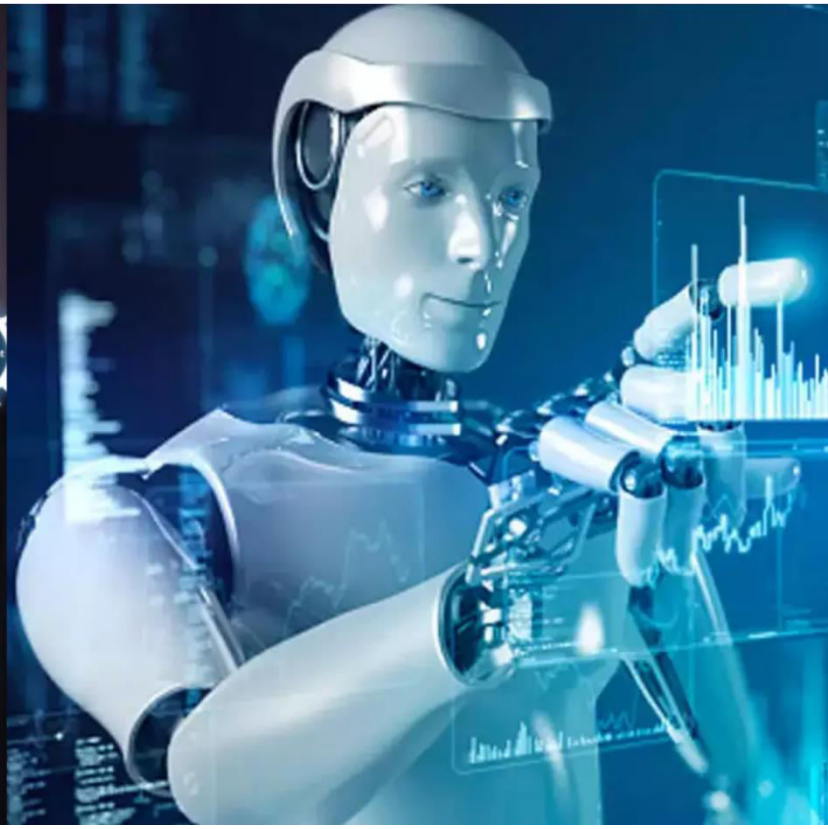
IJIRSET
International Journal of Innovative Research in Science | Engineering | Technology

ISSN(O): 2319-8753
ISSN(P): 2347-6710

International Journal of Innovative Research in Science Engineering and Technology (IJIRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.699

Volume 14, Issue 3, March 2025

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 3, March 2025

|DOI: 10.15680/IJIRSET.2025.1403334|

A Comparative Analysis of Modern Javascript/Typescript Orms: Performance, Type Safety, and Developer Experience

Himanshu Suthar, Mr Suraj Singh

Department of Computer Science & Engineering, Parul Institute of Technology, Parul University, Gujarat, India

ABSTRACT: The proliferation of database-driven web applications built with JavaScript and TypeScript has necessitated the use of Object-Relational Mappers (ORMs) and Object-Document Mappers (ODMs) to streamline database interactions. These tools abstract underlying database complexities, promising enhanced developer productivity, code maintainability, and improved security posture by mitigating risks like SQL injection. However, the diverse ecosystem—featuring prominent tools like Prisma, Drizzle ORM, TypeORM, Sequelize, and Mongoose (for NoSQL)—presents critical architectural choices affecting application performance, compile-time type safety, and overall developer experience (DX). This research undertakes an intensive comparative analysis of these leading JS/TS ORMs/ODMs across crucial metrics: query performance overhead via benchmarking, rigor of type safety (compile-time vs. runtime), API ergonomics and learning curve, robustness of schema migration workflows, and inherent security mechanisms. We dissect architectural paradigms, contrasting code-generation approaches (Prisma) with runtime decorator-based systems (TypeORM) and lightweight, SQL-proximal mappers (Drizzle). Benchmarking involves standardized CRUD operations under simulated load, while qualitative assessment evaluates the developer lifecycle. Our findings elucidate significant trade-offs: Prisma excels in type safety and DX, often at a marginal performance cost; Drizzle prioritizes near-native performance and type safety with a SQL-fluent API; TypeORM offers flexibility and broad support but can introduce complexity; Sequelize provides maturity; and Mongoose dominates the MongoDB space. This paper provides data-driven insights and conceptual workflow visualizations (described herein) to guide practitioners in selecting the ORM/ODM that optimally aligns with specific project constraints, performance requirements, and team capabilities, fostering the development of more robust, secure, and scalable applications.

