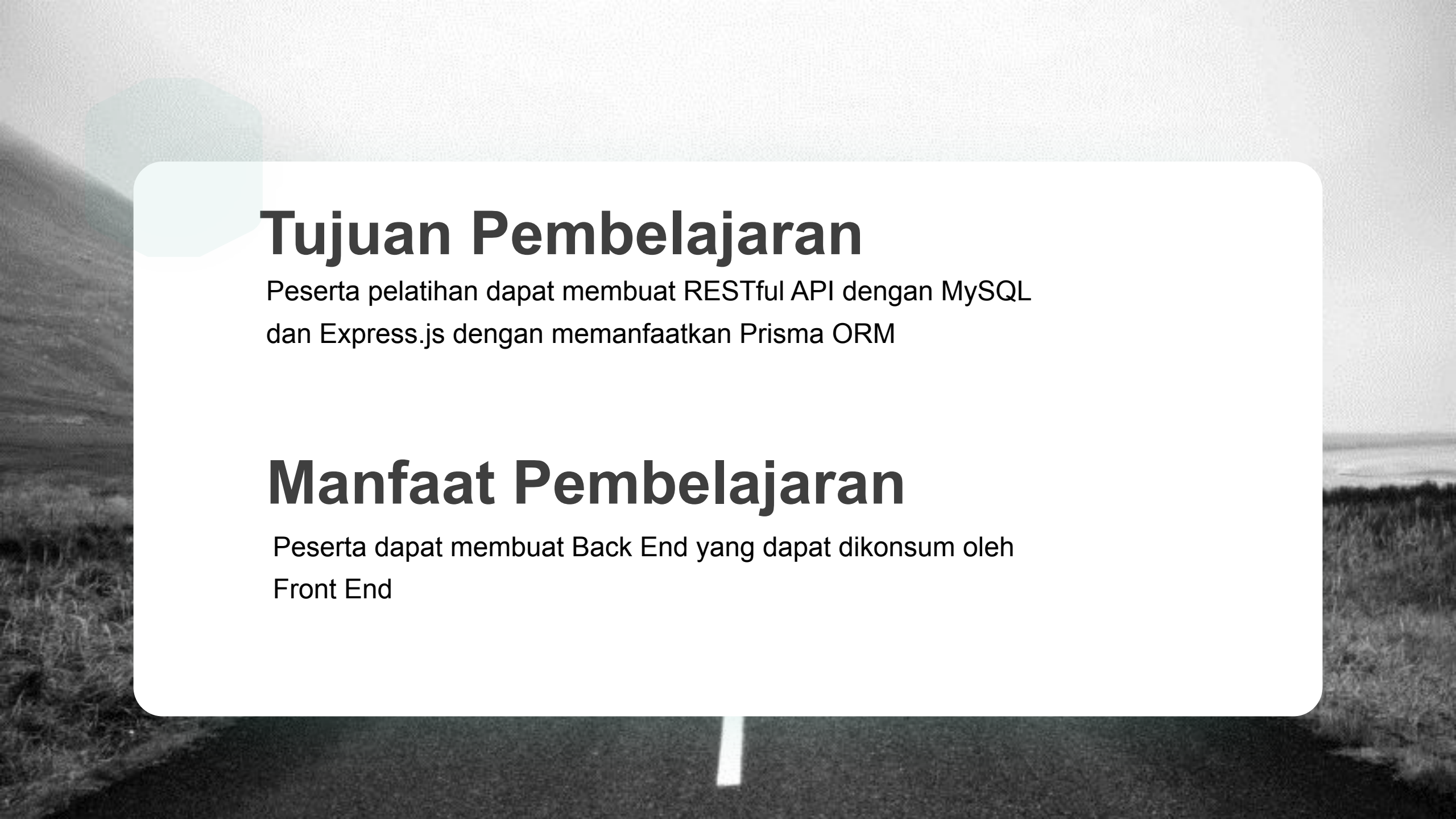

Penggunaan Express Js untuk Membuat RESTFul API dengan Database MySQL menggunakan Prisma ORM

Zulka Ali Ajuab, S.Kom

2013 - 2017 : Instruktur di BBLKI Serang

2017 - Sekarang : Instruktur di BBPVP Bekasi



Tujuan Pembelajaran

Peserta pelatihan dapat membuat RESTful API dengan MySQL dan Express.js dengan memanfaatkan Prisma ORM

Manfaat Pembelajaran

Peserta dapat membuat Back End yang dapat dikonsum oleh Front End

Express.JS

- Express.js adalah framework web untuk **Node.js** yang digunakan untuk:
  - Membuat aplikasi web atau API RESTful
  - Menangani HTTP request dan response
  - Menyusun middleware (fungsi penghubung antara permintaan dan respons)

