ToDo List 项目文档

负责人:刘佳琦

1.技术选型: Kotlin语言

是安卓官方推荐的语言 并且更加熟悉

框架:Jetpack compose + MVVM

mvvm的架构中view只负责展示，而不包含逻辑

viewmodel负责状态管理

这样在更改ui时候不用更改业务逻辑 也更加容易扩展 利于维护

数据库:Room

数据持久化本地化 结构清晰简单

2.

本项目采用 MVVM架构，并结合 Jetpack Compose、Room 数据库以及 Kotlin Flow 实现响应式数据驱动 UI

```
com.fit2081.todoapp
│
├── data
│   ├── local
│   │   ├── Todo.kt
│   │   ├── TodoDao.kt
│   │   └── TodoDatabase.kt
│   │
│   └── repository
│       └── TodoRepository.kt
│
├── ui
│   └── theme
│       ├── TodoScreen.kt
│       ├── viewModel
│       │   └── TodoViewModel.kt
│       ├── Color.kt
│       ├── Theme.kt
│       └── Type.kt
│
└── MainActivity.kt
```

本项目在结构设计上严格按照功能职责进行模块划分，以确保系统具有良好的可维护性与扩展性。

data.local 模块负责应用的本地数据管理与持久化操作。其中，Todo.kt 定义了 Todo 任务的数据结构，并作为 Room 数据库中的实体（Entity），用于描述任务的标题、所属分类以及完成状态等关键信息。TodoDao.kt 作为数据访问对象（DAO），集中定义了对 Todo 数据的增、删、改、查操作，屏蔽了底层 SQL 细节。TodoDatabase.kt 则用于配置和创建 Room 数据库实例，统一管理数据库的生命周期。该模块仅关注数据的存储与读取，不涉及任何业务逻辑或界面逻辑。

data.repository 模块用于封装数据访问逻辑，是 ViewModel 与本地数据库之间的中间层。TodoRepository.kt 对外提供统一的数据操作接口，将具体的数据库实现细节隐藏在内部，从而降低上层模块对数据来源的依赖。这种设计使得 ViewModel 不需要直接与 Room 打交道，增强了系统的解耦性，同时也为未来引入网络数据源或多数据源融合提供了良好的扩展空间。

ui.theme.viewModel 模块负责业务逻辑处理与界面状态管理。TodoViewModel.kt 作为 ViewModel 层的核心组件，负责接收 UI 层的用户操作请求，并通过 Repository 完成数据的增删改查。同时，ViewModel 使用 Kotlin Flow 和 StateFlow 对 Todo 列表与搜索关键字进行状态管理，使 UI 能够在数据变化时自动更新。通过将业务逻辑集中在 ViewModel 中，项目有效避免了界面层逻辑膨胀的问题。

ui.theme 模块主要负责界面展示与用户交互。TodoScreen.kt 使用 Jetpack Compose 构建应用界面，展示 Todo 列表、搜索输入框、任务分类以及删除等功能。UI 层仅通过观察 ViewModel 暴露的状态进行渲染，不直接访问数据库或处理业务逻辑，从而保证了界面的简洁性与可维护性。

通过上述模块划分，项目实现了清晰的分层结构，各模块之间职责明确、依赖关系单向，有效提升了整体代码质量和系统的可扩展性。

3
1.描述不是必须填写，todo的title为必填项
当标题为空时，禁止创建待办事项，UI 层会阻止提交操作
当描述为空时，允许创建待办事项，在数据层中使用空指针表示该字段缺失
2.已完成的选项我们可以通过点击前面的按钮 表示已完成，最后的ui显示会变成删除线的箱子
3.默认按照时间顺序来排序
4.作为进阶扩展功能，本项目实现了任务搜索功能。用户可以通过输入关键词，对任务标题和分类进行实时过滤。
设计思路是：在viewmodel中维护一个搜索关键字的状态，讲原始任务列表和搜索关键子进行组合，当搜索为空的时候显示所有，当搜索不为空的时候，仅显示标题或者分类包含关键子的部分

4.使用了chatGPT帮我生成了初步的文档以及Screen部分的ui的代码

Ai建议我加入Inmemory repo但是我选择直接room + SQLite的做法这样更直接也更符合真是的设计思路

5.运行：使用Android Studio 直接运行项目代码

本项目已在以下环境中进行测试：

- 操作系统：macOS
- 开发工具：Android Studio
- Android 系统版本：Android 13 / Android 14（模拟器）
- 编程语言：Kotlin
- 架构模式：MVVM（Model–View–ViewModel）

目前主要通过人工测试验证功能正确性，尚未编写完整的单元测试或 UI 自动化测试。

6.

如果有更多时间对本项目进行完善，主要会从以下几个方面进行改进：

首先，在数据层方面，可以实现完整的数据库迁移（Migration）机制，替代当前开发阶段使用的破坏式迁移方式，从而在应用版本升级时更好地保护用户数据。这将使应用更接近真实生产环境的要求。

其次，在功能层面，可以引入任务优先级、截止时间和提醒通知功能。通过为任务设置优先级或到期时间，并结合系统通知机制，进一步提升待办事项应用的实用性。

此外，在测试方面，可以补充单元测试和 UI 自动化测试，对 ViewModel 和 Repository 层进行更系统的验证，从而提高代码的稳定性和可维护性。

本次实现的最大亮点在于整体架构设计的清晰性和可扩展性。

项目采用 MVVM 架构，将 UI、业务逻辑和数据访问进行清晰分离。UI 层仅负责状态展示与用户交互，ViewModel 负责状态管理与业务逻辑，而数据访问则统一通过 Repository 与 Room 数据库完成。这种设计使代码结构清晰，降低了模块之间的耦合度。

同时，应用基于响应式数据流（Flow）实现 UI 自动更新，避免了手动刷新界面的复杂逻辑。当数据发生变化时，界面能够自动同步更新，提升了代码的可读性与可维护性。

总体而言，该实现不仅满足了作业的功能要求，也体现了对真实 Android 应用开发模式的理解，为后续功能扩展和维护奠定了良好基础。