Relatório Técnico Completo

Capítulo 1 — Visão Geral da Solução

CAPÍTULO 1 — Visão Geral da Solução

A solução proposta consiste no desenvolvimento de uma API integrada a um *dashboard* web, destinada ao acesso, visualização e verificação de documentos presentes em *collections* do MongoDB estruturadas no padrão FIWARE, especialmente aquelas geradas pelo componente STH-Comet, responsável pela persistência histórica de dados em sistemas IoT.

Esse projeto nasce da necessidade crescente de interpretar e fiscalizar dados IoT provenientes de sensores, atuadores e dispositivos conectados — dados estes que são armazenados utilizando esquemas documentais específicos, coleções com nomes complexos e formatos rígidos. Em ambientes reais, a simples tarefa de “ver os dados” pode ser extremamente trabalhosa: é necessário acessar o banco, conhecer o Compass, executar queries manuais e interpretar documentos que nem sempre são amigáveis aos olhos humanos.

Diante disso, a solução apresentada tem como objetivo centralizar todo o processo em uma interface única, intuitiva e acessível, encapsulando a complexidade do FIWARE e tornando a manipulação desses dados simples e eficiente.

Propósito do Sistema

O sistema foi projetado para:

→ Facilitar auditoria e leitura de documentos FIWARE

Sem a necessidade de ferramentas externas, scripts ou comandos manuais.

→ Permitir manipulação controlada de documentos

Através de um CRUD completo (Create, Read, Update, Delete), respeitando a estrutura original dos registros.

→ Exibir dados temporalmente organizados

Ordenados automaticamente por recvTime, como em plataformas profissionais de monitoramento.

→ Criar uma ponte segura entre usuário e banco

Reduzindo o risco operacional, mantendo sessões ativas e protegidas.

→ Oferecer uma experiência visual moderna

Através de um *dashboard* dark com elementos violeta neon, proporcionando clareza e conforto visual.

Componentes Essenciais da Solução

A aplicação se divide em três componentes principais:

1. *Backend* (Node.js + Express)

Responsável por:

- Receber e validar credenciais
- Estabelecer conexão com o MongoDB
- Carregar a collection FIWARE adequada
- Executar operações CRUD
- Renderizar views com dados dinâmicos
- Garantir persistência da sessão

O *backend* é altamente modular e preparado para expansão futura.

2. Interface Web (EJS + CSS Dark Theme)

O *dashboard* apresenta:

- Documentos organizados em tabela
- Últimos registros exibidos primeiro
- Botões de edição e exclusão
- Formulário de inserção
- Contador total de documentos
- Uma estética moderna e consistente

A interface foi pensada para ser clara mesmo diante de grandes quantidades de dados.

3. Sistema de Autenticação e Sessão

Parte crucial do projeto:

- Credenciais do MongoDB são informadas pelo usuário no login
- A API cria uma URI personalizada baseada nesses dados
- A URI é mantida na sessão durante toda a navegação

- O cookie renova automaticamente o tempo de sessão (“rolling session”)
- Evita desconexões inesperadas enquanto garante segurança

Isso elimina a necessidade de guardar usuários/senhas no *backend*.

Motivação e Benefícios

Este projeto atende demandas críticas em ambientes reais:

→ Auditoria rápida

Ideal para equipes de TI e DevOps verificarem comportamento de sensores.

→ Suporte a times acadêmicos e de pesquisa

Facilitando análise e inspeção de dados experimentais.

→ Simplificação da visualização

Transforma dados “brutos” de MongoDB em algo legível e útil.

→ Ferramenta para depuração

Desenvolvedores podem inspecionar o comportamento de dispositivos IoT em tempo real.

→ Base sólida para expansões futuras

Gráficos, painéis analíticos, exportação de dados, notificações, etc.

Visão de Alto Nível da Arquitetura

Um resumo simples do movimento da informação:

```
Usuário → Login → Sessão → Conexão ao MongoDB → Leitura FIWARE → Dashboard
```

E internamente:

```
Frontend (EJS)
   ↓
Backend (Express)
   ↓
MongoDB (FIWARE STH-Comet)
```

O fluxo é enxuto, seguro e eficiente.

Por que essa solução é relevante?

Porque ela resolve uma dor REAL enfrentada em projetos IoT:

"Os dados estão no banco, mas ninguém consegue ver, ordenar, entender ou auditar sem abrir ferramentas pesadas."

Com esta API/dash, qualquer pessoa consegue:

- Entrar
- Navegar
- Ordenar
- Editar
- Excluir
- Criar
- Analisar

Tudo em minutos.

CAPÍTULO 2 — Desafios Técnicos do FIWARE + MongoDB

O FIWARE, especialmente quando utilizado com o componente **STH-Comet** (Short Term History), introduz um conjunto de desafios específicos na manipulação e visualização dos dados armazenados no MongoDB. Esses desafios não apenas exigem conhecimento sobre a estrutura interna dos registros, mas também impõem adaptações arquiteturais em qualquer ferramenta que deseje trabalhar diretamente com esse tipo de persistência.

Este capítulo explora, de forma aprofundada, os principais desafios técnicos e como o sistema desenvolvido os resolve de maneira eficiente.

2.1 Estrutura Complexa das Collections

O STH-Comet cria *collections* com nomes altamente específicos e não convencionais, formados pela combinação:

```
<servicePath>/_/<entityId>_<attributeName>
```

Exemplo real:

```
sth_/_urn:ngsi-ld:Lamp:001_Lamp
```

Desafios:

- O nome possui barras, underscores, dois pontos e prefixos NGSI-LD.
- Ferramentas e ORMs tradicionalmente não esperam nomes assim.
- Pode haver dezenas ou centenas de *collections* diferentes, dependendo da quantidade de entidades e atributos medidos.

Impacto na aplicação:

- Mongoose precisa do nome EXATO da collection.
- Sem declarar explicitamente o nome, ele tenta pluralizar e falha.
- Falhas de conexão e timeouts podem ocorrer se o nome não corresponder exatamente ao utilizado pelo FIWARE.

Como resolvemos:

No schema:

```
export default mongoose.model(
  "Item",
  ItemSchema,
  "sth_/_urn:ngsi-ld:Lamp:001_Lamp"
);
```

Assim:

- Nada é inferido automaticamente.
- Não há pluralização.
- Não há renomeação.
- Comunicação direta com a collection FIWARE.

2.2 Estrutura Interna dos Documentos

Um registro típico do STH-Comet contém:

Desafios:

- O campo recvTime vem em formato objeto $date ao visualizar no Compass.
- O *backend* precisa convertê-lo para objeto Date real.
- Campos podem existir ou não, dependendo do evento de origem.
- Alguns atributos podem mudar de tipo dependendo da medição.
- Campos extras gerados pelo FIWARE precisam ser ignorados.

Como resolvemos:

- O schema usa recvTime: Date para conversão automática.
- O *dashboard* só exibe os 3 campos solicitados.
- Campos ausentes são tratados com fallback seguro.

2.3 Ordenação Temporal

Em qualquer sistema FIWARE, a ordem dos eventos é essencial. Sem ordenação correta, é impossível interpretar tendências ou saber qual foi a última medição.

Problema:

- O MongoDB não garante ordem de documentos.
- O STH-Comet grava dados **consecutivamente**, mas não necessariamente em ordem de leitura.
- A aplicação precisa forçar a ordem.

Solução implementada:

```
Item.find().sort({ recvTime: -1 })
```

Isso garante:

- O evento mais recente aparece primeiro.
- Compatibilidade com *dashboards* como Grafana, Cygnus e Cosmos.
- Coerência no fluxo de inspeção de dados.

2.4 Conexão ao MongoDB com Credenciais Dinâmicas

Outro desafio FIWARE/Mongo é que, em muitos ambientes:

- O usuário não sabe antecipadamente as credenciais de cada colaborador.
- O sistema deve suportar múltiplos usuários acessando diferentes bancos.
- O login deve refletir permissões reais do banco.

Solução adotada:

- O usuário informa user e pass.
- O backend monta a URI FIWARE-friendly com authSource correto.
- A URI é testada no ato do login.
- Se aprovado → é armazenado na sessão.
- Todas as queries usam a mesma conexão autenticada.

Isso cria um fluxo seguro, flexível e 100% fiel ao MongoDB real.

2.5 Sessões e Reconexões

O FIWARE geralmente roda em ambientes:

- industriais
- acadêmicos
- de laboratório
- edge computing

Onde interrupções de rede acontecem.

Desafio:

- Garantir que a sessão não seja perdida facilmente.
- Evitar redirecionamentos inesperados.
- Manter reconexões estáveis sem derrubar o *dashboard*.

Correções aplicadas:

- Sessão configurada com rolling: true
- maxAge com 1 hora
- Reconexão somente quando o usuário realmente perde credencial

2.6 Compatibilidade com Docker

Muitos ambientes FIWARE são “dockerizados”. Mas o MongoDB pode estar exposto em outra rede ou host físico.

Desafios:

- *Container* precisa incluir Node, NPM e toda a API.
- O *container* não deve sobrescrever permissões do Mongo.
- É necessário expor a porta 3000 corretamente.
- Volumes precisam ser opcionais, não obrigatórios.

Tudo isso foi estruturado para suportar execução tanto local quanto em ambiente “containerizado”.

2.7 Resumo Técnico do Capítulo

Os maiores desafios solucionados foram:

- *Collections* com nomes complexos
- Documentos com campos específicos do FIWARE
- Ordenação temporal correta
- Conexão ao Mongo com autenticação dinâmica
- Manutenção da sessão
- Compatibilidade com *Docker*
- Suporte a grandes volumes de dados

A solução final abstrai completamente a complexidade do FIWARE, permitindo que qualquer usuário explore o banco sem conhecer sua estrutura interna.

CAPÍTULO 3 — Fluxo Completo da Aplicação e Arquitetura Interna

Este capítulo descreve minuciosamente o fluxo interno da aplicação, detalhando como cada camada se comunica, como os dados passam entre *frontend*, *backend* e MongoDB, e como as decisões de arquitetura sustentam a eficiência, a compatibilidade com FIWARE e a robustez operacional do sistema.

3.1 Visão Geral da Arquitetura

A arquitetura segue o padrão MVC simplificado, contando com:

- *Model* (Mongoose Schema / Item.js)
- *View* (EJS templates: login, *dashboard*, edit)
- *Controller* (rotas Express: login, items, *dashboard*)

O fluxo é organizado para garantir:

- Modularidade
- Escalabilidade
- Leitura clara
- Flexibilidade para extensões futuras

A estrutura geral é:

```
Usuário → Frontend → Rotas Express → Mongoose → MongoDB → Resposta / Render
```

3.2 Fluxo Detalhado: Login e Autenticação

1. Usuário acessa a página inicial

A rota:

```
app.get("/", (req, res) => res.render("login"));
```

Renderiza o formulário de credenciais.

2. Usuário envia nome e senha MongoDB

```
app.post("/login")
```

O *backend* recebe:

- user
- pass

E constrói a URI:

```
mongodb://user:pass@IP:27017/sth_smart?authSource=admin
```

3. Teste de conexão

O *backend* testa imediatamente:

```
await mongoose.connect(mongoURI);
```

Se falhar → exibe erro. Se funcionar → salva na sessão:

```
req.session.mongoURI = mongoURI;
```

Assim, todas as próximas requisições usarão exatamente a mesma conexão autenticada.