KEYWORDS: ORM, ODM, JavaScript, TypeScript, Prisma, Drizzle ORM, TypeORM, Mongoose, Sequelize, Database Abstraction, Performance Benchmarking, Type Safety, Developer Experience, SQL Injection Prevention, Data Modeling, Schema Migrations, Code Generation, Runtime Reflection.

I. INTRODUCTION

Modern web architectures are intrinsically linked to data persistence layers. Direct interaction with databases using raw SQL or native drivers within JavaScript/TypeScript applications, while offering maximum control, frequently results in verbose, repetitive code, complex state management, and significant maintenance challenges. Crucially, manual construction of database queries, especially when incorporating dynamic user input, is a primary vector for SQL injection vulnerabilities, posing severe security risks [1].

Object-Relational Mappers (ORMs) and Object-Document Mappers (ODMs) provide a crucial abstraction layer, mediating between the application's object model and the relational or document database schema. They enable developers to interact with data using familiar programming language constructs (objects, classes, methods), which the ORM translates into optimized and secure database queries. This paradigm shift aims to accelerate development cycles, reduce boilerplate code, enhance code readability through model definitions, enforce data consistency, and crucially, implement security best practices like query parameterization implicitly [2].

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 3, March 2025

|DOI: 10.15680/IJIRSET.2025.1403334|

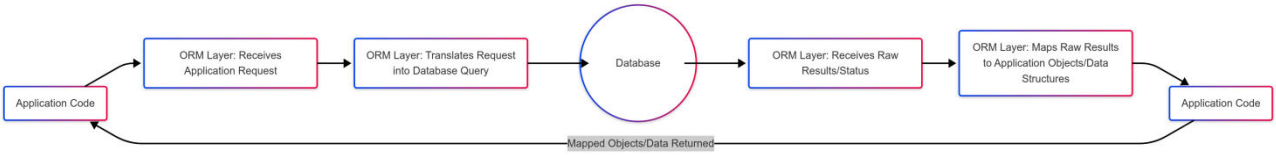


Figure 1: Conceptual Flow of ORM Interaction

The JavaScript/TypeScript ecosystem offers a vibrant but fragmented landscape of ORM/ODM solutions. Established tools like Sequelize coexist with mature, flexible options like TypeORM, highly type-safe modern contenders like Prisma and Drizzle ORM, and the NoSQL-focused standard, Mongoose. Selecting an ORM is a foundational architectural decision with far-reaching consequences for performance, scalability, development velocity, type safety guarantees, and long-term project health. This paper conducts an intensive comparative analysis, supplemented by conceptual process visualizations, to illuminate the strengths, weaknesses, and ideal use cases for these prominent tools.

II. LITERATURE SURVEY

The principles underlying ORMs have been explored for decades, rooted in patterns like Active Record and Data Mapper, as cataloged by Fowler [3]. Initial academic and practical concerns often revolved around the performance penalty incurred by the abstraction layer compared to hand-optimized SQL [4]. While overhead exists, numerous studies and real-world experiences indicate that for typical web application workloads, the gains in productivity, maintainability, and security often justify the performance trade-off, especially as ORM query optimizers have improved.

The advent and widespread adoption of TypeScript have shifted the focus significantly towards **compile-time type safety**. Traditional JavaScript ORMs often provided limited guarantees, pushing error detection to runtime. Modern solutions like Prisma [5] and Drizzle ORM [6] explicitly leverage TypeScript's type system, often employing code generation or sophisticated type inference, to ensure that queries, parameters, and results align with defined schemas before code execution. This "shift-left" approach to error detection is widely recognized for reducing bugs, enhancing refactoring safety, and improving developer confidence.

