

系统开发报告

报告题目： To-Do List

智能任务规划APP

课 程： 移动智能开发

指导老师： 曹 阳

小组成员： 何子亨、陈浩

黎炜堂、黄润

学 院： 计算机学院

# 一、产品设计

## 1.项目可行性分析

通过调研发现,日益繁忙的生活带来的任务管理问题日益突出。传统的纸质备忘录存在信息收集、分类、查找不便等问题。移动设备日益普及,应用软件日趋成熟,基于APP应用的任务管理解决方案具有明显优势。消费者对智能化日渐增长,商户及企业对效率的追求也不断提高。

市场上已有多款TASK APP,但多数只满足基本需求而缺少差异化点。与传统任务管理软件相比,APP产品交互方式更便捷,且可集成互联网功能及多端联动。

根据趋势分析,TASK APP具备可观市场:

1)全球智能设备用户量持续增长,带动APP需求;

2)越来越多的企业和个人需要提高工作效率和生产力。

综上,基于Android开发的智能To-Do List具备可行性。

## 2. 产品定位和目标

经调研并综合用户需求,定位To-Do List为一款智能高效的工作管理助手。目标用户为20-45岁上班族群体,帮助他们实现"有条不紊的效率生活"。

主要目标:

1) 提供多项智能功能如计划模式、提醒监督、统计分析等,满足复杂任务管理;

2)提供多终端支持,随时随地管理任务;

3)优化UI交互体验,即使繁忙也易于使用;

4)丰富定制功能如黑名单/宽限期设置等,让用户更多管理选择。

定位为"优雅高效"的TASK APP应用:

1)表面上的优雅:简洁现代的UI界面;

