Знакомство с Spring Boot

Программирование на Spring Boot начинается с подготовки репозитория и заканчивается запуском приложения. В этом уроке мы пройдем этот путь — вы напишете свое первое работающее приложение на Spring Boot. Начнем с установки Spring Boot. Ее можно выполнить тремя способами:

- С помощью пакетных менеджеров. Это самый удобный способ, если вы умеете пользоваться терминалом
- С помощью сайта start.spring.io. На этом сайте вы можете выбрать нужную конфигурацию и скачать архив с кодом или скопировать исходники прямо с сайта
- С помощью нашего шаблона. По ссылке вы найдете репозиторий, который мы подготовили специально для этого курса. В нем мы разместили начальный код приложения, а еще подключили и настроили разные полезные подсистемы, которые понадобятся для работы почти во всех случаях. Выбрав этот способ установки, вы сэкономите немало времени на настройке

В целях обучения мы пойдем по пути ручной настройки — будем добавлять и разбирать каждый файл в проекте. Начнем с настройки Gradle-проекта. Создадим директорию *spring-example* и инициализируем Gradle-проект внутри нее:

```
# Выполните эти команды в вашей домашней директории
mkdir spring-example
cd spring-example

# Инициализируем проект
gradle init

Select type of project to generate:
  1: basic
  2: application
  3: library
  4: Gradle plugin
Enter selection (default: basic) [1..4]

Select build script DSL:
  1: Kotlin
  2: Groovy
```

```
Enter selection (default: Kotlin) [1..2]

Project name (default: spring-example):

Generate build using new APIs and behavior (some features may change in the next minor rel

> Task :init
To learn more about Gradle by exploring our Samples at https://docs.gradle.org/8.3/samples

BUILD SUCCESSFUL in 14s
2 actionable tasks: 2 executed
```

Далее настроим *build.gradle.kts*. В качестве *group* можно указать любой префикс — например, *io.hexlet*:

```
plugins {
    java
    id("org.springframework.boot") version "3.2.2"
    id("io.spring.dependency-management") version "1.1.3"
}

group = "io.hexlet"
version = "0.0.1-SNAPSHOT"

repositories {
    mavenCentral()
}

dependencies {
    implementation("org.springframework.boot:spring-boot-starter")
  implementation("org.springframework.boot:spring-boot-starter-web")
  implementation("org.springframework.boot:spring-boot-devtools")
    testImplementation("org.springframework.boot:spring-boot-starter-test")
}

tasks.withType<Test> {
    useJUnitPlatform()
}
```

Здесь мы видим три зависимости в виде **starter-пакетов**. Это метапакеты, то есть они содержат код. Их задача — подключить целый набор пакетов, связанных по какому-то

признаку. Рассмотрим несколько примеров:

- В базовый starter-пакет входит не только Spring Boot, но еще и пакет для логирования, пакет для автоконфигурирования и тому подобное
- В пакет *spring-boot-starter-test* входит все необходимое для тестирования
- В пакет *spring-boot-starter-web* входит все нужное для работы с HTTP

В будущих уроках мы еще не раз столкнемся с подобными пакетами, которые команда Spring Boot заботливо подготовила для нас.

Кроме указания зависимостей, Spring Boot поставляется с двумя плагинами Gradle:

- Плагин *dependency-management* упрощает работу с зависимостями. Обратите внимание, что зависимости указаны без версий. Работая с плагином, вы можете просто указать версию самого плагина, а он самостоятельно проставит эту же версию всем пакетам Spring Boot
- Другой плагин добавляет команды необходимые для сборки, тестирования и запуска приложения на Spring Boot

Подробнее с плагинами и командами мы познакомимся позже. На этом базовая настройка закончена, и теперь мы можем добавить код нашего приложения. Создадим соответствующую структуру директорий:

```
.
└── src
    └── main
        └── java
            └── io
                └── hexlet
                    └── spring
                        └── Application.java
```

Теперь добавим код приложения в файл *Application.java*:

```
package io.hexlet.spring;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
```

```java
@SpringBootApplication
@RestController
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @GetMapping("/")
    String home() {
        return "Hello World!";
    }
}
```

В этом коде мы видим такие элементы:

- Метод `main()` — внутри него запускается Spring Boot
- Две аннотации, которые настраивают приложение:
  - `@SpringBootApplication` выполняет автоконфигурацию приложения по установленным зависимостям. Например, стартер *web* добавляет Tomcat и Spring MVC, настраивая Spring Boot на работу в режиме веб-приложения
  - `@RestController` отмечает классы, которые содержат обработку маршрутов
- Обработчик маршрута `/` — он определяется аннотацией `@GetMapping` и маршрутом, переданным в нее. Сам обработчик называется `home()`, но это не принципиально, потому что в Spring Boot нет ограничений на именование. Обработчик возвращает строку, которая вернется как тело HTTP-ответа, что удобно в простых ситуациях

Запустим приложение. Для этого понадобится команда `bootRun`, добавленная плагином *org.springframework.boot*:

```
./gradlew bootRun

> Task :bootRun

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.1.3)
```

```
Здесь будет вывод логов запущенного приложения
<==========---> 80% EXECUTING [10s]
> :bootRun
```

Дальше есть два варианта — можно открыть в браузере *localhost:8080* или выполнить запрос через *curl*. В обоих случаях мы увидим на экране фразу *Hello World!*:

```
curl localhost:8080
Hello World!
```

Чтобы остановить приложение, можно набрать комбинацию клавиш `Ctrl-C`.

Spring Boot DevTools

Во время установки мы добавили в Gradle несколько пакетов, в том числе этот:

```
implementation("org.springframework.boot:spring-boot-devtools")
```

Он включает автоматический рестарт приложения при его изменениях, что значительно упрощает и ускоряет работу. Пакет начинает работать автоматически после установки, но это не всегда срабатывает из-за особенностей работы разных IDE. Если вы столкнулись с проблемой, откройте документацию и выполните инструкции для вашей IDE.

Дополнительные материалы

1. Установка, настройка и запуск

Далее →