编程规范

一、通用规范

1. 版本控制

使用Git

- 所有项目成员应使用Git作为版本控制系统。
- 熟悉基本的Git操作，包括克隆、提交、推送、拉取、合并和分支管理。

分支策略

- 采用 GitHub Flow作为项目的分支管理策略。
- 主要分支包括：`main` 、 `develop` 等分支。

提交准则

- 提交小而频繁。
- 每次提交必须附带清晰、描述性的信息。

2. 代码审查

- 所有代码在合并到主分支之前必须至少由一名组员审查。
    - 覆盖一次完整的迭代成果前需要全组成员共同审查
- 审查应关注代码质量、设计模式、代码一致性和安全漏洞。
- 鼓励建设性的反馈和讨论。

3. 编码规范

代码格式

- 使用统一的缩进（4个空格）。
- 代码行长度不超过120字符。
- 程序中不要出现局部变量和全局变量同名的现象。
    - 尽管两者作用域不同不会发生语法错误，但会使人误解

命名约定

- 变量、函数和类名应清晰、准确，避免使用缩写。
- 类型名和函数名均以大写字母开头的单词组合而成
- 变量名和参数名采用第一个单词首字母小写而后面的单词首字母大写的单词组合(驼峰命名法)
- 符号常量用全大写的单词组合而成，单词之间用下划线分割
- 使用英文单词或其组合进行命名，切忌使用汉语拼音来命名，注意用词准确性。
- 全局函数的名字应当使用"动词"或者"动词+名词"
- 变量的名字应当使用"名词"或者"形容词+名词"的格式命名
- 程序中不要出现仅靠大小写来区分的相似标识符

- 尽量避免名字中出现数字编号，除非逻辑上确实需要如此

4. 文档

代码内文档

- 使用注释描述复杂逻辑。
- 注释应清晰、简洁。

开发文档

- 编写详细的开发文档，包括安装、配置和API描述。
- 文档应与代码同步更新。

5. 安全性

- 遵守安全编码标准，防止常见的安全威胁。
- 密钥和敏感数据不得硬编码在源代码中。

6. 性能优化

- 编写高效代码，避免不必要的计算和资源消耗。
- 合理管理内存和网络资源，避免资源泄露。

二、UI界面设计规范

1. 设计原则

- **一致性**：确保所有界面元素在不同页面保持一致的风格和行为。
- **简洁性**：界面应清晰不拥挤，避免不必要的元素。
- **直观性**：设计直观易懂的UI，确保用户能自然地使用。
- **反馈**：对用户操作提供及时和明确的反馈。

2. 布局和结构

- **导航**：提供直观的导航系统，顶部为界面导航栏。
- **布局**：顶部留有70px空白，左右35px空白。顶部操作栏TopBar占高`40px`，周视图日历占高`75px`，时间轴占宽`40px`。
- **组件**：输入框、大按钮`360px-50px`，菜单选择按钮`150px-40px`，小按钮`50px-25px`。
- **间距**：相关联的按钮之间间距`10px`，不同部分按钮之间间距`30px`，以做区分。

3. 色彩和图标

- **色彩方案**：采用较为活泼清新的颜色。主色调为`#002FA7` `#F16326`,背景颜色为`#FF9E9E` `#72C4FF`。
- **图标使用**：使用风格一致的图标来增强界面直观性。

4. 字体和文本

- **字体选择**：优先使用系统默字体，保证文本在各种设备上的可读性。认
- **文本大小**：标题大小为`20px`，正文为`16px`，小字为`14px`。

5. 交互设计

- **触控目标大小**：确保所有可触控元素大小适宜，大小大于`50px-25px`。
- **动画和过渡**：按钮点按后加深进行颜色处理，形成交互动画。
- **错误处理**：清晰展示表单填写错误或操作错误信息。

三、前端规范

使用React Native进行前端开发

1. 代码风格

- **缩进**：使用2个空格进行缩进。
- **组件命名**：使用PascalCase（大驼峰命名法）命名React组件。
- **变量和函数命名**：使用camelCase（小驼峰命名法），清晰表达意图。
- **调试日志**：使用 文件名:函数名:数据名:数据值的格式，方便定位错误，跟踪程序进程。

2. 文件结构

- `/screen`-包含应用中各个屏幕的组件。
- `/component`-存放可复用的UI组件。每个组件都应该是独立的，与业务逻辑解耦。
- `/service`-包括与外部API交互的服务。
- `/utils`-包含各种工具函数。这些工具函数应该是纯函数，不依赖应用的其他部分。
- `/css`-存放所有的样式文件。使用统一的样式规范来保持界面的一致性。
- `/router`-包括应用的路由配置。使用React Navigation或类似的库来管理屏幕导航。