MySQL

- Adalah sistem manajemen basis data relasional (RDBMS - Relational Database Management System) yang menggunakan Structured Query Language (SQL) sebagai bahasa utama untuk mengelola data
- MySQL bersifat open-source dan banyak digunakna untuk aplikasi web karena cepat, stabil dan gratis
- MySQL digunakan untuk
  - menyimpan data dalam tabel relasional
  - Mengelola data dengan operasi CRUD

Prisma

- Adalah ORM (Object-Relational Mapping) modern untuk Node.js dan TypeScript/JavaScript yang digunakan untuk mempermudah pengelolaan database.
- Dengan Prisma, kita tidak perlu menulis query SQL secara manual untuk melakukan operasi Database, karena Prisma menyediakan API yang type-safe dan mudah digunakan.
- Biasanya digunakan dalam pengembangan aplikasi web yang membutuhkan interaksi dengan database, misalnya di Express.js, NestJS, Next.js atau Remix

Komponen Utama Prisma

1. Prisma Client
2. Prisma Migrate
3. Prisma Studio

Komponen Utama Prisma - Prisma Client

- Ini adalah library auto-generated yang digunakan untuk mengakses database dari kode JavaScript/TypeScript
- Prisma Client dibuat berdasarkan schema Prisma yang kita definisikan
- Contoh Penggunaan :

```
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

const users = await prisma.user.findMany(); // Ambil semua data user
```

Komponen Utama Prisma - Prisma Migrate

- Alat untuk mengelola skema database melalui migrasi
- Dengan Prisma Migrate, kita bisa membuat perubahan pada model di file schema.prisma dan secara otomatis menghasilkan migrasi SQL untuk database
- Contoh perintah :

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

Komponen Utama Prisma - Prisma Studio

- GUI (Graphical User Interface) untuk melihat dan mengedit data di database secara visual
- Contoh perintah :

```
npx prisma studio
```

Struktur Dasar Prisma

- File utama Prisma adalah schema.prisma yang mendefinisikan :
  - Datasource : Database yang digunakan (MySQL, PostgreSQL, SQLite, MongoDB, dsb)
  - Generate : Menentukan jenis client yang akan di-generate
  - Model : Struktur Tabel dalam database
- Contoh schema.prisma :

```
datasource db {
  provider = "mysql" // Bisa mysql, postgresql, sqlite, mongodb
  url      = env("DATABASE_URL") // URL database dari file .env
}

generator client {
  provider = "prisma-client-js"
}

model User {
  id       Int     @id @default(autoincrement())
  name     String
  email    String  @unique
  createdAt DateTime @default(now())
}
```

Jika dijalankan **npx prisma migrate dev**, maka Prisma akan membuat tabel User di database

Membuat Projek RESTFul API dengan menggunakan Express.js, MySQL dan Prisma ORM

Pengantar Projek

1. Projek yang akan kita buat adalah projek sederhana yaitu RESTful API menggunakan Express.js + MySQL + Prisma ORM. adapun nama projeknya adalah **api-users**
2. Kita akan membuat API untuk :
   a) Membuat User baru (Create)
   b) Menampilkan semua User (Read)
   c) Menampilkan 1 User (Read By Id)
   d) Mengupdate User (Update)
   e) Menghapus User (Delete)
3. Adapun tabel User nantinya berisi : id, name, email, age, createdAt

Persiapan Lingkungan

Pastikan sudah menginstall :

1. Node.js dan npm
2. MySQL
3. Visual Studio Code
4. PhpMyAdmin / MySQL Workbench (optional)

Langkah-langkah

1. Masuk ke folder tempat projek akan dibuat dengan Command Prompt
2. Buat folder projek dan inisialisasi dengan npm

```
mkdir api-users
cd api-users
npm init -y
```

3. Install package yang diperlukan

```
npm install express cors body-parser @prisma/client
npm install prisma --save-dev
npm install --save-dev nodemon
```

- express -> framework server Node.js
- body-parser -> parsing data JSON dari request
- cors -> agar API bisa diakses dari frontend (misal React/Vue)
- nodemon -> untuk auto restart server saat coding
- @prisma/client dan prisma -> Library Prisma

Langkah-langkah

4. Ubah value pada “scripts” pada package.json

```
"scripts": {
  "start": "nodemon server.js"
}
```

5. Buat struktur proyek. Folder prisma dan file .env akan dibuat pada perintah berikutnya

```
api-users/
|-- controllers/
|    └── userController.js
|-- routes/
|    └── userRoutes.js
|-- prisma/
|    └── schema.prisma
|-- .env
|-- server.js
|-- package.json
```

Langkah-langkah

6. Inisialisasi Prisma, dengan perintah

```
npx prisma init
```

7. Konfigurasi database di .env. jika belum ada database yang dibuat, nanti bisa dibuat dengan Prisma migrate. Pada .env tambahkan script:

```
DATABASE_URL="mysql://root:@localhost:3307/users_db"
```

