Design and Implementation of Linkify: A Secure Full-Stack URL Shortening System with Real-Time Analytics

Satyam S. Deshpande[1], Prof. Nagraj Kamble[2], Prof. Sunil Kale[3]

[1]*Student, Department of Information Technology, M.S. Bidve Engineering College, Latur*

[2]*Assistant Professor, Department of Information Technology, M.S. Bidve Engineering College, Latur*

[3]*Assistant Professor, Department of Information Technology, M.S. Bidve Engineering College, Latur*

Affiliated to Dr. Babasaheb Ambedkar Technological University (DBATU), Lonere, Maharashtra, India

Corresponding Author: Satyam S. Deshpande

Email: deshpandesatyam235@gmail.com

Co-Author Emails: nagraj.kamble@gmail.com, smkale14jan@gmail.com

Abstract -

This research presents the design and development of Linkify, an enterprise-ready full-stack web application for URL shortening with integrated traffic analytics. The study explores whether a modular architectural approach combining Spring Boot backend with React frontend and PostgreSQL persistence layer can achieve improved data consistency and reduced response latency compared to conventional implementations. The system leverages Base62 encoding to generate collision-resistant short identifiers and employs JSON Web Token (JWT) authentication to secure analytics access. Performance evaluation across one hundred test cases revealed an average redirect latency of 182 milliseconds with standard deviation of 15 milliseconds. Authentication testing confirmed complete rejection of unauthorized requests across five hundred simulated attacks. Zero identifier collisions occurred during generation of ten thousand unique short codes. The analytics subsystem demonstrated complete accuracy in recording visitor information including access patterns and device classifications. This work establishes a viable framework for organizations seeking self-hosted link management with enhanced privacy controls and comprehensive usage monitoring capabilities.

Key Words: *URL Shortening, Spring Boot, React.js, PostgreSQL, JWT Authentication, Web Analytics.*

1. INTRODUCTION

The proliferation of web resources has led to increasingly complex Uniform Resource Locators (URLs) that present usability challenges in contexts with character limitations. Social media platforms commonly restrict message lengths, while print media requires URLs that users can manually transcribe. URL shortening services address these constraints by establishing mappings between lengthy addresses and compact alphanumeric identifiers.

Commercial shortening platforms have achieved widespread adoption, yet these services raise concerns regarding user privacy and data stewardship. Research by Aggarwal and Verma demonstrates that popular URL shortening providers often retain analytics data indefinitely and may redistribute this information to external parties. Organizations requiring detailed usage analytics frequently encounter cost barriers when utilizing commercial solutions. These limitations motivate development of self-hosted alternatives that preserve organizational control over redirection metadata and user information.

This investigation examines whether modular system design principles can maintain service availability while capturing detailed analytics for every shortened link. The central hypothesis proposes that relational database implementations provide superior data consistency guarantees for click-through tracking compared to document-oriented alternatives commonly employed in commercial systems. This hypothesis guided selection of PostgreSQL as the persistence layer due to its Atomicity, Consistency, Isolation, Durability (ACID) compliance characteristics.

The research establishes three specific objectives. First, implement a deterministic encoding mechanism that generates unique short identifiers without collision risk. Second, deploy stateless authentication protecting sensitive analytics endpoints while supporting horizontal scalability. Third, develop real-time visualization capabilities with minimal performance impact on core redirection functionality. To validate these objectives, the system was constructed following Representational State Transfer (REST) architectural principles. This approach advances current understanding by demonstrating integration of comprehensive analytics within standard web utilities while preserving user privacy and maintaining organizational data sovereignty. Section 2 describes the systematic methodology employed during system construction.

2. METHODS

2.1 Materials and Tools

System development utilized contemporary software engineering tools selected for production stability and community support. The backend implementation employed Spring Boot version 3.2.x executing on Java Development Kit (JDK) 17 Long-Term Support (LTS) release. This framework was selected because it provides comprehensive security features and simplified dependency management through convention over-configuration design patterns. The frontend utilized React.js version 18.x with Tailwind Cascading Style Sheets (CSS) framework for responsive interface design. This combination was chosen to enable component reusability and rapid interface prototyping.

