猫扑走路 APP

一、开发环境

1) 开发系统：Windows，Mac OS，Linux

2) 编程 IDE：Android Studio 吗，Sublime Text

3) 编程语言：Java，html，css，javascript，sql

二、项目阐述

1. 名称

猫扑跑步

2. 简介

猫扑跑步是一款跑步计数软件，支持音乐播放和对跑步轨迹进行绘画显示，每次跑步结束后会对跑步记录进行保存，并且显示当次跑步的步数，时间，以及通过以 60kg，170cm 正常人为模板，计算出消耗卡路里，跑步距离和步速。软件的主界面可以通过跳转查询历史记录，历史记录可以查询运动轨迹和跑步距离，步速，卡路里等数据。同时，软件支持对本机音乐进行播放以及软件内置 7 首歌曲支持循环播放，使跑步不会那么枯燥。

3. 功能

1.**计时、计步器**，当点击开始按钮后，计时器开始工作，统计跑步时间和总步数；当计时器暂停后，暂停计时和计步；重新点击开始后，应用继续统计步数和时间。点击停止之后，计时器清零并把时间和步数存入数据库。支持后台计时和计步。

2.**地图定位**，可以定位到当前位置，并把地图放缩到适合的比例。在网络或 GPS 信号正常时，应用支持实时更新定位。根据手机朝向，应用主页面的黑色指针即为当前前进的方向。

3.**绘制跑步轨迹**：

当点击开始按钮后，会在地图上绘制红色跑步轨迹，当点击暂停的时候，轨迹会停止绘制，点击暂停后，会将轨迹数据存储进数据库。同时在统计页面显示刚才跑步的轨迹。

4. **起终点标记。**在应用开始/结束时，会在起点/终点加上绿色/红色标记。

5. **统计时长速度卡路里**：点击停止，APP 会跳转到新页面并且根据此次跑步的时间，步数，距离，设计算出平均速度，热量等信息，同时显示出此次跑步的轨迹。

6. **数据库**：每次跑步的时间步数还有时长，跑步的轨迹都会存进数据库，点击跑步记录中的任意一项，会跳转到详细记录页面，里面记录了历史跑步记录的始终点和轨迹，同时会显示此次跑步的步数，时长，卡路里等信息。

7. **Widget 及静态广播**：在手机桌面新建 widget 后，每次跑步结束后，会自动更新桌面 Widget，显示最近一次跑步的时间、步数和消耗的卡路里。同时在通知栏静态广播相同的信息

8. **后台音乐播放功能**：点击音乐选项会播放默认音乐，如果想更换歌曲，可选择更换音乐 BUTTON，从本地文件夹中选取新的歌曲进行播放。

9. **开场图片动画播放及后台测量数据**：APP 刚打开的时候会显示几组健身图片的 2D 动画，APP 点击退出之后，会继续在后台绘制轨迹以及步数时间的测量。

10.**可以通过网页登陆注册并查看步数：**登陆 catpu 的网页，可以进行登陆注册用户，并在网页中查看近几天的步数。

4. 亮点

1.技术要求

(1)界面框架设计（用户友好，控件布局合理，美观）

(2)事件处理的实现（点击事件、activity 跳转等）

(3)SQLite、ContentProvider 等数据存储方式

(4)Notification、Widegt

(5)Broadcast

(6)Service

(7)传感器使用（GPS，步数传感器）

(8)2D、3D 动画应用

(9) OverlayOptions 绘制轨迹（百度地图 API）

(10)服务器后端采用 expressjs 框架

(11)后端链接 mysql 数据库，并将用户 session 存入 mysql 缓存池

(12)后端使用 gulp 对资源进行更新及优化处理

(13)web 前端使用 vue 框架及 muse-ui 框架

(14)使用 webpack 对前端进行管理打包，配置跨域请求

2.实现亮点

(1) 本次实验基本囊括了之前课程所学的大部分知识，上面技术要求已经介绍

(2) 此次组员通过查阅资料，系统可通过加速度传感器获取的数值进行胳膊甩动动作的判断，从而达到测量步数的要求，以此达到精确测量步数的目的。

(3) 根据百度地图 API 的 OverlayOptions 以及 LIST<latlng>实现了实时绘制跑步的轨迹 ,这是跑步软件的一个基础也是精髓核心

(4) 数据库实现了存储 LIST<latlng>的功能，这样不仅可以记录跑步数据，也能在任何时刻显示以前跑步自己跑步的轨迹。

5. 实验步骤&功能对应代码解释

// 贴代码解释

1.开机动画

开场动画与 recycleView 不同，可以设置不同的动画播放方式，5s 进行翻转。由于 github 上提供了很多开源的安卓库，可以调用 github 上的库对图片进行处理：

```
compile 'com.bigkoo:convenientbanner:2.0.5'
compile 'com.ToxicBakery.viewpager.transforms:view-pager-transforms:1.2.32@aar'
compile 'com.tbruyelle.rxpermissions:rxpermissions:0.9.0@aar'
compile 'io.reactivex:rxandroid:1.2.1'
compile 'io.reactivex:rxjava:1.1.6'
compile files('libs/universal-image-loader-1.9.5.jar')
```

对软件中 mipmap 图片进行加载 ,viewpager.transform 库可以进行设置动画的翻转方式，设置了 14 种播放方式，可以通过点击 BeginActivity 的 button 进行播放方式的切换：

```
//各种翻页效果
transformerList.add(DefaultTransformer.class.getSimpleName());
transformerList.add(AccordionTransformer.class.getSimpleName());
transformerList.add(BackgroundToForegroundTransformer.class.getSimpleName());
transformerList.add(CubeInTransformer.class.getSimpleName());
transformerList.add(CubeOutTransformer.class.getSimpleName());
transformerList.add(DepthPageTransformer.class.getSimpleName());
transformerList.add(FlipHorizontalTransformer.class.getSimpleName());
transformerList.add(FlipVerticalTransformer.class.getSimpleName());
transformerList.add(ForegroundToBackgroundTransformer.class.getSimpleName());
transformerList.add(RotateDownTransformer.class.getSimpleName());
transformerList.add(RotateUpTransformer.class.getSimpleName());
transformerList.add(StackTransformer.class.getSimpleName());
transformerList.add(ZoomInTransformer.class.getSimpleName());
transformerList.add(ZoomOutTranformer.class.getSimpleName());
```

设置 convenientBanner 定义广告栏播放形式，在此新建一个本地图片 Holder：

```
initImageLoader();
loadTestDatas();
//本地图片例子
convenientBanner.setPages(
        (CBViewHolderCreator) () -> {
            return new LocalImageHolderView();
        }, localImages)
        //设置两个点图片作为翻页指示器，不设置则没有指示器，可以根据自己需求自行配合自己的指示器,不需要圆点指示器可用不设
        .setPageIndicator(new int[]{R.mipmap.ic_page_indicator, R.mipmap.ic_page_indicator_focused})
        .setOnItemClickListener(this);
```

