ATIVIDADE	PROJETO INTEGRADOR (PWBE)

CONTEXTO e DESCRIÇÃO

A escola **TecnoVille** está desenvolvendo um projeto de transformação urbana baseado no conceito de **Smart City**. A ideia é implementar sensores em pontos estratégicos da cidade para coletar dados em tempo real sobre:

- **Temperatura**
- **Umidade**
- **Luminosidade**
- **Contador**

Esses sensores serão instalados em locais como praças, corredores, pátios etc.

A gestão escolar contratou sua equipe técnica para desenvolver **o back end** do **sistema de monitoramento** usando **Django e Django Rest Framework**. Os dados dos sensores serão enviados para a API e posteriormente visualizados por um painel web.

Desafio proposto

Você deve **modelar e desenvolver o back end completo da aplicação**, com base nos seguintes requisitos:

Requisitos Funcionais

- O sistema deve permitir **registrar sensores**, com campos como:
  - ID do sensor
  - Sensor (temperatura, umidade, luminosidade, contador)
  - Identificação (mac-address)
  - Localização (latitude e longitude)
  - Status operacional (ativo/inativo)
- Deve ser possível **registrar medições** vindas dos sensores:
  - ID do sensor relacionado
  - Ambiente (relacionado com a tabela de ambientes)
  - Valor da medição
  - Data e hora da leitura (timestamp)
- Deve se registrar os **ambientes** da escola:
  - Local
  - Descrição
  - Responsável
- Deve se registrar os **locais** onde os sensores estarão localizados na escola:
  - Local
- Deve se registrar os **responsáveis** pelos ambientes da escola:
  - nome

- A API deve fornecer endpoints para:
    - Cadastrar, editar, listar e apagar sensores da tabela Sensores.
    - Cadastrar e listar medições por sensor
    - Listar as medições mais recentes (últimas 24h, por exemplo)

Requisitos do Projeto:

Back-End (Django Rest Framework):

Criação de uma API RESTful para gerenciar dados de sensores.

- A API deve ter endpoints para criar, ler, atualizar e deletar (CRUD) dados dos sensores e ambientes.

Os dados dos sensores devem incluir:

Temperatura (°C)

Luminosidade (lux)

Umidade (%)

Contador(num)

Os dados devem ser armazenados em um banco de dados dbsqlite3 ou MySQL.

Implementar autenticação utilizando JSON Web Tokens (JWT) para proteger os endpoints.

