SipSpot 咖啡店评论平台 - 项目规划文档

> 基于 Colt Steele YelpCamp 项目改造的现代化前后端分离应用

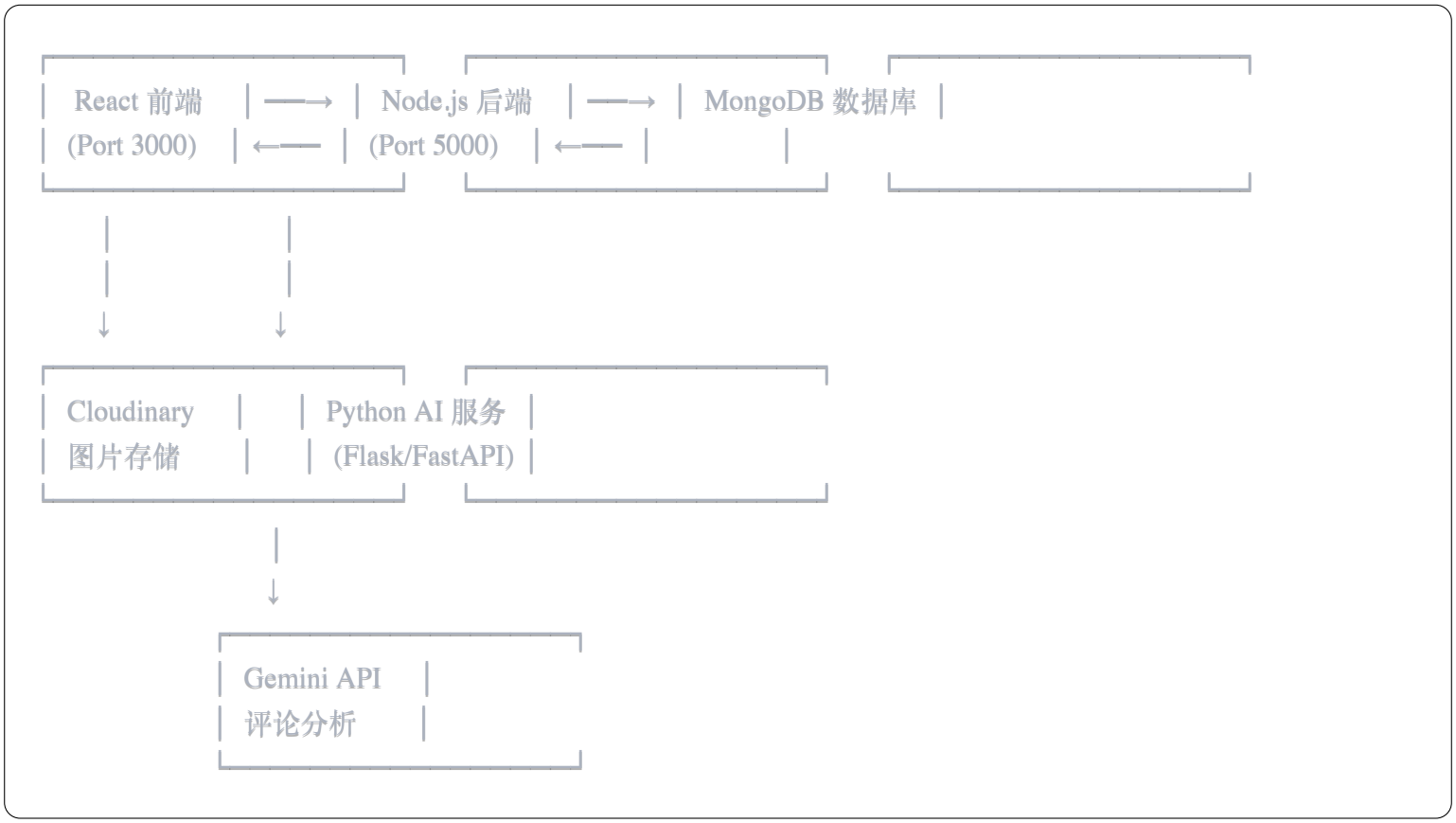
📋 项目概述

SipSpot 是一个全栈咖啡店评论平台，用户可以：

- 📍 基于地理位置搜索附近咖啡店
- ☕ 添加、浏览、评论咖啡店
- 🤖 使用AI智能分析评论情感
- 📊 查看综合评分与用户反馈摘要
- 🗺️ 在地图上可视化咖啡店位置

