移动互联网开发大作业

磨蹭天气

风云千里眼 气象顺风耳

磨蹭天气
Moceng Weather

项目成员：cyj，lhq，lx，why

课程老师：zjh

目录

磨蹭天气
Moceng Weather

产品介绍

磨蹭天气，是一款免费的天气信息查询软件，目前支持**全国3557个地区**的**天气查询**，**精准定位**和及时**推送小时级天气预报**，实时**监测阴晴雨雪**。支持**降雨可视化**和**极端天气预警**，还建立了**用户交流社区**模块，便于用户交流分享。

命名为“**磨蹭天气**”目的是对标“墨迹(磨叽）天气”

我们原创设计的logo，综合了**多云、降雨和晴天**三种气象天气，云朵代表多云，半边太阳代表晴天，下方的横线代表降雨，充满丰富的气象特色，能够突出我们的软件功能。

磨蹭天气
Moceng Weather

现有产品存在的问题

- 界面布局冗余复杂
- 缺乏个性化服务
- 用户粘性不足

磨蹭天气特色

- 界面简约，布局清晰
- 个性化天气提醒
- 丰富的可视化
- 社区交流，内容分析

项目可行性与应用前景

磨蹭天气
Moceng Weather

用户交流社区模块

磨蹭天气还建立了用户交流社区模块，让用户能够分享天气心得、交流相关话题，增强用户之间的互动和参与感。这种社区模块有助于用户建立更紧密的社交网络，提升用户黏性和用户满意度。

精准定位和实时监测

磨蹭天气通过精确的定位技术，能够实时获取用户所在位置的天气信息，并及时推送小时级天气预报。这种精准和实时的功能使得用户能够更好地计划和安排他们的日常活动。

降雨可视化和极端天气预警功能

磨蹭天气提供降雨可视化和极端天气预警功能，帮助用户更直观地了解降雨情况，并及时做出相应的准备和应对措施。这种功能对于户外活动、旅行和灾害应对等方面有着重要的应用价值。

市场需求

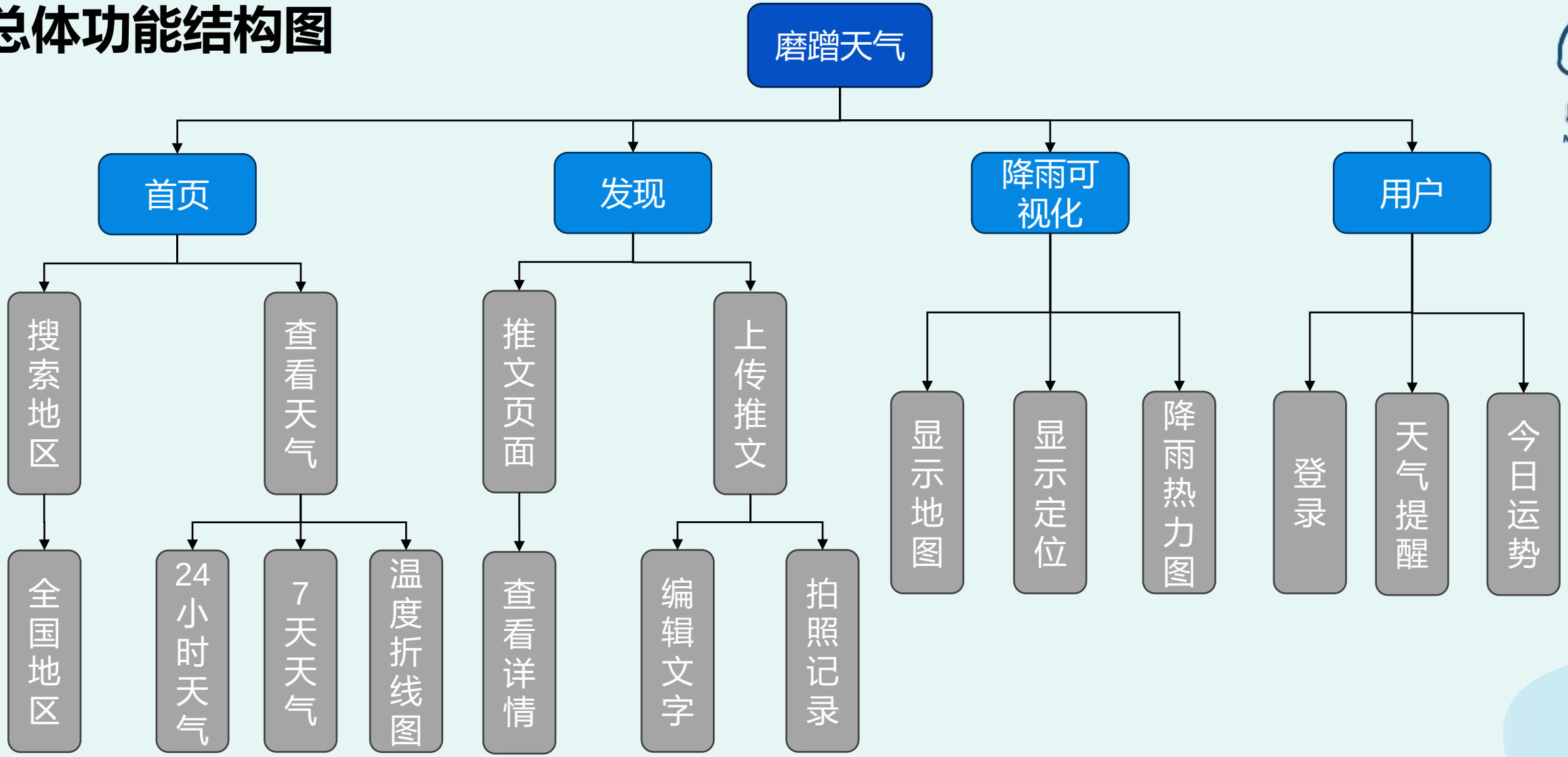
天气对人们的生活和活动有着重要的影响，人们需要及时准确地获取天气信息。磨蹭天气满足了用户对全球范围内天气查询的需求，提供了广泛的城市和地区的天气预报，降低了用户查询天气信息的难度和时间成本。

