Porting to Spring Boot 4 / Spring 7

jim stafford

Fall 2026 v2026-05-10: Built: 2026-05-19 06:17 EST

Table of Contents

Chapter 1. Introduction

This write-up documents key aspects encountered when porting the Ejava course examples from Spring Boot 3.5.x / Spring 6 to Spring Boot 4.x / Spring 7.

Unlike the Spring Boot 2 → 3 migration — which forced a sweeping `javax. to jakarta.` rename across the entire enterprise API surface — the move to Spring Boot 4 has no single breakage. The changes are focused on starter renames, dependency reorganization, HTTP client extraction into dedicated starters, and a handful of API and property renames. Jackson 3 is the most involved change.

The document is organized in three parts:

- **Dependency Changes** — what moved, what was renamed, and what must now be declared explicitly; includes a before/after dependency tree for each affected starter.
- **Details** — per-topic, partially random, deep dives with the error signal that surfaces and the fix.
- **Summary** — a consolidated checklist for the porting effort.

The recommended starting point is to run the OpenRewrite Spring Boot 4 recipe against your codebase, commit the result, then work through the remaining errors using this guide as a map.

1.1. Goals

The student will learn:

- specific changes required to port from Spring Boot 3.x to 4.x using the class examples
- recognize certain error messages and map them to solutions

1.2. Objectives

At the conclusion of this lecture, the student will be able to:

1. update Spring Boot 3.5.x class examples branch to Spring Boot 4.x

Chapter 2. Background

Spring Boot 4.0.0 (with Spring 7) was released in late Nov 2025. I did not physically work with it until 4.0.3 in Feb 2026. My last go around was Apr 2026 with 4.0.6. Unlike the Spring Boot 2 to 3 migration — which forced the `javax. to jakarta.` rename across the entire enterprise API surface — the move to Spring Boot 4 did not have a single sweeping breakage. The changes are more focused on package reorganization within Spring Boot itself, starter renames, and a handful of API and property renames. The examples primarily were impacted by:

- spring-boot-starter renames
- spring-boot-starter refactoring
- dependencies to bring client class modules
- light (relative to SB2⇒SB3) amount of Java package renaming
- Jackson 3

This writeup documents issues encountered — to include the initial signal of error and resolution taken.

Spring provides official Migration Guides for Spring Boot 4 and Spring Framework 7 that should be used as primary references.

Chapter 3. Preparation

The migration guide assumes the project is already on Spring Boot 3.5.x and Java 17+ before starting. The Ejava examples were on Spring Boot 3.5.5 and Java 21 prior to the port — both already passed the prerequisites.

The previous examples version was `6.1.0`, so I updated the root and all sub-modules and references to `6.2.0-SNAPSHOT`.

```
mvn versions:set -DnewVersion=6.2.0-SNAPSHOT -f build -DprocessAllModules ①

mvn versions:set-property -Dproperty=ejava.version -DnewVersion=6.2.0-SNAPSHOT -f
build/ ②

mvn clean install -f build ③

mvn versions:set -DnewVersion=6.2.0-SNAPSHOT -DprocessAllModules ④

mvn versions:update-parent -DparentVersion=6.2.0-SNAPSHOT -DskipResolution=true -N ⑤

mvn versions:set-property -Dproperty=ejava.version -DnewVersion=6.2.0-SNAPSHOT ⑥

mvn clean install ⑦
```

① update pom versions in build modules

② update `ejava.version` property references in build poms

③ install build modules in local repository with new version

④ update `ejava.version` property references in example poms

⑤ update parent of examples

⑥ update pom versions in example modules files

⑦ install example modules files in local repository with new version

Chapter 4. OpenRewrite

The course examples were ported twice during this work. The first pass was a manual error-by-error walkthrough — build, fix, repeat. The second pass used the OpenRewrite Maven Plugin with the Spring Boot 4 migration recipes. The latter approach was my first use of OpenRewrite, and I was pleased with the result. OpenRewite went through the Spring Boot 3 baseline and aggressively made changes all at once. OpenRewrite improved unrelated code blocks (e.g., converted concatenated `String` constants into Java text blocks) and eagerly found property changes that went unnoticed in the first wave. OpenRewrite, however:

- did not fully port the legacy Jackson 2 code — leaving some manual work there to fix some edge cases.
- left a litter of explicit version expressions to delete that ignored the presence of the Spring Boot BOM

This writeup assumes we are taking the OpenRewrite path, which corrects many things very quickly. It does include some error examples from the manual port to provide search strings that can help point to corrections in separate examples.

4.1. Recipe invocation

The Spring Boot 4 migration recipe is supplied by the `rewrite-spring` jar and run through the `rewrite-maven-plugin`. No plugin entry needs to be added to your `pom.xml` — the plugin coordinates are passed on the command line:

```
mvn -U org.openrewrite.maven:rewrite-maven-plugin:run \
  --define rewrite.recipeArtifactCoordinates=org.openrewrite.recipe:rewrite-
spring:RELEASE \
  --define
rewrite.activeRecipes=org.openrewrite.java.spring.boot4.UpgradeSpringBoot_4_0 \
  --define rewrite.exportDatatables=true
```

`recipeArtifactCoordinates`	Adds `org.openrewrite.recipe:rewrite-spring` to the recipe classpath. `RELEASE` resolves to the latest published version (`6.30.x` at the time of writing).
`activeRecipes`	The recipe to run. `org.openrewrite.java.spring.boot4.UpgradeSpringBoot_4_0` is the umbrella Spring Boot 4 migration recipe.
`exportDatatables=true`	Writes a CSV summary of every change the recipe made under `target/rewrite/datatables/⋯`. Useful when reviewing the diff.

Run the command twice

1. First with `-f build` - this updates a few plugin versions and explicitly sets the maven compiler version to 17

    - change the explicit version to at least 21
2. Second with no args - this migrates the class example tree.

Some Modules are Dormant

There are a small number of modules that are not ported to Spring Boot 3 and not referenced in their parent pom. They are not migrated to Spring Boot 4.

After the command completes, run a normal `mvn clean verify`, fix the remaining build errors by hand (this document is the map for that), and commit.

4.2. What the recipe automated

The recipe drove the bulk of the changes documented in the rest of this writeup:

- Spring Boot, Spring, and Spring Security version bumps in `dependencyManagement` / parent POM.
- Deprecated starter renames (e.g., `spring-boot-starter-web` → `-webmvc`, `spring-boot-starter-aop` → `-aspectj`).
- Java class annotations `@MockBean` / `@SpyBean` → `@MockitoBean` / `@MockitoSpyBean`.
- Test auto-configuration annotations that are now required when the relevant `TestRestTemplate` / `WebTestClient` types are injected (e.g., `@AutoConfigureTestRestTemplate`).
- `spring-boot-starter-flyway` insertion when Flyway is in use.
- Property renames in `application.properties` (`spring.data.mongodb.` → `spring.mongodb.`, `spring.datasource.` → `spring.sql.init.`).

Beyond the migration recipe, OpenRewrite also performed unrelated cleanup — concatenated `String` constants converted to text blocks, import ordering normalized. These are convenient but should be reviewed alongside the migration changes when you commit.

4.3. What the recipe did NOT do

The default Spring Boot 4 recipe takes the conservative path on Jackson: it adds the `spring-boot-jackson2` compatibility shim and leaves Jackson 2 imports in place. The full port to Jackson 3 native (`tools.jackson.*` imports, `JsonMapperBuilderCustomizer`, removal of `JavaTimeModule` registration, removal of the `spring-boot-jackson2` dependency) was a manual second pass on top of the recipe output (see Jackson 3).

Likewise the Mongo `actuator hello` command failure (see Mongo Actuator and the `hello` command) and the Spring Security `PathPatternRequestMatcher` substitutions surfaced during the post-recipe build and were resolved by hand.

After the two OpenRewrite passes were complete, the build for the build files completed but the examples failed in common in a Jackson 2/3 area.

Initial post-Rewrite Port Build Error

```
$ mvn clean install -f build
```

```
$ mvn clean install
[ERROR] /Users/jim/proj/ejava-javaee/boot4-port/common/ejava-dto-
util/src/main/java/info/ejava/examples/common/dto/JsonUtil.java:[4,40] cannot find
symbol
  symbol:   class DateDeserializers
  location: package tools.jackson.databind.deser.std
```

That will be the first area of focus after we cover some Spring Boot 3 ⇒ Spring Boot 4 migration basics.

Chapter 5. Dependency Changes

5.1. Spring Boot Version

The first and most obvious change was the update `springboot.version` from `3.5.5` to `4.0.6`. This setting was in both `ejava-build-bom` (identifying dependencyManagement) and `ejava-build-parent` (identifying pluginManagement).

Spring Boot 3 Setting	*Spring Boot 4 Setting*
`<springboot.version>3.5.5</springboot.version>`	`<springboot.version>4.0.6</springboot.version>`

The version setting is used to import the targeted dependency definitions from `spring-boot-dependencies`.

