Modul Praktikum Bagian 1 - Desain dan Pengembangan Sistem Informasi

Pengembangan API menggunakan NodeJS dan Express

Informasi Penulis

Penulis:

Farid Suryanto, Dosen Program Studi Sistem Informasi Universitas Ahmad Dahlan. Modul ini dibuat dengan bantuan ChatGPT menggunakan model GPT-3.5.

E-mail:

farid.suryanto@is.uad.ac.id

Versi

Cetak Versi 1.0 - Rilis pada 25/05/2024

Pengantar

Selamat datang di modul pengembangan API menggunakan Node.js dan Express! Dalam modul ini, kita akan mempelajari cara membangun API yang kuat dan terukur menggunakan dua teknologi yang sangat populer di kalangan pengembang web modern: Node.js dan Express. Node.js adalah runtime JavaScript yang berjalan di server, sementara Express adalah framework minimalis untuk Node.js yang mempermudah pembuatan dan pengelolaan API.

Apa yang Akan Dipelajari?

Dalam modul ini, kita akan melalui serangkaian langkah praktis untuk membangun API, mulai dari tahap dasar hingga fitur yang lebih kompleks. Topik-topik yang akan dibahas meliputi:

1. **Memulai dengan Node.js dan Express**:
    - Pengantar dan pemasangan Node.js dan Express.
    - Membuat dan menjalankan proyek baru menggunakan Express CLI.
2. **Membangun API Sederhana**:
    - Membuat endpoint GET dan POST.
    - Mengelola data produk dalam format JSON.
3. **Menghubungkan API dengan Database MySQL**:
    - Menyiapkan koneksi ke database MySQL menggunakan Sequelize.
    - Membuat model data yang merepresentasikan struktur tabel di database.
4. **CRUD (Create, Read, Update, Delete) Operasi**:
    - Membuat endpoint untuk operasi CRUD pada entitas seperti produk dan kategori produk.
    - Menyusun file dan struktur direktori yang efisien.
5. **Autentikasi dan Otorisasi**:
    - Menerapkan mekanisme login dan otorisasi menggunakan JWT.
    - Mengelola peran pengguna seperti admin dan user.
6. **Mengunggah Gambar**:
    - Mengimplementasikan fitur upload gambar untuk profil pengguna.
7. **Best Practices dan Deployment**:
    - Menerapkan best practices dalam pengembangan API.
    - Menyiapkan aplikasi untuk deployment ke lingkungan produksi.

Persiapan Sebelum Memulai

Sebelum kita mulai, pastikan Anda memiliki persiapan berikut:

1. **Komputer dengan Sistem Operasi yang Mendukung**:
    - Anda dapat menggunakan Windows, macOS, atau Linux.
2. **Instalasi Node.js dan npm**:
    - Unduh dan instal Node.js dari nodejs.org. Node.js sudah termasuk npm (Node Package Manager), yang akan kita gunakan untuk mengelola paket-paket yang diperlukan.
3. **Text Editor atau IDE**:
    - Gunakan text editor atau IDE yang Anda sukai. Beberapa pilihan populer adalah Visual Studio Code, Sublime Text, atau Atom.
4. **MySQL Database**:
    - Pastikan Anda memiliki MySQL terinstal dan berjalan di sistem Anda. Anda dapat mengunduhnya dari mysql.com atau menggunakan layanan MySQL yang di-hosting. Anda juga dapat menggunakan MySQL yang disediakan oleh pihak ketiga seperti XAMPP untuk penggunaan yang lebih mudah pada perangkat lokal Anda.
5. **Postman atau Talend REST Client**:

- Untuk menguji API yang kita buat, Anda memerlukan tool seperti Postman atau Talend REST Client. Postman bisa diunduh dari postman.com.

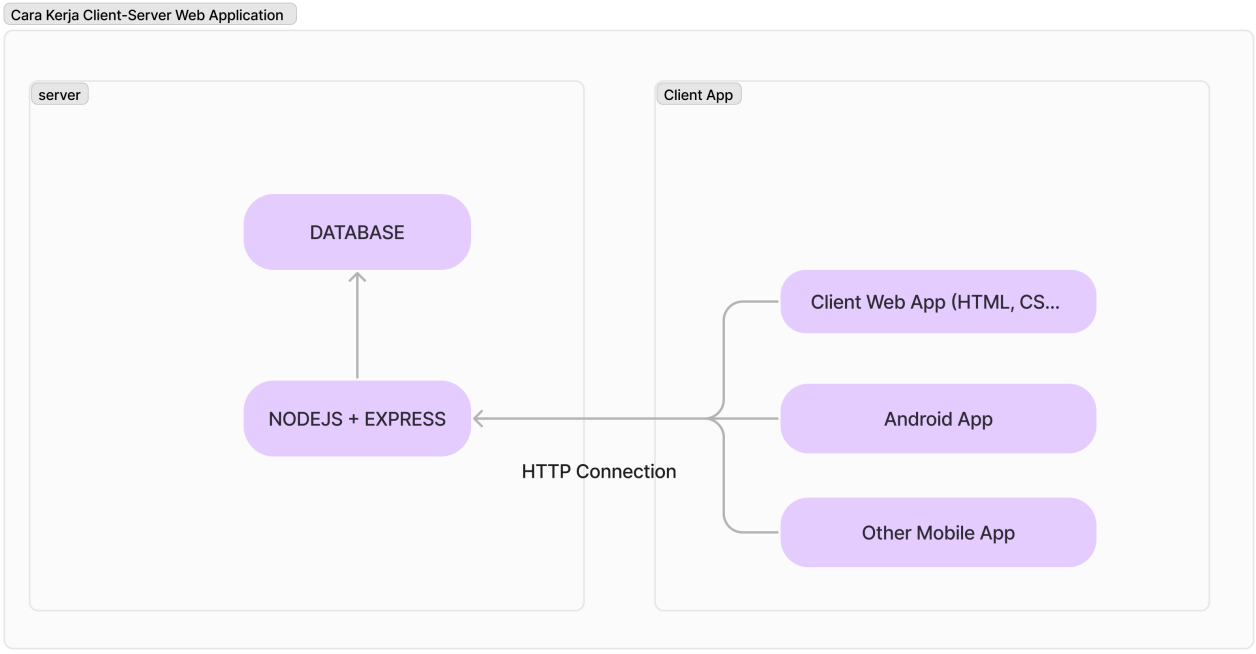
6. **Dasar Pengetahuan JavaScript**:
    - Meskipun kita akan membahas banyak hal dalam modul ini, memiliki pemahaman dasar tentang JavaScript akan sangat membantu.

Dengan persiapan ini, Anda siap untuk memulai perjalanan dalam membangun API yang kuat dan handal menggunakan Node.js dan Express. Mari kita mulai dengan memahami dasar-dasar dan membangun fondasi proyek kita!

Daftar Isi

1 - Pola Pengembangan Aplikasi Web Client-Server

Dalam pengembangan aplikasi web modern, salah satu pola yang paling umum digunakan adalah pola client-server. Pola ini membagi aplikasi menjadi dua bagian utama, yaitu server side (sisi server) dan client side (sisi klien). Pembagian ini memungkinkan pengembangan yang lebih terstruktur, terorganisir, dan mudah dikelola. Gambar berikut menunjukkan pola pengembangan aplikasi dengan pola klien-server menggunakan framework NodeJS dan Express pada sisi server.



Berikut adalah penjelasan sederhana mengenai masing-masing bagian:

1. Server Side (Sisi Server)

Sisi server adalah bagian dari aplikasi yang beroperasi di server, yaitu komputer yang menyediakan layanan kepada klien. Fungsi utama dari sisi server meliputi:

- **Menyimpan dan Mengelola Data**: Server bertanggung jawab untuk menyimpan data aplikasi dalam database. Ketika klien memerlukan data, server akan mengambilnya dari database dan mengirimkannya ke klien.

- **Logika Bisnis**: Server mengeksekusi logika bisnis aplikasi. Misalnya, jika aplikasi web adalah toko online, server akan mengurus pemrosesan pesanan, pembayaran, dan manajemen inventaris.
- **Otentikasi dan Otorisasi**: Server mengelola proses otentikasi (memeriksa identitas pengguna) dan otorisasi (menentukan apa yang dapat diakses oleh pengguna).

Dalam pengembangan menggunakan Node.js dan Express, sisi server biasanya melibatkan pembuatan REST API yang berkomunikasi dengan database dan mengirimkan data dalam format JSON ke klien.

2. Client Side (Sisi Klien)

Sisi klien adalah bagian dari aplikasi yang beroperasi di perangkat pengguna, seperti komputer, tablet, atau smartphone. Fungsi utama dari sisi klien meliputi:

- **Antarmuka Pengguna (UI)**: Klien bertanggung jawab untuk menampilkan data kepada pengguna dan memungkinkan pengguna berinteraksi dengan aplikasi. Ini mencakup elemen-elemen seperti formulir, tombol, dan tampilan data.
- **Permintaan Data**: Klien mengirimkan permintaan ke server untuk mendapatkan data. Misalnya, saat pengguna membuka halaman produk di toko online, klien akan meminta data produk dari server.
- **Mengelola Respons**: Klien menerima data dari server dan menampilkannya kepada pengguna. Klien juga dapat memproses data tersebut sebelum menampilkannya.

Teknologi yang sering digunakan di sisi klien termasuk HTML, CSS, dan JavaScript. Framework dan library populer seperti React, Angular, atau Vue.js sering digunakan untuk mengembangkan sisi klien yang dinamis dan responsif.

