**Work done by Nripendra and Sumit**

**Django Framework (An Beginner’s approach to Website Development and Deployment)**

We decided to use the Django framework as our web development framework.

Choosing the correct framework:

Week 1:

We had the option of selecting between java based framework spring, flask,Django and flask. We moved ahead with Django and flask as our final two choices

as we wanted to use a web-based framework which is based in the language python as the majority of the data preferred to work on Python. The major challenge faced during this phase was the Team’s lack of knowledge of Django Web Framework. Initial one week was majorly spent on understanding its implementation and going through documentation and Tutorial Videos

Week 2,3:

Between Django and flask, we made a comparison we decided to use Django as our primary choice as Django is one of the most mature web frameworks for Python, it supports reducing web application development time, by taking advantage of python syntax rules to produce easily manageable code. The features provided by Django

allows us to build custom web applications rapidly according to varying project requirements. Since the project requires the application to be supported on various devices and platforms, Django enhances the accessibility of web applications by

supporting major operating systems like Windows, Linux and MacOS.

Due to the high amount of support and documentation, both of these tools were chosen for backend development. These two weeks we focused on deploying a dummy website on our local development environment such as Password Generator.

Week 4,5,6

Once the Dummy Applications were Setup and deployed on the development servers. We integrated the midway front end website to the Django framework. This came with its challenges of handling static files which had to be taken into account while writing the HTML Pages and Scripts. During these weeks, the Team was provided Knowledge Transfer to understand the changes that need to be made to the existing webpages to support them onto the Django Framework.

Week 7,8,9,10:

Once the Django Framework was in place and running at the development stage, it was to be pushed onto the AWS Architecture. During this stage, the frontend was experiencing significant changes, and our lack of knowledge made it difficult to implement the changes onto the Django Framework. This caused us many delays

These weeks also involved aws deployment of the Django Framework.

Further details of Deployment of the website on the cloud in specified in the section below.

**Cloud Deployment (Hosting our First Website on AWS)**

Cloud Deployment:

Week 7 :

We had to deploy the website on AWS so that the site can be accessible to users from anywhere. We looked up different techniques on how to

deploy to AWS.

Week 8 :

After, we were successful in having the integration set up between the website and lex set up. The site was attempted to be hosted on

AWS.

Week 9:

The website deployment was initiated by creating an EC2 instance on the server.

Following are commands that we used to deploy the website. The website required a lot of configuration set up it involved the installation of

python, Django, Nginx, supervisor,gunicorn,whitenoise and all the libraries which were used in our project.

Following is the flow of commands used after connecting the instance through putty.

sudo apt-get upgrade -y

sudo apt-get update -y

1. Deploy Python virtual env

sudo apt-get install python3-venv -y

python3 -m venv env

2. Activate env

source env/bin/activate

3. Install Django

pip3 install django

4. Clone the git project

git clone

5. Instal NGINX and GUNICORN

pip3 install gunicorn ## install without sudo..

sudo apt-get install nginx -y

pip install psycopg2-binary

6. Connect gunicorn

gunicorn --bind 0.0.0.0:8000 TestProject.wsgi:application ## similar to runserver

7. Install supervisor

sudo apt-get install -y supervisor ## This command holds the website after we logout

8. Config supervisor

cd /etc/supervisor/conf.d

sudo touch gunicorn.conf

##In the file file do following###

[program:gunicorn]

directory=/home/ubuntu/AwsDemo

command=/home/ubuntu/env/bin/gunicorn --workers 3 --bind  unix:/home/ubuntu/AwsDemo/app.sock TestProject.wsgi:application

autostart=true

autorestart=true

stderr\_logfile=/var/log/gunicorn/gunicorn.err.log

stdout\_logfile=/var/log/gunicorn/gunicorn.out.log

[group:guni]

Program:gunicorn

####endfile####

9. Connect file to supervisor

sudo supervisorctl reread

##if we receive the following error than we need to create the directory:

