Shattuckite

项目计划书

SHADOC-001,SDP, 第五组

版本 *0.0-3-gbde8e97*

表 1: **分工说明**

小组名称	lemon	
学号	姓名	本文档中主要承担的工作内容
16231275	刘瀚骋	内容编写/内容审核/绘图/自动化工具编写

表 2: **版本变更历史**

版本	提交日期	主要编制人	审核人	版本说明
v0.0-3-gbde8e97	2019-03-18 23:59	CNLHC <2463765697@qq.com>	CNLHC	添加表格题注与标题; 补充项目概述及文档概述。
v0.0-2-gc896e12	2019-03-09 03:15	LiuHanCheng <2463765697@qq.com>	CNLHC	添加自动化任务列表生成工具。
v0.0-1-gac343a2	2019-03-09 01:32	LiuHanCheng <2463765697@qq.com>	CNLHC	完善风险控制一节并修复自动部署 Bug
v0.0	2019-03-08 21:12	CNLHC <2463765697@qq.com>	CNLHC	根据给定模板重新组织文档内容。发布 v0.0 版本。

Contents

1 范围

1.1 项目概述

`shattuckite` 项目首先作为 2019 年《软件工程》课程的课程设计, 用于帮助开发团队获取这门课程的学分。

本项目旨在面向家庭及小型民用建筑物, 提供一个可以通过手机及计算机远程访问环境数据及控制机电设备的计算机系统.

终端用户可以使用常见的通信设备, 系统的功能需求包括:

1. 实时访问家中传感器的数据
2. 远程控制可被控制的设备
3. 配置系统自动响应某些特定事件

系统的非功能需求包括:

1. 提供用户友好的 GUI 界面
2. 保证系统运行的稳定性
3. 保证系统的可扩展性

关于更加详细的使用场景描述, **请参考**《*shattuckite 需求分析文档*》

1.2 文档概述

本文档用于阐述项目的开发计划。主要内容包括项目背景概述，开发团队的组织管理方式，开发流程控制，计划交付内容及交付时间节点及风险控制等元信息。

1.3 术语及缩略词表

下表列出在本文档中可能出现的缩写及其全称。

表 3: 缩写及相应全称

缩写	全称
Lora	Long Range
IOT	Internet Of Things
NB-IOT	Narror-Band IOT
REST	Representational State Transfer
RPC	Remote Procedure Call

1.4 引用文档

[1]《shattuckite 需求分析文档》

2 项目任务概要

2.1 工作内容

具体的项目需求 **请参见**《*shattuckite 需求分析文档*》

开发人员角色分配及角色职责参见 节 2.2.1 成员组织。

主要工作划分参见 节 6 进度计划。

2.2 开发人员

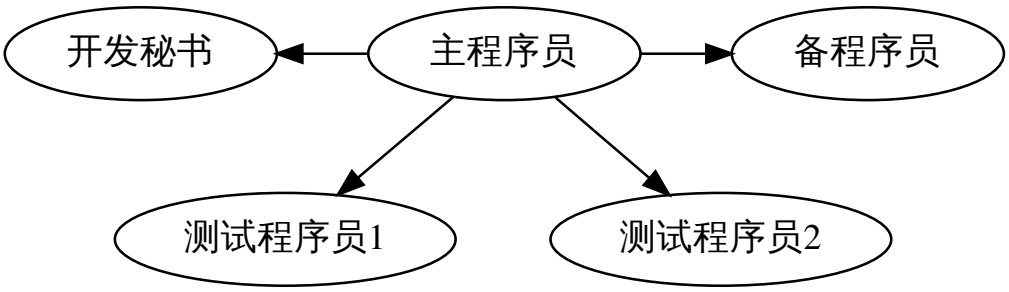
本项目由一个规模为 4 人的小型团队进行开发. 团队成员情况如下

表 4: 开发团队人员组织

姓名	学号	学院	工作经验	工作时间	技术水平
刘瀚骋	16231275	高等理工学院	无经验	3 年	低级
孟巧岚	16061053	计算机学院	无经验	3 年	低级
邓健	16061136	计算机学院	无经验	3 年	低级
许文广	16061069	计算机学院	无经验	3 年	低级
张起铭	16061044	计算机学院	无经验	3 年	低级