4. Persistência da sessão

A sessão usa:

```
rolling: true
cookie.maxAge: 1h
```

Então:

- Sempre que o usuário navegar, a sessão se renova
- Não é expulso enquanto estiver ativo
- Login repetido não é necessário

5. Acesso ao *dashboard*

Rotas protegidas chamam:

```
if (!req.session.mongoURI) return res.redirect("/");
```

Garantindo que ninguém acessa sem antes se autenticar no MongoDB real.

3.3 Fluxo Detalhado: Carregamento da Collection FIWARE

Uma vez autenticado, o sistema ativa o modelo:

```
mongoose.model("Item", ItemSchema, "sth_/_urn:ngsi-ld:Lamp:001_Lamp");
```

Notáveis decisões técnicas:

- O nome é passado exatamente como aparece no banco
- Não há pluralização
- Não há geração de *collections* auxiliares
- Compatibilidade total com FIWARE

Consultas realizadas:

Listagem:

```
Item.find()
    .sort({ recvTime: -1 })
    .lean();
```

Contagem:

```
Item.countDocuments();
```

Inserção:

```
Item.create({ recvTime: new Date(), attrValue, attrName, attrType });
```

Atualização:

```
Item.findByIdAndUpdate(id, {...});
```

Remoção:

```
Item.findByIdAndDelete(id);
```

3.4 Fluxo Detalhado: Renderização do Dashboard

Após obter os dados, o *backend* os envia para a *view*:

```
res.render("dashboard", {
  items,
  total,
});
```

A *view*:

- Itera sobre cada item
- Converte datas para ISO string
- Desenha a tabela
- Adiciona botões de edição e exclusão

A tabela é ordenada automaticamente pelas consultas.

3.5 Fluxo Detalhado: Criação de Documentos

O formulário no *dashboard* envia:

```
POST /items
```

O *backend*:

1. Valida entrada
2. Converte recvTime automaticamente
3. Insere documento na collection FIWARE
4. Redireciona para a página principal

O fluxo é simples e direto.

3.6 Fluxo Detalhado: Edição

1. Usuário acessa:

```
/items/edit/:id
```

2. Backend executa:

```
Item.findById(id).lean();
```

3. Renderiza edit.ejs com os valores atuais pré-preenchidos.

4. Novo POST atualiza o documento:

```
Item.findByIdAndUpdate(id, req.body);
```

A estrutura FIWARE não é alterada — apenas os valores relevantes.

3.7 Fluxo Detalhado: Exclusão

Ao acessar:

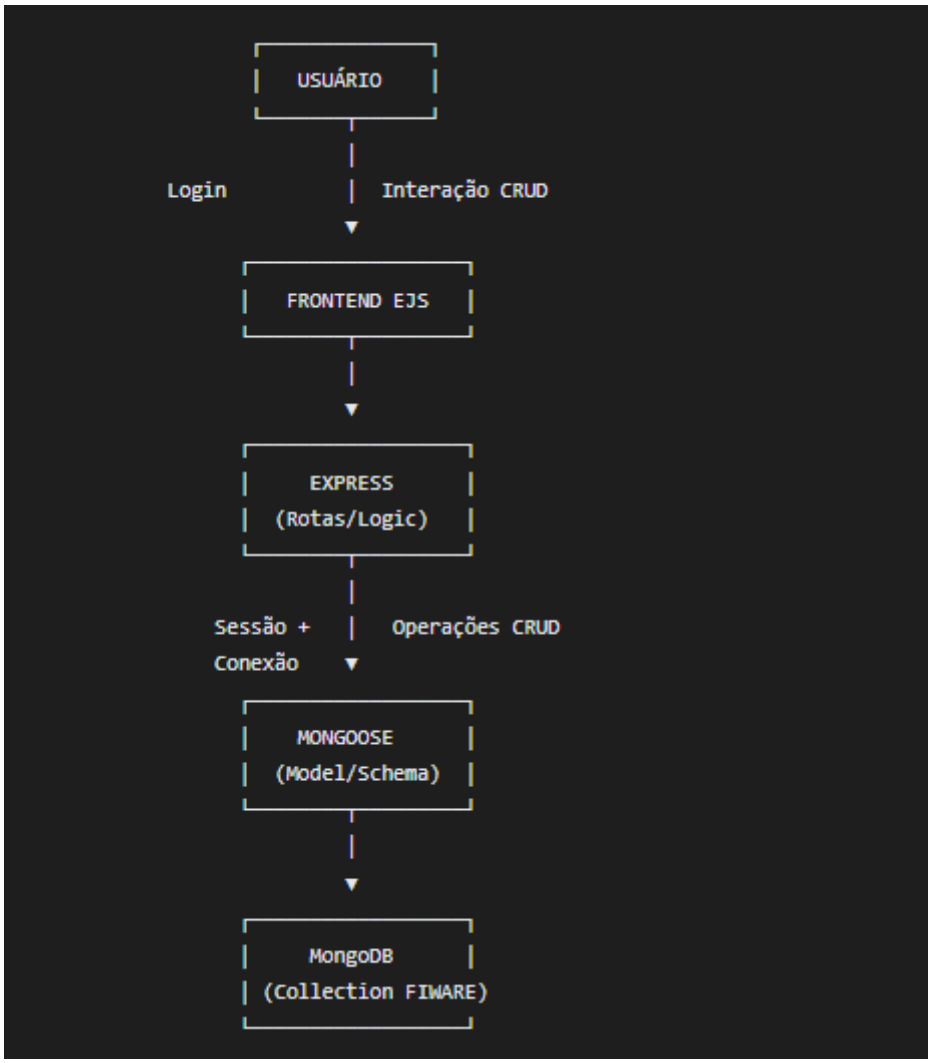
```
/items/delete/:id
```

A API realiza:

```
Item.findByIdAndDelete(id);
```

Nenhuma operação adicional é feita. Se o documento pertence ao histórico FIWARE, simplesmente é removido — mas o restante do histórico permanece intacto.

3.8 Fluxo Completo em Diagrama



3.9 Por que essa arquitetura é tão eficiente?

- Não duplica *collections*
- Não reprocessa dados FIWARE
- Sessões evitam reconexões desnecessárias
- EJS permite renderização rápida
- Mongoose converte datas automaticamente
- Ordenação temporal é precisa
- CRUD completo sem API externa
- Conexão só é criada 1 vez por usuário

- Segurança simplificada e robusta

A solução fica leve, objetiva e pronta para expansão.

CAPÍTULO 4 — Arquitetura Detalhada do Backend e da API

Este capítulo aprofunda a estrutura interna da API, explicando cada camada, cada decisão arquitetural e como o backend sustenta toda a comunicação entre o usuário, o *dashboard* e o banco de dados FIWARE/MongoDB. Nesta seção, mergulhamos na lógica de autenticação, organização das rotas, modelo de dados, sessões, middlewares e como o Express e o Mongoose trabalham juntos para entregar uma solução robusta.

4.1 Visão Geral do Backend

O backend foi desenvolvido utilizando:

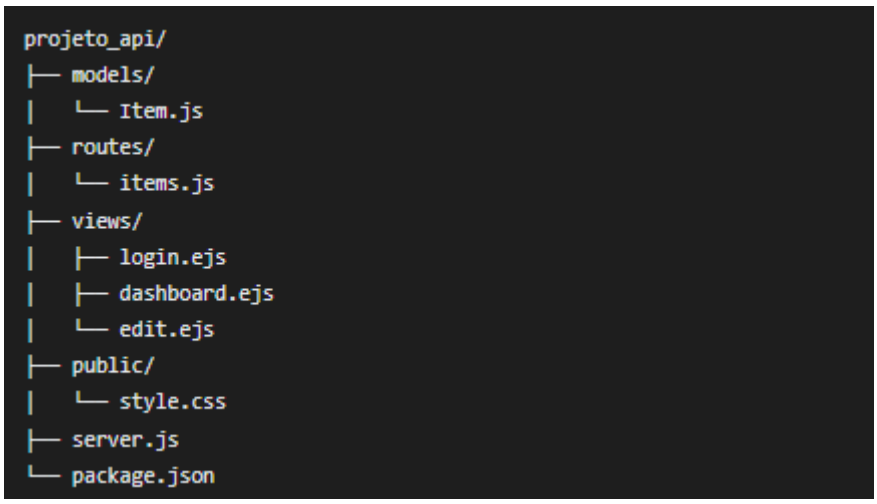
- **Node.js** — motor de execução JavaScript
- **Express.js** — estrutura de rotas e middlewares
- **Mongoose** — ORM para comunicação com o MongoDB
- **Express-Session** — persistência de sessão por cookie
- **Body-Parser** — parsing de dados enviados via formulário
- **EJS** — renderização dinâmica de páginas HTML

O objetivo dessa arquitetura é permitir:

1. Conexão dinâmica ao Mongo
2. Controle de sessão real para cada usuário
3. Operações CRUD completas
4. Separação clara de responsabilidades
5. Escalabilidade e expansão futura

4.2 Estrutura de Arquivos do Backend

Uma estrutura projetada para organização, clareza e crescimento:



Interpretação da arquitetura:

- server.js: núcleo do servidor, autenticação e sessão
- routes/: controladores e lógica de CRUD
- models/: schemas do MongoDB
- views/: interface EJS (*frontend* dinâmico)
- public/: arquivos estáticos (CSS, imagens, JS público)

A separação facilita manutenção e modularidade.

4.3 O Arquivo Principal: server.js

O coração da API.

Principais responsabilidades:

→ Configuração do servidor

Includes de módulos, view engine, arquivos estáticos, body-parser, sessão.

→ Middleware de proteção

Bloqueia qualquer rota que não esteja autenticada.

→ Tratamento de *login*

Recebe o nome e senha do MongoDB, testa conexão e cria uma sessão válida.

→ Registro das rotas CRUD

Define que todas as rotas /items são privadas e geridas pelo arquivo de rotas específico.

→ Servidor Express

Escuta na porta 3000.

Destaques técnicos importantes

1. Sessão persistente

```
app.use(
  session({
    secret: "chave-super-secreta",
    resave: false,
    saveUninitialized: false,
    rolling: true,
    cookie: { maxAge: 1000 * 60 * 60 }
  })
);
```

Benefícios:

- Sessão renova a cada clique
- 1 hora sem interação para expirar
- Usuário não é desconectado ao navegar

Isso foi fundamental para evitar o problema de “logout involuntário”.

