All JavaScript Code Files

File historical_data_service.js

Path: relPath:=

“‘javascript const PowerLog = require(‘./models/PowerLog’);

/** * Mengisi database dengan data historis selama 60 hari terakhir untuk beberapa perangkat. * Masing-masing perangkat punya karakteristik berbeda agar grafik lebih bermakna. */ const populate60DaysData = async () => { try { const logCount = await PowerLog.countDocuments(); if (logCount > 0) { console.log(‘Historical data already exists. Skipping population.’); return; }

```
console.log('Populating database with 60 days of historical data for multiple devices...');
const now = new Date();
const deviceIds = ['digiplug001', 'dp_lamputeras', 'tv', 'dpkamar', 'ac'];
const logs = [];

for (const deviceId of deviceIds) {
  let accumulatedEnergy = 0;

  // Beri karakteristik berbeda tiap device
  const deviceProfile = {
    digiplug001: { baseCurrent: 1.2, currentVar: 0.8 },
    dp_lamputeras: { baseCurrent: 0.4, currentVar: 0.3 },
    tv: { baseCurrent: 0.8, currentVar: 0.5 },
    dpkamar: { baseCurrent: 0.6, currentVar: 0.4 },
    ac: { baseCurrent: 2.0, currentVar: 1.0 },
  }[deviceId];

  for (let day = 60; day >= 0; day--) {
    // Variasikan jumlah titik per hari
    const pointsPerDay = (day <= 2) ? 8 : (day <= 7 ? 4 + Math.floor(Math.random() * 2) : 2

    for (let point = 0; point < pointsPerDay; point++) {
      // Timestamp yang sedikit diacak agar tidak terlalu seragam
      const hourOffset = (24 / pointsPerDay) * point + Math.random() * 2;
      const timestamp = new Date(now.getTime() - (day * 24 * 60 * 60 * 1000) - (hourOffset *

      const isWeekend = timestamp.getUTCDay() === 0 || timestamp.getUTCDay() === 6;
      const hour = timestamp.getUTCHours();

      const usageMultiplier = (hour >= 18 || hour <= 5 || isWeekend) ? 1.5 : 0.8;

      const voltage = 220 + Math.random() * 10 - 5; // 215 - 225 V
      const current = (deviceProfile.baseCurrent + Math.random() * deviceProfile.currentVar)
```

```
      const powerFactor = 0.9 + Math.random() * 0.09;
      const power = voltage * current * powerFactor;

      accumulatedEnergy += (power / 1000);

      logs.push({
        deviceId,
        timestamp,
        voltage,
        current,
        power,
        energyKWh: accumulatedEnergy,
        powerFactor,
      });
    }
  }

  console.log(`Generated data for device: ${deviceId}`);
}

await PowerLog.insertMany(logs);
console.log(`Successfully populated database with ${logs.length} historical logs for ${devic
```

} catch (error) { console.error(‘Error populating initial data:’, error.message); } };

module.exports = { populate60DaysData }; “‘

File index.js

Path: relPath:=

“‘javascript // index.js

const express = require(‘express’); const dotenv = require(‘dotenv’); const http = require(‘http’); const { WebSocketServer } = require(‘ws’); const url = require(‘url’); const jwt = require(‘jsonwebtoken’);

const connectDB = require(‘./config/db’); const { connectMqtt } = require(‘./services/mqtt_service’); // const { startRealtimeSimulation } = require(‘./services/realtime_service’); // Sudah tidak dipakai const { startScheduler } = require(‘./services/scheduler_service’); const { notFound, errorHandler } = require(‘./middleware/errorMiddleware’);

const powerLogRoutes = require(‘./routes/powerLogRoutes’); const deviceRoutes = require(‘./routes/deviceRoutes’); const userRoutes = require(‘./routes/userRoutes’); const roomRoutes = require(‘./routes/roomRoutes’); const scheduleRoutes = require(‘./routes/scheduleRoutes’);

dotenv.config();

const app = express(); app.use(express.json());

const server = http.createServer(app); const wss = new WebSocketServer({ server });

const clientConnections = new Map();

wss.on(‘connection’, (ws, req) => { const token = url.parse(req.url, true).query.token; if (!token) { console.log(‘[WebSocket] Koneksi ditolak: Tidak ada token.’); return ws.terminate(); }

try { const decoded = jwt.verify(token, process.env.JWT_SECRET); const userId = decoded.id; console.log(`[WebSocket] Klien terhubung untuk user: ${userId}`); clientConnections.set(userId, ws);

```
ws.on('close', () => {
  console.log(`[WebSocket] Klien terputus untuk user: ${userId}`);
  clientConnections.delete(userId);
});
ws.on('error', (error) => {
  console.error(`[WebSocket] Error untuk user ${userId}:`, error);
  clientConnections.delete(userId);
});
```

} catch (error) { console.log(‘[WebSocket] Koneksi ditolak: Token tidak valid.’); ws.terminate(); } });

app.get(‘/api’, (req, res) => res.send(‘API sedang berjalan...’));

// Daftarkan semua rute app.use(‘/api/users’, userRoutes); app.use(‘/api/devices’, deviceRoutes); app.use(‘/api/rooms’, roomRoutes); app.use(‘/api/logs’, powerLogRoutes); app.use(‘/api/schedules’, scheduleRoutes);

// Error Middleware app.use(notFound); app.use(errorHandler);

const PORT = process.env.PORT || 5000;

const startServer = async () => { try { await connectDB(); // Berikan clientConnections ke service MQTT agar bisa meneruskan data connectMqtt(clientConnections);

```
server.listen(PORT, () => console.log(`[Server] Berjalan di port ${PORT}`));

// Jalankan service latar belakang
startScheduler(clientConnections);
```