Langkah-langkah

8. Definisikan Model di prisma/schema.prisma, dengan menambahkan script :

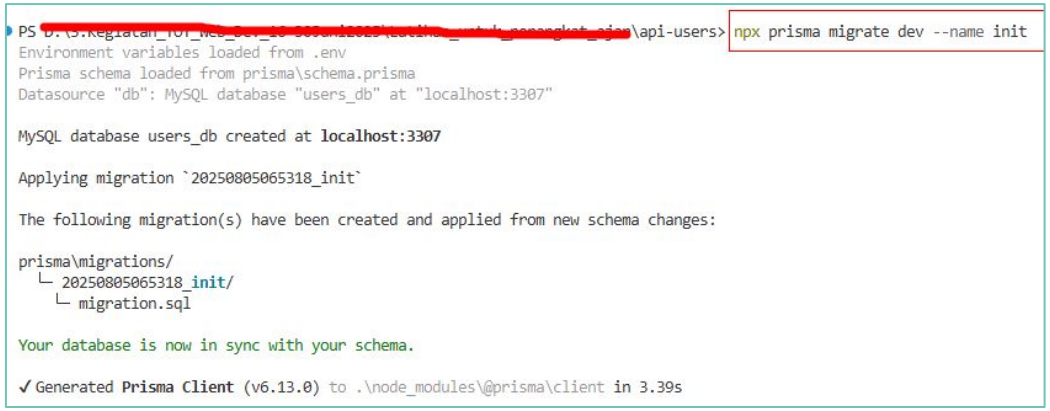
```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
}

model User {
  id        Int      @id @default(autoincrement())
  name      String
  email     String   @unique
  age       Int
  createdAt DateTime @default(now())
}
```

Langkah-langkah

9. Pada terminal atau command prompt, jalankan perintah :

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

Perintah di atas akan membuat database users_db, membuat tabel User dan menghasilkan Prisma Client



10. Tambahkan baris perintah pada file controllers/userController.js

```
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();
```

Langkah-langkah

11. Tambahkan baris perintah untuk fungsi createUser

```
// CREATE - Tambah user
exports.createUser = async (req, res) => {
  try {
    const { name, email, age } = req.body;
    const user = await prisma.user.create({
      data: { name, email, age: Number(age) },
    });
    res.json({ message: 'User created successfully', user });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

Langkah-langkah

12. Masih pada file yang sama tambahkan fungsi getAllUsers

```
// READ - Semua user
exports.getAllUsers = async (req, res) => {
  try {
    const users = await prisma.user.findMany();
    res.json(users);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

Langkah-langkah

13. Tambahkan fungsi getUserById

```
// READ - User by ID
exports.getUserById = async (req, res) => {
  try {
    const { id } = req.params;
    const user = await prisma.user.findUnique({
      where: { id: Number(id) },
    });
    if (!user) return res.status(404).json({ message: 'User not found' });
    res.json(user);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

Langkah-langkah

14. Tambahkan fungsi updateUser

```
// UPDATE - User by ID
exports.updateUser = async (req, res) => {
  try {
    const { id } = req.params;
    const { name, email, age } = req.body;
    const user = await prisma.user.update({
      where: { id: Number(id) },
      data: { name, email, age: Number(age) },
    });
    res.json({ message: 'User updated successfully', user });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

Langkah-langkah

15. Tambahkan fungsi deleteUser

```
// DELETE - User by ID
exports.deleteUser = async (req, res) => {
  try {
    const { id } = req.params;
    await prisma.user.delete({ where: { id: Number(id) } });
    res.json({ message: 'User deleted successfully' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

Langkah-langkah

16. Pada file routes/userRoutes.js tambahkan baris perintah

```
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');

router.post('/', userController.createUser);
router.get('/', userController.getAllUsers);
router.get('/:id', userController.getUserById);
router.put('/:id', userController.updateUser);
router.delete('/:id', userController.deleteUser);

module.exports = router;
```

Langkah-langkah

17. Pada file server.js tambahkan baris perintah

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const userRoutes = require('./routes/userRoutes');

const app = express();
app.use(cors());
app.use(bodyParser.json());

app.use('/api/users', userRoutes);

const PORT = 3001; // berbeda dengan api-produk
app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

Langkah-langkah

18. Generate Prisma Client dengan perintah **npx prisma generate**. Baris perintah ini digunakan untuk membuat / update Prisma Client agar query database sesuai model terbaru. Jalankan perintah ini untuk :
    - Setelah mengubah schema.prisma
    - Setelah menjalankan npx prisma migrate dev
    - Setelah pertama kali install project Prisma

19. Jalankan server dengan perintah **npm start**

Langkah-langkah

15. Jalankan aplikasi dengan mengetikkan “npm start” di dalam folder proyek dalam lingkungan **command prompt**

