SECURITY ARCHITECTURE AND DATA MANAGEMENT IN A MICROSERVICE-BASED CAREER GUIDANCE PLATFORM

Eyvazli V.
Alekberov C.
Xalilov F.
Nuraliyeva F.
Master's degree student, no academic rank, Master's student, Department of Applied Mathematics and Artificial Intelligence, Azerbaijan Technical University, Baku, Azerbaijan
https://doi.org/10.5281/zenodo.19573283

Abstract

This paper presents the security architecture and data management approach implemented in Career Bridge, a microservice-based platform designed to match university graduates with relevant employment opportunities. The study addresses two interconnected aspects of the system: the multi-layered information security model and the structured relational data storage strategy. The security implementation is based on the CIA triad, JSON Web Token authentication, role-based access control, and OWASP API Security Top 10 guidelines. The data layer employs a database-per-service isolation pattern using cloud-hosted PostgreSQL via the Neon DBaaS platform, with Spring Data JPA and Hibernate managing object-relational mapping. The paper describes the rationale behind each design decision, the identified threats relevant to the platform, and the mechanisms applied to mitigate them. The proposed architecture demonstrates how security and data integrity can be effectively embedded into a distributed system from the design stage.

Keywords: information security, JWT authentication, OWASP, microservice architecture, PostgreSQL, DBaaS, Spring Security, role-based access control.

Introduction

The increasing adoption of distributed software platforms for career guidance and recruitment introduces significant challenges in the areas of data protection and system integrity. Platforms that process personal information about students — including educational background, technical skills, and career preferences — must ensure that this data is protected from unauthorized access, modification, and disclosure. At the same time, the distributed nature of microservice architectures requires a carefully designed data management strategy to maintain consistency, isolation, and performance across independent service boundaries.

Career Bridge is a microservice-based platform developed to address the employment gap faced by university graduates and junior developers by automating the matching of student competency profiles with employer vacancy requirements. The platform consists of five independent microservices responsible for profile management, vacancy storage, skill matching, notification delivery, and external vacancy integration. Given that the system processes sensitive personal data and operates across multiple independently deployed components, security and data management constitute foundational architectural concerns rather than supplementary features.

This paper focuses specifically on these two aspects of the Career Bridge platform. The security section describes the threat model, the adopted authentication and authorization mechanisms, and their alignment with established standards. The data management section describes the storage architecture, the schema isolation strategy, and the tools employed to ensure efficient and reliable data access. Together, these elements form the technical backbone of a platform designed to operate securely and consistently in a cloud-hosted environment.

1. Information Security Architecture

1.1. Threat Model and Theoretical Foundation

The security design of Career Bridge is grounded in the CIA triad — a foundational framework in information security that defines three core properties any secure system must preserve. Confidentiality ensures that personal data is accessible only to authorized parties. Integrity guarantees that data cannot be altered without authorization. Availability ensures that the platform remains accessible to legitimate users at all times [1, p. 38].

In the context of Career Bridge, confidentiality is enforced through JWT-based authentication and HTTPS transport encryption, ensuring that student profile data is protected both in transit and at the application layer. Integrity is maintained through input validation via Spring Validation and the use of parameterized JPA queries, which prevent injection-based data manipulation. Availability is supported through Docker containerization and cloud deployment on Render.com, which provides automatic service restart capabilities and infrastructure resilience [2, p. 14].

The principle of security by design, described by Ransome and Misra, was adopted as a guiding methodology [3, p. 62]. Rather than treating security as a post-development concern, all protective mechanisms were integrated into the system architecture during the initial design phase. This approach ensures that security boundaries align with service boundaries, reducing the attack surface of each individual component.

1.2. Authentication Mechanism: JSON Web Token

Authentication in Career Bridge is implemented using JSON Web Token, a stateless token-based mechanism that allows each microservice to independently verify the identity of a requesting party without maintaining a shared session store. When a user authenticates via the login endpoint, the server generates a signed JWT containing the user identifier and assigned role. This token is returned to the client and must be included in the Authorization header of all subsequent requests.

