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 notificationRoutes = require(‘./routes/notificationRoutes’);

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); app.use(‘/api/notifications’, notificationRoutes); // 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 asyncHandler = require('../middleware/asyncHandler');

// Impor fungsi-fungsi yang dibutuhkan dari service MQTT const { isDeviceConfirmed, confirmDeviceClaim, unclaimedDevices, publishMqttMessage, } = require('../services/mqtt_service');

// @desc Get claim status for a new device // @route GET /api/devices/claim-status // @access Private const getClaimStatus = asyncHandler(async (req, res) => { let confirmedDeviceId = null; for (const [deviceId, status] of unclaimedDe-
```

vices.entries()) { if (status.confirmed) { confirmedDeviceId = deviceId; break; } }

if (confirmedDeviceId) { res.status(200).json({ status: ‘ready’, deviceId: confirmedDeviceId }); } else { res.status(202).json({ status: ‘waiting’ }); } });

// @desc Claim a new device for a user // @route POST /api/devices/claim // @access Private 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); });

// @desc Get all devices for a logged-in user // @route GET /api/devices // @access Private const getDevices = asyncHandler(async (req, res) => { const devices = await Device.find({ owner: req.user.id }); res.json(devices); });

// @desc Update a device’s properties (name, room, favorite, active) // @route PUT /api/devices/:id // @access Private 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;

```
// Perbarui nama dan ruangan jika ada di body request
device.name = req.body.name || device.name;
device.room = req.body.room || device.room;

// ================================================================
// ================= SOLUSI MASALAH FAVORIT ADA DI SINI ==================
// Tambahkan blok ini untuk menangani pembaruan field 'isFavorite'.
// Kode ini akan memeriksa apakah ada data 'isFavorite' di dalam request,
// dan jika ada, akan memperbarui nilainya di database.
if (typeof req.body.isFavorite === 'boolean') {
  device.isFavorite = req.body.isFavorite;
}
// ================================================================
// ================================================================

// Perbarui status aktif (ON/OFF)
```

```
if (typeof req.body.active === 'boolean') {
  device.active = req.body.active;
}

const updatedDevice = await device.save();

// Jika status 'active' berubah, kirim perintah MQTT ke perangkat
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 // @route DELETE /api/devices/:id // @access Private 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(); res.json({ message: 'Perangkat berhasil dihapus dari akun dan direset.' }); } else { res.status(404); throw new Error('Perangkat tidak ditemukan atau Anda tidak berwenang'); } });

// @desc Update a device's specific configuration // @route PUT /api/devices/:id/config // @access Private 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()) { if (configKey === 'overcurrentThreshold' && typeof value === 'number' && value > 0) { device.config.overcurrentThreshold = value; await device.save();

```
  const topic = `digihome/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.’ }); } });

// @desc Command a device to enter re-provisioning mode // @route POST /api/devices/:id/re-provision // @access Private 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 = `digihome/devices/${device.deviceId}/command`; const message = { action: ‘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, getClaimStatus, setDeviceConfig, enterReProvisioningMode, }; “‘ . . ## File notificationController.js *Path: relPath:=* . “‘javascript // controllers/notificationController.js const Notification = require(‘../models/Notification’); const asyncHandler = require(‘../middleware/asyncHandler’);

/** * @desc Mengambil semua notifikasi untuk pengguna yang sedang login. * @route GET /api/notifications * @access Private */ const getUserNotifications = asyncHandler(async (req, res) => { // — BLOK DEBUGGING & PERBAIKAN — // Log 1: Verifikasi bahwa middleware ‘protect’ telah berjalan dengan benar. if (!req.user || !req.user._id) { console.error(‘[Debug] FATAL: req.user atau req.user._id tidak ditemukan. Middleware `protect` mungkin gagal atau token tidak valid.’); // Kirim respons error yang jelas jika pengguna tidak terautentikasi. res.status(401); throw new Error(‘User tidak terautentikasi dengan benar.’); }

console.log(`[Debug] Mencoba mengambil notifikasi untuk user ID: ${req.user._id}`);

try { // Log 2: Jalankan query ke database dengan user ID yang sudah divalidasi. console.log(`[Debug] Menjalankan query: Notification.find({ user: "${req.user._id}" })`); const notifications = await Notification.find({ user: req.user._id }).sort({ createdAt: -1 });

