

本项目的开发进程是以周为进度来记录的

11.13-11.19: 项目启动与基础架构设置

一、小组讨论与赛题确认:

首先经过小组内讨论确认了我们赛题的选择为 OS 应用开发赛道，并且打算制作一个和做饭相关的应用

二、系统选择与工具设置

开发应用的操作系统选择了 openharmony，开发软件则是华为的 DevEco Studio 在开发的过程中遇到的第一个问题就是没有搞清楚 harmony OS 和 openharmony 的区别。相信第一次接触 openharmony 的初学者应该都会遇到这种问题:

1.最新版的 DevEco Studio 软件里并没有和老版一样可以直接选择 openharmony 框架，开发者只能通过先创建一个 harmony OS 的文件然后通过修改里面的 build-profile.json5 文件来实现，如下图 1 和图 2 所示:

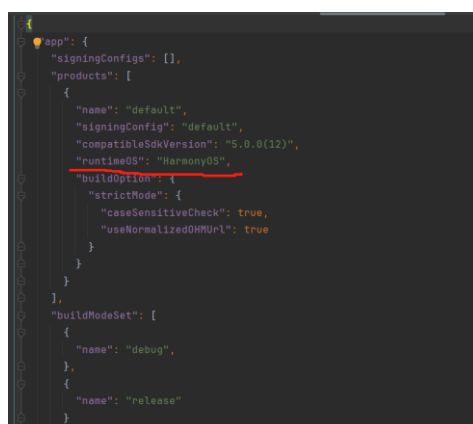


图 1



图 2

2.虽然 DevEco Studio 软件里面可以自行下载模拟器代替设备调试，但是这个模拟器是指能够运行 harmony OS 的，并不支持 openharmony 的运行，经过协商，我们小组购入一台 1 加 6T 二手机，并进行刷机安装 openharmony-5.0.0 系统来代替开发板进行调试。装系统资源和教材都是通过 B 站视频来实现的效果如下图所示:

(视频链接: https://www.bilibili.com/opus/995934398360584217?spm_id_from=333.999.0.0)



图 3

11.20-11.26：基础功能开发与框架搭建

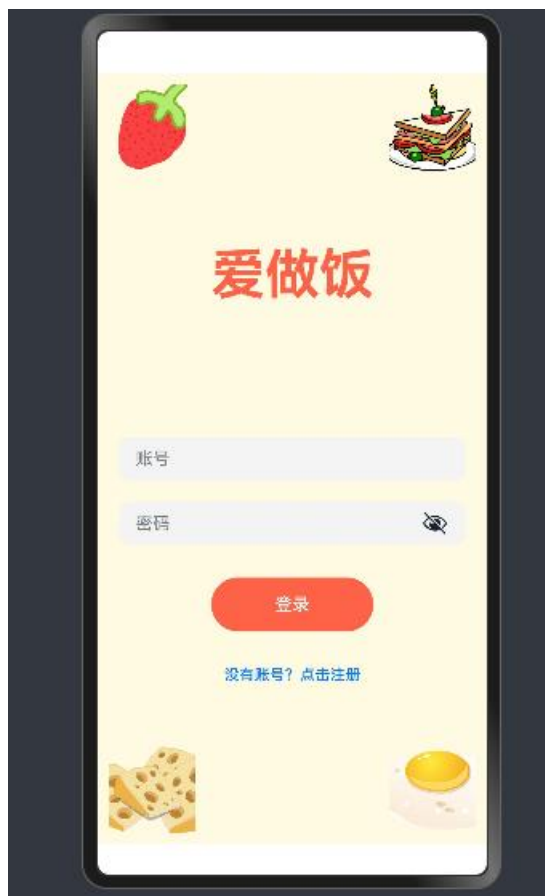
一、基础学习以及框架选择

首先代码都是基于 openharmony 的官方文档来撰写(<https://www.openharmony.cn/mainPlay/>) 选择了 ArkTS (OpenHarmony 生态的应用开发语言) 语言, ArkUI (OpenHarmony 上原生 UI 框架) 框架以及 Stage 模型来进行开发。

二、功能模块开发

- **登录功能开发：**设计和实现用户注册、登录等功能

1. 界面展示：



在背景插入了一些食物图片来点缀, 然后有账号和密码的输入框 (密码可以通过点击眼睛按钮来切换是否可见), 以及实现了登录和注册功能。整体上实现了一个简约美观的登陆界面

2. 核心代码展示：

```
// 背景部分：使用 Column 作为背景，确保背景填充整个屏幕
Column()
  .width('100%')
  .height('100%')
  .backgroundColor('#FFFAE1') // 背景色
  .borderRadius(0) // 背景容器的圆角

// 四个角的食物图片
// 左上角图片
Image($r('app.media.food1'))
  .position({ left: 10, top: 10 })
  .width(80)
  .height(80)
```

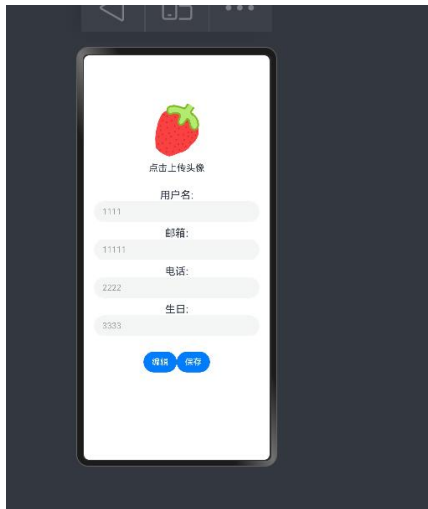
(插入图片以及设置背景色)

文本输入框，登录、注册按钮以及点击逻辑的实现：

```
Column() {  
  // 显示应用名称  
  Text('爱做饭')  
    .fontSize(50) // 设置字体大小  
    .fontWeight('bold') // 设置字体粗细  
    .fontColor('#FF6347') // 设置字体颜色  
    .textAlign(TextAlign.Center) // 文字居中  
    .margin({ top: 20, bottom: 100 }) // 设置距离顶部的间距，并增加底部间距  
  
  // 登录部分，增加间距  
  Column() {  
    // 账号输入框  
    TextInput({ placeholder: '账号' })  
      .margin({ top: 20 })  
      .borderRadius(10) // 圆角边框  
      .padding({ left: 15, right: 15 }) // 内边距  
      .backgroundColor('#F4F4F4')  
  
    // 密码输入框  
    TextInput({ placeholder: '密码' })  
      .type(InputType.Password)  
      .margin({ top: 20 })  
      .borderRadius(10)  
      .padding({ left: 15, right: 15 })  
      .backgroundColor('#F4F4F4')  
  
    // 登录按钮  
    Button('登录')  
      .width(150)  
      .margin({ top: 30 })  
      .borderRadius(25) // 按钮圆角  
      .height(50) // 按钮高度  
      .backgroundColor('#FF6347') // 按钮背景色  
      .fontColor('FFF') // 按钮文字颜色  
      .onClick(() => {  
        console.info('登录按钮被点击');  
        // 实现登录逻辑  
      })  
  
    // 注册按钮  
    Button('没有账号? 点击注册')  
      .margin({ top: 20 })  
      .fontSize(14)  
      .fontColor('#007BFF') // 注册按钮颜色  
      .backgroundColor('transparent')  
      .onClick(() => {  
        // 跳转到注册界面  
        console.info('跳转到注册页面');  
      })  
  }  
}
```