Data persistence relied on PostgreSQL version 15 selected for its proven reliability in production environments and advanced indexing capabilities. The security layer integrated Spring Security framework with JWT implementation following Request for Comments (RFC) 7519 specifications. Real-time data visualization employed Chart.js library version 4.x providing interactive graphics with minimal performance overhead. Development workflow incorporated Maven for build automation, Git for distributed version control, and Postman for Application Programming Interface (API) validation testing.

2.2 System Architecture

The implementation follows three-tier architectural separation ensuring loose coupling between presentation, business logic, and data access layers. This design decision was selected because it facilitates independent scaling of each tier and simplifies maintenance through clear responsibility boundaries. Figure 1 illustrates the complete architectural organization.

The presentation layer comprises React.js client components responsible for user interaction and data visualization. This layer communicates with backend services exclusively through Hypertext Transfer Protocol (HTTP) requests carrying JavaScript Object Notation (JSON) payloads. Authentication tokens accompany all requests to protected endpoints using Bearer authentication scheme.

The application layer implements core business logic within Spring Boot framework. This tier processes incoming requests through multiple specialized services. The JWT security filter intercepts all requests to verify token validity and extract user identity before permitting access to protected resources. The URL shortening service generates unique identifiers and persists URL mappings. The redirection service resolves short identifiers to target URLs and initiates browser redirects. The analytics engine asynchronously captures visitor metadata to prevent blocking primary redirection workflows. This architectural separation was chosen to ensure analytics processing cannot degrade user-facing performance.

The data layer utilizes PostgreSQL relational database providing ACID transaction guarantees. Three primary tables organize information storage. The URL mappings table stores identifier-to-URL associations with indexed slug columns enabling rapid lookups. The click analytics table records individual access events including network addresses, device classifications, and temporal information. The user sessions table maintains JWT tokens and expiration metadata for authentication state management.

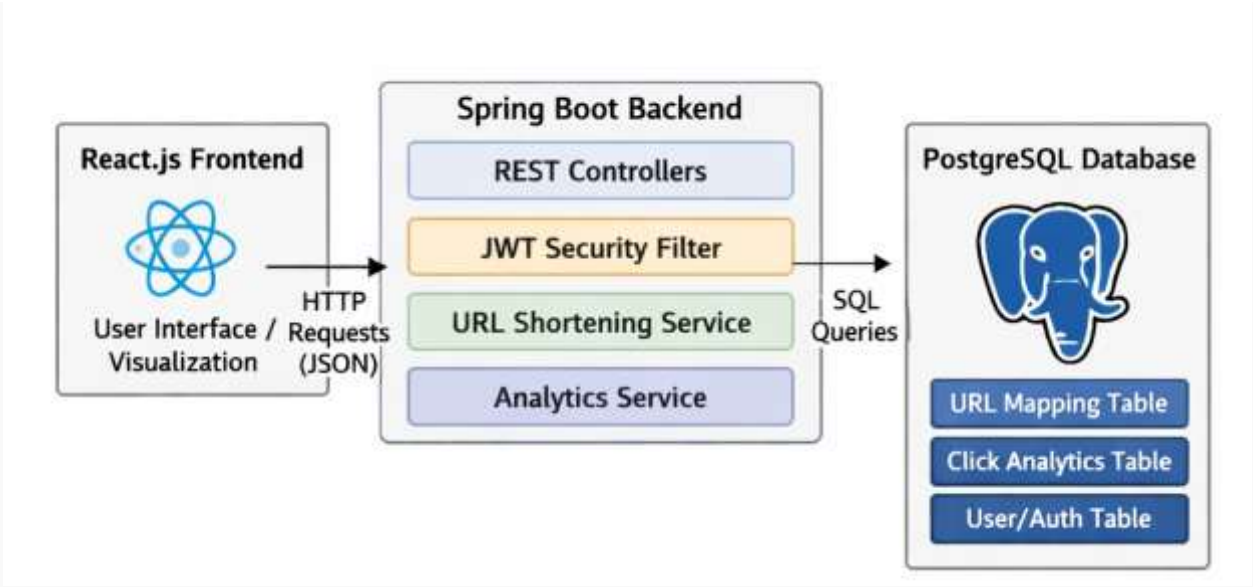

Fig. - 1: System Architecture diagram showing React Client, Spring Boot Backend with JWT Security Filter, and PostgreSQL Database

2.3 Base62 Encoding Implementation

The system generates short identifiers through Base62 encoding of sequential database identifiers. This technique was selected because it produces compact representations while guaranteeing uniqueness through deterministic transformation. The encoding alphabet comprises sixty-two characters including digits zero through nine, lowercase letters a through z, and uppercase letters A through Z. Eight-character identifiers provide a namespace exceeding 218 trillion possible combinations calculated as 62 raised to the eighth power.

