实验四 系统登录注册模块(Android app)开发说明文档

天气查询APP详细设计说明书本

一：整体介绍

1.APP说明：

进一步理解Java、SDK、ADT、IEDEA、Springboot的彼此依赖关系,并且能熟练使用java语言来编写Android工程，并利用爬虫爬取不同城市的天气信息，实现不同语言之间的相互调用。掌握Android应用开发环境的搭建方法以及虚拟机的配制方法，掌握Android工程创建方法，掌握基于虚拟机与真机的Android工程运行方法，了解Activity生命周期，理解Activity事件回调，onRestoreInstanceState()和onSaveInstanceState两个函数的使用。在此基础上,掌握Android客户端与服务器通信的原理并且运用到项目中,理解Android发送http网络请求，包括GET请求和POST请求，熟悉Android异步任务的处理方法，包括各种回调函数。能够运用Springboot和数据库来实现一个登录界面。最后用json实现android客户端与服务端的通讯。

2.开发人员介绍和APP功能说明：

前端：韦刚永，李曾一

后端： 李曾一，陈光月

天气爬虫：陈光月，李曾一，韦刚永

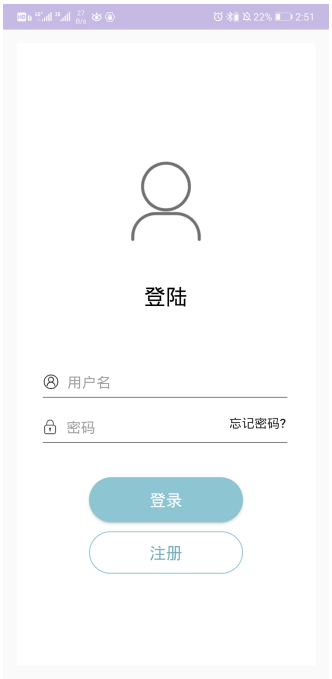
前端界面包含：登陆界面，注册界面，密保问题设置界面，天气显示界面，设置界面（包含城市切换，修改密码，退出登录等功能），修改密码界面，找回密码界面等近10个界面。

附加了天气查询功能。该功能通过前端返回请求，后台调用python爬虫代码爬取中国天气网给出的绑定城市的实时天气信息。详细的界面展示见附录。

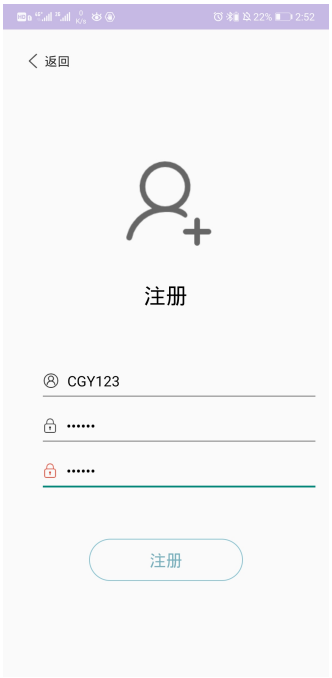
二：模块说明

1. 前端模块：

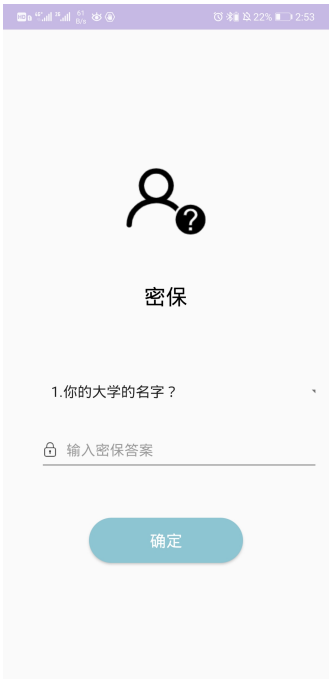
1. 登录界面：
    1. 用户名和密码的输入，前端进行动态验证。
    2. 登录按钮，点击后将用户名和密码发送至后端进行校验。若成功则跳转至（7）主界面。若失败则弹出信息提示用户名或密码错误。
    3. 注册按钮，点击后跳转至（2）注册界面。
    4. 忘记密码按钮，点击后跳转至（4）忘记密码界面



