Getting started with Prisma ORM

Binayak G Shankar · Follow

6 min read · Oct 10, 2023

Listen Share

Source: Google images

Hello! In this tech blog, we will learn about Prisma, a new-age ORM. We are going to cover various topics like:

1. What is an ORM?
2. Why Prisma is better?
3. Initializing our project
4. Creating models & relationships

5. Prisma migrate
6. CRUD using Prisma
7. Prisma Studio (GUI)

Additionally, I will provide the code walkthrough here and also add the link to GitHub.

Let's get started!

1. What is an ORM?
- An ORM stands for Object Relationship Mapping, it is basically a technique that helps you bridge OOPs program to your relational database.
With this, you can speed up your development eventually lowering the cost of development. You can easily interact with the database, and the logic will be handled by the ORM. You can imagine it as a layer that connects your database to your object-oriented program. ***Prisma is an ORM.***

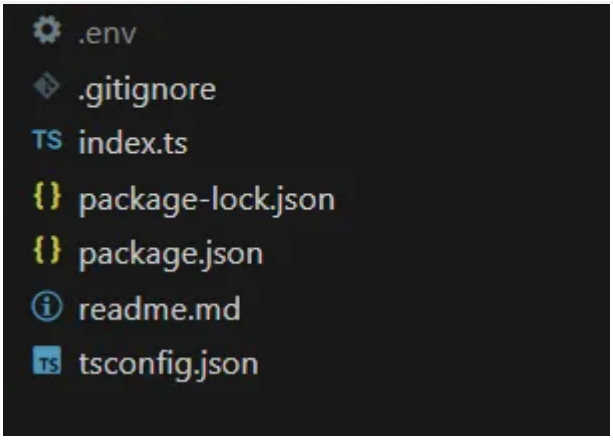
2. Why Prisma is better?
- Prisma offers 3 basic tools, Prisma Client, Prisma Migrate, and Prisma Studio. No worries, we are going to discuss these further.
As per the documentation, *Prisma ORM is an Auto-generated and type-safe query builder for Node.js & TypeScript.*
Prisma helps you to create models that are type-safe and also assists in creating queries by providing you with an interface.

3. Initializing your project
- This is an amazing part of this blog, we are finally going to initialize our project and start using Prisma. I am using VS Code for this project, you can use any IDE of your choice.

```
// Create a folder and make it your current working directory
// Now execute these queries in the terminal

npm install typescript ts-node @types/node -D

npx tsc --init

npm i prisma -D
```

```
npx prisma init
```

After executing these, you can see a folder `Prisma` in your current working directory or folder. Like in the below picture. (ignore *index.ts* as of now)



So, we have installed Prisma successfully and we can see `schema.prisma` file inside the folder. Additionally, an environment file i.e. `.env` is also created along with it in the *root* directory. These files are very important to us. Here, we will configure our further operations.

The file contains the database connection part. We will provide the connection string in order to connect to the database.

> *Here we are using **Postgres** as our database server, it is by default database configured in Prisma client.*

In your `.env` file, `DATABASE_URL` should hold the value of your connection string. That is imported directly to your `prisma.schema` file.

We have successfully configured our database server to Prisma now. This marks the end of your project initialization.
** You may find the connection string in your PgAdmin4 here*

4. Creating models and relationships
- This is an important part as we are going to define our models and relationships among them.

Append these lines in your `prisma.schema` file.

```
model User {
  id Int @id @default(autoincrement())
  email String @unique
  name String?
  articles Article[]
}

model Article {
  id Int @id @default(autoincrement())
  title String
  body String?
  author User @relation(fields: [authorId], references: [id])
  authorId Int
}
```

Let's understand what it means, we have created two models, namely `User` and `Article` , both have a primary key as `id` , also we have set it as `@default(autoincrement())` it means it will set an auto-incremented value as default for each inserted row, `@unique` signifies that the values passed to this field (*here, email*) should be unique. `Int,String` are the scalar data types supported by Prisma, whereas `Article` which is another model acting as a user-defined data type.

What is `?` and `[]` ? Okay, `?` signifies that the field is optional and can be left blank, whereas `[]` means an Array of a particular data type.

So, `articles Article[]` means, a `User` can have multiple `articles` of type `Article`. It is actually establishing a relationship between the `User` and `Article` , other important lines in our `Article` model are:

```
author User @relation(fields: [authorId], references: [id])
authorId Int // Basically the id of an existing user, who is writing this artic
```

`@relation(fields: [authorId], references: [id])` means `author` is of `User` type and contains the details of the user whose `id` is the same as the `authorId` provided. Now, we have successfully created the relationship among the models.