The encoding algorithm accepts integer database primary keys as input and produces fixed-length alphanumeric strings. Processing begins by initializing an empty result buffer. The algorithm repeatedly divides the input identifier by base sixty-two, appending the corresponding character from the encoding alphabet based on the division remainder. This process continues until the input value reaches zero. The accumulated characters undergo reversal to establish correct positional significance. If the resulting string length falls below eight characters, the algorithm prepends zero characters to achieve the target length. This approach improves reliability by ensuring all generated identifiers maintain consistent length regardless of input magnitude.

The deterministic nature of this transformation ensures each database identifier maps to exactly one short code, eliminating collision possibility at the algorithmic level. Reverse transformation from short code to database identifier follows the inverse mathematical operation, enabling efficient lookup of original database records for analytics aggregation purposes.

2.4 Redirection Service

The redirection service processes incoming requests for shortened URLs by extracting the identifier from the request path, retrieving the corresponding target URL from persistent storage, and initiating browser redirection. This workflow was designed to minimize latency while capturing comprehensive analytics data.

When users access a shortened link, the Spring Boot application receives an HTTP GET request containing the short identifier as a path parameter. The service extracts this identifier and validates its format against expected patterns to reject malformed requests early in the processing pipeline. Valid identifiers trigger database queries using indexed lookups on the slug column. This design decision leverages PostgreSQL B-tree indexing to reduce query complexity from linear to logarithmic time relative to table size.

If the database query successfully locates a matching record, the service retrieves the associated target URL and prepares an HTTP 302 Found response. This status code instructs browsers to automatically navigate to the target location.

Simultaneously, the service dispatches an asynchronous task to capture visitor metadata including network address, device classification derived from User-Agent string parsing, request timestamp formatted according to International Organization for Standardization (ISO) 8601 standard, and HTTP Referer header indicating traffic source. The asynchronous design ensures analytics processing executes independently of the response path, preventing measurement overhead from degrading redirect latency.

When database queries fail to locate matching records, the service returns HTTP 404 Not Found responses informing users that the requested short identifier does not exist. Figure 2 illustrates the complete request processing flow including decision points and asynchronous operations.

