

Express.JS

Express.JS Concept
Installation and
implementation
Routing and view engines



Express JS

- ☐ Pengenalan Express JS
- ☐ Setup Environment Express JS
- ☐ Konsep Routing dan implementasinya

Objektif sesi

- Siswa memahami tentang Express JS
- Siswa memahami cara setup environment untuk menggunakan Express JS
- Siswa memahami tentang konsep routing Express JS dan implementasinya

Rancangan Subtopik

Di bawah ini merupakan list subtopik yang sudah dirancang. Namun masih sangat bisa dirubah dan disesuaikan dengan kebutuhan

- Pengenalan Express JS
- Setup Environment Express JS
- Konsep Routing dan Implementasinya

Express JS



Pengenalan Express JS



Setup Environment Express JS



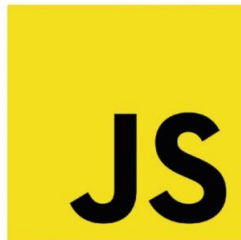
Konsep Routing dan implementasinya

Pengenalan Express JS

[Express.js](#) adalah framework web app untuk Node.js yang ditulis dengan bahasa pemrograman JavaScript. Framework open source ini dibuat oleh TJ Holowaychuk pada tahun 2010 lalu.

Express.js adalah framework back end. Artinya, ia bertanggung jawab untuk mengatur fungsionalitas website, seperti pengelolaan routing dan session, permintaan HTTP, penanganan error, serta pertukaran data di server.

