**DECEPTIFORM AND SPEECHER**

**J COMPONENT PROJECT REPORT**

**Fall Semester 2020-21**

Submitted by

**KSHITIJ DHYANI (18BIT0131)**

**ISHAAN JAIN (18BIT0300)**

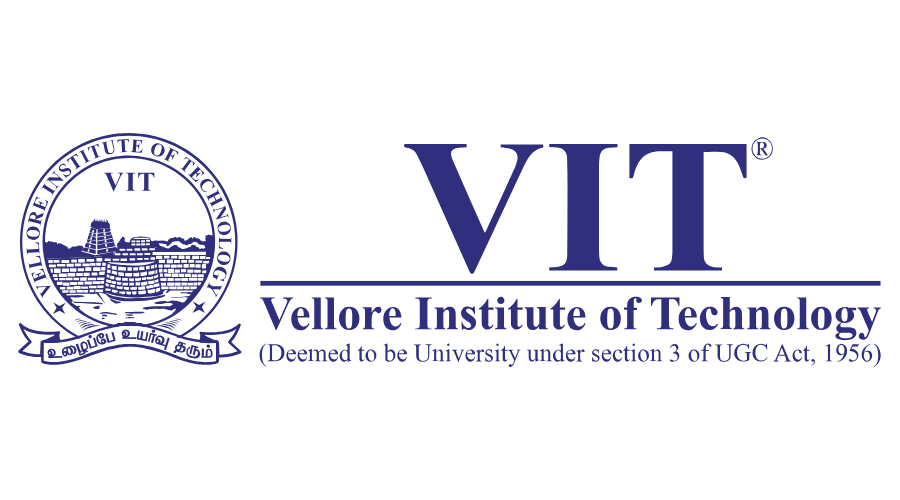
**PRANAV GARG (18BIT0320)**

*in partial fulfillment for the award* of the degree of

**B.Tech**

in

**Information Technology**



Vellore-632014, Tamil Nadu, India

*Under* ***Prof. Sharmila Banu K.***

***Natural Language Processing (CSE4022)***

**School of Information Technology**

November, 2020

Important links

PROJECT LINKS -

1. DECEPTIFORM WEBSITE: <https://deceptiform.herokuapp.com/>
2. GIT HUB REPOSITORY: <https://github.com/wimpywarlord/women_techies>
3. SPEECHR: <https://github.com/ishaan14112000/NLP-PROJECT-SPEECHR>

HANDS ON TASKS -

1) GITHUB LINK TO MY HANDS ON : <https://github.com/wimpywarlord/nlp_hands_on>

TEAMS TASKS -

1. <https://github.com/ishaan14112000/NLP-NATGEO-GROUP-TASK>

**ABSTRACT**

The name of our product is deceptiform, let us break it down. The first half of the name of the product comes from the movie transformers and the main villains in the same movie are called Decepticons which are humongous robots with autonomous interstellar technology.

The second half is a simple word “Form” which in its entirety means a method to ease the process of exchange of information via a set of input fields.

Lets us Fuse these two words together in an attempt to explain a product. Deceptiform, in other words, means the automation of filling of cumbersome forms which we often find ourselves stuck with, leading to either just staring at the complexly constructed pieces of papers or paying huge loads of money to some professional to decode the form and help us fill it.

Programmed speech recognition is an innovation which has real time genuine applications. From programmed replying mail administrations, to registry enquiries and computerized conveyance administrations, anyplace where people interface with an independent framework, speech recognition gives the connection among man and machine. This venture applies speech recognition to the field of meeting perusing, mechanizing a significant number of the assignments attempted constantly taker. Utilizing an open source speech recognizer framework, that is JAVA, singular speaker accounts are prepared and timestamped. The transcriptions are then processed and integrated into a meeting browser system, in order to facilitate the quick and easy access of meeting information.

**MOTIVATION FOR PROJECT**

Lets us take an example to better understand the usability of the product. Let us say that you want to buy a piece of land and in order to do that you cant just simply smash a pile of cash on the desk and go on with your life celebrating the new purchase.

On a more simpler alternative reality, you would first have to hire a lawyer to bring about all the paperwork and then you have to sit down multiple times throughout the week, paying for every sitting to that notorious lawyer, trying to decode that encrypted piece of paper that lawyers have carefully curated to make it unreadable to a normal person in order to promote their business model. Enjoy hassle-free documentation with Deceptiform.

Secondly, if you were in a documentation sector you would easily realise that an agreement or a formal doc could go up to 1000-2000 pages in length.

In the face of such a situation, a legal professional or a documentation personal just simply go skimming through the document since they won't be sitting down reading a whole of 2000 pages for every single customer, this assures a guarantee of atleast missing out on atleast a few details, but not with deceptiform. Enjoy a 100% error-free documentation and form filling in lightening fast terms.