} catch (error) { console.error(‘[Server] Gagal memulai server:’, error); process.exit(1); } };

startServer(); “‘

File realtime_service.js

Path: relPath:=

“‘javascript const PowerLog = require(‘./models/PowerLog’); const Device = require(‘./models/Device’); // <– Impor model Device const { WebSocket } = require(‘ws’);

/** * Memulai simulasi yang menghasilkan data baru setiap 3 detik * dan hanya untuk perangkat yang statusnya aktif. * @param {WebSocketServer} wss Instance WebSocket Server untuk menyiarkan data. */ const startRealtimeSimulation = (wss) => { console.log(‘[Simulation] Memulai simulasi real-time...’);

const deviceIds = [‘digiplug001’, ‘lampu’]; let currentDeviceIndex = 0;

setInterval(async () => { try { const deviceId = deviceIds[currentDeviceIndex];

```
// --- VALIDASI ON/OFF ---
// 1. Cek status perangkat di database
const device = await Device.findOne({ deviceId: deviceId });

// 2. Jika perangkat tidak ada atau statusnya 'active: false', lewati iterasi ini
if (!device || !device.active) {
  console.log(`[Simulation] Perangkat ${deviceId} sedang OFF. Melewatkan pengiriman data.`
  // Pindah ke perangkat berikutnya
  currentDeviceIndex = (currentDeviceIndex + 1) % deviceIds.length;
  return;
}

console.log(`[Simulation] Perangkat ${deviceId} sedang ON. Menghasilkan data...`);
// --- Lanjutan Logika Simulasi ---

const random = Math.random;
const voltage = 220 + random() * 4 - 2;
const powerFactor = 0.95 + random() * 0.04 - 0.02;

let current, power;
if (deviceId === 'digiplug_kulkas_01') {
  current = 1.0 + random() * 0.5;
  power = voltage * current * powerFactor;
} else {
  current = 0.1 + random() * 0.1;
  power = voltage * current * powerFactor;
}

const lastLog = await PowerLog.findOne({ deviceId: deviceId }).sort({ timestamp: -1 });
const lastEnergy = lastLog ? lastLog.energyKWh : 0;
const newEnergy = lastEnergy + (power / 1000) * (3 / 3600);
```

```
  const newLogData = new PowerLog({
    deviceId: deviceId,
    timestamp: new Date(),
    voltage, current, power, energyKWh: newEnergy, powerFactor,
  });

  const createdLog = await newLogData.save();
  const payload = JSON.stringify(createdLog);

  // Siarkan ke semua klien
  wss.clients.forEach((client) => {
    if (client.readyState === WebSocket.OPEN) {
      client.send(payload);
    }
  });
  console.log(`[WebSocket] Berhasil mengirim data untuk ${deviceId}: ${power.toFixed(2)} W`)

  // Pindah ke perangkat berikutnya
  currentDeviceIndex = (currentDeviceIndex + 1) % deviceIds.length;

} catch (error) {
  console.error('[Simulation] Error:', error.message);
}
```

}, 3000); };

module.exports = { startRealtimeSimulation }; “‘

File simulation_service.js

Path: relPath:=

“‘javascript const PowerLog = require(‘./models/PowerLog’); const { WebSocket } = require(‘ws’);

// — Fungsi Baru untuk Mengisi Data Historis — const populateInitialData = async () => { try { const logCount = await PowerLog.countDocuments(); if (logCount > 0) { console.log(‘Historical data already exists. Skipping population.’); return; }

```
console.log('No historical data found. Populating a 60-day history...');
const now = new Date();
const logs = [];
const random = Math.random;

// Generate data untuk 60 hari terakhir
for (let i = 60; i >= 0; i--) {
```

```
  // Buat beberapa log data per hari untuk membuatnya lebih variatif
  for (let j = 0; j < 8; j++) {
    const timestamp = new Date(now.getTime() - (i * 24 * 60 * 60 * 1000) - (j * 3 * 60 * 60

    const isWeekend = timestamp.getDay() === 0 || timestamp.getDay() === 6;

    // Simulasi pemakaian lebih tinggi di malam hari dan akhir pekan
    const hour = timestamp.getHours();
    const usageMultiplier = (hour >= 18 || hour <= 6 || isWeekend) ? 1.5 : 1;

    const voltage = 220 + random() * 10 - 5;
    const current = (1.0 + random()) * usageMultiplier;
    const powerFactor = 0.9 + random() * 0.09;
    const power = voltage * current * powerFactor;
    const energyKWh = (power / 1000) * (j * 3); // Simulasi akumulasi per 3 jam

    const newLog = new PowerLog({
      deviceId: 'digiplug_001',
      timestamp: timestamp,
      voltage: voltage,
      current: current,
      power: power,
      energyKWh: energyKWh,
      powerFactor: powerFactor,
    });
    logs.push(newLog);
  }
}

await PowerLog.insertMany(logs);
console.log(`Successfully populated database with ${logs.length} historical logs.`);
```

} catch (error) { console.error(‘Error populating initial data:’, error.message); } };