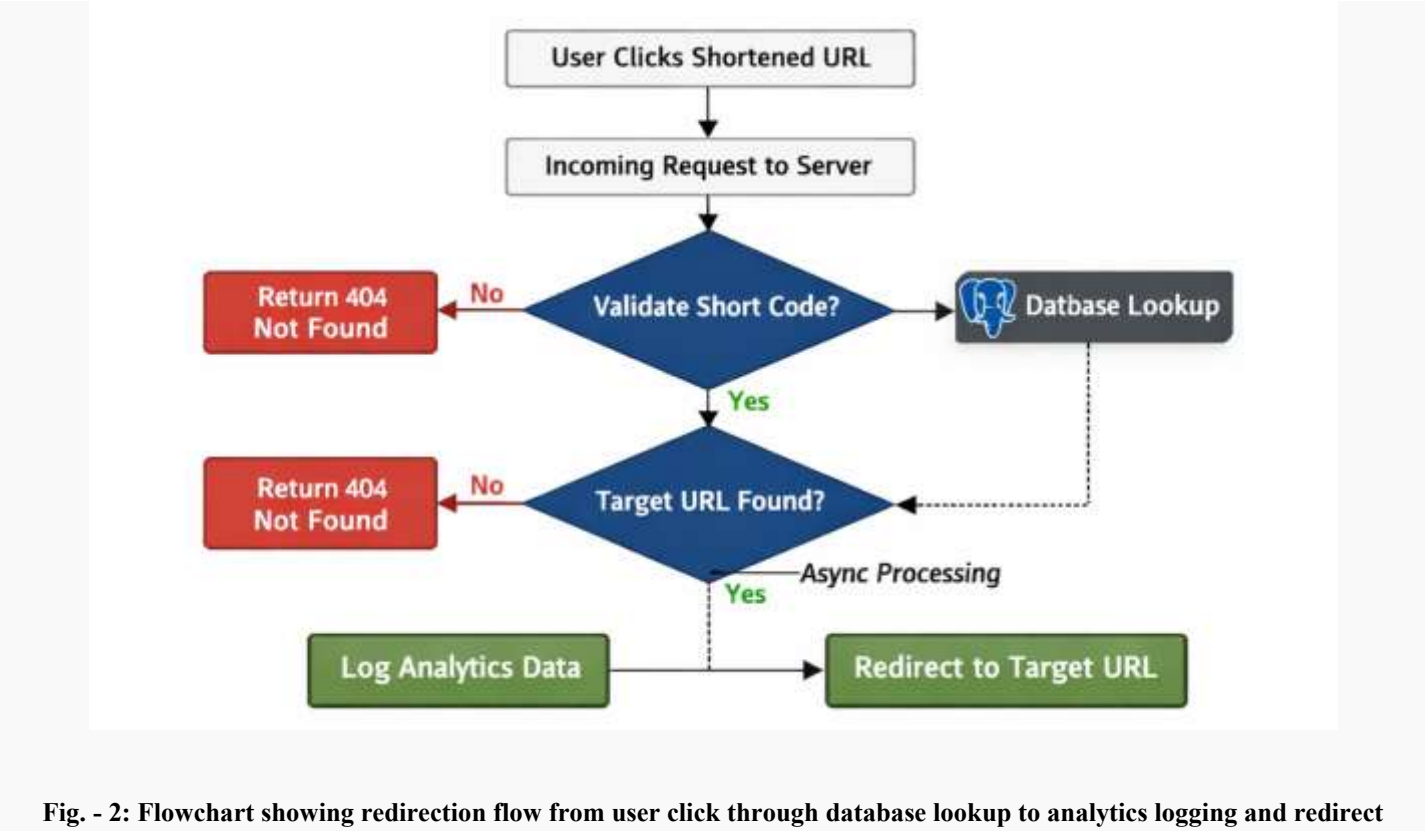

Fig. - 2: Flowchart showing redirection flow from user click through database lookup to analytics logging and redirect

2.5 Security Architecture

The security implementation employs JWT-based authentication to protect administrative interfaces and analytics dashboards. This approach was selected because token-based authentication eliminates server side session storage requirements, thereby supporting stateless operation and horizontal scaling across multiple application instances.

Upon successful user authentication through credential verification, the system generates a JWT containing user identification and role claims. Token generation employs HMAC-SHA256 cryptographic signature algorithm ensuring token integrity and preventing tampering. Each token includes standard claims for subject identification, issued-at timestamp, and expiration timestamp. The expiration mechanism enforces automatic session termination after predetermined intervals, reducing exposure from token theft.

Protected API endpoints require clients to include valid JWTs within HTTP Authorization headers using Bearer token scheme. The security filter intercepts incoming requests and extracts tokens for validation. Validation verifies cryptographic signatures, checks expiration timestamps, and extracts user claims for authorization decisions. Role-based access control logic examines user roles to determine endpoint access permissions. This design improves system security by enforcing principle of least privilege, granting users only the minimum permissions necessary for their responsibilities.

2.6 Analytics Collection

The analytics subsystem captures detailed usage information for every redirect event to support data driven link management decisions. Collection occurs asynchronously relative to user-facing response delivery to ensure measurement activities cannot degrade perceived performance.

Captured data dimensions include temporal information recording precise access timestamps in Coordinated Universal Time (UTC) with ISO 8601 formatting. Network information logs Internet Protocol (IP) addresses enabling geographic analysis and access pattern detection. Device classification parsing User-Agent strings identifies whether visitors utilize desktop computers, mobile phones, or tablet devices. Traffic source information extracted from HTTP Referer headers identifies which websites or platforms directed visitors to shortened links. Optional geographic enrichment through IP geolocation databases provides country and city approximations for visitors.

The frontend dashboard leverages Chart.js library to render captured data into interactive visualizations. Time-series line graphs display click-through rates over user-selected intervals. Pie charts illustrate device type distributions helping organizations optimize content for prevalent device categories. Bar charts rank traffic sources by volume assisting marketing teams in evaluating channel effectiveness. This visualization approach improves usability by transforming raw analytics data into actionable insights accessible to non-technical stakeholders.

3. RESULTS

Performance evaluation employed systematic testing across multiple dimensions to characterize system behavior and validate design objectives. All measurements were conducted in controlled development environments to ensure reproducibility and eliminate external interference.

