“Build your imagination with Naratass Software”

Backend Dasar dengan Node.js + Express Bagian 4

Mengimplementasikan fitur tambahan (Pagination, Softdelete, Restore dan Force Delete) untuk data Catatan.

Pertemuan 15— Belajar JavaScript Tingkat Basic
Disampaikan oleh: Aries Indrabayu Yunandar

Tujuan Pembelajaran

- Dasar JavaScript untuk backend
- Cara kerja Express dan routing
- Struktur proyek backend modern
- Membaca dan memodifikasi controller & router
- Validasi dan migrasi database

Struktur Folder

```
catatan-api/
├── config/
│   └── db.js
├── controller/
│   └── catatanController.js
├── middlewares/
│   └── validate.js
├── migrations/
├── node_modules/
├── routes/
│   └── catatan.js
├── validators/
│   ├── catatanValidator.js
│   ├── paramValidator.js
│   └── queryValidator.js
├── .env
├── knexfile.js
├── package-lock.json
├── package.json
└── server.js
```

migrations/20250706064531_update_catatan_add_fields.js

Buat migrasi baru untuk update tabel catatan:
npx knex migrate:make update_catatan_add_fields

```
exports.up = function (knex) {
  return knex.schema.createTable("catatan", (table) => {
    table.increments("id");
    table.text("isi").notNullable();
    table.string("kategori").nullable();
    table.text("keterangan").nullable();
    table.boolean("status").defaultTo(true);
    table.timestamp("deleted_at").nullable();
    table.timestamps(true, true); // created_at & updated_at
  });
};
exports.down = function (knex) {
  return knex.schema.alterTable("catatan", function (table) {
    table.dropColumn("kategori");
    table.dropColumn("keterangan");
    table.dropColumn("status");
    table.dropColumn("deleted_at");
    table.dropTimestamps();
  });
};
```

controller/catatanController.js #1

```
const db = require("../config/db");

// CREATE
const createCatatan = async (req, res) => {
  try {
    const { isi, kategori, keterangan, status } =
req.body;
    const [result] = await db.query(
      `INSERT INTO catatan (isi, kategori,
keterangan, status, created_at, updated_at) VALUES
(?, ?, ?, ?, NOW(), NOW())`,
      [isi, kategori, keterangan, status ?? true]
    );
    res.status(201).json({ status: "success", id:
result.insertId });
  } catch (err) {
    res.status(500).json({ status: "error", message:
err.message });
  }
};

// READ BY ID
const getCatatanById = async (req, res) => {
  try {
    const [rows] = await db.query(
      `SELECT * FROM catatan WHERE id = ? AND
deleted_at IS NULL`,
      [req.params.id]
    );
    if (rows.length === 0) return
res.status(404).json({ status: "not_found" });
    res.json({ status: "success", data: rows[0] });
  } catch (err) {
    res.status(500).json({ status: "error", message:
err.message });
  }
};

// UPDATE
const updateCatatan = async (req, res) => {
  try {
    const { isi, kategori, keterangan, status } =
req.body;
    const [result] = await db.query(
      `UPDATE catatan SET isi = ?, kategori = ?,
keterangan = ?, status = ?, updated_at = NOW() WHERE
id = ? AND deleted_at IS NULL`,
      [isi, kategori, keterangan, status,
req.params.id]
    );
    res.json({ status: result.affectedRows ?
"success" : "not_found" });
  } catch (err) {
    res.status(500).json({ status: "error", message:
err.message });
  }
};
```

```
// READ ALL (tanpa yang terhapus)
const getAllCatatan = async (req, res) => {
  try {
    const page = parseInt(req.query.page) || 1;
    const size = parseInt(req.query.size) || 10;
    const search = req.query.search || "";
    const sort_by = req.query.sort_by || "id";
    const sort_dir =
      req.query.sort_dir?.toUpperCase() === "DESC"
? "DESC" : "ASC";
    const deleted = req.query.deleted || "0"; // 0
= aktif, 1 = terhapus, all = semua

    // Kondisi WHERE untuk filtering berdasarkan
deleted_at
    let deletedWhere = "";
    if (deleted === "0") {
      deletedWhere = "deleted_at IS NULL";
    } else if (deleted === "1") {
      deletedWhere = "deleted_at IS NOT NULL";
    } else {
      deletedWhere = "1"; // semua (tanpa filter
deleted_at)
    }

    // WHERE untuk search + deleted
    const whereClause = `WHERE (${deletedWhere})
AND (isi LIKE ? OR kategori LIKE ? OR keterangan
LIKE ?)`;

    const offset = (page - 1) * size;

    // Total data
    const [totalResult] = await db.query(
      `SELECT COUNT(*) as count FROM catatan
${whereClause}`,
      [`%${search}%`, `%${search}%`,
`%${search}%`]
    );
    const total = totalResult[0].count;
    const total_page = Math.ceil(total / size);

    // Data hasil query
    const [rows] = await db.query(
      `SELECT * FROM catatan ${whereClause} ORDER
BY ${sort_by} ${sort_dir} LIMIT ? OFFSET ?`,
      [`%${search}%`, `%${search}%`,
`%${search}%`, size, offset]
    );

    res.json({
      status: "success",
      page,
      size,
      total,
      total_page,
      has_next: page < total_page,
      has_prev: page > 1,
      data: rows,
    });
  } catch (err) {
    res.status(500).json({ status: "error",
message: err.message });
  }
};
```

