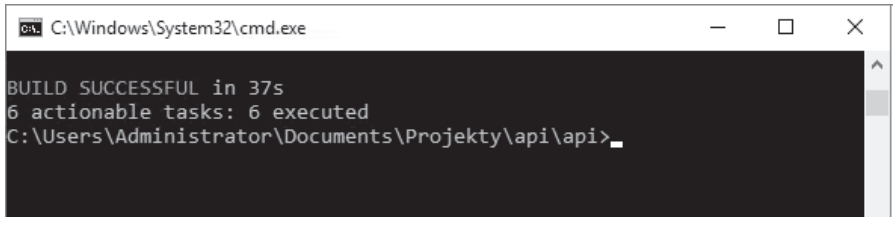
Narzędzie Gradle pobierze wymagane zależności:



Powyższy komunikat oznacza, że zależności zostały pomyślnie pobrane. Aby skompilować kod, wpisz następujące polecenie:

```
C:\Users\Administrator\Documents\Projekty\api\api>gradlew build
```

Jeżeli kompilacja zakończy się pomyślnie, pojawi się taki komunikat:



Aby uruchomić aplikację, przejdź do folderu *build/libs* i wpisz polecenie:

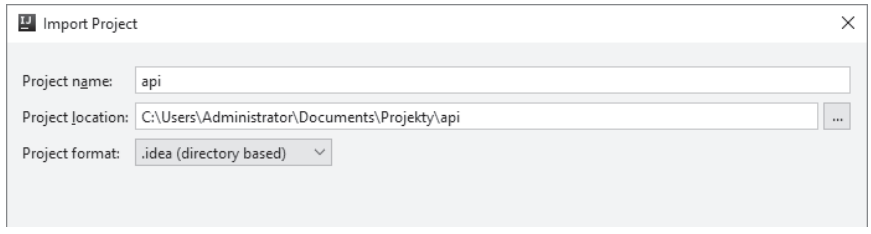
```
C:\Users\Administrator\Documents\Projekty\api\api\build\
↪libs>java -jar ./api-0.0.1-SNAPSHOT
```

Aplikacja wyświetli następującą treść:

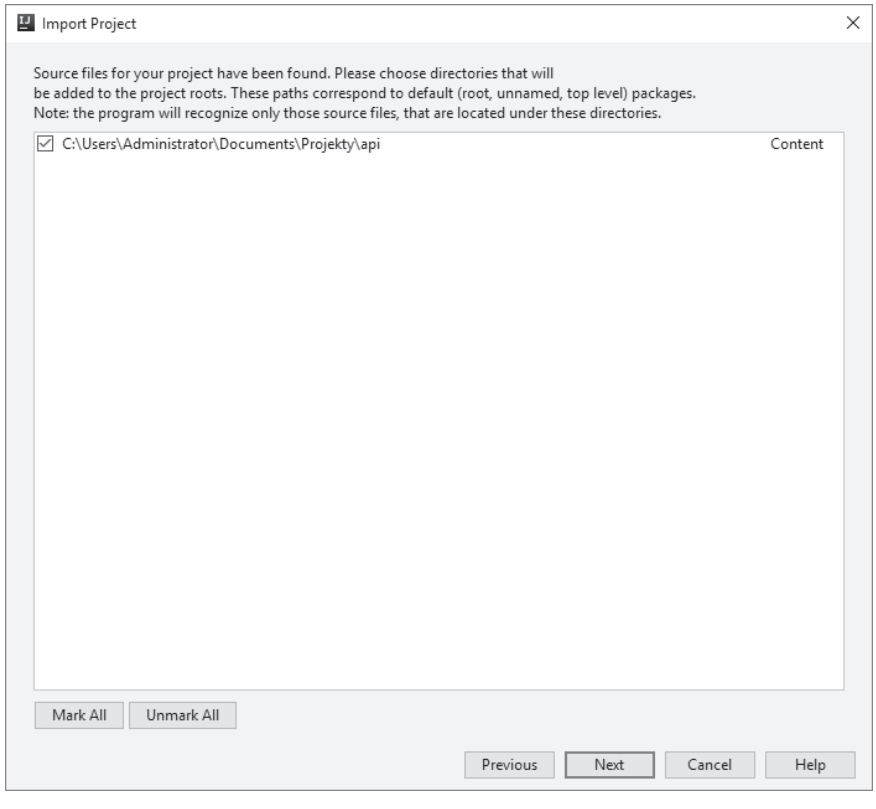
Na razie nie będziemy szczegółowo analizować wyświetlanych komunikatów, ponieważ musisz najpierw przejść długą drogę do wysłania pierwszego zapytania.

