```
package com.rtnimageresizer;
import android.annotation.SuppressLint;
import android.graphics.Bitmap;
import android.net.Uri;
import android.os.AsyncTask;
import android.util.Log;
import androidx.annotation.Nullable;
import androidx.annotation.NonNull;
import com.facebook.react.bridge.Arguments;
import com.facebook.react.bridge.GuardedAsyncTask;
import com.facebook.react.bridge.Promise;
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.bridge.ReadableMap;
import com.facebook.react.bridge.WritableMap;
import com.facebook.react.bridge.ReactMethod;
import com.rtnimageresizer.NativeImageResizerSpec;
import java.io.File;
import java.io.IOException;
import java.util.UUID;

public class ImageResizerModule extends NativeImageResizerSpec {
    public static final String NAME = "RTNImageResizer";

    ImageResizerModule(ReactApplicationContext reactContext) {
        super(reactContext);
    }
```

```
@Override
@NonNull
public String getName() {
    return NAME;
}

@ReactMethod
public void createResizedImage(String uri, double width, double height, String format, double quality, String mode, boolean onlyScaleDown, Double rotation, @Nullable String outputPath, Boolean keepMeta, Promise promise) {
    WritableMap options = Arguments.createMap();
    options.putString("mode", mode);
    options.putBoolean("onlyScaleDown", onlyScaleDown);

    new GuardedAsyncTask<Void, Void>(this.getReactApplicationContext()) {
    @Override
    protected void doInBackgroundGuarded(Void... params) {
    try {
        Object response = createResizedImageWithExceptions(uri, (int) width, (int) height, format, (int) quality, rotation.intValue(), outputPath, keepMeta, options);
        promise.resolve(response);
    }
    catch (IOException e) {
        promise.reject(e);
    }
    }
```

```
        }.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);

        }


        @SuppressLint("LongLogTag")

        private Object createResizedImageWithExceptions(String
imagePath, int newWidth, int newHeight,

        String compressFormatString, int quality, int rotation, String
outputPath,

        final boolean keepMeta,

        final ReadableMap options) throws IOException {


        Bitmap.CompressFormat compressFormat =
Bitmap.CompressFormat.valueOf(compressFormatString);

        Uri imageUri = Uri.parse(imagePath);


        Bitmap scaledImage =
ImageResizer.createResizedImage(this.getReactApplicationContext(),
imageUri, newWidth, newHeight, quality, rotation,

        options.getString("mode"),
options.getBoolean("onlyScaleDown"));


        if (scaledImage == null) {

                throw new IOException("The image failed to be resized;
invalid Bitmap result.");

        }


        File path = this.getReactApplicationContext().getCacheDir();

                if (outputPath != null) {

                        path = new File(outputPath);
```

```
            }


        File resizedImage = ImageResizer.saveImage(scaledImage, path,
UUID.randomUUID().toString(), compressFormat, quality);

        WritableMap response = Arguments.createMap();


        if (resizedImage.isFile()) {

                response.putString("path",
resizedImage.getAbsolutePath());

                response.putString("uri",
Uri.fromFile(resizedImage).toString());

                response.putString("name", resizedImage.getName());

                response.putDouble("size", resizedImage.length());

                response.putDouble("width", scaledImage.getWidth());

                response.putDouble("height", scaledImage.getHeight());


        if(keepMeta){

                try{


        ImageResizer.copyExif(this.getReactApplicationContext(),
imageUri, resizedImage.getAbsolutePath());

        }

        catch(Exception ignored){

                Log.e("ImageResizer::createResizedImageWithExceptions",
"EXIF copy failed", ignored);

        }

        }

        } else {

                throw new IOException("Error getting resized image path");
```

```
        }
 scaledImage.recycle();
            return response;
    }
}
```