Contoh Interaksi Client-Server

Mari kita lihat contoh sederhana untuk memahami bagaimana interaksi antara klien dan server terjadi dalam aplikasi web:

1. **Pengguna Membuka Aplikasi Web**: Pengguna membuka browser dan mengakses URL aplikasi web. Browser (klien) mengirim permintaan HTTP GET ke server.
2. **Server Mengirimkan Halaman HTML**: Server menerima permintaan dan mengirimkan file HTML, CSS, dan JavaScript ke klien.
3. **Klien Memuat Halaman Web**: Browser memuat dan menampilkan halaman web kepada pengguna.
4. **Pengguna Melakukan Aksi**: Pengguna mengklik tombol "Lihat Produk". Klien mengirim permintaan HTTP GET ke server untuk mendapatkan data produk.
5. **Server Mengirimkan Data Produk**: Server mengambil data produk dari database dan mengirimkannya dalam format JSON ke klien.

6. **Klien Menampilkan Data Produk**: Klien menerima data JSON, memprosesnya, dan menampilkan daftar produk di halaman web.

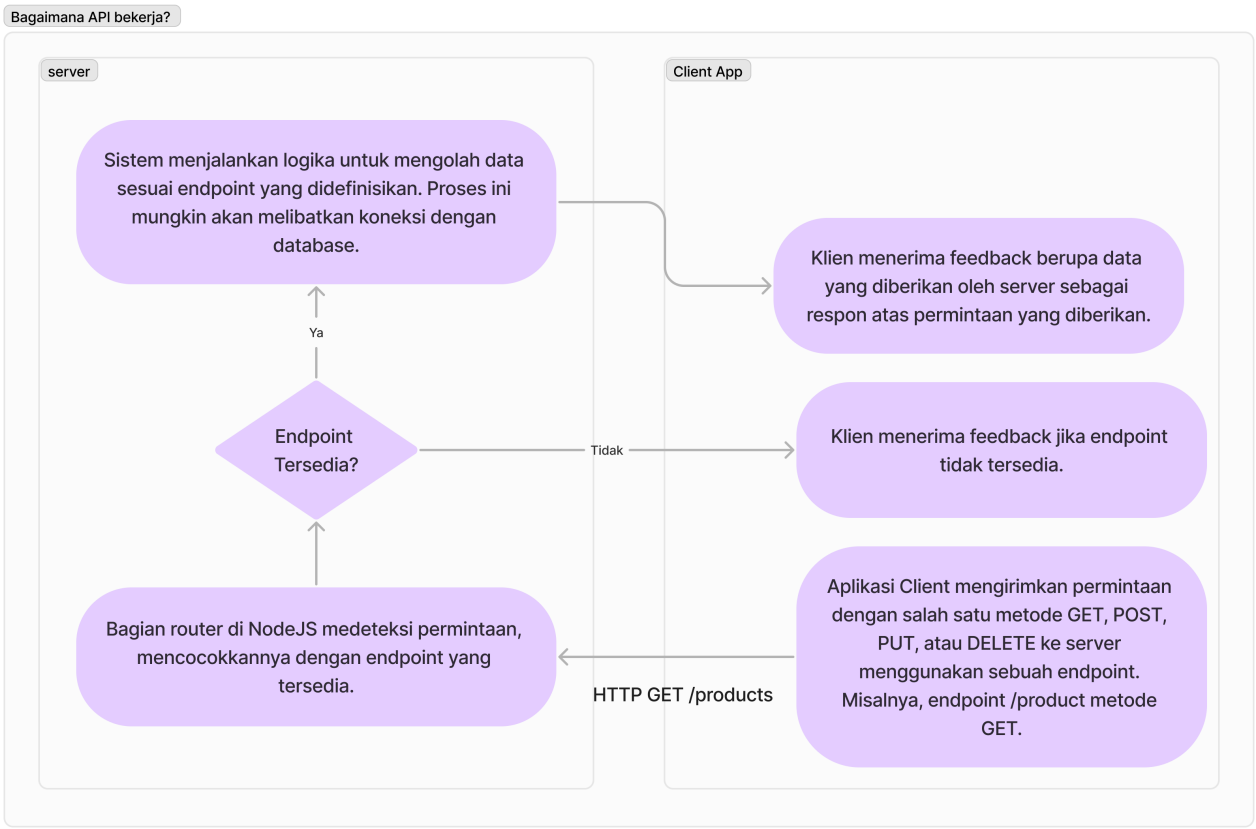
Manfaat Pola Client-Server

- **Pemeliharaan yang Mudah**: Dengan memisahkan logika aplikasi dan antarmuka pengguna, pengembangan dan pemeliharaan menjadi lebih mudah.
- **Pengembangan Paralel**: Tim pengembang dapat bekerja secara paralel di sisi server dan sisi klien, yang mempercepat proses pengembangan.
- **Skalabilitas**: Server dapat dioptimalkan untuk menangani banyak permintaan secara efisien, sementara klien dapat berjalan di berbagai perangkat dengan kemampuan berbeda.

2 - Apa Itu REST API

Apa itu REST API?

REST API (Representational State Transfer Application Programming Interface) adalah salah satu jenis API yang paling umum digunakan dalam pengembangan web. REST API memungkinkan aplikasi untuk berkomunikasi satu sama lain melalui protokol HTTP. Konsep utama dari REST adalah sumber daya (resources), yang diidentifikasi oleh URL, dan tindakan terhadap sumber daya tersebut ditentukan oleh metode HTTP (GET, POST, PUT, DELETE, dll).



Prinsip Dasar REST

1. **Client-Server Architecture**: REST API memisahkan klien dan server. Klien bertanggung jawab untuk antarmuka pengguna dan pengalaman pengguna, sementara server menangani penyimpanan data dan logika backend.

2. **Statelessness**: Setiap permintaan dari klien ke server harus berisi semua informasi yang diperlukan untuk memahami permintaan tersebut. Server tidak menyimpan informasi apapun tentang klien antara permintaan.
3. **Cacheability**: Respons dari server dapat dicache oleh klien untuk mengurangi jumlah permintaan yang dilakukan ke server, meningkatkan efisiensi dan kinerja.
4. **Layered System**: Struktur dari API REST memungkinkan penggunaan lapisan perantara seperti load balancers dan proxies untuk meningkatkan skalabilitas dan keamanan.
5. **Uniform Interface**: REST mengharuskan adanya antarmuka yang seragam dan standar untuk berinteraksi dengan sumber daya. Ini memudahkan interoperabilitas antara berbagai sistem.

Bagaimana Cara Kerja REST API?

Untuk memahami cara kerja REST API, mari kita lihat bagaimana operasi dasar dilakukan menggunakan metode HTTP:

- **GET**: Mengambil data dari server. Misalnya, `GET /users` mungkin digunakan untuk mengambil daftar semua pengguna.
- **POST**: Mengirim data baru ke server untuk disimpan. Misalnya, `POST /users` digunakan untuk membuat pengguna baru.
- **PUT**: Memperbarui data yang ada di server. Misalnya, `PUT /users/1` digunakan untuk memperbarui data pengguna dengan ID 1.
- **DELETE**: Menghapus data dari server. Misalnya, `DELETE /users/1` digunakan untuk menghapus pengguna dengan ID 1.

Setiap operasi ini dilakukan melalui permintaan HTTP yang mengarah ke URL tertentu yang mewakili sumber daya yang dimaksud.

Contoh Sederhana

Misalkan kita memiliki aplikasi yang mengelola daftar buku. REST API untuk aplikasi ini mungkin memiliki endpoint seperti berikut:

- `GET /books` - Mengambil daftar semua buku.
- `GET /books/{id}` - Mengambil detail buku dengan ID tertentu.
- `POST /books` - Menambahkan buku baru ke daftar.
- `PUT /books/{id}` - Memperbarui informasi buku dengan ID tertentu.
- `DELETE /books/{id}` - Menghapus buku dengan ID tertentu.

Setiap endpoint tersebut akan menerima dan mengirim data dalam format yang dapat dibaca oleh manusia, biasanya JSON.

Manfaat Menggunakan REST API

1. **Interoperabilitas**: REST API memungkinkan berbagai aplikasi dari platform yang berbeda untuk saling berkomunikasi.
2. **Skalabilitas**: Arsitektur REST yang sederhana dan tidak menyimpan status memudahkan untuk mengembangkan aplikasi yang skalabel.
3. **Fleksibilitas**: Klien dan server dapat dikembangkan secara terpisah dan dapat diperbarui tanpa saling mempengaruhi satu sama lain.
4. **Efisiensi**: Dengan mendukung caching, REST API dapat mengurangi beban server dan meningkatkan kecepatan respon.

3 - Memulai NodeJS dan Express

Pengantar

Node.js adalah platform berbasis JavaScript yang memungkinkan Anda untuk menjalankan kode JavaScript di sisi server. Node.js sangat populer karena kemampuannya untuk menangani banyak permintaan secara efisien dan non-blokir (non-blocking). Dengan Node.js, Anda bisa membuat aplikasi web yang cepat dan skalabel.

Express adalah framework untuk Node.js yang mempermudah dalam membangun aplikasi web dan API. Dengan Express, Anda dapat menangani routing, middleware, dan banyak lagi dengan lebih mudah dan terstruktur.

Dalam bagian ini, kita akan membahas cara memulai dengan Node.js dan Express, mulai dari instalasi hingga mencoba aplikasi sederhana.

Prerequisite

Sebelum memulai, pastikan Anda memiliki:

- Komputer dengan sistem operasi Windows, macOS, atau Linux
- Koneksi internet
- Akses terminal atau command prompt
- Pengetahuan dasar tentang JavaScript

Instalasi Node.js

1. Kunjungi situs resmi Node.js di nodejs.org.
2. Unduh installer untuk sistem operasi Anda (versi LTS direkomendasikan untuk stabilitas).
3. Jalankan installer dan ikuti instruksi pemasangan.
4. Setelah instalasi selesai, verifikasi instalasi dengan membuka terminal atau command prompt dan menjalankan perintah berikut:

```
node -v
```

```
npm -v
```

Jika instalasi berhasil, Anda akan melihat versi Node.js dan npm yang terinstal.

