Architectural Paradigms and Comprehensive Learning Curriculum for Spring WebFlux and Project Reactor

The transition from imperative, blocking architectures to reactive, non-blocking systems represents a fundamental shift in the development of modern JVM-based web applications. As distributed systems and microservices architectures become the standard for enterprise-level software, the limitations of traditional thread-per-request models have become increasingly apparent. Spring WebFlux, introduced in Spring Framework 5, provides a robust alternative to the classic Spring MVC stack, designed specifically to address the requirements of high-concurrency, low-latency, and resource-efficient applications.[1] This report provides an exhaustive analysis of the reactive ecosystem, exploring the theoretical underpinnings of Project Reactor, the architectural design of Spring WebFlux, and a rigorous, structured learning plan for developers seeking to master this complex paradigm.

Theoretical Foundations: The Reactive Manifesto and Non-Blocking I/O

The evolution of the Spring ecosystem toward reactive programming is driven by the necessity for systems that are responsive, resilient, elastic, and message-driven, as outlined in the Reactive Manifesto.[3] In a traditional blocking environment, such as that utilized by Spring MVC, each incoming HTTP request is assigned a dedicated thread from a server's thread pool, typically managed by a Servlet container like Tomcat.[2] This thread is bound to the request for its entire duration, including time spent waiting for I/O-bound operations like database queries or external service calls.[6] Under heavy load, this model can lead to thread exhaustion, where the server consumes significant memory and CPU resources just to maintain idle threads, eventually resulting in performance degradation or system failure.[2]

In contrast, Spring WebFlux utilizes an asynchronous, non-blocking model built on the Reactive Streams specification.[10] Instead of blocking a thread during an I/O operation, the application registers a callback and frees the thread to process other tasks.[5] This model is particularly effective for I/O-bound workloads, allowing a small, fixed number of threads—often corresponding to the number of CPU cores—to handle thousands of concurrent connections with minimal overhead.[2]

Architectural Comparison: Spring MVC vs. Spring WebFlux

The following table synthesizes the fundamental differences between the synchronous Spring MVC model and the asynchronous Spring WebFlux model, providing a baseline for

architectural decision-making.

Architectural Feature	Spring Web MVC	Spring WebFlux
I/O Model	Synchronous, Blocking	Asynchronous, Non-blocking
Threading Strategy	Thread-per-request (Heavyweight)	Event loop (Lightweight)
Primary Dependency	Servlet API	Reactive Streams (Project Reactor)
Concurrency Limit	Bound by Thread Pool Size	Bound by System Resources (CPU/RAM)
Runtime Environment	Servlet 3.0+ (Tomcat, Jetty)	Netty, Undertow, Servlet 3.1+
Programming Style	Imperative	Declarative / Functional
Optimal Workload	CPU-bound, Simple CRUD	I/O-bound, High Concurrency, Streaming
Data Handling	Collection-based (List)	Stream-based (Flux, Mono)

The shift from Spring MVC to WebFlux is not merely a change in libraries but a fundamental "reboot" of the developer's mental model.[9] Developers must move away from commanding step-by-step execution and instead focus on declaring how data should flow through a system of asynchronous publishers and subscribers.[11]

Project Reactor: The Engine of the Reactive Stack

Project Reactor is the foundational library for Spring WebFlux, serving as a concrete implementation of the Reactive Streams specification.[14] It introduces a composable API that allows developers to build complex asynchronous pipelines using two primary reactive types: Mono and Flux.[14]

Reactive Types: Mono and Flux

Project Reactor's strength lies in its semantic cardinality, which provides explicit information about the nature of the asynchronous result.

- **Mono**: This type represents an asynchronous sequence that emits zero or one element before completing.[10] It is the reactive equivalent of a CompletableFuture or a single object return in imperative code. A common use case for Mono is a database lookup by a unique identifier or an HTTP request to an external service that returns a single JSON object.[1]
- **Flux**: This type represents an asynchronous sequence of zero to N elements.[14] It is suitable for streaming data, such as real-time updates from a sensor, a series of rows from a database, or the continuous delivery of social media messages.[10]

The distinction between Mono and Flux allows for better API design; a method returning a Mono guarantees to the consumer that it will receive at most one item, which influences how operators are applied downstream.[14]

