移动软件开发 课程项目

THelper 软件详细设计说明书

杨怿飞 2015011089

曹衷珩 2017011308

隋唯一 2017011430

版本 1.1

创建日期：2020/04/16

最后修改：2020/04/26

修订记录

版本	修改者	简述	日期
1.0	杨怿飞、曹衷珩、隋唯一	简述、市场需求、用户说明、产品特性及界面初步设计	2020/04/16
1.1	杨怿飞、曹衷珩、隋唯一	增加架构设计、数据类设计、数据库设计、界面设计	2020/04/26

1. 概述

1.1 项目背景

根据选题 3 的项目“清华帮帮忙”的要求，我们从不同角度出发，结合不同用户反映的实际需求，对 THelper 的需求进行详细分析，最终编写成为本需求文档。

本文档主要是为开发 THelper 安卓 APP 编写，力求让需求方、用户以及开发者在阅读本文档之后，对我们所要开发的 THelper 形成全面清晰的认识。本文档从需求出发，阐明产品的定位，理清该产品逻辑，并对产品特性进行详尽的说明，侧重于描述目标用户所需的功能和相应的界面逻辑。

THelper 的最终目标是搭建一个完整的平台，专注于校内生活服务众包，提供一个发布悬赏校内跑腿任务及接收任务、获得悬赏等基本功能，并在此基础上继续迭代，持续优化功能，增加新的特色功能。

THelper 平台将由本小组进行开发，该平台的用户范围是所有有众包需求或愿意代跑腿挣外快的清华在校学生。

1.2 市场需求

清华同学在学校有各类跑腿，二手交易等需求，另一侧可以满足需求的同学难以获得相关信息，没有针对性的平台可以将二者匹配。

1.3 用户说明

本项目只设计一类用户，用户根据需要可完成两种类型的功能

角色	行为	共同功能	不同功能
平台用户	发布任务	注册账号	描述任务、给出悬赏、与接单人员交流、查看/更新任务状态、评价
	执行任务		查看任务、接单、与发布人员交流、查看/更新任务状态、查看评价

2. 产品特性

2.1 用户需求点

2.1.1 初次使用

输入清华邮箱、密码完成注册

邮箱验证？

完善用户信息，设置头像、昵称，完善宿舍楼、院系等信息

2.1.2 用户登录

通过注册的账户（邮箱）密码登录

找回忘记的密码

2.1.3 接收任务

查看每项任务的具体内容、悬赏、发布者的部分信息

根据条件对任务进行筛选

选择接受任务

2.1.4 发布任务

添加任务的标题、具体内容、截止日期、悬赏

2.1.5 已接收（未完成）任务管理

查看任务具体信息
对任务状态进行更新
与发布者在线交流
放弃任务

2.1.6 已发布（未完成）任务管理

查看任务具体信息
对任务细节进行更新
与接收者在线交流
放弃任务

2.1.7 已接收（失效）任务管理

查看任务具体信息

2.1.8 已发布（失效）任务管理

查看任务具体信息
对接收者进行评价

2.1.9 用户个人信息管理

查看已接收任务评价
修改密码
修改个人信息
注销

2.2 界面设计及功能点

2.2.1 登录界面

输入账号（邮箱）、密码
忘记密码
新用户注册

2.2.2 注册界面

输入邮箱
验证？
输入密码

2.2.3 用户信息修改界面

填写个人信息

2.2.4 主界面

主界面分为三个 tab

2. 2. 4. 1. Tab1：发布界面浏览

全部未失效未被接收任务的标题、悬赏、截止日期，按发布时间排列
几个筛选的过滤器
上滑刷新
入口：发布新任务

2. 2. 4. 2. Tab2：个人管理

个人头像、昵称、已获取评分、发布的/接收的任务数
入口：个人信息修改

入口：我发布的任务
入口：我接收的任务
入口：我获得的评价
退出登录

2.2.4.3. Tab3：我的消息

按照任务更新的时间排序，显示最新消息
未失效的任务交流
已失效的任务交流
入口：查看任务详情

2.2.5 任务详情页（我发布的）

接收者（昵称、评分、接收过的任务数）
入口：与接收者交流
入口：修改任务详情
任务截止时间
任务标题
任务描述
悬赏
任务最新状态
入口：评价（点击弹出评分界面、文字输入框）
我对该任务的评价（任务失效后出现）