Instalasi Express CLI

Express CLI adalah alat baris perintah yang mempermudah pembuatan aplikasi Express. Untuk menginstalnya, gunakan npm (Node Package Manager) yang telah terpasang bersama Node.js:

1. Buka terminal atau command prompt.
2. Jalankan perintah berikut untuk menginstal Express CLI secara global:

```
npm install -g express-generator
```

Membuat Project Baru dengan Express CLI

Setelah Express CLI terinstal, Anda dapat membuat proyek baru dengan sangat mudah:

1. Buka terminal atau command prompt.
2. Arahkan ke direktori tempat Anda ingin membuat proyek baru.
3. Jalankan perintah berikut untuk membuat proyek baru (ganti `nama-proyek-anda` dengan nama proyek Anda):

```
express nama-proyek-anda
```

4. Pindah ke direktori proyek yang baru dibuat:

```
cd nama-proyek-anda
```

5. Instal dependensi proyek dengan menjalankan:

```
npm install
```

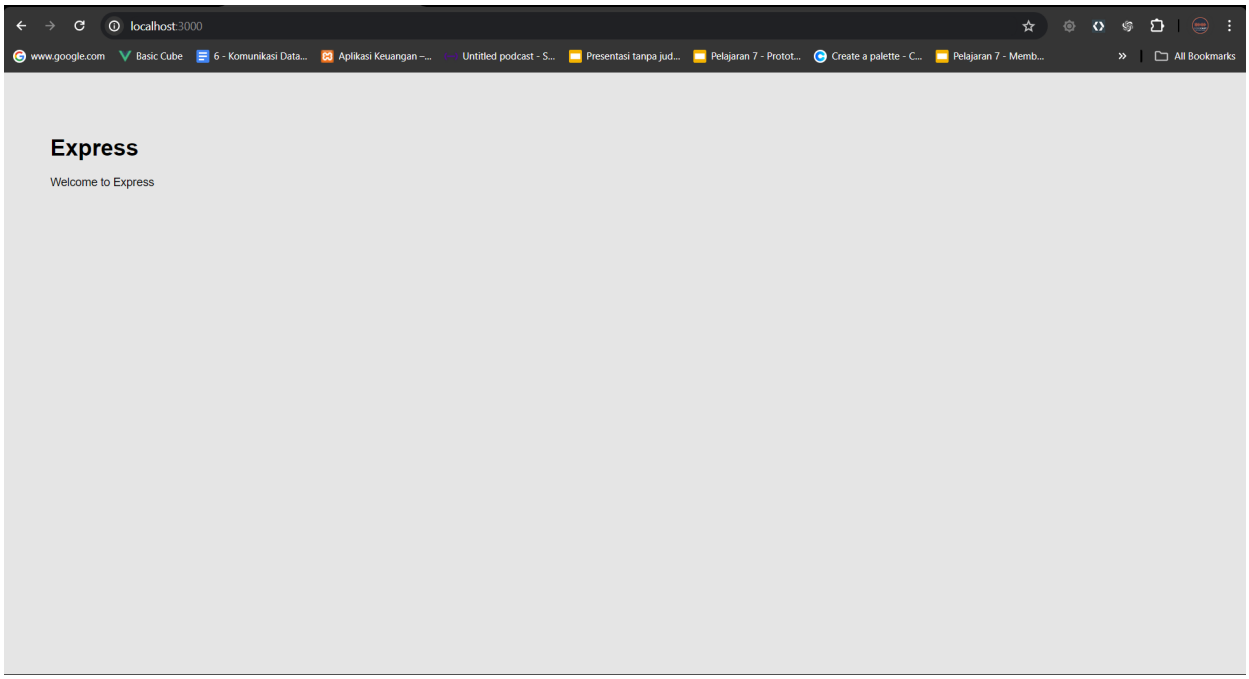
Mencoba Aplikasi

Setelah semua dependensi terinstal, Anda bisa menjalankan aplikasi dan melihat hasilnya:

1. Jalankan perintah berikut di terminal atau command prompt:

```
npm start
```

2. Buka browser web dan akses `http://localhost:3000`. Anda akan melihat halaman awal dari aplikasi Express yang baru saja Anda buat. Tampilan akan seperti pada gambar berikut:



Selamat! Anda telah berhasil memulai dengan Node.js dan Express. Pada bagian selanjutnya, kita akan membahas struktur proyek dan mulai membangun API pertama Anda.

4 - Membuat API Sederhana dengan Metode GET untuk Menampilkan Data 'Products'

Pada bagian ini, kita akan membuat API sederhana menggunakan metode GET untuk menampilkan data produk dalam format JSON. Kita akan memulai dari proyek Express yang telah kita buat sebelumnya.

Struktur Proyek

Berikut adalah struktur direktori proyek yang dibuat oleh Express CLI:

```
nama-proyek-anda/
├── app.js
├── bin/
│   └── www
├── package.json
├── public/
├── routes/
│   ├── index.js
│   └── users.js
├── views/
│   ├── error.jade
│   ├── index.jade
│   └── layout.jade
└── node_modules/
```

Kita akan menambahkan rute baru di dalam direktori `routes` untuk API `products`.

Langkah-langkah Membuat API Sederhana

1. **Tambahkan File Rute Baru untuk Products**
   Buat file baru bernama `products.js` di dalam direktori `routes`.
2. **Edit File `products.js`**
   Buka file `routes/products.js` dan tambahkan kode berikut:

```
var express = require('express');
var router = express.Router();

// Data produk yang akan kita tampilkan
const products = [
    { id: 1, name: 'Product A', price: 100 },
    { id: 2, name: 'Product B', price: 150 },
    { id: 3, name: 'Product C', price: 200 }
];

// Rute GET untuk mendapatkan daftar produk
router.get('/', function(req, res, next) {
    res.json(products);
});

module.exports = router;
```

3. **Update `app.js` untuk Menggunakan Rute `products`**
   Buka file `app.js` dan tambahkan rute `products` ke aplikasi Express Anda. Tambahkan baris berikut setelah bagian rute `users`.

```
var productsRouter = require('./routes/products');

// Tambahkan ini di bagian setelah rute users
app.use('/products', productsRouter);
```

   Secara keseluruhan, bagian rute di `app.js` akan terlihat seperti ini:

```
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var productsRouter = require('./routes/products');

app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/products', productsRouter);
```

4. **Menjalankan Server dan Mencoba API**
   Pastikan server Express berjalan. Jika belum, jalankan perintah berikut di terminal:

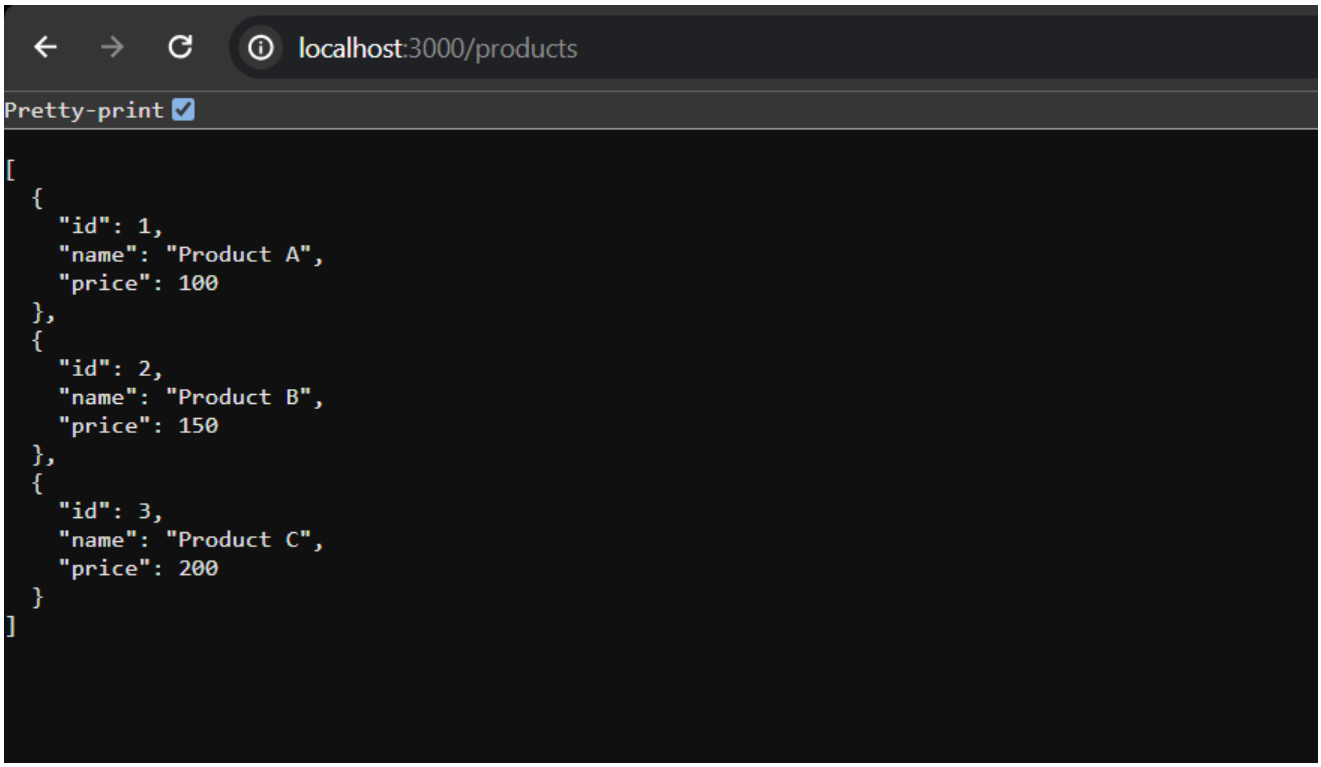
```
npm start
```

   Setelah server berjalan, buka browser atau alat API seperti Postman dan akses URL berikut:

```
http://localhost:3000/products
```

Anda akan melihat output JSON yang berisi daftar produk:

```
[
    { "id": 1, "name": "Product A", "price": 100 },
    { "id": 2, "name": "Product B", "price": 150 },
    { "id": 3, "name": "Product C", "price": 200 }
]
```



Dengan langkah-langkah di atas, Anda telah berhasil membuat API sederhana menggunakan metode GET untuk menampilkan data produk dalam format JSON. Selanjutnya, Anda bisa mengembangkan API ini dengan menambahkan fitur lain seperti POST, PUT, DELETE, dan lain-lain.

