

# Prisma on Next.js

Asst.Prof.Drusawin Vongpramate

Department of Information Technology

Faculty of Science, BRU

**Hint**



Main Topic



Sub Topic

# What's Prisma?

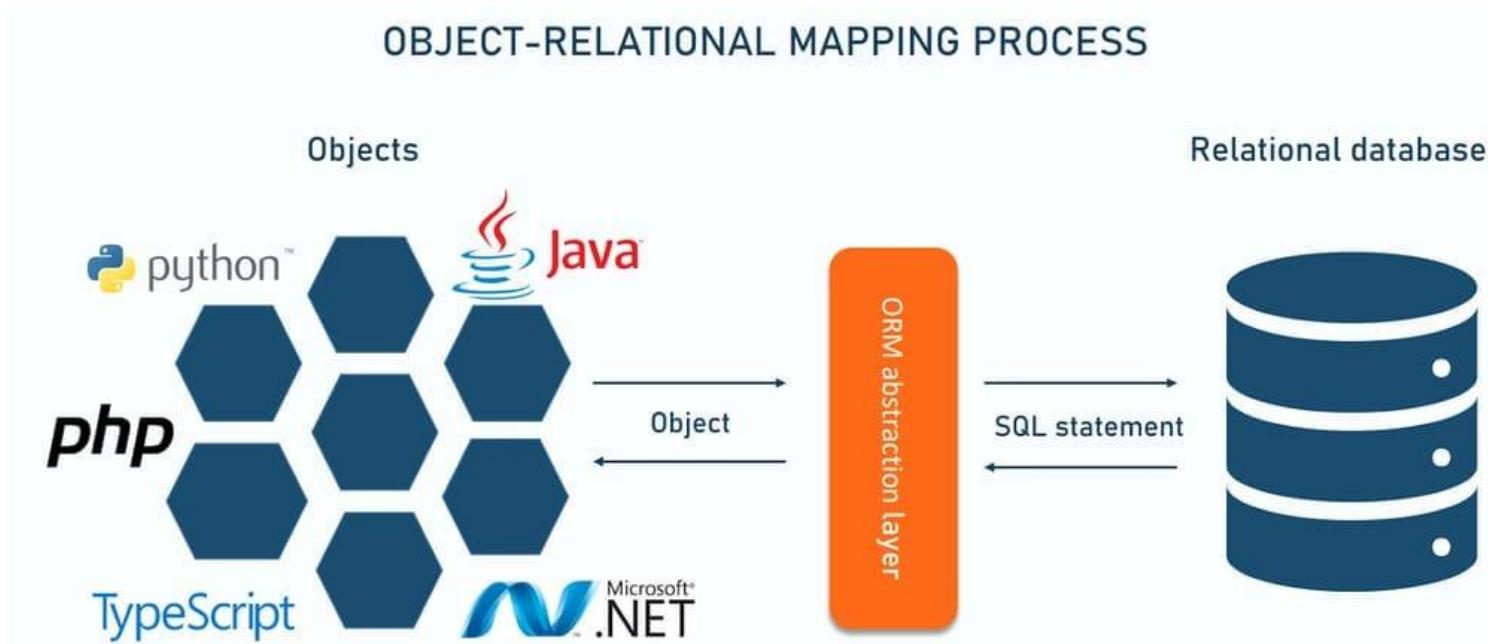
Next-generation Node.js and TypeScript **ORM**.  
Prisma ORM unlocks a new level of developer experience when working with databases thanks to its intuitive data model, automated migrations, type-safety & auto-completion.