Additionally, we are all familiar with the scams that go around these paper works. There could be an insurance paper amongst the other pile of papers, and the lawyer could make you sign with a blind eye everywhere. The only thing you need to keep a blind eye on is the quality and safety of Deceptiform.

Who is to save us amongst such dismay? Now, what if I told you that there was free service that would automatically fill the whole legal document just based on a few questions.

All you need to do is enter the form template which is quite freely available on the internet and sit back while our service using NLP and certain algorithms, constructs a small questionnaire about the details which it will need from you to fill the form.

Just answer a few super simple straightforward question and witness our product smarty fill up the whole document. BOOM, enjoy your document hassle-free.

Let's not limit this application to legal documents, we come across such documents on very frequent occasions, for instance when you wanted to Join Vellore Institute of Technology – Vellore, you were handed hundreds of papers to fill and a thousand places to sign, or whenever you are looking to join a new organisation as an employee or whenever you end up going to the hospital or the notary.

Be it a birthday invite or a simple non-disclosure agreement, simply enter the form template and enjoy the end product while you sip on your morning tea.

Natural Language Processing NLP refers to the evolving set of computer and AI-based technologies that allow computers to learn, understand, and produce content in human languages. The technology works closely with speech/voice recognition and text recognition engines. While text/character recognition and speech/voice recognition allows computers to input the information, NLP allows making sense of this information. So, in this report we are making our effort to show the use of NLP programs. Java speech recognition is one of the project as we are using Sphinx 4 library.

**METHODS AND TECHNOLOGY USED**

DECEPTIFORM – Deceptiform is a very simple yet novel implementation of google’s text to speech api and microsoft’s text summarization api. The interface is a web based application, built upon HTML CSS and Vanilla JS. The architecture can be divided into client side and server side. The client side utilizes technologies such as bootstrap, SCSS, where the server side is built upon technologies such as NODE.js, express.js and mongodb. The api used are the following and can be viewed from the respective links:

1. Text to speech api : <https://cloud.google.com/text-to-speech>
2. Text summarization api : <https://rapidapi.com/textanalysis/api/text-summarization>

SPEECHER - The programming language used in the project was Java. The IBM EclipseIDE was used to build and manage the speech recognition package, the CMUSphinx 4 Recognizer. One of the requirements of the program was the installation of ApacheAnt to compile the Sphinx package.

Apache Ant is a program for creatingXML build files for Java, and also integrates fully into the Eclipse IDE.With Sphinx being a complicated package containing many separate files and folders. Components of the platform use the Java Speech API, and this was given the source code. The JSAPI is a Java package for use with speech recognition frameworks and speech blend frameworks, and isn't given the Standard Edition of the Java platform.

Using Eclipse gave a steady platform to fabricate and run the Sphinx project. It was conceivable to alter all info boundaries and a debugger was accessible when things turned out badly. The Sphinx site gave directions on the best way to get everything going utilizing Eclipse.The Sphinx framework was picked due to the way that it was executed in Java, and furthermore on the grounds that it was an open source framework and thus liberated from permitting issues. The adaptable idea of the framework, and the way that it was configurable to suit a scope of various errands likewise demonstrated helpful to the project.

**CODE AND OUTPUT**

**1. DECEPTIFORM**

var express = require("express");

var bodyParser = require("body-parser");

var fs=require('fs');

var deepai=require('deepai');

deepai.setApiKey('quickstart-QUdJIGlzIGNvbWluZy4uLi4K');

var resp;

async function asyncall() {

    resp = await deepai.callStandardApi("summarization", {

            text: fs.createReadStream("proxies.txt"),

    });

    console.log(resp);

}

asyncall();

var global\_variable\_for\_the\_form\_text;

var app = express();

app.use(bodyParser.urlencoded({extended : true}));

app.use(express.static(\_\_dirname + '/assests'));

app.use(express.static(\_\_dirname + '/public'));

////

var express = require("express");

var bodyParser = require("body-parser");

var global\_variable\_for\_the\_form\_text;

var app = express();

app.use(bodyParser.urlencoded({extended : true}));

app.use(express.static(\_\_dirname + '/assests'));

app.use(express.static(\_\_dirname + '/public'));

var unirest = require("unirest");

var req = unirest("POST", "https://textanalysis-keyword-extraction-v1.p.rapidapi.com/keyword-extractor-text");

req.headers({

    "x-rapidapi-host": "textanalysis-keyword-extraction-v1.p.rapidapi.com",

    "x-rapidapi-key": "7703ef81e0mshd5555ce8567e172p148f04jsn2f562e2f9571",

    "content-type": "application/x-www-form-urlencoded"

});

req.form({

    "text": "\_\_\_\_ (NAME)"

    +"\_\_\_\_"

    +"\_\_\_\_ (DATE)"

    +"\_\_\_\_ (EMAIL)"

    +"To \_\_\_\_\_(COMPANY),"

    +"I am formally requesting you to give me \_\_\_\_\_(NO OF DAYS) days leave of absence."