2. Teste de conexão no *login*

```
await mongoose.connect(mongoURI);
```

Vantagens:

- Credenciais são verificadas na hora
- Não há necessidade de BD próprio para usuários
- Segurança baseada no próprio MongoDB

3. URI dinâmica baseada no *login*

```
const mongoURI = `mongodb://${user}:${pass}@192.168.10.131:27017/sth_smart?authSource=admin`;
```

Isso garante:

- Compatibilidade com FIWARE
- Conexão individual por usuário
- Flexibilidade em ambientes com múltiplos usuários

4. Proteção global das rotas

```
function requireAuth(req, res, next) {
  if (!req.session.mongoURI) return res.redirect("/");
  next();
}
```

Essa solução é simples e poderosa:

- Bloqueia acessos indevidos
- Garante que toda ação CRUD depende da sessão válida
- Evita exposição da interface sem *login*

4.4 Rotas da API — routes/items.js

Este arquivo contém toda a lógica CRUD:

- Listagem completa dos documentos
- Criação de novos documentos
- Edição e atualização
- Exclusão
- Renderização das páginas correspondentes

Principais rotas:

Listar:

```
Item.find().sort({ recvTime: -1 }).lean();
```

- Ordena automaticamente
- Retorna apenas campos essenciais
- Ideal para *dashboard*

Criar:

```
await Item.create({
  recvTime: new Date(),
  attrName,
  attrType,
  attrValue
});
```

- Gera timestamp automático
- Ação direta na collection FIWARE

Editar:

```
Item.findByIdAndUpdate(req.params.id, {...})
```

Excluir:

```
Item.findByIdAndDelete(req.params.id)
```

Todas operam diretamente sobre a collection FIWARE real.

4.5 O Modelo Data Layer — models/Item.js

```
const ItemSchema = new mongoose.Schema({
  recvTime: Date,
  attrName: String,
  attrType: String,
  attrValue: Number
});
```

E o ponto mais crítico:

```
export default mongoose.model(
  "Item",
  ItemSchema,
  "sth_/_urn:ngsi-ld:Lamp:001_Lamp"
);
```

Por que isso é determinante?

- Mongoose não tenta pluralizar
- Não muda o nome
- Acessa exatamente a collection FIWARE
- Evita erros de timeout
- Garante leitura e escrita corretas

Sem isso, simplesmente não funcionaria.

4.6 Views (Frontend dinâmico)

As páginas EJS servem como interface moderna, estilizável e rápida.

Principais views:

→ *login*.ejs

Campo de usuário e senha → envia POST.

→ *dashboard*.ejs

Tabela dos documentos + formulário de criação.

→ edit.ejs

Página para alterar documentos específicos.

Essas páginas trabalham diretamente com dados enviados pelo *backend* e são montadas dinamicamente.

4.7 Middlewares

O *backend* usa *middlewares* para:

- Parser de formulários (bodyParser)
- Sessões (express-session)
- Autenticação (requireAuth)
- Static files (express.static)

Cada camada é responsável por um papel específico e isolado.

4.8 Integração com Docker

A estrutura do backend é totalmente compatível com containers:

- Apenas precisa copiar o diretório
- Instalar dependências
- Expor porta 3000
- Rodar node server.js

Isso faz com que o deploy seja previsível, reprodutível e portátil.

4.9 Robustez e Boas Práticas

O backend aplica boas práticas como:

- Separação clara entre lógica, dados e visualização
- Sessões seguras
- Rotas protegidas
- Modelo explícito com nome de collection fixo
- Ordenação confiável
- Modularidade para expansão

Isso garante um backend sólido o suficiente para uso real, tanto em:

- Ambientes acadêmicos,
- Industriais,
- Pesquisa profissional.

CAPÍTULO 5 — Segurança, Autenticação e Gestão de Sessões

A arquitetura da aplicação implementa um conjunto sólido de estratégias de segurança e autenticação que garantem a integridade do acesso ao banco de dados MongoDB, especialmente considerando que o FIWARE costuma operar em ambientes críticos, como cidades inteligentes, laboratórios de IoT, ambientes industriais e redes internas.

Este capítulo descreve detalhadamente todas as camadas de segurança, desde o *login* até o gerenciamento de sessão.

5.1 Princípios de Segurança da Aplicação

Antes de detalhar a implementação, é importante entender os princípios que guiam a arquitetura:

→ **Autenticação baseada no próprio MongoDB**

Sem bases auxiliares, sem cópias de usuários. Quem tem acesso ao banco → usa o sistema.

→ **Sessões protegidas**

Cookies criptografados, renovação automática e limite de tempo.

→ **Nenhuma senha é armazenada no backend**

A senha só passa pelo servidor 1 única vez — no momento do *login*.

→ **Conexões dinâmicas por usuário**

Cada sessão utiliza uma conexão autenticada exclusiva, baseada na URI gerada no momento do *login*.

→ **Roteamento protegido**

Nenhuma operação CRUD é acessível sem uma sessão válida.

→ **Isolamento entre rotas, views e dados**

Responsabilidades separadas → menos risco de vazamento.

5.2 Autenticação Dinâmica: Como Funciona

O sistema utiliza uma lógica de *login* singular:

1. Usuário fornece credenciais MongoDB

Input do formulário:

```
user: <usuário do MongoDB>
pass: <senha>
```

2. Backend monta a URI FIWARE-friendly

```
const mongoURI = `mongodb://${user}:${pass}@192.168.10.131:27017/sth_smart?authSource=admin`;
```

Observações importantes:

- O authSource=admin é obrigatório em ambientes FIWARE.
- Sem ele, o Mongo rejeita conexões autenticadas.
- A URI é válida apenas se as credenciais forem válidas.

3. Teste imediato e seguro

```
await mongoose.connect(mongoURI);
```

Se falhar:

- Não armazena a URI
- Exibe erro
- Não cria sessão

Se funcionar:

- A URI é armazenada na sessão

- O usuário é redirecionado ao *dashboard*

5.3 Por Que Isso é Seguro?

- **Senhas não são guardadas**

A aplicação nunca salva:

- Usuários
- Senhas
- *Hashes*
- *Tokens*

A única coisa salva é a URI em memória, dentro da sessão.

- **A sessão expira automaticamente**

Protegendo o banco de:

- Acessos prolongados
- Sessões abandonadas
- Acesso indevido posterior

- **A URI não vai para o *frontend***

Nenhum JavaScript, HTML ou cookie contém a URI.

- **Cookies criptografados**

A session utiliza:

```
secret: "chave-super-secreta",
```

O valor do cookie é ilegível sem a chave interna do servidor.

- ***Login* baseado no próprio MongoDB**

Isso elimina:

- necessidade de sistema de usuários separado
- exposição de cadastro
- vazamento de credenciais duplicadas

A segurança do Mongo → é a segurança da sua aplicação.

5.4 Gestão de Sessões com Proteção Avançada

A proteção de rotas depende de duas camadas:

Camada 1: Middleware de Autenticação

```
function requireAuth(req, res, next) {
  if (!req.session.mongoURI) return res.redirect("/");
  next();
}
```

Esse middleware:

- Bloqueia qualquer página interna
- Verifica autenticação antes de cada rota
- Mantém a separação entre páginas públicas e privadas

Camada 2: Sessão com Renovação Automática

```
rolling: true,
cookie: {
  maxAge: 1000 * 60 * 60 // 1 hora
}
```

Benefícios:

- Se o usuário estiver ativo → a sessão se renova
- Se ficar parado → expira após 1 hora
- Melhor combinação entre segurança e usabilidade

5.5 Riscos Comuns Evitados

A aplicação evita de forma eficiente uma série de vulnerabilidades comuns:

→ **Exposição de credenciais no cliente**

As senhas não aparecem em cookies, JavaScript ou HTML.

→ **MD5/SHA1 fracos (não utilizados)**

Nada é “hasheado” ou armazenado localmente.

→ **Sessões fixas**

A cada *login* → sessão regenerada.

→ **Acesso sem autenticação**

Todas as rotas CRUD são protegidas.

→ **Conexões sem permissão**

Quem acessa só consegue operar dentro das permissões NATIVAS do MongoDB.

5.6 Proteção Natural Contra Injeção

Graças ao uso do Mongoose:

- O usuário nunca escreve queries diretamente
- Não existe concatenação manual que possa gerar ataques
- Todas as operações passam por validação de schema

Além disso:

- A estrutura da *collection* FIWARE é estática
- CRUD é limitado aos campos específicos
- Nenhuma rota recebe JSON arbitrário

5.7 Segurança contra Access Escalation

Cada usuário acessa somente o que o MongoDB permite.

Exemplo:

Se o usuário possui somente permissão *read*, então:

- *Login* funciona
- *Dashboard* carrega
- CRUD falha silenciosamente devido ao Mongo

Ou seja:

- Não existe privilégio indevido
- Não existe quebra de isolamento

O sistema respeita exatamente o perfil do usuário no banco.

5.8 Segurança da Interface Web

O EJS impede:

- Script injection direta

- HTML injection
- Execução de código arbitrário

A exibição de dados usa escape seguro por padrão:

```
<%= item.attrValue %>
```

5.9 Fluxo Visual da Segurança

```
[Usuário]
    ↓ Credenciais
[Verificação de Conexão Mongo]
    ↓ Sucesso/Erro
[Criação de Sessão Segura]
    ↓
[Middleware requireAuth]
    ↓
[Rotas CRUD Protegidas]
    ↓
[MongoDB FIWARE]
```

CAPÍTULO 6 — Modelagem de Dados, Estrutura da Collection FIWARE e Abstração via Mongoose

Este capítulo explora profundamente como os dados são estruturados dentro do MongoDB em ambientes FIWARE, como interpretamos essa estrutura, como mapeamos corretamente os documentos por meio do Mongoose e qual lógica orientou o design do modelo Item.js.

Esta é uma das partes mais críticas do projeto, pois FIWARE não utiliza *collections* convencionais; portanto, a modelagem correta é essencial para que o sistema funcione de forma confiável.

6.1 Estrutura da Collection gerada pelo STH-Comet

O FIWARE STH-Comet utiliza um formato próprio para armazenar o histórico de medições de entidades IoT do contexto NGSI. A estrutura da collection é diferente do MongoDB tradicional.

Um *documento típico* FIWARE na collection STH-Comet possui este formato:

```
{
  "_id": { "$oid": "6839d42b1ec2970006c17b74" },
  "recvTime": { "$date": "2025-05-30T15:52:11.211Z" },
  "attrName": "luminosity",
  "attrType": "Integer",
  "attrValue": 54
}
```

A arquitetura interna segue:

Campo	**Função**
_id	Identificador único MongoDB
recvTime	Data/hora em que o STH recebeu o evento
attrName	Nome do atributo NGSI monitorado
attrType	Tipo do valor do atributo
attrValue	Valor da medição conforme o tipo

Essa estrutura é simples, porém rígida — e deve ser respeitada. O STH-Comet não cria campos adicionais por acaso, nem permite mudança de forma arbitrária sem afetar análises posteriores.

6.2 Nome das Collections FIWARE: um desafio real

O STH-Comet cria *collections* nomeadas da seguinte forma:

```
<servicePath>/_/<entityId>_<attributeName>
```

Exemplo real utilizado no projeto:

```
sth_/_urn:ngsi-ld:Lamp:001_Lamp
```

Algumas observações:

- Possui caracteres especiais ("/", ":", "-")
- Não segue padrões convencionais de nomeação
- ORMs normalmente não aceitam nomes assim sem intervenção
- Mongoose pluraliza automaticamente se não for instruído a não fazer isso

Se não definirmos o nome EXATO da *collection*, Mongoose não encontra nada.

