

# Prisma on Next.js

Asst.Prof.Drusawin Vongpramate

Department of Information Technology

Faculty of Science, BRU

**Hint**



Main Topic



Sub Topic

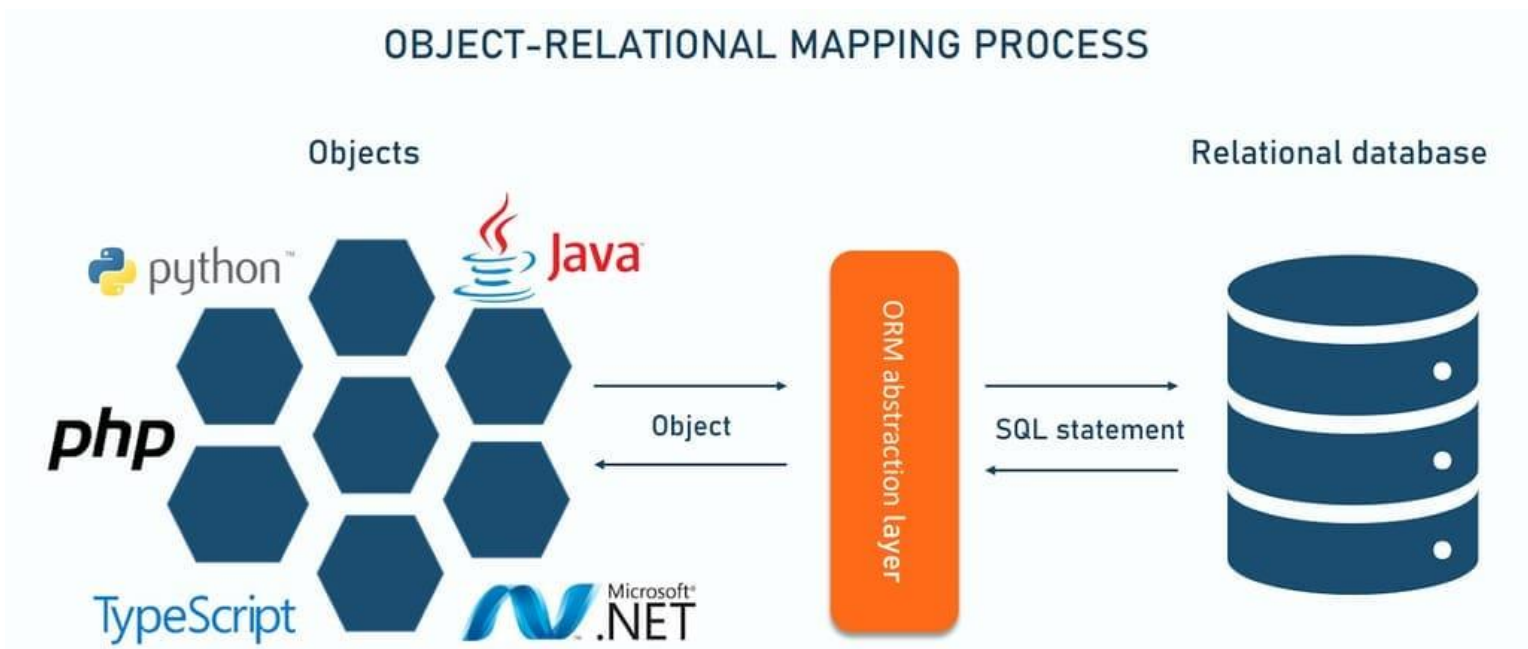
# What's Prisma?

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.