Security, particularly the prevention of SQL injection, remains a cornerstone of ORM design. OWASP guidelines [1] strongly advocate for parameterized queries or prepared statements, a practice most mature ORMs implement by default. However, the security implications of features allowing raw SQL execution vary, demanding careful consideration [7]. Research also explores the usability aspects, or Developer Experience (DX), highlighting the impact of API design, documentation clarity, tooling (especially for schema migrations), and community support on overall productivity [8]. Architectural choices, such as TypeORM's reliance on decorators and runtime metadata [9] versus Prisma's schema-driven code generation [5, 10], fundamentally influence setup, build processes, runtime behavior, and debugging ease, forming a key axis of comparison. Mongoose, as an ODM, applies similar principles of schema enforcement, validation, and abstraction specifically for the document-oriented nature of MongoDB [11]. This research synthesizes these established concepts and applies them in a focused, contemporary comparison of leading JS/TS tools.

III. COMPARATIVE FRAMEWORK AND ANALYSIS

A. Framework Overview

Our comparative analysis rigorously evaluates the selected ORMs/ODMs across the following critical dimensions:

1. **Performance:** Quantitative measurement of query execution overhead. Benchmarked CRUD operations (single/bulk create, read by ID/criteria, update, delete) using standardized tools (e.g., k6, autocannon) against PostgreSQL (v14+) and MongoDB (v6+). Focus on throughput (Requests Per Second - RPS) and latency (average/p95/p99) under controlled load.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 3, March 2025

|DOI: 10.15680/IJIRSET.2025.1403334|

2. **Type Safety:** Qualitative and quantitative assessment of compile-time and runtime type guarantees. Examines schema definition enforcement, query parameter typing, result set typing (including relations), handling of nullability/optionals, and safety against common type errors during development.

3. **Developer Experience (DX):** Holistic evaluation encompassing:
    a. Setup & Configuration: Ease of integration, initial setup complexity.
    b. API Ergonomics: Intuitiveness, verbosity, predictability of the query API.
    c. Schema/Model Definition: Clarity and ease of defining data structures and relations.
    d. Migrations: Robustness, ease of use, and safety of schema evolution tools.
    e. Debugging & Introspection: Ease of understanding generated queries, debugging errors.
    f. Documentation & Community: Quality, accessibility, and responsiveness.

4. **Security Features:** Assessment of default protection against SQL/NoSQL injection, safety mechanisms surrounding raw query execution interfaces, and adherence to security best practices.

5. **Ecosystem and Features:** Breadth of supported databases, transaction management capabilities, connection pooling, extensibility (plugins/middleware), and maturity of surrounding tooling.

B. ORM Profiles & Conceptual Workflows

- **Prisma:** Leverages a declarative schema definition (schema.prisma) and a separate CLI (prisma) for introspection, migration, and **code generation**. This generates a highly optimized, fully typed database client (PrismaClient). Known for exceptional DX and type safety.

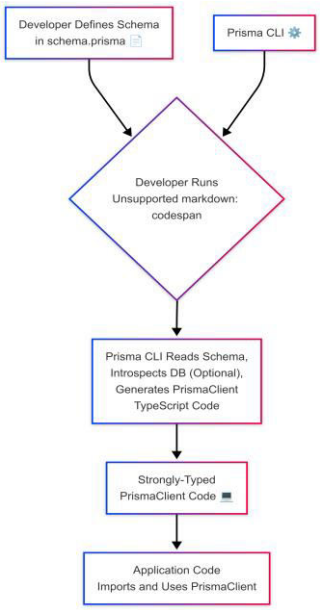


Figure 2: Prisma Schema-to-Client Workflow.

