# 1) Arquitetura final do sistema (com ajustes/melhorias)

**Visão geral (end-to-end):**

**Frontend (Next.js 16 + React 18/TS)**: dashboards (admin e usuário), reservas (FullCalendar), painel de sala, gráficos (Recharts), toasts (Sonner), tema (next-themes), validações (Zod + RHF).

**Backend (Next API Routes)**: CRUD de salas/usuários/reservas, logs e auditoria, endpoints de sensores/equipamentos; comunicação com:

- **Servidor MIO (Node/Express)**: gateway HTTP→UDP para a placa **DBX MIO Flex**, recebe leituras (webhook) e repassa ao Next via /api/sensores/leitura.

- **MQTT** (broker externo ou interno): tópicos de eventos/telemetria e controle para dispositivos IoT (ESP32/RFID).

**Dispositivos IoT**:

- **DBX MIO Flex** via **UDP** (20108 comandos / 20109 leituras).

- **ESP32/Arduino** com **RFID/NFC** (autenticação local/edge) e sensores auxiliares.

**Banco de Dados (MySQL via Prisma)**: entidades de usuários, reservas, salas, equipamentos, leituras de sensores, logs.

**Filas/tempo real**: **WebSockets** para atualizações em dashboard e **MQTT** para eventos de campo.

**Relés**: 10 canais mapeados (iluminação, ar, projetor, etc.).

**Relatórios**: consumo, uso de salas, alertas, economia estimada.

**Ajustes/melhorias sugeridos (factíveis sem refator total):**

**Segurança/API**

Trocar body-parser por express.json() e habilitar **limite de payload**.

Validar payloads com **Zod** também no servidor MIO (já usado no front).

Adicionar **CORS restrito** ao domínio do Next, não *.

Adotar **API key** ou **mTLS** entre Next ↔ Servidor MIO para o webhook /mio/leitura.

**Observabilidade**

**Winston/Pino** com níveis (info/warn/error), **correlation-id** por requisição, e logs estruturados (JSON).

**Healthcheck** dedicado (/healthz) com verificação de variáveis de ambiente e reachability do MIO.

**Confiabilidade**

**Retry/backoff** para envio ao MIO e para POST no Next.

**Circuit breaker** para o endpoint do Next se estiver fora.

**Topologia**

Isolar o **Servidor MIO** numa subnet/host protegido; **firewall UDP** apenas para MIO-IP.

Adicionar **fila** (ex. Redis Stream) para desacoplar as leituras do persist (opcional).

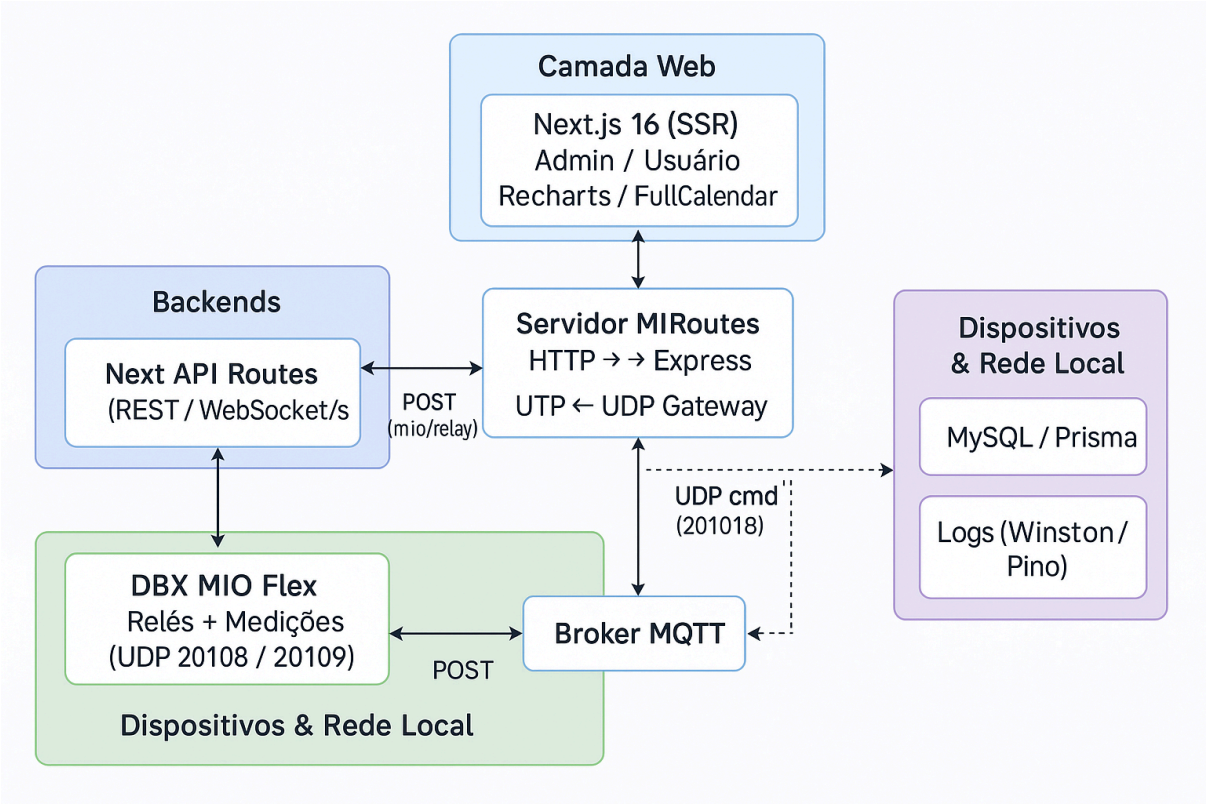
**Dados/energia**

Normalizar leituras (tensão, corrente, potência instantânea) e **agregar** por janelas (ex.: 1 min/5 min) para gráficos mais suaves.