Express



Kelebihan VS Kekurangan

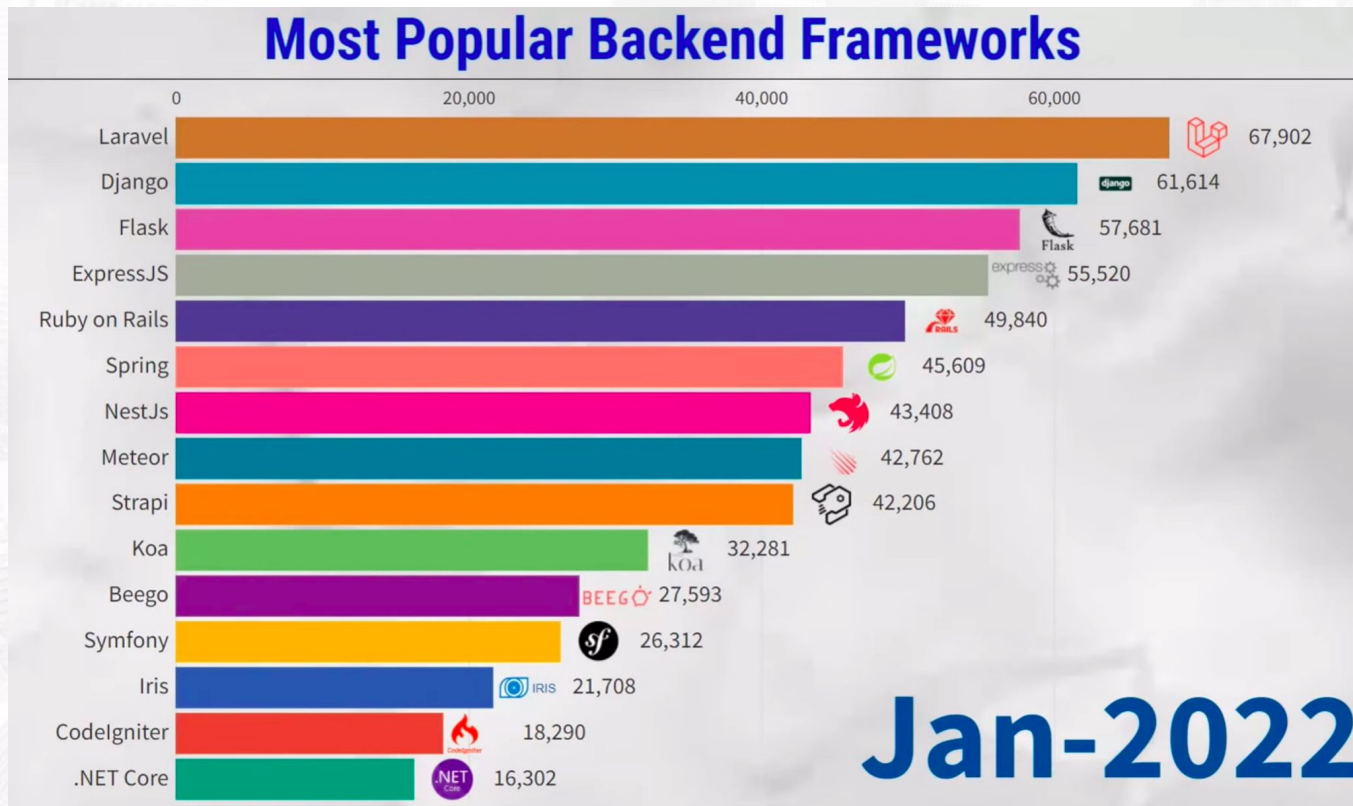
Kelebihan

- Bebas menentukan sendiri arsitektur dan struktur website yang akan dikembangkan
- Ukuran framework lebih kecil dan ringan
- Kinerja website yang dihasilkan jadi lebih baik karena ringannya framework

Kekurangan

- Terlalu banyak pilihan plugin dan library
- Pilihan metode pengembangan yang berbeda-beda

Express JS VS Others



Express JS



Pengenalan Express JS



Setup Environment Express JS



Konsep Routing dan implementasinya

Setup Express JS

Sebelum menginstall Express JS pastikan Node dan npm sudah terinstall terlebih dahulu dan lakukan initalisasi Node seperti yang dijelaskan pada module Node JS sebelumnya.

Setelah itu kita bisa menginstall Express JS dengan menjalankan perintah

```
npm install --save express
```

Setup Express JS

Setelah melakukan install express kita sudah bisa melakukan development. Namun untuk mempermudah proses development selanjutnya kita bisa menginstall package tambahan yaitu **nodemon** dengan cara menjalankan perintah

```
npm install -g nodemon
```

Package ini akan membuat Node server yang kita jalankan akan otomatis restart ketika kita melakukan perubahan pada code sehingga kita tidak perlu melakukan restart manual node setiap melakukan perubahan.

Run Express JS

Selanjutnya kita bisa memulai membuat aplikasi menggunakan Express. Buatlah file baru dengan nama **index.js**.

```
var express = require('express');
var app = express();

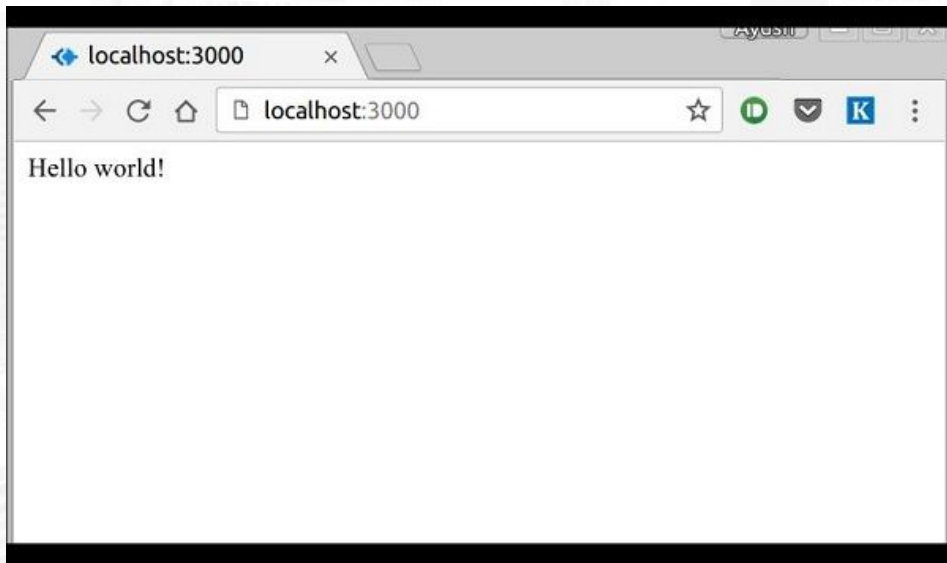
app.get('/', function(req, res){
  res.send("Hello world!");
});

app.listen(3000);
```

Save file lalu jalankan perintah berikut untuk menjalankan

```
nodemon index.js
```

Run Express JS



Dengan menjalankan perintah tersebut maka aplikasi akan berjalan di url <http://localhost:3000> dengan tampilan seperti disamping.