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>${springboot.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
```

5.2. Dependency Analysis

I used Claude Code/Sonnet 4.6 to analyze the dependency and code base changes after the port was complete. These are some insights Claude came up with my input.

5.3. Starter Dependency Map: SB3 → SB4

The tree below uses the SB3 starter structure as the baseline. Red (`-`) lines mark artifacts that were renamed, moved, or removed in SB4 — and where each went. Unchanged starters and their unchanged sub-tree nodes appear in neutral. The analysis covers starters used across the `app`, `svc-api`, `svc-security`, `db-jpa`, and `db-mongo` modules; `springdoc-openapi-starter-webmvc-ui` is excluded.

SB3 ⇒ SB4 Where Did They Go

```
 -- Main Starters --

-spring-boot-starter-web (renamed → spring-boot-starter-webmvc)
 +- spring-boot-starter
-|  +- spring-boot (moved → spring-boot-http-converter)
 |  +- spring-boot-autoconfigure
```

```
 |  \- spring-boot-starter-logging
-+- spring-boot-starter-json (renamed → spring-boot-starter-jackson)
-|  +- jackson-databind 2.x (replaced by tools.jackson.core:jackson-databind 3.x)
-|  +- jackson-datatype-jdk8 (removed; built into Jackson 3)
-|  +- jackson-datatype-jsr310 (removed; built into Jackson 3)
-|  \- jackson-module-parameter-names (removed; built into Jackson 3)
 +- spring-boot-starter-tomcat
-|  +- tomcat-embed-core 10.1.44 (moved → spring-boot-starter-tomcat-runtime; 11.0.21)
-|  +- tomcat-embed-el 10.1.44 (moved → spring-boot-starter-tomcat-runtime; 11.0.21)
-|  \- tomcat-embed-websocket 10.1.44 (moved → tomcat-runtime; 11.0.21)
-+- spring-web (moved → spring-boot-http-converter)
-|  |  [RestTemplate + RestClient lived here; no explicit dep needed in SB3]
-|  +- spring-beans
-|  \- micrometer-observation
-+- spring-webmvc (moved → spring-boot-webmvc)
-   +- spring-context (moved → spring-boot-http-converter)
    +- spring-aop
    \- spring-expression

 spring-boot-starter-security (leaf in SB3; gains spring-boot-security sub-module in
SB4)

 spring-boot-starter-test
 +- spring-boot-test
 \- spring-boot-test-autoconfigure

 spring-boot-starter-data-jpa (no sub-modules visible in SB3; hierarchy added in SB4)
 \- spring-boot-starter-jdbc

 spring-boot-starter-data-mongodb (no sub-modules visible in SB3; hierarchy added in
SB4)

 -- WebClient (db-jpa, db-mongo, svc-api) --

-spring-boot-starter-webflux (use spring-boot-starter-webclient for client-only use)

 -- Embedded Mongo (db-mongo) --

-de.flapdoodle.embed.mongo.spring3x (→ de.flapdoodle.embed.mongo.spring4x)

 -- New Dedicated Starters in SB4 --

 (classes in spring-web, no starter) → spring-boot-starter-restclient
 (required spring-boot-starter-webflux) → spring-boot-starter-webclient
 (absent)                              → spring-boot-starter-webmvc-test
 (absent)                              → spring-boot-starter-security-test
 (absent)                              → spring-boot-resttestclient
 (absent)                              → spring-boot-webtestclient

 -- Test Framework (via spring-boot-starter-test) --
```

```
-org.junit.jupiter:junit-jupiter 5.12.2
-org.junit.platform:junit-platform-* 1.12.2
-org.spockframework:spock-bom 2.4-M1-groovy-4.0 (Groovy 4)
```

5.4. Starter Dependency Map: SB4 ← SB3

The tree below uses the SB4 starter structure. Green (+) lines mark artifacts that are new, renamed, or reparented — with their SB3 origin in parentheses. Unchanged nodes appear in neutral.

```
 -- Main Starters --

+spring-boot-starter-webmvc (from spring-boot-starter-web)
 +- spring-boot-starter
 |  +- spring-boot-autoconfigure
 |  \- spring-boot-starter-logging
++- spring-boot-starter-jackson (from spring-boot-starter-json)
+|  \- spring-boot-jackson (NEW — Jackson 3 wrapper)
+|     \- tools.jackson.core:jackson-databind (replaces com.fasterxml 2.x)
+|        +- jackson-annotations 2.x (compat shim; was a peer)
+|        \- tools.jackson.core:jackson-core (replaces com.fasterxml 2.x)
 +- spring-boot-starter-tomcat
++|  +- spring-boot-starter-tomcat-runtime (NEW)
+|  |  +- spring-boot-web-server (NEW)
+|  |  +- tomcat-embed-core 11.0.21 (from spring-boot-starter-tomcat direct)
+|  |  +- tomcat-embed-el 11.0.21 (from spring-boot-starter-tomcat direct)
+|  |  \- tomcat-embed-websocket 11.0.21 (from spring-boot-starter-tomcat direct)
++|  \- spring-boot-tomcat (NEW)
++- spring-boot-http-converter (NEW)
+|  +- spring-boot (from spring-boot-starter)
+|  |  +- spring-core (from spring-boot-starter)
+|  |  |  +- commons-logging (NEW; replaced spring-jcl)
+|  |  |  \- jspecify (NEW)
+|  |  \- spring-context (from spring-webmvc)
+|  \- spring-web (from spring-boot-starter-web direct)
+|     |  [RestTemplate + RestClient here; auto-config moved to spring-boot-
restclient]
+|     +- spring-beans
+|     \- micrometer-observation
+\- spring-boot-webmvc (NEW)
+   +- spring-boot-servlet (NEW)
+   \- spring-webmvc (from spring-boot-starter-web direct)
+      +- spring-aop
+      \- spring-expression

 spring-boot-starter-security
++\- spring-boot-security (NEW internal module)

 spring-boot-starter-test
 +- spring-boot-test
```

```
 \- spring-boot-test-autoconfigure

 spring-boot-starter-data-jpa
 +- spring-boot-starter-jdbc
++|  +- spring-boot-jdbc (NEW)
+|  |  +- spring-boot-sql (NEW)
+|  |  \- spring-boot-transaction (NEW)
++\- spring-boot-data-jpa (NEW)
+   +- spring-boot-data-commons (NEW)
+   |  \- spring-boot-persistence (NEW)
+   \- spring-boot-hibernate (NEW)
+      \- spring-boot-jpa (NEW)

 spring-boot-starter-data-mongodb
++\- spring-boot-starter-mongodb (NEW)
+|  \- spring-boot-mongodb (NEW)
++\- spring-boot-data-mongodb (NEW)

 -- WebClient (db-jpa, db-mongo, svc-api) --

+spring-boot-starter-webclient (NEW — replaces starter-webflux for client-only use)
+\- spring-boot-reactor (NEW)
+\- spring-boot-webclient (NEW)

 -- Embedded Mongo (db-mongo) --

+de.flapdoodle.embed.mongo.spring4x (from spring3x; 4.24.0)

 -- New Starters with no SB3 Counterpart --

+spring-boot-starter-restclient (NEW — RestTemplate + RestClient)
+\- spring-boot-restclient
+   \- spring-boot-http-client

+spring-boot-starter-webmvc-test (NEW — MockMvc test slice)
+\- spring-boot-starter-jackson-test
+\- spring-boot-webmvc-test
+   \- spring-boot-web-server

+spring-boot-starter-security-test (NEW)
+\- spring-boot-security-test

+spring-boot-resttestclient (NEW — TestRestTemplate)
+spring-boot-webtestclient (NEW — WebTestClient)
+\- spring-boot-http-codec

 -- Test Framework (via spring-boot-starter-test) --

+org.junit.jupiter:junit-jupiter 6.0.3  (from 5.12.2)
+org.junit.platform:junit-platform-* 6.0.3 (from 1.12.2; versions now aligned)
```

```
+org.spockframework:spock-bom 2.4-M7-groovy-5.0 (from 2.4-M1-groovy-4.0)
```

5.5. Change Details

5.5.1. Web Starter: spring-boot-starter-web → spring-boot-starter-webmvc

The rename is the most visible change — every `pom.xml` that declared `spring-boot-starter-web` must be updated. OpenRewrite handles this automatically.

Two non-Spring-Boot artifacts changed in the web starter sub-tree:

- `spring-jcl` was removed. Spring 7's `spring-core` now declares a direct dependency on `commons-logging` instead of bundling its own bridge.
- `jspecify` (null-safety annotations) is new, pulled by `spring-core` in Spring 7.

5.5.2. Jackson 3

`spring-boot-starter-json` was renamed to `spring-boot-starter-jackson`. Its sub-tree changed substantially: the new `spring-boot-jackson` module pulls Jackson 3 (`tools.jackson.core`) rather than Jackson 2 (`com.fasterxml.jackson.core`). The three Jackson 2 extension modules — `jackson-datatype-jdk8`, `jackson-datatype-jsr310`, and `jackson-module-parameter-names` — are absent from the SB4 tree because their functionality is built into Jackson 3 core.

A compatibility shim remains: `jackson-annotations 2.x` (from `com.fasterxml.jackson.core`) survives as a transitive dependency of `tools.jackson.core:jackson-databind 3.x`. Code importing from `com.fasterxml.jackson.*` will still compile if Jackson 2 annotation classes are used, but the active mapper is Jackson 3. See Jackson 3 and Jackson2ObjectMapperBuilderCustomizer / JsonMapperBuilderCustomizer for the full porting path.

5.5.3. Tomcat 10 → 11 and Jakarta EE Alignment

The embedded Tomcat version advanced from 10.1.44 to 11.0.21. Tomcat 11 targets Jakarta EE 11; this ripples to two Jakarta API version bumps managed by the Spring Boot BOM:

Artifact	SB3	SB4
`jakarta.annotation:jakarta.annotation-api`	2.1.1	3.0.0
`jakarta.persistence:jakarta.persistence-api`	3.1.0	3.2.0

No code changes are required for the Tomcat or Jakarta EE version bumps; the BOM manages all versions.

5.5.4. HTTP Clients: RestTemplate, RestClient, and WebClient

Where the classes lived in SB3

`RestTemplate` and `RestClient` both live in `spring-web`. In SB3, `spring-boot-starter-web` pulled `spring-web` as a direct child, so both client types landed on the classpath for free — no explicit dependency

required. `RestClient` was added to `spring-web` in Spring 6.1 (SB 3.2), so it was also free-riding the web starter from SB3.2 onward.

`RestTemplateBuilder` is a Spring Boot construct (not a Spring Framework class). In SB3 it lived in `spring-boot-autoconfigure` under the package `org.springframework.boot.web.client`, which is pulled in by `spring-boot-starter` — again a free transitive of `spring-boot-starter-web`.

In SB3 the net effect was: declare `spring-boot-starter-web`, get `RestTemplate`, `RestClient`, and `RestTemplateBuilder` with zero additional entries in `pom.xml`.

What changed in SB4

`spring-web` is still on the classpath in SB4 — it moved under `spring-boot-http-converter` inside `spring-boot-starter-webmvc`. The `RestTemplate` and `RestClient` classes therefore remain accessible. What moved out is the Spring Boot infrastructure that wraps them:

- `RestTemplateBuilder` and its auto-configuration moved from `spring-boot-autoconfigure` into `spring-boot-restclient`, inside `spring-boot-starter-restclient`.
- The package for `RestTemplateBuilder` changed accordingly.

SB3 import	SB4 import
`org.springframework.boot.web.client.RestTemplateBuilder`	`org.springframework.boot.restclient.RestTemplateBuilder`

The practical result: code that injected or constructed a `RestTemplateBuilder` compiles fine in SB3 but fails to resolve the symbol in SB4 without `spring-boot-starter-restclient` on the classpath. Code that constructs `new RestTemplate()` directly still compiles (the class is still in `spring-web`), but loses Spring Boot auto-configuration.

Dependency mapping

Type	SB3 pom.xml entry	SB4 pom.xml entry
`RestTemplate`, `RestClient` (classes)	(transitive via `spring-boot-starter-web`)	(transitive via `spring-boot-starter-webmvc`)
`RestTemplateBuilder`, client auto-config	(transitive via `spring-boot-starter-web`)	`spring-boot-starter-restclient`
`WebClient`, `WebClient.Builder`	`spring-boot-starter-webflux` (pulls full Reactive server)	`spring-boot-starter-webclient` (client only, no server)

This resulted in some common dependency changes/additions.

