

## 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 * 60 * 1000));

      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) * usageMultiplier;
```

```

    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 ${deviceId}`);
} 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 } = re-
quire('./services/mqtt_service'); // const { startRealtimeSimulation } =
require('./services/realtime_service'); // Sudah tidak dipakai const { startSched-
uler } = require('./services/scheduler_service'); const { notFound, errorHandler
} = require('./middleware/errorMiddleware');

const powerLogRoutes = require('./routes/powerLogRoutes'); const de-
viceRoutes = require('./routes/deviceRoutes'); const userRoutes = re-
quire('./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', power-
LogRoutes); 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 connect-
Mqtt(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); pro-
cess.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 startRealtimeSim-
ulation = (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 (log-
Count > 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 * 1000));

  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 asyncHandler = require('../middleware/asyncHandler');

// ===== PERBAIKAN UTAMA DI SINI =====
// Impor kembali 'publishMqttMessage' dari mqtt_service const { isDeviceConfirmed, confirmDeviceClaim, unclaimedDevices, publishMqttMessage } = require('../services/mqtt_service'); // =====

// FUNGSI BARU: Untuk mengecek status klaim const getClaimStatus = asyncHandler(async (req, res) => { let confirmedDeviceId = null; for (const [deviceId,

```

```

status] of unclaimedDevices.entries()) { if (status.confirmed) { confirmedDeviceId
= deviceId; break; } }

if (confirmedDeviceId) { res.status(200).json({ status: 'ready', deviceId: con-
firmedDeviceId }); } else { res.status(202).json({ status: 'waiting' }); } });

const claimDevice = asyncHandler(async (req, res) => { const { deviceId } =
req.body; if (!deviceId) { res.status(400); throw new Error('deviceId diperlukan.')}
}

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

if (!isDeviceConfirmed(deviceId)) { res.status(404); throw new Error('Perangkat
belum dikonfirmasi secara fisik. Tekan tombol pada perangkat.')} }

const newDevice = new Device({ owner: req.user._id, deviceId: deviceId, name:
DigiPlug `${deviceId.slice(-6)}`, type: 'plug', });

const createdDevice = await newDevice.save(); confirmDeviceClaim(deviceId);
res.status(201).json(createdDevice); });

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

const updateDevice = asyncHandler(async (req, res) => { const device = await
Device.findById(req.params.id);

if (device && device.owner.toString() === req.user.id.toString()) { const was-
Active = device.active;

// Perbarui nama, ruangan, atau status 'active' dari body request
device.name = req.body.name || device.name;
device.room = req.body.room || device.room;
if (typeof req.body.active === 'boolean') {
    device.active = req.body.active;
}

const updatedDevice = await device.save();

// ===== PERBAIKAN UTAMA DI SINI =====
// Jika status 'active' (ON/OFF) berubah, kirim perintah MQTT
// Pastikan kode ini tidak dikomentari lagi
if (wasActive !== updatedDevice.active) {
    const topic = `dighome/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'); } });

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 = `dighome/devices/${device.deviceId}/command`;
publishMqttMessage(topic, { action: "FACTORY_RESET" }); await de-
vice.deleteOne(); 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 setDeviceConfig =
asyncHandler(async (req, res) => { const { configKey, value } = req.body;
const device = await Device.findById(req.params.id);

if (device && device.owner.toString() === req.user.id.toString()) { // Validasi
input if (configKey === 'overcurrentThreshold' && typeof value === 'number'
&& value > 0) { // Update di database device.config.overcurrentThreshold =
value; await device.save();

// Kirim perintah ke perangkat via MQTT
const topic = `dighome/devices/${device.deviceId}/command`;
const message = {
  action: "SET_CONFIG",
  payload: { [configKey]: value }
};
publishMqttMessage(topic, message);

res.status(200).json({ message: `Konfigurasi ${configKey} berhasil diperbarui.` });
} else {
  res.status(400).send({ message: 'Kunci konfigurasi atau nilai tidak valid.' });
}

} else { res.status(404).send({ message: 'Perangkat tidak ditemukan atau Anda
tidak berwenang.' }); } });

// — FUNGSI BARU: Untuk memerintahkan perangkat masuk mode
pairing — const enterReProvisioningMode = asyncHandler(async (req,
res) => { const device = await Device.findById(req.params.id); if (device
&& device.owner.toString() === req.user.id.toString()) { const topic =
`dighome/devices/${device.deviceId}/command`; const message = { ac-
tion: "ENTER_PROVISIONING" }; publishMqttMessage(topic, message);
res.status(200).json({ message: 'Perintah re-provisioning terkirim.' }); } else {
res.status(404).send({ message: 'Perangkat tidak ditemukan atau Anda tidak
berwenang.' }); } });

module.exports = { getDevices, claimDevice, updateDevice, deleteDevice, get-
ClaimStatus, setDeviceConfig, enterReProvisioningMode, };“ . . ## File

```