2.2.1 成员组织

本次开发计划使用类似主程序员小组的方式进行小组组织。组织结构如下图所示。



组织方式示意图

各组员小组角色分配如下

表 5: 小组成员

姓名	学号	学院	小组角色
刘瀚骋	16231275	高等理工学院	主程序员
孟巧岚	16061053	计算机学院	备程序员
邓健	16061136	计算机学院	测试程序员
张起铭	16061044	计算机学院	测试程序员
许文广	16061069	计算机学院	开发秘书

2.2.2 角色职责分配

本节叙述不同小组角色的职责

主程序员

- 管理项目开发进度。
- 设计项目开发路线与体系结构。
- 具体实施软件开发及硬件设计工作。
- 分割开发任务, 与备程序员协同完成开发任务。
- 与测试程序员交流, 完成软件测试工作。
- 与开发秘书共同完成文档的撰写。

备程序员

- 与主程序员交流，具体实施软件开发任务。
- 与开发秘书共同完成文档的撰写。

测试程序员

- 与主程序员交流，设计各个构件的测试方法，并编写《测试文档》。
- 编写测试代码，完成软件测试。
- 软件可持续集成与可持续交付的相关工作。

开发秘书

- 与主程序员交流，完成《需求分析文档》及《设计文档》的撰写
- 管理并发布软件开发过程中产生的各种文档
- 负责与甲方交流，确保开发团队及甲方始终保持共识。

2.3 产品

本项目结束时计划交付的内容包括

表 6: 计划交付内容

项目名称	类型	负责人
《项目开发计划书》	文档	主程序员
《需求分析》	文档	主程序员，备程序员，开发秘书
《系统设计说明》	文档	主程序员，备程序员，开发秘书
《测试文档》	文档	测试程序员，开发秘书
《系统部署指南》	文档	主程序员，开发秘书
测试程序	源代码	测试程序员
系统程序	源代码	主程序员，备程序员

2.3.1 程序和和设备

测试程序

本项目计划使用黑盒测试的方法，编写单元测试。

系统程序

系统程序将以构件为单位进行发布。

对于运行在嵌入式平台 (非 x86_64 架构 CPU) 上的程序，计划直接发布交叉编译后的二进制文件及安装脚本。

对于运行在通用平台 (x86_64 架构 CPU) 上的程序, 计划以 docker 容器和源代码 + 构建脚本两种形式进行发布。

2.3.2 文档

《项目开发计划书》

本文档用于阐述项目的开发计划。主要内容包括项目背景介绍，开发团队的组织管理方式，开发流程控制，计划交付内容及交付时间节点及风险控制等元信息。

《需求分析》

基于场景和数据两方面, 尽可能全面的分析本项目的需求。

基于场景的需求分析将从用户的视角出发, 讨论用户会以何种方式与本系统发生交互。使用自然语言和 UML 活动图, 对所有可能发生的交互场景进行建模。

基于数据的需求分析将结合基于场景的分析结果, 讨论为满足用户使用场景, 系统需要维护的数据和系统需要暴露的控制接口。并使用 DFD(Data flow Diagram) 分别对

1. 数据在系统中的传递与持久化
2. 控制信号的传递

进行建模

《系统设计说明》

系统设计说明将自顶向下的描述系统的具体实现方式。自顶向下的顺序分别是设备级, 构件级, 类/方法级, 实现级四级。

设备定义为物理上独立的实体。设备级设计将规划系统运行所需要的设备以及设备间互联的基本方案。计划使用 UML 部署图来建模这一级的设计。

构件是运行在设备上, 且逻辑功能较为独立的软件。一个设备上可能会运行多个构件。构件级设计将规划每个构件实现的具体功能，使用的具体技术栈以及构件间的互联方式。计划使用 UML 部署图来建模这一级的设计。