```
<dependency> <!-- to get RestTemplateBuilder and RestClient.Builder -->
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-restclient</artifactId>
    <scope>...</scope>
</dependency>
```

```
<dependency> <!-- server dependency replaced, not needed ... -->
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
    <scope>test</scope>
</dependency>
<dependency> <!-- ... to get WebClient and WebClient.Builder -->
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webclient</artifactId>
    <scope>test</scope>
</dependency>
```

WebClient

In SB3, the most common way to get `WebClient.Builder` auto-configured was `spring-boot-starter-webflux`, which also started a Reactive server — a heavyweight dependency for tests that only needed an HTTP client. SB4 separates the client from the server: `spring-boot-starter-webclient` provides `WebClient.Builder` auto-configuration with no WebFlux server. For test-scope-only use, `spring-boot-webclient` (no `starter-` prefix) is a lighter option.

`WebClient.Builder` is no longer auto-configured by `spring-boot-starter-webmvc`. Code that injects `WebClient.Builder` will fail at startup until a webclient dependency is declared.

See WebFlux Not Needed for WebClient and WebClient.Builder for the error messages and detailed fix steps.

5.5.5. Test Clients and Test Slices

Spring Boot 4 extracts test client types and test slice support into separate artifacts.

TestRestTemplate

In SB3, `TestRestTemplate` lived in `spring-boot-test` (`org.springframework.boot.test.web.client`), which is pulled by `spring-boot-starter-test`. It was available with no extra dependency; injecting it into a `@SpringBootTest` worked out of the box.

In SB4 it moved to the new `spring-boot-resttestclient` artifact (`org.springframework.boot.resttestclient`), which is not pulled by `spring-boot-starter-test`. The dependency must be declared explicitly, and `@AutoConfigureTestRestTemplate` must be added to the test class for injection to succeed. See TestRestTemplate for the compile error and the fix.

WebTestClient

In SB3, `WebTestClient` (from `spring-test`, package `org.springframework.test.web.reactive.server`) was also brought in by `spring-boot-starter-test` via `spring-test`. The auto-configuration for it (`@AutoConfigureWebTestClient`) lived in `spring-boot-test-autoconfigure`, also part of `spring-boot-starter-test`. However, `WebTestClient` relies on reactive infrastructure — in practice you also needed `spring-boot-starter-webflux` on the test classpath to make full use of it.

In SB4, `WebTestClient` support is backed by the dedicated `spring-boot-webtestclient` artifact, which pulls `spring-boot-http-codec` for codec support. The explicit dependency is required; `spring-boot-`

`starter-test` alone no longer provides it.

MockMvc Test Slice

In SB3, `@AutoConfigureMockMvc` and the full MockMvc test slice lived in `spring-boot-test-autoconfigure` (`org.springframework.boot.test.autoconfigure.web.servlet`). That module is pulled by `spring-boot-starter-test`, so MockMvc support was free — no separate dependency, no separate import package.

In SB4 it moved out of `spring-boot-test-autoconfigure` into a dedicated module. `@AutoConfigureMockMvc` is now in `org.springframework.boot.webmvc.test.autoconfigure`, and the dependency `spring-boot-starter-webmvc-test` (or bare `spring-boot-webmvc-test`) must be declared explicitly.

SB3 import	SB4 import
`org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc`	`org.springframework.boot.webmvc.test.autoconfigure.AutoConfigureMockMvc`

The dependency is `spring-boot-starter-webmvc-test` (which pulls `spring-boot-webmvc-test` and `spring-boot-starter-jackson-test`). The bare `spring-boot-webmvc-test` artifact is also available for test scope without the transitive jackson-test pull. See AutoConfigureMockMvc for the compile error and the fix.

Security Test

Spring Security test support is now a first-class Spring Boot starter. In SB3 this was a direct `spring-security-test` dependency; SB4 wraps it as `spring-boot-starter-security-test` → `spring-boot-security-test`.

5.5.6. Security Starter

`spring-boot-starter-security` is structurally unchanged at the starter level. Internally, a new `spring-boot-security` module now backs the starter; this is an implementation detail requiring no `pom.xml` changes.

The two security API changes that require code fixes are covered in separate sections:

- `AntPathRequestMatcher` removed → see AntPathRequestMatcher / PathPatternRequestMatcher
- `MvcRequestMatcher` / `HandlerMappingIntrospector` removed → see MvcRequestMatcher / HandlerMappingIntrospector removed
- `@MockBean` / `@SpyBean` removed → see MockBean / MockitoBean

5.5.7. Data JPA Starter

`spring-boot-starter-data-jpa` is internally restructured into a module hierarchy. The new modules — `spring-boot-data-jpa`, `spring-boot-data-commons`, `spring-boot-persistence`, `spring-boot-hibernate`, `spring-boot-jpa`, `spring-boot-jdbc`, `spring-boot-sql`, `spring-boot-transaction` — are implementation details of the starter and require no `pom.xml` changes.

Two JPA-related items do require code or dependency changes:

- `@EntityScan` moved packages — see EntityScan.
- `flyway-core` alone no longer integrates cleanly with the JPA lifecycle; use `spring-boot-starter-flyway` — see Flyway as a starter.

`spring-boot-starter-flyway` pulls its own JDBC stack and `flyway-core 12.4.0`:

```
+spring-boot-starter-flyway (NEW — replaces bare flyway-core dependency)
 +- spring-boot-starter
 |  +- spring-boot-starter-logging
 |  +- spring-boot-autoconfigure
 |  +- jakarta.annotation-api 3.0.0
 |  \- snakeyaml
 +- spring-boot-starter-jdbc
 |  \- HikariCP 7.0.2
++- spring-boot-flyway (NEW)
+|  +- spring-boot
+|  |  +- spring-core
+|  |  |  +- commons-logging
+|  |  |  \- jspecify
+|  |  \- spring-context
+|  \- flyway-core 12.4.0
+|     \- tools.jackson.core:jackson-databind 3.x
 \- spring-boot-jdbc
    +- spring-boot-sql
    +- spring-boot-transaction
    |  +- spring-boot-persistence
    |  \- spring-tx
    \- spring-jdbc
```

Note that `flyway-core 12.4.0` itself pulls `tools.jackson.core:jackson-databind` (Jackson 3) — the same Jackson 3 artifact that `spring-boot-starter-webmvc` brings in via `spring-boot-jackson`. Projects that use Flyway without the web starter will get Jackson 3 on the classpath through this path.

5.5.8. MongoDB Starter and Flapdoodle

`spring-boot-starter-data-mongodb` grew two new wrapper modules; the MongoDB driver and Spring Data artifacts moved under them.

```
-spring-boot-starter-data-mongodb 3.5.5
-+- org.mongodb:mongodb-driver-sync 5.5.1 (moved → spring-boot-starter-mongodb)
-|  +- org.mongodb:bson 5.5.1
-|  \- org.mongodb:mongodb-driver-core 5.5.1
-|     \- org.mongodb:bson-record-codec 5.5.1 (runtime)
-\- spring-data-mongodb 4.5.3 (moved → spring-boot-data-mongodb)

+spring-boot-starter-data-mongodb 4.1.0-RC1
++- spring-boot-starter-mongodb (NEW)
```

```
+|  +- spring-boot-mongodb (NEW)
+|  \- org.mongodb:mongodb-driver-sync 5.7.0-beta1 (from starter direct)
+|     +- org.mongodb:bson 5.7.0-beta1
+|     \- org.mongodb:mongodb-driver-core 5.7.0-beta1
+|        \- org.mongodb:bson-record-codec 5.7.0-beta1 (runtime)
++\- spring-boot-data-mongodb (NEW)
+   \- spring-data-mongodb 5.1.0-RC1 (from starter direct)
```

`de.flapdoodle.embed.mongo.spring4x` is a pure artifact rename; its transitive structure is unchanged from `spring3x`:

```
-de.flapdoodle.embed.mongo.spring3x
+de.flapdoodle.embed.mongo.spring4x 4.24.0
```

Three MongoDB items require action:

- Replace `de.flapdoodle.embed.mongo.spring3x` with `de.flapdoodle.embed.mongo.spring4x` (4.24.0) — see Flapdoodle for Spring 4.
- Auto-configuration class packages moved — see Mongo Autoconfigure Imports.
- Property prefix changed from `spring.data.mongodb.` to `spring.mongodb.` — see MongoProperties / property namespace.

5.5.9. Test Framework

Framework	SB3 version	SB4 version
JUnit Jupiter	5.12.2	6.0.3
JUnit Platform	1.12.2	6.0.3
Mockito	5.17.0	5.23.0
AssertJ	3.27.4	3.27.7

JUnit 6

JUnit Platform 6 ships with Spring Boot 4, and its version number now aligns with JUnit Jupiter (both 6.x), eliminating the historic 5.x / 1.x split. `junit-vintage-engine` (JUnit 4 compatibility) is no longer pulled transitively by `spring-boot-starter-test`; modules that still depend on JUnit 4 must declare it explicitly. See JUnit 6 and IntelliJ for an IntelliJ compatibility issue that affects running tests from the IDE.

Spock

The BOM version advances from `2.4-M1-groovy-4.0` to `2.4-M7-groovy-5.0`. Spock M7 requires Groovy 5 and is incompatible with Groovy 4; the BOM entry must be updated together. Two new transitive artifacts arrive with M7: `io.leangen.geantyref:geantyref` (Spock M7 internals) and `org.jspecify:jspecify` (pulled by JUnit Platform 6). See Spock Version for the Groovy incompatibility error and the version fix.

Chapter 6. Details

The following sections list individual issues with symptoms and corrections needed post OpenRewrite.

6.1. Jackson 3

Some of the things OpenRewrite did was to

- change the Jackson dependency groupIds from `com.fasterxml.jackson.*` to `tools.jackson.*`
- change the Jackson Java packages from `com.fasterxml.jackson.*` to `tools.jackson.*`

Some of the legacy Jackson 2 annotations still exist and are supplied through transient dependencies from Jackson 3 artifacts. The figure below shows the Jackson 3 dependency tree and where the one surviving Jackson 2 artifact comes in.

Jackson 3 Dependency Tree

