International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 06 | June - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

Relevance and Resilience: Exploring the Continued Importance of Spring Boot in Modern Application Development

Dr. B.M Sagar
Department of ISE
R. V. College of Engineering®
Bengaluru, India

Dileep Sharma
Department of ISE
R. V. College of Engineering®
Bengaluru, India

***Abstract*—Spring Boot has emerged as a vital technology in modern software development, offering a robust framework for building resilient and scalable applications. This paper investigates the ongoing relevance of Spring Boot in the field of software development. Starting by looking at its beginnings, evolution of Spring Boot from Spring, its features, ecosystem, and adoption trends, we highlight the enduring significance of Spring Boot in enabling developers to streamline development processes, enhance productivity, and achieve architectural resilience. By simplifying dependency management through starter POMs and reducing boilerplate configuration with auto-configuration, Spring Boot has significantly enhanced the usability of the Spring framework. By delving into some analytics, the paper explains how Spring Boot continues to empower developers to meet the challenges posed by modern application development, including microservices architectures, cloud-native deployments, and DevOps practices.**

***Index Terms*—Spring Boot, Spring , Microservice,Spring MVC ,Spring Security , Java**

I. Introduction

Spring Boot, initiated by Rod Johnson then working in Pivotal Software (now part of VMware) in 2012, has evolved from a revolutionary idea to a foundational technology in modern application development. Its inception marked a paradigm shift in the Entreprise Application Development, introducing a convention-over-configuration approach that streamlined application setup and eliminated boilerplate code. The very first version of Spring Boot laid the groundwork for subsequent releases, setting the stage for its widespread adoption and continued evolution.

With each iteration, Spring Boot has matured and adapted to the evolving needs of developers and the changing landscape of software development. Version 2.0, released in 2018, introduced significant enhancements, including support for reactive programming with Spring WebFlux and improved support for Kotlin, further expanding its appeal to a broader audience of developers.

As Spring Boot continued to gain traction in the industry, version 3.x, released in 2021, focused on enhancing developer productivity and simplifying cloud-native development. With features such as auto-configuration updates and improved support for cloud-native deployment platforms like Kubernetes, Spring Boot 3.x solidified its position as a cornerstone technology for building resilient and scalable applications in modern cloud environments.

II. Literature Review

In recent times, Spring Boot has gained considerable prominence as a potent platform for developing Java-based systems, especially in the fields of web development and microservices. Walls [1] claim that Spring Boot makes application management and deployment simpler for developers by streamlining the configuration and testing procedures [2]. As noted by Walls [3] and Gutierrez in their respective works [9], the framework's Actuator module is especially noteworthy for its capacity to offer insights into application health and metrics, further assisting in the auditing process. Applications developed with Spring Boot are more reliable and maintainable because to this thorough monitoring and auditing strategy. Heckler [4] also emphasises how effective Spring Boot is at expediting the development process, which helps explain why it has become so popular.

Recent studies have explored the integration of Spring Boot with other technologies to enhance its capabilities. For instance, Jovanovic et al. [12] discuss the integration of Spring Boot REST web services with the Weka framework for artificial intelligence applications, demonstrating how the framework can support advanced machine learning tasks. Similarly, Guntupally et al. [13] illustrate how Spring Boot can be leveraged to improve quality of data report generated in big data contexts, highlighting its scalability and efficiency in handling large datasets. Additionally, the development of domain-specific languages (DSLs) for generating Spring Boot REST APIs, as presented by Gomez et al. [10], showcases the framework's potential for facilitating rapid development and deployment of web services. The evolution of Spring Boot is further evidenced by its adaptation for reactive web development, as reviewed by Rakshith and Swamy [11], and its comprehensive documentation and continuous updates provided by Webb et al. [5]. The practical applications of Spring Boot in real-world scenarios, such as the development

III. Spring Boot

Spring Boot has become an essential framework in the world of modern application development. This section describes various aspects of Spring Boot, including its core principles, how it differs from the traditional Spring Framework, its architecture, and its standout features.

A. Dependency Injection

Dependency Injection (DI) and Inversion of Control (IoC) are core principles of the Spring Framework, and they play a crucial role in Spring Boot. DI allows objects to be injected into other objects, rather than being created directly by those objects. This leads to loosely coupled code, which is easier to maintain and test.

In Spring Boot, DI is achieved using annotations such as `@Autowired`, `@Component`. The Spring IoC container manages the lifecycle and dependencies of these components, injecting the required dependencies at runtime. This automated injection process reduces boilerplate code and simplifies the wiring of application components, leading to more modular and testable codebases.This makes spring boot highly maintanable and developer friendly by delegating object state management completely to spring boot.

