BlossomAccents

下载链接：https://www.jianguoyun.com/p/DZZs1tgQgo--CRjOq_4D

文档链接：https://hello-cloudbase-1gu7n3nx256c8a7e-1305329184.tcloudbaseapp.com/zly/#0（因为域名没有备案所以没法用，只能用默认的了）

操作视频：下载版本：https://www.jianguoyun.com/p/DQa9qWMQgo--CRiqrv4D（坚果云）

在线版本：https://tube.nocturlab.fr/videos/watch/86700019-ee9f-4c82-95be-6363a132708a（PeerTube private link）

设计

想法

做一个类似背单词的软件，在这里用户可以创建一个集合（比如命名为“东北话”），然后在这里上传短音频，一段音频对应它的标准释义。别的用户可以点击这些音频，看它的释义，听它用东北话怎么说出来。

该想法以“艺术与文化”主题中的“文化”部分为重点，符合我国地域辽阔方言众多的特性，契合了当代年轻人对传统文化家乡方言的求知欲和玩乐精神，也在一定程度上为有丰富方言知识的老年人开放了平台。将流传至今的语言文化与现代科技相结合，令其焕发新的生机。

本项目的核心功能为音频文件的录入、上传和下载。

应用场景

做这个项目的初衷是希望有更多的用户加入。用户行为可以分为两种

- 录音的，想到什么词汇就可以加入解释和音频。而且一个人可以创建多个集合，不同的地区语言都可以自由添加。
- 听的，点击一个集合进去，可以看到这个用户上传的各种音频和释义，点击可以听到别人的语音，可以见识到各个地区的语言文化差异。

实际问题

方言和普通话的转换比较麻烦，希望可以借这个软件开发出更加全面系统的词典库。另外听各个地区的人对同一个词汇的不同发音也是一件很有人文价值的事情。

功能划分

- 登录/注册
- 创建集合、删除自己的集合、修改集合名字
- 用集合名搜索集合
- 查看集合的具体信息
- 在自己的集合中加入音频片段和标准释义
- 修改用户名和头像
- 退出登录
- 查看自己创建的集合列表

技术

关键的几个：

- 数据库：cloudbase
- 文件存储：cloudbase
- 头像选取：image_picker
- 录音：flutter_audio_recorder
- 播放：audioplayers
- 本地文件：path_provider

cloudbase

使用flutter开发语言，版本为2.10.0

检验数据库和云存储的函数是否起作用可以在控制台页面直接查看。

配置

参考腾讯的文档，使用匿名登陆的方式，确保可以获取登陆成功的信息。

登录过程将放在主函数入口处，进入页面之前。

为了方便调用，将常用的数据库和文件存储对象放在一个全局文件中，如下：

```
import 'package:cloudbase_database/cloudbase_database.dart';
import 'package:cloudbase_storage/cloudbase_storage.dart';
//用户表
Collection userCollection;
//集合表
Collection listCollection;
//单个表
Collection audioCollection;
//存储
CloudBaseStorage storage;
CloudBaseDatabase db;
```

数据库

使用数据库之前先初始化，这一过程可以在cloudbase登录时进行。

本项目中用到两个表user和collection

user

内容

昵称userName、邮箱userEmail、密码userPwd、头像userImg，以及userId

操作

- 增：用户注册
- 查：查找用户名和密码用于登录、用userId查剩余值
- 改：修改头像和用户名

collection

内容

作者UserId、名称header、字典个数wordCount、词典列表audioList，以及集合id

注：这一部分的数据库设计有问题，需要再修改

操作

- 增：用户创建集合
- 删：作者删除集合
- 查：主页显示全部、根据用户找集合、根据标题关键词找集合、根据集合id找到对应的内容
- 改：增加audioList，作者修改集合名字

