课表小助手

软件架构文档

版本 0.2

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 2024.5.14 | 0.2 | 技术原型软件架构 | 张倍宜 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

1. 简介 4

1.1 目的 4

1.2 参考资料 4

2. 用例视图 4

3. 逻辑视图 4

3.1 概述 4

3.2 在构架方面具有重要意义的设计包 4

4. 进程视图 4

5. 部署视图 4

6. 实现视图 5

7. 技术视图 5

8. 数据视图（可选） 5

9. 核心算法设计（可选） 5

10. 质量属性的设计 5

软件架构文档

# 简介

## 目的

本文档将从构架方面对系统进行综合概述，其中会使用多种不同的构架视图来描述系统的各个方面。它用于记录并表述已对系统的构架方面作出的重要决策。

此文档在项目文档体系中扮演着至关重要的角色。它不仅是一份静态的记录，更是一个动态的指南，指导团队成员在系统开发过程中做出与架构一致的决策。此外，本文档还旨在为项目的非技术利益相关者提供足够的架构信息，使他们能够理解系统的整体结构和设计。

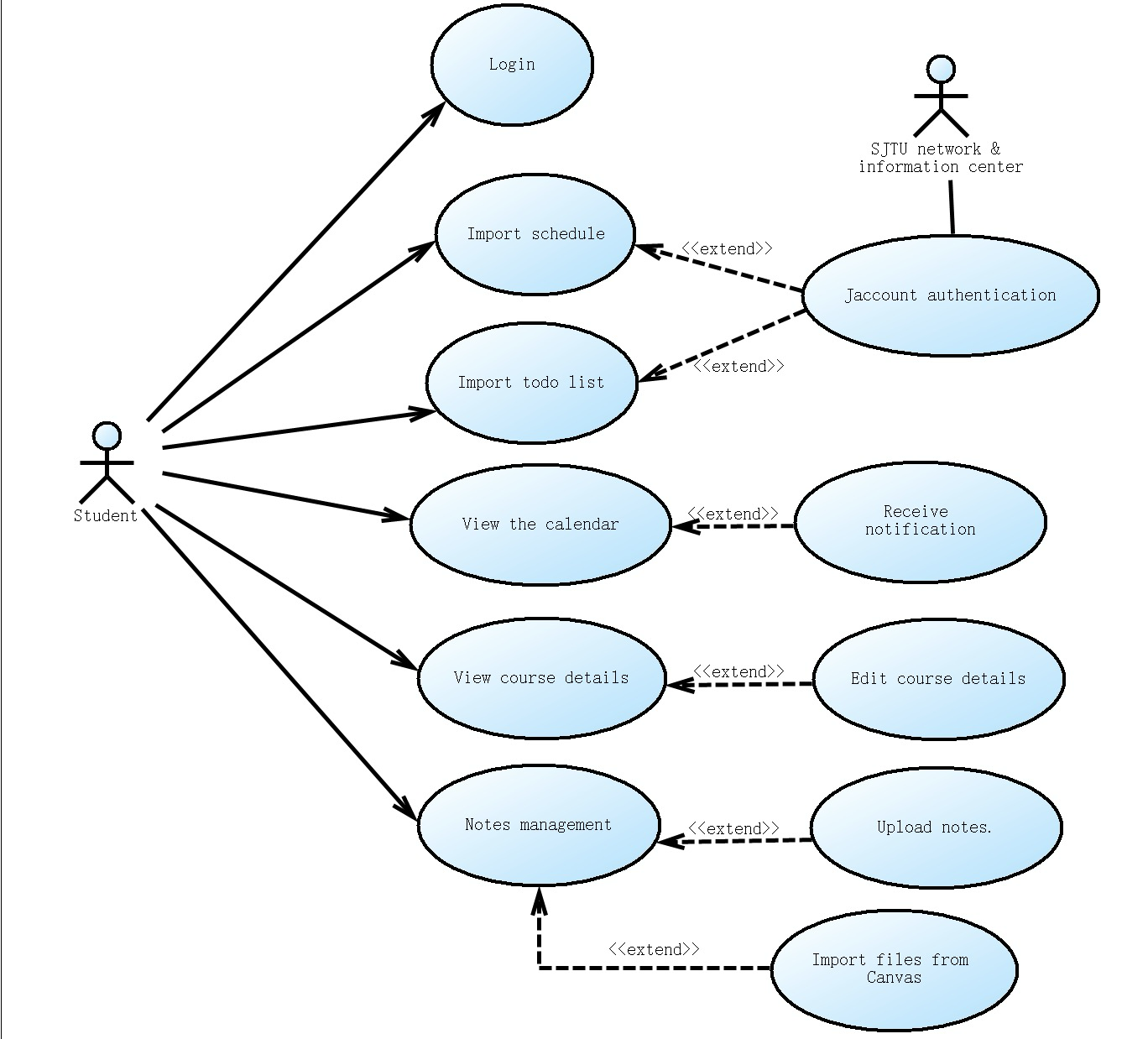
本文档的特定读者包括项目团队成员以及任何对系统架构有深入了解需求的其他利益相关者。建议读者在开始阅读本文档之前，先熟悉项目的Vision文档和设计规范，以便更好地理解本文档中描述的架构决策。