# Prisma

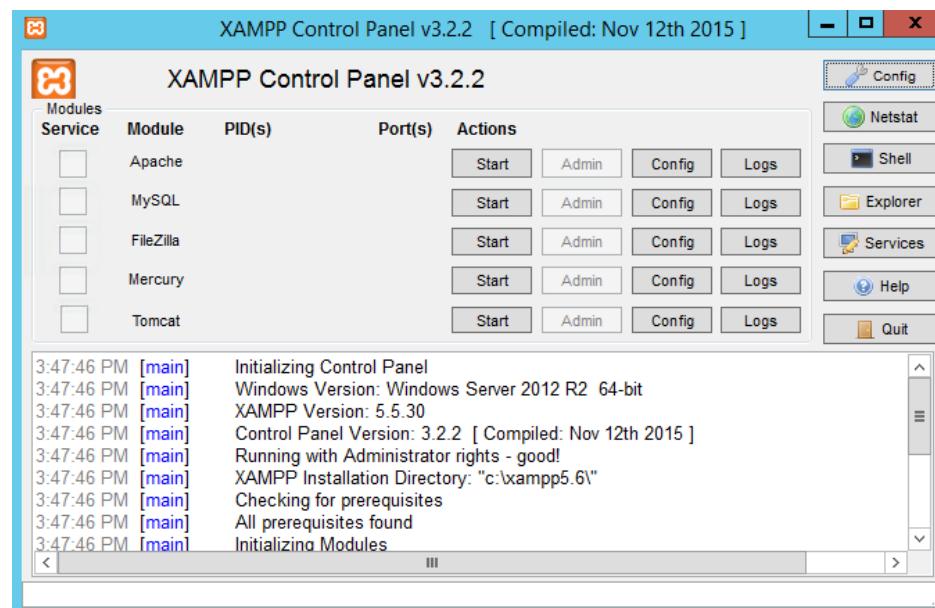
# ORM

An ORM, or **Object Relational Mapper**, is a piece of software designed to translate between the data representations used by databases and those used in object-oriented programming.



# Extra Tools for LAB

XAMPP is a completely free, easy to install Apache distribution containing MariaDB, PHP, and Perl. The XAMPP open source package has been set up to be incredibly easy to install and to use.



# Extra Tools for LAB

## Prisma extension for VS Code

The screenshot shows the Visual Studio Code Marketplace interface. The search bar at the top contains the text "prisma". In the main pane, the top result is the official "Prisma" extension by Prisma, which is highlighted with a blue background. This extension is version 5.15.0, has over 1,700,000 installs, and a perfect 5-star rating from 23 reviews. It is described as adding syntax highlighting, formatting, auto-completion, and jump-to-definition features. Below the main extension card, there are other related extensions listed, such as "Prisma - Insider", "Prisma NextJS", and "Prisma Import".

EXTENSIONS: MARKETPLACE

File Edit Selection View Go Run Terminal Help

4133305-1-67

prisma

**Prisma** v5.15.0

Prisma [prisma.io](#) | 1,735,288 | ★★★★★(23)

Adds syntax highlighting, formatting, auto-completion, jump-to-definition...

[Disable](#) [Uninstall](#)

DETAILS FEATURES CHANGELOG

Prisma VS Code Extension

Adds syntax highlighting, linting, code completion, formatting, jump-to-definition...

Features

Syntax highlighting of Prisma schema files

# Get Started

Create next app

```
>npx create-next-app@latest
```

Change work directory

```
>cd <your app>
```

Install prisma

```
>npm install prisma --save-dev
```

Setup prisma

```
>npx prisma init
```

# Config .env

## MySQL connection URL

CONNECTOR    USER:PASSWORD@HOST:PORT    DATABASE    KEY1=VALUE&KEY2=VALUE

mysql :// janedoe:mypassword@localhost:5432 / mydb ? connection\_limit=5

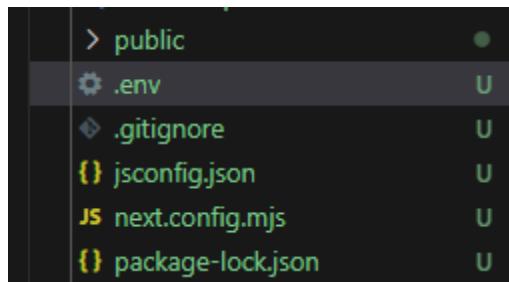
Protocol

Base URL

Path

Arguments

example



6

```
7 # DATABASE_URL="postgresql://johndoe:randompassword@localhost:5432/blog"
8 DATABASE_URL="mysql://dba:1234@localhost:3306/blog"
```