It is advised to visualize your relationships among the models before creating them. It makes your task easy and relationships can be established hassle-free during coding.

5. Prisma Migrate.
- The purpose of this service provided by Prisma is to create tables along with the fields and relationships in less time by running a single command.

Now, run this command in your terminal, `npx prisma migrate` Congratulations! You have created the schema now. Wasn't it easy?
Let's move on to CRUD operation now.

6. CRUD Operation.
- As the name suggests, we are going to perform some basic operations like Create, Read, Update, and Delete.
For this, we have to create a file `index.ts` in the *root* directory, here we will write our code for the operations abovementioned. Follow the steps below:

- Initialize Prisma Client: Add the below lines to your `index.ts` file.

```
import { PrismaClient } from "@prisma/client"; // Importing Prisma Client

const prisma = new PrismaClient(); // Creating Prisma client object

async function main() {
  // We will write code here for CRUD
}

main()
.then(async () => {
    await prisma.$disconnect(); // disconnect after task is done
}).catch(async (e)=>{
    console.error(e); // log exception
    await prisma.$disconnect(); // disconnect if exception occur
    process.exit(1); // end process with some failure
})
```

Having added these lines, we are going to modify `main` function to perform CRUD.

- Create User: We are going to create `User` now.

```
const user = await prisma.user.create({
    data:{
          name:"Peter",
          email:"peter.p@example.com"
      },
   })
```

- Create Article: Let's see again, how can we create `Article`

```
const article = await prisma.article.create({
        data: {
         title:"Peter's Secret",
         body:"It isn't easy to know my secrets.",
         author:{
            connect:{
            id:1  // here we are providing the id of a user, to associate with
         }
        },
        },
   });
```

`connect` basically means to establish a relationship by providing the `id` of a `user`. You have seen this in the schema, right?

- Read Articles, Users: Having done with creating a `user` and `schema` , lets see how can we fetch them!

```
const users = await prisma.user.findMany(); // it will return an array
console.log(users);
```

```
const articles = await prisma.article.findMany();
console.log(articles);
```

The `findMany` function helps you to fetch all the data of a table, you may also provide conditions to filter. Read more here.

> *Note: Make sure to comment other queries before running the current query in the `main` function.*

- Update a user: Updating a user will introduce us to `where` clause from RDBMS, interesting, isn't it?

```
const user = await prisma.user.update({
     where: {
         id:1,
     },
     data:{
         name:"John"
     }
 })
 console.log(user); // Updated user
```

here, in `where` condition, we have provided the `id` of the user, for which we are going to change the name. In `data` object, we have provided the fields that we are going to change. We can add other fields from the schema as well, but here I am going to change the `name` .

- Remove/Delete an article: Before proceeding here, you are advised to create 2–3 more articles, so that you are left with some rows after deleting one article.

```
const article = await prisma.article.delete({
    where:{
        id: 1
    }
});

console.log(article); // deleted article
```

The delete method helps us to delete from a particular table. In return, we get the deleted object.

Since we have reached to second last step, I am adding the ***Repository of the project*** , take a clone, and get started, if you have not yet.

7. Prisma Studio: This is a GUI tool available locally to visualize your data. You can also add, remove, or update data through it. It opens in a local browser.

`npx prisma studio` upon running this command in your terminal, you can open `localhost:5555` on your machine. It is very intuitive and easy to use.

This marks the end of this small demonstration of Prisma. You can now explore more on this interesting topic.

Do not forget to find the repository link of the project above.

Peace out!