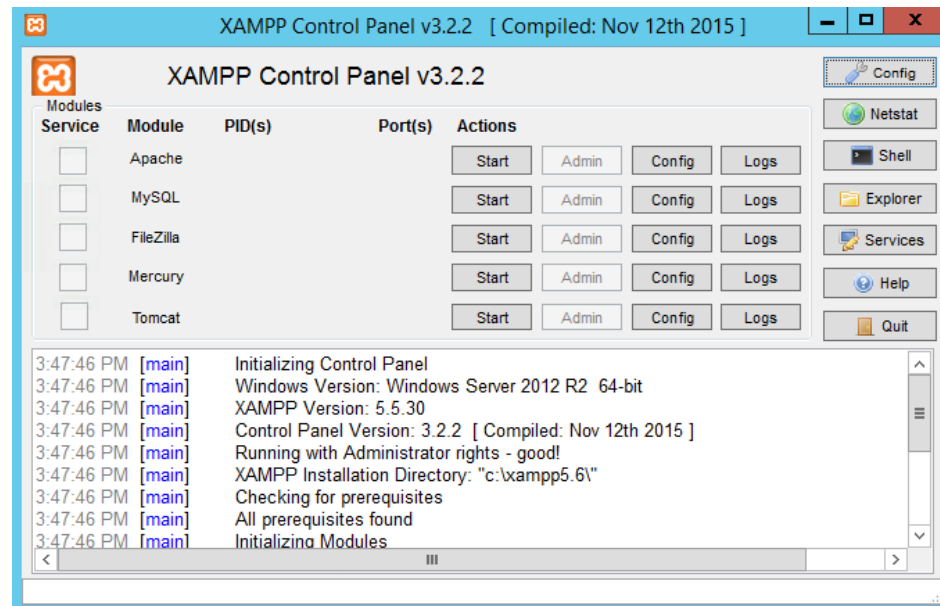
# ORM

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.



