LAB-12: Database Access using Prisma ORM(Object-Relation Mapping) in NextJS

- **Object-Relational Mapping (ORM):** A technique in programming that maps database tables (relational) to objects in code (object-oriented), allowing developers to work with data using familiar objects instead of writing raw SQL.
- **Examples:** Tools like Hibernate (Java) or SQLAlchemy (Python) are ORMs.

STEP 1: DATABASE SETUP (MySQL)

Create Database:
```
CREATE DATABASE awt_db;
USE awt_db;
```

Create Table:
```
CREATE TABLE products (
  id INT PRIMARY KEY AUTO_INCREMENT,
  title VARCHAR(100),
  price INT,
  category VARCHAR(50),
  description TEXT
);
```

Insert Sample Data:
```
INSERT INTO products (title, price, category, description) VALUES
('Laptop', 55000, 'Electronics', 'High performance laptop'),
('Mobile', 25000, 'Electronics', 'Android smartphone'),
('Headphones', 3000, 'Accessories', 'Noise cancelling'),
('Chair', 7000, 'Furniture', 'Office chair');
```

STEP 2: CREATE NEXTJS PROJECT

```
npx create-next-app awt-lab12
cd awt-lab12
npm run dev
```

We have done this in the last lab so we will continue with the previous project.

STEP 3: INSTALL PRISMA

npm install prisma --save-dev
npm install prisma@5 @prisma/client@5
For new prisma
Also install @prisma/client @prisma/adapter-mariadb dotenv

Initialize Prisma:
npx prisma init

This will create a root folder named prisma and a file named schema.prisma inside it.---------
---
STEP 4: CONFIGURE DATABASE CONNECTION

File: .env

DATABASE_URL="mysql://root@localhost:3306/awt_db"

STEP 5: DEFINE PRISMA MODEL
File: prisma/schema.prisma

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

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

model Product {
  id          Int     @id @default(autoincrement())
  title       String
  price       Int
  category    String
  description String
}
```

STEP 6: GENERATE PRISMA CLIENT
npx prisma db pull
npx prisma generate
---
npx prisma studio->you can also use this command. By this browser will open -> live db ui

STEP 7: CREATE PRISMA CLIENT INSTANCE

File: lib/prisma.ts

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

const globalForPrisma = global as unknown as {
  prisma: PrismaClient;
};

export const prisma =
  globalForPrisma.prisma ||  new PrismaClient();
```

STEP 8: FETCH ALL RECORDS (GET ALL)

File: app/products/page.tsx

```
import { prisma } from "@/lib/prisma";
import Link from "next/link";

export default async function Products() {
  const products = await prisma.product.findMany();

  return (
    <div>
      <h1>Product List</h1>
      <table border={1} cellPadding={10}>
        <thead>
          <tr>
            <th>ID</th>
            <th>Title</th>
            <th>Price</th>
            <th>Category</th>
            <th>Details</th>
          </tr>
        </thead>

        <tbody>
          {products.map((p:any) => (
            <tr key={p.id}>
              <td>{p.id}</td>
              <td>{p.title}</td>
              <td>₹{p.price}</td>
              <td>{p.category}</td>
              <td><Link href={"/products/"+p.id}>Detail</Link></td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
```

```
}
```

STEP 9: FETCH RECORD USING ID (GET BY ID)

Dynamic Route:
app/products/[id]/page.tsx

Code:

```
import { prisma } from "@/lib/prisma";

export default async function Product({ params }: {params:Promise<{id:string}>}) {
 const {id} = await params;
 const product = await prisma.product.findUnique({
  where: {
   id: id,
  },
 });

 if (!product) {
  return <h2>Product not found</h2>;
 }

 return (
  <div>
   <h1>{product.title}</h1>
   <p><strong>Price:</strong> ₹{product.price}</p>
   <p><strong>Category:</strong> {product.category}</p>
   <p>{product.description}</p>
  </div>
 );
}
```