// — Fungsi yang Sudah Ada, Diperbarui Sedikit — const startRealtimeSimulation = (wss) => { console.log(‘Starting Digi-Plug real-time simulation...’);

setInterval(async () => { try { const random = Math.random; const voltage = 220 + random() * 4 - 2; const current = 1.2 + random() * 0.4 - 0.2; const powerFactor = 0.95 + random() * 0.04 - 0.02; const power = voltage * current * powerFactor;

```
  const lastLog = await PowerLog.findOne({ deviceId: 'digiplug_001' }).sort({ timestamp: -1
  const lastEnergy = lastLog ? lastLog.energyKWh : 0;
  const newEnergy = lastEnergy + (power / 1000) * (3 / 3600);

  const newLogData = new PowerLog({
```

```
    deviceId: 'digiplug_001',
    timestamp: new Date(),
    voltage: voltage,
    current: current,
    power: power,
    energyKWh: newEnergy,
    powerFactor: powerFactor,
  });

  const createdLog = await newLogData.save();
  console.log('New real-time log saved:', createdLog.power.toFixed(2), 'W');

  const payload = JSON.stringify(createdLog);
  wss.clients.forEach((client) => {
    if (client.readyState === WebSocket.OPEN) {
      client.send(payload);
    }
  });

} catch (error) {
  console.error('Real-time simulation error:', error.message);
}

}, 3000); };

module.exports = { populateInitialData, startRealtimeSimulation }; "'

. ## File db.js Path: relPath:= . "'javascript const mongoose = require('mongoose');

const connectDB = async () => { try { const conn = await mongoose.connect(process.env.MONGO_URI); console.log(`MongoDB Connected: ${conn.connection.host}`); } catch (error) { console.error(`Error: ${error.message}`); process.exit(1); // Keluar dari proses jika koneksi gagal } };

module.exports = connectDB;"' . . ## File deviceController.js Path: relPath:= . "'javascript // controllers/deviceController.js

const Device = require('../models/Device'); const Schedule = require('../models/Schedule'); const asyncHandler = require('../middleware/asyncHandler'); const { publishMqttMessage } = require('../services/mqtt_service');

// — FUNGSI BARU UNTUK KLAIM PERANGKAT — // @desc Claim a new device and assign it to the user // @route POST /api/devices/claim // @access Private const claimDevice = asyncHandler(async (req, res) => { const { deviceId, secretKey } = req.body; if (!deviceId || !secretKey) { res.status(400); throw new Error('deviceId dan secretKey diperlukan.'); }

// --- Validasi di Dunia Nyata ---
```

```
// Di sini, Anda akan memvalidasi secretKey ke database registri pabrik.
// Untuk TA, kita asumsikan kuncinya benar dan hanya cek duplikasi.
// PENTING: Pastikan secretKey yang dikirim sama dengan yang di-flash ke firmware.

const existingDevice = await Device.findOne({ deviceId });
if (existingDevice) {
    res.status(400);
    throw new Error('Perangkat ini sudah terdaftar di akun lain.');
}

const newDevice = new Device({
    owner: req.user._id,
    deviceId: deviceId,
    name: `DigiPlug ${deviceId.slice(-4)}`, // Nama default
    type: 'plug',
});

const createdDevice = await newDevice.save();
console.log(`[API] Perangkat ${deviceId} berhasil diklaim oleh user ${req.user._id}`);
res.status(201).json(createdDevice);
```

});

// @desc Update a device owned by the user const updateDevice = asyncHandler(async (req, res) => { const device = await Device.findById(req.params.id); if (device && device.owner.toString() === req.user._id.toString()) { const wasActive = device.active; Object.assign(device, req.body); const updatedDevice = await device.save(); if (wasActive !== updatedDevice.active) { const topic = `digihome/devices/${updatedDevice.deviceId}/command`; const message = { action: "SET_STATUS", payload: updatedDevice.active ? "ON" : "OFF" }; publishMqttMessage(topic, message); } res.json(updatedDevice); } else { res.status(404); throw new Error('Perangkat tidak ditemukan atau Anda tidak berwenang'); } });

// @desc Delete a device and trigger factory reset const deleteDevice = asyncHandler(async (req, res) => { const device = await Device.findById(req.params.id); if (device && device.owner.toString() === req.user._id.toString()) { const topic = `digihome/devices/${device.deviceId}/command`; publishMqttMessage(topic, { action: "FACTORY_RESET" }); await device.deleteOne(); await Schedule.deleteMany({ deviceId: device.deviceId, owner: req.user._id }); res.json({ message: 'Perangkat berhasil dihapus dari akun dan direset.' }); } else { res.status(404); throw new Error('Perangkat tidak ditemukan atau Anda tidak berwenang'); } });

const getDevices = asyncHandler(async (req, res) => { const devices = await Device.find({ owner: req.user._id }); res.json(devices); });

// Fungsi addDevice yang lama sudah tidak relevan karena sekarang menggu-

nakan claimDevice // Namun kita biarkan untuk potensi penggunaan lain atau debugging. const addDevice = asyncHandler(async (req, res) => { const { deviceId, name, type } = req.body; const newDevice = new Device({ deviceId, name, type, owner: req.user._id }); const createdDevice = await newDevice.save(); res.status(201).json(createdDevice); });

module.exports = { getDevices, claimDevice, updateDevice, deleteDevice, addDevice }; “‘ . . ## File powerLogController.js *Path: relPath:=* . “‘javascript const PowerLog = require(‘../models/PowerLog’);

// @desc Ambil semua log data, bisa difilter berdasarkan deviceId // @route GET /api/logs // @route GET /api/logs?deviceId=xxxxx const getPowerLogs = async (req, res) => { try { let query = {};

```
// Jika ada parameter deviceId di URL, tambahkan ke filter query
if (req.query.deviceId) {
  query.deviceId = req.query.deviceId;
}

// Ambil data dari MongoDB, urutkan dari yang terbaru, batasi 1000 data terakhir
const logs = await PowerLog.find(query).sort({ timestamp: -1 }).limit(1000);

res.json(logs);
```