Express JS



Pengenalan Express JS



Setup Environment Express JS



Konsep Routing dan implementasinya

Pengenalan Routing

Routing adalah metode yang digunakan website (server) untuk merespons permintaan dari browser (client). Misalnya, permintaan untuk menampilkan halaman tertentu.

Cara kerja routing di Express.js adalah dengan sebuah metode bernama `app`. Metode tersebut akan merespons setiap permintaan berbentuk **HTTP**. Misalnya **GET**, **POST**, **PUT**, dan **DELETE**.

Berikut contoh skrip **index.js** routing di Express

```
app.get('/hello', function(req, res){
  res.send("Hello World!");
});

app.post('/hello', function(req, res){
  res.send("You just called the post method at '/hello'!\n");
});
```

Implementasi Router Demi Efisiensi

Untuk maintain semua routes di dalam satu file saja merupakan hal yang sangat menyulitkan dan tidak efisien. Oleh karena itu kita bisa memanfaatkan metode **Express.Route** pada express untuk memudahkan manage file. Buat file baru **things.js** dan tuliskan skrip berikut.

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

router.get('/', function(req, res){
  res.send('GET route on things.');
```



```
});
router.post('/', function(req, res){
  res.send('POST route on things.');
```



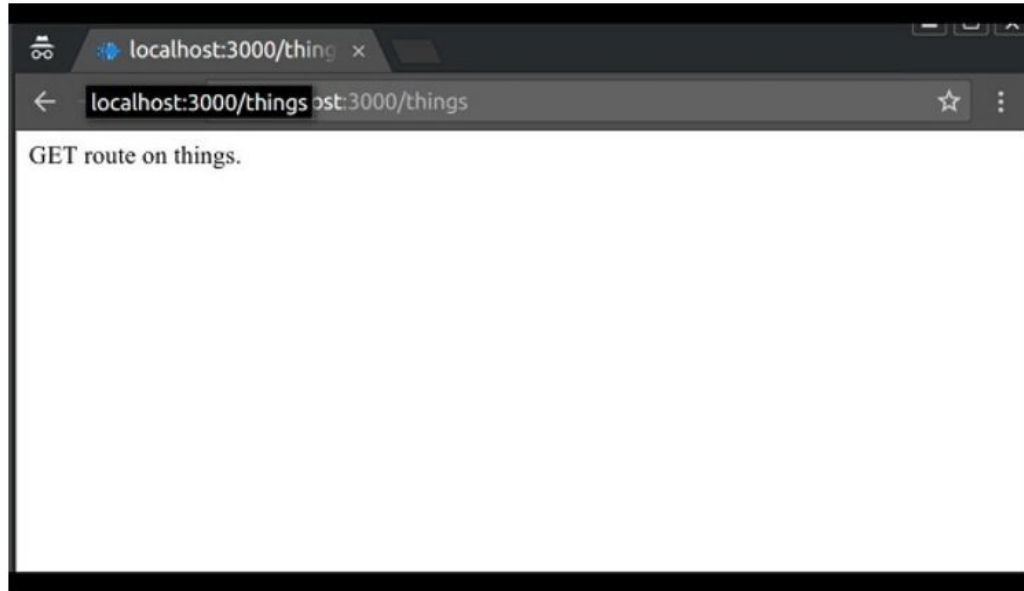
```
});

//export this router to use in our index.js
module.exports = router;
```

Selanjutnya untuk mengimplementasikan skrip tersebut di **index.js** tuliskan skrip berikut sebelum function **app.listen**

```
var express = require('Express');  
var app = express();  
  
var things = require('./things.js');  
  