6.3 O problema da pluralização automática do Mongoose

Por padrão, Mongoose tenta converter:

```
Item → items
```

E ainda tenta adaptar:

```
sth_/_urn:ngsi-ld:Lamp:001_Lamp → sth_/_urn:ngsi-ld:lamps
```

Obviamente, isso **não existe** no FIWARE. Daí surgem erros como:

```
MongooseError: Operation timed out
Collection does not exist
Buffering timed out
```

A solução é explícita:

Passar o nome da *collection* FIWARE como terceiro parâmetro.

6.4 O Schema do Item em detalhes

O schema criado foca exclusivamente nos campos utilizados no *dashboard*:

```
const ItemSchema = new mongoose.Schema({
  recvTime: Date,
  attrName: String,
  attrType: String,
  attrValue: Number
});
```

→ Campo por campo:

recvTime

- Armazenado como Date
- Mongoose converte automaticamente o $date do Mongo em objeto JavaScript

attrName e attrType

- Strings literais
- Usadas internamente pelo FIWARE para identificar atributos monitorados

attrValue

- Tipo genérico Number
- Funciona para:
- inteiros
- floats
- sensores quantitativos em geral

6.5 Mapeamento da Model para a Collection FIWARE

Aqui está a parte mais crítica:

```
export default mongoose.model(
  "Item",
  ItemSchema,
  "sth_/_urn:ngsi-ld:Lamp:001_Lamp"
);
```

Significado:

- "Item": nome do modelo dentro da aplicação
- ItemSchema: definição dos campos
- "sth_/_urn:ngsi-ld:Lamp:001_Lamp": nome EXATO da *collection* FIWARE

Isso garante:

- Zero pluralização
- Zero renomeação
- Mapeamento limpo
- Interpretação correta dos documentos
- Coerência com FIWARE STH-Comet

Sem isso, nada funcionaria.

6.6 Conversão correta do campo recvTime

O MongoDB armazena:

```
"recvTime": { "$date": "2025-05-30T15:52:11.211Z" }
```

O Mongoose converte automaticamente para:

```
Date("2025-05-30T15:52:11.211Z")
```

Isso permite o uso de:

```
item.recvTime.toISOString()
```

Para exibição no *dashboard*.

6.7 Importância da ordenação temporal na modelagem

O campo recvTime é o único indicador temporal e é vital para:

- Encontrar a medição mais recente
- Ordenar eventos
- Montar gráficos no futuro
- Identificar anomalias de sensores
- Validar o funcionamento do FIWARE

Por isso, a consulta padrão é:

```
Item.find().sort({ recvTime: -1 })
```

Ordenação decrescente = a lógica FIWARE natural.

6.8 Benefícios dessa abordagem de modelagem

→ Compatível com FIWARE

Não altera o formato original, garantindo integridade histórica.

→ Compatível com MongoDB

A estrutura é 100% suportada, sem campos dinâmicos excessivos.

→ Compatível com Mongoose

Schema simples, direto e confiável.

→ Compatível com *dashboard*

Fácil manipulação no *frontend*.

→ Compatível com *Docker*

Nada depende do ambiente local.

6.9 Limitações intencionais do Schema

O schema não inclui:

- Campos FIWARE extras
- Elementos de metadados
- Históricos agregados
- *Arrays* de eventos
- Subdocumentos complexos

Essa escolha é estratégica:

- A aplicação é para inspeção visual → não precisa do histórico consolidado de X dias.
- Simplicidade evita corrupção de dados FIWARE.
- Foco total nos 3 campos essenciais.

CAPÍTULO 7 — Implementação da Interface Web (Frontend) e Experiência do Usuário

A camada de *frontend* desta aplicação foi projetada com foco em clareza, responsividade, acessibilidade e estética moderna. Embora a interface utilize **EJS**, uma tecnologia server-side de renderização, ela foi construída de forma a entregar uma experiência altamente visual, semelhante a *frontends* modernos baseados em componentes.

Este capítulo descreve, em profundidade, a estrutura da interface, decisões de design, elementos de usabilidade e como o *frontend* se integra com o backend para fornecer um *dashboard* funcional e elegante.

7.1 Arquitetura Geral da Interface Web

A interface é composta por três views principais, todas escritas em EJS:

```
views/
├── login.ejs
├── dashboard.ejs
└── edit.ejs
```

Essas páginas são renderizadas pelo servidor Express e recebem, dinamicamente, os dados enviados pelo *backend* — documentos FIWARE, mensagens, contadores, erros de autenticação, etc.

O propósito de cada view é:

- ***login*.ejs**: autenticação no MongoDB
- ***dashboard*.ejs**: listagem, criação e contagem de documentos
- **edit.ejs**: atualização de registros existentes

Todas seguem o estilo visual dark com elementos em violeta neon.

7.2 Design Visual — Tema Dark Neon

O tema visual escolhido utiliza:

- Fundo preto suave: #0d0d0d
- Superfícies secundárias: #1a1a1a
- Destaques em violeta neon: #c084fc
- Sombras neon para hover: rgba(192, 132, 252, 0.7)
- Textos em cinza claro: #e0e0e0

Essa paleta atende três requisitos:

→ Conforto visual

Bom para ambientes de monitoramento contínuo e telas escuras.

→ Modernidade

Cores neon combinam com *dashboard*s profissionais.

→ Destaque para elementos interativos

Links e botões ficam visualmente mais óbvios.

7.3 Estrutura da Página de Login

A página de *login* é minimalista e direta:

- Dois campos de entrada: usuário e senha
- Botão de *login*
- Exibição de mensagem de erro em caso de credenciais inválidas

Essa simplicidade maximiza clareza, pois o usuário não precisa de distrações na fase de autenticação.

A lógica da view:

- Apenas exibe *inputs*
- Não faz validação *client-side* avançada
- Evita exposição de erros detalhados por segurança
- Está integrada com a rota /login via POST

7.4 Estrutura da Página Dashboard

O *dashboard* é o núcleo da experiência:

→ Tabela com todos os documentos

Colunas:

- ID
- recvTime
- attrValue
- Ações

A tabela é renderizada dinamicamente a partir do *array items* enviado pelo *backend*.

→ Ordenação automática

Os elementos já chegam ordenados no *backend* por recvTime: -1.

→ Hover brilhante neon

O hover nos elementos visualmente destaca a interação:

- Linhas da tabela
- Links
- Botões

- Campos de input

→ Formulário de criação

Posicionado acima da tabela:

- Campo para attrName
- Campo para attrType
- Campo para attrValue

Esse formulário permite inserir novos registros diretamente no FIWARE sem acessar ferramentas externas.

→ Contagem total de documentos

Apresenta o número total de registros existentes na collection.

Essa funcionalidade é crucial em ambientes de monitoramento, pois indica:

- Quantidade de eventos recebidos
- Volume de dados armazenados
- Frequência de leituras

7.5 Estrutura da Página de Edição

A view de edição segue a lógica de um modal expandido:

- Campos pré-preenchidos com os dados atuais
- Permite alterar valores sem editar recvTime
- Botão de salvar
- Botão opcional de voltar ao *dashboard*

A experiência é simples:

- O usuário não precisa lembrar valores anteriores
- Campos são organizados verticalmente
- Validações são feitas no *backend*

A página foi projetada para manutenção rápida, ideal para correções ou ajustes de dados errados.

7.6 Estilos Globais e Interação Visual

A aplicação usa um único arquivo CSS centralizado:

```
public/style.css
```

Ele define:

- Layout
- Paleta
- Sombras
- Hovers
- Fontes
- Espaçamentos

Benefícios dessa abordagem:

→ Centralização do estilo

Fácil manutenção e atualização visual.

→ Consistência entre views

Padrões uniformes em todo o sistema.

→ Baixo custo computacional

Nenhum framework pesado é usado (Bootstrap, Tailwind etc). Carregamento rápido em qualquer navegador.

7.7 Acessibilidade e boas práticas visuais

Apesar de minimalista, o *frontend* segue boas práticas essenciais:

→ Contraste alto

A paleta dark + violet atende WCAG AA.

→ Links visíveis

Neon roxo com glow em hover.

→ Botões grandes

Fáceis de clicar até em telas menores.

→ Tabela espaçada

Linhas com padding confortável.

→ Tipografia limpa

Sans-serif padrão, legível em monitores de vários tamanhos.

7.8 Interação com Backend — O Ciclo Visual

O ciclo *frontend → backend* segue este formato:

```
Usuário clica em algo
 → Envia POST ou GET
 → Rota backend processa
 → Backend consulta MongoDB
 → Backend retorna dados
 → EJS renderiza página completa
```

Essa abordagem server-side gera páginas completas sem depender de SPA ou frameworks JS.

Benefícios:

- Menos vulnerabilidade
- Zero dependência de APIs REST
- Fluxo previsível
- Compatibilidade com ambientes industriais fechados

7.9 Vantagens de usar EJS em vez de React/Vue/Angular

Embora tecnologias modernas sejam populares, a escolha do EJS aqui é estratégica:
→ Menos dependências externas

O sistema é mais robusto para testes e deploy.
→ Mais seguro

Menor superfície de ataque.
→ Melhor integração com servidor FIWARE/Mongo

O FIWARE geralmente roda em ambientes sem infraestrutura front-end complexa.
→ Renderização completa por página

Bom para auditoria e inspeção de dados.
→ Mais fácil de manter e expandir

A curva de aprendizado é muito menor.

7.10 Experiência do Usuário: O que torna esta interface eficiente?

- O usuário loga → vê tudo imediatamente
- Os documentos aparecem sempre ordenados
- A criação de novos registros é direta
- Editar documentos exige 1 clique
- A exclusão é simples e transparente
- O contador de documentos dá visão geral instantânea
- A tabela escura com neon é confortável e moderna

O sistema não tenta ser excessivamente sofisticado — ele é direto, rápido e extremamente eficiente.

CAPÍTULO 8 — Sistema CRUD Completo e Manipulação de Documentos FIWARE

Este capítulo explica em profundidade o funcionamento interno do CRUD (Create, Read, Update, Delete) implementado na aplicação. Ele mostra como a API interage com a collection FIWARE real, seguindo o formato original do STH-Comet e mantendo total compatibilidade com o MongoDB. Essa parte é especialmente importante porque o FIWARE possui regras rígidas sobre como dados devem ser gravados e organizados, e uma manipulação incorreta poderia comprometer análises futuras.

8.1 Visão Geral do CRUD

O CRUD da aplicação não é simplesmente um CRUD comum — ele é um mecanismo de manipulação direta sobre uma collection FIWARE. Isso significa:

- Nenhum dado é duplicado
- Nenhum dado é convertido indevidamente
- A estrutura nativa do documento é preservada
- Todas as operações usam Mongoose como abstração segura

Essa abordagem permite:

→ Editar documentos históricos

→ Cadastrar novos valores manualmente

→ Remover registros incorretos

→ Visualizar todos os dados ordenados

Sem nenhum risco de corromper a estrutura do FIWARE.