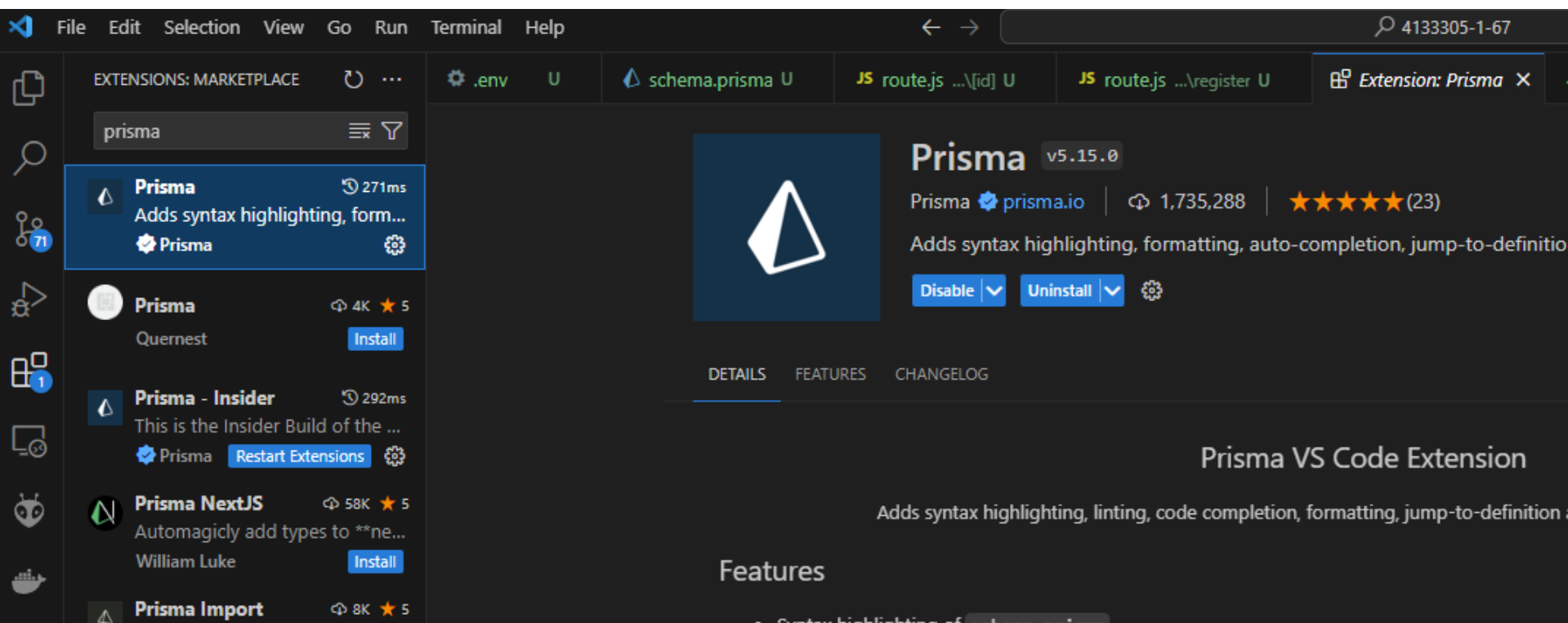
# Extra Tools for LAB

**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 interface with the Extensions Marketplace open. The search bar contains 'prisma'. The extension 'Prisma' by Prisma (prisma.io) is selected, showing its details. The extension is version 5.15.0 and has 1,735,288 installations and a 5-star rating (23 reviews). It is currently installed, with 'Disable' and 'Uninstall' buttons visible. The description states: 'Adds syntax highlighting, formatting, auto-completion, jump-to-definition...'. The 'Features' section is partially visible at the bottom.

File Edit Selection View Go Run Terminal Help

EXTENSIONS: MARKETPLACE

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

- Prisma: Adds syntax highlighting, form... (271ms)
- Prisma: Quernest (4K ★ 5) [Install]
- Prisma - Insider: This is the Insider Build of the ... (292ms) [Restart Extensions]
- Prisma NextJS: Automagically add types to \*\*ne... (58K ★ 5) [Install]
- Prisma Import: (8K ★ 5)

# 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

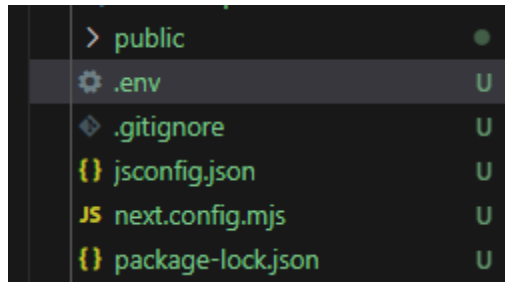
## MySQL connection URL

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@loca
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.

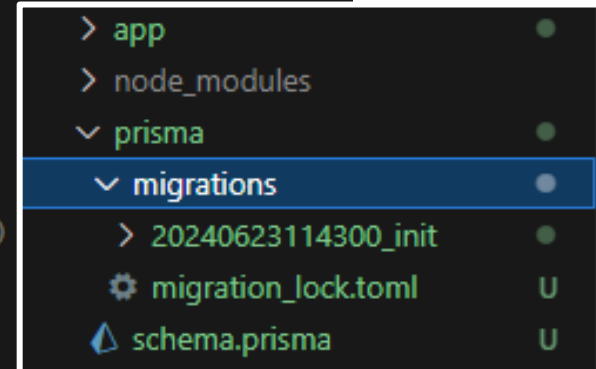
Running generate... (Use --skip-generate to skip the generators)

