

# Mapify Turismo — Sistema de Mapa Turístico Gamificado

Documentação Técnica Completa

## 1. Visão geral do projeto

O Mapify Turismo é uma aplicação web que combina mapa interativo, gamificação e gestão de pontos turísticos. O usuário visualiza pontos de interesse em um mapa, cria novos locais, avalia lugares, marca pontos como 'cartão postal' (visitados) e acompanha sua evolução em um perfil com sistema de XP e níveis. O sistema é dividido em duas camadas principais: um frontend em React, responsável pela interface e experiência do usuário, e um backend em Node.js com Express, que gerencia autenticação, usuários e CRUD de lugares em memória para fins de protótipo.

## 2. Tecnologias utilizadas

### 2.1 Frontend

- React + Vite: base do frontend SPA (Single Page Application). O Vite cuida do build e desenvolvimento rápido, enquanto o React oferece componentes e controle de estado.
- React Leaflet + Leaflet: biblioteca de mapas utilizada para renderizar o mapa, marcadores, círculos de raio e rotas. O Leaflet é a engine de mapas; o React Leaflet fornece componentes React para integração fluida.
- CSS puro (styles.css): responsável pelo layout geral (topbar, sidebar, modais, cartões de perfil, páginas de salvos/avaliados/visitados), com um visual clean e focado em usabilidade.
- LocalStorage: usado no navegador para persistir autenticação (usuário + token), avaliações de lugares e lista de pontos visitados, tudo separado por usuário.

### 2.2 Backend.

- Node.js + Express: implementam a API REST do projeto, incluindo autenticação, rotas de usuários e CRUD de lugares.
- JSON Web Token (JWT): usado para autenticação stateless. O backend emite um token assinado no login/registro e o frontend envia esse token no cabeçalho Authorization.
- bcryptjs: biblioteca para hash e verificação de senha, garantindo que senhas não sejam armazenadas em texto plano.
- Nodemailer: responsável pelo envio de e-mails de recuperação de senha, usando credenciais e servidor SMTP configurados via variáveis de ambiente.
- dotenv: carrega variáveis sensíveis do arquivo .env (JWT\_SECRET, SMTP\_HOST, SMTP\_USER, SMTP\_PASS, DATABASE\_URL etc.).

### 3. Arquitetura do Frontend

A camada de frontend é organizada em torno do componente App.jsx, que funciona como orquestrador global. Ele mantém o estado principal da aplicação e decide qual 'tela' renderizar (mapa, perfil, locais salvos, avaliados ou visitados).

#### 3.1 Estrutura de pastas (frontend/src)

- assets/: recursos estáticos.
- components/: componentes reutilizáveis da interface. – AuthPage.jsx: tela de login/registro e recuperação de senha. – MapView.jsx: encapsula o MapContainer do Leaflet e renderiza marcadores, raio e rota. – PlaceDetailsModal.jsx: modal de detalhes de um lugar, avaliações e cartão postal. – PlaceCreateModal.jsx: modal para criação de novos pontos turísticos. – ProfilePage.jsx: tela de perfil com estatísticas de progressão (XP, nível, contadores). – SavedPlacesPage.jsx: lista de lugares criados pelo usuário logado. – ReviewedPlacesPage.jsx: lista de lugares que o usuário avaliou. – VisitedPlacesPage.jsx: lista de lugares marcados como visitados (cartões postais).
- data/basePlacesNatal.js: conjunto inicial de pontos turísticos pré-cadastrados.
- services/api.js: comunicação com o backend (auth e /places).
- services/places.js: integração com a API pública OpenTripMap para buscar pontos próximos.
- services/routes.js: integração com serviço externo de rotas (OpenRouteService ou similar).
- styles.css: estilos principais da aplicação.

### 4. Estado global e fluxo de dados (App.jsx)

O App.jsx é responsável por:

- Gerenciar a autenticação (currentUser, token) e sincronizá-la com o LocalStorage.
- Armazenar posição do usuário (geolocalização) e raio de busca.
- Controlar listas de lugares: apiPlaces (OpenTripMap), backendPlaces (API própria) e basePlacesNatal (locais fixos), combinadas em allPlaces.
- Manter reviewsByPlace (avaliações por lugar) e visitedPlaces (IDs de lugares visitados) por usuário.
- Gerenciar informações de rota: routeTarget, routeGeometry e routeInfo.
- Calcular estatísticas de perfil e XP (profileStats).