    +"I have finished almost all the projects assigned to me with only one remaining which I have given to my assistant to take care of it since he is as experienced just as I am in handling that remaining project. "

    +"Thank you for your consideration."

    +"Yours Sincerely,"

    +"\_\_\_\_\_\_(NAME)",

    "wordnum": "5"

});

var a,b;

req.end(function (res) {

    if (res.error) throw new Error(res.error);

    var keywords\_arr = ["name","date of leave","company name","email"];

    res.body.keywords =  (res.body.keywords).concat(keywords\_arr);

    console.log(res.body);

    var questionable = require('questionable');

questionable("employee name", function (err, titles) {

  if (err) throw err;

  //console.log(titles);

  console.log(titles)

    });

questionable("company name", function (err, titles) {

  if (err) throw err;

  //console.log(titles);

  console.log(titles);

    });

    questionable("date of leave", function (err, titles) {

        if (err) throw err;

        //console.log(titles);

        console.log(titles);

          });

          questionable("email", function (err, titles) {

            if (err) throw err;

            //console.log(titles);

            console.log(titles);

              });

});

app.get("/",function(req,res){

    res.render("home.ejs");

})

var questions={

    "q" : [

    "what is the name of your name?",

    "what is the name of your company?",

    "what is your name?",

    "what is the date of application of leave?"

]

}

app.get("/intake",function(req,res){

    res.render("intake.ejs");

})

app.get("/info",function(req,res){

    res.render("form\_des.ejs");

})

app.get("/offer\_letter",function(req,res){

    res.render("offer\_letter.ejs",{summary:resp.output});

})

app.get("/autoform",function(req,res){

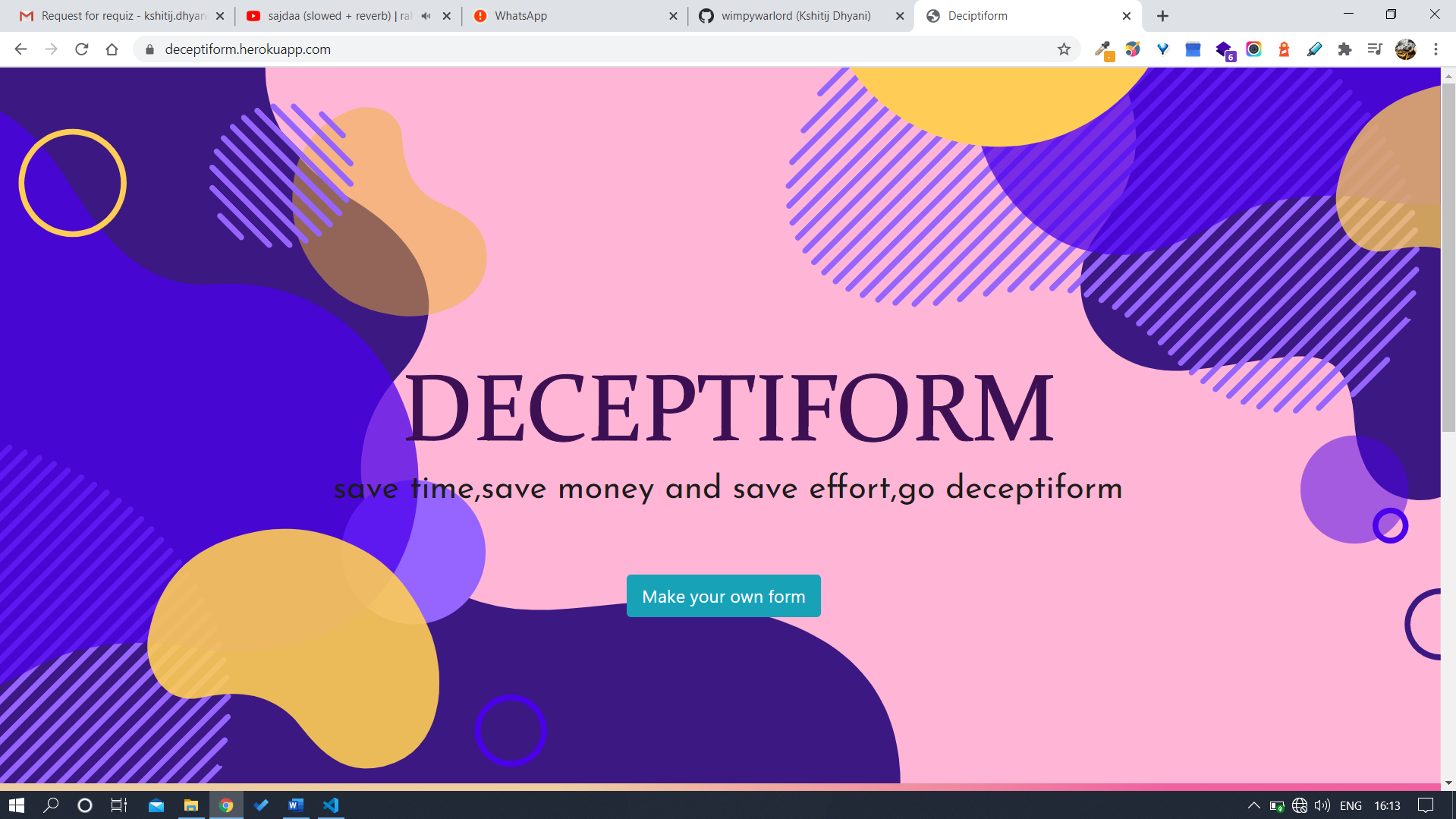
    res.render("autoform.ejs",{questions:questions});