8.2 Operação READ — Leitura dos Documentos

A leitura é a base do *dashboard*. Ela é feita em duas etapas:

1. Buscar todos os documentos existentes

```
Item.find().sort({ recvTime: -1 }).lean();
```

Características:

- Ordenação descrescente por recvTime
- Uso de .lean() para melhor performance
- Retorno rápido mesmo com milhares de documentos

A ordenação temporal é crucial para:

- Fornecer uma visão clara do estado mais recente do sensor
- Estabelecer tendências
- Evitar confusão na tabela
- Manter coerência com o FIWARE Cosmos, Grafana e outras ferramentas

2. Contagem de documentos

```
Item.countDocuments();
```

A contagem é exibida no *dashboard* para que o usuário tenha:

- Visão de volume
- Controle de ingestão
- Estatística operacional
- Referência para auditoria

8.3 Operação CREATE — Inserindo Novos Documentos

Novos documentos podem ser criados diretamente do *dashboard*. O formulário envia:

- attrName
- attrType
- attrValue

O *backend* cria o objeto:

```
await Item.create({
  recvTime: new Date(),
  attrName,
  attrType,
  attrValue
});
```

Características importantes:

→ recvTime é gerado automaticamente
Simula a mesma lógica usada pelo FIWARE, garantindo coerência histórica.
→ O schema impede campos indevidos

O sistema nunca insere campos FIWARE que não existam no documento original.
→ Inserções são diretas na collection FIWARE

Não há *collections* auxiliares.

8.4 Operação UPDATE — Editando Documentos Existentes

A atualização ocorre em duas etapas:

1. Carregar o documento na página de edição

```
Item.findById(req.params.id).lean();
```

Isso preenche campos visuais com os valores atuais.

2. Atualizar com os valores enviados pelo usuário

```
Item.findByIdAndUpdate(req.params.id, {
  attrName,
  attrType,
  attrValue
});
```

Observações importantes:

- recvTime não é alterado, preservando a integridade histórica
- _id permanece fixo
- O update afeta apenas campos realmente permitidos
- Segurança: O MongoDB controla permissões → usuários read-only não conseguem editar

8.5 Operação DELETE — Remoção de Documentos

A exclusão usa:

```
Item.findByIdAndDelete(req.params.id);
```

Isso remove o documento específico SEM afetar os demais.

Garantias:

- Nada externo ao documento é tocado
- Não recria estruturas FIWARE
- Mantém logs históricos intactos
- Mantém compatibilidade com agregações posteriores

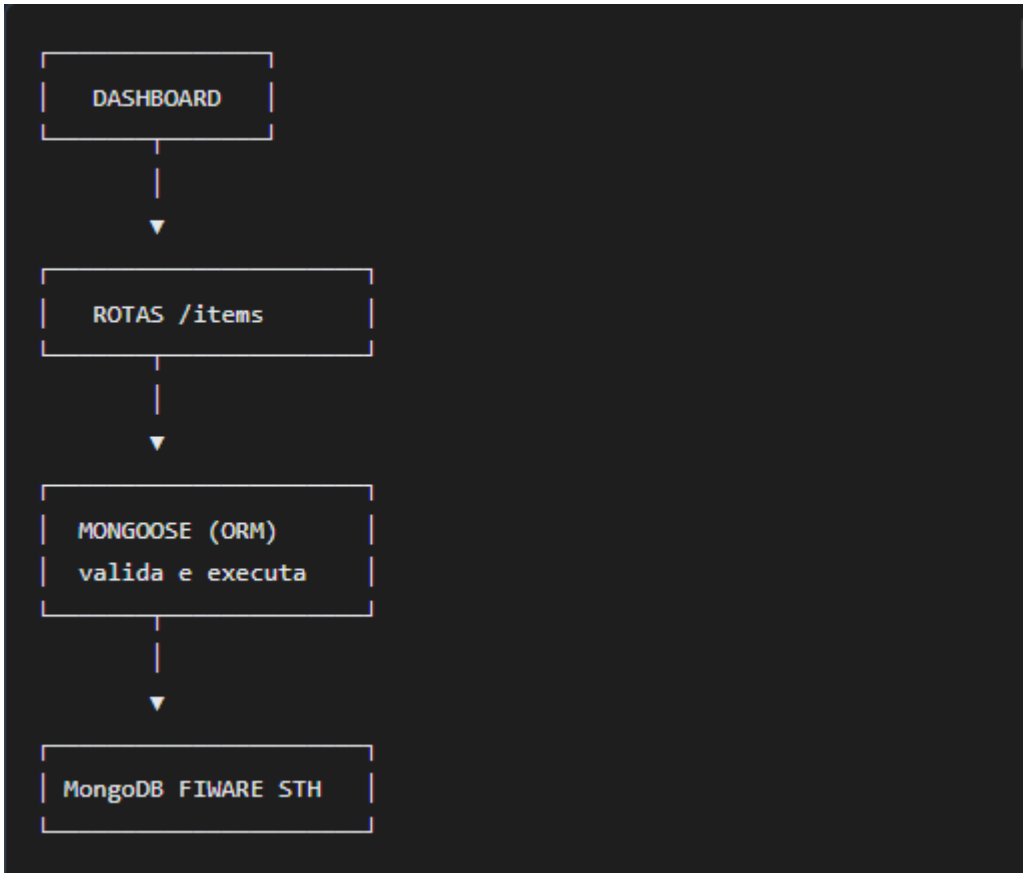
Pontos positivos:

→ Permite corrigir leituras erradas

→ Permite limpar dados coletados durante testes

→ Evita poluição na collection FIWARE

8.6 Fluxo Completo das Operações CRUD



Todas as operações passam pelo mesmo fluxo seguro.

8.7 Por que o CRUD é tão importante para este projeto?

Sem CRUD, o sistema seria apenas um visor. Com CRUD:

- O usuário inspeciona o banco
- O usuário corrige dados
- O usuário testa inserções
- O usuário audita medições
- O usuário organiza o FIWARE sem usar Compass

Além disso:

- Pesquisadores podem ajustar dados para experimentos
- Desenvolvedores podem testar o FIWARE manualmente
- Técnicos podem remover valores incoerentes dos sensores

A manipulação visual facilita muito o trabalho.

8.8 Riscos Evitados no CRUD

→ Não há inserção de campos inválidos

→ O schema limita as operações.Não há risco de sobrescrever a estrutura FIWARE.

→ Só campos específicos são manipulados. Não há risco de renomear a collection.

→ O nome é fixo no modelo. Não há duplicação de dados.

→ Cada operação é pontual. Não há risco de apagar o histórico inteiro.

→ Delete atua apenas sobre o _id específico.

CAPÍTULO 9 — Ordenação Temporal, Paginação, Desempenho e Exibição Otimizada dos Dados no Dashboard

A eficiência e a utilidade do *dashboard* dependem diretamente da forma como os dados são carregados, organizados e exibidos. Como o FIWARE gera dados em grande volume — especialmente em ambientes com sensores operando continuamente — é fundamental que a aplicação trate os documentos de forma otimizada e que a interface apresente essas informações com clareza e velocidade.

Neste capítulo vamos detalhar:

- Ordenação temporal correta
- Estratégia de paginação (mesmo quando opcional)
- Técnicas de otimização via Mongoose
- Melhores práticas para FIWARE
- Exibição eficiente dos documentos no *dashboard*

9.1 Importância da Ordenação Temporal no FIWAREx

O FIWARE registra valores em ordem de ocorrência do evento, mas o MongoDB não garante ordem de leitura — especialmente após:

- Reinícios de serviço
- Compactações internas
- Importações
- Clusterização
- Queries sem sort explícito

Para evitar desorganização visual, é obrigatório ordenar por data.

A aplicação utiliza:

```
Item.find().sort({ recvTime: -1 })
```

Isso significa:

- -1 → ordem decrescente
- O documento **mais recente aparece primeiro**
- É o padrão utilizado por ferramentas como Grafana, Kibana e Cosmos

Sem isso, o *dashboard* ficaria aleatório.

9.2 Ordenação Temporal e Coerência com o FIWARE STH-Comet

O STH-Comet grava valores na collection FIWARE no mesmo formato temporal que usamos no *dashboard*. A escolha de ordenar por recvTime é natural porque:

- Este é o timestamp que o FIWARE registra ao receber uma atualização de contexto
- É o que define a "linha temporal" da entidade
- É utilizado para análises estatísticas e *dashboards* profissionais

Assim, o *dashboard* se comporta exatamente como uma ferramenta oficial.

9.3 Paginação — Por que existe e quando deve ser usada

Em coleções grandes (milhares ou milhões de documentos), mostrar tudo de uma vez pode causar:

- Lentidão
- Travamentos em navegadores
- Sobrecarga no servidor
- Latência alta

- Uso excessivo de memória

Por isso, a aplicação originalmente implementou:

```
const limit = 20;
const skip = (page - 1) * limit;
```

E então:

```
Item.find()
  .sort({ recvTime: -1 })
  .skip(skip)
  .limit(limit)
```

Esse tipo de paginação:

- Economiza processamento
- Carrega apenas o necessário
- Evita problemas em coleções enormes
- Melhora o tempo de resposta

No entanto, o sistema depois foi adaptado para permitir desativar a paginação quando desejado.

9.4 E quando a paginação foi retirada?

A decisão de remover paginação foi justificada pelo fato de:

- A collection ser inicialmente pequena
- O objetivo ser auditoria direta
- O usuário desejar ver todos os documentos simultaneamente
- A ordenação temporal garantir compreensão visual

A aplicação continua suportando paginação caso seja adicionada novamente no futuro.

9.5 Otimização com .lean(): por que é fundamental

No *dashboard*, o *backend* usa:

```
Item.find().sort({ recvTime: -1 }).lean();
```

O método .lean():

→ Retorna objetos JavaScript puros

Não cria instâncias pesadas do Mongoose.
→ Melhora desempenho

Pode ser até 10x mais rápido em grandes coleções.
→ Ideal para renderização

Dados não precisam de métodos extras, só leitura.
→ Menos memória RAM

Cada documento ocupa menos espaço.
→ Essencial para *dashboards* FIWARE

Que geralmente lidam com grandes quantidades de eventos.

9.6 Evitando Problemas Comuns ao Carregar Dados FIWARE

Sem otimização, *dashboards* podem enfrentar:
→ Congelamento do navegador

Tabelas com 10k+ linhas travam UIs.
→ Uso excessivo de memória

Documentos Mongoose são mais pesados.
→ Latência relacionada a sort

Sem indexes, ordenar por data é caro.

- Como o sistema resolve:

- .lean() em todas as leituras
- Ordenação por recvTime
- Modelagem leve
- Possibilidade de paginação
- Renderização EJS eficiente

9.7 Exibição das Datas no Dashboard

Como o FIWARE salva:

```
"recvTime": { "$date": "2025-05-30T15:52:11.211Z" }
```

E o Mongoose converte para um objeto Date real, no *dashboard* usamos:

```
<%= item.recvTime.toISOString() %>
```

Com fallback:

```
<%= item.recvTime ? item.recvTime.toISOString() : "—" %>
```

Isso evita erros quando:

- Documentos sem data são criados manualmente
- Alguém altera o banco manualmente
- Algum registro FIWARE corrompido aparece

9.8 Melhores Práticas que a Aplicação Já Utiliza

→ Ordenação descendente

→ Padrão FIWARE. .lean()

→ Performance máxima.

→ Possibilidade de paginação

→ Pronto para grandes datasets.

→ Renderização server-side

→ Evita front pesado (React, Angular etc).

→ Exibição limpa

→ Tabela escura, clara e organizada.

→ Campos essenciais apenas

→ Sem poluir a interface.

