Understand Prisma

Prisma Introduction: What, Why & How

Learn about Prisma's use cases, main benefits and how it fits into your stack.

What is Prisma?

Prisma replaces traditional ORMs and simplifies database workflows:

- *Access*: **Type-safe database access with the auto-generated Prisma client** (in JavaScript, TypeScript, Go)
- *Migrate*: **Declarative data modelling and migrations** (optional)

- *Manage*: **Visual data management with Prisma Admin**

It is used to build **GraphQL, REST, gRPC APIs** and a lot more. Prisma currently supports MySQL, PostgreSQL, MongoDB.

A note on Prisma Cloud

Use cases

Prisma is useful in any context where you're working with databases.

Building GraphQL servers

Prisma is the perfect tool for building GraphQL servers. The Prisma client is compatible with the Apollo ecosystem, has default support for GraphQL subscriptions and Relay-style pagination, provides end-to-end type safety and comes with a built-in dataloader to solve the N+1 problem.

Building REST & gRPC APIs

Prisma is a great fit for building REST& gRPC APIs where it can be used in place of traditional ORMs. It provides

many benefits such as type-safety, a modern API and flexible ways for reading and writing relational data.

CLIs, Scripts, Serverless Functions & a lot more

Prisma has an extremely flexible API which makes it a great fit for a variety of use cases. Whenever you need to talk to one or more databases, Prisma will be of great help by simplifying database workflows.

Why use Prisma?

Simple database workflows

Prisma's overall goal is to remove complexity from common database workflows and simplify data access in your applications:

- **Type-safe database access** thanks to the custom and auto-generated Prisma client.
- Simple and powerful API for working with **relational data and transactions**.
- Visual **data management** with Prisma Admin.
- Prisma unifies access to multiple databases at once (*coming soon*) and therefore drastically **reduces complexity in cross-database workflows**.

- **Realtime streaming & event system for your database** ensuring you're getting updates for all important events happening in your database.
- **Automatic database migrations (optional)** based on a declarative datamodel expressed using GraphQL's schema definition language (SDL).
- Other database workflows such as **data import, export & more**.

A realtime layer for your database

Some databases, such as RethinkDB or DynamoDB provide a realtime API out-of-the box. Such an API lets clients *subscribe* to any changes happening in the database. The vast majority of conventional databases however does not offer such a realtime API, and implementing it manually gets very complex.

Prisma offers a realtime API for every supported database, letting you subscribe to any database event, such as *creating*, *updating* or *deleting* data.

End-to-end type safety

Programming in a type safe way is the default for modern application development. Here are some of the core benefits type safety provides:

- **Confidence**: Developers can have strong confidence in their code thanks to static analysis and compile-time error checks.

- **Developer experience**: Developer experience increases drastically when having clearly defined data types. Type definitions are the foundation for IDE features like *intelligent auto-completion* or *jump-to-definition*.
- **Code generation**: It's easy to leverage code generation in your development workflows to avoid writing boilerplate.
- **Cross-system contracts:** Type definitions can be shared across systems (e.g. between client and server) and serve as contracts that define the respective interfaces/APIs.

End-to-end type safety refers to having type safety across the entire stack, from client to database. An end-to-end type safe architecture might look as follows:

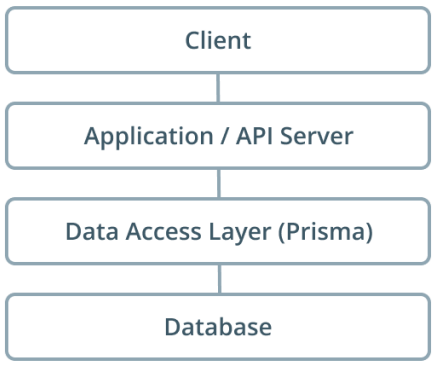
- **Database**: Prisma provides the strongly typed database layer, the *datamodel* defines the data types that are being stored in the database
- **Application server**: The application server defines its own schema (e.g. using GraphQL or OpenAPI/Swagger) where it either reuses or transforms the data types from the database. The application server needs to be written in a type safe language (e.g. TypeScript, Scala, Go).
- **Client**: A client being aware of the application server's schema can validate API requests and

potential responses at build-time.

Clean and layered architecture

When developing application servers, most complexity lies in implementing a safe and well-organized database access with respect to *synchronization*, *query optimization/performance* and *security*. This becomes even more complicated when *multiple* databases are involved.

One common solution to this problem is the introduction of a dedicated *data access layer* (DAL) that abstracts away the complexities of database access. The DAL's API is consumed by the application server, allowing API developers to simply think about *what* data they need instead of worrying about *how* to securely and performantly retrieve it from the database.



Using a DAL ensures a **clear separation of concerns** and therefore **improves maintainability and reusability** of your code. Having some sort of database abstraction (be

it a simple ORM library or a standalone infrastructure component) is **best practice** for smaller sized applications as well as for applications running at scale. It ensures the application server can talk to your database(s) in a **secure and performant** way.

Prisma is an auto-generated DAL following the same principles as industry-leading DALs (such as Twitter's Strato or Facebook's TAO) while still being accessible for smaller applications.

Prisma lets you start your project with a clean architecture from the beginning and saves you from writing the boilerplate that is otherwise required to glue together database and application server.

How does Prisma fit into your stack?

Prisma is a standalone infrastructure component that sits on top of your database. You're then using a Prisma client (which is available in various languages) in your application server to connect to Prisma.

This enables you to talk to your database(s) through a simple and modern API ensuring highly performant and secure database access.

Edit this page on Github

Last updated 4 days ago

Was this page helpful?

Create an Issue

Join the Prisma newsletter

your@email.com

JOIN

Products

Prisma Client

Prisma Migrate

Prisma Admin

Prisma Cloud

Prisma Enterprise

Resources

Docs

Get Started

Tutorials

Examples

Community

Meet the Community

Slack

Forum

Conference

Company

About

Jobs

Blog

Terms & Privacy