2)内在的高效:智能化的任务计划与执行。

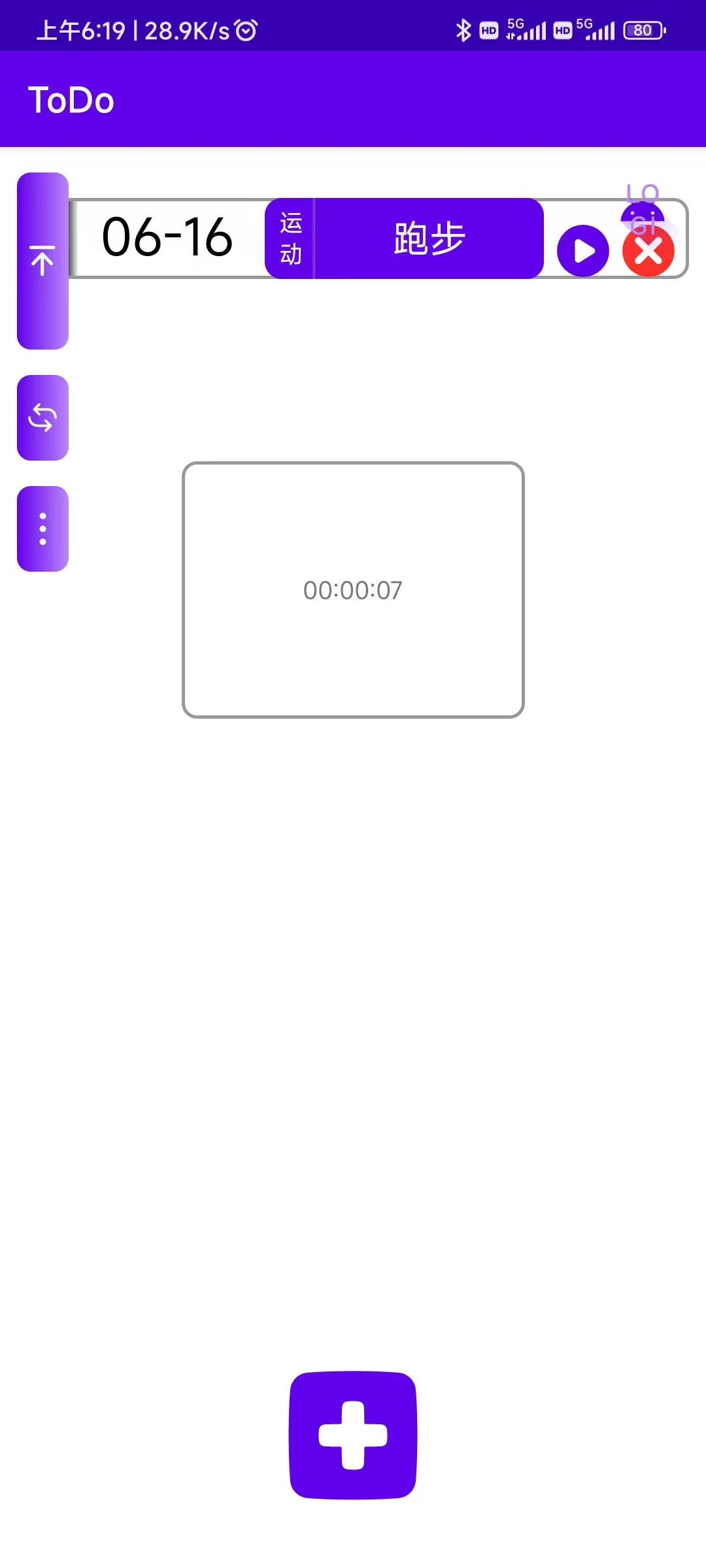
**3. 产品内容策划**

1. 新建任务:输入任务名称、描述、标签、优先级、截止日期、 estimate时间等。
2. 多种任务分类:重点事项、待办事项、计划项目、日程安排、重复事项、已完成任务、黑名单等。
3. 计划模式:按周、月、季度设置任务计划。
4. 提前提醒:Email、短信、应用内、桌面闹钟等方式,按倒数时间提醒。
5. 统计分析:各类型任务数量,任务完成率、平均用时等。
6. 黑名单:超出宽限期自动列入黑名单,直至完成。
7. 项目 management:项目采Agile工作方式,卡片管理子任务。
8. 时间轴:直观呈现任务安排和时间节点。
9. 多设备联动:手机、平板、PC端多端支持,数据实时同步。
10. 云端备份:自动云端备份,恢复数据。
11. 定时任务:通过WorkManager在指定时间自动执行。
12. 标签管理:自定义标签,快速定位相关任务。
13. 支持导入任务:导入外部任务文件。

# 二、产品实现

## 1.各界面UI及实现代码逻辑

### 1.1主页面



主要功能:

* 显示TODO列表,分为多个列表(重点事项、重复事项、其他事项)
* 新建TODO事项,设置类型、名称、开始时间、结束时间
* 每个TODO事项包含开始时间、结束时间、类型、名称、完成按钮、删除按钮
* 完成按钮可以计时,记录TODO操作时间
* 删除按钮可以删除该TODO事项

主要技术点:

* 使用SQLite数据库存储TODO数据
* 使用Parcelable实现数据传递
* 使用Handler实现计时功能
* ValueAnimator实现TODO列表高度变化动画

主要代码逻辑:

* 定义TODO的相关数据类Todo和AllToDo
* AppData对象存储全局数据
* 读取assets数据初始化TODO全局数据
* viewTodo()方法根据TODO类型显示到对应列表
* pushTodo()方法添加TODO项至列表,并使用动画
* getEntry()方法构建一个TODO项View
* 点击完成按钮实现计时功能
* 点击删除按钮从列表和数据库中删除当前TODO项

data class Time(var data : Long)

data class AllToDo(

@SerializedName("top") var top : Array<Todo>,

@SerializedName("rep") var rep : Array<Todo>,

@SerializedName("other") var other : Array<Todo>) {

fun push(todo : AppData.Todo) {

if (todo.isTop) this.top.plus(todo)

else if (todo.isRep) this.rep.plus(todo)

else this.other.plus(todo)

}

fun del(todo: AppData.Todo) {

if (todo.isTop) {

for (i in this.top) {

if (i == todo) {

this.top = this.top.filterIndexed { index, \_ -> index != this.top.indexOf(i) }.toTypedArray()

}

}

} else if (todo.isRep) {

for (i in this.rep) {

if (i == todo) {

this.rep = this.rep.filterIndexed { index, \_ -> index != this.rep.indexOf(i) }.toTypedArray()

}

}

} else {

for (i in this.other) {

if (i == todo) {

this.other = this.other.filterIndexed { index, \_ -> index != this.other.indexOf(i) }.toTypedArray()