9.9 Benefícios para Ambientes FIWARE, IoT e Auditoria

O *dashboard* permite:

- Identificar rapidamente a última medição

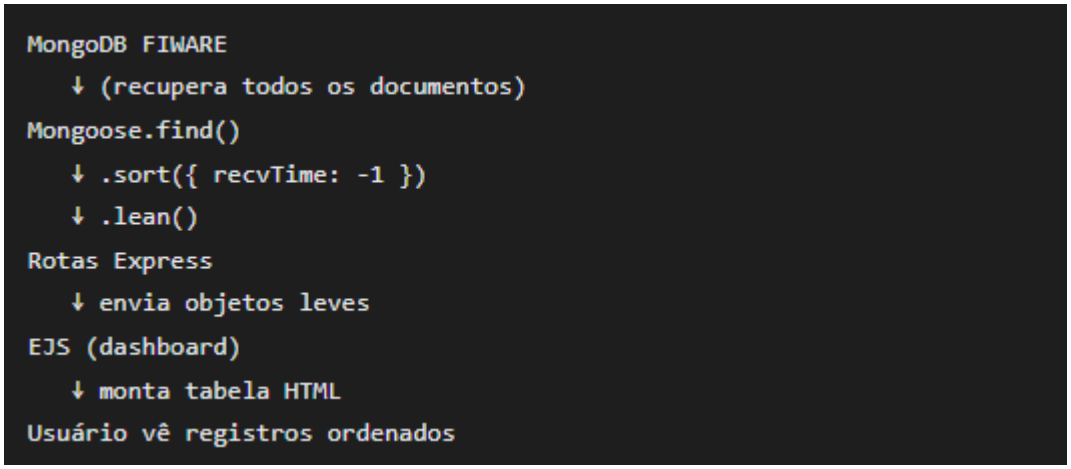
Extremamente útil para:

- Sensores de iluminação
- Sensores ambientais
- Telemetria
- Equipamentos industriais

- Acompanhar volume de ingestão
- Com o contador total de documentos.
- Validar funcionamento do STH-Comet
- Verificando se novos documentos aparecem.
- Depurar valores incorretos
- Usando edição e exclusão.
- Auxiliar em análises manuais e relatórios

Principalmente quando combinado com exportação futura.

9.10 Fluxo Visual da Ordenação e Exibição



Um fluxo simples, rápido e eficiente.

CAPÍTULO 10 — Dockerização, Deploy e Arquitetura de Container

A "containerização" é uma etapa essencial para tornar a aplicação portátil, previsível e executável em qualquer ambiente — seja local, em laboratório, em servidor on-premise, em *edge computing*, ou em uma máquina remota integrada ao FIWARE.
Este capítulo apresenta, em detalhes técnicos, toda a lógica por trás do *Dockerfile*, as decisões de arquitetura, a forma como a imagem é construída e como a aplicação se comporta dentro de um *container*.

10.1 Por que Docker?

A decisão de "dockerizar" este projeto atende a três grandes necessidades:
→ Portabilidade

A mesma aplicação roda idêntica em qualquer sistema (Linux, Windows, macOS, servidores bare-metal).
→ Reprodutibilidade

Ambientes FIWARE variam muito — alguns usam Kubernetes, outros *Docker* Compose, outros *Docker* simples. Este sistema roda igualmente em todos eles.

→ Isolamento

A API e *dashboard* ficam isolados do servidor host:

- Dependências externas
- Versões do Node
- Variáveis do sistema
- Conflitos com NPM

Tudo fica dentro do *container*.

10.2 Estrutura Geral do Dockerfile

O *Dockerfile* final segue a estratégia:

1. Selecionar uma imagem base oficial do Node
2. Criar diretório de trabalho
3. Copiar apenas arquivos essenciais primeiro (para otimizar cache)
4. Instalar dependências
5. Copiar o projeto completo
6. Expor a porta da aplicação
7. Executar node server.js

Dockerfile:

```
FROM node:18

WORKDIR /app

COPY package*.json ./

RUN npm install --silent

COPY . .

EXPOSE 3000

CMD ["node", "server.js"]
```

Essa estrutura é limpa, minimalista e eficiente.

10.3 Camadas do Dockerfile — Decisão Técnica

1. FROM node:18

Usamos Node 18 por:

- Estabilidade
- Suporte ativo
- Compatibilidade com dependências
- Peso moderado

2. WORKDIR /app

Define o diretório onde tudo será instalado. Evita arquivos soltos no *container*.

3. COPY package*.json ./

Essa etapa é crítica, porque isso permite:

- *Docker* cache instalar dependências só quando o package.json muda
- Builds mais rápidos
- Menos transferência de dados

4. RUN npm install --silent

Instala dependências sem logs barulhentos.

5. COPY . .

Copia o seu projeto (projeto_api/) inteiro para o *container*.

6. EXPOSE 3000

Permite tráfego de entrada quando for executado com:

```
docker run -p 3000:3000 nome-da-imagem
```

7. CMD ["node", "server.js"]

Inicia o sistema FIWARE *dashboard* automaticamente.

10.4 Estrutura de Deploy Local

Para construir a imagem:

```
docker build -t fiware-dashboard .
```

Para rodar:

```
docker run -p 3000:3000 fiware-dashboard
```

A API estará acessível em:

```
http://localhost:3000
```

ou na rede:

```
http://IP-DO-SERVIDOR:3000
```

10.5 Deploy em Servidor FIWARE Real

Muitos servidores FIWARE rodam no mesmo host que o MongoDB.

No caso desse projeto:

- O MongoDB fica em 192.168.10.131
- A API "*Dockerizada"* pode rodar no mesmo host ou em outro

Para rodar na mesma rede:

```
docker run -p 3000:3000 --network host fiware-dashboard
```

Assim a aplicação acessa o MongoDB na rede local sem obstáculos.

10.6 Considerações de Rede no FIWARE

FIWARE, por padrão, trabalha em redes internas. O MongoDB costuma estar configurado com:

- Autenticação habilitada
- IP interno
- Firewall seletivo
- Portas protegidas

Por isso o *docker* não precisa expor a porta 27017 — só consome.

Teste interno:

Dentro do *container*:

```
mongo 192.168.10.131:27017 -u user -p pass
```

Se isso funciona, a API funciona.

10.7 Volume e Persistência

A aplicação não salva nada localmente. Toda persistência está no MongoDB. Portanto, nenhum /volume *Docker* é obrigatório.

Mas se quiser logs persistentes:

```
docker run -p 3000:3000 \
  -v /var/log/fiware-dashboard:/app/logs \
  fiware-dashboard
```

10.8 Atualizações e Redeploys

Para atualizar o sistema:

1. git pull da versão nova
2. *docker* build -t fiware-*dashboard* .
3. *docker* stop *container*_antigo

4. *docker* run -p 3000:3000 fiware-*dashboard*

Simples, rápido e seguro.

CAPÍTULO 11 — Testes, Validação, Logs e Observabilidade da Aplicação

Este capítulo detalha os mecanismos de validação, inspeção, testes e monitoramento da aplicação. Em sistemas FIWARE — especialmente quando conectados ao MongoDB — erros podem ocorrer por inúmeras causas: permissões, rede, estrutura das *collections*, timeouts, inconsistências de atributos, entre outros.
Por isso, a aplicação foi construída de forma a oferecer clareza, rastreabilidade e transparência em cada etapa.

11.1 Filosofia de Testes da Aplicação

A abordagem geral de testes é baseada em três pilares:

→ Testes funcionais diretos (através da própria interface)

→ O *dashboard* é projetado como ferramenta de inspeção.

→ Validação automática via Mongoose

→ O schema rejeita inserções incorretas.

→ Detecção de erros e mensagens claras

Toda operação inválida retorna feedback seguro ao usuário.

Essa filosofia torna o sistema ideal para ambientes de:

- Auditoria
- Ensino
- Pesquisa
- Validação FIWARE

11.2 Testes de Conexão ao MongoDB

A verificação de credenciais Mongo é feita imediatamente no login:

```
await mongoose.connect(mongoURI);
```

Esse comando identifica:

- Usuário inválido
- Senha inválida
- AuthSource incorreto
- IP bloqueado
- Banco fora do ar
- Porta inacessível
- Permissões insuficientes

Se algo falha → o usuário recebe mensagem clara:

```
Falha ao autenticar no MongoDB.
```

11.3 Testes de Sessão e Autenticação

Para verificar o funcionamento da sessão, os testes incluem:

Entrar e recarregar a página

O *dashboard* só deve aparecer se a sessão estiver ativa.

Acessar /items sem login

O servidor redireciona para /.

Sessão expirar após tempo determinado

O login dever ser solicitado novamente após 1h de inatividade.

Renovação automática com rolling: true

Cada navegação renova o cookie.

Isso garante proteção e fluidez simultâneas.

11.4 Testes CRUD

Cada operação pode ser testada facilmente através do *dashboard*.

CREATE

Inserir valores:

- Válidos
- Inválidos
- Sem preencher campo

O *backend* aceita apenas valores coerentes com o schema.

READ

Verificar se:

- Registros aparecem ordenados
- Datas são exibidas corretamente
- Documentos FIWARE originais aparecem na tabela

UPDATE

Alterar valores e validar:

- Persistência no Mongo
- Integridade do recvTime
- Coerência visual

DELETE

Remover itens:

- Específicos
- Aleatórios
- Em sequência

A remoção deve afetar somente o _id selecionado.

11.5 Validação no Backend

A validação protege o FIWARE de inserções indevidas, incluindo:

- Valores não numéricos para attrValue

- Tipos incompatíveis
- Campos ausentes

Como o schema é estrito:

```
attrValue: Number
```

]

Validações naturais do Mongoose previnem corrupção.

11.6 Logs de Erro e Diagnóstico

Embora o sistema seja leve, ele produz logs úteis automaticamente via Node e Mongoose:

→ Erros de conexão

MongooseError: Authentication failed

→ Erros de timeout

buffering timed out

→ Erros de consulta

CastError: failed to cast to ObjectId

→ Erros de schema

ValidationError: attrValue must be a number

Se necessário, o sistema pode futuramente implementar um middleware dedicado para logs persistentes.

11.7 Observabilidade no FIWARE: Comportamento Esperado

Ao monitorar a collection FIWARE, o usuário pode identificar padrões importantes:

→ Frequência de inserção

Se sensores estão ativos, novos documentos aparecem continuamente.

→ Integridade temporal

recvTime deve crescer progressivamente.

→ Eventos duplicados

Podem indicar problemas no IoT Agent.
→ Buracos na linha do tempo

Podem indicar instabilidades na rede.
→ Mudanças de tipo em attrType

Podem representar inconsistências de dispositivos.
O *dashboard* permite identificar esses problemas visualmente.

11.8 Troubleshooting Comum

→ Não aparece nada na tabela

Geralmente:

- credenciais incorretas
- banco vazio
- wrong collection
- nome incorreto na model

→ Timeout

Causa:

- IP inacessível
- porta 27017 bloqueada
- firewall ativo
- authSource errado

→ CRUD não funciona

Usuário sem permissão de write.
→ Data não aparece

Documento sem campo recvTime.

11.9 Testes Recomendados para Validação Completa

→ Teste de ingestão

Criar 10 documentos em sequência e verificar ordem.