Multiple implementations

```
public class LocalImageHolderView implements Holder<Integer> {
    private ImageView imageView;
    @Override
    public View createView(Context context) {
        imageView = new ImageView(context);
        imageView.setScaleType(ImageView.ScaleType.FIT_XY);
        return imageView;
    }

    @Override
    public void UpdateUI(Context context, int position, Integer data) {
        imageView.setImageResource(data);
    }
}
```

2.计时器和计步器：

计时器：单独使用一个特定线程管理计时。使用一个变量`chronometerState`记录当前计时器状态。

使用 Handler 与 Runnable 的消息传递机制来更新 UI 线程的信息，以在主页面更新计时器的数字。每 0.01 秒更新一次数据（Thread.sleep(10)，并发出信息），Handler 根据消息类型执行相应动作（更新当前时间或重置时间），从而更新主界面 UI。

```
private Runnable runnable = new Runnable() {
    @Override
    public void run() {
        while (true) {
            if (chronometerState == STATE_START) {
                try {
                    Thread.sleep(10);
                    Message message = new Message();
                    message.what = STATE_START;
                    handler.sendMessage(message);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            } else if (chronometerState == STATE_STOP) {
                Message message = new Message();
                message.what = STATE_STOP;
                handler.sendMessage(message);
                break;
            }
        }
    }
};
```

```
private Handler handler = handleMessage(message) -> {
        switch (message.what) {
            case STATE_START:
                currentTime++;
                break;
            case STATE_STOP:
                currentTime = 0;
                stepCount = 0;
                stepStamp = 0;
                break;
        }
        setTime(currentTime);
        setStep(stepCount);
        super.handleMessage(message);
};
```

```
            } else if (chronometerState == STATE_START) {
                if (clockThread == null) {
                    clockThread = new Thread(runnable);
                }
                chronometerState = STATE_PAUSE;
                buttonStartAndPause.setText("开始");
                LatLng desLatLng1 = new LatLng(0, 0);
                map_database.insertdb(order, desLatLng1.latitude, desLatLng1.longitude);
            } else if (chronometerState == STATE_PAUSE) {
                stepStamp = actualStep - stepCount;
                if (clockThread == null) {
                    clockThread = new Thread(runnable);
                }
                chronometerState = STATE_START;
                buttonStartAndPause.setText("暂停");
            }
    });
```

根据这个变量来判断当前计时器的状态，然后执行对应的操作。点击开始后，标记开始位置，把位置信息存入数据库，以用于标记跑步路线。

在开始按钮的回调函数中，点击开始后，开始一次新计时的。开启一个多线程计时。

```
clockThread = new Thread(runnable);
clockThread.start();
```

若点击停止，则置空这个线程，关闭计时，并且重置 UI 界面。

计步器：从 sensorManager 处获取计步器的实例

```
private Sensor stepCounter;
```

```
stepCounter = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
```

在 onResume 方法中注册计步器 stepCounter

```
}
@Override
protected void onResume() {
    //在activity执行onResume时执行mMapView. onResume (), 实现地图生命周期管理
    mMapView.onResume();
    // register magnetic and accelerometer sensor into sensor manager (onResume
    mSensorManager.registerListener(mSensorEventListener, mMagneticSensor,
            SensorManager.SENSOR_DELAY_GAME);
    mSensorManager.registerListener(mSensorEventListener, mAccelerometerSensor,
            SensorManager.SENSOR_DELAY_GAME);
    // register location update listener
    if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
        mLocationManager.requestLocationUpdates(providerName, 0, 0, mLocationListener);
    }
    super.onResume();
    if (` != null) {
        sensorManager.registerListener(this, stepCounter, SensorManager.SENSOR_DELAY_UI);
    } else {
        Toast.makeText(this, "Count sensor not available!", Toast.LENGTH_SHORT).show();
    }
}
```

当应用打开后，actualStep 为记录当前走路步数的变量，这个步数值是从 onSensorChanged 的 event 处获取的。如果跑步状态为开始 STATE_START，我们要更新当前已跑的总步数。

```
@Override
public void onSensorChanged(SensorEvent event) {
    actualStep = (long) event.values[0];
    if (chronometerState == STATE_START) {
        stepCount = actualStep - stepStamp;
    }
}
```

在这之前，先要点击开始跑步按钮，那么在按钮的点击回调函数中，会重置 stepStamp（我们在点击按钮时的跑步步数）

```
stepStamp = actualStep;
```

然后每当 sensorChanged 的时候，就会自动更新步数了。

当然我们点击结束的时候，因为状态变为 STATE_STOP，因此 stepCount 不再更新。当下次再点击开始的时候，则再次统计。

3.后台音乐

Filepicker 可以调用 github 上提供的开源安卓库：

```
compile 'com.nbsp:library:1.1'
```

```
// 调用github中安卓开源库进行文件管理器的打开
search_.setOnClickListener((v) -> {
        new MaterialFilePicker()
                .withActivity(MainActivity.this)
                .withRequestCode(1)
                .withHiddenFiles(true) // Show hidden files and folders
                .start();
});
```

MaterialFilePicker 是对根目录中的文件进行过滤，因此如果使用 filter 函数无法彻底过滤音乐文件，所以在这里不设置过滤条件。

打开文件管理器选择文件，选中文件之后会返回 result_ok，将文件载入到 mediaPlayer：

```
// 打开文件管理器后选择音乐文件加载进入mediaPlayer
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    // 文件存在并且成功选择
    if (requestCode == 1 && resultCode == RESULT_OK) {
        try {
            String filePath = data.getStringExtra(FilePickerActivity.RESULT_FILE_PATH);
            Toast.makeText(this, filePath, Toast.LENGTH_SHORT).show();
            // Do anything with file
            if (mediaPlayer.isPlaying()) {
                music.setText("播放音乐");
                mediaPlayer.reset();
                mediaPlayer.setDataSource(filePath);
                mediaPlayer.prepare();
            } else {
                mediaPlayer.setDataSource(filePath);
                mediaPlayer.prepare();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4.数据库存储跑步记录

```
public class Run {
    private String date;
    private String time;
    private String distance;
    private String order;
```

跑步记录包括四项，分别是日期，跑步时长，跑步步数，以及第几次记录。

```
public class Run_Adapter extends ArrayAdapter<Run> {
    private int resourceId;

    public Run_Adapter(Context context, int textViewResourceId, List<Run> objects) {
        super(context, textViewResourceId, objects);
        resourceId = textViewResourceId;
    }


    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Run runlist = getItem(position);
        View view;
        if (convertView == null) {
            view= LayoutInflater.from(getContext()).inflate(resourceId, null);
        } else {
            view = convertView;
        }
        TextView date = (TextView)view.findViewById(R.id.date);
        date.setText(runlist.getDate());
        TextView time = (TextView)view.findViewById(R.id.time);
        time.setText(runlist.getTime());
        TextView dis = (TextView)view.findViewById(R.id.distance);
        dis.setText(runlist.getDistance());
        return view;
    }
}
```

使用自定义 Adapter 来提高了 Listview 效率，重写了父类的一组构造函数，用于将上下文、ListView 子项布局的 id 和数据都传递进来。另外又重写了 getView()方法，这个方法在每个子项被滚动到屏幕内的时候会被调用。getView 方法中，首先通过 getItem()方法得到当前项的 RunList 实例，然后使用 LayoutInflater 来为这个子项加载我们传入的布局，接着调用 ViewfindViewById()方法分别获取到 TextView 的实例，并分别为他们附上 item 的四项属性最后将布局返回。

数据库采用的是 SQLite 存储，方法实现和实验 8 类似，再此不再展示。