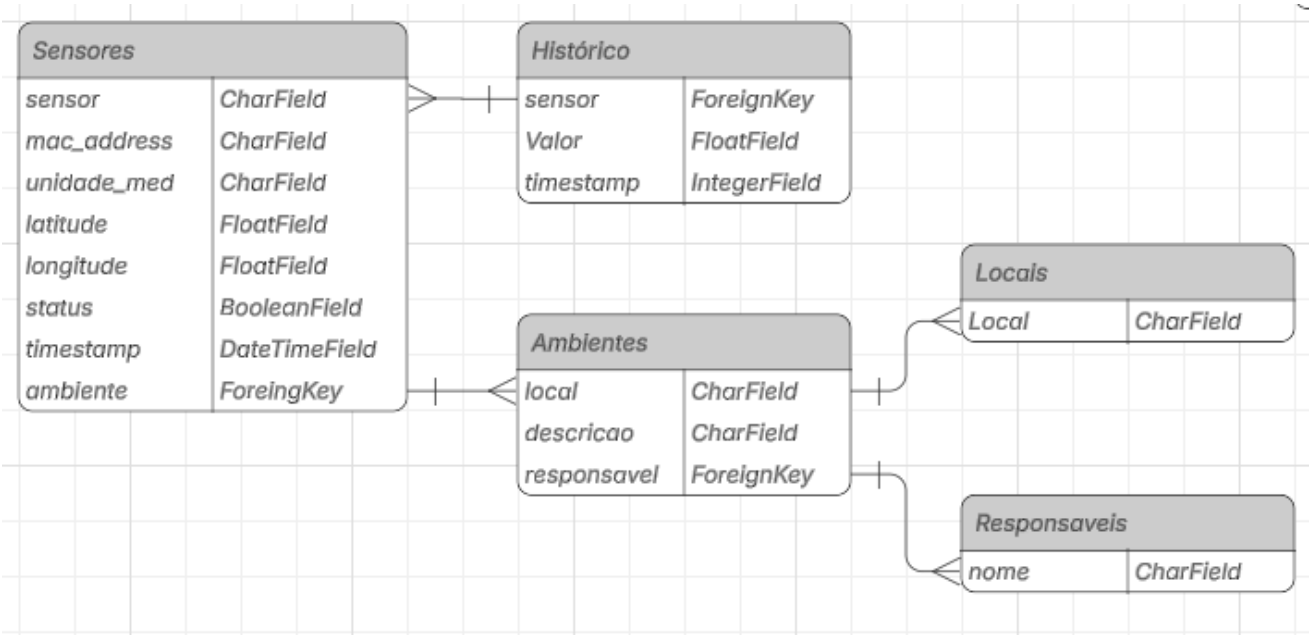

Figura 1 - Diagrama de Entidade e Relacionamento

- **Login**

  Criar um super usuário para o nosso api_smart.
    - **username = senai**
    - **password = 123**

- **Relacionamento entre tabelas**
    - Os relacionamentos deverão ser aplicados nas tabelas conforme diagrama já mencionado acima.

- **Gerenciamento dos Sensores:**
  - Em todas as páginas os elementos deverão ser listados na forma de tabela.
- **Dados:**
  - Criar método para popular banco de dados de sensores, ambientes, locais e responsáveis que estão nas planilhas disponibilizadas.
  - Para popular o banco deve se criar os seguintes arquivos com os respectivos códigos para esse fim: pop_sensores.py, pop_ambientes.py, pop_locais.py, pop_responsaveis.py.
- **Integração entre Front End e Back End:**
  - Utilizar **httpClient** no Angular para consumir a API Django.
  - Inicie com uma página de **login**.
  - Ao logar direcione para a página **home** em que teremos os 4 sensores que devem ser botões ou links diretos para página específica de cada sensor.
  - Ao acessar a página do sensor específico deve-se listar os dados em forma de tabela.

Tarefas a Serem Realizadas:

- **Desenvolvimento do Back-End:**
  - **Histórias de Usuário:**
    - Como administrador, eu quero criar um endpoint para registrar dados de sensores, para armazenar os dados de temperatura, luminosidade, umidade e contador.
    - Criar um endpoint para visualizar os dados dos sensores, para poder monitorar as condições ambientais.
    - Implementar autenticação JWT, para garantir que apenas usuários autorizados acessem os dados.
  - **Tarefas:**
    - Configurar projeto Django e instalar o Django Rest Framework..
    - Criar modelos para dados de sensores.
    - Implementar serializers e views.
    - Configurar URLs e autenticação JWT.

- **Desenvolvimento do Front-End:**
  - **Histórias de Usuário:**
    - Como supervisor, eu quero visualizar os dados dos sensores em uma lista, para monitorar as condições em tempo real.
    - Realizar login na aplicação, para acessar os dados de forma segura.
  - **Tarefas:**
    - Configurar projeto Angular.
    - Criar tela de login.
    - Conectar a aplicação à API utilizando httpClient.
    - Implementar autenticação JWT no front-end.

- **Testes e Simulação:**
  - Implementar scripts para gerar dados simulados de sensores.
  - Testar a API com ferramentas como Postman ou **Insomnia**.
  - Garantir que a aplicação móvel exiba corretamente os dados simulados.

- **Algo a mais** (não serão cobrados para nota)**:**
  - Paginação de dados no front.
  - Estilização
  - Página de cadastro de usuário
  - Página de cadastro de sensores
  - Página de entrada manual de dados para sensores
  - Página de gráficos com períodos pré-definidos
  - Visualizar gráficos dos dados dos sensores, para analisar as variações ao longo do tempo.
  - Em cada página acrescentar ícones de edição e exclusão de cada item
  - Criar opção de exportar dados no formato XLSX
  - Não aceitar medicao se sensor.status == inativo
  - Acrescentar CHOICES nos models

- **Observações:**
  - Os dados para popular serão fornecidos pelo professor.
  - Trabalho individual.
  - Na data de entrega combinada o professor irá clonar o repositório, corrigindo o trabalho do jeito como está.
  - Dúvidas somente em relação à interpretação do enunciado.
  - Entrega dia 12/12/2025.