controller/catatanController.js #2

```
// SOFT DELETE
const softDeleteCatatan = async (req, res) => {
  try {
    const [result] = await db.query(
      `UPDATE catatan SET deleted_at = NOW() WHERE
id = ? AND deleted_at IS NULL`,
      [req.params.id]
    );
    if (result.affectedRows) {
      res.json({ status: "success", message:
"Catatan berhasil dihapus" });
    } else {
      res
        .status(404)
        .json({
          status: "not_found",
          message: "Data tidak ditemukan atau sudah
dihapus",
        });
    }
  } catch (err) {
    res.status(500).json({ status: "error", message:
err.message });
  }
};

// RESTORE
const restoreCatatan = async (req, res) => {
  try {
    const [result] = await db.query(
      `UPDATE catatan SET deleted_at = NULL WHERE id
= ? AND deleted_at IS NOT NULL`,
      [req.params.id]
    );
    if (result.affectedRows) {
      res.json({ status: "success", message:
"Catatan berhasil dipulihkan" });
    } else {
      res
        .status(404)
        .json({
          status: "not_found",
          message: "Data tidak ditemukan atau belum
dihapus",
        });
    }
  } catch (err) {
    res.status(500).json({ status: "error", message:
err.message });
  }
};
```

```
// FORCE DELETE
const forceDeleteCatatan = async (req, res) => {
  try {
    const [result] = await db.query(`DELETE FROM
catatan WHERE id = ?`, [
      req.params.id,
    ]);
    if (result.affectedRows) {
      res.json({
        status: "success",
        message: "Catatan berhasil dihapus
permanen",
      });
    } else {
      res
        .status(404)
        .json({ status: "not_found", message:
"Data tidak ditemukan" });
    }
  } catch (err) {
    res.status(500).json({ status: "error",
message: err.message });
  }
};

module.exports = {
  createCatatan,
  getAllCatatan,
  getCatatanById,
  updateCatatan,
  softDeleteCatatan,
  restoreCatatan,
  forceDeleteCatatan,
};
```

validators/catatanValidator.js

```
const { body } = require("express-validator");

// Validasi untuk CREATE
const createCatatanRules = () => [
  body("isi")
    .notEmpty()
    .withMessage("Isi tidak boleh kosong")
    .isLength({ min: 3 })
    .withMessage("Isi minimal 3 karakter"),

  body("kategori")
    .optional()
    .isLength({ max: 50 })
    .withMessage("Kategori maksimal 50 karakter"),

  body("keterangan")
    .optional()
    .isString()
    .withMessage("Keterangan harus berupa teks"),

  body("status")
    .optional()
    .isBoolean()
    .withMessage("Status harus berupa boolean"),
];

// Validasi untuk UPDATE (semua optional, tapi tetap divalidasi jika dikirim)
const updateCatatanRules = () => [
  body("isi")
    .optional()
    .isLength({ min: 3 })
    .withMessage("Isi minimal 3 karakter"),

  body("kategori")
    .optional()
    .isLength({ max: 50 })
    .withMessage("Kategori maksimal 50 karakter"),

  body("keterangan")
    .optional()
    .isString()
    .withMessage("Keterangan harus berupa teks"),

  body("status")
    .optional()
    .isBoolean()
    .withMessage("Status harus berupa boolean"),
];

module.exports = {
  createCatatanRules,
  updateCatatanRules,
};
```