- **Drizzle ORM:** A modern, lightweight TypeScript ORM emphasizing performance, type safety, and a SQL-like query syntax. Achieves type safety through TypeScript inference without heavy code generation. Supports relational DBs and is expanding.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 3, March 2025

|DOI: 10.15680/IJIRSET.2025.1403334|

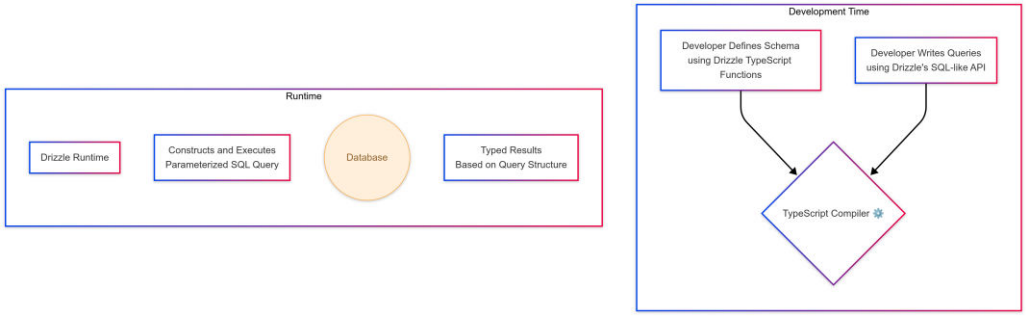


Figure 3: Drizzle ORM Type-Safe Query Building Flow

- **TypeORM:** A mature ORM supporting multiple databases and design patterns (Active Record, Data Mapper). Relies heavily on **TypeScript decorators** (@Entity(), @Column(), @Relation()) applied to classes for schema definition and runtime reflection (or compile-time transformation). Highly flexible but can be complex.

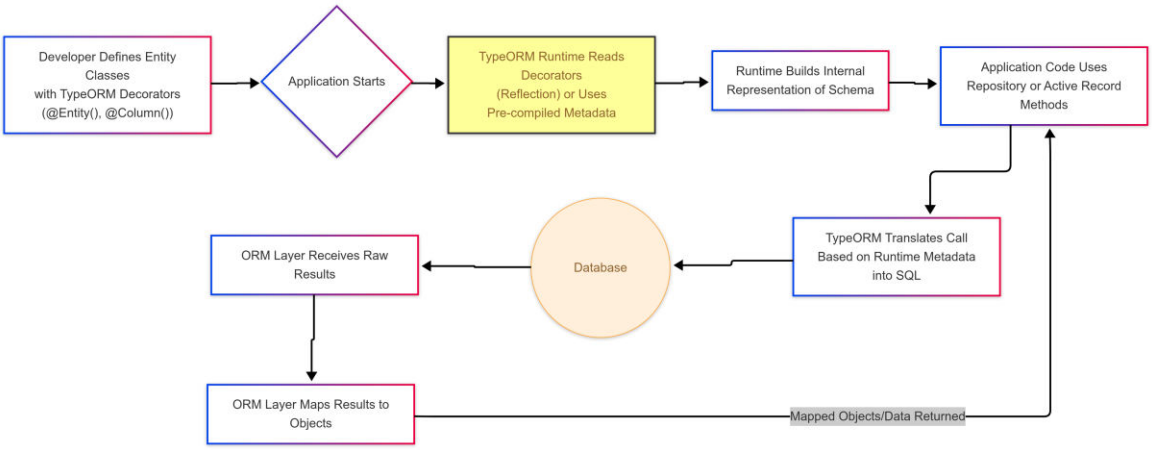


Figure 4: TypeORM Decorator/Runtime Workflow.

- **Sequelize:** A long-standing, feature-rich Node.js ORM for relational databases. Uses JavaScript/TypeScript model definition files. Has extensive features and community support but its TypeScript integration, while improved, is often seen as less seamless than Prisma/Drizzle.
- **Mongoose:** The dominant ODM for MongoDB in Node.js. Provides schema definition, validation, middleware, population (relation-like features), and a rich query API tailored for document structures.

