PEMROGRAMAN WEB LANJUT — EDISI KOMPREHENSIF

Arsitektur Full-Stack: Next.js App Router & Prisma ORM pada PostgreSQL

Kuliah mendalam 25+ slide mengenai teknik pengembangan aplikasi web enterprise modern.

Agenda Perkuliahan (Bagian 1)

Modul 1 hingga 4: Fondasi Arsitektur Web Modern dan Routing.

1. Ekosistem Modern

Evolusi web, tren full-stack, dan perbandingan pola arsitektur masa kini.

2. App Router Next.js

Mekanisme file-system routing dan filosofi perancangan direktori app.

3. Komponen Render

Perbedaan mendasar antara Server Components dan Client Components.

Agenda Perkuliahan (Bagian 2)

Modul 5 hingga 8: Manajemen Database dan Studi Kasus Implementasi.

4. Prisma & PostgreSQL

Pemodelan data relasional, skema, migrasi, dan keamanan type-safety.

5. API & Actions

Penggunaan Route Handlers dan Server Actions untuk mutasi data aman.

6. Studi Kasus

Membangun sistem inventaris perguruan tinggi end-to-end secara lengkap.

Modul 1: Evolusi Pengembangan Web

Perubahan paradigma dari aplikasi monolitik tradisional ke arsitektur modern.

Traditional SSR

Server merender seluruh halaman HTML secara utuh setiap kali terjadi perpindahan route.

Single Page App (SPA)

Browser memuat bundle JavaScript besar dan merender antarmuka sepenuhnya di sisi klien.

Hybrid Full-Stack

Menggabungkan kecepatan Server Components dengan interaktivitas Client Components.

Keunggulan Arsitektur Next.js

Alasan utama framework React tingkat lanjut ini menjadi standar industri.

Performa Optimal

Rendering berbasis server mengurangi ukuran bundle JavaScript yang dikirim ke browser pengguna.

SEO Ramah Mesin Pencari

Halaman dimuat lengkap dengan teks HTML, memudahkan crawler melakukan indeks konten.

Integrasi Backend Terpadu

Kemampuan menulis kode frontend dan logika database/API dalam satu repositori yang sama.

Modul 2: Konsep Dasar App Router

Revolusi routing berbasis direktori file-system pengganti system lama.

Konvensi Direktori /app

Setiap folder secara otomatis mencerminkan segmen path URL tanpa konfigurasi router eksternal.

Server Components Default

Komponen dieksekusi di server secara default untuk meminimalkan beban komputasi klien.

```
// Struktur Folder App Router

app/
  ├── layout.js   // Kerangka global utama
  ├── page.js     // Halaman utama (/)
  └── produk/
    ├── page.js   // Halaman daftar (/produk)
    └── [id]/
      └── page.js  // Detail dinamis (/produk/1)
```

File Khusus dalam App Router

File standar untuk mengatur layout, status muatan, error, dan penanganan khusus.

layout.js

Membagikan antarmuka UI di antara beberapa halaman dan mempertahankan state saat navigasi.

loading.js

Menampilkan status muatan instan berbasis React Suspense secara otomatis saat data diambil.

error.js

Mencegah crash aplikasi total dengan batas error spesifik per segmen direktori.

Dynamic Routing Lanjutan

Menangani parameter URL tunggal, opsional, dan catch-all segments.

Dynamic [id]

Menangkap satu parameter variabel unik seperti ID produk atau profil pengguna.

Catch-all [...slug]

Menangkap seluruh segmen URL bersarang untuk dokumentasi atau sistem artikel.

Optional [[...slug]]

Menangkap parameter bersarang sekaligus mengizinkan route tanpa parameter tersebut.

Route Handlers dalam Next.js

Membuat endpoint API kustom menggunakan standar Request & Response Web.

Metode HTTP yang Didukung

Mendukung fungsi standar `GET`, `POST`, `PUT`, dan `DELETE` dalam satu file `route.js`.

```
// app/api/produk/route.js
import { NextResponse } from 'next/server';

export async function GET(request) {
  const data = [{ id: 1, nama: 'Laptop Lab' }];
  return NextResponse.json(data);
}
```

Modul 3: Server vs Client Components

Memahami batas eksekusi kode di server dan interaktivitas di browser.

Server Components (RSC)

Default di Next.js. Ideal untuk akses database langsung, menjaga kerahasiaan API key, dan fetch data.

Client Components

Diaktifkan dengan `'use client'`. Diperlukan untuk event listener, state React, dan useEffect.

Panduan Memilih Komponen

Matriks keputusan kapan harus menggunakan Server atau Client Component.

Gunakan Server Component jika:

Mengambil data dari database, mengakses backend langsung, atau merender konten statis berat.

Gunakan Client Component jika:

Membutuhkan interaksi klik, form input useState, hook browser, atau animasi dinamis.

Modul 4: Pengantar Prisma ORM & Postgres

Menghubungkan aplikasi web modern ke database relasional tangguh secara type-safe.

PostgreSQL

Database relasional object-relasional enterprise yang handal, mendukung integritas data dan transaksi ACID.

Prisma Schema

Definisi model data deklaratif tunggal yang bersih dan mudah dibaca untuk memetakan tabel database.

Prisma Client

Klien query database yang aman secara tipe (type-safe) otomatis berdasarkan skema yang Anda tentukan.

Konfigurasi Skema Database

Mendefinisikan koneksi datasource dan generator client pada file skema.

```
// prisma/schema.prisma

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

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

Environment Variables

Koneksi URL disimpan aman di file `.env` untuk menghindari kebocoran kredensial.

Client Generation

Perintah `npx prisma generate` membuat helper TypeScript otomatis.

Pemodelan Entitas & Relasi One-to-Many

Menghubungkan entitas Kategori dan Produk dalam relasi relational database.