本文档的结构分为多个部分，每个部分都针对系统架构的不同方面进行阐述。读者应该按照文档的顺序阅读，以便逐步构建对系统架构的全面理解。在阅读过程中，读者应该关注每个架构视图如何与整体架构设计相契合，并理解每个决策背后的设计意图和考虑因素。

总之，本文档的目的是确保项目的架构设计得到清晰的记录和传达，为项目的成功实施提供一个坚实的基础。通过提供对系统架构的全面概述，本文档旨在帮助读者理解系统的设计选择，并指导他们在项目生命周期中做出与架构一致的决策。

## 参考资料

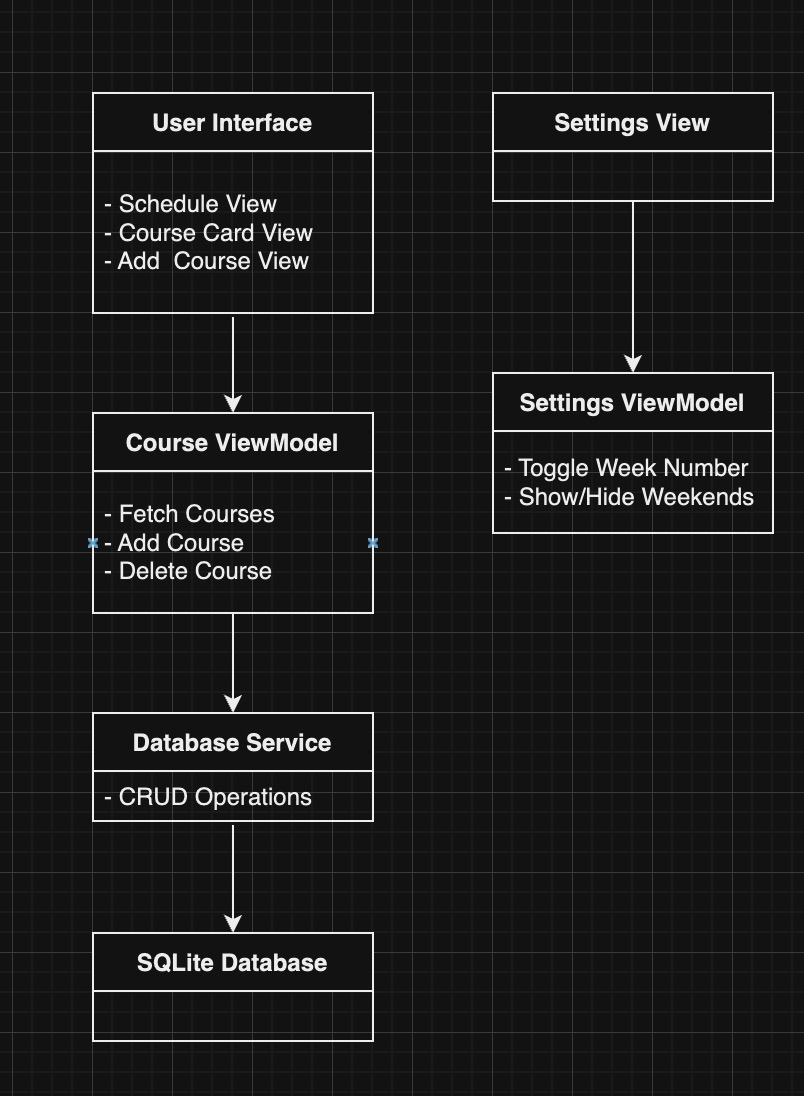
# 用例视图



# 逻辑视图

## 概述

本课表软件采用了Model-View-ViewModel (MVVM) 架构模式。这种架构模式将应用程序的逻辑层、视图层和数据层进行了明确的分离，提高了代码的可维护性和可测试性。