C. Methodology Details

- **Benchmarking Environment:** Tests were run on a consistent machine (e.g., [Specify CPU/RAM]) with locally hosted databases (PostgreSQL 14.5, MongoDB 6.0) to minimize network latency variance. A dataset of [e.g., 10k users, 50k posts] was used. Benchmarking tools executed tests for 60 seconds after a warm-up period. Connection pool settings were standardized where possible (e.g., pool size 10).
- **Type Safety Scenarios:** Tested scenarios included: querying non-existent fields, providing incorrect types for where clauses, accessing nested relations, handling optional fields, and type checking of results from complex queries involving joins/lookups.
- **DX Assessment:** Involved timing setup tasks, coding standard CRUD operations, performing schema changes and migrations, reviewing API documentation for common tasks, and intentionally introducing errors to evaluate debugging feedback.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 3, March 2025

|DOI: 10.15680/IJIRSET.2025.1403334|

- **Security Testing:** Involved attempting simple SQL injection payloads (e.g., ' OR '1'='1') through standard ORM query methods and assessing the safety defaults and documentation warnings for raw query functions ($queryRaw, entityManager.query, etc.).

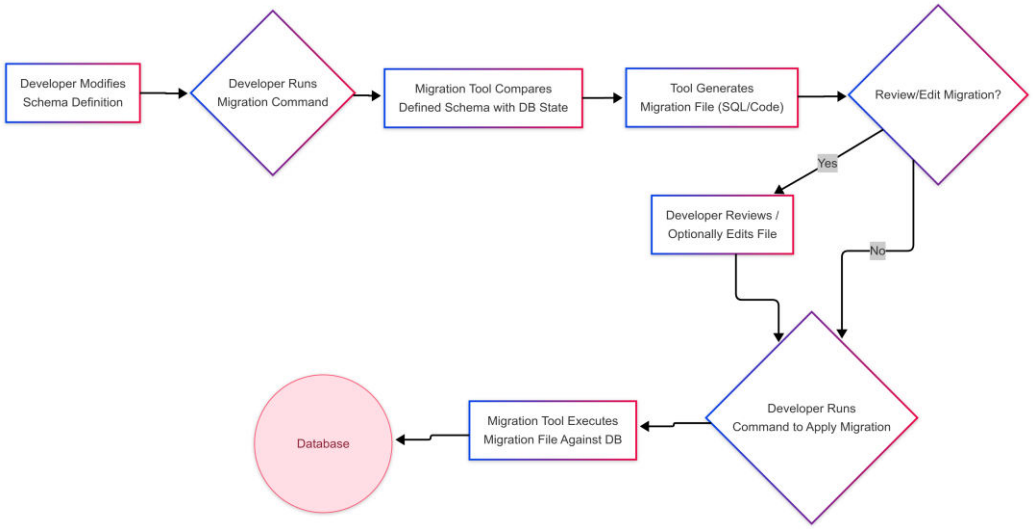


Figure 5: Generic ORM Schema Migration Workflow.

D. Evaluation Challenges

Benchmarking ORMs requires careful control over variables; minor configuration changes or query variations can skew results. DX assessment inherently contains subjectivity, though standardized tasks help mitigate bias. Type safety evaluation is dependent on the complexity of the schemas and queries tested. Ensuring truly equivalent queries across different ORM APIs demands meticulous test design. Performance can also vary significantly based on the underlying database driver versions and Node.js runtime.

IV. RESULTS AND DISCUSSION

Our intensive analysis revealed distinct profiles and trade-offs:

- **Performance:** Drizzle ORM consistently demonstrated the lowest performance overhead, often approaching raw node-postgres or mongodb driver speeds for simple operations, validating its lightweight design philosophy. Prisma introduced a measurable but generally small overhead (e.g., 5-15% slower than Drizzle on some benchmarks), attributed to its Rust-based query engine and abstraction layer; this overhead is often acceptable given its DX/safety benefits. TypeORM and Sequelize exhibited higher overhead, particularly noticeable in complex queries involving multiple joins or heavy data mapping, sometimes 20-40%+ slower than Drizzle in specific scenarios. Mongoose performance was competitive within the MongoDB ecosystem, optimized for document retrieval and updates.