类/方法是组织逻辑的最小单元。若干个类/方法相互协作，组合成构件。计划使用 ER 图，UML 类图来建模这一级的设计

实现指类/方法的具体代码。实现级文档不单独撰写，将会根据编码过程中的注释自动生成。

《测试文档》

测试文档计划阐述本项目的测试策略以及运行测试程序的方法。

2.4 运行与开发环境

2.4.1 运行环境

嵌入式终端

表 7: 嵌入式终端运行环境

项目	约束
操作系统	Linux 内核版本 > 3.4.1
嵌入式 CPU 架构	Cortex-A8 或 Cortex-A9
嵌入式 CPU 主频	主频 >= 800Mhz
内存	容量 >= 500MBytes
接口	USB2.0/RJ45/RS232(可选)/RS485(可选)

云服务器

表 8: 云服务器规格要求

项目	约束
操作系统	Ubuntu>=16.04/Debian 9
内存	容量 >=1GBytes
网络	带宽 >=2Mbps, 公网 IPV4 地址
CPU 架构	x86_64
CPU 核心数	核心数 >=1 core

移动端 APP

表 9: 移动端 App 运行环境

项目	约束
操作系统	Android >=7.0 IOS>=11.0
Android	API Level >= 26

桌面端 APP 及 Web 端接口

表 10: 桌面端 APP 及 Web 端接口

项目	约束
操作系统	Windows 7/10/Linux (With GNOME or KDE)
浏览器	Chrome 版本 >60.0 Firefox 版本 >49.0

2.4.2 开发环境

表 11: 开发环境

项目	名称	提供方
代码托管平台	Github	免费服务
可持续集成	私人服务器, 使用 Jenkins 服务	自行准备
代码编写/交叉编译	私人 PC	自行准备
Cortex-A8 代码调试	飞凌嵌入式 AM3354 开发板	自行准备
Cortex-A9 代码调试	Terasic Cyclone SOC 开发板	自行准备
Wifi 传感器节点	乐鑫 ESP8266	自行准备
NB-IOT 传感器节点	移远 BC-26 NB-IOT 模组	TBD
Lora 传感器网关	Semetech SX1301 评估版	学校提供
Lora 传感器节点	SX1278 / SX1276	学校提供

2.5 项目期限

参见甲方交付时间节点要求

本项目开发活动起始于 2019 年 3 月 4 日, 结束于 2019 年 6 月 14 日

3 风险控制

3.1 风险分析

本项目面临的主要风险是系统无法按期交付导致无法获取成绩。

项目无法按期交付的可能原因包括

1. 开发过程中遇到无法解决的技术难题, 导致项目难于继续进行。
2. 自动化工具的建立不够完善，导致在版本管理/部署/发布等非开发环节消耗过多的时间。
3. 设计规模超出控制，引入不必要的工作量。
4. 某些成员经验不足，导致项目进度缓慢。

3.2 风险规避

为了规避风险，本项目计划尽早发布第一个原型版本。

针对上述列出的几点导致项目延期交付的原因，项目组计划采用以下方案，尽可能规避风险:

1. 在设计阶段采用自顶向下的设计与严格的构件化设计，并控制构件规模。确保某一构件设计无法实现时, 能够快速进行重新设计。
2. 指派专人进行自动化工具的搭建。(本开发组中，将由测试程序员承担这项工作。参见成员组织)
3. 严格践行原型模式，按照计划按时发布每一个小版本。避免好高骛远，一次引入过多未实现功能。

4. 执行 KPI 制度, 根据工作量分配浮动分数, 激励开发小组内每位成员努力学习新技术栈。

4 过程模型

4.1 模型

本项目要求一定要在规定的日期前交付至少一个版本, 因此决定采用类似快速原型模型的流程进行开发。

本项目将开发过程中的一次迭代划分成四个环节:

1. 讨论调研
2. 需求分析建模与软件设计
3. 编码与测试
4. 发布

4.1.1 讨论调研

讨论调研时期，开发团队以访谈调查问卷等方式，与甲方以及其他项目面向的用户进行交流，获取用户的最新需求，并确保用户的需求是可行的。

4.1.2 需求分析建模与软件设计

对讨论调研得到的信息进行抽象, 整理; 并撰写或修改《需求分析》文档。

在需求分析建模结束后，根据《需求分析》文档的修改情况，更新《系统说明文档》的内容，对系统实现的设计进行修订。

4.1.3 编码与测试

根据最新版本的《系统说明文档》, 对系统构件进行增删或修改；与此同时，更新相应的测试用例。

4.1.4 发布

完成编码与测试后，根据修改情况，进行版本号控制 (详见版本号控制)。并编写相应的脚本，进行数据库，配置文件等的迁移，将新系统部署到生产服务器。

4.2 模型实施

在开发初期我们将按照: 讨论调研-需求分析建模-编码与测试-发布的流程完成第一次迭代。

在第一次迭代完成后, 讨论调研-需求分析建模与编码测试-发布这两个过程将独立异步进行。

为了保证时间节点与甲方的要求相同, 我们计划在 2019 年 4 月 4 日前完成首次迭代过程, 发布原型版本 (系统版本号 `0.1.0`)。

关于模型实施的更多细节，请参照进度计划

5 资源计划

运行环境 及开发环境 两节中的内容已经足以充分说明项目开发的软硬件资源需求, 在此不再赘述。

6 进度计划

6.1 里程碑计划

6.2 甲方交付时间节点要求

本项目甲方要求按照下表列出的时间节点提交相应文件

表 12: 甲方交付时间要求

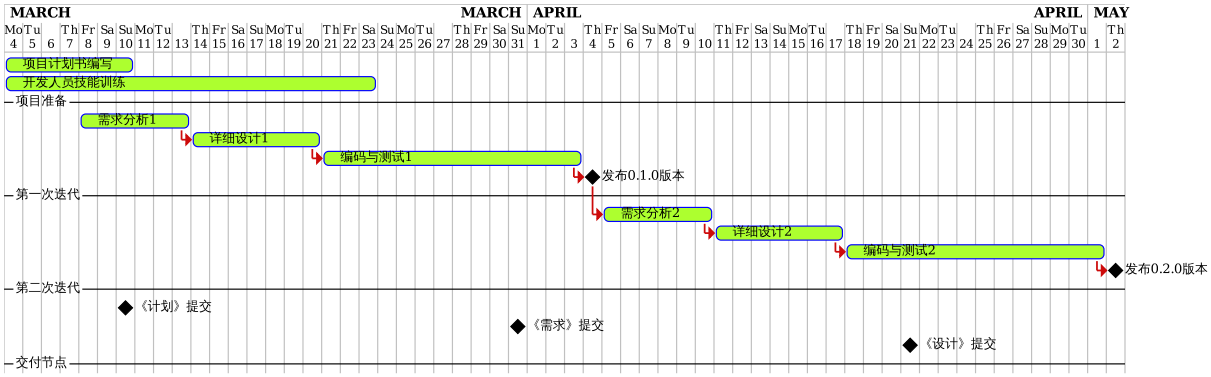
工作名称	DDL
自由组队	2019 年 3 月 4 日
《项目开发计划书》编写	2019 年 3 月 10 日
《需求分析文档》提交	2019 年 3 月 31 日
需求分析评审	2019 年 4 月 2 日
设计文档提交	2019 年 4 月 21 日
设计文档评审	2019 年 4 月 23
代码评审	2019 年 5 月 14 日
测试文档评审 1	2019 年 6 月 4 日
测试文档评审 2	2019 年 6 月 11 日
课程总结	2019 年 6 月 18 日

6.3 团队里程碑设置

在 2019 年 5 月中旬前, 开发团队计划实现两个里程碑

1. 于 2018 年 4 月 3 日发布 *0.1.0* 版本
2. 于 2018 年 5 月 2 日发布 *0.2.0* 版本

6.4 里程碑任务映射



6.5 任务详情分配

本项目使用 `Github` 的 `issue` 功能进行任务发布。本节的内容为脚本自动生成。

6.5.1 《测试文档》编写计划

成员:Dicky35,baixusata

KPI: 200

DDL: 2019/3/30 24:00

设计测试方案并考虑《测试文档》的编写。在本次任务结束前，请提交《测试文档》的编写大纲。

请 @Dicky35 与 @baixusata 共同完成本任务。请二位自行商议各自负责的部分。

注: 截止 2019/03/22 本项目后端可能使用的技术栈包括

1. Scala (`Akka` + `Kafka`+ `gRPC`)：嵌入式终端与服务器数据交互
2. C/C++ (Linux system call)：嵌入式终端设备驱动
3. Python (`Django`+`gRPC`)：数据库管理与提供 Web 接口

