**Plano do Projeto (do zero ao pronto)**

**Fase 0 — Preparação**

* Definir nome do projeto e criar repositório Git.
* Decidir versão do Node e instalar dependências básicas que certamente usaremos: express, ejs, (opcional) method-override para PUT/DELETE, e um gerador de id simples (ou id incremental).
* Estrutura inicial de pastas (sem implementar nada ainda):
  + /src
    - /routes
    - /controllers
    - /models
    - /views (com /partials)
    - /middleware (se precisarmos)
  + /public (CSS, imagens, JS do front)
* **Critério de saída:** servidor “Hello World” sobe, EJS configurado, express.static("public") ativo.

**Fase 1 — Domínio e Dados (modelo em memória)**

* Definir o **shape** do Post (sem banco):
  + id, title, content, createdAt, updatedAt.
* Definir onde a lista de posts vive (array em memória) e como gerar id.
* Planejar operações de dados (ainda sem rotas): createPost, getAllPosts, getPostById, updatePost, deletePost.
* **Critério de saída:** módulo do “modelo” pronto (só funções e estrutura, ainda sem telas/rotas).

**Fase 2 — Rotas (mapa REST, sem lógica)**

* Mapear rotas que teremos (apenas assinatura, sem implementar):
  + GET / → listar posts
  + GET /posts/new → form de criação
  + POST /posts → criar
  + GET /posts/:id → ver um post
  + GET /posts/:id/edit → form de edição
  + PUT /posts/:id → atualizar (ou POST se optar por não usar method-override)
  + DELETE /posts/:id → excluir (ou POST para excluir)
* **Critério de saída:** arquivo(s) de rota(s) com endpoints vazios chamando controllers “stub”.

**Fase 3 — Controllers (fluxo, sem render bonito)**

* Implementar controllers chamando o modelo:
  + Lista, criação, leitura, edição, atualização, exclusão.
* Decidir estratégia de erros (ex.: se id não existe → 404).
* **Critério de saída:** controllers retornam dados “crus” ou redirecionam, mas ainda sem páginas finalizadas.

**Fase 4 — Views base (EJS)**

* Criar **layout básico** com partials:
  + partials/header.ejs (link do CSS, título, nav mínima)
  + partials/footer.ejs (ano dinâmico)
* Criar páginas **mínimas**:
  + index.ejs (lista de posts)
  + new.ejs (form criar)
  + show.ejs (ver post)
  + edit.ejs (form editar)
* **Critério de saída:** ciclo completo manual funciona (criar → listar → ver → editar → excluir), mesmo que “feio”.

**Fase 5 — UX & Validação simples**

* Validações mínimas no servidor (ex.: título obrigatório, tamanho mínimo).
* Mensagens de erro/sucesso simples (pode ser com variáveis passadas à view).
* Prevenir perda de dados em caso de erro (re-render com valores do formulário).
* **Critério de saída:** usuário comum consegue usar sem “quebrar” nada com dados vazios.

**Fase 6 — Estilo (CSS) e responsividade**

* Criar public/css/styles.css e aplicar estilos básicos:
  + Tipografia, espaçamentos, container, botões, formulários, cards de post.
* Responsividade com grid/flex e ajustes para mobile (sem exagero).
* **Critério de saída:** layout limpo e usável em desktop e mobile.

**Fase 7 — Qualidade de Vida**

* Páginas de **404** e **500** simples.
* Pequenos aprimoramentos de acessibilidade (labels, contraste, foco).
* Navegação clara (links entre páginas).
* **Critério de saída:** app está “redondinho” para demonstração.

**Fase 8 — Refinos opcionais (se quiser depois)**

* **Method override** para ter rotas REST sem hack em formulários.
* **Flash messages** (ex.: criação/edição/exclusão OK).
* **Semeadura** de posts de exemplo quando app inicia (modo demo).
* **Separar CSS por páginas** se crescer.
* Escrever um **README** com instruções de uso.

**Ordem de execução recomendada (resumo)**

1. Preparar projeto e pastas
2. Definir modelo em memória (shape + funções CRUD)
3. Esboçar rotas (assinaturas)
4. Implementar controllers chamando o modelo
5. Montar views mínimas (index/new/show/edit)
6. Validação leve e mensagens
7. CSS + responsividade
8. Erros (404/500) e refinamentos