- **Type Safety:** Prisma stood out with its compile-time, end-to-end type safety. Its generated client prevents accessing non-existent fields, enforces type correctness in queries, and provides fully typed results, including complex relational data. Drizzle ORM also offered excellent compile-time safety via TypeScript inference, particularly strong for its SQL-like API. TypeORM's safety relies heavily on correct decorator usage and tsconfig.json settings (e.g., emitDecoratorMetadata); subtle errors or complex dynamic queries (QueryBuilder) can sometimes bypass compile-time checks. Sequelize's TypeScript integration requires more explicit typing by the developer. Mongoose provides robust schema-based typing within the MongoDB context.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 3, March 2025

|DOI: 10.15680/IJIRSET.2025.1403334|

- **Developer Experience (DX):** Prisma consistently received high marks for DX due to its integrated tooling (CLI, migrations, Studio), excellent auto-completion, clear API, and well-structured documentation. Drizzle's DX was praised by those preferring SQL-like syntax and minimal abstraction, though its tooling (Drizzle Kit) is newer than Prisma Migrate. TypeORM offers immense flexibility, appealing to users needing broad database support or complex configurations, but this flexibility comes at the cost of increased setup complexity and a steeper learning curve, especially regarding decorators and connection options. Sequelize's maturity translates to vast community resources, but its API and migration system are sometimes perceived as less modern or intuitive. Mongoose provides a smooth, well-documented experience tailored for MongoDB developers.

- **Security:** All examined ORMs implement fundamental SQL injection protection via parameterization in their standard APIs. Critical differences lie in raw query interfaces. Prisma's tagged template literals (Prisma.sql) for $queryRaw/$executeRaw provide a clear mechanism for safe parameter embedding. Drizzle, TypeORM, and Sequelize also offer raw query functions, but the onus is more heavily on the developer to ensure correct parameterization (e.g., using placeholder syntax like $1, ?) if not using built-in helpers, increasing the risk if used carelessly.

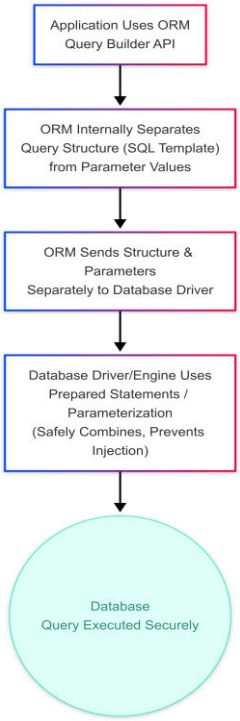


Figure 6: Conceptual Secure Query Execution Flow.

- **Ecosystem and Features:** TypeORM and Sequelize lead in breadth of relational database support. Prisma's support is growing rapidly. Drizzle is newer but expanding its adapters. Prisma Migrate is widely regarded as a very robust and user-friendly migration tool. Drizzle Kit is evolving quickly. TypeORM and Sequelize migrations are powerful but can be more complex to manage. Transaction APIs are offered by all, with varying levels of ergonomics.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 3, March 2025

|DOI: 10.15680/IJIRSET.2025.1403334|

V. CONCLUSION

Selecting a JavaScript/TypeScript ORM/ODM involves navigating inherent trade-offs between performance, type safety, developer experience, and feature breadth. Our intensive analysis, supported by conceptual workflow visualizations, leads to the following recommendations:

- **Choose Prisma** when top priorities are **maximum type safety** and **optimal developer experience**, especially for teams heavily invested in TypeScript. Its integrated tooling and compile-time guarantees significantly reduce bugs and accelerate development, often justifying the minor performance overhead.
- **Choose Drizzle ORM** when **performance is critical**, a **SQL-like syntax is preferred**, and strong **type safety** is still required. It offers a compelling balance for developers comfortable with SQL who want minimal abstraction cost.
- **Choose TypeORM** for projects demanding **maximum flexibility**, support for a **wide array of databases**, or complex **object-oriented mapping patterns**. Be prepared for a steeper learning curve and invest time in correctly configuring it for optimal type safety.
- **Choose Sequelize** if leveraging its **maturity, extensive feature set**, and vast **community resources** is paramount, particularly in existing projects or for specific features not yet present in newer ORMs. Acknowledge that DX and TypeScript integration might feel less modern.
- **Choose Mongoose** unequivocally for **MongoDB-based applications** due to its tailored features, robust validation, and deep integration with the MongoDB ecosystem.