前景

随着用户对天气信息的需求不断增长，磨蹭天气有着巨大的发展潜力，并能够持续吸引和留住用户，成为用户们获取天气信息的首选软件。

总体结构设计

总体功能结构图



磨蹭天气
Moceng Weather

功能模块设计

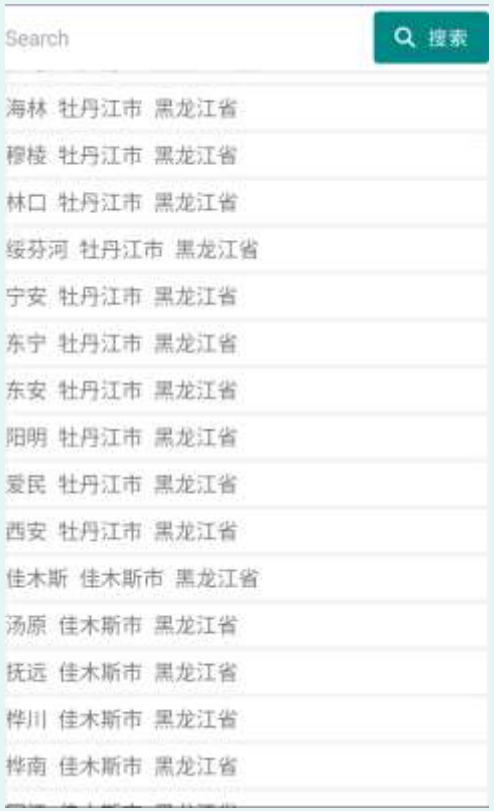
磨蹭天气
Moceng Weather

首页模块

本界面能够显示当前位置地区的实时天气、风向、湿度、空气质量、24小时天气和气温折线图、未来七天天气、以及当天各项生活指数。并且可以切换所要查看天气的地点。

实时天气+24小时温度折线图

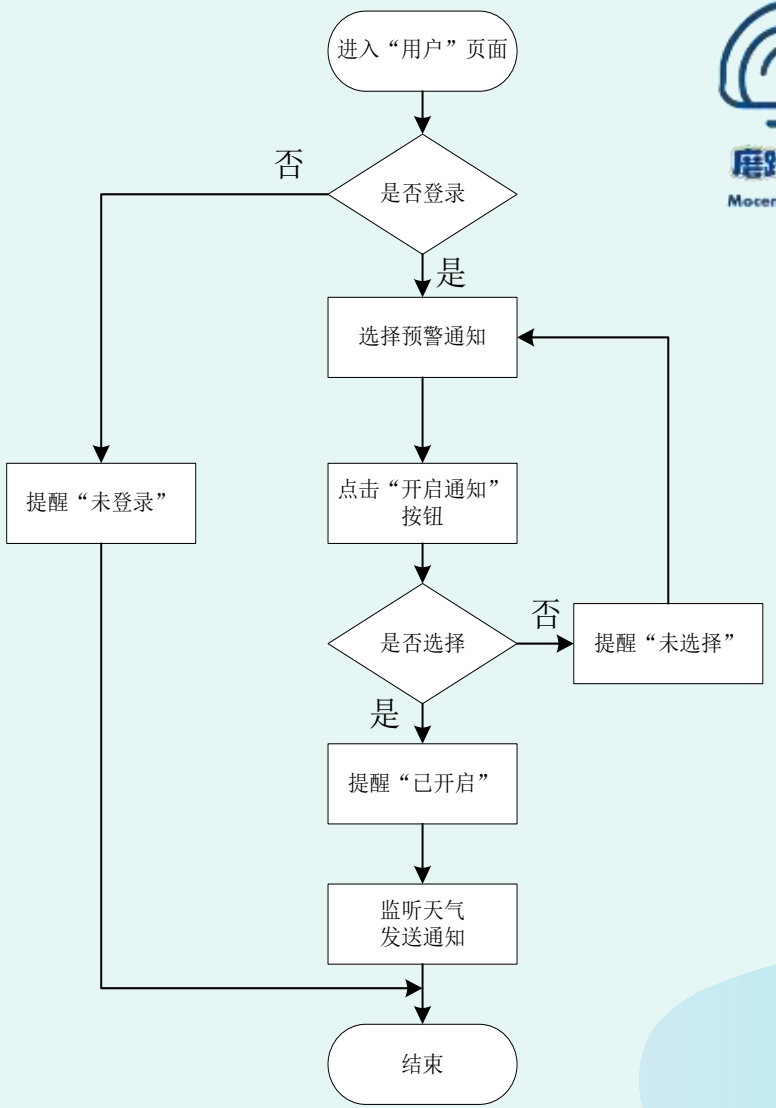
24小时天气+未来七天天气



地区搜索

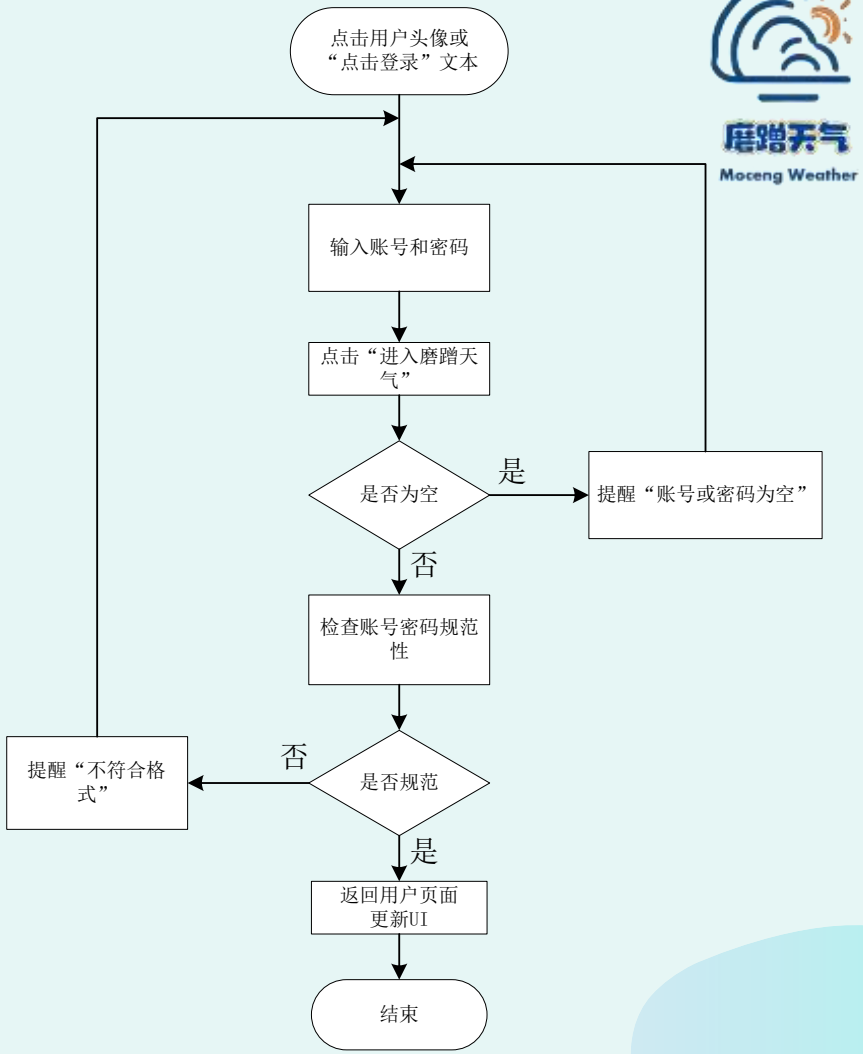
用户页面

用户主页在美观性上做足了功夫，尽力做到布局让用户舒适，用户能够通过点击左上角的用户头像进行登录，也可以点击“点击登录”文字进行登录。在这一页面，增添了天气预警功能，在登陆后，用户能够通过选择需要接收的预警，来开启手机的预警通知。同时右上角设置“运势”图标，能够点击进入“运势”页，增加了软件的趣味性可玩性。



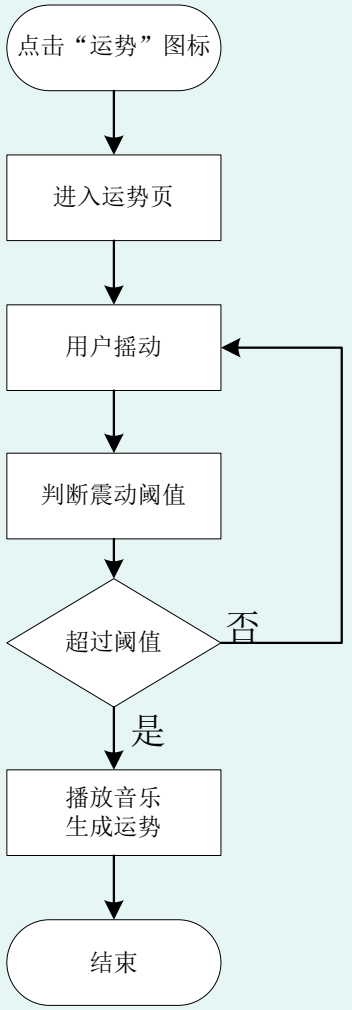
登录页

在登录页上，我们的布局简约且耐看，精心选择了背景。在这里，我们可以输入账号和密码进行登录，我们会对输入的账号和密码进行规范性检验，在账号密码正确填写后点击“进入磨蹭天气”按钮便可返回用户页面，并且能够观察到UI的变化。