具体的使用都参考腾讯文档，这里以collection表的操作为例

```
class CollectionTable {
  //新建集合
  Future<String> addCollection(String header,  int time) async {
    var a=await listCollection.add({
      'header': header,
      'time': time,
      'userId': curUserId,
      'wordCount': 0,
      'audioList':[]
    });
    return a.id;
  }
  //删除集合
  deleteCollection(String listId) async{
    await listCollection.doc(listId).remove();
  }
  //修改名字
  modifyCollectionName(String listId,String newName) async{
    print(newName);
    print(listId);
```

```
    var a=await listCollection.doc(listId).update({
      'header':newName
    });
    print(a);
  }
  //根据id查找集合
  Future<List<ListClass>> getMyListById() async{
    List<ListClass> resultListItems=List<ListClass>();
    var value=await listCollection.where({
      'userId':curUserId
    }).get();
    var datas=value.data;
    for(var data in datas) {
      String header = data['header'];
      int wordCount = data['wordCount'];
      String userId = data['userId'];
      String listId=data['_id'];
      String authorName=await UserTable().getUserNameById(userId);
      resultListItems.add(ListClass(header,userId,
 wordCount,authorName,listId));
    }
    return resultListItems;
  }
}
```

云存储

云存储的使用过程是：上传文件到云端-获得fileId，通过fileId可以获得文件的实际下载链接。

本项目涉及到文件存储的功能：

- 用户上传头像（图片，可以获得下载url后直接存储到数据库中，因为flutter可以直接展示网络图片）
- 用户上传语音（音频文件不能直接读取，所以它需要将fileId存储到数据库中，用于后面的下载）
- 用户下载语音（通过数据库查询到fileId，再获得下载url后下载到本地，这一过程用到了path_provider）

代码如下：

```
class StorageMethod{
//音频上传
  Future<String> audioUpload(String filePath,String fileName) async {
    var res=await storage.uploadFile(
        cloudPath: 'userAudios/'+fileName+'.aac',
        filePath: filePath,
        onProcess: (int count, int total) {
          // 当前进度
          print(count);
```

```
            // 总进度
            print(total);
          }
      );
      return res.data.fileId;
    }
  //头像上传
    Future<String> avatarUpload(String imagePath) async{
      var res=await storage.uploadFile(
        cloudPath: 'userImgs/'+curUserId+'.png',
        filePath: imagePath,
        onProcess: (int count, int total) {
          // 当前进度
          print(count);
          // 总进度
          print(total);
        }
      );
      return res.data.fileId;
    }
  //音频下载
    audioDownload(String cloudPath,String localPath) async{
      await storage.downloadFile(
          fileId: cloudPath,
          savePath: localPath,
          onProcess: (int count, int total) {
            // 当前进度
            print(count);
            // 总进度
            print(total);
          }
      ).catchError((onError)=>print(onError));
    }
  }
```

路由跳转

使用fluro

路由管理页面。用index的跳转为例，其它页面类似。

```
//routers.dart 定义
import 'package:fluro/fluro.dart';
import 'routers_handler.dart';

///路由配置
class Routers {
  static String index = "/index";
```

```
  static void configureRoutes(FluroRouter router) {
    router.notFoundHandler = new Handler(
        handlerFunc: (BuildContext context, Map<String, List<String>>
params) {
          print("ROUTE WAS NOT FOUND !!!");
        });

    /// 第一个参数是路由地址，第二个参数是页面跳转和传参，第三个参数是默认的转场动画
    router.define(index, handler: indexHandler);
  }
}
```

```
//routers_handler.dart 定义实际的跳转页面
import 'package:fluro/fluro.dart';
import '../pages/index/index_screen.dart';

//跳转到主页
var indexHandler =  Handler(
    handlerFunc: (BuildContext context, Map<String, List<String>>
params) {
      return IndexScreen();});
//跳转到欢迎
var welcomeHandler =  Handler(
    handlerFunc: (BuildContext context, Map<String, List<String>>
params) {
      return WelcomeScreen();});
```

这样就可以方便地进行路由跳转。

登录注册

主要用的是shared preferences ,判断用户是否登录过。实际登录/注册时，是先从数据库中获取邮箱（作为不重复的用户名）和密码保存到全局变量map mockUsers中，然后将这个map和用户输入的用户名密码进行匹配。