前端使用的主要技术是 `React.js`

其中 `Scala` 和 `Python` 自带单元测试框架，`React.js` 可以使用 `jest` 测试框架，`C/C++` 可以使用 `gTest` 测试框架。

此外，移动端可以使用 `ADB` 进行自动化测试，Web 端可以使用 `Selenium` 进行自动化测试。

6.5.2 前端开发-移动端 UI 原型设计

成员:mqlKKK

KPI: 200

DDL: 2019/3/30 24:00

根据需求文档的描述，进行移动端用户界面的原型。设计 UI 时请注意:

1. 使用 Axure 原型工具进行绘制
2. 最终产品需要严格按照本版原型进行开发，在设计时请考虑可实现性。
3. 尽可能 cover 用户可见的每一个页面
4. 尽量保证设计的美观
5. 在 buaase 下新建仓库 shattuckite-gui-mobile, 并将设计稿提交到该仓库。

此外，如果时间允许的话，请为 UI 原型添加简单的交互逻辑。

注: Antd library 提供 ant-design 及 ant-design-mobile 组件的 Axure 模型。

6.5.3 非 x86 平台测试环境搭建

成员:Dicky35

请使用 qemu 仿真非 x86 体系结构的 cpu，搭建测试环境。

具体要求

继续修改 #7 中的 Dockerfile，在其内部配置 qemu 环境

建立首个 qemu 虚拟机，需要满足以下几点要求

1. 基于 Cortex-A9 架构
2. 指令长度为 64 位
3. 安装 Debian 系统

注意建立 Qemu 过程中的一切静态文件（包括系统配置文件及虚拟磁盘文件）应当保存在宿主机而非 Docker 中，Docker 容器应该通过 VOLUME 挂载的方式访问这些静态文件。

该任务计 200KPI

该任务应当在 2019 年 3 月 22 日 24:00 前完成

6.5.4 《需求文档》绘制用户用例活动图

成员:wenguang1998

请为 shattuckite-doc /requirements/source/userOriented/cont.json目录下的每个 case 绘制 UML 活动图

基本要求

1. 使用 plantuml 进行绘制
2. 在/source/userOriented 目录下新建一个 uml 文件夹，并将绘制 (编写) 的 UML 图置于此处
3. 每一个 case 对应一个 uml 文件, 文件名应该是 case 的 name 字段, 文件扩展名是 plantuml。该文件应当置于 uml 文件夹内。
4. 请注意 uml 活动图的规范, 建议阅读Activity Diagram

该任务计 200KPI

该任务应当在 2019 年 3 月 22 日 24:00 前完成

注: 至少完成 #9 提交的 case.json 版本中用例的建模。

6.5.5 《需求文档》继续细化面向用户的需求建模

成员:mqlKKK

文档仓库shattuckite-doc的 PRD-dev 分支内正在进行需求文档的编写。

请参考/requirements/source/userOriented/intro.rst 文件内的概述信息，并自行扮演用户的角色，完成面向用户的需求建模。

1. 查看监控数据用例拆分

该用例至少可进一步拆分为 查看实时监控数据及 查看历史数据。

1. 管理传感器用例重新设计

管理传感器实际应为管理 设备。设备包括传感器，执行器，摄像头等本系统可能会用到的物理设备。

管理设备请细分为批量设备管理，传感器管理 执行器管理等

1. 添加操作执行器的用例

用户可能会直接操作隶属于执行器类的设备。**操作**与 **管理**属于不同的逻辑，应当为操作执行器单独设计用例。

1. 关于 prerequisites 字段

该字段主要体现用例的相互依赖关系。例如在 A 用例的 prerequisites 字段中列出 B 用例，则说明 A 用例一定要在用户执行了 B 用例后才会执行。目前部分用例该字段有些许不妥，请修改。