运势页

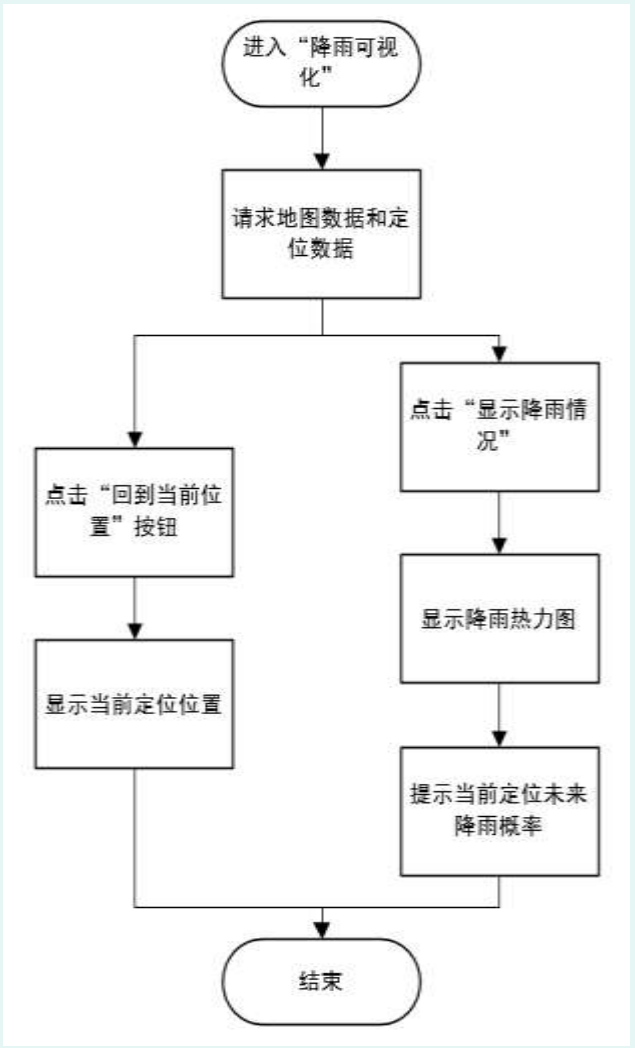
在运势页上，我们根据系统时间对日期进行更新，并且，我们将使用震动传感器，通过模仿“微信”的摇一摇功能，捕获用户手机的震动来查看今日运势，更新UI，在增添功能的同时，也提高了本软件的可玩性。



降雨可视化模块

本界面能够显示当前位置地区和其他大城市的24小时降雨热力图的变化，信息窗提示当前位置未来24小时内的最大降雨概率。拖动地图，再点击“回到当前位置”按钮，地图视角会锁定当前位置。

界面设计图



操作流程图

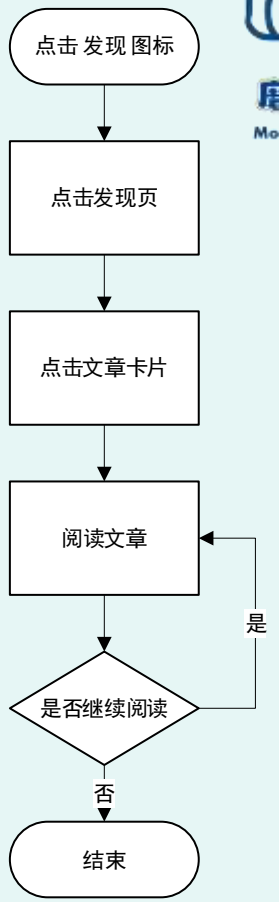
文章发现页

发现页采用流式网格布局，能够显示所有上传的文章。每张卡片的图片高度随机，通过差异化的图片显示达成交错的效果。

阅读页采用滚动布局，通过上下滑动页面可以浏览整篇文章。文章数据通过DBHelper存储并由intent实现Activity间的数据传递。

发现页界面设计图

阅读页界面设计图



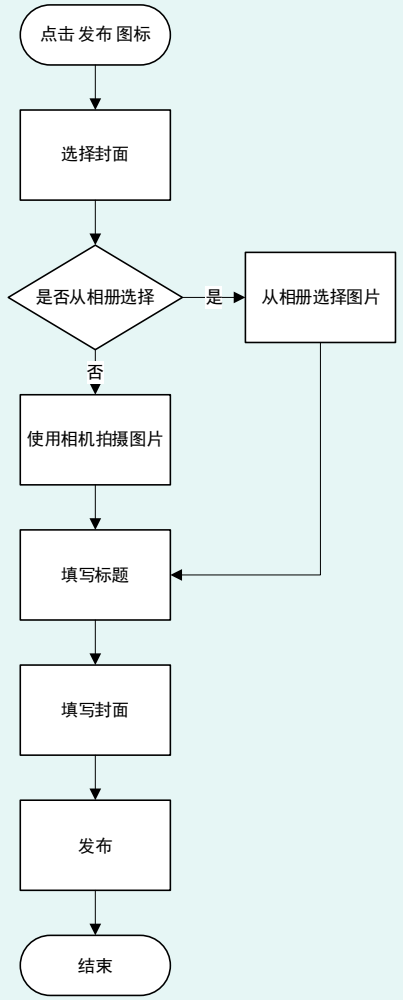
发现页流程图

文章发布页

用户可以在本界面上传他的所见所闻并将其分享给其他用户。用户可以通过拍照或选择图库的图片作为封面，然后填写标题与正文内容并上传到发现页。

用户所填写的文章内容与封面将通过DBHelper存入数据库中。

发布页界面设计图



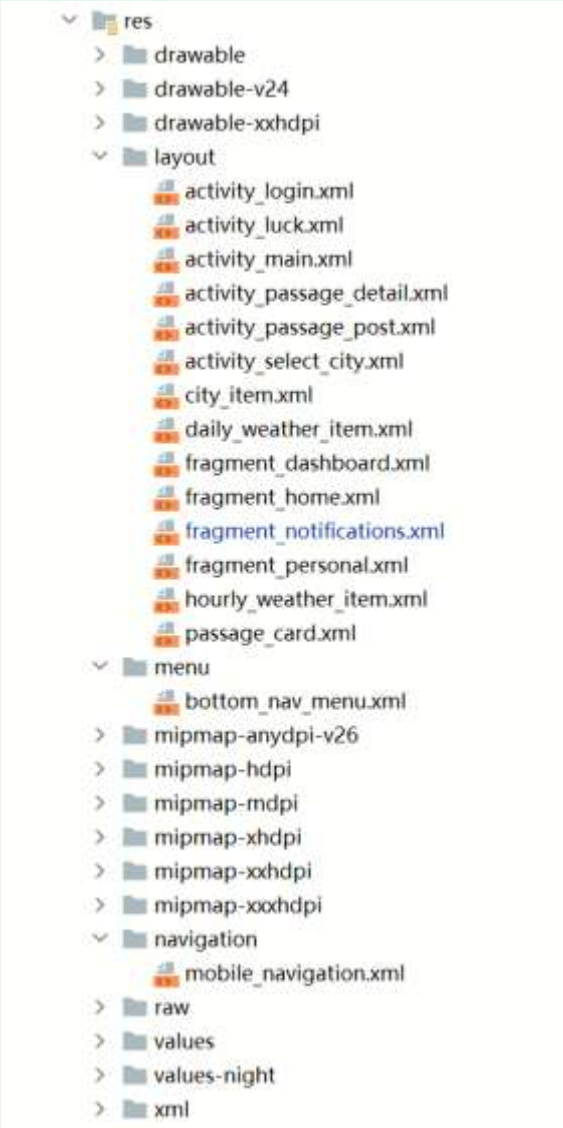
发布页流程图

总体结构设计

磨蹭天气
Moceng Weather

文件架构

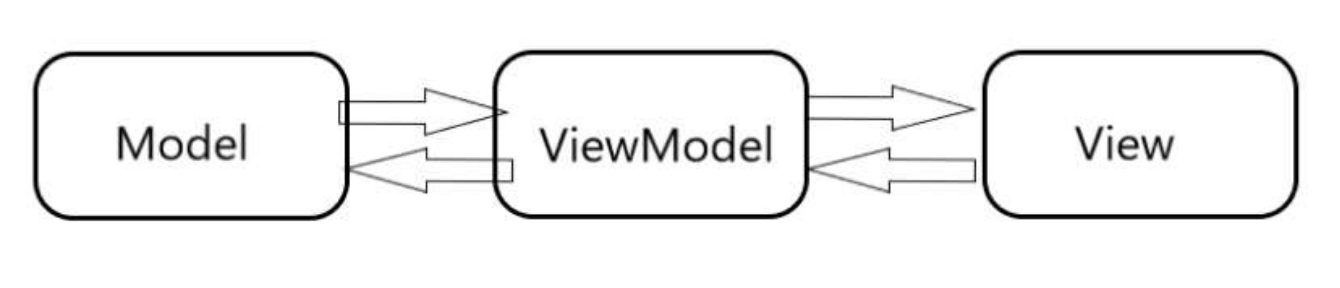
本项目通过采用Bottom Navigation View来控制多个Fragment之间的切换，menu文件夹内的单个文件用于存储导航栏的图标，而navigation文件夹的单个文件用于管理导航栏UI以及每个导航栏图标对应的Fragment。Layout文件夹则存放对应Fragment以及Activity的布局文件。在drawable文件夹我们存放各种图片资源文件等等。