added 1 package, and audited 29 packages in 4s

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

found 0 vulnerabilities

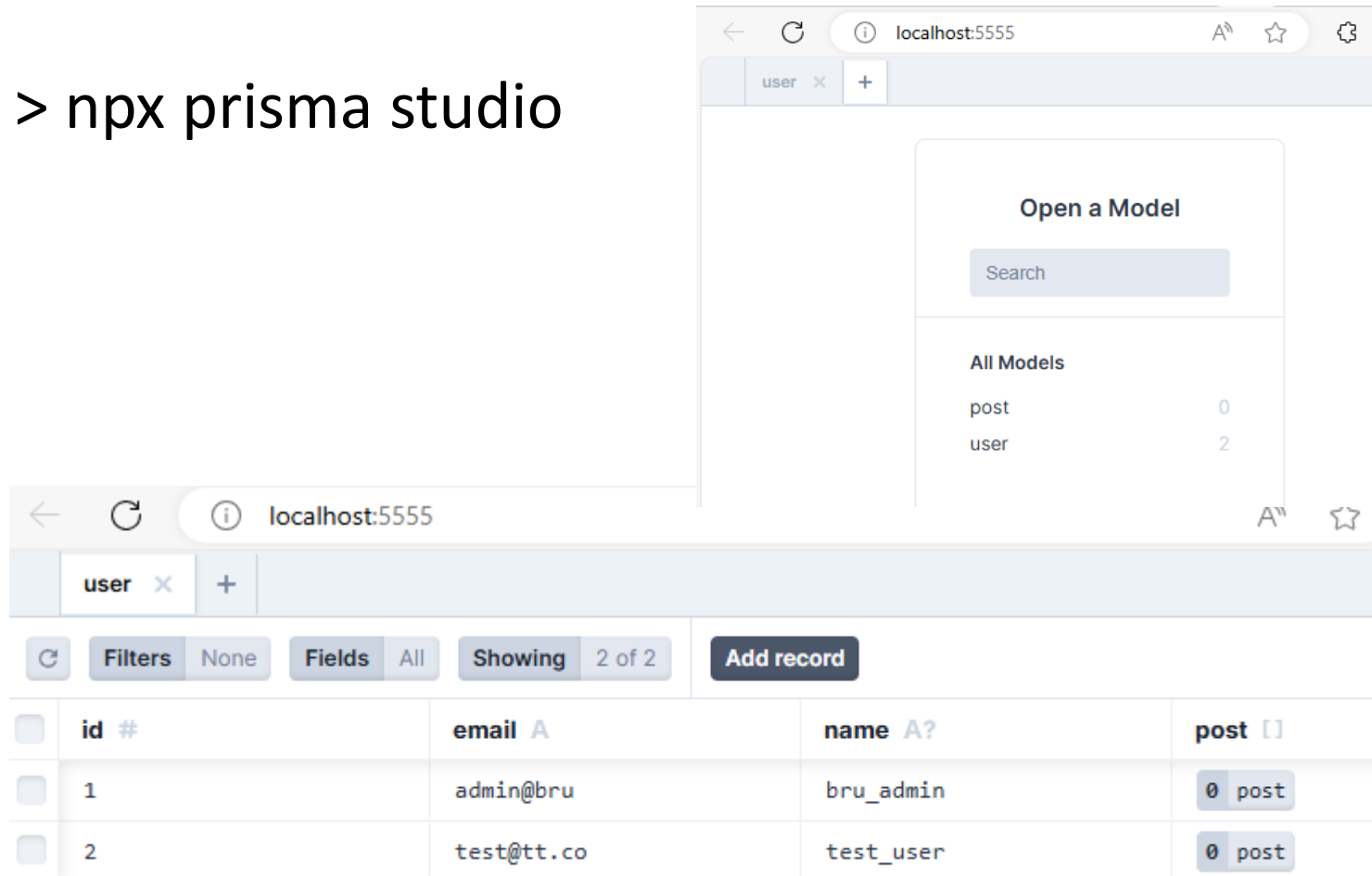
✓ Generated Prisma Client (v5.15.1) to .\node_modules\@prisma\client in 72ms
```



> npx prisma generate //if change model

# View and edit data

> npx prisma studio

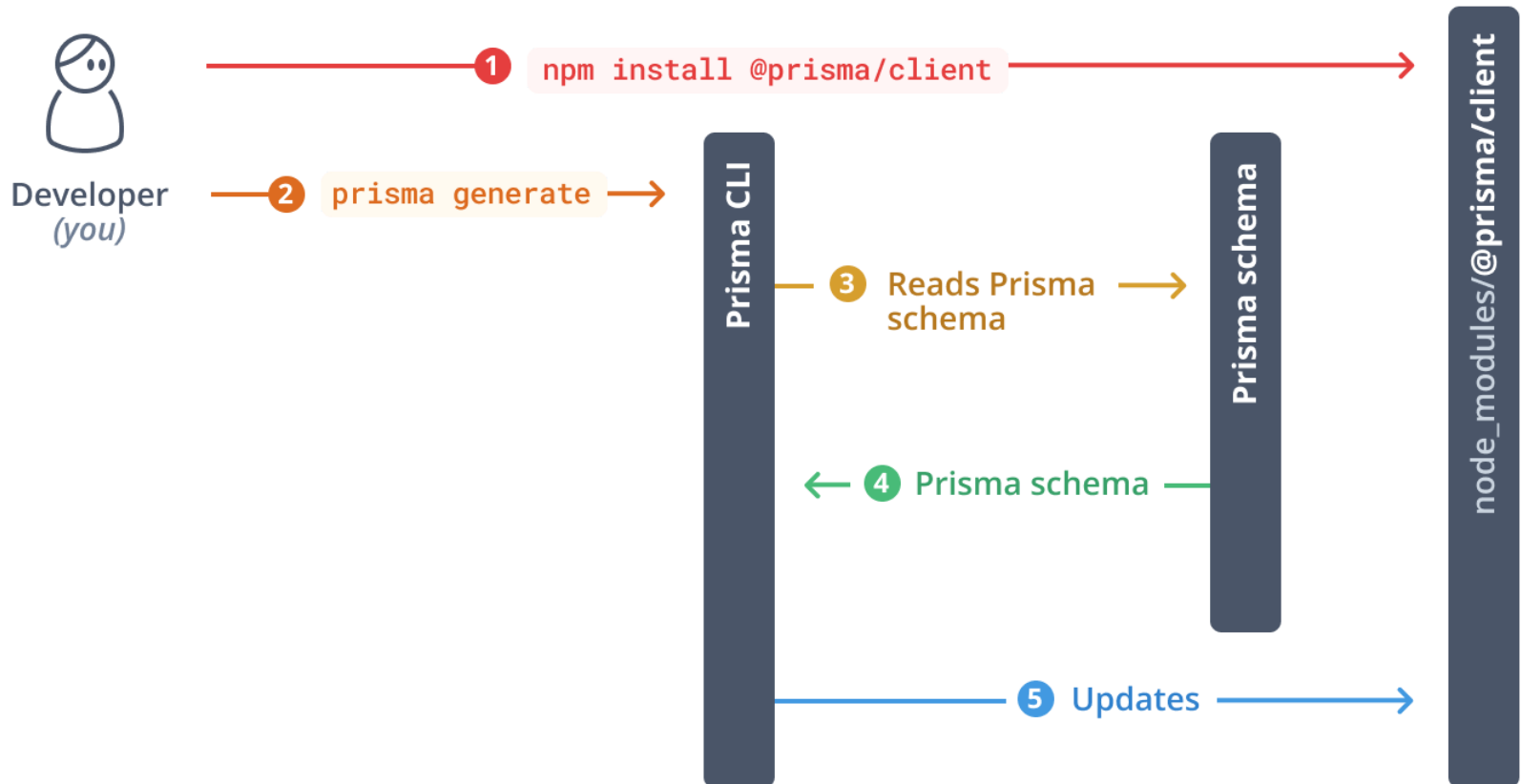


The image shows a screenshot of the Prisma Studio web interface. The browser address bar indicates the URL is localhost:5555. The interface is divided into two main sections. The top section is a modal titled "Open a Model" with a search input field and a list of "All Models" showing "post" with 0 records and "user" with 2 records. The bottom section is a table view for the "user" model, showing 2 records. The table has columns for "id #", "email A", "name A?", and "post []". The "post" column contains a dropdown menu with "0" selected and "post" as an option. The "Add record" button is visible in the top right of the table view.

<input type="checkbox"/>	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

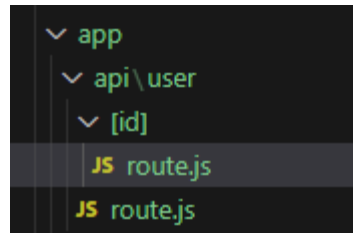