2.2.6 任务详情页（我接收的）

发布者（昵称、发布过的任务数）
入口：与发布者交流
入口：更新任务状态（点击弹出文字输入框）
任务截止时间
任务标题
任务描述
悬赏
任务最新状态
我收到的评价（任务失效后出现）

2.2.7 我发布的任务

所有我发布过的任务，按发布时间次序排列
对任务状态用 UI 进行区分

2.2.8 我接收的任务

所有我接收过的任务，按发布时间次序排列
对任务状态用 UI 进行区分

2.2.9 任务详情编辑

任务截止时间
任务标题
任务描述
悬赏

2.2.10 在线交流页

输入框

任务标题、悬赏及截止日期
发布者发送消息显示
接收者发送消息显示
系统消息显示（任务状态更新、任务详情更新）

2.2.11 我获得的评价

所有我接收过的任务的评价（评分），按发布时间次序排列
总评分

2.3 任务的生命周期

为了进一步阐明平台的核心功能，我们对几类常见的任务生命周期进行示例，其中横线代表着一个特定的时间点，并对应一条聊天界面中弹出的系统消息；其他则代表着后台中的任务状态，通过后端维护。

2.3.1 正常的任务周期

已发布未接单
- 接单

已接单
- 接收者发送状态更新消息 A
- 接收者发送状态更新消息 B
- 到达截止时间

已超时
- 任务完成

已完成待评价
- 任务评价

已终止

2.3.2 任务无人接受

已发布未接单
- 发布者关闭任务

已终止

2.3.3 任务超时，被一方主动关闭

已发布未接单
- 接单

已接单
- A 发起关闭请求

任务关闭请求中
- B 拒绝

已接单
- 到达截止时间

已超时
- A 关闭任务

已失效待评价

- 任务评价

已终止

2.3.4 任务到期前被发起关闭请求，另一方无响应

已发布未接单

- 接单

已接单

- A 发起关闭请求

任务关闭请求中

- 到达截止时间
- 自动关闭任务

已失效待评价

- 任务评价

已终止

2.4 非功能性需求

2.4.1 并发性能优化

对多用户同时访问平台的性能进行优化、对多个用户同时抢单的情况保证原子性。

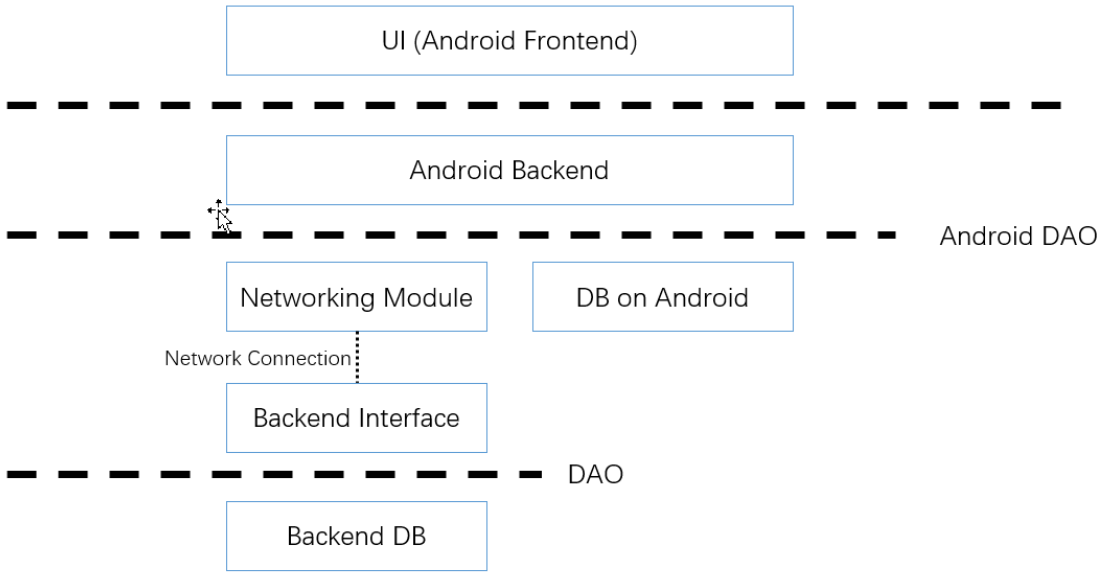
2.4.2 网络传输加密

基于 HTTPS 对后台进行访问。

3. 工程设计

3.1 软件架构设计

为了便于整个产品各层次之间解耦，我们将整个软件分隔为如下四层：



自上而下，这四层分别为

UI 层：UI 层负责 APP UI 的绘制工作，对不同的用户输入进行相应的数据处理，并通

过访问 APP 内主要对象的对应方法来对 Android 逻辑层进行控制。

Android 逻辑层：Android 逻辑层负责维护 APP 内各页面的跳转、手机上各类状态间的逻辑状态转换、后台消息的接收和更新等，并在各类的方法内部通过访问统一的 DAO 接口来对下层的数据进行读写操作。