🏗️ 项目架构

整体架构图



🔄 从 YelpCamp 到 SipSpot 的核心迁移

YelpCamp 原有功能映射

YelpCamp 功能	SipSpot 对应功能	改进点
营地列表	咖啡店列表	添加地理位置筛选
营地详情	咖啡店详情	整合AI评论分析
用户评论	咖啡店评论	AI情感分析 + 关键词提取
图片上传	咖啡店照片	多图上传 + 图片压缩
用户认证	JWT认证	前后端分离认证
MapBox地图	OpenStreetMap	Leaflet.js集成

💻 技术栈详细说明

1. 前端技术栈

React.js 项目结构

```
frontend/
├── src/
│   ├── components/
│   │   ├── Navbar.jsx          # 导航栏
│   │   ├── CafeCard.jsx        # 咖啡店卡片
│   │   ├── CafeList.jsx        # 咖啡店列表
│   │   ├── CafeDetail.jsx      # 咖啡店详情
│   │   ├── ReviewForm.jsx      # 评论表单
│   │   ├── ReviewList.jsx      # 评论列表
│   │   ├── Map.jsx             # 地图组件
│   │   └── AIAnalysis.jsx      # AI分析展示
│   ├── pages/
│   │   ├── Home.jsx            # 首页
│   │   ├── Login.jsx           # 登录
│   │   ├── Register.jsx        # 注册
│   │   ├── CafeDetailPage.jsx  # 咖啡店详情页
│   │   └── Profile.jsx         # 用户个人页
│   ├── hooks/
│   │   ├── useAuth.js          # 认证Hook
│   │   ├── useGeolocation.js   # 地理位置Hook
│   │   └── useAPI.js           # API调用Hook
│   ├── contexts/
│   │   └── AuthContext.jsx     # 认证上下文
│   ├── services/
│   │   ├── api.js              # Axios配置
│   │   └── cafesAPI.js         # 咖啡店API
│   └── App.jsx
```

关键前端技术选型

- **Tailwind CSS**: 快速开发响应式UI
- **Ant Design**: 复杂表单和组件（评分、上传等）
- **Leaflet.js**: 轻量级地图库，与OpenStreetMap集成
- **React Router v6**: 路由管理
- **React Query**: 数据缓存与状态管理

2. 后端技术栈

Node.js + Express 项目结构

```
backend/
├── src/
│   ├── models/
│   │   ├── User.js              # 用户模型
│   │   ├── Cafe.js              # 咖啡店模型
│   │   └── Review.js            # 评论模型
│   ├── routes/
│   │   ├── auth.js              # 认证路由
│   │   ├── cafes.js             # 咖啡店路由
│   │   ├── reviews.js           # 评论路由
│   │   └── users.js             # 用户路由
│   ├── controllers/
│   │   ├── authController.js
│   │   ├── cafeController.js
│   │   └── reviewController.js
│   ├── middleware/
│   │   ├── auth.js              # JWT验证中间件
│   │   ├── errorHandler.js      # 错误处理
│   │   └── upload.js            # Multer图片上传
│   ├── services/
│   │   ├── cloudinary.js        # Cloudinary配置
│   │   └── aiService.js         # AI服务调用
│   ├── utils/
│   │   ├── validation.js        # 数据验证
│   │   └── geocoding.js         # 地理编码
│   └── server.js
```