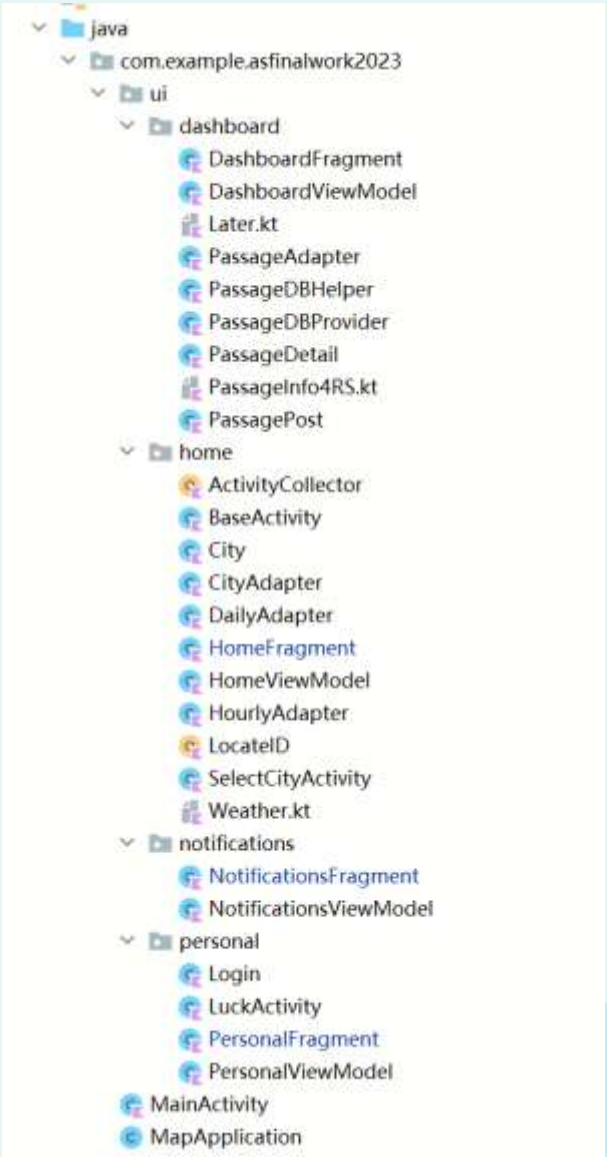


文件架构

dashboard,home,notifications,personal这四个文件夹对应的是底部的四个导航栏，每个文件夹中的fragment是用于承载控件的，而另一个就是mvvm框架下的viewModel。fragment用于显示ui界面，而viewModel则是给ui界面提供数据。

MainActivity则用于控制四个导航栏所对应的Fragment。





关键技术

磨蹭天气
Moceng Weather

首页模块

获取天气数据

```
//获取实时天气url
urlWeatherNow = "https://devapi.qweather.com/v7/weather/now?location=$locate&key=$key"
//获取空气质量url
urlAir = "https://devapi.qweather.com/v7/air/now?location=$locate&key=$key"
//获取24小时天气url
urlWeather24 = "https://devapi.qweather.com/v7/weather/24h?location=$locate&key=$key"
//获取未来七天天气url
urlWeather7 = "https://devapi.qweather.com/v7/weather/7d?location=$locate&key=$key"
//获取生活指数url
urlLive = "https://devapi.qweather.com/v7/indices/1d?type=0&location=$locate&key=$key"
```

和风天气提供的查询各类信息API

```
//读取JSON数据
private fun readJSONData(url:String){
    thread {
        try {
            val client = OkHttpClient()
            val request = Request.Builder().url(url).build()
            val response = client.newCall(request).execute()
            val responseData = response.body?.string()
            if (responseData != null){
                //根据传入的不同url调用不同的函数
                when(url){
                    urlWeatherNow->{weatherNow(responseData)}
                    urlAir->{air(responseData)}
                    urlWeather24->{weather24Hour(responseData)}
                    urlWeather7->{weather7Day(responseData)}
                    urlLive->{live(responseData)}
                }
            }
        } catch (e: Exception){
            e.printStackTrace()
        }
    }
}
```

通过okhttp进行http请求，获取JSON数据，并调用不同的函数对JSON进行处理

首页模块

处理天气数据

```
data class WeatherNow(
    var icon: String,
    var temp: String,        //温度，默认单位：摄氏度
    var feelsLike: String,   //体感温度，默认单位：摄氏度
    var text: String,        //天气状况的文字描述，包括阴晴雨雪等天气状态的描述
    var windDir: String,     //风向
    var windScale: String,   //风力等级
    var humidity: String,    //相对湿度，百分比数值
)
```

数据类来存储数据
(以实时天气数据类为例)

```
//处理空气质量JSON数据
private fun air(JsData: String){
    requireActivity().runOnUiThread {
        val gson = Gson()
        val data: Air = gson.fromJson(JsData, Air::class.java)
        air.text = data.now.category
        when(air.text){
            "优"->{air.setTextColor(Color.rgb( red: 0, green: 228, blue: 0))}
            "良"->{air.setTextColor(Color.rgb( red: 255, green: 255, blue: 0))}
            "轻度污染"->{air.setTextColor(Color.rgb( red: 255, green: 126, blue: 0))}
            "中度污染"->{air.setTextColor(Color.rgb( red: 255, green: 0, blue: 0))}
            "重度污染"->{air.setTextColor(Color.rgb( red: 153, green: 0, blue: 76))}
            "严重污染"->{air.setTextColor(Color.rgb( red: 126, green: 0, blue: 35))}
        }
    }
}
```

处理JSON数据的函数 通过GSON将数据存入数据类
(以处理空气质量JSON数据函数为例)

首页模块

获取城市列表

```
//读取城市列表
private fun readCityMap(){
    val csvFile = context?.assets?.open( fileName: "citylist.csv")
    var line: String?
    val csvSplitBy = ","
    try {
        BufferedReader(InputStreamReader(csvFile)).use { br ->
            // 跳过CSV文件的标题行
            br.readLine()
            while (br.readLine().also { line = it } != null) {
                val data = line!!.split(csvSplitBy).toTypedArray()
                val locationId = data[0]
                val locationName = data[2]
                val province = data[7]
                val cityName = data[9]
                val latitude = data[11]
                val longitude = data[12]
                val city = City(locationId, locationName, province, cityName, latitude, longitude)
                cityMap[locationId] = city
            }
        }
    } catch (e: IOException) {
        e.printStackTrace()
    }
}
```

HomeFragment中读取城市列表csv文件中的数据并存入cityMap中

SelectCityActivity中读取城市列表csv文件中的数据并存入cityList中

```
private fun createCityList(){
    val csvFile = assets.open( fileName: "citylist.csv")
    var line: String?
    val csvSplitBy = ","
    try {
        BufferedReader(InputStreamReader(csvFile)).use { br ->
            // 跳过CSV文件的标题行
            br.readLine()
            while (br.readLine().also { line = it } != null) {
                val data = line!!.split(csvSplitBy).toTypedArray()
                val locationId = data[0]
                val locationName = data[2]
                val province = data[7]
                val cityName = data[9]
                val latitude = data[11]
                val longitude = data[12]
                val city = City(locationId, locationName, province, cityName, latitude, longitude)
                cityList.add(city)
            }
        }
    } catch (e: IOException) {
        e.printStackTrace()
    }
}
```