B. Evolution from Spring Framework to Spring Boot

Dependency inversion was made available to developers with the release of the Spring Framework in 2003. Because it made dependency management easier, developers were initially rather happy with this. But with time, the lengthy process of creating and managing beans, along with the need to write a lot of boilerplate code, got tedious. In response to these problems, Spring Boot surfaced, providing a thorough method that prioritised convention above configuration. Some of the most important distinctions between Spring Boot and the Spring Framework are shown in Table I.

C. Architecture

Spring Boot's architecture is crafted to streamline the creation of Java applications by automating numerous configurations and setups, thereby reducing developer workload. This architecture consists of several layers, each performing distinct responsibilities, ensuring a clean separation of concerns and promoting a more maintainable and scalable codebase. The primary layers include the client layer, service layer, and model layer, often integrated with JPA (Java Persistence API) for database interactions.

Spring Framework	Spring Boot
Requires extensive manual configuration, often through XML or Java annotations.	Provides auto-configuration to simplify setup, reducing the need for manual configuration.
More time-consuming due to the need to configure dependencies, application context, etc.	Faster setup with starter dependencies and embedded servers, enabling quick project initialization.
Developers must manually manage dependencies and include necessary libraries.	Offers starter POMs that bundle commonly used libraries, simplifying dependency management.
Requires an external application server for deployment.	Includes embedded servers such as Tomcat, Undertow and Jetty allowing applications to run as standalone executables.

TABLE I
Differences between Spring in comparison to Spring Boot

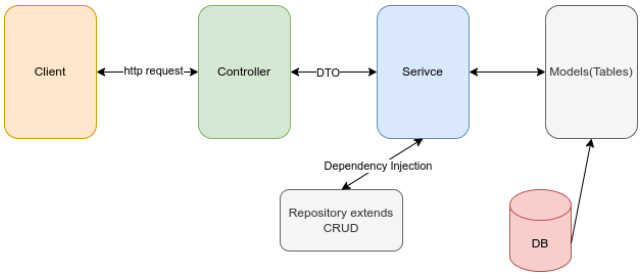

Fig. 1. Spring Boot Architecture

1) Client Layer: The client layer is the entry point of the application, responsible to handle all the incoming HTTP requests from clients. It typically consists of controllers that map requests to appropriate service methods. These controllers are annotated with `@RestController` or `@Controller` and use `@RequestMapping` or similar annotations to define request handling methods.

2) Service Layer: The service layer contains the business logic of the application. It acts as an intermediate bridge between the client layer and the model layer. The layer's major responsibility lies in processing client requests, applying core business rules, and orchestrating interactions with the model layer. Services are annotated with `@Service`, and they can leverage dependency injection to interact with repositories and other services.

3) Model Layer: The model layer manages the data and database interactions. It includes entities that represent the data structure and repositories that provide CRUD operations on the database. Entities are typically annotated with `@Entity` and mapped to database tables, while repositories extend interfaces like `JpaRepository` to facilitate data access using JPA.

The use of JPA in the model layer allows for object-

relational mapping, making it easier to work with relational databases by using Java objects. This integration ensures that the application can perform complex database operations with minimal boilerplate code.

Together, these layers form a cohesive architecture that promotes modularity, testability, and scalability.A typical flow

of a request would be from client to service (through dto) ,then the db interactions from service via JPA (typically Hibernate an ORM is used to create entities) and finally the response is mapped by another dto which is wrappend in ResponseEntity from controller.Throughout the process the auto-configuration and exhaustive rich libraries of Spring Boot further simplify the process , allowing developers to focus more on implementing the problem statement and less on infrastructure concerns.

IV. Features Making Spring Boot Industry Relevant

The below are few of the features that really make spring-boot have an edge over other frameworks

A. Auto-Configuration

Referred many a times in the paper, Spring Boot's auto-configuration feature automatically configures Spring applications based on various dependencies present on the classpath. This eliminates the need for manual configuration setup in many cases, significantly reducing boilerplate code and setup time. Developers can focus on building business logic instead of dealing with complex configurations. Spring Boot achieves this through a set of annotations, some of the key annotations include:

- `@SpringBootApplication`: This is a composite annotation that combines `@Configuration`, `@EnableAutoConfiguration`, `@ComponentScan`. It configures the Spring application context, activates auto-configuration, and performs component scanning within the package.
- `@RestController`: This annotation is a specialized version of `@Controller` used in Spring MVC. It indicates that the class is a web controller and that the return value of methods should be written directly to the HTTP response body.
- `@SpringBootTest`: This annotation is used for integration testing in Spring Boot. It also be used alongside mockito to perform unit testing , further each layers do have annotations for testing .
- `@EnableScheduling`: This annotation enables Spring's scheduled task execution capability, allowing methods annotated with `@Scheduled` to be run at specified intervals.