```

powerLogController.js Path: relPath:= . “javascript const PowerLog = re-
quire('./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 schedule-
Controller.js Path: relPath: = . “javascript // controllers/scheduleController.js

const Schedule = require('../models/Schedule'); const Device = re-
quire('../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()) { sched-
ule.scheduleName = req.body.scheduleName || schedule.scheduleName; sched-
ule.startTime = req.body.startTime || schedule.startTime; schedule.endTime =
req.body.endTime || schedule.endTime; schedule.days = req.body.days || sched-
ule.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 berwe-
nang.')} });

module.exports = { getAllUserSchedules, createSchedule, getSchedulesForDe-
vice, updateSchedule, deleteSchedule, }; “ . . ## File userController.js Path:
relPath:= . “javascript const User = require('../models/User'); const generate-
Token = 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 const mongoose = require(‘mongoose’);

const deviceSchema = new mongoose.Schema( { owner: { type: mon-
goose.Schema.Types.ObjectId, required: true, ref: ‘User’ }, deviceId: { type:
String, required: true, unique: 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: {} },

// --- PENAMBAHAN BARU ---
isOnline: { type: Boolean, default: false },
wifiSsid: { type: String, default: ‘N/A’ },
wifiRssi: { type: Number, default: 0 },
config: {
  overcurrentThreshold: { type: Number, default: 5.0 }
}
// -----
}, { timestamps: true, } );

// Hapus index lama jika ada, dan pastikan deviceId unik // de-
viceSchema.index({ owner: 1, deviceId: 1 }, { unique: true }); // Ini
bagus, tapi deviceId sendiri harus unik di seluruh sistem

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); mod-
ule.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: mon-
goose.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', ...)) user-
Schema.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 (enteredPass-
word) { return await bcrypt.compare(enteredPassword, this.password); };

const User = mongoose.model('User', userSchema); module.exports = User; “
. . ## File deviceRoutes.js Path: relPath:= . “javascript const express =
require('express'); const router = express.Router();

const { getDevices, claimDevice, updateDevice, deleteDevice, getClaimStatus,
setDeviceConfig, // Impor fungsi baru enterReProvisioningMode, // Impor
fungsi baru } = require('../controllers/deviceController'); const { protect } =
require('../middleware/authMiddleware');

router.route('/').get(protect, getDevices); router.route('/claim-status').get(protect,
getClaimStatus); router.route('/claim').post(protect, claimDevice); router.route('/:id').put(protect,

```



```

updateDevice).delete(protect, deleteDevice);

// — PENAMBAHAN RUTE BARU — router.route('/:id/config').put(protect,
setDeviceConfig); router.route('/:id/re-provision').post(protect, enterReProvi-
sioningMode); // —————

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 diak-
ses 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 = ex-
press.Router(); const { getRooms, addRoom, deleteRoom } = re-
quire('./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 { getAl-
lUserSchedules, // <- 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, deleteSched-
ule);

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');

```

```

let client = null;

// — STATE MANAGEMENT UNTUK CLAIMING — // Struktur:
Map<deviceId, { confirmed: boolean }> const unclaimedDevices = new Map();
const unclaimedDeviceTimers = new Map();

// — LOGIKA BATCH PROCESSING UNTUK TELEMETRI — const telemetryBuffer = new Map(); const BATCH_INTERVAL_MS = 5 * 60 * 1000;

async function processTelemetryBuffer() { if (telemetryBuffer.size === 0) return; console.log([Batch] Memproses buffer telemetri untuk ${telemetryBuffer.size} perangkat...); const logsToInsert = []; for (const [deviceId, messages] of telemetryBuffer.entries()) { if (messages.length === 0) continue; const avgVoltage = messages.reduce((sum, msg) => sum + (msg.voltage || 0), 0) / messages.length; const avgCurrent = messages.reduce((sum, msg) => sum + (msg.current || 0), 0) / messages.length; const avgPower = messages.reduce((sum, msg) => sum + (msg.power || 0), 0) / messages.length; const avgPowerFactor = messages.reduce((sum, msg) => sum + (msg.powerFactor || 0), 0) / messages.length; const lastEnergyKWh = messages.length > 0 ? messages[messages.length - 1].energyKWh : 0; logsToInsert.push({ deviceId, timestamp: new Date(), voltage: avgVoltage, current: avgCurrent, power: avgPower, energyKWh: lastEnergyKWh, powerFactor: avgPowerFactor, }); } try { if (logsToInsert.length > 0) { await PowerLog.insertMany(logsToInsert); console.log([Batch] Berhasil menyimpan ${logsToInsert.length} log agregat.); } } catch (error) { console.error([Batch] Gagal menyimpan data batch:', error); } telemetryBuffer.clear(); }

setInterval(processTelemetryBuffer, BATCH_INTERVAL_MS);