```
PS [redacted]api-users> npm start

> api-users@1.0.0 start
> nodemon server.js

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Server running at http://localhost:3001
```

Ujicoba Backend

Langkah-langkah Ujicoba

1. Sama dengan uji coba API sebelumnya, bisa menggunakan Postman atau extension thunder client pada Vs.Code
2. Untuk menjalankannya url yang digunakan adalah http://localhost:3001 + Endpoint RESTfull seperti pada list di bawah:

Endpoint	Method	Keterangan	Body Request (JSON)	Contoh Response (JSON)
`/api/users`	GET	Menampilkan semua user	-	`[{"id":1,"name":"John Doe","email":"john@example.com","age":30,"createdAt":"2025-08-05T07:00:00.000Z"}, {...}]`
`/api/users/:id`	GET	Menampilkan satu user berdasarkan ID	-	`{"id":1,"name":"John Doe","email":"john@example.com","age":30,"createdAt":"2025-08-05T07:00:00.000Z"}`
`/api/users`	POST	Menambahkan user baru	`{ "name": "John Doe", "email": "john@example.com", "age": 30 }`	`{"message":"User created successfully","user":{"id":1,"name":"John Doe","email":"john@example.com","age":30,"createdAt":"2025-08-05T07:00:00.000Z"}}`
`/api/users/:id`	PUT	Mengupdate data user berdasarkan ID	`{ "name": "John Updated", "email": "johnupd@example.com", "age": 31 }`	`{"message":"User updated successfully","user":{"id":1,"name":"John Updated","email":"johnupd@example.com","age":31,"createdAt":"2025-08-05T07:00:00.000Z"}}`
`/api/users/:id`	DELETE	Menghapus user berdasarkan ID	-	`{"message":"User deleted successfully"}`

Langkah-langkah Ujicoba

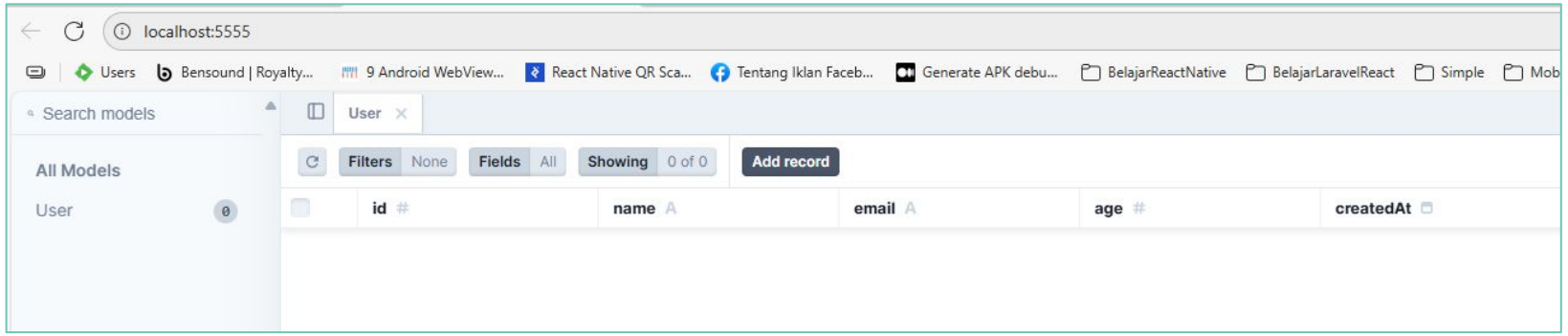
Adapun syarat dan keterangan masing - masing API adalah :

1. GET /api/users
    1. Mengambil semua user dari database.
    2. Cocok untuk menampilkan daftar user di frontend.
2. GET /api/users/:id
    1. Mengambil 1 user berdasarkan ID.
    2. Jika user tidak ditemukan, mengembalikan status 404 dengan { "message": "User not found" }.
3. POST /api/users
    1. Membuat user baru.
    2. Field email harus unik.
4. PUT /api/users/:id
    1. Mengupdate data user berdasarkan ID.
    2. Jika ID tidak ditemukan, akan memunculkan error.
5. DELETE /api/users/:id
    1. Menghapus user dari database.
    2. Biasanya butuh konfirmasi di frontend agar tidak salah hapus.

npx prisma studio

npx prisma studio adalah perintah untuk menampilkan data pada database dalam bentuk GUI

```
PS D:\3.Kegiatan_TOT_Web_Dev_16-30Juni2023\Latihan_untuk_peserta\api-users> npx prisma studio
Environment variables loaded from .env
Prisma schema loaded from prisma\schema.prisma
Prisma Studio is up on http://localhost:5555
```




TERIMA
KASIH