}

}

}

}

}

data class Todo(

@SerializedName("name") var name: String,

@SerializedName("type") var type : Int,

@SerializedName("startTime") var startTime: Long,

@SerializedName("endTime") var endTime: Long,

@SerializedName("isRep") var isTop : Boolean,

@SerializedName("isTop") var isRep : Boolean) : Parcelable {

@RequiresApi(Build.VERSION\_CODES.Q)

constructor(parcel: Parcel) : this(

parcel.readString()!!,

parcel.readInt(),

parcel.readLong(),

parcel.readLong(),

parcel.readBoolean(),

parcel.readBoolean()

)

@RequiresApi(Build.VERSION\_CODES.Q)

override fun writeToParcel(parcel: Parcel, flags: Int) {

parcel.writeString(name)

parcel.writeInt(type)

parcel.writeLong(startTime)

parcel.writeLong(endTime)

parcel.writeBoolean(isTop)

parcel.writeBoolean(isRep)

}

override fun describeContents(): Int {

return 0

}

companion object CREATOR : Parcelable.Creator<Todo> {

@RequiresApi(Build.VERSION\_CODES.Q)

override fun createFromParcel(parcel: Parcel): Todo {

return Todo(parcel)

}

override fun newArray(size: Int): Array<Todo?> {

return arrayOfNulls(size)