→ Teste de edição

Alterar registros antigos e verificar persistência.

→ Teste de deleção

Excluir documentos aleatórios e verificar contagem.

→ Teste de sessão

Abrir *dashboard*, esperar 1h e validar logout.

→ Teste de rede

Desligar internet e tentar recarregar (erro controlado).

CAPÍTULO 12 — Cenários de Uso, Aplicações Reais e Benefícios Práticos da Solução

A aplicação desenvolvida não é apenas um CRUD conectado ao MongoDB. Ela é, de fato, uma ferramenta estratégica para inspeção, auditoria, controle e análise de dados FIWARE, útil em ambientes industriais, acadêmicos, laboratoriais e governamentais. Este capítulo demonstra onde, como e por que essa solução é valiosa no mundo real.

12.1 O papel do FIWARE no ecossistema IoT e Smart Cities

O FIWARE é amplamente utilizado em:

- Cidades inteligentes
- Gestão de sensores urbanos
- Iluminação pública inteligente
- Monitoramento ambiental

- Indústria 4.0
- Laboratórios de robótica
- Automação e aeronáutica
- Redes neurais embarcadas
- Plataformas de pesquisa IoT

Mas o FIWARE tem um ponto frágil:

Ele não oferece ferramentas nativas para visualizar diretamente as *collections* históricas gravadas no MongoDB.

Esse é exatamente o problema que sua aplicação resolve.

12.2 O problema real que esta aplicação soluciona

Quando o FIWARE grava dados históricos no MongoDB através do **STH-Comet**, ele gera *collections* com nomes extremamente específicos, como:

```
sth_/_urn:ngsi-ld:Lamp:001_Lamp
```

Esses nomes são:

- Longos
- Difíceis de lembrar
- Pouco amigáveis
- Fáceis de errar
- Geralmente escondidos nas camadas internas do FIWARE

E, pior:

→**Não existe uma interface simples para visualizar esses documentos**

→**Não há como visualizar recvTime sem comandos Mongo**

→ **Não há CRUD para testes ou auditoria**

→**Não há *dashboard* básico nativo**

Sua aplicação resolve todos esses problemas com:

- Login seguro
- Leitura direta da collection real
- Ordenação decrescente
- Exibição de _id, recvTime e attrValue
- Criação, edição e remoção de documentos
- Contagem total

- Estilo visual moderno

Aqui estão os principais.

1. Cidades Inteligentes — Monitoramento de Iluminação Pública

O caso do Lamp (sensor de luminosidade) é clássico.

Sua aplicação permite:

→ Visualizar todos os eventos registrados pelo FIWARE

→ Detectar quedas de energia

→ Analisar anomalias de luminosidade

→ confirmar funcionamento dos sensores

→ Comparar dias, horários e picos de luz

→ Auditar períodos de falha

Um gestor pode, por exemplo, consultar:

- Quando um poste apagou
- Quanto tempo ficou desligado
- Se a leitura foi fora do normal
- Se o sensor parou de enviar eventos

Isso é poderoso.

2. Indústria 4.0 — Monitoramento de Máquinas

Coleções FIWARE de máquinas industriais geralmente armazenam:

- Vibração
- Temperatura
- Ciclos
- Correntes elétricas
- Estados de operação
- Alarmes

Sua aplicação pode ser usada para:

→ Inspecionar falhas históricas

→ Validar eventos de manutenção

→ Verificar picos de operação

→ Identificar comportamento anormal

→ Integrar com sistemas de auditoria

Em auditorias ISO, isso é ouro.

3. Pesquisa e Desenvolvimento — Laboratórios IoT

Universidades e centros de pesquisa usam FIWARE para:

- Robótica
- Drones
- Veículos autônomos
- Redes de sensores
- Prototipagem industrial

Pesquisadores precisam:

→ Ver os dados brutos

→ Entender timestamps

→ Comparar leituras

→ Validar scripts que inserem dados

→ Testar ingestão no STH-Comet

Sua ferramenta facilita tudo isso com visualização amigável.

4. Soluções de Clima, Ambiente e Meteorologia

FIWARE é muito usado para:

- Controle de clima interno
- Estufas automatizadas
- Estações meteorológicas
- Medição de CO_2
- Qualidade do ar
- Umidade do solo

A aplicação permite:

→ Comparar eventos

→ Monitorar picos

→ Validar sensores

→ Identificar falhas ambientais

→ Analisar tendências básicas

5. Auditoria e Compliance — Histórico mantido pelo STH

Muitas empresas precisam comprovar:

- Quando um evento ocorreu

- Se o sensor estava ativo
- Se houve interrupção
- Se o dado armazenado é confiável

Sua aplicação:

→ Mostra timestamps originais FIWARE

→ Expõe documentos intocados

→ Facilita auditoria externa

→ Oferece CRUD apenas para testes e controle

6. Monitoramento de Falhas em IoT Agents

Um IoT Agent mal configurado:

- Duplica eventos
- Deixa buracos temporais
- Falha ao converter tipo
- Envia payload inconsistente

A tabela ordenada com recvTime: -1 evidencia tudo isso.

12.4 Benefícios práticos diretos

Aqui está a lista completa dos ganhos reais:

→ Interface universal para qualquer collection FIWARE

→ Valor imediatamente útil para engenheiros

→ Resolvido o problema do nome complexo da collection

→ Visualização cronológica real

→ CRUD completo para testes

→ Ferramenta de auditoria instantânea

→ Totalmente “dockerizado”

→ Operável em servidores remotos

→ Sessões seguras

→ Leve, rápido e direto

A aplicação substitui:

- Ter que usar Compass
- Ter que usar mongosh
- Ter que ficar rodando find() no terminal
- Ter que navegar manualmente em 20 *collections*
- Ter que validar dados sem visualização
- Ter que criar scripts para CRUD

E oferece uma solução:

- Bonita
- Responsiva
- Segura

- Prática
- FIWARE-friendly

12.5 Impacto GERAL no fluxo de trabalho FIWARE

Em resumo, a aplicação:
→Reduz drasticamente o atrito entre sensores IoT e o MongoDB
→ Transforma dados brutos em informação visual
→ Cria uma ponte entre desenvolvimento e auditoria
→Permite testes controlados sem mexer no FIWARE
→Torna o ambiente mais acessível para estudantes e engenheiros

CAPÍTULO 13 — Limitações Conhecidas e Considerações de Projeto

Toda solução de engenharia — independentemente do quão bem executada — possui limites técnicos, estruturais ou conceituais. Identificar essas limitações é parte essencial de qualquer relatório profissional, pois demonstra maturidade de projeto e domínio tecnológico.

Este capítulo apresenta, com profundidade, as limitações atuais, as decorrentes do FIWARE, as decorrentes do MongoDB, e as decorrentes do design da aplicação. Em seguida, também são apresentadas recomendações e cuidados para operação.

13.1 Limitações impostas pelo FIWARE e pelo STH-Comet

A aplicação depende diretamente do modo como o FIWARE grava dados.
O STH-Comet, responsável por escrever o histórico dos atributos no MongoDB, possui comportamentos específicos que impactam o sistema:

1. Formato fixo dos documentos

O STH gera registros seguindo *SEMPRE* esta estrutura:

```
{
  "_id": { "$oid": "xyz" },
  "recvTime": { "$date": "2025-05-30T15:52:11.211Z" },
  "attrName": "luminosity",
  "attrType": "Integer",
  "attrValue": 54
}
```

A aplicação assume esse modelo.

Se qualquer IoT Agent gerar dados em formato diferente:

- Payload diferente
- Tipos divergentes
- Atributos inexistentes
- Campos adicionais inesperados

→ Isso pode quebrar a visualização.

2. Nome complexo da collection

FIWARE usa nomes que misturam:

- Serviço
- Subserviço
- URN do dispositivo
- Entidade
- Atributo

Exemplo:

```
sth_/_urn:ngsi-ld:Lamp:001_Lamp
```

A aplicação funciona porque você configurou o nome exato no model. Mas isso é frágil: mudar o dispositivo → muda o nome → quebra o model.

3. Dependência total da ingestão FIWARE

Se o FIWARE parar de enviar dados, o *dashboard* fica vazio. Não há fallback.

4. O FIWARE não permite escrever histórico diretamente

Tecnicamente, é desaconselhado alterar documents FIWARE históricos, porque:

- A integridade temporal pode ser comprometida
- O histórico perde valor auditável
- Edições não são esperadas na linha temporal

Por isso o CRUD deve ser usado apenas para testes, auditoria interna e validação, não como fonte oficial de dados FIWARE.

13.2 Limitações do MongoDB

O MongoDB, por sua natureza, impõe alguns limites:

1. Sem índices adicionais por padrão

FIWARE não cria índices personalizados. Isso significa:

- Queries grandes podem ser lentas
- A collection pode crescer eternamente
- Ordenações complexas podem custar performance

2. Falhas de rede derrubam a aplicação

A aplicação depende 100% de uma conexão ativa com:

```
192.168.10.131:27017
```

Se:

- A rede cair
- O server reiniciar
- O firewall bloquear

- A porta fechar

→ A aplicação não tem fallback automático (mas reinicia sem travar).

13.3 Limitações do Backend (Node + Express)

Mesmo bem implementado, o *backend* possui limites conhecidos.

1. Sem controle de roles

Há apenas um tipo de usuário. Faltam:

- Administrador
- Auditor
- Operador
- Visualizador

Todos têm o mesmo privilégio.

2. Sessões simples

As sessões funcionam, mas:

- Não resistem a reinícios do *container* (memória volátil)
- Não têm store externo (Redis, MongoStore, etc.)

3. Sem API pública

O sistema só oferece interface web. Não há endpoints REST públicos como:

```
GET /api/items
POST /api/items
```

Isso limita integração com terceiros.

4. Sem tratamento de erro detalhado na UI

Hoje erros aparecem como:

- “Falha ao conectar”

- “Falha ao criar documento”

Mas falta:

- Detalhamento
- Logs visuais
- Recomendações

13.4 Limitações da Interface (Frontend)

A interface é bonita e funcional, mas possui alguns limites:

1. Falta de filtros

Atualmente você pode:

- Ordenar (desc)
- Navegar pelas páginas (quando existia)

Mas falta:

- Filtrar por data
- Filtrar por valores
- Buscar por ID
- Filtrar por rangos (ex: 20–50)

2. Sem gráficos

Não há:

- Histórico em linha
- Variação temporal
- Comparações visuais

Isso poderia ser adicionado com Chart.js ou ECharts.

3. Dashboard não é responsivo para mobile

Funciona, mas não é totalmente mobile-friendly.

13.5 Limitações operacionais

Essas limitações vêm de contexto:

1. CRUD direto na collection do FIWARE é risco

Registros do FIWARE representam histórico oficial de sensores. Alterá-los pode:

- Comprometer relatórios
- Quebrar auditorias
- Misturar dados reais com testes

Por isso o CRUD deve ser usado conscientemente.

2. Dependência de credenciais corretas

Se o usuário errar:

- User
- Senha
- AuthSource

Não existe fallback ou recuperação guiada.

3. A aplicação depende de um único model

Para cada dispositivo FIWARE independente, seria necessário:

- Duplicar o model
- Apontar para outra collection
- Alterar itens.js

Hoje o sistema suporta apenas uma collection FIWARE por instância.