} catch (error) { console.error(`Error fetching logs: ${error.message}`); res.status(500).json({ message: ‘Server Error’ }); } };

module.exports = { getPowerLogs }; “‘ . . ## File roomController.js *Path: relPath:=* . “‘javascript const Room = require(‘../models/Room’); const Device = require(‘../models/Device’);

// @desc Get all rooms for a logged-in user // @route GET /api/rooms const getRooms = async (req, res) => { try { const rooms = await Room.find({ owner: req.user._id }); res.json(rooms); } catch (error) { res.status(500).json({ message: ‘Server Error’ }); } };

// @desc Add a new room for a logged-in user // @route POST /api/rooms const addRoom = async (req, res) => { const { name } = req.body;

if (!name) { return res.status(400).json({ message: ‘Nama ruangan tidak boleh kosong’ }); }

try { const roomExists = await Room.findOne({ owner: req.user._id, name }); if (roomExists) { return res.status(400).json({ message: ‘Nama ruangan sudah ada’ }); }

```
const room = new Room({
  name,
  owner: req.user._id,
});
```

```
const createdRoom = await room.save();
res.status(201).json(createdRoom);
```

} catch (error) { res.status(500).json({ message: ‘Server Error’ }); } };

// @desc Delete a room owned by the user // @route DELETE /api/rooms/:id const deleteRoom = async (req, res) => { try { const room = await Room.findById(req.params.id);

```
// Cek kepemilikan ruangan
if (!room || room.owner.toString() !== req.user._id.toString()) {
  return res.status(404).json({ message: 'Ruangan tidak ditemukan' });
}

// Cek apakah masih ada perangkat di dalam ruangan ini
const devicesInRoom = await Device.countDocuments({ owner: req.user._id, room: room.name });
if (devicesInRoom > 0) {
  return res.status(400).json({ message: 'Tidak bisa menghapus ruangan yang masih berisi per
}

await room.deleteOne();
res.json({ message: 'Ruangan berhasil dihapus' });
```

} catch (error) { res.status(500).json({ message: ‘Server Error’ }); } };

module.exports = { getRooms, addRoom, deleteRoom }; “‘ . . ## File scheduleController.js *Path: relPath:=* . “‘javascript // controllers/scheduleController.js

const Schedule = require(‘../models/Schedule’); const Device = require(‘../models/Device’); const asyncHandler = require(‘../middleware/asyncHandler’);

// @desc Get all schedules for the logged-in user // @route GET /api/schedules // @access Private const getAllUserSchedules = asyncHandler(async (req, res) => { const schedules = await Schedule.find({ owner: req.user._id }).sort({ createdAt: -1, }); res.json(schedules); });

// @desc Membuat jadwal baru untuk sebuah perangkat // @route POST /api/schedules // @access Private const createSchedule = asyncHandler(async (req, res) => { const { deviceId, scheduleName, startTime, endTime, days, action, isEnabled, } = req.body;

if (!deviceId || !scheduleName || !startTime || !endTime || !days || !action) { res.status(400); throw new Error(‘Mohon lengkapi semua field yang diperlukan.’); }

// — PERBAIKAN KRUSIAL: Gunakan field ‘deviceId’ untuk query — const device = await Device.findOne({ deviceId: deviceId, // Query berdasarkan field ‘deviceId’ yang benar owner: req.user._id, });

if (!device) { res.status(404); throw new Error(‘Perangkat tidak ditemukan atau Anda tidak berwenang.’); }

const schedule = new Schedule({ owner: req.user._id, deviceId: device.deviceId, // Pastikan kita menyimpan deviceId yang sama scheduleName, startTime, endTime, days, action, isEnabled, });

const createdSchedule = await schedule.save(); res.status(201).json(createdSchedule); });

// @desc Mengambil semua jadwal untuk satu perangkat // @route GET /api/schedules/device/:deviceId // @access Private const getSchedulesForDevice = asyncHandler(async (req, res) => { const schedules = await Schedule.find({ owner: req.user._id, deviceId: req.params.deviceId, }); res.json(schedules); });

// @desc Memperbarui sebuah jadwal // @route PUT /api/schedules/:id // @access Private const updateSchedule = asyncHandler(async (req, res) => { const schedule = await Schedule.findById(req.params.id);

if (schedule && schedule.owner.toString() === req.user._id.toString()) { schedule.scheduleName = req.body.scheduleName || schedule.scheduleName; schedule.startTime = req.body.startTime || schedule.startTime; schedule.endTime = req.body.endTime || schedule.endTime; schedule.days = req.body.days || schedule.days; schedule.action = req.body.action || schedule.action; schedule.isEnabled = req.body.isEnabled ?? schedule.isEnabled;

```
const updatedSchedule = await schedule.save();
res.json(updatedSchedule);
```

} else { res.status(404); throw new Error('Jadwal tidak ditemukan atau tidak berwenang.'); } });

// @desc Menghapus sebuah jadwal // @route DELETE /api/schedules/:id // @access Private const deleteSchedule = asyncHandler(async (req, res) => { const schedule = await Schedule.findById(req.params.id);

if (schedule && schedule.owner.toString() === req.user._id.toString()) { await schedule.deleteOne(); res.json({ message: 'Jadwal berhasil dihapus.' }); } else { res.status(404); throw new Error('Jadwal tidak ditemukan atau tidak berwenang.'); } });

module.exports = { getAllUserSchedules, createSchedule, getSchedulesForDevice, updateSchedule, deleteSchedule, }; "' . . ## File userController.js *Path: relPath:=* . "'javascript const User = require('../models/User'); const generateToken = require('../utils/generateToken');