Ostatnim zadaniem będzie utworzenie projektu od podstaw w środowisku IntelliJ IDEA. Wcześniej jednak zaimportuj pobrany projekt. W tym celu otwórz środowisko i kliknij opcję *Import Project*. Następnie wskaż folder z rozpakowanymi plikami i kliknij przycisk *OK*. Pojawi się okno kreatora projektu. Zaznacz opcję *Create project from existing sources* (utwórz projekt na podstawie istniejących źródeł), jak na poniższym rysunku:

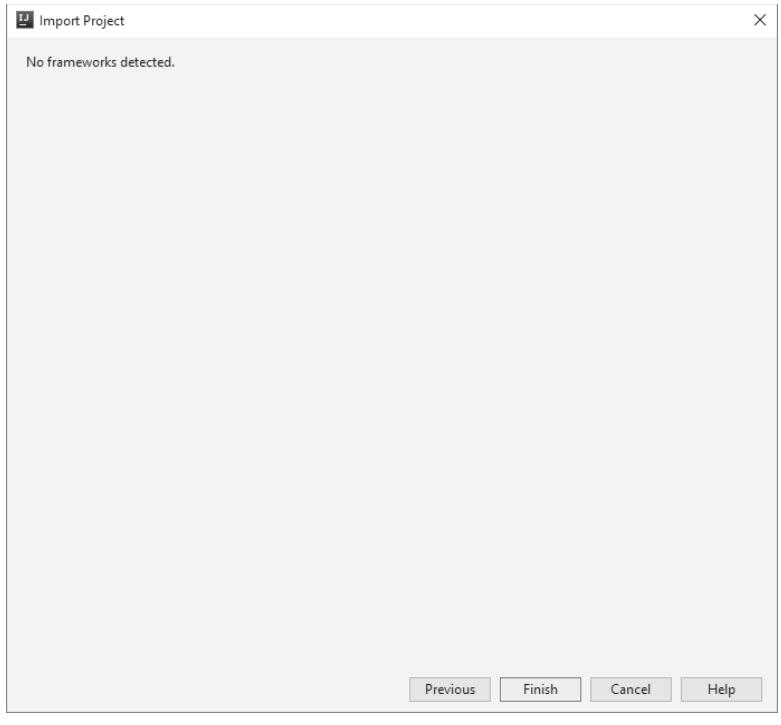
Kliknij przycisk *Next*. Pojawi się kolejne okno z polem *Project name* zawierającym nazwę projektu:



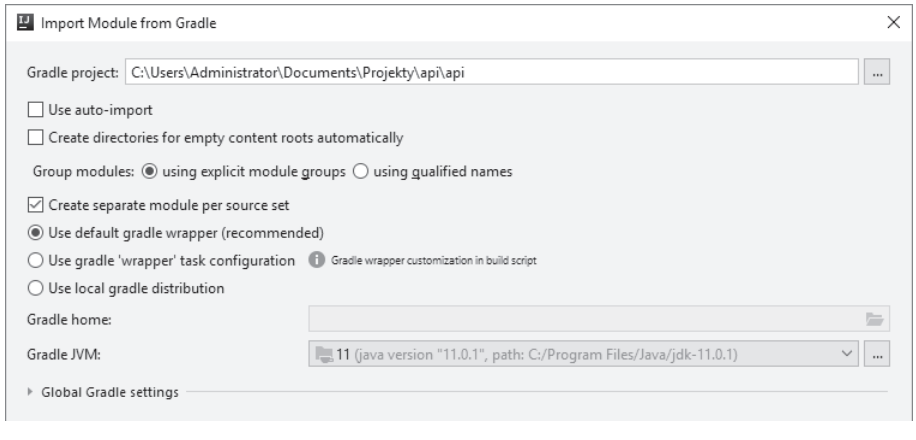
Jeżeli nazwy projektu i folderu Ci odpowiadają, kliknij przycisk *Next*. W następnym oknie kliknij przycisk *Mark All* (zaznacz wszystko):