和数据库类似的增删查改，只是像文件存储一样保存在手机里。

将操作封装到一个文件里，方便后面的调用

```
//shared_utils.dart
import 'package:blossom_accents/common/application.dart';
import 'package:shared_preferences/shared_preferences.dart';
//加
sharedAddData(String key,Object dataType,Object data) async{
  SharedPreferences prefs=await SharedPreferences.getInstance();
  switch(dataType){
    case bool:
```

```
        prefs.setBool(key, data as bool);break;
      case double:
        prefs.setDouble(key, data as double);break;
      case int:
        prefs.setInt(key, data as int);break;
      case String:
        prefs.setString(key, data as String);break;
      case List:
        prefs.setStringList(key, data as List<String>);break;
      default:
        prefs.setString(key, data as String);break;
    }
  }
  //返回
  Future<Object> sharedGetData(String key) async{
    SharedPreferences prefs=await SharedPreferences.getInstance();
    return prefs.get(key);
  }
  //清除当前登录信息（用于退出登录）
  Future<void>sharedDeleteAll() async{
    SharedPreferences prefs=await SharedPreferences.getInstance();
    prefs.remove(USER_EMAIL);
    prefs.remove(USER_LOGIN);
  }
```

主页

在进入主页之前会先采集到用户信息，将其显示在drawer中。除此以外，整个软件运行过程中会经常用到的用户唯一标识（即userId，邮箱并不方便数据库搜索）也在这里进行赋值。避免多次调用数据库。

使用的drawer组件是：flutter_slider_drawer

主页的组成是：body+menu。menu中有若干个类别：我的、设置、主页等，随着用户选中不同的menu，body会返回不同的页面。

换头像

主要用的是image_picker，有从相机拍照和从相册获取的功能（但是模拟手机不能覆盖全型号，其它型号的手机能不能用我也不清楚。）

获得照片后，会将该照片上传到云端，获得fileId，因为flutter可以直接用url展示图片，所以在数据库中存入该图片的实际链接即可方便地获取头像。

关键代码如下

```
  //avatar_change.dart
  class ImagePickerPage extends StatefulWidget {
```

```
  ImagePickerPage({Key key}) : super(key: key);
  _ImagePickerPageState createState() => _ImagePickerPageState();
}
class _ImagePickerPageState extends State<ImagePickerPage> {
  //记录选择的照片
  File _image;
  //拍照
  Future _getImageFromCamera() async {
    var image = await ImagePicker.pickImage(source: ImageSource.camera,
maxWidth: 400);
  }
  //相册选择
  Future _getImageFromGallery() async {
    var image = await ImagePicker.pickImage(source:
ImageSource.gallery);
  }

//  上传图片到服务器并保存为用户头像
  _uploadImage() async {
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
    );
  }
}
```

语音

录音

录音的流程：（判断作者）▶ 用户点击按钮▶ 输入名称并校验 ▶ 输入没问题后启动录音 ▶ 录音过程 ▶ 用户点击停止按钮 ▶ 上传到云端并加载数据库 ▶ 页面更新 ▶ 启动下一次录音