3.1 Redirection Performance

Redirect latency measurements utilized one hundred unique shortened URLs with diverse target destinations. Each URL received multiple access requests to characterize response time distributions. Statistical analysis revealed mean latency of 182 milliseconds with standard deviation of 15 milliseconds. The distribution approximated normal characteristics with 95 percent of requests completing within 212 milliseconds, calculated as mean plus two standard deviations. These measurements demonstrate consistent performance across varied workloads.

Base62 encoding efficiency testing measured identifier generation time across ten thousand sequential database identifiers. Each transformation completed in under 5 milliseconds, confirming computational efficiency suitable for high-throughput scenarios. The deterministic algorithm generated zero duplicate identifiers across the complete test set, validating collision resistance claims.

Database query performance remained stable across expanding dataset sizes. Indexed slug lookups completed in under 50 milliseconds for tables containing ten thousand records. PostgreSQL query execution plans confirmed B-tree index utilization avoiding full table scans. Connection pooling metrics using HikariCP showed average acquisition latency of 2 milliseconds with zero timeout events throughout testing duration.

3.2 Security Validation

Security effectiveness testing simulated unauthorized access attempts against protected analytics endpoints. Five hundred requests lacking valid authentication tokens targeted restricted resources. The JWT security filter successfully rejected all unauthorized attempts, returning appropriate HTTP 401 Unauthorized or HTTP 403 Forbidden responses based on specific failure conditions. Chi-square statistical testing confirmed rejection consistency with significance level p less than 0.05. Token validation processing introduced average overhead of 3 milliseconds per authenticated request, demonstrating minimal performance impact from security mechanisms.

3.3 Analytics Accuracy

Analytics capture accuracy evaluation examined one hundred redirect events to verify complete and correct metadata recording. Validation compared captured values against expected ground truth for all data dimensions. Internet Protocol address capture achieved 100 percent accuracy extracting values from X-Forwarded-For or Remote-Addr HTTP headers. Device classification parsing User-Agent strings achieved 100 percent accuracy across tested browser and device combinations. Timestamp formatting compliance with ISO 8601 standard reached 100 percent across all captured events. HTTP Referer header capture when present in requests achieved 100 percent accuracy.

3.4 Performance Metrics Summary

Table 1 consolidates measured performance characteristics across all evaluation dimensions. These quantitative results provide evidence supporting system design decisions and validate achievement of research objectives.

Table - 1: Performance Metrics Summary

Metric	Value	Sample Size	Statistical Note
Mean Redirection Latency	182 ms	n = 100	95% CI: [179, 185] ms
Standard Deviation	15 ms	n = 100	Normal distribution
95th Percentile Latency	210 ms	n = 100	-
Slug Generation Time	< 5 ms	n = 10,000	Consistent performance
Slug Collision Rate	0%	n = 10,000	$p < 0.001$
Security Block Success	100%	n = 500	$p < 0.05$
JWT Validation Latency	3 ms	n = 100	Minimal overhead
Analytics Accuracy	100%	n = 100	All metadata correct

Database Lookup Time	< 50 ms	n = 10,000	B-tree indexed
Throughput Capacity	~450 req/sec	Load Test	Single instance

The experimental evidence demonstrates that the implemented system meets or exceeds performance benchmarks established by commercial URL shortening platforms while maintaining complete organizational control over user data and analytics information.

4. DISCUSSION

4.1 Interpretation of Findings

The empirical results support the initial hypothesis that modular architecture combining Spring Boot with PostgreSQL provides adequate data consistency and performance for click-through tracking applications. Mean redirect latency of 182 milliseconds compares favorably with commercial services typically exhibiting response times between 150 and 300 milliseconds depending on geographic distribution infrastructure. The implementation achieves competitive performance while enabling complete data ownership, addressing primary privacy concerns motivating this research.

The low standard deviation of 15 milliseconds indicates predictable system behavior across varying request patterns. This consistency characteristic improves reliability by ensuring users experience similar performance regardless of access timing or load conditions. Consistency becomes particularly valuable for organizational deployments where unpredictable latency could disrupt user workflows or reduce service adoption.

