Svelte 应用 API 使用手册 (已解耦)

本文档旨在帮助 Svelte 开发者理解并使用后端提供的 Flask API 服务。API 服务划分为几个模块（蓝图）：认证 (Auth)、博客 (Blog)、待办事项 (Todo)、成就 **(Achievements)**、未来计划 **(Plans)**、用户资料 (Anchor/Profile) 和人工智能 (AI)。

Svelte 集成通用指南

（此部分与之前版本相同，保持不变）

1. 基础 **URL (Base URL)**：
   - 在 Svelte 应用中定义一个 API 请求的基础 URL。例如，在开发环境中可能是 http://localhost:5000/api，在生产环境中则是您的部署域名。
   - 建议将基础 URL 存储在环境变量或配置文件中。
2. **HTTP** 请求：
   - 使用 JavaScript 内置的 fetch API 或如 axios 这样的 HTTP 客户端库来发送请求。
   - 对于 POST, PUT, PATCH 请求，确保将 Content-Type 请求头设置为 application/json（如果API期望接收JSON格式的数据）。
3. 请求体 **(Request Body)**：
   - 对于需要发送数据的请求（如 POST, PUT），通常需要将 JavaScript 对象通过 JSON.stringify() 转换为 JSON 字符串作为请求体。
4. 认证 **(Authentication)**：
   - 许多 API 端点受 JWT (JSON Web Tokens) 保护。
   - 登录后：从登录接口 (/auth/login) 获取 access_token 和 refresh_token。将这些令牌安全地存储起来（例如，使用 Svelte 的 store，并可选择性地持久化到 localStorage 或 sessionStorage）。
   - 发送令牌：对于需要认证的请求，在 Authorization 请求头中附带 access_token，格式为 Bearer <your_access_token>。
   - 令牌刷新：access_token 通常有较短的有效期。当它过期时（API 返回 401 Unauthorized），您需要使用 refresh_token 调用 /auth/refresh 接口来获取新的 access_token。
   - 登出：调用 /auth/logout (撤销 access token) 和 /auth/logout-refresh (撤销 refresh token) 来使令牌失效。
5. 响应处理 **(Response Handling)**：
   - 检查 HTTP 响应状态码。2xx 通常表示成功，4xx 表示客户端错误（如输入无效、未授权），5xx 表示服务器端错误。
   - 解析响应体。大多数 API 会返回 JSON 格式的数据。使用 .json() 方法处理 fetch 的响应。
6. 错误处理 **(Error Handling)**：
   - 在 Svelte 组件中实现健壮的错误处理逻辑。向用户显示有意义的错误信息。

    - 捕获网络错误以及 API 返回的特定错误信息。
7. 加载状态 **(Loading States)**:
    - 在发起 API 请求和等待响应期间，向用户显示加载指示器（如 spinners 或 loading 消息），以改善用户体验。
8. 跨域请求 **(CORS)**:
    - 在开发过程中，Flask 后端需要配置 CORS (Cross-Origin Resource Sharing) 以允许来自 Svelte 开发服务器 (通常是 http://localhost:xxxx，如 http://localhost:5173) 的请求。生产环境中也需要正确配置。

API 模块详解

1. 认证 **API (/api/auth)**

（此部分与之前版本相同，保持不变 ）

- 基础路径: /api/auth
    - GET /ping
    - POST /register
    - POST /login
    - POST /logout
    - POST /logout-refresh
    - GET /me
    - POST /refresh
    - POST /change-password
    - *(详细描述请参考之前版本)*

2. 博客 **API (/api/blog)**

（此部分与之前版本相同，保持不变，假设存在）

- 基础路径: /api/blog
    - GET /ping
    - GET /posts
    - GET /posts/<post_id>
    - POST /posts
    - PUT /posts/<post_id>
    - DELETE /posts/<post_id>
    - *(详细描述请参考之前版本)*

3. 待办事项 **API (/api/todo)**

（此部分与之前版本相同，保持不变 ）

- 基础路径: /api/todo
    - GET /ping

- GET /todos
- POST /todos
- GET /todos/<todo_id>
- PUT /todos/<todo_id>
- DELETE /todos/<todo_id>
- *(详细描述请参考之前版本)*

4. 成就 **API (/api/achievements)**

此模块管理用户的"成就" (Done) 信息。

- 基础路径: /api/achievements

4.1 Ping 成就服务

- 用途: 测试成就 API 是否可用。
- 方法: GET
- **URL**: /api/achievements/ping
- 成功响应 **(200)**: {"message": "Achievements API is alive!"}