```
// Log 3: Laporkan hasilnya dan kirim respons sukses.
console.log(`[Debug] Ditemukan ${notifications.length} notifikasi untuk pengguna ${req.user.
res.status(200).json(notifications);
```

} catch (error) { // Log 4: Tangkap error spesifik dari database atau proses lainnya. console.error(`[Debug] Terjadi error saat query database untuk user ${req.user._id}:`, error); res.status(500); // Set status ke 500 Internal Server Error throw new Error('Terjadi kesalahan pada server saat mengambil notifikasi.'); } });

/** * @desc Menandai satu notifikasi sebagai sudah dibaca. * @route PUT /api/notifications/:id/read * @access Private */ const markAsRead = asyncHandler(async (req, res) => { // Pastikan req.user ada sebelum melanjutkan if (!req.user || !req.user._id) { res.status(401); throw new Error('User tidak terautentikasi.'); }

const notification = await Notification.findById(req.params.id);

// Pastikan notifikasi ada dan dimiliki oleh pengguna yang sedang login if (notification && notification.user.toString() === req.user._id.toString()) { notification.isRead = true; await notification.save(); res.json({ message: 'Notification marked as read' }); } else { res.status(404); throw new Error('Notification not found or you are not authorized'); } });

module.exports = { getUserNotifications, markAsRead }; "' . . ## 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 // controllers/userController.js const User = require(‘../models/User’); const generateToken = require(‘../utils/generateToken’); const asyncHandler = require(‘../middleware/asyncHandler’); const { sendNotificationToUser } = require(‘../services/notification_service’); // Impor service notifikasi

// @desc Mendaftarkan pengguna baru const registerUser = asyncHandler(async (req, res) => { const { name, email, password } = req.body; const userExists = await User.findOne({ email }); if (userExists) { res.status(400); throw new Error(‘Email sudah terdaftar’); } const user = await User.create({ name, email, password }); if (user) { res.status(201).json({ _id: user._id, name: user.name, email: user.email, token: generateToken(user._id), }); } else { res.status(400); throw new Error(‘Data pengguna tidak valid’); } });

// @desc Login pengguna & mendapatkan token const loginUser = asyncHandler(async (req, res) => { const { email, password } = req.body; const user = await User.findOne({ email });

if (user && (await user.matchPassword(password))) { // ================== PERUBAHAN PENTING (1/2) ================== // Logika pengiriman notifikasi DIHAPUS dari sini untuk // memperbaiki race condition. Backend sekarang hanya fokus // memberikan token otentikasi. // ==

```
res.json({
  _id: user._id,
  name: user.name,
  email: user.email,
  token: generateToken(user._id),
});
```

} else { res.status(401); throw new Error(‘Email atau password salah’); } });

// @desc Mendaftarkan atau memperbarui FCM token untuk pengguna const registerFcmToken = asyncHandler(async (req, res) => { const { token } = req.body; if (!token) { res.status(400); throw new Error(‘Token tidak ditemukan di body request’); } const user = await User.findById(req.user._id);

if (user) { const isNewToken = !user.fcmTokens.includes(token);

```
// Simpan token ke database
user.fcmTokens = [...new Set([...user.fcmTokens, token])];
await user.save();