RESTful API 设计

javascript

```
// 认证路由
POST   /api/auth/register      # 用户注册
POST   /api/auth/login         # 用户登录
GET    /api/auth/me            # 获取当前用户

// 咖啡店路由
GET    /api/cafes              # 获取咖啡店列表（支持地理位置查询）
GET    /api/cafes/:id          # 获取咖啡店详情
POST   /api/cafes              # 创建咖啡店（需认证）
PUT    /api/cafes/:id          # 更新咖啡店（需认证+权限）
DELETE /api/cafes/:id          # 删除咖啡店（需认证+权限）
GET    /api/cafes/nearby       # 查找附近咖啡店

// 评论路由
GET    /api/cafes/:id/reviews  # 获取咖啡店评论
POST   /api/cafes/:id/reviews  # 创建评论（需认证）
PUT    /api/reviews/:id        # 更新评论（需认证+权限）
DELETE /api/reviews/:id        # 删除评论（需认证+权限）
POST   /api/reviews/:id/analyze # 触发AI分析（需认证）
```

3. 数据库设计

MongoDB Schema 设计

javascript

```
// User Schema
{
  username: String,
  email: String (unique),
  password: String (hashed),
  role: String (enum: ['user', 'admin']),
  avatar: String,
  createdAt: Date,
  favorites: [ObjectId] // 收藏的咖啡店
}

// Cafe Schema
{
  name: String,
  description: String,
  location: {
    type: "Point", // GeoJSON格式
    coordinates: [Number] // [经度, 纬度]
```

```
  },
  address: String,
  city: String,
  images: [String], // Cloudinary URLs
  price: Number (1-4), // 价格区间
  amenities: [String], // 设施标签：WiFi、插座、安静等
  rating: Number,
  reviewCount: Number,
  owner: ObjectId (ref: 'User'),
  createdAt: Date
}

// Review Schema
{
  cafe: ObjectId (ref: 'Cafe'),
  author: ObjectId (ref: 'User'),
  rating: Number (1-5),
  content: String,
  images: [String],
  aiAnalysis: {
    sentiment: String (enum: ['positive', 'negative', 'neutral']),
    keywords: [String],
    summary: String
  },
  createdAt: Date
}
```

地理位置索引

```
javascript

// 在Cafe模型中创建2dsphere索引
cafeSchema.index({ location: '2dsphere' });

// 查询附近咖啡店
db.cafes.find({
  location: {
    $near: {
      $geometry: { type: "Point", coordinates: [经度, 纬度] },
      $maxDistance: 5000 // 5公里内
    }
  }
});
```

4. AI 智能分析模块

4.4.2 智能分析模块

Python 微服务架构 (FastAPI)

```python
```

```python
# ai-service/main.py
from fastapi import FastAPI
from pydantic import BaseModel
import google.generativeai as genai

app = FastAPI()
genai.configure(api_key="YOUR_GEMINI_API_KEY")

class ReviewRequest(BaseModel):
    content: str
    cafe_name: str

class AnalysisResponse(BaseModel):
    sentiment: str
    keywords: list[str]
    summary: str

@app.post("/analyze", response_model=AnalysisResponse)
async def analyze_review(review: ReviewRequest):
    model = genai.GenerativeModel('gemini-pro')

    prompt = f"""
    分析以下咖啡店评论，并提取：
    1. 情感倾向（positive/negative/neutral）
    2. 关键特征（最多5个关键词）
    3. 一句话摘要（15字内）

    咖啡店名称：{review.cafe_name}
    评论内容：{review.content}

    请以JSON格式返回：
    {{
      "sentiment": "positive",
      "keywords": ["环境好", "咖啡香", "服务周到"],
      "summary": "环境舒适，咖啡品质优秀"
    }}
    """

    response = model.generate_content(prompt)
    # 解析并返回结果
    return parse_gemini_response(response.text)
```