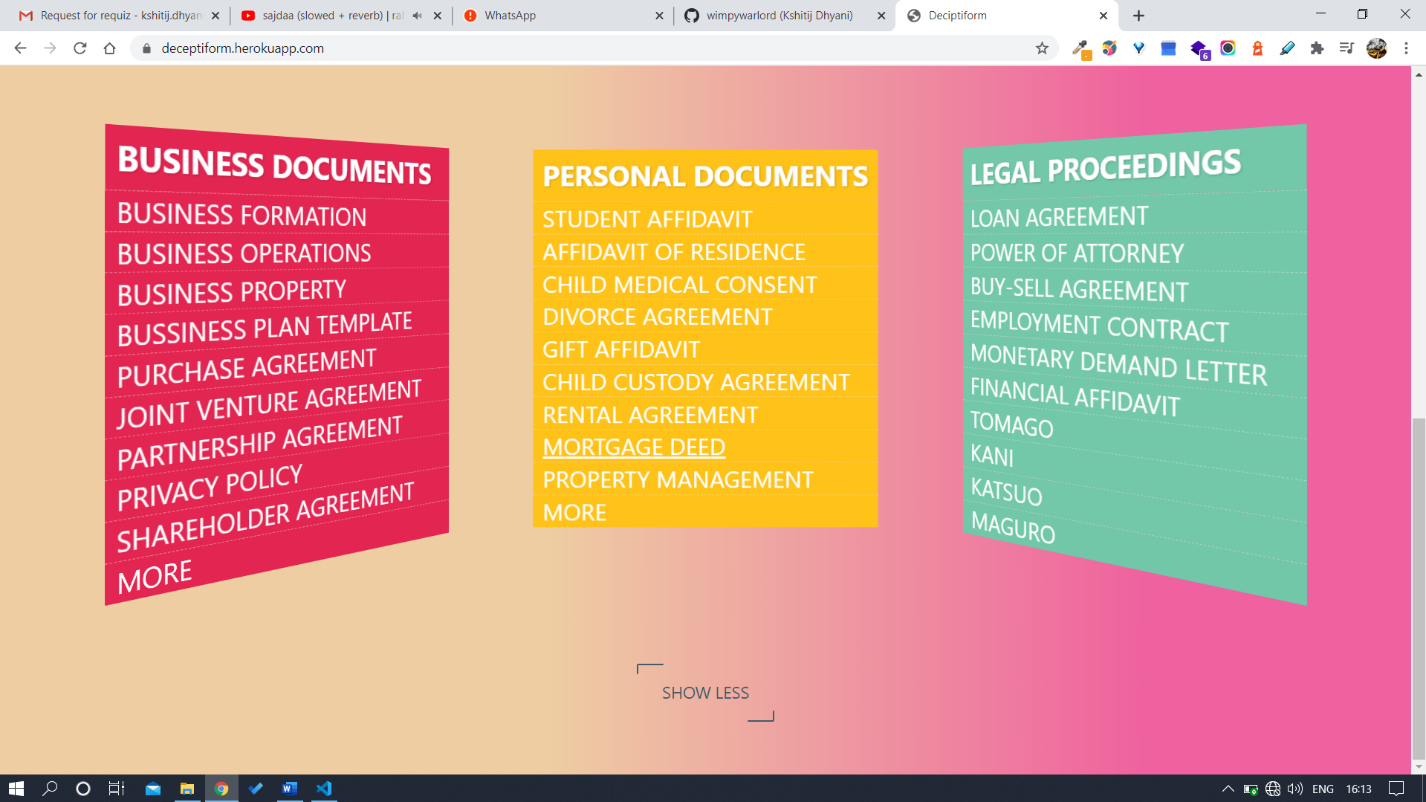
})