// ================= PERUBAHAN PENTING (2/2) =================
```

```
// Notifikasi "Selamat Datang" sekarang dikirim dari sini,
// SETELAH token berhasil disimpan.
// Ini juga hanya akan dikirim jika token tersebut baru,
// untuk menghindari spam setiap kali pengguna membuka aplikasi.
if (isNewToken) {
  await sendNotificationToUser(
    user._id,
    'Selamat Datang di DigiHome!',
    `Akun Anda sekarang siap untuk menerima notifikasi di perangkat ini.`
  );
}
// ============================================================

res.status(200).json({ message: 'Token berhasil disimpan' });
```

} else { res.status(404); throw new Error(‘Pengguna tidak ditemukan’); } });

const testBudgetNotification = asyncHandler(async (req, res) => { // Untuk menyederhanakan, kita panggil fungsi yang sama dengan yang dijalankan cron // NOTE: Anda perlu mengekspor checkUserBudgets dari scheduler_service.js // Jika belum, untuk sementara kita bisa pindahkan logikanya ke sini. // Mari kita asumsikan kita akan memindahkannya ke service terpisah nanti. // Untuk sekarang, kita panggil dummy function.

// Kirim notifikasi tes langsung ke pengguna yang sedang login await sendNotificationToUser(req.user._id, ‘Uji Coba Notifikasi Budget’, ‘Jika Anda melihat ini, maka pemicu notifikasi berfungsi!’);

res.status(200).json({ message: ‘Perintah tes notifikasi budget telah dijalankan.’ }); });

module.exports = { registerUser, loginUser, registerFcmToken, testBudgetNotification, }; “‘ . . ## 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: mongoose.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 // deviceSchema.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 Notification.js *Path: relPath:=* . “‘javascript // models/Notification.js const mongoose = require(‘mongoose’);

/** * Skema untuk menyimpan riwayat notifikasi yang dikirim ke pengguna. * Ini memungkinkan aplikasi untuk menampilkan halaman riwayat notifikasi. */ const notificationSchema = new mongoose.Schema({ // Referensi ke pengguna yang menerima notifikasi user: { type: mongoose.Schema.Types.ObjectId, required: true, ref: ‘User’, }, // Judul notifikasi yang ditampilkan title: { type: String, required: true, }, // Isi pesan notifikasi body: { type: String, required: true, }, // Menyimpan data tambahan (payload) yang dikirim bersama notifikasi. // Berguna untuk navigasi di aplikasi (misal: membuka halaman detail perangkat tertentu). dataPayload: { type: Map, of: String, }, // Menandai apakah notifikasi sudah dibaca oleh pengguna atau belum. isRead: { type: Boolean,

required: true, default: false, }, }, { // Secara otomatis menambahkan field `createdAt` dan `updatedAt` timestamps: true, });

const Notification = mongoose.model('Notification', notificationSchema);

module.exports = Notification; "' . . ## 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 // models/User.js 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 }, fcmTokens: { type: [String], default: [], }, // ========== FIELD BARU UNTUK BUDGET & TARIF ========== settings: { monthlyBudget: { type: Number, default: 250000, // Anggaran default Rp 250.000 }, tariffTier: { type: String, enum: [‘tier900’, ‘tier1300’, ‘tier2200’, ‘other’], default: ‘tier1300’, // Golongan tarif default }, // Menambahkan flag untuk memastikan notifikasi budget hanya dikirim sekali per bulan budgetNotificationSent: { month: { type: Number, default: 0 }, // Menyimpan bulan (1-12) year: { type: Number, default: 0 }, // Menyimpan tahun }, }, // ==
}, { timestamps: true });

// Middleware dan method lainnya tidak berubah userSchema.pre(‘save’, async function (next) { if (!this.isModified(‘password’)) { next(); } const salt = await bcrypt.genSalt(10); this.password = await bcrypt.hash(this.password, salt); });

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 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, enterReProvisioningMode); // ——————-

module.exports = router;“‘ . . ## File notificationRoutes.js *Path: relPath:=* . “‘javascript // routes/notificationRoutes.js const express = require(‘express’); const router = express.Router(); const { getUserNotifications, markAsRead, } = require(‘../controllers/notificationController’); const { protect } = require(‘../middleware/authMiddleware’);

// Route untuk GET /api/notifications -> akan memanggil getUserNotifications router.route(‘/’).get(protect, getUserNotifications);

// Route untuk PUT /api/notifications/:id/read -> akan memanggil markAsRead router.route(‘/:id/read’).put(protect, markAsRead);

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 // routes/userRoutes.js const express = require('express'); const router = express.Router(); const { registerUser, loginUser, registerFcmToken, testBudgetNotification, } = require('../controllers/userController'); const { protect } = require('../middleware/authMiddleware');

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

// Rute baru untuk mendaftarkan token FCM router.post('/fcm-token', protect, registerFcmToken); router.get('/test-budget-notif', protect, testBudgetNotification);

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 { sendNotificationToUser } = require('./notification_service');

let client = null;

// Peta untuk mengelola perangkat yang sedang dalam proses provisioning const unclaimedDevices = new Map(); const unclaimedDeviceTimers = new Map();

// Buffer untuk mengumpulkan data telemetri sebelum disimpan ke DB secara massal const telemetryBuffer = new Map(); const BATCH_INTERVAL_MS = 5 * 60 * 1000; // Proses buffer setiap 5 menit

/** * Memproses buffer telemetri secara periodik. * Mengagregasi data telemetri yang terkumpul untuk setiap perangkat * dan menyimpannya sebagai satu entri log untuk efisiensi database. */ 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;