// @desc Mendaftarkan pengguna baru // @route POST /api/users/register const registerUser = async (req, res) => { const { name, email, password } = req.body;

try { const userExists = await User.findOne({ email }); if (userExists) { return res.status(400).json({ message: 'Email sudah terdaftar' }); }

```
const user = await User.create({
  name,
```

```
  email,
  password, // Password akan di-hash secara otomatis oleh pre-save hook di model
});

if (user) {
  res.status(201).json({
    _id: user._id,
    name: user.name,
    email: user.email,
    token: generateToken(user._id),
  });
} else {
  res.status(400).json({ message: 'Data pengguna tidak valid' });
}
```

} catch (error) { res.status(500).json({ message: ‘Server Error’ }); } };

// @desc Login pengguna & mendapatkan token // @route POST /api/users/login const loginUser = async (req, res) => { const { email, password } = req.body;

try { const user = await User.findOne({ email });

```
// Cek apakah pengguna ada DAN password-nya cocok
if (user && (await user.matchPassword(password))) {
  res.json({
    _id: user._id,
    name: user.name,
    email: user.email,
    token: generateToken(user._id),
  });
} else {
  res.status(401).json({ message: 'Email atau password salah' });
}
```

} catch (error) { res.status(500).json({ message: ‘Server Error’ }); } };

module.exports = { registerUser, loginUser }; “‘ . . ## File asyncHandler.js *Path: relPath:=* . “‘javascript // middleware/asyncHandler.js

/** * Wrapper untuk fungsi async route handler. * Menangkap error dan meneruskannya ke error handler Express. * Ini menghilangkan kebutuhan blok try-catch di setiap controller. * @param {Function} fn - Fungsi controller async yang akan dieksekusi. */ const asyncHandler = (fn) => (req, res, next) => Promise.resolve(fn(req, res, next)).catch(next);

module.exports = asyncHandler; “‘ . . ## File authMiddleware.js *Path: relPath:=* . “‘javascript const jwt = require(‘jsonwebtoken’); const User = require(‘../models/User.js’);

const protect = async (req, res, next) => { let token;

// Cek jika header Authorization ada dan dimulai dengan ‘Bearer’ if (req.headers.authorization && req.headers.authorization.startsWith(‘Bearer’)) { try { // 1. Ambil token dari header (Contoh: “Bearer ”, kita hanya ambil bagian tokennya) token = req.headers.authorization.split(’ ’)[1];

```
  // 2. Verifikasi keaslian token menggunakan secret key kita
  const decoded = jwt.verify(token, process.env.JWT_SECRET);

  // 3. Jika token valid, ambil data pengguna dari database berdasarkan ID di dalam token
  //    Kita tidak menyertakan password saat mengambil data (`.select('-password')`)
  req.user = await User.findById(decoded.id).select('-password');

  // 4. Lanjutkan ke fungsi controller selanjutnya (misalnya, getDevices)
  next();
} catch (error) {
  console.error(error);
  res.status(401).json({ message: 'Tidak terotorisasi, token gagal' });
}
```

}

if (!token) { res.status(401).json({ message: ‘Tidak terotorisasi, tidak ada token’ }); } };

module.exports = { protect }; “‘ . . ## File errorMiddleware.js *Path: relPath:=* . “‘javascript // middleware/errorMiddleware.js

/** * Middleware untuk menangani Not Found (404) errors. * Ini akan berjalan jika tidak ada route handler lain yang cocok. */ const notFound = (req, res, next) => { const error = new Error(`Not Found - ${req.originalUrl}`); res.status(404); next(error); };

/** * Middleware error handler terpusat. * Ini akan menangkap semua error yang dilempar di dalam aplikasi. * Memastikan semua response error dikirim dalam format JSON. */ const errorHandler = (err, req, res, next) => { // Terkadang error datang dengan statusCode 200, kita ubah ke 500 jika begitu let statusCode = res.statusCode === 200 ? 500 : res.statusCode; let message = err.message;

```
// Khusus untuk Mongoose CastError (e.g., ObjectId tidak valid)
if (err.name === 'CastError' && err.kind === 'ObjectId') {
  statusCode = 404;
  message = 'Resource not found';
}

// Khusus untuk Mongoose Duplicate Key Error
if (err.code === 11000) {
  statusCode = 400; // Bad Request
```

```
  const field = Object.keys(err.keyValue);
  message = `Duplicate field value entered for: ${field}. Please use another value.`;
}

res.status(statusCode).json({
  message: message,
  // Hanya tampilkan stack trace jika kita tidak dalam mode production
  stack: process.env.NODE_ENV === 'production' ? null : err.stack,
});
```

};

module.exports = { notFound, errorHandler }; “‘ . . ## File Device.js *Path: relPath:=* . “‘javascript // models/Device.js

const mongoose = require(‘mongoose’); // Hapus dependensi ke mqtt_service untuk memutus siklus // const { publishMqttMessage } = require(‘../services/mqtt_service’);

const deviceSchema = new mongoose.Schema({ owner: { type: mongoose.Schema.Types.ObjectId, required: true, ref: ‘User’ }, deviceId: { type: String, required: true }, name: { type: String, required: true }, type: { type: String, required: true }, room: { type: String, default: ‘Unassigned’ }, active: { type: Boolean, default: false }, isFavorite: { type: Boolean, default: false }, attributes: { type: mongoose.Schema.Types.Mixed, default: {} }, }, { timestamps: true, indexes: [{ fields: { owner: 1, deviceId: 1 }, unique: true }], });