Prisma Orm GraphQL Database Software Development

Follow

Written by Binayak G Shankar

19 Followers

DAIICT Alumnus. https://binayakgourishankar.github.io/

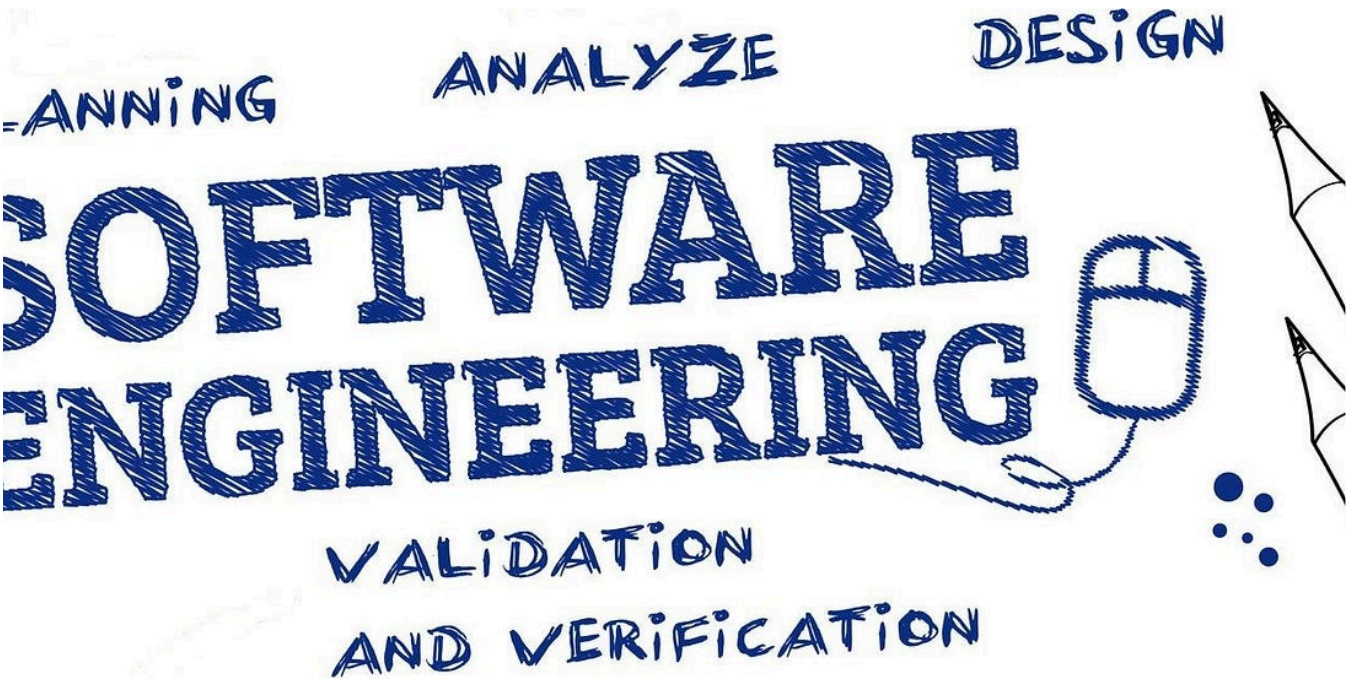
More from Binayak G Shankar

Binayak G Shankar

Get Rejected Like a Boss

Rejection is priceless. There is no course in any university or school that teaches us how to face rejection! How to get over it! How to...

Sep 8, 2018 159 1



Binayak G Shankar

Build Good Build Better.

Hey pals, you all might have heard a famous saying— Anything planned is half done. The era we live in is very unorganized. We are...

Oct 20, 2018 132 2

Binayak G Shankar

Demystifying YouTube Ads: A Technical Deep Dive into Placement and Execution

In the ever-evolving landscape of online content consumption, YouTube stands out as a powerhouse for video content. Central to its revenue...

Jan 17 2

	default	balance	housing	loan	contact	day	month	duration	
	no	1787	no	no	cellular	19	oct	79	1
	no	1787	no	no	cellular	19	oct	79	1
	no	1787	no	no	cellular	19	oct	79	1
	no	1787	no	no	cellular	19	oct	79	1
	no	4789	yes	yes	cellular	11	may	220	1
	no	1350	yes	no	cellular	16	apr	185	1
	no	1476	yes	yes	unknown	3	jun	199	4
	no	0	yes	no	unknown	5	may	226	1
	no	747	no	no	cellular	23	feb	141	2
	no	307	yes	no	cellular	14	may	341	1
	no	147	yes	no	cellular	6	may	151	2

Binayak G Shankar

Stored Procedures in MySQL

As a data analyst, You have to retrieve data from various sources, process it and analyze, eventually helping in decision-making. While...

Nov 18, 2022 55

See all from Binayak G Shankar

Recommended from Medium

Craftsman

Setting Up Prisma with MongoDB: A Step-by-Step Guide

1. Create project setu

Feb 16 3

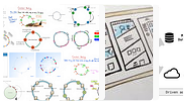
Stephen Klop

Introduction to Prisma: Testing Prisma Applications—Part 10/15

In this tenth entry of our “Introduction to Prisma” series, we explore the critical role of testing in Prisma applications. Effective...

