SMK Telkom Malang

JOB SHEET

NODE JS

Sequelize Bagian II

Disusun Oleh:
TIM MGMP RPL
SMK TELKOM MALANG

Restful API Node.js dan MySQL menggunakan Sequelize

Pada modul sebelumnya kita telah membuat struktur project dan menginisiasi Sequelize mulai dari konfigurasi database, migration, model, dan seeder. Pada modul ini kita akan melanjutkan tahapan selanjutnya pada project yang kita buat. Pada tahapan ini kita akan membuat bagian CRUD tabel member.

A. CRUD menggunakan Sequelize

1. Inisiasi Model

   Proses inisiasi model merupakan langkah awal sebelum kita membuat proses CRUD menggunakan Sequelize. Proses ini merupakan pemanggilan file model yang dibutuhkan untuk proses CRUD. Misalkan kita akan memanggil file model "member" maka kita dapat menggunakan code seperti berikut.

   ```
   const memberModel = require(`../models/index`).member
   ```

   File yang dipanggil adalah file "index.js" yang ada di dalam folder "models". Kemudian memanggil key "member" sebagai merupakan model yang kita kehendaki.

2. Creating Data

   Untuk proses membuat atau menambahkan data baru kita dapat menggunakan function "create()" dari model yang telah kita panggil.

   ```
   await member.create({
       name: "John Cena",
       gender: "Male",
       contact: "088 2321",
       address: "Madinah"
   })
   ```

   Fungsi create() merupakan sebuah **promise** yang harus di-*handle* menggunakan **then()** dan **catch()** atau dapat menggunakan **await** yang harus di dalam scope ***async function***. Parameter yang di-passing pada function create() adalah object yang berisi data yang akan ditambahkan. Adapun struktur dari object tersebut sesuai dengan struktur tabel atau model yang digunakan. Code di atas jika dijalankan akan menjalankan perintah SQL seperti berikut.

   ```
   INSERT INTO `members` (name, gender, contact, address)
   VALUE ('John Cena', 'Male', '088 2321', 'Madinah')
   ```

3. Updating Data

   Untuk proses update data pada Sequelize dapat menggunakan function update() dari model yang kita panggil.

```
await member.update({ address: "Mina" },{
    where: {
        id: 1
    }
})
```

Fungsi update() merupakan sebuah **promise** yang harus di-*handle* menggunakan **then()** dan **catch()** atau dapat menggunakan **await** yang harus di dalam scope ***async function***. Parameter yang di-passing pada function update() adalah object yang berisi perubahan data dan object parameter yang dijadikan patokan data yang akan diubah. Code di atas jika dijalankan akan menjalankan perintah SQL seperti berikut.

```
UPDATE `members` SET address = `Mina` WHERE id = 1
```

4. Deleting Data
   Untuk proses update data pada Sequelize dapat menggunakan function destroy() dari model yang kita panggil.

```
await member.destroy({
    where: {
        id: 1
    }
})
```

Fungsi destroy() merupakan sebuah **promise** yang harus di-*handle* menggunakan **then()** dan **catch()** atau dapat menggunakan **await** yang harus di dalam scope ***async function***. Parameter yang di-passing pada function destroy() adalah object parameter yang dijadikan patokan data yang akan diubah. Code di atas jika dijalankan akan menjalankan perintah SQL seperti berikut.

```
DELETE FROM `members` WHERE id = 1
```

5. Read Data
   Untuk proses *read data* pada Sequelize dapat menggunakan function findAll(), findByPk(), findOne(), findAndCountAll().
   - **findAll**() digunakan untuk mendapatkan semua data dari tabel dan parameter pencarian yang dikehendaki. Hasil perintah ini berupa **array object** dari data yang didapatkan. Perhatikan contoh berikut ini.
     Contoh 1:

```
await member.findAll()
```

     Perintah di atas akan men-generate SQL berikut.

```
SELECT * FROM `members`
```

     Contoh 2:

```
await member.findAll({
    where: {
        gender: `Male`
    }
})
```

Perintah di atas akan men-generate SQL berikut.

```
SELECT * FROM `members` WHERE gender = `Male`
```

- **findByPk**() digunakan untuk mendapatkan data berdasarkan nilai data primary key yang dikehendaki. Hasil perintah ini berupa **object** dari data yang didapatkan. Perhatikan contoh berikut ini.

  ```
  await member.findByPk(29)
  ```

