ShadowMap 智能出行阴影规划系统 - 开发文档

🎯 项目愿景与目标

核心理念

基于 shademap.app 的阴影模拟技术，打造**智能出行规划系统**，为用户提供基于阴影和天气数据的出行建议。

产品差异化定位

- **基础功能**：复现shademap.app的全球阴影模拟能力
- **核心创新**：面向出行场景的智能规划功能
  - 🚶 **最少阳光路径规划** - 为用户规划晒太阳最少的出行路线
  - 📊 **光照时间序列分析** - 分析GPS轨迹的完整光照暴露情况
  - 🌤️ **天气数据融合** - 结合云层、UV指数等实时天气数据
  - 📈 **一日光照统计** - 类似健康应用的步数统计功能

📋 技术栈架构

前端技术栈

- **框架**: React 19 + TypeScript + Vite
- **地图**: Leaflet + leaflet-shadow-simulator (阴影计算引擎)
- **状态管理**: Zustand
- **样式**: Tailwind CSS
- **数据获取**: 自建API服务

后端技术栈

- **服务器**: Node.js + Express + TypeScript
- **数据库**: MongoDB (地理数据、用户轨迹、缓存)
- **地理数据**: 自建DEM瓦片服务 + OSM建筑物数据
- **天气数据**: 集成 earth.nullschool.net API

数据源策略

- **DEM地形数据**: AWS Terrarium格式，支持全球覆盖
- **建筑物数据**: OSM Overpass API + 智能缓存策略
- **天气数据**: nullschool.net (UV指数、云层覆盖、温度等)

🚀 开发阶段规划

✅ 第一阶段：基础架构 (已完成)

- [x] 前后端分离架构搭建
- [x] React + TypeScript + Leaflet 集成
- [x] Express API服务器
- [x] leaflet-shadow-simulator 插件集成
- [x] DEM瓦片服务 (支持北京地区真实数据)
- [x] OSM建筑物API集成
- [x] 多底图支持系统

🔄 第二阶段：核心功能完善 (进行中)

- [x] 建筑物高度与阴影计算联动
- [x] 时间控制和动画系统
- [x] 点击分析功能
- [x] 高级缓存管理系统
- [ ] **MongoDB数据库集成** (替换文件缓存)
- [ ] **OSM API速率限制优化** (预处理热门城市数据)

🎯 第三阶段：出行规划功能 (核心创新)

3.1 路径规划算法

```
// 最少阳光路径优化
class ShadowRouteOptimizer {
  async findMinSunRoute(start: LatLng, end: LatLng, time: Date) {
    // 1. 获取多条可选路径
    const routes = await this.getAllPossibleRoutes(start, end);

    // 2. 分析每条路径的阴影暴露
    const analyzed = await Promise.all(
      routes.map(route => this.analyzeSunExposure(route, time))
    );

    // 3. 综合评分：阴影覆盖率 + 路径长度 + 舒适度
    return this.rankRoutesByComfort(analyzed);
  }

  private async analyzeSunExposure(route: Route, time: Date) {
    const segments = this.divideRouteIntoSegments(route);
    let totalSunExposure = 0;

    for (const segment of segments) {
      const shadowData = await this.calculateShadowForSegment(segment, time);
      const weatherData = await this.getWeatherData(segment.center, time);

      // 考虑云层遮挡效果
```

```
      const effectiveSun = shadowData.sunlight * (1 - weatherData.cloudCover *
0.8);
      const uvExposure = effectiveSun * weatherData.uvIndex;

      totalSunExposure += uvExposure * segment.duration;
    }

    return {
      route,
      totalSunExposure,
      comfort: this.calculateComfort(totalSunExposure),
      estimatedTime: route.duration
    };
  }
}
```

3.2 GPS轨迹分析

```
// 光照时间序列分析
class TrajectoryAnalyzer {
  async analyzeLightExposure(gpsTrack: GPSPoint[]) {
    const timeSeriesData: LightExposurePoint[] = [];

    for (let i = 0; i < gpsTrack.length - 1; i++) {
      const segment = {
        start: gpsTrack[i],
        end: gpsTrack[i + 1],
        duration: gpsTrack[i + 1].timestamp - gpsTrack[i].timestamp
      };

      // 获取阴影数据
      const shadowData = await this.calculateShadowForSegment(segment);

      // 获取天气数据
      const weatherData = await this.getWeatherData(segment.start,
segment.start.timestamp);

      timeSeriesData.push({
        time: gpsTrack[i].timestamp,
        location: gpsTrack[i],
        sunlightRatio: shadowData.sunlightRatio,
        uvExposure: shadowData.uvExposure * weatherData.uvIndex,
        comfort: this.calculateComfort(weatherData),
        cloudCover: weatherData.cloudCover
      });
    }

    return this.generateDailyReport(timeSeriesData);
  }

  private generateDailyReport(data: LightExposurePoint[]) {
    return {
      totalSunlightMinutes: data.reduce((sum, point) => sum + point.sunlightRatio
* point.duration, 0),
      totalUVExposure: data.reduce((sum, point) => sum + point.uvExposure, 0),
```

```
      averageComfort: data.reduce((sum, point) => sum + point.comfort, 0) /
data.length,
      peakExposureTime: data.reduce((max, point) => point.uvExposure >
max.uvExposure ? point : max),
      recommendations: this.generateRecommendations(data)
    };
  }
}
```

