东莞理工学院网络空间安全学院

实验报告

课程名称：移动应用开发 学期：2020年秋季

|  |  |  |  |
| --- | --- | --- | --- |
| 实验名称 | 移动应用开发课程设计 | | |
| 组 长 | 林俊杰 201841404119 | 班 级 | 18软卓1班 |
| 组 员 | 蔡俊谦 201841404101 | 指导老师 | 吴宇 |
| 教师评语 |  | 实验成绩 |  |
| 一、实验内容  项目简介:  完成一个课程表的APP，前后端分离，实现以下功能：  管理员(web端)：   * 添加新学期 * 添加学生 * 添加课程 * 添加通知事件(作业也是一种通知形式) * 为移动端APP提供接口   学生用户(移动端)：   * 用户登录、密码修改 * 查看本周课程 * 检索上/下周的课程 * 推送当天课程通知到手机 * 接收管理员发布的通知(作业也是一种形式) * 通知事件的忽略 * 通知事件的添加和删除   **二、需求分析**  **2.1需求分析得到用例图：**    **2.2进行数据库设计：**    **三、所采用的技术实现**  **3.1 技术使用：**  后端：MySql+SSM  移动端：Service+ Fragment+ Retrofit2+AlarmManager+WorkManager+ RecyclerView  +第三方开源课程表控件[TimetableView](https://github.com/zfman/TimetableView)  **3.2 技术实现：**  **3.2.1后端接口**  **用户接口**  @PostMapping(value = "/verify",produces = "application/json;charset=utf-8")  @ResponseBody  public String verify(@RequestParam("num") String num,@RequestParam("pwd")String pwd){}  @PutMapping(value = "/updatePwd",produces = "application/json;charset=utf-8")  @ResponseBody  public String updatePwd(Integer userId,String oldPwd,String newPwd){}  **课程接口**  @GetMapping(value = {"/getCourseOfWeek/{userId}","/getCourseOfWeek/{userId}/{week}"},produces = "application/json;charset=utf-8")  @ResponseBody  public String getCourseOfWeek(@PathVariable("userId") Integer userId,@PathVariable(value = "week",required = false) Integer week){}  @GetMapping(value = "/getCourseOfNextDay/{userId}",produces = "application/json;charset=utf-8")  @ResponseBody  public String getCourseOfNextDay(@PathVariable("userId") Integer userId) {}  **事件接口**  //添加事件{userId:userId，name:name，detail:name，endTime:yyyy-MM-dd，canDelete:1}  @PostMapping(value = "/item",produces = "application/json;charset=utf-8")  @ResponseBody  public String insertItem(Item item) {}    //删除事件  @DeleteMapping(value = "/item/{itemId}", produces = "application/json;charset=utf-8")  @ResponseBody  public String deleteItem(@PathVariable("itemId")Integer itemId){}    //获取最近20件，已完成(isFinished=1)或未完成事件(isFinished=0)  @GetMapping(value = "/items/{userId}/{isFinished}", produces = "application/json;charset=utf-8")  @ResponseBody  public String selectItems(@PathVariable("isFinished")Integer isFinished,@PathVariable("userId")Integer userId){}  //获取明天截止的未标记完成的事件  @GetMapping(value = "/expiringItems/{userId}", produces = "application/json;charset=utf-8")  @ResponseBody  public String selectDeadlineItems(@PathVariable("userId")Integer userId){}  //更改事件的完成状态  @PutMapping(value = "/markFinished/{itemId}/{isFinished}",produces = "application/json;charset=utf-8")  @ResponseBody  public String markFinished(@PathVariable("itemId")Integer itemId,@PathVariable("isFinished")Integer isFinished){}  **3.2.2 网络模块：（以用户接口为例）**  使用retrofit2进行网络请求，retrofit2基于OkHttp并且支持restful风格接口，实现如下  //编写网络请求工具类  **class** RetrofitUtils {  **companion object**{  **fun** getRetrofit():Retrofit=  Retrofit.Builder()  .baseUrl(**"http://8.129.29.84:8080/course/"**)  .addConverterFactory(GsonConverterFactory.create())  .build()  } }  //编写网络api接口,以ItemApi为例  **interface** ItemApi {  @GET(**"items/{userId}/{isFinished}"**)  **fun** selectItems(@Path(**"userId"**) userId:Int,@Path(**"isFinished"**) isFinished:Int):Call<ItemBean>   @GET(**"expiringItems/{userId}"**)  **fun** selectDeadlineItems(@Path(**"userId"**) userId: Int):Call<ItemBean>   @DELETE(**"item/{itemId}"**)  **fun** deleteItem(@Path(**"itemId"**) itemId:Int):Call<ItemBean> @POST(**"item"**)  @FormUrlEncoded  **fun** insertItem(@Field(**"userId"**) userId:Int,@Field(**"name"**)name:String,@Field(**"detail"**)detail:String  ,@Field(**"endTime"**)endTime:String,@Field(**"canDelete"**)canDelete:Int):Call<ItemBean>   @PUT(**"markFinished/{itemId}/{isFinished}"**)  **fun** markFinished(@Path(**"itemId"**) itemId:Int,@Path(**"isFinished"**) isFinished:Int): Call<ItemBean> }  //编写接收数据的bean  **data class** ItemBean(  @SerializedName(**"code"**)  **var code**: Int?,  @SerializedName(**"msg"**)  **var msg**: String?,  @SerializedName(**"data"**)  **var `data`**: List<Data>? ){  **data class** Data(  @SerializedName(**"canDelete"**)  **var canDelete**: Int?,  @SerializedName(**"detail"**)  **var detail**: String?,  @SerializedName(**"endTime"**)  **var endTime**: String?,  @SerializedName(**"isFinished"**)  **var isFinished**: Int?,  @SerializedName(**"itemId"**)  **var itemId**: Int?,  @SerializedName(**"name"**)  **var name**: String?,  @SerializedName(**"userId"**)  **var userId**: Int?  ) }  **3.2.3用户模块**  使用activity和fragment进行页面的跳转  主要代码实现：  登录  **val** api = RetrofitUtils.getRetrofit().create(InfoApi::**class**.*java*) api.verifyInfo(num, pwd)  .enqueue(**object** : Callback<InfoBean> {  **override fun** onResponse(call: Call<InfoBean>, response: Response<InfoBean>) {  response.*let* **{** it **->** it.body()?.*let* **{  if** (**it**.**code** == 200) {  GlobalMsg.**info**= **it**.**data**!! *//登录后把登录信息存入全局变量* **val** intent = Intent(**this**@Login, MainNavigation::**class**.*java*)  intent.addFlags(Intent.*FLAG\_ACTIVITY\_CLEAR\_TASK*)  intent.addFlags(Intent.*FLAG\_ACTIVITY\_NEW\_TASK*)  startActivity(intent)  } **else if** (**it**.**code** == 400) {  Toast.makeText(**this**@Login, **"登录密码错误"**, Toast.*LENGTH\_LONG*).show()   } **else** {  Toast.makeText(**this**@Login, **"网络请求失败，请稍后再试"**, Toast.*LENGTH\_LONG*)  .show()   }  **}  }** }   **override fun** onFailure(call: Call<InfoBean>, t: Throwable) {  Toast.makeText(**this**@Login, **"网络请求失败，请稍后再试！"**, Toast.*LENGTH\_LONG*).show()   }  })  退出：  btnLogout.setOnClickListener **{** *activity*?.*let* **{  val** intent = Intent(**it**, Login::**class**.*java*)  *//退出登录后，栈中的所有activity都会清空，然后跳转到LoginActivity* intent.addFlags(Intent.*FLAG\_ACTIVITY\_CLEAR\_TASK*)  intent.addFlags(Intent.*FLAG\_ACTIVITY\_NEW\_TASK*)  **it**.startActivity(intent)  **} }**  **3.2.4 通知事件模块**  使用RecyclerView实现事件的展示添加和删除，使用WorkManager配合PeriodicWorkRequest实现截止通知的推送展示。  核心代码：  **private fun** createNotificationChannel() {  **if** (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*O*) {  **val** name = **"my channel no.1"  val** descriptionText = **"my channel descriptionText no.1"  val** importance = NotificationManager.*IMPORTANCE\_DEFAULT* **val** channel = NotificationChannel(*CHANNEL\_ID*, name, importance).*apply* **{** *description* = descriptionText  **}  val** notificationManager: NotificationManager =  getSystemService(*NOTIFICATION\_SERVICE*) **as** NotificationManager  notificationManager.createNotificationChannel(channel)  } }  **private fun** makeNoticeWorker(){  createNotificationChannel() *//创建channelId用于worker中创建通知* **val** constrain = Constraints.Builder()  .setRequiredNetworkType(NetworkType.