- **个人信息功能：**设计用户的个人信息管理模块，支持修改个人资料，退出登录功能。

1.界面展示：



实现了用户可以自己上传图库图像，以及用户名，邮箱，电话等一些基本信息的输入，点击编辑按钮后用户可以编辑相关信息（字体颜色由暗转亮代表可以编辑），点击保存后后用户自动退出编辑状态

2. 核心代码展示：

由于其他代码和之前的类似，此处不再说明，以下是读取图库图片并且更换的函数展示
该函数是参考 [openharmony 文档里的示例代码实现](#)

([zh-cn/third-party-cases/photo-pixmap-transfer.md](#) • [OpenHarmony/docs - Gitee.com](#))

```
async pickProfile() {
  const photoSelectOptions = new picker.PhotoSelectOptions();
  photoSelectOptions.MIMETYPE = picker.PhotoViewMIMETypes.IMAGE_TYPE;
  photoSelectOptions.maxSelectNumber = 1;
  const photoViewPicker = new picker.PhotoViewPicker();
  photoViewPicker.select(photoSelectOptions)
    .then((photoSelectResult: picker.PhotoSelectResult) => {
      let imageUri = photoSelectResult.photoUris[0];
      console.info('photoViewPicker.select to file succeed and uris are:' + imageUri);
      let context = getContext();
      let filesDir = context.filesDir;
      let fileName = "userProfileImage";
      let path = filesDir + "/" + fileName + "." + imageUri.split(".")[1];
      let file = fs.openSync(imageUri, fs.OpenMode.READ_ONLY);
      let file2 = fs.openSync(path, fs.OpenMode.READ_WRITE | fs.OpenMode.CREATE);
      fs.copyFileSync(file.fd, file2.fd);
      fs.closeSync(file.fd);
      fs.closeSync(file2.fd);
      this.uri = fileUri.getUriFromPath(path);
      this.getPixmap();
    })
    .catch((err: BusinessError) => {
      console.error('MinePage',
        'Invoke photoViewPicker.select failed, code is ${err.code}, message is ${err.message}');
    });
}

// 获取图片的 Pixmap
getPixmap() {
  let file = fs.openSync(this.uri, fs.OpenMode.READ_WRITE);
  const imageSourceApi = image.createImageSource(file.fd);
  imageSourceApi.createPixmap().then(pixmap => {
    this.profileImage = pixmap;
    console.log('Succeeded in creating pixmap object through image decoding parameters.');
```

3. 问题说明

但是后续实机调试过程中发现手机的图库打开后立刻闪退后导致功能无法实现,所以最后只能删除上传图片自定义头像的功能,改为使用默认头像。查询资料以及在论坛上寻求帮助后依旧无法解决,情况如下图所示:



本来还准备调用百度的菜品识别 API 来实现对一个菜品的识别以及营养的分析,但由于图片无法上传和相机无法使用的原因导致删除了这个功能

调用 API 代码展示

```
// 调用菜品识别 API
async AI_request() {
  let httpRequest = http.createHttp();
  try {
    // 发送请求到菜品识别 API
    const response = await httpRequest.request(
      "https://aip.baidubce.com/rest/2.0/image-classify/v2/advanced_general?access_token=" + this.access_token,
      {
        method: http.RequestMethod.POST,
        header: {
          'Content-Type': 'application/x-www-form-urlencoded'
        },
        extraData: {
          'image': this.base64Str.split(',')[1], // 获取 Base64 数据部分
          'baike_num': 1
        },
        connectTimeout: 60000,
        readTimeout: 60000,
      }
    );
  }
};
```

转化图片为 base64 格式代码展示:

```
// 将缩略图转为 Base64
const imagePackageApi: image.ImagePacker = image.createImagePacker();
let packOpts: image.PackingOption = {
  format: 'image/jpeg',
  quality: 100,
};

const readBuffer = await imagePackageApi.packing(pixelMap, packOpts);

// Base64 编码
let base64Helper = new util.Base64Helper();
let uint8Arr = new Uint8Array(readBuffer);
console.info('selectPhoto Uint8Array, uint8Arr: ' + JSON.stringify(uint8Arr));

let pixelStr = base64Helper.encodeToStringSync(uint8Arr);
this.base64Str = 'data:image/jpeg;base64,' + pixelStr;
console.info('selectPhoto base64, base64Str: ' + this.base64Str);
```

11.27-12.3：核心功能开发

先确认了应用的核心功能为菜谱查找和根据添加菜谱来生成食材的购物清单

- **菜谱查找功能：**设计并实现菜谱数据库，用户可以根据菜谱名字以及难易程度查找菜谱。

1. 界面展示



按照菜品制作难度对菜谱进行分类，然后通过输入菜谱名字可以查找菜谱，点击进去后可以显示菜谱的食材，制作步骤，标签等信息，然后点击添加按钮会在食物购买清单那里添加。底部是设置了到不同功能的导航栏，菜谱的数据是通过 json 文件来保存的。

2. 核心代码展示

添加菜谱到购物清单的代码：

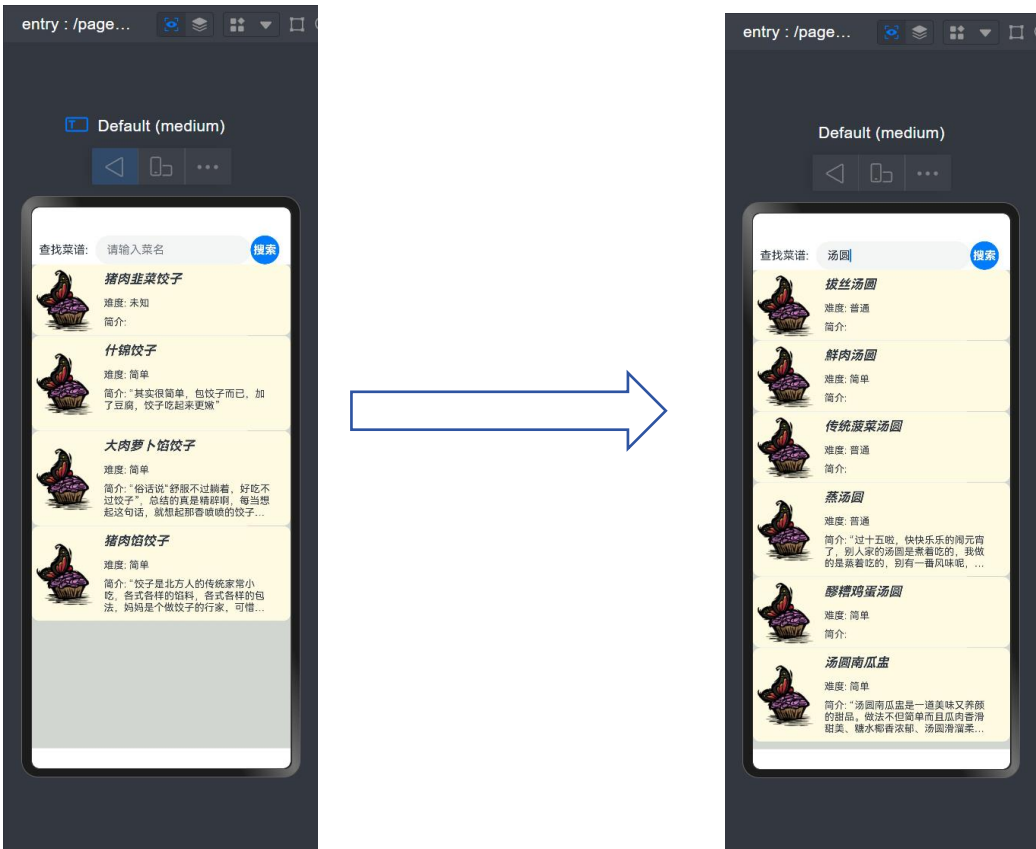
```
aboutToAppear(): void {
  console.info("result foodtitle"+this.response)
  let context = getContext(this)
  let fileDir = context.filesDir
  try {
    context.resourceManager.getRawFileContent("food_menu_results.json").then(value => {
      console.info('rawFile_utf: ' + value)
      let rawFile:string = this.convertUint8ArrToStr(value);
      console.info('rawFile: ' + json.stringify(rawFile))
      this.response = rawFile
      this.result= this.response
    })
  } catch (error) {
    console.error(`promise getRawFileContent failed, error code: ${error.code}, message: ${error.mess
  }
```