The Lifecycle of a Reactive Stream

Understanding the lifecycle of a reactive stream is critical for debugging and performance tuning. The lifecycle is typically divided into three distinct phases: assembly time, subscription time, and execution time.[13]

1. **Assembly Time**: This occurs when the developer defines the pipeline using operators like map, filter, and flatMap. At this stage, no data is moving; the system is essentially building a blueprint of the execution logic.[11]
2. **Subscription Time**: When a consumer calls .subscribe(), the publisher begins the process of data emission. In Reactor, "nothing happens until you subscribe".[11]
3. **Execution Time**: Data flows through the operators, and transformations are applied. This is when the actual asynchronous work, such as network I/O or database access, takes place.[13]

Essential Transformation and Composition Operators

The vocabulary of Project Reactor is vast, often leading to confusion for beginners. The following table provides a comparison of the most critical mapping operators used for asynchronous transformation.

Operator	Subscription Mechanism	Ordering and Interleaving	Primary Use Case
map	Synchronous	Maintains order, no interleaving	1:1 data transformation

			(e.g., entity to DTO).[10]
flatMap	Eager (Parallel)	No guaranteed order, allows interleaving	High-throughput parallel API calls where order is irrelevant.[22]
concatMap	Sequential	Maintains order, no interleaving	Sequential operations that must happen in a specific order.[22]
flatMapSequential	Eager (Parallel)	Maintains order via queuing	Parallel calls that require results in the original subscription order.[22]
switchMap	Dynamic (Latest)	Only latest inner publisher is active	Search-as-you-type or scenarios where old requests are obsolete.[21]

A critical insight for developers is that flatMap has a default concurrency limit of 256.[23] For massive fan-out operations, this limit must be tuned to avoid connection pool exhaustion or memory spikes.[12]

Spring WebFlux Architecture: The Netty Event Loop

Spring WebFlux departs from the Servlet container model by default, opting instead for Netty, an asynchronous event-driven network application framework.[2] Netty's execution model is centered around the concept of the EventLoop, which handles all I/O events for one or more Channel objects.[25]

The Event Loop Mechanism

In a WebFlux application, the number of event loop threads is typically small, often calculated as:

$$N_{threads} = \max(2, \text{CPU Cores} \times 2)$$

This lean threading model is the key to WebFlux's scalability.9 When a request arrives, the Boss thread group accepts the connection and hands it over to a Worker thread group.19 The worker thread executes the reactive pipeline. If the pipeline requires an I/O operation, the thread does not wait; it registers a "watcher" and moves on to the next request in the FIFO task queue.7

This efficiency comes with a strict requirement: **never block the event loop**. If a developer introduces a blocking call (such as Thread.sleep() or a synchronous JDBC query) on an event loop thread, the entire server's throughput can collapse because all other requests assigned to that thread will be stalled.[12]

Schedulers: Managing Threading Contexts

Project Reactor provides the Schedulers utility to safely move execution off the event loop when necessary. This is essential for integrating with blocking legacy systems or performing CPU-bound computations.[12]

Scheduler Type	Intended Workload	Characteristics
Schedulers.parallel()	CPU-intensive tasks	Bounded to the number of CPU cores.[12]
Schedulers.boundedElastic()	Blocking I/O (JDBC, File I/O)	Dynamic thread pool that caps growth to prevent OOM.[10]
Schedulers.single()	Low-latency, stateful work	Single dedicated thread for sequential tasks.[12]
Schedulers.immediate()	No context switch	Executes on the current thread.[12]

The golden rule of WebFlux development is to "Stay on the event loop by default" and only hop to a scheduler when interacting with blocking resources.[12]

Strategic Learning Curriculum for Spring WebFlux

Mastering Spring WebFlux requires a phased approach that builds from theoretical concepts to advanced production patterns. This twelve-week curriculum integrates research-backed resources and practical milestones.

Phase 1: Paradigms and Reactive Streams (Weeks 1–2)

The objective of the first phase is to break the imperative mental model and understand the

"why" of reactive systems.