Each microservice validates the incoming token by verifying its cryptographic signature using a shared secret key. This eliminates the need for a centralized session management component and reduces inter-service dependency. A systematic literature review by de Almeida et al., covering more than fifty primary studies on authentication patterns in microservice architectures, identified the JWT combined with an Identity Provider as the most widely adopted and practically effective approach for distributed systems [4, p. 7]. This finding directly informed the authentication design of Career Bridge.

The token contains three components: a header specifying the signing algorithm, a payload encoding the user identity and role claims, and a cryptographic signature. The HS256 algorithm is used for signing. Tokens are configured with a fixed expiration period, after which re-authentication is required. This limits the validity window of any compromised credential.

1.3. Authorization and Role-Based Access Control

Career Bridge implements role-based access control with two defined roles: STUDENT and ADMIN. The role is embedded in the JWT payload and is evaluated at the service layer using Spring Security's method-level authorization annotations. Public endpoints — specifically the registration and login routes — are accessible without authentication. All other endpoints require a valid token.

The most critical authorization concern in the platform is the prevention of Broken Object Level Authorization, identified as the top-ranked vulnerability in the OWASP API Security Top 10 2023 report [5]. This vulnerability occurs when a user can access resources belonging to other users by manipulating object identifiers in API requests. In Career Bridge, this is mitigated by extracting the authenticated user's identifier from the security context and comparing it against the resource identifier in each request. A student user requesting the profile of another student receives a 403 Forbidden response regardless of whether the target resource exists.

Administrative operations — including user enumeration and deletion — are restricted to the ADMIN role. This separation ensures that student-facing endpoints cannot be used to enumerate or manipulate other users' data, even if a valid token is present.

1.4. Infrastructure Security

All communication between clients and the platform is encrypted via HTTPS, enforced by the Render.com deployment environment. Database connections to Neon PostgreSQL require SSL with the channel_binding parameter set to require, ensuring that no unencrypted connection can be established between the application layer and the data layer.

Application secrets — including the JWT signing key, database credentials, and service configuration — are stored exclusively as environment variables on the deployment platform and are never present in the source code or version control history. This practice aligns with the Twelve-Factor App methodology for configuration management and eliminates a common source of credential exposure in cloud-hosted applications.

Mamidi identifies these practices — HTTPS enforcement, secret management through environment variables, and input validation at every API boundary — as the essential baseline for securing REST APIs built on Spring Boot [6, p. 4]. All three elements are present in the Career Bridge implementation.

2. Data Management Architecture

2.1. Database Isolation Strategy

Career Bridge adopts the database-per-service pattern, in which each microservice maintains its own isolated schema within the cloud-hosted PostgreSQL instance. This pattern, described by Richardson as a fundamental requirement for genuine service independence, ensures that no service can directly access or modify the data of another service [7, p. 103]. All cross-service data access occurs exclusively through defined API contracts.

The platform maintains five isolated schemas: student_schema for user profiles and skill records, vacancy_schema for vacancy data and required competencies, matching_schema for computed matching results and scores, notif_schema for notification records, and parser_schema for imported vacancy data awaiting processing. This isolation reduces coupling between services and ensures that a failure or schema change in one service does not propagate to others.

The underlying database infrastructure is provided by Neon, a serverless PostgreSQL platform that eliminates the operational overhead of database server administration. Backup, monitoring, and scaling operations are managed by the provider, allowing the development team to focus on application logic. Kleppmann notes that the choice of data storage technology in distributed applications has direct implications for scalability, reliability, and consistency guarantees [8, p. 89]. The Neon DBaaS model was selected in recognition of these implications, as it provides a managed PostgreSQL environment with built-in high availability and SSL-enforced connections.

2.2. Object-Relational Mapping and Query Optimization

Data access across all services is implemented using Spring Data JPA with Hibernate as the underlying ORM provider. This combination enables declarative repository definitions, reducing the volume of boilerplate data access code and allowing the development team to express queries at the domain object level rather than the SQL level.

A significant performance consideration in the vacancy service is the management of the skill association

between vacancy records and their required competencies. Each competency is stored as an individual record in a related table rather than as a delimited string field, enabling the matching service to perform accurate set-based comparisons. However, this design introduces the risk of the N+1 query problem, in which loading a collection of vacancies triggers an individual database query for each associated skill set. Mihalcea identifies this as the most prevalent performance issue in Hibernate-based applications [9, p. 215]. The platform addresses this through the use of JOIN FETCH clauses and batch loading strategies, consolidating related data retrieval into a single query where possible.