5 - Membuat API Sederhana dengan Metode POST untuk Mengirimkan Data Produk

Pada bagian ini, kita akan membuat API sederhana menggunakan metode POST untuk mengirimkan data produk dari klien ke server. Kita akan menggunakan proyek Express yang telah dibuat sebelumnya. Kita juga akan memberikan respons untuk memberitahu klien bahwa data berhasil diterima.

Langkah-langkah Membuat API Sederhana dengan Metode POST

1. **Edit File `products.js`**
   Kita akan menambahkan rute POST di file `routes/products.js` untuk menerima data produk.

   Buka file `routes/products.js` dan tambahkan kode berikut di bawah rute GET:

```
var express = require('express');
var router = express.Router();

// Data produk yang akan kita tampilkan dan simpan
let products = [
    { id: 1, name: 'Product A', price: 100 },
    { id: 2, name: 'Product B', price: 150 },
    { id: 3, name: 'Product C', price: 200 }
];

// Rute GET untuk mendapatkan daftar produk
router.get('/', function(req, res, next) {
    res.json(products);
});

// Rute POST untuk menambahkan produk baru
router.post('/', function(req, res, next) {
    const newProduct = {
        id: products.length + 1,
        name: req.body.name,
        price: req.body.price
```

```
    };
    products.push(newProduct);
    res.status(201).json({ message: 'Product added successfully', product:
newProduct });
});

module.exports = router;
```

2. **Update `app.js` untuk Menggunakan Middleware `express.json()`**
   Agar server dapat memproses data JSON yang dikirimkan oleh klien, kita perlu menambahkan middleware `express.json()` di `app.js`.

   Buka file `app.js` dan tambahkan baris berikut sebelum mendefinisikan rute:

```
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var productsRouter = require('./routes/products');

var app = express();

app.use(logger('dev'));
app.use(express.json()); // Middleware untuk memproses data JSON
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/products', productsRouter);

module.exports = app;
```

3. **Menjalankan Server dan Mencoba API dengan Postman**
   Pastikan server Express berjalan. Jika belum, jalankan perintah berikut di terminal:

```
npm start
```

Menggunakan Talend REST Client untuk Menguji POST /products

Talend REST Client adalah salah satu dari banyak alat yang dapat digunakan untuk menguji API RESTful. Anda dapat memasang aplikasi tersebut pada browser Chrome melalui Chrome webstore. Berikut adalah langkah-langkah untuk menggunakan Talend REST Client dalam menguji endpoint POST `/products`.

Langkah-langkah Menguji POST /products dengan Talend REST Client

1. **Unduh dan Instal Talend REST Client pada Web Browser**
   Jika Anda belum memiliki Talend REST Client, Anda dapat mengunduh dan menginstalnya dari situs resmi Talend atau melalui plugin di browser Anda.
2. **Buka Talend REST Client**
   Setelah terinstal, buka Talend REST Client. Anda akan melihat antarmuka untuk mengirim permintaan HTTP.
3. **Konfigurasi Permintaan POST**
   Untuk menguji endpoint POST `/products`, ikuti langkah-langkah berikut:
   - **Pilih Metode HTTP**: Pada dropdown metode HTTP, pilih `POST`.
   - **Masukkan URL**: Di bidang URL, masukkan endpoint API Anda:

     ```
     http://localhost:3000/products
     ```

   - **Konfigurasi Header**: Tambahkan header `Content-Type` dengan nilai `application/json` untuk memastikan bahwa server mengenali data yang dikirim sebagai JSON.
     - Klik tombol `Add Header`.
     - Masukkan `Content-Type` sebagai nama header.
     - Masukkan `application/json` sebagai nilai header.
   - **Masukkan Body Permintaan**: Pada tab `Body`, pilih opsi `Raw` dan pilih tipe `JSON`. Kemudian masukkan data produk yang ingin Anda kirim, misalnya:

     ```
     {
         "name": "Product E",
         "price": 300
     }
     ```

4. **Kirim Permintaan**
   Setelah semua konfigurasi di atas selesai, klik tombol `Send` untuk mengirim permintaan POST ke server.
5. **Lihat Respons**
   Setelah mengirim permintaan, Anda akan melihat respons dari server di bagian bawah atau di tab `Response`. Jika permintaan berhasil, Anda akan melihat respons seperti ini:

```
{
    "message": "Product added successfully",
    "product": {
        "id": 5,
        "name": "Product E",
        "price": 300
    }
}
```

Contoh Konfigurasi di Talend REST Client

- **Method**: POST
- **URL**: http://localhost:3000/products
- **Headers**:
  - Content-Type: application/json
- **Body**:

```
{
    "name": "Product E",
    "price": 300
}
```

Kesimpulan

Dengan mengikuti langkah-langkah di atas, Anda dapat menggunakan Talend REST Client untuk menguji endpoint POST `/products` pada server Express Anda. Talend REST Client adalah alat yang berguna untuk pengujian API karena menyediakan antarmuka yang mudah digunakan dan fitur yang kuat untuk mengkonfigurasi permintaan HTTP. Anda dapat menggunakannya untuk menguji berbagai jenis permintaan HTTP dan memastikan bahwa API Anda berfungsi dengan benar.

6 - Mengatur Koneksi Database MySQL Menggunakan Sequelize

0Dalam bagian ini, kita akan mengatur koneksi ke database MySQL menggunakan Sequelize dan memastikan koneksi tersebut berhasil dengan memberikan umpan balik melalui console.

Langkah-langkah Mengatur Koneksi Database Menggunakan Sequelize

1. **Instalasi Sequelize dan MySQL2**
   Pertama, kita perlu menginstal Sequelize dan driver MySQL2. Jalankan perintah berikut di direktori proyek Anda:

   ```
   npm install sequelize mysql2
   ```

2. **Buat File Konfigurasi Sequelize**
   Buat direktori baru bernama `models` di direktori root proyek Anda jika belum ada. Kemudian, buat file `index.js` di dalam direktori `models` untuk menyimpan pengaturan koneksi database.

   ```
   mkdir models
   touch models/index.js
   ```

3. **Edit File `models/index.js`**
   Buka file `models/index.js` dan tambahkan kode berikut untuk mengatur koneksi ke database MySQL menggunakan Sequelize:

   ```
   const { Sequelize } = require('sequelize');

   // Konfigurasi koneksi Sequelize
   const sequelize = new Sequelize('your_database_name', 'root', 'password', {
       host: 'localhost',
       dialect: 'mysql'
   ```

```
});

// Uji koneksi
sequelize.authenticate()
    .then(() => {
        console.log('Connection has been established successfully.');
    })
    .catch(err => {
        console.error('Unable to connect to the database:', err);
    });

// Ekspor instance sequelize untuk digunakan di tempat lain
module.exports = sequelize;
```

Gantilah `your_database_name`, `root`, dan `password` dengan nama database, username, dan password MySQL Anda yang sesuai.

4. **Update `app.js` untuk Menggunakan Koneksi Database**
   Pastikan aplikasi menggunakan file `models/index.js` agar koneksi database diperiksa saat aplikasi dijalankan. Buka file `app.js` dan tambahkan kode berikut di bagian atas setelah `require('express')`:

```
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var productsRouter = require('./routes/products');
var sequelize = require('./models/index'); // Tambahkan ini untuk memuat
koneksi database

var app = express();

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/products', productsRouter);

module.exports = app;
```

5. **Menjalankan Server dan Memeriksa Koneksi Database**
   Jalankan server Express. Jika belum berjalan, jalankan perintah berikut di terminal:

```
npm start
```