- **Core Study**: Research the limitations of the Servlet thread-per-request model and the advantages of non-blocking I/O.[2]
- **The Reactive Manifesto**: Review the four pillars of reactive systems—Responsiveness, Resilience, Elasticity, and Message-Driven.[3]
- **Key Resource**:(https://www.pluralsight.com/courses/getting-started-spring-webflux). This course provides the foundational knowledge of exactly what reactive programming is and why it's useful.[27]
- **Practical Milestone**: Set up a basic Spring Boot project using the spring-boot-starter-webflux dependency and analyze the default Netty logging to observe the worker threads.[1]

Phase 2: Mastering Project Reactor Fundamentals (Weeks 3–4)

This phase focuses on the core library that powers WebFlux. Without a deep understanding of Reactor, WebFlux code will remain opaque and difficult to debug.

- **Flux and Mono Exercises**: Practice creating reactive types using just(), fromIterable(), fromCallable(), and defer().[10]
- **The "Nothing Happens Until Subscription" Rule**: Experiment with creating pipelines without subscribing to observe the lack of execution.[10]
- **Interactive Tutorial**: Complete the(https://github.com/reactor/lite-rx-api-hands-on) on GitHub. This is an industry-standard resource for learning the Reactor API through unit tests.[29]
- **Key Operators**: Master map, filter, flatMap, and zip. Understand how zip can be used to combine multiple asynchronous calls (e.g., fetching a user profile and account balance simultaneously).[10]

Phase 3: WebFlux Programming Models (Weeks 5–6)

WebFlux offers two ways to expose endpoints: annotated controllers and functional routing.[31]

- **Annotated Controllers**: Leverage existing Spring MVC knowledge by using @RestController and @GetMapping, but with Mono and Flux return types.[17]
- **Functional Endpoints (WebFlux.fn)**: Learn the lightweight functional model using RouterFunction and HandlerFunction. This model is more explicit and often preferred for microservices.[31]
- **Resource**:(https://spring.io/guides/gs/reactive-rest-service/) guide from Spring.io.[35]
- **Practical Milestone**: Implement a simple CRUD API twice—once with annotations and once with functional routing—to compare the two styles.[32]

Phase 4: Reactive Persistence and Data Streams (Weeks 7–8)

End-to-end reactivity requires a non-blocking data access layer. Traditional JDBC will block

the event loop, negating WebFlux's benefits.[26]

- **R2DBC (Reactive Relational Database Connectivity)**: Understand how R2DBC provides non-blocking drivers for SQL databases like PostgreSQL and MySQL.[37]
- **Reactive NoSQL**: Explore Spring Data MongoDB Reactive, which supports Tailable Cursors for streaming data from a collection.[17]
- **Backpressure in Data Streams**: Learn how to use limitRate() or onBackpressureBuffer() to handle scenarios where the database produces data faster than the web client can consume it.[11]
- **Key Resource**:(https://bell-sw.com/blog/mastering-reactive-programming-with-spring-data-r2dbc/).[37]

Phase 5: Advanced Security and Resilience (Weeks 9–10)

Security in a reactive world differs from the Servlet model because ThreadLocal cannot be used to store security context.[41]

- **Spring Security Reactive**: Configure the SecurityWebFilterChain and learn how the Reactor Context is used to propagate authentication information across threads.[41]
- **Cross-Cutting Concerns**: Implement reactive filters (WebFilter) for logging, tracing, and header manipulation.[1]
- **Resilience Patterns**: Integrate Resilience4j for circuit breaking and rate limiting. Learn how to apply timeouts at both the Netty level and the Reactor level (.timeout()).[12]
- **Resource**:(https://docs.spring.io/spring-security/reference/reactive/index.html).[41]

Phase 6: Testing, Debugging, and Production Readiness (Weeks 11–12)

The final phase addresses the "complexity tax" of reactive systems: testing and troubleshooting.[9]

- **StepVerifier**: Master the use of StepVerifier to test reactive sequences step-by-step, verifying emissions, errors, and completion.[15]
- **WebTestClient**: Use WebTestClient for end-to-end integration testing of your reactive endpoints.[2]
- **Debugging Tools**: Learn to use ReactorDebugAgent.init() for production-ready stack traces and BlockHound for detecting blocking calls in the event loop.[13]
- **Resource**:(https://spring.io/blog/2019/03/28/reactor-debugging-experience/).[13]

Deep Dive into Reactive Data Access: R2DBC and NoSQL

The persistence layer is often the most significant bottleneck in reactive applications. Developers must choose between R2DBC for relational data and native reactive drivers for