Critérios de avaliação

Nº	Critério	Nota	Críticos	Desejáveis
1	Estrutura inicial do projeto Django + DRF (apps, settings, INSTALLED_APPS, middleware, CORS)	2	2	
2	Model **Ambiente** com local, descricao, responsavel (**tipagem correta**)	5	5	
3	Model **Sensor** com id, sensor (temperatura/umidade/luminosidade/contador), mac_address, unidade_med, latitude, longitude, status, timestamp, ambiente (**tipagem correta)**	5	5	
4	Model **Histórico** com FK para Sensor , valor, timestamp (auto/definível) **- tipagem correta**	5	5	
5	Relacionamentos	4	4	
6	Migrações aplicadas e dbsqlite3 operando (makemigrations/migrate)	1	1	
7	Serializers	2	2	
8	**URLs**: Todas as rotas da API devem ser definidas manualmente usando path() no **urls.py** principal. As rotas devem incluir os CRUDs e rotas de autenticação JWT (/api/token/ e /api/token/refresh/). A organização deve seguir uma estrutura clara: project/urls.py → inclui app/urls.py;	2	2	
9	CRUD **Ambientes** e **Locais** (testado via **Insomnia**/Postman)	8		8
10	CRUD **Sensores** e **Responsáveis** (testado via **Insomnia**/Postman)	8		8
11	Endpoints para **registrar e listar medições por sensor**	8	8	
12	**Filtros** por: sensor (Sensores), sensor (Histórico), status, data, hora, local	6		6
13	Endpoint para **medições recentes** (últimas 24h; ?hours=24) por sensor e geral	6		6
14	**Autenticação JWT (SimpleJWT)**: /api/token/ e /api/token/refresh/; proteção dos endpoints	8		8
15	**Importação de planilhas** (sensores/ambientes/locais/responsaveis) management command (CSV/Pandas)	8		8
16	**Proteção estrita das listas**: GET de **/sensores** e **/medicoes** requer JWT; sem token retorna **401**; evidências de teste (**Insomnia**/Postman)	4	4	
17	*front* - **Login (Angular)**: tela com FormBuilder + validação (required), submit via **HttpClient** para /api/token/	1	1	
18	*front* - **Armazenamento do token** (**localStorage** ou sessionStorage) e **AuthInterceptor** anexando Authorization: Bearer …	1	1	
19	*front* - **Route Guard (canActivate)** protegendo rotas de listas; nega acesso sem token	1	1	
20	*front* - **Home** pós-login com **4 botões** (Temperatura, Umidade, Luminosidade, Contador, Histórico) e navegação correta	1	1	
21	*front* - **Lista por tipo**: componente consome /api/sensores?tipo= (ou rota equivalente) e exibe resultados	1	1	
22	*front* - **Tratamento de 401** no Interceptor: redireciona para /login quando o token é inválido/expira	1	1	
23	*front* - **Logout**: limpa token e redireciona para /login	1	1	
24	*front* - **Organização visual mínima** (títulos, espaçamento, layout simples nas listas)	1	1	
25	*front* - **Configuração de ambiente** (environment.ts) com apiBaseUrl e uso centralizado no serviço	1	1	
26	Código limpo	1		1
27	Organização do código conforme visto em aula	1	1	
28	Criar e manter atualizado repositório **privado** (GitHub) onde o nome do repositório deve ser o nome_completo (sem abreviações, sem acentuação e tudo em minúsculo) com palavras separadas por "_" (ex.: maria_da_silva_souza), contendo duas pastas na raiz: back e front. O repositório deve ser privado e conceder acesso somente a lindomarbatistao@gmail.com. O professor disponibilizará um forms para recolher o endereço do github/url.	2	2	
29	Pergunta oral direta no código	5		5
		100	50	50