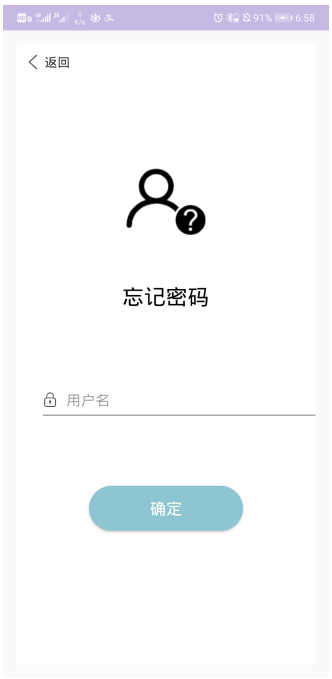
2. 注册界面:
    1. 用户名，密码，确认密码三个输入框，动态校验，其中用户名至少包含一个大写字母，确认密码和密码一致，长度符合实验要求。若某一项不符合要求，则对应前部图标变红。
    2. 返回按钮，点击后返回（1） 登录界面。
    3. 注册按钮，点击后提交用户名和密码到后端校验，根据校验结果，若注册成功则跳转至（3）设置密保界面。若用户名已存在给出提示信息。



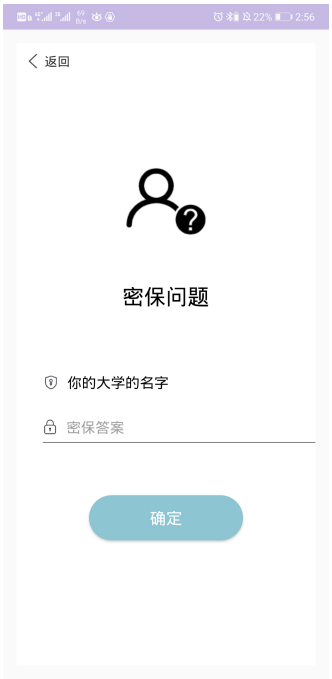
3. 设置密保界面：
    1. 密保问题选择，下拉菜单，不同的问题有不同的id
    2. 输入密保答案
    3. 确认按钮，点击后将密保问题的id和密保答案传至后端，后端返回保存成功后跳转至（1）登录界面。若失败弹出提示信息



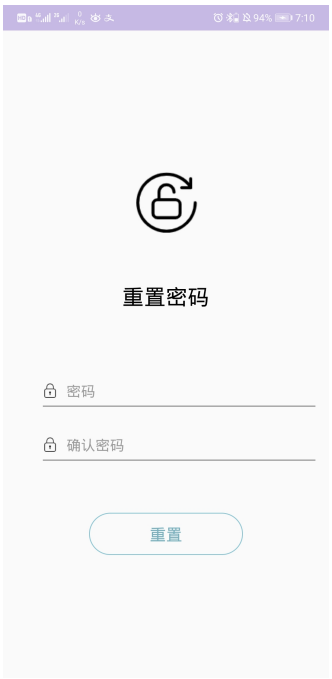
4. 忘记密码界面：
    1. 返回按钮，点击后返回（1） 登录界面。
    2. 用户名输入框，用户输入忘记密码的用户名，动态校验。
    3. 确定按钮，点击后将输入的用户名信息传输到后端校验，若用户名存在，跳转至（5）密保验证界面，若用户名不存在，则提示用户名不存在。