```
// Hitung rata-rata dari semua metrik yang terkumpul
const avgVoltage = messages.reduce((sum, msg) => sum + (msg.voltage || 0), 0) / messages.len
const avgCurrent = messages.reduce((sum, msg) => sum + (msg.current || 0), 0) / messages.len
const avgPower = messages.reduce((sum, msg) => sum + (msg.power || 0), 0) / messages.length;
const avgPowerFactor = messages.reduce((sum, msg) => sum + (msg.powerFactor || 0), 0) / mess
// Ambil nilai energi terakhir sebagai nilai akumulasi saat ini
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); }

// Kosongkan buffer setelah diproses telemetryBuffer.clear(); }

// Jalankan pemroses buffer secara periodik setInterval(processTelemetryBuffer, BATCH_INTERVAL_MS);

// Fungsi helper untuk proses klaim perangkat 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 dan dihapus dari daftar tunggu.`); };

/** * Fungsi utama untuk menghubungkan ke broker MQTT dan mengatur semua listener. * @param {Map} clientConnections - Peta koneksi WebSocket dari server 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|mqtts|ws|wss):///.test(brokerUrl)) { brokerUrl = `mqtt://${brokerUrl}`; }

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

```
try {
  const topicParts = topic.split('/');
  if (topicParts.length < 3) return;

  const topicType = topicParts[1];
  const deviceIdFromTopic = topicParts[2];

  if (topicType === 'devices' && topic.endsWith('/status')) {
    const isOnline = message.online === true;
    const device = await Device.findOne({ deviceId: deviceIdFromTopic });

    if (device && device.isOnline !== isOnline) {
      device.isOnline = isOnline;
      await device.save();
      console.log(`[MQTT-Status] Status perangkat ${device.name} diperbarui menjadi: ${isOnl

      if (!isOnline) {
        await sendNotificationToUser(
          device.owner,
          `Perangkat Offline`,
          `${device.name} tidak terhubung ke jaringan. Mohon periksa koneksi listrik dan Wi-
          { 'screen': 'digiPlugDetailRoute', 'deviceId': device._id.toString() }
```

```
      );
    }
  }
} else if (topicType === 'devices' && topic.endsWith('/telemetry')) {
  const device = await Device.findOne({ deviceId: deviceIdFromTopic });
  if (!device) return;

  await device.updateOne({ isOnline: true, lastSeen: new Date() });

  if (device.owner) {
      const ownerSocket = clientConnections.get(device.owner.toString());
      if (ownerSocket && ownerSocket.readyState === 1) {
          ownerSocket.send(payload.toString());
      }
  }

  if (!telemetryBuffer.has(deviceIdFromTopic)) telemetryBuffer.set(deviceIdFromTopic, []);
  telemetryBuffer.get(deviceIdFromTopic).push(message);

} else if (topicType === 'devices' && topic.endsWith('/alert')) {
  const device = await Device.findOne({ deviceId: deviceIdFromTopic });
  if (!device) return;

  let alertTitle = 'Peringatan Keamanan';
  let alertBody = `Terdeteksi masalah pada perangkat ${device.name}.`;

  if (message.error === 'OVERCURRENT_DETECTED') {