// — PERBAIKAN: Hapus hook ‘pre’ atau ‘post’ dari sini — // Logika pengiriman MQTT akan dipindahkan ke controller. // Ini membuat model lebih bersih dan fokus pada struktur data saja.

const Device = mongoose.model(‘Device’, deviceSchema); module.exports = Device; “‘ . . ## File PowerLog.js *Path: relPath:=* . “‘javascript const mongoose = require(‘mongoose’);

const powerLogSchema = new mongoose.Schema({ deviceId: { type: String, required: true, index: true }, timestamp: { type: Date, default: Date.now, index: true }, voltage: { type: Number, required: true }, current: { type: Number, required: true }, power: { type: Number, required: true }, energyKWh: { type: Number, required: true }, powerFactor: { type: Number, required: true }, });

const PowerLog = mongoose.model(‘PowerLog’, powerLogSchema); module.exports = PowerLog; “‘ . . ## File Room.js *Path: relPath:=* . “‘javascript const mongoose = require(‘mongoose’);

const roomSchema = new mongoose.Schema({ // Menambahkan referensi ke model User owner: { type: mongoose.Schema.Types.ObjectId, required: true, ref: ‘User’, }, name: { type: String, required: true, }, // deviceCount tidak perlu disimpan di DB, karena bisa dihitung secara dinamis // dari jumlah perangkat yang memiliki nama ruangan ini. }, { timestamps: true, // Membuat index

gabungan untuk memastikan nama ruangan unik per pengguna indexes: [{ fields: { owner: 1, name: 1 }, unique: true }] });

const Room = mongoose.model('Room', roomSchema); module.exports = Room; "' . . ## File Schedule.js *Path: relPath:=* . "'javascript // models/Schedule.js

const mongoose = require('mongoose');

const scheduleSchema = new mongoose.Schema({ owner: { type: mongoose.Schema.Types.ObjectId, required: true, ref: 'User', }, deviceId: { type: String, // Kita gunakan deviceId unik dari model Device required: true, }, scheduleName: { type: String, required: true, }, startTime: { type: String, // Format "HH:mm" e.g., "08:00" required: true, }, endTime: { type: String, // Format "HH:mm" e.g., "22:30" required: true, }, days: { type: [String], // e.g., ["Sen", "Sel", "Rab"] required: true, }, action: { type: String, // "ON" or "OFF" required: true, enum: ['ON', 'OFF'], }, isEnabled: { type: Boolean, default: true, }, }, { timestamps: true, indexes: [// Index untuk memastikan nama jadwal unik per perangkat milik satu user { fields: { owner: 1, deviceId: 1, scheduleName: 1 }, unique: true },], });

const Schedule = mongoose.model('Schedule', scheduleSchema);

module.exports = Schedule; "' . . ## File User.js *Path: relPath:=* . "'javascript const mongoose = require('mongoose'); const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({ name: { type: String, required: true }, email: { type: String, required: true, unique: true }, password: { type: String, required: true }, }, { timestamps: true });

// Middleware yang berjalan SEBELUM data disimpan (.pre('save', ...)) userSchema.pre('save', async function (next) { // Hanya lakukan hashing jika password diubah (atau baru) if (!this.isModified('password')) { next(); }

// Generate "salt" untuk memperkuat hash, lalu hash password-nya const salt = await bcrypt.genSalt(10); this.password = await bcrypt.hash(this.password, salt); });

// Method untuk membandingkan password yang dimasukkan dengan hash di database userSchema.methods.matchPassword = async function (enteredPassword) { return await bcrypt.compare(enteredPassword, this.password); };

const User = mongoose.model('User', userSchema); module.exports = User; "' . . ## File deviceRoutes.js *Path: relPath:=* . "'javascript // routes/deviceRoutes.js

const express = require('express'); const router = express.Router(); const { getDevices, claimDevice, updateDevice, deleteDevice, addDevice // Tetap ekspor untuk kompatibilitas } = require('../controllers/deviceController'); const { protect } = require('../middleware/authMiddleware');

router.route('/') .get(protect, getDevices) .post(protect, addDevice); // Route POST lama tetap ada

router.route(‘/claim’).post(protect, claimDevice); // Route baru untuk klaim

router.route(‘/:id’) .put(protect, updateDevice) .delete(protect, deleteDevice);

module.exports = router; “‘ . . ## File powerLogRoutes.js *Path: relPath:=* . “‘javascript const express = require(‘express’); const router = express.Router(); const { getPowerLogs } = require(‘../controllers/powerLogController’);

// Definisikan rute untuk endpoint /api/logs // Saat URL ini diakses dengan metode GET, ia akan menjalankan fungsi getPowerLogs router.route(‘/’).get(getPowerLogs);

module.exports = router; “‘ . . ## File roomRoutes.js *Path: relPath:=* . “‘javascript const express = require(‘express’); const router = express.Router(); const { getRooms, addRoom, deleteRoom } = require(‘../controllers/roomController’); const { protect } = require(‘../middleware/authMiddleware’); // <– Impor middleware

// PERBAIKAN: Terapkan middleware ‘protect’ pada semua rute router.route(‘/’).get(protect, getRooms).post(protect, addRoom); router.route(‘/:id’).delete(protect, deleteRoom);

module.exports = router; “‘ . . ## File scheduleRoutes.js *Path: relPath:=* . “‘javascript // routes/scheduleRoutes.js

const express = require(‘express’); const router = express.Router(); const { getAllUserSchedules, // <– IMPORT BARU createSchedule, getSchedulesForDevice, updateSchedule, deleteSchedule, } = require(‘../controllers/scheduleController’); const { protect } = require(‘../middleware/authMiddleware’);