#### 4.1 LocalStorage e chaves por usuário

Para isolar os dados de cada conta, o sistema utiliza uma 'userKey', derivada do e-mail do usuário (ou id/nome como fallback). Essa chave é usada nos prefixos:

- mapify\_auth\_v1 → dados de autenticação (user + token).
- place\_reviews\_v1\_ → avaliações feitas pelo usuário.
- visited\_places\_v1\_ → IDs de lugares marcados como cartão postal. ■ Dessa forma, ao trocar de conta, cada usuário enxerga apenas suas próprias avaliações e locais visitados.

## 4.2 Sistema de XP e gamificação

O objeto `profileStats` é recalculado via `useMemo` e inclui:

- `savedPlaces`: total de lugares criados pelo usuário.
- `reviewedPlacesCount`: quantidade de lugares distintos avaliados.
- `totalReviews`: total de avaliações (caso se queira múltiplas por local).
- `visitedCount`: quantidade de locais marcados como visitados.
- `xp`: pontuação calculada combinando ações (criar, avaliar, visitar).
- `level`: nível do usuário ( $xp / 200 + 1$ ).
- `currentLevelXp` e `xpToNext`: usados para desenhar a barra de progresso.

## 5. Funcionalidades principais do Frontend

### 5.1 Mapa interativo (MapView.jsx)

- Mostra o mapa centrado na posição do usuário (geolocalização) ou em uma coordenada padrão.
- Desenha um círculo representando o raio de busca selecionado.
- Para cada lugar em `places`, adiciona um `Marker` com `Popup` exibindo nome, endereço/tipo e distância.
- Ao clicar num marcador, o `App.jsx` abre o `PlaceDetailsModal`.
- Ao clicar com o botão direito (`contextmenu`), dispara `onAddPlace` para abrir o modal de criação de novo ponto turístico.

### 5.2 Detalhes do lugar e avaliações (PlaceDetailsModal.jsx)

- Exibe nome, categoria, distância estimada e endereço do local.
- Calcula distância e tempo estimado a pé e de carro, usando `routeInfo` ou a propriedade `dist` do lugar.
- Mostra média das avaliações (estrelinhas) e lista de comentários.
- Permite que o usuário autenticado crie, edite ou remova sua própria avaliação (apenas uma por lugar). O autor é derivado do nome da conta logada.
- Integração com rotas: botão 'Traçar rota' e 'Limpar rota', que se conectam ao serviço de rotas do `App.jsx`.

### 5.3 Cartão postal / locais visitados

No modal de detalhes, o usuário pode marcar o local como 'cartão postal', indicando que já visitou aquele ponto. Essa ação aciona `onToggleVisited(place.id)`, que atualiza `visitedPlaces` no `App.jsx`.

Esses IDs são persistidos no `LocalStorage` e utilizados para:

- Contabilizar 'Pontos turísticos visitados' no perfil.
- Renderizar cards na `VisitedPlacesPage.jsx` com um PIN colorido indicando a categoria (ponto turístico, restaurante ou comércio).

## 6. Telas de perfil e listagens

### 6.1 ProfilePage.jsx

A tela de perfil apresenta:

- Nome e e-mail do usuário, com avatar baseado nas iniciais.
- Nível atual e XP, com barra de progresso até o próximo nível.
- Três cartões de estatísticas clicáveis: – Locais criados: leva para SavedPlacesPage. – Locais avaliados: leva para ReviewedPlacesPage. – Pontos turísticos visitados: leva para VisitedPlacesPage.

### 6.2 SavedPlacesPage.jsx

Lista somente os lugares criados pelo usuário logado (`created_by === currentUser.id`). Os dados vêm do backend via `/places`. A tela segue o mesmo padrão visual de cartões, com título, endereço/categoria e botão para voltar ao mapa.

### 6.3 ReviewedPlacesPage.jsx

Recebe de `App.jsx` a lista `profileStats.reviewedPlaces` (`placeId + review`). Para cada item, faz o join com o array `places` para recuperar os dados completos do lugar. Exibe nome, nota, comentário e data da avaliação, em um layout de cartões responsivos. 6.4 `VisitedPlacesPage.jsx` Similar à tela de avaliados, mas com foco em locais visitados. Recebe a lista completa de `places` e o array `visitedPlaces` (IDs). A partir disso, filtra e monta `visitedList`.

