

# AI Traveller 提交材料

- 仓库地址: <https://github.com//ai-traveller>
- 生成时间: 2025-10-27T11:43:59.065Z

## AI Traveller

AI Traveller 是一套基于 JavaScript 的全栈智能旅行规划平台，通过大语言模型、语音识别与地图服务帮助用户快速生成并管理个性化的出行方案。系统支持账号密码登录（用户名唯一），注册时会发送邮箱验证码/确认链接，验证成功后即可使用用户名或邮箱 + 密码登录。

### 🌟 核心功能

- 智能行程规划**: 输入旅行目的地、预算、同行人数等信息，AI 自动生成包含交通、住宿、景点、美食的多日行程。
- 预算估算与费用管理**: AI 估算预算结构，支持语音或文字记录每天开销并同步汇总。
- 账号密码登录体系**: 注册时填写唯一用户名、邮箱和密码，系统向邮箱发送验证码/确认邮件；登录支持“用户名或邮箱 + 密码”，并保留教学模式下的临时 Token 演示。
- 语音识别支持**: 内置 Web Speech API 输入，也可上传语音文件由后端代理科大讯飞识别。
- 地图可视化**: 集成高德地图 (Amap) 展示行程路线与 POI，并在缺少坐标时自动解析。
- 一体化部署**: 前端 (Vite + React)、后端 (Express) 与 Docker 打包，单命令启动或部署。

### 📦 技术栈

- 前端**: Vite, React 19, Tailwind CSS, React Router, SWR, Zustand
- 后端**: Node.js 20, Express, Supabase JS, OpenAI 兼容 SDK, Axios, Multer
- 共享模块**: @ai-traveller/common (费用类别、常量、类型)
- 第三方服务**: Supabase (认证 & 数据库)、科大讯飞语音识别、大语言模型 API、高德地图开放平台
- 工程工具**: npm workspaces, ESLint, Vitest, Docker, GitHub Actions (预留)

### 📁 目录结构

```
.
├── backend/           # Express API 服务
├── frontend/         # React Web 前端
├── packages/common/  # 前后端共享常量
├── docs/             # 架构及提交文档
├── scripts/          # 构建辅助脚本
├── Dockerfile
└── docker-compose.yml
```

### 🚀 快速开始

#### 1. 克隆与安装依赖

```
git clone <your-repo-url> ai-traveller
cd ai-traveller
npm install
```

2. 配置环境变量

复制示例环境文件，按照需求填写真实密钥（切勿提交到仓库）：

```
cp backend/.env.example backend/.env
cp frontend/.env.example frontend/.env
```

关键变量说明：

变量	说明
SUPABASE_URL / SUPABASE_ANON_KEY / SUPABASE_SERVICE_ROLE_KEY	Supabase 项目地址与密钥，后端使用 service role，前端只使用 anon key
LLM_API_URL / LLM_API_KEY / LLM_MODEL	大语言模型服务地址、密钥与模型名称（支持 OpenAI 兼容接口）
IFLYTEK_APP_ID / IFLYTEK_API_KEY / IFLYTEK_API_SECRET	科大讯飞实时语音转写密钥
AMAP_WEB_SERVICE_KEY	高德开放平台「Web 服务」Key，用于服务端 POI/逆地理接口
VITE_AMAP_JS_KEY（前端）	高德开放平台「Web 端 (JS API)」Key
VITE_AMAP_JS_SECURITY_CODE（前端）	高德 Web JS 安全密钥 securityJsCode，需与 JS Key 一起使用

 如果暂未配置 Supabase，可使用登录页底部的“教学模式”临时 Token 登录进行演示。

3. 初始化 Supabase 表结构

在 Supabase 控制台 → SQL Editor 依次执行以下脚本，创建所需表及 RLS 策略：

```
-- 用户资料表（用户名唯一）
create table if not exists public.profiles (
  id uuid primary key references auth.users (id) on delete cascade,
  username text not null unique,
  email text not null,
  created_at timestampz default now()
);

alter table public.profiles enable row level security;
create policy "Users manage own profile"
```

```
on public.profiles
using (auth.uid() = id)
with check (auth.uid() = id);
```

-- 行程表

```
create table if not exists public.trips (
  id uuid primary key,
  user_id uuid not null references auth.users (id) on delete cascade,
  destination text not null,
  start_date date,
  end_date date,
  budget numeric,
  currency text,
  travelers int,
  preferences jsonb,
  notes text,
  ai_summary jsonb,
  ai_budget jsonb,
  created_at timestamptz default now(),
  updated_at timestamptz default now()
);
```

```
alter table public.trips enable row level security;
create policy "Trips belong to user"
on public.trips
using (auth.uid() = user_id)
with check (auth.uid() = user_id);
```