// — PERBAIKAN: Gabungkan GET dan POST untuk root route — router .route(‘/’) .get(protect, getAllUserSchedules) // <– ROUTE BARU .post(protect, createSchedule);

router .route(‘/:id’) .put(protect, updateSchedule) .delete(protect, deleteSchedule);

router.route(‘/device/:deviceId’).get(protect, getSchedulesForDevice);

module.exports = router; “‘ . . ## File userRoutes.js *Path: relPath:=* . “‘javascript const express = require(‘express’); const router = express.Router(); const { registerUser, loginUser } = require(‘../controllers/userController’);

router.post(‘/register’, registerUser); router.post(‘/login’, loginUser);

module.exports = router; “‘ . . ## File mqtt_service.js *Path: relPath:=* . “‘javascript // services/mqtt_service.js

const mqtt = require(‘mqtt’); const PowerLog = require(‘../models/PowerLog’); const Device = require(‘../models/Device’); const { WebSocket } = require(‘ws’);

let client = null;

/** * Menghubungkan ke MQTT broker dan mengatur listener. * @param {Map<string, WebSocket>} clientConnections Peta koneksi WebSocket pengguna. */ const connectMqtt = (clientConnections) => { let brokerUrl = process.env.MQTT_BROKER_URL; if (!brokerUrl) { console.error("[MQTT] Error: MQTT_BROKER_URL tidak terdefinisi."); return; } if (!/^(mqtt|mqtts|ws|wss):///.test(brokerUrl)) { brokerUrl = `mqtt://${brokerUrl}`; }

const options = { clientId: `digihome_backend_${Math.random().toString(16).slice(2, 8)}`, username: process.env.MQTT_USERNAME, password: process.env.MQTT_PASSWORD, };

console.log(`[MQTT] Menghubungkan ke broker di ${brokerUrl}...`); client = mqtt.connect(brokerUrl, options);

client.on('connect', () => { console.log('[MQTT] Berhasil terhubung ke broker.'); const telemetryTopic = 'digihome/devices/+/telemetry'; client.subscribe(telemetryTopic, (err) => { if (!err) { console.log(`[MQTT] Berlangganan ke topik: ${telemetryTopic}`); } }); });

client.on('message', async (topic, payload) => { try { if (!topic.includes('/telemetry')) return;

```
  const message = JSON.parse(payload.toString());
  const { deviceId } = message;

  if (!deviceId) return;

  const newLog = new PowerLog(message);
  const savedLog = await newLog.save();

  const device = await Device.findOne({ deviceId: deviceId });
  if (!device) return;

  const ownerSocket = clientConnections.get(device.owner.toString());
  if (ownerSocket && ownerSocket.readyState === WebSocket.OPEN) {
    ownerSocket.send(JSON.stringify(savedLog));
  }
} catch (error) {
  console.error('[MQTT] Gagal memproses pesan telemetri:', error.message);
}
```

});

client.on('error', (error) => console.error('[MQTT] Error koneksi:', error)); client.on('reconnect', () => console.log('[MQTT] Menyambung ulang...')); client.on('close', () => console.log('[MQTT] Koneksi MQTT ditutup.')); };

/** * Menerbitkan (publish) pesan ke topik MQTT. * @param {string} topic - Topik tujuan. * @param {object|string} message - Pesan yang akan dikirim.

*/ const publishMqttMessage = (topic, message) => { // — PERBAIKAN: Implementasi lengkap fungsi publish — if (client && client.connected) { const payload = typeof message === 'string' ? message : JSON.stringify(message); client.publish(topic, payload, (err) => { if (err) { console.error(`[MQTT] Gagal publish ke topik ${topic}:`, err); } else { console.log(`[MQTT] Publish ke ${topic}: ${payload}`); } }); } else { console.error('[MQTT] Tidak bisa publish. Klien tidak terhubung.'); } };

module.exports = { connectMqtt, publishMqttMessage }; "' . . ## File realtime_service.js *Path: relPath:=* . "'javascript // services/realtime_service.js

const PowerLog = require('../models/PowerLog'); const Device = require('../models/Device'); const { WebSocket } = require('ws');

/** * Memulai simulasi yang menghasilkan data baru setiap 3 detik * untuk SEMUA perangkat yang statusnya 'active: true' di database. * @param {Map<string, WebSocket>} clientConnections - Map dari userId ke koneksi WebSocket. */ const startRealtimeSimulation = (clientConnections) => { console.log('[Simulation] Memulai simulasi real-time berbasis database...');

setInterval(async () => { try { // 1. Ambil semua perangkat yang sedang aktif dari database const activeDevices = await Device.find({ active: true });

```
if (activeDevices.length === 0) {
  // console.log('[Simulation] Tidak ada perangkat aktif saat ini.');
  return;
}

console.log(`[Simulation] Found ${activeDevices.length} active device(s). Generating data.

// 2. Loop melalui setiap perangkat aktif dan hasilkan data
for (const device of activeDevices) {
  const logData = generateLogForDevice(device);
  const lastLog = await PowerLog.findOne({ deviceId: device.deviceId }).sort({ timestamp:
  const lastEnergy = lastLog ? lastLog.energyKWh : 0;

  // Interval adalah 3 detik
  const newEnergy = lastEnergy + (logData.power / 1000) * (3 / 3600);

  const newLog = new PowerLog({
    deviceId: device.deviceId,
    timestamp: new Date(),
    voltage: logData.voltage,
    current: logData.current,
    power: logData.power,
    energyKWh: newEnergy,
    powerFactor: logData.powerFactor,
  });
```

```
    const createdLog = await newLog.save();
    const payload = JSON.stringify(createdLog);

    // --- PEMBARUAN UTAMA: PENGIRIMAN BERTARGET ---
    const ownerId = device.owner.toString();
    const userSocket = clientConnections.get(ownerId);

    if (userSocket && userSocket.readyState === WebSocket.OPEN) {
      userSocket.send(payload);
      console.log(`[WebSocket] Sent data for ${device.deviceId} to user ${ownerId}`);
    }
  }
} catch (error) {
  console.error('[Simulation] Error:', error.message);
}
```