3.3 天气数据集成

```
// 天气数据增强阴影计算
class WeatherEnhancedShadowCalculator {
  async calculateWithWeather(location: LatLng, time: Date) {
    // 1. 基础阴影计算
    const baseShadow = await this.calculateBaseShadow(location, time);

    // 2. 获取实时天气数据
    const weather = await this.fetchWeatherData(location, time);

    // 3. 云层修正
    const cloudCover = weather.cloudCover; // 0-1
    const effectiveSunlight = baseShadow.sunlight * (1 - cloudCover * 0.8);

    // 4. UV指数修正
    const uvIndex = weather.uvIndex;
    const uvExposure = effectiveSunlight * uvIndex;

    return {
      ...baseShadow,
      effectiveSunlight,
      uvExposure,
      comfort: this.calculateComfort(weather),
      recommendations: this.generateRecommendations(weather, baseShadow)
    };
  }

  private async fetchWeatherData(location: LatLng, time: Date) {
    // 集成 nullschool.net API
    const response = await fetch(`/api/weather?
lat=${location.lat}&lng=${location.lng}&time=${time.toISOString()}`);
    return response.json();
  }
}
```

🎨 第四阶段：用户体验优化

4.1 智能推荐系统

- **最佳出行时间推荐**: 基于天气预报和阴影预测
- **实时路径调整**: 根据云层变化动态调整路线
- **个性化设置**: 用户可设置对阳光的敏感度

4.2 可视化增强

- **实时路径建议**: 类似导航，但优化目标是舒适度
- **一日光照报告**: 类似健康应用的活动统计
- **阴影轨迹可视化**: 显示太阳路径和阴影变化

💾 MongoDB数据库设计

集合结构设计

```
// 建筑物数据集合
db.buildings.createIndex({ "geometry": "2dsphere" }); // 地理空间索引
{
  _id: ObjectId,
  geometry: {
    type: "Polygon",
    coordinates: [[[lng, lat], ...]]
  },
  properties: {
    height: 25.5,
    levels: 8,
    building_type: "residential"
  },
  tile: { z: 15, x: 26976, y: 13487 }, // 瓦片索引
  last_updated: ISODate
}

// 用户出行轨迹集合
db.user_tracks.createIndex({ "route.geometry": "2dsphere" });
{
  _id: ObjectId,
  user_id: "user123",
  route: {
    type: "LineString",
    coordinates: [[lng, lat, timestamp], ...]
  },
  analysis: {
    total_sunlight_minutes: 45,
    total_uv_exposure: 2.3,
    comfort_score: 7.5,
    shadow_segments: [...]
  },
  created_at: ISODate
}

// DEM瓦片元数据集合
db.dem_tiles.createIndex({ "z": 1, "x": 1, "y": 1 });
{
  _id: "15/26976/13487",
  z: 15, x: 26976, y: 13487,
  bounds: [lng1, lat1, lng2, lat2],
  file_path: "/data/dem/15/26976/13487.png",
  cached: true,
  created_at: ISODate
```

```
}
```

数据分片策略

- **地理分片**: 按经纬度范围分片，支持全球数据分布
- **时间分片**: 用户轨迹按时间分片，优化查询性能

🌐 API接口设计

核心API端点

```
// 阴影计算相关
POST /api/shadow/calculate          // 计算指定区域阴影
GET  /api/shadow/realtime           // 实时阴影数据
POST /api/shadow/analyze-route      // 分析路径阴影暴露

// 出行规划相关 (新增)
POST /api/routes/optimize           // 最少阳光路径规划
POST /api/routes/analyze            // GPS轨迹分析
GET  /api/routes/recommendations    // 获取出行建议

// 天气数据相关 (新增)
GET  /api/weather/current           // 当前天气数据
GET  /api/weather/forecast          // 天气预报
GET  /api/weather/uv-index          // UV指数查询

// 地理数据相关
GET  /api/dem/:z/:x/:y.png          // DEM瓦片
GET  /api/buildings/:z/:x/:y.json   // 建筑物数据
GET  /api/terrain/elevation         // 高程查询

// 用户数据相关 (新增)
POST /api/users/tracks              // 保存用户轨迹
GET  /api/users/tracks/:id          // 获取轨迹分析
GET  /api/users/stats              // 用户光照统计
```