查询菜谱相关的代码：

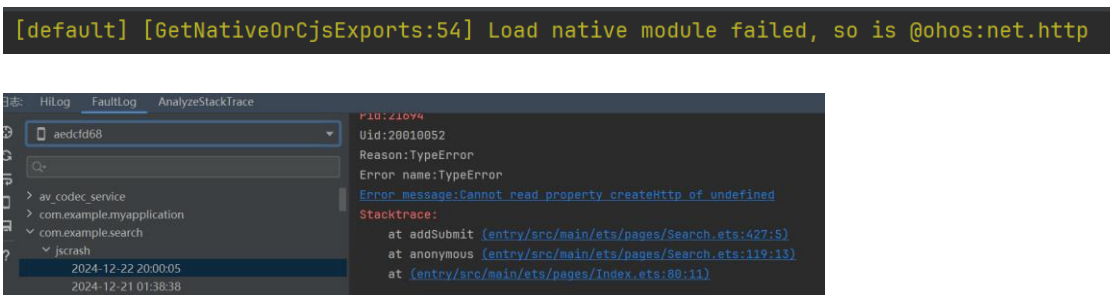
```
addSubmit() {
  promptAction.showToast({ message: '获取菜谱中~~~' });
  let res:object[] = [];
  (JSON.parse(this.response)as object[])?.forEach(item => {
    if (item['title'] && item['title'].includes(this.foodTitle)) { // 检查 title 是否包含关键词
      res.push(item); // 如果找到匹配项，将元素添加到 result 数组
    }
  });
  this.result = JSON.stringify(res)
  console.info("result foodtitle"+JSON.stringify( (JSON.parse(this.response))))
  console.info("result foodtitle"+this.foodTitle)
  console.info("get result "+this.result)
}
```

3. 问题

初步计划以调用外部 API 的方法实现菜谱的查询，但是在实机操作时发生了无法使用 Http 请求模块的错误，因此改用了搜索本地数据的方式。下图是预览器下成功调用 API 的示例



下图是日志里的报错情况：



- **购物清单功能：**用户可以根据选择的菜谱生成购物清单，并可以勾选已购买的物品。

1. 界面展示



上半部分是需要制作的菜谱（点击红色删除按钮可以删除选择的菜谱），然后点击统计食材就会在下半部分生成购物清单，家里如果拥有的食材或者已经购买完毕的可以勾选食材清除

2. 核心代码

勾选拥有的或者已经购买完毕食材会在食材名称上面划线

```

List(){
  ForEach(this.ingredients, (item: IIngredient, index) => {
    ListItem(){
      Flex({ justifyContent: FlexAlign.SpaceBetween, alignItems: ItemAlign.Center }) {
        Row({ space: 4 }) {
          Circle()
            .width(24)
            .height(24)
            .fill(Color.White)
            .borderWidth(3)
            .borderRadius(30)
            .borderColor('#ffdcdfdf')
            .foregroundColor(this.Task[index] == true ? "green" : "white")
            .margin({ right: 10 })
            .onClick(() => {
              this.Task[index] = !this.Task[index]
              this.Task = [...this.Task]
            })
        }
        Text(item.name + item.quantity)
          .maxLines(1)
          .fontSize(15)
          .decoration({ type: this.Task[index] == true ? TextDecorationType.LineThrough: TextDecorationType.None })
      }
    }
  })
}

```

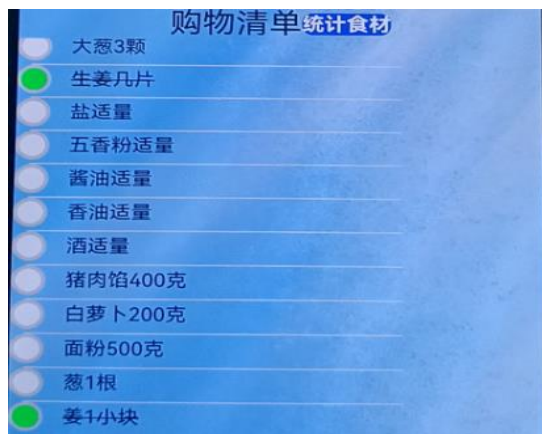

对菜谱的食材进行累加并且生成购物清单的函数：

```
CountIngredients() {
  this.ingredients=[]
  this.Rlist.forEach((item: IRecipe) => {
    console.info("item.ingredients:" +JSON.stringify(typeof item.ingredients))
    let ingredientsArray: IIngredient[] = Object.entries(item.ingredients as object).map((entry: [string, string]) => {
      const name = entry[0]; // 第一个元素是食材名称
      const quantity = entry[1]; // 第二个元素是食材数量
      return { name, quantity } as IIngredient; // 返回一个符合 IIngredient 接口的对象
    });console.info("ingredientsArray:" +JSON.stringify(ingredientsArray))
    ingredientsArray.forEach((ingre1: IIngredient) => {
      let same = false;
      this.ingredients.forEach((ingre2: IIngredient) => {
        // 检查两者的原料名称是否相同
        if (ingre1.name === ingre2.name) {
          same = true;
          // 尝试从数量中匹配数字和单位
          let match1 = ingre1.quantity.match(/(\d+)(\D+)?/);
          let match2 = ingre2.quantity.match(/(\d+)(\D+)?/);

          // 如果两个原料的数量都能匹配到数字和单位
          if (match1 && match2) {
            // 数字相加
            let totalQuantity = parseInt(match1[1]) + parseInt(match2[1]);
            let unit = match1[2] || match2[2]; // 默认取第一个的单位
            ingre2.quantity = totalQuantity.toString() + (unit || ''); // 合并数量和单位
          } else {
            // 如果没有数字和单位的匹配，设置为“适量”
            ingre2.quantity = '适量';
          }
        }
      });
    });
    // 如果没有找到相同的原料，就添加一个新的原料
    if (!same) {
      this.ingredients.push({ name: ingre1.name, quantity: ingre1.quantity });
    }
  });
  this.ingredients=[...this.ingredients]
}
```