The Specification API provided by Spring Data JPA is employed in the vacancy service to support dynamic multi-criteria filtering. This allows queries to be constructed programmatically based on the parameters present in a given request, without requiring a separate repository method for each possible filter combination. The result is a flexible and maintainable query layer that can accommodate evolving search requirements without structural refactoring.

2.3. Connection Management and Data Integrity

Database connections are managed through HikariCP, the default connection pool implementation provided by Spring Boot. A configured pool of persistent connections is maintained and reused across requests, avoiding the overhead of establishing new connections for each database interaction. Pool parameters are tuned to balance resource utilization with connection availability under expected load conditions.

All mutating operations in the platform are executed within transactional boundaries managed by Spring's declarative transaction framework. The @Transactional annotation is applied to service-layer methods to ensure that related operations either complete atomically or are rolled back in their entirety. Read-only transactions are explicitly marked to enable database-level optimizations. PostgreSQL's MVCC concurrency model ensures that concurrent read and write operations do not interfere with each other, supporting the simultaneous use of the platform by multiple users without locking conflicts.

Schema normalization is applied throughout the data model to eliminate redundancy and ensure data integrity. The first, second, and third normal forms are observed: skill associations are stored in dedicated relational tables rather than denormalized into parent records, foreign key relationships enforce referential integrity, and no transitive dependencies exist within any table structure. Delaney emphasizes that correct normalization is a prerequisite for both query performance and long-term data consistency in relational systems [10, p. 178].

Conclusion

This paper has described the security architecture and data management design of the Career Bridge platform. The security model integrates JWT-based stateless authentication, role-based access control, BOLA mitigation through ownership verification, HTTPS enforcement, and environment-based secret management. Each of these mechanisms directly addresses specific threat categories identified in the OWASP API Security Top 10 standard and is grounded in established principles from the information security literature.

The data management architecture employs the database-per-service isolation pattern, cloud-hosted PostgreSQL via Neon DBaaS, Spring Data JPA with Hibernate for object-relational mapping, and HikariCP for connection pool management. The design addresses known performance challenges including the N+1 query problem and supports flexible multi-criteria querying through the Specification API. Normalization principles are consistently applied to maintain data integrity and minimize redundancy.

Together, these architectural choices demonstrate that security and data integrity can be effectively embedded into a distributed microservice platform from the earliest stages of design. Future work will focus on extending the security model to include OAuth2 integration, API rate limiting, and automated security scanning within the CI/CD pipeline, as well as evaluating the introduction of read replicas and caching layers to further improve data access performance under increased load.

References:

1. Mattord H. J., Whitman M. E. Principles of Information Security. 7th ed. Boston: Cengage Learning; 2022. 752 p.

2. Kothapalli M. Securing microservices architecture: best practices and challenges. Journal of Scientific and Engineering Research. 2021; 8(10): 187–192.

3. Ransome J., Misra A. Core Software Security: Security at the Source. Boca Raton: CRC Press; 2021. 380 p.

4. de Almeida M. G., Silva R. A., Pereira T. F. Authentication and authorization in microservices architectures: a systematic literature review. Applied Sciences. 2022; 12(6): 3023. https://doi.org/10.3390/app12063023

5. OWASP Foundation. OWASP API Security Top 10 – 2023. Available at: https://owasp.org/API-Security/ (accessed 10.03.2025).

6. Mamidi S. Building secure REST APIs in Spring Boot: techniques and tools. ResearchGate Preprint. 2025. Available at: https://www.ijetcsit.org/index.php/ijetcsit/article/download/561/502

7. Richardson C. Microservices Patterns: With Examples in Java. Shelter Island: Manning Publications; 2019. 520 p.

8. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol: O'Reilly Media; 2017. 611 p.

9. Mihalcea V. High-Performance Java Persistence. Self-published; 2021. 474 p.

10. Delaney K. Introducing Microsoft SQL Server 2019. Redmond: Microsoft Press; 2019. 264 p.