    alertTitle = `Arus Berlebih pada ${device.name}!`;
    alertBody = `Perangkat telah dimatikan secara otomatis untuk keamanan. Arus terdeteksi

    device.active = false;
    await device.save();
    console.log(`[ALERT] Status perangkat ${device.name} diubah menjadi OFF di database.`)

    const commandTopic = `digihome/devices/${device.deviceId}/command`;
    const commandMessage = { action: 'SET_STATUS', payload: 'OFF' };
    publishMqttMessage(commandTopic, commandMessage);
    console.log(`[ALERT] Perintah OFF dikirim ke topik ${commandTopic}.`);
  }

  await sendNotificationToUser(device.owner, alertTitle, alertBody);

} else if (topicType === 'provisioning') {
  const deviceIdFromMessage = message.deviceId;
  if (!deviceIdFromMessage) return;
```

```
    if (topic.endsWith('/online')) {
        console.log(`[Provisioning] Perangkat ${deviceIdFromMessage} online, menunggu konfi
        unclaimedDevices.set(deviceIdFromMessage, { confirmed: false });
        const timer = setTimeout(() => {
            if (unclaimedDevices.has(deviceIdFromMessage)) {
                unclaimedDevices.delete(deviceIdFromMessage);
                unclaimedDeviceTimers.delete(deviceIdFromMessage);
                console.log(`[Provisioning] Batas waktu klaim untuk ${deviceIdFromMessage} h
            }
        }, 5 * 60 * 1000); // Timeout 5 menit
        unclaimedDeviceTimers.set(deviceIdFromMessage, timer);
    } else if (topic.endsWith('/confirm')) {
        if (unclaimedDevices.has(deviceIdFromMessage)) {
            unclaimedDevices.set(deviceIdFromMessage, { confirmed: true });
            console.log(`[Provisioning] Konfirmasi fisik untuk ${deviceIdFromMessage} DITERI
        }
    }
  }
} catch (error) {
  console.error(`[MQTT] Gagal memproses pesan dari topik ${topic}:`, error.message);
}

});
```

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

/** * Mempublikasikan pesan ke topik MQTT tertentu. * @param {string} topic - Topik tujuan. * @param {object} message - Objek pesan yang akan dikirim (akan di-stringify). */ 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 notification_service.js *Path: relPath:=* . “‘javascript // services/notification_service.js const admin = require(‘firebase-admin’); const User = require(‘../models/User’); const Notification = require(‘../models/Notification’); // <– BARU

try { const serviceAccount = require(‘../firebase-service-account_digihome.json’); admin.initializeApp({ credential: admin.credential.cert(serviceAccount), }); console.log(‘[FCM] Firebase Admin SDK berhasil diinisialisasi.’); } catch (error) { console.error(‘[FCM] GAGAL inisialisasi Firebase Admin SDK. Pastikan file “firebase-service-account_digihome.json” ada dan path-nya benar.’, error.message); }

// /** // * Mengirim notifikasi ke semua perangkat milik satu pengguna dan menyimpannya ke database. // * @param {string} userId ID pengguna (dari MongoDB) yang akan dikirimi notifikasi. // * @param {string} title Judul