//both index.js and things.js should be in same directory  
app.use('/things', things);  
  
app.listen(3000);
```

Selanjutnya buka browser kembali dan akses <http://localhost:3000/things> seperti ini.



Router sangat membantu untuk memisahkan file kode kita sehingga lebih mudah untuk dibaca dan menemukan bug lebih mudah. Selain itu file **index.js** akan lebih rapih dan terstruktur.

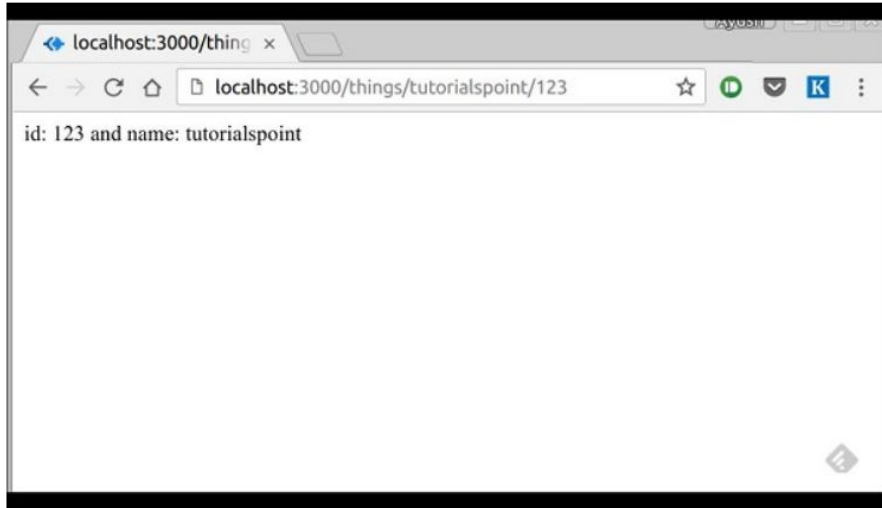
Dynamic Routes

Sekarang kita sudah bisa membuat routes namun masih static atau tidak bisa diubah. Bagaimana dengan routes seperti **/things/cloth** atau **/things/pants** yang menggunakan nama dynamic? Untuk kasus itu kita perlu menggunakan tipe routes yang berbeda. Berikut contoh kode dynamic routes express.

```
var express = require('express');
var app = express();

app.get('/things/:name/:id', function(req, res) {
  res.send('id: ' + req.params.id + ' and name: ' + req.params.name);
});