**Automação inteligente**

Regras com **estado** (ex.: presença + reserva + luminosidade) e **histerese** para evitar “piscadas” de relé.

## 2) Diagrama atualizado (hardware + software + rede)




## 3) Código-fonte final versionado e comentado (observações rápidas)

Trecho enviado (**Servidor MIO**) está funcional e claro. Sugestões pontuais:

**Env/validação**: defina um esquema mínimo para .env.

**Express**: usar app.use(express.json({ limit: "256kb" })).

**CORS**: restringir origin.

**Tratamento de erro**: middleware de erro central.

**Idempotência**: opcional, mas útil para reenvio de leituras.

**Zod** no payload:

```
// exemplo curto de validação
import { z } from "zod";
const RelayCmd = z.object({
  action: z.enum(["ON", "OFF", "STATUS"]),
  relay: z.number().int().min(1).max(10),
});
app.post("/mio/relay", async (req, res) => {
  const parsed = RelayCmd.safeParse(req.body);
  if (!parsed.success) return res.status(400).json({ error: parsed.error.issues });
  const { action, relay } = parsed.data;
  ...
});
```

# 4) Relato técnico da integração (sensores, atuadores, lógica embarcada)

**Atores**:

*Atuadores*: relés (MIO) → iluminação/ar/projetor.

*Sensores*: tensão, corrente, potência, energia (MIO); presença, temperatura, luminosidade (ESP32); credenciais (RFID).

**Fluxo**:

**Reserva** (Next) ativa regra: no *start* da reserva, enviar action=ON para relés mapeados; no *end*, enviar action=OFF.

**Presença/RFID** (ESP32) via MQTT → Next valida usuário/sala → mantém/encerra sessão.

**Leituras MIO**: MIO → UDP 20109 → Servidor MIO /mio/leitura → Next /api/sensores/leitura → DB (Prisma).

**Automação** (regras): se **sem presença** + **sem reserva ativa** + **limiar de tempo** ⇒ desligar cargas; se **consumo anômalo** ⇒ alerta/log.

**Sincronismo**: WebSockets atualizam dashboards; backoff/queue evitam perdas quando Next estiver indisponível.

# 5) Demonstração funcional (operação simulada)

**Cenário 1 - Reserva inicia e liga iluminação (relay 1):**

```
# 1) Disparar comando ON (simula início da reserva)
curl -X POST http://localhost:3001/mio/relay \
  -H "Content-Type: application/json" \
  -d '{"action":"ON","relay":1}'
# Esperado: { ok: true, message: ... } e MIO comutando o relé 1
```

**Cenário 2 - Solicitar leitura de um relé (status/telemetria):**

```
curl -X POST http://localhost:3001/mio/solicitar-leitura \
  -H "Content-Type: application/json" \
  -d '{"relay_id":1}'
# Esperado: resposta ok, e posterior POST de leitura para o Next.
```

**Cenário 3 - Injeção de leitura (simulação local via HTTP):**

```
curl -X POST http://localhost:3001/mio/leitura \
  -H "Content-Type: application/json" \
  -d
'{"relay_id":1,"voltage":220.3,"current":2.1,"power":462.6,"energy":0.015,"temperature":24.1}'
# Esperado: encaminhar ao Next -> persistir no DB -> refletir no dashboard.
```

# 6) Testes e validações

# 6.1 Rotinas de automação

**Desligamento automático**:

Dado: sala sem reserva ativa **e** sem presença por *N* minutos

Quando: cron job/worker avalia regra

Então: action=OFF para relés mapeados; log de ação; alerta no dashboard

Teste: simular ausência (ESP pausa presença), medir estado dos relés e log.

**Alertas** (ex.: *power* > *threshold* fora de horário):

Gatilho: potência > X W entre 22h–6h

Ação: notificação no painel e opcional e-mail (Nodemailer)

## 6.2 Leituras da rede elétrica

**Validações**:

Verificar faixas plausíveis: 200V ≤ V ≤ 245V, 0 ≤ I ≤ 20A (exemplo), P ≈ V * I * fp (se fp~1 p/ cargas resistivas).

**Consistência temporal**: timestamps monotônicos, sem regressos.

**Qualidade de dados**: outliers → marcar como suspeito e não usar no cálculo de economia.

## 6.3 Eficiência energética (estimada)

**Consumo base (sem automação)**:

Medir (ou estimar) tempo ligado de iluminação/ar/projetor por reserva e fora de reserva.