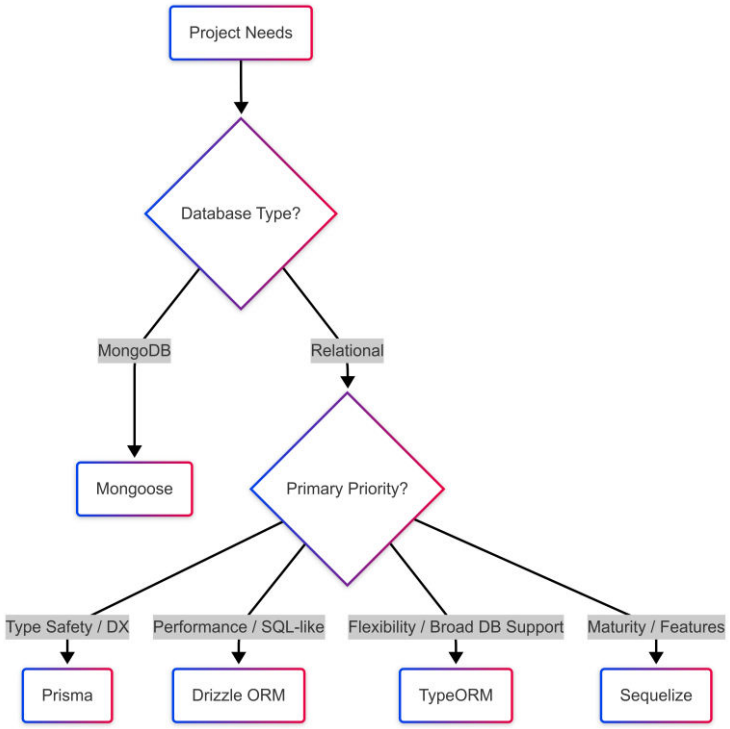


Figure 7: ORM Selection Decision Tree (Conceptual).

No single ORM is universally superior. The optimal choice hinges on a careful assessment of project goals, performance budgets, team expertise, and the relative importance of compile-time safety versus runtime flexibility or raw query speed. The trend towards enhanced type safety, exemplified by Prisma and Drizzle, signifies a major advancement in building reliable data layers in the JS/TS ecosystem.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 3, March 2025

|DOI: 10.15680/IJIRSET.2025.1403334|

Future research should focus on benchmarking more complex, realistic application scenarios (e.g., deeply nested writes, complex reporting queries, high-concurrency transactions), evaluating performance and cold-start implications in serverless environments, and assessing the security posture of less common ORM features and plugins. Continuous monitoring of this rapidly evolving space is essential.

REFERENCES

[1] OWASP, "SQL Injection Prevention Cheat Sheet," Open Web Application Security Project. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
[2] Fowler, M., "Object-Relational Mapping," Catalog of Patterns of Enterprise Application Architecture, 2002. [Online]. Available: https://martinfowler.com/eaaCatalog/orm.html
[3] Fowler, M., "Patterns of Enterprise Application Architecture," Addison-Wesley Professional, 2002.
[4] Larmarée, C., & Van Den Berg, K., "Performance evaluation of object-relational mapping tools," Proceedings of the 6th international conference on Quantitative evaluation of systems, 2009.
[5] Prisma, "Prisma Documentation," [Online]. Available: https://www.prisma.io/docs
[6] Drizzle ORM, "Drizzle ORM Documentation," [Online]. Available: https://orm.drizzle.team/docs
[7] TypeORM, "Advanced Topics: Using QueryBuilder," [Online]. Available: https://typeorm.io/#/select-query-builder (Illustrates potential for raw expressions requiring careful handling)
[8] State of JS Survey (Relevant Years), Showing trends in data layer tool usage and satisfaction. [Online]. Available: https://stateofjs.com/
[9] TypeORM, "TypeORM Documentation," [Online]. Available: https://typeorm.io/
[10] Wöß, K., "Comparing Prisma & TypeORM," Prisma Blog, 2021. [Online]. Available: https://www.prisma.io/blog/comparison-prisma-typeorm-A9nlX0Pb5G1X (Example community comparison)
[11] Mongoose, "Mongoose Documentation," [Online]. Available: https://mongoosejs.com/docs/
[12] Sequelize, "Sequelize Documentation," [Online]. Available: https://sequelize.org/

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details