```
model Kategori {
  id      Int     @id @default(autoincrement())
  nama    String
  produk  Produk[]
}

model Produk {
  id      Int     @id @default(autoincrement())
  nama    String
  harga   Float
  stok    Int     @default(0)
  katId   Int
  kategori Kategori @relation(fields: [katId], references:
[id])
}
```

Atribut Primary & Foreign Key

Penggunaan `@id` dan `@relation` menyusun integritas referensial secara instan.

Manajemen Migrasi

Perintah `prisma migrate dev` mencatat riwayat perubahan skema database.

Inisialisasi Prisma Client (Singleton)

Mencegah kelebihan koneksi database pada mode hot-reloading development.

```
// lib/prisma.js
import { PrismaClient } from '@prisma/client';

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

if (process.env.NODE_ENV !== 'production') globalForPrisma.prisma = prisma;

export default prisma;
```

Modul 5: Studi Kasus Sistem Inventaris

Penerapan nyata integrasi Next.js dan Prisma dalam studi kasus inventaris kampus.

1. Fitur Daftar (Read)

Menampilkan tabel inventaris barang dengan relasi kategori langsung dari database PostgreSQL.

2. Tambah Data (Create)

Menggunakan Server Actions untuk memproses form input dan memasukkan data baru.

3. Hapus Data (Delete)

Melakukan penghapusan baris data dengan validasi ID dan penyegaran halaman otomatis.

Implementasi Read: Menampilkan Data

Contoh kode Server Component untuk mengambil daftar produk beserta kategorinya.

```
// app/produk/page.js

import prisma from '@/lib/prisma';

export default async function HalamanProduk() {
  const daftarProduk = await prisma.produk.findMany({
    include: { kategori: true },
  });

  return (
    <div>
      <h1>Daftar Inventaris Produk</h1>
      {daftarProduk.map(p => (<p key={p.id}>{p.nama} - {p.kategori.nama}</p>))}
    </div>
  );
}
```

Server Actions untuk Mutasi Data

Menjalankan fungsi sisi server langsung dari form HTML tanpa membuat API terpisah.

```
// app/produk/actions.js

'use server'
import prisma from '@/lib/prisma';
import { revalidatePath } from 'next/cache';

export async function tambahProduk(formData) {
  const nama = formData.get('nama');
  const harga = parseFloat(formData.get('harga'));
  await prisma.produk.create({ data: { nama, harga, katId: 1 } });
  revalidatePath('/produk');
}
```

Validasi Data & Penanganan Error

Menjamin keandalan aplikasi dengan memeriksa masukan pengguna sebelum masuk database.

Validasi Sisi Server

Selalu validasi tipe data string, angka, dan batas rentang nilai di dalam Server Actions.

Blok Try-Catch

Menangkap kegagalan koneksi atau pelanggaran constraint unik database dengan elegan.

Umpan Balik Pengguna

Memberikan pesan status berhasil atau gagal yang jelas pada antarmuka pengguna.

Optimasi Performa & Caching

Strategi penyegaran cache dan pemanfaatan ISR (Incremental Static Regeneration).

Router Cache

Next.js menyimpan cache halaman di memori browser untuk perpindahan rute instan.

revalidatePath

Fungsi untuk membersihkan cache secara spesifik setelah terjadi perubahan data.

Database Indexing

Menambahkan indeks pada kolom sering dicari untuk mempercepat query PostgreSQL.

Keamanan Aplikasi Web (Security)

Mitigasi kerentanan umum dalam pengembangan aplikasi web modern.

Pencegahan SQL Injection

Prisma secara otomatis menggunakan parameterized queries untuk mencegah serangan injeksi SQL.

Sanitasi Masukan

Melindungi aplikasi dari serangan Cross-Site Scripting (XSS) pada form input pengguna.

Proteksi Kredensial

Memastikan variabel penting lingkungan tidak pernah terekspos ke kode sisi klien.

Pengujian & Debugging Aplikasi

Teknik memastikan kualitas kode sebelum aplikasi dirilis ke server produksi.

Prisma Studio

Antarmuka berbasis web visual untuk melihat dan memodifikasi isi database secara langsung.

TypeScript Checking

Mendeteksi kesalahan tipe data secara dini pada waktu proses kompilasi kode.

Logging Kueri

Mengaktifkan log query SQL pada Prisma untuk menganalisis performa kueri database.

Deployment ke Lingkungan Produksi

Langkah publikasi aplikasi web Next.js dan database PostgreSQL ke cloud.

Vercel Hosting

Platform optimal buatan pencipta Next.js untuk deployment otomatis berbasis Git.

Managed PostgreSQL

Penyedia database cloud seperti Supabase, Neon, atau AWS RDS untuk skalabilitas tinggi.

Build Command

Menjalankan `prisma migrate deploy` secara otomatis pada siklus build produksi.

Rangkuman & Praktik Terbaik

Poin kunci dalam merancang aplikasi web dengan Next.js dan Prisma.

1. Manfaatkan File-System Routing

Atur struktur folder dengan rapi di dalam direktori `app/` untuk kemudahan pemeliharaan.

2. Jaga Koneksi Database

Gunakan pola singleton untuk Prisma Client agar tidak terjadi kelebihan koneksi pada PostgreSQL.

3. Prioritaskan Server Components

Lakukan query database langsung di server untuk keamanan kredensial dan performa rendering.

Sesi Tanya Jawab & Penutup

Terima kasih. Ada pertanyaan seputar arsitektur Next.js atau Prisma ORM?