NoSQL databases.[37]

R2DBC vs. JDBC: The Technical Divide

R2DBC is not a wrapper around JDBC; it is a completely new SPI designed from the ground up for reactive streams.[37]

Feature	JDBC	R2DBC
I/O Nature	Blocking (Thread per connection)	Non-blocking (Asynchronous)
API Style	Imperative	Functional / Reactive Streams
ORM Support	Mature (Hibernate/JPA)	Limited (Spring Data R2DBC)
Transaction Model	Thread-bound (ThreadLocal)	Context-bound (Reactor Context)
Relationship Mapping	Automatic (@OneToMany, etc.)	Manual (@Transient + manual fetch).[37]

A major challenge with Spring Data R2DBC is its lack of automated relationship mapping (like JPA's @OneToMany). Developers must manually orchestrate the fetching of related entities using flatMap and zip operators.[37]

Reactive MongoDB: A Gateway to Real-Time Data

MongoDB is frequently paired with WebFlux because its reactive driver is highly mature and supports features like Server-Sent Events (SSE) natively.[18] By using a Tailable Cursor, a Flux can be kept open to continuously stream new documents as they are inserted into a capped collection, facilitating real-time dashboards or chat applications.[38]

Security Architectures in Reactive WebFlux

Spring Security's adaptation for WebFlux represents a significant departure from its Servlet roots. In a blocking environment, SecurityContextHolder uses a ThreadLocal variable to store the SecurityContext. In a reactive environment, where a single request may hop across

multiple threads, ThreadLocal is unreliable.[41]

The Role of Reactor Context

Reactive Spring Security stores the security context within the Reactor Context, which is a key-value store that travels "up" the stream from the subscriber to the publisher.[41] This ensures that even when execution switches from an event loop thread to a boundedElastic thread (e.g., for a database call), the security information remains accessible.

Security Implementation Best Practices

1. **CORS First**: Configure CORS via a CorsWebFilter to ensure it is processed before authentication filters, as pre-flight requests do not contain credentials.[41]
2. **CSRF Customization**: For JavaScript-based applications (Angular/React), use CookieServerCsrfTokenRepository.withHttpOnlyFalse() to allow the frontend to read the CSRF token.[41]
3. **Method Security**: Enable method-level security with @EnableReactiveMethodSecurity. Ensure that methods return a Publisher (Mono or Flux) for the security context to be properly managed.[41]
4. **Logout Handling**: In a reactive environment, a custom WebSessionServerLogoutHandler may be required to ensure that the WebSession is invalidated correctly.[41]

Advanced Debugging: Overcoming the Reactive Stack Trace

One of the primary complaints regarding Project Reactor is the "archaeological" nature of its stack traces. Because the assembly of the pipeline happens separately from its execution, a standard stack trace often shows only framework-level orchestration code, making it impossible to identify the line of business logic that failed.[9]

Diagnostics Toolkit

To maintain production stability, developers should employ the following diagnostic tools:

- **Checkpointing**: The .checkpoint("Description") operator can be added to a pipeline to provide a "breadcrumb" in the stack trace if an error occurs at that point. This is low-overhead and suitable for production.[45]
- **The Log Operator**: .log() can be placed between operators to observe signals (onNext, onError, onComplete) and backpressure demand in real-time. This is invaluable during local development.[19]
- **Reactor Debug Agent**: By initializing the ReactorDebugAgent, the system instruments your classes at load time to capture assembly information. This is significantly more performant than Hooks.onOperatorDebug() and is the recommended approach for

production environments.[13]
- **BlockHound**: This agent should be integrated into the test suite. If a test path accidentally hits a blocking I/O operation on a non-blocking thread, BlockHound will throw an exception, preventing performance regressions from reaching production.[13]

Performance Tuning and Production Best Practices

Scaling a WebFlux application requires more than just non-blocking code; it requires careful management of system resources and downstream pressure.

Concurrency and Fan-out Management

A common mistake in reactive programming is the use of unbounded flatMap. If an upstream Flux emits 1,000 items and each item triggers an external API call via flatMap, the system will attempt 1,000 concurrent network connections.[23] This can lead to:

1. **Connection Pool Starvation**: Exhausting the client's HTTP connection pool.
2. **Memory Blowups**: Buffering large numbers of pending responses.
3. **Network Saturation**: Overwhelming the target service.