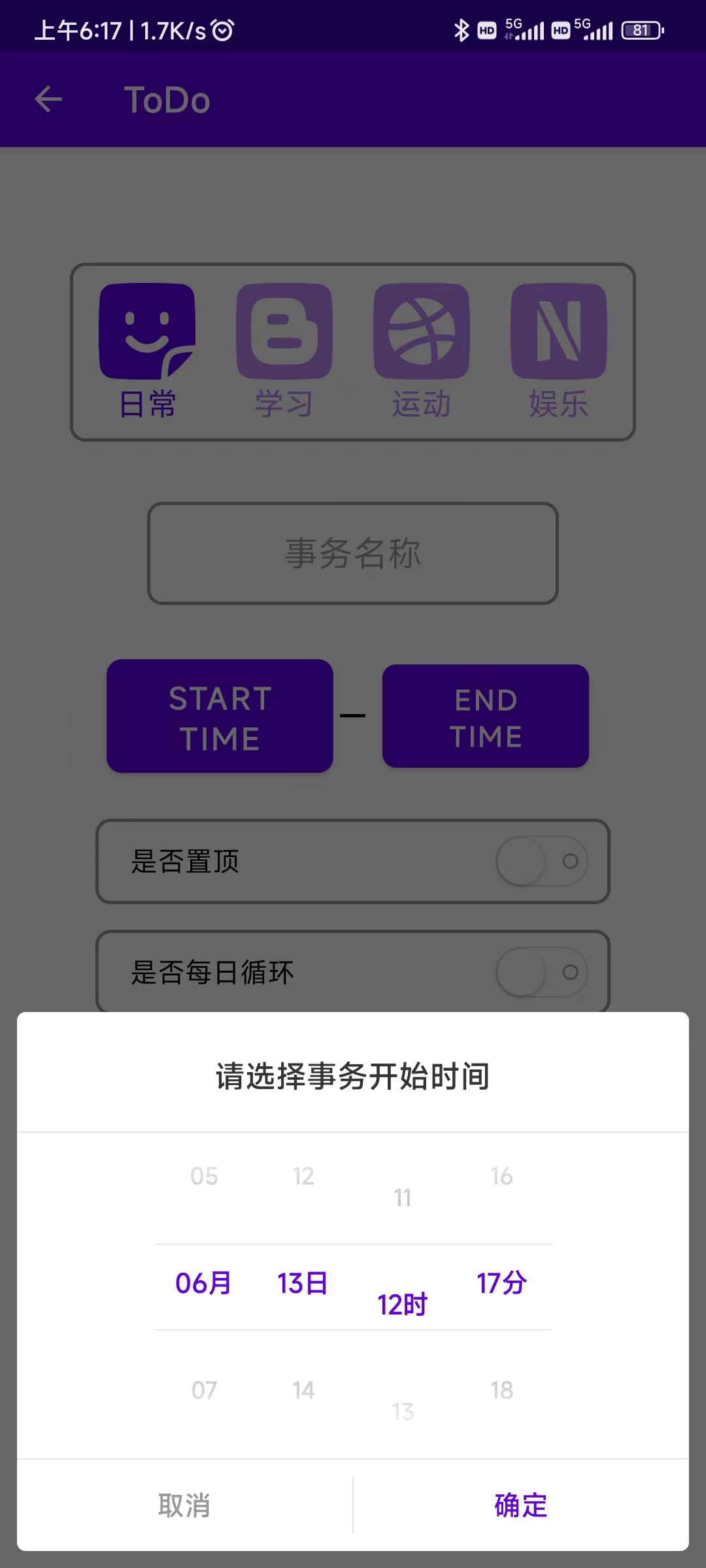
}

}

}

lateinit var global: AllToDo

### 1.2 新任务添加页面



主要功能:

添加任务时可以选择开始时间和结束时间,并保存到数据类Todo中

添加任务时可以设置任务类型,是否重点或重复任务

添加任务成功后通过Intent返回结果数据给前一个Activity

使用ValueAnimator实现Switch和时间选择按钮的背景颜色变化动画

使用ShaderDrawable和LayerDrawable实现Switch状态变化时的渐变背景效果

使用ScaleAnimation实现按钮和RadioGroup选项的缩放动画

根据输入内容校验检查添加任务的数据是否合法,不合法时给出Toast提示

重置EditText焦点和软键盘,实现更好的用户体验

使用字符插值器实现时间选择按钮文字从旧值到新值的平滑变化

val startTime = AppData.Time(-1)

val endTime = AppData.Time(-1)

val radioGroup = findViewById<RadioGroup>(R.id.radioGroup)

val linearLayout3 = findViewById<LinearLayout>(R.id.linearLayout3)

val linearLayout4 = findViewById<LinearLayout>(R.id.linearLayout4)

val textView2 = findViewById<TextView>(R.id.textView2)

val textView3 = findViewById<TextView>(R.id.textView3)

val switch = findViewById<com.suke.widget.SwitchButton>(R.id.switch1)

val switch2 = findViewById<com.suke.widget.SwitchButton>(R.id.switch2)

animateSwitch(switch, linearLayout3, textView2)

animateSwitch(switch2, linearLayout4, textView3)

// clear editText focus

val editText = findViewById<EditText>(R.id.editText)

val back = findViewById<View>(R.id.back)

back.setOnClickListener {

clearEditTextFocus(editText)

}

// radioButton animation

val radioButton = findViewById<RadioButton>(R.id.radioButton)

val radioButton2 = findViewById<RadioButton>(R.id.radioButton2)

val radioButton3 = findViewById<RadioButton>(R.id.radioButton3)

val radioButton4 = findViewById<RadioButton>(R.id.radioButton4)

val checkedColor = ContextCompat.getColor(

this, R.color.purple\_500)

val uncheckedColor = ContextCompat.getColor(this, R.color.purple\_200)

animateRadioButton(radioButton, checkedColor, uncheckedColor)

animateRadioButton(radioButton2, checkedColor, uncheckedColor)

animateRadioButton(radioButton3, checkedColor, uncheckedColor)

animateRadioButton(radioButton4, checkedColor, uncheckedColor)

// time picker dialog and text&color animation

val button3 = findViewById<Button>(R.id.button3)

val button4 = findViewById<Button>(R.id.button4)

animateTimePickerButton(button3, "请选择事务开始时间", startTime)

animateTimePickerButton(button4, "请选择事务结束时间", endTime)

//

val data = intent.getStringExtra("extra\_data")

if (data != null) {

Log.d("SecondActivity", data)

}

val button2 = findViewById<Button>(R.id.button2)

button2.setOnClickListener {

var type = radioGroup.indexOfChild(findViewById<RadioButton>(radioGroup.checkedRadioButtonId))

val name = editText.text.toString()

val isTop = switch.isChecked

val isRep = switch2.isChecked

if (type > 5) type = 1

val todo = AppData.Todo(name, type, startTime.data, endTime.data, isTop, isRep)

if (checkTodo(todo)) {

Log.d("2222222222", todo.toString())

val intent = Intent()

intent.putExtra("todo", todo)

setResult(RESULT\_OK, intent)

finish()