```
 spring-boot-starter-jackson
 \- spring-boot-jackson
    \- tools.jackson.core:jackson-databind:3.1.2      (Jackson 3)
+      +- com.fasterxml.jackson.core:jackson-annotations:2.21  (Jackson 2 compat shim)
       \- tools.jackson.core:jackson-core:3.1.2        (Jackson 3)
```

`jackson-annotations` (`com.fasterxml.jackson.core`) is the one Jackson 2 artifact still pulled in — as a compatibility shim by `jackson-databind` 3.x. Code that imports `com.fasterxml.jackson.annotation.*` (e.g., `@JsonProperty`, `@JsonIgnore`) will therefore compile without a separate Jackson 2 dependency. The rest of the Jackson 2 groupId (`jackson-databind`, `jackson-core`) is absent; only the annotations jar survives as a bridge.

We hit most of the Jackson issues in the first module built: `common`.

6.1.1. DateDeserializers / JsonMapper

One of the first errors we run into is a failed build in `common` module with a Jackson issue.

Jackson Imports Changed to Jackson 3

```
import tools.jackson.databind.deser.std.DateDeserializers;
import tools.jackson.databind.ser.std.DateSerializer;
```

Missing Jackson 3 Dependency

```
[ERROR] /Users/jim/proj/ejava-javaee/boot4-port/common/ejava-dto-
util/src/main/java/info/ejava/examples/common/dto/JsonUtil.java:[4,40] cannot find
symbol
[ERROR]   symbol:   class DateDeserializers
```

```
[ERROR]   location: package tools.jackson.databind.deser.std
```

That specific class was renamed to be `JDKDateDeserializers`, but we actually don't need it anymore if we migrate this code more fully to Jackson 3. In Jackson 2, we had to configure the ObjectMapper with date constructs

Jackson 2 Initialization Code for JsonUtil

```
protected final ObjectMapper mapper;
...

public JsonUtil() {
    mapper = new JsonMapper();
    SimpleDateFormat sdf = new SimpleDateFormat(DATETIME_FORMAT);
    sdf.setTimeZone(TimeZone.getTimeZone("UTC"));

    SimpleModule dateSerialization = new SimpleModule();
    dateSerialization.addDeserializer(Date.class, DateDeserializers.DateDeserializer
.instance);
    dateSerialization.addSerializer(Date.class, new DateSerializer(false, sdf));

    mapper.registerModule(dateSerialization);
}

public JsonUtil(ObjectMapper mapper) {
    this.mapper = mapper;
}
```

To port this more fully, we change the `ObjectMapper` to `JsonMapper`, leveraged the builder, and remove all the custom date settings that are now active by default.

Jackson 3 Initialization Code for JsonUtil

```
protected final JsonMapper mapper;
...

public JsonUtil() {
    this(JsonMapper.builder()
            .enable(SerializationFeature.INDENT_OUTPUT) //option unchanged
            .build());
}
public JsonUtil(JsonMapper mapper) {
    this.mapper = mapper;
}
```

JsonMapper is Unmodifiable

Unlike `ObjectMapper`, JsonMapper` is unmodifiable. This makes it safer to use since there is no longer a chance that a shared mapper gets adjusted by one of its

> injections and seen by others. This makes sharing mappers more sane.

Many of the serialization/deserialization dependencies are included by default. The following shows relevant setting defaults:

Serialization Feature	Jackson 2 default	Jackson 3 default
INDENT_OUTPUT	off	off (unchanged)
WRITE_DATES_AS_TIMESTAMPS	on	**off**

6.1.2. XmlMapper

With that corrected, we get the same error with the XmlUtil.

```
[ERROR] /Users/jim/proj/ejava-javaee/boot4-port/common/ejava-dto-
util/src/main/java/info/ejava/examples/common/dto/XmlUtil.java:[3,40] cannot find
symbol
[ERROR]   symbol:   class DateDeserializers
```

Jackson 2 Initialization Code for XmlUtil

```
protected XmlMapper mapper = new XmlMapper();

public XmlUtil() {
    SimpleDateFormat sdf = new SimpleDateFormat(DATETIME_FORMAT);
    sdf.setTimeZone(TimeZone.getTimeZone("UTC"));

    SimpleModule dateSerialization = new SimpleModule();
    dateSerialization.addDeserializer(Date.class, DateDeserializers.DateDeserializer
.instance);
    dateSerialization.addSerializer(Date.class, new DateSerializer(false, sdf));

    mapper.registerModule(dateSerialization);
}
public XmlUtil(XmlMapper mapper) {
    this.mapper = mapper;
}
```

The `XmlMapper` has the same unmodifiable traits as `JsonMapper`.

Jackson 3 Initialization Code for XmlUtil

```
protected XmlMapper mapper;

public XmlUtil() {
    this(XmlMapper.builder()
            .enable(SerializationFeature.INDENT_OUTPUT) //option unchanged
            .build());
}
```

```
public XmlUtil(XmlMapper mapper) {
    this.mapper = mapper;
}
```

6.1.3. HttpHeaders / MultiValueMap

Unrelated to Jackson, but included within the `ejava-util-dto` — HttpHeaders no longer implements MultiValueMap.

```
import org.springframework.http.HttpHeaders;
...
public MultiValueMap<String, String> getQueryParams() {
    MultiValueMap<String, String> queryParams = new HttpHeaders();
```

```
[ERROR] /Users/jim/proj/ejava-javaee/boot4-port/common/ejava-dto-
util/src/main/java/info/ejava/examples/common/dto/paging/PageableDTO.java:[114,53]
incompatible types: org.springframework.http.HttpHeaders cannot be converted to
org.springframework.util.MultiValueMap<java.lang.String,java.lang.String>
```

We can quickly make that work by changing the data type to `LinkedMultiValueMap`;

```
import org.springframework.util.LinkedMultiValueMap;
...
public MultiValueMap<String, String> getQueryParams() {
    MultiValueMap<String, String> queryParams = new LinkedMultiValueMap<>();
```

6.1.4. Ambiguous Parent/Child XML Elements

Moving on to the `ejava-web-util` module, we encounter some hard-core Jackson 3 migration issues that took a while to get through. There were many spots to address, which make it tough to explain when we get to the weeds. See what you can benefit from this portion of the write-up.

The first of which is that the tests were in the wrong module.

```
tools.jackson.databind.exc.InvalidDefinitionException:
Invalid definition for property 'content' (of type
`info.ejava.examples.common.web.paging.PageDTOTest$IntegerPageDTO`): Could not find
creator property with name 'content' (known Creator properties: [value, totalElements,
pageSpec])
```

I moved them back to the `ejava-dto-util`. The DTO classes where likely refactored as some point in the past and the tests did not follow with them. I also added `spring-boot-starter-test` since this was the first test added to the module.

Adding Test Classes

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

Now back to the error. When I looked at the total results, the JSON format was passing and only the XML was failing.

```
14:46:43.048 [main] INFO info.ejava.examples.common.dto.paging.PageDTOTest -- <intPage
xmlns="urn:ejava.common.dto.test" totalElements="100">
  <content>
    <value xmlns="">1</value>
    <value xmlns="">2</value>
    <value xmlns="">3</value>
  </content>
  <wstxns1:pageable xmlns:wstxns1="urn:ejava.common.dto" pageNumber="1" pageSize="25"
sort="title:ASC,artist:ASC"/>
</intPage>


tools.jackson.databind.exc.InvalidDefinitionException: Invalid definition for property
'content' (of type
`info.ejava.examples.common.dto.paging.PageDTOTest$IntegerPageDTO`): Could not find
creator property with name 'content' (known Creator properties: [value, totalElements,
pageSpec])
```

The first fix was to add an explicit reference of the wrapper class to the setter() in the base class.

Added Explicit Mapping of Parent Setter to Wrapper Class

```
public class PageDTO<T> {
    private List<T> content;
    public PageDTO(List<T> content, Long totalElements, PageableDTO pageSpec) { ...

    /**
     * Jackson 3 XML (version 3.0.4) required this parent setter to be annotated
     * with the wrapper property name to resolve ambiguity between wrapper and element
     * coming from the derived class.
     * @param content list of elements
     */
    @JsonProperty("content")
    public void setContent(List<T> content) {
        this.content = content;
    }
```

Now the problem shifted to a different abiguity issue.

```
java.lang.IllegalStateException: Conflicting/ambiguous property name definitions
(implicit name 'content'): found multiple explicit names: [content, {}value], but also
implicit accessor: [parameter #0, annotations:
[null]][visible=true,ignore=false,explicitName=false]
```

This class marshals itself to a facade and the Jackson 2 facade markings are not working the same with Jackson 3.

This ambiguity gets corrected by adding a `@JsonRootname` to the outer `PageableDTO` and that is carried forward to the facade.

Outer Class now has Root Name

```
@JsonRootName(value="pageable", namespace = "urn:ejava.common.dto") //serializer was
ignoring root on facade class
@JsonDeserialize(builder= PageableDTO.DtoFacade.class)
@JsonSerialize(using= PageableDTO.Serializer.class)
public class PageableDTO {
```

Facade no longer has the Root Name.

```
    @JsonPOJOBuilder
    public static class DtoFacade {
```

Choose JsonRootName over JacksonXmlRootElement

Jackson has 2 legacy annotations: `JacksonXmlRootElement` and `JsonRootName`. `JsonRootName` is a more generalized than its explicit XML counterpart. The Javadocs state you should use `JsonRootName` instead. They both work.

The last mysterious change needed to get beyond the ambiguity errors was to annotate the constructor collection parameter with the name of the wrapper class (the actual type of the payload).

Concrete Class Needed an explicit Property Mapping on the wrapper class

```
public IntegerPageDTO(
        /**
         * Jackson 3 XML (v3.0.4) requires this ctor property to be annotated
         * with the wrapper property name to resolve base/derived class references
         * to property with different annotations.
         */
        @JsonProperty("content") List<Integer> content,
        Long totalElements, PageableDTO pageSpec) {
    super(content, totalElements, pageSpec);
}
```

6.1.5. Ignores

I found where properties were ignored in the past were no longer being ignored. This caused extra properties to be marshaled into the payload. The simple fix was to make a pass through the code and make sure that every getter/setter/is that should not be a property is marked with `@JsonIgnore`

Add @JsonIgnore to Accessor Methods

```
    @JsonIgnore
    public boolean isUnpaged() {
        return pageNumber==null || pageSize==null;
    }
```

6.1.6. JacksonException

One small issue and I am not sure why my OpenRewrite did not change it — Jackson 3 now throws an unchecked `JacksonException` versus a checked `IOException`. Existing signatures ...