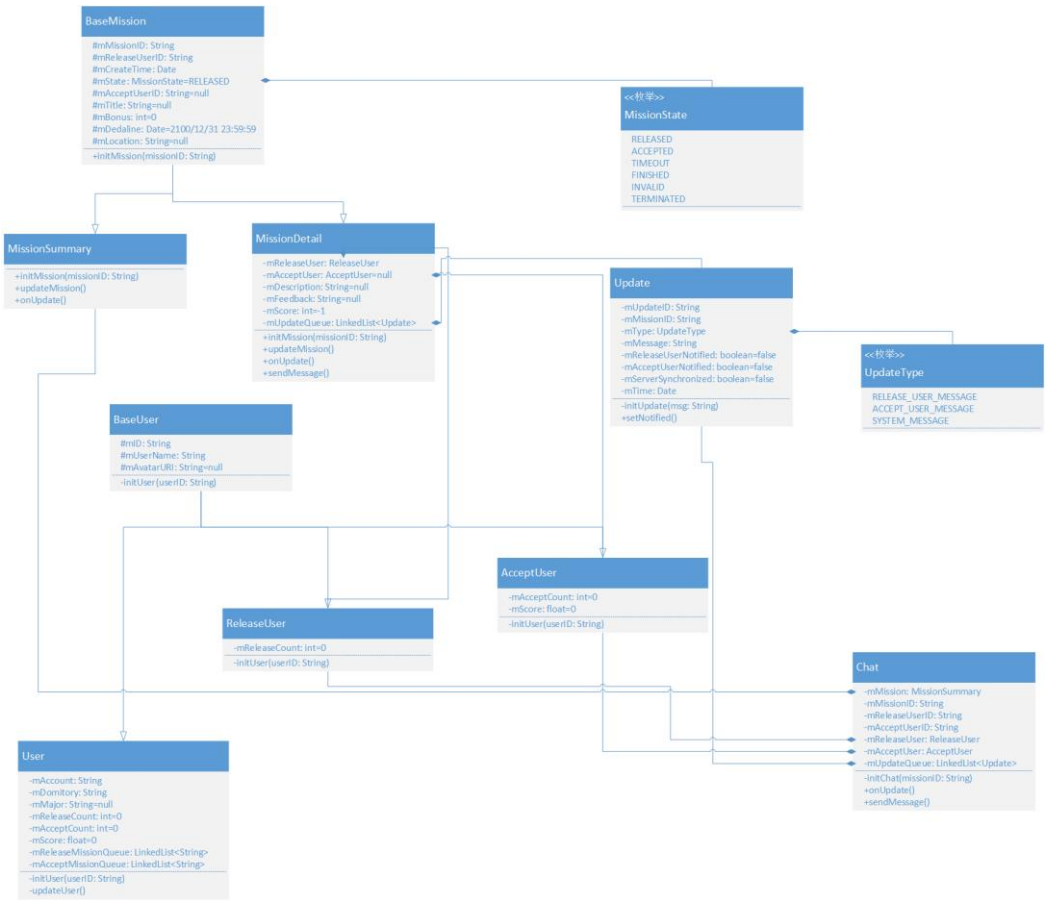
数据层：数据层将数据的具体访问与上层逻辑抽离开，在该层，上层的数据访问将被分发为对 Android 端上数据库的访问和对远程后台数据库的访问，并执行相应的访问。对 Android 端上数据库的访问将直接执行，而对后台数据库的访问，将会通过 RPC 或 WebSocket 方式进行。为了开发便利，我们没有在前后端之间做层次分离，而是在逻辑层与数据层之间做接口分离，并统一由数据层维护前后端的完整通信过程。考虑到后端接口调整本身非常方便，且其与 Android 端底层的强耦合特点，我们认为这样的抽象比前后端分离更有助于开发。

后台数据层：在后台暴露的网络接口之下，我们通过后台的一系列 DAO 接口来将上层的通信和后台的数据处理进行分离，并设立最底层的后台数据层。该层主要保持并发稳定性，并执行最底层的相应数据库操作。