}, 3000); // Berjalan setiap 3 detik };

/** * Helper function untuk menghasilkan data log palsu berdasarkan tipe perangkat. * @param {object} device - Objek perangkat dari Mongoose. * @returns {object} - Objek berisi data telemetri yang disimulasikan. */ *function generateLogForDevice(device) { const random = Math.random; const voltage = 220 + random()* 10 - 5; // 215V - 225V

let baseCurrent = 0.5, currentVar = 0.2, powerFactor = 0.9;

// Berikan karakteristik berbeda untuk setiap tipe perangkat switch (device.type) { case 'AC': baseCurrent = 2.0; currentVar = 1.0; powerFactor = 0.85; break; case 'Kulkas': // Kulkas memiliki siklus on/off, kita simulasikan secara sederhana baseCurrent = (new Date().getMinutes() % 10 < 5) ? 0.8 : 0.1; currentVar = 0.3; break; case 'Smart TV': baseCurrent = 0.7; currentVar = 0.4; break; case 'Lampu': baseCurrent = 0.05; currentVar = 0.01; break; default: // Untuk DigiPlug dan lainnya baseCurrent = 0.5; currentVar = 1.0; // Beri variasi lebih besar }

const current = baseCurrent + random() * currentVar; const finalPowerFactor = powerFactor + random() * 0.09; const power = voltage * current * finalPowerFactor;

return { voltage, current, power, powerFactor: finalPowerFactor }; }

module.exports = { startRealtimeSimulation }; "' . . ## File scheduler_service.js *Path: relPath:*= . "'javascript // services/scheduler_service.js

const cron = require('node-cron'); const Schedule = require('../models/Schedule'); const Device = require('../models/Device'); const { WebSocket } = require('ws');

// Helper untuk memetakan nama hari ke angka (Minggu=0, Senin=1, ...) const dayMap = { 'Min': 0, 'Sen': 1, 'Sel': 2, 'Rab': 3, 'Kam': 4, 'Jum': 5, 'Sab':

6 };

/** * Memulai Scheduler Engine yang berjalan setiap menit. * @param {Map<string, WebSocket>} clientConnections - Map dari userId ke koneksi WebSocket. */ const startScheduler = (clientConnections) => { console.log(‘[Scheduler] Engine is running. Checking for tasks every minute.’);

// Jadwalkan tugas untuk berjalan setiap menit: ’* * * * * ’ *cron.schedule(’* * * * *’*, async () => { const now = new Date(); const currentDay = Object.keys(dayMap).find(key => dayMap[key] === now.getDay()); const currentTime = `${now.getHours().toString().padStart(2, '0')}:${now.getMinutes().toString().padStart(2, '0')}`;

```
console.log(`[Scheduler] Checking for tasks at ${currentTime} on ${currentDay}...`);

try {
  // Cari semua jadwal yang aktif, cocok dengan hari dan waktu saat ini
  const dueSchedules = await Schedule.find({
    isEnabled: true,
    days: currentDay,
    $or: [
      { startTime: currentTime },
      { endTime: currentTime }
    ]
  });

  if (dueSchedules.length === 0) {
    return; // Tidak ada tugas, selesai untuk menit ini.
  }

  console.log(`[Scheduler] Found ${dueSchedules.length} due tasks.`);

  for (const schedule of dueSchedules) {
    const targetState = schedule.startTime === currentTime ? schedule.action === 'ON' : sche

    const device = await Device.findOne({ deviceId: schedule.deviceId, owner: schedule.owner

    if (!device) {
      console.log(`[Scheduler] Device ${schedule.deviceId} not found for schedule ${schedule
      continue;
    }

    // Hanya eksekusi jika status perangkat berbeda dengan target
    if (device.active !== targetState) {
      device.active = targetState;
      await device.save();
      console.log(`[Scheduler] Executed: Set device ${device.name} to ${targetState ? 'ON'
```

```
      // Kirim pembaruan real-time ke pengguna yang tepat
      const ownerSocket = clientConnections.get(device.owner.toString());
      if (ownerSocket && ownerSocket.readyState === WebSocket.OPEN) {
        // Kita kirim pesan "telemetry" palsu agar UI bereaksi
        const payload = JSON.stringify({
          deviceId: device.deviceId,
          timestamp: new Date().toISOString(),
          power: targetState ? (Math.random() * 50 + 10) : 0, // Data daya acak
          voltage: 220,
          current: targetState ? (Math.random() * 0.5) : 0,
          energyKWh: 0, // Tidak relevan untuk update status
          powerFactor: 0.9
        });
        ownerSocket.send(payload);
        console.log(`[Scheduler] Notified user ${device.owner} about status change.`);
      }
    }
  }
} catch (error) {
  console.error('[Scheduler] Error during task execution:', error);
}

}); };
```

module.exports = { startScheduler }; “‘ . . ## File generateToken.js *Path: relPath:=* . “‘javascript const jwt = require(‘jsonwebtoken’);

const generateToken = (id) => { return jwt.sign({ id }, process.env.JWT_SECRET, { expiresIn: ‘30d’, // Token akan valid selama 30 hari }); };

module.exports = generateToken; “‘ .