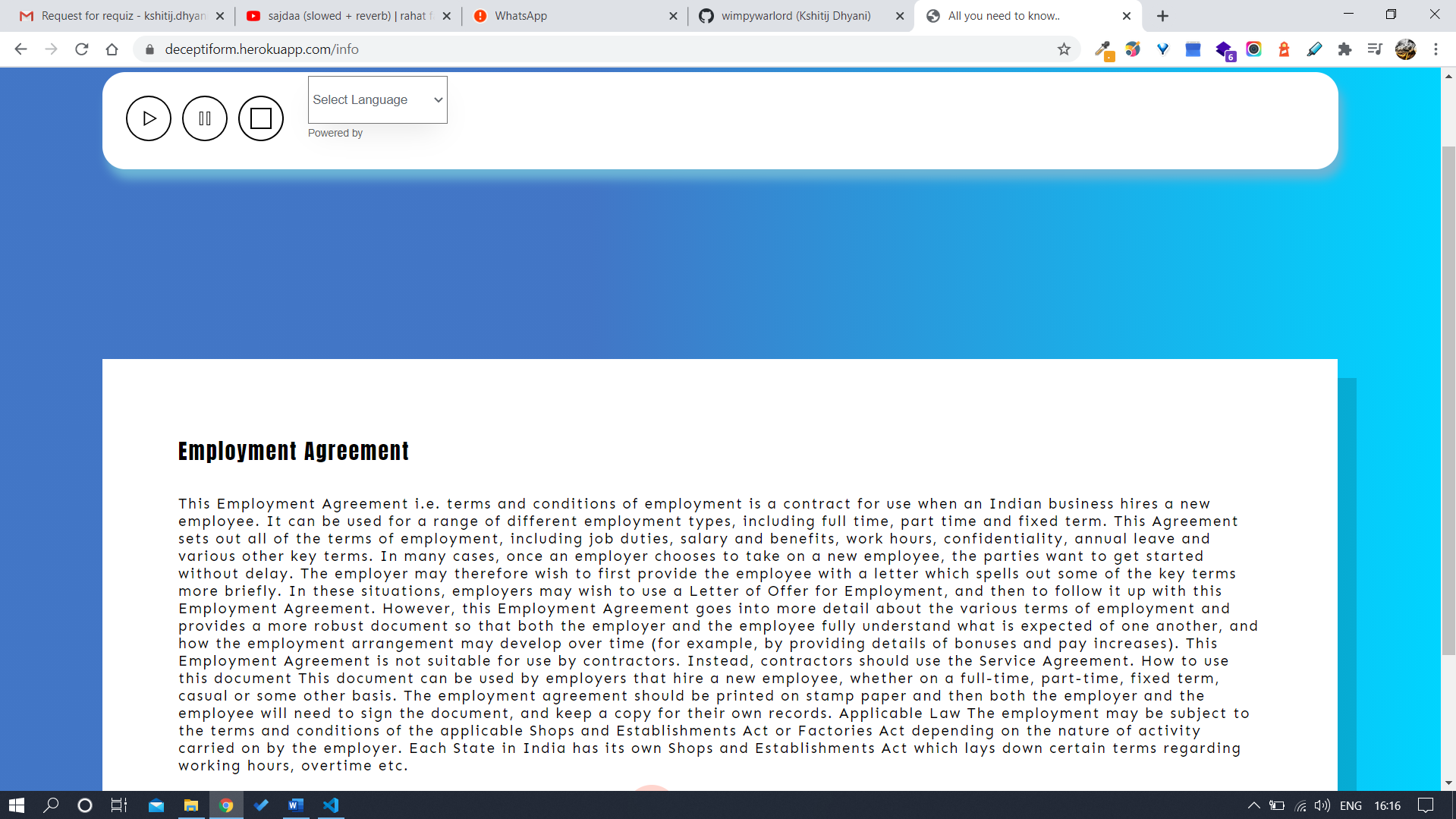
app.listen( process.env.PORT || 8000 , function(){