在我们的模型里，前两层在行为上强耦合（如，当 UI 执行某一个更新操作时，必然在逻辑层有对应的调用出现），而后三层在数据上强耦合（如，当逻辑层执行某一个数据更新操作时，后两层也需要随之进行更新）。在未来开发中，我们可以隔离其他层来对某一层进行独立开发，因而更有利于小组合作。

3.2 类设计

在数据上，我们主要对 Android 逻辑层内的类进行了抽象和划分，具体类 UML 图如下：



总体而言，我们将数据分为四类：

任务 Mission：平台上的所有众包任务

聊天 Chat：针对某一任务的消息流

消息 Update：由任务发起者、任务执行者、系统三方之一发送的任务更新消息

用户 User：发起/执行任务的个体

3.3 接口设计

如前所述，我们没有对前后端之间的接口在本阶段进行具体设计，但是对于逻辑层和数据层之间的接口，我们在这里给出大致实现范围：

任务：实现初始化 initMission、详情更新 updateMission、接收后台更新 onUpdate、发送新的更新消息 sendMessage 四类接口

聊天：实现初始化 initChat、接收后台更新 onUpdate 两类接口

消息：实现初始化接口 initUpdate

用户：实现初始化接口 initUser、详情更新 updateUser

这里需要具体指明 updateMission 与 sendMessage 的区别。对任务的更新有两类，一类为改变任务的标题、内容等详情，通过 updateMessage 进行更新；一类为向系统/另一方发送新的消息通知，通过 sendMessage 进行更新

3.4 数据库设计

基于上述的类图设计，按照 3NF 分解后，我们定义了三类数据表，其具体设计如下

MissionDtail				
列名称	列含义	列类型	默认值	主键
mMissionID	任务的编号	String		pri
mReleaseUserID	任务发起人的ID	String		
mCreateTime	任务发起的时间	Date		
mState	任务此时的处理状态	MissionState	RELEASED	
mAcceptUserID	接受任务人的ID	String	"none"	
mTitle	任务的标题	String	"none"	
mBonus	任务的奖金	float	0	
mDeadline	任务的DDL	Date	2100/12/31	
mDescription	任务的详细描述	String	"none"	
mFeedback	任务的反馈	String	"none"	
mScore	任务的得分	int	0	

Update				
列名称	列含义	列类型	默认值	主键
mMissionID	更新的是哪个任务	String		
mUpdateID	该次更新的编号	String		pri
mType	更新的种类	UpdateType		
mMessage	更新的具体信息	String		
mReleaseUserNotified	是否通知发起人	boolean	FALSE	
mAcceptUserNotified	是否通知接收人	boolean	FALSE	
mServerSynchronized	管理端是否已成功接受更新	boolean	FALSE	
mTime;	更新的时间	Date		

User				
列名称	列含义	列类型	默认值	主键
mID	用户的编号	String		pri
mUserName	用户的昵称	String		
mAvatarURL	用户的头像	String		
mAccount	用户的账号	String		
mDormitory	用户的寝室	String	"none"	
mMajor	用户的专业	String	"none"	