首页模块

搜索城市

```
private fun searchCity(){
    val searchText = searchEditText.text.toString()
    // 遍历 CityList 根据条件保留部分 City
    for (i in 0 ≤ until < cityList.size) {
        val city = cityList[i]
        if (city.Location_Name.contains(searchText)
            || city.cityName.contains(searchText) || city.Province.contains(searchText)) {
            newCityList.add(city)
        }
    }
}
```

根据搜索框中的文本和cityList中的文本进行匹配，将数据存入newCityList中

用户模块

用户主页

注册广播接收器并更新状态

```
private val loginStatusReceiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        val isLogin = intent?.getBooleanExtra( name: "isLogin",  defaultValue: false) ?: false
        if (isLogin) {
            personalViewModel.updateLoginState( state: "已登录")
            personalViewModel.updateUserImageRes(R.drawable.user)
            personalViewModel.setLoginText("七叔张达")
        }
    }
}
```

```
// 点击头像跳转到Login登录页
val userImage = binding.userImage
userImage.setOnClickListener {  it: View!
    val intent = Intent(requireContext(),Login::class.java)
    startActivity(intent)
}

//点击"点击登录"文本可以跳转到Login登录页
val login = binding.login
login.setOnClickListener {  it: View!
    val intent = Intent(requireContext(),Login::class.java)
    startActivity(intent)
}

//点击"运势"图标可以跳转到运势页
val luckIcon = binding.luckIcon
luckIcon.setOnClickListener {  it: View!
    val intent = Intent(requireContext(),LuckActivity::class.java)
    startActivity(intent)
}
```

设置页面跳转

```
personalViewModel.loginText.observe(viewLifecycleOwner) {  it: String!
    login.text = it
}
personalViewModel.loginState.observe(viewLifecycleOwner) { state ->
    Glide.with( fragment: this@PersonalFragment).load(personalViewModel.userImageRes.value).centerCrop().into(binding.userImage)
    loginState = state
}
```

使用ViewModel更新数据

用户模块

用户主页

```
//点击"开启通知"按钮会进行登录确认，没有登录就会弹出窗口要求登录，登陆了就将选择的checkBox进行提醒
val openMessage = binding.openMessage
openMessage.setOnClickListener {  it: View!
    if (loginState == "已登录") {
        // TODO: 提醒选择的checkBox
        box1Check =checkBox1.isChecked
        box2Check =checkBox2.isChecked
        box3Check =checkBox3.isChecked
        personalViewModel.boxCheckLiveData.value = Triple(box1Check, box2Check, box3Check)
        if (box1Check||box2Check||box3Check){
            Toast.makeText(requireContext(),  text: "已开启", Toast.LENGTH_SHORT).show()
        }else{
            Toast.makeText(requireContext(),  text: "未选择", Toast.LENGTH_SHORT).show()
        }
    } else {
        Toast.makeText(requireContext(),  text: "请先登录", Toast.LENGTH_SHORT).show()
    }
}
```

获取复选框选中状态

根据选中状态进行天气提醒

```
var Weather=HomeFragment().dailyWeather
var dayIndex:Int
if(box1Check){
    if(Weather.maxByOrNull { it.tempMax }?.tempMax!!.toFloat()>40.0){
        val notification = NotificationCompat.Builder(requireContext(),  channelId: "notification")
            .setContentTitle("高温预警").setContentText("未来七天内温度较高，注意降高")
            .setSmallIcon(R.drawable.ic_launcher_foreground)
            .build()
        manager.notify( id: 1,notification)
    }

}
if(box2Check){
    for (i in 0 ..  until  < 7){
        if (Weather[i].textDay.equals("雷阵雨")){
            dayIndex=i+1
            val notification = NotificationCompat.Builder(requireContext(),  channelId: "notification")
                .setContentTitle("暴雨预警").setContentText("未来$dayIndex 有连续暴雨")
                .setSmallIcon(R.drawable.ic_launcher_foreground)
                .build()
            manager.notify( id: 2,notification)
        }
    }
}
if(box3Check){
    if(Weather.minByOrNull { it.tempMin }?.tempMin!!.toFloat()<-6.0) {
        val notification = NotificationCompat.Builder(requireContext(),  channelId: "notification")
            .setContentTitle("低温预警").setContentText("未来七天内存在低温，注意保暖！")
            .setSmallIcon(R.drawable.ic_launcher_foreground)
            .build()
        manager.notify( id: 3,notification)
    }
```

用户模块
用户主页

```
//在其他地方,比如Activity中,可以调用ViewModel的方法改变LiveData值,触发Fragment的UI更新.
class PersonalViewModel : ViewModel() {
    //在ViewModel中定义LiveData用于监听数据变化,这里是_text
    private val _loginText = MutableLiveData<String>().apply {  this: MutableLiveData<String>
        value = "点击登录"
    }
    val loginText: LiveData<String> = _loginText
    fun setLoginText(text: String) {
        _loginText.value = text
    }

    private val _loginState = MutableLiveData<String>().apply {  this: MutableLiveData<String>
        value = "未登录"
    }
    val loginState: LiveData<String> = _loginState
    fun updateLoginState(state: String) {
        _loginState.value = state
    }

    private val _userImageRes = MutableLiveData<Int>().apply {  this: MutableLiveData<Int>
        value = R.drawable.personal
    }
    val userImageRes: LiveData<Int> = _userImageRes
    fun updateUserImageRes(resId: Int) {
        _userImageRes.value = resId
    }

    val boxCheckLiveData = MutableLiveData<Triple<Boolean, Boolean, Boolean>>()
}
```

通过ViewModel监听数据变化

用户模块
登录页

```
loginBtn.setOnClickListener { it: View!
    // 对输入进行合法性检查
    val account:String = accountEdit.text.toString()
    val password:String = passwordEdit.text.toString()
    val isValidAccount:Boolean = (android.util.Patterns.EMAIL_ADDRESS.matcher(account).matches())||(android.util.Patterns.PHONE.matcher(account).matc
    val isValidPassword:Boolean = password.length > 5 && password.length < 13

    if(account.isEmpty() || password.isEmpty()){
        Toast.makeText( context: this, text: "账号或密码为空", Toast.LENGTH_SHORT).show();
    }else if (!isValidAccount || !isValidPassword){
        Toast.makeText( context: this, text: "账号或密码的格式不正确", Toast.LENGTH_SHORT).show()
    }else{
        loginState = "已登录"
        // 发送广播，通知其他组件当前的登录状态已经改变了
        val intent = Intent( action: "LOGIN_STATUS")
        intent.putExtra( name: "isLogin", value: true)
        sendBroadcast(intent)
        finish()
    }
}
}
```

进行合法性检查并发送广播

用户模块

运势页

```
private lateinit var sensorManager: SensorManager
private lateinit var accelerometer: Sensor
private val shakeThreshold = 50 // 定义摇晃的阈值
private var lastShakeTime: Long = 0 // 上一次摇晃的时间
private var mediaPlayer: MediaPlayer? = null // 音效

private lateinit var goodView: TextView
private lateinit var badView: TextView
val date = Calendar.getInstance()  // 获取当前日期
val year = date.get(Calendar.YEAR)  // 年
val month = date.get(Calendar.MONTH) + 1  // 月,因为Calendar月份从0开始所以+1
val day = date.get(Calendar.DATE)  // 日
```

必要参数定义

```
override fun onResume() {
    super.onResume()
    // 注册加速度传感器的监听器
    sensorManager.registerListener( listener: this, accelerometer, SensorManager.SENSOR_DELAY_NORMAL)
}
```