4.2 创建成就

- 用途: 为当前用户添加一条新的成就记录。
- 方法: POST
- **URL**: /api/achievements/
- 认证: @jwt_required()
- 请求体 **(JSON)**:

```
{
  "title": "项目A成功上线", // 必填, 非空字符串
  "description": "负责项目A的前后端开发...", // 可选, 字符串
  "quantifiable_results": "用户增长20%", // 可选, 字符串
  "core_skills_json": ["Svelte", "API设计", "项目管理"], // 可选, 字符串列表
  "date_achieved": "2024-10-15" // 可选, YYYY-MM-DD
}
```

- 成功响应 **(201)**: 新创建的成就对象。
- 错误响应:
  - 400 Bad Request: 字段无效。

4.3 获取所有成就

- 用途: 获取当前用户的所有成就记录，按完成日期降序、创建时间降序排列。
- 方法: GET
- **URL**: /api/achievements/
- 认证: @jwt_required()

- 成功响应 **(200)**: 成就对象数组。

4.4 获取单个成就

- 用途: 根据 ID 获取单个成就记录。
- 方法: GET
- **URL**: /api/achievements/<achievement_id>
- 认证: @jwt_required()
- 成功响应 **(200)**: 成就对象。
- 错误响应:
  - 403 Forbidden: 无权访问。
  - 404 Not Found: 未找到。

4.5 更新成就

- 用途: 更新指定的成就记录。
- 方法: PUT
- **URL**: /api/achievements/<achievement_id>
- 认证: @jwt_required()
- 请求体 **(JSON)**: 包含要更新的字段。
- 成功响应 **(200)**: 更新后的成就对象。
- 错误响应:
  - 400 Bad Request: 字段无效。
  - 403 Forbidden: 无权更新。
  - 404 Not Found: 未找到。

4.6 删除成就

- 用途: 删除指定的成就记录。
- 方法: DELETE
- **URL**: /api/achievements/<achievement_id>
- 认证: @jwt_required()
- 成功响应 **(204 No Content)**: 响应体为空。
- 错误响应:
  - 403 Forbidden: 无权删除。
  - 404 Not Found: 未找到。

5. 未来计划 **API (/api/plans)**

此模块管理用户的“未来计划” (Plan) 信息。

- 基础路径: /api/plans

5.1 Ping 计划服务

- 用途: 测试计划 API 是否可用。

- 方法: GET
- **URL**: /api/plans/ping
- 成功响应 **(200)**: {"message": "Plans API is alive!"}

5.2 创建未来计划

- 用途: 为当前用户添加一个新的未来计划。
- 方法: POST
- **URL**: /api/plans/
- 认证: @jwt_required()
- 请求体 **(JSON)**:
  {
  "title": "学习AI技术", // 必填, 非空字符串
  "description": "系统学习机器学习和深度学习基础。", // 必填, 非空字符串
  "goal_type": "学习提升", // 可选, 字符串, 最多50字符
  "status": "active", // 可选, 默认 'active'。允许值: 'active', 'achieved', 'deferred', 'abandoned'
  "target_date": "2026-12-31" // 可选, YYYY-MM-DD
  }

- 成功响应 **(201)**: 新创建的未来计划对象。
- 错误响应:
  - 400 Bad Request: 字段无效。

5.3 获取所有未来计划

- 用途: 获取当前用户的所有未来计划，按目标日期升序、创建时间降序排列。
- 方法: GET
- **URL**: /api/plans/
- 认证: @jwt_required()
- 成功响应 **(200)**: 未来计划对象数组。

5.4 获取单个未来计划

- 用途: 根据 ID 获取单个未来计划。
- 方法: GET
- **URL**: /api/plans/<plan_id>
- 认证: @jwt_required()
- 成功响应 **(200)**: 未来计划对象。
- 错误响应:
  - 403 Forbidden: 无权访问。
  - 404 Not Found: 未找到。

5.5 更新未来计划

- 用途: 更新指定的未来计划。
- 方法: PUT
- **URL**: /api/plans/<plan_id>
- 认证: @jwt_required()
- 请求体 **(JSON)**: 包含要更新的字段。
- 成功响应 **(200)**: 更新后的未来计划对象。
- 错误响应:
  - 400 Bad Request: 字段无效。
  - 403 Forbidden: 无权更新。
  - 404 Not Found: 未找到。

5.6 删除未来计划

- 用途: 删除指定的未来计划。
- 方法: DELETE
- **URL**: /api/plans/<plan_id>
- 认证: @jwt_required()
- 成功响应 **(204 No Content)**: 响应体为空。
- 错误响应:
  - 403 Forbidden: 无权删除。
  - 404 Not Found: 未找到。