// — FUNGSI HELPER YANG DIEKSPOR — const isDeviceConfirmed = (deviceId) => unclaimedDevices.has(deviceId) && unclaimedDevices.get(deviceId).confirmed === true; const confirmDeviceClaim = (deviceId) => { unclaimedDevices.delete(deviceId); if (unclaimedDeviceTimers.has(deviceId)) { clearTimeout(unclaimedDeviceTimers.get(deviceId)); unclaimedDeviceTimers.delete(deviceId); } console.log([Claim] Perangkat ${deviceId} berhasil diklaim.); };

// — FUNGSI KONEKSI UTAMA — const connectMqtt = (clientConnections) => { let brokerUrl = process.env.MQTT_BROKER_URL; if (!brokerUrl) { return console.error("[MQTT] Error: MQTT_BROKER_URL tidak terdefinisi."); } if (!/^(\mqtt|mqttps|ws|wss):\/\/.test(brokerUrl)) { brokerUrl = mqtt://${brokerUrl}; }

const options = { clientId: digihome_backend_${Math.random().toString(16).slice(2, 8)} }; client = mqtt.connect(brokerUrl, options);

client.on('connect', () => { console.log([MQTT] Berhasil terhubung ke broker.); client.subscribe('digihome/#', { qos: 1 }, (err) => { if (!err) console.log([MQTT] Berlangganan ke topik generik: digihome/#'); }); });

```

```

client.on('message', async (topic, payload) => { console.log([MQTT]
Pesan diterima di topik [${topic}]); let message; try { message =
JSON.parse(payload.toString()); } catch (error) { console.error([MQTT] Gagal
parse JSON dari topik ${topic}:, payload.toString()); return; }

// --- PERBAIKAN: Selalu gunakan deviceId dari payload untuk konsistensi ---
const { deviceId } = message;
if (!deviceId) {
  console.warn(`[MQTT] Menerima pesan tanpa deviceId di topik ${topic}`);
  return;
}

try {
  const topicType = topic.split('/')[1]; // 'provisioning', 'devices'

  if (topicType === 'provisioning') {
    if (topic.endsWith('/online')) {
      console.log(`[Provisioning] Perangkat ${deviceId} online, menunggu konfirmasi fisik.`);
      unclaimedDevices.set(deviceId, { confirmed: false });
      const timer = setTimeout(() => {
        if (unclaimedDevices.has(deviceId)) {
          unclaimedDevices.delete(deviceId);
          unclaimedDeviceTimers.delete(deviceId);
          console.log(`[Provisioning] Batas waktu klaim untuk ${deviceId} habis.`);
        }
      }, 5 * 60 * 1000);
      unclaimedDeviceTimers.set(deviceId, timer);
    } else if (topic.endsWith('/confirm')) {
      if (unclaimedDevices.has(deviceId)) {
        unclaimedDevices.set(deviceId, { confirmed: true });
        console.log(`[Provisioning] Konfirmasi fisik untuk ${deviceId} DITERIMA & STATUS DI...`);
        console.log(`[Provisioning] Status daftar tunggu saat ini:`, Array.from(unclaimedDev...));
      }
    }
  } else if (topicType === 'devices') {
    if (topic.endsWith('/status')) {
      await Device.findOneAndUpdate({ deviceId }, { isOnline: message.isOnline }, { new: true });
    } else if (topic.endsWith('/telemetry')) {
      const device = await Device.findOne({ deviceId });
      if (device) {
        // Update status WiFi
        await device.updateOne({ wifiSsid: message.wifiSsid, wifiRssi: message.wifiRssi, is... });
        // Teruskan ke WebSocket
        if (device.owner) {
          const ownerSocket = clientConnections.get(device.owner.toString());
          if (ownerSocket && ownerSocket.readyState === 1) {

```

```

        ownerSocket.send(payload.toString());
    }
}
}
// Simpan ke buffer
if (!telemetryBuffer.has(deviceId)) telemetryBuffer.set(deviceId, []);
telemetryBuffer.get(deviceId).push(message);
}
}
} catch (error) {
    console.error(`[MQTT] Gagal memproses pesan dari topik ${topic}:`, error.message);
}
});

client.on('error', (error) => console.error('[MQTT] Error koneksi:', error));

const publishMqttMessage = (topic, message) => { if (client &&
client.connected) { const payload = JSON.stringify(message); client.publish(topic,
payload); } else { console.error('[MQTT] Tidak bisa publish. Klien tidak
terhubung.');
```

module.exports = { connectMqtt, publishMqttMessage, isDeviceConfirmed, confirmDeviceClaim, unclaimedDevices }; “ . . ## 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 * finalPow-
erFactor;

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

module.exports = { startRealtimeSimulation }; ““ . . ## File sched-
uler_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) => { con-
sole.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.ownerId });

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

  // 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; “ .

```