}

}

}

// add animation to RadioButton

private fun animateRadioButton(radioButton: RadioButton,

checkedColor: Int, uncheckedColor: Int,

checkedSize: Float = 1.15f, uncheckedSize: Float = 1f) {

radioButton.setOnCheckedChangeListener { buttonView, isChecked ->

val editText = findViewById<EditText>(R.id.editText)

clearEditTextFocus(editText)

val colorAnimator = ValueAnimator.ofObject(ArgbEvaluator(),

if (isChecked) uncheckedColor else checkedColor,

if (isChecked) checkedColor else uncheckedColor)

colorAnimator.duration = 300

colorAnimator.addUpdateListener {

buttonView.setTextColor(it.animatedValue as Int)

buttonView.compoundDrawables[1].colorFilter =

PorterDuffColorFilter(it.animatedValue as Int, PorterDuff.Mode.SRC\_IN)

}

colorAnimator.start()

val scaleAnimation = ScaleAnimation(if (isChecked) uncheckedSize else checkedSize,

if (isChecked) checkedSize else uncheckedSize,

if (isChecked) uncheckedSize else checkedSize,

if (isChecked) checkedSize else uncheckedSize,

Animation.RELATIVE\_TO\_SELF, 0.5f, Animation.RELATIVE\_TO\_SELF, 0.5f)

scaleAnimation.duration = 200

scaleAnimation.fillAfter = true

buttonView.startAnimation(scaleAnimation)

}

}

// add animation to TimePickerButton

private fun animateTimePickerButton(button : Button, title : String, time: AppData.Time) {

button.setOnClickListener {

val editText = findViewById<EditText>(R.id.editText)

clearEditTextFocus(editText)

val scaleAnimation = ScaleAnimation(

1f, 1.1f, // 水平方向的起始和结束缩放比例

1f, 1.1f, // 垂直方向的起始和结束缩放比例

Animation.RELATIVE\_TO\_SELF, 0.5f, // 缩放中心点的 x 坐标

Animation.RELATIVE\_TO\_SELF, 0.5f // 缩放中心点的 y 坐标

).apply {

duration = 200 // 动画持续时间，单位为毫秒

interpolator = AccelerateDecelerateInterpolator() // 加速减速插值器

fillAfter = true // 动画结束后保持最终状态

}

button.startAnimation(scaleAnimation)

CardDatePickerDialog.builder(this)

.setTitle(title)

.setThemeColor(Color.parseColor("#6200EE"))

.setDisplayType(DateTimeConfig.MONTH, DateTimeConfig.DAY, DateTimeConfig.HOUR, DateTimeConfig.MIN)

.showBackNow(false)

.showFocusDateInfo(false)

.setOnChoose {millisecond->

time.data = millisecond

val sdf = SimpleDateFormat("MM-dd\nHH:mm", Locale.getDefault())

val dateString: String = sdf.format(Date(millisecond))

val oldValue = button.text.toString()

val textAnimator = ValueAnimator.ofFloat(0f, 1f)

textAnimator.duration = 200

textAnimator.addUpdateListener { valueAnimator ->

val fraction = valueAnimator.animatedFraction

val text = if (fraction == 1f) dateString else interpolate(

oldValue,

dateString,

fraction

)

button.text = text

}

val originalColor = button.currentTextColor

val colorAnimator = ObjectAnimator.ofInt(button, "textColor", originalColor, getRandomColor())

colorAnimator.duration = 200

colorAnimator.repeatCount = 1

colorAnimator.repeatMode = ValueAnimator.REVERSE

colorAnimator.setEvaluator(ArgbEvaluator())

colorAnimator.doOnEnd {

button.setTextColor(originalColor)