```
public <T> void marshalThrows(T object, OutputStream os) throws IOException {
```

can be declared as throwing the `JacksonException` or leave it out entirely since it is unchecked.

```
public <T> void marshalThrows(T object, OutputStream os) throws JacksonException {
```

6.2. Jackson2ObjectMapperBuilderCustomizer / JsonMapperBuilderCustomizer

With the change from Jackson 2 to 3, we also need to change how the mappers are built/customized.

Instead of the following customizer that accepts a `Jackson2ObjectMapperBuilder`...

Jackson 2 ObjectMapper Customizer

```
@Bean
@Order(Ordered.HIGHEST_PRECEDENCE)
public Jackson2ObjectMapperBuilderCustomizer jacksonMapper() {
    return (builder) -> { builder
        //spring.jackson.serialization.indent-output=true
        .featuresToEnable(SerializationFeature.INDENT_OUTPUT)
        //spring.jackson.serialization.write-dates-as-timestamps=false
        .featuresToDisable(DateTimeFeature.WRITE_DATES_AS_TIMESTAMPS)
        //spring.jackson.date-
format=info.ejava.examples.svc.content.quotes.dto.ISODateFormat
        .dateFormat(new ISODateFormat());
    };
```

```
}
```

... we provided a customizer thar accepts a `JsonMapper.Builder`.

Jackson 3 JsonMapper Customizer

```
@Bean
@Order(Ordered.HIGHEST_PRECEDENCE)
public JsonMapperBuilderCustomizer jacksonMapper() {
    return (builder) -> builder
        //spring.jackson.serialization.indent-output=true
        .enable(SerializationFeature.INDENT_OUTPUT)
        //spring.jackson.date-
format=info.ejava.examples.svc.content.quotes.dto.ISODateFormat
        .defaultDateFormat(new ISODateFormat());
}
```

6.3. Jacksonize

There are times when you don't want Jackson to directly construct a DTO object. You want Jackson to use a builder to construct the DTO object and have the builder be a Lombok builder.

You can trigger that option with the use of `@Jacksonize` on the class.

Jacksonized Class

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Jacksonized
public class QuoteListDTO {
```

However, without doing anything, Lombok will create a Jackson 2 Jacksonized capability.

Jacksonize Jackson 2 Error

```
JacksonXmlTest.unmarshal:36 » InvalidDefinition Invalid definition for property
'quotes' (of type `info.ejava.examples.svc.content.quotes.dto.QuoteListDTO`): Could
not find creator property with name 'quotes' (known Creator properties: [offset,
limit, total, keywords, quote])
```

In order to implement a Jackson 3 solution for the `@Jacksonize` annotation, we need to add a `lombok.config` in a directory above the source code. This file was placed next to the pom.xml. Two options defined:

- config.stopBubbling - tells lombok whether to keep searching for other `lombok.config` files
- `lombok.jacksonized.jacksonVersion` - tells lombok the version of Jackson to author the

Jacksonized changes for.

jackson.config

```
config.stopBubbling = true
# This is the critical flag for Jackson 3 / Spring Boot 4
# Use += to specify the version in 1.18.44
lombok.jacksonized.jacksonVersion += 3

# Required if you want annotations copied to the builder constructor
#lombok.copyableAnnotations +=
tools.jackson.dataformat.xml.annotation.JacksonXmlProperty
#lombok.copyableAnnotations +=
tools.jackson.dataformat.xml.annotation.JacksonXmlElementWrapper
```

6.4. Spock Version

When building the spock examples with the legacy BOM version, the following error occurs.

Error Message Building Spock Test with legacy Versions

```
Groovyc: While compiling [tests of apptesting-spock-example]: Could not instantiate
global transform class org.spockframework.compiler.SpockTransform specified at
jar:file:/Users/jim/.m2/repository/org/spockframework/spock-core/2.4-M1-groovy-
4.0/spock-core-2.4-M1-groovy-4.0.jar!/META-
INF/services/org.codehaus.groovy.transform.ASTTransformation  because of exception
org.spockframework.util.IncompatibleGroovyVersionException: The Spock compiler plugin
cannot execute because Spock 2.4.0-M1-groovy-4.0 is not compatible with Groovy 5.0.5.
For more information (including enforce mode), see https://docs.spockframework.org
(section 'Known Issues').
```

Legacy Spock Version

```
<spock.version>2.4-M1-groovy-4.0</spock.version>
```

Changiung the version to the following allows the spock module to complete.

Compatible Spock Version

```
<spock.version>2.4-M7-groovy-5.0</spock.version>
```

6.4.1. TypeExcludeFilters Package

The package for TypeExcludeFilters has changed.

Example TypeExcludeFilters Package Error

```
 Compilation failure: Compilation failure:
```

```
[ERROR] /Users/jim/proj/ejava-javaee/boot4-port/app/app-testing/apptesting-scanpath-
example/componentscan/src/test/java/info/ejava/examples/app/testing/scanpath/component
scan/app/MyTypeExcludeFilterNTest.java:[6,58] package
org.springframework.boot.test.autoconfigure.filter does not exist
```

before

```
import org.springframework.boot.test.autoconfigure.filter.TypeExcludeFilters;
```

Changed the package to the following:

after

```
import org.springframework.boot.test.context.filter.annotation.TypeExcludeFilters;
```

6.5. WebFlux Not Needed for WebClient

Spring Boot 4 has separated `WebClient` from WebFlux so that we no longer need to add the `spring-boot-starter-webflux` server dependency to our client applications/tests if we use `WebClient`. We can instead add a dependency on `spring-boot-starter-webclient`.

Replace webflux Server Dependency for Clients

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
        <scope>test</scope>
    </dependency>
```

Use WebClient Dependency for Clients

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webclient</artifactId>
        <scope>test</scope>
    </dependency>
```

The symptoms for this are when the tests fail to start because the `WebClient.Builder` cannot be found for injection.

Application Failing to Start

```
.IllegalStateException: ApplicationContext failure threshold (1) exceeded: skipping
repeated attempt to load context for [WebMergedContextConfiguration@6501b105 testClass
= info.ejava.examples.svc.rpc.greeter.GreeterSyncWebClientNTest
```

6.6. UnprocessableEntity / UnprocessableContent

HTTP 422 was renamed from "Unprocessable Entity" to "Unprocessable Content" in RFC 9110. Spring 7 follows suit — `HttpStatus.UNPROCESSABLE_ENTITY` is now `UNPROCESSABLE_CONTENT` and the matching `HttpClientErrorException.UnprocessableEntity` is now `UnprocessableContent`.

This causes some minor test failures.

Test Failure

```
java.lang.AssertionError:
Expecting actual throwable to be an instance of:
  org.springframework.web.client.HttpClientErrorException.UnprocessableEntity
but was:
  org.springframework.web.client.HttpClientErrorException$UnprocessableContent: 422
Unprocessable Content: "boy named blue?"
```

Erroneous References to 422 being UnprocessableEntity

```
import org.springframework.web.client.HttpClientErrorException;
...
RestClientResponseException ex = catchThrowableOfType(
        HttpClientErrorException.UnprocessableEntity.class,
        () -> greeterAPI.createBoy("blue"));

then(ex.getStatusCode()).isEqualTo(HttpStatus.UNPROCESSABLE_ENTITY);
```

Simple fix is to change all UnprocessableEntity references to UnprocessableContent.

Correct References to 422 being UnprocessableContent

```
RestClientResponseException ex = catchThrowableOfType(
        HttpClientErrorException.UnprocessableContent.class,
        () -> greeterAPI.createBoy("blue"));

then(ex.getStatusCode()).isEqualTo(HttpStatus.UNPROCESSABLE_CONTENT);
```

The same rename applies to wherever you find the 422 being applied.

6.7. MvcRequestMatcher / HandlerMappingIntrospector removed

`MvcRequestMatcher` and `HandlerMappingIntrospector` were removed. In Spring Boot 3, Spring MVC changed to an `MvcRequestMatcher` under the hood for Spring MVC servlets and AntMatcher for non-Spring MVC servlets. There were several class examples created that showed how to explicitly choose between one or the other versus leaving to chance. The explicit choice of `MvcRequestMatcher` seems to no longer be an option — nor doo we need to, so let's let it default.

Spring Boot 4 Compile Error

```
cannot find symbol
symbol:   class HandlerMappingIntrospector
cannot find symbol
symbol:   class MvcRequestMatcher
```

Legacy Spring Boot 3 SecurityFilterChain

```
import org.springframework.security.web.servlet.util.matcher.MvcRequestMatcher;
import org.springframework.web.servlet.handler.HandlerMappingIntrospector;

@Bean
@Order(100)
public SecurityFilterChain swaggerSecurityFilterChain(HttpSecurity http,
                                                       HandlerMappingIntrospector
introspector) throws Exception {
    MvcRequestMatcher.Builder mvc = new MvcRequestMatcher.Builder(introspector);
    http.securityMatchers(cfg->cfg.requestMatchers(
                    mvc.pattern("/swagger-ui*"),
                    mvc.pattern("/swagger-ui/**"),
                    mvc.pattern("/v3/api-docs/**")));
```

Update Spring Boot 4 SecurityFilterChain

```
public SecurityFilterChain swaggerSecurityFilterChain(HttpSecurity http) throws
Exception {
    http.securityMatchers(cfg->cfg.requestMatchers(
                    "/swagger-ui*",
                    "/swagger-ui/**",
                    "/v3/api-docs/**"));
```

6.8. AntPathRequestMatcher / PathPatternRequestMatcher

`AntPathRequestMatcher` was removed in Spring Security 7. The replacement is `PathPatternRequestMatcher.pathPattern(…)`.