# Install Prisma Client

> 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

`findUnique()`  
`findUniqueOrThrow()`  
`findFirst()`  
`findFirstOrThrow()`  
`findMany()`  
`create()`  
`update()`  
`upsert()`  
`delete()`  
`createMany()`  
`createManyAndReturn()`  
`updateMany()`  
`deleteMany()`  
`count()`  
`aggregate()`  
`groupBy()`  
`findRaw()`  
`aggreagateRaw()`

## Model query options

`select`  
`include`  
`omit` (Preview)  
`relationLoadStrategy` (Preview)  
`where`  
`orderBy`  
`distinct`

## Nested queries

`create`  
`createMany`  
`set`  
`connect`  
`connectOrCreate`  
`disconnect`  
`update`  
`upsert`  
`delete`  
`updateMany`  
`deleteMany`

## Filter conditions and operators

`equals`  
`not`  
`in`  
`notIn`  
`lt`  
`lte`  
`gt`  
`gte`  
`contains`  
`search`  
`mode`  
`startsWith`  
`endsWith`  
`AND`  
`OR`  
`NOT`

## Relation filters

`some`  
`every`  
`none`  
`is`  
`isNot`

## Scalar list methods

`set`  
`push`  
`unset`

## Scalar list filters

Remarks  
`has`  
`hasEvery`  
`hasSome`  
`isEmpty`  
`isSet`  
`equals`

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