Node.js 调用 AI 服务

```javascript
// backend/src/services/aiService.js
const axios = require('axios');

const AI_SERVICE_URL = process.env.AI_SERVICE_URL || 'http://localhost:8000';

async function analyzeReview(content, cafeName) {
  try {
    const response = await axios.post(`${AI_SERVICE_URL}/analyze`, {
      content,
      cafe_name: cafeName
    });
    return response.data;
  } catch (error) {
    console.error('AI分析失败:', error);
    return null;
  }
}

module.exports = { analyzeReview };
```

🚀 实施步骤

Phase 1: 后端基础搭建（1-2周）

1. 初始化项目

```bash
mkdir sipspot-backend && cd sipspot-backend
npm init -y
npm install express mongoose dotenv cors bcryptjs jsonwebtoken
npm install -D nodemon
```

2. 配置MongoDB连接
   - 设置MongoDB Atlas或本地MongoDB
   - 配置环境变量（.env文件）
3. 实现用户认证
   - User模型
   - JWT生成与验证

- 注册/登录路由

4. 实现咖啡店**CRUD**
   - Cafe模型（包含GeoJSON）
   - 基础CRUD路由
   - 图片上传（Cloudinary集成）

Phase 2: 前端基础搭建（1-2周）

1. 创建**React**项目

```bash
npx create-react-app sipspot-frontend
cd sipspot-frontend
npm install axios react-router-dom react-query
npm install tailwindcss @ant-design/icons antd
```

2. 配置**Tailwind CSS**
3. 实现基础组件
   - Navbar、Footer
   - 认证页面（Login/Register）
   - 咖啡店列表页
4. 实现状态管理
   - AuthContext
   - React Query配置

Phase 3: 核心功能开发（2-3周）

1. 地图集成
   - 安装Leaflet：`npm install leaflet react-leaflet`
   - 实现地图展示组件
   - 咖啡店标记点击交互
2. 评论系统
   - 评论表单（含图片上传）
   - 评论列表展示