1. 关于 case.json 文件编辑

9 中提交的 case.json 版本存在些许语法错误。建议使用带有 json lint 功能的编辑器对本文件进行编辑。

该任务计 100KPI

该任务应当在 2019 年 3 月 20 日 24:00 前完成。

6.5.6 《需求文档》面向用户的需求建模编写

成员:mqlKKK

文档仓库shattuckite-doc的 PRD-dev 分支内正在进行需求文档的编写。

请参考/requirements/source/userOriented/intro.rst 文件内的概述信息，并自行扮演用户的角色，完成面向用户的需求建模。

在编写用例时，应当编辑/requirements/source/userOriented/case.json 文件。该文件内已经设计了两个用例供参考。

请尽可能多的提出需求并以标准化用例的形式记录。

该任务计 100KPI

该任务应当在 2019 年 3 月 17 日 24:00 前完成。

6.5.7 《需求文档》编译配置

成员:wenguang1998

修改shattuckite-doc仓库 requirements 分支,requirements/source 目录下的 conf.py 文件, 实现

1. (100kpi) 输出 pdf 的 title page 和项目计划书的相同。
2. (100kpi) 自动根据符合特定格式的 commit 信息修改文档内的修订历史记录

注: 上述符合格式的 commit 信息, 格式定义为: 使用 git log 输出提交历史记录后, 可被正则表达式匹配并提取分组的格式。

该任务应当在 2019 年 3 月 15 日 24:00 前完成。

6.5.8 arm 交叉编译环境搭建

成员:Dicky35

编写 Dockerfile, 新建一个 docker img。在使用该 img 实例化 container 后, 应当实现

1. 内部含有交叉编译各种 arm 架构二进制程序的编译环境
2. /root 目录下提供一个 shell 脚本 armenv.sh, source 该脚本后, 系统内和编译有关的环境变量会被自动替换为 arm 交叉编译器版本。

在完成 Dockerfile 的编写后, 使用该环境交叉编译 grpc到 Cortex-A9 架构。

该任务应当在 2019 年 3 月 15 日 24:00 前完成。

6.5.9 《需求文档》自动化发布

成员:baixusata

KPI: 200

DDL: 2019/03/15 24:00:00

150kpi 自动发布逻辑编写

修改文档仓库(shattuckite-doc)项目文件夹内的 Jenkinsfile, 在 Jenkins 流程被触发后

1. 自动编译更新后的《需求文档》, 生成 pdf 文件。
2. 将生成的 pdf 重新命名为 需求文档-$version 的格式。$version 为使用 git describe 生成的版本号。
3. 将生成的文档自动 push 到Team105仓库

注意

1. 文档仓库内可能同时存在多个分支。在编译及重命名文档时，可能需要通过检查当前分支的名称，来选择不同的编译/发布逻辑。
2. 没有开放文档仓库的写权限，请自行 fork 一个仓库进行调试。

50kpi Jenkins 项目配置

请配置 Jenkins 项目，使其响应除 master 分支和 *-dev 分支外的所有分支的更新。(可以通过项目配置页面里的正则表达式过滤器实现)

该任务应当在 2019 年 3 月 15 日 24:00 前完成。

6.5.10 React+Redux 前端: UI 框架前端路由与 scss 样式

成员:mqlKKK

(主要是替换按钮/输入框/列表等组件)

注意，可能需要改变 Webpack 配置文件。如果是用 create-react-app 生成的项目需要 eject 操作。

a. 设计一个主界面，主界面上有连接到 TickTocToe 和 TodoList 的按钮
b. 在 TickTocToe 和 TodoList 的页面上分别加上返回主界面的按钮
c. 请使用 Hash-Router

此外，关于一些代码组织上的建议

1. 组件源代码主要存储在 src/container 和 src/presentation 下；Reducer.ts(Root)，index.tsx 等存放在 src/ 下
2. 每个组件拥有自己的独立文件夹
3. 所有组件的主类均在文件夹内的 index.tsx 文件实现与导出
4. 对于容器组件，文件夹内，除了 index.tsx 外，通常情况下还会拥有 action.ts 和 reducers.ts
5. 如果需要添加样式的话，可以在组件所属的文件夹内添加 index.scss