*CONNECTED*)  .build()  **val** oneTimeWorkRequest = *OneTimeWorkRequestBuilder*<NotificationWorker>()  .setConstraints(constrain)  .build()  **val** periodicWorkRequest =  *PeriodicWorkRequestBuilder*<NotificationWorker>(15,TimeUnit.*MINUTES*)  .setConstraints(constrain)  .build()  **workManager**.enqueue(periodicWorkRequest) }  **val** intent = Intent(  *applicationContext*,  MainNavigation::**class**.*java* ).*apply* **{** *flags* =  Intent.*FLAG\_ACTIVITY\_NEW\_TASK* **} val** pendingIntent: PendingIntent =  PendingIntent.getActivity(  *applicationContext*,  0,  intent,  Intent.*FLAG\_ACTIVITY\_NEW\_TASK* ) **val** builder =  NotificationCompat.Builder(*applicationContext*, *CHANNEL\_ID*)  .setSmallIcon(R.drawable.*ic\_notifications\_black\_24dp*)  .setContentTitle(**"截止提醒"**)  .setContentText(noticeName)  .setPriority(NotificationCompat.*PRIORITY\_DEFAULT*)  .setContentIntent(pendingIntent)  .setAutoCancel(**true**) *with*(NotificationManagerCompat.from(*applicationContext*)) **{**notify(*NOTIFICATION\_ID\_OF\_ITEM*, builder.build()) **}**  **3.2.5课程表模块：**  **课程表信息：使用[TimetableView](https://github.com/zfman/TimetableView" \t "_blank)（一款开源、完善、高效的Android课程表控件）展示课程表信息，课程表推送通知：使用Service+AlarmManager用轮询的方式实现课程表的推送**    **@RequiresApi(Build.VERSION\_CODES.*O*) override fun onCreate() {  //初始化通知栏配置  createNotificationChannel() }  override fun onStart(intent: Intent?, startId: Int) {  PollingThread().start() }  override fun onDestroy() {  super.onDestroy()  *println*("Service:onDestroy") }  */\*\*  \* Polling thread  \* 模拟向Server轮询的异步线程  \*/* internal inner class PollingThread : Thread() {  @RequiresApi(Build.VERSION\_CODES.*KITKAT*)  override fun run() {  //网络请求是否有推送信息  getMessage()   } }  //获取推送信息 private fun getMessage(){  val api = RetrofitUtils.getRetrofit().create(CourseApi::class.*java*)  api.getCourseOfNextDay(GlobalMsg.info.userId!!)  .enqueue(object : Callback<CourseBean> {  override fun onResponse(call: Call<CourseBean>, response: Response<CourseBean>) {  response.*let* { it ->  it.body()?.*let* {  if (it.code == 200) {  Log.d("jj",it.toString())  title = "课程提醒"  nextSubjects = it.data  bigText = jointString()  count++  *println*("轮询次数为${count}")  //实际每隔一分钟轮询一次，程序已每隔30次即30分钟推送  if(count == 480) {  //推送到通知栏  createNotification()  count = 0  }  }else if(it.code == 204){  title = "课程提醒"  bigText = "明天没有课程"  count++  *println*("轮询次数为${count}")  //实际每隔一分钟轮询一次，程序已每隔30次即30分钟推送  if(count == 480){  //推送到通知栏  createNotification()  count=0  }  }  else {  return  }  }  }  }   override fun onFailure(call: Call<CourseBean>, t: Throwable) {  return  }  })  }  fun jointString(): String {  var string:String = "明天课程有："  for(item:Subject in nextSubjects!!){  string += item.*courseName* }  return string }  @SuppressLint("WrongConstant") fun createNotification() {    // Create an explicit intent for an Activity in your app  val intent = Intent(*application*, MainNavigation::class.*java*).*apply* {  *flags* = Intent.*FLAG\_ACTIVITY\_NEW\_TASK* }  val pendingIntent: PendingIntent = PendingIntent.getActivity(*application*, 0, intent, Intent.