s

The detailed summary of the migration from Spring Boot 2.x with Java 8 to Spring Boot 4.0.2 with Java 17 or 21.
When upgrading from Spring Boot 2.x to Spring Boot 3.x or 4.0.2 generation + Java 21, the following areas must be handled:

1. Java version upgrade
2. javax package to jakarta package
3. Hibernate 5 → Hibernate 6
4. Spring Security configuration changes
5. CsrfTokenRequestAttributeHandler: Spring Boot 3.x vs 2.x
6. Swagger replacement
7. Removed deprecated APIs
8. Third-party dependency compatibility
9. All native JOIN queries were reviewed during migration to prevent duplicate alias conflicts
10. ResponseEntityExceptionHandler override method signatures changed
11. Hibernate dialect changes between Spring Boot versions

Upgrade JDK 1.8 to JDK 21 Changes

1 Update Maven/Gradle (Pom.xml file changes)

Java 8 + Spring Boot 2.x	Java 17 or Java 21 + Spring Boot 3.x/4.x
<properties> <java.version>1.8</java.version> </properties>	<properties> <java.version>21</java.version> </properties>
<dependency> <groupId>javax.servlet</groupId> <artifactId>javax.servlet-api</artifactId> <scope>provided</scope> </dependency>	<dependency> <groupId>jakarta.servlet</groupId> <artifactId>jakarta.servlet-api</artifactId> <scope>provided</scope> </dependency>

---

2 Code Changes

This is one of the most important changes when upgrading from Java 8 + Spring Boot 2.x to Java 17 or Java 21 + Spring Boot 3.x/4.x

2.1) You must change:

javax.validation.* package to→ **jakarta.validation.*** package

s

Java 8 + Spring Boot 2.x	Java 17 or Java 21 + Spring Boot 3.x/4.x

import javax.validation.*
 	
import javax.validation.constraints.NotBlank;
public class User {
@NotBlank(message =”userId must not be blank”)
private String userId;
//setters and getters
}
	import jakarta.validation.*
	
	import jakarta.validation.constraints.NotBlank;
	public class User {
	@NotBlank(message =”userId must not be blank”)
	private String userId;
	//setters and getters
	}

2.2) You must change:

javax.persistence.* → jakarta.persistence.*

Java 8 + Spring Boot 2.x	Java 17 or Java 21 + Spring Boot 3.x/4.x

javax.persistence.*
 	
import javax.persistence.Entity;
import javax.persistence.Table;
import javax.persistence.Id;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
 	
@Entity
@Table(name = "users")
public class Users {
 	
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
//setter and getters
}
	Jakarta.persistence.*
	
	import jakarta.persistence.Entity;
	import jakarta.persistence.Table;
	import jakarta.persistence.Id;
	import jakarta.persistence.GeneratedValue;
	import jakarta.persistence.GenerationType;
	
	@Entity
	@Table(name = "users")
	public class Users {
	
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	//setter and getters
	}

Annotation	New (jakarta)
@Entity	jakarta.persistence.Entity
@Table	jakarta.persistence.Table
@Id	jakarta.persistence.Id

@Column	jakarta.persistence.Column
@OneToMany	jakarta.persistence.OneToMany
@ManyToOne	jakarta.persistence.ManyToOne
@JoinColumn	jakarta.persistence.JoinColumn
@Transient	jakarta.persistence.Transient
@Enumerated	jakarta.persistence.Enumerated

2.3) Security Configuration Migration Code Changes

Area	Java 8 + Spring Boot 2.x	Java 17 or Java 21 + Spring Boot 3.x/4.x
Authorize method	authorizeRequests()	authorizeHttpRequests()
URL matchers	antMatchers()	requestMatchers()
CSRF ignoring	ignoringAntMatchers()	ignoringRequestMatchers()
Headers config	.headers().xssProtection()	Lambda style config
Chaining style	.and() chaining	Lambda DSL
Security Adapter	Extended class earlier	SecurityFilterChain bean only