Cada card exibe:

- Nome e endereço do lugar.
- Um 'PIN' colorido indicando categoria: azul/roxo para pontos turísticos, verde para restaurantes e vermelho para comércio.
- Tag textual ('Ponto turístico', 'Restaurante' ou 'Comércio') e, se disponível, distância estimada.

## 7. Backend — Arquitetura e rotas

A API backend foi implementada em Node.js com Express, com foco em fornecer autenticação, gestão simples de usuários e CRUD de lugares em memória. Para fins de protótipo, não há persistência real dos dados em banco (exceto pelas estruturas preparadas com Prisma, para uso futuro).

### 7.1 Autenticação e usuários (index.js)

O backend mantém arrays em memória: ■ • `users`: lista de usuários cadastrados.

- `places`: lista de lugares cadastrados via API.
- `resetTokens`: tokens temporários de recuperação de senha.

Os endpoints principais de autenticação são:

- `POST /auth/register`: cria um novo usuário (`name`, `email`, `password`). A senha é hashada com `bcrypt`.
- `POST /auth/login`: valida credenciais e retorna token JWT e dados do usuário.
- `POST /auth/forgot-password`: gera token de reset, armazena em `resetTokens` e dispara e-mail com link de recuperação usando `Nodemailer`.

- POST /auth/reset-password: recebe token + nova senha, valida o token, atualiza o hash e invalida o token.

O middleware authMiddleware verifica o cabeçalho Authorization ('Bearer '), decodifica o JWT e injeta as informações do usuário em req.user. Rotas protegidas, como /places, exigem esse middleware.

## 7.2 Rotas de Places (index.js)

As rotas de lugares operam sobre o array em memória places:

- GET /places (autenticado): retorna todos os lugares cadastrados.
- POST /places (autenticado): cria um novo lugar com os campos name, type, address, lat, lon, description, imageUrl. Também grava created\_by (id do usuário) e created\_at.
- PUT /places/:id (autenticado): atualiza campos de um lugar existente.
- DELETE /places/:id (autenticado): remove um lugar existente.

## 7.3 Integração futura com banco de dados (schema.prisma, db.js)

O projeto já inclui arquivos de preparação para uso de PostgreSQL via Prisma:

- schema.prisma: define modelos Place e Review, com relacionamento (um lugar possui várias avaliações).
- db.js: exemplo de conexão com PostgreSQL usando pg Pool, preparado para ambientes como Supabase.

Atualmente, essas estruturas não são usadas em produção, mas servem como base para evoluir o backend de um protótipo em memória para uma API persistente

## 8. Integração Frontend ↔ Backend

O módulo frontend services/api.js centraliza chamadas HTTP para o backend.

Exemplos típicos:

- register(name, email, password) → POST /auth/register.
- login(email, password) → POST /auth/login.
- forgotPassword(email) → POST /auth/forgot-password.
- resetPassword(token, password) → POST /auth/reset-password.
- getPlaces(token) → GET /places com cabeçalho Authorization.
- createPlace(data, token) → POST /places. Após login/registro bem-sucedido, o App.jsx salva user + token no LocalStorage e repassa o token para essas funções de serviço.

## 9. Configuração de ambiente e execução

Para executar o projeto localmente:

1. Backend:

- Criar arquivo backend/.env com variáveis: – JWT\_SECRET=uma\_chave\_segura – SMTP\_HOST, SMTP\_PORT, SMTP\_USER, SMTP\_PASS (para recuperação de senha).
- Instalar dependências: npm install • Rodar servidor: npm run dev ou node src/index.js

2. Frontend:

- Instalar dependências: npm install
- Rodar frontend: npm run dev (Vite, porta padrão 5173).

## **10. Conclusão**

O Mapify Turismo entrega um MVP funcional de plataforma de turismo gamificada, combinando mapa interativo, perfil de usuário, sistema de avaliações e coleção de 'cartões postais'. A arquitetura foi pensada para ser simples de entender e fácil de evoluir, tanto no frontend quanto no backend, permitindo que futuras versões adotem banco de dados real, novas mecânicas de jogo e integrações com plataformas de turismo e cidades inteligentes.