}

val animatorSet = AnimatorSet()

animatorSet.playTogether(colorAnimator, textAnimator)

animatorSet.start()

val sizeAnimation = ScaleAnimation(

1f, 0.92f, // 水平方向的起始和结束缩放比例

1f, 0.92f, // 垂直方向的起始和结束缩放比例

Animation.RELATIVE\_TO\_SELF, 0.5f, // 缩放中心点的 x 坐标

Animation.RELATIVE\_TO\_SELF, 0.5f // 缩放中心点的 y 坐标

).apply {

duration = 200 // 动画持续时间，单位为毫秒

interpolator = AccelerateDecelerateInterpolator() // 加速减速插值器

fillAfter = true // 动画结束后保持最终状态

}

button.startAnimation(sizeAnimation)

}.build().show()

}

}

private fun clearEditTextFocus(editText: EditText) {

editText.clearFocus()

val inputMethodManager = getSystemService(Context.INPUT\_METHOD\_SERVICE) as InputMethodManager

inputMethodManager.hideSoftInputFromWindow(editText.windowToken, 0)

}

private fun interpolate(oldValue: String, newValue: String, fraction: Float): String {

val oldLength = oldValue.length

val newLength = newValue.length

val length = if (oldLength > newLength) oldLength else newLength

val builder = StringBuilder(length)

for (i in 0 until length) {

val oldChar = if (i < oldLength) oldValue[i] else ' '

val newChar = if (i < newLength) newValue[i] else ' '

builder.append(interpolateChar(oldChar, newChar, fraction))

}

return builder.toString()

}

private fun interpolateChar(oldChar: Char, newChar: Char, fraction: Float): Char {

val oldInt = oldChar.code

val newInt = newChar.code

val interpolatedInt = (oldInt + fraction \* (newInt - oldInt)).toInt()

return interpolatedInt.toChar()

}

private fun getRandomColor(): Int {

val random = Random()

// 生成 0 到 255 之间的随机数

val r = random.nextInt(256)

val g = random.nextInt(256)

val b = random.nextInt(256)

// 将随机生成的颜色值转换为 ARGB 格式

return Color.argb(255, r, g, b)

}

private fun checkTodo(todo: AppData.Todo):Boolean {

var f = true

if (todo.name.isEmpty()) {

Toast.makeText(applicationContext, "请输入事务名称", Toast.LENGTH\_SHORT).show()

f = false

}

if (todo.startTime == -1L) {

Toast.makeText(applicationContext, "请选择开始时间", Toast.LENGTH\_SHORT).show()

f = false

}

if (todo.endTime == -1L) {

Toast.makeText(applicationContext, "请选择结束时间", Toast.LENGTH\_SHORT).show()

f = false

}

return f

}

// add animation to Switch

private fun animateSwitch(switch : com.suke.widget.SwitchButton,

linearLayout : LinearLayout, textView: TextView) {

val originalBackground = linearLayout.background

switch.setOnCheckedChangeListener { \_, isChecked ->

val startColor = Color.WHITE // 起始颜色

val endColor = 0xFF6200EE.toInt() // 结束颜色

val gradientDrawable = GradientDrawable(

GradientDrawable.Orientation.RIGHT\_LEFT,

intArrayOf(startColor, startColor)

) // 创建渐变背景

gradientDrawable.cornerRadius = 25f // 设置圆角

val layerDrawable = LayerDrawable(

arrayOf(gradientDrawable, originalBackground)

) // 创建层叠背景

if (isChecked) {

val backgroundAnimator = ValueAnimator.ofObject(

ArgbEvaluator(),

startColor,

endColor

).apply {

duration = 300 // 动画持续时间，单位为毫秒

addUpdateListener { animator ->

val color = animator.animatedValue as Int

gradientDrawable.colors = intArrayOf(startColor, color)

linearLayout.background = layerDrawable

}

}

backgroundAnimator.start()

val textAnimator = ValueAnimator.ofObject(

ArgbEvaluator(),

Color.BLACK,

Color.WHITE

).apply {

duration = 200 // 动画持续时间，单位为毫秒

addUpdateListener { animator ->

val color = animator.animatedValue as Int

textView.setTextColor(color)

}

}

textAnimator.start()

} else {

val backgroundAnimator = ValueAnimator.ofObject(

ArgbEvaluator(),

endColor,

startColor

).apply {

duration = 300 // 动画持续时间，单位为毫秒

addUpdateListener { animator ->

val color = animator.animatedValue as Int

gradientDrawable.colors = intArrayOf(startColor, color)

linearLayout.background = layerDrawable

}

}

backgroundAnimator.start()

val textAnimator = ValueAnimator.ofObject(

ArgbEvaluator(),

Color.WHITE,

Color.BLACK

).apply {

duration = 200 // 动画持续时间，单位为毫秒

addUpdateListener { animator ->

val color = animator.animatedValue as Int

textView.setTextColor(color)