**Consumo com automação**:

Tempo ligado **apenas** durante uso real (reserva + presença).

**Economia estimada**:

Para cada equipamento *k*:
Economiak≈(horas_ligado_sem_auto−horas_ligado_com_auto)×Poteˆnciak
\text{Economia}_k \approx (\text{horas_ligado_sem_auto} - \text{horas_ligado_com_auto}) \times \text{Potência}_kEconomiak≈(horas_ligado_sem_auto−horas_ligado_com_auto)×Poteˆnciak

**Relatar**: economia mensal (kWh) e percentual.

**Procedimento**:

Logar tempo ligado por relé (por dia/semana).

Agregar kWh do MIO (se disponível) ou integrar potência ao longo do tempo.

Comparar **antes/depois** em janelas equivalentes.

# 7) Backlog final — 8 sprints (reconstruído pelos artefatos)

**Sprint 1** – Setup base do monorepo + Next

Objetivo principal: Configurar a estrutura inicial do projeto e ambiente de desenvolvimento.
Itens chave: Criação da pasta src automacao, integração do framework shadcn/ui, configuração do Tailwind CSS e das páginas iniciais.
Status: Concluído

**Sprint 2** – Modelagem de Banco de Dados + Prisma

Objetivo principal: Definir o modelo de dados e integração com o ORM.
Itens chave: Criação do arquivo schema.prisma, execução de migrações e definição das entidades salas, usuários, reservas, equipamentos e leituras.
Status: Concluído

**Sprint 3** – Servidor MIO (HTTP com UDP)

Objetivo principal: Implementar o servidor intermediário de comunicação entre o sistema web e a placa DBX MIO Flex.
Itens chave: Desenvolvimento dos endpoints /mio/relay e /mio/leitura, configuração do arquivo .env e criação de logs básicos.
Status: Concluído

**Sprint 4** – Integração MIO com Next

Objetivo principal: Integrar o servidor MIO ao backend do Next.js.
Itens chave: Implementação do endpoint POST /api/sensores/leitura, criação do painel administrativo para monitoramento de status e mapeamento dos relés no sistema.
Status: Concluído

**Sprint 5** – RFID/ESP32 + MQTT

Objetivo principal: Adicionar suporte à autenticação via RFID e comunicação MQTT.
Itens chave: Criação do hook use-rfid-stream, configuração do tópico base MQTT e validação de acesso de usuários.
Status: Concluído

**Sprint 6** – Reservas + Automação

Objetivo principal: Automatizar o acionamento e desligamento de equipamentos com base nas reservas.
Itens chave: Integração do FullCalendar, implementação das regras de início e término de reserva (liga/desliga automático) e criação de logs de automação.
Status: Concluído

**Sprint 7** – Dashboards e Relatórios

Objetivo principal: Desenvolver visualizações gráficas e métricas de uso energético.
Itens chave: Utilização do Recharts para exibir potência e energia, criação de métricas de uso e sistema de alertas no dashboard.
Status: Concluído

**Sprint 8** – Segurança, Testes e Hardening

Objetivo principal: Reforçar a segurança e a confiabilidade do sistema.
Itens chave: Implementação de CORS restrito, validação de dados com Zod no servidor MIO, criação de healthcheck e execução de testes de carga e robustez.
Status: Concluído

# 8) Considerações finais (melhorias, escalabilidade, integração futura)

**Melhorias imediatas**

Segurança de API (Zod no MIO, API key entre serviços, CORS fechado).

Observabilidade (logs estruturados, métricas de erro, healthchecks).

Debounce/histerese nas regras de automação para evitar comutação excessiva.

**Escalabilidade**

Containerizar (Docker Compose) Next + Servidor MIO + Broker MQTT + MySQL.

Usar fila (Redis/Kafka) para absorver picos de leituras.

Sharding por campus/bloco (multi-tenant) com mapeamento de relés por unidade.

Integração com escolas reais

SSO (SAML/OIDC) com diretório acadêmico; perfis (professor/aluno/admin).

Import de turmas/horários do sistema acadêmico (CSV/API) para reservas automáticas.

Políticas de retenção de dados e conformidade (LGPD).

**Risco & mitigação:**

Perda de pacotes UDP → confirmação via leitura e retries.

Quedas de rede → fila local e *replay* quando reconectar.

Anomalias elétricas → thresholds, cutoffs e alertas pró-ativos.

# Apêndice - Review rápido do server.js enviado

Pontos positivos:

Endpoints claros (/mio/relay, /mio/leitura, /mio/solicitar-leitura, /).

Encaminhamento de leituras para o Next com timestamp.

Logs úteis na inicialização e checagem de .env.

Aprimoramentos práticos:

express.json({ limit: "256kb" }) no lugar de body-parser.

Validação (Zod) dos corpos de requisição.

Middleware de erro e de request id.

CORS com origin: process.env.ALLOWED_ORIGINS.

Timeout e retry (ex.: fetch-retry) ao chamar o Next.

Rate limit leve (ex.: 20 req/seg por IP) em /mio/leitura.