```
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 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

The screenshot shows a database client interface with a table named 'user'. The table has three columns: 'id', 'email', and 'name'. The data is as follows:

id	email	name
1	admin@bru	bru_admin
2	test@tt.co	test_user

## JSON on API

The screenshot shows an API client interface. The request is a GET request to the endpoint `{{host3000}}/api/user`. The response is a JSON array of two user objects.

```
4133305 / user • From [get all]
```

GET `{{host3000}}/api/user`

Params Authorization Headers (6) Body Scripts

Query Params

Key	Value
Key	Value

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON

```
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

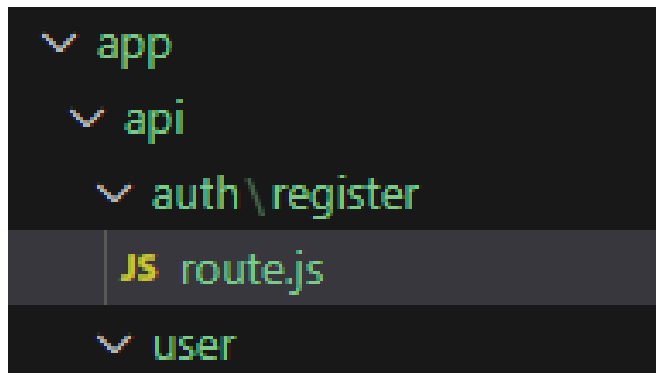
```
15 export async function DELETE(req, { params }) {
16   try {
17     const user_id = Number(params.id);
18     const result = await prisma.user.delete({
19       where: { id: user_id },
20     });
21
22     return Response.json({ des: "Deleted user:", result }, { status: 200 });
23   } catch (err)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

