Projeto (21/01 → 05/02) — Mini-Sistema de Registros (API + Front)

Turma: Ensino Médio Técnico
Equipe: 5 pessoas *(pode ser 4)*
Tecnologias (sugestão):

- **Front:** React **ou** Vue (ou outro framework web equivalente)
- **UI:** TailwindCSS
- **Backend:** API (framework livre: Flask, Express, FastAPI etc.)
- **Integração:** consumo via **fetch/axios** *(InertiaJS é opcional — use se o seu stack suportar e o grupo dominar)*

Entrega: 05/02 (Projeto **70 pontos** + Artigo **30 pontos**)

---

1) Ideia do projeto

Você vai construir um **Mini-Sistema de Registros**, ou seja: um app para **cadastrar e gerenciar itens** de um tema escolhido pelo grupo.

Exemplos de temas (escolha 1)

- **Biblioteca:** livros
- **Agenda:** eventos
- **Finanças:** lançamentos (entradas/saídas)
- **Escola:** avisos/ocorrências
- **Saúde:** treinos/refeições
- **Loja:** produtos

> Regra principal: **o projeto tem 1 entidade principal (uma "tabela"/coleção)**. Nada de login e nada de relacionamentos entre tabelas.

---

2) Modelo de dados obrigatório (contrato do projeto)

Entidade única: `Item`

Campos **obrigatórios**:

- `id` *(number)* — gerado pelo backend
- `titulo` *(string)* — obrigatório (mín. 3 caracteres)
- `tipo` *(string)* — obrigatório (definido pelo grupo)
- `status` *(string)* — obrigatório (definido pelo grupo)

Campos **opcionais** *(escolha no máximo 2)*:

- `descricao` *(string)*

- `data` *(string no formato YYYY-MM-DD)*
- `valor` *(number; se usar, não pode ser negativo)*
- `tags` *(array de strings)*

Escolhas do grupo para personalizar o tema

Cada grupo deve definir:

- Quais valores de **`tipo`** vão existir no tema
  - Ex.: `["livro"]` ou `["entrada", "saida"]` ou `["aviso", "ocorrencia"]`
- Quais valores de **`status`** vão existir no tema
  - Sugestão simples: `["ativo", "concluido", "arquivado"]`

> Dica: coloquem esses valores em um arquivo de configuração (ex.: `config.js` / `settings.py`) para facilitar.

---

3) API — rotas obrigatórias

Base URL sugerida: `http://localhost:5000`

3.1 Endpoints mínimos

- `GET /items` → lista itens
  - **Extra simples (recomendado):** suportar filtro por query string:
    - `GET /items?tipo=livro`
    - `GET /items?status=ativo`
- `POST /items` → cria um item
- `PUT /items/:id` → edita um item (edição completa)
- `PATCH /items/:id/status` → altera apenas o status
- `DELETE /items/:id` → remove um item

3.2 Regras de validação (obrigatórias)

- `titulo` é obrigatório e deve ter **mínimo 3 caracteres**
- `tipo` é obrigatório e deve estar dentro da lista permitida pelo grupo
- `status` é obrigatório e deve estar dentro da lista permitida pelo grupo
- Se o grupo usar `valor`, então `valor >= 0`

3.3 Respostas padronizadas (sugestão)

- Sucesso: retornar JSON do item (ou lista) e status HTTP apropriado (200/201)
- Erros de validação: retornar status **400** com JSON contendo uma mensagem clara

Exemplo de erro:

```
{
  "error": "titulo é obrigatório e deve ter no mínimo 3 caracteres"
}
```

4) Persistência

Escolha **uma**:

- **Opção A (recomendada):** salvar em um arquivo `items.json`
- **Opção B (opcional):** SQLite (somente se o grupo estiver seguro)

> Não é permitido "sumir com os dados" a cada reinício **se o grupo escolher a opção A ou B**.
> Se optarem por "memória", deve estar **claramente explicado** no README e será considerado **menos completo**.

---

5) Frontend — requisitos mínimos (1 tela bem feita)

Você deve entregar uma **tela única** que tenha:

1. **Formulário para criar/editar**
   - `titulo`, `tipo`, `status` (+ até 2 campos opcionais)
2. **Lista (ou tabela) de itens**
   - Exibe os itens e seus campos
   - Ações: **Editar**, **Remover**, **Mudar status**
3. **Filtros**
   - Por `tipo` e por `status` (dropdowns ou botões)

5.1 Usabilidade mínima (obrigatória)

- Mostrar estado de **carregando** (loading) ao buscar dados
- Mostrar mensagem de **erro** quando a API falhar
- Mostrar feedback simples de **sucesso** (ex.: "Item cadastrado!")

---

6) Integração Front ↔ API

- O front deve consumir a API via **fetch** ou **axios**
- CORS deve estar configurado se front e back rodarem em portas diferentes
- InertiaJS é **opcional**: vale usar, mas **não é exigido**

---

7) Entregáveis obrigatórios (o que enviar em 05/02)

7.1 Projeto (repositório)

No repositório, deve existir:

- Código do **backend** e do **frontend**
- Um **README.md** com:
  - Como rodar o backend
  - Como rodar o frontend
  - Endpoints (com exemplos de request/response)
  - Regras de validação

    - Prints (ou gif) do sistema funcionando

7.2 Evidências

- Pelo menos **3 prints**:
    1. lista carregada
    2. criação/edição
    3. filtro ou mudança de status

7.3 Artigo (PDF)

Um artigo curto (mínimo de 5 páginas), no formato indicado, contendo:

- Título + Resumo + Palavras-chave
- Introdução (tema e objetivo)
- Metodologia (stack, divisão de tarefas, endpoints)
- Resultados (prints + exemplo de JSON)
- Conclusão (limitações e melhorias futuras)
- Referências (2 a 4 fontes confiáveis)

---

8) Cronograma sugerido (para não acumular)

- **21/01:** kickoff + tema + contrato do Item + repo
- **22–24/01:** API pronta (CRUD + validação + persistência)
- **25–29/01:** front pronto (tela única + integração + filtros)
- **30/01–02/02:** polimento + README + rascunho do artigo
- **03–04/02:** revisão final + artigo final
- **05/02:** entrega

---

9) Exemplos de JSON (para guiar)

Exemplo de Item (tema “Biblioteca”)

```
{
  "id": 1,
  "titulo": "Dom Casmurro",
  "tipo": "livro",
  "status": "ativo",
  "descricao": "Machado de Assis",
  "data": "2026-01-25"
}
```

Exemplo de Item (tema “Finanças”)

```
{
  "id": 7,
```

```
  "titulo": "Internet",
  "tipo": "saida",
  "status": "ativo",
  "valor": 99.90,
  "data": "2026-02-01"
}
```

---

10) Dicas finais

- Fechem o escopo no **dia 21/01** e não aumentem depois.
- Façam commits pequenos e frequentes.
- Primeiro “funcionar”, depois “ficar bonito”.
- Se travar em Inertia, use fetch/axios e entregue funcionando.