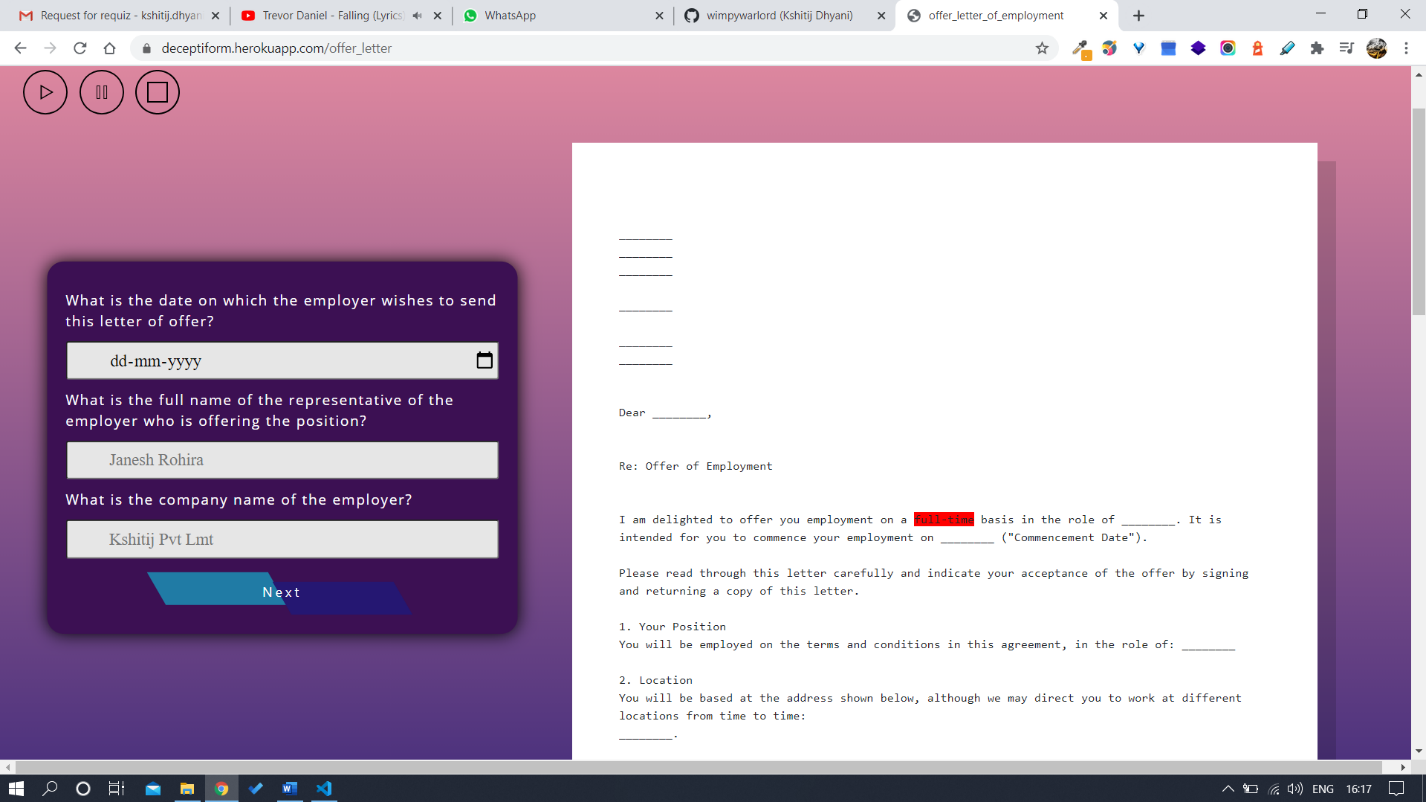
    console.log("SERVER 8000 HAS STARTED");

});