9

10

# Creating the database schema

/app/prisma/schema.prisma

```
11 datasource db {  
12   provider = "mysql"  
13   url      = env("DATABASE_URL")  
14 }  
15  
16 model User {  
17   id    Int    @id @default(autoincrement())  
18   email String @unique  
19   name  String?  
20   posts Post[]  
21 }  
22  
23 model Post {  
24   id      Int    @id @default(autoincrement())  
25   title   String  
26   content String?  
27   published Boolean @default(false)  
28   author  User    @relation(fields: [authorId], references: [id])  
29   authorId Int  
30 }
```

Create relation

Auto generate

Default attribute by table name

# Map data model to the database schema

```
> npx prisma migrate dev --name init
```

```
Applying migration `20240623114300_init`

The following migration(s) have been created and applied from new schema changes:

migrations/
└── 20240623114300_init/
    └── migration.sql

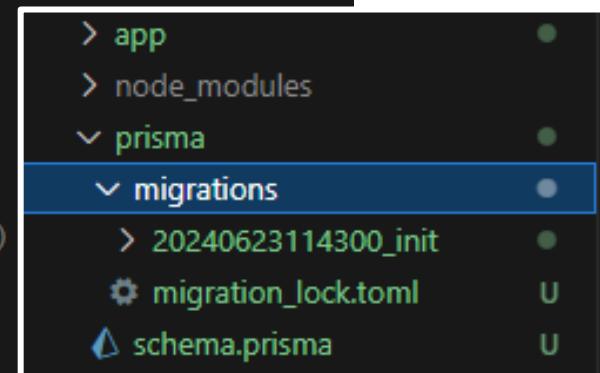
Your database is now in sync with your schema.

Running generate... (Use --skip-generate to skip the generators)
added 1 package, and audited 29 packages in 4s

3 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

✓ Generated Prisma Client (v5.15.1) to .\node_modules\@prisma\client in 72ms
```



```
> npx prisma generate //if change model
```