代码如下：

```
class RecorderView extends StatefulWidget {
  //onSaved为保存后启动的函数（见home中）
  final Function onSaved;
  //tableId 是这个list对应的id，用于后面的数据库存储
  final String tableId;
}
//录制的状态，用于更换按钮的显示信息
enum RecordingState {
  UnSet,
  Set,
```

```
  Recording,
  Stopped,
}

class _RecorderViewState extends State<RecorderView> {
  FlutterAudioRecorder audioRecorder;
  @override
  void initState() {
    super.initState();
    //检查权限
  }
  //页面展示
  @override
  Widget build(BuildContext context) {
    return Stack(
    );
  }
//按下按钮之后的操作
  Future<void> _onRecordButtonPressed() async {
    switch (_recordingState) {
    }
  }
//初始化录音-即找到对应的文件位置
  _initRecorder(String name) async {
    Directory appDirectory = await getApplicationDocumentsDirectory();
    //这个是录音的，保存的文件放缓存 /userAudio/tableId/下面，用explain作为文件名。
      audioRecorder =
          FlutterAudioRecorder(filePath, audioFormat: AudioFormat.AAC);
    await audioRecorder.initialized;
  }
//开始录音
  _startRecording() async {
    await audioRecorder.start();
  }
//结束录音
  _stopRecording() async {
    await audioRecorder.stop();
    String explain=filePath.substring(
        filePath.lastIndexOf('/') + 1, filePath.lastIndexOf('.'));
    //上传到云端。
    String audioId=await StorageMethod().audioUpload(filePath,explain);
    //加到collection数据库里头
    CollectionTable().addSingleAudio(widget.tableId, audioId,explain);
    widget.onSaved();
  }
    //状态修改
  Future<void> _recordVoice() async {
  }
```

```
}
```

播放

设置语音条

```
Future<void> _onPlay({@required String filePath, @required int index})
async {
    AudioPlayer audioPlayer = AudioPlayer();
    if (!_isPlaying) {
      audioPlayer.play(filePath, isLocal: true);
      setState(() {
        _selectedIndex = index;
        _completedPercentage = 0.0;
        _isPlaying = true;
      });
      audioPlayer.onPlayerCompletion.listen((_) {
        setState(() {
          _isPlaying = false;
          _completedPercentage = 0.0;
        });
      });
      audioPlayer.onDurationChanged.listen((duration) {
        setState(() {
          _totalDuration = duration.inMicroseconds;
        });
      });

      audioPlayer.onAudioPositionChanged.listen((duration) {
        setState(() {
          _currentDuration = duration.inMicroseconds;
          _completedPercentage =
              _currentDuration.toDouble() / _totalDuration.toDouble();
        });
      });
    }
  }
```

权限

本项目是安卓应用，因此只关注了android端的权限管理，ios端和web端等开发可以自行参考文档。安卓平台的权限管理位置:

blossom_accents/android/app/src/main/AndroidManifest.xml

在application同级的位置加入

```
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

版本管理

位置：blossom_accents/android/app/build.gradle

（安卓）minSdkVersion 23

总结

数据库和云存储的功能实现过程中突出的一点就是控制异步操作。由于不熟练，一开始我在开发过程中经常遇到先加载页面，后读取数据库等问题（当然页面会报错）。但实际上异步操作的优势在我的代码中并没有得到完全体现——我几乎是把它塞在整个主线程的时间线里处理的，凡是有async必有await。

其次，在项目初期我的代码也就是一个main函数带动所有，后来参阅了许多项目的代码，才逐渐学会合理的文件布局，对项目的理解也更加深刻，逐渐认识到一开始设计的缺漏。除了参考别人代码的架构和使用方式，在pub.dev上寻找别人封装好的功能和插件也是我项目开发过程中的一大乐趣。合理的调用可以大大减少工作量，但也可能会造成项目累赘——有太多代码只使用了一次，仅仅是为了一个微不足道的小功能。

最后，在写的过程中是痛苦伴随着快乐的。痛苦在“我写不下去了”“我现在换主题还来得及吗”之间反复横跳，快乐在攻克完一个难点后的轻松，以及我最终测试可以录入语音并播放的时候。

拓展

在让朋友试了一下我的应用之后，总结出一些可以增加的功能：

- 看看国内有无成熟的方言识别接口，让用户可以测试识别自己的方言语系
- 优化注册功能，增加邮箱验证码和找回密码功能（不用手机，手机号注册我的一生之敌）
- 用云函数提高数据库的搜索速度
- 增加收藏功能，并累计收藏数量
- 优化页面跳转，采用更新state的方式
- 将音频独立出来，可以用关键词搜索语音词条
- 优化音频文件的下载方式
- 用户可以在单个语音条下面进行评论
- 首页的更新以用户最后上传语音条的时候为基准
- 允许用户删除指定语音条