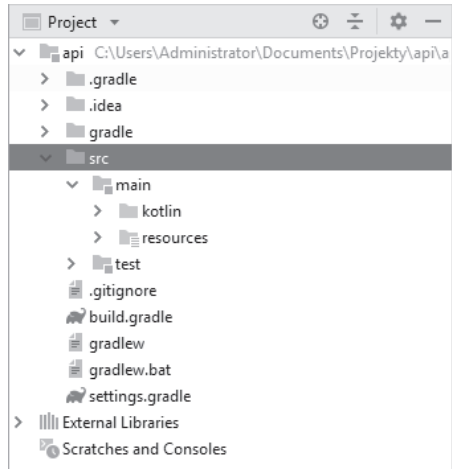
W ostatnim oknie kliknij przycisk *Finish*:



Następnie kliknij polecenie menu *File/New/Module from Existing Sources* (plik/nowy/moduł z istniejących źródeł) i zaimportuj moduł Gradle. W oknie z ustawieniami importu kliknij przycisk *OK*:



Import modułu zajmie chwilę. Rysunek na następnej stronie przedstawia strukturę projektu aplikacji gotowej do uruchomienia:



Jeżeli środowisko programistyczne nie rozpozna automatycznie konfiguracji projektu, rozwiń sekcję *api/src/main/kotlin/com.journaler.api*, kliknij prawym przyciskiem myszy plik *ApiApplication.kt* i wybierz polecenie *Run 'com.journaler.api.ApiApplication.kt'*. Aplikacja powinna wyświetlić takie same komunikaty jak plik JAR, który uruchomiłeś wcześniej.

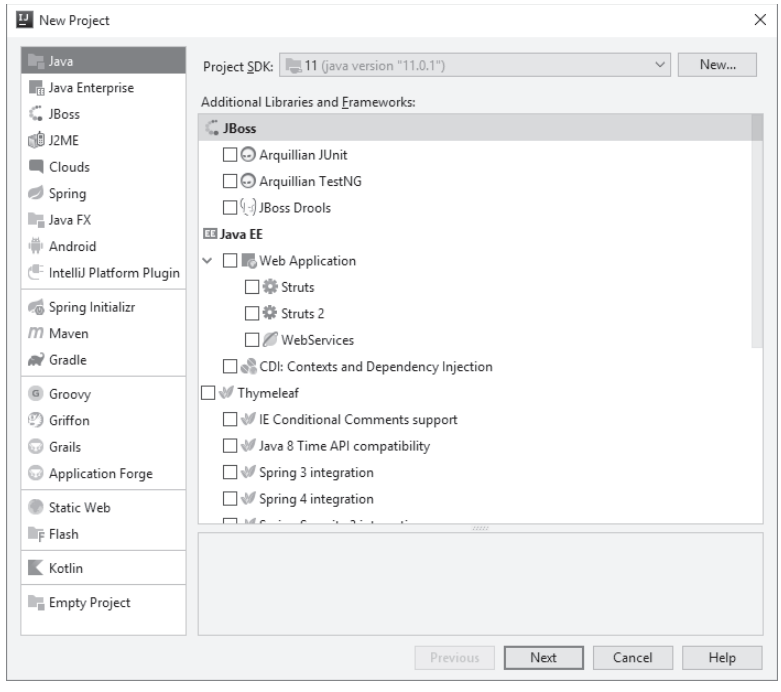
W pliku *.gitignore* wpisz poniższe wiersze (których narzędzie Spring Initializr nie utworzyło) wyłączające kontrolę wersji w przypadku plików narzędzia Gradle:

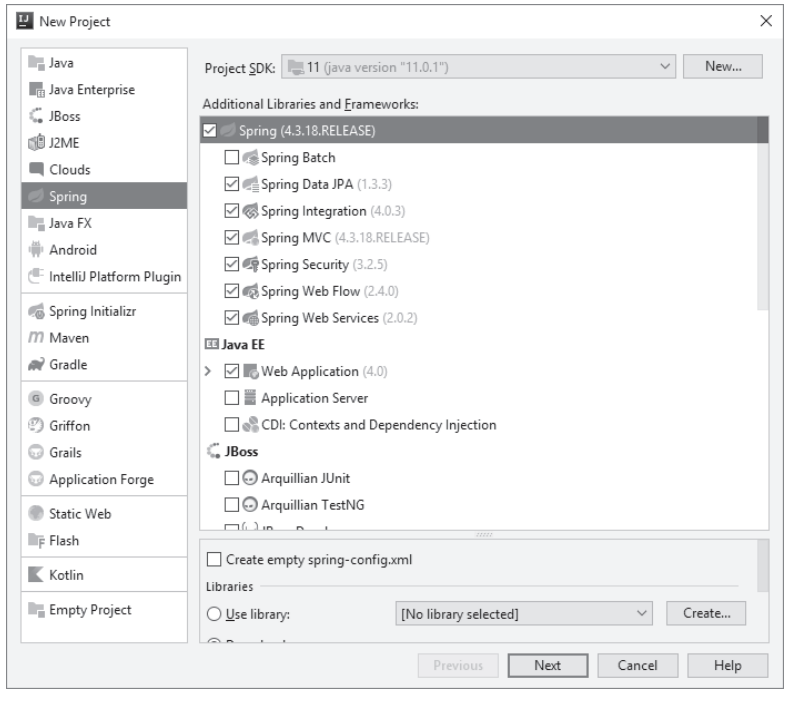
```
gradlew
gradlew.bat
gradle/*
```

Tworzenie projektu w środowisku IntelliJ IDEA

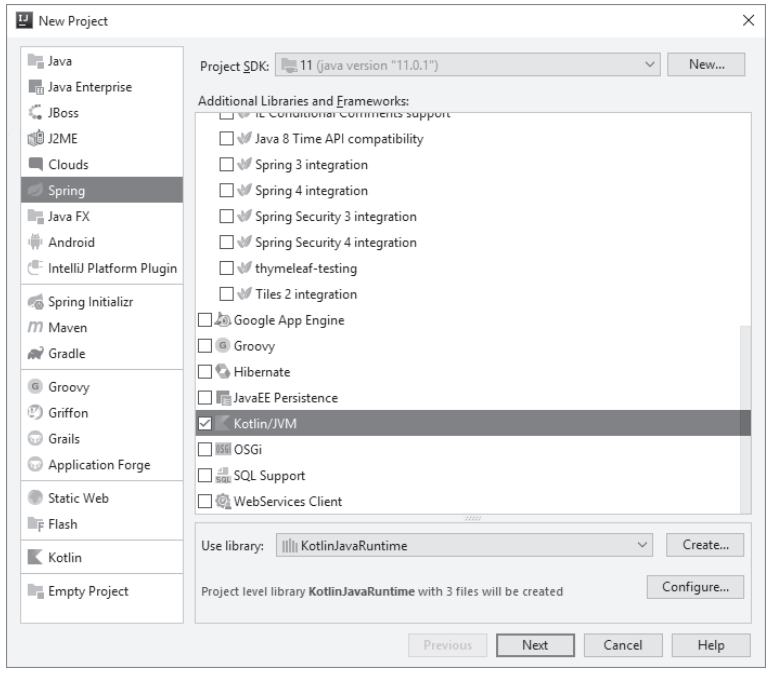
Na zakończenie rozdziału dowiesz się, jak inicjuje się projekty w środowisku IntelliJ IDEA. Efekt będzie taki sam jak ten uzyskany za pomocą narzędzia Spring Initializr. Uruchom środowisko IntelliJ IDEA i kliknij opcję *Create New Project* (utwórz nowy projekt). Pojawi się okno *New Project* (zobacz pierwszy rysunek na następnej stronie).

Kliknij sekcję *Spring* i zaznacz opcje jak na drugim rysunku na następnej stronie.





Ponadto zaznacz opcję *Kotlin/JVM*, jak niżej:

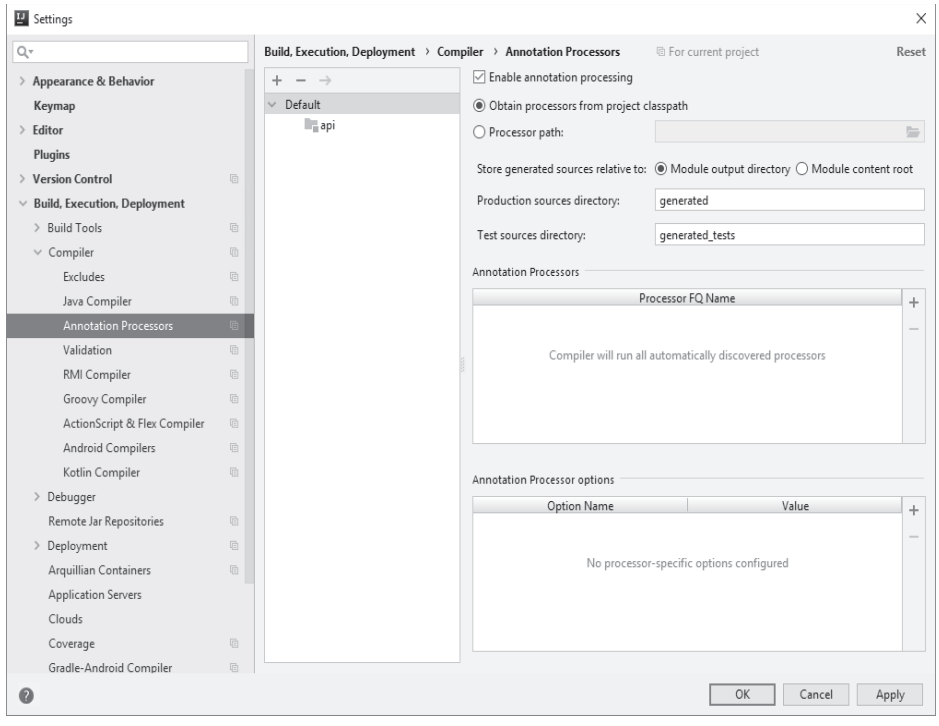


Kliknij przycisk *Next* i dokończ tworzenie projektu zgodnie ze wskazówkami kreatora. Następnie kliknij polecenie menu *File/Settings*. W oknie, które się pojawi, rozwiń po lewej stronie sekcję *Build, Execution, Deployment/Compiler/Annotation Processing* (kompilacja, uruchomienie, wdrożenie/kompilator/przetwarzanie adnotacji) i zaznacz opcję *Enable annotation processing* (włącz przetwarzanie adnotacji) — zobacz rysunek na następnej stronie.

Jak widać na rysunku na następnej stronie, w środowisku brak jest ustawień platformy Spring. Dlatego będziesz tworzył projekty przy użyciu narzędzia Spring Initializr — zawiera ono wszystkie niezbędne opcje.

Podsumowanie

Ten rozdział był wprowadzeniem do platformy Spring i jej funkcjonalności. Utworzyłeś swój pierwszy projekt, pomyślnie go skompilowałeś i uruchomiłeś. W następnym rozdziale będziesz kontynuował rozpoczętą podróż i zajmiesz się bardziej konkretnymi zadaniami: utworzysz swoją pierwszą usługę REST, która będzie się składać z kontrolera, klasy danych i komponentów usługowych.



Skorowidz

J

K

L

M

N

O

P

R

S

Ś

T

U

W

X

Z

Kotlin: tak świetny jak Java, ale o wiele łatwiejszy i przyjemniejszy!

Kotlin jest interoperacyjnym, obiektowym i coraz popularniejszym językiem programowania. Charakteryzuje się statycznie typowanymi zmiennymi, czytelną składnią i znakomitą reaktywnością. Z kolei Spring jest wyjątkowo cenioną platformą do budowania stron WWW. Służy do tworzenia aplikacji internetowych udostępniających interfejs REST i inne usługi. Spring znakomicie współpracuje z Kotlinem, co pozwala programiście na wyższy poziom skutecznego programowania i efektywne tworzenie funkcjonalnych, stabilnych, skalowalnych i reaktywnych aplikacji. Przy tym wszystkim obydwa te produkty w całości są otwartym oprogramowaniem!

Ta książka jest przewodnikiem dla osób, które chcą szybko poznać platformę Spring i zacząć posługiwać się językiem Kotlin w stopniu pozwalającym na tworzenie aplikacji internetowych. Zawiera przystępne wprowadzenie do pracy na platformie Spring i jej konfiguracji dla potrzeb Kotlina; omawia też zasady projektowania aplikacji za pomocą tych narzędzi. Sporo uwagi autor poświęca budowaniu systemu mikrousług udostępniających interfejs REST. Opisuje również techniki posługiwania się tak pożytecznymi narzędziami jak Spring Data, Spring Security czy biblioteka JUnit, a także sposoby wdrażania aplikacji w chmurze AWS.

W książce między innymi:

- przygotowanie platformy Spring do pracy z Kotlinem
- korzystanie z bazy danych MySQL
- usługi REST i programowanie reaktywne
- zabezpieczanie aplikacji za pomocą Spring Security
- dobre praktyki programistyczne, testowanie i wdrażanie aplikacji

Miloš Vasić jest serbskim programistą, autorem książek o programowaniu i entuzjastą otwartego oprogramowania. Ukończył studia na uniwersytecie w Singidunum, ze specjalizacją w dziedzinie grafiki komputerowej i programowania dla Androida. Stara się dzielić swój czas między pisanie kolejnej książki a pracę nad nowym — oczywiście otwartym — projektem.

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 978-83-283-5183-7
9 788328 351837

INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 49,00 zł