The solution is to use the concurrency parameter: .flatMap(id -> client.get(id), 16). This limits the number of active inner publishers to 16, ensuring predictable resource usage.[12]

Timeouts and Fault Tolerance

Every external call in a reactive pipeline should have a timeout. While Netty provides connection and read timeouts, Reactor's .timeout(Duration) operator provides an "SLA timeout" that cancels the entire subscription if the downstream does not respond within the window.[12]

Level	Mechanism	Scope
Network (Netty)	Connect/Read Timeout	Low-level socket behavior.[12]
Pipeline (Reactor)	.timeout(Duration)	The entire asynchronous operation.[12]
Resilience (Resilience4j)	Circuit Breaker	Long-term fault isolation.[42]

Combining these layers ensures that a single slow dependency does not cause a cascading failure throughout the microservices ecosystem.[12]

Recommended Learning Resources and Platforms

The learning landscape for Spring WebFlux has matured significantly. The following table highlights the most effective resources categorized by learning style and depth.

Platform	Recommended Resource	Strength
Official Documentation	(https://projectreactor.io/docs/core/release/reference/)	Exhaustive detail on operators and internals.[11]
Spring Academy	Pro Content (Now Free)	Hands-on labs with browser-based IDEs.[47]
Interactive Labs	(https://github.com/reactor/lite-rx-api-hands-on)	Coding exercises to master the Reactor API.[29]
Video Courses	(https://www.udemy.com/topic/spring-webflux/)	Practical microservice patterns and R2DBC.[28]
Community Content	(https://www.baeldung.com/spring-webflux)	Deep dives into specific use cases like security and testing.[28]
GitHub Demos	(https://github.com/AthirsonSilva/reactive-microservices)	End-to-end microservices example with multiple databases.[51]

Conclusion and Future Outlook

The adoption of Spring WebFlux and Project Reactor is a strategic decision that offers immense scalability rewards but carries a significant "complexity tax".[9] For applications characterized by massive I/O concurrency, real-time data streaming, or high-throughput microservice orchestration, the reactive stack is indispensable.[2] However, for standard business applications with moderate traffic, the simplicity and mature ecosystem of Spring MVC often provide a better balance of productivity and performance.[7]

As the Java ecosystem evolves, the introduction of Project Loom and Virtual Threads may alter the value proposition of reactive programming by providing a simpler way to achieve high concurrency. Nevertheless, the declarative power, backpressure management, and functional composition offered by Project Reactor ensure that Spring WebFlux will remain a cornerstone

of sophisticated, cloud-native backend engineering for the foreseeable future.[4] Successful mastery of this stack requires persistent practice, a commitment to understanding low-level threading models, and a disciplined approach to testing and diagnostics.

Works cited

1. An Introduction to Spring WebFlux: Reactive Programming Made Easy | by Bolot Kasybekov, accessed January 18, 2026, https://medium.com/@bolot.89/an-introduction-to-spring-webflux-reactive-programming-made-easy-f70050f4c6c6
2. Spring MVC vs WebFlux: When to Use Which Framework? - GeeksforGeeks, accessed January 18, 2026, https://www.geeksforgeeks.org/blogs/spring-mvc-vs-spring-web-flux/
3. Spring WebFlux Guide - centron GmbH, accessed January 18, 2026, https://www.centron.de/en/tutorial/spring-webflux-guide/
4. Spring Reactive for Beginners: Quick Start Guide - Mindbowser, accessed January 18, 2026, https://www.mindbowser.com/spring-reactive-guide/
5. Spring Web MVC vs Spring WebFlux: differences between Spring web frameworks, accessed January 18, 2026, https://symflower.com/en/company/blog/2024/spring-mvc-spring-webflux/
6. Comparing & Contrasting Spring WebMVC & WebFlux, accessed January 18, 2026, http://www.dre.vanderbilt.edu/~schmidt/cs891/2023-PDFs/17.1.2-comparing-and-contrasting-WebMVC-and-WebFlux.pdf
7. Spring WebFlux vs. Spring MVC: Cutting Through the Hype — A Pragmatic Guide - Medium, accessed January 18, 2026, https://medium.com/@mesfandiari77/spring-webflux-vs-spring-mvc-cutting-through-the-hype-a-pragmatic-guide-881e484056a9
8. Spring MVC Async vs Spring WebFlux | Baeldung, accessed January 18, 2026, https://www.baeldung.com/spring-mvc-async-vs-webflux
9. Spring WebFlux: When to Use It and How to Build With It - DEV Community, accessed January 18, 2026, https://dev.to/adamthedeveloper/spring-webflux-when-to-use-it-and-how-to-build-with-it-5a6e
10. Reactive Streams with Project Reactor | by Rahul | Dec, 2025 - Medium, accessed January 18, 2026, https://medium.com/@rahulhind/reactive-streams-with-project-reactor-3b0ec1b0fc07
11. Introduction to Reactive Programming :: Reactor Core Reference Guide, accessed January 18, 2026, https://projectreactor.io/docs/core/release/reference/reactiveProgramming.html
12. Advanced Reactive Programming with Spring WebFlux: Threads, Schedulers, Timeouts, and Hybrid Architectures | by Burak KOCAK | Jan, 2026 | Medium, accessed January 18, 2026, https://medium.com/@burakkocakeu/advanced-reactive-programming-with-spri