- 评分系统

3. **地理位置搜索**
   - 使用浏览器Geolocation API
   - 实现"附近咖啡店"功能
   - MongoDB地理查询

Phase 4: AI功能集成（1-2周）

1. **Python AI服务**

```bash
pip install fastapi uvicorn google-generativeai
```

2. **Gemini API配置**
   - 获取API密钥
   - 实现评论分析端点

3. **前后端集成**
   - 后端调用AI服务
   - 前端展示AI分析结果

Phase 5: 优化与部署（1周）

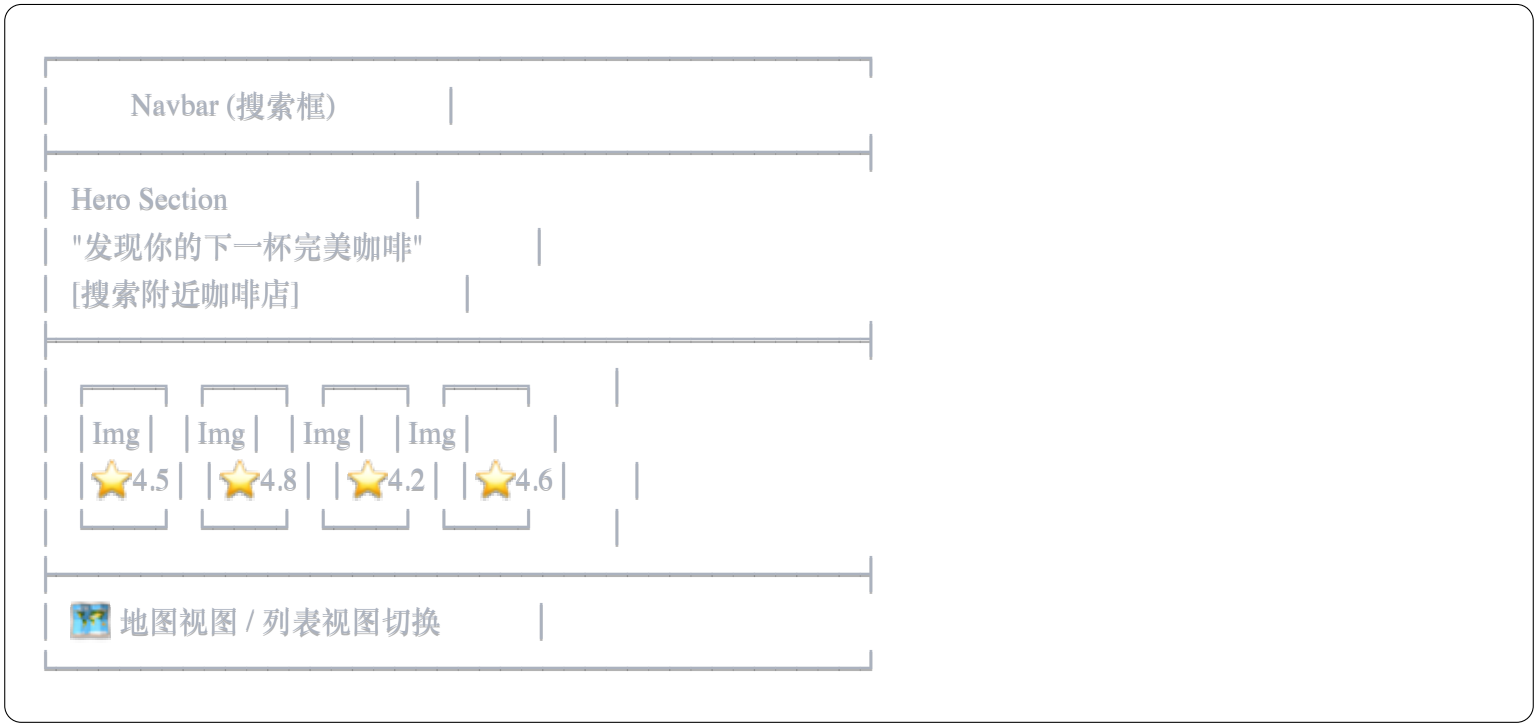
1. **性能优化**
   - 图片压缩
   - API缓存策略
   - 前端代码分割

2. **部署**
   - 后端：Render/Railway
   - 前端：Vercel
   - 数据库：MongoDB Atlas
   - AI服务：Render/Railway