```
while(cursor.moveToNext()){
    times++;
    date = cursor.getString(cursor.getColumnIndex("date"));
    time = cursor.getString(cursor.getColumnIndex("time"));
    distance = cursor.getString(cursor.getColumnIndex("distance"));
    String a = cursor.getString(cursor.getColumnIndex("_order"));
      Toast.makeText(DB_Activity.this, a+"", Toast.LENGTH_SHORT).show();
    String h = time.substring(0,2);
    int hi = Integer.parseInt(h);
    String m = time.substring(3,5);
    int mi = Integer.parseInt(m);
    String s = time.substring(6,8);
    int si = Integer.parseInt(s);

    second += si;
    if (second >= 60) {
        second -= 60;
        min++;
    }
    min += mi;
    if (min >= 60) {
        min -= 60;
        hour++;
    }
    hour += hi;
```

数据库界面上方的总的数据统计，是通过 select*遍历整个数据库，然后通过 cursor 循环获取每个列的值，通过几个变量累加，最后赋值给 TextView 控件即可。

5. 数据库存储跑步轨迹

跑步轨迹同跑步记录是两个数据库来存储的。因为跑步记录是通过 List<LatIng>来记录各个点的坐标的，然而这个类型无法直接通过 SQL 存储，因此采用间接的方法，通过存储每个坐标的经纬度（float 类型），因为每次跑步轨迹是有很多坐标组成的，所以需要给每个点加个 order 的记录来表示是哪一次的跑步记录。具体实现如下：

```
public class Map_DB extends SQLiteOpenHelper{
    private static final String DB_NAME = "MapDB";
    private static final String TABLE_NAME = "MapTable";
    private static final int DB_VERSION = 1;

    private static String CREATE_TABLE = "CREATE TABLE " + TABLE_NAME
            +" (id INTEGER PRIMARY KEY,_order INTEGER,latitude DOUBLE,longitude DOUBLE)";
    private Context mcontext;
    public Map_DB(Context context, String name, SQLiteDatabase.CursorFactory cursorFactory, int v) {
        super(context, name, cursorFactory, DB_VERSION);
        mcontext = context;
    }
    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) { sqLiteDatabase.execSQL(CREATE_TABLE); }
    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int il) {

    }

    public void insertdb(int order, double la, double lo) {
        SQLiteDatabase db = getWritableDatabase();
        ContentValues cv = new ContentValues();
        cv.put("_order", order);
        cv.put("latitude", la);
        cv.put("longitude", lo);
        db.insert(TABLE_NAME, null, cv);
        db.close();
    }
```

点击数据库 item 会绘制出轨迹在地图上面。是通过如下代码实现的。首先是数据通过 order 查询出

此

次跑步的所有的点：

```
    public List<LatLng> selectdb(int order) {
        SQLiteDatabase db = getReadableDatabase();
        List<LatLng> polylines = new ArrayList<~>();
        Cursor cursor = db.rawQuery("select * from MapTable", null);
        int judge;
        while(cursor.moveToNext()){
            judge = cursor.getInt(cursor.getColumnIndex("_order"));
            if (judge == order) {
                double la = cursor.getDouble(cursor.getColumnIndex("latitude"));
                double lo = cursor.getDouble(cursor.getColumnIndex("longitude"));
                LatLng l = new LatLng(la, lo);
                polylines.add(l);
            }
        }
        db.close();
        return polylines;
    }
}
```

```
int a = getIntent().getIntExtra("orderr", 0);
List<LatLng> guiji = map_database.selectdb(a);
```

此函数返回一个 List<LatLng>的坐标点集合。然后通过将这些点击通过特殊的方法绘制成轨迹（后面
会讲）。置于点击的存储是通过一个每 250ms 更新一次的线程，将获取的坐标存进数据库的。

6. 绘制实时轨迹：

1）创建一个 LatLng 的 List 用来记录本次跑步经过的坐标。

2）在开始跑步之后，每 0.25 秒将当前坐标加入 List，并使用 OverlayOptions 将当前坐标和 List 中记录的上一个坐标连成折线，显示在地图上。

```
LatLng desLatLng = convertLocationToLatLng(currentLocation);
polylines.add(desLatLng);

List<LatLng> temp = new ArrayList<LatLng>();
temp.add(polylines.get(polylines.size()-1));
LatLng desLatLng = convertLocationToLatLng(currentLocation);
temp.add(desLatLng);

polylines.add(desLatLng);
    OverlayOptions ooPolyline = new PolylineOptions().width(5)
            .color(Color.RED).points(temp);
    mMapView.getMap().addOverlay(ooPolyline);
```

3）当点击暂停并再次开始跑步后，保留暂停前的轨迹，清空 List 重新记录坐标绘制新的轨迹。

7.绘制历史轨迹：

1）在开始跑步后，每 0.25 秒将当前坐标存入数据库。

```
map_database.insertdb(order, polylines.get(polylines.size()-1).latitude, polylines.get(polylines.size()-1).longitude);
```

2）当点击暂停时，将一个（0,0）坐标存入数据库，作为暂停的标识。

```
LatLng desLatLng1 = new LatLng(0, 0);
map_database.insertdb(order, desLatLng1.latitude, desLatLng1.longitude);
```

3）在 Message 页面，读取数据库中的坐标 List 并绘制轨迹。

```
List<LatLng> guiji = map_database.selectdb(a);

int i = 0;
List<LatLng> temp1 = new ArrayList<~>();
LatLng pause = new LatLng(0,0);
while (i < guiji.size()) {
    if (guiji.get(i).toString().equals(pause.toString())&&temp1.size() >= 2) {
        OverlayOptions ooPolyline = new PolylineOptions().width(5)
                .color(Color.RED).points(temp1);
        mMapView2.getMap().addOverlay(ooPolyline);
        temp1.clear();
        i++;
        continue;
    }
    if (guiji.get(i).toString().equals(pause.toString())) {
        temp1.clear();
        i++;
        continue;
    }
    if (i == guiji.size()-1 && temp1.size() >= 2) {
        OverlayOptions ooPolyline = new PolylineOptions().width(5)
                .color(Color.RED).points(temp1);
        mMapView2.getMap().addOverlay(ooPolyline);
        temp1.clear();
        i++;
        continue;
    }
    temp1.add(guiji.get(i));
    i++;
}
```

8. 起终点的标记和在跑步轨迹页面中的显示：

使用百度地图的 Overlay 类存储开始、终止点，使用 ArrayList 存储中间的点。

```
private Overlay startOverlay = null;
private Overlay endOverlay = null;
private ArrayList<Overlay> pathOverlay = new ArrayList<>();
```

点击跑步开始后，在图中标记初始点，并使用 addOverlay 往图中标记出跑步初始点。同理，在点击停止时，往图中标记终点。

