1. 环境规范
2. 文件夹及其文件的结构和命名规范
    2.1. 文件夹结构
    2.2. 命名规范
3. 编码规范
    3.1. HTML/Vue模板
        3.1.1. 缩进和空格
        3.1.2. 标签和属性
        3.1.3. 注释
        3.1.4. 指令和事件
        3.1.5. 条件渲染和列表渲染
    3.2. CSS/SCSS样式
        3.2.1. 缩进和空格
        3.2.2. 属性值
        3.2.3. 选择器命名
        3.2.4. 注释
        3.2.5. 避免使用内联样式
        3.2.6. 重用和扩展
    3.3. JavaScript
        3.3.1. 变量命名
        3.3.2. 函数和方法
        3.3.3. 代码风格和格式
        3.3.4. 异步处理
        3.3.5. 错误处理
        3.3.6. 代码注释
        3.3.7. 第三方库和插件
4. 接口与数据规范
    4.1. API调用
    4.2. 数据状态管理
5. 注释规范
6. 其他规范
    6.1. 代码审查
    6.2. 版本控制

前端开发规范说明书

1. 环境规范

- 开发工具：HBuilder X 4.06
- 开发框架: uni-app
- UI 框架：uView 1.0
- 第三方API集成：和风天气
- 图标：iconFont图标库
- 数据可视化：ECharts

2. 文件夹及其文件的结构和命名规范

2.1. 文件夹结构

- `pages/`：页面目录，每个页面有一个单独的文件夹。
- `static/`：静态资源目录，如图片、图标、字体等，各自用一个文件夹存放，外部引入的资源默认使用其文件夹。

2.2. 命名规范

- 文件名直接使用小写英文单词命名
- Vue页面使用驼峰命名法进行命名，如`aboutUs.vue`
- 避免使用拼音或缩写，尽量使用英文描述

3. 编码规范

3.1. HTML/Vue模板

3.1.1. 缩进和空格

- 使用制表符进行缩进。
- 在元素之间、属性和值之间添加适当的空格，以提高代码的可读性。

3.1.2. 标签和属性

- 标签应小写，并闭合。
- 属性值应使用双引号括起来，避免使用单引号。
- 避免使用内联样式，尽量将样式定义在`<style>`标签或外部CSS文件中。
- 若样式在三个以内，可接受内联样式。

3.1.3. 注释

- 在需要解释或说明的地方添加注释，注释应简洁明了，避免冗余。
- 对于复杂的逻辑或特殊的实现方式，应添加详细的注释以帮助其他开发者理解。

3.1.4. 指令和事件

- 使用`v-bind`或简写`:`绑定属性。
- 使用`v-on`或简写`@`绑定事件。
- 避免在模板中直接操作数据，应通过methods或computed属性来处理逻辑。

3.1.5. 条件渲染和列表渲染

- 使用`v-if`、`v-else-if`和`v-else`进行条件渲染时，确保条件逻辑清晰明了。
- 使用`v-for`进行列表渲染时，为每一项提供一个唯一的`key`属性以提高性能。

3.2. CSS/SCSS样式

3.2.1. 缩进和空格

- 使用制表符进行缩进。
- 在选择器、属性和值之间添加适当的空格。

3.2.2. 属性值

- 属性值应使用小写，并确保使用正确的CSS属性名。
- 颜色值应使用十六进制（#fff）、RGB、RGBA或HSL格式，并尽量使用有意义的颜色变量。
- 长度值应使用合适的单位，如`px`、`rpx`、`em`、`rem`或`%`。

3.2.3. 选择器命名

- 选择器命名应简洁、描述性强，并使用小写字母和短横线分隔。
- 避免使用过于复杂的选择器，以提高性能。

3.2.4. 注释

- 在需要解释或说明的地方添加注释，注释应简洁明了，避免冗余。
- 对于复杂的样式规则或特殊实现，应添加详细的注释。

3.2.5. 避免使用内联样式

- 尽量避免在HTML标签中直接使用`style`属性来定义内联样式，以保持样式和结构的分离。

3.2.6. 重用和扩展

- 尽量重用已有的样式规则，避免重复编写相同的样式。

3.3. JavaScript

3.3.1. 变量命名

- **变量名**：使用驼峰命名法，避免使用下划线。
- **常量**：使用全大写字母和下划线分隔，如`MAX_COUNT`。
- **避免全局变量**：尽量使用局部变量或模块作用域变量。

3.3.2. 函数和方法

- **函数名**：使用动词或动词短语，描述函数的功能，使用驼峰命名法。
- **参数**：参数名应清晰明了，使用有意义的名称。
- **注释**：对于复杂的函数或方法，应添加注释说明其功能、参数和返回值。

3.3.3. 代码风格和格式

- **缩进**：使用制表符，不要使用4个空格进行缩进。
- **分号**：在语句末尾添加分号，避免因自动分号插入导致的错误。
- **空格**：操作符两边不用有空格，如`a+b`即可，而不一定是`a + b`。
- **大括号**：即使只有一条语句，也应使用大括号包围。

3.3.4. 异步处理

- **async/await**：在可能的情况下，使用`async/await`语法使异步代码更易于阅读和理解。

3.3.5. 错误处理

- **try-catch**：使用`try-catch`块捕获和处理可能发生的错误。
- **错误日志**：打印错误信息，方便调试和排查问题。

3.3.6. 代码注释

- **注释内容**：应简洁明了，解释代码的目的、实现方式或注意事项。
- **避免冗余注释**：不要对显而易见的代码进行注释。
- **特殊注释**：对于某些特殊的代码段或逻辑，可以使用特殊注释标记，如TODO、FIXME等。

3.3.7. 第三方库和插件

- **按需引入**：只引入项目中实际使用的库和插件，避免不必要的依赖。

4. 接口与数据规范

4.1. API调用

- 使用统一的API调用方式，如uni.request。
- 对API返回的数据进行统一的错误处理和格式转换。

4.2. 数据状态管理

- 使用简单的data对象进行状态管理。
- 避免在组件间直接传递大量数据，使用状态管理进行共享或使用数据缓存的方法。

5. 注释规范

- 在关键代码处添加注释，解释其功能和逻辑。
- 使用统一的注释格式和风格
  - 在`<template>`区域使用`<!--  -->`。
  - 在`<script>`区域使用单行注释或多行注释，单行注释用于简要说明，多行注释用于解释复杂的代码块
  - 在`<style>`区域使用`/* */`注释

6. 其他规范

6.1. 代码审查

- 定期进行代码审查，确保代码质量。
- 进行代码格式检查和格式化。

6.2. 版本控制

- 使用Git进行版本控制，遵循分支管理策略。
- 定期合并代码，保持代码库的更新和稳定。