3. 问题：

由于生成购物清单的算法比较简单，然后加上是刚开始学习 Ark 语言，网上的资料也比较少，所以它在生成清单的时候譬如姜末和生姜这种它是不会合在一起的，而是分成两个独立的东西加入到购物清单里的，准备后续再进行优化。问题如下图所示：

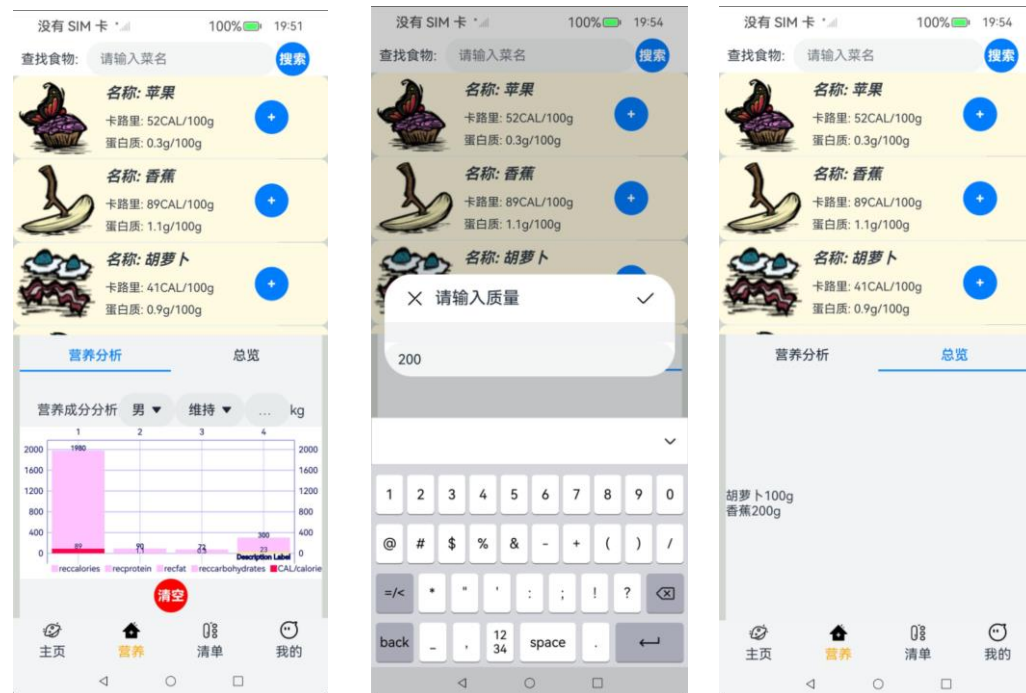


12.4-12.10: 进阶功能开发

在原先核心的功能基础上增加了对食材营养成分分析以及自定义设置通知的进阶功能

- **营养成分分析功能:** 根据菜谱和食材, 计算并显示每道菜的营养成分(如热量、蛋白质、脂肪等)。

1. 界面展示



上半部分是一些基本的食材以及它的营养成分(食材及营养成分的数据是通过分布式数据库导入的, 可以根据需求后续添加更多数据), 也可以通过上面的搜索食材框来搜索食材, 点击添加按钮然后输入食材的重量(每个食材的营养成分默认为 100g), 在下半部分生成营养成分分析图, 营养分析图的达标值用户可以根据性别, 要求(减肥, 增肌, 维持)以及体重的设定来实现, 可以通过柱形图观察, 总览则是就可以查看用户总计添加了哪些食材

2. 核心代码

添加食材函数:

```
AddToList(name:string,weight:number){
  let same = false;
  this.Foodlist.forEach((item:object,index)=>{
    if(item['name'] == name) {
      this.Foodlist[index].weight +=weight
      this.Foodlist=[...this.Foodlist]
      same=true
    }
  })
  if(!same) {
    this.Foodlist[this.Foodlist.length]= { name:name,weight:weight }
    promptAction.showToast({ message: '添加成功!' });
  }
  console.info("Listofrecipe:" + this.Foodlist)
}
```