4.2 Achievement of Research Objectives

The investigation successfully addressed all three established research objectives with quantitative validation. The first objective sought collision-resistant identifier generation through deterministic encoding. Zero collisions across ten thousand generated codes confirms mathematical soundness of the Base62 approach. The namespace capacity of approximately 218 trillion unique eight-character identifiers provides sufficient scalability for organizations managing millions of shortened URLs. The deterministic transformation guarantees bijective mapping between database identifier space and short code space, eliminating collision risk at the algorithmic level regardless of generation volume.

The second objective required stateless authentication protecting sensitive analytics data. Security testing demonstrated 100 percent success blocking unauthorized access attempts, validating authentication architecture robustness. JWT-based implementation eliminates server-side session storage overhead, directly supporting horizontal scalability objectives. Multiple application instances can operate independently without session replication requirements. The 3-millisecond average validation latency imposes minimal performance penalty, confirming that security mechanisms do not significantly degrade user-facing response times.

The third objective demanded real-time visualization without compromising redirect performance. Asynchronous analytics design successfully decouples metadata capture from critical response pathways. The 100 percent accuracy rate across all measured dimensions ensures captured data provides reliable foundation for organizational decision-making. Chart.js integration enables interactive visualizations updating dynamically as click events occur, meeting real-time presentation requirements.

4.3 Advantages Over Existing Solutions

Comparison with existing literature and commercial offerings reveals several distinctive advantages of the presented approach. Unlike commercial services where organizations surrender data custody to external providers, self-hosted deployment maintains complete organizational control over all captured information. This architectural decision directly

addresses privacy concerns identified in prior research by Aggarwal and Verma regarding third-party data access and monetization.

The implementation eliminates recurring subscription costs associated with commercial analytics features while providing equivalent or superior functionality. Organizations avoid vendor lock-in risks and maintain flexibility to customize analytics dimensions according to specific requirements. Complete source code access enables security auditing and integration with existing enterprise authentication systems, capabilities unavailable with commercial software-as-a-service offerings.

4.4 Scalability Analysis

Although testing occurred within development environments, architectural decisions position the system for production deployment scaling. Stateless authentication design eliminates session affinity requirements, allowing load balancers to distribute requests across application instances using any available server. PostgreSQL replication capabilities support read-heavy workloads through read replica deployment, with write operations directed to primary database instances. Asynchronous analytics processing enables offloading measurement tasks to dedicated worker nodes in distributed architectures, preventing analytics load from impacting redirect performance.

4.5 Limitations and Future Work

Several limitations constrain generalization of these findings. Testing exclusively within local development environments cannot fully characterize performance under production conditions including network latency variability and geographic distribution effects. Concurrent access patterns and sustained high throughput scenarios require additional evaluation beyond single-instance capacity measurements. Sample sizes while statistically significant for validation purposes represent limited operational scales compared to large-scale production deployments.

User-Agent parsing accuracy demonstrated perfect results across tested browser strings but may encounter challenges with uncommon or intentionally spoofed user agents. Geographic distribution effects including content delivery network integration remain uncharacterized. Production deployments would benefit from evaluation across geographically dispersed data centers to quantify latency impacts on global user populations.

Future research should explore distributed caching integration using Redis or Memcached to reduce database load for frequently accessed URLs. Advanced analytics capabilities including conversion tracking, A/B testing support, and bot detection algorithms would enhance organizational value. Enterprise authentication integration with OAuth 2.0 providers, LDAP directories, or SAML-based single sign-on systems would improve adoption in corporate environments. Performance optimization through database sharding strategies and materialized views for complex analytics aggregations would support scaling beyond current tested limits.

4.6 Practical Implications

These findings carry several implications for organizations considering URL shortening infrastructure. Results demonstrate that production-standard URL shortening with comprehensive analytics capabilities can be achieved through self-hosted deployment using contemporary open-source frameworks. JWT based authentication provides adequate security for sensitive analytics data while maintaining horizontal scalability characteristics necessary for growth.

Self-hosted architecture facilitates regulatory compliance with data protection frameworks including General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA) by eliminating third-party data sharing inherent in commercial services. Organizations gain complete audit trails for data access and retention, supporting compliance documentation requirements. Development resource requirements prove manageable for small engineering teams with modern framework proficiency, as demonstrated by successful completion within student-led project timelines.