ng-webflux-threads-schedulers-timeouts-and-hybrid-3fb123a12cc6
13. Reactor Debugging Experience - Spring, accessed January 18, 2026, https://spring.io/blog/2019/03/28/reactor-debugging-experience/
14. Reactor Core Features :: Reactor Core Reference Guide, accessed January 18, 2026, https://projectreactor.io/docs/core/release/reference/coreFeatures.html
15. Reactor 3 Reference Guide, accessed January 18, 2026, https://projectreactor.io/docs/core/3.5.15/reference
16. Spring Boot - Reactive Programming Using Spring Webflux Framework - GeeksforGeeks, accessed January 18, 2026, https://www.geeksforgeeks.org/springboot/spring-boot-reactive-programming-using-spring-webflux-framework/
17. A Step-by-Step Guide to Reactive Programming in Spring Boot - DEV Community, accessed January 18, 2026, https://dev.to/devcorner/a-step-by-step-guide-to-reactive-programming-in-spring-boot-5f97
18. How to Build a Reactive Microservice with Spring WebFlux & MongoDB - YouTube, accessed January 18, 2026, https://www.youtube.com/watch?v=y5CsYhadZiA
19. Debugging Spring Reactive Applications | by Kalpa Senanayake | The Startup | Medium, accessed January 18, 2026, https://medium.com/swlh/debugging-spring-reactive-applications-660be7989968
20. Spring Reactive Programming WebFlux | Complete Enterprise Guide (2025), accessed January 18, 2026, https://www.inexture.com/spring-reactive-programming-webflux-complete-2025-guide/
21. Reactive Programming: The Hitchhiker's Guide to map operators | by Benedikt Jerat | Digital Frontiers — Das Blog | Medium, accessed January 18, 2026, https://medium.com/digitalfrontiers/reactive-programming-the-hitchhikers-guide-to-map-operators-7d8bbc1d8465
22. Whats the difference between flatMap, flatMapSequential and concatMap in Project Reactor? - Stack Overflow, accessed January 18, 2026, https://stackoverflow.com/questions/71971062/whats-the-difference-between-flatmap-flatmapsequential-and-concatmap-in-project
23. Project Reactor — flatMap vs concatMap vs flatMapSequential | by Abhishek Gautam, accessed January 18, 2026, https://medium.com/@abugautam/project-reactor-flatmap-vs-concatmap-vs-flatmapsequential-dc74ca58829e
24. More Operators For Transforming Sequences | Project Reactor Course - Esteban Herrera, accessed January 18, 2026, https://eherrera.net/project-reactor-course/04-using-other-reactive-operators/transforming-sequences.html
25. Spring Boot WebFlux Complete Flow | Netty Event Loop Explained ! | Reactive Programming, accessed January 18, 2026, https://www.youtube.com/watch?v=7OGeUQu3rEQ
26. How to detect blocking calls in Spring Webflux - DEV Community, accessed