注册加速度传感器监听器

```
override fun onSensorChanged(event: SensorEvent?) {
    event?.let { it: SensorEvent
        val currentTime = System.currentTimeMillis() // 获取当前时间
        if ((currentTime - lastShakeTime) > 1000) { // 限制摇晃的频率
            val x = it.values[0]
            val y = it.values[1]
            val z = it.values[2]
            val acceleration = sqrt( x * x + y * y + z * z) // 计算加速度向量的模长
            if (acceleration > shakeThreshold) { // 如果加速度向量的模长大于阈值，则认为发生了摇晃
                // 更新上一次摇晃的时间
                lastShakeTime = currentTime
                // 播放音效
                mediaPlayer?.start()

                // TODO: 获取今天的运势信息，并更新 UI
                val goodLuck = "乔迁 入学 嫁种 搬公司"
                val badLuck = "结婚 开业 提车 签合同"
                goodView.text = goodLuck
                badView.text = badLuck
            }
        }
    }
}
```

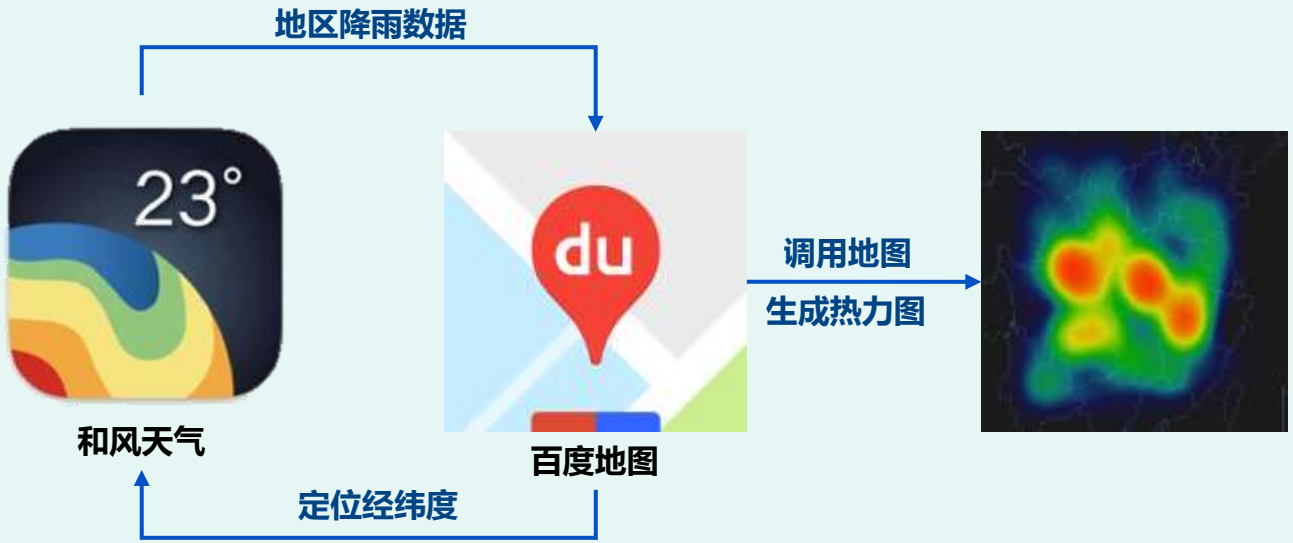
设备摇晃后事件

```
// 获取设备的加速度传感器
sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)

mediaPlayer = MediaPlayer.create( context: this, R.raw.shake_sound)
mediaPlayer?.setOnCompletionListener { mediaPlayer?.reset() }
```

获取加速度传感器和音乐播放器

降雨可视化模块



降雨可视化模块业务逻辑图

service注册定位传感器　　配置key

```
<!-- 百度地图定位服务-->
<service android:name="com.baidu.location.f"
         android:enabled="true"
         android:process=":remote">
</service>
<!--百度地图配置key-->
<meta-data
         android:name="com.baidu.lbsapi.API_KEY"
         android:value="xQx4sfS5dFI7GQzeBknVxdEhPqm0G36K" />
```

导入okhttp、gson和百度地图sdk依赖

```
dependencies {
    ...
     implementation 'com.squareup.okhttp3:okhttp:4.8.1' //http请求
    implementation 'com.google.code.gson:gson:2.8.8' //GSON解析JSON
    implementation files('libs\\BaiduLBS_Android.jar') //百度地图sdk
}//导入了sdk就不用导入其他的百度地图以来了了，sdk基本都包含了，否则会发生冲突
```

本节详细介绍见cyj的个人博客：[Android Studio降雨热力图 | Joel Station (jjuprising.github.io)]

降雨可视化模块

生成地图

初始化地图，设置定位监听器

```
//地图初始化
private fun initloc() {
    //定位初始化
    mLocationClient = LocationClient(requireActivity())
    //通过LocationClientOption设置LocationClient相关参数
    val option = LocationClientOption()
    option.isOpenGps = true // 打开gps
    //设置locationClientOption
    mLocationClient!!.locOption = option
    //注册LocationListener监听器
    val myLocationListener = MyLocationListener()
    mLocationClient!!.registerLocationListener(myLocationListener)
    //开启地图定位图层
    mLocationClient!!.start()
    //设置当前视图位置
    mBaiduMap?.animateMapStatus(preStatus) //动画的方式到中间
}

//定位监听器
inner class MyLocationListener : BDAbstractLocationListener() {
    override fun onReceiveLocation(location: BDLocation) {
        //mapView 销毁后不在处理新接收的位置
        if (location == null || mMapView == null) {
            return
        }
        val locData = MyLocationData.Builder()
        .accuracy(location.radius) // 此处设置开发者获取到的方向信息，顺时针0-360
        .direction(location.direction).latitude(location.latitude)
        .longitude(location.longitude).build()
        mBaiduMap?.setMyLocationData(locData)
        //获取经纬度，并保留两位小数''
        preLatitude = String.format("%.2f", location.latitude).toDouble()
        preLongitude = String.format("%.2f", location.longitude).toDouble()
        //获取当前地址
        preAddress = location.addrStr
        //保证经纬度
        val ll = LatLng(location.latitude, location.longitude)
        //设置位置状态
        preStatus = MapStatusUpdateFactory.newLatLng(ll)
    }
}
```

显示定位按钮事件

```
//点击按钮回到当前位置
val setStatusBtn = binding.setStatusBtn
setStatusBtn.setOnClickListener {
    //设置当前视图位置
    mBaiduMap?.animateMapStatus(preStatus) //动画的方式到中间
}
```

降雨可视化模块

获取降雨数据

构造接收api数据的类

```
//接收api逐小时数据hourly类
data class HourlyData(
    val fxTime: String,
    val temp: String,
    val icon: String,
    val text: String,
    val wind360: String,
    val windDir: String,
    val windScale: String,
    val windSpeed: String,
    val humidity: String,
    val pop: String,
    val precip: String,
    val pressure: String,
    val cloud: String,
    val dew: String
)

//获取降雨概率方法
class CityRainProbability(private val hourlyData: List<HourlyData>) {
    //降雨概率
    fun getHourlyPop(): List<Double> {
        return hourlyData.map { it.pop.toDouble() }
    }
}
```

请求并解析api数据

```
//请求数据
private fun readJSONData(url: String, city: Pair<Double, Double>) {
    thread {
        try {
            val client = OkHttpClient()
            val request = Request.Builder().url(url).build()
            val response = client.newCall(request).execute()
            val responseData = response.body?.string()
            if (responseData != null) {
                dealRainData(responseData, city)
                Log.d("responseData", responseData)
                //                println(responseData)
            }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
}

//提取降雨数据
private fun dealRainData(JsData: String, city: Pair<Double, Double>) {
    val gson = Gson()
    val cityData = gson.fromJson(JsData, JsonObject::class.java)
    val hourlyData =
    gson.fromJson(cityData.getAsJsonArray("hourly"), Array<HourlyData>::class.java).toList()
    val City = CityRainProbability(hourlyData)
    hourlyPop = City.getHourlyPop()// 获取到一个城市24小时天气数据
    val hourlyRainProb = hourlyPop?.toList()
    val thiscity = City(LatLng(city.first, city.second), hourlyRainProb) //单个城市降雨数据
    citiesRain.add(thiscity) //插入城市降雨数据列表
}
```