*FLAG\_ACTIVITY\_NEW\_TASK*)    val builder = CHANNEL\_CODE?.*let* {  NotificationCompat.Builder(*application*, it)  .setSmallIcon(R.mipmap.*ic\_launcher\_round*)  .setContentTitle(title!!)  .setContentText(bigText!!)  .setStyle(  NotificationCompat.BigTextStyle()  .bigText(bigText!!)  )  .setPriority(NotificationCompat.*PRIORITY\_DEFAULT*)  // Set the intent that will fire when the user taps the notification  .setContentIntent(pendingIntent)  .setAutoCancel(true)  }   *with*(NotificationManagerCompat.from(*application*)) {  // notificationId is a unique int for each notification that you must define  notify(NOTIFICATION\_ID, builder!!.build())  } }   private fun createNotificationChannel() {  // Create the NotificationChannel, but only on API 26+ because  // the NotificationChannel class is new and not in the support library  if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*O*) {  val name = "test\_channel"  val descriptionText = "this is my test channel"   //other Importance types are  //IMPORTANCE\_HIGH  //IMPORTANCE\_LOW  //IMPORTANCE\_MAX  //IMPORTANCE\_MIN  val importance = *IMPORTANCE\_HIGH* //define your own channel code here i used a predefined constant  mNotificationChannel = NotificationChannel(CHANNEL\_CODE, name, importance).*apply* {  *description* = descriptionText  }   // Register the channel with the system  // I am using application class's context here  mManager = *application*.getSystemService(Context.*NOTIFICATION\_SERVICE*) as NotificationManager  mManager!!.createNotificationChannel(mNotificationChannel!!)  } }  companion object {  const val ACTION = "com.example.timetable.PollingService" }**  //开启轮询服务  @RequiresApi(Build.VERSION\_CODES.KITKAT)  fun startPollingService(context: Context, seconds: Int, cls: Class<\*>?, action: String?) {  //获取AlarmManager系统服务  val manager = context.getSystemService(Context.ALARM\_SERVICE) as AlarmManager  //包装需要执行Service的Intent  val intent = Intent(context, cls)  intent.action = action  val pendingIntent = PendingIntent.getService(  context, 0,  intent, PendingIntent.FLAG\_UPDATE\_CURRENT  )  //触发服务的起始时间  val triggerAtTime = SystemClock.elapsedRealtime()  //使用AlarmManger的setRepeating方法设置定期执行的时间间隔（seconds秒）和需要执行的Service  print("初始时间"+triggerAtTime)  print("间隔为"+seconds\*1000.toLong())  manager.setRepeating(  AlarmManager.ELAPSED\_REALTIME, triggerAtTime+5000,  seconds \* 10000.toLong(), pendingIntent  )  }  //停止轮询服务  fun stopPollingService(context: Context, cls: Class<\*>?, action: String?) {  val manager = context  .getSystemService(Context.ALARM\_SERVICE) as AlarmManager  val intent = Intent(context, cls)  intent.action = action  val pendingIntent = PendingIntent.getService(  context, 0,  intent, PendingIntent.FLAG\_UPDATE\_CURRENT  )  //取消正在执行的服务  manager.cancel(pendingIntent)  }   1. **实验的主要功能模块**   **4.1.服务端功能模块实现：**          **4.2.移动端功能模块实现**  4.2.1登录：    4.2.2我的    4.2.3密码修改：    4.2.4：课程信息    4.4.5 事件详情      4.4.6消息提醒：    **五、实验总结**  通过这次课程实验，对于巩固自身开发基础和熟练运用所学知识起到了极大的作用，让我们了解了git协作开发，服务器部署，jenkins持续集成，kotlin语言开发、retrofit2网络模块的使用，以及熟悉了fragment、service、notification、alarmManager、workManager等其它安卓开发的技术点。不足之处是对于很多技术的使用也存在一知半解，部分需求实现不如人意，例如定时从服务器获取信息并推送到手机上，推送时间无法实际把控；API版本更新迭代问题，例如推送功能的时钟AlarmManager的setRepeating的方法经过安卓API版本的迭代出现时间间隔不准确的问题。 | | | |