  Perintah di atas akan men-generate SQL berikut.

  ```
  SELECT * FROM `members` WHERE id = 29
  ```

- **findOne**() digunakan untuk mendapatkan data berdasarkan parameter pencarian yang dikehendaki (penggunaan ***where clause***). Hasil perintah ini berupa **object** dari data yang didapatkan. Perhatikan contoh berikut ini.

  ```
  await member.findOne({
      where: {
          address: `Madinah`
      }
  })
  ```

  Perintah di atas akan men-generate SQL berikut.

  ```
  SELECT * FROM `members` WHERE address = `Madinah`
  ```

- **findAndCountAll**() digunakan untuk mendapatkan semua data dari tabel dan parameter pencarian yang dikehendaki beserta jumlah data yang didapatkan. Hasil perintah ini berupa **object** yang berisi dua key yaitu key "count" bernilai jumlah data yang didapatkan dan key "rows" berisi **array object** data yang didapatkan.

Semua fungsi find() di atas merupakan sebuah **promise** yang harus di-*handle* menggunakan **then()** dan **catch()** atau dapat menggunakan **await** yang harus di dalam scope ***async function***.

B. Filtering Data (Where Clause)

Klausa "*where*" digunakan untuk menyaring data yang akan diambil dari sebuah tabel. Jika terdapat beberapa filter yang digunakan, maka kalusa "where" membutuhkan "operator" untuk menjadikan filtering data lebih kompleks seperti operator kurang dari, lebih dari, AND, OP, NOT, dan sebagainnya. Untuk

memanfaatkan penggunaan operator pada Sequelize, kita harus menginisiasi seperti berikut ini.

```
const Op = require(`sequelize`).Op
```
atau
```
const {Op} = require(`sequelize`)
```

Perhatikan beberapa *case* berikut untuk memahami penggunaan Operator pada *Where Clause*.

1. Case Pertama: Mendapatkan data member dengan gender "Male" dan address "Mina".

```
await member.findAll({
    where: {
        gender: `Male`,
        address: `Mina`
    }
})
```

Atau

```
await member.findAll({
    where: {
        [Op.and]: [
            {
                gender: `Male`,
                address: `Mina`
            }
        ]
    }
})
```

Code di atas akan di-generate menjadi perintah SQL berikut.

```
SELECT * FROM `members`
WHERE gender = "Male" and address = "Mina"
```

2. Case Kedua: Medapatkan data member yang masuk diantara tanggal 1 Mei 2022 sampai 30 Mei 2022.

```
await member.findAll({
    where: {
        createdAt: {
            [Op.between] : ["2022-05-01","2022-05-30"]
        }
    }
})
```

Code di atas akan di-generate menjadi perintah SQL berikut.

```
SELECT * FROM `members`
WHERE createdAt BETWEEN "2022-05-01" AND "2022-05-30"
```

3. Case Ketiga: Mendapatkan data member yang namanya mengandung kata "John".

```
await member.findAll({
    where: {
        name: {
            [Op.substring] : "John"
        }
    }
})
```

Code di atas akan di-generate menjadi perintah SQL berikut.

```
SELECT * FROM `members`
WHERE name LIKE "%John%"
```

4. Case Keempat: Mendapatkan data member yang masuk sebelum tanggal 30 Januari 2022

```
await member.findAll({
    where: {
        createdAt: {
            [Op.lt] : "2022-01-30"
            // lt is "less than"
        }
    }
})
```

Code di atas akan di-generate menjadi perintah SQL berikut.

```
SELECT * FROM `members` WHERE createdAt < "2022-01-30"
```

5. Case Kelima: Mendapatkan data member yang mengandung kata "hai".

```
await member.findAll({
    where: {
        [Op.or]: [
            { name: { [Op.substring]: "hai" } },
            { address: { [Op.substring]: "hai" } },
            { gender: { [Op.substring]: "hai" } }
        ]
    }
})
```

Code di atas akan di-generate menjadi perintah SQL berikut.

```
SELECT * FROM `members`
WHERE name LIKE "%hai%"
OR address LIKE "%hai%"
OR gender LIKE "%hai%"
```

Untuk lebih banyak referensi tentang Sequelize Operator dapat dilihat pada laman ini https://sequelize.org/docs/v6/core-concepts/model-querying-basics/#operators

