**1) Escolha & entendimento da API**

* **API de previsão:** Open-Meteo /v1/forecast (sem autenticação). Variáveis diárias úteis:
  + precipitation\_probability\_mean (probabilidade média de chuva por dia),
  + precipitation\_sum/rain\_sum (acumulado diário).  
    Requer latitude, longitude e timezone. [open-meteo.com](https://open-meteo.com/en/docs)
* **Geocoding:** Open-Meteo /v1/search?name=... para converter cidade → coordenadas (sem autenticação). [open-meteo.com](https://open-meteo.com/en/docs/geocoding-api?utm_source=chatgpt.com)
* **Por que ela?** Sem chave, ampla cobertura global e documentação clara. (No-key explicitado nos docs.) [open-meteo.com](https://open-meteo.com/en/about?utm_source=chatgpt.com)
* **CORS:** vamos consumir a API **no servidor (Express)** via Axios; assim, evitamos qualquer bloqueio de CORS no navegador.

**2) Escopo funcional (MVP)**

* Usuário informa uma **cidade**.
* Sistema resolve **latitude/longitude** via Geocoding.
* Sistema busca **previsão diária** e decide: **“vai chover amanhã?”** (boolean + explicação).
* Exibe no EJS: mensagem clara (ex.: “Alta chance de chuva amanhã em Goiânia”), probabilidade (%) e, opcionalmente, acumulado (mm).

**3) Regras de negócio (primeira versão)**

* “Amanhã” = **próximo dia local da cidade** (usar timezone=auto na chamada da previsão). [open-meteo.com](https://open-meteo.com/en/docs)
* Critério de chuva (podemos ajustar depois):
  + Chove se precipitation\_probability\_mean >= 50% **ou** precipitation\_sum >= 0.5 mm.
* Em caso de múltiplas correspondências no geocoding, pegar **primeiro resultado**; depois refinamos (país/UF).

**4) Arquitetura & rotas (sem código, só desenho)**

* **GET /** → formulário (cidade).
* **POST /consulta** →
  1. chama Geocoding (cidade → lat/lon),
  2. chama Forecast (diário para 2 dias),
  3. calcula decisão “vai chover amanhã?”,
  4. renderiza index.ejs com { cidade, amanhaISO, prob%, mm, vaiChover }.
* **GET /health** (simples) para ver se o servidor está de pé.
* **/public** para CSS/JS estático.

**5) Organização de pastas**

/project

/public (css, imagens)

/views (EJS)

/src

app.js (Express)

services/

meteo.js (funções Axios p/ geocoding & forecast)

lib/

decide.js (regra “vai chover amanhã?”)

README.md

**6) Apresentação (UI mínima)**

* Topo: título + formulário com input de cidade.
* Resultado: card com **mensagem** (verde/vermelho), **probabilidade (%)**, **acumulado (mm)**, **data** (“Amanhã, 03/09/2025”) e **cidade normalizada**.
* Erros: caixa discreta com texto amigável (ex.: “Cidade não encontrada” ou “Falha na previsão”).

**7) Tratamento de erros**

* **Geocoding**: zero resultados → feedback “Cidade não encontrada”.
* **Forecast**: resposta sem daily ou sem o índice de amanhã → feedback “Dados indisponíveis agora”.
* **Rede/timeout**: log no server e mensagem genérica ao usuário; status HTTP coerente.

**8) Critérios de aceite (para cada etapa)**

1. **Geocoding OK**: cidade “Goiânia” retorna lat/lon e timezone presentes. [open-meteo.com](https://open-meteo.com/en/docs/geocoding-api?utm_source=chatgpt.com)
2. **Forecast OK**: chamada com daily=precipitation\_probability\_mean,precipitation\_sum&forecast\_days=2&timezone=auto retorna arrays com 2 dias. [open-meteo.com](https://open-meteo.com/en/docs)
3. **Decisão**: regra aplicada ao índice “dia+1” no fuso local.
4. **UI**: mensagem e números aparecem, sem quebrar layout.
5. **Erros**: mensagens amigáveis evidentes no EJS.

**9) Documentação & GitHub**

* README.md: como rodar (ex.: npm i, npm run dev), variáveis de ambiente (se usarmos), endpoints e exemplos de uso.
* Comentários em pontos-chave (serviços/decisão).
* Commits pequenos e descritivos.