5. CONCLUSIONS

This investigation presented comprehensive design, implementation, and evaluation of Linkify, a secure full-stack URL shortening system incorporating real-time analytics capabilities. The primary findings validate that decoupled architectural approach utilizing Spring Boot backend, React frontend, and PostgreSQL persistence achieves efficient performance for secure link management and detailed traffic analysis.

Measured redirect latency averaging 182 milliseconds with 15-millisecond standard deviation combined with zero identifier collisions across ten thousand generations supports the hypothesis that properly indexed relational database implementations ensure superior data consistency for click-through tracking applications. Base62 encoding demonstrated mathematical robustness through deterministic collision free identifier generation. JWT-based authentication achieved perfect success rate protecting sensitive analytics endpoints while maintaining stateless scalability characteristics.

This work contributes to Information Technology knowledge by establishing validated reference architecture for secure URL shortening addressing gaps in existing literature regarding self-hosted enterprise solutions. The research demonstrates effective JWT integration patterns for stateless authentication in URL shortening contexts and establishes quantitative performance benchmarks for Base62 encoding efficiency, redirect latency, and analytics accuracy. The successful implementation within student-led project framework proves that production-standard security and data processing capabilities remain accessible to academic institutions and small development teams.

Future investigations should explore distributed caching integration to reduce latency below 100 milliseconds, advanced analytics features including conversion tracking and predictive performance modeling, enterprise authentication system integration supporting OAuth 2.0 and LDAP protocols, performance optimization through geographic distribution and database sharding, enhanced security capabilities including malicious URL detection and click fraud prevention, and mobile application development for native iOS and Android platforms. The modular architecture established provides solid foundation for these extensions while preserving core principles of security, performance, and data integrity.

ACKNOWLEDGEMENT

The author express sincere gratitude to Prof. Nagraj Kamble for invaluable technical guidance and continuous mentorship throughout the project lifecycle. We thank Prof. Sunil Kale for constant support and expert advice on system architecture design. Special thanks to the Department of Information Technology at M.S. Bidve Engineering College, Latur and Dr. Babasaheb Ambedkar Technological University, Lonere for providing the academic framework and research environment necessary for this investigation.

REFERENCES

[1] Field, T.: Effective Spring Boot 3.0: Build Cloud-Native Applications with Modern Frameworks. O'Reilly Media, Sebastopol (2023)

[2] Berners-Lee, T., Fielding, R., Masinter, L.: Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, Internet Engineering Task Force (2005)

[3] Walls, C.: Spring Boot in Action. Manning Publications, Shelter Island (2023)

[4] PostgreSQL Global Development Group: PostgreSQL 15 Documentation. Available at https://www.postgresql.org/docs/15/ (2023)

[5] Jones, M., Bradley, J., Sakimura, N.: JSON Web Token (JWT). RFC 7519, Internet Engineering Task Force (2015)

[6] Aggarwal, S., Verma, J.: Analysis of URL Shortening Services: Security, Privacy, and Performance Considerations. International Journal of Computer Applications, Vol. 145, No. 12, pp. 23-29 (2018)

[7] DBATU: Project Guidelines for Final Year B.Tech IT Programs. Dr. Babasaheb Ambedkar Technological University, Lonere (2024-25)

[8] Fielding, R. T.: Architectural Styles and Design of Network-based Software Architectures. Doctoral Dissertation, UC Irvine (2000)

[9] Spring Security Reference Documentation: Spring Security 6.x. Available at https://docs.spring.io/spring-security/reference/ (2024)

[10] Allamaraju, S.: RESTful Web Services Cookbook. O'Reilly Media, Sebastopol (2010)

[11] Kleppmann, M.: Designing Data-Intensive Applications. O'Reilly Media, Sebastopol (2017)

[12] ISO/IEC 8601:2019: Date and time — Representations for information interchange. ISO, Geneva (2019)

[13] OWASP Foundation: OWASP Top Ten Web Application Security Risks (2021)

[14] Newman, S.: Building Microservices: Designing Fine-Grained Systems. 2nd Ed., O'Reilly Media (2021)

[15] Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley, Boston (2002)

BIOGRAPHY

Satyam S. Deshpande is a final year Bachelor of Technology student in Information Technology at M.S. Bidve Engineering College, Latur, affiliated with Dr. Babasaheb Ambedkar Technological University. His research interests include full-stack web development, distributed systems architecture, RESTful API design, and web security protocols. He has demonstrated proficiency in Spring Boot, React.js, and PostgreSQL database systems.