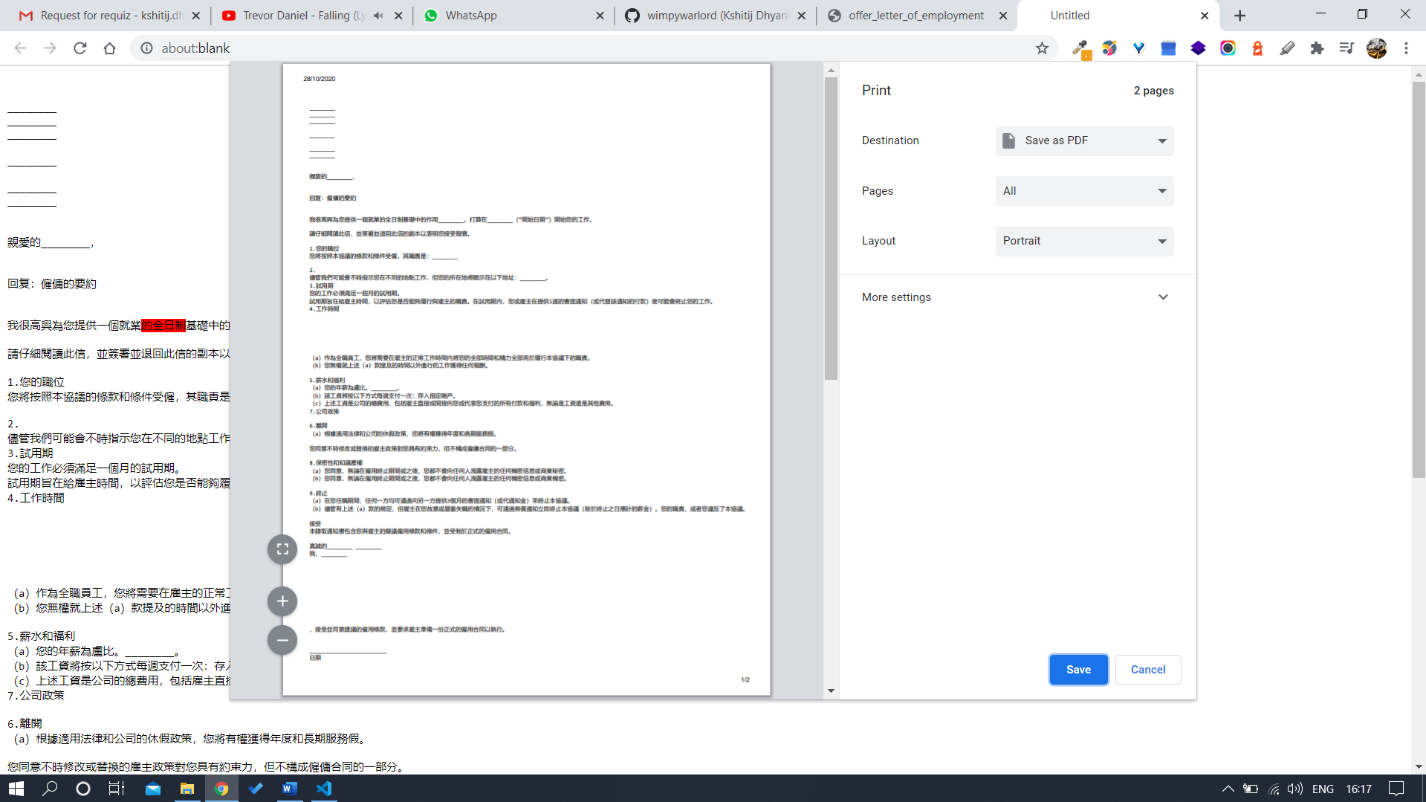












**2. SPEECHR**

package test;

import java.io.IOException;

import java.awt.Desktop;

import java.net.URI;

import edu.cmu.sphinx.api.Configuration;

import edu.cmu.sphinx.api.LiveSpeechRecognizer;

import edu.cmu.sphinx.api.SpeechResult;

public class voicerecognizer {

public static void main(String[] args) throws IOException {

Configuration configuration= new Configuration();

configuration.setAcousticModelPath("resource:/edu/cmu/sphinx/models/en-us/en-us");

configuration.setDictionaryPath("/Users/ishaa/eclipse-workspace/voice recognition/src/resources/3345.dic");

configuration.setLanguageModelPath("/Users/ishaa/eclipse-workspace/voice recognition/src/resources/3345.lm");

LiveSpeechRecognizer recognize=new LiveSpeechRecognizer(configuration);

recognize.startRecognition(true);

SpeechResult speechResult;

while((speechResult=recognize.getResult())!=null) {

String command=speechResult.getHypothesis();

System.out.println("INPUT GIVEN IS:"+command);

if(command.equalsIgnoreCase("internet")) {

try {

URI uri= new URI("http://www.google.com");

java.awt.Desktop.getDesktop().browse(uri);

} catch (Exception e) {

e.printStackTrace();