app.listen(3000);
```



Selanjutnya buka browser kembali dan akses

<http://localhost:3000/things/tutorialspoint/123> seperti di samping.

Kita bisa mengakses berbagai parameter dynamic dengan menggunakan **req.params**.

Study Case

Terdapat sebuah web app yang memiliki beberapa page yang terdiri dari home, about, products, blog, dan contact. Untuk page home, about, dan contact kita bisa membuat page static karena konten di dalamnya tidak berubah. Namun untuk kasus page products dan blog terdapat beberapa sub-sub bagian yang dimana jika kita membuat page satu per satu maka akan sangat tidak efisien dan sulit.

Untuk mengatasi hal ini kita bisa menggunakan express js untuk mengaplikasikan dynamic routing agar kita bisa menampilkan konten secara dinamis dengan hanya membuat satu page untuk products dan blog.

Integrasi Database

- ☐ Menghubungkan Express JS ke Database
- ☐ Query Database Express JS
- ☐ Pengenalan Seeding Database
- ☐ Implementasi Seeding Express JS
- ☐ Pengenalan Migration Database
- ☐ Implementasi Migration Express JS

Objektif sesi

- Siswa memahami cara menghubungkan Database di Express JS
- Siswa memahami cara query database di Express JS
- Siswa memahami tentang konsep seeding
- Siswa memahami cara pengaplikasian seeding pada Express JS
- Siswa memahami tentang konsep migration database
- Siswa memahami cara pengaplikasian migration database

Rancangan Subtopik

Di bawah ini merupakan list subtopik yang sudah dirancang. Namun masih sangat bisa dirubah dan disesuaikan dengan kebutuhan

- Menghubungkan Express JS ke Database
- Query Database Express JS
- Pengenalan Seeding Database
- Implementasi Seeding Express JS
- Pengenalan Migration Database
- Implementasi Migration Database

Integrasi Database



Menghubungkan Express JS ke Database



Query Database Express JS



Pengenalan Seeding Database



Implementasi Seeding Express JS

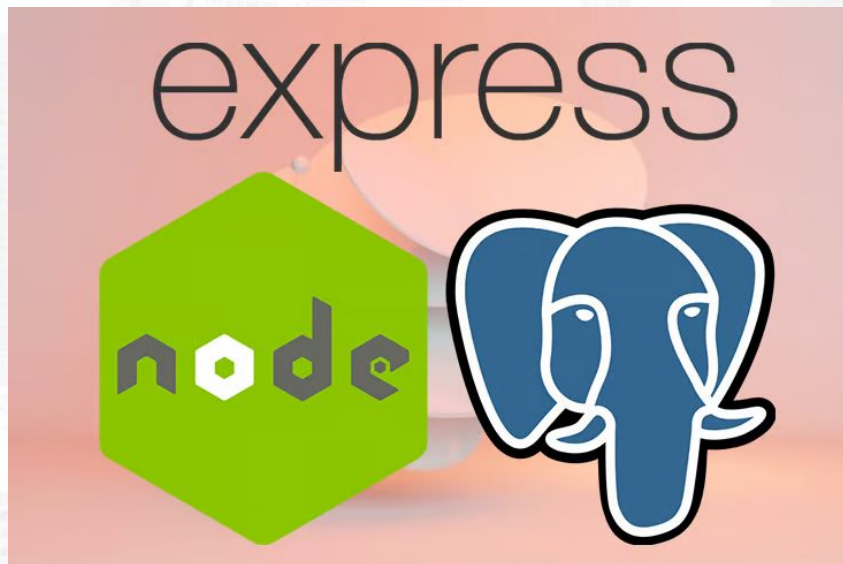


Pengenalan Migration Database



Implementasi Migration Express JS

Menghubungkan Database



Di modul sebelumnya kita sudah mengetahui cara membuat dan menjalankan aplikasi Express JS. Namun belum menghubungkan aplikasi tersebut dengan database. Kita butuh database untuk menyimpan data yang telah dikirim ke server. Untuk pembelajaran kali ini kita akan mencoba menghubungkan database **PostgreSQL** di aplikasi Express JS.

Untuk menghubungkan PostgreSQL kita membutuhkan package **node-postgres** atau **pg** sebagai penghubung aplikasi Express dengan database.

node-postgres merupakan nonblocking client PostgreSQL untuk Node JS. Pada dasarnya, node-postgres adalah kumpulan modul Node.js untuk berinteraksi dengan database PostgreSQL.

Untuk menginstall node-postgress jalankan perintah berikut

```
npm install --save pg
```

Selanjutnya buatlah file baru bernama **queries.js**. Di dalam file ini kita akan menuliskan function untuk menghubungkan ke database dan kumpulan queries untuk mengakses database.

Buatlah schema baru di PostgreSQL untuk menyimpan data yang dikirimkan ke server.

Dalam file **queries.js** tuliskan kode berikut untuk menghubungkan ke database.

```
const Pool = require('pg').Pool
const pool = new Pool({
  user: 'me',
  host: 'localhost',
  database: 'api',
  password: 'password',
  port: 5432,
})
```

Masukkan data credentials sesuai dengan database postgresQL yang sudah kita buat sebelumnya.

Integrasi Database



Menghubungkan Express JS ke Database



Query Database Express JS



Pengenalan Seeding Database



Implementasi Seeding Express JS



Pengenalan Migration Database



Implementasi Migration Express JS

Query Database

Untuk melakukan query kita akan menggunakan function yang telah kita buat pada **queries.js** sebelumnya. Contoh **GET** query seperti ini

```
const getUsers = (request, response) => {
  pool.query('SELECT * FROM users ORDER BY id ASC', (error, results) => {
    if (error) {
      throw error
    }
    response.status(200).json(results.rows)
  })
}
```

Untu **PUT** seperti ini

```
const updateUser = (request, response) => {
  const id = parseInt(request.params.id)
  const { name, email } = request.body

  pool.query(
    'UPDATE users SET name = $1, email = $2 WHERE id = $3',
    [name, email, id],
    (error, results) => {
      if (error) {
        throw error
      }
      response.status(200).send(`User modified with ID: ${id}`)
    }
  )
}
```

Untuk **DELETE** seperti ini