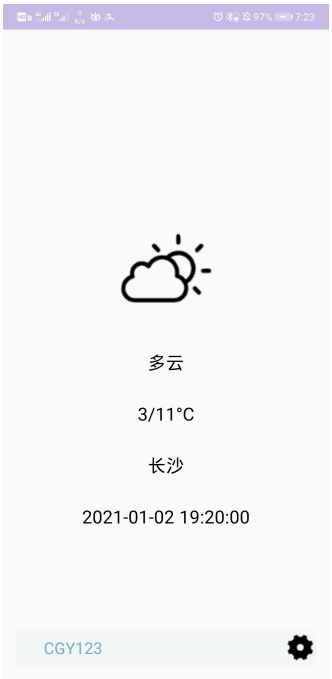
5. 密保验证界面：
    1. 返回按钮，点击后返回（1） 登录界面。
    2. 后端传入注册时选择的密保问题，显示在该页面上。
    3. 输入框输入密保问题答案。
    4. 确定按钮，点击后将输入的密保问题答案传入后端，与注册时设置的答案进行对比。根据后端的返回值，若成功跳转至（6）密码重置界面，失败则提示验证失败。



6. 密码重置界面：
    1. 密码和确认密码的输入框，由用户输入修改后的密码和确定密码，动态校验，二者要求一致。
    2. 重置按钮，点击后将数据发送至后端，更新数据库中用户名对应的密码的MD5值。页面显示成功并显示修改后的密码。点击“我知道了” 按钮，跳转至（1）登录界面。

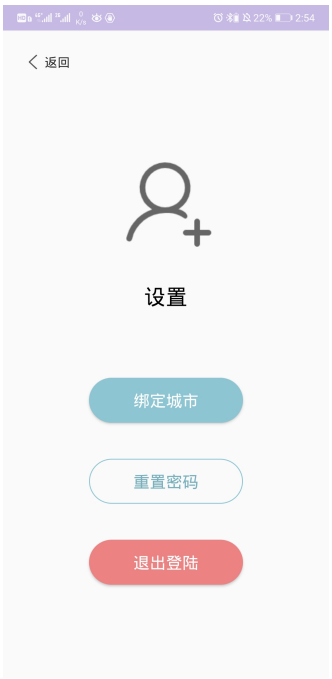


7. 主界面：
    1. 天气显示部分，显示天气、对应的图标，温度、城市名。
    2. 时间显示部分，显示页面刷新时，服务器时间。
    3. 用户名显示部分，显示当前登录的用户名。
    4. 下拉刷新模块，此页面支持下拉刷新，每次刷新调用后端接口，获取相对应的数据。
    5. 齿轮为设置按钮，点击后跳转至（8）设置界面。



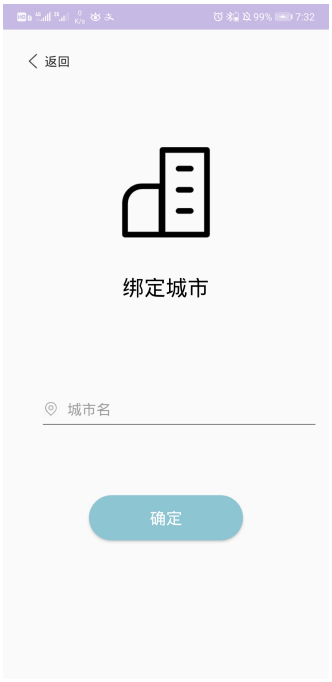
8. 设置界面：

    1. 返回按钮，点击后返回（7）主界面。
    2. 更换城市按钮，点击后跳转至（9）绑定/更换 城市界面。
    3. 重置密码按钮，点击后跳转至（10）修改密码界面。
    4. 退出登录按钮，点击后退出当前用户，跳转至（1）登录界面。

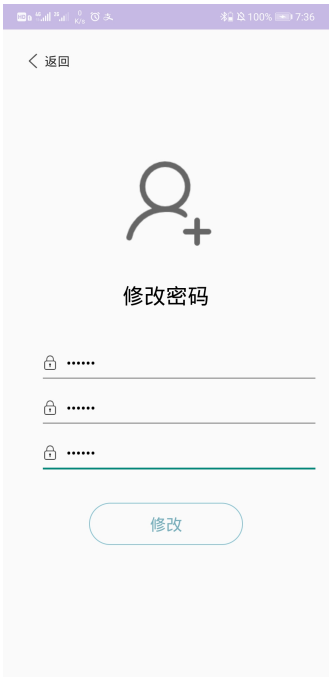


9. 绑定/更换 城市界面：

    1. 返回按钮，点击后返回（8）设置界面。
    2. 城市名输入框，用户输入需要绑定/更换 的城市名。
    3. 确定按钮，点击后将输入的城市名传入后端，查询数据库中该城市名是否存在，返回结果。若不存在，提示“城市不存在”，若存在，弹出提示消息“成功....”，点击 “我知道了” 返回至（8）设置界面。