```
buttonStartAndPause.setOnClickListener((view) -> {
        updateTop();
        if (chronometerState == STATE_STOP) {
            mToggleButton.setChecked(true);
            // init startIcon
            LatLng limit = convertLocationToLatLng(currentLocation);
            startOverlay = mMapView.getMap().addOverlay(new MarkerOptions().position(limit)
                    .icon(BitmapDescriptorFactory
                            .fromResource(R.drawable.start)));
            // clear path
            for (Overlay overlay : pathOverlay) {
                overlay.remove();
            }
            pathOverlay.clear();
            stepStamp = actualStep;
```

在跑步数据记录的线程中，往数据库和本地变量中插入并在图中画出途经点。

```
Runnable runnable1 = new Runnable() {
    @Override
    public void run() {
map_database.insertdb(order, polylines.get(polylines.size()-1).latitude, polylines.get(polylines.size()-1).longitude)

polylines.add(desLatLng);
OverlayOptions ooPolyline = new PolylineOptions().width(5)
        .color(Color.RED).points(temp);
Overlay point = mMapView.getMap().addOverlay(ooPolyline);
pathOverlay.add(point);
```

9.静态广播

通过在 messageActivity 进行计算跑步距离、卡路里、跑步速度（均以 60kg，170cm 正常人为模板）：

```
String time_ = getIntent().getStringExtra("time");
String []temp = time_.split(":");
int mins = (Integer.parseInt(temp[0])) * 3600
        + Integer.parseInt(temp[1]) * 60
        + (Integer.parseInt(temp[2]));
String step_ = getIntent().getStringExtra("step");

// 以60公斤1m7身高的正常人，通过步数测里距离
double distance_ = Integer.parseInt(step_) * step_distance / 1000;
time.setText(time_);

// 测里数据保持两位小数的形式显示
velocity.setText(String.format("%.2f", mins/distance_/60));
distance.setText(String.format("%.2f", distance_));
calorie.setText(String.format("%.2f", Math.ceil(distance_*0.06)));
back_btn.setOnClickListener((v) -> { MessageActivity.this.finish(); });
```

获取数据后发送静态广播：

```
// Static Broadcast
Intent intent = new Intent(STATICACTION);
Bundle bundle = new Bundle();
bundle.putString("step", String.valueOf(step_));
bundle.putString("time", time_);
bundle.putString("calorie", String.format("%.2f", Math.ceil(distance_*0.06)));
intent.putExtras(bundle);
sendBroadcast(intent);
```

接收到广播后显示 notification：

```
if (intent.getAction().equals("com.example.shick.stepcounter.MyBroadcast")) {
    Bundle bundle = intent.getExtras();
    String step = bundle.getString("step");
    String calorie = bundle.getString("calorie");
    String time = bundle.getString("time");
    Bitmap icon = BitmapFactory.decodeResource(context.getResources(), R.mipmap.new_con);
    PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, new Intent(context, MainActivity.class), 0);
    Notification.Builder builder = new Notification.Builder(context);
    String text = "跑步步数: " + step + "  用时: " + time + "  卡路里 " + calorie;
    builder.setContentTitle("猫扑跑步").setContentText(text).setTicker("您有一条新消息").setLargeIcon(icon).setSmallIcon(R.mipmap.new_con).setAutoCancel(true).setDefaults(Notification.DEFAULT_SOUND)
            .setContentIntent(pendingIntent);
    NotificationManager manager = (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
    Notification notify = builder.build();
    manager.notify(0, notify);
}
```

10.Widget：

在项目中新建一个 App Widget

```
public class StepCounterWidget extends AppWidgetProvider {

    static void updateAppWidget(Context context, AppWidgetManager appWidgetManager,
                                int appWidgetId) {
```

在重载的 onUpdate 方法中，通过 PendingIntent 跳转 Activity 和 RemoteViews 控制 Widget，实现点击 Widget 返回主界面。

```
@Override
public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
    // There may be multiple widgets active, so update all of them
    super.onUpdate(context, appWidgetManager, appWidgetIds);
    Intent clickInt = new Intent(context, MainActivity.class);
    PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, clickInt, 0);
    RemoteViews remoteViews = new RemoteViews(context.getPackageName(), R.layout.step_counter_widget);
    remoteViews.setOnClickPendingIntent(R.id.widget_image, pendingIntent);
    appWidgetManager.updateAppWidget(appWidgetIds, remoteViews);
}
```

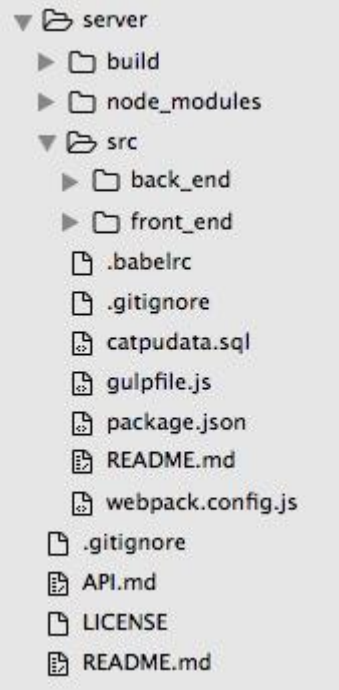
重载的 onReceive 方法中，使用 remoteViews 获取 Widget，因为在静态广播中，会发出关于跑步数据的更新数据，因此只需根据 Intent 的消息类型判断是否是静态广播传来的数据，若是，则更新 Widget 的内容。

```
@Override
public void onReceive(Context context, Intent intent) {
    super.onReceive(context, intent);
    RemoteViews remoteViews = new RemoteViews(context.getPackageName(), R.layout.step_counter_widget);
    AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
    int [] widgetIDs = appWidgetManager
            .getAppWidgetIds(new ComponentName(context,
                    StepCounterWidget.class));
    Bundle bundle = intent.getExtras();

    if (intent.getAction().equals(MessageActivity.STATICACTION)) {
        remoteViews.setTextViewText(R.id.step, bundle.getString("step") + "步");
        remoteViews.setTextViewText(R.id.time, bundle.getString("time"));
        remoteViews.setTextViewText(R.id.calorie, bundle.getString("calorie") + "卡");
        appWidgetManager.updateAppWidget(widgetIDs, remoteViews);
    }
}
```

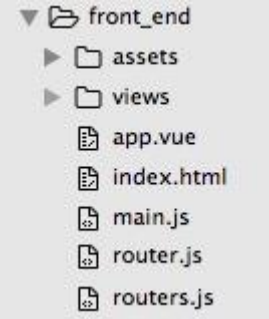
Widget 的 UI 即显示最新跑步的步数、时间和卡路里，以及一个小图标。

11.服务器框架



源代码在 src 文件夹，build 文件夹存放经 gulp 优化后资源，node_modules 是 nodejs 的模块，其他文件为说明、配置等。src 中有 back_end 和 front_end 文件夹，分别存放服务器后端和 web 前端代码。

12.web 前端



前端有存放图片等资源的 assets 文件夹及存放子页面的 views 文件夹。

app.vue 为主模块，index.html 作为前端页面的入口， main.js 调用 vue 模块，作为前端服务器启动的入口，在 webpack.config.js 中配置如下:

```
const webpack = require('webpack')
const path = require('path')
const HtmlwebpackPlugin = require('html-webpack-plugin')

module.exports = {
  entry: './src/front_end/main.js',
  output: {
    path: path.resolve(__dirname, './build/front_end'),
    publicPath: '/',
    filename: 'bundle.js'
  },
  module: {...
  },
  plugins: [...
  ],
  resolve: {
    alias: {
      'vue$': 'vue/dist/vue.common.js'
    }
  },
  devServer: {
    historyApiFallback: true,
    noInfo: true,
    hot: true,
    inline: true,
    proxy: {
      '*': {
          target: 'http://localhost:3002',
          host: 'localhost',
          secure: false,
      }
    }
  }
}
```

Bundle.js 是 webpack 对前端资源的打包，使浏览器对资源作“一次请求”，devServer 是代理服务器的配置，目标为后端 api 的 url

router.js 和 routers.js 是对前端页面路由的配置

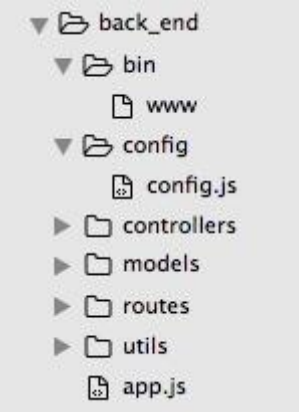
页面实现如下：

```
1   <template>
2       <div id='login-register-page'>
46      </div>
47  </template>
48
49
50
51  <script>
52  import md5 from 'md5'
53
54  export default {
128 }
129 </script>
130
131
132 <style>
272
273 </style>
```

Vue 中 template 标签里类似 html 的标签，script 标签中 export default 里 data 属性是对虚拟 DOM 数据的双向绑定，methods 方法可以处理事件等，其中还有 vue 的生命周期的属性等

Style 标签通过 css 语言调整样式

13．服务器后端



bin 中 www 作为 express 服务器启动项之一，其开启端口并对端口进行监听

```
var port = normalizePort(process.env.PORT || '3002');
app.set('port', port);

/** ...

var server = http.createServer(app);

/** ...

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);

/** ...

function normalizePort(val) { ...
}

/** ...

function onError(error) { ...
}

/**
 * Event listener for HTTP server "listening" event.
 */

function onListening() {
  var addr = server.address();
  var bind = typeof addr === 'string'
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('Listening on ' + bind);
}
```

config.js 是连接数据库，session，cookie 等配置

utils 连接数据库池

```
const mysql = require('mysql')
const Promise = require('bluebird')
const config = require('../config/config')
const conn = require('mysql/lib/Connection')
const po = require('mysql/lib/Pool')

Promise.promisifyAll(conn.prototype)
Promise.promisifyAll(po.prototype)

let getConnection = () => {
  return pool.getConnectionAsync()
    .disposer(connect => {
        connect.release()
    })
}

let dbconfig = Object.assign({ connectionLimit: 10 }, config.db)
let pool = mysql.createPool(dbconfig)

exports.queryDB = (sql, vals) => {
  return Promise.using(getConnection(), connect => {
    return connect.queryAsync(sql, vals)
  })
}