6. 用户资料 **API (/api/anchor)**

此模块现在主要管理用户的“看板资料” (Profile) 信息。原有的“成就”和“未来计划”已移至独立的 API 模块。

- 基础路径: /api/anchor

6.1 Ping 用户资料服务

- 用途: 测试用户资料 API 是否可用。
- 方法: GET
- **URL**: /api/anchor/ping
- 成功响应 **(200)**: {"message": "Anchor (Profile) API is alive!"}

6.2 用户看板资料 **(User Profile)**

6.2.1 获取用户看板资料

- 用途: 获取当前认证用户的看板资料。如果用户尚无资料，会自动创建一条空的资料记录。
- 方法: GET
- **URL**: /api/anchor/profile

- 认证: @jwt_required()
- 成功响应 **(200)**: 用户资料对象。

```
{
  "id": 1, // user_id
  "professional_title": "软件工程师", // 可为 null
  "one_liner_bio": "热衷于构建可扩展的 Web 应用。", // 可为 null
  "skill": "Svelte, Python, Flask", // 可为 null
  "summary": "详细的个人总结...", // 可为 null
  "created_at": "YYYY-MM-DDTHH:MM:SS.ffffffZ",
  "updated_at": "YYYY-MM-DDTHH:MM:SS.ffffffZ"
}
```

6.2.2 更新用户看板资料

- 用途: 更新当前认证用户的看板资料。
- 方法: PUT
- **URL**: /api/anchor/profile
- 认证: @jwt_required()
- 请求体 **(JSON)**: 包含要更新的字段。

```
{
  "professional_title": "高级软件工程师", // 可选
  "one_liner_bio": "更新的简介。" // 可选
  // "skill": ..., "summary": ...
}
```

- 成功响应 **(200)**: 更新后的用户资料对象。
- 错误响应:
  - 400 Bad Request: 请求体为空或字段类型错误。

7. AI API (/api/ai)

（此部分与之前版本相同，保持不变，假设存在）

- 基础路径: /api/ai
  - GET /ping
  - POST /generate-text
  - *(详细描述请参考之前版本)*

Svelte 代码示例片段

（此部分与之前版本相同，保持不变，仅需注意端点 URL 的更新）

<script>

```
  import { onMount } from 'svelte';
  // import { accessToken } from './authStore'; // 假设您有一个 authStore

  let data = null;
  let error = null;
  let loading = false;
  const API_BASE_URL = 'http://localhost:5000/api'; // 或者从环境变量读取

  async function fetchData(endpoint, method = 'GET', body = null, requiresAuth =
false) {
    loading = true;
    error = null;
    try {
      const headers = {
        'Content-Type': 'application/json',
      };
      if (requiresAuth) {
        const token = localStorage.getItem('accessToken'); // 或者从 store 获取
        if (!token) {
          throw new Error('No access token found. Please log in.');
        }
        headers['Authorization'] = `Bearer ${token}`;
      }

      const options = {
        method,
        headers,
      };

      if (body && (method === 'POST' || method === 'PUT')) {
        options.body = JSON.stringify(body);
      }

      const response = await fetch(`${API_BASE_URL}${endpoint}`, options);

      if (!response.ok) {
        const errorData = await response.json().catch(() => ({ message:
response.statusText }));
        throw new Error(errorData.error || errorData.message || `HTTP error! status:
```

```
${response.status}`);
      }

      if (response.status === 204) { // No Content
        return null;
      }
      return await response.json();

    } catch (err) {
      error = err.message;
      console.error(`Failed to ${method} ${endpoint}:`, err);
      return null; // 或抛出错误供上层处理
    } finally {
      loading = false;
    }
  }

  async function getMyAchievements() {
    // 示例：调用新的成就 API
    data = await fetchData('/achievements/', 'GET', null, true);
  }

  async function createNewPlan() {
    // 示例：调用新的计划 API
    const newPlanData = { title: "My new Svelte plan", description: "Details of the plan"
};
    const result = await fetchData('/plans/', 'POST', newPlanData, true);
    if (result) {
      // 成功创建
      console.log("New plan created:", result);
      // getMyPlans(); // 刷新计划列表
    }
  }

  onMount(() => {
    if (localStorage.getItem('accessToken')) {
      getMyAchievements(); // 例如，加载成就数据
    }
  });
```

```
</script>

{#if loading}
  <p>正在加载...</p>
{/if}

{#if error}
  <p style="color: red;">错误: {error}</p>
{/if}

{#if data}
  <pre>{JSON.stringify(data, null, 2)}</pre>
{/if}

<button on:click={createNewPlan} disabled={loading}>创建新计划</button>
<button on:click={getMyAchievements} disabled={loading}>刷新成就列表</button>
```

请根据您的具体需求调整上述代码。确保在 Svelte 应用中妥善管理认证令牌、用户状态和错误信息，并使用新的 API 端点。