10. 修改密码界面
    1. 返回按钮，点击后返回（8）设置界面。
    2. 原密码输入框，输入当前用户的密码。
    3. 新密码输入框，输入新密码。
    4. 新密码确认框，重复输入新密码。三者均有动态校验。
    5. 修改按钮，点击后将数据发送至后端，更新数据库中用户名对应的密码的MD5值。页面显示成功并显示修改后的密码。点击“我知道了” 按钮，跳转至（1）登录界面。



2. 后端模块：

数据结构

天气的数据结果为

```
{
    "tq": 天气,
    "wd": 温度
}
```

接口

1. 注册 `/logon`
    1. 接收格式：post

        ```
        {
            "id": 用户名,
            "passwd": 密码
        }
        ```

    2. 接口作用

        根据用户名和密码进行用户的注册，查询user表，若用户存在，返回注册失败；若用户不存在，计算密码的MD5值，将该值和用户名结合，向表中插入该行数据。

    3. 返回格式

        成功

        ```
        {
            "user": 用户名和密码的MD5值结合成的对象,
            "msg":  "注册成功",
            "status": 1
        }
        ```

        失败

        ```
        {
            "user": 用户名和密码的MD5值结合成的对象,
            "msg":  "已存在此用户名",
            "status": 0
        }
        ```

2. 登录 `login`
    1. 接收格式：post

        ```
        {
            "id": 用户名,
            "passwd": 密码
        }
        ```

    2. 接口作用

        根据用户名和密码进行用户的登录，查询user表，计算密码的MD5值，调用用户名密码比对函数。若成功，返回此用户token（使用jwt加密）；若失败，返回错误。

    3. 返回格式

        成功

```
{
    "user": 用户名和密码的MD5值结合成的对象,
    "msg":  "登录成功",
    "status": 1,
    "token":  使用jwt加密的token
}
```

失败

```
{
    "user": 用户名和密码的MD5值结合成的对象,
    "msg":  "用户名或密码错误",
    "status": 0
}
```

3. 设置密保 `setpassguard`
    1. 接收格式：post

```
{
    "id": 用户名,
    "qestion_id": 密保问题id,
    "qestion_ans": 密保问题答案
}
```

    2. 接口作用

    根据用户名，向user_qestion表中插入此用户名，密保问题id和其对应的密保问题答案。

    3. 返回格式

    成功

```
{
    "guard": 用户名，密保id，密保答案结合成的对象,
    "msg":  "密保设定成功",
    "status": 1
}
```

    失败

```
{
    "guard": 用户名，密保id，密保答案结合成的对象,
    "msg":  "已存在",
    "status": 0
}
```

4. 密保验证 `checkpassguard`
    1. 接收格式：post

```
{
    "id": 用户名,
    "qestion_ans": 密保问题答案
}
```

2. 接口作用

   根据用户名，查询user_qestion表，调用密保问题答案对比函数，若成功，返回校验成功。

3. 返回格式

   成功

```
{
    "guard": 用户名，密保id，密保答案结合成的对象,
    "msg":  "验证成功",
    "status": 1
}
```

   失败

```
{
    "guard": 用户名，密保id，密保答案结合成的对象,
    "msg":  "验证失败",
    "status": 0
}
```

5. 密码修改 `changepass`

   1. 接收格式：post

```
{
    "old": 旧密码,
    "new": 新密码
}
```

   2. 接口作用

      从请求头中jwt加密的token中取出用户名，调用用户名密码校验函数进行原密码校验，成功后调用密码修改函数进行密码的修改，会修改user表中的值。

   3. 返回格式

      成功

```
{
    "user": 用户名，密码结合成的对象,
    "msg":  "修改密码成功",
    "status": 1
}
```

      失败

```
{
    "user": 用户名，密码结合成的对象，
    "msg":  "原密码错误",
    "status": 0
}
```

6. 用户名查询 `/checkid`
    1. 接收格式：get

       id

    2. 接口作用

       接收获得的用户名，调用用户查询函数，根据id在user表中查询此用户是否存在，同时会从user_qestion表中获取对应的密保id，再到qestion表中获取对应的密保问题