绘制营养成分分析柱形图的函数：

```
DrawBarChart():void{
    //绘制柱状图
    let values1:JArrayList<BarEntry> = new JArrayList<BarEntry>();
    let values2:JArrayList<BarEntry> = new JArrayList<BarEntry>();
    interface RecObj{
        calories:number;
        protein:number;
        fat:number;
        carbohydrates:number
    }
    let Rec:RecObj[] = [
        {calories:33 , protein:1.5, fat:1.2 ,carbohydrates:5},//基础男性
        {calories:32 , protein:1.5, fat:1 ,carbohydrates:4.5},//基础女性
    ]

    let plan:number[]=[1,0.85,1.15]
    values2.add(new BarEntry(1, Rec[this.selectedtype].calories*plan[this.selectedplan]*this.inputweight))
    values2.add(new BarEntry(2, Rec[this.selectedtype].protein*plan[this.selectedplan]*this.inputweight))
    values2.add(new BarEntry(3, Rec[this.selectedtype].fat*plan[this.selectedplan]*this.inputweight))
    values2.add(new BarEntry(4, Rec[this.selectedtype].carbohydrates*plan[this.selectedplan]*this.inputweight))

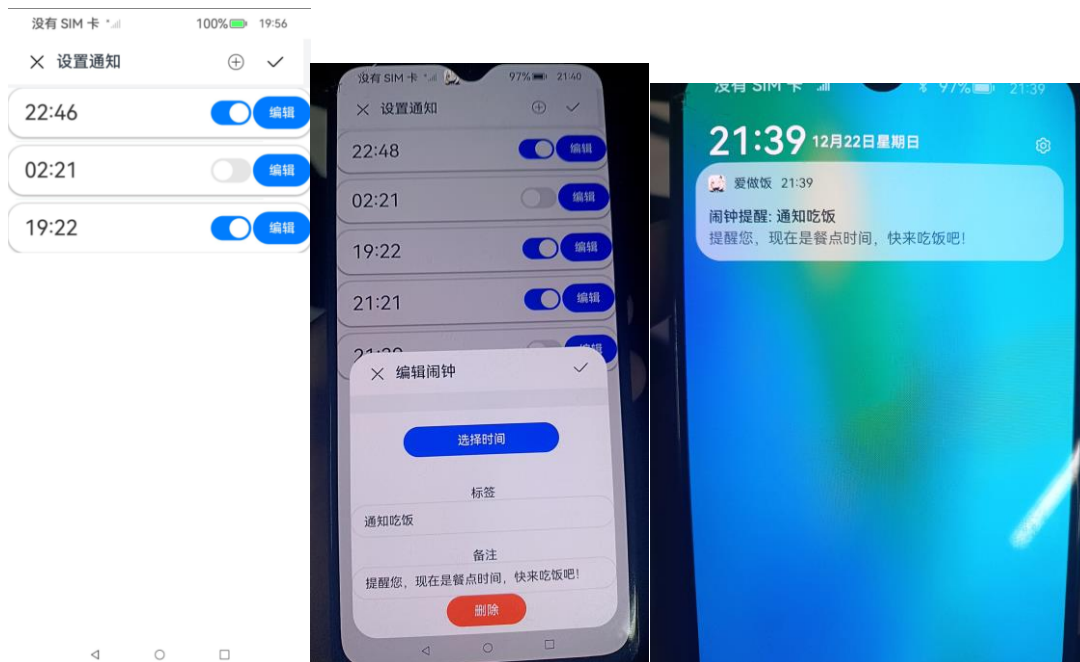
    let set1: BarDataSet,set2: BarDataSet,set3: BarDataSet,set4: BarDataSet
    let dataSets: JArrayList<IBarDataSet> =
        new JArrayList<IBarDataSet>();
    set1 = new BarDataSet(values2, "reccalories");
    set1.setColorByColor(Color.Pink);
    set2 = new BarDataSet(values2, "recprotein");
    set2.setColorByColor(Color.Pink);
    set3 = new BarDataSet(values2, "recfat");
    set3.setColorByColor(Color.Pink);
    set4 = new BarDataSet(values2, "reccarbohydrates");
    set4.setColorByColor(Color.Pink);
    dataSets.add(set1);
    dataSets.add(set2);
    dataSets.add(set3);
    dataSets.add(set4);
    values1 = new JArrayList<BarEntry>();
    set1 = new BarDataSet(values1.add(new BarEntry(1, this.Foodall.calories)), "CAL/calories");
    set1.setColorByColor(Color.Red);
    values1 = new JArrayList<BarEntry>();
    set2 = new BarDataSet(values1.add(new BarEntry(2, this.Foodall.protein)), "g/protein");
    set2.setColorByColor(Color.Green);
    values1 = new JArrayList<BarEntry>();
    set3 = new BarDataSet(values1.add(new BarEntry(3, this.Foodall.fat)), "g/fat");
    set3.setColorByColor(Color.Blue);
    values1 = new JArrayList<BarEntry>();
    set4 = new BarDataSet(values1.add(new BarEntry(4, this.Foodall.carbohydrates)), "g/carbohydrates");
    set4.setColorByColor(Color.Yellow);

    dataSets.add(set1);
    dataSets.add(set2);
    dataSets.add(set3);
    dataSets.add(set4);

    let data:BarData = new BarData(dataSets)
    this.model.setData(data)
}
```

- **通知时间功能：**为用户设置烹饪过程中的提醒（例如，某个步骤需要在 5 分钟后执行时提醒用户）或者为用户设置需要吃饭，买菜的时间点来提醒。

1. 界面展示



整个界面是类似于闹钟的，左上角的导航栏的 x 是返回上一个界面，然后+号则是用来添加新的提醒。提醒的通知闹钟用一个列表来展示，左边是用户设置通知的时间，右边通过一个切换开关来实现通知是否开启，然后点击编辑会进去一个弹窗，可以选择时间（通过 Ark 自带的时间选择器来实现），设置标签和备注内容（通知内容），点击删除即可删除通知

2. 核心代码

添加闹钟以及编辑闹钟：

```
// 添加新的闹钟，跳转到编辑页面
addNewAlarm() {
  this.selectedAlarmIndex = -1 // 新增闹钟时没有选择现有闹钟

  this.EditTitle = '添加闹钟'
  this.editingTime = new Date() // 默认选择当前时间
  this.editingLabel = '通知吃饭' // 设置默认标签
  this.editingNote = '提醒您, 现在是餐点时间, 快来吃饭吧!' // 设置默认备注
  this.dialogController.open()
}

// 编辑现有闹钟，跳转到编辑页面
editAlarm(index: number) {
  this.selectedAlarmIndex = index
  this.EditTitle = '编辑闹钟'
  const alarm = this.alarms[index]
  this.editingTime = new Date(alarm.time)
  this.editingLabel = alarm.label
  this.editingNote = alarm.note
  this.dialogController.open()
}
```

保存闹钟的函数：

```
saveEditedAlarm() {  
  if (this.selectedAlarmIndex === -1) {  
    // 新添加的闹钟  
    const newAlarm: Alarm = {  
      time: this.editingTime,  
      label: this.editingLabel,  
      isEnabled: true,  
      note: this.editingNote,  
      id: 0  
    };  
    (this.alarms as Alarm[]).push(newAlarm);  
    AppStorage.SetOrCreate('Alarms', this.alarms as Alarm[])  
    this.scheduleNotification(newAlarm); // 发布通知  
  } else {  
    // 编辑现有的闹钟  
    const editedAlarm = this.alarms[this.selectedAlarmIndex];  
    editedAlarm.time = this.editingTime;  
    editedAlarm.label = this.editingLabel;  
    editedAlarm.note = this.editingNote;  
    this.alarms = [...this.alarms];  
    AppStorage.SetOrCreate('Alarms', this.alarms)  
    this.scheduleNotification(editedAlarm); // 更新通知  
  }  
}
```

用来通知的函数：

```

scheduleNotification(alarm: Alarm) {
  const notifyTime = alarm.time.getTime() - new Date().getTime(); // 计算到通知时间的毫秒数

  if (notifyTime > 0) { // 检查时间是否在未来
    // 使用 setTimeout 在指定的时间发布通知
    setTimeout(() => {
      const notificationTitle = `闹钟提醒: ${alarm.label}`;
      const notificationContent = alarm.note;
      // 设置通知的内容类型
      let publishCallback = (err: BusinessError): void => {
        if (err) {
          console.error(`publish failed, code is ${err.code}, message is ${err.message}`);
        } else {
          console.info("publish success");
        }
      }
      let notificationRequest: notificationManager.NotificationRequest = {
        id: alarm.id as number, // 使用传入的 alarm.id 作为通知 ID
        content: {
          notificationContentType: notificationManager.ContentType.NOTIFICATION_CONTENT_BASIC_TEXT,
          normal: {
            title: notificationTitle,
            text: notificationContent, // 这里用 alarm.note 作为通知的具体内容
          }
        }
      };
      notificationManager.publish(notificationRequest, publishCallback);
      alarm.isEnabled=false
      // 发布通知
    }, notifyTime);
  } else {
    console.warn("设置的时间已经过去");
  }
}