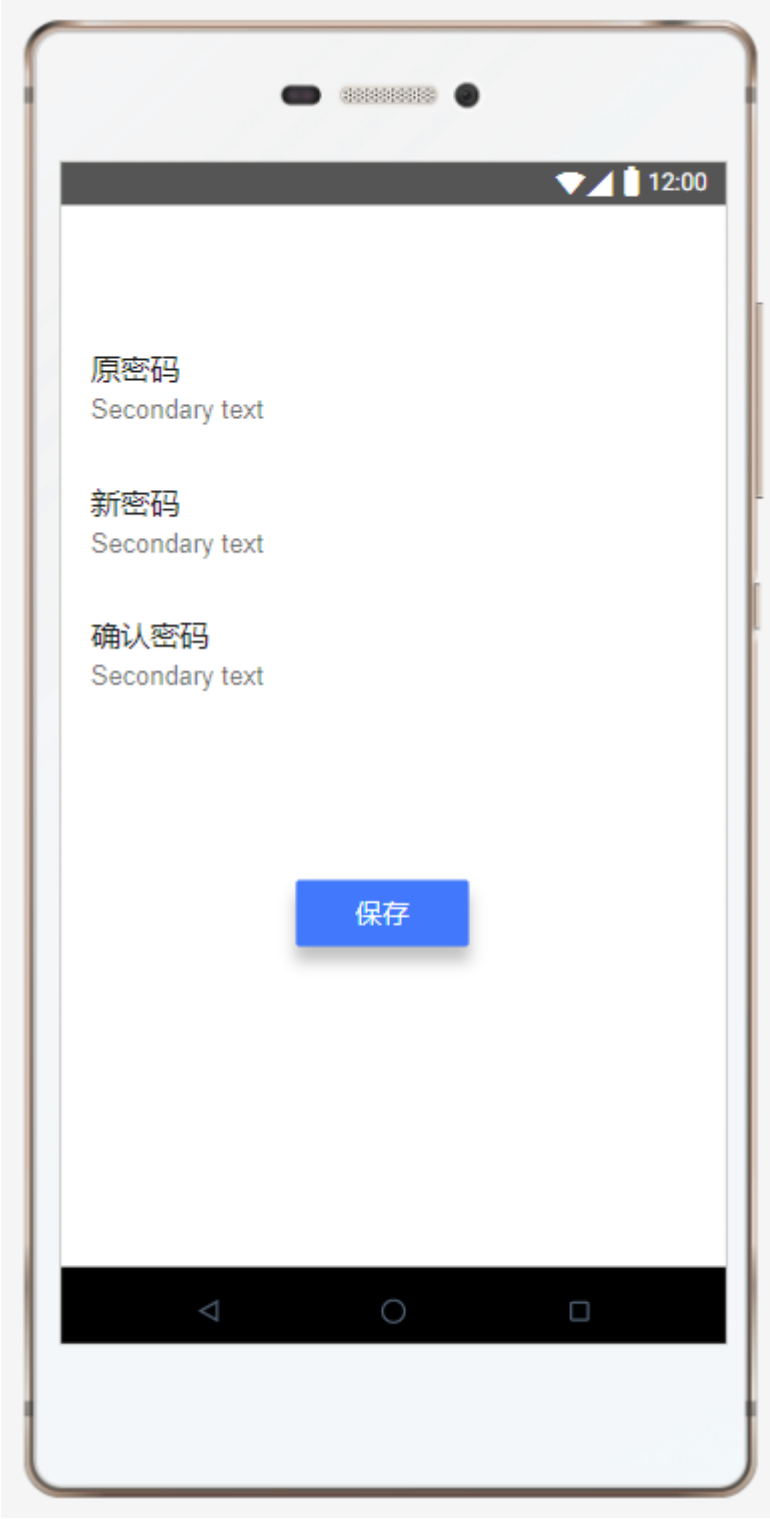
3.5 用户界面模型

3.5.1 登录界面

3.5.2 个人信息设置


12:00
avater
uid:114514
dorm:zijing#1
昵称
Secondary text
邮箱
Secondary text
更改密码
Secondary text
保存