```
const deleteUser = (request, response) => {
  const id = parseInt(request.params.id)

  pool.query('DELETE FROM users WHERE id = $1', [id], (error, results) => {
    if (error) {
      throw error
    }
    response.status(200).send(`User deleted with ID: ${id}`)
  })
}
```

Integrasi Database

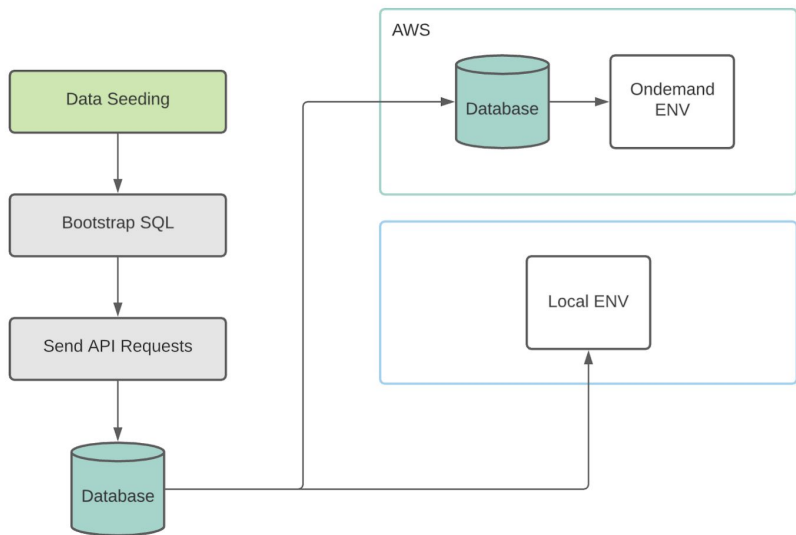
- ☒ Menghubungkan Express JS ke Database
- ☒ Query Database Express JS



Pengenalan Seeding Database

- ☐ Implementasi Seeding Express JS
- ☐ Pengenalan Migration Database
- ☐ Implementasi Migration Express JS

Pengenalan Seeding



Data seeding adalah suatu kegiatan menambahkan data awal pada database sebelum aplikasi dijalankan. Seringkali berguna untuk pra-populasi database dengan *hard-coded* data ketika kita mengatur aplikasi pertama kali. Hal ini akan mempermudah development dengan adanya dummy data untuk diolah.

Ketika membuat suatu aplikasi, pasti perlu melakukan input data dengan initial data yang nantinya akan digunakan untuk *development*, *demo purposes*, *proof of concept*, dan lain-lain. Tentu semakin besar suatu proyek juga akan menyebabkan jumlah data yang semakin banyak pula (**big data**), maka dari itu *data seeding* sangat bermanfaat jika perubahan-perubahan besar terjadi pada database sehingga kita tidak perlu memasukkan data satu per satu dari awal karena kita sudah punya data *copy* yang siap di *seeding*.



Integrasi Database

- ☒ Menghubungkan Express JS ke Database
- ☒ Query Database Express JS
- ☒ Pengenalan Seeding Database



Implementasi Seeding Express JS



Pengenalan Migration Database



Implementasi Migration Express JS

Implementasi Seeding

Untuk melakukan seeding di Express JS langkah pertama yang bisa kita lakukan adalah membuat file sql untuk input dummy data. Contohnya seperti ini

```
DROP TABLE IF EXISTS _USER;  
  
CREATE TABLE _USER(  
  ID INT PRIMARY KEY NOT NULL,  
  FIRST_NAME VARCHAR(255) NOT NULL,  
  LAST_NAME VARCHAR(50) NOT NULL  
);  
  
INSERT INTO _USER (ID, FIRST_NAME, LAST_NAME)  
VALUES (1, 'John', 'Doe');  
  
INSERT INTO _USER (ID, FIRST_NAME, LAST_NAME)  
VALUES (2, 'Alice', 'Wonderland');
```

Simpanlah file di dalam folder **db** dengan nama **seeding.sql**

Selanjutnya kita bisa menggunakan function pool pada library **node-postgress** untuk menjalankan kode sql yang telah dibuat seperti berikut