notifikasi. // * @param {string} body - Isi pesan notifikasi. // * @param {object} [data={}] - Data tambahan yang ingin dikirim (misal: deviceId). // */ async function sendNotificationToUser(userId, title, body, data = {}) { try { const user = await User.findById(userId); if (!user || !user.fcmTokens || user.fcmTokens.length === 0) { console.log(`[FCM] Pengguna ${userId} tidak memiliki token untuk dikirimi notifikasi.`); return; }

```
const message = {
  notification: {
    title: title,
    body: body,
  },
  tokens: user.fcmTokens,
  data: data,
  android: {
    priority: 'high',
    notification: {
      sound: 'default',
      channel_id: 'high_importance_channel',
    },
  },
  apns: {
    payload: {
      aps: {
        sound: 'default',
        badge: 1,
      },
    },
  },
};

const response = await admin.messaging().sendEachForMulticast(message);
console.log(`[FCM] Berhasil mengirim notifikasi ke ${response.successCount} token untuk peng

// --- BLOK BARU: Simpan notifikasi ke database ---
// Ini dieksekusi HANYA jika pengiriman FCM berhasil
if (response.successCount > 0) {
  await Notification.create({
    user: userId,
    title: title,
    body: body,
    dataPayload: data,
  });
  console.log(`[Notification] Notifikasi untuk user ${userId} juga disimpan ke DB.`);
}
// --- AKHIR BLOK BARU ---
```

```
if (response.failureCount > 0) {
  const tokensToRemove = [];
  response.responses.forEach((resp, idx) => {
    if (!resp.success) {
      tokensToRemove.push(user.fcmTokens[idx]);
    }
  });
  console.log(`[FCM] Menghapus token tidak valid: ${tokensToRemove}`);
  await User.updateOne({ _id: userId }, { $pullAll: { fcmTokens: tokensToRemove } });
}
```

} catch (error) { console.error(`[FCM] Gagal mengirim atau menyimpan notifikasi ke pengguna ${userId}:`, error); } }

module.exports = { sendNotificationToUser }; “‘ . . ## 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 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 User = require(‘../models/User’); // Ditambahkan untuk budget const PowerLog = require(‘../models/PowerLog’); // Ditambahkan untuk budget const { WebSocket } = require(‘ws’); const { publishMqttMessage } = require(‘./mqtt_service’); const { sendNotificationToUser } = require(‘./notification_service’); // Ditambahkan untuk budget

// Helper untuk memetakan nama hari (dari kode asli Anda) const dayMapEnToId = { Sun: ‘Min’, Mon: ‘Sen’, Tue: ‘Sel’, Wed: ‘Rab’, Thu: ‘Kam’, Fri: ‘Jum’, Sat: ‘Sab’, };

// Definisikan tarif listrik di backend agar konsisten const TARIFF_RATES = { tier900: 1352.0, tier1300: 1444.7, tier2200: 1444.7, other: 1699.5, };

// ================== FUNGSI BARU UNTUK PENGECEKAN BUDGET ================== async function checkUserBudgets() { console.log(‘[Budget] Memulai pengecekan budget pengguna...’); const now = new Date(); // Menggunakan waktu Jakarta untuk konsistensi const jakartaTime = new Date(now.toLocaleString(‘en-US’, { timeZone: ‘Asia/Jakarta’ })); const currentMonth = jakartaTime.getMonth() + 1; const currentYear = jakartaTime.getFullYear();

const users = await User.find({});

for (const user of users) { if (user.settings.budgetNotificationSent?.month === currentMonth && user.settings.budgetNotificationSent?.year === currentYear) { continue; }

```
const startOfMonth = new Date(currentYear, currentMonth - 1, 1);
const endOfMonth = new Date(currentYear, currentMonth, 0, 23, 59, 59);

const userDevices = await Device.find({ owner: user._id }).select('deviceId');
const userDeviceIds = userDevices.map(d => d.deviceId);

let totalKwhThisMonth = 0;
for (const deviceId of userDeviceIds) {
  const firstLog = await PowerLog.findOne({ deviceId, timestamp: { $gte: startOfMonth } }).s
  const lastLog = await PowerLog.findOne({ deviceId, timestamp: { $lte: endOfMonth } }).sort

  if (firstLog && lastLog && lastLog.energyKWh > firstLog.energyKWh) {
    totalKwhThisMonth += (lastLog.energyKWh - firstLog.energyKWh);
  }
}

const userTariff = TARIFF_RATES[user.settings.tariffTier] || TARIFF_RATES['tier1300'];
const estimatedCost = totalKwhThisMonth * userTariff;
const budget = user.settings.monthlyBudget;
```

```
if (budget > 0) {
    const usagePercentage = (estimatedCost / budget) * 100;

    if (usagePercentage > 80) {
      const title = 'Peringatan Anggaran Listrik';
      const body = `Pemakaian bulan ini telah mencapai ${usagePercentage.toFixed(0)}% (Rp $

      await sendNotificationToUser(user._id, title, body);

      user.settings.budgetNotificationSent = { month: currentMonth, year: currentYear };
      await user.save();
    }
}
```