3. 组件设计

- 尽可能使用函数式组件和Hooks。
- 对于所有props使用`PropTypes`进行类型检查。

4. 状态管理

- 局部状态使用`useState`。
- 前端需要多次跨界面使用的数据采用`AsyncStorage`方法暂时存储在前端。
- 前端只需要一次跨界面使用的数据可以通过`route.params`直接解构赋值。
- 通过维护isLoading状态变量，确保在需要的数据拿到之后再进行页面渲染。
- 通过`axios`的`.then`方法以及`async`函数实现必要的异步操作。

5. 性能优化

- 利用`React.memo`，`useMemo`，`useEffect`和`useCallback`来避免不必要的渲染。
- 使用`FlatList`或`SectionList`来优化长列表的渲染。

四、后端规范

使用Spring Boot进行后端开发

1. 文件结构

- `/src/main/java` - 存放所有Java源代码。
- `/controller` - 存放控制层代码，处理外部请求并返回响应。
- `/service` - 存放业务逻辑层代码，编写业务处理逻辑。
- `/repository` - 存放数据访问层代码，直接与数据库交互。
- `/model` - 存放实体类代码，定义数据模型和返回值类型。
- `/Config` - 存放配置类代码，配置项目如数据库、安全、网络、跨域问题处理等。
- `/util` - 存放工具类。

2. 编码规范

命名约定

- **类和接口名**：使用PascalCase（大驼峰命名法），如`UserProfileService`。
- **方法和变量名**：使用camelCase（小驼峰命名法），如`saveUserProfile`。
- **常量**：使用全大写字母和下划线分隔，如`MAX_SIZE`。

注释和文档

- **类和方法注释**：使用JavaDoc标准，为所有公开的类和方法提供文档注释。
- **内部代码注释**：在复杂的逻辑和算法处添加必要的解释性注释。

数据库交互

- **使用JPA**：采用Spring Data JPA进行数据持久化操作，避免使用原生SQL语句以减少SQL注入风险。
- **事务管理**：明确事务边界，合理使用`@Transactional`注解确保数据一致性。

错误处理

- **全局异常处理**：使用`@ControllerAdvice`或`@RestControllerAdvice`来处理全局异常。
- **自定义异常**：定义业务相关的异常类，使错误处理更加清晰。

安全性

- **API安全**：使用Spring Security或相应的库来保护应用程序，确保敏感接口的访问控制。
- **密码处理**：使用强哈希算法（如BCrypt）来存储和验证用户密码,同时将密码在后端处理，避免前端直接处理密码。

3. 配置管理

- **外部化配置**：使用`application.properties`或`application.yml`外部化配置信息，如数据库连接、API密钥等。

五、数据库规范

使用MySQL支持数据库操作。

1. 数据库设计

表结构设计

- **命名约定**：字段名使用小写字母和下划线（snake_case），表名使用小写单词拼接。例如：`eventtable`。
- **主键**：每个表应有一个唯一的主键，通常使用自增的整数`id`。
- **外键**：明确定义外键关系，以保证数据的完整性。
- **索引**：为提高查询效率，对常查询的列如日期、时间段等使用索引。

数据类型选择

- 选择适当的数据类型以优化存储空间和查询效率，例如：
  - **日期和时间**：使用`DATE`和`TIME`数据类型存储相关数据。
  - **字符串**：对于固定长度的字符串使用`CHAR`，对于可变长度的字符串使用`VARCHAR`。

2. SQL编写规范

- **SQL关键字**：使用大写字母书写SQL关键字，如`SELECT`，`INSERT`，`UPDATE`，`DELETE`。
- **避免使用SELECT ***：明确指定需要查询的列，提高查询效率。
- **参数化查询**：为防止SQL注入，使用参数化查询。

3. 数据库连接与安全性

数据库连接

- **连接池管理**：使用数据库连接池技术管理数据库连接，提高资源利用率和应用性能。
- **配置外部化**：数据库连接字符串、用户名和密码等敏感信息应配置在外部文件中。

安全性考虑

- **权限控制**：为应用配置最小必要的数据库权限，避免使用root账户。
- **数据备份**：定期备份数据库，确保数据的安全性。

4. 维护与优化

日志记录

- **SQL日志**：记录关键操作的SQL日志，便于问题的追踪和调试。
- **性能监控**：定期检查查询性能，及时优化长时间运行的查询。

定期审查

- **代码审查**：定期进行代码审查，确保遵循本规范。
- **数据库结构审查**：定期审查数据库结构，根据业务发展调整表结构和索引。