🔧 技术挑战与解决方案

1. OSM API速率限制问题

问题: Overpass API限制每分钟60个请求
解决方案:

- **数据预处理**: 预下载热门城市建筑物数据到MongoDB
- **智能缓存**: 实现多级缓存(内存 + 数据库 + 文件)
- **请求队列**: 实现速率控制和请求去重

2. 大规模地理数据处理

问题: 全球DEM和建筑物数据量巨大
解决方案:

- **分级数据策略**: 核心城市预存储，其他地区API调用
- **数据压缩**: 使用高效的地理数据压缩算法
- **边缘缓存**: CDN缓存静态瓦片数据

3. 实时天气数据集成

问题: 需要集成多个天气数据源
解决方案:

- **统一天气API**: 封装nullschool.net等多个数据源
- **数据缓存**: 天气数据按时间和位置缓存
- **降级策略**: 数据源失效时的备用方案

📊 当前开发状态

✅ 已完成功能

- [x] 基础阴影模拟系统 (React + leaflet-shadow-simulator)
- [x] DEM地形数据集成 (AWS Terrarium格式)
- [x] OSM建筑物数据API (Overpass API集成)
- [x] 多底图支持 (OSM、CartoDB、ESRI、Mapbox等)
- [x] 时间控制和动画系统
- [x] 点击分析功能 (日照时长计算)
- [x] 高级缓存管理系统
- [x] 现代化UI设计 (Tailwind CSS)

🔄 进行中任务

- [ ] MongoDB数据库集成 (替换文件缓存系统)
- [ ] 天气数据API集成 (nullschool.net)
- [ ] 路径规划算法实现
- [ ] GPS轨迹分析功能

📝 下一阶段计划 (优先级排序)

高优先级 (1-2周)

1. **MongoDB集成**: 替换现有文件缓存，提升数据查询性能
2. **OSM数据预处理**: 预下载热门城市数据，减少API依赖
3. **天气数据集成**: 实现nullschool.net API集成

中优先级 (2-4周)

4. **路径规划算法**: 实现最少阳光路径计算
5. **GPS轨迹分析**: 实现光照时间序列分析
6. **用户系统**: 实现轨迹保存和统计功能

低优先级 (1-2个月)

7. **性能优化**: WebGL渲染、数据压缩等
8. **移动端适配**: 响应式设计和PWA支持
9. **生产部署**: Docker容器化和CI/CD流水线

🔄 替补需求 (未来考虑)

10. **自研阴影计算引擎**: 替换leaflet-shadow-simulator，实现完全自主的阴影计算
    - 自研太阳位置算法
    - 自研光线追踪算法
    - 自定义GPU着色器程序
    - 针对出行场景的性能优化

🎯 商业化考虑

目标用户群体

- **个人用户**: 注重防晒和舒适出行的用户
- **健康应用开发商**: 需要光照数据的应用集成
- **城市规划师**: 需要阴影分析的专业用户
- **建筑设计师**: 需要采光分析的设计师

商业模式

- **API调用收费**: 按请求量计费的API服务
- **SaaS订阅**: 面向企业用户的订阅服务
- **数据授权**: 向其他应用提供地理和阴影数据

竞争优势

- **自主技术栈**: 完全基于自有服务器，不依赖第三方
- **出行场景优化**: 针对实际出行需求的功能创新
- **全球数据覆盖**: 支持全球任意位置的阴影分析
- **实时天气集成**: 结合天气数据的智能推荐

📚 技术架构总结

系统架构图

```
用户界面 (React + Leaflet)
    ↓
状态管理 (Zustand Store)
    ↓
API服务层 (自建Express API)
    ↓
数据存储层 (MongoDB + 文件缓存)
    ↓
外部数据源 (AWS DEM + OSM + Weather APIs)
```

关键技术决策

1. **前端**: React生态系统，现代化开发体验
2. **地图**: Leaflet + leaflet-shadow-simulator，成熟的阴影计算方案
3. **数据源**: 完全自主的DEM和建筑物数据，摆脱第三方依赖
4. **数据库**: MongoDB，原生支持地理空间数据
5. **缓存**: 多级缓存策略，平衡性能和成本
6. **部署**: Docker容器化，支持水平扩展

🔧 当前技术架构说明

- **阴影计算**: 使用leaflet-shadow-simulator作为计算引擎，但数据完全来自自建服务
- **数据独立性**: DEM瓦片和建筑物数据均通过自建API提供
- **计算逻辑**: 基于物理光线追踪算法，支持真实的阴影模拟
- **扩展性**: 为未来替换为自研引擎预留了架构空间

这个更新后的文档清晰地体现了项目从基础阴影模拟到智能出行规划的升级路径，为后续开发提供了明确的技术路线图。