} console.log(‘[Budget] Pengecekan budget selesai.’); } // ==========

const startScheduler = (clientConnections) => { console.log(‘[Scheduler] Engine is running.’);

// ========== JADWAL PERANGKAT (DARI KODE ASLI ANDA) ========== // Tugas ini tetap berjalan setiap menit untuk mengeksekusi jadwal ON/OFF perangkat. cron.schedule(’* * * * *‘, async () => { const now = new Date(); const dayFormatter = new Intl.DateTimeFormat(’en-US’, { timeZone: ‘Asia/Jakarta’, weekday: ‘short’ }); const jakartaDayShort = dayFormatter.format(now); const currentDay = dayMapEnToId[jakartaDayShort]; const timeFormatter = new Intl.DateTimeFormat(‘en-GB’, { timeZone: ‘Asia/Jakarta’, hour: ‘2-digit’, minute: ‘2-digit’, hour12: false }); const currentTime = timeFormatter.format(now);

```
// ... (SELURUH BLOK LOGIKA 'try-catch' UNTUK JADWAL PERANGKAT DARI KODE ASLI ANDA TETAP DI
try {
    const dueSchedules = await Schedule.find({
      isEnabled: true,
      days: currentDay,
      $or: [{ startTime: currentTime }, { endTime: currentTime }],
    });

    if (dueSchedules.length === 0) return;

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

    for (const schedule of dueSchedules) {
      let targetState;
      if (schedule.startTime === currentTime) {
        targetState = schedule.action === 'ON';
      } else {
        targetState = !(schedule.action === 'ON');
```

```
      }

      const device = await Device.findOne({ deviceId: schedule.deviceId, owner: schedule.own
      if (!device) {
        console.log(`[Scheduler] Device ${schedule.deviceId} not found for schedule ${schedu
        continue;
      }

      if (device.active !== targetState) {
        device.active = targetState;
        await device.save();
        console.log(`[Scheduler] Executed DB update: Set device ${device.name} to ${targetSt

        const topic = `digihome/devices/${device.deviceId}/command`;
        const message = { action: 'SET_STATUS', payload: targetState ? 'ON' : 'OFF' };
        publishMqttMessage(topic, message);
        console.log(`[Scheduler] Published MQTT command to ${topic}`);

        const ownerSocket = clientConnections.get(device.owner.toString());
        if (ownerSocket && ownerSocket.readyState === WebSocket.OPEN) {
          const payload = JSON.stringify({
            deviceId: device.deviceId,
            timestamp: new Date().toISOString(),
            power: targetState ? Math.random() * 50 + 10 : 0,
            voltage: 220,
            current: targetState ? Math.random() * 0.5 : 0,
            energyKWh: 0,
            powerFactor: 0.9,
            status: targetState ? 'ON' : 'OFF',
          });
          ownerSocket.send(payload);
          console.log(`[Scheduler] Notified user ${device.owner} about status change.`);
        }
      }
    }
} catch (error) {
console.error('[Scheduler] Error during task execution:', error);
}
```

}, { scheduled: true, timezone: 'Asia/Jakarta' }); // ==========

// ========== JADWAL BARU UNTUK BUDGET ========== // Jadwalkan pengecekan budget untuk berjalan sekali setiap hari pukul 08:00 pagi. cron.schedule('0 8 * * *', checkUserBudgets, { scheduled: true, timezone: 'Asia/Jakarta', }); // ==========

};

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; “‘ .