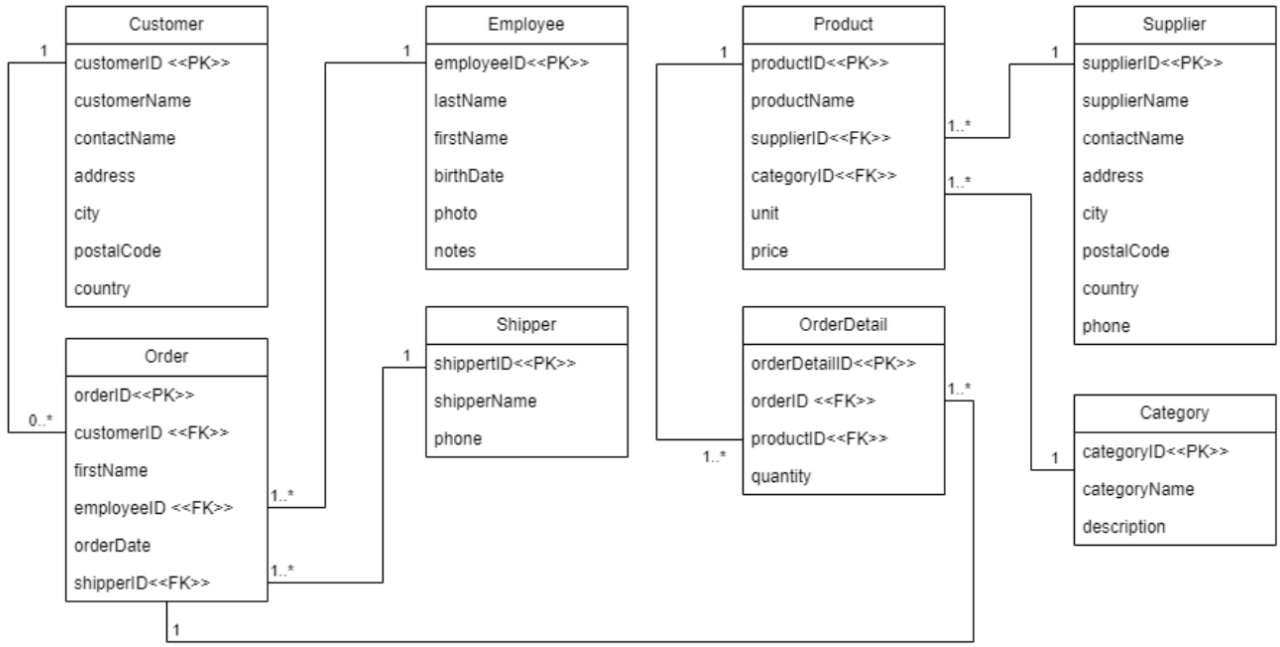
Anda akan melihat pesan di terminal yang mengindikasikan apakah koneksi ke database berhasil atau gagal.

Kesimpulan

Dengan langkah-langkah di atas, Anda telah berhasil mengatur koneksi ke database MySQL menggunakan Sequelize dan memverifikasi koneksi di terminal. Jika koneksi berhasil, Anda akan melihat pesan yang menyatakan bahwa koneksi berhasil. Jika koneksi gagal, Anda akan melihat pesan error yang menjelaskan masalah koneksi. Langkah ini memastikan bahwa aplikasi Anda dapat terhubung ke database, memungkinkan Anda untuk melanjutkan ke langkah berikutnya dalam pengembangan API Anda.

7 - Membuat Data Model untuk Product dan Category Menggunakan Sequelize

Dalam bagian ini, kita akan membuat data model untuk tabel `Product` dan `Category` berdasarkan ERD pada gambar dibawah. Kita akan menggunakan Sequelize untuk mendefinisikan model-model ini dan mengatur relasi antara keduanya.



Langkah-langkah Membuat Data Model dengan Sequelize

1. **Pastikan Sequelize Terpasang**
   Jika belum, pastikan Anda telah menginstal Sequelize dan driver MySQL2 dengan perintah berikut:

   ```
   npm install sequelize mysql2
   ```

2. **Konfigurasi Koneksi Sequelize**
   Jika belum ada, buat file `models/index.js` untuk mengatur koneksi ke database

MySQL menggunakan Sequelize. Kode berikut mengatur koneksi dan memberikan feedback di console:

```
const { Sequelize } = require('sequelize');

// Konfigurasi koneksi Sequelize
const sequelize = new Sequelize('your_database_name', 'root', 'password', {
    host: 'localhost',
    dialect: 'mysql'
});

// Uji koneksi
sequelize.authenticate()
    .then(() => {
        console.log('Connection has been established successfully.');
    })
    .catch(err => {
        console.error('Unable to connect to the database:', err);
    });

// Ekspor instance sequelize untuk digunakan di tempat lain
module.exports = sequelize;
```

3. **Definisikan Model Category**
   Buat file `category.js` di dalam direktori `models` untuk mendefinisikan model `Category`:

```
const { DataTypes } = require('sequelize');
const sequelize = require('./index');

const Category = sequelize.define('Category', {
    categoryID: {
        type: DataTypes.INTEGER,
        autoIncrement: true,
        primaryKey: true
    },
    categoryName: {
        type: DataTypes.STRING,
        allowNull: false
    },
    description: {
        type: DataTypes.STRING,
        allowNull: true
    }
}, {
    timestamps: false
});
```

```
module.exports = Category;
```

4. **Definisikan Model Product**
   Buat file `product.js` di dalam direktori `models` untuk mendefinisikan model `Product` dan mengatur relasi dengan `Category`:

```
const { DataTypes } = require('sequelize');
const sequelize = require('./index');
const Category = require('./category'); // Impor model Category

const Product = sequelize.define('Product', {
    productID: {
        type: DataTypes.INTEGER,
        autoIncrement: true,
        primaryKey: true
    },
    productName: {
        type: DataTypes.STRING,
        allowNull: false
    },
    supplierID: {
        type: DataTypes.INTEGER,
        allowNull: false
    },
    categoryID: {
        type: DataTypes.INTEGER,
        allowNull: false,
        references: {
            model: Category,
            key: 'categoryID'
        }
    },
    unit: {
        type: DataTypes.STRING,
        allowNull: true
    },
    price: {
        type: DataTypes.DECIMAL(10, 2),
        allowNull: false
    }
}, {
    timestamps: false
});

// Definisikan relasi antara Product dan Category
Product.belongsTo(Category, { foreignKey: 'categoryID' });
Category.hasMany(Product, { foreignKey: 'categoryID' });
```

```
module.exports = Product;
```

5. **Sinkronisasi Model dengan Database**
   Untuk memastikan model dan tabel terbuat di database, tambahkan sinkronisasi di `app.js` atau buat skrip sinkronisasi terpisah. Dalam contoh ini, kita akan melakukannya di `app.js`.

   Buka file `app.js` dan tambahkan sinkronisasi database:

```
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var productsRouter = require('./routes/products');
var sequelize = require('./models/index'); // Impor sequelize
var Category = require('./models/category'); // Impor model Category
var Product = require('./models/product'); // Impor model Product

var app = express();

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/products', productsRouter);

// Sinkronkan model dengan database
sequelize.sync()
    .then(() => {
        console.log('Database synchronized');
    })
    .catch(err => {
        console.error('Error synchronizing database:', err);
    });

module.exports = app;
```

Kesimpulan

Dengan langkah-langkah di atas, Anda telah berhasil mendefinisikan data model `Product` dan `Category` menggunakan Sequelize dan mengatur relasi antara keduanya. Model `Product` memiliki relasi `belongsTo` dengan `Category`, dan `Category` memiliki relasi `hasMany` dengan `Product`. Ini memastikan bahwa setiap produk dikaitkan dengan satu kategori dan setiap kategori dapat memiliki banyak produk. Selanjutnya, model-model ini dapat digunakan dalam rute API untuk melakukan operasi CRUD pada data produk dan kategori.

8 - Membuat Endpoints untuk Mengelola Product Category

Dalam bagian ini, kita akan membuat endpoints untuk menambahkan, menampilkan, memperbarui, dan menghapus kategori produk. Kita akan menambahkan rute-rute ini ke dalam proyek Express kita.

Struktur File Proyek yang Dimodifikasi

Berikut adalah struktur file proyek setelah menambahkan pengaturan koneksi database, model, dan rute untuk kategori produk:

```
project-root/
├── app.js
├── package.json
├── models/
│   ├── index.js
│   ├── category.js
│   └── product.js
├── routes/
│   ├── index.js
│   ├── users.js
│   ├── products.js
│   └── categories.js
├── public/
│   └── ... (static files)
├── views/
│   └── ... (view templates)
└── ... (other files and directories)
```

Langkah-langkah Membuat Endpoints untuk Kategori Produk

1. **Buat Rute Categories**
   Buat file `categories.js` di dalam direktori `routes` untuk mendefinisikan endpoints

CRUD untuk kategori produk.

```
touch routes/categories.js
```

2. **Edit File** `routes/categories.js`
   Tambahkan kode berikut untuk mendefinisikan endpoints untuk menambahkan, menampilkan, memperbarui, dan menghapus kategori:

```
const express = require('express');
const router = express.Router();
const Category = require('../models/category'); // Impor model Category

// Endpoint untuk menambahkan kategori baru
router.post('/', async (req, res, next) => {
    try {
        const { categoryName, description } = req.body;
        const newCategory = await Category.create({ categoryName,
description });
        res.status(201).json(newCategory);
    } catch (err) {
        next(err);
    }
});

// Endpoint untuk menampilkan semua kategori
router.get('/', async (req, res, next) => {
    try {
        const categories = await Category.findAll();
        res.json(categories);
    } catch (err) {
        next(err);
    }
});

// Endpoint untuk menampilkan kategori berdasarkan ID
router.get('/:id', async (req, res, next) => {
    try {
        const category = await Category.findByPk(req.params.id);
        if (category) {
            res.json(category);
        } else {
            res.status(404).json({ message: 'Category not found' });
        }
    } catch (err) {
        next(err);
    }
});
```

```
// Endpoint untuk memperbarui kategori berdasarkan ID
router.put('/:id', async (req, res, next) => {
    try {
        const { categoryName, description } = req.body;
        const category = await Category.findByPk(req.params.id);
        if (category) {
            category.categoryName = categoryName;
            category.description = description;
            await category.save();
            res.json(category);
        } else {
            res.status(404).json({ message: 'Category not found' });
        }
    } catch (err) {
        next(err);
    }
});

// Endpoint untuk menghapus kategori berdasarkan ID
router.delete('/:id', async (req, res, next) => {
    try {
        const category = await Category.findByPk(req.params.id);
        if (category) {
            await category.destroy();
            res.json({ message: 'Category deleted' });
        } else {
            res.status(404).json({ message: 'Category not found' });
        }
    } catch (err) {
        next(err);
    }
});

module.exports = router;
```

3. **Tambahkan Rute Categories ke `app.js`**
   Buka file `app.js` dan tambahkan rute untuk kategori produk:

```
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var productsRouter = require('./routes/products');
var categoriesRouter = require('./routes/categories'); // Impor rute
categories
var sequelize = require('./models/index'); // Impor sequelize
```

```
var app = express();

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/products', productsRouter);
app.use('/categories', categoriesRouter); // Gunakan rute categories

// Sinkronkan model dengan database
sequelize.sync()
    .then(() => {
        console.log('Database synchronized');
    })
    .catch(err => {
        console.error('Error synchronizing database:', err);
    });

module.exports = app;
```

Kesimpulan

Dengan langkah-langkah di atas, Anda telah berhasil membuat endpoints untuk mengelola kategori produk dalam aplikasi Express Anda. Anda sekarang memiliki endpoints untuk menambahkan, menampilkan, memperbarui, dan menghapus kategori produk. Struktur file proyek Anda juga telah diatur dengan baik untuk mendukung pengembangan lebih lanjut. Selanjutnya, Anda dapat menguji endpoints ini menggunakan alat seperti Postman atau Talend REST Client.