compile error

```
cannot find symbol
symbol:   class AntPathRequestMatcher
```

Legacy Spring Boot 3 AntPathRequestMatcher

```
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
...
```

```
    .requestMatchers(RequestMatchers.allOf(
            htmlRequestMatcher,
            AntPathRequestMatcher.antMatcher("/error")
        ))

import static
org.springframework.security.web.util.matcher.AntPathRequestMatcher.antMatcher;
...
    http.csrf(cfg->cfg.ignoringRequestMatchers(antMatcher("/h2-console/**")));
```

Update Spring Boot 4 PathPatternRequestMatcher

```
import
org.springframework.security.web.servlet.util.matcher.PathPatternRequestMatcher;

    .requestMatchers(RequestMatchers.allOf(
            htmlRequestMatcher,
            PathPatternRequestMatcher.pathPattern("/error")
        ))

    http.csrf(cfg -> cfg.ignoringRequestMatchers(
            PathPatternRequestMatcher.pathPattern("/h2-console/**")));
```

6.9. Mongo Autoconfigure Imports

The Mongo AutoConfiguration classes moved to a dedicated `org.springframework.boot.mongodb.autoconfigure` package (and the data variant under `boot.data.mongodb.autoconfigure`).

compile error

```
[ERROR] ...package org.springframework.boot.autoconfigure.data.mongo does not exist
[ERROR] ...package org.springframework.boot.autoconfigure.jdbc does not exist
[ERROR] ...package org.springframework.boot.autoconfigure.mongo does not exist
```

Spring Boot 3 Mongo Java Packages

```
import org.springframework.boot.autoconfigure.data.mongo.MongoDataAutoConfiguration;
import org.springframework.boot.autoconfigure.mongo.MongoAutoConfiguration;
import org.springframework.boot.autoconfigure.mongo.PropertiesMongoConnectionDetails;
```

Notice how the legacy SB3 Mongo autoconfigure was in the Spring Boot Autoconfigure and the SB4 autoconfigure has been broken out into its own set of packages.

Spring Boot 4 Mongo Java Packages

```
import org.springframework.boot.data.mongodb.autoconfigure.DataMongoAutoConfiguration;
import org.springframework.boot.mongodb.autoconfigure.MongoAutoConfiguration;
import
org.springframework.boot.mongodb.autoconfigure.PropertiesMongoConnectionDetails;
```

The name of `MongoAutoConfiguration` also changed to `DataMongoAutoConfiguration`. These would be referenced when trying to disable during a non-mongo test.

```
@EnableAutoConfiguration(exclude = {
        MongoAutoConfiguration.class, ①
        DataMongoAutoConfiguration.class ②
})
@Slf4j
class HelloApiContainerIT {
```

① Bootstraps the Mongo Connection

② Registers the `MongoTemplate` client

6.10. MongoProperties / property namespace

The Mongo property prefix moved from `spring.data.mongodb. to spring.mongodb.`.

Spring Boot 3 Mongo URI Expression

```
spring.data.mongodb.uri=mongodb://admin:secret@${ejava-
parent.docker.hostname}:${host.mongodb.port}/test?authSource=admin
```

Spring Boot 4 Mongo URI Expression

```
spring.mongodb.uri=mongodb://admin:secret@${ejava-
parent.docker.hostname}:${host.mongodb.port}/test?authSource=admin
```

Any local `MongoProperties` bound needs to be changed to the new prefix:

Bind Mongo ConfigurationProperties to new Property Prefix

```
@ConfigurationProperties("spring.mongodb")
public class MongoProperties {
```

External overrides supplied via command line / env need to follow:

before

```
if [[ -n "${MONGODB_URI:-}" ]]; then
  OPTIONS="${OPTIONS} --spring.data.mongodb.uri=${MONGODB_URI}"
```

```
fi
```

after

```
if [[ -n "${MONGODB_URI:-}" ]]; then
  OPTIONS="${OPTIONS} --spring.mongodb.uri=${MONGODB_URI}"
fi
```

before

```
@ParameterizedTest
@CsvSource(value={

"spring.data.mongodb.uri=mongodb://admin:secret@localhost:56295/test?authSource=admin,
localhost:56295",

"spring.data.mongodb.uri=mongodb://admin:secret@host.docker.internal:56295/test?authSo
urce=admin,host.docker.internal:56295",
})
```

after

```
@ParameterizedTest
@CsvSource(value={

"spring.mongodb.uri=mongodb://admin:secret@localhost:56295/test?authSource=admin,local
host:56295",

"spring.mongodb.uri=mongodb://admin:secret@host.docker.internal:56295/test?authSource=
admin,host.docker.internal:56295",
})
```

before

```
public static void addLateSpringContextProperties(DynamicPropertyRegistry registry) {
    registry.add("spring.data.mongodb.uri",()-> mongoDB.getReplicaSetUrl(
"testcontainers"));
}
```

after

```
public static void addLateSpringContextProperties(DynamicPropertyRegistry registry) {
    registry.add("spring.mongodb.uri",()-> mongoDB.getReplicaSetUrl("testcontainers")
);
}
```

6.11. Flapdoodle for Spring 4

The Flapdoodle embedded Mongo support has a Spring-4-compatible artifact.

startup error

```
Caused by: java.lang.IllegalStateException: Error processing condition on
de.flapdoodle.embed.mongo.spring.autoconfigure.EmbeddedMongoAutoConfiguration$SyncClie
ntServerWrapperConfig.syncClientServerWrapper
```

The Flapdoodle `spring3x` dependency must be updated to `spring4x`.

Spring Boot 3 Flapdoodle Dependency

```
<flapdoodle.spring3x.version>4.21.0</flapdoodle.spring3x.version> ①
        <version>${flapdoodle.spring3x.version}</version>
...
<dependency> ②
    <groupId>de.flapdoodle.embed</groupId>
    <artifactId>de.flapdoodle.embed.mongo.spring3x</artifactId>
    <scope>test</scope>
</dependency>
```

① dependency management

② child dependency

Spring Boot 4 Flapdoodle Dependency

```
<flapdoodle.spring4x.version>4.24.0</flapdoodle.spring4x.version> ①
        <version>${flapdoodle.spring4x.version}</version>
...
<dependency> ②
    <groupId>de.flapdoodle.embed</groupId>
    <artifactId>de.flapdoodle.embed.mongo.spring4x</artifactId>
    <scope>test</scope>
</dependency>
```

① dependency management

② child dependency

6.12. Mongo Actuator and the `hello` command

The Spring Boot Mongo health indicator switched from the deprecated `isMaster` ping to the Stable-API-compliant `hello` command. Older MongoDB images (pre-4.4.2) reject `hello` and report the application as DOWN.

symptom

```
$ curl http://localhost:56514/actuator/health -v
< HTTP/1.1 503
{"groups":["liveness","readiness"],"status":"DOWN"}
```

```
com.mongodb.MongoCommandException: Command execution failed on MongoDB server with
error 59 (CommandNotFound): 'no such command: 'hello'' on server mongodb:27017. The
full response is {"ok": 0.0, "errmsg": "no such command: 'hello'", "code": 59,
"codeName": "CommandNotFound"}
```

`hello` was introduced in MongoDB 4.4.2. The fix is to bump the image:

Update Mongo Versions to Support Health Probe

```
was: image: mongo:4.4.0-bionic
now: image: mongo:4.4.2-bionic
```

If a server upgrade is not possible, the indicator can be disabled:

Disable Mongo Actuator Health Probe

```
management.health.mongo.enabled=false
```

6.13. ServletWebServerFactory

The Tomcat `ServletWebServerFactory` types moved out of `org.springframework.boot.web.embedded.tomcat` / `org.springframework.boot.web.servlet.server` into more focused packages. These calls are necessary to redirect clients from one port to another, normally for the reason of shifting from HTTP to HTTPS.

compile error

```
[ERROR] ...package org.springframework.boot.web.embedded.tomcat does not exist
[ERROR] ...package org.springframework.boot.web.servlet.server does not exist
[ERROR] ...cannot find symbol
[ERROR]   symbol:   class ServletWebServerFactory
```

context

```
@Bean
@Profile("redirect") //only enable on demand
public ServletWebServerFactory servletContainer() {
    TomcatServletWebServerFactory tomcat = new TomcatServletWebServerFactory() {
...
    tomcat.addAdditionalTomcatConnectors(redirectConnector());
```

Spring Boot 3 Web Server Java Class Packages

```
import org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory;
import org.springframework.boot.web.servlet.server.ServletWebServerFactory;

tomcat.addAdditionalTomcatConnectors(redirectConnector());
```

The add connector method also lost the `Tomcat`-prefixed method name.

Spring Boot 4 Web Server Java Class Packages

```
import org.springframework.boot.tomcat.servlet.TomcatServletWebServerFactory;
import org.springframework.boot.web.server.servlet.ServletWebServerFactory;

tomcat.addAdditionalConnectors(redirectConnector());
```

6.14. Docker Maven Plugin / Docker Desktop Error

With an updated Docker Desktop (Engine 29.4.0) and the legacy `io.fabric8:docker-maven-plugin` (0.44.0), the build would fail when the plugin encounters an error calling Docker. The following snippet shows that error.

Error

```
Execution start-container of goal io.fabric8:docker-maven-plugin:0.44.0:start failed:
Cannot invoke "com.google.gson.JsonElement.isJsonNull()" because the return value of
"com.google.gson.JsonObject.get(String)" is null
```

The source of the problem was that Docker Desktop removed a property and the response was a null versus a JsonElement with a null.

There were various fixes, but the easiest and long term fix was to advance the version of the plugin in `ejava-build`.

before

```
<docker-maven-plugin.version>0.44.0</docker-maven-plugin.version>
```

after

```
<docker-maven-plugin.version>0.48.1</docker-maven-plugin.version>
```

6.15. RestTemplate / RestClient / WebClient / Test Client

`spring-boot-starter-web` (now `-webmvc`) no longer transitively brings in `RestTemplateBuilder`, `RestClient.Builder`, and `WebClient.Builder`. `TestRestTemplate` has moved to a dedicated artifact. The mapping below shows which starter to add when the corresponding type is needed.

Type	New Spring Boot 4 dependency
`RestTemplate`, `RestTemplateBuilder`	`spring-boot-starter-restclient`
`WebClient`, `WebClient.Builder`	`spring-boot-starter-webclient`
`TestRestTemplate`	`spring-boot-starter-resttestclient` (test scope)

Spring 6 also introduced `RestClient` enhancements and additional `@HttpExchange` interface-binding options. Those are covered in the API lecture; this document only addresses the import / dependency changes needed to compile.