exports.pool = pool
```

Models 通过 sql 对数据库查询

```
exports.retrieveData = (itemVal, item) => {
  let sql =
        'SELECT * \n' +
        'FROM userlist \n' +
        'WHERE ' + item + ' = ? \n' +
    ';'
  let vals = [itemVal]
  return query(sql, vals)
}
```

Controller 调用 model，实现了 api 的逻辑

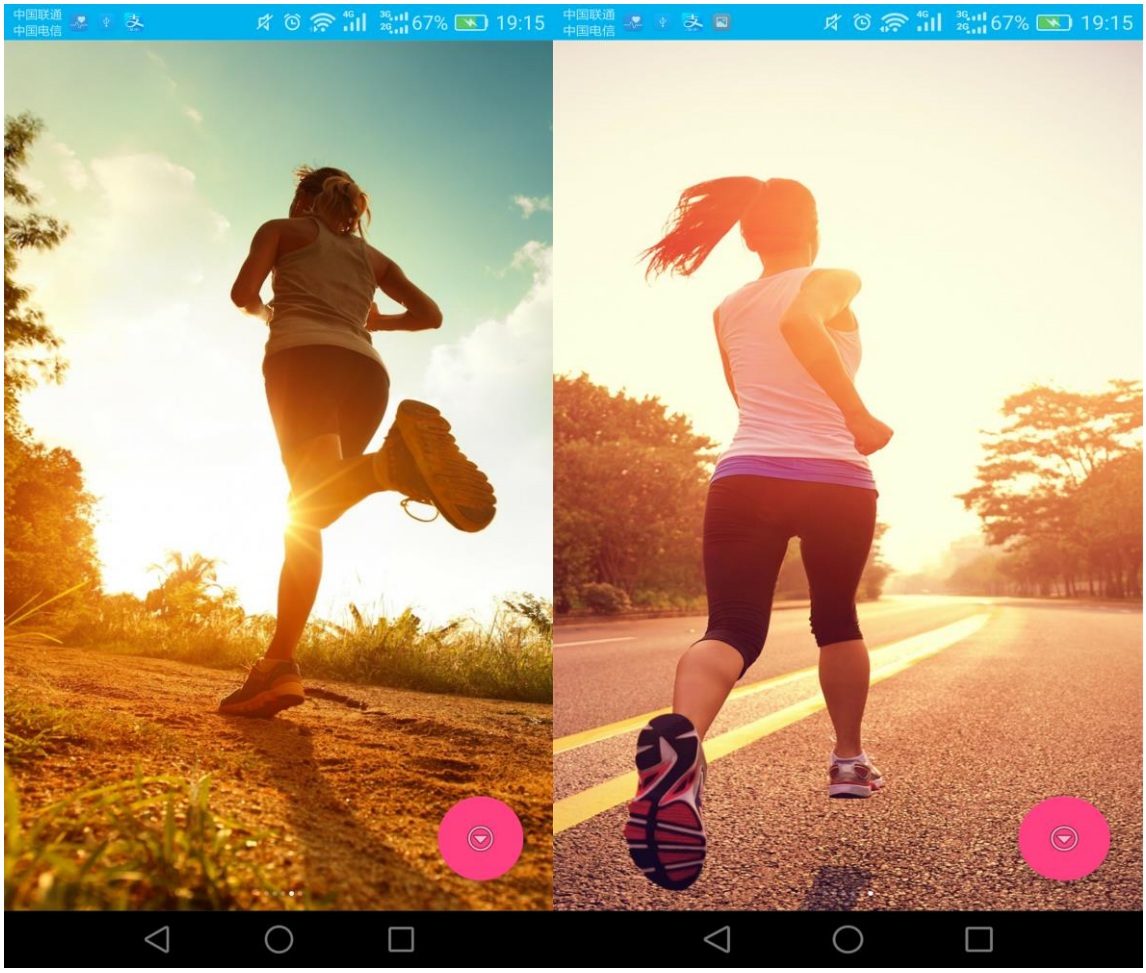
```
exports.register = (req, res, next) => {
  user_model.retrieveData(req.body.username, 'username')
    .then(([user]) => {
      if (user != null) {...
      } else if (req.body.username == '' || req.body.phone == ''...
      } else {...
      }
    })
    .catch(err => { console.log(err) })
}
```

Route 是 api 的路由

app.js 配置 express 的路径，组件等

三、项目展示

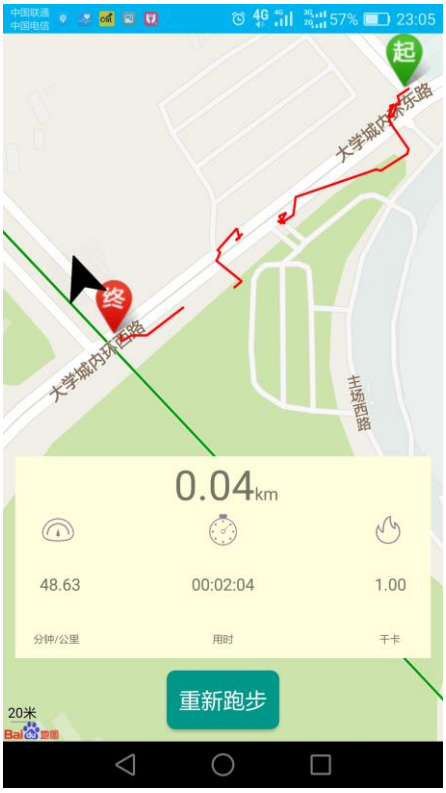
1.开场动画



2.主界面

3.点击开始跑步之后记录轨迹以及各项数据

4. 点击暂停会停止记录轨迹以及步数还有时间等



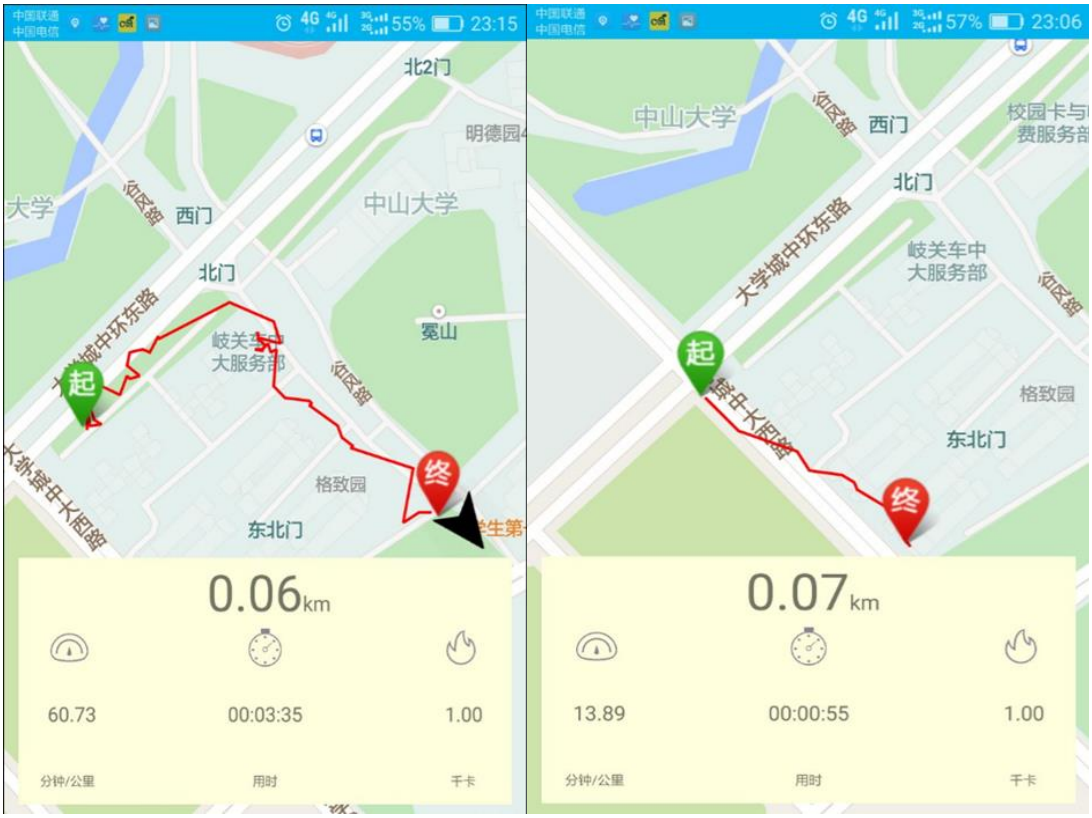
5.点击停止会跳转到一个统计界面：

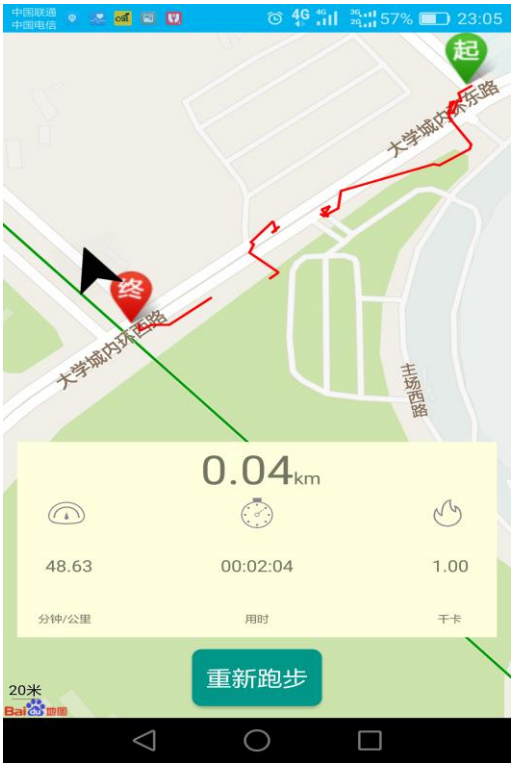
显示速度时长卡路里轨迹数据。

6.数据库界面显示：


猫扑走路
总步数：608
总次数：5
总时长: 00:09:36
跑步时间
跑步时长
跑步步数
2016-12-20 10:59:45
00:01:33
132
2016-12-20 11:02:44
00:02:04
85
2016-12-20 11:11:35
00:03:35
118
2016-12-20 11:16:27
00:01:29
141

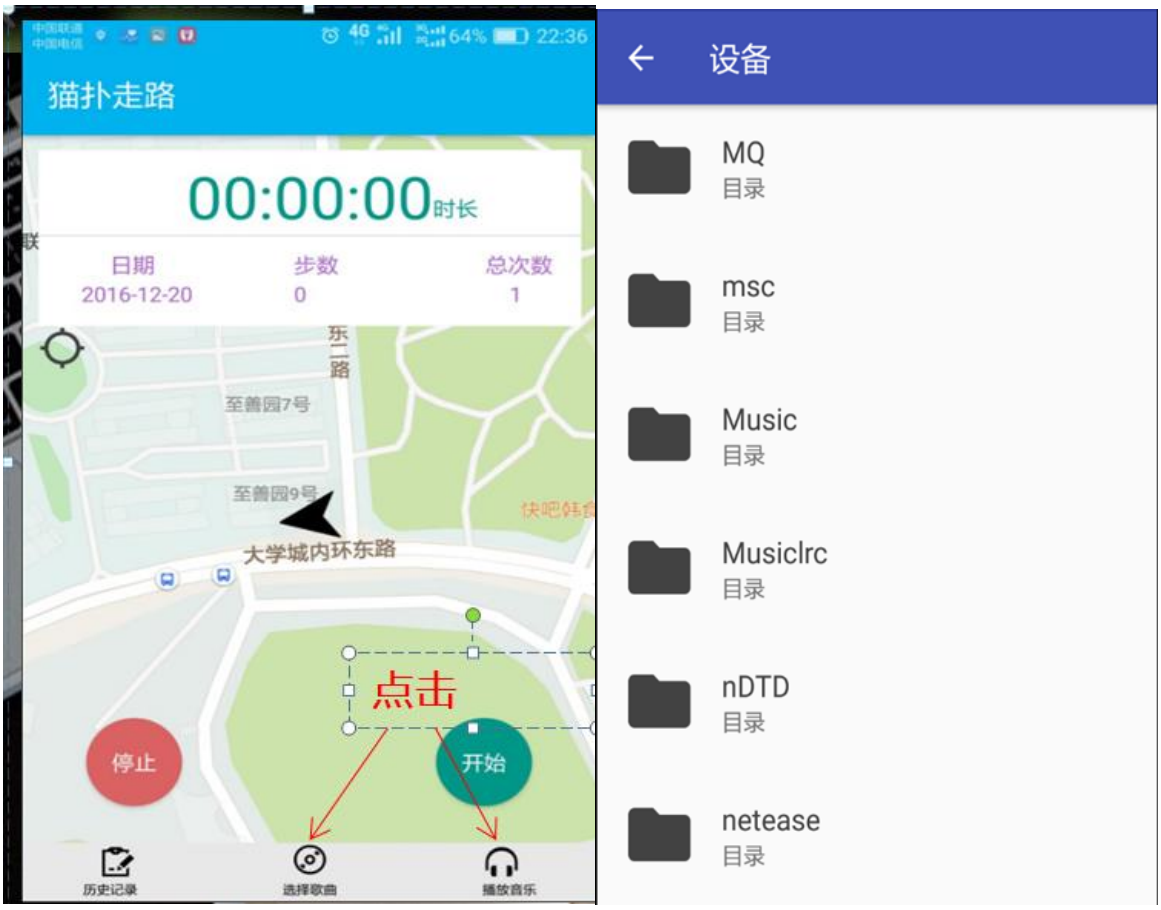
7.历史跑步轨迹展示：


0.06km
60.73
00:03:35
1.00
分钟/公里
用时
千卡
0.07km
13.89
00:00:55
1.00
分钟/公里
用时
千卡


起
大学城内环东路
终
大学城内环西路
主场西路
0.04km
48.63
00:02:04
1.00
分钟/公里
用时
千卡
重新跑步
20米


格致园
东北门
学生第一食堂
起
中山大学
慎思园5号
慎思园6号
大学城中大北路
0.07km
23.48
00:01:33
1.00
分钟/公里
用时
千卡

8. 后台音乐播放

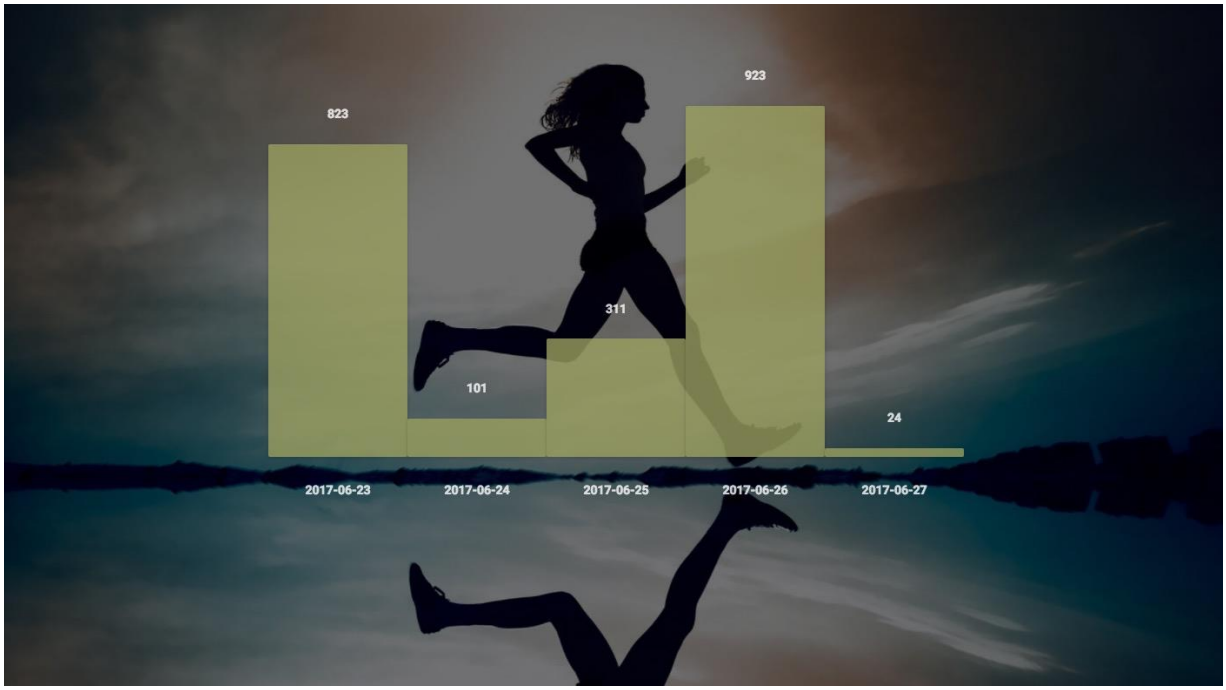

猫扑走路
00:00:00时长
日期
2016-12-20
步数
0
总次数
1
东二路
至善园7号
至善园9号
大学城内环东路
点击
停止
开始
历史记录
选择歌曲
播放音乐
设备
MQ
目录
msc
目录