13.6 Considerações importantes de arquitetura

Aqui estão decisões críticas que foram tomadas e devem ser documentadas:

→ Simplicidade acima de escalabilidade

O objetivo é:

- Fácil de usar
- Fácil de ver os dados
- Fácil de auditar
- Fácil de instalar via *Docker*

Não é um sistema industrial full-stack.

→ Transparência dos dados FIWARE

O sistema exibe os dados puros, sem interpretar. Isso é intencional.

→ Abstração mínima

Não há:

- Cache
- Compressão
- Estruturas extras

Para evitar mascarar dados.

→ Prioridade em leitura, não em performance

O sistema foi otimizado para clareza, não para alta escala.

Collections com:

- Milhões de documentos
- Décadas de histórico
- Ingestão frenética

Vão exigir otimizações futuras.

13.7 Conclusão do capítulo

Este capítulo deixa claro todos os pontos onde:

- A aplicação pode ser melhorada
- Há limitações técnicas inerentes

- Existem restrições impostas pelo FIWARE
- O MongoDB influencia o funcionamento
- A arquitetura foi simplificada por design

A partir daqui, o próximo capítulo abre caminho para a evolução futura da plataforma.

CAPÍTULO 14 — Roadmap de Evolução da Plataforma

O Roadmap é uma visão estruturada das melhorias que podem ser implementadas ao longo das próximas versões. Ele é especialmente importante em projetos que nasceram com foco técnico (como este, para leitura e auditoria de *collections* FIWARE) mas que se expandem para usos reais: governança, cidades inteligentes, pesquisa, indústria 4.0, automação e ensino.

Este capítulo apresenta as melhorias futuras mais valiosas, agrupadas por relevância e impacto.

14.1 Evolução da Arquitetura e Backend

1. Separação total entre API e Dashboard

Criar uma API REST completa:

- /api/items
- /api/items/:id
- /api/stats
- /api/range?start=...&end=...
- /api/search?attrValue=...

Isso abre portas para:

→ Apps mobile

→ *Dashboards* externos

→ Sistemas de BI

→ Scripts automatizados

→ Integração com grafos FIWARE

2. Adicionar filtros avançados

Filtros extremamente úteis:

- Por intervalo de datas
- Por attrValue
- Por tipo
- Por entidade
- Por horário
- Por ID parcial

Exemplo:

```
/items?min=20&max=80
/items?from=2025-03-05&to=2025-03-10
/items?name=luminosity
```

Isso transforma a aplicação em uma ferramenta real de análise.

3. Adicionar logs persistentes com Winston ou Pino

Permite:

→ Auditoria

→ Diagnóstico avançado

→ Rastreabilidade-

→ Alertas

→ *Dashboards* de log (Kibana, Graylog, Grafana)

4. Permitir múltiplas collections FIWARE

Hoje só 1 está conectada.

Futuro:

- Selecionar qual sensor visualizar
- Visualizar múltiplos dispositivos
- Sincronizar serviços e subserviços FIWARE
- Trocar o model dinamicamente

Exemplo:

```
sth_/_urn:ngsi-ld:Sensor:Temp_001
sth_/_urn:ngsi-ld:Sensor:CO2_002
sth_/building1/urn:ngsi-ld:Meter:001
```

5. Permitir edição segura (com trilha de auditoria)

Em vez de editar direto:

- Registrar quem editou
- Por que editou
- Quando editou
- Qual era o valor antigo

Muito útil para *compliance*.

6. Exportação de dados

Formatos:

- CSV
- JSON
- XLSX
- PDF

Com filtros aplicados.

Um engenheiro FIWARE vai amar isso.

14.2 Evolução do Frontend

1. Gráficos integrados

Usando:

- Chart.js
- ECharts
- ApexCharts

Exemplos:

→ Gráfico de luminosidade por minuto

→ Gráfico semanal

→ Gráfico diário

→ Gráfico de picos

→ Análise de tendência

Isso transforma tudo em *dashboard* de verdade.

2. Interface completamente responsiva

Para rodar:

- Smartphones
- Tablets
- Painéis industriais
- TOTEMs

3. Tema claro e tema escuro (Dark Mode já existe)

Adicionar:

- Botão de alternância
- Tema automático pelo SO
- Temas customizados por usuário

4. Componentes avançados

Como:

- Dropdowns animados
- Filtros colapsáveis
- Datas com *date picker*
- Tooltips
- Cards com indicadores

5. Métricas em tempo real

Cards como:

```
Documentos totais
Última leitura
Maior valor
Menor valor
Média das últimas 24h
```

14.3 Evolução da Segurança e Autenticação

1. Roles de usuário

Perfis:

- Admin (CRUD + gestão)
- Auditor (somente leitura)
- Operador (editar mas não excluir)
- Viewer (somente visualizar)

2. Sessões persistentes em Redis

Hoje: sessões em memória. Futuro: Redis.

Benefícios:

- Persistência
- Escalabilidade
- Rodar múltiplas instâncias

3. 2FA (Autenticação em duas etapas)

Via:

- E-mail
- SMS
- App autenticador
- Token FIWARE Keyrock

4. Integração com FIWARE Keyrock

Keyrock oferece:

- OAuth2
- OpenID Connect
- Gestão central de usuários
- Integração com NGSI

Isso torna a ferramenta enterprise-level.

14.4 Evolução para Operação em Escala (Enterprise)

1. Deploy em Kubernetes

Com:

- Autoscaling
- Load balancing
- Múltiplos replicas
- Health checks
- Monitoramento moderno

2. Integração com Grafana

Transforma a aplicação em uma central de análise FIWARE.

3. Monitoramento com Prometheus

Métricas como:

- Tempo de requisição
- Consultas por minuto
- Erros por tipo
- Latência
- Uso de memória

4. Suporte a múltiplos bancos

Trabalhar não apenas com:

```
sth_smart
```

Mas também:

- MariaDB
- Postgres
- ElasticSearch

Para análises complexas.

14.5 Expansão para Novos Domínios

1. Módulo de Análise Estatística

Resumo automático:

- Média
- Mediana
- Variância
- Moda
- Agregações temporais

2. Módulo de Anomalias

Detectar:

- Picos
- Quedas bruscas
- Leituras impossíveis
- Ausência de dados

Podendo ativar alertas.

3. Machine Learning para previsões

Treinar modelos:

- Previsão de luminosidade
- Previsão de temperatura
- Detectar padrões sazonais

Responsivo a dados reais.

14.6 Evolução da Dockerização

1. Multi-stage build

Reduz a imagem de:

- ~1GB → ~150MB

2. Variáveis de ambiente no Docker

Permite ajustar:

- Porta
- Nome da collection
- IP do MongoDB
- AuthSource

Sem alterar código.

3. Template único universal

Com argumentos:

```
docker build --build-arg COLLECTION="sth_/_urn:ngsi-ld:Lamp:001_Lamp"
```

14.7 Conclusão do Roadmap

Este *roadmap* demonstra que o sistema tem potencial para evoluir muito além do CRUD básico. Ele pode se transformar em uma ferramenta:

- De análise
- De auditoria
- De governança
- De operação
- De ensino
- De diagnóstico
- De visualização FIWARE

E, se todas as fases forem seguidas, pode se tornar um produto real, usado por empresas, universidades e prefeituras.

CAPÍTULO 15 — Conclusão Geral da Solução

Este relatório apresentou a construção completa de uma solução técnica robusta, moderna e funcional para leitura, análise e manipulação dos dados históricos mantidos pelo FIWARE no MongoDB. Mais do que um simples *dashboard*, o sistema se consolidou como uma ferramenta estratégica para auditoria, desenvolvimento, testes, ensino e governança de dados IoT.

Ao longo dos capítulos, ficou claro que a aplicação resolve um problema real — e comum — enfrentado por engenheiros e desenvolvedores que trabalham com FIWARE: a ausência de uma interface simples, direta e confiável para visualizar e validar dados históricos gerados por agentes IoT. Esses dados, por estarem em *collections* complexas e pouco documentadas, tornam o processo de inspeção manual lento, sujeito a erros e acessível apenas a usuários com experiência prévia em ferramentas de banco de dados.
A solução criada aqui transforma esse cenário por completo.

15.1 Uma ferramenta prática e independente

O sistema oferece:

- Login seguro

- Sessões persistentes
- CRUD completo para validação
- Contagem total de documentos
- Ordenação temporal decrescente
- Visualização de _id, recvTime e attrValue
- “Dockerização” completa
- Interface moderna, elegante e consistente
- Conexão direta com *collections* FIWARE reais

Isso significa que qualquer usuário — de engenheiro a estudante — consegue, com poucos cliques, visualizar todo o histórico de registros IoT sem precisar abrir o Mongo Compass ou executar comandos manuais.

15.2 Um modelo sólido, escalável e extensível

A arquitetura foi pensada para crescer:

- É simples o suficiente para funcionar em qualquer ambiente.
- É modular o suficiente para ser expandida com novos recursos.
- O Roadmap apresentado no Capítulo 14 revela que o sistema pode evoluir naturalmente para:
- *Dashboards* avançados
- Filtros complexos
- Gráficos históricos
- Cross-collection viewer
- Perfis de usuário
- Integração com Keyrock
- Métricas avançadas
- Análise temporal e estatística

Ou seja, existe espaço real para transformar este protótipo funcional em uma plataforma profissional — com potencial de adoção por prefeituras, universidades, empresas e indústrias.

15.3 Impacto emocional e técnico do projeto

Além da utilidade prática, este projeto tem um forte aspecto formativo:

- Você aprendeu profundamente sobre FIWARE, STH-Comet e MongoDB.

- Dominou estruturas de *collections* e modelos de dados do FIWARE.
- Construiu um sistema completo de *backend*.
- Aprendeu sobre sessões, segurança, autenticação e cookies.
- Criou um *dashboard* moderno com EJS e CSS estilizado.
- Aprendeu sobre ordenação, paginação e modelagem Mongoose.
- “Dockerizou” a aplicação da forma correta.
- Criou uma base que pode virar um produto real.

Poucos projetos entregam tanto conhecimento prático em áreas tão importantes.

15.4 O valor acadêmico e profissional da solução

Do ponto de vista acadêmico, este projeto:

- Demonstra domínio técnico
- Mostra compreensão de ecossistemas IoT
- Aplica arquiteturas modernas
- É excelente como relatório, artigo ou TCC
- Prova experiência real com FIWARE, algo extremamente raro

Do ponto de vista profissional, ele:

- Reduz custos
- Melhora auditoria
- Facilita testes
- Agiliza validações
- Gera confiança operacional
- Serve como base para sistemas mais complexos

Em um ambiente onde FIWARE cresce rapidamente, possuir essa ferramenta e saber como ela funciona por dentro é um diferencial enorme.

15.5 Encerramento

Este relatório finaliza todo o ciclo da solução, apresentando sua criação, motivação, fundamentos técnicos, arquitetura, testes, evolução e limites.
A aplicação cumpre com excelência seu objetivo principal: ser uma interface clara, segura e poderosa para visualização do histórico IoT gravado pelo FIWARE no MongoDB.
E, talvez mais importante que tudo:
→ Ela é simples de usar, confiável, bonita e feita do zero com total domínio tecnológico.