    3. 返回格式

       成功

       ```
       {
           "qestion": 密保问题，
           "msg":  "用户存在",
           "status": 1
       }
       ```

       失败

       ```
       {
           "msg":  "用户不存在",
           "status": 0
       }
       ```

7. 密码重置 `resetpass`
    1. 接收格式：post

       ```
       {
           "id": 用户名，
           "passwd": 密码
       }
       ```

    2. 接口作用

       根据用户名和密码进行用户密码的重置，计算密码的MD5值，调用用户名密码重置函数。

    3. 返回格式

       ```
       {
           "user": 用户名和密码的MD5值结合成的对象，
           "msg":  "修改密码成功",
           "status": 1
       }
       ```

8. 获取服务器时间 `message`
    1. 接收格式：get
    2. 接口作用

获取服务器时间，格式为yyyy-MM-dd HH-mm:ss

3. 返回格式

```
{
    "time": 时间
}
```

9. 绑定城市 `/bindcity`

1. 接收格式：get

   cityname

2. 接口作用

   从请求头中jwt加密的token中取出用户名，并结合城市名，首先从city表中查询城市名是否正确，然后向user_city表中写入相应数据。

3. 返回格式

   成功

```
{
    "city": 用户名和城市结合成的对象,
    "msg":  "绑定成功",
    "status": 1
}
```

   失败

```
{
    "city": 用户名和城市结合成的对象,
    "msg":  "无效城市名",
    "status": 0
}
```

10. 更换绑定城市

1. 接收格式：get

   cityname

2. 接口作用

   从请求头中jwt加密的token中取出用户名，并结合城市名，首先从city表中查询城市名是否正确，然后向user_city表中写入相应数据，会修改原有的城市。

3. 返回格式

   成功

```
{
    "city": 用户名和城市结合成的对象,
    "msg":  "修改成功",
    "status": 1
}
```

   失败

```
    {
        "city": 用户名和城市结合成的对象,
        "msg":  "无效城市名",
        "status": 0
    }
```

11. 获取城市天气
    1. 接收格式：get
    2. 接口作用

       从请求头中jwt加密的token中取出用户名，调用爬虫程序进行天气的爬取，然后从user_city中取出对应的城市天气
    3. 返回格式

       成功

```
    {
        "city": 用户名和城市和天气结合成的对象,
         "msg":  "天气获取成功",
        "status": 1
     }
```

       失败

```
  {
         "msg":  "天气未获取",
         "status": 0
  }
```

3. 爬虫模块：

1. 数据库连接：连接云端服务器，使用pysql
2. 获取爬取城市对应的citycode

   从city表中查询对应城市名的城市代码

```
def get_citynum(cityname):
    try:
        cursor=conn.cursor()
        cursor.execute("select * from city where cityname = %s",cityname)
        cityCode=cursor.fetchone() #匹配一个
        cursor.close()
        return cityCode[1]
    except:
        print('未找到该城市')
        return 1
```

3. 爬取天气信息，返回天气信息

```
def spider(url,headers):
    response = requests.get(url,headers)
    content = response.content.decode('utf-8')
    pat_weather = re.compile('<input type="hidden" id="hidden_title" value="
(.*?)" />')
    pat_up_time = re.compile('<input type="hidden"
id="fc_24h_internal_update_time" value="(.*?)"/>')
    weather = pat_weather.findall(content)
    up_time = pat_up_time.findall(content)
    #print(type(weather[0]))
    a= weather[0].split(' ')
    whether_inf={}
    whether_inf["tq"]=a[3]
    whether_inf["wd"]=a[5]
    return whether_inf
```