}

}

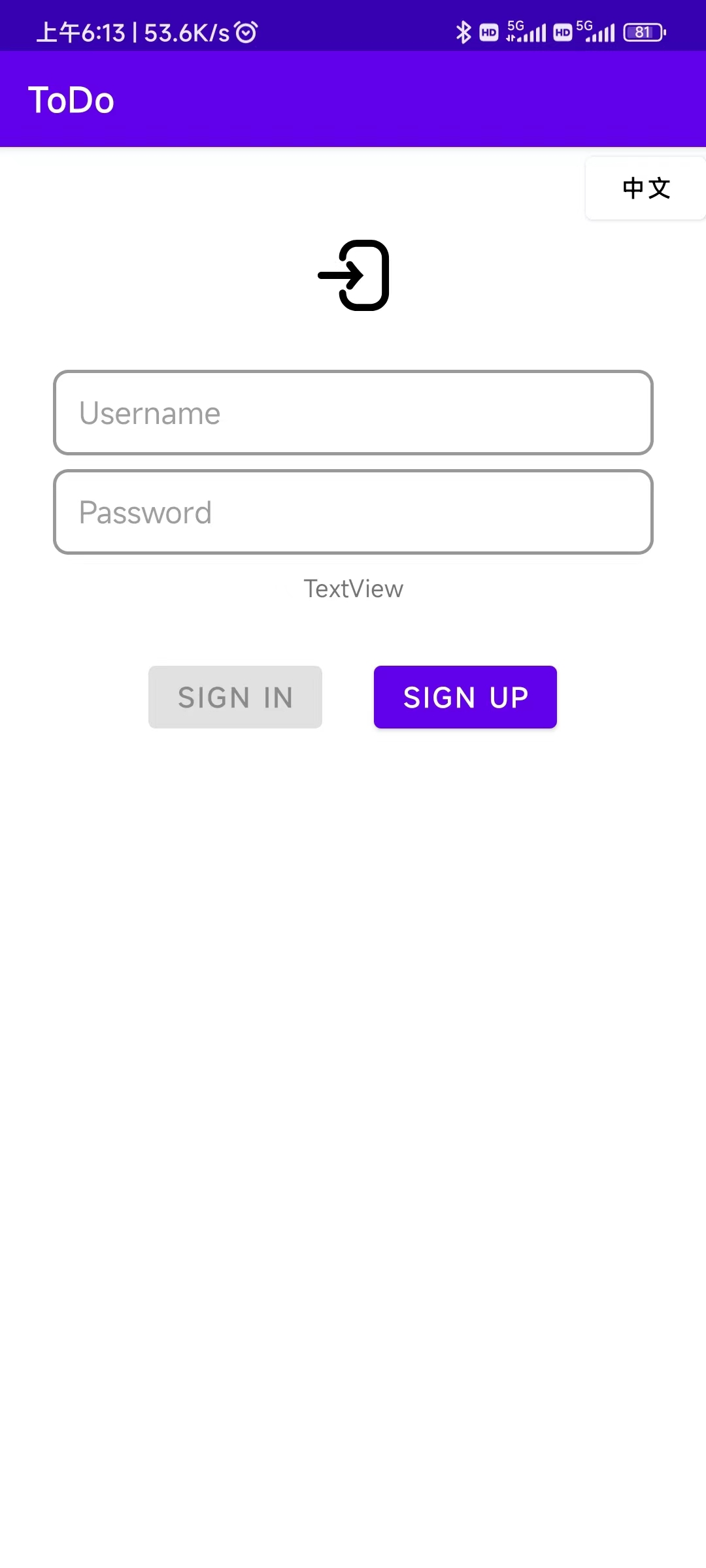
textAnimator.start()

}

}

}

### 1.3登录注册界面



登录注册界面:

提供用户名和密码输入框

输入框内容变化时,会调用usernameValid()、passwordValid()方法校验格式

根据校验结果,更新state状态,从而控制登录按钮是否可用

登录按钮点击时,判断用户名密码是否匹配,弹出成功或者失败提示

有注册按钮可跳转到注册界面

有语言切换按钮,点击后切换界面语言为中文或英文

实现逻辑:

使用State类存储输入框状态

注册TextWatcher监听输入框内容变化

在onTextChanged方法中,调用校验方法后更新state和登录按钮的enable状态

校验方法usernameValid()和passwordValid()实现格式校验

注册和语言切换按钮实现界面跳转和更新配置

class MainActivity3 : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main3)

val editText\_username = findViewById<EditText>(R.id.username)

val editText\_password = findViewById<EditText>(R.id.password)

val button\_login = findViewById<Button>(R.id.login)

var usernameState = State(1)

var passwrodState = State(1)

var loginState = State(-2)

checkText(editText\_username, usernameState, ::usernameValid, button\_login, loginState)

checkText(editText\_password, passwrodState, ::passwordValid, button\_login, loginState)

button\_login.setOnClickListener {

val username = editText\_username.text.toString()

val password = editText\_password.text.toString()

if (username == "123456" && password == "123456") {

Toast.makeText(this, "登陆成功", Toast.LENGTH\_LONG)

finish()

} else {

Toast.makeText(this, "登陆失败", Toast.LENGTH\_LONG)

}

}

val button\_register = findViewById<Button>(R.id.register)

button\_register.setOnClickListener{

val intent = Intent(this, MainActivity4::class.java)

startActivity(intent)

}

val button\_language = findViewById<Button>(R.id.button\_language)

button\_language.setOnClickListener {

var locale: Locale = if (button\_language.text != "ENG") { Locale("zh") }

else { Locale("en") }

Locale.setDefault(locale)

val config = Configuration()

config.setLocale(locale)

resources.updateConfiguration(config, resources.displayMetrics)

val intent = packageManager.getLaunchIntentForPackage(packageName)

intent?.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP)

startActivity(intent)

}

}

private fun usernameValid(username : String) : Int {

if (username.length !in 5..15) return -1

return 1

}

private fun passwordValid(password : String) : Int {

if (password.length !in 5..15) return -1

return 1

}

private fun checkText(editText: EditText, state : State, onChangeFun : (String)->Int, button: Button, loginState: State) {

editText.addTextChangedListener(object : TextWatcher {

override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {

}

override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {

if (onChangeFun(s.toString()) == 1 && state.value == 1) {

loginState.value += 1

state.value = 0

}

if (loginState.value >= 0) button.isEnabled = true

}

override fun afterTextChanged(s: Editable?) {

}

})

}

}

**三、测试情况**

## 1.功能测试

测试所有应用程序的功能是否正常工作，包括增加、删除、修改备忘事项、分组管理事项、单个事项开启计时等。

测试结果：

基本功能以及数据可合理性正常

## 2.性能测试

测试应用程序的响应时间、处理速度、数据传输速度、内存使用率等性能指标，以确保应用程序的性能符合用户的期望。

测试结果：

内存占用与cpu占用都非常小

内存占用最高使用量56MB

## 3. 兼容性测试

在各种不同的设备、操作系统上测试应用程序的功能，确保应用程序的跨平台兼容性。

测试结果：

HarmonyOS 3.0、MIUI 13等系统都正常运行

## 4. 用户反馈

测试应用程序的用户界面和用户体验，包括布局、字体、颜色、图标、按钮、标签等视觉设计元素，以及用户交互、反馈、导航等交互设计元素。

测试结果：

测试应用程序的用户界面和用户体验，包括布局、字体、颜色、图标、按钮、标签等视觉设计元素，以及用户交互、反馈、导航等交互设计元素。

**四、团队分工**

**何子亨：**

**负责整体ui风格，数据结构的设计，添加页面的具体实现，包括ui动画以及功能。**

**陈浩：**

**负责注册页面的设计与实现，以及效果调查，对所有资料进行整理及总结，文档的编写。**

**黎炜堂：**

**负责主页面的设计与实现，包括数据传递的方式以及存储方式。**

**黄润：**

**负责登录页面设计与实现，以及对app实施全面的测试。**