C. Langkah Praktikum

1. Buatlah file baru pada folder controllers dengan nama "member.controller.js"!
2. Buat code untuk memanggil model untuk tabel "member" dan object Operation dari library Sequelize seperti berikut!

```
/** load model for `members` table */
const memberModel = require(`../models/index`).member

/** load Operation from  Sequelize  */
const Op = require(`sequelize`).Op
```

3. Buat fungsi untuk mendapatkan semua data member seperti berikut!

```
/** create function for read all data */
exports.getAllMember = async (request, response) => {
    /** call findAll() to get all data */
    let members = await memberModel.findAll()
    return response.json({
        success: true,
        data: members,
        message: `All Members have been loaded`
    })
}
```

4. Buat fungsi untuk pencarian data member berdasarkan keyword yang ditentukan seperti berikut!

```
/** create function for filter */
exports.findMember = async (request, response) => {
    /** define keyword to find data */
    let keyword = request.body.keyword

    /** call findAll() within where clause and operation
     * to find data based on keyword  */
    let members = await memberModel.findAll({
        where: {
            [Op.or]: [
                { name: { [Op.substring]: keyword } },
                { gender: { [Op.substring]: keyword } },
                { address: { [Op.substring]: keyword } }
            ]
        }
    })
```

```
    return response.json({
        success: true,
        data: members,
        message: `All Members have been loaded`
    })
}
```

5. Buat fungsi untuk penambahan data member baru seperti berikut!

```
/** create function for add new member */
exports.addMember = (request, response) => {
    /** prepare data from request */
    let newMember = {
        name: request.body.name,
        address: request.body.address,
        gender: request.body.gender,
        contact: request.body.contact
    }

    /** execute inserting data to member's table */
    memberModel.create(newMember)
        .then(result => {
            /** if insert's process success */
            return response.json({
                success: true,
                data: result,
                message: `New member has been inserted`
            })
        })
        .catch(error => {
            /** if insert's process fail */
            return response.json({
                success: false,
                message: error.message
            })
        })
}
```

6. Buat fungsi untuk pengubahan data member seperti berikut!

```
/** create function for update member */
exports.updateMember = (request, response) => {
    /** prepare data that has been changed */
    let dataMember = {
        name: request.body.name,
        address: request.body.address,
```

```
        gender: request.body.gender,
        contact: request.body.contact
    }

    /** define id member that will be update */
    let idMember = request.params.id

    /** execute update data based on defined id member */
    memberModel.update(dataMember, { where: { id: idMember } })
        .then(result => {
            /** if update's process success */
            return response.json({
                success: true,
                message: `Data member has been updated`
            })
        })
        .catch(error => {
            /** if update's process fail */
            return response.json({
                success: false,
                message: error.message
            })
        })
}
```

7. Buat fungsi untuk penghapusan data member seperti berikut!

```
/** create function for delete data  */
exports.deleteMember = (request, response) => {
    /** define id member that will be update */
    let idMember = request.params.id

    /** execute delete data based on defined id member */
    memberModel.destroy({ where: { id: idMember } })
        .then(result => {
            /** if update's process success */
            return response.json({
                success: true,
                message: `Data member has been updated`
            })
        })
        .catch(error => {
            /** if update's process fail */
            return response.json({
                success: false,
                message: error.message
```

```
            })
        })
}
```

8. Buatlah file baru pada folder "routes" dengan nama "member.route.js"!
9. Buat code untuk memanggil dan inisiasi library "express" seperti berikut!

```
/** load library express */
const express = require(`express`)

/** initiate object that instance of express */
const app = express()
```

10. Buat code untuk membuka akses membaca data request yang bertipe JSON seperti berikut ini!

```
/** allow to read 'request' with json type */
app.use(express.json())
```

11. Buat code untuk memanggil file controller dari member seperti berikut ini!

```
/** load member's controller */
const memberController =
require(`../controllers/member.controller`)
```

12. Buat code untuk menentukan route pada proses mendapatkan data member dengan method "GET" seperti berikut ini!

```
/** create route to get data with method "GET" */
app.get("/", memberController.getAllMember)
```

13. Buat code untuk menentukan route pada proses menambahkan data member dengan method "POST" seperti berikut ini!

```
/** create route to add new member using method "POST" */
app.post("/", memberController.addMember)
```

14. Buat code untuk menentukan route pada proses mencari data member dengan method "POST" seperti berikut ini!

```
/** create route to find member
 * using method "POST" and path "find" */
app.post("/find", memberController.findMember)
```

