**Ingress 打卡点系统 - 代码规范文档**

**一、命名规范**

**1. 组件命名**

- 使用 PascalCase 命名组件文件和组件名

- 组件名应当是描述性的，如 LoginPage、CheckpointCard

- 避免使用缩写，保持名称的完整性和可读性

**2. 变量命名**

- 使用 camelCase 命名变量

- 布尔值变量使用 is、has、can 等前缀

- 常量使用全大写，下划线分隔

示例：

```javascript

const isLoggedIn = true;

const hasCheckedIn = false;

const API\_BASE\_URL = 'http://localhost:5000/api';

```

**3. 方法命名**

- 使用动词开头的 camelCase

- 事件处理方法使用 handle 前缀

- 清晰表达方法的功能和意图

示例：

```javascript

handleLogin()

handleCheckIn()

handleCapture()

```

**二、代码组织**

**1. 文件结构**

- 按功能模块组织文件

- 相关文件放在同一目录下

- 保持目录结构清晰和合理

```

src/

├── components/ // 组件文件

├── store/ // 状态管理

├── router/ // 路由配置

└── config/ // 配置文件

```

**2. 组件结构**

- template 在前，script 在中，style 在后

- 组件选项按固定顺序排列：name、components、props、data、computed、methods

- 使用 scoped style 避免样式污染

**三、代码风格**

**1. 缩进和空格**

- 使用 2 空格缩进

- 运算符前后加空格

- 对象和数组的最后一项后不加逗号

**2. 注释规范**

- 关键逻辑添加注释说明

- 避免无意义的注释

- 使用 JSDoc 风格的方法注释

示例：

```javascript

/\*\*

\* 处理用户打卡操作

\* @param {string} checkpointId - 打卡点ID

\* @param {string} userType - 用户类型

\* @returns {Promise<void>}

\*/

async handleCheckIn(checkpointId, userType) {

// 实现逻辑

}

```

**四、最佳实践**

**1. 组件设计**

- 单一职责原则

- 组件功能独立

- 避免过度嵌套

- 提取可复用逻辑

**2. 状态管理**

- 合理使用 LocalStorage

- 集中管理全局状态

- 避免状态混乱

**3. 错误处理**

- 统一的错误处理机制

- 友好的错误提示

- 错误日志记录

示例：

```javascript

try {

await handleOperation();

} catch (error) {

console.error('Operation failed:', error);

alert('操作失败，请重试。');

}

```

 **持久性**：数据保存在浏览器中，即使关闭页面或浏览器后，数据也不会丢失。

 **键值对存储**：数据以字符串的键值对方式存储。

 **存储容量**：大约为 5MB，容量根据浏览器和设备可能有所不同。

 **同源访问**：同一域名下的页面可以访问相同的 localStorage 数据，但不同域名或子域之间无法访问彼此的 localStorage 数据。

**五、性能优化**

**1. 代码优化**

- 避免不必要的计算

- 使用计算属性代替方法

- 及时清理不需要的监听器

**2. 资源优化**

- 路由懒加载

- 组件按需加载

- 合理使用缓存

**六、安全性**

**1. 数据安全**

- 敏感信息加密存储

- 输入数据验证

- XSS/CSRF 防护

**2. 权限控制**

- 路由级别的权限控制

- API 访问权限验证

- 用户身份验证

**七、测试规范**

**1. 单元测试**

- 组件测试

- 方法测试

- 路由测试

**2. 集成测试**

- 功能流程测试

- 用户交互测试

- 异常情况测试

**八、文档规范**

**1. 代码文档**

- 必要的注释说明

- API 文档

- 组件文档

**2. 项目文档**

- README 文件

- 技术栈说明

- 部署文档

**九、版本控制**

**1. Git 规范**

- 清晰的提交信息

- 合理的分支管理

- 版本标签管理

**2. 发布规范**

- 语义化版本号

- 更新日志维护

- 发布流程规范

**十、持续改进**

**1. 代码审查**

- 定期代码审查

- 及时修复问题

- 优化代码质量

**2. 重构原则**

- 识别代码异味

- 合理的重构时机

- 保持向后兼容