```

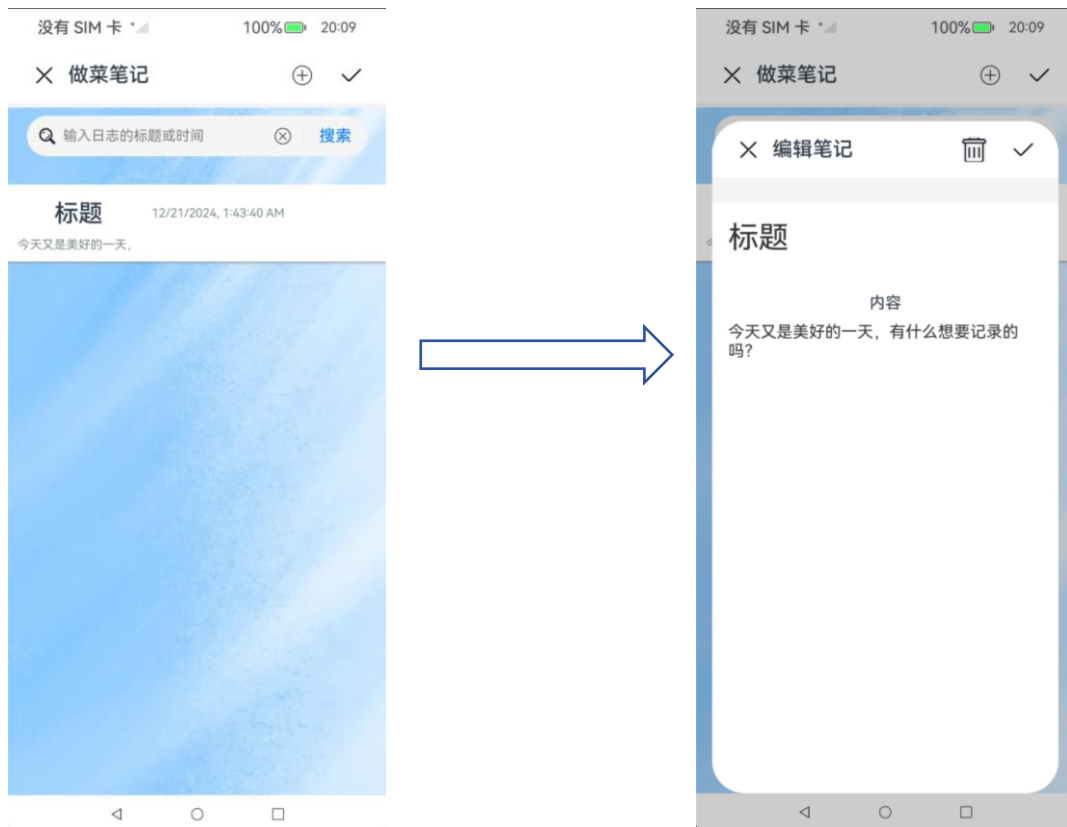
12.11-12.17：功能补充与优化

一、功能补充

在原先功能的基础上再添加了一个记事本的功能，用户可以用来记录自己的做饭心得和当成备忘录来使用，还增加了一个做饭宝典的界面，用来给用户提示一些厨具使用方法，食材保存建议等。

- **记事本功能：**实现用户在应用中记下烹饪心得或其他相关内容的功能。

1. 界面展示



2. 核心代码

增加笔记和编辑笔记的代码：

```
addNewNote() {
  this.selectedNoteIndex = -1 // 新增页面时没有选择现有页面
  this.EditTitle = '添加笔记'
  this.editingTitle = '标题' // 设置默认标题
  this.editingContent = '今天又是美好的一天，有什么想要记录的吗?' // 设置默认内容
  this.dialogController.open()
}

// 编辑记事本页面
editNote(index: number) {
  this.selectedNoteIndex = index
  const selectedNote = this.notebookPages[index]
  this.editingTitle = selectedNote.title || '新页面'
  this.editingContent = selectedNote.content || ''
  this.EditTitle = '编辑笔记'
  this.dialogController.open()
}
```

3. 问题

在后续应用的调试中发现了一开始写的代码里并没有永久存储的功能，在下次打开应用的时候，上次保存的数据就会消失，经过查阅文档得知可以试用本地存储的方式来实现应用数据的永久存储。在此过程中又发现前面诸如个人信息，设置通知等模块也出现了这样的问题，并没有实现数据的永久化存储，后续将用本地存储对代码进一步优化。

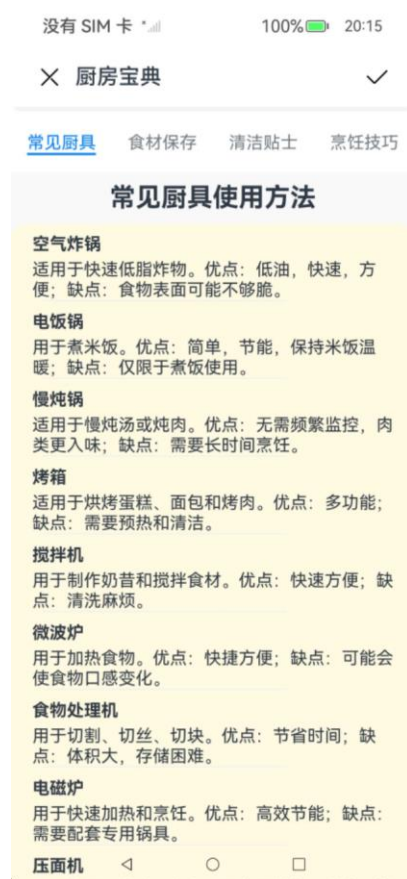
```

saveEditedNote() {
  const currentTime = new Date().toISOString(); // 获取当前时间并转化为 ISO 格式字符串
  if (this.selectedNoteIndex === -1) {
    // 新增的记事本
    const newNote: NotebookPage = {
      title: this.editingTitle,
      content: this.editingContent,
      createdAt: currentTime,
    };
    this.notebookPages.push(newNote);
  } else {
    // 编辑现有的记事本
    this.notebookPages[this.selectedNoteIndex].title = this.editingTitle;
    this.notebookPages[this.selectedNoteIndex].content = this.editingContent;
    this.notebookPages = [...this.notebookPages];
  }
  AppStorage.SetOrCreate('notebookPages', this.notebookPages); // 保存数据到本地存储
}

```

- **厨房贴士功能：**开发初步的厨房贴士模块，包含食材保存、烹饪技巧等基础内容。

1. 界面展示



2. 核心代码

一些相关数据简单的插入

```

List() {
  ListItemGroup({ header: this.itemHead('蔬菜') }) {
    ListItem() { Text('大多数蔬菜应存放在冰箱的蔬菜抽屉中，避免潮湿。') }
  }
  ListItemGroup({ header: this.itemHead('肉类') }) {
    ListItem() { Text('生肉应存放在冰箱冷藏区，若不立即食用可冷冻保存。') }
  }
  ListItemGroup({ header: this.itemHead('谷物') }) {
    ListItem() { Text('保持干燥阴凉处，密封存储防虫害。') }
  }
  ListItemGroup({ header: this.itemHead('奶制品') }) {
    ListItem() { Text('存放在冰箱冷藏区，开封后尽快食用。') }
  }
  ListItemGroup({ header: this.itemHead('水果') }) {
    ListItem() { Text('水果通常应存放在室温下，避免存放在塑料袋中。') }
  }
  ListItemGroup({ header: this.itemHead('面包') }) {
    ListItem() { Text('应存放在阴凉干燥处，保持面包的新鲜度。') }
  }
  ListItemGroup({ header: this.itemHead('坚果') }) {
    ListItem() { Text('坚果应存放在密封袋中，并保持干燥，避免潮湿。') }
  }
  ListItemGroup({ header: this.itemHead('巧克力') }) {
    ListItem() { Text('巧克力应存放在阴凉干燥处，避免高温。') }
  }
  ListItemGroup({ header: this.itemHead('调味料') }) {
    ListItem() { Text('调味料应密封存放，避免受潮影响质量。') }
  }
  ListItemGroup({ header: this.itemHead('冷冻食品') }) {
    ListItem() { Text('冷冻食品应尽量避免反复解冻，以保持其品质。') }
  }
  ListItemGroup({ header: this.itemHead('茶叶') }) {
    ListItem() { Text('茶叶应存放在干燥阴凉处，并密封保存，避免受潮。') }
  }
}