-- 行程详情与预算

```
create table if not exists public.itineraries (
  trip_id uuid primary key references public.trips (id) on delete cascade,
  data jsonb not null,
  updated_at timestamptz default now()
);
```

```
alter table public.itineraries enable row level security;
create policy "Itinerary belongs to trip owner"
on public.itineraries
using (exists (select 1 from public.trips t where t.id = trip_id and t.user_id =
auth.uid()))
with check (exists (select 1 from public.trips t where t.id = trip_id and t.user_id =
auth.uid()));
```

```
create table if not exists public.budgets (
  trip_id uuid primary key references public.trips (id) on delete cascade,
  data jsonb not null,
```

```

    updated_at timestamptz default now()
);

alter table public.budgets enable row level security;
create policy "Budget belongs to trip owner"
  on public.budgets
  using (exists (select 1 from public.trips t where t.id = trip_id and t.user_id =
auth.uid()))
  with check (exists (select 1 from public.trips t where t.id = trip_id and t.user_id =
auth.uid()));

-- 费用记录
create table if not exists public.expenses (
  id uuid primary key,
  trip_id uuid not null references public.trips (id) on delete cascade,
  user_id uuid not null references auth.users (id) on delete cascade,
  title text,
  category text,
  amount numeric,
  currency text,
  spent_at timestamptz,
  notes text,
  voice_note_url text,
  transcript text,
  created_at timestamptz default now(),
  updated_at timestamptz default now()
);

alter table public.expenses enable row level security;
create policy "Expenses belong to user"
  on public.expenses
  using (auth.uid() = user_id)
  with check (auth.uid() = user_id);

```

确保在 **Authentication** → **Configuration** 中打开 “Email confirmations”，并把 `http://localhost:5173`（以及部署地址）填入 Site URL 和 Additional Redirect URLs。

## 4. 启动开发环境

```
npm run dev
```

该命令将并行启动：

- `http://localhost:8080` : Express API (含健康检查 `/health`)
- `http://localhost:5173` : React 前端开发服务

## 5. 质量检查与构建

```
npm run lint
npm run test:backend
npm --workspace frontend run build # 验证前端构建
```

### 模块说明

- **认证与用户管理**: `backend/src/services/authService.js` 负责注册、登录与用户名解析。使用 `service role` 创建 Supabase 用户并写入 `profiles` , 注册后发送邮箱验证码; 登录支持用户名或邮箱。前端 `LoginGate` 提供注册 + 登录表单, 并在未配置 Supabase 时回退到临时 Token 或内存账号模式。
- **\*\*AI 规划 ( `backend/src/services/aiService.js` )\*\***: 封装 LLM Prompt 和 JSON 解析, 在未配置 LLM Key 时提供静态示例。
- **\*\*语音识别 ( `backend/src/services/speechService.js` )\*\***: 实现科大讯飞 REST API 签名流程, 缺省时给出提示; 前端提供 Web Speech + 音频上传两种方式。
- **\*\*地图服务 ( `backend/src/services/mapService.js` )\*\***: 调用高德 Web 服务获取 POI 与逆地理结果, 并在缺少 Key 时返回示例坐标; 前端地图组件会自动补全行程项坐标。
- **\*\*费用管理 ( `backend/src/services/expenseService.js` )\*\***: Supabase 表结构访问 + 内存回退, 包含分类校验和金额处理。
- **\*\*前端状态 ( `frontend/src/store/useSessionStore.js` )\*\***: 统一 Supabase Session、教学模式 Token, 与 Axios 鉴权拦截器配合使用。

### Docker 打包与运行

1. 准备 `backend/.env.docker` 并填入生产环境密钥。
2. 构建镜像:

```
docker build -t ai-traveller:latest .
```

3. 运行:

```
docker run --rm -p 8080:8080 --env-file backend/.env.docker ai-traveller:latest
```

或使用 Compose:

```
docker compose up --build
```

容器会同时提供 `/api` 接口与前端静态页面。

### 文档与提交


- `docs/ARCHITECTURE.md` : 系统架构、模块、配置说明。
- `docs/submission.pdf` : 执行 `REPO_URL=<仓库地址> npm run generate:pdf` 自动生成 (基于 README) 。

## API 密钥与安全建议

- 所有密钥仅配置在 `.env`，禁止提交到仓库。
- 前端需要公开的 Key 使用 `VITE_` 前缀，通过 Vite 注入。
- Supabase service role 只在后端使用，可通过部署平台的环境变量管理。
- 部署前建议为账号密码登录启用强密码策略与邮箱域名白名单。

## 下一步计划

- 接入 Supabase Realtime，实现多人协作与实时通知。
- 扩展 CI/CD (GitHub Actions) 自动化测试、Docker 构建与推送。
- 增强移动端适配与 PWA 缓存能力。

 如需调试或替换新的 API Key，请在 `.env` 中更新并重启服务；若仅用于教学/演示，可继续使用登录页提供的临时 Token 登录模式。