Java 8 + Spring Boot 2.x (OLD)

```
package com.example.security;

import java.util.Arrays;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.security.web.header.writers.StaticHeadersWriter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled =true)
public class SecurityConfig {
```

```
 @Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
      CustomCSRFRepository csrfTokenRepository = CustomCSRFRepository.withHttpOnlyFalse();
      repo.setSecure(true);
      repo.setCookiePath("/");

  http
         .headers()
            .xssProtection().disable()
            .contentSecurityPolicy("script-src 'self'")
            .and()
            .addHeaderWriter(new StaticHeadersWriter(
                  "Expect-CT", "max-age=31536000, enforce"))
         .and()

         .csrf()
            .csrfTokenRepository(csrfTokenRepository)
            .ignoringAntMatchers("/V1/token")
         .and()

         .cors()
            .configurationSource(corsConfigurationSource())
         .and()

         .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
         .and()

         .authorizeRequests()
            .antMatchers("/V1 /token").permitAll()
            .antMatchers("/role").authenticated()
            .antMatchers("/country/**").hasAuthority("/V1/businessusers/**")
            .antMatchers("/nonbusinessadmin/**").hasAnyAuthority("/V1/admin/role/**")
            .anyRequest().authenticated();

      http.addFilterBefore(jwtFilter(),
            UsernamePasswordAuthenticationFilter.class);
  }


  //  CORS Configuration
  @Bean
  public CorsConfigurationSource corsConfigurationSource() {

      CorsConfiguration configuration = new CorsConfiguration();
      configuration.setAllowedOrigins(Arrays.asList("*"));
      configuration.setAllowedMethods(
            Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS"));
```

```
        configuration.setAllowedHeaders(Arrays.asList("*"));
        configuration.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source =
                new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);

        return source;
    }

    //  JWT Filter Bean
    @Bean
    public JwtFilter jwtFilter() {
        return new JwtFilter();
    }
}
```

Java 17 or Java 21 + Spring Boot 3.x/4.x (New)

```
package com.example.security;
import java.util.List;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.security.web.header.writers.StaticHeadersWriter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true, securedEnabled = true)
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        CsrfTokenRequestAttributeHandler csrfHandler = new CsrfTokenRequestAttributeHandler();
 http
            .headers(headers -> headers
                .xssProtection(xss -> xss.disable())
                .contentSecurityPolicy(csp ->
```

```
            csp.policyDirectives("script-src 'self'"))
        .addHeaderWriter(new StaticHeadersWriter(
            "Expect-CT", "max-age=31536000, enforce"))
    )

    .csrf(csrf -> csrf
        .csrfTokenRequestHandler(csrfHandler).csrfTokenRepository(csrfTokenRepository)
        .ignoringRequestMatchers("/V1/token")
    )

    .cors(cors -> cors
        .configurationSource(corsConfigurationSource())
    )

    .sessionManagement(session ->
        session.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
    )

    .authorizeHttpRequests(auth -> auth
        .requestMatchers("/V1 /token").permitAll()
        .requestMatchers("/role").authenticated()
        .requestMatchers("/country/**").hasAuthority("/V1/businessusers/**")
        .requestMatchers("/nonbusinessadmin/**")
            .hasAnyAuthority("/V1/admin/role/**")
        .anyRequest().authenticated()
    )

    .addFilterBefore(jwtFilter(),
        UsernamePasswordAuthenticationFilter.class);

  return http.build();
}

// CSRF Repository Bean
@Bean
public CustomCSRFRepository csrfTokenRepository() {
  CustomCSRFRepository repo =
      CustomCSRFRepository.withHttpOnlyFalse();
  repo.setSecure(true);
  repo.setCookiePath("/");
  return repo;
}

// CORS Configuration
@Bean
public CorsConfigurationSource corsConfigurationSource() {

  CorsConfiguration configuration = new CorsConfiguration();
```

```
    configuration.setAllowedOrigins(List.of("*"));
    configuration.setAllowedMethods(
            List.of("GET", "POST", "PUT", "DELETE", "OPTIONS"));
    configuration.setAllowedHeaders(List.of("*"));
    configuration.setAllowCredentials(true);

    UrlBasedCorsConfigurationSource source =
            new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", configuration);

    return source;
}
```