# View and edit data

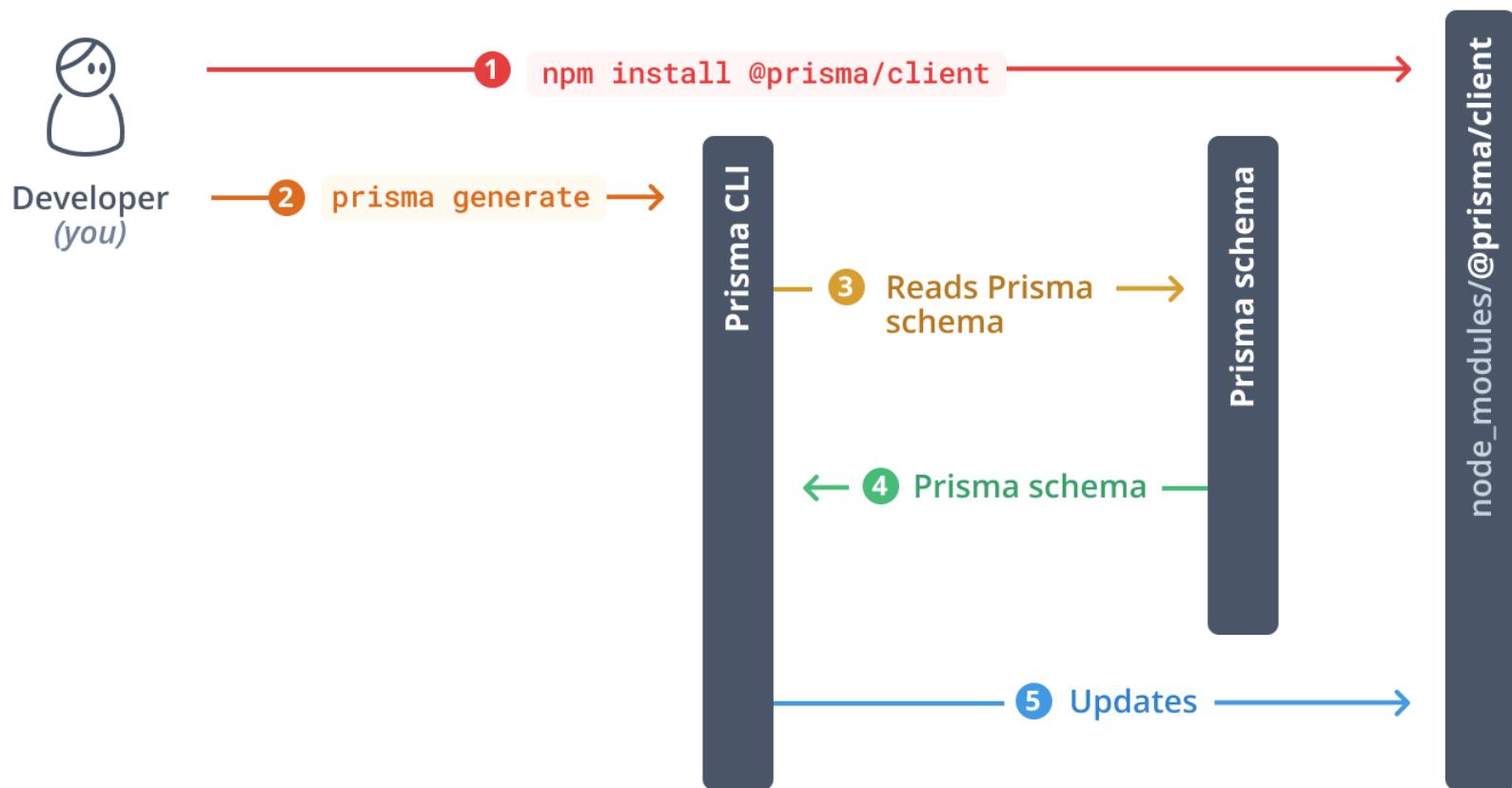
> npx prisma studio

The screenshot shows two views of the Prisma Studio interface. The top view is a search interface with a 'Search' bar and a list of models: post (0) and user (2). The bottom view is a detailed view of the 'user' model. It includes a toolbar with 'Filters' (set to 'None'), 'Fields' (set to 'All'), and 'Showing 2 of 2'. There is also an 'Add record' button. The main table displays two records:

	id #	email A	name A?	post []
<input type="checkbox"/>	1	admin@bru	bru_admin	0 post
<input type="checkbox"/>	2	test@tt.co	test_user	0 post

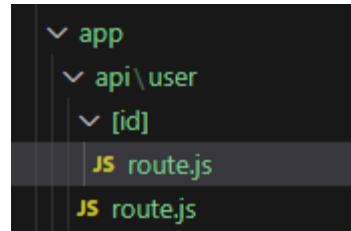
# Install Prisma Client

> npm install @prisma/client



# Querying the database

Create API



edit `route.js` at `/app/api/user`

```
1 import { PrismaClient } from "@prisma/client";
2
3 const prisma = new PrismaClient()
4
5 export async function GET(){
6     const allUsers = await prisma.user.findMany()
7
8     return Response.json(allUsers)
9 }
```