}

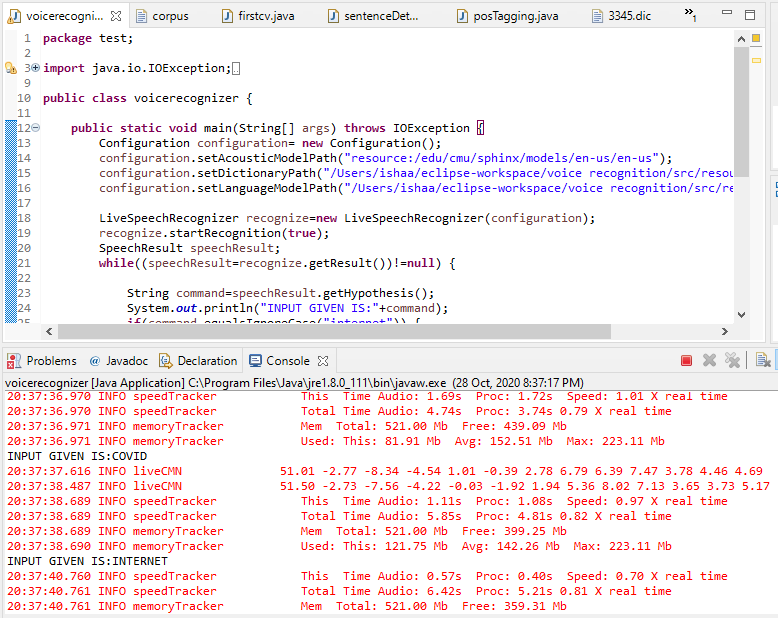
}

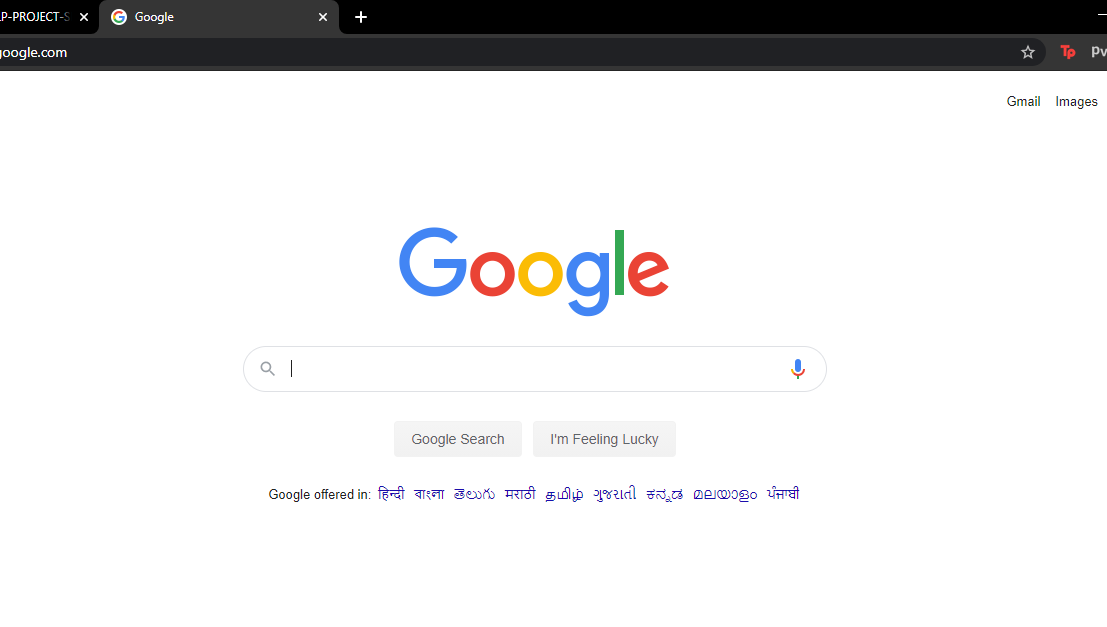
}

}

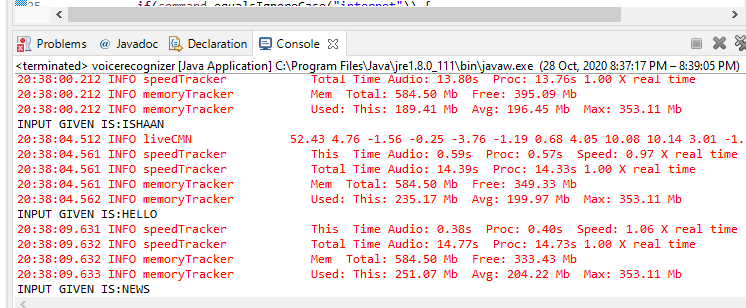
}

When input is given as internet, google chrome opens up with google as the web page being opened,

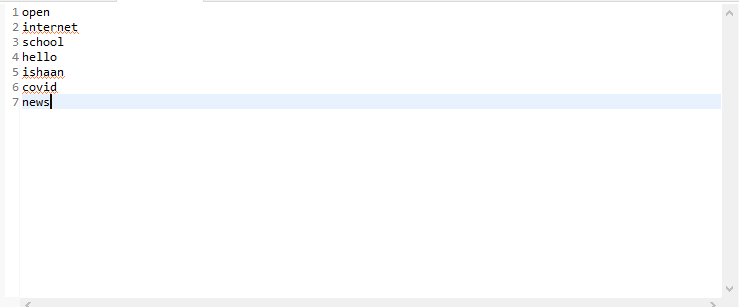




Speech being recognized,



Corpus,



**CONCLUSION**

This future of Deceptiform aims to make a central database of all the forms that are generally used in the country and make it a cakewalk to fill such forms. Giving a time benefit of a magnitude much greater than that of current standards. Save time, save money, and save the effort, go deceptiform.

**REFERENCES**

Deceptiform website: <https://deceptiform.herokuapp.com/>

Github repository: <https://github.com/wimpywarlord/women_techies>

Text to speech api : <https://cloud.google.com/text-to-speech>

Text summarization api : <https://rapidapi.com/textanalysis/api/text-summarization>