15. Buat code untuk menentukan route pada proses mengubah data member dengan method "PUT" seperti berikut ini!

```
/** create route to update member
 * using method "PUT" and define parameter for "id" */
app.put("/:id", memberController.updateMember)
```

16. Buat code untuk menentukan route pada proses menghapus data member dengan method "DELETE" seperti berikut ini!

```
/** create route to delete member
 * using method "DELETE" and define parameter for "id" */
app.delete("/:id", memberController.deleteMember)
```

17. Buat code untuk mengekspor objek "app" agar dapat dimuat pada file lain!

```
/** export app in order to load in another file */
module.exports = app
```

18. Buat file baru pada root folder project dengan nama server.js!
19. Tuliskan code berikut ini pada file server.js!

```
/** load library express */
const express = require(`express`)

/** create object that instances of express */
const app = express()

/** define port of server */
const PORT = 8000

/** load library cors */
const cors = require(`cors`)

/** open CORS policy */
app.use(cors())

/** define all routes */
const memberRoute = require(`./routes/member.route`)

/** define prefix for each route */
app.use(`/member`, memberRoute)

/** run server based on defined port */
```

```
app.listen(PORT, () => {
    console.log(`Server of School's Library runs on port
${PORT}`)
})
```

20. Jalankan perintah "**npm start**" pada terminal atau command prompt!

```
D:\Nodejs\school_library>npm start

> school_library@1.0.0 start
> nodemon server.js

[nodemon] 2.0.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Server of School's Library runs on port 8000
```

21. Buka Postman dan buat *collection* baru misalkan dengan nama "**School Library**"!
22. Buat folder baru dengan nama "**member**" pada *collection* yang dibuat pada langkah sebelumnya!
23. Buat *request* baru untuk mengambil semua data member, pilih method GET dan tuliskan *request URL* dengan **http://localhost:8000/member** dan klik Send untuk mendapatkan *response*. Jangan lupa untuk "***Save***" request tersebut.
24. Tambahkan *request* baru untuk menambah data member baru, pilih method POST dan tuliskan *request URL* dengan **http://localhost:8000/member**. Kemudian tambahkan data request pada tab **Body**, pilih option "**raw**" dan pilih format **JSON**. Sebagai contoh data *request* yang ditambahkan adalah sebagai berikut.

```
{
    "name":"Eddie",
    "address":"Kongo",
    "contact":"099 182",
    "gender":"Male"
}
```

    Klik Send untuk mendapatkan *response*. Jangan lupa untuk "***Save***" request tersebut.
25. Tambahkan *request* baru untuk mengubah data member, pilih method PUT dan tuliskan *request URL* dengan **http://localhost:8000/member/4**. Kemudian tambahkan data request pada tab **Body**, pilih option "**raw**" dan pilih format **JSON**. Sebagai contoh data *request* yang ditambahkan adalah sebagai berikut.

```
{
    "name":"Eddie",
    "address":"Zimbabwe",
    "contact":"099 182",
    "gender":"Male"
}
```

Klik Send untuk mendapatkan *response*. Jangan lupa untuk "***Save***" request tersebut.

26. Tambahkan *request* baru untuk menghapus data member, pilih method DELETE dan tuliskan *request URL* dengan **http://localhost:8000/member/4**. Klik Send untuk mendapatkan *response*. Jangan lupa untuk "***Save***" request tersebut.
27. Tambahkan *request* baru untuk mencari data member, pilih method POST dan tuliskan *request URL* dengan **http://localhost:8000/member/find**. Kemudian tambahkan data request pada tab **Body**, pilih option "**raw**" dan pilih format **JSON**. Sebagai contoh data *request* yang ditambahkan adalah sebagai berikut.

```
{
    "keyword":"John"
}
```

Klik Send untuk mendapatkan *response*. Jangan lupa untuk "***Save***" request tersebut.

D. Tugas Praktikum

Buatlah proses CRUD untuk tabel **admin** dengan menambah controller, route, dan menambahkan konfigurasi di file server.js! Pada data admin, terdapat *password* yang harus dilakukan *hashing* untuk menyembunyikan informasi yang sebenarnya. *Hashing* dilakukan menggunakan library md5. Berikut contoh penggunaan library md5.

```
const md5 = require(`md5`)
let password = md5(`password`)
/** result: 5f4dcc3b5aa765d61d8327deb882cf99 */
```