# Prisma Client API reference

Model queries	Model query options	Filter conditions and operators	Relation filters
<code>findOne()</code>	<code>select</code>	<code>equals</code>	<code>some</code>
<code>findOneOrThrow()</code>	<code>include</code>	<code>not</code>	<code>every</code>
<code>findFirst()</code>	<code>omit</code> (Preview)	<code>in</code>	<code>none</code>
<code>findFirstOrThrow()</code>	<code>relationLoadStrategy</code> (Preview)	<code>notIn</code>	<code>is</code>
<code>findMany()</code>	<code>where</code>	<code>lt</code>	<code>isNot</code>
<code>create()</code>	<code>orderBy</code>	<code>lte</code>	
<code>update()</code>	<code>distinct</code>	<code>gt</code>	
<code>upsert()</code>		<code>gte</code>	
<code>delete()</code>		<code>contains</code>	
<code>createMany()</code>		<code>search</code>	
<code>createManyAndReturn()</code>		<code>mode</code>	
<code>updateMany()</code>		<code>startsWith</code>	
<code>deleteMany()</code>		<code>endsWith</code>	
<code>count()</code>		<code>AND</code>	
<code>aggregate()</code>		<code>OR</code>	
<code>groupBy()</code>		<code>NOT</code>	
<code>findRaw()</code>			
<code>aggregateRaw()</code>			
			Scalar list methods
			<code>set</code>
			<code>push</code>
			<code>unset</code>
			Scalar list filters
			Remarks
			<code>has</code>
			<code>hasEvery</code>
			<code>hasSome</code>
			<code>isEmpty</code>
			<code>isSet</code>
			<code>equals</code>

# Querying the database

```
const createMany = await prisma.user.createMany({  
  data: [  
    { name: 'Bob', email: 'bob@prisma.io' },  
    { name: 'Bobo', email: 'bob@prisma.io' }, // Duplicate unique key!  
    { name: 'Yewande', email: 'yewande@prisma.io' },  
    { name: 'Angelique', email: 'angelique@prisma.io' },  
  ],  
  skipDuplicates: true, // Skip 'Bobo'  
})
```

```
const findUser = await prisma.user.findFirst({  
  where: {  
    posts: {  
      some: {  
        likes: {  
          gt: 100,  
        },  
      },  
    },  
    orderBy: {  
      id: 'desc',  
    },  
  }  
})
```

```
const user = await prisma.user.findUnique({  
  where: {  
    email: 'emma@prisma.io',  
  },  
  select: {  
    email: true,  
    posts: {  
      select: {  
        likes: true,  
      },  
    },  
  },  
})
```

```
const deleteUsers = await prisma.user.deleteMany({  
  where: {  
    email: {  
      contains: 'prisma.io',  
    },  
  },  
})
```

```
const updateUser = await prisma.user.update({  
  where: {  
    email: 'viola@prisma.io',  
  },  
  data: {  
    name: 'Viola the Magnificent',  
  },  
})
```

```
const users = await prisma.user.findMany({  
  where: {  
    email: {  
      endsWith: 'prisma.io',  
    },  
  },  
})
```

```
const users = await prisma.user.findMany({  
  where: {  
    OR: [  
      {  
        name: {  
          startsWith: 'E',  
        },  
      },  
      {  
        AND: {  
          profileViews: {  
            gt: 0,  
          },  
          role: {  
            equals: 'ADMIN',  
          },  
        },  
      },  
    ],  
  },  
})
```

# Querying the database

edit `route.js` at `/app/api/user/[id]`

```
1 import { PrismaClient } from "@prisma/client";
2
3 const prisma = new PrismaClient();
4
5 export async function GET(req, { params }) {
6   const user_id = Number(params.id);
7
8   const allUsers = await prisma.user.findUnique({
9     where: { id: user_id },
10    });
11
12   return Response.json(allUsers);
13 }
```

# Querying the database

## Data on database

A screenshot of a database management interface. At the top, there are tabs for 'Properties', 'Data' (which is selected), and 'ER Diagram'. Below the tabs, a search bar contains the text 'user'. A message in the search bar says 'Enter a SQL expression to filter results (use Ctrl+Space)'. The main area shows a grid of data with three columns: 'id', 'email', and 'name'. There are 6K rows of data. The first two rows are visible: row 1 has id 1, email admin@bru, and name bru\_admin; row 2 has id 2, email test@tt.co, and name test\_user. The 'test\_user' entry is highlighted with a blue selection bar.