ERROR: CANT\_REREAD: The directory named as part of the path /var/log/gunicorn/gunicorn.out.log does not exist. in section 'program:gunicorn' (file: '/etc/supervisor/conf.d/gunicorn.conf')

sudo mkdir -p /var/log/gunicorn

sudo supervisorctl reread

sudo supervisorctl update

10. Check if gunicorn is running in background

sudo supervisorctl status

11. Create nginx configuration for django site

sudo vim /etc/nginx/sites-available/django.conf

###inhalt##

server {

    listen 80;

    server\_name 18.195.252.82;

    location / {

        include proxy\_params;

        proxy\_pass http://unix:/home/ubuntu/AwsDemo/app.sock;

    }

}

12. Test file configuration

sudo ln /etc/nginx/sites-available/django.conf /etc/nginx/sites-enabled/

sudo nginx -t

13. restart NGINX server

sudo service nginx restart

**Amazon Lex (Chatbot Development, Deployment and Integration)**

The chatbot creation started by making a decision, to begin with, our own chatbot implementation that involved the creation of Machine Learning and Textual Understanding

Week 1:

Involved making the above understanding and analysing the libraries that can be used to implement a chatbot manually from Scratch since we had experience in Text Processing. On reflecting upon Simon’s advice, we started digging more into market available API Based chatbots that would automate manual efforts of creating the entire Chatbot from Scratch.

Week 2:

This week we tried to implement chatbots using DialogFlow(Google) and Lex(Amazon).

Since Lex provided better acceptable documentation and Team had prior experience in AWS, we opted to go ahead with AWS based Lex Bot Development.

This week involved testing an existing Chatbot in the AWS that involved testing Voice-based and Text-based input. This week focused on how Lex worked from a very top view.

Week 3 and 4:

In these two weeks, we focused on making a dummy chatbot which involved basic tasks such as Ordering Flowers and Booking Tables in a Restaurant by using existing templates and development documentation provided in AWS for understanding the functionality that we could incorporate into our Chatbot.

Week 5:

In this week, our first dummy version of covidchatbot was implemented which would ask username and would allow the user to interact with the bot being limited to the exchange of first greetings and understanding the region of the user’s interest.

Week 6,7:

These weeks were focused on deploying a database that would connect to the Lex bot. This database contained a set of dummy records that would allow us to develop and test the python scripts in our local Machine. We then focused on the JSON Response formats of Lex-Lambda Communication to understand the functionality provided by Lambda Functions and how these Message Responses would be used

Week 8,9,10:

These weeks involved making up of the Lambda Functions to create intents. We created two intents that were GetCountryData and GetCountyData\_IRE. GetCountryData intent is responsible for collecting data from the databases of the respective countries. The cases were Active, Recovered, Deaths and Confirmed Cases. It also involved Validation of User input for the name of the countries as well as the date being entered. The Intent was responsible for triggering the appropriate lambda function and fetching result based on the user input. The challenge that we faced at this stage is that many countries are known by different names such as Iran is also known as the Islamic Republic of Iran. Also, United States of America is also known as the USA, states, the United States depending on the user’s personal preferences. We could not mitigate these variations of user input within our Chatbot due to time constraints.

The second Intent we developed was focused on getting county wise data for the Republic of Ireland. These involved generating numbers for case types as the previous Intent.

We also focused on error handling around the lex chatbot. This included validation of User Input, validation of data in the database for the given user query. We also tried adding an Image response to the Chatbot Query Answers but were unable to do so as the Lex-Lambda Interaction happens only using JSON Format and Lex does not provide a website style rendering within itself. Also, triggering a store-and-supply image methodology would not have worked as it would involve storing many user inputs based generated Images into S3 buckets and then calling them to the Lex Bot. This would have added overhead to the existing chatbot solution, and thus the method was dropped. Due to time constraints, the image functionality had been put on hold to focus on Deployment of the lex chatbot.

Nripendra and Sumit were responsible for Lex and Django Framework Deployment, so the above Lex implementation was done in Parallel with Django Framework Deployment.