validators/paramValidator.js

```
const { param } = require("express-validator");

const validateIdParam = () => [
  param("id")
    .exists()
    .withMessage("ID wajib disertakan")
    .bail()
    .isInt({ gt: 0 })
    .withMessage("ID harus berupa angka lebih dari 0"),
];

module.exports = { validateIdParam };
```

validators/queryValidator.js

```
const { query } = require("express-validator");

// Kolom yang diizinkan untuk sorting
const allowedSortBy = ["id", "isi", "kategori", "updated_at", "created_at"];

const validateCatatanQuery = () => [
  query("page").optional().isInt({ min: 1 }).withMessage("Page minimal 1"),

  query("size")
    .optional()
    .isInt({ min: 1, max: 100 })
    .withMessage("Size harus antara 1-100"),

  query("search")
    .optional()
    .isLength({ max: 100 })
    .withMessage("Search maksimal 100 karakter"),

  query("sort_by")
    .optional()
    .isIn(allowedSortBy)
    .withMessage(`Sort_by hanya boleh: ${allowedSortBy.join(", ")}`),

  query("sort_dir")
    .optional()
    .isIn(["asc", "desc", "ASC", "DESC"])
    .withMessage("Sort_dir hanya boleh asc / desc"),

  query("deleted")
    .optional()
    .isIn(["0", "1", "all"])
    .withMessage("Deleted hanya boleh 0, 1, atau all"),
];

module.exports = { validateCatatanQuery };
```

middlewares/validate.js

Fungsi `validate` ini adalah middleware untuk:

- Mengecek hasil validasi dari `express-validator`
- Menyederhanakan penanganan error input
- Membantu menjaga kualitas dan keamanan data masuk ke backend

```
const { validationResult } = require("express-validator");

const validate = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(422).json({
      status: "validation_error",
      errors: errors.array(),
    });
  }
  next();
};

module.exports = validate;
```

routes/catatan.js

```
const express = require("express");
const router = express.Router();
const controller =
require("../../controllers/catatanV1Controller");
const {
  createCatatanRules,
  updateCatatanRules,
} = require("../../validators/catatanValidator");
const validate =
require("../../middlewares/validate");

const { validateIdParam } =
require("../../validators/paramValidator");
const { validateCatatanQuery } =
require("../../validators/queryValidator");


// CREATE → Validasi penuh
router.post(
  "/catatan",
  createCatatanRules(),
  validate,
  controller.createCatatan
);

// READ
router.get(
  "/catatan",
  validateCatatanQuery(),
  validate,
  controller.getAllCatatan
);
// GET BY ID
router.get(
  "/catatan/:id",
  validateIdParam(),
  validate,
  controller.getCatatanById
);

// UPDATE → Validasi fleksibel
router.put(
  "/catatan/:id",
  validateIdParam(),
  updateCatatanRules(),
  validate,
  controller.updateCatatan
);

// DELETE (soft)
router.delete(
  "/catatan/:id",
  validateIdParam(),
  validate,
  controller.softDeleteCatatan
);
```

```
// RESTORE
router.patch(
  "/catatan/restore/:id",
  validateIdParam(),
  validate,
  controller.restoreCatatan
);

// FORCE DELETE
router.delete(
  "/catatan/force/:id",
  validateIdParam(),
  validate,
  controller.forceDeleteCatatan
);

module.exports = router;
```

server.js

🧩 Fungsi:

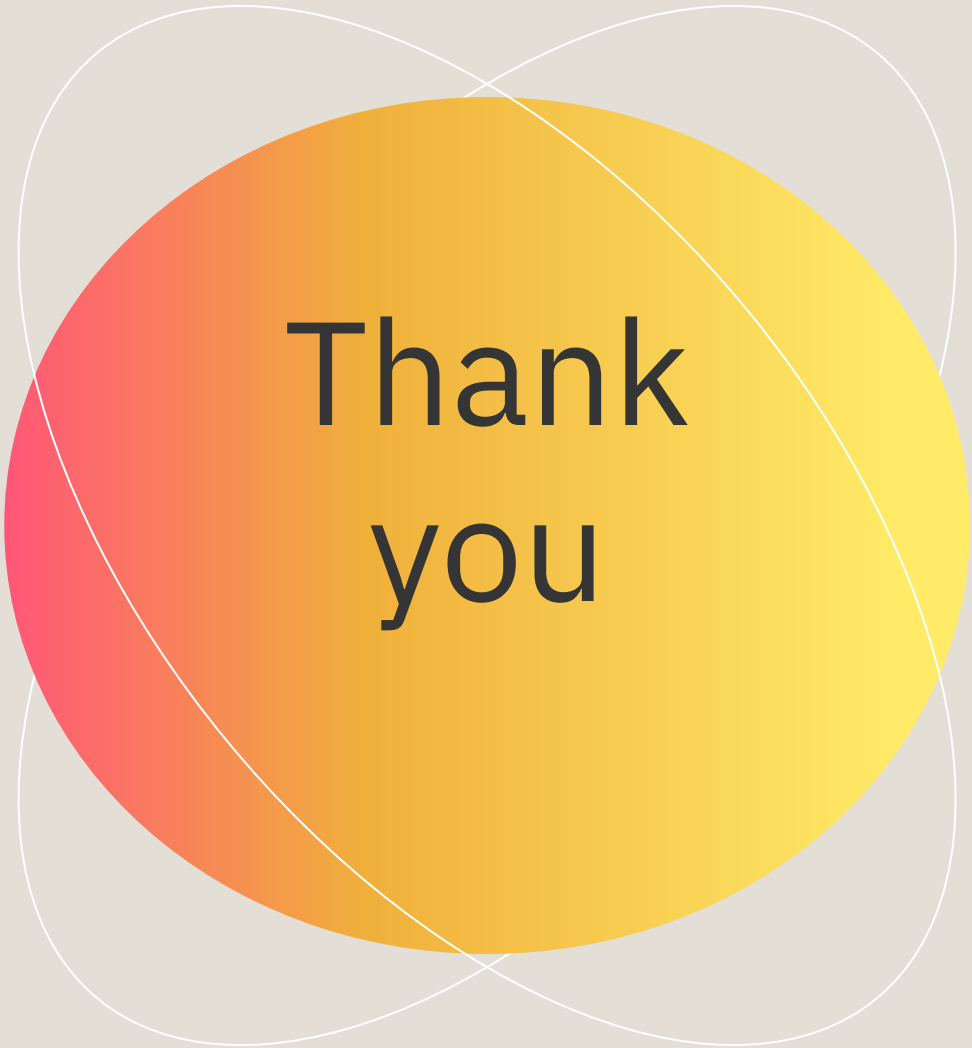
- Inisialisasi Express
- Parsing JSON
- Menghubungkan router
- Menyalakan server

```
const express = require("express");
const cors = require("cors");
const dotenv = require("dotenv");
const catatanRoutes = require("./routes/catatan");
// Load environment variables
dotenv.config();
const PORT = process.env.PORT || 7000;

// Middleware agar bisa baca JSON
app.use(cors()); // ✅ Aktifkan CORS agar frontend bisa ambil data

// ⛔ Menonaktifkan cache untuk file JS
app.use(
  express.static("frontend", {
    setHeaders: (res, path) => {
      if (path.endsWith(".js") || path.endsWith(".html")) {
        res.set("Cache-Control", "no-store");
      }
    },
  })
);
app.use(express.json());

// Prefix api
app.use("/api", catatanRoutes);
// Jalankan server
app.listen(PORT, () => {
  console.log(`Server berjalan di http://localhost:${PORT}`);
});
```


Thank
you