4.拼接城市码，合成合法的URL，爬取天气

```
def main(cityCode):
    headers = {
        'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2)
AppleWebKit'\
                      '/604.4.7 (KHTML, like Gecko) Version/11.0.2
Safari/604.4.7'
    }
    url = 'http://www.weather.com.cn/weather1d/%d.shtml' % cityCode  #得到城市
天气网址
    temp=spider(url,headers)
    return temp
```

5.获取天气信息，转化为json格式存入user_city表中。

```
def opdatabase(user,temperature):
    cursor = conn.cursor()
    d_json=json.dumps(temperature)
    sql1="update user_city set weather='{json}'  where id ='{id1}';"
    conn.autocommit(1) #此行一定要加
    sql=sql1.format(json=pymysql.escape_string(d_json),id1=user)
    conn.commit()
    cursor.execute(sql)
    cursor.close()
    conn.close()
```

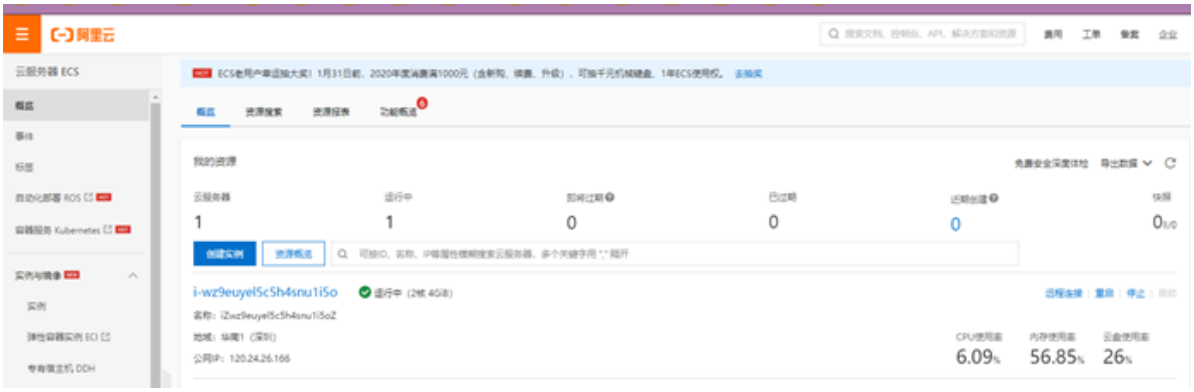
4. 数据库说明：

1. user表：用于存储用户信息，包括用户名（id）、密码的MD5值（passwd）。其中用户名为主键。
2. question表：用于储存密保问题，包括密保问题ID（qestion_id)、问题（qestion）。其中密保问题ID为主键。
3. user_question表：用于储存每个用户自身设置的密保问题，以及对应的答案，包括用户名（id）、密保问题ID（qestion_id)、密保答案（qestion_ans）。其中外键为用户名和密保问题，分别对应user表和question表。

4. city表：该表中大约有2500行信息，包括了全国个城市的名称（cityname）和城市代码（citycode），作用是根据不同用户绑定的不同城市查询其对应的城市码，返回给python爬虫构成完整的URL，以便爬取该地区的天气信息。同时，用户在绑定城市时，用户提交的城市名称也会在该表中检索一次，以判定用户绑定城市的合法性
5. user_city表：用于储存每个用户绑定的城市以及对应的城市天气，包括用户名（id）、城市名（city）、天气（weather）。其中用户名和城市名为外键，对应user表和city表。天气使用json格式保存。

三：项目亮点

1. 使用了部署在阿里云上的Linux服务器
2. 系统为CentOS 8.0 64位；硬件2核 4G



3. 后端使用Springboot框架
4. 后端对密码进行了MD5校验，数据库中存储的是密码的MD5值。
5. 进行了不同语言之间的互相调用，后端java程序会调用python爬虫。
6. 前端在输入用户名密码等数据是可以进行动态校验，图标的颜色可以进行实时的变化。
7. 使用了jwt加密的token，用户端保留token，当用户登陆过一次后，再次点击软件可以实现自动登陆。
8. 一些后端接口实现了拦截功能，需要有相应的合法token才能进行访问。