Video Transcription Application - Complete Project Code

Backend Implementation (Go)

```
package main

import (
    "context"
    "fmt"
    "io"
    "io/ioutil"
    "log"
    "net/http"
    "os"

    "cloud.google.com/go/videointelligence/apiv1"
    videopb "google.golang.org/genproto/googleapis/cloud/videointelligence/v1"
)

func uploadHandler(w http.ResponseWriter, r *http.Request) {
    file, header, err := r.FormFile("video")
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }
    defer file.Close()

    out, err := os.Create("./uploads/" + header.Filename)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }
    defer out.Close()

    _, err = io.Copy(out, file)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }
    fmt.Fprintln(w, "File uploaded successfully: ", header.Filename)
}

func transcribeHandler(w http.ResponseWriter, r *http.Request) {
    filePath := r.URL.Query().Get("file")
    transcription, err := transcribeVideo(filePath)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }
    fmt.Fprintln(w, transcription)
}
func transcribeVideo(filePath string) (string, error) {
```

```
    ctx := context.Background()
    client, err := videointelligence.NewClient(ctx)
    if err != nil {
        return "", err
    }
    defer client.Close()

    fileBytes, err := ioutil.ReadFile(filePath)
    if err != nil {
        return "", err
    }

    operation, err := client.AnnotateVideo(ctx, &videopb.AnnotateVideoRequest{
        Features: []videopb.Feature{videopb.Feature_SPEECH_TRANSCRIPTION},
        InputContent: fileBytes,
    })
    if err != nil {
        return "", err
    }

    response, err := operation.Wait(ctx)
    if err != nil {
        return "", err
    }

    var transcription string
    for _, result := range response.AnnotationResults {
        for _, speechTranscription := range result.SpeechTranscriptions {
            for _, alternative := range speechTranscription.Alternatives {
                transcription += alternative.Transcript + "\n"
            }
        }
    }
    return transcription, nil
}

func main() {
    http.HandleFunc("/upload", uploadHandler)
    http.HandleFunc("/transcribe", transcribeHandler)
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

Frontend Implementation (Kotlin)

```
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import okhttp3.*
import java.io.File
import java.io.IOException
class MainActivity : AppCompatActivity() {
```

```
private val PICK_VIDEO_REQUEST = 1
private lateinit var uploadButton: Button
private lateinit var transcribeButton: Button
private lateinit var transcriptionText: TextView
private var selectedVideoPath: String? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    uploadButton = findViewById(R.id.uploadButton)
    transcribeButton = findViewById(R.id.transcribeButton)
    transcriptionText = findViewById(R.id.transcriptionText)

    uploadButton.setOnClickListener {
        val intent = Intent(Intent.ACTION_PICK)
        intent.type = "video/*"
        startActivityForResult(intent, PICK_VIDEO_REQUEST)
    }

    transcribeButton.setOnClickListener {
        selectedVideoPath?.let { path ->
            uploadVideo(File(path))
        }
    }
}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == PICK_VIDEO_REQUEST && resultCode == RESULT_OK && data != null) {
        val uri: Uri = data.data!!
        selectedVideoPath = uri.path
    }
}

private fun uploadVideo(file: File) {
    val client = OkHttpClient()
    val requestBody = RequestBody.create(MediaType.parse("video/*"), file)
    val request = Request.Builder()
        .url("http://localhost:8080/upload")
        .post(requestBody)
        .build()

    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            e.printStackTrace()
        }

        override fun onResponse(call: Call, response: Response) {
            if (response.isSuccessful) {
                requestTranscription(file.absolutePath)
            }
        }
    })
}

private fun requestTranscription(filePath: String) {
```

```
        val client = OkHttpClient()
        val request = Request.Builder()
            .url("http://localhost:8080/transcribe?file=$filePath")
            .build()

        client.newCall(request).enqueue(object : Callback {
            override fun onFailure(call: Call, e: IOException) {
                e.printStackTrace()
            }

            override fun onResponse(call: Call, response: Response) {
                runOnUiThread {
                    if (response.isSuccessful) {
                        transcriptionText.text = response.body?.string()
                    }
                }
            }
        })
    }
}
```

Generated for GitHub repository inclusion. Includes complete backend (Go) and frontend (Kotlin) source code for the Video Transcription Application project.