降雨可视化模块

渲染热力图

将24小时降雨数据构造为24帧数据

```
private fun showheat() {
    // 创建热力图数据
    val builder = HeatMap.Builder()
    // 添加热力图数据点
    //构造24帧数据
    val frames = MutableList(24) { mutableListOf<WeightedLatLng>() }
    // 遍历每个城市
    citiesRain.forEach { city ->
       val hourProbabilities = city.hourlyProbabilities
       // 遍历每个小时的降雨概率
        hourProbabilities?.forEachIndexed { index, probability ->
           val weightedLatLng = WeightedLatLng(city.latLng, probability)
           frames[index].add(weightedLatLng)
        }
     }
    builder.weightedDatas(frames)

}
```

配置热力图，在地图上添加覆盖物

```
private fun showheat(){
    // 设置开始动画属性：开启初始动画，时长100毫秒，动画缓动函数类型为线性
    val init = HeatMapAnimation(true, 100, HeatMapAnimation.AnimationType.Linear)
    // 设置帧动画属性：开启帧动画，时长10000毫秒，动画缓动函数类型为线性
    val frame = HeatMapAnimation(true, 10000, HeatMapAnimation.AnimationType.Linear)
    builder.initAnimation(init)
    builder.frameAnimation(frame)
    // 设置热力图半径范围
    builder.radius(35)
    // 设置热力图渐变颜色
    val colors = intArrayOf(
        Color.rgb(255, 0, 0), Color.rgb(0, 225, 0), Color.rgb(0, 0, 200)
    )
    builder.gradient(Gradient(colors, floatArrayOf(0.2f, 0.5f, 1.0f)))
    builder.maxIntensity(100.0f)
    builder.opacity(0.8)
    val heatMapData = builder.build()
    Log.d("showHeat", "添加覆盖物")
    // 添加热力图覆盖物
    mBaiduMap?.addHeatMap(heatMapData)
    mBaiduMap?.startHeatMapFrameAnimation()
}
```

发现模块

文章数据库

```
class PassageDBHelper(val context: Context, name: String, version: Int) :
    SQLiteOpenHelper(context, name, factory: null, version) {

    private val createPassageDB = "create table Passage (" +
            " id integer primary key autoincrement," +
            "title text," +
            "content text," +
            "picture BLOB)"

    override fun onCreate(db: SQLiteDatabase) {
        db.execSQL(createPassageDB)
```

数据库建立

读取

```
holder.passageTitle.text = passage.title//设置标题
//Log.d("Title",passage.title)
//用id查图片
val db = PassageDBHelper(context, name: "Passage.db", version: 1).writableDatabase
val cursor =
    db.query( table: "Passage", columns: null, selection: "id=?", arrayOf(passage.picture.toString())
val cw = CursorWindow( name: "name", windowSizeBytes: 500000000)
val ac = cursor as AbstractWindowedCursor
ac.window = cw
if (cursor.moveToFirst()) {//查到了
    val array = cursor.getBlob(cursor.getColumnIndex("picture"))//找图片列，得到字节数组
    if(array!=null){
        val bitmap = BitmapFactory.decodeByteArray(array, offset: 0, array.size)//转成bitmap
        Glide.with(context).load(bitmap).into(holder.passageImage)//设置图片
    }
}
cursor.close()
db.close()
var parm = holder.passageImage.layoutParams
parm.height=((200+Math.random()*500).toInt())//随机图片高度
holder.passageImage.layoutParams = parm
```

Adapter

写入

文章（Passage）
PK 文章ID（ID）
文章标题（Title）
文章内容（Content）
图片（Picture）

文章关系模式

```
private fun savePassagetoDB(passage: PassageInfoByte) {//文章数据传入数据库
    val db = dbHelper.writableDatabase
    val values = ContentValues()

    values.put("title", passage.title)
    values.put("content", passage.content)
    values.put("picture", passage.picture)

    db.insert( table: "Passage", nullColumnHack: null, values)
    db.close()
}
```

Post

发现模块

发布页

发布页

调用摄像机/相册

fileprover

上传

```
private fun submitPost(title: String, content: String, Image: Bitmap) {//提交文章动作
    // Convert Bitmap to byte array
    val stream = ByteArrayOutputStream()
    Image.compress(Bitmap.CompressFormat.PNG, quality: 10, stream)//压缩图片
    val byteArray = stream.toByteArray()//转成字节数组
    if (byteArray.size < 100) {
        Toast.makeText( context: this, text: "请上传更大的封面~", duration: 1).show()
    } else {
        val passage = PassageInfoByte(title, content, byteArray)
        savePassagetoDB(passage)
        Toast.makeText( context: this, text: "发布成功", duration: 1).show()
        finish()
    }
}
```

数据校验

```
private fun savePassagetoDB(passage: PassageInfoByte) {//文章数据传入数据库
    val db = dbHelper.writableDatabase
    val values = ContentValues()

    values.put("title", passage.title)
    values.put("content", passage.content)
    values.put("picture", passage.picture)

    db.insert( table: "Passage", nullColumnHack: null, values)
    db.close()
}
```

封装数据并提交

发现模块

发现页

最终呈现

```
initPassage()//初始化文章流

val layoutManager = StaggeredGridLayoutManager( spanCount: 2,StaggeredGridLayoutManager.VERTICAL)
layoutManager.setGapStrategy(StaggeredGridLayoutManager.GAP_HANDLING_NONE)
sallayout = layoutManager
```

设置布局

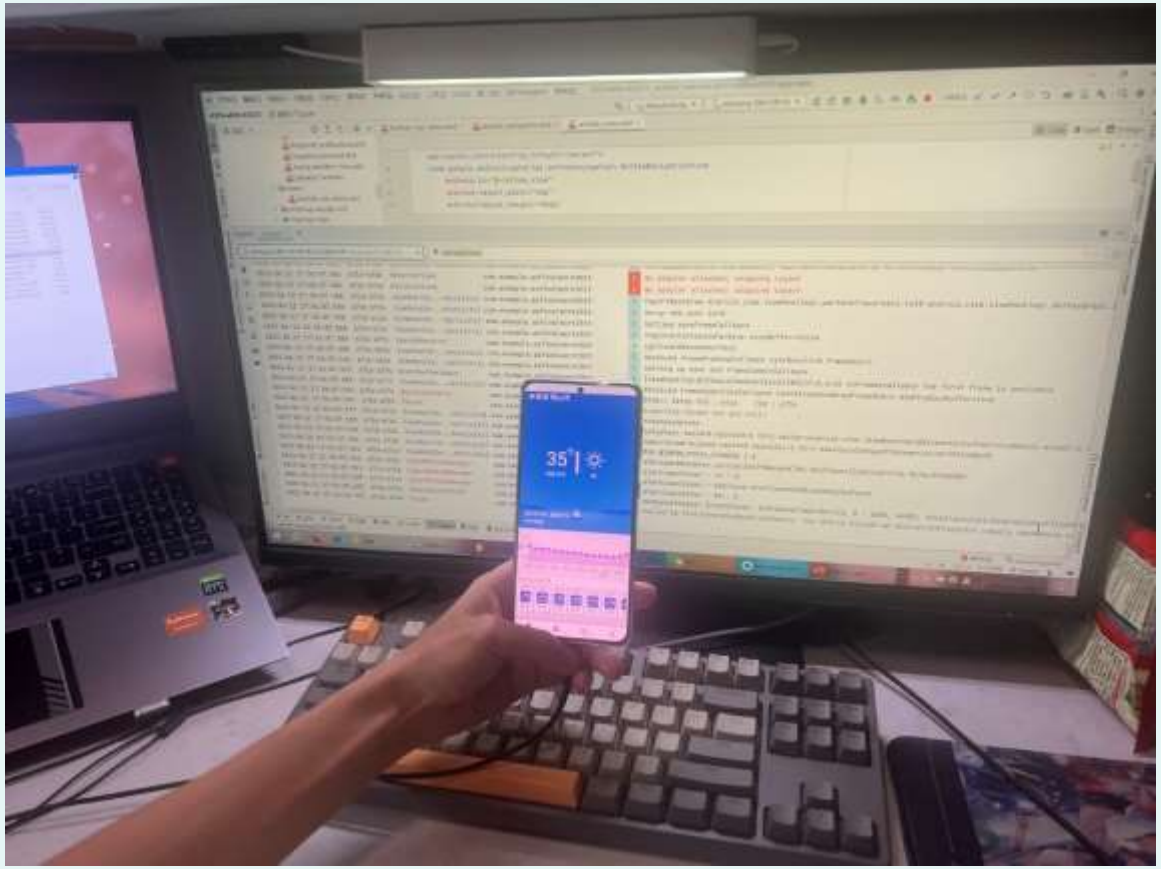
```
holder.passageTitle.text = passage.title//设置标题
//Log.d("Title",passage.title)
//用id查图片
val db = PassageDBHelper(context,  name: "Passage.db",  version: 1).writableDatabase
val cursor =
    db.query( table: "Passage",  columns: null,  selection: "id=?",  arrayOf(passage.picture.toString())
val cw = CursorWindow( name: "name",  windowSizeBytes: 500000000)
val ac = cursor as AbstractWindowedCursor
ac.window = cw
if (cursor.moveToFirst()) {//查到了
    val array = cursor.getBlob(cursor.getColumnIndex("picture"))//找图片列，得到字节数组
    if(array!=null){
        val bitmap = BitmapFactory.decodeByteArray(array,  offset: 0, array.size)//转成bitmap
        Glide.with(context).load(bitmap).into(holder.passageImage);//设置图片
    }
}
cursor.close()
db.close()
var parm = holder.passageImage.layoutParams
parm.height=((200+Math.random()*500).toInt())//随机图片高度
holder.passageImage.layoutParams = parm
```