B. Starter Dependencies

Spring Boot provides a set of starter POMs (Project Ob- ject Models) that aggregate commonly used libraries and frameworks. These starter dependencies simplify dependency management by bundling the necessary libraries for specific functionalities, such as web development, JPA, and security.

For example, the `spring-boot-starter-web` starter includes dependencies for building web applications, such as Spring MVC and an embedded Tomcat server. Similarly, the `spring-boot-starter-data-jpa` starter bundles Hibernate, Spring Data JPA, and an embedded H2 database for working with JPA.

Other notable starters include:

- `spring-boot-starter-security`: Includes Spring Security for authentication and authorization.
- `spring-boot-starter-test`: Provides libraries for testing, including JUnit, Hamcrest, and Mockito.
- `spring-boot-starter-thymeleaf`: Bundles the Thymeleaf template engine for server-side rendering of web pages.
- `spring-boot-starter-actuator`: Includes features for monitoring and managing applications, such as health checks and metrics.
- `spring-boot-starter-mail`: Configures JavaMailSender with commonly used email settings.

These starter dependencies not only simplify the initial setup of a Spring Boot application but also ensure that the included libraries are compatible with each other, reducing the risk of version conflicts and other dependency-related issues.

C. Spring Boot Actuator

Spring Boot Actuator provides a range of production-ready features that help monitor and manage applications. It includes endpoints for health checks, metrics, application information,auditing of the calls and more. These endpoints are useful for gaining insights into the application's performance and behavior, making it easier to maintain and troubleshoot in production environments.

D. Spring Boot DevTools

Spring Boot DevTools enhances the development experience by offering automatic restarts, live reload,remote cli monitor and customisations for a smoother workflow. DevTools can reduce the time needed for testing and debugging by automatically restarting the programme or reloading specific components when developers make changes to their code.

E. Security Features

Applications can benefit from a complete security architecture thanks to the smooth integration between Spring Security and Spring Boot. It has functions for authorization, authentication, and defence against typical security risks. Developers can make their applications secure by default by implementing

strong security mechanisms with little configuration.

F. Microservices Support

In particular, Spring Boot works incredibly well for creating microservices architectures. It offers functions like distributed tracing, circuit breakers, configuration management, service discovery, and Spring Cloud integration. These solutions make it easier for businesses to create scalable, CI-CD apps and resilient systems by streamlining the development and man- agement of microservices.

G. Exhaustive Libraries

Be it a small use case as utilities or a bigger feature such as scheduling, asynchronous tasks, or transactional queries, Spring Boot has libraries for all. The extensive range of libraries and built-in functionalities provided by Spring Boot significantly simplifies the development process, making it a go-to choice for developers. Here are some examples of the libraries and features available in Spring Boot:

1) Utility Libraries: Spring Boot offers numerous utility libraries that cater to common development needs. Libraries such as Apache Commons IO and Spring Utils provide ready-to-use functions for file handling, string manipulation, and other common tasks, reducing the need for custom implementations.

2) Scheduling: For scheduling tasks, Spring Boot integrates with the Spring Scheduling framework. This integration allows developers to schedule tasks using simple annotations a classical annotation used is `@Scheduled` with necessary props , enabling the execution of periodic tasks such as batch jobs, automated reports, database cleanups, consistency management and maintenance routines without the need for complex configuration.

3) Asynchronous Processing: By integrating with the `@Async` annotation and the `TaskExecutor` interface, Spring Boot facilitates asynchronous processing. Through the use of this functionality, developers can run tasks in the background and enhance application performance by transferring laborious actions to different threads.

4) Transactional Management: Spring Boot provides robust transactional management with Spring Data JPA. By using the `@Transactional` annotation, developers can ensure data integrity and manage transactions declaratively. This feature is particularly useful for complex database operations that require consistency and rollback capabilities.

5) Security: Spring Boot's integration with Spring Security allows developers to implement authentication, authorization, and other security features with ease. The framework provides ready-to-use security components and configurations that help protect applications against common security threats.