9 - Mengelola Product (CRUD) Menggunakan Sequelize

Pada bagian ini, kita akan mengatur rute `products` untuk mendukung operasi CRUD menggunakan model `Product` yang telah dibuat sebelumnya. Kita akan membuat rute untuk menambahkan, menampilkan, memperbarui, dan menghapus produk.

Langkah-langkah Membuat Rute CRUD untuk Produk

1. **Buat dan Edit File** `routes/products.js`
   Kita akan mengganti konten rute `products` untuk mendukung operasi CRUD. Jika file ini belum ada, buat file tersebut di direktori `routes`.

   ```
   touch routes/products.js
   ```

   Tambahkan kode berikut untuk mendefinisikan rute CRUD:

   ```
   const express = require('express');
   const router = express.Router();
   const Product = require('../models/product'); // Impor model Product

   // Endpoint untuk menambahkan produk baru
   router.post('/', async (req, res, next) => {
       try {
           const { productName, supplierID, categoryID, unit, price } =
   req.body;
           const newProduct = await Product.create({ productName, supplierID,
   categoryID, unit, price });
           res.status(201).json(newProduct);
       } catch (err) {
           next(err);
       }
   });

   // Endpoint untuk menampilkan semua produk
   router.get('/', async (req, res, next) => {
   ```

```
    try {
        const products = await Product.findAll();
        res.json(products);
    } catch (err) {
        next(err);
    }
});

// Endpoint untuk menampilkan produk berdasarkan ID
router.get('/:id', async (req, res, next) => {
    try {
        const product = await Product.findByPk(req.params.id);
        if (product) {
            res.json(product);
        } else {
            res.status(404).json({ message: 'Product not found' });
        }
    } catch (err) {
        next(err);
    }
});

// Endpoint untuk memperbarui produk berdasarkan ID
router.put('/:id', async (req, res, next) => {
    try {
        const { productName, supplierID, categoryID, unit, price } =
req.body;
        const product = await Product.findByPk(req.params.id);
        if (product) {
            product.productName = productName;
            product.supplierID = supplierID;
            product.categoryID = categoryID;
            product.unit = unit;
            product.price = price;
            await product.save();
            res.json(product);
        } else {
            res.status(404).json({ message: 'Product not found' });
        }
    } catch (err) {
        next(err);
    }
});

// Endpoint untuk menghapus produk berdasarkan ID
router.delete('/:id', async (req, res, next) => {
    try {
        const product = await Product.findByPk(req.params.id);
        if (product) {
```

```
            await product.destroy();
            res.json({ message: 'Product deleted' });
        } else {
            res.status(404).json({ message: 'Product not found' });
        }
    } catch (err) {
        next(err);
    }
});

module.exports = router;
```

2. **Tambahkan Rute Products ke `app.js`**
   Buka file `app.js` dan pastikan rute untuk produk diatur dengan benar:

```
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var productsRouter = require('./routes/products'); // Impor rute products
var categoriesRouter = require('./routes/categories'); // Impor rute
categories
var sequelize = require('./models/index'); // Impor sequelize

var app = express();

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/products', productsRouter); // Gunakan rute products
app.use('/categories', categoriesRouter); // Gunakan rute categories

// Sinkronkan model dengan database
sequelize.sync()
    .then(() => {
        console.log('Database synchronized');
    })
    .catch(err => {
        console.error('Error synchronizing database:', err);
    });
```

```
module.exports = app;
```

Struktur File Akhir

```
project-root/
├── app.js
├── package.json
├── models/
│   ├── index.js
│   ├── category.js
│   └── product.js
├── routes/
│   ├── index.js
│   ├── users.js
│   ├── products.js
│   └── categories.js
├── public/
│   └── ... (static files)
├── views/
│   └── ... (view templates)
└── ... (other files and directories)
```

Selanjutnya, Anda dapat menguji endpoints ini menggunakan alat seperti Postman atau Talend REST Client.

Kesimpulan

Dengan langkah-langkah di atas, Anda telah berhasil mengatur rute CRUD untuk mengelola produk menggunakan Sequelize dalam aplikasi Express Anda. Anda sekarang memiliki endpoints untuk menambahkan, menampilkan, memperbarui, dan menghapus produk. Struktur file proyek Anda telah diatur dengan baik untuk mendukung pengembangan lebih lanjut.

10 - Penerapan Autentikasi dan Otorisasi

Pengantar Autentikasi dan Otorisasi

Autentikasi adalah proses verifikasi identitas pengguna, sedangkan otorisasi adalah proses menentukan apakah pengguna yang terautentikasi memiliki izin untuk mengakses sumber daya tertentu. Dalam aplikasi web, penerapan autentikasi dan otorisasi penting untuk melindungi data dan memastikan bahwa hanya pengguna yang berwenang yang dapat melakukan operasi tertentu.

Prasyarat

- Instalasi library yang diperlukan:

```
npm install jsonwebtoken bcryptjs
```

- Struktur pengguna dalam database.

Membuat Model Pengguna (User)

Buat model `User` untuk menyimpan informasi pengguna dan kata sandi terenkripsi.

1. **Definisi Model User**

```
// models/user.js
const { DataTypes } = require('sequelize');
const sequelize = require('./index');
const bcrypt = require('bcryptjs');

const User = sequelize.define('User', {
    username: {
        type: DataTypes.STRING,
        allowNull: false,
        unique: true
    },
    password: {
        type: DataTypes.STRING,
```

```
        allowNull: false
    },
    role: {
        type: DataTypes.ENUM('admin', 'user'),
        allowNull: false
    }
}, {
    hooks: {
        beforeCreate: async (user) => {
            const salt = await bcrypt.genSalt(10);
            user.password = await bcrypt.hash(user.password, salt);
        }
    }
});

module.exports = User;
```

Membuat Rute Pendaftaran dan Login

2. **Rute Pendaftaran**

```
// routes/auth.js
const express = require('express');
const router = express.Router();
const User = require('../models/user');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

// Rute pendaftaran
router.post('/register', async (req, res, next) => {
    try {
        const { username, password, role } = req.body;
        const newUser = await User.create({ username, password, role });
        res.status(201).json({ message: 'User registered successfully' });
    } catch (err) {
        next(err);
    }
});

// Rute login
router.post('/login', async (req, res, next) => {
    try {
        const { username, password } = req.body;
        const user = await User.findOne({ where: { username } });
        if (!user) {
            return res.status(401).json({ message: 'Invalid credentials' });
        }
        const isMatch = await bcrypt.compare(password, user.password);
```

```
        if (!isMatch) {
            return res.status(401).json({ message: 'Invalid credentials' });
        }
        const token = jwt.sign({ id: user.id, role: user.role },
'your_jwt_secret', { expiresIn: '1h' });
        res.json({ token });
    } catch (err) {
        next(err);
    }
});

module.exports = router;
```

3. **Tambahkan Rute Autentikasi ke `app.js`**

```
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var productsRouter = require('./routes/products');
var categoriesRouter = require('./routes/categories');
var authRouter = require('./routes/auth');
var sequelize = require('./models/index');

var app = express();

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/products', productsRouter);
app.use('/categories', categoriesRouter);
app.use('/auth', authRouter);

sequelize.sync()
    .then(() => {
        console.log('Database synchronized');
    })
    .catch(err => {
        console.error('Error synchronizing database:', err);
    });
```

```
module.exports = app;
```

Menerapkan Middleware Autentikasi

4. **Middleware untuk Memverifikasi Token JWT**

```
// middleware/auth.js
const jwt = require('jsonwebtoken');

const authenticate = (req, res, next) => {
    const token = req.header('Authorization').replace('Bearer ', '');
    if (!token) {
        return res.status(401).json({ message: 'No token, authorization
denied' });
    }
    try {
        const decoded = jwt.verify(token, 'your_jwt_secret');
        req.user = decoded;
        next();
    } catch (err) {
        res.status(401).json({ message: 'Token is not valid' });
    }
};

const authorize = (roles = []) => {
    return (req, res, next) => {
        if (!roles.includes(req.user.role)) {
            return res.status(403).json({ message: 'Forbidden' });
        }
        next();
    };
};

module.exports = { authenticate, authorize };
```