CsrfTokenRequestAttributeHandler: Spring Boot 3.x vs 2.x

- **Spring Boot 2.x + Spring Security 5 (Java 8)**
    - CSRF tokens were automatically exposed via _csrf request attribute.
    - Templates (e.g., Thymeleaf) could directly use ${_csrf.token}.
    - No extra configuration was needed; Spring Security handled token exposure.
- **Spring Boot 3.x / 4.x + Spring Security 6 (Java 17/21)**
    - CSRF handling was refactored; tokens are **not automatically exposed**.
    - CsrfTokenRequestAttributeHandler is required to map the token to request attributes:
- CsrfTokenRequestAttributeHandler csrfHandler = new CsrfTokenRequestAttributeHandler();

```
.csrf(csrf -> csrf
    .csrfTokenRequestHandler(csrfHandler).csrfTokenRepository(csrfTokenRepository)
    .ignoringRequestMatchers("/V1/token")
)
```

    - Explicit token exposure improves control and security.
    - Ensures compatibility with REST APIs, Thymeleaf forms, and AJAX requests.
    - Aligns with the Lambda-based configuration style in Spring Security 6.

2.4) As part of the migration from Java 8 + Spring Boot 2.x to Java 21 + Spring Boot 3.x/4.x, we encountered an issue related to **native queries with JOIN operations.**

Issue Description

Java 8 + Spring Boot 2.x (Working)

```
@Query(value = "SELECT * FROM loan.orders o LEFT JOIN loan.customers c
ON o.customer_id = c.customer_id  WHERE o.status = :status", nativeQuery = true)
List<Order> findOrders(@Param("status") String status);
```

This query worked correctly in Spring Boot 2.x (Hibernate 5).

Issue in Java 21 + Spring Boot 3.x / 4.x

The same query now throws the following error:

Error: Encountered a duplicate SQL alias 'customer_id' during auto-discovery

Why This Happens

Both tables contain the same column:

- o.customer_id
- c.customer_id

When using SELECT *, Hibernate tries to automatically map the result set to the Order entity.

However, since both tables return a column named customer_id, Hibernate 6 detects duplicate aliases and cannot determine which column belongs to the mapped entity.

Hibernate 6 (used in Spring Boot 3+) is stricter than Hibernate 5.
That's why the query worked earlier but now fails.

Root Cause

The result set contains two customer_id columns:

o.customer_id

c.customer_id

This causes:

Duplicate SQL alias → Mapping ambiguity → Runtime failure

Correct Solution

Never use SELECT * in a native query with JOIN.

Instead, explicitly select only the columns of the entity you are mapping.

If the return type is List<Order>, use:

```
@Query(value = "SELECT o.* FROM loan.orders o LEFT JOIN loan.customers c
ON o.customer_id = c.customer_id  WHERE o.status = :status", nativeQuery = true)
```

```
List<Order> findOrders(@Param("status") String status);
```

Why This Fix Works

- Hibernate now maps only o (orders) columns
- No duplicate alias
- Clean and safe
- Fully compatible with Hibernate 6

Please ensure all native queries with JOINs are reviewed during migration to avoid similar alias conflicts.

2.5) Exception Handler

When moving Java 8 + Spring Boot to Java 21 + Spring Boot / 4.x

the **ResponseEntityExceptionHandler override method signatures changed**.

What Changed in Spring Boot 3 / 4?

HttpStatus → HttpStatusCode

ResponseEntityExceptionHandler Override methods:

1 handleMethodArgumentNotValid

Spring Boot 2.x (OLD)

```
@Override
protected ResponseEntity<Object> handleMethodArgumentNotValid(
        MethodArgumentNotValidException ex, HttpHeaders headers,
        HttpStatus status,  WebRequest request)
```

Spring Boot 3.x / 4.x (NEW)

```
@Override
protected ResponseEntity<Object> handleMethodArgumentNotValid(
        MethodArgumentNotValidException ex, HttpHeaders headers,
        HttpStatusCode status, WebRequest request)
```

---

s

2 handleHttpMessageNotReadable

Spring Boot 2.x

```
@Override
protected ResponseEntity<Object> handleHttpMessageNotReadable(
        HttpMessageNotReadableException ex,HttpHeaders headers,
        HttpStatus status, WebRequest request)
```

Spring Boot 3.x / 4.x (NEW)

```
@Override
protected ResponseEntity<Object> handleHttpMessageNotReadable(
        HttpMessageNotReadableException ex, HttpHeaders headers,
        HttpStatusCode status, WebRequest request)
```

3 handleMissingServletRequestParameter

OLD

```
@Override
protected ResponseEntity<Object> handleMissingServletRequestParameter(
        MissingServletRequestParameterException ex,  HttpHeaders headers,
        HttpStatus status, WebRequest request)
```

NEW

```
@Override
protected ResponseEntity<Object> handleMissingServletRequestParameter(
        MissingServletRequestParameterException ex, HttpHeaders headers,
        HttpStatusCode status,  WebRequest request)
```

Why This Changed?