3.5.3 密码修改页


12:00
原密码
Secondary text
新密码
Secondary text
确认密码
Secondary text
保存

3.5.4 主界面

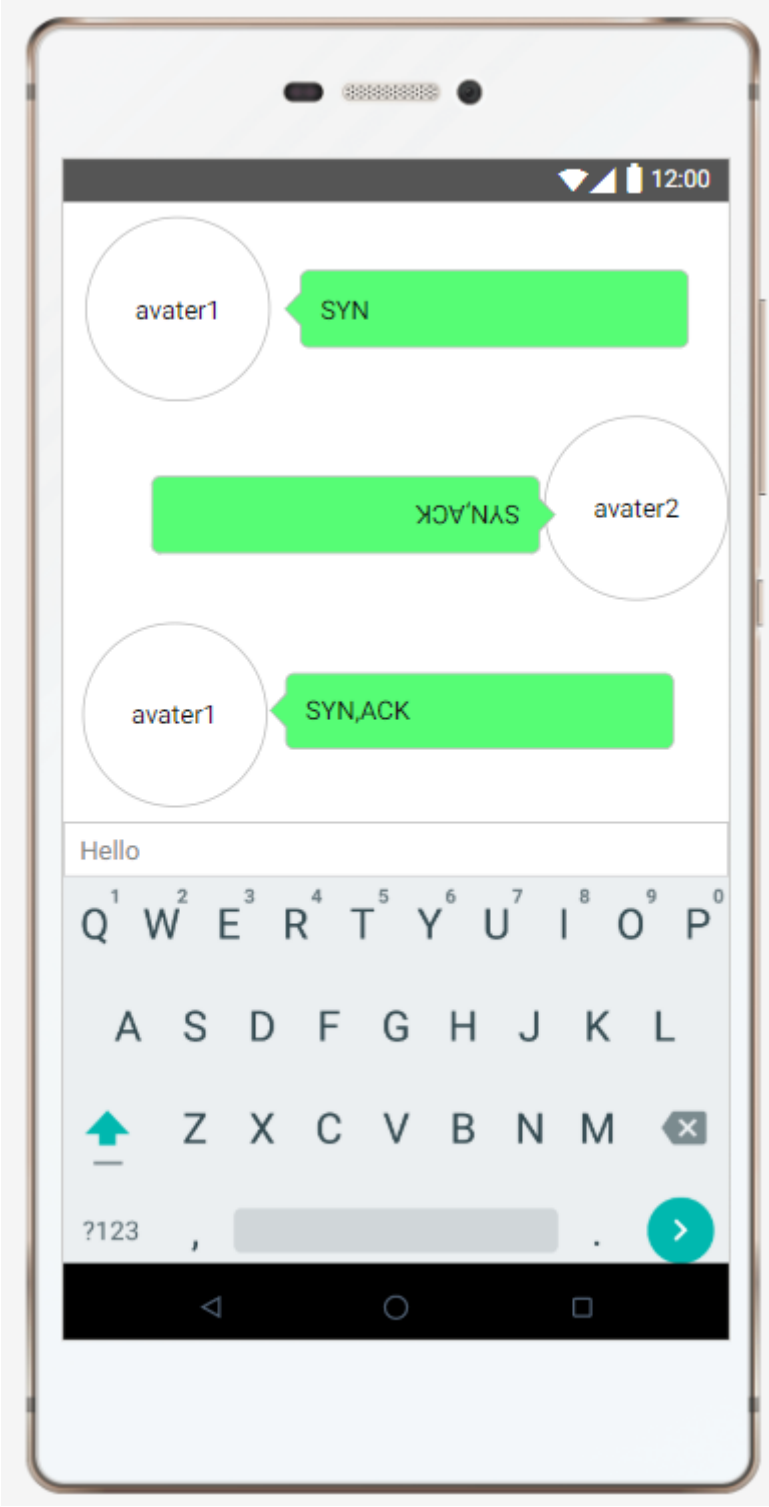

12:00
带帮手
我的下单
订单市场
我的接单
avatar1
摘要1
ddl1
bonus1
avater2
摘要2
ddl2
bonus2
avater3
摘要3
ddl3
bonus3
+

3.5.5 任务详情页


12:00
commiter's avater
score1
nickname1
major1 dorm1
receiver's avater
score2
mCounts
nickname2
major2 dorm2
title
missionId
details
status: 有人接单/无人接单
time to ddl
抢单
cancel
返回首页

3.5.6 任务详情编辑

3.5.7 聊天界面



3.6 算法设计

当前，我们认为我们的主要操作来自数据库的读写和简单的状态转移，暂时没有考虑具体算法，算法将就未来性能瓶颈再具体设计。

3.7 待敲定细节

受限于当前设计时考虑的周全程度、当前课程进度导致的知识体系不全等原因，我们尚未考虑如下内容，将在之后学到相关知识/实际开发中再行敲定：

密码存储：根据初步调研，安卓平台上通过 AccountManager 进行密码管理，具体实现方式需要进一步学习。

后台消息通知：根据初步调研，安卓平台 APP 的消息通知主要通过 Service 实现，具体实现方式估计后面课程会覆盖。

多线程：根据初步调研，安卓平台一般将主线程更新 UI，用其他线程执行其他计算任务，实现方式估计后面课程会覆盖。

网络通信：根据初步调研，RPC 可能是比较便于实现通信的方式，对一般数据库访问将通过请求/应答来处理，对聊天将通过双向流式通信+定时心跳包的方式来处理，具体实现方式需要进一步学习。

后台并发控制：对于并发冲突情况，应由后台发送相应状态码到安卓端上进行通信，具体方式待定义。

4. 开发时间表

按照优先级，我们大致总结出如下任务：

1. 主界面或 Tab 的静态实现、任务详情界面的静态实现
2. 任务状态动态更新、实时通信模块
3. 登录界面、其他零碎功能完善
4. UI 界面美化

其中，从 11 到 15 周大致 5 个星期的时间，计划用前三个星期完成前两项，后两个星期完成后两项。

对应到最开始的产品架构上，我们将优先着手上方的两个层次（UI 层和 Android 逻辑层）的开发，通过模拟数据访问的方式来隔离底层两个层次的实现。在上方两个层次大致完成后，再着手开发数据层和后台数据层。