6.5.11 React+Redux 前端 - TikTacToe 编写

成员:mqlKKK

TikTakToe 是 React 官方文档中的例子请参照官方文档中的实现，使用 Typescript 和 Webpack 完成该示例程序。

6.5.12 测试与持续集成-基础

成员:Dicky35,baixusata

https://github.com/buaaembeddedse/group-training/commit/2a72c991a8e5f940e23a6acd8766e87bb6f99368 上传了两个文件，分别用 c++ 和 python 实现了最简单的 a+b 功能，需要分别使用 `gTest` 和 `unittest` 为这两个函数添加单元测试。

该任务计 100KPI

应当在 `2019 年 3 月 8 日 24:00` 前完成本任务

6.5.13 文档撰写和 UML 建模 -PlantUML 的使用

成员:wenguang1998

https://github.com/buaaembeddedse/group-training/commit/6783b777dfd1fcc55897a52e3b3fcf21065b8d9c 上传了一个简单的基于场景的需求用例。

1. 请使用 UML 活动图对该用例进行建模
2. 使用 reStructuredText 语法将上述活动图添加到文档中

请注意

1. 不要使用 GUI 工具进行 UML 图绘制，理由是 GUI 工具绘制的 UML 图一致性差且难于维护。推荐使用 `plantUML` 或 `GraphViz` 进行绘制。
2. rst 文档中图片的引用文法，请确保 github 能够识别并且能在网页预览页面上渲染图片。### React+Redux 前端 - TodoList 编写

成员:mqlKKK

`TodoList` 是 `Redux` 官方教程中的例子。

1. 使用 `create-react-app` 脚手架创建项目
2. 使用 `redux` 进行状态管理
3. 编码时将 `Container Component`, `Presentation Component` 分开处理, 并在注释中标明。
4. 使用 `Redux` 时，注意结合 `TS` 的模板编程技术进行 `Type Hint`。`Action`，`Root Reducers` 都应该是强类型的，而不是 `any`

开发的基本模式是 `Container Component` 使用 `connect` 作为一个高阶组件的输入，并作为其他若干 `Presentation Component` 的父组件。仅有 `Container Component` 会与 `Redux` 维护的状态树产生交互。

该任务计 100KPI

该任务应当在 `2019 年 3 月 8 日 24:00` 前完成

7 其他

7.1 版本号控制

7.1.1 构件版本号控制

我们将基于构件进行版本号控制。即每一个构件均拥有自己独立的版本号。

构件的版本号将按照 Semantic Versioning 2.0 标准进行编制。

在版本号变动时，必须在构件仓库目录添加新版本的发布说明，简要的阐述该版本的修改。

构件版本号的主版本号和系统版本号控制 中的系统版本号的主版本号相同。

7.1.2 系统版本号控制

整个系统拥有一个独立的系统版本号。系统版本号同样使用 Semantic Versioning 2.0 标准进行编制。

在开发过程中，系统版本号的修订号不修改，始终为 0. 完成一个迭代后，在发布阶段递增系统版本号的次版本号。

系统次版本号递增时，需要同时发布当前版本的系统程序包含的构件版本以及概述本版改动。

7.1.3 文档版本号控制

开发过程中产生的文档应当拥有独立版本号。文档版本号使用 `X.Y` 两级格式。X 和 Y 分别是主版本号和次版本号。

文档的次版本号在每次迭代时进行递增。

文档的主版本号和系统及构件的主版本号一致。

对文档的内容进行小范围的修改（例如修改错别字，笔误，优化表达方式等）时，不为文档分配新的版本号，在文件名后加入独立的且不会重复的字符串以示区分。

7.2 团队 KPI 管理

为了合理分配甲方提供的开发报酬 (浮动分数)，开发团队将使用 KPI 衡量个人绩效。具体规则如下:

1. 团队中的每个成员有 1000 分基础 KPI 点数
2. 项目经理（主程序员）进行任务划分，为每位成员安排相应的任务，并根据任务的复杂程度，为任务规定一个分数。
3. 若成员未能在规定时间内完成任务，则扣除任务分数的一半。
4. 若成员在延期 1/3 的原定工作时间后仍未完成任务，扣除全部任务分数。
5. 最后根据每个人的剩余 KPI 比例来分配分数