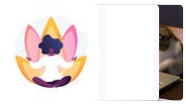
Jan 31 31

Lists



General Coding Knowledge

20 stories · 1430 saves

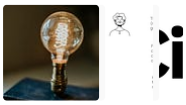


Stories to Help You Grow as a Software Developer

19 stories · 1232 saves

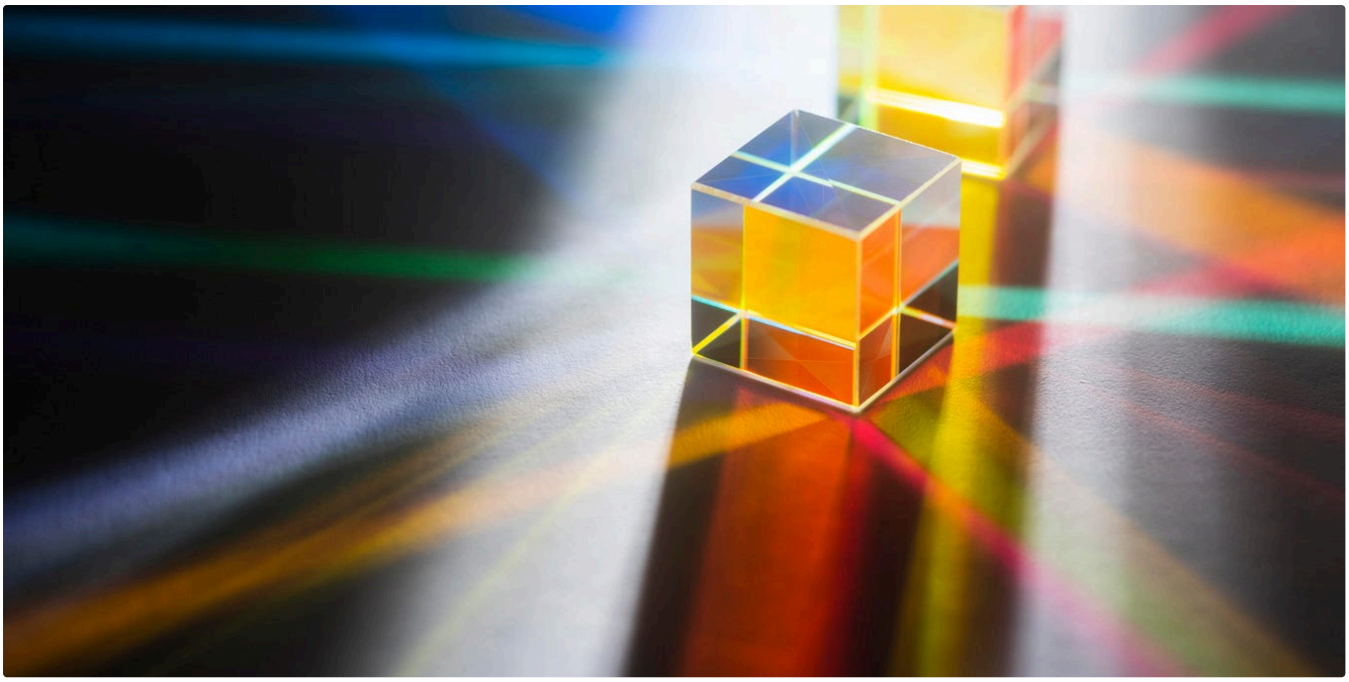
Coding & Development

11 stories · 714 saves



Good Product Thinking

11 stories · 649 saves



Marcos Lombog in JavaScript in Plain English

Multiple Database Connections with NextJS and Prisma ORM

A Hands-On Guide for Database Migration with NextJS and Prisma ORM

Mar 13

Louis Trinh

Advanced Node.js API Logging with Winston and Morgan

Installation:

Apr 6 26

Stephen Klop

Introduction to Prisma: Migrations and Database Management with Prisma—Part 8/15

Welcome to the eighth installment of our “Introduction to Prisma” series. In this part, we focus on Prisma Migrate, a critical feature...

Jan 31 2

Ashish Ansurkar in JavaScript in Plain English

Drizzle ORM—Nestjs: database migration set-up

If you have not set up a basic Nestjs application with Drizzle ORM, then head over here:

May 30 4

See more recommendations