```
const seedQuery = fs.readFileSync('db/seeding.sql', { encoding: 'utf8' })
pool.query(seedQuery, (err, res) => {
  console.log(err, res)
  console.log('Seeding Completed!')
  pool.end()
})
```

Simpan file di dalam folder yang sama dengan skrip seeding sebelumnya.

Untuk mempermudah kita bisa membuat command di **package.json** untuk menjalankan perintah seeding

```
...
"seed": "node db/index.js"
...
```

Integrasi Database

- ☒ Menghubungkan Express JS ke Database
- ☒ Query Database Express JS
- ☒ Pengenalan Seeding Database
- ☒ Implementasi Seeding Express JS



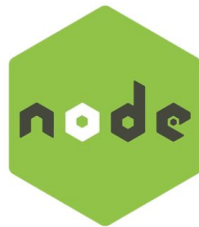
Pengenalan Migration Database

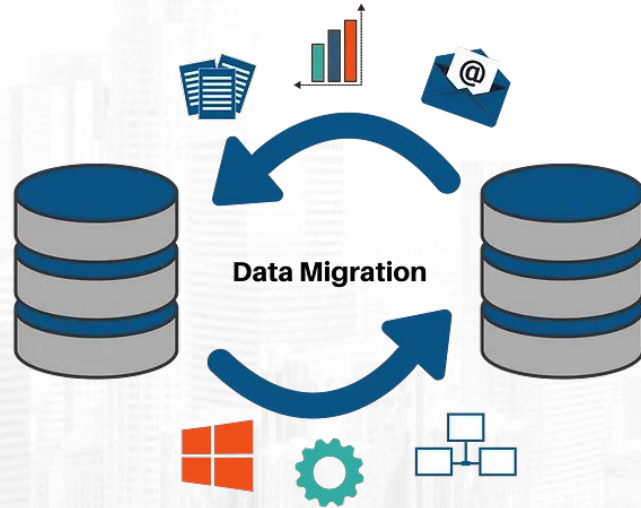


Implementasi Migration Express JS

Pengenalan Database Migration

Migration atau database migration, adalah salah satu fitur yang cukup memudahkan kita ketika ada pembuatan atau perubahan pada tabel-tabel di database aplikasi kita. Entah itu penambahan kolom, indexs dan lain sebagainya. Dengan database migration, kita dapat membuat tabel-tabel tanpa harus membuka aplikasi administrasi database, seperti phpmyadmin, navicat, sqlyog, dan lainnya.





Dengan fitur migration, proses pembuatan dan modifikasi tabel database dapat dilakukan dalam script. Penggunaan database migration ini akan sangat “terasa” bagi developer yang terbiasa dalam pengembangan sistem bersama tim dimana dalam pengerjaan sistem biasanya dikerjakan lebih dari 1 orang. Setiap ada perubahan tabel, maka akan tersimpan di source control seperti git. Selain itu, kita juga dapat melakukan undo dan redo perubahan database dengan mudah.



git

Integrasi Database

- ☒ Menghubungkan Express JS ke Database
- ☒ Query Database Express JS
- ☒ Pengenalan Seeding Database
- ☒ Implementasi Seeding Express JS
- ☒ Pengenalan Migration Database



Implementasi Migration Express JS

Implementasi Migration

Langkah pertama untuk melakukan migrasi kita perlu menginstall library secara global **db-migrate** dan **db-migrate-pg** dengan cara seperti berikut

```
npm install -g db-migrate  
npm install -g db-migrate-pg
```

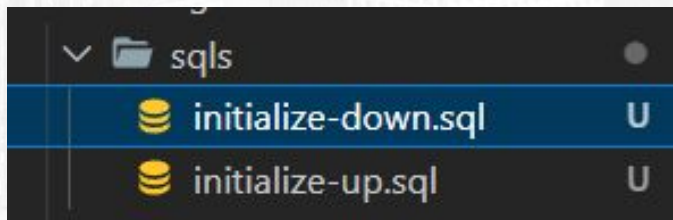
Setelah proses install selesai buat file baru bernama **database.json** dan tuliskan kode berikut.