5. **Gunakan Middleware dalam Rute yang Ada**

```
// routes/products.js
const express = require('express');
const router = express.Router();
const Product = require('../models/product');
const { authenticate, authorize } = require('../middleware/auth');

// Endpoint untuk menambahkan produk baru
router.post('/', authenticate, authorize(['admin']), async (req, res, next)
=> {
```

```
    try {
        const { productName, supplierID, categoryID, unit, price } =
req.body;
        const newProduct = await Product.create({ productName, supplierID,
categoryID, unit, price });
        res.status(201).json(newProduct);
    } catch (err) {
        next(err);
    }
});

// Endpoint untuk menampilkan semua produk
router.get('/', authenticate, async (req, res, next) => {
    try {
        const products = await Product.findAll();
        res.json(products);
    } catch (err) {
        next(err);
    }
});

// Endpoint untuk menampilkan produk berdasarkan ID
router.get('/:id', authenticate, async (req, res, next) => {
    try {
        const product = await Product.findByPk(req.params.id);
        if (product) {
            res.json(product);
        } else {
            res.status(404).json({ message: 'Product not found' });
        }
    } catch (err) {
        next(err);
    }
});

// Endpoint untuk memperbarui produk berdasarkan ID
router.put('/:id', authenticate, authorize(['admin']), async (req, res,
next) => {
    try {
        const { productName, supplierID, categoryID, unit, price } =
req.body;
        const product = await Product.findByPk(req.params.id);
        if (product) {
            product.productName = productName;
            product.supplierID = supplierID;
            product.categoryID = categoryID;
            product.unit = unit;
            product.price = price;
            await product.save();
```

```
            res.json(product);
        } else {
            res.status(404).json({ message: 'Product not found' });
        }
    } catch (err) {
        next(err);
    }
});

// Endpoint untuk menghapus produk berdasarkan ID
router.delete('/:id', authenticate, authorize(['admin']), async (req, res,
next) => {
    try {
        const product = await Product.findByPk(req.params.id);
        if (product) {
            await product.destroy();
            res.json({ message: 'Product deleted' });
        } else {
            res.status(404).json({ message: 'Product not found' });
        }
    } catch (err) {
        next(err);
    }
});

module.exports = router;
```

Pengujian

6. **Menguji Pendaftaran dan Login Pengguna**
   - Gunakan Postman atau Talend REST Client untuk mengirim permintaan POST ke `/auth/register` dengan data pengguna baru.
   - Kirim permintaan POST ke `/auth/login` dengan kredensial pengguna untuk mendapatkan token JWT.
7. **Menguji Akses ke Rute yang Dilindungi**
   - Kirim permintaan dengan header `Authorization: Bearer <token>` ke rute yang dilindungi seperti `/products` untuk memastikan hanya pengguna yang terautentikasi dan berperan sesuai yang dapat mengakses.

Struktur File Akhir

```
project-root/
├── app.js
├── package.json
├── models/
```

```
│   ├── index.js
│   ├── category.js
│   ├── product.js
│   └── user.js
├── routes/
│   ├── index.js
│   ├── users.js
│   ├── products.js
│   ├── categories.js
│   └── auth.js
├── middleware/
│   └── auth.js
├── public/
│   └── ... (static files)
├── views/
│   └── ... (view templates)
└── ... (other files and directories)
```

Dengan demikian, Anda telah berhasil menambahkan autentikasi dan otorisasi ke aplikasi Express Anda, memastikan bahwa hanya pengguna yang terdaftar yang dapat mengelola data sesuai dengan peran mereka.

11 - Upload Gambar untuk Profil Pengguna

Pada bagian ini, kita akan menambahkan fitur untuk mengunggah gambar profil pengguna. Kita akan menggunakan `multer`, middleware untuk menangani multipart/form-data, yang digunakan untuk mengunggah file.

Langkah-langkah untuk Menambahkan Fitur Upload Gambar

1. **Instalasi Multer**

   Pertama, instal `multer` dengan perintah berikut:

   ```
   npm install multer
   ```

2. **Pengaturan Multer**

   Buat file konfigurasi `multer` di direktori `middleware` untuk mengatur penyimpanan dan filter file yang diunggah.

   ```
   // middleware/upload.js
   const multer = require('multer');
   const path = require('path');

   // Konfigurasi penyimpanan
   const storage = multer.diskStorage({
       destination: function (req, file, cb) {
           cb(null, 'uploads/');
       },
       filename: function (req, file, cb) {
           cb(null, `${Date.now()}-${file.originalname}`);
       }
   });

   // Filter file yang diizinkan
   const fileFilter = (req, file, cb) => {
       const filetypes = /jpeg|jpg|png/;
       const mimetype = filetypes.test(file.mimetype);
   ```

```
    const extname =
filetypes.test(path.extname(file.originalname).toLowerCase());

    if (mimetype && extname) {
        return cb(null, true);
    } else {
        cb('Error: Images Only!');
    }
};

const upload = multer({
    storage: storage,
    limits: { fileSize: 1024 * 1024 * 5 }, // Batasan ukuran file 5MB
    fileFilter: fileFilter
});

module.exports = upload;
```

3. **Menambahkan Rute untuk Mengunggah Gambar Profil**

   Buat atau tambahkan kode ke file `routes/users.js` untuk mendukung pengunggahan gambar profil.

```
// routes/users.js
const express = require('express');
const router = express.Router();
const User = require('../models/user');
const upload = require('../middleware/upload');
const { authenticate } = require('../middleware/auth');

// Endpoint untuk mengunggah gambar profil
router.post('/uploadProfilePic', authenticate, upload.single('profilePic'),
async (req, res, next) => {
    try {
        const user = await User.findByPk(req.user.id);
        if (!user) {
            return res.status(404).json({ message: 'User not found' });
        }
        user.profilePic = req.file.path; // Simpan path gambar ke database
        await user.save();
        res.json({ message: 'Profile picture uploaded successfully',
filePath: req.file.path });
    } catch (err) {
        next(err);
    }
});
```

```
module.exports = router;
```

4. **Modifikasi Model User**

   Tambahkan kolom `profilePic` pada model `User` untuk menyimpan path gambar profil.

```
// models/user.js
const { DataTypes } = require('sequelize');
const sequelize = require('./index');
const bcrypt = require('bcryptjs');

const User = sequelize.define('User', {
    username: {
        type: DataTypes.STRING,
        allowNull: false,
        unique: true
    },
    password: {
        type: DataTypes.STRING,
        allowNull: false
    },
    role: {
        type: DataTypes.ENUM('admin', 'user'),
        allowNull: false
    },
    profilePic: {
        type: DataTypes.STRING
    }
}, {
    hooks: {
        beforeCreate: async (user) => {
            const salt = await bcrypt.genSalt(10);
            user.password = await bcrypt.hash(user.password, salt);
        }
    }
});

module.exports = User;
```

5. **Tambahkan Middleware untuk Menyajikan File Statis**

   Tambahkan middleware untuk menyajikan file statis yang diunggah pada `app.js`.

```
// app.js
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
```

```
var logger = require('morgan');
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var productsRouter = require('./routes/products');
var categoriesRouter = require('./routes/categories');
var authRouter = require('./routes/auth');
var sequelize = require('./models/index');

var app = express();

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));
app.use('/uploads', express.static('uploads')); // Middleware untuk
menyajikan file statis

app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/products', productsRouter);
app.use('/categories', categoriesRouter);
app.use('/auth', authRouter);

sequelize.sync()
    .then(() => {
        console.log('Database synchronized');
    })
    .catch(err => {
        console.error('Error synchronizing database:', err);
    });

module.exports = app;
```

6. **Pengujian**
   - Gunakan Postman atau Talend REST Client untuk menguji pengunggahan gambar profil.
   - Kirim permintaan POST ke `/users/uploadProfilePic` dengan header `Authorization: Bearer <token>` dan form-data berisi file dengan key `profilePic`.

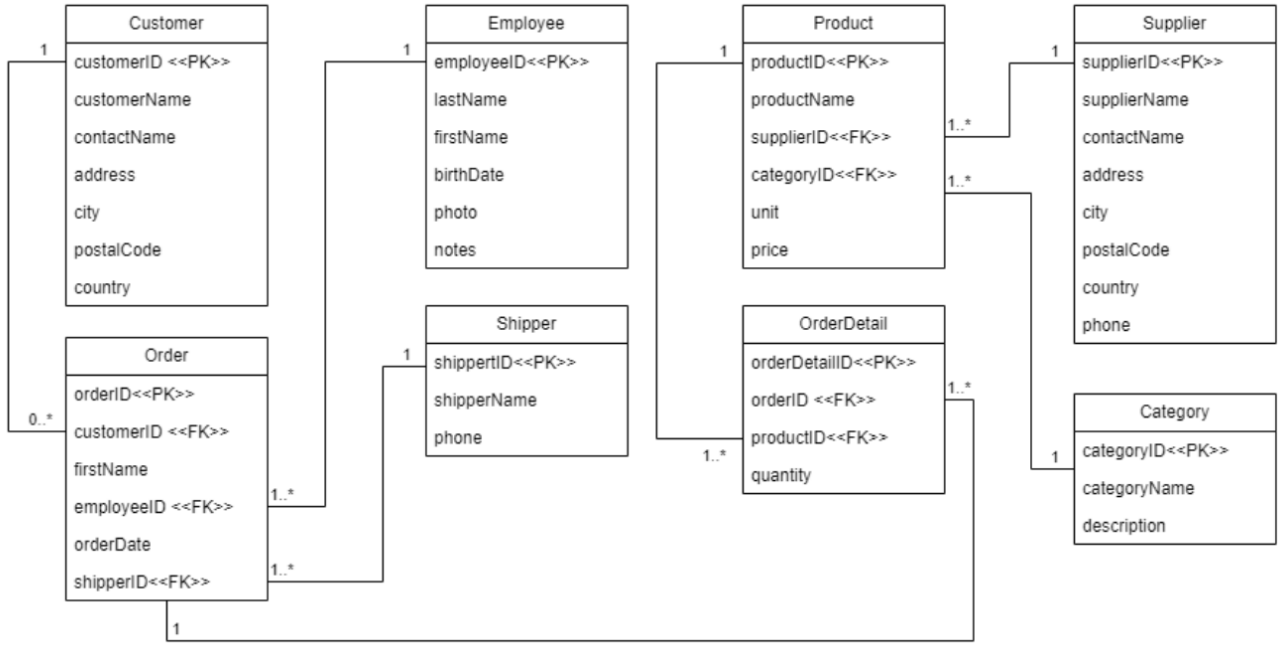
Struktur File Akhir

```
project-root/
├── app.js
├── package.json
├── models/
│   ├── index.js
```

```
│   ├── category.js
│   ├── product.js
│   └── user.js
├── routes/
│   ├── index.js
│   ├── users.js
│   ├── products.js
│   ├── categories.js
│   └── auth.js
├── middleware/
│   ├── auth.js
│   └── upload.js
├── public/
│   └── ... (static files)
├── uploads/
│   └── ... (uploaded files)
├── views/
│   └── ... (view templates)
└── ... (other files and directories)
```