January 18, 2026, https://dev.to/dayanandaeswar/detect-blocking-calls-using-blockhound-in-spring-webflux-51jc
27. Spring WebFlux 5: Getting Started - Pluralsight, accessed January 18, 2026, https://www.pluralsight.com/courses/getting-started-spring-webflux
28. Top 8 Courses to learn Reactive Spring Boot and WebFlux in 2025 - Best of Lot, accessed January 18, 2026, https://javarevisited.blogspot.com/2021/04/best-reactive-spring-and-webflux-courses-for-java-developers.html
29. Lite Rx API Hands-On with Reactor Core 3 - GitHub, accessed January 18, 2026, https://github.com/reactor/lite-rx-api-hands-on
30. Learn - Project Reactor, accessed January 18, 2026, https://projectreactor.io/learn
31. Functional Endpoints :: Spring Framework, accessed January 18, 2026, https://docs.spring.io/spring-framework/reference/web/webflux-functional.html
32. Registering Functional Endpoints with Spring Boot and RouterFunction - Medium, accessed January 18, 2026, https://medium.com/@AlexanderObregon/registering-functional-endpoints-with-spring-boot-and-routerfunction-4dbac9a4e61b
33. Functional Endpoints: Alternative to Controllers in WebFlux - DZone, accessed January 18, 2026, https://dzone.com/articles/functional-endpoints-alternative-to-controllers-webflux
34. Course Module 5. Spring - Lecture: Functional endpoints - CodeGym, accessed January 18, 2026, https://codegym.cc/quests/lectures/en.questspring.level05.lecture03
35. Guides - Spring, accessed January 18, 2026, https://spring.io/guides/
36. What is the difference between Router and Annotated Controllers? - Stack Overflow, accessed January 18, 2026, https://stackoverflow.com/questions/51786154/what-is-the-difference-between-router-and-annotated-controllers
37. A guide to Spring Data R2DBC - BellSoft, accessed January 18, 2026, https://bell-sw.com/blog/mastering-reactive-programming-with-spring-data-r2dbc/
38. Reactive Spring Data: When and How to Use Non-Blocking Repositories - Java Code Geeks, accessed January 18, 2026, https://www.javacodegeeks.com/2025/07/reactive-spring-data-when-and-how-to-use-non-blocking-repositories.html
39. Spring Data JDBC and R2DBC, accessed January 18, 2026, https://docs.spring.io/spring-data/relational/reference/index.html
40. Build a Reactive App with Spring Boot and MongoDB | Okta Developer, accessed January 18, 2026, https://developer.okta.com/blog/2019/02/21/reactive-with-spring-boot-mongodb
41. Reactive Applications :: Spring Security, accessed January 18, 2026, https://docs.spring.io/spring-security/reference/reactive/index.html
42. Top Spring WebFlux Courses Online - Updated [January 2026] - Udemy, accessed

January 18, 2026, https://www.udemy.com/topic/spring-webflux/
43. Spring Boot & Microservices Masterclass All-in-One 4 Hour Full Course - YouTube, accessed January 18, 2026, https://www.youtube.com/watch?v=XZcvYhgY39A
44. Testing :: Reactor Core Reference Guide - Project Reactor, accessed January 18, 2026, https://projectreactor.io/docs/core/release/reference/testing.html
45. Debugging Reactor :: Reactor Core Reference Guide, accessed January 18, 2026, https://projectreactor.io/docs/core/release/reference/debugging.html
46. Reactive Streams | IntelliJ IDEA Documentation - JetBrains, accessed January 18, 2026, https://www.jetbrains.com/help/idea/reactor.html
47. Spring Academy: Home, accessed January 18, 2026, https://spring.academy/
48. 10 Best Spring/Boot Courses for 2026: Microservices & Web Apps - Class Central, accessed January 18, 2026, https://www.classcentral.com/report/best-spring-boot-courses/
49. Spring Academy Pro Content Now Free To Access, accessed January 18, 2026, https://spring.io/blog/2024/04/10/spring-academy-pro-content-now-free-to-access/
50. 7 Best WebFlux and Reactive Spring Boot Courses for Java Programmers in 2025 - Medium, accessed January 18, 2026, https://medium.com/javarevisited/7-best-webflux-and-reactive-spring-boot-courses-for-java-programmers-33b7c6fa8995
51. AthirsonSilva/reactive-microservices: A simple project following the microservice architecture with the objective of learn more about microservices in a reactive way - GitHub, accessed January 18, 2026, https://github.com/AthirsonSilva/reactive-microservices