```
{  
  "dev": {  
    "driver": "pg",  
    "user": "postgres",  
    "password": "password",  
    "host": "localhost",  
    "database": "migrationdb"  
  }  
}
```

Untuk menuliskan migration pertama kali kita bisa menjalankan command seperti ini

```
db-migrate create initialize --sql-file
```

Jika berhasil maka secara otomatis system akan membuat folder baru bernama migration. Folder ini berisi sub-folder bernama sqls yang didalamnya berisi dua file, initialize-up.sql dan initialize-down.sql. Di dalam file inilah kita akan menuliskan kode migration.



Migration up berisi command sql yang akan kita jalankan untuk melakukan migrasi sedangkan migration down berisi command untuk mengembalikan migrasi ke versi sebelumnya.

Untuk mengaplikasikan migration up yang baru dengan cara menjalankan command

```
db-migrate up 20150207135259-myFancyMigration
```

Perintah tersebut akan menjalankan migration spesifik yang telah kita tentukan. Dalam kasus diatas perintah akan menjalankan migrasi yg bernama *-myFancymigration.sql. Sedangkan jika ingin menjalankan seluruh migrasi sekaligus bisa hanya dengan command **db-migrate up** tanpa mendefinisikan nama file.

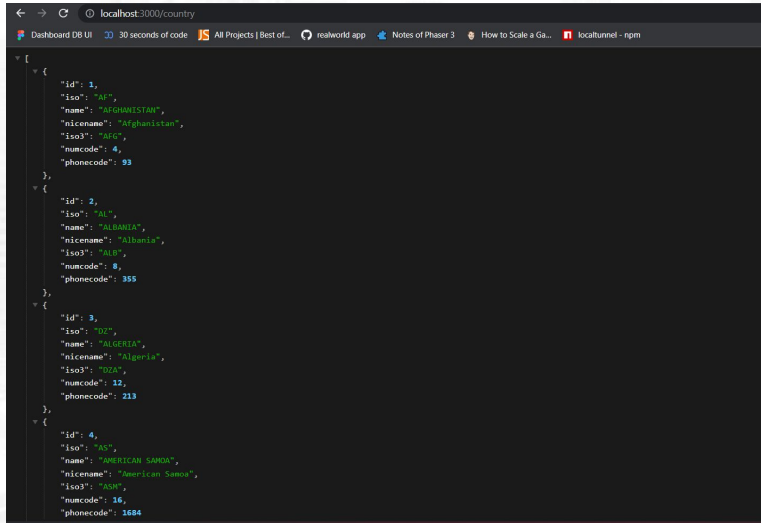
Sedangkan untuk migration down dilakukan dengan perintah **db-migrate down** atau jika ingin menjalankan migration down secara sekaligus dapat dilakukan dengan perintah **db-migrate reset**.

Study Case

Berikut merupakan sample data kumpulan list data tentang kota, negara, bahasa yang digunakan, dan lainnya. [Sample data](#)

Import sample data tersebut ke dalam database postgresQL dan gunakan express js dengan mengkoneksikannya ke database untuk menampilkan list negara yang ada di dunia.

Jika berhasil maka akan menampilkan halaman seperti disamping.



```

{
  "id": 1,
  "iso": "AF",
  "name": "AFGHANISTAN",
  "nickname": "Afghanistan",
  "iso3": "AFG",
  "numcode": 4,
  "phonecode": 93
},
{
  "id": 2,
  "iso": "AL",
  "name": "ALBANIA",
  "nickname": "Albania",
  "iso3": "ALB",
  "numcode": 8,
  "phonecode": 355
},
{
  "id": 3,
  "iso": "DZ",
  "name": "ALGERIA",
  "nickname": "Algeria",
  "iso3": "DZA",
  "numcode": 12,
  "phonecode": 213
},
{
  "id": 4,
  "iso": "AS",
  "name": "AMERICAN SAMOA",
  "nickname": "American Samoa",
  "iso3": "ASM",
  "numcode": 16,
  "phonecode": 1684
}

```


Integrasi Database

- ✓ Menghubungkan Express JS ke Database
- ✓ Query Database Express JS
- ✓ Pengenalan Seeding Database
- ✓ Implementasi Seeding Express JS
- ✓ Pengenalan Migration Database
- ✓ Implementasi Migration Express JS

Reference material

- [Menghubungkan Database Express JS](#)
- [Seeding Database](#)
- [Database Migration](#)

Reference material

- [Pengenalan Express JS](#)
- [Setup Environment Express JS](#)
- [Konsep Routing](#)



Thank you