Adapter中读取数据库数据

安装测试

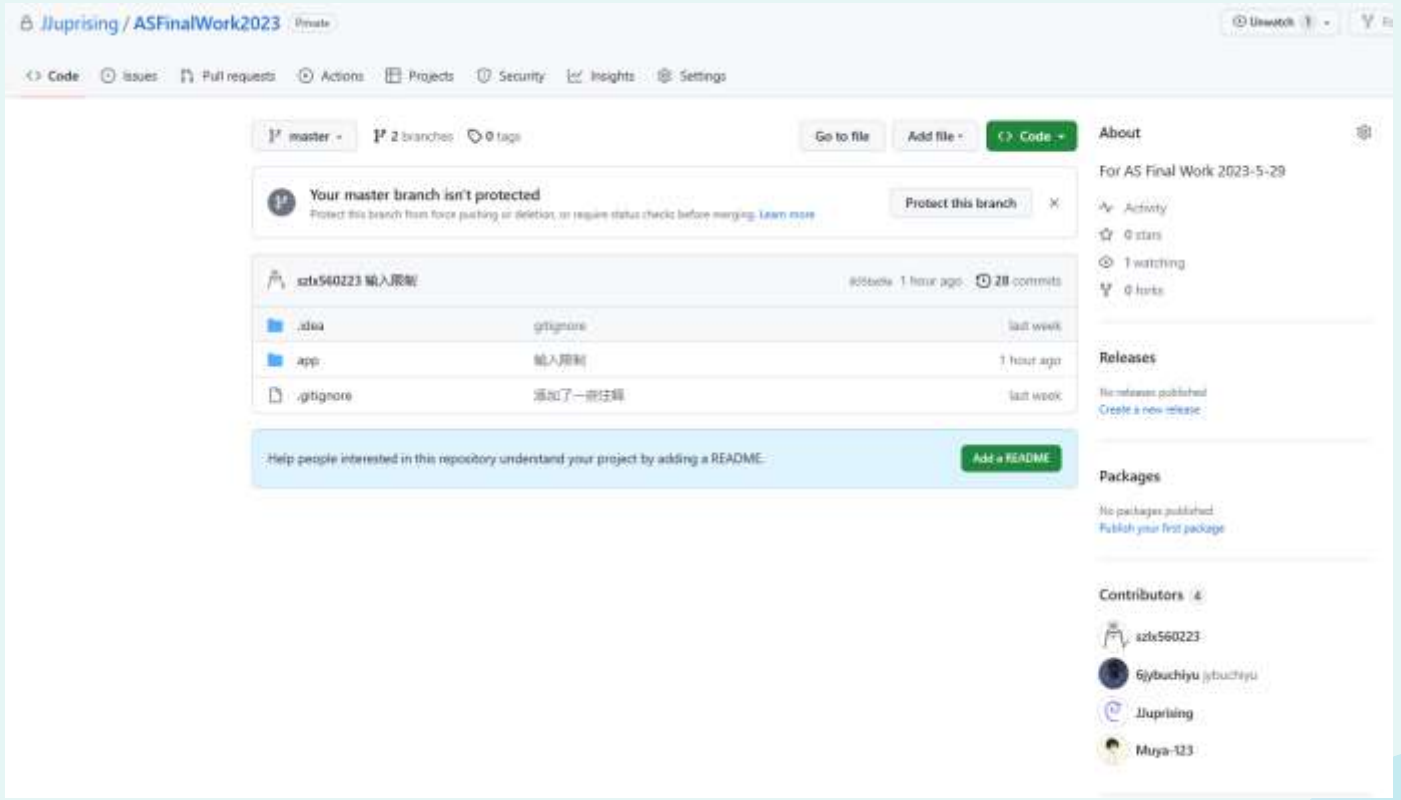
磨蹭天气
Moceng Weather

“磨蹭天气” app在华为P60设备的鸿蒙3.0系统上能够正常编译运行，安装过程流程，程序运行稳定，所有数据能够正常调用和显示，几乎没有卡顿的现象，符合预期效果。



版本控制和仓库管理的使用

本次大作业，我们充分利用了版本控制工具git，并将项目代码托管到了Github仓库中，我们利用Android Studio进行项目开发，使用Github Desktop进行代码上传提交，大大提高了协助开发效率。回溯版本的功能也减少了代码丢失情况



仓库链接：[JJuprising/ASFinalWork2023: For AS Final Work 2023-5-29 (github.com)]

感想与总结

- 本次大作业，我将本学期的大部分安卓知识复盘了一遍，不仅完善了最基本的布局、Acitivity等知识，还补充了调用外部sdk等课外的知识，扎实了自己的安卓开发基础。——cyj

- 在本次大作业的完成过程中，我对本学期学习的知识进行运用，编写布局对页面进行排版，调取API，使用okhttp进行http请求，使用GSON对JSON数据进行处理以及文件的读取操作等。这让我觉得自己学的东西能够被好好运用，收获满满的成就感。——lhq

- 在这次大作业里，我们是选择了一个新的Bottom Navigation Activity项目的，对于Navigation中的Fragment的更新，是我想了很久的问题，最终我选择换个思路，不纠结于Fragment的更新上，才得以继续，并且在Fragment和它对应的ViewModel上，我花了一些时间去理解，慢慢的，我便能炉火纯青地进行数据的同步更新了。学到了很多，也很有成就感。——why

- 通过这次大作业，我完整地复习了这学期所学的大部分知识，并将其实际落地到项目上。除此之外，我也学习了一些课本上没讲到的知识，比如BLOB对象和Bitmap转数组等，总而言之，受益匪浅。——lx

- 总结：后期我们也将完善博客内容，争取能够将项目知识转化为优质的文档内容。我们在开发的过程中遇到了很多的bug，也踩过一些坑，目前网上大部分的资料和文章都是还是基于java的，而基于最新的kotlin还比较少，我们也希望能够提炼我们知识，为安卓开发社区贡献绵薄之力。

感谢观看

请老师批评指正

磨蹭天气
Moceng Weather