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

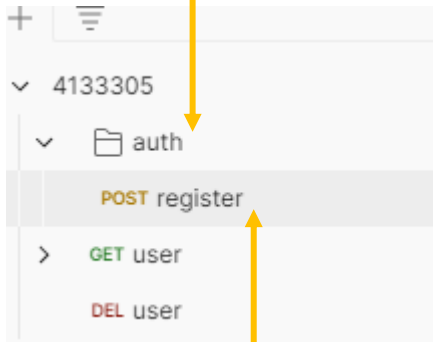
```

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 }

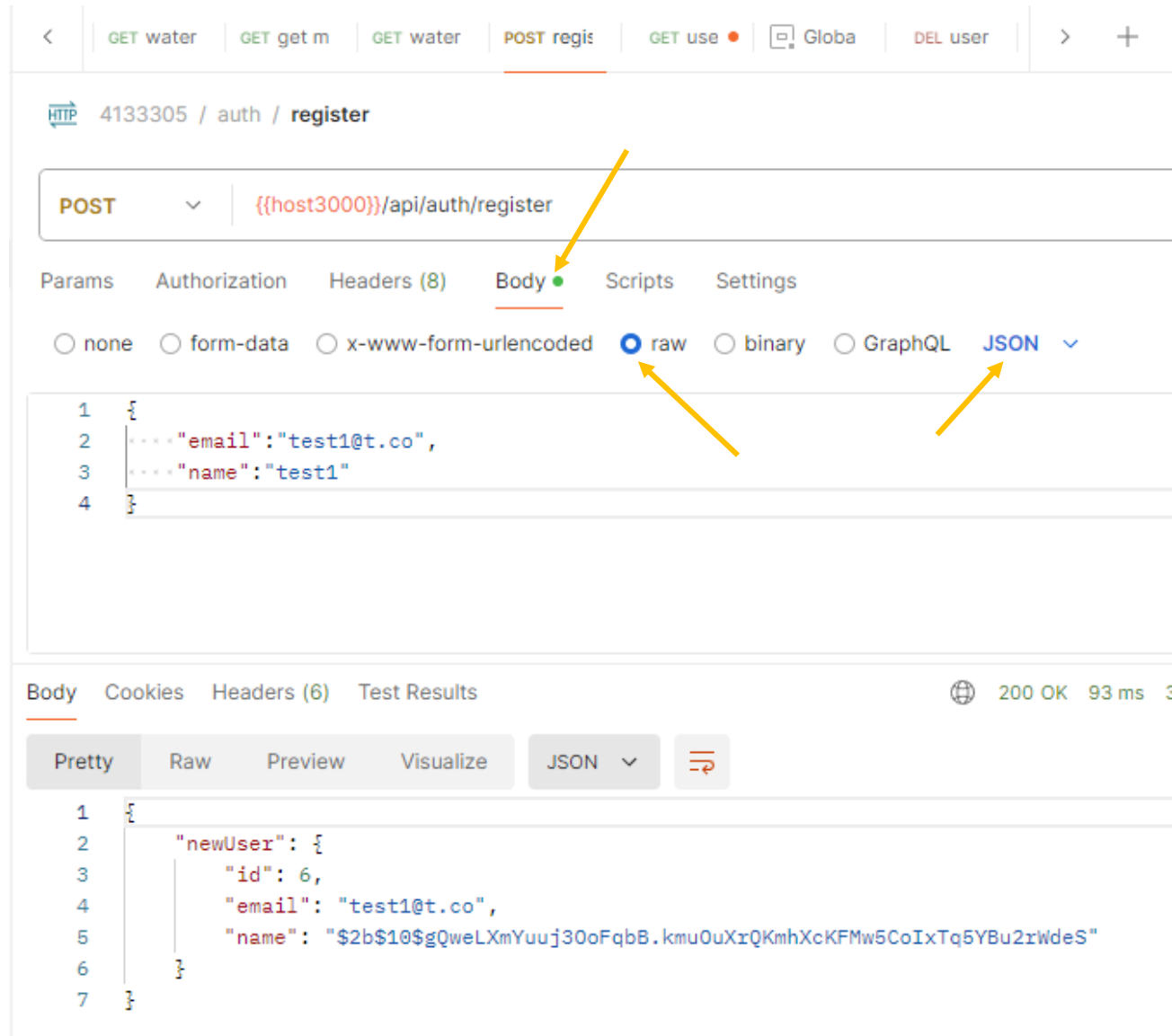
```

# Querying the database

Create **auth** folder



Create **register** by **POST** request



# 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 }
```