## JSON on API

A screenshot of a REST API testing tool. The URL is 'HTTP 4133305 / user'. Below it, a 'GET' method is selected with the endpoint '({{host3000}})/api/user'. The 'Headers (6)' tab is selected. Under 'Query Params', there are two empty rows for key and value. The 'Body' tab is selected, showing a JSON response. The response is a list of two user objects:

```
1 [  
2 {  
3   "id": 1,  
4   "email": "admin@bru",  
5   "name": "bru_admin"  
6 },  
7 {  
8   "id": 2,  
9   "email": "test@tt.co",  
10  "name": "test_user"  
11 }]  
12 ]
```

# Querying the database

Add `DELETE` method

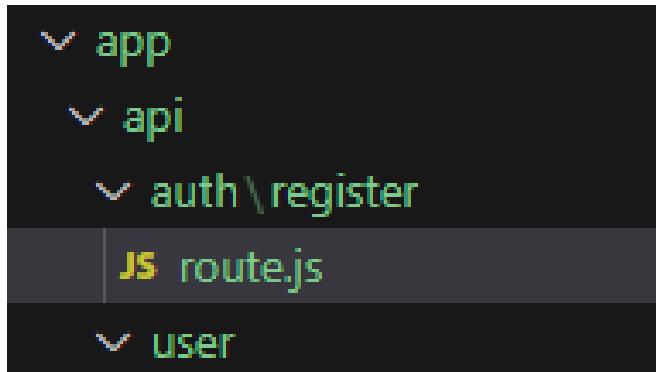
```
15 export async function DELETE(req, { params }) {
16   try {
17     const user_id = Number(params.id);
18     const result = await prisma.user.delete({
19       where: { id: user_id },
20     });
21
22     return Response.json({ des: "Deleted user:", result }, { status: 200 });
23   } catch (err) {
24     return Response.json({ err }, { status: 500 });
25   }
26 }
27
```

Blank -> return 200

```
DELETE /api/user/1 200 in 57ms
✓ Compiled in 47ms (28 modules)
```

# Querying the database

Create `register` in  
api/auth and add  
POST method



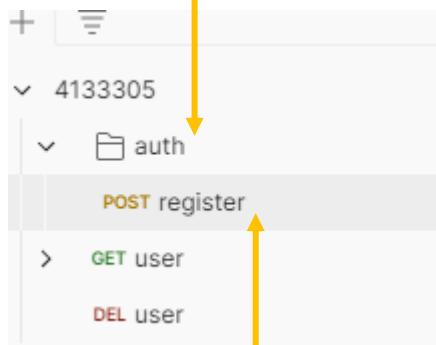
\*\*\* Install `bcrypt`  
package

```
6  export async function POST(req) {
7    const { email, name } = await req.json();
8    const newUser = await prisma.user.create({
9      data: {
10        email,
11        name: await hash(name, 10),
12      },
13    });
14
15    try {
16      return Response.json(
17        {
18          newUser,
19        },
20        { status: 200 }
21      );
22    } catch (err) {
23      Response.json({ err }, { status: 500 });
24    }
25 }
```

A yellow arrow originates from the text "Install bcrypt package" at the bottom left and points diagonally upwards towards the `hash` function call in the code snippet.

# Querying the database

Create auth folder



Create register by  
POST request

HTTP 4133305 / auth / register

POST {{host3000}}/api/auth/register

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {  
2   "email": "test1@t.co",  
3   "name": "test1"  
4 }
```

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "newUser": {  
3     "id": 6,  
4     "email": "test1@t.co",  
5     "name": "$2b$10$gQweLXmYuuj3OoFqbB.kmu0uXrQKmhXcKFMw5CoIxTq5YBu2rWdeS"  
6   }  
7 }
```

# Create update method



Let's Try

## Guideline

1. Where is route.js?
2. What is function name?
3. What is ORM method?
4. What is response?

**Q & A**

```
30 export async function PUT(req, { params }) {
31   try {
32     const userId = Number(params.id);
33     const { email, name } = await req.json();
34     const updateUser = await prisma.user.update({
35       where: {
36         id: userId,
37       },
38       data: {
39         email,
40         name,
41       },
42     });
43     return Response.json(updateUser, { status: 200 });
44   } catch (err) {
45     return Response.json(err, { status: 500 });
46   }
47 }
```