Spring Framework 6 introduced: **HttpStatusCode**Instead of tightly coupling everything to HttpStatus enum.This gives more flexibility for custom status codes.

Fully Correct Boot 3 / 4 Example

```
@RestControllerAdvice
public class GlobalExceptionHandler extends ResponseEntityExceptionHandler {
  @Override
   protected ResponseEntity<Object> handleMethodArgumentNotValid(
         MethodArgumentNotValidException ex, HttpHeaders headers,
        HttpStatusCode status, WebRequest request) {
      Map<String, String> errors = new HashMap<>();
      ex.getBindingResult().getFieldErrors()
            .forEach(error ->   errors.put(error.getField(),   error.getDefaultMessage())   );
      return ResponseEntity.status(status).body(errors);
   }
}
```

2.6) Using @GeneratedValue(strategy = **GenerationType.AUTO**) without an IDENTITY column or SEQUENCE works in Spring Boot 2.x because Hibernate 5 often defaulted to IDENTITY for SQL Server.
In Spring Boot 3.x / 4.x, Hibernate 6 prefers SEQUENCE if the database supports it.
Since SQL Server (2012+) supports SEQUENCE, Hibernate now attempts to use one.
But no SEQUENCE exists in the DB and the column is not IDENTITY.
So Hibernate tries select next value for hibernate_sequence and the insert fails.
Fix: Explicitly use GenerationType.IDENTITY with an IDENTITY column or define a proper SEQUENCE and avoid AUTO in Boot 3/4.

2.7) **Hibernate dialect changes between Spring Boot versions.**

Spring Boot 2.x uses **Hibernate 5.x**.

Hibernate 5 supported the following dialect configurations in application.properties:

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.SQLServerDialect

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.SQLServer2012Dialect

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.SQLServer2008Dialect

Therefore, in Spring Boot 2.x:

👉 SQLServer2012Dialect works perfectly fine.

Spring Boot 3.x and 4.x

Spring Boot 3.x and 4.x use:

• Hibernate 6

Hibernate 6 removed version-specific dialect classes such as:

❌ SQLServer2012Dialect
❌ SQLServer2008Dialect

Hibernate 6 simplified the dialect strategy. Now the correct configuration is:

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.SQLServerDialect

Error Observed When Using Old Dialect in Boot 3.x / 4.x

If SQLServer2012Dialect is still configured, the following error occurs:

```
WARN: HHH1000046: Could not obtain connection to query JDBC database metadata

org.hibernate.boot.registry.selector.spi.StrategySelectionException:

Unable to resolve name [org.hibernate.dialect.SQLServer2012Dialect]

as strategy [org.hibernate.dialect.Dialect]

at org.hibernate.boot.registry.selector.internal.StrategySelectorImpl.selectStrategyImplementor

at org.hibernate.boot.registry.selector.internal.StrategySelectorImpl.resolveStrategy
```

This happens because the class no longer exists in Hibernate 6.

For Spring Boot 3.x / 4.x applications, please ensure the dialect is updated to:

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.SQLServerDialect

---

2.8) Swagger replacement to Springdoc OpenAPI

Swagger has not been removed in Spring Boot 4.x. The issue arises because **Springfox** (the older Swagger integration library) is not compatible with Spring Boot 3.x and 4.x.

Spring Boot 2.x was based on Java 8, javax.* packages, and Hibernate 5. Since Springfox was built using javax.*, it worked without issues in Boot 2.x.

However, Spring Boot 3+ migrated from javax.* to jakarta.*, and Spring Boot 4.x is fully aligned with Jakarta EE 10, Hibernate 6, and Spring Framework 6+. As Springfox was never upgraded to properly support Jakarta, it fails in Boot 3.x/4.x environments.

Common errors include:

- ClassNotFoundException: javax.servlet.*

- Swagger UI not loading
- Bean creation failures during startup

Since Springfox is no longer actively maintained, it is not suitable for modern Spring Boot versions.

The recommended replacement is **Springdoc OpenAPI**, which fully supports Spring Boot 3.x/4.x and Jakarta packages and is actively maintained.