上述架构图展示了课表软件的逻辑架构。在此架构中，用户与视图层交互，视图层通过ViewModel处理用户请求，ViewModel负责业务逻辑的处理并与数据层交互。数据层通过Database Service与SQLite数据库进行数据的存取操作。这样的分层架构提高了应用程序的模块化程度，增强了代码的可维护性和可测试性。

## 在构架方面具有重要意义的设计包

### **Model（数据层）**

Course Model：代表课程的基本数据结构，包括课程名称、上课时间、老师、上课地点、备注等信息。

Database Service：负责与SQLite数据库交互，执行数据的增删查改操作。

### **View（视图层）**

Schedule View：显示当前周的课程表。用户可以在此视图中查看课程信息，并点击查看课程详细信息。

Course Card View：当用户点击课程方块时，弹出显示详细信息的课程卡片视图。用户可以在此视图中删除课程。

Add Course View：提供用户添加新课程的界面，用户可以在此视图中填写课程相关信息。

Settings View：用户可以在此视图中进行应用程序的设置，例如切换周数和是否展示周末课程。

### **ViewModel（逻辑层）**

Course ViewModel：处理与课程相关的业务逻辑，包括从数据库中读取课程数据、添加课程、删除课程以及更新课程信息。

Settings ViewModel：处理应用程序设置相关的逻辑，包括切换周数和是否展示周末课程的逻辑。

### **模块协同**

用户打开应用程序，Course ViewModel 从数据库中读取当前周的课程数据，通过数据绑定的方式将数据传递给 Schedule View，显示在界面上。

用户点击课程方块，Course ViewModel 获取点击的课程信息并传递给 Course Card View，展示详细信息。用户可以在课程卡片中选择删除课程，Course ViewModel 将请求传递给 Database Service，然后更新界面显示。

用户在 Add Course View 中添加课程时，填写课程信息并提交，Course ViewModel 将信息传递给 Database Service，插入数据库后更新课程表视图。

用户在 Settings View 中切换周数或设置是否显示周末课程，Settings ViewModel 将更新的设置信息传递给 Course ViewModel，后者重新从数据库中读取相应的课程数据并更新课程表视图。

# 进程视图

## 系统分解

本课表软件运行在单个Flutter应用程序内，不涉及重量级进程（成组的轻量级进程）的划分，主要依赖于Flutter的单线程事件循环机制。因此，本节将主要讨论在Flutter应用程序内的轻量级进程（单个控制线程）和组件之间的通信和交互。