```

二、功能优化

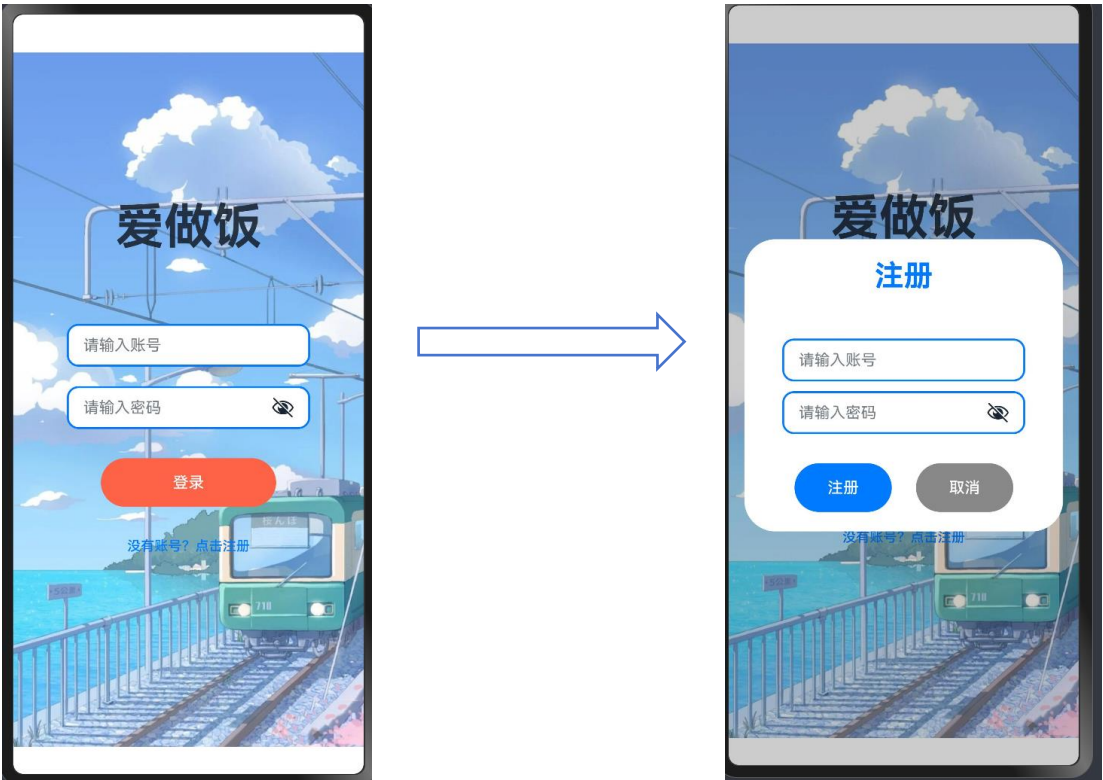
对之前的登录功能，个人信息，设置通知不能存储数据的问题进行了优化。并且

将设置通知，记事本，个人信息，厨房宝典，退出登录的功能合并到一个我的界面

● 登录功能优化：

增加了本地存储数据的功能，并且优化了界面设计，补充了完整的登录验证以及注册逻辑。还增加了一些错误提示，如登录时用户名不能为空，注册的时候用户名不能少于三位，密码不能少于六位等。

1.界面展示：



2.核心代码展示：

存储本地数据

```
interface GeneratedTypeLiteralInterface_1 {
    username: string;
    password: string;
}

interface GeneratedObjectLiteralInterface_1 {
    username: string;
    password: string;
}

PersistentStorage.PersistProp<boolean>('isLogin', false);
PersistentStorage.PersistProp<GeneratedTypeLiteralInterface_1>('credentials', null);

@Entry
@Component
export struct LoginPage {
    @State username: string = '' // 存储用户名
    @State password: string = '' // 存储密码
    @State errorMessage: string = '' // 错误提示信息
    @StorageLink('isLogin') isLogin: boolean = false
    @StorageLink('credentials') storedCredentials: GeneratedTypeLiteralInterface_1 | null = null // 本地存储的用户凭证
    private registerDialogId: number = 0
```

登录逻辑：

```
// 处理登录逻辑
handleLogin() {
  if (this.validateUsername(this.username) && this.validatePassword(this.password)) {
    if (this.storedCredentials) {
      const username = this.storedCredentials.username;
      const password = this.storedCredentials.password;

      if (this.username === username && this.password === password) {
        console.info('登录成功');
        this.errorMessage = '';
        promptAction.showDialog({
          title: '登录成功',
          message: '欢迎回来!',
          buttons: [{ text: '确认', color: '#FF6347' }],
        });
        this.islogin = true;
      } else {
        this.showErrorDialog('用户名或密码错误');
      }
    } else {
      this.showErrorDialog('没有找到已注册的账号');
    }
  } else {
    console.info('登录失败');
  }
}
}
```

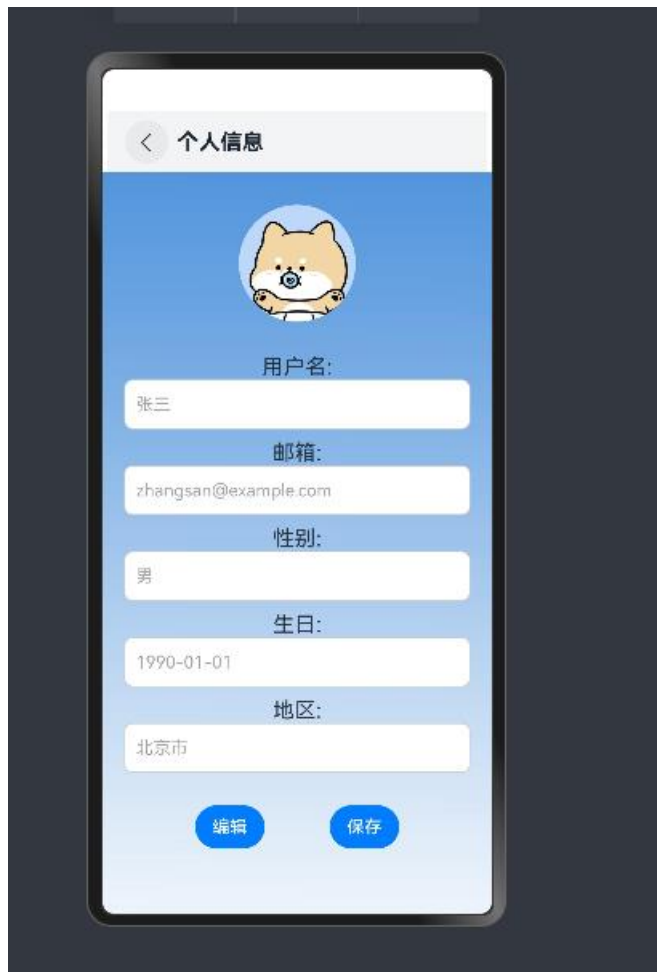
注册逻辑:

```
// 处理注册逻辑
handleRegister() {
  if (this.validateUsername(this.username) && this.validatePassword(this.password)) {
    const credentials: GeneratedObjectLiteralInterface_1 = { username: this.username, password: this.password };
    // 保存用户信息到本地存储
    AppStorage.set('credentials', credentials);
    console.info('注册成功');
    this.errorMessage = '';
    promptAction.showDialog({
      title: '注册成功',
      message: '您已成功注册!',
      buttons: [{ text: '确认', color: '#007BFF' }],
    });
    promptAction.closeCustomDialog(this.registerDialogId); // 关闭注册弹窗
  } else {
    console.info('注册失败');
  }
}
}
```