Music
目录
Musiclrc
目录
nDTD
目录
netease
目录

9.Wedgit 及广播

10.网页界面


登录
注册
昵称
电话
邮箱
密码
注 册

近几天步数


823
101
311
923
24
2017-06-23
2017-06-24
2017-06-25
2017-06-26
2017-06-27

四、项目难点及解决方案

1．开场动画设计的时候想使用 recycleView，但不能实现自动翻转效果，通过搜索资料发现 github 上有广告栏循环播放的开源库可以直接进行调用，通过自己设计符合的图片加载的 holder 可以进行实现动画的播放，并且添加了不同的播放形式。

2．filePicker 部分如果是自己实现代码刚开始发现工程量挺大的，而且安卓 sdk 中没有相关的 api 接口。Github 上相关控件的 api 开源库真的很强大；

3．计时器的难点主要是多线程的消息传递机制。因为 Android 的 UI 线程和计时器的线程必须分开，我使用的是 Handler 和 Runnable 的消息传递机制实现计时线程和 UI 线程的同步。最麻烦的问题是 MainActivity 的 main 线程、UI 线程、计时线程的同步，如果消息传递不同步的话，当计时器停止之后，UI 没有来得及更新；更麻烦的是，点击停止图标后，计时线程没有停止，但由于计时状态已经进入停止，计时线程还会跳动 20ms，因此主页面里的 UI 时间不会变成 00:00:00。所以必须通过合理的消息传递和对时间的一些处理实现时间更新。

4．百度的 API 变动比较快，而且百度地图上的 Overlay 不能直接删除，必须保存到一个 Overlay 变量里，然后调用这个 Overlay 的 remove 方法，否则不能在图中删除。同理，对于轨迹来说，我们必须对点的集合里的每一个点做移除，否则不能在跑完步后实现轨迹重置（把原来的轨迹删除）。

5．Widget 实现过程基本没遇到很大的问题。

6．百度地图定位时的坐标抖动问题：这个我至今没找到很好的办法，因为更新坐标频率过慢的话，跑步轨迹画出来效果比较差；如果更新过快的话，由于定位的不精准性，坐标在信号差的地方会飞来飞去，按照画轨迹的逻辑，会产生一些“乱点”，其他跑步应用（如 Keep、虎扑跑步）也会有相同的问题。定位问题和手机传感器 Sensor 本身感觉没有关系。我想通过降低 Location 更新频率，但使用 Sensor 更新手机指向方向来优化用户体验，但效果还是很差，定位一样会抖动。

7．数据库此次实现存储轨迹可谓是一个难点，主要是存储轨迹的类型是 List<LatIng>，首先如果用课堂上学习的 SQLite 语句来创建的话，肯定不能创建一个 List<LatIng>类型的列名，经过查阅资料发现 SQL 要存储 List 类型需要重载一个类，新建一个类来储存 LatIng，显然这对我们的项目来说无异于太麻烦也没有任何意义，仔细想来之后，决定把每个轨迹的坐标点的经纬度存进去数据库，因为是 float 类型所以会方便很多。除此之外，因为每次跑步记录需要记录很多坐标点，因此数据库还需要一个 order 类型来存储这些轨迹点是哪一次跑步的坐标。

8．数据库存储的时候有一个列名写成了 order，导致一直报语义的异常，因为类似 order by 这些语句是 SQL 语句，不要用作 SQL 的列名为好。

9．使用百度地图的鹰眼 API 的轨迹绘制功能绘制的轨迹与 APP 中的图标出现分离，即绘制的轨迹为鹰眼 API 自行优化过的轨迹并不是 APP 中图标走过的轨迹。解决方法：放弃了使用鹰眼 API，用实时记录 APP 中图标的经纬度绘制轨迹并使用数据库保存至本地。