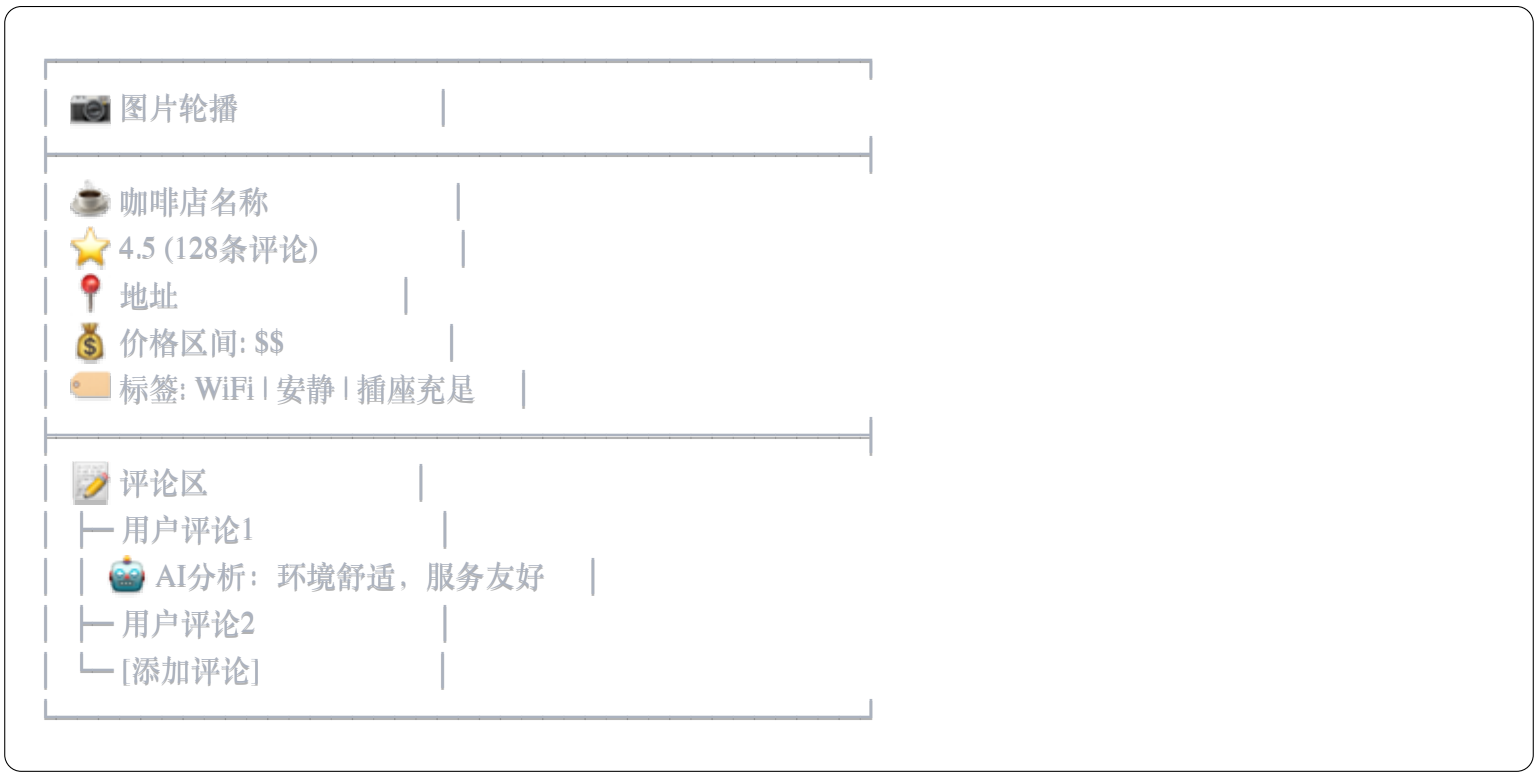
🎨 UI/UX 设计建议

首页布局

首页布局



咖啡店详情页



安全性考虑

1. 认证安全
   - 密码使用bcrypt哈希
   - JWT存储在httpOnly cookie
   - 实现刷新token机制
2. 输入验证

- 使用Joi/express-validator
- 防止XSS攻击
- SQL/NoSQL注入防护

3. **API限流**
   - 使用express-rate-limit
   - 防止暴力破解

📊 测试策略

1. 后端测试

```bash
npm install -D jest supertest
```

- API端点测试
- 数据库操作测试
- 认证流程测试

2. 前端测试

```bash
npm install -D @testing-library/react @testing-library/jest-dom
```

- 组件单元测试
- 用户交互测试

🎯 未来扩展功能

1. 社交功能
   - 用户关注系统
   - 咖啡店打卡
   - 分享到社交媒体
2. 高级搜索
   - 按标签筛选（WiFi、安静、有插座等）

- 按价格区间筛选
- 营业时间筛选

3. 推荐系统
   - 基于用户偏好的推荐
   - 类似咖啡店推荐
4. 实时功能
   - Socket.io实现实时评论
   - 咖啡店拥挤度实时更新

📚 学习资源

- React官方文档
- Express.js指南
- MongoDB地理查询
- Leaflet.js文档
- Gemini API文档

🤝 项目管理建议

1. 版本控制
   - 使用Git + GitHub
   - 遵循Git Flow工作流
   - 写清晰的commit message
2. 开发规范
   - ESLint + Prettier代码格式化
   - 统一的命名规范
   - 代码审查机制
3. 文档维护
   - API文档（Swagger/Postman）
   - README.md