### **轻量级进程**

在Flutter应用程序中，主要的轻量级进程包括：

1. UI 线程（主线程）：

说明：处理所有的UI渲染和用户交互。

职责：

渲染界面

处理用户的输入事件（点击、滑动等）

触发状态更新和重绘界面

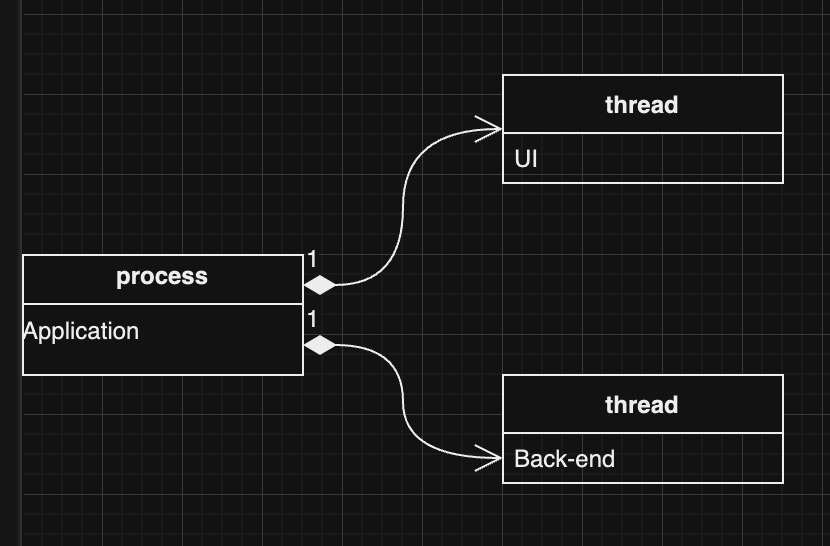
1. 后台线程：

说明：使用Future和async/await进行异步操作，如数据库访问、网络请求等。

职责：

执行耗时操作（如从SQLite数据库读取数据）

确保UI线程保持流畅，不被阻塞



### **进程间的通信模式**

在Flutter应用程序中，主要的通信模式包括消息传递和异步操作。

1. 消息传递（Provider）：

说明：Flutter的Provider包用于在不同的Widget树之间共享状态和数据。

通信模式：当ViewModel中的数据发生变化时，通过Provider通知相应的视图（UI组件）更新。

视图与ViewModel之间通过Provider进行双向绑定，实现状态的自动同步。

1. 异步操作（async/await）：

说明：Flutter中广泛使用async和await关键字来进行异步编程，确保耗时操作不会阻塞UI线程。

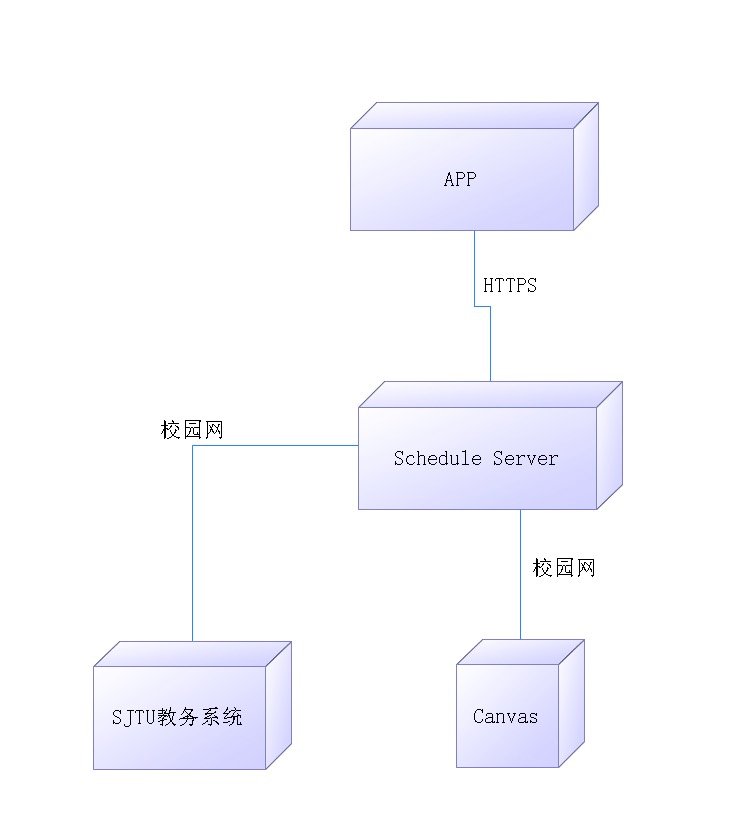
通信模式：后台线程执行异步操作（如数据库查询），完成后通过回调或Future通知UI线程更新视图。

# 部署视图

用户使用课表APP，通过互联网连接到服务端。

服务端是一个云服务端，用于处理和响应用户的所有请求。

通过jAccount系统，允许用户使用jAccount账号登陆，并可以直接从SJTU教务网站导入用户的课程安排信息，从Canvas平台导入待办事项。



# 实现视图

用户界面层（UI Layer）：负责展示用户界面，接收用户输入，向用户展示信息。

应用逻辑层（Application Logic Layer）：包括ViewModel层和部分业务逻辑，负责处理用户输入、业务逻辑处理和状态管理。

数据访问层（Data Access Layer）：负责与SQLite数据库进行交互，执行数据的增删查改操作。

# 技术视图

编程语言：Dart

开发工具：Android Studio，VScode

框架：Flutter

数据库：SQLite

# 数据视图（可选）

# 核心算法设计（可选）

# 质量属性的设计

## 性能

在课表软件中，合理设计的数据访问层和ViewModel层可以减少数据库访问次数和数据传输量，从而提高应用程序的响应速度和性能表现。

## 可扩展性

课表软件通过采用MVVM架构，将UI层与业务逻辑分离，使得在不影响界面显示的情况下可以轻松修改业务逻辑或增加新功能，提高了系统的可扩展性。

## 可靠性：

课表软件通过在ViewModel层进行状态管理和错误处理，确保了系统在面对异常情况时的稳定性，提高了系统的可靠性。

## 易用性

课表软件通过设计直观的用户界面和清晰的交互逻辑，使得用户可以轻松地添加、编辑和删除课程，提高了系统的易用性。

## 可移植性：

课表软件通过采用Flutter作为跨平台开发框架，使得应用程序可以在不同的移动设备上运行，提高了系统的可移植性。