10．刚开始只有后端，测试后端 api 因为 POST 方法不能用浏览器的方式，使用 postman 工具进行测试。

11．后端 session 储存不能作一般的储存，多并发可能导致丢失，需储存到缓存池，可用 redis，此项目用的是 mysql。

12．做 web 前后端分离时无从入手，因为要使用 vue 框架，前端用 webpack 热更新，并配置 es6 的转译，不知前从哪个入手，因为要测试是否成功，都跟另外两方面有关系。会出现 vue 找不到模板，或者不支持 es6 语法等等的情况。于是先自己写例子测试 webpack 热更新功能及其使用，再写出简单的 vue 模板 ,逐点排插错误 ,发现为了让浏览器刷新停留在本页面的历史模式配置存在问题，所以去掉了历史模式，刷新页面会跳回登陆，这点有待改善。

13．前端的自适应问题比较难调节，会因窗口大小改变而出现了样式问题，所以进行了长时间的细节调整。

五、项目总结

1. 这次实验实现了一个跑步健身类的软件，感觉还是十分的实用的，总体上，基本的跑步软件的功能也可以满足。在这次实验中我主要负责软件的开场动画的设计以及实现查询本级音乐和音乐循环播放部分，如果要具体实现这些功能，实现的代码基本上是很大的，在这部分的学习中，

我学会了对 github 上的安卓控件开源库的引用，通过其提供的相关 api 接口的调用，可以实现很强大的功能，极大地锻炼了阅读别人代码的能力以及与其代码 api 进行对接。并且，在这次实验中也更好地培养了团队协作的能力，一个项目的实现有时候并不能依靠一个人，很多时候要学会团队沟通以及对代码功能部分进行分工以及后期代码整合。

2. 此外有个难点在于之前并没有了解过地图轨迹绘制相关知识。在尝试使用百度地图鹰眼 API 时，在接口信息请求和获取解析方面花费了很长时间。实现之后发现鹰眼 API 通过去除定位不准确的坐标来实现的优化轨迹很难与 APP 中黑色图标重合，即轨迹与目标经常处于分离状态，所以放弃了使用鹰眼 API，转而使用本地记录经纬度绘制轨迹。而本地记录的轨迹则需要解决数据储存的问题，幸好查阅资料后使用数据库成功保存了经纬度数据，实现了轨迹数据的储存。

3.在 UI 方面，因为之前的一个设想是将数据统计界面做成一个悬浮界面放在地图之上，UI 也参考了虎扑跑步。最后经过讨论决定采用帧布局的方式来设置主界面达到一种嵌套效果。数据库方面，因为要精益求精，最后在轨迹终于可以实时绘制的情况下，组员决定将轨迹也加到数据库中，这是一个新的挑战和难点，但是更加完善和丰富了我们的 APP。最终采用的是存储坐标点的经纬度到数据库顺利解决完成了轨迹的存储。最后一晚，为了展示 PPT 的实验截图，组员特地装好 APP 去内环疯狂跑步，截取各种各样的轨迹。当时就是感觉组内氛围非常好，感觉看到轨迹在 APP 上顺利地跑出来没有出问题，之前所有的努力都很值得，有很棒的队友真的是一件很幸运的事情。

4.web 前端及服务器方面，了解了整个后端架构的运作，及前端对其 api 的请求及处理等。更加深入地了解了“服务器”的概念，以前认为的服务器是远程主机之类，现在了解到，可以启动服务、提供服务的程序也是一种服务器，这类服务器间的交流问题（比如跨域请求）需要通过配置方式来解决，而不是在同一主机上程序开启的服务就是共享所有资源的。通过撰写 api 了解到

http 是无状态的，需要通过 cookie 及后端的 session 来“验证身份”。通过使用 vue 框架了解到 DOM 渲染、数据绑定的优化等。通过前后端分离了解到跨域攻击和跨域请求的原理。