The following two dependency trees identifies where each client and client builder exists in Spring Boot 3 and 4.

Spring Boot 3 RestTemplate, RestClient, and WebClient Dependencies

```
-spring-boot-starter-web         ← name changed
 +- spring-boot-starter
 |  +- spring-boot-autoconfigure  ①
-|  +- [ RestTemplateBuilder, RestClient.Builder, WebClient.Builder auto-configs ]
 |  \- spring-boot-starter-logging
-+- spring-boot-starter-json     ← Jackson 2
 +- spring-boot-starter-tomcat
 +- spring-web
 +- [ RestTemplate, RestClient, RestClient.Builder classes here ]
 \- spring-webmvc

-spring-boot-starter-webflux     ← required for WebClient (full Reactive server) ②
 \- spring-webflux
  +- [ WebClient, WebClient.Builder classes here ]
- \- spring-boot-starter-json     ← Jackson 2
```

① client builder auto-configs removed from spring-boot-autoconfigure

② no longer need full Reactive server dependency to obtain WebClient client classes

Spring Boot 4 RestTemplate, RestClient, and WebClient Dependencies

```
+spring-boot-starter-webmvc         ← renamed
 +- spring-boot-starter
-|  +- spring-boot-autoconfigure ① ②
-|  +- [RestTemplateBuilder, Rest/WebClient.Builders auto-configs REMOVED]
 |  \- spring-boot-starter-logging
 +- spring-boot-starter-jackson     ← Jackson 3
 +- spring-boot-starter-tomcat
++- spring-boot-http-converter
 |  +- spring-boot
 |  \- spring-web
     \- [ RestTemplate, RestClient, RestClient.Builder classes STILL here ]
 \- spring-boot-webmvc
    \- spring-webmvc
```

```
+spring-boot-starter-restclient
++- spring-boot-starter-jackson     ← Jackson 3
+\- spring-boot-restclient ①
+ +- [RestTemplateBuilder, RestClient.Builder auto-configs MOVED here]
+ \- spring-web

+spring-boot-starter-webclient
++- spring-boot-starter-jackson     ← Jackson 3
+\- spring-boot-webclient       ← WebClient.Builder auto-config MOVED here ②
  \- spring-webflux             ← WebClient, WebClient.Builder classes STILL here
```

① New `spring-boot-restclient` now hosts non-reactive auto-configs

② New `spring-boot-webclient` now hosts reactive auto-config

The net result clients:

- Non-reactive:
  - If you want just the `RestTemplate`/`RestTemplateBuilder` or `RestClient`/`RestClient.Builder` classes only, continue to add a dependency on `spring-web`.
  - If you want the `RestTemplateBuilder` or `RestClient.Builder` auto-configs, add a dependency on the new `spring-boot-restclient`. These no longer come in automatically from `spring-boot-autoconfigure`.
  - If you would like Jackson and client dependencies, add a dependency on `spring-boot-starter-restclient`. These all (some do) no longer come in with the new `spring-boot-starter-webmvc`.
- Reactive clients:
  - If you want just the `WebClient`/`WebClient.Builder` classes only, continue to add a dependency on `spring-webflux`.
  - If you want the `WebClient.Builder` auto-configs, add a dependency on the new `spring-boot-webclient`.
  - If you would like Jackson and Reactive client dependencies, add a dependency on `spring-boot-starter-webclient`.

6.16. junit-vintage-engine no longer transitive

`spring-boot-starter-test` no longer pulls in `junit-vintage-engine` (the JUnit 4 compatibility layer). Modules that still rely on JUnit 4 must add the dependency directly. The Ejava examples are primarily JUnit 5-only, with a sample of JUnit 4 in the testing section for historical coverage reasons. In general, excludes are no longer needed. If you do encounter JUnit 4 classes that are meant to be kept, an explicit dependency is required.

Explicit JUnit 4 Dependencies (likely not needed)

```
<!-- for legacy JUnit 4 examples -->
<dependency>
```

```
    <groupId>org.junit.vintage</groupId>
    <artifactId>junit-vintage-engine</artifactId>
    <scope>test</scope>
</dependency>
```

6.17. JUnit 6 and IntelliJ

JUnit Platform 6 ships with Spring Boot 4. Older IntelliJ builds dispatch to a bundled JUnit Platform that does not match the JUnit 6 selectors:

```
Exception in thread "main" java.lang.NoSuchMethodError: 'java.lang.String
org.junit.platform.engine.discovery.MethodSelector.getMethodParameterTypes()'
```

I found the error using the following Community Editions.

Affected

```
IntelliJ IDEA 2025.1.3 (Community Edition)
Build #IC-251.26927.53, built on June 23, 2025

IntelliJ IDEA 2025.1.7 -- still broken
```

Updating to the most current overall edition (in Community Mode) solved the issue.

Resolved

```
IntelliJ IDEA 2025.3.3
```

JetBrains issue tracker

6.17.1. Testcontainers Module Renames

Testcontainers 2.x reorganized its module artifacts under a `testcontainers-*` prefix. The Spring Boot 4 BOM now imports `testcontainers-bom` 2.x, so child modules just need to update their artifactIds.

Spring Boot 3.x artifactId	Spring Boot 4.x artifactId
`org.testcontainers:junit-jupiter`	`org.testcontainers:testcontainers-junit-jupiter`
`org.testcontainers:postgresql`	`org.testcontainers:testcontainers-postgresql`
`org.testcontainers:mongodb`	`org.testcontainers:testcontainers-mongodb`
`org.testcontainers:spock`	`org.testcontainers:testcontainers-spock`

- Testcontainers 2.0.3 BOM

6.18. Code / API Changes

6.18.1. MultiValueMap construction

Spring 7 separated the `HttpHeaders` class from the `MultiValueMap<String, String>` interface it used to implement. Code that constructed a `MultiValueMap` by `new HttpHeaders<>()` no longer compiles.

compile error

```
PageableDTO.java:[115,53] incompatible types: org.springframework.http.HttpHeaders
cannot be converted to
org.springframework.util.MultiValueMap<java.lang.String,java.lang.String>
```

before

```
public MultiValueMap<String, String> getQueryParams() {
    MultiValueMap<String, String> queryParams = new HttpHeaders<>();
    ...
```

after

```
public MultiValueMap<String, String> getQueryParams() {
    MultiValueMap<String, String> queryParams = new LinkedMultiValueMap<>();
    ...
```

Where `addQueryParams()` previously iterated `getQueryParams().toSingleValueMap().entrySet()` to push each entry onto a `UriBuilder`, the call can now be a single `uriBuilder.queryParams(getQueryParams())`.

before

```
public UriBuilder addQueryParams(UriBuilder uriBuilder) {
    uriBuilder.queryParams(getQueryParams());
    for (Map.Entry<String, String> entry:getQueryParams().toSingleValueMap().entrySet
()) {
        uriBuilder.queryParam(entry.getKey(), entry.getValue());
    }
    return uriBuilder;
}
```

after

```
public UriBuilder addQueryParams(UriBuilder uriBuilder) {
    return uriBuilder.queryParams(getQueryParams());
}
```

6.18.2. RestTemplateBuilder

`RestTemplateBuilder` moved out of `org.springframework.boot.web.client` and into `org.springframework.boot.restclient`, alongside the new `spring-boot-starter-restclient`.

compile error

```
[ERROR] .../common/ejava-web-
util/src/main/java/info/ejava/examples/common/web/RestTemplateConfig.java:[3,43]
package org.springframework.boot.web.client does not exist
[ERROR] .../common/ejava-web-
util/src/main/java/info/ejava/examples/common/web/RestTemplateConfig.java:[15,43]
cannot find symbol
[ERROR]   symbol:   class RestTemplateBuilder
```

add the starter

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-restclient</artifactId>
</dependency>
```

update the import

```
//import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.boot.restclient.RestTemplateBuilder;
```

6.18.3. MockBean / MockitoBean

`@MockBean` was deprecated in Spring Boot 3 and removed in Spring Boot 4. The replacement is `@MockitoBean` from `org.springframework.test.context.bean.override.mockito`.

before

```
import org.springframework.boot.test.mock.mockito.MockBean;
...
    @MockBean
    private GraderRepository graderRepositoryMock;
```

after

```
import org.springframework.test.context.bean.override.mockito.MockitoBean;
...
    @MockitoBean
    private GraderRepository graderRepositoryMock;
```

6.18.4. WebClient.Builder

In Spring Boot 4, `WebClient.Builder` is no longer auto-configured by `spring-boot-starter-webmvc`. Code that injected `WebClient.Builder` will fail at startup until a starter or webclient dependency is added.

startup error

```
Unsatisfied dependency expressed through method 'webClient' parameter 0: No qualifying
bean of type 'org.springframework.web.reactive.function.client.WebClient$Builder'
available: expected at least 1 bean which qualifies as autowire candidate.
```

@Bean factory remains the same

```
@Bean
public WebClient webClient(WebClient.Builder builder) {
    return builder.build();
    //or just the following will work in the simple cases like this
    //return WebClient.builder().build();
}
```

The legacy `spring-boot-starter-webflux` was previously a heavyweight way to bring in `WebClient`. For test-only use, the lighter `spring-boot-webclient` artifact is enough:

remove

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

add

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-webclient</artifactId>
    <scope>test</scope>
</dependency>
```

For non-test usage, add `spring-boot-starter-webclient` instead:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webclient</artifactId>
</dependency>
```

6.18.5. ex.getRawStatusCode()

`RestClientResponseException.getRawStatusCode()` has been removed. Use `getStatusCode()` and compare against an `HttpStatusCode` directly.

compile error

```
[ERROR] .../svc/svc-api/httpapi-gestures-example/httpapi-gestures-
svc/src/test/java/info/ejava/examples/svc/httpapi/gestures/GesturesRestTemplateClientN
Test.java:[79,16] cannot find symbol
[ERROR]   symbol:   method getRawStatusCode()
```

before

```
RestClientResponseException ex = assertThrows(RestClientResponseException.class,
        ()->gesturesClient.getGesture("unknown", null));

then(ex.getRawStatusCode()).isEqualTo(HttpStatus.NOT_FOUND.value());
```

after

```
then(ex.getStatusCode()).isEqualTo(HttpStatus.NOT_FOUND);
```

6.18.6. AutoConfigureMockMvc

`@AutoConfigureMockMvc` and the `MockMvc` test slice support moved from `org.springframework.boot.test.autoconfigure.web.servlet` to `org.springframework.boot.webmvc.test.autoconfigure`, and now require an explicit dependency.

compile error