- 个人信息界面优化:

由于存储本地数据的代码相似，本处不再过多赘述。在原先的基础上优化了界面，并且增加了一个导航栏可以点击返回按钮返回我的界面

1.界面展示:



2.核心代码展示:

导航栏功能实现

```
// 主界面导航栏
EditableTitleBar({
  leftIconStyle: EditableLeftIconType.Cancel, // 设置左侧为返回箭头
  title: '个人信息', // 修改为“个人信息”作为标题
  isSaveIconRequired: false,

  onCancel: () => {
    promptAction.showToast({ message: "返回操作" })
    router.pushUrl({ url: 'pages/me' }).then(() => {
      console.info('Succeeded in jumping to the second page.')
    }).catch((err: BusinessError) => {
      console.error('Failed to jump to the second page. Code is ${err.code}, message is ${err.message}')
    })
  },
})
.height(60)
.backgroundColor('#F3F4F6') // 浅灰色背景
.padding({ left: 0, right: 10 });
```

- 我的界面整合优化:

对一些功能进行整合，可以实现点击跳转以及返回的功能，并且设置了默认头像和默认背景，用户可以通过点击来切换。

1.界面展示：



核心代码展示：

点击切换图片的代码，头像的和这个类似

```
private images1: Resource[] = [
    $r('app.media.bj1'),
    $r('app.media.bj7'),
    $r('app.media.bj3'),
    $r('app.media.bj4'),
    $r('app.media.bj5'),
    $r('app.media.bj6'),
    $r('app.media.bj7'),
    $r('app.media.bj8'),
    $r('app.media.bj9'),
    $r('app.media.bj10'),
];

@State currentIndex1: number = 0;
@State currentIndex: number = 0;

build() {
    RelativeContainer() {
        // 背景图片：占据整个屏幕
        Image(this.images1[this.currentIndex1])
            .objectFit(ImageFit.Cover) // 背景图像保持比例，覆盖整个容器
            .height('100%')
            .width('100%')
            .zIndex(0)
            .borderRadius(0) // 背景图像不使用圆角
            .margin({top: 0, left: 0, right: 0, bottom: 0})
            .onClick(() => {
                // 点击后切换到下一张图片，并保持循环
                this.currentIndex1 = (this.currentIndex1 + 1) % this.images1.length;
            })
    }
}
```

12.18-12.25：综合测试与最终整理交付

一、各个功能的整合：对所有功能进行整合，并且可以通过路由跳转，实现一个完整的应用功能

二、测试与调试：在设备上进行应用的最终测试，确保没有 bug，并解决任何遗留的问题。

三、演示视频与文档制作：拍摄演示视频，完成文档编写，包括项目总结、功能介绍、开发过程等。

四、答辩 PPT 制作与资料上传：制作答辩 PPT，整理并上传所有参赛资料，准备好项目的答辩。

五、应用功能的不足与改进建议：

1. 不足：用户登录与个人信息管理功能

- **问题：**当前的用户登录与个人信息管理功能较为简单，未能充分实现数据同步与个性化服务。用户的个人数据（如浏览历史、收藏菜谱、饮食偏好）无法在不同设备和功能模块之间共享与同步。
- **改进建议：**
 - **数据同步：**引入云端同步功能，确保用户在不同设备或平台登录时，可以自动同步个人信息、浏览历史和收藏菜谱。
 - **个性化服务：**结合用户的历史行为和偏好，为用户提供更加个性化的推荐和营养分析，提升用户粘性和体验。
 - **社交功能：**可考虑加入社交互动功能，用户可以与朋友分享菜谱、评论和评分，提高用户参与度。

2. 不足：营养分析的精准性与个性化

- **问题：**目前的营养分析模块虽然实现了基本的营养成分计算，但没有提供足够精准和个性化的优化建议。算法仅根据食材重量进行分析，忽略了更多个性化因素（如用户的具体健康目标、活动量等）。
- **改进建议：**
 - **增强算法精准性：**引入更复杂的营养分析算法，结合更多个性化信息（如年龄、健康状况、活动量等）提供更精准的建议。
 - **动态调整建议：**根据用户的饮食行为、体重变化等数据，实时调整营养建议，使其更加符合用户的健康目标。
 - **食材替代建议：**提供食材替代方案，帮助用户根据预算、健康需求等因素，选择更适合的替代食材。

3. 不足：购物清单功能的智能化与优化

- **问题：**购物清单功能虽然可以根据菜谱生成食材清单，但在食材管理上存在一定的局限性。比如，食材的智能合并（如姜末和生姜未能合并为一个食材）和个性化推荐仍不完善。
- **改进建议：**
 - **智能合并食材：**优化食材识别算法，能够根据不同名称和描述合并相同食材（如姜末和生姜合并为姜），避免重复添加。
 - **智能购物推荐：**基于用户的饮食偏好和过往购买记录，智能推荐用户可能需要购买的食材，减少用户的操作负担。
 - **价格对比与商家推荐：**整合线上超市或本地商家的 API，提供价格对比功能，帮助用户选择最合适的购买渠道和价格。

4. 不足：食谱更新与个性化推荐

- **问题：**菜谱推荐功能目前主要基于用户的搜索与选择，未能实现智能化的个性化推荐。且菜谱更新较为缓慢，用户可能感到内容单一。
- **改进建议：**
 - **智能推荐算法：**引入基于用户历史行为、兴趣偏好、健康目标的智能推荐算法，自动推送用户可能感兴趣的菜谱，提升用户粘性。
 - **菜谱更新机制：**与知名美食平台合作，定期更新菜谱内容，保证菜谱库的丰富性和时效性。也可以考虑支持用户提交自定义菜谱，增强社区互动。
 - **社交功能扩展：**加入用户之间的互动与交流功能，如分享菜谱、评论评分、举办烹饪挑战等，提升用户参与度和内容活跃度。