H. Cloud-Native Deployment

Spring Boot's compatibility with cloud platforms makes it an ideal choice for cloud-native applications. It supports containerization with Docker, orchestration with Kubernetes, and integration with cloud services from providers like AWS, Azure, and Google Cloud.For instance sdks provided by AWS are sufficient for integrating advanced features such as SQS, STS etc. This flexibility ensures that Spring Boot applications can easily be deployed and managed in various cloud environments along with helping in ease of utilization of various cloud features.

I. Comprehensive Documentation and Community Support

Spring Boot's extensive documentation and vibrant community support are among its advantages. The official documentation offers thorough instructions, resources, and case studies to aid developers in becoming up to speed as soon as possible. Furthermore, through forums, blogs, and open-source projects, the active community surrounding Spring Boot offers helpful tools and contributes to its ongoing development.

Together, these characteristics make Spring Boot a strong and useful framework for the industry, empowering developers to create excellent applications quickly and effectively.

V. Current Industry and Spring Boot

Particularly for building microservices, Spring Boot has grown to be a significant framework in the programming world. It is made to create production-ready, stand-alone apps and was developed by Pivotal Team. Comparing Spring Boot to other well-known programming frameworks, this section looks at its market share, usage trends, and regional distribution.

A. Geographical Distribution

Spring Boot is most widely used in the United States, with 3,275 companies utilizing the framework. Other countries with significant usage include India (1,429 companies) and Germany (609 companies), reflecting a strong global presence.

B. Comparison with Other Frameworks

Figure 2 provides a graph of top 5 popular frameworks used to develop backend . This figure highlights the number of companies using each framework.As clearly seen Spring is the third popular framework is utilized by a significant number of companies, indicating its robust capabilities and reliability. The consistent adoption of Spring and thus Spring Boot in various industries reflects its versatility and effectiveness in building scalable applications.

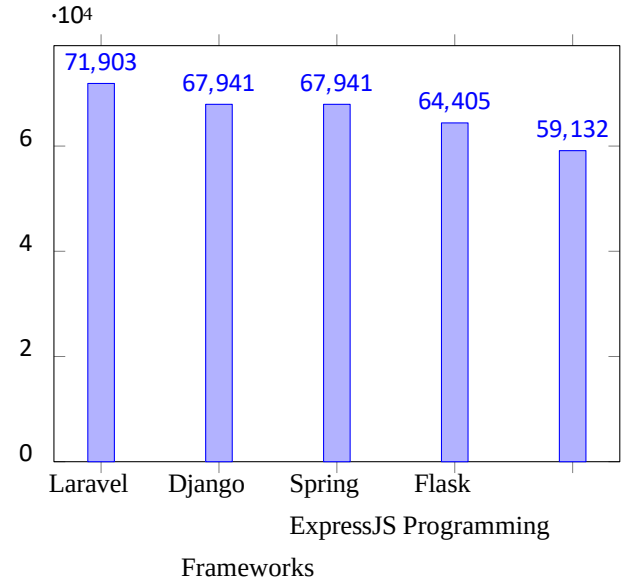

Fig. 2. Comparison of Number of Companies Using Various Programming Frameworks

C. Adoption Trends

Figure 3 illustrates the distribution of Spring Boot The distribution of Spring Boot clients by firm size as gathered via a survey in 2022 is shown in Figure 3, indicating the widespread use of Spring Boot in the sector.

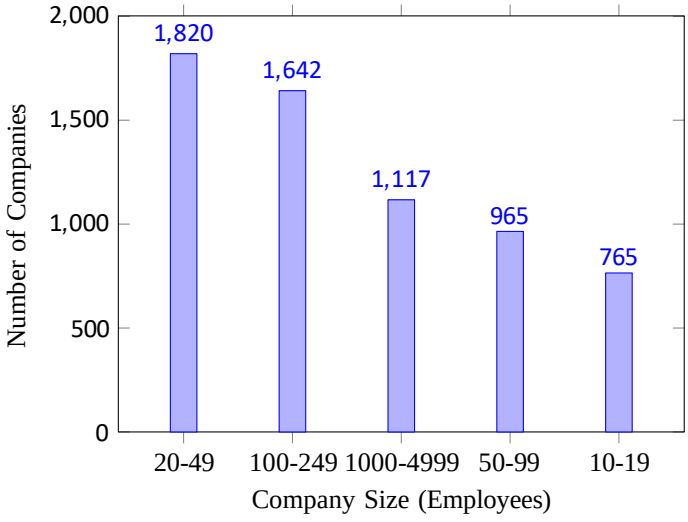

Fig. 3. Spring Boot Customers by Company Size

D. Big Tech Giants Using Spring Boot

Several prominent companies across various industries have adopted Spring Boot for their applications. Some of the notable users include:

- JPMorgan: Located in New York City, New York, United States, JPMorgan is one of the leading global financial services firms.
- National Auto Auction Association: Based in Frederick, Maryland, United States, this association represents the interests of auto auction businesses.
- Axel Springer Verlag AG: A major digital publishing house headquartered in Berlin, Germany.
- Capgemini SA: Headquartered in Paris, Ile-de-France, France, Capgemini is a global leader in consulting, technology services, and digital transformation.
- Palo Alto Networks, Inc.: Based in Santa Clara, California, United States, Palo Alto Networks is a global cybersecurity leader.
- American Express Company: A multinational financial services corporation headquartered in New York City, New York, United States.

VI. Conclusion

Spring Boot has demonstrated its effectiveness in simplifying the development of backend application which are Java-based through its auto-configuration and starter dependencies. By reducing configuration complexity and bundling essential libraries, Spring Boot enhances developer productivity and accelerates application development.The extensive use of Spring Boot by prominent companies such as JPMorgan, American Express, Palo Alto Networks, Springer highlights its reliability and versatility in various industry sectors. As a robust and scalable framework, Spring Boot is go-to framework for modern software development needs. Overall, Spring Boot's innovative features and industry acceptance affirm its pivotal role in the development of efficient and scalable applications.

References

[1] C. Walls, "Customizing Configuration," In book: Spring Boot in Action, 2016. Manning Publications Co., ISBN: 1617292540.
[2] C. Walls, "Testing with Spring Boot," In book: Spring Boot in Action, 2016. Manning Publications Co., ISBN: 1617292540.
[3] C. Walls, "Taking a Peek Inside with the Actuator," In book: Spring Boot in Action, 2016. Manning Publications Co., ISBN: 1617292540.
[4] M. Heckler, "Spring Boot: Up and Running," O'Reilly Media, 2021.
[5] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis, S. Deleuze, M. Simons, and others, "Spring Boot reference documentation," Retrieved June, vol. 22, pp. 36, 2020.
[6] H. Suryotrisongko, D. P. Jayanto, and A. Tjahyanto, "Design and Development of Backend Application for Public Complaint Systems Using Microservice Spring Boot," *Procedia Computer Science*, vol. 124, pp. 736–743, 2017. DOI: 10.1016/j.procs.2017.12.136.
[7] M. Zhang et al., "Intelligent business cloud service platform based on SpringBoot framework," in *2020 Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*, pp. 201–207, 2020. DOI: 10.1109/IPEC49694.2020.9115131.
[8] F. Gutierrez, "Deploying Spring Boot," In book: Pro Spring Boot, 2016. DOI: 10.1007/978-1-4842-1431-2_12.
[9] F. Gutierrez, "Spring Boot Actuator," In book: Pro Spring Boot, 2016. DOI: 10.1007/978-1-4842-1431-2_11.
[10] O. S. Gomez, R. R. Miranda, and K. C. Karen, K. C, "CRUDyLeaf: A DSL for Generating Spring Boot REST APIs from Entity CRUD Operations," *Cybernetics and Information Technologies*, vol. 20, pp. 3-14, 2020. DOI: 10.2478/cait-2020-0024.
[11] R. R. Rakshith and S. R. Swamy, "Review on Spring Boot and Spring Webflux for Reactive Web Development," *International Research Journal of Engineering and Technology (IRJET)*, vol. 7, no. 4, pp. 3834-3837, 2020.

[12] Z. Jovanovic, D. Jagodic, D. Vujicic, and S. Randic, "Java Spring Boot Rest Web Service Integration with Java Artificial Intelligence Weka Framework," in *UNITECH 2017 International Scientific Conference*, Gabrovo, 2017.
[13] K. Guntupally, R. Devarakonda, and K. Kehoe, "Spring Boot based REST API to Improve Data Quality Report Generation for Big Scientific Data: ARM Data Center Example," in *2018 IEEE International Confer- ence on Big Data (Big Data)*, pp. 5328-5329, 2018. DOI: 10.1109/Big- Data.2018.8621924.
[14] M. M. Garc´ıa, "Learn Microservices with Spring Boot," Jan. 2020. DOI: 10.1007/978-1-4842-6131-6.