```
[ERROR] .../svc/svc-api/httpapi-gestures-example/httpapi-gestures-
svc/src/test/java/info/ejava/examples/svc/httpapi/gestures/GesturesMockMvcNTest.java:[
11,63] package org.springframework.boot.test.autoconfigure.web.servlet does not exist
```

add the dependency

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-webmvc-test</artifactId>
    <scope>test</scope>
</dependency>
```

update the import

```
//import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.webmvc.test.autoconfigure.AutoConfigureMockMvc;
```

Spring 7 changed when `@WithMockUser` is applied during MockMvc requests. Annotations declared on the test method are now overwritten by request-level post-processors:

```
@Test
@WithMockUser("user1") //useless, set early, gets overwritten by request in Spring 7
void can_be_accessed_authenticated() throws Exception {
    mockMvc.perform(get(authnHelloURI)
                    .with(SecurityMockMvcRequestPostProcessors.user("user1")) //needed
                    .queryParam("name", "jim"))
            .andExpect(status().isOk())
            .andExpect(content().string("hello, jim :caller=user1"))
            .andDo(MockMvcResultHandlers.print());
}
```

6.18.7. TestRestTemplate

`TestRestTemplate` moved from `org.springframework.boot.test.web.client` to `org.springframework.boot.resttestclient`, requires a new dependency, and now requires explicit `@AutoConfigureTestRestTemplate` to inject the bean.

compile error

```
[ERROR] ...cannot find symbol
[ERROR]   symbol:   class TestRestTemplate
```

add the dependency

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-resttestclient</artifactId>
    <scope>test</scope>
</dependency>
```

update the import

```
//import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.boot.resttestclient.TestRestTemplate;
```

Without `@AutoConfigureTestRestTemplate`, autowiring will now fail:

```
UnsatisfiedDependencyException: ...No qualifying bean of type
'org.springframework.boot.resttestclient.TestRestTemplate' available: expected at
least 1 bean which qualifies as autowire candidate.
```

add the annotation

```
import
org.springframework.boot.resttestclient.autoconfigure.AutoConfigureTestRestTemplate;
...
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@AutoConfigureTestRestTemplate
public class AuthzTestRestTemplateNTest {
    @Autowired
    private TestRestTemplate client; //automatically tracks @LocalPort
```

6.18.8. UriComponentsBuilder.fromHttpUrl()

`UriComponentsBuilder.fromHttpUrl(String)` has been removed in favor of `fromUriString(String)`, which works for both http(s) and other schemes.

compile error

```
[ERROR] .../JacksonLeniencyApplicationTests.java:[50,47] cannot find symbol
[ERROR]   symbol:   method fromHttpUrl(java.lang.String)
[ERROR]   location: class org.springframework.web.util.UriComponentsBuilder
```

before

```
URI url = UriComponentsBuilder.fromHttpUrl(baseUrl).path("api/dates").build().toUri();
```

after

```
URI url = UriComponentsBuilder.fromUriString(baseUrl).path("api/dates").build().toUri
();
```

6.19. Security

6.19.1. http.csrf() no-arg removed

The no-arg `http.csrf()` builder form is gone; the lambda-style `csrf(cfg → cfg.disable())` is the only supported entry point.

compile error

```
HttpSecurity cannot be applied to given types;
[ERROR]   required:
org.springframework.security.config.Customizer<org.springframework.security.config.ann
otation.web.configurers.CsrfConfigurer<org.springframework.security.config.annotation.
web.builders.HttpSecurity>>
[ERROR]   found:    no arguments
[ERROR]   reason: actual and formal argument lists differ in length
```

before

```
http.csrf().disable();
```

after

```
http.csrf(cfg -> cfg.disable());
```

6.20. Persistence / JPA

6.20.1. EntityScan

`@EntityScan` moved from `org.springframework.boot.autoconfigure.domain` to `org.springframework.boot.persistence.autoconfigure`.

compile error

```
[ERROR] ...package org.springframework.boot.autoconfigure.domain does not exist
[ERROR] ...cannot find symbol
[ERROR]   symbol: class EntityScan
```

before

```
import org.springframework.boot.autoconfigure.domain.EntityScan;
```

after

```
import org.springframework.boot.persistence.autoconfigure.EntityScan;
```

6.20.2. Flyway as a starter

Direct `flyway-core` dependency no longer pulls in the schema-management auto-configuration in a way that integrates cleanly with the JPA `entityManagerFactory` lifecycle. Switch to `spring-boot-starter-flyway`, which configures Flyway to run before Hibernate's schema validation.

startup error

```
Caused by: jakarta.persistence.PersistenceException: Unable to build Hibernate
SessionFactory  [persistence unit: default] ; nested exception is
org.hibernate.tool.schema.spi.SchemaManagementException: Schema validation: missing
table [vote]
```

before

```
<dependency>
    <groupId>org.flywaydb</groupId>
```

```
    <artifactId>flyway-core</artifactId>
    <scope>runtime</scope>
</dependency>
```

after

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-flyway</artifactId>
    <scope>runtime</scope>
</dependency>
```

6.21. Integration Testing / Docker

6.21.1. ComposeContainer.withLocalCompose removed

Testcontainers 2.x removed `ComposeContainer.withLocalCompose(boolean)`. The local-compose code path is now selected automatically based on whether the host has a `docker-compose` binary on `PATH`.

compile error

```
[ERROR] ...cannot find symbol
[ERROR]   symbol:   method withLocalCompose(boolean)
[ERROR]   location: class org.testcontainers.containers.ComposeContainer
```

before

```
return new ComposeContainer("testcontainers-ntest",
...
    .withLocalCompose(true) //false works when host mounts files, true works when not
mounted
```

Drop the `.withLocalCompose(⋯)` line; behaviour is now picked automatically.

6.22. Spring MVC `@Validated`

The Spring `@Validated` annotation is no longer required to trigger Jakarta validation in MVC controllers — Spring 7 activates Jakarta validation automatically when constraint annotations are present. Existing `@Validated` annotations remain harmless and continue to be honored when supplying validation groups.

The way I noticed: an example was set up to compare validation-on vs. validation-off using two profile-selected `@Bean`s with the same base controller class. Without `@Validated` on the class, Spring 7 still triggered validation and threw `MethodArgumentNotValidException` — where Spring 6 would have skipped it.

Jakarta constraint without @Validated

```
@RestController
//no @Validated annotation here
public class ContactsController extends ContactsController {
    @RequestMapping(path= CONTACT_PATH,
            method=RequestMethod.GET,
            produces={MediaType.APPLICATION_JSON_VALUE, MediaType
.APPLICATION_XML_VALUE})
        public ResponseEntity<PersonPocDTO> createPOC(
                @Validated(PocValidationGroups.CreatePlusDefault.class)
                @RequestBody PersonPocDTO personDTO) {
            return super.createPOC(personDTO);
        }
```

To explicitly turn validation OFF in a controller (the new "non-default" case), use `@Validated` with a marker group that does not match any constraint:

```
static interface NoValidation {}

@Validated(NoValidation.class) //turning off default activation
public class NonValidatingContactsController extends ContactsController {
    public ResponseEntity<PersonPocDTO> createPOC(
        @Validated(NoValidation.class) //overriding parent with non-existent group
        PersonPocDTO personDTO) {
            return super.createPOC(personDTO);
        }
```

Chapter 7. Summary

During the Spring Boot 3 to Spring Boot 4 port of the class examples, we covered:

- **Dependency changes**
  - name changes
    - `spring-boot-starter-web` → `spring-boot-starter-webmvc`
    - `spring-boot-starter-json` → `spring-boot-starter-jackson`
  - refactoring and new dependencies
    - `spring-boot-starter-restclient`, `spring-boot-starter-webclient` are client-side focused
    - test clients and test slices extracted from `spring-boot-starter-test`
      - `spring-boot-resttestclient` — `TestRestTemplate`
      - `spring-boot-webtestclient` — `WebTestClient`
      - `spring-boot-starter-webmvc-test` — `@AutoConfigureMockMvc` and MockMvc slice
      - `spring-boot-starter-security-test` — `@WithMockUser` and security test support
- **Spring Web API changes**
  - `spring-boot-starter-webflux` is server-side focused
- **Spring Security changes**
  - `AntPathRequestMatcher` removed → `PathPatternRequestMatcher`.
  - `MvcRequestMatcher` / `HandlerMappingIntrospector` removed.
- **Persistence tier changes**
  - Property namespace: `spring.data.mongodb.` → `spring.mongodb.`
  - Flapdoodle: `spring3x` → `spring4x` artifact rename
  - `de.flapdoodle.embed.mongo.spring3x` → `de.flapdoodle.embed.mongo.spring4x`
  - Flyway: declare `spring-boot-starter-flyway` instead of bare `flyway-core`.
- **Jackson 3 changes**
  - Jackson 3 (`tools.jackson.core`) replaces Jackson 2 (`com.fasterxml.jackson.core`). A `jackson-annotations 2.x` compat shim remains, so existing import statements still compile — but the active mapper is Jackson 3.
  - `WRITE_DATES_AS_TIMESTAMPS` defaults to **off**; date fields now serialize as ISO strings unless explicitly reconfigured.
  - `jackson-datatype-jsr310`, `jackson-datatype-jdk8`, and `jackson-module-parameter-names` are gone — their functionality is built into Jackson 3 core.
  - `Jackson2ObjectMapperBuilderCustomizer` → `JsonMapperBuilderCustomizer`.
- **API renames**
  - `UnprocessableEntity` → `UnprocessableContent`
  - `getRawStatusCode()` removed. Use `getStatusCode().value()` or `getStatusCode()` and compare

with `HttpStatus` enum

- `UriComponentsBuilder.fromHttpUrl` removed
- `@Validated` behavior change on Spring MVC controllers

- **Other**
  - `@MockBean` / `@SpyBean` removed → `@MockitoBean` / `@MockitoSpyBean`.

For the changes, I tried to make sure you were well aware what

- Spring Boot 3 constructs existed and where they went
- Spring Boot 4 constructs existed and where they came from

The recommended porting sequence: run the OpenRewrite Spring Boot 4 recipe, commit the result, then walk through remaining build errors using this guide.