Dengan langkah-langkah di atas, Anda telah berhasil menambahkan fitur untuk mengunggah gambar profil pengguna dalam aplikasi Express Anda. Anda dapat melanjutkan dengan mengembangkan fitur lain atau melakukan pengujian lebih lanjut sesuai kebutuhan.

12 - Membuat Data Model Lengkap Berdasdarkan ERD



Model ERD tersebut terdiri dari beberapa entitas dengan relasi antara satu sama lain, termasuk entitas `Customer`, `Employee`, `Product`, `Supplier`, `Order`, `Shipper`, `OrderDetail`, dan `Category`. Berikut adalah beberapa detail dari ERD tersebut:

- **Customer**: Menyimpan informasi pelanggan.
- **Employee**: Menyimpan informasi karyawan.
- **Product**: Menyimpan informasi produk.
- **Supplier**: Menyimpan informasi pemasok.
- **Order**: Menyimpan informasi pesanan.
- **Shipper**: Menyimpan informasi pengirim.
- **OrderDetail**: Menyimpan informasi detail pesanan.
- **Category**: Menyimpan informasi kategori produk.

Relasi antara entitas tersebut mencerminkan hubungan seperti:

- `Customer` memiliki banyak `Order`.
- `Order` memiliki banyak `OrderDetail`.
- `Product` berada di dalam `OrderDetail`.
- `Supplier` menyediakan `Product`.
- `Category` mengelompokkan `Product`.
- `Shipper` mengirimkan `Order`.

Mari kita lanjutkan dengan membuat data model untuk beberapa entitas ini di Sequelize sesuai dengan struktur ERD yang Anda berikan.

Data Model dengan Sequelize

Model Customer

```
// models/customer.js
const { DataTypes } = require('sequelize');
const sequelize = require('./index');

const Customer = sequelize.define('Customer', {
    customerID: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    customerName: {
        type: DataTypes.STRING,
        allowNull: false
    },
    contactName: {
        type: DataTypes.STRING,
        allowNull: false
    },
    address: {
        type: DataTypes.STRING,
        allowNull: false
    },
    city: {
        type: DataTypes.STRING,
        allowNull: false
    },
    postalCode: {
        type: DataTypes.STRING,
        allowNull: false
    },
    country: {
        type: DataTypes.STRING,
        allowNull: false
```

```
    }
});

module.exports = Customer;
```

Model Employee

```
// models/employee.js
const { DataTypes } = require('sequelize');
const sequelize = require('./index');

const Employee = sequelize.define('Employee', {
    employeeID: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    lastName: {
        type: DataTypes.STRING,
        allowNull: false
    },
    firstName: {
        type: DataTypes.STRING,
        allowNull: false
    },
    birthDate: {
        type: DataTypes.DATE,
        allowNull: false
    },
    photo: {
        type: DataTypes.STRING
    },
    notes: {
        type: DataTypes.TEXT
    }
});

module.exports = Employee;
```

Model Product

```
// models/product.js
const { DataTypes } = require('sequelize');
const sequelize = require('./index');

const Product = sequelize.define('Product', {
    productID: {
```

```
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    productName: {
        type: DataTypes.STRING,
        allowNull: false
    },
    supplierID: {
        type: DataTypes.INTEGER,
        allowNull: false
    },
    categoryID: {
        type: DataTypes.INTEGER,
        allowNull: false
    },
    unit: {
        type: DataTypes.STRING,
        allowNull: false
    },
    price: {
        type: DataTypes.DECIMAL,
        allowNull: false
    }
});

module.exports = Product;
```

Model Supplier

```
// models/supplier.js
const { DataTypes } = require('sequelize');
const sequelize = require('./index');

const Supplier = sequelize.define('Supplier', {
    supplierID: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    supplierName: {
        type: DataTypes.STRING,
        allowNull: false
    },
    contactName: {
        type: DataTypes.STRING,
        allowNull: false
    },
```

```
    address: {
        type: DataTypes.STRING,
        allowNull: false
    },
    city: {
        type: DataTypes.STRING,
        allowNull: false
    },
    postalCode: {
        type: DataTypes.STRING,
        allowNull: false
    },
    country: {
        type: DataTypes.STRING,
        allowNull: false
    },
    phone: {
        type: DataTypes.STRING,
        allowNull: false
    }
});

module.exports = Supplier;
```

Model Order

```
// models/order.js
const { DataTypes } = require('sequelize');
const sequelize = require('./index');

const Order = sequelize.define('Order', {
    orderID: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    customerID: {
        type: DataTypes.INTEGER,
        allowNull: false
    },
    firstName: {
        type: DataTypes.STRING,
        allowNull: false
    },
    employeeID: {
        type: DataTypes.INTEGER,
        allowNull: false
    },
```

```
    orderDate: {
        type: DataTypes.DATE,
        allowNull: false
    },
    shipperID: {
        type: DataTypes.INTEGER,
        allowNull: false
    }
});

module.exports = Order;
```

Model Shipper

```
// models/shipper.js
const { DataTypes } = require('sequelize');
const sequelize = require('./index');

const Shipper = sequelize.define('Shipper', {
    shipperID: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    shipperName: {
        type: DataTypes.STRING,
        allowNull: false
    },
    phone: {
        type: DataTypes.STRING,
        allowNull: false
    }
});

module.exports = Shipper;
```

Model OrderDetail

```
// models/orderDetail.js
const { DataTypes } = require('sequelize');
const sequelize = require('./index');

const OrderDetail = sequelize.define('OrderDetail', {
    orderDetailID: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
```

```
    },
    orderID: {
        type: DataTypes.INTEGER,
        allowNull: false
    },
    productID: {
        type: DataTypes.INTEGER,
        allowNull: false
    },
    quantity: {
        type: DataTypes.INTEGER,
        allowNull: false
    }
});

module.exports = OrderDetail;
```

Model Category

```
// models/category.js
const { DataTypes } = require('sequelize');
const sequelize = require('./index');

const Category = sequelize.define('Category', {
    categoryID: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    categoryName: {
        type: DataTypes.STRING,
        allowNull: false
    },
    description: {
        type: DataTypes.TEXT
    }
});

module.exports = Category;
```

Menambahkan Relasi antara Model

Setelah mendefinisikan semua model, tambahkan relasi di file `index.js`.

```
// models/index.js
const { Sequelize } = require('sequelize');
const sequelize = new Sequelize('database', 'username', 'password', {
```

```
    host: 'localhost',
    dialect: 'mysql'
});

const Customer = require('./customer')(sequelize, Sequelize.DataTypes);
const Employee = require('./employee')(sequelize, Sequelize.DataTypes);
const Product = require('./product')(sequelize, Sequelize.DataTypes);
const Supplier = require('./supplier')(sequelize, Sequelize.DataTypes);
const Order = require('./order')(sequelize, Sequelize.DataTypes);
const Shipper = require('./shipper')(sequelize, Sequelize.DataTypes);
const OrderDetail = require('./orderDetail')(sequelize, Sequelize.DataTypes);
const Category = require('./category')(sequelize, Sequelize.DataTypes);

// Relasi antara model
Customer.hasMany(Order, { foreignKey: 'customerID' });
Order.belongsTo(Customer, { foreignKey: 'customerID' });

Employee.hasMany(Order, { foreignKey: 'employeeID' });
Order.belongsTo(Employee, { foreignKey: 'employeeID' });

Shipper.hasMany(Order, { foreignKey: 'shipperID' });
Order.belongsTo(Shipper, { foreignKey: 'shipperID' });

Supplier.hasMany(Product, { foreignKey: 'supplierID' });
Product.belongsTo(Supplier, { foreignKey: 'supplierID' });

Category.hasMany(Product, { foreignKey: 'categoryID' });
Product.belongsTo(Category, { foreignKey: 'categoryID' });

Order.hasMany(OrderDetail, { foreignKey: 'orderID' });
OrderDetail.belongsTo(Order, { foreignKey: 'orderID' });

Product.hasMany(OrderDetail, { foreignKey: 'productID' });
OrderDetail.belongsTo(Product, { foreignKey: 'productID' });

module.exports = sequelize;
```

Dengan demikian, model ERD yang diberikan sudah tercakup dalam model Sequelize dan relasinya. Anda dapat melanjutkan dengan implementasi CRUD untuk setiap entitas sesuai dengan kebutuhan aplikasi Anda.